بسم الله الرحمن الرحیم

آموزش ویژوال #C 2005

گردآوری:
سید محمد هاشمیان

این کتاب آموزشی یک کتاب رایگان است و می توان مطالب موجود در آن را با/بدون اجازه گردآورنده توزیع کرده و یا تغییر داد. این کتاب به این امید گردآوری شده است که بتواند در ارتقای سطح علمی دوستان موثر واقع شود، اما هیچ تضمینی در مورد آن و یا در مورد مناسب بدون مطالب موجود برای اهداف خاص وجود ندارد.

از خواننده محترم تقاضا می شود با ارسال نظرات و پیشنهادات خود در مورد این کتاب و یا اشکالات موجود در محتوی یا متن آن به آدرس m.hashemian@gmail.com در بهبود این کتاب سهیم شود.

با تشکر،

سید محمد هاشمیان

تابستان ۱۳۸۵

فصل اول: به ویژوال #C ۲۰۰۵ خوش آمدید ۱۸
نصب ویژوال #C ۲۰۰۵: ۱۸
محیط توسعه ویژوال #C ۲۰۰۵: ۲۳
صفحه Profile Setup: ۲۳
منو: ۲۴
نوار ابزارها: ۲۶
ایجاد یک برنامه ساده: ۲۷
پنجره ها در IDE ویژوال استودیو ۲۰۰۵: ۲۸
امتحان کنید: ساختن پروژه Hello User ۲۹
جعبه ابزار: ۳۱
نشانه گذاری مجارستانی تغییر یافته: ۳۵
ویرایشگر کد: ۳۶
استفاده از سیستم راهنمای ویژوال استودیو: ۴۱
خلاصه: ۴۳
تمرین: ۴۳
فصل دوم: چارچوب .NET. و ارتباط آن با #C ۴۴
چارچوب .NET. چیست؟ ۴۴
چارچوب .NET. از چه اجزایی تشکیل شده است؟ ۴۵
چگونه با استفاده از چارچوب .NET. برنامه بنویسیم؟ ۴۶
MSIL و JIT: ۴۶
اسمبلی ها: ۴۷
کدهای مدیریت شده: ۴۸
مدیریت حافظه در .NET.: ۴۸
مراحل اجرای برنامه در .NET. ۴۸
لینک دادن: ۵۰
#C چیست؟ ۵۰
چه نوع برنامه هایی را میتوان با استفاده از #C انجام داد؟ ۵۱
ویژوال استودیو ۲۰۰۵: ۵۲
راه حلهای ویژوال استودیو: ۵۳
نتیجه: ۵۳
فصل سوم: نوشتن نرم افزار ۵۴
داده ها و اطلاعات: ۵۴
الگوریتم ها: ۵۴
یک زبان برنامه نویسی چیست؟ ۵۶
متغییر ها: ۵۶
کار با متغییر ها: ۵۶
توضیحات و فضاهای خالی: ۶۰

توضیحات: ۶۰
فضاهای خالی: ۶۲
نوع های داده ای: ۶۲
کارکردن با اعداد: ۶۲
عملیات ریاضی معمول روی اعداد صحیح: ۶۳
تند نویسی در عملیات ریاضی: ۶۶
محدودیت کار با اعداد صحیح: ۶۷
اعداد اعشاری: ۶۸
حالتهای دیگر: ۷۰
اعداد اعشاری با دقت معمولی: ۷۰
کار با رشته ها: ۷۱
اتصال رشته ها: ۷۳
استفاده از عملگر اتصال رشته در درون برنامه: ۷۴
عملیات بیشتر روی رشته ها: ۷۵
زیر رشته ها: ۷۷
قالب بندی رشته ها: ۷۹
قالب بندی بومی: ۸۰
جایگزینی زیررشته ها: ۸۱
تبدیل نوع های داده ای: ۸۲
استفاده از تاریخها: ۸۵
قالب بندی تاریخها: ۸۶
استفاده از خاصیتهای DateTime: ۸۸
کار با تاریخها: ۸۹
بولین: ۹۰
نگهداری متغییر ها: ۹۱
دودویی: ۹۱
بیتها و بایت ها: ۹۲
نمایش مقادیر: ۹۲
متدها: ۹۴
چرا از متدها استفاده می کنیم؟ ۹۵
متدهایی که تاکنون دیده اید: ۹۵
ایجاد یک متد: ۱۰۰
انتخاب نام برای متد: ۱۰۳
محدوده ها: ۱۰۴
نتیجه: ۱۰۶
تمرین: ۱۰۷
تمرین ۱: ۱۰۷
تمرین ۲: ۱۰۷

فصل چهارم: کنترل روند اجرای برنامه ۱۰۸
تصمیم گیری در برنامه: ۱۰۸
دستور if: ۱۰۸
دستور Else: ۱۱۱
بررسی چند شرط با else if: ۱۱۲
دستورات if تودرتو: ۱۱۴
عملگرهای مقایسه ای: ۱۱۵
استفاده از عملگر مخالف: ۱۱۵
استفاده از عملگر های مقایسه ای: ۱۱۷
عملگر های And و Or منطقی : ۱۲۱
استفاده از عملگر And منطقی: ۱۲۳
مطالب بیشتر در رابطه با عملگر های And و Or منطقی: ۱۲۵
مقایسه رشته ها: ۱۲۶
انتخاب بین حالتها با استفاده از switch: ۱۲۸
استفاده از switch با و بدون حساسیت به نوع حروف: ۱۳۲
انتخابهای چند گانه: ۱۳۶
دستور default: ۱۳۸
استفاده از نوع های داده ای گوناگون در دستور switch: ۱۳۹
حلقه ها: ۱۳۹
حلقه for: ۱۴۰
شمارش معکوس در حلقه: ۱۴۵
حلقه های foreach: ۱۴۶
حلقه های do: ۱۴۸
حلقه while: ۱۵۰
شرطهای قابل قبول برای حلقه های do و while: ۱۵۲
حلقه های تودرتو: ۱۵۳
خروج زودهنگام از حلقه: ۱۵۵
دستور continue: ۱۵۷
حلقه های بی نهایت: ۱۵۸
نتیجه: ۱۵۹
تمرین: ۱۶۰
تمرین ۱: ۱۶۰
تمرین ۲: ۱۶۰
فصل پنجم: کارکردن با ساختارهای داده ای ۱۶۱
مفهوم آرایه: ۱۶۱
تعریف و استفاده از آرایه ها: ۱۶۱
استفاده از foreach: ۱۶۴

انتقال آرایه ها به عنوان پارامتر: ۱۶۷
مرتب سازی آرایه ها: ۱۷۰
حرکت به عقب در آرایه ها: ۱۷۱
مقدار دهی اولیه به آرایه ها: ۱۷۳
مفهوم شمارنده ها: ۱۷۴
استفاده از شمارنده ها: ۱۷۵
تعیین موقیت: ۱۸۰
مفهوم ثابت ها: ۱۸۲
استفاده از ثابت ها: ۱۸۲
ثابتها با نوعهای داده ای گوناگون: ۱۸۵
ساختارها: ۱۸۶
ایجاد ساختارها: ۱۸۶
اضافه کردن خاصیت به ساختارها: ۱۹۰
کار با لیست های پیوندی: ۱۹۱
استفاده از لیست های پیوندی: ۱۹۲
حذف یک عنصر از لیست های پیوندی: ۱۹۷
نمایش عناصر موجود در لیست پیوندی: ۲۰۰
ایجاد جداول قابل جستجو با Hashtableها: ۲۰۲
استفاده از Hashtable: ۲۰۳
جلوگیری از وارد شدن عناصر تکراری: ۲۰۸
نتیجه: ۲۱۰
تمرین: ۲۱۰
تمرین ۱: ۲۱۱
فصل ششم: ایجاد برنامه های ویندوزی ۲۱۲
پاسخ به رویدادها: ۲۱۲
تنظیم یک رویداد برای کنترل Button: ۲۱۲
ایجاد یک برنامه ساده: ۲۱۸
ایجاد فرم: ۲۱۸
شمارش کاراکتر ها: ۲۲۰
شمارش کلمات: ۲۲۲
ایجاد برنامه های پیچیده تر: ۲۲۷
برنامه ویرایشگر متن: ۲۲۷
ایجاد نوار ابزار: ۲۲۸
ایجاد نوار وضعیت: ۲۳۲
ایجاد قسمت ویرایش متن: ۲۳۴
پاک کردن بخش ویرایشگر متن: ۲۳۵
پاسخ به رویدادهای نوار ابزار: ۲۳۷
مفهوم فوکوس: ۲۴۲

استفاده از چندین فرم در برنامه: ۲۴۴
فرم About: ۲۴۵
نتیجه: ۲۴۸
تمرین: ۲۴۹
تمرین ۱: ۲۴۹
تمرین ۲: ۲۴۹
فصل هفتم: نمایش کادرهای محاوره ای ۲۵۰
کادر محاوره ای MessageBox: ۲۵۰
آیکونهای قابل استفاده در یک کادر پیغام: ۲۵۱
دکمه های موجود برای کادر پیغام: ۲۵۱
تنظیم دکمه ی پیش فرض: ۲۵۲
گزینه های مختلف کادر پیغام: ۲۵۲
حالتهای مختلف استفاده از متد Show: ۲۵۳
کادرهای پیغام نمونه: ۲۵۵
کنترل OpenFileDialog: ۲۵۹
خاصیتهای کنترل OpenFileDialog: ۲۶۰
متدهای OpenFileDialog: ۲۶۲
استفاده از کنترل OpenFileDialog: ۲۶۲
کنترل SaveFileDialog: ۲۶۸
خاصیتهای کنترل SaveFileDialog: ۲۶۸
متدهای کنترل SaveFileDialog: ۲۶۹
استفاده از کنترل SaveFileDialog: ۲۶۹
کنترل FontDialog: ۲۷۳
خاصیتهای کنترل FontDialog: ۲۷۳
متدهای کنترل FontDialog: ۲۷۴
استفاده از کنترل FontDialog: ۲۷۴
کنترل ColorDialog: ۲۷۸
خاصیتهای کنترل ColorDialog: ۲۷۹
استفاده از کنترل ColorDialog: ۲۸۰
کنترل PrintDialog: ۲۸۲
خاصیتهای کنترل PrintDialog: ۲۸۳
استفاده از کنترل PrintDialog: ۲۸۳
کلاس PrintDocument: ۲۸۳
خصوصیات کلاس PrintDocument: ۲۸۴
چاپ یک سند: ۲۸۴
کنترل FolderBrowserDialog: ۲۹۳
خاصیتهای کنترل FolderBrowser: ۲۹۴
استفاده از کنترل FolderBrowser: ۲۹۵

نتیجه:.....۲۹۸
تمرین:.....۲۹۹
تمرین ۱:.....۲۹۹
تمرین ۲:.....۲۹۹
فصل هشتم: منو ها.....۳۰۱
درک ویژگیهای یک منو:.....۳۰۱
تصاویر:.....۳۰۱
کلیدهای دسترسی:.....۳۰۲
شورت کات ها:.....۳۰۲
علامت تیک:.....۳۰۲
پنجره Properties:.....۳۰۳
ایجاد منو ها:.....۳۰۴
طراحی منو ها:.....۳۰۴
اضافه کردن نوار ابزارها و کنترل ها:.....۳۰۷
نوشتن کد برای منو ها:.....۳۰۹
کد نویسی منوی View و نوار ابزارها:.....۳۱۵
امتحان برنامه:.....۳۱۷
منو های فرعی:.....۳۲۰
ایجاد منو های فرعی:.....۳۲۱
فعال و غیر فعال کردن گزینه های منو و دکمه های نوار ابزار:.....۳۲۴
نتیجه:.....۳۳۰
تمرین:.....۳۳۰
فصل نهم: ساختن اشیا.....۳۳۱
مفهوم اشیا:.....۳۳۱
کپسولی بودن:.....۳۳۳
متد ها و خاصیت ها:.....۳۳۳
رویدادها:.....۳۳۴
قابل رویت بودن:.....۳۳۴
یک کلاس چیست؟.....۳۳۵
ایجاد کلاسها:.....۳۳۶
قابلیت استفاده مجدد:.....۳۳۷
طراحی یک شیئ:.....۳۳۸
حالت:.....۳۳۹
رفتار:.....۳۳۹
نگهداری حالت:.....۳۴۰
خاصیت های فقط-خواندنی:.....۳۴۳
خاصیتهای خواندنی-نوشتنی:.....۳۴۸
متد IsMoving:.....۳۵۱

امتحان کنید: اضافه کردن متد IsMoving ۳۵۲
متدهای سازنده: ۳۵۴
ایجاد یک متد سازنده: ۳۵۴
وراثت: ۳۵۶
اضافه کردن متدها و خاصیت های جدید: ۳۵۷
اضافه کردن متد GetPowerToWeightRatio: ۳۶۱
تغییر دادن پیش فرض ها: ۳۶۲
چند شکلی بودن: کلمه ای ترسناک، مفهومی ساده ۳۶۴
Override کردن متدهای بیشتر: ۳۶۵
به ارث بردن از کلاس Object: ۳۶۹
اشیا و ساختارها: ۳۶۹
کلاسهای چارچوب NET.: ۳۷۰
فضای نام: ۳۷۰
راهنمای using: ۳۷۲
وراثت در چارچوب NET.: ۳۷۳
نتیجه: ۳۷۳
فصل دهم: مباحث پیشرفته برنامه نویسی شیئ گرا ۳۷۵
سربار گذاری متد ها: ۳۷۵
استفاده از خاصیت ها و متدهای Static: ۳۷۸
استفاده از خاصیت های Static: ۳۷۹
استفاده از متدهای Static: ۳۸۵
سربار گذاری عملگر ها: ۳۸۶
درک عملگر ها: ۳۸۷
چگونگی سربار گذاری عملگر ها: ۳۸۸
کلاسهای Abstract: ۳۹۴
کلاسهای sealed: ۴۰۱
Interfaceها: ۴۰۱
ایجاد یک برنامه ی کاربردی: ۴۱۴
شورت کات های اینترنتی و Favoritesها: ۴۱۴
استفاده از کلاسها: ۴۱۶
پیدا کردن گزینه های Favorites: ۴۲۳
مشاهده ی لینک ها: ۴۲۸
ایجاد نمونه ای دیگر از برنامه ی Favorite Viewer: ۴۳۱
ایجاد برنامه ی Favorites Tray: ۴۳۱
نمایش گزینه های Favorites: ۴۳۴
نتیجه: ۴۳۹
فصل یازدهم: اشکال زدایی و کنترل خطا در برنامه ۴۴۱

انواع مختلف خطاها: ۴۴۱
خطاهای دستوری: ۴۴۱
خطاهای اجرایی: ۴۴۴
خطاهای منطقی: ۴۴۴
اشکال زدایی: ۴۴۶
ایجاد یک برنامه نمونه: ۴۴۶
کنترل اجرای برنامه با استفاده از Breakpointها: ۴۵۱
گزینه های پر کاربرد در نوار ابزار Debug: ۴۵۲
پنجره ی Breakpoints: ۴۵۴
کنترل استثنا ها در برنامه: ۴۵۶
چگونگی یافتن بلاک Catch برای یک استثنا: ۴۵۷
کلاس Exception: ۴۵۸
دستور throw: ۴۵۹
دستورات try و catch: ۴۶۲
ایجاد بلاک های catch اختصاصی: ۴۶۵
خاصیت ها و متدهای کلاس Exception: ۴۶۸
نتیجه: ۴۷۱
فصل دوازدهم: ایجاد کتابخانه های کلاس ۴۷۳
مفهوم کتابخانه های کلاس: ۴۷۴
ایجاد یک کتابخانه ی کلاس: ۴۷۴
ایجاد یک کتابخانه ی کلاس برای Favorites Viewer: ۴۷۶
برنامه های چند لایه: ۴۸۰
استفاده از نامگذاری قوی ۴۸۲
امضا کردن اسمبلی ها: ۴۸۳
نسخه های یک اسمبلی: ۴۸۶
ثبت کردن یک اسمبلی: ۴۸۷
ابزار GacUtil: ۴۸۷
طراحی کتابخانه های کلاس: ۴۹۰
استفاده از یک کتابخانه ی کلاس شخص ثالث: ۴۹۱
استفاده از فایل InetrnetFavorites.dll: ۴۹۱
استفاده از Object Browser: ۴۹۲
نتیجه: ۴۹۴
تمرین: ۴۹۴
فصل سیزدهم: ایجاد کنترلهای سفارشی ۴۹۵
کنترلهای ویندوزی: ۴۹۵
ایجاد و تست کردن کنترل های سفارشی: ۴۹۶
ایجاد کردن خاصیت برای کنترل های سفارشی: ۵۰۰

اضافه کردن خاصیت ها: ۵۰۱
اضافه کردن متد به کنترل های سفارشی: ۵۰۳
اضافه کردن رویداد به کنترل: ۵۰۴
زمان اجرا یا زمان طراحی: ۵۰۹
ایجاد یک کتابخانه ی فرم: ۵۱۲
ایجاد یک کتابخانه ی فرم حاوی فرم ورود: ۵۱۲
استفاده از کتابخانه ی فرم ایجاد شده: ۵۲۳
استفاده از رویداد های موجود در کتابخانه ی فرم: ۵۲۶
نتیجه: ۵۳۰
تمرین: ۵۳۱
فصل چهاردهم: ایجاد برنامه های گرافیکی ۵۳۲
ایجاد یک برنامه ی Paint ساده: ۵۳۲
ایجاد یک پروژه همراه با کنترل های سفارشی: ۵۳۲
برنامه های گرافیکی چگونه کار می کنند؟ ۵۳۳
ترسیم بیت مپی: ۵۳۴
ترسیم برداری: ۵۳۵
ایجاد کلاس GraphicsItem: ۵۳۵
مختصات صفحه و مختصات برنامه: ۵۳۸
بررسی حرکات ماوس و رسم اشیای GraphicsCircle: ۵۴۰
نا معتبر سازی: ۵۴۷
بهینه سازی کردن رسم: ۵۴۸
انتخاب رنگ: ۵۴۹
ایجاد کنترل ColorPalette: ۵۴۹
پاسخ دادن به کلیک ها: ۵۵۶
استفاده از دو رنگ در برنامه: ۵۶۰
مشخص کردن رنگهای مورد استفاده: ۵۶۳
استفاده از رنگهای بیشتر در برنامه: ۵۷۲
استفاده از کادر Color: ۵۷۴
استفاده از رنگهای سیستمی: ۵۷۶
استفاده از ابزارهای متفاوت: ۵۷۷
استفاده از دایره های توخالی: ۵۷۸
کار با عکسها: ۵۸۴
نمایش تصاویر: ۵۸۵
تغییر اندازه ی تصاویر: ۵۸۷
متد های دیگر کلاس Graphics: ۵۹۰
نتیجه: ۵۹۱
فصل پانزدهم: استفاده از بانکهای اطلاعاتی ۵۹۲
بانک اطلاعاتی چیست؟ ۵۹۲

اشیای موجود در Access: ۵۹۳
جدولها: ۵۹۳
پرس وجو ها: ۵۹۴
دستور SELECT در زبان SQL: ۵۹۵
پرس وجو ها در Access: ۵۹۷
ایجاد یک پرس وجو: ۵۹۸
کامپوننت های دسترسی اطلاعات: ۶۰۳
DataSet: ۶۰۳
DataGridView: ۶۰۴
BindingSource: ۶۰۴
BindingNavigator: ۶۰۵
TableAdapter: ۶۰۵
اتصال داده ها: ۶۰۵
نتیجه: ۶۱۳
تمرین: ۶۱۴
تمرین ۱: ۶۱۴
تمرین ۲: ۶۱۵
فصل شانزدهم: برنامه نویسی بانک اطلاعاتی با SQL Server و ADO.NET ۶۱۶
ADO.NET ۶۱۷
فضای نام Data: ۶۱۷
کلاس SqlConnection: ۶۱۹
ایجاد بخشهای مختلف ConnectionString: ۶۱۹
متصل شدن و قطع کردن اتصال به یک بانک اطلاعاتی: ۶۲۱
کلاس SqlCommand: ۶۲۱
خاصیت Connection: ۶۲۲
خاصیت CommandText: ۶۲۲
خاصیت Parameters: ۶۲۳
متد ExecuteNonQuery: ۶۲۴
کلاس SqlDataAdapter: ۶۲۵
خاصیت SelectCommand: ۶۲۶
استفاده از Command Builder برای ایجاد دستورات SQL دیگر: ۶۲۸
متد Fill: ۶۲۸
کلاس DataSet: ۶۳۰
کلاس DataView: ۶۳۱
خاصیت Sort: ۶۳۲
خاصیت RowFilter: ۶۳۲
متد Find: ۶۳۳
استفاده از کلاسهای ADO.NET در عمل: ۶۳۴

کاربرد DataSet در برنامه: ۶۳۵
اتصال داده ها: ۶۴۶
BindingContext و CurrencyManager: ۶۴۷
اتصال کنترل ها: ۶۴۸
مثال ایجاد اتصال: ۶۴۹
نتیجه: ۶۸۹
تمرین: ۶۹۰
تمرین ۱: ۶۹۰
فصل هفدهم: برنامه های مبتنی بر وب ۶۹۱
معماری برنامه های تحت وب: ۶۹۱
برنامه های تحت وب در مقایسه با برنامه های تحت ویندوز: ۶۹۳
مزایای برنامه های تحت ویندوز: ۶۹۳
برنامه های تحت وب: ۶۹۴
اجزای اصلی برنامه های تحت وب: ۶۹۴
سرور وب: ۶۹۵
مرورگر: ۶۹۵
HTML: ۶۹۵
VBScript و JavaScript: ۶۹۶
CSS: ۶۹۶
ASP: ۶۹۶
مزایا: ۶۹۷
فایلهای خاص در یک برنامه ی تحت وب: ۶۹۷
فایل Global.asax: ۶۹۷
فایل web.config: ۶۹۸
استفاده از محیط ویژوال استودیو: ۶۹۸
کنترلهای موجود در جعبه ابزار: ۶۹۸
ایجاد برنامه ها تحت وب: ۶۹۹
ایجاد یک فرم وب برای پردازش سمت سرور و سمت کلاینت: ۷۰۰
دریافت اطلاعات و اعتبار سنجی آنها: ۷۰۷
طراحی ظاهر سایت: ۷۱۳
استفاده از کنترل GridView برای نمایش داده ها در فرم وب: ۷۲۷
محل قرارگیری یک برنامه ی تحت وب در ویژوال استودیو: ۷۳۵
نتیجه: ۷۳۸
تمرین: ۷۳۸
فصل هجدهم: تشخیص هویت در برنامه های تحت وب ۷۴۰
تشخیص هویت در یک سایت وب: ۷۴۰
تشخیص هویت با استفاده از ویندوز: ۷۴۰
تشخیص هویت با استفاده از فرمهای وب: ۷۴۱

ابزار مدیریت سایت وب (WAT):..... ۷۴۱
کنترلهای Login: ۷۵۲
نتیجه:..... ۷۶۵
تمرین: ۷۶۶
تمرین ۱:..... ۷۶۶
فصل نوزدهم: XML و ویژوال C# ۲۰۰۵..... ۷۶۷
درک XML:..... ۷۶۷
XML شبیه به چیست؟..... ۷۶۸
XML برای افراد مبتدی: ۷۷۱
پروژه ی دفتر تلفن: ۷۷۱
ایجاد پروژه:..... ۷۷۱
کلاس SerializableData: ۷۷۳
دریافت داده ها از یک فایل XML: ۷۸۰
تغییر در داده ها: ۷۸۴
فرستادن ایمیل:..... ۷۸۵
ایجاد لیستی از آدرسها:..... ۷۸۶
در نظر نگرفتن اعضا:..... ۷۹۲
استخراج رکورد ها از فایل XML: ۷۹۵
اضافه کردن رکورد های جدید: ۷۹۶
حرکت در بین داده ها: ۷۹۸
حذف کردن داده ها از برنامه:..... ۸۰۰
بررسی لبه ها: ۸۰۲
ایجاد یکپارچگی بین برنامه ی دفتر تلفن و دیگر برنامه ها:..... ۸۰۳
توضیح اصول یکپارچه سازی:..... ۸۰۳
خواندن اطلاعات برنامه ی دفتر تلفن در یک برنامه ی دیگر:..... ۸۰۵
نتیجه:..... ۸۱۱
تمرین: ۸۱۱
تمرین ۱:..... ۸۱۲
تمرین ۲:..... ۸۱۲
فصل بیستم: وب سرویس ها و .NET Remoting..... ۸۱۳
وب سرویس چیست؟..... ۸۱۳
نحوه ی عملکرد وب سرویس ها: ۸۱۴
SOAP:..... ۸۱۵
ایجاد یک وب سرویس:..... ۸۱۶
ایجاد یک وب سرویس ساده:..... ۸۱۷
اضافه کردن متدهای دیگر:..... ۸۲۰
ایجاد سرور پروژه ی Picture Service: ۸۲۲

ایجاد پروژه:..... ۸۲۲
برگرداندن آرایه ها به عنوان نتیجه ی متد:..... ۸۲۵
برگرداندن یک ساختار به عنوان نتیجه ی یک متد در وب سرویس:..... ۸۳۰
ایجاد برنامه ی کلاینت: ۸۳۵
WSDL: ۸۳۵
ایجاد برنامه ی کلاینت: ۸۳۵
اضافه کردن یک وب سرویس به برنامه:..... ۸۳۷
نمایش لیست فولدر ها در برنامه: ۸۳۹
نمایش لیست فایلهای موجود و انتخاب آنها:..... ۸۴۳
.NET Remoting: ۸۴۸
ایجاد پروکسی: ۸۵۲
نتیجه:..... ۸۵۶
تمرین: ۸۵۶
تمرین ۱:..... ۸۵۶
تمرین ۲:..... ۸۵۷
فصل بیست و یکم: توزیع برنامه های کاربردی ۸۵۸
منظور از توزیع برنامه چیست؟ ۸۵۸
توزیع برنامه با استفاده از روش ClickOnce:..... ۸۵۹
توزیع برنامه با استفاده از روش XCOPY: ۸۶۵
ایجاد یک برنامه ی نصب با استفاده از ویژوال استودیو ۲۰۰۵:..... ۸۶۵
ایجاد یک برنامه ی نصب کننده: ۸۶۶
ویرایشگر رابط کاربری برنامه ی نصب: ۸۷۰
توزیع راه حلهای گوناگون: ۸۷۴
اسمبلی های خصوصی: ۸۷۴
اسمبلی های عمومی: ۸۷۵
توزیع برنامه های ویندوزی:..... ۸۷۶
توزیع برنامه های مبتنی بر وب: ۸۷۶
توزیع وب سرویس ها: ۸۷۶
ابزارهای مفید: ۸۷۶
نتیجه:..... ۸۷۷
تمرین: ۸۷۸
تمرین ۱:..... ۸۷۸
تمرین ۲:..... ۸۷۸
فصل بیست و دوم: ایجاد برنامه های موبایل..... ۸۷۹
درک محیط: ۸۷۹
Common Language Runtime: ۸۸۰
ActiveSync: ۸۸۰
نوع های داده ای در CF: ۸۸۱

کلاسهای موجود در Compact Framework: ۸۸۲
ایجاد یک بازی برای Pocket PC: ۸۸۵
نتیجه: ۸۹۷
تمرین: ۸۹۸
ضمیمه ی ۱: ادامه ی مسیر ۸۹۹
منابع آنلاین: ۹۰۰
منابع مایکروسافت: ۹۰۰
منابع دیگر: ۹۰۱
ضمیمه ۲: برنامه نویسی تجاری و چند لایه در .NET ۹۰۲
چرا .NET ؟ ۹۰۲
مشکلات تجاری رفع شده توسط .NET ۹۰۲
کارایی و مقیاس پذیری: ۹۰۴
مزایای .NET : ۹۰۴
پذیرش استانداردهای همگانی: ۹۰۵
سرویس های وب: ۹۰۶
ویژگی های محیط توسعه Visual Stadio.NET : ۹۰۶
Common Language Runtime: ۹۰۶
زبان های برنامه نویسی .NET: ۹۰۶
Intermediate Language: ۹۰۷
تکامل برنامه نویسی لایه ای: ۹۰۷
تعریف: ۹۰۷
مدیریت متمرکز: ۹۰۷
محاسبات توزیع شده: ۹۰۸
کارایی: ۹۰۸
مقیاس پذیری: ۹۰۸
قواعد و دستورات تجاری: ۹۰۸
راحتی کاربر: ۹۰۹
برنامه های دو لایه: ۹۰۹
مدیریت کد در برنامه های دو لایه: ۹۰۹
کارایی: ۹۱۰
دسترسی داده ها: ۹۱۰
قواعد تجاری: ۹۱۱
برنامه های سه لایه: ۹۱۱
سرویس های کاربران: ۹۱۱
سرویس های تجاری: ۹۱۲
سرویس های اطلاعاتی: ۹۱۲
مدیریت کد: ۹۱۳
مقیاس پذیری: ۹۱۳

قواعد تجاری:.....۹۱۳
برنامه نویسی چند لایه: ۹۱۴
کلاس خارجی: ۹۱۴
کلاس اصلی تجاری:.....۹۱۵
کلاسهای دسترسی اطلاعات: ۹۱۶
سرویس های وب:.....۹۱۶
مدل لایه وب سرویس ها: ۹۱۷
چرا وب سرویس ها؟ ۹۱۷
ضمیمه ی ۳: معماری پلت فرم .NET Framework.....۹۱۹
کامپایل سورس کد به ماژول های مدیریت شده: ۹۱۹
ترکیب ماژول های مدیریت شده در اسمبلی ها: ۹۲۲
بارگذاری Common Language Runtime:.....۹۲۴
اجرای کد های مدیریت شده:.....۹۲۵
مجموعه کتابخانه کلاس .NET Framework:.....۹۲۹
سیستم نوع داده ای عمومی:.....۹۳۱
خصوصیات عمومی زبانهای برنامه نویسی:.....۹۳۳
ضمیمه ۴: مدیریت حافظه در .NET.....۹۳۵
درک مبانی کار Garbage Collector:.....۹۳۵
الگوریتم Garbage Collection:.....۹۳۸
ارجاع های ضعیف: ۹۴۲
نسلها: ۹۴۴
دیگر نتایج کارایی Garbage Collector: ۹۴۸
اشیای بزرگ: ۹۵۰

فصل اول: به ویژوال C# ۲۰۰۵ خوش آمدید

نوشتن برنامه برای یک کامپیوتر همانند یاد دادن گره زدن بند کفش به یک کودک است. تا زمانی که شما نتوانید درست مراحل کار را بیان کنید، هیچ کاری انجام نمی شود. ویژوال C# ۲۰۰۵ یک زبان برنامه نویسی است که به وسیله ی آن می توانید به کامپیوتر خود بگویید چه کارهایی را انجام دهد. اما کامپیوتر نیز مانند یک کودک است و فقط کارهایی را می تواند انجام دهد که مراحل آن به وضوح مشخص شوند. اگر تا کنون هیچ برنامه ای ننوشته باشید ممکن است این کار بسیار مشکل به نظر برسد، البته در بعضی مواقع نیز به همین صورت است. اما خوشبختانه، ویژوال C# ۲۰۰۵ زبانی است که سعی کرده است این موضوع را تا حد ممکن ساده کند و به شما اجازه می دهد تا کارهای بسیار مشکل را به سادگی انجام دهید. درک اتفاقاتی که در سطوح پایین برای اجرای یک برنامه رخ می دهد هیچ وقت ضرری نداشته است، اما در ویژوال C# ۲۰۰۵ برای نوشتن یک برنامه نیازی به درگیری با مسائلی از این قبیل ندارید و می توانید به راحتی بر الگوریتم برنامه ای که می خواهید بنویسید تمرکز کنید.
برنامه هایی که به وسیله ی ویژوال C# ۲۰۰۵ نوشته می شوند می توانند بر روی سیستم عامل ویندوز اجرا شوند. حتی اگر تاکنون هیچ برنامه ای برای کامپیوتر ننوشته باشید، در طول کتاب و اجرای تمرینات بخش "امتحان کنید"، بیشتر با جنبه های مختلف این زبان برنامه نویسی و همچنین.NET Framework آشنا می شوید. به زودی متوجه خواهید شد که برنامه نویسی برای کامپیوتر به این اندازه که تصور می کنید، مشکل نیست. بعد از مدت کمی که با آن آشنا شدید، به راحتی می توانید انواع مختلف برنامه ها را با ویژوال C# ۲۰۰۵ ایجاد کنید. ویژوال C# ۲۰۰۵ (همانطور که از اسم .NET مشخص است) میتواند برای ایجاد برنامه های قابل استفاده در اینترنت مورد استفاده قرار گیرد. شما می توانید با این زبان به راحتی برای دستگاه های موبایل و یا PocketPC ها برنامه بنویسید. نوشتن این نوع برنامه ها آنقدر ساده است که در پایان این کتاب، برای نمونه یک برنامه برای این دستگاه ها خواهیم نوشت. اما احتمالاً با این نکته موافقید که قبل از یادگیری دویدن، باید راه رفتن را آموخت. بنابراین در این قسمت از کتاب در برنامه نویسی ویندوز تمرکز می کنیم.
در این فصل:

- نصب ویژوال استودیو ۲۰۰۵
- گشتی در محیط توسعه متمرکز (IDE) ویژوال C# ۲۰۰۵
- چگونگی ایجاد یک برنامه ساده تحت ویندوز
- چگونگی استفاده از سیستم راهنمای جامع ویژوال استودیو ۲۰۰۵

نصب ویژوال C# ۲۰۰۵:

قبل از اینکه بتوانیم استفاده از ویژوال استودیو و ویژوال C# را شروع کنیم، باید آن را در کامپیوتر خودئ نصب کنیم. ویژوال C# به گ.نه های مختلفی به وسیله ی مایکروسافت ارائه شده است. نسخه ی ویژوال C#ای که شما از آن استفاده می کنید، ممکن است به یکی از حالتهای زیر باشد:

- به عنوان بخشی از ویژوال استودیو ۲۰۰۵، که یک مجموعه شامل ابزارها و زبانهای برنامه نویسی است: این مجموعه همچنین شامل ویژوال بیسیک، J# و نیز Visual C++ میشود. ویژوال استودیو در نسخه های Standard، Professional، Tools For Office و ویژوال استودیو Team

System منتشر شده است. هر کدام از این نسخه ها نسبت به نسخه قبلی از امکانات و ابزارهای بیشتری برای برنامه نویسی بهره مند هستند.

- به عنوان نگارش Express: این نگارش نسخه ای از ویژوال C# است که شامل یک سری امکانات و ویژگیهای محدود موجود در ویژوال استودیو میشود.

هر دوی این نسخه ها از C# به شما این امکان را می دهد تا برای ویندوز برنامه بنویسید. مراحل نصب هر دوی آنها نیز کاملاً واضح است. در حقیقت، باید گفت که ویژوال استودیو آنقدر باهوش است که بفهمد برای اجرا شدن روی کامپیوتر شما به چه چیزهایی نیاز دارد.
توضیحاتی که در بخش "امتحان کنید" زیر آمده است، برای نصب ویژوال استودیو ۲۰۰۵ نسخه ی Team System است. بیشتر قسمتهای نصب کاملاً واضح هستند و فقط باید گزینه های پیش فرض را انتخاب کنید. بنابراین صرف نظر از اینکه از کدامیک از نگارش های ویژوال استودیو استفاده می کنید، اگر هنگام نصب گزینه های پیش فرض را انتخاب کنید، در نصب برنامه نباید مشکلی داشته باشید.

امتحان کنید: نصب ویژوال C# ۲۰۰۵

۱) با قرار دادن CD ویژوال استودیو ۲۰۰۵ در درایو، برنامه ی نصب به صورت اتوماتیک اجرا می شود. اما اگر اجرا نشد میتوانید فایل setup.exe را از درون درایو اجرا کنید. برای این کار به منوی Start بروید و روی گزینه Run کلیک کنید. در پنجره ای که باز میشود، D:\setup.exe را تایپ کنید (D نام درایو ی است که CD یا DVD ویژوال استودیو در آن قرار دارد). بعد از اجرای برنامه Setup باید صفحه ای مشابه شکل ۱-۱ ببینید.
۲) این پنجره مراحلی را که برای نصب باید طی کنید نشان می دهد. برای اجرای درست فرآیند نصب، ویژوال استودیو نیاز دارد که یک سری از برنامه های روی سیستم عامل را به روز رسانی کند مثل سرویس پک ۱ برای ویندوز XP. برنامه ی نصب، لیستی از مواردی را که باید در سیستم به روز رسانده شوند را به شما نشان می دهد و شما باید قبل از ادامه ی نصب ویژوال استودیو، این برنامه ها را نصب کنید. بعد از اینکه ویژوال استودیو تغییرات لازم در سیستم را انجام داد، وارد نصب خود برنامه می شویم. برای این مرحله روی لینک Install Visual Studio کلیک کنید.
۳) بعد از قبول کردن قرارداد نوشته شده توسط شرکت، روی Continue کلیک کنید تا به مرحله بعد بروید.
۴) در این مرحله نوع های مختلفی که می توانید ویژوال استودیو را به آن صورت نصب کنید نمایش داده می شوند. همانطور که مشاهده می کنید ویژوال استودیو سه گزینه ی مختلف را در این قسمت در اختار شما قرار می دهد که عبارتند از:

- Default: این گزینه باعث می شود ویژوال استودیو با ابزارهایی که به صورت پیش فرض انتخاب شده اند در سیستم نصب شوند.
- Full: با انتخاب این گزینه، ویژوال استودیو و تمام ابزارهای جانبی آن به صورت کامل در سیستم شما نصب می شوند. اگر از نظر فضایی که با انتخاب این گزینه در سیستم شما اشغال می شود مشکلی ندارید، بهتر است که هنگام نصب این گزینه را انتخاب کنید تا ویژوال استودیو به صورت کامل نصب شود.
- Custom: با انتخاب این گزینه، لیستی از تمام قسمتهای موجود در ویژوال استودیو نمایش داده می شوند و می توانید انتخاب کنید که کدام قسمتها باید نصب شوند و کدامیک نباید نصب شوند.

در این مرحله برای اینکه با قسمتهای موجود در ویژوال استودیو نیز آشنا شویم، گزینه ی Custom را انتخاب کرده و دکمه ی Next را فشار دهید.

۵) با وارد شدن به این قسمت، لیست اجزای ویژوال استودیو را که می توانید نصب کنید مشاهده خواهید کرد. بدین ترتیب می توانید فقط قسمتهایی را که به آنها نیاز دارید نصب کنید. برای مثال اگر فضای دیسک شما کم است و از ویژوال C++ ۲۰۰۵ استفاده نمی کنید، می توانید آن را نصب نکنید. در این قسمت همچنین می توانید مکان نصب برنامه را نیز تعیین کنید (معمولا مکان اولیه مناسب است، مگر آنکه به دلیل خاصی بخواهید آن را تغییر دهید). تمام قسمتهایی را که در این مرحله مشاهده می کنید بعداً نیز میتوانند نصب شده و یا از حالت نصب خارج شوند. البته اگر می خواهید برنامه های بانک اطلاعاتی بنویسید، همانند برنامه هایی که در فصل ۱۶ توضیح داده شده اند، در این قسمت باید SQL Server 2005 Express که آخرین گزینه ی لیست است را نیز انتخاب کنید.

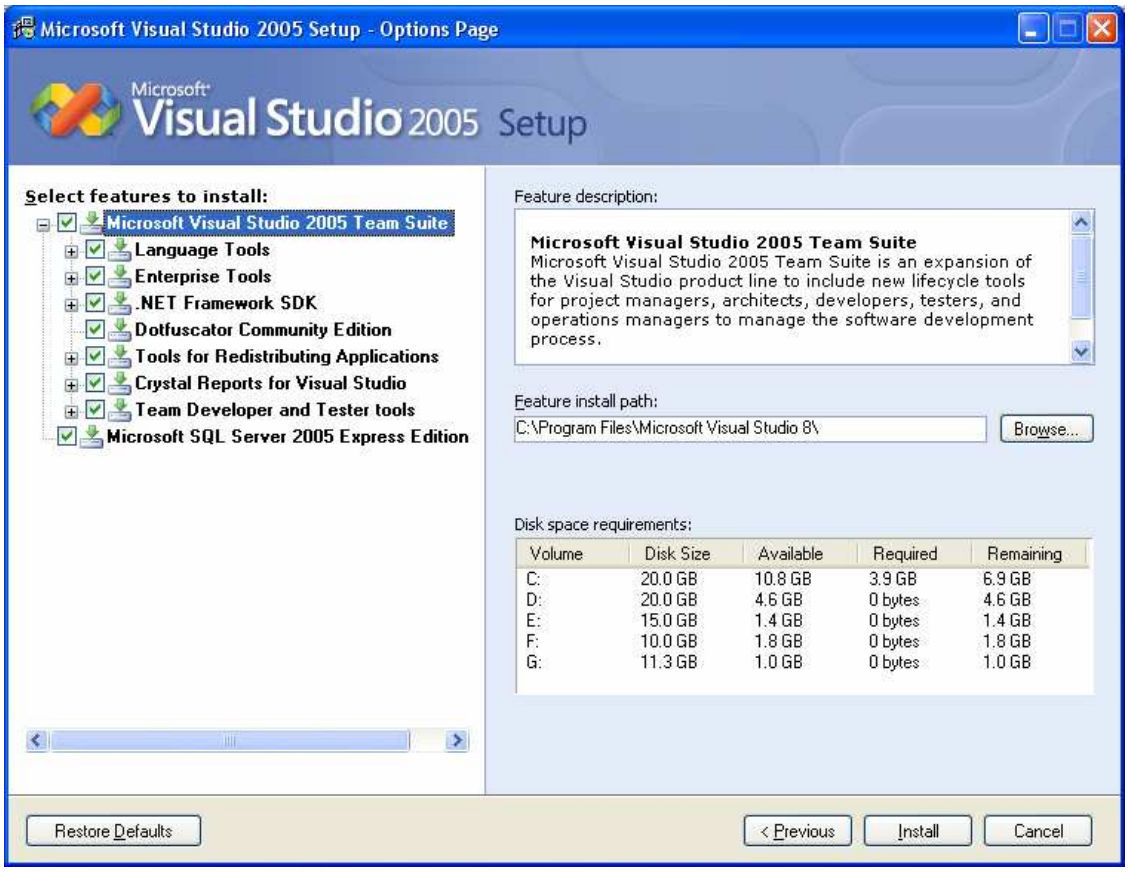


شکل ۱-۱

برای هر گزینه از لیست سه قسمت وجود دارند که اطلاعات آن را نمایش میدهند:

- قسمت Feature Description یک طرح کلی و کارایی قسمت انتخاب شده را شرح میدهد.
- قسمت Feature Install Path مکانی که فایلهای بخش انتخاب شده در آن نصب میشوند را نمایش میدهد.
- در نهایت، قسمت Disk Space Requirements به شما نشان میدهد که با نصب بخش انتخاب شده، فضای دیسک شما چگونه تغییر می کند.

بعد از اینکه نصب ویژوال استودیو به پایان رسید، زمانی که ویژوال C# ۲۰۰۵ را اجرا می کنید اطلاعات زیادی از دیسک به حافظه و برعکس منتقل می شوند. بنابراین داشتن مقداری فضای خالی در دیسک برای کار ضروری است. تعیین مقدار دقیق فضای مورد نیاز برای این مورد، امکان پذیر نیست. اما اگر می خواهید از این سیستم برای برنامه نویسی استفاده کنید، حداقل به ۱۰۰ مگا بایت فضای خالی نیاز دارید.



شکل ۱-۲

۶) بعد از انتخاب قسمتهایی که میخواهید نصب کنید، روی گزینه `Install` کلیک کنید. حالا شما میتوانید مقداری استراحت کنید، تا برنامه بر روی سیستم نصب شود. زمان مورد نیاز برای نصب برنامه بسته به قسمتهایی که برای نصب انتخاب کرده اید متفاوت است. ولی نصب برنامه در یک سیستم با پردازنده ی ۲,۴ گیگا هرتز و ۵۱۲ مگا بایت حافظه با ویندوز XP نسخه `Professional` حدود ۲۰ دقیقه طول میکشد.

۷) هنگامی که نصب برنامه تمام شد، صفحه ای را مشاهده می کنید که پایان نصب را اطلاع میدهد. در این مرحله، برنامه ی نصب هر مشکلی که با آن روبرو شده باشد را گزارش می دهد. همچنین در این مرحله میتوانید گزارش عملکرد نصب را نیز بررسی کنید. در این گزارش تمام کارهایی که در طول نصب برنامه انجام شده، نوشته شده است. این گزارش در مواقعی که برنامه نصب با مشکل روبرو می شود، می تواند مفید باشد. خوب، بدین ترتیب نصب ویژوال استودیو ۲۰۰۵ تمام شد. روی گزینه `Done` کلیک کنید تا وارد بخش نصب مستندات یا راهنمای ویژوال استودیو شویم.

۸) نصب `MSDN Library` کاملاً ساده و واضح است و در این بخش فقط قسمتهای مهم آن را ذکر می کنیم. اولین صفحه ای که در این مرحله میبینید، صفحه خوش آمد گویی است. برای ادامه کار روی `Next` کلیک کنید.

۹) همانطور که در شکل ۱-۳ نشان داده شده است، به شما اجازه داده می شود که مقدار مستنداتی کـه مـی خواهیـد نـصب کنید را مشخص کنید. برای شروع نصب روی گزینه Next کلیک کنید.

اگر دیسک شما فضای کافی دارد، بهتر است که MSDN را به صورت کامل نصب کنید. بدین ترتیب به تمـام اطلاعـات این کتابخانه دسترسی خواهید داشت. این مورد به خصوص در مواقعی که فقط قسمتی از ویژوال استودیو را نـصب کـرده اید و می خواهید بعدا قسمتهای دیگری را نیز نصب کنید، بسیار مفید خواهد بود.

۱۰) بعد از اینکه نصب MSDN به پایان رسید، به صفحه اول بـاز خواهیـد گـشت. در ایـن مرحلـه گزینـه Service Releases نیز فعال خواهد بود.



شکل ۱-۳

به وسیله ی این گزینه می توانید نسخه های Update این برنامه را با استفاده از اینترنت دریافت کنید. این نسخه ها می توانند شامل هر چیزی، از مستندات اضافی تا تعمیر خطاهای موجود در برنامه باشند. برای دریافت آنها نیز می توانید از اینترنت و یا از بسته های Service Pack استفاده کنید. البته مشخص است که برای نصب ایـن برنامـه هـا از طریق اینترنت، باید یک اتصال فعال به اینترنت داشته باشید و تا حد ممکن این اتصال سریع باشد، چون حجم این بـسته ها معمولا زیاد است.

بعد از طی کردن مراحل بالا ویژوال استودیو نصب شده و آماده استفاده است. از این قسمت به بعد تفریح واقعی شـروع میـشود! پـس اجازه بدهید وارد دنیای ویژوال C# ۲۰۰۵ بشویم.

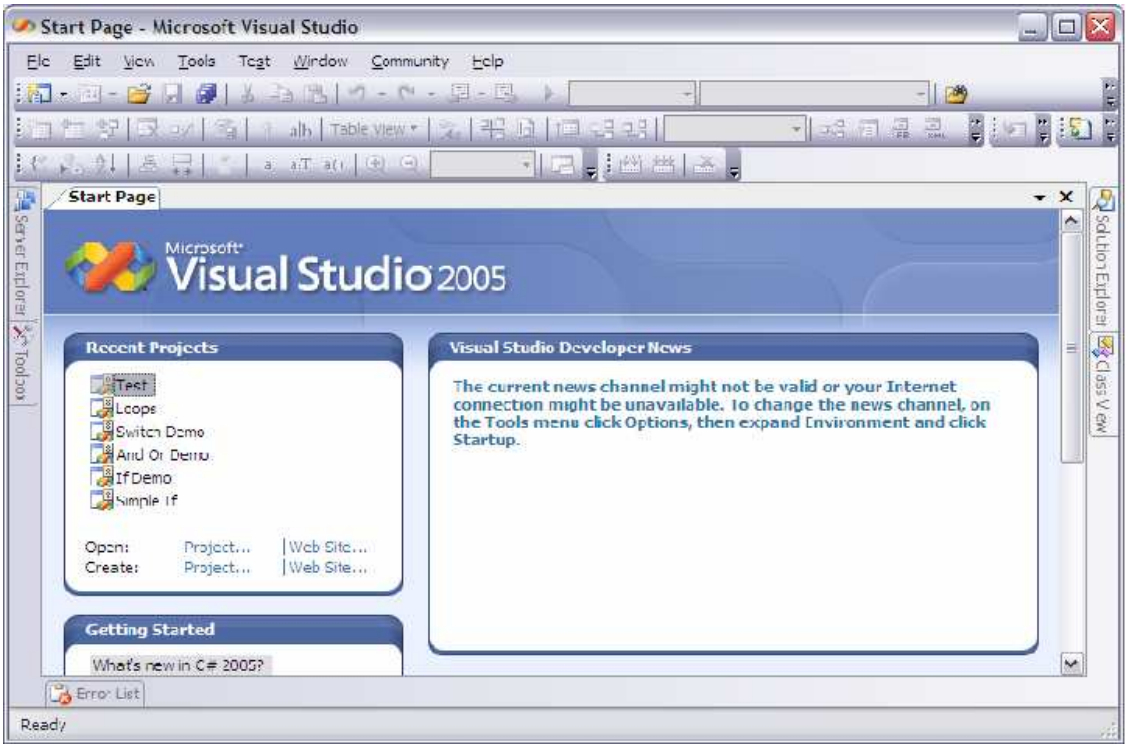
محیط توسعه[1] ویژوال C# ۲۰۰۵:

در ابتدا جالب است بدانید که برای برنامه نویسی به زبان C#، به برنامه ی ویژوال C# ۲۰۰۵ نیازی ندارید! شما می توانید برنامه های خود را با یک ویرایشگر متنی مانند Notepad نیز بنویسید. اما برنامه های ویژوال C# معمولا طولانی هستند و نوشتن آنها با notepad زمان زیادی را صرف می کند. اه بهتر برای انجام این کار استفاده از محیط توسعه مجتمع ویژوال استودیو که به عنوان IDE نیز شناخته می شود. IDE ویژوال استودیو امکانات بسیار زیادی را در اختیار شما قرار می دهد که مسلماً با استفاده از ویرایشگر های متنی به آنها دسترسی نخواهید داشت. برای مثال این محیط می تواند درستی کدهای نوشته شده را بررسی کند، قسمتهای تمام شده از برنامه را به صورت بصری نمایش دهد، خطاهای موجود در برنامه را تشخیص دهد و ....

صفحه Profile Setup:

یک IDE، محیطی است شامل یک سری ابزار که موجب سهولت کار توسعه و طراحی نرم افزار می شود. ویژوال استودیو ۲۰۰۵ را اجرا کنید تا ببینید با چه چیزی روبرو می شوید. اگر شما مراحل پیش فرض نصب را انتخاب کرده اید، به منوی استارت بروید و Programs را انتخاب کنید (All Programs در ویندوز XP یا ویندوز ۲۰۰۳) سپس از زیر منوی Microsoft Visual Studio 2005 گزینه ی Microsoft Visual Studio 2005 را انتخاب کنید. صفحه آغازین ویژوال استودیو به سرعت نمایش داده می شود و بعد از آن پنجره Choose Default Environment Settings را خواهید دید. از لیست ظاهر شده گزینه Visual C# Development Settings را انتخاب کرده و روی Start Visual Studio کلیک کنید. محیط توسعه مایکروسافت همانند شکل ۱-۴ نمایش داده خواهد شد.

[1] Integrated Development Environment (IDE)



شکل ۱-۴

منو:

احتمالا اشتیاق زیادی برای شروع کد نویسی دارید. اما در ابتدا بهتر است کمی IDE را بررسی کنیم. گردش خودمان را در IDE از منو ها و نوارهای ابزار شروع می کنیم. همانطور که میبینید، منو ها و نوار ابزارها در این برنامه نیز تفاوت چندانی با برنامه های دیگر مایکروسافت از قبیل Word و یا Excel ندارد.

نوار منوی Visual Studio 2005 به صورت دینامیک است، یعنی بر حسب کاری که می خواهید انجام دهید یک سری از گزینه ها به منو اضافه شده و یا از آن حذف می شوند. وقتی فقط محیط IDE خالی را در مقابل خود دارید، منوی ویژوال استودیو شامل گزینه های File، Edit، View، Data، Tools، Test، Window، Community و منوی Help است. اما هنگامی که کار بر روی یک پروژه را شروع کنید منوی کامل ویژوال استودیو ۲۰۰۵ همانند شکل ۱-۵ نمایش داده خواهد شد.



شکل ۱-۵

در اینجا به توضیح کامل در مورد همه ی منو ها نیازی نداریم. در طول این کتاب به مرور با تمامی آنها آشنا خواهید شد. اما در زیر برای آشنایی اولیه، شرح مختصری از عملکرد هر یک از منو ها آورده شده است:

- **File**: به نظر میرسد که همه برنامه های ویندوزی یک منوی فایل دارند. در این منو حداقل چیزی کـه پیـدا میـشود، راهی برای خارج شدن از برنامه است. البته در منوی File این برنامه، گزینه های بیشتری مثل بـاز کـردن، بـستن یـا ذخیره کردن یک فایل خاص و یا تمام پروژه هم وجود دارد.
- **Edit**: این منو هم مثل برنامه های دیگر شامل گزینه هایی است که انتظار آن را داریـد:Undo ، Redo، Cut، Copy، Paste و Delete.
- **View**: منوی View به شما اجازه می دهد تا به سرعت به پنجره های موجود در IDE مثـل Solution Explorer، پنجره Properties، پنجره Output، Toolbarها و ... دسترسی داشته باشید.
- **Project**: این منو به شما اجازه میدهد تا فایلهای مختلف از قبیل فرمهای جدید و یا کلاسها را به برنامـه ی خـود اضافه کنید.
- **Build**: این منو زمانی مفید خواهد بود که برنامه ی خود را تمام کنید و بخواهید کـه آن را بـدون اسـتفاده از محـیط Visual C# اجرا کنید (احتمالا از طریق منوی استارت، مثل همه برنامه های ویندوزی دیگر از قبیل Word و یا Excel)
- **Debug**: این منو به شما اجازه میدهد تا برنامه خودتان را در داخل محیط ویـژوال اسـتودیو خـط بـه خـط اجـرا کنیـد. همچنین از طریق این منو شما به دیباگر ویژوال استودیو ۲۰۰۵ نیز دسترسی خواهید داشت. به وسیله دیبـاگر مـی توانیـد عملکرد کد خود را در هنگام اجرای برنامه خط به خط بررسی کرده و مشکلات آن را متوجه شوید.
- **Data**: این منو به شما کمک می کند تا از اطلاعات بدست آمده از یک بانک اطلاعاتی استفاده کنیـد. البتـه ایـن منـو زمانی نمایش داده می شود که در حال کار بر روی قسمتهای بصری برنامه خود باشید (در پنجره اصلی ویژوال اسـتودیو، قسمت [Design] فعال باشد)، نه زمانی که در حال نوشتن کد هستید. در فصول ۱۵ و ۱۶ کتاب، کار بـا بانکهـای اطلاعاتی را بررسی خواهیم کرد.
- **Format**: این منو نیز فقط زمانی که در حال کار با قسمت های بصری برنامه باشید نمایش داده می شود. بـه وسـیله گزینه های این منو می توانید طریقه قرار گرفتن اشیای موجود در فرم برنامه (از قبیل TextBoxها، دکمه ها و ...) را کنترل کنید.
- **Tools**: در این قسمت می توانید محیط IDE ویژوال استودیو ۲۰۰۵ را کنترل و یا تنظیم کنید. همچنین لینکـی بـه برنامه های اضافی نصب شده در کنار ویژوال استودیو نیز، در این قسمت وجود دارد.
- **Test**: منوی Test به شما اجازه می دهد برنامه هایی ایجاد کنید تا به وسیله آن بتوانید بعد از اتمـام یـک برنامـه، قسمتهای مختلف آن را از نظر کارایی و یا عملکرد بررسی کنید.
- **Window**: این منو در همه برنامه هایی که امکان باز کردن بیش از یک پنجره در هر لحظه را به کاربر می دهند، مثل Word و یا Excel، نیز وجود دارد. گزینه های موجود در این منو به شما اجازه می دهند کـه در بـین پنجـره هـای موجود در IDE جا به جا شوید. نام پنجره هایی که در هر لحظه در محیط ویژوال استودیو باز هستند، در پایین نوار ابـزار نمایش داده می شوند که با کلیک کردن روی هر کدام از آنها، پنجره مربوطه نمایش داده می شود.
- **Community**: این منو، دسترسی به منابع برنامه نویسی، مکان هایی برای پرسیدن سوالات و نیز جستجو بین نمونه کدها را در اینترنت فراهم می کند.
- **Help**: منوی Help به شما اجازه دسترسی به مستندات ویژوال استودیو ۲۰۰۵ را می دهـد. راه هـای زیـادی بـرای دسترسی به این اطلاعات وجود دارند (برای مثال از طریق محتویات، اندیس و یا جستجو). این منو همچنین دارای گزینـه هایی برای وصل شدن به وب سایت مایکروسافت، دریافت آخرین نسخه های بـه روز رسـانی و همچنـین گـزارش دادن مشکلات برنامه است.

نوار ابزارها:

نوار ابزارهای زیادی در IDE ویژوال استودیو وجود دارند، مانند Image Editor، Formatting و یا Text Editor. برای حذف و یا اضافه این نوار ابزارها می توانید از گزینه Toolbars در منوی View استفاده کنید. هر کدام از این نوار ابزارها، دسترسی سریع شما را به یک دستور پرکاربرد فراهم می کند. بدین صورت مجبور نخواهید بود که هر بار برای اجرای آن دستور منو ها را زیر و رو کنید. برای مثال، گزینه ی ...File ➔ New ➔ Project از نوار منو، به وسیله ی سمت چپ ترین آیکون در نوار ابزار پیش فرض که نوار ابزار استاندارد نیز نامیده میشود (شکل ۶-۱) نیز قابل دسترسی است.



شکل ۱-۶

نوار ابزار استاندارد به چند بخش که شامل گزینه های مرتبط به هم هستند تقسیم شده است. هر بخش به وسیله یک خط عمودی از بخشهای دیگر تفکیک شده است. پنج آیکون اول، شامل کارهای عمومی بر روی فایل و یا پروژه هستند که از طریق منوی File و یا منوی Project قابل دسترسی اند، مانند باز کردن و یا ذخیره کردن فایلها.

گروه بعدی آیکون ها، برای ویرایش استفاده میشود (Cut، Copy و Paste). گروه بعدی نیز برای لغو کردن آخرین عمل انجام شده، دوباره انجام دادن آن و یا جا به جا شدن بین کدها است.

گروه چهارم از آیکون ها به شما اجازه می دهد اجرای برنامه خود را شروع کنید (به وسیله مثلث سبز رنگ). در این قسمت همچنین میتوانید پیکر بندی برنامه تان را مشخص کرده و یا نحوه اجرای آن را تعیین کنید.

در بخش بعدی میتوانید متن خاصی را در بین کدهای فایلی که هم اکنون باز است، در بین مستندات برنامه و یا در بین کل پروژه جستجو کنید.

گروه آخر از آیکون ها دسترسی سریع شما را به قسمتهای مختلف ویژوال استودیو مانند Solution Explorer، پنجره Properties، Toolbox، Object Browser، Start Page و یا صفحات دیگر فراهم می کند (در قسمتهای بعدی با این پنجره ها بیشتر آشنا خواهیم شد). اگر هر کدام از این پنجره ها بسته شده باشد، با کلیک بر روی آیکون آن در این قسمت، پنجره مورد نظر نمایش داده خواهد شد.

نکته: اگر فراموش کردید که هر آیکون چه کاری انجام میدهد، اشاره گر ماوس خود را برای چند لحظه بر روی آن نگه دارید. بدین ترتیب کادری ظاهر میشود که نام آیکون مورد نظر را نمایش میدهد.

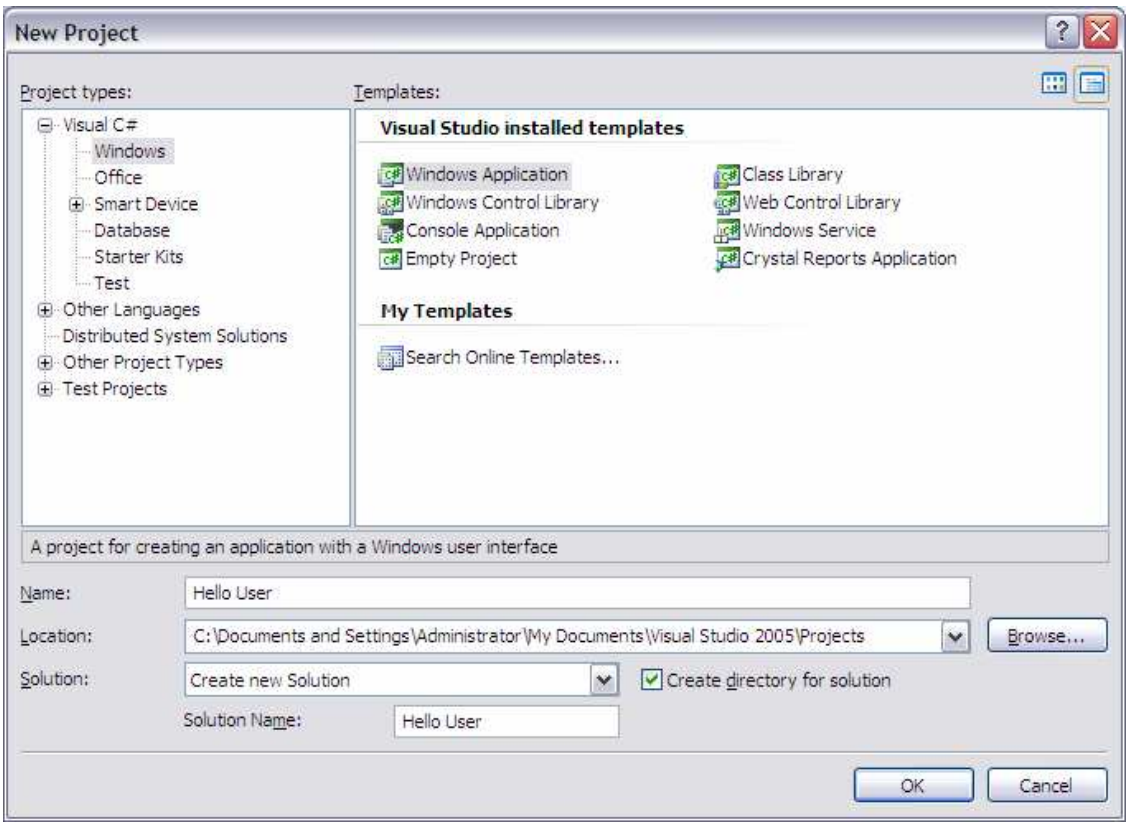
برای دیدن بقیه پنجره های ویژوال استودیو می توانید از منوی View، پنجره مورد نظرتان را انتخاب کنید. اما همانطور که میبینید، بیشتر آنها در حال حاظر خالی هستند و نمیتوان عملکرد آنها را فهمید. بهترین راه فهمیدن کاربرد این قسمتها، کار کردن با IDE و استفاده از این قسمتها در طول نوشتن کد برای یک برنامه است.

ایجاد یک برنامه ساده:

برای اتمام گردش در IDE ویژوال استودیو، بهتر است یک برنامه ساده بسازیم. به این ترتیب، در پنجره های قبلی مقداری اطلاعات واقعی و جالب قرار می گیرند که می توانید آنها را بررسی کنید. در بخش "امتحان کنید" زیر، یک برنامه کاملاً ساده به نام HelloUser خواهید ساخت که در آن کاربر میتواند نام خود را در یک کادر متنی وارد کند. سپس برنامه یک پیغام خوش آمد گویی به کاربر، با نام او، نمایش خواهد داد.

امتحان کنید: ایجاد یک پروژه HelloUser

۱) بر روی دکمه ی New Project در نوار ابزار کلیک کنید.
۲) پنجره New Project نمایش داده خواهد شد. مطمئن شوید که در قسمت Project Type در سمت چپ، گزینه Visual C# انتخاب شده باشد. سپس در بخش Templates در سمت راست، گزینه Windows Applications را انتخاب کنید. در کادر Name کلمه Hello User را تایپ کرده و در انتها روی OK کلیک کنید. پنجره New Project شما باید چیزی مشابه شکل ۱-۷ باشد.



شکل ۱-۷

۳) با کلیک کردن روی OK، IDE ویژوال استودیو یک برنامه ویندوزی خالی برای شما ایجاد میکند. در حال حاظر، برنامه ی Hello User فقط دارای یک پنجره ویندوزی خالی است که یک **فرم ویندوزی**[1] (یا به اختصار یک **فرم**) نامیده می شود. نام پیش فرض این فرم، همانطور که در شکل ۱–۸ نشان داده شده است، Form1.cs است.

نکته: هر زمانی که ویژوال استودیو بخواهد یک فایل جدید را ایجاد کند، چه این فایل در هنگام ساختن پروژه ایجاد شود و چه بعداً به برنامه اضافه شود، نامی به آن فایل اختصاص می دهد که از دو قسمت تشکیل شده است. قسمت اول نوع فایل را توصیف می کند و قسمت دوم نیز یک عدد است که مشخص می کند این فایل، چندمین فایل از این نوع است.

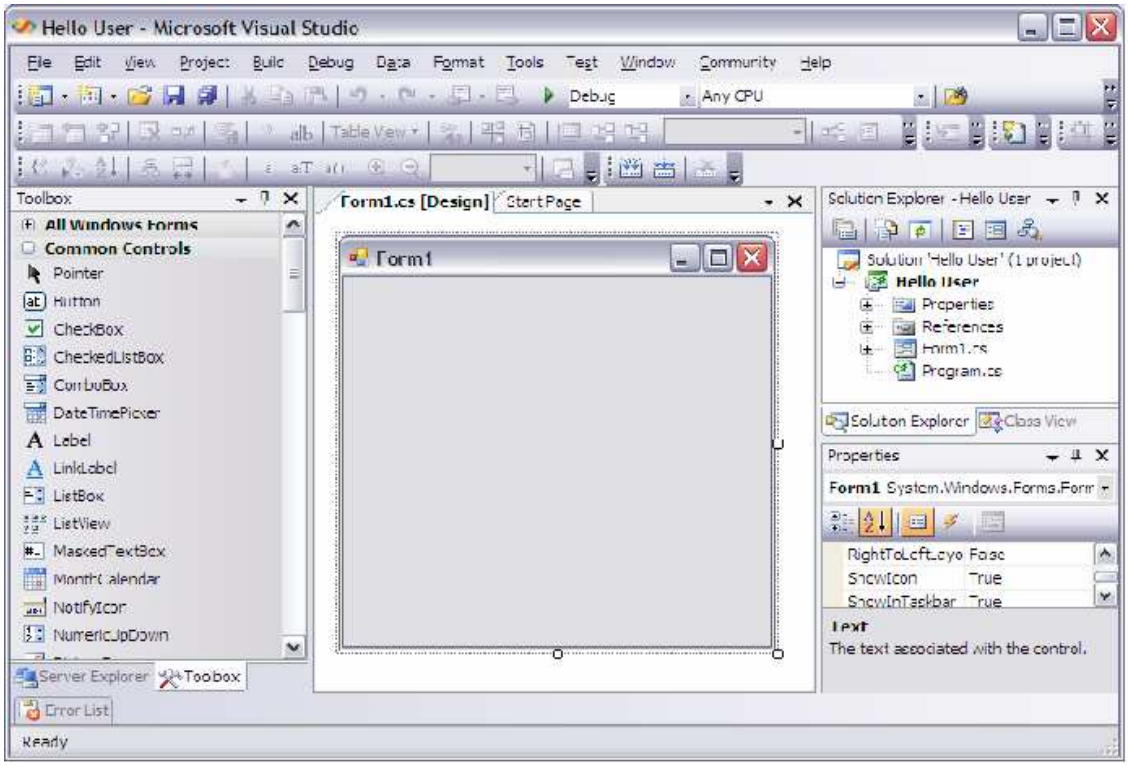
پنجره ها در IDE ویژوال استودیو ۲۰۰۵:

در محیط ویژوال استودیو پنجره های زیادی را مشاهده می کنید که هر کدام کاربرد خاصی دارند. بهتر است قبل از ادامه ی بخش "امتحان کنید"، تعدادی از آنها را به اختصار بررسی کنیم. یادآوری می کنم که اگر هر یک از این پنجره ها در کامپیوتر شما نمایش داده نمی شوند، از منوی View گزینه مربوط به آن را انتخاب کنید تا آن پنجره دیده شود. همچنین اگر از مکان قرارگیری یک پنجره خاص راضی نیستید، با کلیک بر روی نوار عنوان پنجره (نوار آبی رنگ بالای پنجره مورد نظر) و کشیدن آن به مکان جدید، جای آن را تغییر دهید. پنجره ها میتوانند درون IDE شناور باشند و یا به یکی از لبه ها وصل شوند(همانند شکل ۱–۸). لیست زیر عمومی ترین پنجره ها را معرفی میکند.

- **Server Explorer**: این پنجره دسترسی شما را به سرورهای بانک اطلاعاتی که برای برنامه تعریف کرده اید فراهم می کند. در این قسمت می توانید اتصالات جدیدی را به این سرورها ایجاد کنید و یا اطلاعات موجود در بانکهای اطلاعاتی کنونی را مشاهده کنید. در تصویر ۱–۸، پنجره Server Explorer، تب[2] موجود در زیر پنجره Toolbox است.
- **Toolbox**: این پنجره شامل کنترل ها و کامپوننت هایی است که می توانید به برنامه خود اضافه کرده و با استفاده از آن پنجره ی برنامه ی خود را طراحی کنید. این کنترل ها شامل کنترلهای عمومی مانند دکمه ها یا اتصال دهنده های داده ای، کنترلهای خریداری شده و یا کنترل هایی است که خودتان طراحی کرده اید.
- **Design Window**: این قسمت، بخشی است که بیشترین فعالیتها در آن صورت میگیرد. در این بخش شما رابط کاربری برنامه تان را بر روی فرم برنامه طراحی می کنید. این پنجره در بعضی مواقع Designer هم نامیده می شود.
- **Solution Explorer**: این پنجره یک نمای درختی از **راه حل**[3] شما را نمایش می دهد. یک راه حل می تواند شامل چندین پروژه باشد، که هر یک از این *پروژه* ها خود نیز می توانند شامل فرم ها، کلاسها، ماژول ها، و یا کامپوننت هایی باشند که یک مسئله خاص را حل می کند. در فصل دوم بیشتر در مورد یک راه حل صحبت خواهیم کرد.
- **Properties**: پنجره Properties خاصیتهای قابل تغییر شیئ انتخاب شده را نمایش می دهد. اگرچه می توانید این خاصیت ها را از طریق کد تنظیم کنید، اما در بعضی از مواقع تنظیم کردن آنها در زمان طراحی برنامه راحت تر است (برای مثال، موقع قرار دادن کنترل ها در فرم). دقت کنید که خاصیت File Name دارای مقدار Form1.cs است. این نام، نام فیزیکی فایل حاوی کدهای فرم و اطلاعات ظاهری آن است.

---

[1] Windows Form - WinForm
[2] Tab
[3] Solution



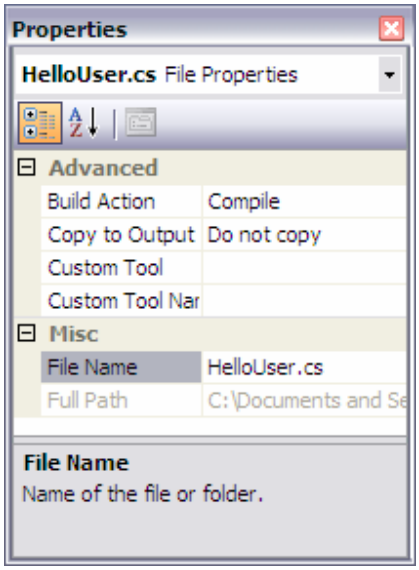
شکل ۱–۸

امتحان کنید: ساختن پروژه Hello User

۱) ابتدا نام فرم خود را به چیزی تغییر دهید که بیشتر معرف برنامه شما باشد. برای این کار روی `Form1.cs` در `Solution Explorer` کلیک کنید. سپس، در پنجره `Properties` خاصیت `File Name` را از `Form1.cs` به `HelloUser.cs` تغییر داده (شکل ۱–۹) و `Enter` را فشار دهید. بعد از تغییر هر خاصیت در پنجره `Properties`، برای اعمال آن باید کلید `Enter` را فشار دهید و یا در جایی خارج از پنجره کلیک کنید.

۲) توجه کنید که اسم فایل در پنجره `Solution Explorer` هم به `HelloUser.cs` تغییر می کند.

۳) حالا روی فرمی که در پنجره ی `Design` نمایش داده شده است کلیک کنید. پنجره `Properties` تغییر کرده و خاصیتهای `Form` انتخاب شده را نمایش می دهد (به جای خاصیتهای فایل `HelloUser.cs` که در قسمت قبلی در حال نمایش آن بود). مشاهده می کنید که خاصیتهای این قسمت کاملاً متفاوت با قسمت قبلی است. تفاوتی که در این جا وجود دارد به علت دو نگاه متفاوت به یک فایل است. زمانی که نام فرم در `Solution Explorer` انتخاب شده است، خاصیتهای مربوط به فایل فیزیکی فرم نمایش داده می شود. اما زمانی که فرم موجود در بخش `Designer` انتخاب شود، خاصیتهای منطقی و بصری فرم نمایش داده می شود.



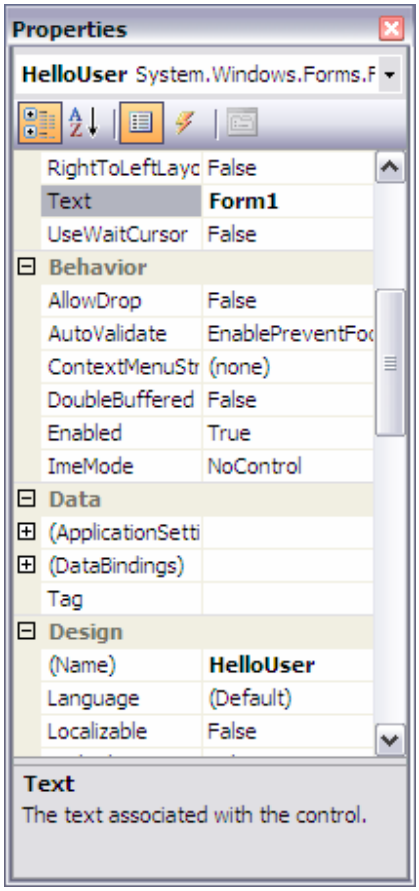
شکل ۱-۹

با استفاده از پنجره Properties میتوانید خاصیتهای یک کنترل را به راحتی تغییر دهید. خاصیت ها یک مجموعه ی ویژه، داخل اشیا هستند. آنها معمولا رفتار یا ظاهر یک شیئ را توصیف می کنند. همانطور که در شکل ۱-۱۰ می بینید خاصیت ها در گروه های مختلف قرار میگیرند که عبارتند از: Accessibility(نمایش داده نشده است)، Appearance(نام این گروه نیز در شکل مشخص نیست)، Behavior، Data، Design، Focus(نمایش داده نشده است)، Layout(نمایش داده نشده است)، Misc(نمایش داده نشده است) و Window-Style(نمایش داده نشده است).

نکته: همانطور که در شکل مشخص است، با وجود اینکه خاصیت نام فایل مربوط به فرم را به Hello User تغییر داده ایم، اما عنوان فرم همچنان Form1 است.

۴) در حال حاظر، عنوان این فرم Form1 است. این عنوان کاربرد برنامه را مشخص نمی کند. پس آن را تغییر می دهیم تا بیشتر معرف برنامه باشد. خاصیت Text را در بخش Appearance در پنجره Properties انتخاب کرده و مقدار آن را به Hello From Visual C# 2005 تغییر داده، سپس Enter را فشار دهید. توجه کنید که عنوان فرم در بخش Designer برابر با مقداری می شود که در کادر مربوطه وارد کرده اید.
اگر پیدا کردن خاصیت مورد نظرتان از لیست در حالت گروه بندی شده مشکل است، بر روی گزینه AZ در نوار ابزار بالای پنجره Properties کلیک کنید. بدین ترتیب لیست خاصیت ها به صورت الفبایی مرتب می شوند.

۵) بر روی دکمه Start در نوار ابزار ویژوال استودیو کلیک کنید (مثلث سبز رنگ) تا برنامه اجرا شود. در طول این کتاب هر جا عبارتهای "برنامه را اجرا کنید" و یا "برنامه را شروع کنید" دیدید، روی کلید Start کلیک کنید. بعد از کلیک روی این دکمه، یک پنجره خالی با عنوان Hello From Visual C# 2005 نمایش داده خواهد شد.



شکل ۱-۱۰

خیلی راحت بود، اما برنامه کوچک شما کاری انجام نمی دهد. اجازه بدهید مقداری برنامه را محاوره ای تر کنیم. برای این کار، باید تعدادی کنترل به فرم اضافه کنیم، دو دکمه، یک لیبل و یک کادر متنی. به زودی خواهید دید که اضافه کردن اینها با استفاده از Toolbox چه قدر راحت است. یکی از مزیت های Visual C# این است که شما میتوانید مقدار زیادی از برنامه خودتان را طراحی کنید بدون اینکه کدی بنویسید. البته برای آنها کد نوشته می شود، اما این کد دور از دید شما است و ویژوال C# آنها را برای شما می نویسد.

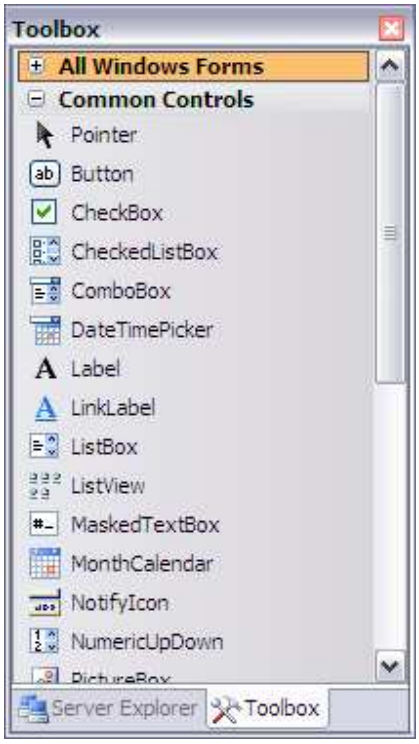
جعبه ابزار:

برای دسترسی به جعبه ابزار سه راه وجود دارد:

۱. از منوی View گزینه Toolbox را انتخاب کنید.
۲. از نوار ابزار استاندارد آیکون مربوط به آن را انتخاب کنید.
۳. کلیدهای Ctrl+Alt+X را فشار دهید.

بدین ترتیب جعبه ابزار در قسمت چپ IDE نمایش داده میشود. جعبه ابزار شامل کنترل ها و کامپوننت هـایی مـی شـود کـه مـی توانید بر روی فرم خود قرار دهید. کنترل ها مانند دکمه ها، کادر های متنی، دکمه های رادیویی و یا لیست های ترکیبی می توانند از جعبه ابزار انتخاب شوند و روی فرم قرار گیرنـد. بـرای برنامـه HelloUser شـما فقـط از کنترلهـای قـسمت Common Controls در جعبه ابزار استفاده می کنید. در شکل ۱-۱۱ می توانید لیستی از کنترلهای عمومی بـرای فرمهـای ویندوزی را مشاهده کنید.

کنترل ها می توانند به هر ترتیبی که بخواهید به فرم اضافه شوند. بنابراین این مورد که شما دکمه ها را قبل از کادرهـای متنـی بـر روی فرم قرار دهید و یا لیبل ها را قبل از دکمه ها رسم کنید اهمیتی ندارد. در "امتحان کنید" بعدی، قرار دادن کنتـرل هـا بـر روی فرم را شروع می کنیم.



شکل ۱-۱۱

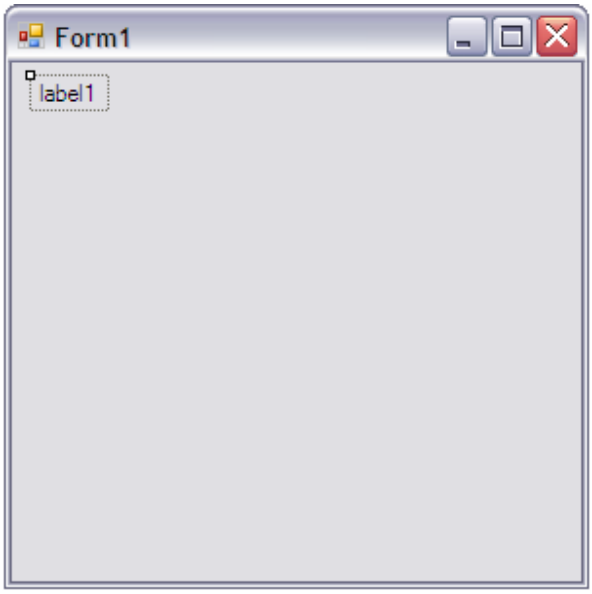
امتحان کنید: اضافه کردن کنترل ها به برنامه ی HelloUser

۱) اگر برنامه هم اکنون در حال اجرا است آن را متوقف کنید، زیرا باید تعدادی کنترل به فرم اضافه کنیـد. بهتـرین راه بـرای بستن برنامه کلیک کردن روی دکمه ی X در سمت راست نوار عنوان است.همچنین می توانید بر روی مربع آبی رنگ در IDE کلیک کنید (اگر اشاره گر ماوس خود را بر روی آن نگه دارید عبـارت "Stop Debugging" در کـادر زرد نمایش داده می شود).

۲) یک کنترل لیبل به فرم اضافه کنید. برای این کار، در جعبه ابزار روی کنترل Label کلیک کنید و آن را تا محل مورد نظرتان بر روی فرم بکشید و سپس آن را رها کنید. همچنین برای قرار دادن یک کنترل روی فرم میتوانید بر روی آیکون آن در جعبه ابزار دو بار کلیک کنید.

۳) اگر کنترل لیبلی که بر روی فرم قرار داده اید در مکان مناسبی قرار نگرفته است، مشکلی نیست. هنگامی که کنترلی بر روی فرم قرار می گیرد، میتوانید آن را جا به جا کنید و یا اندازه آن را تغییر دهید. شکل ۱-۱۲ فرم برنامه را بعد از قرار دادن کنترل بر روی آن نشان میدهد. برای حرکت دادن کنترل روی فرم، بر روی ناحیه نقطه چین در فرم کلیک کنید و آن را به مکان مورد نظرتان بکشید. از نظر اندازه هم، کنترل لیبل خود را با متنی که درون آن وارد می کنید هم اندازه میکند. پس خیالتان می تواند از این نظر راحت باشد.

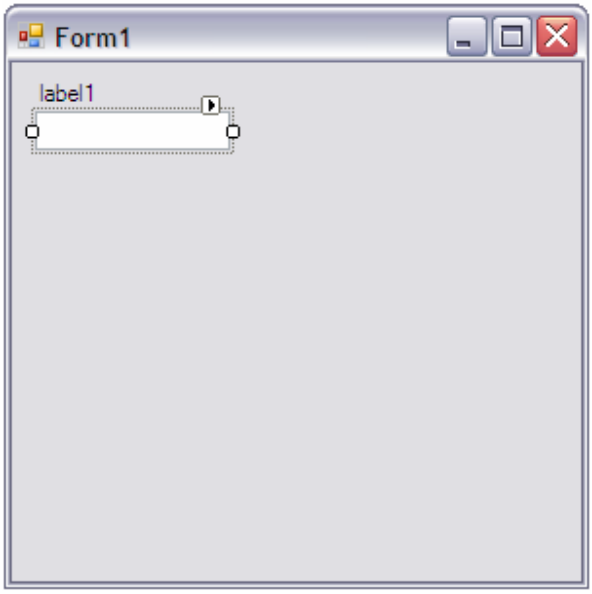
۴) بعد از رسم یک کنترل بر روی فرم، حداقل باید نام و متنی که نمایش می دهد را اصلاح کنید. با انتخاب کنترل `Label` بر روی فرم، مشاهده خواهید کرد که پنجره `Properties` در سمت چپ `Designer`، خاصیتهای `Label1` را نمایش می دهد. در پنجره `Properties` خاصیت `Text` این کنترل را به `Enter Your Name` تغییر دهید. توجه کنید که با فشار کلید `Enter` و یا کلیک در خارج از خاصیت مورد نظر، اندازه لیبل به صورتی تغییر میکند تا متن شما را در خود جای دهد. حالا، خاصیت `Name` کنترل را به `lblName` تغییر دهید.



شکل ۱-۱۲

۵) حالا، دقیقا زیر کنترل `Label`، یک کنترل `TextBox` قرار دهید تا در آن بتوانید نام را وارد کنید. برای اضافه کردن یک `TextBox` به فرم همانند لیبل عمل کنید، اما در این بار به جای لیبل، کنترل `TextBox` را از جعبه ابزار انتخاب کنید. بعد از اینکه `TextBox` را در جای خود بر روی فرم قرار دادید (همانند شکل ۱-۱۳) با استفاده از پنجره `Properties` خاصیت `Name` آن را به `txtName` تغییر دهید.

به دستگیره های تنظیم اندازه در سمت چپ و راست کنترل توجه کنید. به وسیله آنها، میتوانید اندازه افقی کنترل را تغییر دهید.

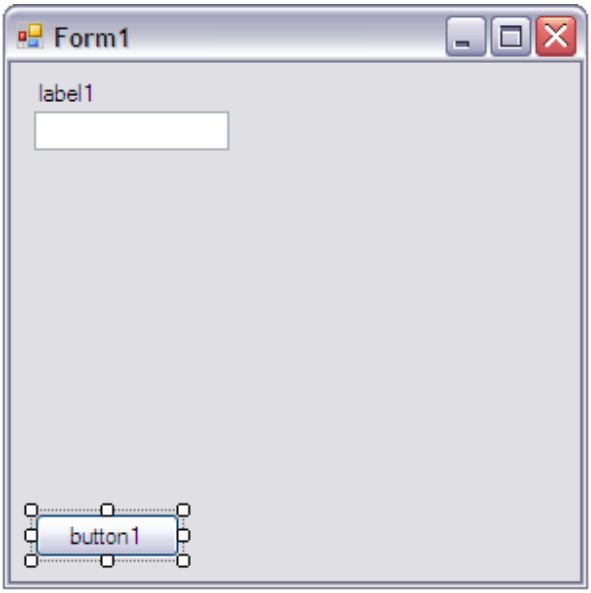


شکل ۱-۱۳

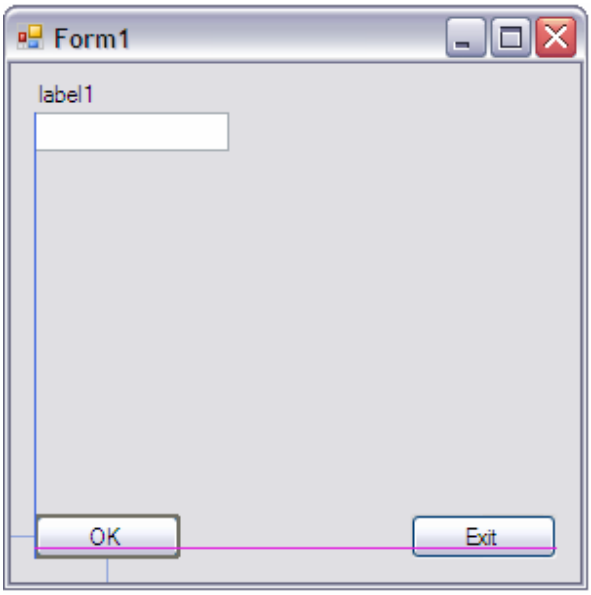
۶) در سمت چپ پایین فرم، به همان صورت که `Label` و یا `TextBox` را بـر روی فـرم قـرار دادیـد، یـک کنتـرل `Button` هم اضافه کنید. سپس خاصیت `Name` آن را بـه `btnOK` و خاصـیت `Text` آن را بـه `&OK` تغییـر هید. فرم شما هم اکنون باید مشابه شکل ۱-۱۴ باشد.

کاراکتر `&` که در خاصیت `Text` دکمه های فرمان استفاده می شود، برای ایجاد شـورت کـات بـرای آن دکمـه اسـت. حرفی که کاراکتر `&` قبل از آن قرار می گیرد، به صورت زیر خـط دار نمـایش داده میـشود(هماننـد شـکل ۱-۱۴). بـدین ترتیب کاربر میتواند به جای کلیک کردن با ماوس بر روی دکمه، با فشار کلید `Alt` و حرف مشخص شـده کلیـد مـورد نظر را انتخاب کند (در بعضی مواقع، تا کاربر کلید `Alt` را فشار ندهد، حروف مورد نظر زیر خط دار نمـی شـوند). بـرای مثال، در این جا فشار دادن کلید `Alt + O` همانند کلیک کردن بر روی دکمه `OK` است. برای انجام دادن این کـار لازم نیست که شما کدی را وارد کنید.

۷) حالا دکمه دوم را همانند دکمه اول، با کشیدن از جعبه ابزار و رها کردن بر روی فرم، در گوشه سـمت راسـت پـایین فـرم قرار دهید. دقت کنید، به محض اینکه دکمه مورد نظر را به گوشه سمت راست فرم ببرید، یک خط افقی آبی رنگ، همانند شکل ۱-۱۵، ظاهر میشود. این خط به شما اجازه می هد که مکان کنترل جدید را، با کنترلهای موجود در فرم تـراز کنیـد. به وسیله این خط می وانید کنترلهای جدید را دقیقاً در سمت چپ، راست، بالا و یا پایین یک کنترل خاص قرار دهید. بـه وسیله خط آبی کمرنگ کنار کنترل، می توانید یک فاصله خاص را همواره بین لبه فرم خود و لبه کنترلهای موجود در فرم رعایت کنید. خاصیت `Name` کنترل جدید را به `btnExit` و خاصیت `Text` آن را بـه `E&xit` تغییـر دهیـد. فرم شما هم اکنون باید چیزی مشابه شکل ۱-۱۵ باشد.



شکل ۱-۱۴



شکل ۱-۱۵

خوب، قبل از اینکه اولین برنامه را تمام کنید، بهتر است مقداری کد که باید در این برنامه استفاده کنید را به اختصار توضیح دهیم.

نشانه گذاری مجارستانی تغییر یافته:

احتمالا متوجه شده اید که کنترل هایی که تا کنون ایجاد کرده ایم، ساختار نام جالبی دارند. تمامی آنها دارای یک پیشوند هستند که نوع کنترل را مشخص می کند. این کار باعث می شود که هنگام کد نویسی به راحتی نوع کنترلی که با آن کار می کنید را تشخیص

دهید. مثلاً فرض کنید در برنامه خود یک کنترل دارید که نام آن Name است، بدون هیچ پیشوندی از قبیل lbl یا txt. چطور می خواهید تشخیص دهید که این کنترل، یک کادر متنی (TextBox) است که نام را از کاربر دریافت می کند یا یک لیبل که یک عبارت مربوط به نام را در فرم نمایش می دهد؟ فرض کنید که در بخش "امتحان کنید" قبلی، کنترل لیبل را Name1 و کنترل TextBox را Name2 نام گذاری می کردیم، به این ترتیب مسلماً گیج می شدید. اگر بعد از چند ماه می خواستید کد را تغییر دهید، چطور میتوانستید کنترل ها را از هم تشخیص دهید؟

هنگامی که با چند برنامه نویس به صورت گروهی کار می کنید، این مورد که، استیل و قالب برنامه را به صورت ثابت و مشخصی نگه دارید مهم خواهد شد. یکی از عمومی ترین ساختارهای نام برای کنترل ها در برنامه نویسی به هر زبانی، توسط دکتر چارلز سیمونی به وجود آمد که قبل از پیوستنش به مایکروسافت برای شرکت XPARC[1] کار میکرد. او پیشوند های کوتاهی را به وجود آورده بود که برنامه نویسان با استفاده از آنها می توانستند به راحتی تشخیص دهند که هر متغییر چه نوع داده ای را نگهداری می کند. به دلیل اینکه دکتر سیمونی اهل مجارستان بود و این پیشوند ها هم مقداری مانند یک زبان خارجی می ماندند، نام "نشانه گذاری مجارستانی" بر روی این سیستم ماند. همچنین به این دلیل که سیستم معرفی شده برای زبانهای C و C++ به کار میرفت، ما این سیستم را در ویژوال C# ۲۰۰۵ "نشانه گذاری مجارستانی تغییر یافته" می نامیم. جدول زیر لیستی از پیشوندهایی است که به شما پیشنهاد می کنم در این کتاب از آنها استفاده کنید.

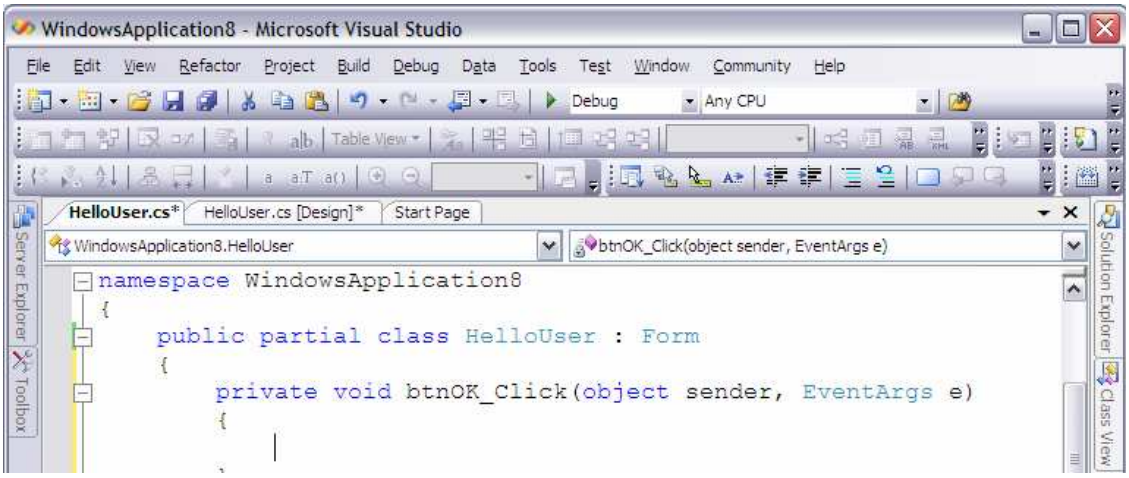
نشانه گذاری مجارستانی زمانی که در حال بررسی کد فرد دیگری هستید و یا کدی که خودتان چند ماه پیش نوشته اید را مطالعه می کنید، بسیار کار را سریع می کنند. اما گذشته از این مورد، بهترین فایده استفاده از این سیستم، ایجاد یکپارچگی در کد است. پیشنهاد می کنم از پیش فرض هایی که به صورت غیر رسمی در ویژوال C# ۲۰۰۵ به صورت استاندارد در آمده اند استفاده کنید، اما این کار واجب نیست. مهم این است که در طول برنامه خود از یک قاعده خاص برای نامگذاری پیروی کنید. تعدادی از این پیش فرض ها به همراه نام کنترل مربوط به آن در جدول زیر نمایش داده شده اند.

کنترل	پیشوند
دکمه فرمان (Button)	Btn
جعبه ترکیبی (ComboBox)	Cbo
جعبه انتخاب (CheckBox)	Chk
لیبل (Label)	lbl
جعبه لیست (ListBox)	lst
منوی اصلی (Menu)	mnu
دکمه رادیویی (RadioButton)	rdb
جعبه تصویر (PictureBox)	pic
جعبه متنی (TextBox)	txt

ویرایشگر کد:

[1] Xerox Palo Alto Research Center

حالا که فرم HelloUser را ایجاد کرده اید، باید مقداری کد به آن اضافه کنید تا کارهای مورد نظر را برایتان انجام دهد. تا کنون دیده اید که اضافه کردن یک کنترل به فرم تا چه حد ساده است. فراهم کردن یک کارایی خاص برای این کنترل ها به وسیله کد نیز، زیاد سخت تر از اضافه کردن کنترل به فرم نیست. برای اضافه کردن کد به کنترل مورد بحث، فقط کافی است که روی آن دوبار کلیک کنید. با این کار، صفحه ویرایشگر کد همانطور که در شکل ۱-۱۶ نشان داده شده است باز می شود.



شکل ۱-۱۶

دقت کنید که یک لبه دیگر به لبه های بالای صفحه اصلی در ویژوال استودیو اضافه شد. حالا در محیط ویژوال استودیو به دو پنجره **Design** و **Code** دسترسی دارید. برای طراحی ظاهر و رابط کاربری برنامه باید از قسمت Design، و برای نوشتن کد برنامه باید از قسمت Code استفاده کنید. توجه کنید که ویژوال استودیو ۲۰۰۵ برای کد مربوط به یک فرم، یک فایل مجزا ایجاد می کند. قسمتهای بصری و قسمتهایی که به ظاهر فرم مربوط هستند در فایلی به نام HelloUser.Designer.cs و کدهای مربوط به چگونگی عملکرد فرم در HelloUser.cs قرار می گیرند. این مورد خود یکی از دلایلی است که موجب راحتی برنامه نویسی با ویژوال C# ۲۰۰۵ می شود. با استفاده از قسمت Design می توانید ظاهر برنامه خود را طراحی کنید، سپس با استفاده از قسمت Code کد مربوط به فرم را بنویسید.

قسمت مهم دیگر در پنجره مربوط به کد، دو جعبه ترکیبی موجود در بالای صفحه است. به وسیله این دو می توانید به سرعت به قسمتهای مختلف فرم خود دسترسی داشته باشید. اشاره گر ماوس خود را بر روی جعبه ترکیبی سمت چپ ببرید و مقداری بر روی آن نگه دارید. راهنمایی ظاهر شده و میگوید که این کادر، مربوط به Types است. اگر این لیست را باز کنید، لیستی از تمامی کلاسهای موجود در فرم خود مشاهده خواهید کرد. اگر اشاره گر ماوس خود را بر روی جعبه ترکیبی سمت راست ببرید، راهنمای ظاهر شده به شما می گوید که این قسمت مربوط به Members است. اگر این لیست را باز کنید، نام تمام توابع و زیربرنامه هایی که در کلاس انتخاب شده در سمت چپ قرار دارند را خواهید دید. اگر فرم جاری محتوی مقدار زیادی کد است، به وسیله این قسمت میتوانید به راحتی بین توابع آن جا به جا شوید.

امتحان کنید: اضافه کردن کد به برنامه HelloUser

۱) برای شروع اضافه کردن کد به برنامه، بر روی قسمت Design در پنجره اصلی کلیک کنید تا بار دیگر قسمت طراحی را ببینید. سپس روی دکمه ی OK دو بار کلیک کنید. پنجره ی کد با کدی که در زیر نوشته شده است باز می

شود. این کد که به صورت اتوماتیک نوشته شده است، پوسته یا قالب رویداد Click برای کنترل Button است. در این قسمت میتوانید کدی را وارد کنید که با هر بار کلیک کردن روی این کنترل اجرا شود. این کد به عنوان **کنترل کننده ی رویداد**[1] و یا **زیر برنامه رویداد**[2] نامیده می شود. در فصلهای بعد با این موارد بیشتر آشنا خواهیم شد:

```
private void btnOK_Click(object sender, EventArgs e)
{

}
```

در کد بالا کلمات void و یا private نمونه ای از کلمات کلیدی در C# هستند. در اصطلاحات برنامه نویسی، **کلمات کلیدی**[3] کلماتی هستند که به ویژوال C# میگویند کارهای خاصی را انجام دهد. مثلاً در این جا، کلمه void به ویژوال C# می گوید که تابع تعریف شده هیچ مقداری را برنمی گرداند. همه ی کدهایی که شما در بین خطوط مربوط به باز شدن آکولاد ({) و بسته شدن آن (}) بنویسید، تابع رویداد مربوط به دکمه OK را تشکیل می دهند. در فصل سوم بیشتر در مورد این مباحث گفتگو خواهیم کرد.

۲) حالا کدهای مشخص شده در این قسمت را در تابع وارد کنید (در بین آکولادها بنویسید):

```
private void btnOK_Click(object sender, EventArgs e)
{
    //Display a message box greeting the user
    MessageBox.Show("Hello " + txtName.Text +
        "! Welcome to Visual C# 2005.",
        "Hello User Message");
}
```

نکته: به علت کمبود جا در این صفحه نمی توانیم تمام یک دستور را پشت سر هم بنویسیم، اما در محیط ویژوال استودیو، شما می توانید این کدها را پشت سر هم بنویسید. **در ویژوال C# یک خط کد، زمانی تمام میشود که علامت نقطه ویرگول (;) بعد از آن بیاید.** بنابراین میتوانید یک دستور را در چند خط بنویسید و تا زمانی که کاراکتر ; را وارد نکرده اید نیز دستور را ادامه دهید.

نکته: در طول کتاب، با قسمتهایی روبرو می شوید که باید کدهایی را در برنامه خود وارد کنید. معمولا هر جا که چنین موردی پیش بیاید، مکان دقیق وارد کردن کد را برای شما مشخص می کنم. کدهایی که با رنگ پیش زمینه خاکستری مشخص می شوند، کدهایی هستند که باید در برنامه وارد کنید.

۳) بعد از اینکه کد قسمت قبلی را وارد کردید، مجددا به قسمت Design برگردید و روی دکمه Exit دوبار کلیک کنید. کد مشخص شده در زیر را در تابع btnExit_Click وارد کنید.

```
private void btnExit_Click(object sender, EventArgs e)
{
```

---

[1] Event Handler
[2] Event Procedure
[3] Keywords

```
    //End the program and close the form
    this.Close();
}
```

احتمالا کلمه `this` برایتان جدید است. `this` یک کلمه کلیدی در C# است که به شیئ که در آن، در حال کدنویسی هستیم، اشاره می کند. در این جا چون کدهای نوشته شده مربوط به فرم `HelloUser` است، کلمه `this` به فرم `HelloUser` اشاره می کند.

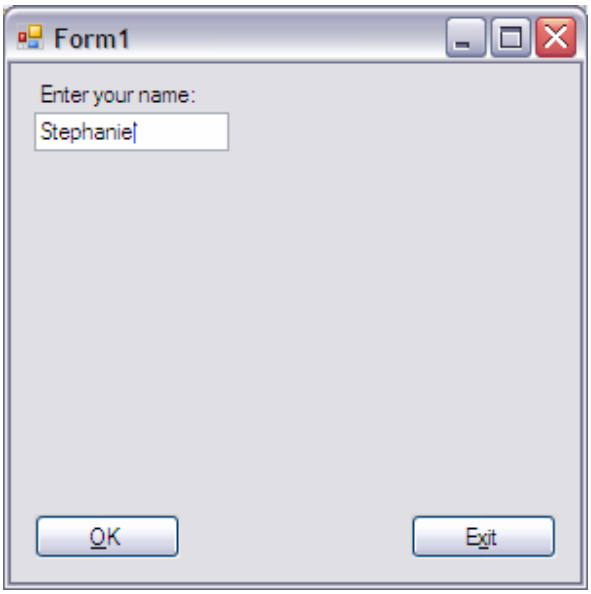
۴) حالا که کدنویسی برنامه به پایان رسید، زمان تست کردن آن شده است و می توانید ساخته خودتان را مشاهده کنید. ابتدا برنامه را از طریق منوی `File` ➔ `Save HelloUser.cs` و یا با استفاده از کلید `Save` روی نوار ابزار، ذخیره کنید.

۵) روی دکمه `Start` بر روی نوار ابزار (مثلث سبز رنگ) کلیک کنید. پنجره `Output` در پایین صفحه، انجام فعالیتهای زیادی را نمایش می دهد. اگر در وارد کردن کدهای برنامه هیچ خطایی به وجود نیاید، اطلاعات این پنجره فقط شامل اسم فایلهایی هستند که برای اجرای برنامه بارگذاری می شوند.

در این مرحله، به اصطلاح، ویژوال استودیو در حال **کامپایل** برنامه شما است. کامپایل کردن به مرحله ای گفته می شود که در آن، از کد ویژوال C# ۲۰۰۵ که توسط شما نوشته شده است، کدی ساخته میشود که توسط کامپیوتر قابل فهم باشد[1]. بعد از اینکه کامپایل برنامه شما با موفقیت به پایان رسید، ویژوال استودیو ۲۰۰۵ آن را اجرا می کند و می توانید نتیجه کار خود را مشاهده کنید.

اگر در مرحله کامپایل کردن، ویژوال C# ۲۰۰۵ با هر خطایی در کد مواجه شود، آن را به عنوان یک وظیفه در پنجره ی `Task List` نمایش می دهد. با دوبار کلیک کردن روی وظیفه مورد نظر در بخش `Task List` به قسمتی از کد که به آن مرتبط است، منتقل می شوید. در فصل ۱۱ در مورد خطا یابی برنامه ها و اصلاح آنها بیشتر یاد خواهیم گرفت.

۶) بعد از اینکه برنامه اجرا شد، صفحه اصلی آن نمایش داده می شود. یک نام را وارد کرده و روی کلید `OK` کلیک کنید (یا کلیدهای `Alt + O` را فشار دهید). (شکل ۱-۱۷)



[1] `Compile` – در این مورد در فصل دوم بیشتر توضیح خواهیم داد.

شکل ۱-۱۷

۷) پنجره ای که به کادر پیغام[1] معروف است، نمایش داده خواهد شد و به شخصی که نـام او در TextBox داخـل فـرم آمده است خوش آمد می گوید. (شکل ۱-۱۸)



شکل ۱-۱۸

۸) بعد از اینکه کادر پیغام را با کلیک کردن روی دکمه OK بستید، روی دکمه Exit بر روی فـرم کلیـک کنیـد. برنامـه بسته خواهد شد و شما به محیط ویژوال C# ۲۰۰۵ برخواهید گشت.

چگونه کار می کند؟

کدی که در رویداد Click برای دکمه OK وارد کرده اید نام کاربری را که در TextBox فرم وارد شده است دریافت کـرده و آن را به عنوان بخشی از پیغام همانند شکل ۱-۱۸ نشان می دهد.

خط اولی که در کد مربوط به این رویداد نوشته شده است، فقط یک توضیح است. این توضیح برای راهنمایی کردن برنامه نویسی که روی پروژه کار می کند و یا کسی که بعدها می خواهد کد برنامه را بخواند نوشته مـی شـود و توسـط کـامپیوتر خوانـده نمـی شـود. توضیحات در ویژوال C# با (//) مشخص می شوند و هر متنی که بعد از این دو کاراکتر وارد شود هنگام کامپایل نادیده گرفته می شود. در رابطه با اضافه کردن توضیحات در فصل سوم بحث شده است.

تابع MessageBox.Show یک پیغام را در صفحه نمایش می دهد. این تابع پارامترهای مختلفی را دریافت می کند. مـثلا در برنامه قبلی، یک رشته متنی را به این تابع فرستادید تا در صفحه نمایش دهد. رشته متنی شما از اتصال دو مقدار ثابت متنی که در علامت نقل قول (") قرار گرفته بود تشکیل می شد. برای اتصال چند رشته متنی به یکدیگر و ایجاد یک رشته طولانی در C# مـی توانید همانند کاری که برای جمع کردن اعداد انجام می دهید، از علامت + استفاده کنید.

در حقیقت کدی که بعد از خط توضیحات در برنامه قبلی آمده است، ثابت رشته ای "Hello " را با مقـدار خاصـیت Text مربوط به کنترل txtName جمع کرده و عبـارت "! Welcome to Visual C# 2005" را بـه رشـته حاصل اضافه می کند. پارامتر دومی که به متد MessageBox.Show فرستاده شده است، توسط تابع به عنوان متنی کـه باید در نوار عنوان پنجره نمایش داده شود استفاده می شود.

نکته دیگری که در این کد مهم است این است که یک دستور را در چند خط نوشته ایم. این کار هنگامی که می خواهیم یک دستور طولانی را وارد کنیم بسیار مفید است. همانطور که ذکر شد، در C# یک دستور زمانی تمام میشود که کاراکتر ; بعد از آن قرار گیرد. تا زمانی که کامپایلر به این کاراکتر برخورد نکرده باشد، تمام متن را به عنوان یک دستور در نظر می گیرد.

[1] MessageBox

```
private void btnOK_Click(object sender, EventArgs e)
{
    //Display a message box greeting the user
    MessageBox.Show("Hello " + txtName.Text +
        "! Welcome to Visual C# 2005.",
        "Hello User Message");
}
```

کد بعدی که وارد کردید، مربوط به رویداد Click برای دکمه Exit بود. در آنجا برای خروج از برنامه، به راحتی کد this.Close() را نوشتید. همانطور که قبلا هم توضیح داده شد، کلمه کلیدی this، به فرمی که هم اکنون در آن هستیم اشاره می کند. متد Close از فرم جاری، باعث می شود که فرم بسته شده و تمام منابعی که سیستم در اختیار آن قرار داده است آزاد شوند، بنابراین فرم (و در نتیجه برنامه) بسته می شود.

```
private void btnExit_Click(object sender, EventArgs e)
{
    //End the program and close the form
    this.Close();
}
```

احتمالاً قسمتهای زیادی از نوشته های بالا را متوجه نشده اید و یا برایتان نامفهوم است. اما اصلاً جای نگرانی نیست، در فصلهای بعدی تمام این موارد به تفصیل توضیح داده خواهند شد.

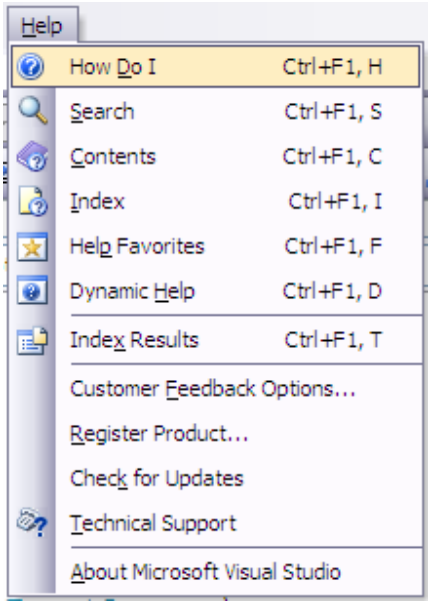
استفاده از سیستم راهنمای ویژوال استودیو:

سیستم راهنمایی که در ویژوال C# ۲۰۰۵ به کار رفته است، نسخه ی ارتقا یافته ی نسخه های قبلی این برنامه است. به تدریج که شروع به یادگیری ویژوال C# ۲۰۰۵ کنید، با این سیستم راهنما بیشتر آشنا خواهید شد. اما خوب است در اینجا به طور مختصر این سیستم را بررسی کنیم تا بتوانید سریعتر اطلاعات مورد نیاز خود را در آن پیدا کنید.

گزینه های منوی Help در شکل ۱-۱۹ نمایش داده شده اند.

همانطور که می بینید، این منو نسبت به منوی Help دیگر برنامه های ویندوزی گزینه های بیشتری دارد. دلیل این موضوع هم حجم زیاد مستنداتی است که در این برنامه وجود دارد. افراد کمی هستند که بیشتر موارد مورد نیاز خود را در .NET از حفظ باشند. اما خوشبختانه نیازی به حفظ بودن آنها نیست، چون می توانید به راحتی و به سرعت به سیستم راهنمای ویژوال استودیو مراجعه کنید.

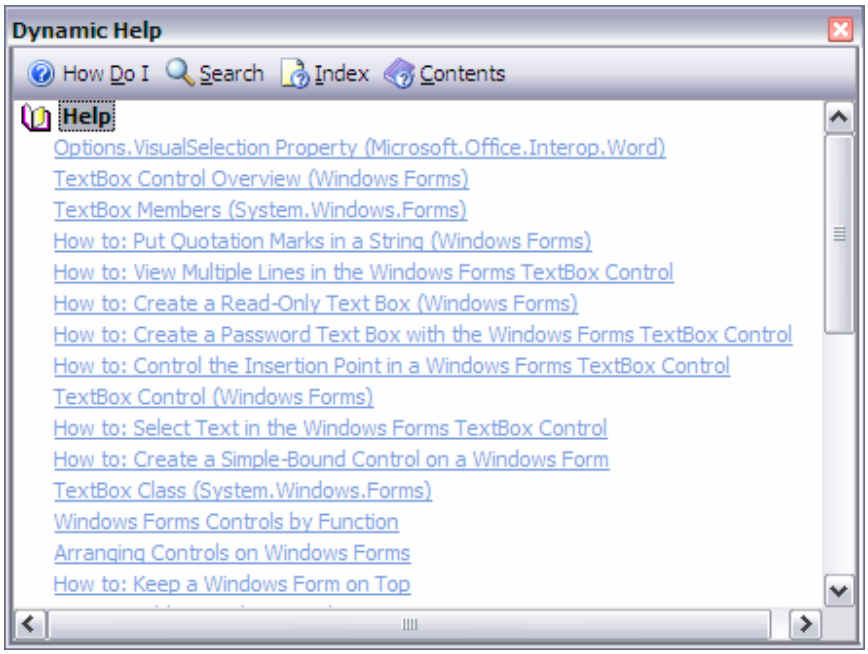
یکی از امکانات جالب راهنمای ویژوال استودیو، سیستم راهنمای دینامیک آن است. وقتی شما گزینه Dynamic Help را از منوی Help انتخاب می کنید، پنجره مربوط به راهنمای دینامیک باز می شود و لیستی از موضوعات مرتبط با کاری که در حال انجام آن هستید را به شما نمایش می دهد. پنجره راهنمای دینامیک از طریق انتخاب Help ➔ Dynamic Help در نوار منو قابل دسترسی است. با انتخاب آن پنجره Dynamic Help در کنار پنجره ی Properties در محیط ویژوال استودیو نمایش داده خواهد شد.



شکل ۱-۱۹

مثلاً، فرض می کنیم که شما در حال کار با یک کنترل `TextBox` هستید (برای مثال `TextBox`ای که در برنامه ی `Hello User` قرار دادیم) و می خواهید مقداری اطلاعات راجع به آن کسب کنید. کافی است که در فرم و یا در قسمت کد، `TextBox` را انتخاب کنید تا تمام موضوعات مرتبط با آن را در پنجره راهنمای دینامیک مشاهده کنید. (شکل ۱-۲۰)

گزینه های دیگر موجود در منوی `Help` (`Serach`، `Content` و `Index`) همانند دیگر برنامه های ویندوزی عمل میکنند. گزینه `How Do I` در این منو، لیستی از موارد موجود در راهنمای ویژوال استودیو که در آن کارهای عمومی دسته بندی شده اند را نمایش می دهد.



شکل ۱-۲۱

خلاصه:

خوشبختانه تا کنون متوجه شده اید که برنامه نویسی با ویژوال C# ۲۰۰۵ سخت نیست. مقداری محیط ویژوال استودیو را بررسی کرده اید و دیده اید که این محیط چگونه به شما کمک می کند تا به سرعت یک برنامه را طراحی کنید. دیدید که جعبه ابزار می تواند به شما در طراحی راحت و سریع رابطهای کاربری کمک بسیاری کند. به کمک پنجره Properties می توانید خاصیت های کنترل های فرم خود را به راحتی تغییر داده و به صورت دلخواه تنظیم کنید. به کمک پنجره Solution Explorer توانستید نمای درختی تمام فایلهای تشکیل دهنده ی پروژه را مشاهده کنید. حتی در این بخش مقداری کد نوشتید.
در فصلهای بعدی با جزییات بیشتری درگیر می شوید و بیشتر به کد نویسی عادت می کنید. قبل از اینکه زیاد وارد ویژوال C# ۲۰۰۵ بشویم، در فصل بعد چارچوب .NET را معرفی خواهیم کرد.
در پایان فصل باید با موارد زیر آشنا شده باشید:

- محیط توسعه مجتمع ویژوال استودیو یا IDE
- اضافه کردن کنترل به فرم در حالت طراحی
- تنظیم خاصیتهای کنترل های روی فرم
- اضافه کردن کد به کنترل ها در پنجره ی کد

تمرین:

برنامه ای بنویسید که شامل یک کنترل TextBox و یک Button باشد. با فشار داده شدن دکمه توسط کاربر، متنی که در TextBox نوشته شده است در یک کادر پیغام نمایش داده شود.

نکته: پاسخ این تمرین و دیگر تمرین ها، در انتهای کتاب در ضمیمه ی ۵ آورده شده است.

فصل دوم: چارچوب .NET و ارتباط آن با C#

در فصل قبل مقداری با زبان برنامه نویسی C# و محیط ویژوال استودیو آشنا شدیم. اما بهتر است قبل از اینکه درگیر برنامه نویسی با این زبان و کار با این محیط شویم، مقداری در رابطه با چارچوب .NET[1] و ارتباط آن با ویژوال استودیو و C# صحبت کنیم.
در این فصل ابتدا نگاهی کلی به تکنولوژی .NET خواهیم داشت و بعد از معرفی اجزای آن سعی می کنیم ارتباط آنها را با یکدیگر شرح دهیم. .NET هنوز یک تکنولوژی جدید محسوب می شود و دارای مباحث فنی زیادی است که فراگیری آنها در ابتدا کمی مشکل به نظر می رسد. مشکل بودن آن نیز به این علت است که .NET یک چارچوب یا فریم ورک است و یک فریم ورک، راه و روش جدیدی را برای طراحی و توسعه نرم افزار ارائه می دهد. در طول این فصل سعی می کنیم مفاهیم جدید ارائه شده در .NET را به طور خلاصه و اجمالی بررسی کنیم.
در این فصل:

- چارچوب .NET چیست؟
- چارچوب .NET چگونه کار می کند و چه چیزی باعث شده است که به یک فریم ورک پرطرفدار تبدیل شود؟
- با زبان C# چه برنامه هایی را می توان نوشت؟

چارچوب .NET چیست؟

قبل از هر چیز بهتر است که تعریف دقیقی از کلمات فریم ورک یا چارچوب و همچنین پلتفرم ارائه دهیم.

در تعریف .NET می توانیم بگوییم که: "چارچوب .NET یک پلتفرم جدید است که توسط مایکروسافت برای طراحی و توسعه نرم افزار ایجاد شده است."
نکته جالبی که در این تعریف وجود دارد ابهام زیادی است که در این تعریف به کار برده ام، اما برای این کار دلیل خوبی وجود دارد. برای شروع، توجه کنید که در این تعریف نگفته ام "طراحی و توسعه نرم افزار برای سیستم عامل ویندوز." اگرچه مایکروسافت چارچوب .NET را برای اجرا بر روی سیستم عامل ویندوز منتشر کرده است، به زودی نسخه های دیگری از این چارچوب را مشاهده خواهید کرد که بر روی سیستم عامل های دیگر مانند لینوکس نیز اجرا می شوند. یکی از این نسخه ها مونو[2] است. مونو یک نسخه متن باز از چارچوب .NET است (که شامل یک کامپایلر C# نیز هست) که برای سیستم عامل های گوناگونی مانند نسخه های مختلف لینوکس و مکینتاش منتشر شده است. پروژه های بسیار دیگری مشابه مونو در حال اجرا هستند که ممکن است هنگام انتشار این کتاب در اختیار شما قرار گرفته باشند. به علاوه می توانید با استفاده از نسخه فشرده این چارچوب به نام Microsoft .NET Compact Framework که زیر مجموعه ای از چارچوب .NET است برای وسایل هوشمند مانند دستیار دیجیتال شخصی[3] و یا موبایل ها نیز برنامه بنویسید (با این چارچوب در فصل بیست و دوم بیشتر آشنا خواهیم شد).
اگر به تعریفی که در بالا برای چارچوب .NET آورده شده است دقت کنید، مشاهده می کنید که این تعریف محدود به نوع خاصی از برنامه ها نیست. در حقیقت در مورد نوع برنامه هایی که می توان با .NET نوشت هیچ محدودیتی وجود ندارد که بخواهیم آن را

---

[1] .NET Framework
[2] Mono
[3] Personal Digital Assistant (PDA)

ذکر کنیم. از چارچوب .NET می توانید برای طراحی برنامه های تحت ویندوز، برنامه های تحت وب، سرویسهای مبتنی بر وب و ... استفاده کنید.

چارچوب .NET یک چارچوب کلی است و محدود به زبان برنامه نویسی خاصی نیست. شما می توانید برنامه خودتان را به هر زبانی که بخواهید بنویسید. در این کتاب برنامه نویسی به زبان C# را بررسی می کنیم، اما علاوه بر این زبان می توانید از زبانهایی مانند C++، ویژوال بیسیک، جاوا و حتی زبانهای قدیمی مانند COBOL نیز استفاده کنید. برای هر کدام از این زبانها یک کامپایلر خاص .NET ارائه می شود. به وسیله این کامپایلر، برنامه های نوشته شده به این زبانها نه تنها می توانند با چارچوب .NET ارتباط داشته باشند، بلکه می توانند با برنامه های زبانهای دیگر که تحت .NET نوشته شده اند نیز ارتباط داشته باشند. برای مثال یک برنامه که به زبان C# نوشته شده است به راحتی می تواند از کدی استفاده کند که به زبان ویژوال بیسیک نوشته شده است و یا برعکس.

مواردی که تا کنون گفتیم سطح بالای تنوع در .NET را نشان می دهند. این تنوع یکی از دلایلی است که باعث می شود چارچوب .NET چنین دورنمای جذابی داشته باشد.

پس دقت داشته باشید که .NET یک زبان برنامه نویسی، یک مدل برنامه نویسی مانند برنامه نویسی تحت ویندوز، یک نوع برنامه نویسی برای سیستم عاملی خاص مانند برنامه نویسی تحت ویندوز و یا مواردی از این قبیل نیست. بلکه .NET یک روش برای طراحی و توسعه ی نرم افزار است که به وسیله ی مایکروسافت معرفی شده است و می تواند در تمامی مواردی که در بالا ذکر شد مورد استفاده قرار گیرد.

چارچوب .NET از چه اجزایی تشکیل شده است؟

یکی از اجرای اصلی چارچوب .NET کتابخانه کلاس عظیم آن است که می توانید از آن در برنامه های خود استفاده کنید. **کتابخانه کلاس**[1] یک مجموعه از توابع و کلاسها است که برای انجام امور مختلف مورد استفاده قرار می گیرد. برای مثال یک کتابخانه کلاس، شامل توابعی برای کنترل ورودی و خروجی، استفاده از امکانات چاپ، کار با انواع مختلف شبکه ها و ... است. این توابع و کلاسها که با استفاده از تکنیکهای برنامه نویسی شیئ گرا نوشته شده اند[2]، در .NET به گروه ها و یا فضای نامهای مختلفی دسته بندی می شوند. با مفهوم فضای نام در فصل ۹ بیشتر آشنا خواهیم شد.

در نوشتن یک برنامه، می توانید هر کدام از فضای نامها را که نیاز داشتید به برنامه اضافه کنید. برای مثال یکی از این فضای نامها برای برنامه نویسی تحت ویندوز به کار می رود، یکی دیگر برای برنامه نویسی شبکه مورد استفاده قرار می گیرد، فضای نام دیگری برای برنامه نویسی تحت وب به کار می رود. بعضی از این فضای نامها خود به فضای نامهای کوچکتری تقسیم می شوند که برای کاربرد خاصی در آن قسمت استفاده می شوند. برای مثال فضای نام برنامه نویسی تحت وب شامل یک فضای نام کوچکتر است که برای نوشتن سرویسهای تحت وب به کار می رود.

باید توجه داشته باشید که تمام سیستم عامل ها، همه توابع موجود در این فضای نامها را پشتیبانی نمی کنند. برای مثال یک دستیار دیجیتال شخصی از توابع اصلی چارچوب .NET پشتیبانی می کند، اما یک سری از توابع که در این وسایل کاربردی ندارد به وسیله آنها پشتیبانی نمی شود.

بخش دیگری از چارچوب .NET، یک سری نوع های داده ای ابتدایی را تعریف می کند. نوع های داده ای برای نگهداری اطلاعات یک برنامه در طول اجرای آن مورد استفاده قرار می گیرند. نوع های داده ای که در این قسمت از .NET تعریف می شوند به صورت بسیار پایه ای هستند (مانند "عدد صحیح علامت دار ۳۲ بیتی"). نوع های داده ای پیشرفته تری که در زبانهای برنامه نویسی مبتنی بر .NET مانند C# و یا ویژوال بیسیک وجود دارند باید بر اساس یکی از این نوع های داده ای تعریف شده

---

[1] Class Library

[2] با این تکنیک ها در فصول ۹ و ۱۰ آشنا خواهید شد.

در این قسمت از چارچوب .NET باشند. این مورد باعث هماهنگی بین زبانهای برنامه نویسی می شود که از چارچوب .NET استفاده می کنند. این قسمت از چارچوب .NET، سیستم نوع داده ای عمومی و یا به اختصار **CTS**[1] نامیده می شود. با نوع های داده ای در فصل بعد بیشتر آشنا خواهیم شد.
علاوه بر کتابخانه کلاسی که ذکر شد، چارچوب .NET شامل بخشی به نام زبان عمومی زمان اجرا و یا به اختصار **CLR**[2] است. این بخش از چارچوب .NET (که مهمترین بخش آن نیز محسوب می شود) مسئول کنترل و مدیریت اجرای تمام برنامه هایی است که با استفاده از کتابخانه کلاس .NET نوشته شده اند[3].

چگونه با استفاده از چارچوب .NET، برنامه بنویسیم؟

نوشتن برنامه با استفاده از چارچوب .NET به معنی نوشتن کد به هر کدام از زبانهایی که توسط .NET پشتیبانی می شوند، با استفاده از کتابخانه کلاس .NET است. همانطور که گفتم در طول این کتاب از محیط طراحی مجتمع ویژوال استودیو (IDE) برای طراحی و برنامه نویسی استفاده می کنیم. مزیت استفاده از این محیط این است که می توانید به راحتی از ویژگیهایی که در بخشهای قبلی از چارچوب .NET معرفی کردیم استفاده کنید. کدی که شما برای نوشتن برنامه ها در طول این کتاب استفاده می کنید کلاً به زبان C# است، اما در طول برنامه ها از چارچوب .NET و همچنین از یک سری ویژگی ها و ابزارهایی که محیط ویژوال استودیو در اختیار ما قرار می دهد استفاده خواهیم کرد.
یک برنامه که به زبان C# نوشته شده است قبل از اجرا باید به کدی تبدیل شود که برای سیستم عامل قابل فهم باشد. به این کد، کد محلی[4] می گویند. تبدیل یک کد از هر زبانی به کد محلی که برای سیستم عامل قابل فهم باشد را **کامپایل** کردن می گویند و عملی است که به وسیله **کامپایلر** انجام می شود. در چارچوب .NET این بخش از دو مرحله تشکیل شده است.

MSIL و JIT:

هنگامی که برنامه ای که در آن از توابع موجود در کتابخانه کلاس .NET استفاده شده است را کامپایل می کنید، بلافاصله کد قابل فهم برای سیستم عامل و یا کد محلی تولید نمی شود. در عوض کد شما به زبانی به نام زبان سطح میانی مایکروسافت و یا به اختصار MSIL[5] تبدیل می شود. این کد برای سیستم عامل خاصی نیست و همچنین منحصر به زبان C# نیز نیست. به عبارت دیگر کد زبانهای دیگر نیز می تواند به MSIL تبدیل شود (و البته باید تبدیل شوند). کدهای زبانهای دیگری که از چارچوب .NET استفاده می کنند نیز (مانند ویژوال بیسیک) هنگام کامپایل ابتدا به زبان MSIL تبدیل می شوند. هنگام استفاده از ویژوال استودیو برای نوشتن برنامه، این مرحله از کامپایل توسط ویژوال استودیو انجام می شود.
اما برای اجرای یک برنامه توسط سیستم عامل یک مرحله دیگر نیز مورد نیاز است. این مرحله وظیفه ی کامپایلر Just-In-Time و یا به اختصار **JIT کامپایلر** است. این کامپایلر کد MSIL یک برنامه را دریافت کرده و آن را به کدی تبدیل می کند که به وسیله سیستم عامل قابل اجرا باشد. بعد از اینکه این تبدیل توسط JIT انجام شد، سیستم عامل می تواند برنامه را اجرا کند.

---

[1] Common Type System
[2] Common Language Runtime
[3] این مفاهیم به تفصیل و با ذکر جزئیات، در ضمیمه ی ۲ مورد بررسی قرار گرفته اند. البته مطالعه ی مطالب آن ضمیمه ممکن است برای بار اول مقداری مشکل به نظر برسد.
[4] Native Code
[5] Microsoft Intermediate Language

همانطور که از اسم این قسمت نیز مشخص است (Just-In-Time) کدهای زبان MSIL فقط هنگامی به زبان محلی قابل فهم برای سیستم عامل تبدیل می شوند، که بخواهند اجرا شوند.
در گذشته برای اینکه بتوانید برنامه خود را بر روی سیستم عامل های مختلف اجرا کنید نیاز داشتید که برای هر نسخه از سیستم عامل، آن کد را یک مرتبه به طور کامل کامپایل کنید. اما در چارچوب .NET نیازی به این کار نیست. زیرا برای هر نوع پردازنده و نیز هر نوع سیستم عامل یک نسخه از JIT وجود دارد. برنامه شما در هر سیستم عاملی که اجرا شود، کامپایلر JIT موجود در آن سیستم عامل، کد MSIL برنامه ی شما را که مستقل از سیستم عامل و نوع پردازنده است دریافت کرده و کد محلی مناسبی تولید می کنند که برای سیستم عامل قابل فهم باشد.
فایده استفاده از این روش در این است که وظیفه برنامه نویس را به شدت کاهش می دهد. در حقیقت می توان گفت که به عنوان برنامه نویس، هنگام نوشتن کد می توانید سیستم عاملی که قرار است برنامه روی آن اجرا شود را فراموش کرده و فکر خود را بر روی کد و منطق برنامه متمرکز کنید.

اسمبلی ها:

هنگامی که یک برنامه را کامپایل می کنید، کد MSIL تولید شده در فایلهایی به نام **اسمبلی**[1] ذخیره می شوند. فایلهای اسمبلی می توانند شامل برنامه هایی باشند که بدون نیاز به برنامه ای دیگر بتوانند بر روی سیستم عامل اجرا شوند (این گونه فایلها دارای پسوند .exe هستند) و یا شامل کتابخانه هایی از کلاسها و توابع برای استفاده در دیگر برنامه ها باشند (این گونه فایلها دارای پسوند .dll هستند).
فایلهای اسمبلی علاوه بر کدهای MSIL، شامل **اطلاعات متا**[2] (اطلاعاتی راجع به اطلاعات ذخیره شده در فایل اسمبلی) و همچنین منابع اختیاری (اطلاعات اضافی که به وسیله کدهای MSIL استفاده می شوند، همانند فایلهای صوتی و یا فایلهای تصویری) نیز هستند. اطلاعات متا باعث می شوند که یک فایل اسمبلی بتواند اطلاعات داخل خود را به طور کامل توصیف کند. به عبارت دیگر برای استفاده از یک اسمبلی به هیچ اطلاعات و یا کارهای اضافی مانند ثبت آن در رجیستری سیستم نیازی ندارید. به این ترتیب از مشکلاتی که عموماً هنگام کار با این نوع فایلها در محیطهای دیگر به وجود می آمد نیز جلوگیری می شود.
یکی دیگر از خاصیتهای این مورد در این است که توزیع یک نرم افزار به سادگی کپی کردن تمام فایلهای آن بر روی کامپیوتر مقصد است. به علت اینکه برای اجرای یک فایل اسمبلی به هیچ مورد دیگری نیاز نیست، می توانید به سادگی فولدر حاوی برنامه را بر روی کامپیوتر مقصد کپی کنید و سپس با کلیک کردن بر روی فایل اجرایی آن، برنامه را اجرا کنید و از آن استفاده کنید (با فرض اینکه CLR که مهمترین بخش .NET است، قبلا در آن کامپیوتر نصب شده باشد). در مورد چگونگی توزیع یک نرم افزار در فصل بیست و یکم بیشتر صحبت خواهیم کرد.
البته ممکن است در بعضی مواقع بخواهید از توابع موجود در یک فایل DLL، در چند برنامه استفاده کنید. برای این کار لازم نیست فایل مذکور را در فولدر تمام برنامه هایی که از آن استفاده می کنند قرار دهید. بلکه می توانید آن را یک بار در یک مکان مشخص قرار دهید و سپس تمام برنامه های که به آن نیاز دارند، از آن استفاده کنند. در چارچوب .NET این مکان مشخص که برای قرار گرفتن فایلهای اسمبلی عمومی در نظر گرفته شده است، Global Assembly Cache و یا GAC نام دارد. برای اینکه یک فایل اسمبلی را در این قسمت قرار دهید، کافی است به سادگی فایل مورد نظر را در فولدر مشخص شده برای GAC کپی کنید، زیرا برنامه ها می توانند علاوه بر اسمبلی های خود به همه اسمبلی های موجود در این فولدر نیز دسترسی داشته باشند.

---

[1] مفهوم اسمبلی در این کتاب کاملا با زبان برنامه نویسی اسمبلی تفاوت دارد.

[2] Metadata

کدهای مدیریت شده:

همانطور که گفتم برنامه ی شما هنگام کامپایل ابتدا به کد MSIL تبدیل می شود، سپس این کد قبل از اجرا به وسیله JIT به کد محلی تبدیل شده و کد محلی به وسیله سیستم عامل اجرا می شود. تمام این قسمتها بخشی از وظایف CLR است، اما وظایف CLR به این موارد ختم نمی شود. برنامه ای که به وسیله .NET نوشته شده است در طول زمان اجرا توسط CLR مدیریت می شود. به عبارت دیگر در طول اجرای برنامه های نوشته شده با .NET، CLR مسئول کنترل امنیت آنها، مدیریت حافظه برنامه ها، کنترل بخشهای خطا یابی در برنامه ها و ... است. به همین دلیل به برنامه هایی که با .NET نوشته شده اند، برنامه های مدیریت شده می گویند. در مقابل، برنامه هایی که تحت کنترل CLR اجرا نمی شوند به برنامه های مدیریت نشده[1] معروف هستند و زبانهای مشخصی مانند C++ می توانند چنین برنامه هایی را تولید کنند. از کدهای مدیریت نشده بیشتر در مواقعی استفاده می شود که قابلیت استفاده از کدهای مدیریت شده نباشد، همانند فراخوانی توابع سطح پایین سیستم عامل. البته با استفاده از زبان C# نمی توان کدهای مدیریت نشده تولید کرد و تمام کدهای تولید شده به وسیله کامپایلر C# تحت کنترل CLR اجرا می شوند.

مدیریت حافظه در .NET:

یکی از مهمترین ویژگیهای کدهای مدیریت شده، بخش مدیریت حافظه در این نوع کدها است که به وسیله سیستمی به نام Garbage Collection و یا به اختصار GC انجام می شود. چارچوب .NET با استفاده از این سیستم می تواند اطمینان حاصل کند که حافظه ای که به یک برنامه اختصاص داده می شود، با پایان برنامه به طور کامل بازیابی می شود. در زبانهای برنامه نویسی قبل از .NET، این مورد به وسیله برنامه نویس کنترل می شد و امکان داشت که با یک اشتباه کوچک در کد برنامه، مقدار زیادی از فضای حافظه غیر قابل استفاده بماند و برنامه با کمبود حافظه مواجه شود. این گونه مشکلات باعث کاهش سرعت برنامه ها و حتی در بعضی شرایط باعث توقف سیستم می شد.
نحوه کار GC در .NET به این صورت است که در زمانهای مشخصی به بررسی حافظه می پردازد و داده هایی را که دیگر استفاده نمی شوند از حافظه پاک می کند. البته بررسی حافظه توسط GC در فاصله های زمانی ثابت صورت نمی گیرد بلکه ممکن است در شرایطی در هر ثانیه چندین هزار بار اجرا شود و در شرایط دیگر در هر چند ثانیه یک بار اجرا شود[2].

مراحل اجرای برنامه در .NET

قبل از ادامه، مراحل لازم برای ایجاد یک برنامه با .NET را که در قسمتهای قبلی توضیح داده شد جمع بندی می کنیم:

۱) کد برنامه به وسیله یکی از زبانهای سازگار با .NET مانند C# نوشته می شود (شکل ۲-۱):

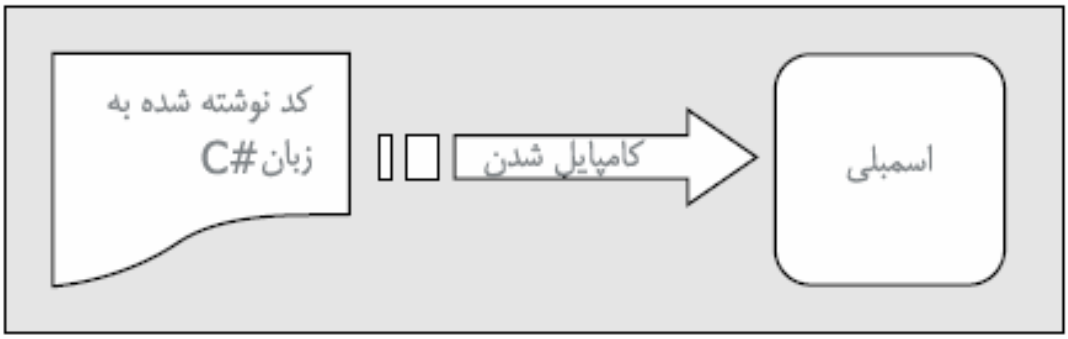
[1] Unmanaged Code

[2] برای آشنایی بیشتر با سیستم مدیریت حافظه در .NET به ضمیمه ی ۴ مراجعه کنید.



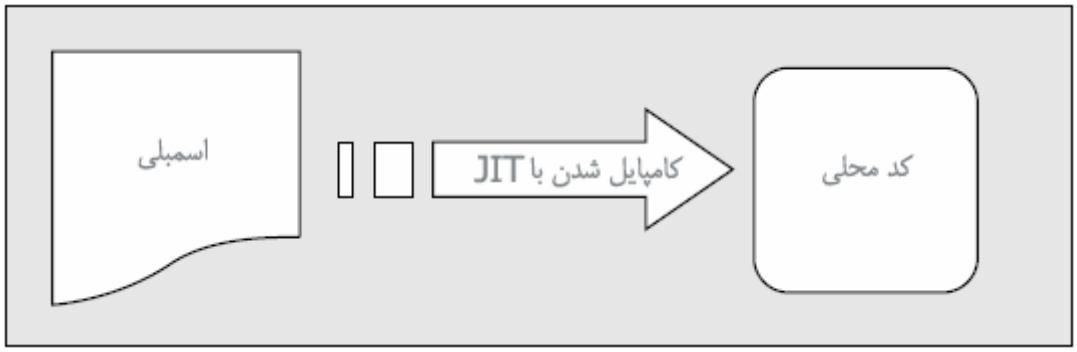
شکل ۱-۱

۲) این کد به زبان MSIL کامپایل می شود و سپس در یک فایل اسمبلی ذخیره می شود (شکل ۲-۲):



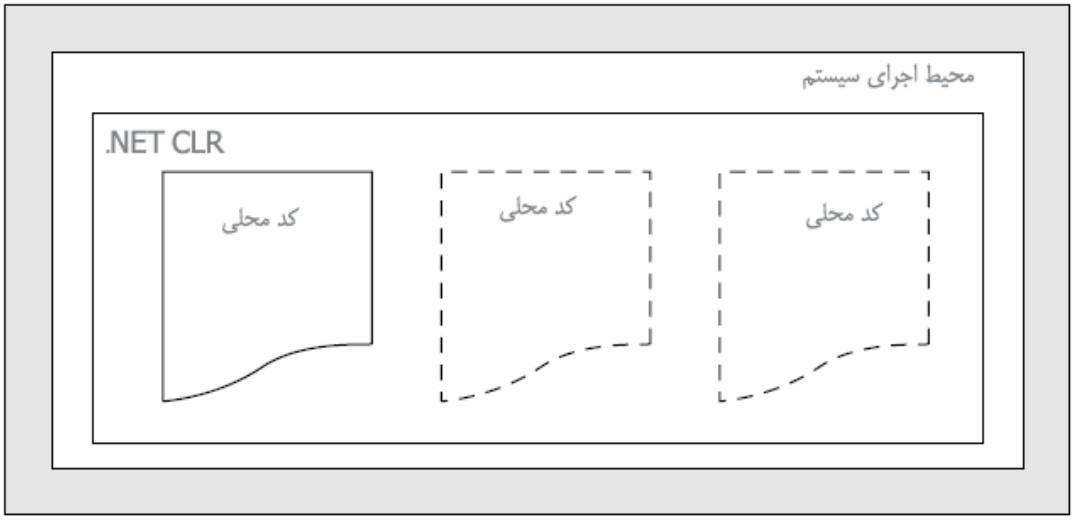
شکل ۲-۲

۳) هنگامی که کد بخواهد اجرا شود (چه خود فایل اجرایی باشد و به تنهایی اجرا شود، چه یک فایل حاوی توابع مورد استفاده باشد و به وسیله دیگر برنامه ها احضار شود) ابتدا باید به وسیله یک کامپایلر دیگر به کد محلی تبدیل شود. ایـن کامپـایلر JIT نام دارد (شکل ۲-۳):



شکل ۲-۳

۴) کد محلی تولید شده به وسیله JIT به همراه دیگر برنامه های در حال اجرا که به وسیله .NET نوشته شده اند، تحت کنترل CLR به اجرا در می آید (شکل ۲-۴):



شکل ۲-۴

لینک دادن:

در تکمیل گفته های قبلی، فقط یک بخش دیگر باقی مانده است. کد C# یک برنامه که در اولین مرحله به زبان MSIL تبدیل می شود حتماً نباید در یک فایل باشد، بلکه می توانیم برنامه را در چندین فایل سورس کد قرار دهیم و سپس آنها را در یک فایل اسمبلی کامپایل کنیم. به این عمل لینک کردن گفته می شود که در برنامه ها کاربرد زیادی دارد. فایده این روش در این است که معمولا برای برنامه نویس کار با چند فایل کوچک راحت تر از کار با یک فایل بزرگ است. برای مثال می توانید سورس یک برنامه را به چندین فایل مجزا تقسیم کنید و سپس به طور جداگانه بر روی هر یک از آنها کار کنید. به این ترتیب در مواقع مورد نیاز، پیدا کردن قسمت خاصی از کد نیز بسیار راحت تر خواهد شد.
یکی دیگر از قابلیتهای این روش در این است که گروههای برنامه نویسی می توانند یک برنامه را به چند قسمت تقسیم کنند. به این ترتیب هر کدام از برنامه نویسان می توانند بر روی یک قسمت خاص کار کند بدون اینکه در مورد نحوه پیشرفت قسمتهای دیگر نگران باشند.

C# چیست؟

همانطور که در قسمتهای قبلی نیز ذکر شد C# یکی از زبانهای برنامه نویسی است که به وسیله آن می توان برنامه هایی با قابلیت اجرا در .NET CLR تولید کرد. زبان C# در حقیقت نسخه کامل شده ی زبانهای برنامه نویسی C و C++ است که به وسیله مایکروسافت برای کار با چارچوب .NET به وجود آمده است. با توجه به جدید بودن این زبان برنامه نویسی، در ایجاد آن سعی شده است که از ویژگیهای خوب زبانهای برنامه نویسی دیگر الهام گرفته شود و نیز کاستی های آن زبانها برطرف شود.

ایجاد یک برنامه در محیط C# بسیار راحت تر از ایجاد یک برنامه در محیط C++ است. علاوه بر این سادگی، C# زبان قدرتمندی نیز محسوب می شود به نحوی که اغلب کارهایی که در C++ امکان پذیر است در C# هم می توان انجام داد. بعضی از ویژگیهای C# که هم سطح با ویژگیهای پیشرفته در C++ هستند، همانند قابلیت دسترسی مستقیم به حافظه و نیز تغییر آن، باعث می شوند که کدهای یک برنامه به عنوان کد نا امن در نظر گرفته شود. استفاده از این تکنیک های پیشرفته ی برنامه نویسی، عموماً خطرناک هستند زیرا ممکن است باعث شوند قسمتهای مهم حافظه که اطلاعات سیستم عامل در آن قرار دارد به طور نا خواسته تغییر کند و سیستم متوقف شود. به همین دلیل این مباحث در این کتاب مورد بررسی قرار نمی گیرند.
بعضی مواقع کدهای زبان C# طولانی تر از کدهای زبان C++ هستند. علت این طولانی تر بودن کدها به خاطر این است که C# بر خلاف C++ یک زبان **نوع-امن**[1] است. در اصطلاح این لغت به معنی این است که هنگامی که نوع داده ای یک متغییر مشخص شد، آن متغییر نمی تواند به یک نوع داده ای دیگر که به آن مرتبط نیست تبدیل شود[2]. علاوه بر این مورد یک سری محدودیتهای دیگر نیز هنگام تبدیل یک نوع داده ای به نوع داده ای دیگر نیز وجود دارد که باعث می شود کدهای C# طولانی تر از کدهای C++ شوند، اما در مقابل کدهای C# از پایداری بیشتری برخوردارند و نیز خطا یابی در آنها ساده تر است.
البته C# فقط یکی از زبانهای برنامه نویسی است که برای طراحی برنامه تحت .NET ایجاد شده است، اما به نظر من مطمئناً بهترین زبان برای این کار است. یکی از دلایل این امر این است که زبان C# از پایه برای استفاده در محیط .NET ایجاد شده است و معمولاً در پروژه هایی که در رابطه با انتقال .NET به سیستم عامل های دیگر است، مانند Mono از این زبان استفاده می کنند. در زبانهای دیگر، مانند نسخه .NET زبان ویژوال بیسیک، برای اینکه شباهت با نسلهای قبلی خود را حفظ کنند، یکسری از قسمتهای CLR پشتیبانی نمی شود. در مقابل با استفاده از زبان C# می توان از تمام ویژگیهای ارائه شده به وسیله .NET در برنامه استفاده کرد.

چه نوع برنامه هایی را میتوان با استفاده از C# انجام داد؟

همانطور که گفتم، در .NET هیچ محدودیتی برای نوع برنامه های قابل اجرا وجود ندارد. زبان C# نیز از چارچوب .NET استفاده می کند، بنابراین هیچ محدودیتی در نوع برنامه هایی که می توان با این زبان انجام داد وجود ندارد. اما بیشتر برنامه هایی که با C# نوشته می شوند جزء یکی از دسته برنامه های زیر هستند:

- **برنامه های مبتنی بر ویندوز:** این نوع برنامه ها همانند Office برنامه هایی هستند که دارای ظاهر آشنای برنامه های ویندوزی هستند. این نوع برنامه ها به وسیله فضای نام مربوط به برنامه های ویندوزی در چارچوب .NET نوشته می شوند. این فضای نام شامل کنترل هایی از قبیل دکمه های فرمان، نوار ابزارها، منو ها و ... است که به وسیله آنها می توان رابط گرافیکی برنامه را طراحی کرد
- **برنامه های مبتنی بر وب:** این نوع برنامه ها شامل یک سری صفحات وب هستند که ممکن است تاکنون به وسیله مرورگر های مختلف اینترنت آنها را مشاهده کرده باشید. چارچوب .NET دارای یک سیستم قوی برای ایجاد اتوماتیک صفحات وب و تامین امنیت آنها و ... است. این سیستم ASP.NET[3] نامیده می شود و شما می توانید با استفاده از زبان C# و سیستم ASP.NET برنامه هایی مبتنی بر وب ایجاد کنید.
- **سرویسهای وب:** وب سرویس ها یک روش جدید و جالب برای ایجاد برنامه های توزیع شدنی مبتنی بر وب هستند. با استفاده از وب سرویس ها می توانید هر نوع اطلاعاتی را از طریق اینترنت بین برنامه ها منتقل کنید. در این مورد زبان

---

[1] Type-Safe

[2] در مورد نوع های داده ای و تبدیل آنها به یکدیگر در فصل بعد صحبت خواهیم کرد.

[3] Active Server Pages .NET

مورد استفاده در برنامه و یا سیستم عاملی که برنامه در آن اجرا می شود اهمیتی ندارد. به عبارت دیگر برنامه ی شما که تحت .NET و سیستم عامل ویندوز نوشته شده است می تواند با برنامه های دیگر که تحت سیستم عامل های دیگر عمل می کند تبادل اطلاعات داشته باشد.

در هر کدام از این برنامه هایی که در بالا ذکر شد ممکن است به دسترسی به یک بانک اطلاعاتی نیاز پیدا کنید. برای این منظور در .NET باید از سیستمی به نام ADO.NET[1] استفاده کنید.

ویژوال استودیو ۲۰۰۵ :

در طراحی برنامه های .NET استفاده از ویژوال استودیو موردی ضروری نیست. اما با استفاده از آن سرعت طراحی برنامه ها افزایش شدیدی پیدا می کند. برای نوشتن برنامه با استفاده از C# تحت .NET می توانید حتی از یک ویرایشگر ساده ی متن مانند Notepad نیز استفاده کنید و سپس با استفاده از کامپایلر خط-فرمان[2] .NET برای C#، آن را به یک برنامه قابل اجرا تبدیل کنید.
در زیر ویژگیهایی از ویژوال استودیو .NET که باعث می شود این محیط انتخابی مناسب برای برنامه نویسی تحت .NET محسوب شود را بررسی خواهیم کرد:

- ویژوال استودیو تمام مراحل کامپایل یک سورس کد به یک برنامه قابل اجرا را به صورت اتوماتیک انجام می دهد و همچنین به برنامه نویس اجازه می دهد هر قسمتی را که بخواهد تغییر داده و تنظیم کند.
- ویرایشگر کد ویژوال استودیو برای کد نویسی زبان های پشتیبانی شده در .NET بسیار هوشمند است و می تواند هنگام نوشتن این کدها خطاهای آنها را تشخیص داده و در تصحیح آنها به برنامه نویس کمک کند.
- ویژوال استودیو شامل محیطهایی برای طراحی برنامه های ویندوزی و نیز برنامه های مبتنی بر وب است که به کمک آنها می توانید به سادگی محیط برنامه خود را طراحی کنید.
- برای اینکه در .NET یک برنامه تحت ویندوز ایجاد کنید باید مقدار زیادی کد را که در اغلب برنامه ها به صورت تکراری هستند بنویسید. این مورد در ایجاد برنامه هایی از نوع های دیگر مانند برنامه های تحت وب نیز وجود دارد. ویژوال استودیو با نوشتن اتوماتیک این کدها در سرعت بخشیدن به طراحی برنامه ها کمک قابل توجهی می کند.
- ویژوال استودیو دارای ویزارد های زیادی است که بسیاری از کارهای عمومی را برای شما انجام داده و کد مربوط به آنها را در برنامه قرار می دهد. به این ترتیب دیگر نیازی نیست در مورد نحوه نوشتن کد آنها نگران باشید.
- ویژوال استودیو دارای ابزارهای قدرتمندی برای کنترل قسمتهای مختلف یک پروژه از قبیل سورس کدهای C# و یا فایلهای مورد نیاز برنامه از قبیل فایلهای صوتی و تصویری است.
- علاوه بر سادگی طراحی برنامه ها در .NET، توزیع آنها نیز بسیار ساده است و به راحتی می توان آن را بر روی کامپیوترهای مقصد اجرا کرد و یا به روز رساند.
- ویژوال استودیو دارای ابزارهای قوی خطا یابی در برنامه است. برای مثال با استفاده از این ابزارها می توان برنامه را خط به خط اجرا کرد و در اجرای هر خط موقعیت برنامه را بررسی کرد.

---

[1] Active Data Objects .NET
[2] Command-Line Compiler

ویژگیهای ویژوال استودیو بسیار بیشتر از موارد ذکر شده است اما همین مقدار برای نمایش مزایای استفاده از آن کافی بـه نظـر مـی رسد.

راه حلهای ویژوال استودیو:

هنگامی که بخواهید یک برنامه را با ویژوال استودیو بنویسید ابتدا باید یک **راه حل**[1] ایجاد کنید. یک راه حـل در اصـطلاح ویـژوال استودیو، از بیش از یک پروژه تشکیل می شود. راه حل ها می توانند شامل چندین پروژه از انواع مختلف باشـند. بـرای مثـال تـصور کنید می خواهید برنامه ای برای یک شرکت تجاری بنویسید که از دو قسمت تشکیل می شود: در قسمت اول باید یک برنامه تحـت ویندوز ایجاد کنید که امور مختلف آن شرکت را کنترل کند، در قسمت دوم نیز باید یک برنامه ی تحت وب ایجاد کنید تـا اطلاعـات مربوط به آن شرکت را در یک وب سایت نمایش دهد. برای هر کدام از این قسمتها به یک پروژه ی مجزا نیاز دارید. بنابراین در کل باید دو پروژه ایجاد کنید. برای در یک گروه قرار دادن این دو پروژه می توانید از راه حل ها استفاده کنید. به این ترتیـب مـی توانیـد کدهای مرتبط به هم را در یک جا گروه بندی کنید، حتی اگر پروژه های آنها در قسمتهای مختلـف هـارد دیـسک باشـند و هنگـام کامپایل چندین فایل اسمبلی مختلف در قسمتهای متفاوت هارد دیسک ایجاد شود.
با استفاده از این ویژگی می توانید همزمان بر روی کدهایی که بین برنامه ها مشترک است (بـرای مثـال اسـمبلی هـای موجـود در GAC) و نیز برنامه اصلی کار کنید. همچنین به علت این که برای کار بر روی این پروژه ها از یک محیط طراحی استفاده می کنید، خطا یابی آنها نیز بسیار ساده تر خواهد شد.

نتیجه:

در این فصل به معرفی .NET پرداختیم و دیدیم که چگونه می تـوان بـا اسـتفاده از .NET بـه سـادگی برنامـه هـای متنـوع و قدرتمندی را نوشت. مشاهده کردید که برای اجرای یک برنامه نوشته شده به زبان C# در .NET چه مراحلی باید طی شوند و نیز مزیت های استفاده از کدهای مدیریت شده در .NET CLR نسبت به کدهای مدیریت نشده چیست.
در پایان این فصل باید با موارد زیر آشنا شده باشید:

- چارچوب .NET چیست، چرا به وجود آمد و چه مواردی باعث قدرت و جذابیت آن می شود؟
- زبان C# چیست و چه چیز باعث می شود که به یک زبان برنامه نویسی مناسب تحت .NET تبدیل شود؟
- ویژوال استودیو ۲۰۰۵ چیست و چگونه می تواند در تسریع نوشتن برنامه در .NET موثر واقع شود؟

[1] Solution

فصل سوم: نوشتن نرم افزار

حالا که توانستید ویژوال C# ۲۰۰۵ را اجرا کرده و حتی یک برنامه کوچک ولی قابل اجرا با آن بنویسید، بهتر است در این فصل مبانی فرآیند نوشتن یک نرم افزار را یاد بگیرید و کدهای خودتان را در کنار هم قرار دهید و برنامه های جذابی طراحی کنید.
در این درس:

- در رابطه با الگوریتم ها مطالبی را خواهید آموخت.
- چگونگی استفاده از متغییر ها را خواهید دید.
- با انواع مختلف داده ها از قبیل اعداد صحیح، اعشاری، رشته ها و تاریخ ها آشنا خواهید شد.
- با دامنه فعالیت متغییر ها آشنا می شوید.
- با خطا یابی در برنامه ها آشنا خواهید شد.
- چگونگی ذخیره شدن داده ها در حافظه کامپیوتر را مشاهده خواهید کرد.

داده ها و اطلاعات:

اطلاعات به توضیحاتی گفته میشود که راجع به واقعیتی بیان می شود. این اطلاعات میتوانند در هر قالبی جمع آوری و یا ارائه شوند، به گونه ای که برای درک توسط کامپیوترها و یا انسانها مناسب باشد. مثلا اگر شما چند نفر را برای بررسی وضعیت ترافیک به چندین چهارراه مختلف بفرستید، در پایان کار با چند گزارش دست نویس که وضعیت عبور ماشینها را در چهارراه های مختلف بیان می کند روبرو می شوید. این گزارشات را میتوان اطلاعاتی در نظر گرفت که برای انسان قابل فهم هستند.
به اطلاعات جمع آوری شده، مرتب شده و قالب بندی شده، به نحوی که توسط قسمتی از برنامه های کامپیوتری قابل استفاده باشد، داده می گویند. اطلاعاتی که شما دارید (چندین دفتر پر از متنهای دست نویس) به وسیله نرم افزارهای کامپیوتری قابل استفاده نیستند. برای تبدیل آنها به داده های قابل استفاده توسط کامپیوتر باید چندین نفر روی آنها کار کنند و قالب آن را تغییر دهند.

الگوریتم ها:

صنعت کامپیوتر به این معروف است که با سرعتی باورنکردنی در حال تغییر است. بیشتر متخصصان این رشته در تمام دوران فعالیت خود، دائما در حال آموزش و یادگیری هستند تا اطلاعات خود را به روز نگه دارند. اما بعضی از قسمتهای کامپیوتر، از زمانی که به وجود آمده اند تا کنون تغییری نکرده اند و احتمالا در طول عمر ما هم تغییری نخواهند کرد. فرآیند و ترتیب موجود در توسعه نرم افزار نمونه ای از جنبه های تکنولوژی کامپیوتر است که ماهیت اصلی آن از ابتدا تاکنون دچار تغییر نشده است.
برای کار کردن یک نرم افزار، یک سری داده نیاز است که روی آنها پردازش انجام شود. نرم افزار این داده ها را دریافت میکند و به فرمی دیگر تبدیل میکند و ارائه می دهد. برای مثال، یک برنامه بانک اطلاعاتی، اطلاعات مشتریان شما را که به صورت صفر و یک نوشته شده است دریافت می کند و آنها را برای شما در مانیتور نمایش می دهد. یا سرویس تلفن شما، مدت زمان تماسهای شما را ذخیره کرده و صورت حساب ها را بر اساس این اطلاعات تولید می کند.
اما اساس فعالیت همه این نرم افزارها، الگوریتم آنها است. قبل از اینکه شما بتوانید برنامه ای بنویسید تا مسئله ای را حل کند، ابتدا باید آن را به چند مسئله کوچکتر تقسیم کنید، و چگونگی حل این مسایل را قدم به قدم توضیح دهید. الگوریتم برنامه ها کاملا

مستقل از زبان برنامه نویسی است که برای نوشتن برنامه از آن استفاده میکنید. بنابراین شما میتوانید الگوریتم یک برنامـه را بـه هـر نحوی که به شما کمک میکند تا مسئله را درک کنید بازگو کنید. مثلا میتوانید آن را به زبانی که با آن صحبت میکنید برای خودتـان توضیح بدهید و یا آن را به وسیله شکلها و نمودارها رسم کنید.
فرض کنید شما برای یک شرکت خدمات تلفنی کار میکنید و میخواهید صورت حساب مشترکین را بر اساس تماسهایی که گرفته اند مشخص کنید. الگوریتمی که برای حل مسئله بالا میتوانید به کار ببرید، میتواند مشابه الگوریتم زیر باشد:

۱) در ابتدای هر ماه شما باید صورت حساب هر یک از مشترکین خود را محاسبه کنید.
۲) برای هر مشترک، شما لیست تماسهایی که آن مشترک در ماه قبل گرفته است را دارید.
۳) شما مدت هر تماس و ساعتی که آن تماس گرفته شده بود را میدانید و بر اساس این اطلاعات میتوانید هزینه هر تماس را مشخص کنید.
۴) برای هر مشترک، صورت حساب او برابر است با مجموع هزینه تمام تماسهایی که داشته است.
۵) مالیات هر صورت حساب را محاسبه می کنید.
۶) بعد از این که صورت حساب نهایی محاسبه شد، باید آن را چاپ کنید.

این شش مرحله، الگوریتم قسمتی از نرم افزاری است که هزینه ماهیانه مشترکین یک مرکز خدمات تلفنی را محاسبه میکند. تفـاوتی ندارد شما این نرم افزار را با چه زبان برنامه نویسی می نویسید، C++، C#، ویژوال بیـسیک، J#، جـاوا و یـا هـر زبـان دیگـری، الگوریتم کلی برنامه تغییری نمیکند. (البته این هفت مرحله هم خیلی کلی هستند و باید قبل از این که به وسـیله یـک زبـان برنامـه نویسی نوشته شوند به مراحل کوچکتری شکسته شوند و جزئیات بیشتری از آن شرح داده شود)
حالا برای افرادی که تازه وارد برنامه نویسی شده اند، یک خبر خوب و یک خبر بد دارم. خبر خوب این است که معمـولا ایجـاد یـک الگوریتم بسیار ساده است. مثلا در الگوریتم قبلی، فکر نکنم قسمتی به نظر شما گنگ باشد و واضح به نظر نرسد. معمولا الگوریتم ها از یک سری استدلال هایی پیروی میکنند که از نظر مردم عادی درست است. البته ممکن است که شما با الگـوریتم هـایی برخـورد کنید که دارای فرمولهای پیچیده ریاضی و یا فرمولهای دیگر علوم باشند و تصور کنید که این فرمول ها با استدلال های شما درست به نظر نمی رسند، اما خوب این فرمول ها هم از نظر افراد دیگری که آنها را میدانند درست به نظر میرسد. اما خبر بد ایـن اسـت کـه معمولا تبدیل یک الگوریتم به کد برنامه کار مشکلی است. بنابراین به عنوان یک برنامه نویس این که بدانید چگونه یک الگـوریتم را توسط یک زبان برنامه نویسی پیاده کنید مهم است.
همه برنامه نویسان حرفه ای بر این اصل عقیده دارند که تقدم یک زبان برنامه نویسی بر زبان برنامه نویسی دیگر، کـاملا بـی ربـط است. زبانهای برنامه نویسی متفاوت میتوانند کارهای مختلفی را راحت تر و سریعتر انجام دهند. مثلا C++ به برنامه نویسان قـدرت زیادی در کنترل نحوه اجرای برنامه میدهد و همچنین برنامه های آن نیز از سرعت اجرای بالایی برخوردار هستند، اما بـدی آن ایـن است که برنامه نویسی به این زبان بسیار مشکل است و یادگیری آن برای مبتدیان معمولا بسیار سخت است. در مقابل برنامه نویسی و یادگیری زبان ویژوال بیسیک بسیار راحت تر است، اما برنامه نویس در این زبان کنتـرل کـافی بـر روی برنامـه نـدارد (زبـان C# همانطور که ادعا شده است، زبانی است که از سادگی زبانی مانند ویژوال بیسیک و قدرت زبانی مانند C++ برخوردار اسـت). چیـزی که شما باید به عنوان یک برنامه نویس یاد بگیرید این است که بتوانید برای حل یک مسئله، زبانهای برنامه نویسی مختلف را بررسی کنید. البته زمانی که شروع می کنید که اولین زبان برنامه نویسی را یاد بگیرید، در آن پیشرفت کنـدی خواهیـد داشـت، امـا معمـولا زبانهای برنامه نویسی در نحوه اجرا و پیاده کردن الگوریتم ها تفاوتی ندارند. بنابراین زمانی که یک زبان برنامه نویسی را یاد گرفتیـد، می توانید از تجربه خود در الگوریتم ها و یا حتی کد زبان قبلی در زبان جدید نیز استفاده کرده و چندین زبان دیگر را نیز بـه سـرعت یاد بگیرید.

یک زبان برنامه نویسی چیست؟

از یک نگاه، میتوانیم بگوییم که یک زبان برنامه نویسی به چیزی گفته میشود که توانایی تصمیم گیری داشته باشد. کامپیوترها معمولا خیلی سریع و با دقت میتوانند تصمیم گیری کنند اما متاسفانه، فقط در زمینه های بسیار ساده میتوانند این کار را انجام بدهند. مثلا "آیا این عدد بزرگتر از سه است؟" و یا "آیا این ماشین آبی است؟".
اگر شما بخواهید یک تصمیم گیری پیچیده را به وسیله کامپیوتر انجام دهید، ابتدا باید آن را به قسمتهای کوچکتری که توسط کامپیوتر قابل فهم باشد تقسیم کنید. معمولا برای فهمیدن این که چگونه یک تصمیم گیری پیچیده را به چند قسمت ساده تقسیم کنید میتوانید از الگوریتم ها استفاده کنید.
برای نمونه یک مسئله که حل آن برای کامپیوتر خیلی سخت است، تشخیص چهره انسانها است. شما نمیتوانید به یک کامپیوتر بگویید که "آیا این عکس تصویری از ماری است؟"، بلکه باید این مسئله را به چند مسئله کوچکتر که توسط کامپیوتر قابل فهم باشد تقسیم کنید تا کامپیوتر بتواند به آنها پاسخ صحیح بدهد.
سوالاتی که شما میتوانید از کامپیوترها بپرسید و بر اساس آنها تصمیم گیری کنید معمولا دارای جواب بله و یا خیر هستند. این دو جواب معمولا با عنوان درست و غلط و یا ۱ و ۰ نیز در نظر گرفته می شوند. در زمان نوشتن نرم افزار نمی توانید بر اساس پاسخ به سوال "۱۰ در مقایسه با ۴ چه قدر بزرگتر است؟" تصمیم گیری کنید. بلکه باید این سوال را به صورت "آیا ۱۰ از ۴ بزرگتر است؟" بپرسید. تفاوت این دو سوال بسیار ریز و در عین حال مهم است. جواب سوال اول به صورت بله و یا خیر نیست اما جواب سوال دوم به این صورت است. البته همانطور که میدانید کامپیوتر قادر است به سوال اول هم پاسخ بدهد، اما بر اساس این پاسخ نمیتوان تصمیم گیری کرد. برای پاسخ سوال اول کامپیوتر مقدار ۴ را از ۱۰ کم میکند و حاصل را نگهداری میکند تا شما بتوانید از آن در قسمتهای دیگر برنامه خود استفاده کنید.
ممکن است که فکر کنید تصمیم گیری فقط بر اساس بله و یا خیر یک محدودیت است، اما واقعاً این طور نیست. حتی در زندگی روزمره نیز تصمیم هایی که می گیریم نوعی از بله و خیر است. وقتی شما میخواهید راجع به موردی تصمیم بگیرید، یا آن را قبول میکنید (بله، درست و یا ۱) یا آن را رد میکنید (نه، غلط یا ۰).
در این کتاب شما از C# به عنوان زبان برنامه نویسی خود استفاده میکنید، اما همانطور که میدانید جنبه مهم برنامه نویسی این است که زبان مورد استفاده در آن مهم نیست. در حقیقت می توانیم بگوییم برنامه نویسی مستقل از زبان برنامه نویسی است. هر برنامه ای، با هر شکل و ظاهری که باشد و هر زبانی که نوشته شده باشد، از متدها (توابع و زیربرنامه ها: چندین خط کد که یک الگوریتم خاص را اجرا میکنند) و متغییر ها (مکانی برای نگهداری و تغییر داده های برنامه ها) تشکیل شده است.

متغییر ها:

متغییر مکانی است که شما در طول کار الگوریتم خود مقداری را در آن ذخیره می کنید. بعدها شما می توانید بر اساس مقدار آن متغییر تصمیم گیری کنید (برای مثال "آیا این متغییر برابر ۷۹ است؟" و یا "آیا این متغییر بیشتر از ۴ است؟") یا عملیات ریاضی و محاسباتی بر روی آن انجام دهید (مثلا "متغییر را با ۲ جمع کن." و یا "متغییر را در ۶ ضرب کن") یا چیزهای دیگر.

کار با متغییر ها:

قبل از اینکه درگیر متغییر ها در کد شویم بهتر است که به الگوریتم زیر توجه کنید:

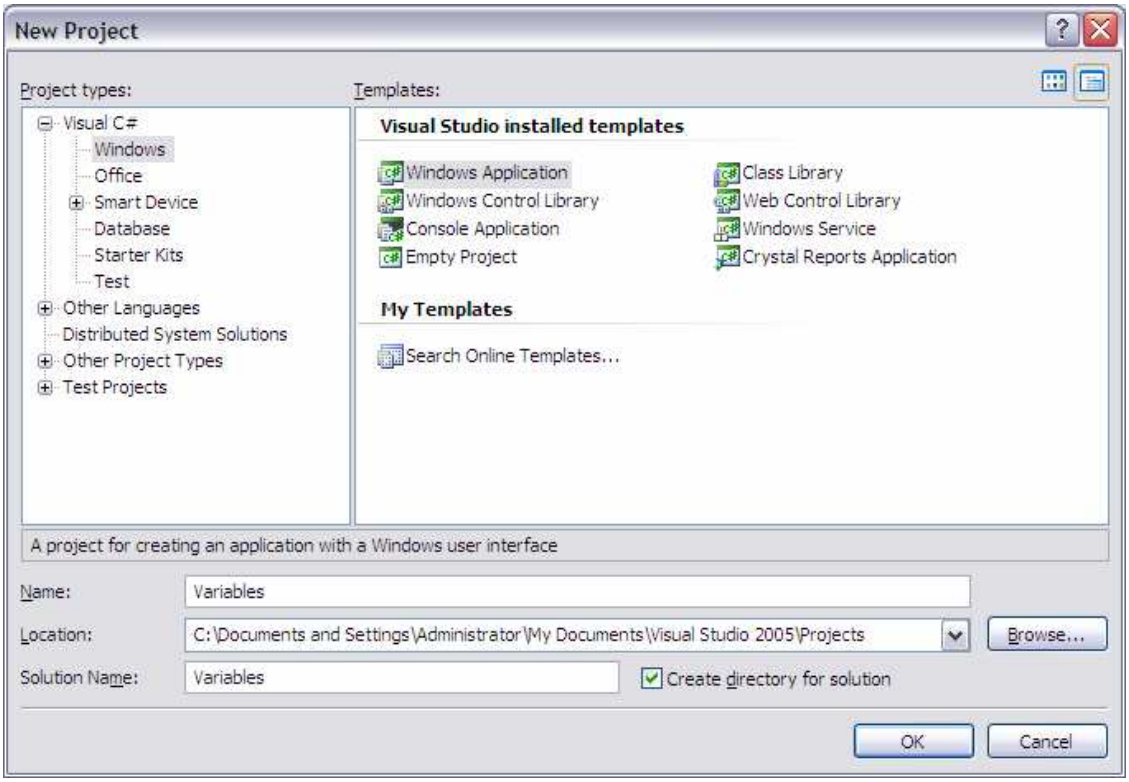
۱) یک متغییر به نام n ایجاد کنید و مقدار ۲۷ را در آن ذخیره کنید.
۲) متغییر n را با عدد ۱ جمع کنید و حاصل را مجدداً در n قرار دهید.
۳) مقدار متغییر n را به کاربر نشان دهید.

در این الگوریتم شما یک متغییر به نام n ایجاد کرده و مقدار ۲۷ را در آن قرار می دهید. این مورد به این معنی است که شما مقداری از حافظه ی کامپیوتر را که مربوط به برنامه شما است گرفته اید و مقدار ۲۷ را در آن قرار داده اید. این قسمت از حافظه کامپیوتر مقدار ۲۷ را برای شما نگه خواهد داشت تا زمانی که آن را تغییر دهید و یا به برنامه بگویید که دیگر به آن نیازی ندارید.
در مرحله دوم شما یک عملیات جمع انجام می دهید. در این مرحله، مقدار متغییر n را از گرفته و آن را با عدد ۱ جمع می کنید. سپس آن را مجدداً در n قرار می دهید. بعد از پایان این عملیات قسمتی از حافظه که متغییر n را نگهداری می کند، حاوی مقدار ۲۸ خواهد بود.
در مرحله آخر می خواهید به کاربر مقدار n را نشان دهید. برای این کار مقدار متغییر n را از حافظه کامپیوتر خوانده و آن را برای کاربر در صفحه نمایش نشان می دهید.
فکر نکنم چیزی در این الگوریتم نامفهوم باشد که متوجه آن نشده باشید. همه چیز واضح است! البته، کد این کار در ویژوال C# ۲۰۰۵ مقداری رمزی تر به نظر می رسد. در بخش "امتحان کنید" زیر، بیشتر در رابطه با کار با متغییر ها در C# آشنا خواهید شد.

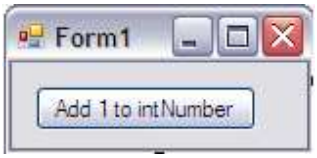
امتحان کنید: کار با متغییر ها

۱) با انتخاب گزینه `File ➔ New ➔ Project` از منوی ویژوال استودیو، یک پروژه جدید را ایجاد کنید. در پنجره `New Project` از قسمت سمت راست `Windows Application` را انتخاب کنید و در قسمت نام پروژه `Variables` را وارد کنید. سپس روی `OK` کلیک کنید. (شکل ۳-۱)
۲) پنجره `Form1` را مقداری کوچکتر کنید و سپس از جعبه ابزار یک کنترل `Button` روی آن قرار دهید. خاصیت `Text` آن را به `Add 1 to intNumber` و خاصیت `Name` آن را به `btnAdd` تغییر دهید. فرم شما باید مشابه شکل ۳-۲ شود.
۳) روی دکمه فرمان دو بار کلیک کنید تا تابع مربوط به رویداد `Click` دکمه فرمان با نام `btnAdd_Click` باز شود. کدهای مشخص شده در زیر را به آن اضافه کنید.

```
private void btnAdd_Click(object sender, EventArgs e)
{
     int intNumber;
     intNumber = 27;
     intNumber = intNumber + 1;
     MessageBox.Show(
          "Value of intNumber + 1 = " + intNumber,
          "Variables");
}
```

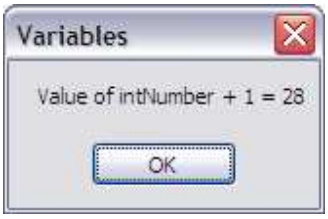


شکل ۳-۱



شکل ۳-۲

۴) برنامه را اجرا کنید و بر روی دکمه `Add 1 to intNumber` کلیک کنید. کادر پیغامی همانند شکل ۳-۳ نمایش داده میشود.



شکل ۳-۳

چگونه کار می کند؟

برنامه از بالاترین خط شروع می شود، یکی یکی خطها را اجرا میکند و به سمت پایین می آید. خط اول یک متغییر جدید به نام intNumber ایجاد میکند.

```
int intNumber;
```

int در C# یک کلمه کلیدی است. همانطور که در فصل یک گفتم کلمه کلیدی، به کلمه ای گفته میشود که برای ویژوال C# ۲۰۰۵ معنی خاصی دارد. مانند دستورات. کلمه کلیدی int، که مخفف کلمه integer به معنای عدد صحیح است، نوع مقداری که می خواهیم در این متغییر ذخیره کنیم را به ویژوال C# ۲۰۰۵ میگوید. این کلمات که برای مشخص کردن نوع داده مورد نظر استفاده میشوند، به **"نوع داده ای"** معروف هستند. فعلا همین کافی است که بدانید، این نوع داده ای به ویژوال C# ۲۰۰۵ میگوید که شما میخواهید یک عدد صحیح را در این متغییر قرار دهید.

نکته: کلمه int برای تعریف نوع داده ای عدد صحیح در C# از زبان C++ گرفته شده است. در C# بعضی از کلمات کلیدی که مربوط به کار با اعداد هستند، از زبان C++ گرفته شده اند، مانند کلمات کلیدی مربوط به تعریف اعداد صحیح، اعداد اعشاری، اعداد صحیح بزرگ و ..

بعد از این که نوع داده ای متغییر خود را مشخص کردیم، باید نامی را به آن اختصاص دهیم تا در طول برنامه برای دسترسی به آن، از آن نام استفاده کنیم. در این برنامه نام intNumber را برای متغییر انتخاب کرده ایم. توجه کنید که در نامگذاری این متغییر از نماد گذاری مجارستانی که در فصل یک توضیح دادم استفاده کرده ایم. در اینجا کلمه int مخفف عبارت Integer است و مشخص می کند که این متغییر یک عدد صحیح را در خود نگهداری می کند. کلمه Number هم نام خود متغییر است. در طول برنامه دیدید که این متغییر یک عدد صحیح را در خود نگهداری میکند، پس این نام برای آن مناسب است.
دستور نوشته شده در خط بعدی، مقدار متغییر intNumber را در آن قرار می دهد. به عبارت دیگر در این خط مقدار ۲۷ در intNumber قرار می گیرد.

نکته: می توانیم مقدار یک متغییر را هنگام تعریف نیز به آن بدهیم. مثلاً در برنامه ی بالا به جای اینکه در یک خط متغییر را تعریف کنیم و در خط بعد مقدار ۲۷ را به آن اختصاص دهیم، می توانستیم به صورت زیر در یک خط هم متغییر را تعریف کرده و هم به آن مقدار بدهیم:

```
int intNumber = 27;
```

دستور خط بعدی، عدد یک را با مقدار متغییر intNumber جمع میکند:

```
intNumber = intNumber + 1;
```

اگر از دید ریاضی به این دستور نگاه کنید ممکن است اشتباه به نظر برسد. اما در حقیقت کاری که این دستور انجام میدهد، این است که مقدار متغییر intNumber را با عدد ۱ جمع میکند و حاصل را مجددا در intNumber ذخیره میکند.
خط آخر یک کادر پیغام را به کاربر نمایش میدهد که در آن مقدار متغییر intNumber + 1 نوشته شده است. عبارتی که این کادر پیغام نمایش میدهد به صورت مجموع رشته ی "Value of intNumber + 1 =" و مقدار کنونی intNumber است. همچنین عنوان این پنجره را نیز "Variables" قرار میدهیم تا با نام برنامه هماهنگ شود:

```
MessageBox.Show(
     "Value of intNumber + 1 = " + intNumber,
     "Variables");
```

توضیحات و فضاهای خالی:

هنگامی که کد یک نرم افزار را می نویسید، باید بدانید که شما یا شخص دیگری در آینده ممکن است بخواهد این کد را تغییر دهد. بنابراین وظیفه شماست که تا جایی که ممکن است کد را قابل فهم کنید.

توضیحات:

توضیحات بخشی از برنامه هستند که به وسیله کامپایلر ویژوال C# ۲۰۰۵ نادیده گرفته می شوند. یعنی شما می توانید در این قسمت از برنامه، هر چه می خواهید و به هر زبانی که می خواهید بنویسید و نیازی نیست که قواعد خاصی را رعایت کنید. این قسمت فقط به برنامه نویسی که در حال مطالعه کد برنامه است کمک می کند تا راحت تر بفهمد آن قسمت از برنامه در حال چه کاری است.

نکته: همه زبانها قابلیت نوشتن توضیحات را دارند، اما نوع مشخص کردن توضیحات در زبانهای مختلف متفاوت است.

اما سوال اینجاست که شما چه موقع به نوشتن توضیحات نیاز دارید؟خوب، این مورد بر حسب موقعیتهای مختلف فرق می کند، اما به عنوان یک قانون کلی می توانیم بگوییم که بهتر است به الگوریتم خودتان دقت کنید. برنامه ای که در "امتحان کنید" قسمت قبل نوشتیم دارای الگوریتم زیر بود:

۱) یک مقدار را برای `intNumber` تعریف کن.
۲) مقدار ۱ را به `intNumber` اضافه کن.
۳) مقدار جدید `intNumber` را به کاربر نشان بده.

می توانید توضیحاتی را به کد برنامه قبلی اضافه کنید تا با قسمتهای الگوریتم هماهنگ شود:

```
// Define a variable for intNumber
int intNumber;
// Set the initial value
intNumber = 27;
// Add 1 to the value of intNumber
intNumber = intNumber + 1;
// Display the new value of intNumber
MessageBox.Show(
     "Value of intNumber + 1 = " + intNumber,
     "Variables");
```

در ویژوال C# ۲۰۰۵ توضیحات خود را با دو کاراکتر اسلش (//) شروع می کنید. هر متنی که بعد از این دو کاراکتر تا انتهای آن خط، قرار بگیرد به عنوان توضیح در نظر گرفته خواهد شد. بنابراین، حتی اگر در یک خط کدی وجود داشته باشد و شما در اواسط خط از // استفاده کنید، متن بعد از آن به عنوان توضیح در نظر گرفته میشود. مانند:

```
intNumber = intNumber + 1; // Add 1 to value of intNumber
```

این مورد منطقی هم هست، زیرا فقط توضیحات بعد از علامت // قرار می گیرند . توجه کنید که توضیحات کد قبلی، کم و بیش مشابه الگوریتم برنامه هستند. یک تکنیک خوب برای اضافه کردن توضیح به برنامه این است که آن قسمت از الگوریتم برنامه که در حال نوشتن کد آن هستید را در چند کلمه توضیح دهید.
اگر در قسمتی از برنامه نیاز داشتید که از بیش از یک خط توضیح استفاده کنید، می توانید به جای اینکه در ابتدای هر خط از // استفاده کنید، در جایی که توضیحات شروع می شوند از علامت /* استفاده کنید. به این ترتیب ویژوال C# تمام متن بعد از این علامت را تا هنگامی که به علامت */ برسد به عنوان توضیح در نظر می گیرد. برای مثال می توانستیم تمام توضیحات نوشته شده در برنامه قبلی را در یک قسمت به صورت زیر وارد کنیم:

```
    /* 1) Define a variable for intNumber
       2) Set the initial value
       3) Add 1 to the value of intNumber
       4) Display the new value of intNumber */
    int intNumber;
```

یکی دیگر از روشهای تولید بلاکهای توضیحات برای متدهای برنامه، استفاده از ویژگی درونی XML Document Comment در ویژوال استودیو ۲۰۰۵ است. برای استفاده از این ویژگی، مکان نما را در خط خالی بالای متد مورد نظرتان قرار دهید و سه کاراکتر اسلش (///) تایپ کنید. یک بلاک توضیحات، همانند کد زیر، به صورت اتوماتیک نمایش داده میشود.

```
/// <summary>
///
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void btnAdd_Click(object sender, EventArgs e)
{
```

اما نکته ای که در مورد این ویژگی جالب است، این است که ویژوال استودیو ۲۰۰۵ مقادیر مربوط به قسمت name را در این بلاک ها بر اساس نام متغییر های متد شما کامل می کند. اگر در متد هیچ پارامتری نداشته باشید، قسمت <param> از این بلاک توضیحات حذف می شود.
هنگامی که این بلاک توضیحات را ایجاد کردید، می توانید یک خلاصه از عملکرد متد را در قسمت summary وارد کنید و یا هر توضیح اضافی که قبل از فراخوانی متد باید در نظر داشت را نیز در این بلاک بنویسید. اگر متد شما مقداری را برگرداند بخش <return> هم به این بلاک اضافه می شود که می توانید در رابطه با مقدار بازگشتی از متد، در آن توضیحاتی بنویسید.
توضیحات داخل برنامه عموماً برای درک بهتر کد مورد استفاده قرار می گیرند، چه برای یک برنامه نویس که تا کنون کد شما را ندیده است چه برای شما اگر مدت زمان طولانی کد را مرور نکرده باشید و اکنون بخواهید آن را تغییر بدهید. در کل توضیحات به

نکاتی راجع به برنامه اشاره میکنند که در نگاه اول به کد مشخص نیستند و یا در مورد عملکرد کد خلاصه ای را بیـان مـی کننـد تـا برنامه نویس بتواند بدون دنبال کردن خط به خط کد از عملکرد آن قسمت مطلع شود.
به زودی متوجه خواهید شد که هر برنامه نویس روشهای خاص خودش را برای توضیح نوشتن دارد. اگر شما برای یک شرکت بزرگ کار می کنید، یا مدیر شما بر کد نویسی استاندارد تاکید داشته باشد، به شما گفته میشود که توضیحات خود را در چه قالبهایی بنویسید و یا اینکه در کدام قسمت از کد توضیح بنویسید و در کدام قسمت ننویسید.

فضاهای خالی:

یکی دیگر از مواردی که موجب خوانایی بیشتر کد میشود استفاده زیاد از فضاهای خالی بین کدها اسـت. همـانطور کـه فـضای بـین کلمات در انگلیسی موجب بهتر خوانده شدن متن می شود، فضاهای خالی در برنامه (فضا هایی بر روی صفحه که به وسیله کـاراکتر ها اشغال نمی شوند) موجب می شوند که کدها راحت تر خوانده شوند. مثلا اگر در مثال قبلی در هر خط، قبـل از توضـیحات از یـک خط خالی استفاده کنیم، موجب می شود کسی که کد را می خواند متوجه شود که هر کدام از این بخشها یک کار خاص را انجام مـی دهند.
در فصل بعدی، هنگامی که بخواهیم کنترل اجرای یک برنامه را به وسیله بلاکهای تکرار (`for`) و یا تصمیم گیری (`if`) توضـیح بدهیم، با دلیل استفاده از فضاهای خالی بیشتر آشنا خواهید شد. همین طور متوجه می شوید که مقدار و چگونگی استفاده از فضاهای خالی در برنامه بین برنامه نویسان متفاوت است. فعلا از زیاد فاصله قرار دادن بین کدها نترسید، این مورد خوانایی کد شما را به مقدار زیادی افزایش می دهد، مخصوصا هنگامی که بخواهید حجم زیادی از کد را در برنامه بنویسید.

نکته: کامپایلر ها، فضاهای خالی بین برنامه و همچنین توضیحات برنامه را نادیده میگیرند. بنابراین بین اجرای یک برنامه با فـضای خالی و توضیحات زیاد و یک برنامه بدون این موارد، از لحاظ سرعت و کارایی هیچ تفاوتی نیست.

نوع های داده ای:

هنگامی که میخواهید یک متغییر را تعریف کنید بهتر است که بدانید تقریبا چه نوع اطلاعاتی را میخواهید در آن ذخیره کنید. تا اینجا در مثال قبل، متغییری را دیدید که مقادیر صحیح را در خود نگه میداشت.
هنگامی که میخواهید یک متغییر را در ویژوال C# ۲۰۰۵ تعریف کنید، باید به آن بگویید که چه نوع اطلاعـاتی را میخواهیـد در آن ذخیره کنید. همانطور که حدس زده اید، این موارد به عنوان نوع های داده ای شناخته می شوند و تمام زبانهای مهـم، تعـداد زیـادی نوع داده ای دارند که شما میتوانید نوع اطلاعات داخل متغییر خود را از بین آنها انتخاب کنید. نوع داده ای یک متغییر تاثیر زیادی بـر این که چگونه کامپیوتر برنامه شما را اجرا کند دارد. در این بخش نگاه دقیقتری بر چگونگی کار متغییر ها خواهیم داشـت. همچنـین چگونگی تاثیر نوع یک متغییر را در کارایی برنامه بررسی خواهیم کرد.

کارکردن با اعداد:

زمانی که شما با اعداد در ویژوال C# ۲۰۰۵ کار می کنید، میتوانید دو نوع عدد داشته باشید: اعداد صحیح و اعداد اعشاری. هر کدام از این اعداد کاربرد خاص خودشان را دارند. مثلا اعداد صحیح توانایی نگهداری بخش اعشاری را ندارند. بنابراین برای محاسباتی که ممکن است نتیجه شامل اعشار هم باشد نباید از متغییر های صحیح استفاده کنید.
در نوشتن نرم افزارها، بیشتر از اعداد صحیح برای شمردن مراتبی که یک اتفاق رخ میدهد استفاده میشود تا برای انجام محاسبات. برای محاسبات بیشتر از اعداد اعشاری استفاده می کنیم.
مثلا، فرض کنید در حال نوشتن برنامه ای هستید که اطلاعات مشترکین را در صفحه نمایش دهد. همچنین فرض میکنیم که شما اطلاعات ۱۰۰ مشترک را در بانک اطلاعاتی خود دارید. زمانی که برنامه شروع به کار کرد، اطلاعات مشترک اول را در صفحه نمایش می دهید. در اینجا شما باید بدانید که اطلاعات کدام مشترک در صفحه چاپ شده تا زمانی که کاربر درخواست کرد اطلاعات مشترک بعدی را مشاهده کند، بدانید کدام اطلاعات را باید نمایش دهید.
معمولا اگر هر بانک اطلاعاتی را که حاوی اطلاعات مشترکین باشند مشاهده کنید، خواهید دید که در آن هر مشترک دارای یک شماره مخصوص به خود است. این به این خاطر است که کامپیوترها راحت تر میتوانند با اعداد کار کنند تا با اسامی. معمولا این شماره های مخصوص که به مشترکین داده میشود، یک عدد صحیح است که به آن شناسه[1] یا ID گفته میشود. بنابراین در مثال قبلی هم که ما لیستی از ۱۰۰ مشترک را در بانک اطلاعاتی خود داشتیم، هر مشترک دارای یک عدد صحیح مخصوص به خود از ۱ تا ۱۰۰ است. برنامه شما برای چاپ اطلاعات مشترکین، میتواند شماره آخرین مشترکی که اطلاعات او چاپ شده است را در یک متغییر نگه دارد. زمانی که کاربر درخواست کرد تا اطلاعات مشترک بعدی را ببیند، کافی است که شما یک واحد به شناسه آخرین کاربری که اطلاعات او چاپ شد اضافه کنید و اطلاعات مشترک جدید را چاپ کنید.

این گونه برنامه ها را زمانی که در مورد مباحث پیشرفته تری مثل بانکهای اطلاعاتی صحبت کردیم، تمرین خواهیم کرد. اما فعلاً بدانید که در برنامه ها معمولا از اعداد صحیح بیشتر از اعداد اعشاری استفاده می شود.

عملیات ریاضی معمول روی اعداد صحیح:

در این قسمت شما پروژه جدیدی برای انجام عملیات ریاضی خواهید نوشت.

امتحان کنید: کار با اعداد صحیح

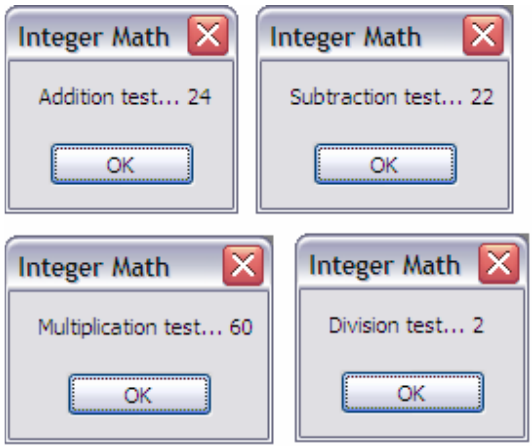
۱) با انتخاب منوی File ➔ New ➔ Project یک پروژه جدید در ویژوال استودیو ۲۰۰۵ ایجاد کنید. در پنجره New Project گزینه Windows Application را از قسمت سمت راست انتخاب کنید (به عکس ۳-۱ رجوع کنید). نام پروژه را IntegerMath وارد کنید و روی OK کلیک کنید.
۲) قبل از هر چیز با استفاده از جعبه ابزار، یک کنترل Button به فرم خود اضافه کنید. خاصیت Name آن را به btnIntMath و خاصیت Text آن را به Math Test تغییر دهید. روی کنترل دو بار کلیک کنید و در متد ایجاد شده برای رویداد Click کد مشخص شده در زیر را تایپ کنید:

```
private void btnIntMath_Click(object sender, EventArgs e)
{
    // Declare variable
```

[1] Identifier

```
        int intNumber;
        // Set number, add numbers, and display results
        intNumber = 16;
        intNumber = intNumber + 8;
        MessageBox.Show("Addition test... " + intNumber,
             "Integer Math");
        // Set number, subtract numbers, and display results
        intNumber = 24;
        intNumber = intNumber - 2;
        MessageBox.Show("Subtraction test... " + intNumber,
             "Integer Math");
        // Set number, multiply numbers, and display results
        intNumber = 6;
        intNumber = intNumber * 10;
        MessageBox.Show("Multiplication test... " +
             intNumber,"Integer Math");
        // Set number, divide numbers, and display results
        intNumber = 12;
        intNumber = intNumber / 6;
        MessageBox.Show("Division test... " + intNumber,
             "Integer Math");
}
```

۳) پروژه را اجرا کنید و روی دکمه Math Test کلیک کنید. چهار کادر پیغام متوالی همانند شکل ۴-۳ نمایش داده میشوند و شما باید روی OK در هر کدام کلیک کنید.



شکل ۳-۴

چگونه کار میکند؟

خوشبختانه، هیچ کدام از کدهایی که در برنامه بالا نوشتید، خیلی گیج کننده نیستند. شما قبلا کاربرد عملگر جمع را دیده اید، در اینجا هم از آن دوباره استفاده کرده ایم.

```
// Set number, add numbers, and display results
intNumber = 16;
intNumber = intNumber + 8;
MessageBox.Show("Addition test... " + intNumber,
     "Integer Math");
```

چیزی که در این کد شما به کامپایلر میگویید این است که:

۱) مقدار متغییر `intNumber` را برابر ۱۶ قرار بده.
۲) سپس مقدار متغییر `intNumber` را برابر مقدار کنونی این متغییر (عدد ۱۶) به اضافه ۸ قرار بده.

همانطور که در کادر پیغام شکل ۴–۳ می بینید، مقدار ۲۴ به عنوان نتیجه نمایش داده خواهد شد که درست است.
عملگر تفریق در اینجا علامت (-) است که عملکرد آن را در کد زیر می بینید:

```
// Set number, subtract numbers, and display results
intNumber = 24;
intNumber = intNumber - 2;
MessageBox.Show("Subtraction test... " + intNumber,
     "Integer Math");
```

دوباره، همانند قسمت جمع به کامپایلر می گویید که:

۱) مقدار متغییر `intNumber` را برابر ۲۴ قرار بده.
۲) مقدار متغییر `intNumber` را برابر مقدار کنونی این متغییر (۲۴) منهای ۲ قرار بده.

عملگر ضرب، علامت ستاره است (*) که کاربرد آن در کد زیر مشخص است:

```
// Set number, multiply numbers, and display results
intNumber = 6;
intNumber = intNumber * 10;
MessageBox.Show("Multiplication test... " +
     intNumber,"Integer Math");
```

الگوریتم این کد به صورت زیر است:

۱) مقدار `intNumber` را برابر ۶ قرار بده.
۲) مقدار `intNumber` را برابر مقدار کنونی این متغییر (۶) ضرب در ۱۰ قرار بده.

در آخر، عملگر تقسیم یک علامت / است که در زیر نشان داده شده است:

```
// Set number, divide numbers, and display results
```

```
intNumber = 12;
intNumber = intNumber / 6;
MessageBox.Show("Division test... " + intNumber,
     "Integer Math");
```

و الگوریتم آن مطابق زیر است:

۱) مقدار `intNumber` را برابر ۱۲ قرار بده.
۲) مقدار `intNumber` را برابر مقدار کنونی این متغییر (یعنی ۱۲) تقسیم بر ۶ قرار بده.

تند نویسی در عملیات ریاضی:

در بخش امتحان کنید بعدی، شما کدهای برنامه قبلی را به نحوی بسیار کوتاهتر خواهید نوشت. البته مسلما در ابتدا این نوع نوشـتن نسبت به نوع طولانی آنها غیر منطقی به نظر می رسد، اما به زودی، بعد از مقداری کار، از آنها بیشتر استفاده خواهید کرد.

امتحان کنید: استفاده از عملگر های مختصر شده

۱) به ویژوال استودیو ۲۰۰۵ بروید و فایل `Form1.cs` را مجددا باز کنید. خطهای مشخص شده در زیر را تغییر دهید:

```
private void btnIntMath_Click(object sender, EventArgs e)
{
     // Declare variable
     int intNumber;
     // Set number, add numbers, and display results
     intNumber = 16;
     intNumber += 8;
     MessageBox.Show("Addition test... " + intNumber,
          "Integer Math");
     // Set number, subtract numbers, and display results
     intNumber = 24;
     intNumber -= 2;
     MessageBox.Show("Subtraction test... " + intNumber,
          "Integer Math");
     // Set number, multiply numbers, and display results
     intNumber = 6;
     intNumber *= 10;
     MessageBox.Show("Multiplication test... " +
          intNumber,"Integer Math");
     // Set number, divide numbers, and display results
     intNumber = 12;
     intNumber /= 6;
```

```
        MessageBox.Show("Division test... " + intNumber,
            "Integer Math");
}
```

۲) برنامه را اجرا کنید و روی دکمه `Math Test` کلیک کنید. مشاهده می کنید که نتیجه ای مشابه برنامه قبلی را دریافت می کنید.

چگونه کار می کند؟

برای استفاده از حالت مختصر شده عملگر ها کافی است که نام متغییر را که برای بار دوم تکرار شده است حذف کنید و علامت ریاضی مرتبط با آن را نیز به قبل از مساوی انتقال دهید. مثلا در حالت:

```
intNumber = intNumber + 8;
```

عبارت به صورت زیر تبدیل می شود:

```
intNumber += 8;
```

محدودیت کار با اعداد صحیح:

محدودیت اصلی که در کار با اعداد صحیح وجود دارد این است که شما نمی توانید عددی داشته باشید که دارای قسمت اعشاری باشد. برای مثال استفاده از کد زیر موجب ایجاد خطا در زمان کامپایل برنامه میشود:

```
        // Try multiplying numbers
        int intNumber = 34;
        intNumber *= 10.234;
```

اجرای برنامه بالا به علت خطایی که در قسمت کامپایل به وجود می آید متوقف می شود. زیرا متغییر `intNumber` از نوع عدد صحیح است و شما نمی توانید آن را در حالت عادی در یک عدد اعشاری ضرب کنید.
در تقسیم دو عدد هم اگر بخواهید یک عدد صحیح را بر یک عدد اعشاری تقسیم کنید، با خطایی مشابه حالت جمع روبرو میشوید. اما کد زیر را در نظر بگیرید:

```
        // Try deviding numbers…
        int intNumber = 12;
        intNumber /= 7;
```

در این کد مقدار متغییر `intNumber` را (۱۲) که یک عدد صحیح است بر ۷ که آن نیز یک عدد صحیح است تقسیم کرده ایم و جواب برابر ۱,۷۱ است. در این مواقع کامپایلر خطایی تولید نمیکند، اما مقدار که در متغییر `intNumber` ذخیره میشود با جوابی که شما انتظار دارید تفاوت دارد. به عبارت دیگر، بعد از اجرای کد بالا، جواب ۱,۷۱ در `intNumber` نخواهد بود، بلکه

قسمت اعشار این عدد حذف میشود و مقدار ۱ در متغییر intNumber قرار میگیرد. همانطور که ممکن است تصور کنید، اگر برنامه ای بنویسید که بخواهد با انواع مختلف داده ها کار کند و از اعداد صحیح استفاده کنید با مشکل مواجه خواهید شد.
در قسمت بعدی خواهید دید که چگونه می توانید با استفاده از اعداد اعشاری، از بروز مشکلاتی مانند قبل جلوگیری کنید.

اعداد اعشاری:

در قسمت قبل متوجه شدید که اعداد صحیح برای انجام محاسبات ریاضی مناسب نیستند. زیرا نتیجه بیشتر این محاسبات دارای قسمت اعشاری است و اعداد صحیح هم نمی توانند قسمت اعشاری را در خود نگهداری کنند. در این بخش چگونگی انجام محاسبات ریاضی با اعداد اعشاری را در برنامه های مختلف مانند محاسبه مساحت و محیط دایره تمرین خواهیم کرد، اما فعلا، در آزمایش کنید زیر، فقط مفاهیم کلی را معرفی می کنیم.

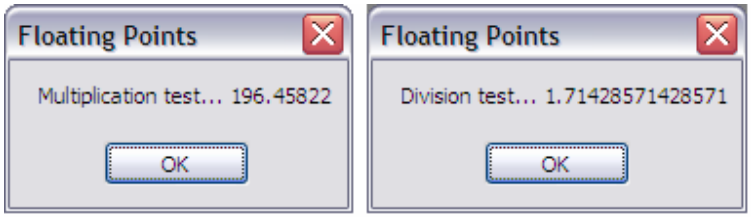
امتحان کنید: اعداد اعشاری

۱) یک پروژه جدید در ویژوال استودیو ۲۰۰۵ به نام Floating-Pt Math ایجاد کنید. قبل از هر چیز، یک کنترل Button روی فرم قرار دهید و خاصیت Name آن را برابر btnFloatMath و خاصیت Text آن را برابر Double Test قرار دهید.
۲) روی دکمه btnFloatMath دو بار کلیک کنید و در متد مربوط به رویداد کلیک آن کدی را که در زیر مشخص شده است وارد کنید.

```
Private void btnFloatMath_Click(object sender,
                                    EventArgs e)
{
     // Declare variable
     double dblNumber;

     // Set number, multiply numbers, and display results
     dblNumber = 45.34;
     dblNumber *= 4.333;
     MessageBox.Show("Multiplication test... " +
          dblNumber, "Floating Points");
     // Set number, divide numbers, and display results
     dblNumber = 12;
     dblNumber /= 7;
     MessageBox.Show("Division test... " + dblNumber,
               "Floating Points");
}
```

۳) برنامه را اجرا کنید و روی کلید Double Test کلیک کنید. نتیجه ای مشابه شکل ۳-۵ را مشاهده خواهید کرد.



شکل ۳-۵

چگونه کار می کند؟

همانطور که متوجه شدید مهمترین تغییر در برنامه بالا، تغییر نوع تعریف متغییر است:

```
// Declare variable
double dblNumber;
```

به جای اینکه برای تعریف متغییر از کلمه کلیدی `int` استفاده کنیم، از کلمه `double` استفاده کرده ایم. این کلمه بـه ویـژوال C# ۲۰۰۵ میگوید که شما می خواهید در این متغییر به جای اعداد صحیح، اعداد با قسمت اعشار قرار دهید. در نتیجه، هر عملیـاتی که بر روی متغییر `dblNumber` انجام دهید از نوع اعشاری خواهد بود و میتواند قسمت اعشاری را نیـز نگهـداری کنـد. نکتـه مهم دیگر در کد بالا این است که به جای استفاده از پیشوند `int` از پیشوند `dbl` استفاده کرده ایم تا مشخص باشـد کـه متغییـر بالا اعداد اعشاری از نوع `Double` را در خود نگهداری می کنند.
البته با این که عملیات روی متغییر `dblNumber` قسمت اعشاری را نیز نگهداری می کنند اما روش انجام عملیـات، همـانطور که در کد زیر مشاهده می کنید، با عملیات روی اعداد صحیح تفاوتی ندارد.

```
// Set number, multiply numbers, and display results
dblNumber = 45.34;
dblNumber *= 4.333;
MessageBox.Show("Multiplication test... " +  dblNumber,
     "Floating Points");
```

اگر کد بالا را اجرا کنید، نتیجه ۱۹۶,۴۵۸۲۲ را مشاهده میکنید که همانند دو عددی که در هم ضرب شدند دارای قسمت اعشاری نیز هست. البته اعدادی که در این محاسبات به کار میروند حتما نباید دارای بخش صحیح باشند، بلکه مانند قسمت تقـسیم برنامـه قبـل می توانند از دو عدد صحیح تشکیل شده باشند که در صورت نیاز حاصل عبارت با قسمت اعشاری نمایش داده خواهد شد. برای مثال به کد زیر توجه کنید:

```
// Set number, divide numbers, and display results
dblNumber = 12;
dblNumber /= 7;
MessageBox.Show("Division test... " + dblNumber,
          "Floating Points");
```

نتیجه این تقسیم دارای قسمت اعشاری نیز خواهد بود زیرا متغییر `dblNumber` به گونه ای تعریف شده است کـه در صـورت نیاز بتواند قسمت اعشاری اعداد را نیز نگهداری کند. بنابراین اگر برنامه بالا را اجرا کنیـد، عـدد ۱,۷۱۴۲۸۵۷۱۴۲۸۵۷۱ را بـه عنـوان نتیجه مشاهده خواهید کرد.

نکته: به اعداد اعشاری، اعداد با ممیز شناور نیز گفته میشود. این نامگذاری به این دلیل است که این اعداد بـه صـورت عـدد علمـی ذخیره می شوند. در نماد گذاری علمی برای اعداد، یک عدد به صورت توانی از ۱۰ و عـددی دیگـر بـین ۱ تـا ۱۰ وجـود دارد. بـرای محاسبه مقدار واقعی عدد، ۱۰ به توان عدد اول می رسد و سپس در عدد دوم ضرب می شود. برای مثـال، عـدد ۱۰۰۰۱ بـه صـورت $۱,۰۰۰۱*۱۰^{۴}$ و عدد ۰,۰۰۱۰۰۰۱ به صورت $۱,۰۰۰۱*۱۰^{-۳}$ نوشته می شود. در این گونه اعداد، ممیـز بـین اعـداد بعـد از رقـم اول شناور است. اعداد اعشاری در کامپیوتر نیز با همین روش نگهداری می شوند با این تفاوت که این اعداد در مبنای ۱۰ بودند اما اعـداد در کامپیوتر در مبنای ۲ ذخیره می شوند.

حالتهای دیگر:

اعداد اعشاری علاوه بر مقادیر عددی، میتوانند حالتهای خاص دیگری را نیز نگهداری کنند. بعضی از این حالتها عبارتند از:

- `NaN` – که به معنی "`Not a Number`" یا "عدد نیست" است.
- بی نهایت منفی
- بی نهایت مثبت

ما در این کتاب در مورد این حالتها صحبت نخواهیم کرد، اما در نوشتن برنامه های محاسباتی و ریاضـی میتـوان از ایـن حالتهـا نیـز استفاده کرد.

اعداد اعشاری با دقت معمولی:

در قسمت قبلی از اعداد اعشاری با دقت مضاعف استفاده کردیم، اما در `.NET` شما برای نگهداری اعداد اعشاری خـود میتوانیـد از دو نوع عدد اعشاری استفاده کنید. در بعضی از مواقع، بخش اعشاری یک عدد ممکن است تا بی نهایت ادامه پیـدا کنـد (ماننـد عـدد پی)، اما کامپیوتر بی نهایت فضا برای نگهداری این اعداد ندارد، بنابراین همیـشه بـرای نگهـداری تعـداد ارقـام اعـشاری یـک عـدد محدودیتهایی وجود دارد. این محدودیت به اندازه یا فضایی بستگی دارد که متغییر برای نگهداری عدد از آن استفاده میکند.
اعداد اعشاری با دقت مضاعف میتوانند اعداد از $۱,۷ * ۱۰^{۳۰۸}$ – تا $۱,۷ * ۱۰^{۳۰۸}$ + را با دقتی بسیار بالا نگهداری کند (بـا دقـت یـک پنی در ۴۵ تریلیون دلار). اعداد صحیح با دقت معمولی میتوانند اعداد از $۳,۴ * ۱۰^{۳۸}$ – تا $۳,۴ * ۱۰^{۳۸}$ + را نگهداری کنند. البته ایـن عدد هم عدد بسیار بزرگی است اما میزان دقت این اعداد خیلی کمتر از اعداد با دقـت مـضاعف اسـت (یـک پنـی در ۳۳۰۰۰۰ دلار). مزیتی که اعداد با دقت معمولی دارند این است که نسبت به اعداد با دقت مضاعف فضای کمتری در حافظه اشغال می کنند و سرعت محاسبه بالاتری دارند.

نکته: به جز در مواردی که به دقت بسیار بالایی نیاز دارید، از اعداد با دقت مضاعف استفاده نکنید و به جای آن اعداد با دقت معمولی را به کار ببرید. استفاده از اعداد با دقت مضاعف به جای اعداد با دقت معمولی، مخصوصا در برنامه های بزرگ میتواند در سرعت برنامه شما به شدت تاثیرگذار باشد.

برای انتخاب اینکه از چه نوع عددی استفاده کنید، باید به محاسباتی که میخواهید انجام دهید توجه کنید. برای تعریف متغییر های عددی با دقت مضاعف از کلمه کلیدی `double` و برای تعریف متغییر های عددی با دقت معمولی از کلمه کلیدی `float` استفاده کنید.

کار با رشته ها:

رشته مجموعه ای از کاراکتر است که ابتدا و انتهای آن به وسیله علامت نقل قول (") مشخص می شود. روش استفاده از رشته ها را در برنامه های قبلی برای نمایش نتیجه برنامه در صفحه نمایش دیده اید. رشته ها عموما برای این منظور استفاده می شوند که به کاربر اطلاع داده شود در برنامه چه اتفاقی افتاده است و چه اتفاقی میخواهد رخ دهد. یک استفاده دیگر از رشته ها، ذخیره قسمتی از یک متن برای استفاده از آن در یک الگوریتم است. در طول این کتاب با رشته های زیادی مواجه می شوید، همانطور که تاکنون از رشته زیر استفاده کرده اید:

```
MessageBox.Show("Multiplication test... " + dblNumber,
          "Floating Points");
```

`"Multiplication test…"` و `"Floating Points"` نمونه هایی از رشته هستند، زیرا دارای علامت (") در ابتدا و انتهای خود هستند. اما عبارت `dblNumber` چی؟ در عبارت بالا، مقدار متغییر `dblNumber` به رشته تبدیل شده و پس از ترکیب با دو رشته دیگر در صفحه نمایش داده خواهد شد (چگونگی تبدیل این متغییر عددی به رشته، بحثی است که جلوتر راجع به آن صحبت شده است. اما فعلا این را بدانید که در این جا یک تبدیل صورت گرفته است). برای مثال اگر مقدار متغییر `dblNumber` برابر با ۲۷ باشد، این مقدار به یک عبارت رشته ای که دو کاراکتر طول دارد تبدیل میشود و سپس روی صفحه نمایش داده می شود.در بخش امتحان کنید بعدی، با یک سری از کارهایی که میتوانید با رشته ها انجام دهید آشنا خواهید شد.

امتحان کنید: استفاده از رشته ها

۱) با استفاده از گزینه `File` ➔ `New` ➔ `Project` یک پروژه جدید در ویژوال استودیو تعریف کنید و نام آن را `Strings` قرار دهید.
۲) با استفاده از جعبه ابزار، یک کنترل `Button` روی فرم قرار دهید. خاصیت `Name` این دکمه را برابر `btnStrings` و خاصیت `Text` آن را برابر `Using Strings` قرار دهید. روی آن دو بار کلیک کنید و در متد ایجاد شده، کد زیر را وارد کنید:

```
private void btnStrings_Click(object sender, EventArgs e)
{
```

```
        // Declare variable
        string strData;
        // Set the string value
        strData = "Hello, world!";
        // Display the result
        MessageBox.Show(strData, "Strings");
}
```

۳) برنامه را اجرا کنید و روی دکمه `Using Strings` کلیک کنید. کادر پیغامی مشابه شکل ۳-۶ نمایش داده خواهد شد.



شکل ۳-۶

چگونه کار می کند؟

برای تعریف متغییری که بتواند رشته ها را در خود نگهداری کند، می توانید مشابه تعریف متغییر های عددی عمل کنید. اما این مرتبه از کلمه کلیدی `string` استفاده کنید:

```
// Declare variable
string strData;
```

همانند قبل، یک مقدار را به متغییر جدید اختصاص می دهید:

```
// Set the string value
strData = "Hello, world!";
```

برای مشخص کردن این که رشته شما از کجا شروع شده و تا کجا ادامه پیدا میکند باید از علامت نقل قول (") استفاده کنید. این مورد اهمیت زیادی دارد، زیرا این علامت به ویژوال C# ۲۰۰۵ میگوید که کدام عبارات را باید به عنوان رشته در نظر بگیرد و آنها را کامپایل نکند. اگر از علامت نقل قول استفاده نکنید، ویژوال C# ۲۰۰۵ با این متن ها به عنوان کد رفتار کرده، سعی میکند آنها را کامپایل کند و نمیتواند. بنابراین کامپایل کل برنامه با خطا روبرو خواهد شد.
وقتی مقدار `"Hello, world!"` را در متغییر `strData` ذخیره کردید میتوانید آن را در به عنوان یک پارامتر به تابع `MessageBox.Show` بفرستید تا روی صفحه نمایش چاپ کند.
همانطور که دیدید نحوه تعریف و استفاده از رشته ها نیز مشابه اعداد است. در قسمت بعدی در مورد چگونگی انجام عملیات مختلف روی رشته ها صحبت خواهیم کرد.

اتصال رشته ها:

اتصال رشته ها به معنی به هم متصل کردن یک سری از رشته ها در امتداد یکدیگر و ایجاد یک رشته جدید است. اتصال برای رشته ها همانند عمل جمع کردن در اعداد است. در بخش امتحان کنید بعدی، با چگونگی این عمل آشنا خواهید شد.

امتحان کنید: اتصال رشته ها

۱) در پروژه ای که در بخش قبلی ایجاد کردید، به قسمت طراحی `Form1` بروید و یک کنترل `Button` جدید اضافه کنید. خاصیت `Name` آن را برابر `btnConcatenation` و خاصیت `Text` آن را برابر `Concatenation` قرار دهید. روی این کنترل دو بار کلیک کنید و کد زیر را در متد ایجاد شده وارد کنید:

```
private void btnConcatenation_Click(object sender,
                                    EventArgs e)
{
     // Declare variables
     string strOne;
     string strTwo;
     string strResults;

     // Set the string values
     strOne = "Hello, ";
     strTwo = "World!";

     // Concatenate the strings
     strResults = strOne + strTwo;

     // Display the results
     MessageBox.Show(strResults, "Strings");
}
```

۲) برنامه را اجرا کنید و روی دکمه `Concatenation` کلیک کنید. کادر پیغامی مشابه شکل ۳-۶ مشاهده خواهید کرد.

چگونه کار می کند؟

در این امتحان کنید، ابتدا سه متغییر از نوع رشته تعریف می کنید:

```
// Declare variables
string strOne;
string strTwo;
```

```
string strResults;
```

سپس به دو متغییر اول مقدار می دهید:

```
// Set the string values
strOne = "Hello, ";
strTwo = "World!";
```

بعد از این که به دو متغییر اول مقدار دادید، دو رشته را با استفاده از علامت جمع (+) با هم جمع می کنید و عبارت جدید را در متغییر سوم به نام `strResults` قرار می دهید:

```
// Concatenate the strings
strResults = strOne + strTwo;
```

در واقع در این قسمت به کامپایلر می گویید: "مقدار متغییر `strResults` را برابر مقدار `strOne` به علاوه مقدار `strTwo` قرار بده". هنگامی که تابع `MessageBox.Show` را فراخوانی کردید، مقدار `strResults` برابر `"Hello, World!"` خواهد بود، بنابراین نتیجه ای مشابه قبل دریافت می کنید.

```
// Display the results
MessageBox.Show(strResults, "Strings");
```

استفاده از عملگر اتصال رشته در درون برنامه:

برای اتصال دو رشته به یکدیگر حتما نباید متغییری تعریف کنید و رشته ها را درون آن قرار دهید. بلکه می توانید درون کد و به سرعت از آنها استفاده کنید. این روش در امتحان کنید این بخش شرح داده شده است.

امتحان کنید: اتصال رشته ها درون برنامه

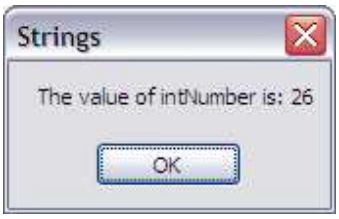
۱) مجددا به قسمت طراحی `Form1` برگردید و یک دکمه فرمان جدید به صفحه اضافه کنید. خاصیت `Name` آن را برابر `btnInlineConcatenation` و خاصیت `Text` آن را برابر `Inline Concatenation` قرار دهید. روی دکمه دو بار کلیک کنید و کد زیر را در آن وارد کنید:

```
private void btnInlineConcatenation_Click(object sender,
                                          EventArgs e)
{
    // Declare variable
    int intNumber;

    // Set the value
    intNumber = 26;
```

```
    // Display the results
    MessageBox.Show("The value of intNumber is: " +
                    intNumber, "Strings");
}
```

۲) کد را اجرا کنید و روی دکمه `Inline Concatenation` کلیک کنید. نتیجه ای مشابه شکل ۳-۷ را مشاهده خواهید کرد.



شکل ۳-۷

چگونه کار می کند؟

استفاده از عملگر اتصال رشته مانند کد بالا را قبلا در مثال های پیش دیده بودید. چیزی که در حقیقت این کد انجام می دهد تبدیل مقدار ذخیره شده در متغییر `intNumber` به رشته است. به این ترتیب این مقدار میتواند در صفحه نمایش چاپ شود. به این کد نگاه کنید:

```
// Display the results
MessageBox.Show("The value of intNumber is: " +
                intNumber, "Strings");
```

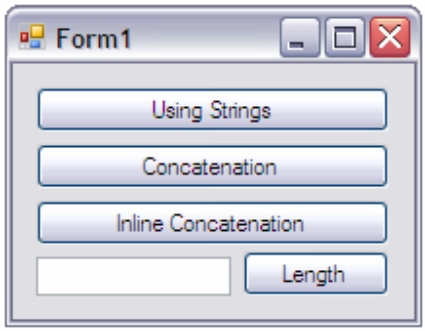
بخش `"The value of intNumber is: "` در حقیقت یک رشته است، اما شما مجبور نیستید که آن را به عنوان یک متغییر رشته ای تعریف کنید. در ویژوال C# ۲۰۰۵ این نوع رشته ها را یک **ثابت رشته ای** می نامند، زیرا از هنگام تعریف تا موقع استفاده، مقدار آنها ثابت است و تغییر نمی کند. زمانی که شما از عملگر اتصال رشته ها روی این رشته و متغییر `intNumber` استفاده کردید، مقدار متغییر `intNumber` به رشته تبدیل خواهد شد و در انتهای `"The value of intNumber is: "` قرار خواهد گرفت. نتیجه این عمل یک رشته جدید شامل هر دو عبارت رشته و عدد خواهد بود که به تابع `MessageBox.Show` فرستاده می شود.

عملیات بیشتر روی رشته ها:

در هر زبان برنامه نویسی از جمله C# توابع زیادی برای کار بر روی یک رشته وجود دارند. بعضی از این توابع در بخش امتحان کنید بعدی توضیح داده شده اند. برای مثال میتوانید طول یک رشته را با استفاده از این توابع بدست آورید.

امتحان کنید: بدست آوردن طول یک رشته

۱) با استفاده از قسمت طراحی `Form1` یک کنترل `TextBox` به فرم خود اضافه کنید. و خاصیت `Name` آن را بـه `txtString` تغییر دهید. کنترل `Button` دیگـری بـه صـفحه اضـافه کنیـد و خاصـیت `Name` آن را برابـر `btnLength` و خاصیت `Text` آن را برابر `Length` قرار دهید. کنترلهای روی فرم را به نحـوی قـرار دهیـد که مشابه شکل ۳–۸ باشند:



شکل ۳–۸

۲) روی دکمه `Length` دو بار کلیک کنید و کد زیر را متد ایجاد شده اضافه کنید:

```
private void btnLength_Click(object sender, EventArgs e)
{
     // Declare variable
     string strData;

     // Get the text from the TextBox
     strData = txtString.Text;

     // Display the length of the string
     MessageBox.Show(strData.Length + " Character(s)",
               "Strings");
}
```

۳) برنامه را اجرا کنید و متن دلخواهی را در `TextBox` وارد کنید.

۴) روی دکمه `Length` کلیک کنید. نتیجه ای مشابه آنچه در شکل ۳–۹ نشان داده شده است را خواهید دید.



شکل ۳–۹

چگونه کار می کند؟

اولین کاری که انجام می دهید تعریف متغییری است که اطلاعات را در خود نگهداری کند. سپس متن وارد شـده در TextBox را دریافت کرده و در متغییر متنی خود که strData نامیده می شود ذخیره می کنید.

```
// Declare variable
string strData;

// Get the text from the TextBox
strData = txtString.Text;
```

زمانی که یک متغییر متنی دارید، می توانید از خاصیت Length آن برای دریافت تعداد حروف رشته استفاده کنید. ایـن خاصـیت مقداری را از نوع عدد صحیح به عنوان طول رشته برمی گرداند. به یاد داشته باشید که کامپیوتر هر کاراکتری از قبیل فضاهای خـالی و علامتها را نیز در محاسبه طول رشته به شمار می آورد.

```
// Display the length of the string
MessageBox.Show(strData.Length + " Character(s)",
          "Strings");
```

زیر رشته ها:

در بیشتر مواقع ممکن است بخواهید در برنامه خود به جای کار با تمام رشته، با قسمتی از آن کار کنید. برای مثال می خواهید از یک مجموعه از کاراکتر ها که در اول رشته و یا در آخر آن آمده اند استفاده کنید. این مجموعه کاراکتر ها که ممکن است از هر جـایی از رشته شروع شوند و به هر جایی در رشته ختم شوند را زیر رشته می نامیم.
در بخش امتحان کنید بعد، برنامه قبلی را به گونه ای تغییر می دهیم که سه کاراکتر ابتدا، وسط و انتهای رشته را نیز نمایش دهد.

امتحان کنید: کار با زیر رشته ها

۱) اگر برنامه Strings در حال اجرا است، آن را ببندید.
۲) دکمه فرمان دیگری به فرم اضافه کنید و خاصیت Name آن را برابر btnSplit و خاصیت Text آن را برابـر Split قرار دهید. روی دکمه دو بار کلیک کنید و کد مشخص شده در زیر را وارد کنید:

```
private void btnSplit_Click(object sender, EventArgs e)
{
     // Declare variable
     string strData;
```

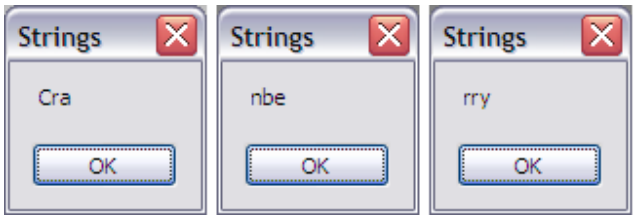
```
        // Get the text from thew TextBox
        strData = txtString.Text;

        // Display the first three characters
        MessageBox.Show(strData.Substring(0, 3), "Strings");

        // Display the middle three characters
        MessageBox.Show(strData.Substring(3, 3), "Strings");

        // Display the last three characters
        MessageBox.Show(
        strData.Substring(strData.Length - 3), "Strings");
}
```

۳) برنامه را اجرا کنید و کلمه `Cranberry` را در جعبه متنی وارد کنید.
۴) روی دکمه `Split` کلیک کنید. سه کادر پیغام متوالی همانند شکل ۳-۱۰ مشاهده خواهید کرد.



شکل ۳-۱۰

چگونه کار می کند؟

متد `Substring` به شما این امکان را می دهد تا از هر قسمتی از رشته، یک مجموعه از کاراکتر ها را جدا کنید. این متد به دو روش می تواند فراخوانی شود. روش اول این است که شماره کاراکتر اول و تعداد کاراکتر هایی را که نیاز دارید به تابع بدهید. برای مثال در اولین بار اجرای تابع در برنامه بالا به ویژوال C# ۲۰۰۵ می گوییم که از کاراکتر صفرم (از ابتدای رشته) شروع کن و سه کاراکتر را جدا کن:

```
// Display the first three characters
MessageBox.Show(strData.Substring(0, 3), "Strings");
```

در مرتبه دوم فراخوانی، به تابع می گویید که از کاراکتر سوم، سه کاراکتر را جدا کند و برگرداند.

```
// Display the middle three characters
MessageBox.Show(strData.Substring(3, 3), "Strings");
```

در مرتبه آخری که تابع را فراخوانی می کنید، فقط یک پارامتر را برای آن مشخص می کنید. این پارامتر به تابع می گوید که از مکان مشخص شده شروع کند و تمام کاراکتر های سمت راست آن را جدا کند. در این قسمت می خواهیم سه کاراکتر آخر رشته را

برگردانیم. بنابراین با استفاده از خاصیت Length به تابع Substring می گوییم که از سه کاراکتر مانده به انتهای رشته شروع کن و تمام کاراکتر های باقی مانده را برگردان.

```
// Display the last three characters
MessageBox.Show(strData.Substring(strData.Length - 3),
          "Strings");
```

قالب بندی رشته ها:

هنگامی که می خواهید با رشته ها کار کنید، ممکن است نیاز داشته باشید که نحوه نمایش کاراکتر ها بر صفحه نمایش را تغییر دهید. مثلا در شکل ۳-۵، کادر پیغام نتیجه تقسیم را نمایش میدهد، اما احتمالا شما به هر ۱۴ رقم اعشار نیاز ندارید و ۲ یا ۳ رقم کفایت میکند! چیزی که شما در این قسمت نیاز دارید این است که رشته را به گونه ای قالب بندی کنید که تمام کاراکتر های سمت چپ ممیز و فقط ۲ یا ۳ رقم اعشار را نشان دهد. در بخش امتحان کنید بعدی این کار را انجام خواهیم داد:

امتحان کنید: قالب بندی رشته ها

۱) برنامه Floating-Pt Math را که قبلا در این فصل ایجاد کرده بودید باز کنید.
۲) ویرایشگر کد را برای Form1 باز کنید و تغیرات زیر را در کد ایجاد کنید:

```
// Set number, divide numbers, and display results
dblNumber = 13;
dblNumber /= 7;

// Display the results without formatting
MessageBox.Show("Without formatting: " + dblNumber,
          "Floating Points");

//Display the results with formatting
MessageBox.Show("With formatting: " +
                String.Format("{0:n3}", dblNumber),
                "Floating Points");
```

۳) برنامه را اجرا کنید. بعد از اینکه کادر پیغام مربوط به تست عمل ضرب نمایش داده شد، کادر پیغام بعدی عدد ۱,۷۱۴۲۸۵۷۱۴۲۸۵۷۱ را به عنوان نتیجه نمایش می دهد.
۴) هنگامی که روی OK کلیک کنید، کادر بعدی نتیجه ۱,۷۱۴ را نمایش خواهد داد.

چگونه کار می کند؟

تنها چیزی که در این کد ممکن است عجیب به نظر برسد، استفاده از تابع String.Format است. این تابع قوی، به شما اجازه میدهد متن و یا اعداد خود را قالب بندی کنید. تنها نکته مهمی که این تابع دارد، پارامتر اول آن است که مشخص می کند خروجی تابع باید در چه قالبی باشد.

```
//Display the results with formatting
MessageBox.Show("With formatting: " +
                String.Format("{0:n3}", dblNumber),
                "Floating Points");
```

برای فراخوانی متد String.Format باید دو پارامتر را به آن بفرستید. پارامتر اول، "{0:n3}"، قالب رشته ای است که شما میخواهید از متد دریافت کنید. پارامتر دوم، dblNumber، مقداری است که میخواهید قالب بندی کنید.
عدد صفر در پارامتر اول مشخص میکند که در حال مشخص کردن قالب اولین پارامتر بعد از پارامتر حاضر یا همان dblNumber هستید. قسمتی که بعد از ":" آمده است مشخص میکند که به چه صورت می خواهید رشته را قالب بندی کنید. در این جا از n3 استفاده کرده اید که به معنای "یک عدد با ممیز شناور و سه رقم اعشار" است. برای این که عدد را با دو رقم اعشار داشته باشید، به راحتی میتوانید از عبارت n2 استفاده کنید.

قالب بندی بومی:

زمانی که برنامه ای را با .NET می نویسید، باید بدانید که کاربر شما ممکن است از قواعد علامت گذاری در زبان خود استفاده کند که برای شما نا آشنا باشد. برای مثال اگر در ایالات متحده زندگی می کنید، از علامت نقطه (.) برای ممیز بین قسمت اعشار و قسمت صحیح عدد استفاده می کنید. اما برای کاربرانی که در فرانسه هستند این علامت یک ویرگول (،) است. همینطور ممکن است برای کاربران کشورهای دیگر این علامت تفاوت داشته باشد.
ویندوز میتواند بر اساس تنظیمات محلی کامپیوتر کاربر، این علامتها را برای شما انتخاب کند. اگر شما از چارچوب .NET درست استفاده کنید، عموما نیازی نیست در مورد این علامت گذاری ها نگران باشید. برای مثال در برنامه قبلی ما با استفاده از n3، به .NET گفتیم که می خواهیم عدد خود را با جدا کننده هزارگان و همچنین با سه رقم اعشار در سمت راست نمایش دهیم. البته تقسیم قسمت قبلی به جداکننده هزارگان نیازی ندارد اما اگر تقسیم را به ۷ / ۱۲۰۰۰ تغییر دهیم، عدد ۱،۷۴۲.۲۸۶ را دریافت می کنیم که در صورت داشتن جداکننده هزارگان، راحت تر خوانده می شود.
حال اگر کاربر تنظیمات محلی کامپیوتر خود را برابر تنظیمات فرانسوی قرار دهد و کد قبلی را اجرا کند (بدون اینکه هیچ تغیری را در کد به وجود آورید)، نتیجه ۱ ۷۱۴،۲۸۶ را دریافت خواهد کرد.

نکته: برای تغییر تنظیمات محلی کامپیوتر خود میتوانید به Control Panel بروید و روی آیکون Regional and Language Options کلیک کنید و زبان خود را به زبانی که تمایل دارید تغییر دهید. به این صورت علامت گذاری زبانی که انتخاب کرده اید در کامپیوتر شما استفاده می شود.

در زبان فرانسه، جداکننده هزارگان یک فضای خالی است نه یک ویرگول. همچنین برای ممیز اعشاری نیز از ویرگول استفاده میکنند نه از نقطه. با استفاده درست از متد String.Format میتوانید برنامه ای بنویسید که بر اساس تنظیمات محلی کاربر، اطلاعات را به نحوی صحیح نمایش دهد.

جایگزینی زیررشته ها:

یکی از کارهای عمومی دیگری که ممکن است هنگام کار با رشته ها به آن نیاز پیدا کنید، جایگزینی یک رشته خاص با رشته دیگری در یک متن است. برای توضیح چگونگی این کار، در بخش امتحان کنید بعدی، برنامه `Strings` خود را به گونه ای تغییر میدهیم که رشته `"Hello"` را با رشته `"Goodbye"` جایگزین کند.

امتحان کنید: جایگزینی زیررشته ها

۱) برنامه `Strings` را که قبلا ایجاد کرده بودید، باز کنید.
۲) کنترل `Button` دیگری را به `Form1` اضافه کنید، خاصیت `Name` آن را برابر `btnReplace` و خاصیت `Text` آن را برابر `Replace` قرار دهید. روی دکمه دو بار کلیک کنید و کد مشخص شده در زیر را به متد مربوط به رویداد `Click` آن اضافه کنید.

```
private void btnReplace_Click(object sender, EventArgs e)
{
     // Declare variables
     string strData;
     string strNewData;

     // Get the text from the TextBox
     strData = txtString.Text;

     // Replace the string occurance
     strNewData = strData.Replace("Hello", "Goodbye");

     // Display the new string
     MessageBox.Show(strNewData, "Strings");
}
```

۳) برنامه را اجرا کنید و جمله `Hello World!` را در جعبه متنی وارد کنید.
۴) روی دکمه `Replace` کلیک کنید. کادر پیغامی را مشاهده خواهید کرد که عبارت `Goodbye World!` را نمایش می دهد.

چگونه کار می کند؟

متد `Replace` دو رشته را دریافت میکند و در متن به دنبال رشته اول می گردد. سپس در هر قسمتی از متن که رشته اول را پیدا کرد آن را با رشته دوم جایگزین می کند. بعد از این که رشته اول در تمام متن با رشته دوم جایگزین شد، عبارت جدید به شما برگردانده می شود و می توانید آن را نمایش دهید.

```
// Replace the string occurance
strNewData = strData.Replace("Hello", "Goodbye");
```

با استفاده از این کد، فقط Hello اول با Goodbye جایگزین نمیشود، بلکه میتوانید در جعبه متنی، عبارتی را وارد کنید که چندین Hello داشته باشد. به این ترتیب تمام Helloهای متن با Goodbye جایگزین میشوند. اما به خاطر داشته باشید که کد بالا، کلمه Hello را با Goodbye جایگزین میکند نه کلمه hello را (یا هر حالت دیگری). به عبارت دیگر این تابع نسبت به کوچکی یا بزرگی حروف حساس است.

تبدیل نوع های داده ای:

در انجام محاسبات بر روی متغییر ها و یا ذخیره مقدار یک متغییر در متغییری دیگر، نوع داده ای آنها همواره باید یکسان باشد. برای مثال یک عدد صحیح را فقط می توان بر یک عدد صحیح دیگر تقسیم کرد و یا مقدار یک متغییر اعشاری را نمی توان به یک متغییر از نوع عدد صحیح نسبت داد[1]. در هنگام محاسبات اگر نوع های متغییر ها با هم تفاوت داشته باشد، یکی از متغییر ها به نوع داده ای متغییر دیگر تبدیل می شود. تبدیل نوع داده ای متغییر ها به دو روش می تواند انجام شود: به صورت اتوماتیک و به صورت صریح. در بیشتر مواقع این تبدیل نوع به صورت اتوماتیک توسط کامپایلر انجام می شود. اما در مواقعی که این تبدیل نوع ممکن است منجر به از دست رفتن اطلاعات شود، برنامه نویس باید آن را به صورت صریح انجام دهد.
برای مثال فرض کنید می خواهید مقدار یک متغییر از نوع اعشاری را در یک متغییر از نوع صحیح قرار دهید. در این حالت نمی توانید به صورت معمولی از عملگر تساوی استفاده کنید، زیرا با این کار قسمت اعشار عدد از بین میرود. به همین علت کامپایلر این تبدیل را به صورت اتوماتیک انجام نمی دهد. اما اگر بخواهید یک متغییر از نوع عدد صحیح را در یک متغییر از نوع اعشاری قرار دهید می توانید از عملگر تساوی استفاده کنید. زیرا با تغییر نوع از عدد صحیح به عدد اعشاری امکان از دست رفتن اطلاعات وجود ندارد و این تبدیل نوع به صورت خودکار توسط کامپایلر انجام می شود.
در بخش امتحان کنید بعد، چگونگی تبدیل نوع داده ای یک متغییر به طور صریح را بررسی خواهیم کرد
.

امتحان کنید: تبدیل نوع های داده ای

۱) در محیط ویژوال استودیو ۲۰۰۵ روی منوی File کلیک کنید و سپس New ➔ Project را انتخاب کنید. در پنجره New Project یک برنامه ویندوزی به نام Casting Demo ایجاد کنید.
۲) در قسمت طراحی Form1 یک کنترل Button قرار داده، خاصیت Name آن را به btnCast و Text آن را به Cast تغییر دهید.
۳) بر روی دکمه فرمان دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس کد مشخص شده در زیر را در آن وارد کنید.

```
private void btnCast_Click(object sender, EventArgs e)
{
```

[1] حتی اگر مقدار آن متغییر فاقد قسمت اعشار باشد، باز هم نمی توان مقدار آن را در یک عدد صحیح قرار داد.

```
      int intNumber = 2;
      double dblNumber = 3.4;

      intNumber = dblNumber;
      MessageBox.Show("The value of intNumber is: " +
                      intNumber);
}
```

۴) برنامه را اجرا کنید. مشاهده می کنید که کامپایلر در خط سوم برنامه با خطا مواجه می شود.

۵) کد متد `btCast_Click` را به صورت زیر تغییر دهید:

```
private void btnCast_Click(object sender, EventArgs e)
{
      int intNumber = 2;
      double dblNumber = 3.4;

      dblNumber = intNumber;
      MessageBox.Show("The value of dblNumber is: " +
                      dblNumber);
}
```

۶) برنامه را اجرا کنید و بر روی دکمه فرمان کلیک کنید. مشاهده خواهید کرد که مقدار متغییر `dblNumber` به عدد ۲ تغییر کرده است.

۷) کد موجود در متد را مجدداً تغییر دهید تا به صورت زیر درآید:

```
private void btnCast_Click(object sender, EventArgs e)
{
      int intNumber = 2;
      double dblNumber = 3.4;

      intNumber = (int)dblNumber;
      MessageBox.Show("The value of intNumebr is: " +
                              intNumber);
}
```

۸) برنامه را اجرا کرده و بر روی دکمه فرمان کلیک کنید. مشاهده خواهید کرد که مقدار متغییر `intNumber` تغیر کرده و برابر با قسمت صحیح عدد `dblNumber` شده است.

چگونه کار می کند؟

در این برنامه ابتدا سعی کرده ایم مقدار یک متغییر اعشاری را در یک متغییر از نوع صحیح قرار دهیم. همانطور که می دانید بـا ایـن کار بخش اعشار عدد از بین می رود. به همین دلیل کامپایلر این تبدیل را به صورت اتوماتیک انجام نمی دهد و برنامه با خطا مواجـه می شود.

```
intNumber = dblNumber;
MessageBox.Show("The value of intNumber is: " +
                intNumber);
```

در بخش بعد سعی کرده ایم مقدار یک متغییر از نوع صحیح را در یک متغییر اعشاری قرار دهیم. به علت اینکه در این حالت احتمـال از دست رفتن اطلاعات وجود ندارد و مقدار موجود در متغییر به صورت کامل در متغییر اعشاری ذخیره می شود، این تبدیل به صورت اتوماتیک توسط کامپایلر انجام می شود.

```
dblNumber = intNumber;
MessageBox.Show("The value of dblNumber is: " +
                dblNumber);
```

با اجرای برنامه و کلیک بر روی دکمه فرمان مشاهده خواهید کـرد کـه مقـدار متغییـر `intNumber` یعنـی عـدد ۲ در متغییـر `dblNumber` قرار گرفته است.
برای اینکه بتوانیم مقدار یک متغییر اعشاری را در یک متغییر از نوع عدد صحیح نگهداری کنیم باید به طـور صـریح آن را بـه عـدد صحیح تبدیل کنیم. در زبان C# این کار به وسیله عملگر پرانتز `( )` انجام می شود. برای تبدیل یک متغییر بـه یـک نـوع داده ای خاص و ذخیره آن، باید قبل از نام متغییر یک پرانتز قرار دهید و سپس داخل پرانتز نام نوع داده ای مقصد را وارد کنید. برای مثال:

```
intNumber = (int)dblNumber;
MessageBox.Show("The value of intNumebr is: " +
                        intNumber);
```

در این کد متغییر `dblNumber` که از نوع `double` است با استفاده از عملگر پرانتز بـه نـوع داده ای `int` تبـدیل شـده است و سپس در متغییری از این نوع ذخیره شده است.
البته دقت داشته باشید که از این روش برای تبدیل متغییر های رشته ای به عددی و بر عکس نمی توانید استفاده کنید. اگر رشته ای که شامل یک عدد است (مانند `"234.14"`) را بخواهید به عدد تبدیل کنید باید از تابع `Parse` در نوع داده ای عدد استفاده کنید. برای مثال فرض کنید متغییر `str1`، رشته ای حاوی یک عدد اعشاری است و می خواهیـد آن را در متغییـر `dblNum1` که از نوع اعشاری است ذخیره کنید. برای این کار باید از تابع `double.Parse()` به صورت زیر استفاده کنید.

```
dblNum1 = double.Parse(str1);
```

به همین ترتیب برای تبدیل یک رشته شامل عدد صحیح به عدد صحیح باید از تابع `int.Parse` استفاده کنید.
برای تبدیل یک عدد به رشته هم باید از تابع `ToString()` مربوط به متغییر عددی استفاده کنید. برای مثـال اگـر بخواهیـد مقدار متغییر `dblNumber` را در یک متغییر رشـته ای ذخیـره کنیـد بایـد از تـابع `dblNumber.ToString()` استفاده کنید.

استفاده از تاریخها:

یکی دیگر از انواع داده ای که کاربرد زیادی دارد و احتمالاً از آن زیاد استفاده خواهید کرد تاریخها هستند. این نـوع متغییـر هـا یـک تاریخ را در خود نگه می دارند. در امتحان کنید بعدی، با متغییر هایی که تاریخ را نگهداری میکنند بیشتر آشنا میشویم.

امتحان کنید: نمایش تاریخ روز

۱) یک پروژه ویندوزی جدید به نام `Date Demo` ایجاد کنید.

۲) با اسـتفاده از جعبـه ابـزار یـک کنتـرل `Button` بـه فـرم جدیـد خـود اضـافه کنیـد. خاصـیت `Name` آن را برابـر `btnDate` و خاصیت `Text` آن را برابر `Show Date` قرار دهید.

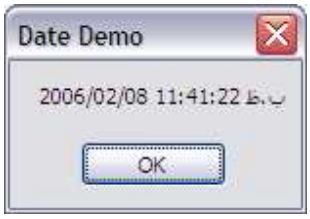
۳) روی دکمه دو بار کلیک کنید و کد زیر را به متد ایجاد شده اضافه کنید:

```
private void btnDate_Click(object sender, EventArgs e)
{
     // Declare variable
     DateTime dteDate;

     // Get the current date and time
     dteDate = DateTime.Now;

     // Display the results
     MessageBox.Show(dteDate.ToString(), "Date Demo");
}
```

۴) برنامه را اجرا کنید و روی دکمه `Show Date` کلیک کنید. کادر پیغامی ظاهر شده و تاریخ و سـاعت جـاری را (بـر اساس تنظیمات محلی کامپیوتر شما) همانند شکل ۳-۱۱ نمایش می دهد.



شکل ۳-۱۱

چگونه کار می کند؟

نوع داده ای DateTime میتواند یک مقدار را که معرف یک تاریخ و زمان خاص است، در خود نگهداری کند. بعد از این که متغییری از این نوع را ایجاد کردید، برای این که به آن مقدار اولیه بدهید میتوانید از خاصیت Now در این نوع داده ای استفاده کنید. این خاصیت مقدار تاریخ و زمان کنونی سیستم را برمیگرداند:

```
// Declare variable
DateTime dteDate;

// Get the current date and time
dteDate = DateTime.Now;
```

متغییر ها بر ای این که به وسیله متد MessageBox.Show نمایش داده شوند، باید به رشته تبدیل شوند. در .NET هر متغییری تابعی به نام ToString دارد که متغییر را به رشته تبدیل میکند[1]. در این جا برای این که بتوانیم متغییر dteDate را نمایش دهیم ابتدا باید آن را به رشته تبدیل کنیم که برای این کار از تابع ToString این متغییر استفاده کرده ایم:

```
// Display the results
MessageBox.Show(dteDate.ToString(), "Date Demo");
```

متغییر های تاریخ و زمان نیز همانند دیگر متغییر ها هستند، با این تفاوت که می توانند تاریخ و زمان را در خود نگهداری کنند و عملیات جمع و تفریق مربوط به آن را نیز انجام دهند. در بخشهای بعدی با نحوه کار با این متغییر ها و نمایش آنها بر روی صفحه نمایش به روشهای گوناگون، بیشتر آشنا خواهید شد.

قالب بندی تاریخها:

در قسمت قبلی یکی از حالتهای قالب بندی تاریخ را دیدیم. اگر متغییری از نوع تاریخ را با استفاده از تابع ToString به رشته تبدیل کنیم، تاریخ و زمان همانند شکل ۳-۱۱ نمایش داده خواهند شد.به دلیل اینکه تنظیمات محلی این کامپیوتر بر اساس ایران است، تاریخ به صورت YYYY/MM/DD و زمان نیز به صورت ۱۲ ساعته نمایش داده میشود. این نیز نمونه دیگری از تاثیر تنظیمات محلی کامپیوتر بر نحوه نمایش متغییر ها است. برای مثال اگر تنظیمات محلی کامپیوتر خود را برابر انگلیس قرار دهید، تاریخ در قالب DD/MM/YYYY و زمان نیز به صورت ۲۴ ساعته نمایش داده می شود.
البته میتوانید نحوه نمایش تاریخ را در برنامه مشخص کنید تا در هر سیستم با هر تنظیمات محلی به یک صورت نمایش داده شود، اما بهتر است که اجازه دهید .NET نحوه نمایش را به صورت اتوماتیک انتخاب کند، تا هر کاربر به هر نحوی که بخواهد آن را مشاهده کند. در بخش امتحان کنید بعد، با چهار روش مفید برای قالب بندی تاریخها آشنا خواهید شد.

امتحان کنید: قالب بندی تاریخها

[1] این در حالتی است که متغییر بتواند به رشته تبدیل شود. بعضی از متغییر ها که در فصلهای بعدی با آنها آشنا میشویم نمیتوانند به رشته تبدیل شوند. در این صورت، این تابع رشته ای را برمیگرداند که معرف متغییر مورد نظر است.

۱) اگر برنامه Date Demo در حال اجرا است آن را ببندید.

۲) با استفاده از ویرایشگر کد برای Form1، تابع مربوط به رویداد Click دکمه فرمان را پیدا کنید و کد مشخص شده در زیر را به آن اضافه کنید:

```
        // Display the results
        MessageBox.Show("Current Date is: " + dteDate,
                        "Date Demo");

        // Display dates
        MessageBox.Show(dteDate.ToLongDateString(),
                        "Date Demo");
        MessageBox.Show(dteDate.ToShortDateString(),
                        "Date Demo");

        // Display times
        MessageBox.Show(dteDate.ToLongTimeString(),
             "Date Demo");
        MessageBox.Show(dteDate.ToShortTimeString(),
                        "Date Demo");
}
```

۳) برنامه را اجرا کنید. با کلیک بر روی دکمه Show Date، پنج کادر پیغام نمایش داده میشوند. همانطور که مشاهده می کنید، کادر پیغام اول تاریخ و زمان را بر اساس تنظیمات محلی کامپیوتر شما نمایش میدهد. کادر دوم تاریخ را به صورت کامل و کادر سوم تاریخ را به صورت خلاصه شده نمایش می دهد. کادر چهارم زمان را به صورت کامل و کادر آخر زمان به صورت مختصر نمایش میدهد.

چگونه کار می کند؟

در برنامه قبلی هیچ نقطه مبهمی وجود ندارد و نام توابع به اندازه کافی واضح هستند و مشخص میکنند که هر تابع چه کاری انجام میدهد:

```
// Display dates
MessageBox.Show(dteDate.ToLongDateString(), "Date Demo");
MessageBox.Show(dteDate.ToShortDateString(),"Date Demo");

// Display times
MessageBox.Show(dteDate.ToLongTimeString(), "Date Demo");
MessageBox.Show(dteDate.ToShortTimeString(),"Date Demo");
```

استفاده از خاصیتهای DateTime:

هنگامی که یک متغییر از نوع DateTime داشته باشید، میتوانید از خاصیتهای گوناگونی که ارائه می دهد استفاده کنید. در امتحان کنید زیر، با قسمتی از آنها آشنا خواهیم شد.

امتحان کنید: استفاده از خاصیتهای DateTime

۱) اگر برنامه Date Demo در حال اجرا است، آن را ببندید.

۲) کنترل Button دیگری به Form1 اضافه کنید، خاصیت Name آن را برابر btnDateProperties و خاصیت Text آن را برابر Date Properties قرار دهید. روی دکمه دو بار کلیک کنید و کد مشخص شده در زیر را در متد ایجاد شده وارد کنید:

```
private void btnDateProperties_Click(object sender,
                                     EventArgs e)
{
     // Declare variable
     DateTime dteDate;

     // Get the current date and time
     dteDate = DateTime.Now;

     // Display the various properties
     MessageBox.Show("Month: " + dteDate.Month,
                     "Date Demo");
     MessageBox.Show("Day: " + dteDate.Day, "Date Demo");
     MessageBox.Show("Year: " + dteDate.Year,
                     "Date Demo");
     MessageBox.Show("Hour: " + dteDate.Hour,
                     "Date Demo");
     MessageBox.Show("Minute: " + dteDate.Minute,
                     "Date Demo");
     MessageBox.Show("Second: " + dteDate.Second,
                     "Date Demo");
     MessageBox.Show("Day of week: " + dteDate.DayOfWeek,
                "Date Demo");
     MessageBox.Show("Day of year: " + dteDate.DayOfYear,
                "Date Demo");
}
```

۳) برنامه را اجرا کنید و روی دکمه فرمان جدید کلیک کنید. یک سری کادرهای پیغام متوالی را خواهید دید که پیغام های واضحی را نمایش میدهند.

چگونه کار می کند؟

همانند قسمت قبل، در این مثال هم نکته مبهمی وجود ندارد که نیاز به توضیح داشته باشد. خود توابع به اندازه کافی واضح هستند. برای استفاده از ساعت، از خاصیت `Hour`، برای استفاده از سال از خاصیت `Year` و ... استفاده کنید.

کار با تاریخها:

یکی از مواردی که کنترل آن همیشه برای برنامه نویسان مشکل بوده است، کار با تاریخها است. یکی از این نمونه مشکلات که ممکن است هنگام کار با تاریخ در برنامه ها ایجاد شود، مشکل سال ۲۰۰۰ بود که در آن همه مردم منتظر بودند تا ببینند برنامه های کامپیوتری چگونه با این مشکل روبرو می شوند. یا مثلا، کار با سالهای کبیسه همواره مشکلات زیادی را در برنامه ها ایجاد کرده است.
در بخش امتحان کنید بعد، تعدادی از توابع و متدهای نوع داده ای `Date` که موجب ساده تر شدن کار با سالهای کبیسه در برنامه می شود را بررسی خواهیم کرد.

امتحان کنید: کار با تاریخهای خاص

۱) اگر برنامه `Date Demo` در حال اجرا است آن را ببندید.
۲) دکمه فرمان دیگری به فرم خود اضافه کنید، خاصیت `Name` آن را برابر `btnDateManipulation` و خاصیت `Text` آن را برابر `Date Manipulation` قرار دهید. روی دکمه فرمان دو بار کلیک کنید و کد مشخص شده در زیر را در متد ایجاد شده وارد کنید:

```
private void btnDateManipulation_Click(object sender,
                                       EventArgs e)
{
    // Declare variables
    DateTime dteStartDate;
    DateTime dteChangedDate;

    // Start off in 2400
    dteStartDate = new DateTime(2400, 2, 28);

    // Add a day and display the results
    dteChangedDate = dteStartDate.AddDays(1);
    MessageBox.Show(dteChangedDate.ToLongDateString(),
                    "Date Demo");

    // Add some months and display the results
```

```
        dteChangedDate = dteStartDate.AddMonths(6);
        MessageBox.Show(dteChangedDate.ToLongDateString(),
                  "Date Demo");

        // Subtract a year and display the results
        dteChangedDate = dteStartDate.AddYears(-1);
        MessageBox.Show(dteChangedDate.ToLongDateString(),
                  "Date Demo");
}
```

۳) برنامه را اجرا کنید و روی دکمه فرمان کلیک کنید. سه کادر پیغام را مشاهده خواهید کرد. کادر پیغام اولی تاریخ ۲۴۰۰/۲/۲۹، کادر پیغام دومی تاریخ ۲۴۰۰/۸/۲۸، و کادر پیغام سوم ۲۳۹۹/۲/۲۸ را نمایش خواهد داد.

چگونه کار می کند؟

نوع داده ای `DateTime` متدهای زیادی برای کار بر روی تاریخ دارد. در زیر سه نمونه از آنها آمده است:

```
// Add a day and display the results
dteChangedDate = dteStartDate.AddDays(1);
MessageBox.Show(dteChangedDate.ToLongDateString(),
          "Date Demo");

// Add some months and display the results
dteChangedDate = dteStartDate.AddMonths(6);
MessageBox.Show(dteChangedDate.ToLongDateString(),
          "Date Demo");

// Subtract a year and display the results
dteChangedDate = dteStartDate.AddYears(-1);
MessageBox.Show(dteChangedDate.ToLongDateString(),
          "Date Demo");
```

این توابع برای اضافه کردن روز، ماه و یا سال به تاریخ به کار می روند. همانطور که مشاهده میکنید، برای کم کردن از این مقادیر یک عدد منفی را به این توابع ارسال کرده ایم. متدهای مهم دیگر در این نوع داده ای عبارتند از: `AddHours`، `AddMinutes`، `AddSeconds` و `AddMiliseconds`.

بولین:

تاکنون با نوع هـای داده ای `int`، `float`، `Double`، `String` و `DateTime` آشـنا شـدید. نـوع داده ای مهـم دیگری که باید نحوه کار با آن را بدانید، `Boolean` است. بعد از اتمام این نوع داده ای،با پرکاربرد تـرین نـوع هـای داده ای در `.NET` آشنا شده اید.
یک متغییر بولین می تواند مقادیر `True` (درست) و یـا `False` (غلـط) را داشـته باشـد. متغییـر هـای `Boolean` بیـشتر هنگامی اهمیت خود را نشان می دهند که برنامه شما بخواهد صحت یک شرط را بررسی کند (با بررسی شرط ها در فـصل ۴ بیـشتر آشنا خواهید شد).

نگـهداری متغییر ها:

عموما محدود ترین قسمت یک سیستم، حافظه آن است. بنابراین برنامه شما باید بـه بهتـرین و کارآمـد تـرین نحـو از ایـن قـسمت استفاده کند. زمانی که یک متغییر (از هر نوعی) تعریف کنید، قسمتی از حافظه را اشغال کرده اید. بنابراین در تعریف متغییر هـا بایـد دقت کنید که هم کمترین مقدار متغییر ممکن را تعریف کنید و هم به بهترین نحو از آن استفاده کنید.
امروزه، به دو دلیل نیازی نیست که شما از جزئیات کامل بهینه سازی متغییر ها آگاه باشید. دلیـل اول ایـن اسـت کـه کامپیوترهـای امروزی مقدار زیادی حافظه دارند. از زمانی که لازم بود برنامه نویسان، برنامه های خود را در ۳۲ کیلو بایـت از حافظـه کـامپیوتر جـا بدهند خیلی گذشته است. دلیل دیگر این است که کامپایلر های امروزی به صورت درونی، به اندازه کافی در این زمینه هوشمند عمل میکنند. به عبارت دیگر، کدهای تولید شده توسط آنها تا حد ممکن بهینه میشود تا بهترین کارایی را داشته باشند.

دودویی:

کامپیوترها برای نگهداری هر اطلاعاتی از سیستم دودویی استفاده میکنند. در حقیقت هر داده ای را که شـما در کـامپیوتر نگهـداری میکنید باید در قالب صفر و یک ذخیره شوند. برای نمونه عدد صحیح ۲۷ را در نظر بگیرید. در سیستم باینری یا دودویی این عدد بـه صورت ۱۱۰۱۱ نمایش داده می شود، که هر کدام از این اعداد در توانی از دو ضرب میشوند. نمودار شکل ۳-۱۲ به شما کمک میکند تا عدد ۲۷ را بهتر در سیستم ده دهی و در سیستم دودویی یا باینری نمایش دهید.
ممکن است در ابتدا این مورد مقداری نامفهوم به نظر برسد. در مبنای ۱۰ یا همان سیستم ده دهی که با آن آشنا هستید، برای تعیین ارزش هر عدد، هر رقم از آن در توانی از ده ضرب میشود. اولین رقم عدد از سمت راست در ده به توان صفر ضرب میشود، عـدد دوم در ده به توان یک ضرب میشود، عدد سوم در ده به توان دو ضرب میشود و به همین ترتیب ادامه پیدا میکند.
همین روش در سیستم باینری هم وجود دارد. برای تبدیل یک عدد در مبنای دو به یک عدد در مبنـای ده، بایـد رقمهـا را از سـمت راست یکی یکی جدا کنید و در دو به توان شماره مکان عدد ضرب کنید (عدد اول از سمت راست در مکان صفرم، عدد دوم در مکان یکم و ... قرار دارد). سپس تمام اعداد به دست آمده را با هم جمع کنید. عددی که به دست می آیـد، نمـایش دهنـده همـان عـدد در مبنای ده است.

۱۰۷	۱۰۶	۱۰۵	۱۰۴	۱۰۳	۱۰۲	۱۰۱	۱۰۰
۱۰۰۰۰۰۰۰	۱۰۰۰۰۰۰	۱۰۰۰۰۰	۱۰۰۰۰	۱۰۰۰	۱۰۰	۱۰	۱
۰	۰	۰	۰	۰	۰	۲	۷

۲ ✻ ۱۰ + ۷ ✻ ۱ = ۲۷

در مبنای ۱۰ هر رقم ضریبی از یکی از توانهای ۱۰ است. برای اینکه مشخص کنید یک رشته از ارقام که در مبنای ۱۰ نوشته شده اند چه عددی را نشان می دهند، کافی است رقم اول را در ۱۰ به توان ۰، رقم دوم را در ۱۰ به توان ۱ و ... ضرب کرده و حاصل را با یکدیگر جمع کنید.

۲۷	۲۶	۲۵	۲۴	۲۳	۲۲	۲۱	۲۰
۱۲۸	۶۴	۳۲	۱۶	۸	۴	۲	۱
۰	۰	۰	۱	۱	۰	۱	۱

۱✻۱۶ + ۱✻۸ + ۰✻۴ + ۱✻۲ + ۱✻۱ = ۲۷

در مبنای ۲ یا اعداد باینری، هر رقم ضریبی از یکی از توانهای ۲ است. برای تشخیص اینکه یک رشته ارقام در مبنای ۲ چه عددی را نمایش می دهند، باید رقم اول را در ۲ به توان ۰، رقم دوم را در ۲ به توان ۱ و ... ضرب کرده و حاصل را با یکدیگر جمع کرد.

شکل ۳-۱۲

بیتها و بایت ها:

در اصطلاحات کامپیوتری، به یک رقم باینری، یک بیت میگویند. یک بیت کوچکترین بخش ممکن برای نمایش داده است و فقط میتواند دارای مقدار درست و یا غلط باشد. این مقدار در مدارهای کامپیوتر به وسیله وجود داشتن و یا وجود نداشتن جریان مشخص می شود. دلیل اینکه در هر قسمت از شکل ۳-۱۲ هشت بیت در کنار هم نشان داده شده اند، این است که هر هشت بیت در کنار هم یک بایت را تشکیل می دهند. بایت واحد اندازه گیری حافظه های کامپیوتر به شمار میرود.

یک کیلو بایت یا KB از ۱۰۲۴ بایت تشکیل شده است. دلیل اینکه به جای ۱۰۰۰ از ۱۰۲۴ بایت تشکیل شده است، این است که ۱۰۲۴ توانی از دو است (دو به توان ده). به دلیل اینکه اعداد در کامپیوتر در مبنای دو ذخیره میشوند، نگهداری ۱۰۲۴ راحت تر از نگهداری ۱۰۰۰ است. همانطور که نگهداری ۱۰۰۰ برای شما که از سیستم ده دهی استفاده میکنید راحت تر از ۱۰۲۴ است.

به همین ترتیب یک مگا بایت یا MB برابر ۱۰۲۴ کیلو بایت یا ۱۰۴۸۵۷۶ بایت، یک گیگا بایت ۱۰۲۴ مگا بایت یا ۱۰۷۳۷۴۱۸۲۴ بایت است (دو به توان سی). همچنین ترا بایت برابر دو به توان چهل و پتا بایت برابر دو به توان پنجاه است.

همه این مطالب به چه کار می آید؟ خوب، دانستن این که کامپیوتر چگونه اطلاعات را ذخیره میکند ممکن است به شما کمک کند که برنامه های خود را بهتر طراحی کنید. فرض کنید کامپیوتر شما ۲۵۶ مگا بایت حافظه دارد. این مقدار حافظه برابر ۲۱۴۷۴۸۳۶۴۸ بایت است. بنابراین هنگامی که در حال نوشتن نرم افزار هستید باید سعی کنید که از حافظه به بهترین نحو ممکن استفاده کنید.

نمایش مقادیر:

بیشتر کامپیوترهای امروزی ۳۲ بیتی هستند، یعنی در هر لحظه میتوانند با داده هایی به طول ۳۲ بیت کار کنند. اعدادی که در بخش قبلی دیدید همگی اعداد هشت بیتی بودند. در یک عدد هشت بیتی حداکثر مقداری را که می توان نگهداری کرد به صورت زیر است:

۱*۱۲۸ + ۱*۶۴ + ۱*۳۲ + ۱*۱۶ + ۱*۸ + ۱*۴ + ۱*۲ + ۱*۱ = ۲۵۶

هر عدد ۳۲ بیتی می تواند مقادیری در محدوده ۲۱۴۷۴۸۳۶۴۷- تا ۲۱۴۷۴۸۳۶۴۷ را نگهداری کند. حال اگر بخواهید یک عدد صحیح را نگهداری کنید متغییری را به صورت زیر تعریف میکنید:

```
int intNumber;
```

در اینجا، .NET ۳۲ بیت از حافظه کامپیوتر را برای این متغییر اختصاص میدهد که شما میتوانید اعداد در بازه ای که گفته شد را در آن ذخیره کنید. اما به خاطر داشته باشید که مقدار حافظه کامپیوتر شما محدود است. برای مثال اگر ۲۵۶ مگابایت حافظه داشته باشد، میتوانید ۶۷۱۰۸۸۶۴ عدد صحیح را در آن نگهداری کنید. ممکن است زیاد به نظر برسد. اما باید دو نکته را در نظر داشته باشید. اول اینکه بعضی از متغییرها که در فصلهای بعدی با آنها آشنا میشویم بسیار بیشتر از یک عدد صحیح فضا اشغال می کنند. دوم اینکه به خاطر داشته باشید حافظه بین همه برنامه ها مشترک است و نباید آن را هدر بدهید.
همچنین برای نگهداری یک عدد اعشاری با دقت مضاعف، متغییری به صورت زیر تعریف میکردید:

```
double dblNumber;
```

برای نگهداری یک عدد اعشاری با دقت مضاعف به ۶۴ بیت از حافظه نیاز دارید. یعنی میتوانید حداکثر ۳۳۵۵۴۴۳۲ عدد اعشاری با دقت مضاعف در حافظه کامپیوتر خود داشته باشید.

نکته: اعداد اعشاری با دقت معمولی ۳۲ بیت از حافظه را اشغال می کنند، یعنی این اعداد نصف اعداد اعشاری با دقت مضاعف و به اندازه ی اعداد صحیح فضا میگیرند.

اگر یک متغییر از نوع صحیح تعریف کنید، چه در آن ۱ را ذخیره کنید، چه ۲۴۹ و یا ۲۱۴۷۴۸۳۶۴۷، دقیقا ۳۲ بیت از فضای حافظه را اشغال کرده اید. اندازه عدد هیچ تاثیری در اندازه فضای مورد نیاز برای ذخیره آن ندارد. این مورد ممکن است هدر رفتن حافظه به نظر رسد. اما دقت داشته باشید که کامپیوتر تصور میکند اعداد از نوع یکسان فضای یکسانی برای نگهداری نیاز دارند. در غیر این صورت، نمیتواند با سرعت قابل قبولی کار کند.
حالا به چگونگی تعریف یک رشته نگاه کنید:

```
string strData = "Hello, World!";
```

بر خلاف اعداد صحیح و اعشاری، رشته ها دارای طول ثابتی نیستند. در رشته ها، هر کاراکتر دو بایت یا ۱۶ بیت از فضای حافظه را اشغال میکند. بنابراین برای نمایش این رشته ی ۱۳ کاراکتری، به ۲۶ بایت یا ۲۰۸ بیت فضا در حافظه نیاز دارید. حافظه کامپیوتری که پیشتر مثال زدیم حدود دو میلیون کاراکتر را میتواند نگهداری کند که نسبت به اعداد صحیح و یا اعداد اعشاری بسیار کمتر است.
اشتباهی که بیشتر برنامه نویسان تازه کار انجام میدهند این است که کمتر به تاثیر نوع ذخیره داده ها در صرفه جویی حافظه فکر می کنند. برای نمونه اگر شما یک متغییر برای نگهداری رشته تعریف کنید و سپس عدد صحیح در آن نگهداری کنید، مانند زیر:

```
string strData = "65536";
```

در اینجا برای نگهداری این عدد صحیح در قالب رشته، از ۱۰ بایت یا ۸۰ بیت استفاده کرده اید. در صورتی که میتوانستید با تعریف این متغییر از نوع عدد صحیح حافظه کمتری استفاده کنید. برای نگهداری این رشته عددی در حافظه هر کدام از این کاراکترها باید به یک عدد خاص تبدیل شوند و سپس در حافظه ذخیره شوند. برای ذخیره متن در حافظه، بر اساس استانداردی به نام یونیکد[1] به هر کاراکتر یک کد خاص داده میشود. به عبارت دیگر هر کاراکتر یک کد از ۰ تا ۶۵۵۳۵ دارد و برای ذخیره آن کاراکتر در حافظه، کد معادل آن ذخیره می شود.
در زیر کد مربوط به هر کاراکتر را در رشته بالا آورده ایم:

- "6" یونیکد ۵۴ که کد باینری معادل آن برابر ۰۰۰۰۰۰۰۰۰۰۱۱۰۱۱۰ است.
- "5" یونیکد ۵۳ که کد باینری معادل آن برابر ۰۰۰۰۰۰۰۰۰۰۱۱۰۱۰۱ است.
- "5" یونیکد ۵۳ که کد باینری معادل آن برابر ۰۰۰۰۰۰۰۰۰۰۱۱۰۱۰۱ است.
- "3" یونیکد ۵۱ که کد باینری معادل آن برابر ۰۰۰۰۰۰۰۰۰۰۱۱۰۰۱۱ است.
- "6" یونیکد ۵۴ که کد باینری معادل آن برابر ۰۰۰۰۰۰۰۰۰۰۱۱۰۱۱۰ است.

هر کدام از این کدها برای ذخیره شدن به دو بایت فضا نیاز دارند. پس برای ذخیره ۵ رقم در قالب رشته به ۸۰ بیت فضا نیاز داریم. برای ذخیره عدد بالا باید متغییر را به صورت زیر تعریف کنیم:

```
int intNumber = 65536;
```

مقدار این متغییر به صورت یک عدد در قالب باینری در حافظه ذخیره میشود و از آنجا که هر عدد صحیح ۳۲ بیت فضا را اشغال می کند، این عدد هم ۳۲ بیت فضا در حافظه اشغال میکند که بسیار کمتر از ۸۰ بیت اشغال شده به وسیله رشته است.

متدها:

یک متد، یک تکه کد است که وظیفه خاصی را انجام می دهد. متدها که پروسیجر هم نامیده میشوند به دو دلیل اهمیت زیادی دارند. دلیل اول این است که آنها برنامه را به قسمتهای کوچکتر تقسیم می کنند و موجب میشوند که برنامه بهتر درک شود. دوم اینکه آنها قابلیت استفاده مجدد از کدها را افزایش می دهند. استفاده مجدد از کدها به اندازه ای مهم است که تا آخر این کتاب بیشتر وقت خود را صرف آن می کنید.
همانطور که میدانید، هنگامی که شروع به نوشتن یک برنامه میکنید، ابتدا باید الگوریتم کلی آن را تهیه کرده و سپس جزئیات هر قسمت از الگوریتم را به کد تبدیل کنید تا کل بخشهای الگوریتم را به صورت کد داشته باشید، سپس با کنار هم قرار دادن این کدها به الگوریتم کلی میرسید. یک متد، یکی از خطهای این الگوریتم کلی را اجرا می کند، برای مثال "یک فایل را باز کن"، "یک متن را روی صفحه نمایش بده"، "یک صفحه را چاپ کن" و یا مواردی از این قبیل.
دانستن این که چگونه یک برنامه را به چند قسمت کوچکتر تقسیم کنید موردی است که به تجربه بستگی دارد. اهمیت تقسیم کردن یک برنامه به چند قسمت کوچک و تاثیر آن در سادگی برنامه را زمانی مشاهده خواهید کرد که برنامه های بسیار پیچیده تری نسبت

[1] Unicode

به آنهایی که تاکنون نوشته اید، بنویسید. در ادامه ی این بخش سعی می کنیم به شما بگوییم چرا و چگونه باید از متدها استفاده کنیم.

چرا از متدها استفاده می کنیم؟

در استفاده از متدها، شما باید اطلاعاتی که یک متد برای اجرا به آنها نیاز دارد را فراهم کنید تا نتیجه ی مطلوبی دریافت کنید. این اطلاعات ممکن است یک عدد صحیح، یک رشته متنی و یا ترکیبی از هر دو باشد. این اطلاعات به عنوان **مقادیر ورودی** شناخته میشوند. البته بعضی از متدها ورودی دریافت نمی کنند، بنابراین داشتن مقدار ورودی برای یک متد لازم نیست. یک متد با استفاده از این اطلاعات ورودی و نیز یک سری اطلاعات درونی (برای مثال دانستن اطلاعاتی در رابطه با وضعیت کنونی برنامه) سعی میکند تا وظیفه خود را انجام دهد.
هنگامی که این اطلاعات را به برنامه می فرستید، در اصطلاح **داده** به تابع فرستاده اید. به این داده ها **پارامتر** هم گفته میشود. در نهایت برای استفاده از یک تابع شما باید آن را **فراخوانی** کنید. خلاصه، شما یک متد را فراخوانی میکنید و داده های مورد نیاز آن را به وسیله پارامتر ها به آن میفرستید.
همانطور که در قبل هم ذکر شد، دلیل استفاده از یک تابع این است که از یک قطعه کد چندین بار استفاده کنیم. برای این کار ابتدا باید بتوانیم الگوریتم برنامه را در حالت کلی بررسی کنیم و سپس قسمتهایی که ممکن است چند بار به آنها نیاز پیدا کنیم را مشخص کنیم. بعد از مشخص کردن این قسمتها، میتوانیم آنها را با استفاده از متدها بنویسیم و سپس متدها را چندین بار در برنامه استفاده کنیم.
برای مثال تصور کنید که برنامه شما از الگوریتم های زیادی تشکیل شده است. بعضی از این الگوریتم ها برای محاسبات خود نیاز دارند که بتوانند محیط دایره را محاسبه کنند. چون بعضی از قسمتهای الگوریتم ما نیاز دارند که نحوه محاسبه محیط دایره را بدانند، استفاده از یک متد در این مورد میتواند مناسب باشد. شما میتوانید متدی بنویسید که بتواند محیط یک دایره را با دانستن شعاع آن محاسبه کند، سپس در هر قسمت از الگوریتم که به محاسبه محیط نیاز بود میتوانید این متد را فراخوانی کنید. به این ترتیب نیازی نیست کدی که یک کار مشخص را انجام میدهد چند بار نوشته شود. شما فقط یک بار کد را می نویسید و چند بار از آن استفاده میکنید.
همچنین ممکن است یک قسمت از الگوریتم بخواهد محیط یک دایره به شعاع ۱۰۰ را بداند و قسمتی دیگر محیط یک دایره به شعاع ۲۰۰. بنابراین متد شما میتواند شعاع دایره را به عنوان پارامتر از ورودی بگیرد و سپس محیط را محاسبه کند. به این ترتیب این متد در هر شرایطی میتواند مورد استفاده قرار بگیرد.

نکته: در ویژوال C# ۲۰۰۵ یک متد یا میتواند مقداری را برگرداند و یا هیچ مقداری را برنگرداند. به متدهایی که مقداری را برمیگردانند یک **تابع**[1] و به متدهایی که هیچ مقداری را برنمیگردانند یک **زیربرنامه**[2] گفته می شود.

متدهایی که تاکنون دیده اید:

[1] Function
[2] Subroutine

بهتر است بدانید که تاکنون در برنامه هایی که در قسمتهای قبلی نوشته ایم، از متدهای زیادی استفاده کرده اید. به عنوان مثال کد زیر را که در ابتدای فصل نوشتید ملاحظه کنید:

```
private void btnAdd_Click(object sender, EventArgs e)
{
     // Define a variable for intNumber
     int intNumber;

     // Set the initial value
     intNumber = 27;

     // Add 1 to the value of intNumber
     intNumber = intNumber + 1;

     // Display the new value of intNumber
     MessageBox.Show("The value of intNumber + 1 = " +
               intNumber, "Variables");
}
```

این کد یک متد است زیرا همانطور که در تعریف متد گفتیم، قطعه کدی مجزا است که کار خاصی را انجام می دهد. در اینجا این قطعه کد عدد یک را به متغییر `intNumber` اضافه میکند و نتیجه را نمایش میدهد.
این متد هیچ مقداری را بر نمی گرداند بنابراین نوع مقدار بازگشتی `void` تعریف شده است. کلمه **`void`** به کامپایلر می گوید که این متد هیچ مقداری را بر نمی گرداند. اگر متدی مقداری را برگرداند به جای استفاده از `void` باید نوع مقداری که برگشت داده می شود را بنویسید (برای مثال `int` یا `double` یا ...). هر دستوری که بین دو علامت آکولاد نوشته شود جزئی از بدنه ی متد محسوب می شود. یک متد به صورت زیر تعریف می شود (البته این متد به صورت اتوماتیک توسط ویژوال C# ۲۰۰۵ ایجاد شده است):

```
private void btnAdd_Click(object sender, EventArgs e)
```

۱) قبل از هر چیز، کلمه `private` را در تعریف تابع مشاهده میکنید. درباره این کلمه در فصلهای بعدی به تفصیل بحث شده است. فعلا، فقط بدانید که این کلمه موجب میشود این متد فقط توسط توابع و یا متدهای دیگر داخل همان بخش استفاده شود.
۲) کلمه بعدی، کلمه `void` است که همانطور که گفتیم به ویژوال C# می گوید این متد هیچ مقداری را برنمی گرداند.
۳) بعد از `void` با کلمه `btnAdd_Click` روبرو میشوید. همانطور که ممکن است حدس زده باشید، این کلمه نام متدی است که تعریف کرده ایم.
۴) چهارمین مورد در تعریف این تابع پارامترهایی است که به آن فرستاده می شود. همانطور که مشاهده میکنید این تابع دو پارامتر را دریافت میکند. پارامتر اول، `sender` از نوع `object` است و پارامتر دوم `e` از نوع `EventArgs` است. در مورد این پارامتر ها در بخشهای بعدی بیشتر صحبت خواهیم کرد.

در امتحان کنید بعدی، یک متد ایجاد خواهید کرد که یک عبارت را با استفاده از کادر پیغام در صفحه نمایش دهد و سپس این متد را به وسیله سه دکمه فرمان متفاوت فراخوانی خواهید کرد.

امتحان کنید: استفاده از متدها

۱) یک پروژه ویندوزی با ویژوال C# ۲۰۰۵ ایجاد کنید و نام آن را برابر Three Buttons قرار دهید.
۲) با استفاده از جعبه ابزار سه دکمه فرمان بر روی فرم خود قرار دهید.
۳) روی دکمه فرمان اول دو بار کلیک کنید و سپس کد مشخص شده در زیر را به آن اضافه کنید:

```
private void button1_Click(object sender, EventArgs e)
{
     // Call your method
     SayHello();
}

private void SayHello()
{
     // Display a message box
     MessageBox.Show("Hello, World!", "Three Buttons");
}
```

۴) برنامه را اجرا کنید. فرمی را با سه دکمه فرمان مشاهده خواهید کرد. بر روی بالاترین دکمه فرمان کلیک کنید. کادر پیغامی با عبارت Hello, World! را خواهید دید.

چگونه کار می کند؟

همانطور که می دانید، هنگامی که در محیط طراحی بر روی یک دکمه فرمان دو بار کلیک می کنید، ویژوال C# ۲۰۰۵ تابعی مانند زیر را به صورت اتوماتیک ایجاد می کند:

```
private void button1_Click(object sender, EventArgs e)
{

}
```

هنگامی که کاربر روی این دکمه در برنامه کلیک کند، اصطلاحا یک **رویداد**[1] در برنامه رخ داده است. ویژوال C# ۲۰۰۵ هنگام کامپایل این متد، کدی را به آن اضافه می کند تا این متد هنگام رخ دادن رویداد کلیک این دکمه اجرا شود. به عبارت دیگر ویژوال C# زمانی که کاربر روی این دکمه فرمان در برنامه کلیک کرد، این متد را فراخوانی می کند. این متد دو پارامتر را به عنوان ورودی دریافت می کند، که فعلا در مورد آنها صحبتی نمی کنیم. در فصلهای بعد بیشتر با رویدادها و نحوه استفاده از آنها در برنامه آشنا خواهید شد.

[1] Event

در بیرون از این متد، متد دیگری را به صورت زیر تعریف می کنید:

```
private void SayHello()
{
      // Display a message box
      MessageBox.Show("Hello, World!", "Three Buttons");
}
```

این متد جدید، SayHello نامیده میشود. هر چیزی که در بین دو آکولاد متد قرار بگیرد، جزئی از کد این متد محسوب میشود و هنگامی که متد فراخوانی شود، اجرا خواهد شد. در این حالت، کد نوشته شده در این متد یک کادر پیغام را نمایش می دهد.
همانطور که میدانید با کلیک کردن بر روی دکمه فرمان، ویژوال C# ۲۰۰۵ متد button1_Click را اجرا میکند و این متد نیز SayHello را احضار میکند. نتیجه این که با کلیک بر روی دکمه فرمان، کد داخل متد SayHello اجرا میشود و کادر پیغامی نمایش داده میشود.

```
private void button1_Click(object sender, EventArgs e)
{
      // Call your method
      SayHello();
}
```

تا اینجا مقداری با تعریف تابع و نیز استفاده از آن آشنا شدید، اما سوالی که ممکن است به وجود آید این است که چرا نیاز دارید متدی بنویسید که کادر پیغامی را نمایش دهد؟ در بخش امتحان کنید بعد دلیل این کار را بیشتر متوجه خواهید شد.

امتحان کنید: استفاده مجدد از متدها

۱) اگر پروژه قبلی در حال اجرا است، آن را ببندید.
۲) روی دکمه فرمان دوم دو بار کلیک کنید و کد زیر را در متد ایجاد شده وارد کنید:

```
private void button2_Click(object sender, EventArgs e)
{
      // Call your method
      SayHello();
}
```

۳) به قسمت طراحی برگردید و روی دکمه فرمان سوم دو بار کلیک کنید. در متد ایجاد شده کد زیر را وارد کنید:

```
private void button3_Click(object sender, EventArgs e)
{
      // Call your method
      SayHello();
}
```

۴) حالا برنامه را اجرا کنید و روی دکمه های فرمان کلیک کنید. مشاهده می کنید که همه دکمه ها کادر پیغـام یکـسانی را نمایش می دهند.

۵) اجرای برنامه را متوقف کنید و متد `SayHello` را در بخش کد برنامه پیدا کنید. متنی که در کادر پیغام نمـایش داده میشود را به صورت زیر تغییر دهید:

```
private void SayHello()
{
     // Display a message box
     MessageBox.Show("I have changed!", "Three Buttons");
}
```

۶) برنامه را مجددا اجرا کنید و روی دکمه های فرمان کلیک کنید. توجه کنید که متن مورد نمایش در کادر پیغام تغییر کـرده است.

چگونه کار می کند؟

هر کدام از متدهای مخصوص به رویداد کلیک دکمه های فرمان[1]، متد `SayHello` را فراخوانی می کنند:

```
private void button1_Click(object sender, EventArgs e)
{
     // Call your method
     SayHello();
}

private void button2_Click(object sender, EventArgs e)
{
     // Call your method
     SayHello();
}


private void button3_Click(object sender, EventArgs e)
{
     // Call your method
     SayHello();
}
```

همانطور که از اسم این متدها نیز مشخص است، هر کدام از آنها برای کنترل رویداد یکی از دکمه فرمانهای روی فرم هستند.

[1] به این متدها که برای کنترل یک رویداد به کار می روند، `Event Handler` نیز گفته می شود.

توجه کنید هنگامی که شما نحوه کارکرد متد SayHello را که هر سه متد از آن استفاده می کنند تغییر دهید، تغییرات بـه هـر سه دکمه فرمان اعمال میشود. این مورد در برنامه نویسی موضوع بسیار مهمی است. اگر شما متدهای کد خود را به نحـوی منطقـی تنظیم کنید، میتوانید با تغییر یک متد در قسمتی از برنامه، بر روی تمام قسمتهای برنامه تاثیر بگذارید. همچنین این مـورد شـما را از وارد کردن کدهای یکسان و مشابه به صورت تکراری نیز نجات میدهد.

ایجاد یک متد:

در امتحان کنید بعد، متدی خواهید ساخت که یک مقدار را برگرداند. تابعی که در این بخش ایجاد می کنید میتواند مساحت یک دایره را بر اساس شعاع آن محاسبه کند. الگوریتمی که برای این کار استفاده میکنید به صورت زیر است:

۱) شعاع را به توان دو برسان.
۲) نتیجه را در عدد پی ضرب کن.

امتحان کنید: ایجاد یک متد

۱) برای نوشتن این برنامه میتوانید از برنامه Three Buttons که در مرحله قبل ساختید استفاده کنید.
۲) کد زیر را برای تعریف یک تابع جدید در قسمت کـد برنامـه اضـافه کنیـد (ایـن متـد مقـداری را بـه برنامـه ی فراخـوان برمیگرداند، پس یک تابع به شمار می رود).

```
// CalculateAreaFromRadius - find the area of a circle
private double CalculateAreaFromRadius(double radius)
{
     // Declare variables
     double dblRadiusSquared;
     double dblResult;

     // Square the radius
     dblRadiusSquared = radius * radius;

     // Multiply it by pi
     dblResult = dblRadiusSquared * Math.PI;

     // Return the result
     return dblResult;
}
```

۳) سپس کد نوشته شد در متد button1_Click را حذف کنید و کد زیر را به جای آن وارد کنید:

```
private void button1_Click(object sender, EventArgs e)
```

```
{
      // Declare variable
      double dblArea;
      // Calculate the area of a circle with radius 100
      dblArea = CalculateAreaFromRadius(100);
      // Print the results
      MessageBox.Show(dblArea, "Area");
}
```

۴) برنامه را اجرا کنید و روی دکمه فرمان Button1 کلیک کنید. نتیجه ای را مشابه شکل ۳-۱۳ مشاهده خواهید کرد:



شکل ۳-۱۳

چگونه کار می کند؟

قبل از هر چیز، یک متد مجزا به نام CalculateAreaFromRadius مانند زیر ایجاد می کنید:

```
private double CalculateAreaFromRadius(double radius)
{

}
```

هر کدی که بین دو آکولاد قرار بگیرد به عنوان بدنه تابع به شمار می رود و با فراخوانی تابع اجرا میشود.
قسمت double radius یک پارامتر از نوع double را برای این متد تعریف می کند. هنگامی که یک متد فراخوانی می شود، NET. قسمتی از حافظه را در اختیار آن قرار می دهد تا متد بتواند از آن در انجام دستورات خود استفاده کند. پارامترهایی که به این ترتیب در بدنه متد تعریف شود، هنگام فراخوانی متد در این فضا کپی می شوند تا متد بتواند از آنها استفاده کند. این عمل موجب می شود هر تغیری که متد در مقدار پارامتر ایجاد کند، در متغییر اصلی تاثیری نداشته باشد.
مثلا در این برنامه، هنگام احضار متد یک متغییر به نام radius در حافظه ی مخصوص متد تعریف می شود و مقدار ارسالی به متد به صورت اتوماتیک در آن قرار می گیرد. بنابراین اگر مقدار ۲۰۰ به متد ارسال شود مقدار متغییر تعریف شده برابر ۲۰۰ خواهد بود. این مورد همانند این است که در ابتدای متد یک متغییر به نام radius تعریف کنید و مقدار آن را برابر ۲۰۰ قرار دهید:

```
double radius = 200;
```

نکته: یکی از مشکلات این روش این است که اگر اندازه متغییر بزرگ باشد، کپی کردن آن در از برنامه اصلی در حافظه ی مخصوص به متد زمان زیادی را می گیرد و همچنین موجب می شود حافظه ی زیادی اشغال شود. برای رفع این مشکل می توان

متغییر های بزرگ را به روش دیگری به نام **ارجاع**[1] به متدها فرستاد. برای اینکه یک پارامتر با استفاده از ارجاع به متد فرستاده شود باید قبل از تعریف پارامتر از کلمه کلیدی `ref` استفاده کنید. هنگامی که یک پارامتر با استفاده از ارجاع به یک متد ارسال شود، دیگر مقدار پارامتر در حافظه ی متد کپی نمی شود بلکه آدرس متغییر اصلی در حافظه، به متد فرستاده خواهد شد. بنابراین هر تغییری که متد در مقدار پارامتر ایجاد کند، بلافاصله در متغییر اصلی نیز تاثیر خواهد گذاشت و مقدار آن نیز تغییر خواهد کرد.

کلمه `double` که قبل از نام متد آمده است، مشخص می کند که این متد مقداری را از نوع عدد اعشاری با دقت مضاعف به جایی که آن احضار شده است برمی گرداند:

```
private double CalculateAreaFromRadius(double radius)
```

حالا بهتر است به بدنه خود متد نگاه دقیقتری بی اندازید. قبل از هر چیز، میدانید که برای محاسبه مساحت یک دایره باید از الگوریتم زیر استفاده کنید:

۱) عددی که بیانگر شعاع دایره است را دریافت کنید.
۲) عدد را به توان دو برسانید.
۳) حاصل را در عدد پی ضرب کنید.

و همانطور که می بینید، این همان کاری است که در بدنه متد انجام داده اید:

```
// Declare variables
double dblRadiusSquared;
double dblResult;

// Square the radius
dblRadiusSquared = radius * radius;

// Multiply it by pi
dblResult = dblRadiusSquared * Math.PI;
```

عبارت `Math.PI` در کد قبلی مقدار ثابتی است که در ویژوال C# ۲۰۰۵ تعریف شده است و مقدار عدد پی را در خود نگهداری می کند. در خط آخر باید نتیجه محاسبات را به برنامه فراخوان بازگردانید. این کار با کد زیر انجام می شود:

```
// Return the result
return dblResult;
```

کدی که در متد `button1_Click` اضافه کرده اید، تابع قبلی را احضار می کند و نتیجه برگردانده شده توسط تابع را به کاربر نمایش می دهد:

```
// Declare variable
```

[1] Reference

```
    double dblArea;
    // Calculate the area of a circle with radius 100
    dblArea = CalculateAreaFromRadius(100);
    // Print the results
    MessageBox.Show(dblArea, "Area");
```

اولین کاری که باید انجام دهید این است که یک متغییر به نام dblArea تعریف کنید که مساحت دایره را در خود نگه دارد. بعد از فراخوانی تابع، مقدار برگشتی از آن را می توانید در این متغییر نگهداری کنید. با استفاده از پرانتز هایی که در آخر نام تابع آمده اند، میتوانید پارامترهای مورد نیاز تابع را به آن ارسال کنید. در این جا فقط باید یک پارامتر را به تابع ارسال کنید و آن هم شعاع دایره خواهد بود.
بعد از اینکه متد را فراخوانی کردید، باید صبر کنید تا محاسبات آن تمام شود. پس از اتمام محاسبات تابع، نتیجه که مساحت دایره است برگردانده می شود و در متغییر dblArea قرار میگیرد. اکنون میتوانید نتیجه را به روش همیشگی در صفحه نمایش دهید.

انتخاب نام برای متد:

در چارچوب .NET یک سری استاندارد برای نامگذاری توابع و متدها تعریف شده اند که بهتر است هنگام انتخاب نام برای متد آنها را رعایت کنید. این مورد باعث می شود برنامه نویسان راحت تر بتوانند کدهای نوشته شده توسط خود را به زبان دیگری منتقل کنند. توصیه میکنیم که هنگام نامگذاری متدها از روش نامگذاری پاسکال استفاده کنید. یعنی فقط اولین حرف نام هر متد را به صورت بزرگ بنویسید. البته این مورد فقط یک توصیه برای خوانا تر شدن کدها در ویژوال C# ۲۰۰۵ است و هیچ اجباری در آن نیست. به مثال های زیر توجه کنید:

- CalculateAreaFromRadius
- OpenXmlFile
- GetEnvironmentValue

توجه کنید که در مثال های بالا حتی در مواردی که از حروف مخفف هم استفاده شده است (XML در مثال دوم) تمام حروف به جز حرف اول به صورت کوچک نوشته شده اند. رعایت این نکته به خصوص در زبانهایی مثل C# که به کوچکی و بزرگی حروف حساس هستند، میتواند از گیج شدن برنامه نویس در برابر نام تابع جلوگیری کند.
قاعده دیگر در مورد نامگذاری پارامتر ها است. در نامگذاری پارامتر ها بهتر است همانند نامگذاری توابع و متدها عمل کنید اما حروف اول هر پارامتر را نیز کوچک قرار دهید. به مثال های زیر در این مورد توجه کنید:

- myAccount
- customerDetails
- updatedDnsRecords

در اینجا نیز، همه چیز حتی کلمات اختصاری هم باید از قواعد نامگذاری پیروی کنند (همانند DNS در مثال بالا).
همانطور که میدانید در .NET کدها به زبان خاصی وابسته نیستند و کدهای موجود در یک زبان میتوانند در زبانهای دیگر مورد استفاده قرار بگیرند. بنابراین بهتر است از این قواعد پیروی کنید تا هم در زبانهایی که به نوع حروف حساس هستند و هم در زبانهایی که نسبت به این مورد حساس نیستند با مشکلی مواجه نشوید.

نکته: زبانهایی مانند Visual Basic به بزرگی و کوچکی حروف حساس نیستند اما زبانهای C#، J# و C++ حساس به حروف هستند. برای مثال در این زبانها متغییر intNumber با متغییر INTNUMBER و یا intnumber متفاوت است.

محدوده ها:

هنگامی که مفاهیم متدها را بیان می کردیم، گفتیم که متدها جامع هستند. این مورد تاثیر مهمی در نوع تعریف و استفاده از متغییر ها در متد دارد. فرض کنید که دو متد به صورت زیر دارید که در هر کدام متغییری رشته ای به نام strName تعریف شده است:

```
private void DisplaySebastiansName()
{
     // Declare variable and set value
     string strName = "Sebastian Blackwood";

     // Display results
     MessageBox.Show(strName, "Scope Demo");
}

private void DisplayBalthazarsName()
{
     // Declare variable and set value
     string strName = "Balthazar Keech";

     //Display results
     MessageBox.Show(strName, "Scope Demo");
}
```

با وجود اینکه هر دوی این متغییرها از متغییری با نام یکسان استفاده می کنند، اما هر کدام از این متغییرها در محدوده متد خود تعریف شده اند. در امتحان کنید بعدی این مورد را مشاهده خواهید کرد.

امتحان کنید: محدوده ها

۱) یک پروژه ویندوزی جدید به نام Scope Demo در ویژوال استودیو ایجاد کنید.
۲) یک کنترل Button جدید به فرم اضافه کنید. خاصیت Name آن را برابر btnScope و خاصیت Text آن را برابر Scope دهید. روی آن دو بار کلیک کنید و در متد ایجاد شده، کد مشخص شده در زیر را اضافه کنید:

```
private void btnScope_Click(object sender, EventArgs e)
{
     // Call a method
```

```
        DisplayBalthazarsName();
}

private void DisplaySebastiansName()
{
        // Declare variable and set value
        string strName;
        strName = "Sebastian Blackwood";
        // Display results
        MessageBox.Show(strName, "Scope Demo");
}

private void DisplayBalthazarsName()
{
        // Declare variable and set value
        string strName;
        strName = "Balthazar Keech";

        // Display results
        MessageBox.Show(strName, "Scope Demo");
}
```

۳) برنامه را اجرا کنید و بر روی دکمه Scope کلیک کنید. کادر پیغامی را مشاهده خواهید کرد که نام Balthazar Keech را نمایش می دهد.

چگونه کار می کند؟

همانطور که مشاهده کردید، در این تمرین با وجود اینکه دو متغییر با نام یکسان ولی در مکانهای متفاوت داریم، برنامه به درستی کار می کند.

```
private void DisplaySebastiansName()
{
        // Declare variable and set value
        string strName;
        strName = "Sebastian Blackwood";
        // Display results
        MessageBox.Show(strName, "Scope Demo");
}

private void DisplayBalthazarsName()
{
        // Declare variable and set value
        string strName;
        strName = "Balthazar Keech";
```

```
        // Display results
        MessageBox.Show(strName, "Scope Demo");
}
```

هنگامی که یک متد شروع به کار می کند، متغییر هایی که در آن تعریف شده اند (در محدوده باز شدن آکولاد و بسته شدن آن) محدوده فعالیت محلی[1] می گیرند. محدوده یک متغییر به این معنی است که کدام قسمت از برنامه میتواند به آن دسترسی پیدا کند. محدوده فعالیت محلی یعنی متغییر فقط در محدوده ی متد جاری قابل دسترسی است.
در برنامه قبلی، متغییر `strName` تا هنگام فراخوانی متد وجود ندارد. هنگامی که متد شروع به کار کرد، `.NET` و ویندوز حافظه مورد نیاز را در اختیار این متغییر قرار می دهد، بنابراین متغییر می تواند در کد مورد استفاده قرار گیرد. در این توابع ابتدا مقدار متغییر را برابر یک نام خاص قرار داده و سپس کادر پیغام را نمایش میدهید. بنابراین در این مثال هنگامی شما متد `DisplayBathazarName` را فراخوانی می کنید، متغییر های تعریف شده در آن متد بلافاصله ایجاد می شوند، متد وظایف خود را انجام می دهد و با پایان یافتن متد، متغییر ها نیز از حافظه حذف می شوند.

نکته: در بخش بعد مشاهده خواهید کرد که محدوده فعالیت یک متغییر حتی می تواند به داخل یک حلقه در داخل متد محدود شود.

نتیجه:

در این فصل مباحث و اصول مورد نیاز برای نوشتن نرم افزار را بیان کردیم. بیشتر این اصول فقط محدود به زبان ویژوال C# نیستند، بلکه در هر زبانی صدق میکنند. فصل را با معرفی الگوریتم ها آغاز کردیم که پایه و اساس تمام نرم افزارها هستند. سپس مفاهیم اولیه متغییر ها را معرفی کردیم و از نزدیک پرکاربرد ترین نوع های داده ای را از قبیل `int`، `double`، `string`، `DateTime` و `Boolean` بررسی کردیم. مشاهده کردید که چگونه می توان کارهای گوناگونی روی این متغییر ها انجام داد. مثلا انجام عملیات ریاضی روی متغییر های عددی، اتصال رشته های مختلف به هم و ایجاد رشته جدید، بدست آوردن طول یک رشته، تقسیم یک متن به چند زیررشته، بدست آوردن تاریخ جاری سیستم و یا دسترسی به قسمتهای مختلف تاریخ از قبیل ماه، سال و ... با استفاده از خاصیتهای آن. سپس نحوه ذخیره شدن متغییر ها در حافظه یک سیستم را مشاهده کردید.
بعد از این موارد نگاهی به متدها انداختیم. این که متدها چیستند، چرا به آنها نیاز پیدا میکنیم، چگونه آنها را ایجاد کنیم و به آنها پارامتر بفرستیم. همچنین محدوده فعالیت متغییر ها را در یک متد را نیز بررسی کردیم.
در پایان باید بدانید:

- الگوریتم ها چیستند و چگونه در توسعه نرم افزار مورد استفاده قرار می گیرند.
- چگونه متغییر هایی از نوع های داده ای مختلف را ایجاد و استفاده کنید.
- چگونه از پرکاربرد ترین توابع کار با رشته هنگامی که متغییری از نوع `string` دارید استفاده کنید.
- چگونه از نوع داده ای `DateTime` برای نگهداری زمان و تاریخ استفاده کنید و آنها را بر اساس تنظیمات محلی کامپیوتر کاربر در صفحه نمایش دهید.
- چگونه متدهای ساده را تعریف و استفاده کنید.

---

[1] `Local Scope`

تمرین:

تمرین ۱:

یک برنامه تحت ویندوز با دو کنترل `Button` ایجاد کنید. در رویداد `Click` دکمه اول دو متغییر از نوع عدد صحیح ایجاد کنید و مقدار اولیه آنها را برابر عددی دلخواه قرار دهید. سپس تعدادی از عملیات ریاضی مختلف را روی این اعداد انجام داده و نتیجه را در خروجی نمایش دهید.

در رویداد `Click` دکمه فرمان دوم دو متغییر از نوع رشته تعریف کنید و مقدار آنها را برابر رشته دلخواه قرار دهید. سپس دو رشته را به هم متصل کرده و نتیجه را نمایش دهید.

تمرین ۲:

یک برنامه تحت ویندوز با یک کنترل `TextBox` و یک کنترل `Button` ایجاد کنید. در رویداد `Click` دکمه فرمان، سه کادر پیغام را نمایش دهید. کادر پیغام اول باید طول رشته ی درون `TextBox` را نمایش دهد. کادر پیغام دوم باید نیمه اول رشته و کادر پیغام سوم نیمه دوم رشته را نمایش دهد.

فصل چهارم: کنترل روند اجرای برنامه

در فصل سوم در مورد الگوریتم ها و کاربرد آنها در نرم افزار مطالبی را آموختید. در این فصل نحوه کنترل روند اجرای برنامه در طول این الگوریتم ها را مشاهده خواهید کرد. برای مثال خواهید دید که چگونه می توانید تصمیماتی از قبیل "اگر X به این حالت بود، A را انجام بده در غیر این صورت B را انجام بده" را در برنامه ی خود پیاده کنید. این قابلیت در الگوریتم برنامه ها به عنوان انشعاب شناخته می شود. همچنین در این فصل مشاهده خواهید کرد چگونه می توانید یک قطعه کد را به تعداد مرتبه مشخص و یا تا زمانی که یک شرط درست است اجرا کنید.
خصوصا در این فصل در مورد موارد زیر صحبت خواهیم کرد:

- دستور `if`
- `switch`
- حلقه های `for` و `foreach`
- حلقه های `while ... do` و `do ... until`

تصمیم گیری در برنامه:

الگوریتم ها همواره دارای تصمیماتی هستند. در واقع، این تصمیمات است که باعث می شود کامپیوتر بتواند وظیفه خود را به خوبی انجام دهد. هنگام کد نویسی با تصمیم گیری های زیادی مواجه می شوید. مثلا فرض کنید که لیستی شامل ده نام در اختیار شما قرار داده اند و باید کدی بنویسید که به اعضای این لیست نامه ای را بفرستد. در هر قسمت از این کد می پرسید "آیا لیست تمام شده است؟" اگر چنین بود الگوریتم تمام می شود. در غیر این صورت نام نفر بعدی از لیست استخراج میشود و مراحل ارسال نامه برای او انجام میشود. به عنوان مثالی دیگر ممکن است که بخواهید فایلی را باز کنید. در این حالت ابتدا می پرسید "آیا فایل مورد نظر وجود دارد؟". در صورت وجود فایل را باز می کنید و در غیر این صورت الگوریتم را به اتمام میرسانید.
تمام این تصمیم گیریها به یک نحو در برنامه پیاده سازی می شوند. در ابتدا به بررسی دستور `if` برای کنترل روند اجرای برنامه می پردازیم.

دستور if

راحت ترین راه برای تصمیم گیری در ویژوال C# ۲۰۰۵ استفاده از دستور `if` است. در بخش امتحان کنید زیر با نحوه کاربرد این دستور آشنا خواهیم شد:

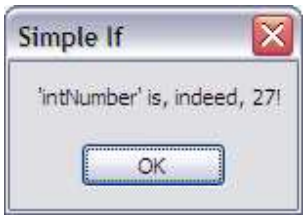
امتحان کنید: یک دستور `if` ساده

۱) یک برنامه تحت ویندوز به نام `Simple If` ایجاد کنید. سپس با استفاده از جعبه ابزار یک کنترل `Button` بـر روی فرم قرار داده، خاصیت `Name` آن را برابر `btnIf` و خاصیت `Text` آن را برابـر `If` قـرار دهیـد. روی ایـن کنترل دو بار کلیک کنید و کد زیر را در آن وارد کنید:

```
private void btnIf_Click(object sender, EventArgs e)
{
     // Declare and set a variable
     int intNumber = 27;

     // Here's where you make a desicion
     // and tell the user what happend
     if (intNumber == 27)
     {
          MessageBox.Show("'intNumber' is, indeed, 27!",
                          "Simple If");
     }
}
```

۲) برنامه را اجرا کنید و بر روی دکمه فرمان `If` کلیک کنید. کادر پیغامی را مشابه شکل ۴-۱ خواهید دید.



شکل ۴-۱

چگونه کار می کند؟

در ابتدا یک متغییر به نام `intNumber` ایجاد می کنید و مقدار آن را برابر ۲۷ قرار می دهید (همانطور که می بینید هم تعریف متغییر و هم مقدار دهی اولیه به آن در یک خط انجام شده است):

```
int intNumber = 27;
```

سپس با استفاده از دستور `if` مشخص می کنید که باید چه کاری انجام دهید. در اینجا شما میگویید "اگر `intNumber` برابر با ۲۷ بود...".

```
// Here's where you make a desicion
// and tell the user what happend
if (intNumber == 27)
```

```
{
    MessageBox.Show("'intNumber' is, indeed, 27!",
                    "Simple If");
}
```

قطعه کدی که درون آکولاد پایین دستور `if` قرار دارد فقط هنگامی اجرا میشود که `intNumber` برابر با ۲۷ باشد. به عبارت دیگر هنگامی که شرط داخل پرانتز برابر `true` و یا درست باشد، کد داخل بلاک `if` اجرا می شود.
بنابراین در اینجا، اجرای برنامه شروع میشود و به دستور `if` میرسد. بعد از ارزیابی عبارت داخل پرانتز چون مقدار آن برابر با `true` است، دستورات درون بلاک `if` اجرا می شوند و سپس اجرای برنامه از خط بعد از بلاک `if` ادامه پیدا می کند.

نکته: دقت کنید که کدهای درون بلاک `if` به صورت اتوماتیک با مقداری تورفتگی[1] نوشته می شوند. این مورد باعث افزایش خوانایی کد می شود، زیرا می توانید با نگاهی سریع بگویید که کدام قسمت از کد در صورت صحیح بودن دستور `if` اجرا می شود. همچنین برای خوانایی بیشتر برنامه بهتر است قبل و بعد از بلاک `if` مقداری فضای خالی قرار دهید.

یک دستور `if` ساده همانند قسمت قبل میتواند بودن هیچ آکولادی نوشته شود. البته این کار هنگامی امکانپذیر است که بلاک `if` فقط شامل یک دستور باشد. به کد زیر نگاه کنید:

```
if (intNumber == 27)
    MessageBox.Show("'intNumber' is, indeed, 27!",
                    "Simple If");
```

این دستور هم مانند دستور `if` در برنامه قبلی کار می کند و مزیت آن فقط این است که کد را کوتاهتر می کند.
اما اگر نتیجه یک شرط نادرست باشد، چه اتفاقی می افتد؟ در امتحان کنید بعدی این حالت را بررسی خواهیم کرد.

امتحان کنید: نادرست بودن شرط

۱) اگر برنامه `Simple If` در حال اجرا است آن را ببندید. کنترل `Button` دیگری به فرم اضافه کنید، خاصیت `Name` آن را برابر `btnAnotherIf` و خاصیت `Text` آن را برابر `Another If` قرار دهید. روی دکمه دو بار کلیک کنید و کد مشخص شده در زیر را در متد مربوط به رویداد کلیک آن وارد کنید:

```
private void btnAnotherIf_Click(object sender,
                                EventArgs e)
{
    // Declare and set a variable
    int intNumber = 27;

    // Here's where you make a decision,
    // and tell the user what happened
    if (intNumber == 1000)
```

[1] Indention

```
    {
        MessageBox.Show("'intNumber' is, indeed, " +
                        "1000!", "Simple If");
    }
}
```

۲) برنامه را اجرا کنید.

چگونه کار می کند؟

در این حالت، جواب سوال "آیا `intNumber` برابر با ۱۰۰۰ است؟" خیر است. بلاک دستورات هم فقط در حالتی اجرا می شود که نتیجه شرط برابر `true` باشد، بنابراین کدهای این بلاک اجرا نخواهند شد. در چنین شرایطی کنترل برنامه بلافاصله به خط بعد از `if` منتقل می شود و کد مربوط به آن را اجرا می کند.

دستور Else:

اگر بخواهید در صورت درست بودن شرط قسمتی از برنامه و در صورت غلط بودن آن قسمتی دیگر اجرا شود، می توانید از دستور `else` استفاده کنید. در امتحان کنید زیر می توانید نحوه کاربرد این دستور را مشاهده کنید.

امتحان کنید: دستور `else`

۱) کد درون رویداد `Click` مربوط به کنترل `btnAnotherIf` را به صورت زیر تغییر دهید:

```
private void btnAnotherIf_Click(object sender,
                                EventArgs e)
{
    // Declare and set a variable
    int intNumber = 27;

    // Here's where you make a decision,
    // and tell the user what happened
    if (intNumber == 1000)
    {
        MessageBox.Show("'intNumber' is, indeed, " +
                        "1000!", "Simple If");
    }
    else
    {
```

```
            MessageBox.Show("'intNumber' is not 1000!",
                "Simple If");
        }
}
```

۲) برنامه را اجرا کنید و روی دکمه Another If کلیک کنید. کادر پیغامی مشابه شکل ۴-۲ را مشاهده خواهید کرد.



شکل ۴-۲

چگونه کار می کند؟

کدی که در بلاک else وارد شده است، فقط در صورتی اجرا می شود که عبارت درون پرانتز if نادرست باشد. در این حالت، مقدار intNumber برابر با ۲۷ است، اما چون در شرط با عدد ۱۰۰۰ مقایسه شده است بنابراین شرط غلط است و کد نوشته شده در بخش else اجرا خواهد شد:

```
else
{
    MessageBox.Show("'intNumber' is not 1000!",
        "Simple If");
}
```

بررسی چند شرط با else if:

اگر می خواهید بیش از یک حالت را تست کنید، باید از ترکیب دستور else و if استفاده کنید. در امتحان کنید بعدی، برنامه Simple If را به نحوی تغییر می دهیم که برابری intNumber را با چند عدد مختلف بررسی کند و نتیجه را نمایش دهد.

امتحان کنید: دستور else if

۱) کد درون متد btnAnotherIf_Click را به صورت زیر تغییر دهید:

```
private void btnAnotherIf_Click(object sender,
```

```
                                    EventArgs e)
{
      // Declare and set a variable
      int intNumber = 27;

      // Here's where you make a decision,
      // and tell the user what happened
      if (intNumber == 1000)
      {
            MessageBox.Show("'intNumber' is, indeed, " +
                  "1000!", "Simple If");
      }
      else if (intNumber == 27)
      {
            MessageBox.Show("'intNumber' is 27!",
                  "Simple If");
      }
      else
      {
            MessageBox.Show("'intNumber' is neither 1000" +
                  " nor 27!", "Simple If");
      }
}
```

۲) برنامه را اجرا کنید و روی دکمه فرمان `Another If` کلیک کنید. کادر پیغامی مشابه شکل ۴-۳ را خواهید دید.



شکل ۴-۳

چگونه کار می کند؟

در این برنامه دستورات بخش `else if` اجرا می شوند، زیرا `intNumber` برابر با عدد ۲۷ است و بنابراین عبارت داخل `else if` درست خواهد بود. توجه داشته باشید که اگر شرط داخل `else if` نیز غلط بود، کدهای بخش `else` اجرا می شدند.

در یک سری دستورات `if` و `else if` متوالی، شرایط از بالاترین `if` به سمت پایین بررسی می شوند و اولین عبارتی که درست ارزیابی شد، دستورات مربوط به آن اجرا می شوند. پس در برنامه قبل اگر شرط اول را به گونه ای تنظیم کنیم که درست باشد (برای مثال داخل پرانتز عبارت `intNumber > 10` را قرار دهیم که به علت بزرگتر بودن `intNumber` از ۱۰ برابر

با true است)، با وجود اینکه شرط دوم هم درست است دستورات شرط اول اجرا می شوند و کنترل برنامه بـه اولـین خـط بعـد از سری دستورات if می رود.

```
else if (intNumber == 27)
{
     MessageBox.Show("'intNumber' is 27!",
               "Simple If");
}
else
{
     MessageBox.Show("'intNumber' is neither 1000" +
               " nor 27!", "Simple If");
}
```

شما می توانید به هر تعداد که بخواهید قسمتهای else if را به یک دستور if برای بررسی حالتهای مختلف اضـافه کنیـد. اما همانطور که ذکر شد، هنگامی که ویژوال C# به اولین دستور if رسید شرط داخل آن را بررسی میکند. اگر عبارت داخل پرانتـز درست ارزیابی شود، دستورات داخل بلاک if اجرا می شوند و کنترل برنامه به اولین خط بعـد از سـری دسـتورات if و else میرود. در غیر این صورت، عبارت مربوط به اولین else if ارزیابی می شود. این روند ادامه پیدا می کند تا برنامه به بلاکـی از دستورات برسد که حاصل آن درست باشد. در این حالت دستورات این بلاک اجرا شده و کنترل برنامه به بعـد از مجموعـه دسـتورات if و else می رود.

نکته: هنگام بررسی یک سری از حالتها، بهتر است آنهایی را که احتمال درست بودنشان بیشتر است، ابتدا بررسی کنیـد. ایـن مـورد باعث می شود کامپایلر هنگام اجرا، شرایط اضافه را بررسی نکند و کد سریعتر اجرا شود.

نکته: در مواردی که تعداد قسمتهای else if در برنامه زیاد باشد و حالتهای زیادی را باید بررسی کنید، به جـای اسـتفاده از if و else if میتوانید از switch استفاده کنید که در قسمتهای بعدی توضیح داده می شوند.

دستورات if تودرتو:

علاوه بر استفاده متوالی از دستورات if، می توانید در داخل یک if از دستورات if دیگری استفاده کنید. بـه مثـال زیـر توجـه کنید:

```
if (intX > 3)
{
     MessageBox.Show("intX is greater that 3",
               "Sample If");

     if (intX == 6)
          MessageBox.Show("intX is 6!", "Sample If");
}
```

در استفاده از دستورات `if` تودرتو هیچ محدودیتی نیست. البته باید دقت کنید که هر چه تعداد `if`های تودرتو در برنامه بیشتر باشد، درک آن مشکلتر می شود. بنابراین سعی کنید تا جایی که میتوانید تعداد `if`های تودرتو را در برنامه کم کنید.

عملگرهای مقایسه ای:

در قسمتهای قبلی، نحوه بررسی برابر بودن یک متغییر را با مقادیر مختلف برای استفاده در شرط یک دستور `if` دیدیم. اما دستور `if` بسیار انعطاف پذیر تر است. شما میتوانید برای شرط این دستور از سوالهایی مثل موارد زیر استفاده کنید که پاسخ همه آنها بله یا خیر است:

- آیا `intNumber` بزرگتر از ۴۹ است؟
- آیا `intNumber` کوچکتر از ۴۹ است؟
- آیا `intNumber` بزرگتر یا مساوی ۴۹ است؟
- آیا `intNumber` کوچکتر یا مساوی ۴۹ است؟
- آیا `strName` برابر با `Ben` است؟

هنگام کار با متغییر های رشته ای، معمولا از شرطهای برابر بودن یا مخالف بودن استفاده میکنید. اما هنگام کار با متغییر های عددی (چه اعداد صحیح و چه اعداد اعشاری) می توانید از تمام عملگر های ریاضی که در بالا ذکر شد استفاده کنید.

استفاده از عملگر مخالف:

تاکنون از عملگر مخالف استفاده نکرده ایم. بنابراین در امتحان کنید بعد نحوه استفاده از این عملگر را با متغییر های رشته ای خواهیم دید.

امتحان کنید: استفاده از عملگر مخالف

۱) یک برنامه تحت ویندوز جدید به نام `If Demo` ایجاد کنید.
۲) هنگامی که محیط طراحی `Form1` را مشاهده کردید، یک کنترل `TextBox` و یک کنترل `Button` را به وسیله جعبه ابزار بر روی فرم قرار دهید. خاصیت `Name` کنترل `TextBox` را برابر با `txtName` و خاصیت `Text` آن را برابر با `Robbin` قرار دهید. سپس خاصیت `Name` کنترل `Button` را برابر `btnCheck` و خاصیت `Text` آن را برابر `Check` قرار دهید.
۳) روی `Button` دو بار کلیک کنید و کد زیر را در بخش رویداد `Click` مربوط به آن قرار دهید:

```
private void btnCheck_Click(object sender, EventArgs e)
{
    // Declare a variable and
    // get the name from the text box
```

```
        string strName;
        strName = txtName.Text;

        // Is the name Gretchen?
        if (strName != "Gretchen")
        MessageBox.Show("The name is *not* Gretchen.",
                        "If Demo");
}
```

۴) برنامه را اجرا کنید و روی دکمه Check کلیک کنید. کادر پیغامی را مشاهده خواهید کرد که میگوید نـام داخـل جعبـه متنی برابر با Gretchen نیست.

چگونه کار می کند؟

عملگر مخـالف در زبـان C# بـه صـورت != اسـت. هنگـامی کـه کـاربر روی دکمـه فرمـان کلیـک مـی کنـد، اول مـتن درون TextBox به وسیله خاصیت Text آن بدست آورده می شود و درون یک متغییر رشته ای قرار می گیرد.

```
// Declare a variable and
// get the name from the text box
string strName;
strName = txtName.Text;
```

بعد از اینکه نام وارد شده در متغییر قرار گرفت، با استفاده از عملگر مخالف مقدار درون متغییر رشته ای با یک عبارت رشته ای دیگـر مقایسه شده و نتیجه این مقایسه برای اجرای دستورات if استفاده می شود.

```
// Is the name Gretchen?
if (strName != "Gretchen")
MessageBox.Show("The name is *not* Gretchen.",
                "If Demo");
```

همانطور که گفتیم، دستورات درون بلاک if فقط در صورتی اجرا می شوند که نتیجه داخل پرانتز برابر بـا true باشـد. ممکـن است عملگر مخالف در ابتدا مقداری گیج کننده به نظر برسد، اما توجه کنید سوالی که در این قسمت پرسیده میشود این است که "آیا مقدار strName مخالف Gretchen است؟". در این حالت پاسـخ بـه صـورت "بلـه، مقـدار strName مخـالف بـا Gretchen است" خواهد بود. بنابراین پاسخ این سوال به صورت بله است که true ارزیابی میـشود. دقـت کنیـد کـه اگـر مقدار Gretchen را در TextBox وارد کنید، مقدار شرط برابر با غلط خواهد بود و دستورات بلاک if اجرا نخواهند شد.

نکته: اگر می خواهید متن Gretchen را در TextBox وارد کنید، توجه کنید که حتما این متن به همان صـورت کـه در کد بررسی می شود نوشته شود (در اینجا با G بزرگ). زیرا بررسی شرط در این قسمت به صورت حساس به حروف انجام می شـود و اگر فرضاً عبارت gretchen را وارد کنید، دو مقدار برابر نخواهد بود و مجددا کادر متنی نمایش داده خواهد شد.

نکته: علامت ! در C# برای معکوس کردن مقدار یک شرط به کار می رود. بنابراین اگر حاصل شرطی برابر با true باشد و قبل از آن از عملگر ! استفاده کنید، نتیجه برابر false ارزیابی می شود. برای مثال اگر مقدار متغییر strName برابر با Robbin باشد، حاصل عبارت زیر false خواهد بود:

```
if (!(strName == "Robbin"))
```

به عبارت دیگر عملگر مخالف در برنامه بالا را می توان به صورت زیر نوشت:

```
if (!(strName == "Gretchen"))
```

استفاده از عملگر های مقایسه ای:

در این بخش چهار عملگر مقایسه ای دیگر را معرفی خواهیم کرد. در امتحان کنید های بعدی با این عملگر ها آشنا می شوید:

امتحان کنید: استفاده از عملگر کوچکتر

۱) اگر برنامه If Demo در حال اجرا است، آن را ببندید. قسمت طراحی فرم را برای Form1 باز کنید، یک Button و یک TextBox به فرم اضافه کنید و خاصیت Name آن را برابر txtValue قرار دهید. سپس یک دیگر به فرم اضافه کرده خاصیت Name آن را برابر btnCheckNumber و خاصیت Text آن را برابر Check Numbers قرار دهید.

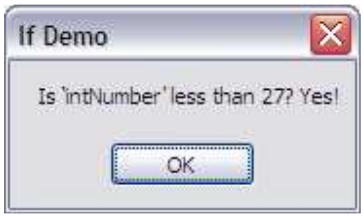
۲) روی کنترل Button دو بار کلیک کنید و کد زیر را در رویداد Click آن وارد کنید:

```
private void btnCheckNumber_Click(object sender,
                                  EventArgs e)
{
    // Declare variable
    int intNumber = 0;
    try
    {
        // Get the number from the text box
        intNumber = Int32.Parse(txtValue.Text);
    }
    catch
    {
    }

    // Is intNumber less than 27?
    if (intNumber < 27)
        MessageBox.Show("Is 'intNumber' less than 27? " +
                        "Yes!", "If Demo");
```

```
        else
            MessageBox.Show("Is 'intNumber' less than 27? " +
                            "No!", "If Demo");
}
```

۳) برنامه را اجرا کنید. عددی را در TextBox وارد کرده و روی دکمه Check Numbers کلیک کنید. کادر پیغامی را مشاهده خواهید کرد که به شما میگوید عدد وارد شده در TextBox بزرگتر از ۲۷ است یا نه؟ (شکل ۴-۴)



شکل ۴-۴

چگونه کار می کند؟

در این برنامه ابتدا باید مقدار عددی وارد شده در TextBox را بدست آورد. برای این کار ابتدا باید مطمئن شویم متن داخل TextBox شامل عدد است. زیرا کاربر برنامه آزاد است که هر متنی را در اینجا وارد کند، و ممکن است متن وارد شده شامل هیچ عددی نباشد که در این حالت برنامه در تبدیل آن به یک عدد صحیح با شکست مواجه می شود و بقیه کد اجرا نمی شود. بنابراین در این قسمت باید کدی را برای مدیریت این استثنا ها وارد می کنیم تا اگر کاربر مقداری غیر از عدد وارد کرد متغییر intNumber برابر با صفر شود، در غیر این صورت عدد وارد شده توسط کاربر در آن قرار گیرد.

```
// Declare variable
int intNumber = 0;
try
{
    // Get the number from the text box
    intNumber = Int32.Parse(txtValue.Text);
}
catch
{
}
```

نکته: دستور try...catch در فصل ۱۱ به صورت کامل بررسی می شود، بنابراین در این قسمت میتوانید از آن صرف نظر کنید.

برای تبدیل یک رشته که شامل عدد است به یک عدد صحیح باید از تابع Parse در Int32 استفاده کنید. این تابع یک رشته را که شامل یک عدد است دریافت کرده و عدد معادل آن را برمیگرداند. اگر رشته ای که به این تابع فرستاده می شود شامل عدد نباشد، یا حتی دارای کاراکتری غیر عددی باشد، تابع با خطا مواجه شده و اجرای برنامه متوقف می شود.

در بخش بعدی، به وسیله دستور `if` بررسی می کنید که عدد وارد شده در `TextBox` بزرگتر از ۲۷ است یا نه و بـر اسـاس آن پیغام مناسبی را به کاربر نمایش می دهید.

```
// Is intNumber less than 27?
if (intNumber < 27)
     MessageBox.Show("Is 'intNumber' less than 27? " +
                     "Yes!", "If Demo");
else
     MessageBox.Show("Is 'intNumber' less than 27? " +
                     "No!", "If Demo");
```

نکته: همانطور که ملاحظه می کنید، در این قسمت چون هم در بخش `if` و هم در بخش `else` فقـط یـک دسـتور وارد شـده است نیازی به استفاده از آکولاد در ابتدا و انتهای دستور نیست.

جالب اینجاست که اگر دقیقا مقدار ۲۷ را در `TextBox` وارد کنید، گفته می شود که عـدد کـوچکتر از ۲۷ نیـست. زیـرا عملگـر کوچکتر فقط در صورتی مقدار `true` را برمی گرداند که عدد کوچکتر باشد، نه بزرگتر یا مساوی. برای اینکه شرط بالا خـود عـدد ۲۷ را نیز شامل شود باید از عملگر کوچکتر مساوی استفاده کنید که در بخش امتحان کنید بعدی شرح داده شده است.

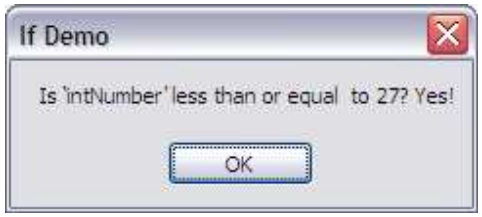
امتحان کنید: استفاده از عملگر کوچکتر یا مساوی

۱) کد موجود در رویداد `Click` مربوط به کنترل `Button` در قسمت قبلی را به صورت زیر تغییر دهید:

```
// Declare variable
int intNumber;
try
{
     // Get the number from the text box
     intNumber = Int32.Parse(txtValue.Text);
}
catch
{
}

// Is intNumber less than or equal to 27?
if (intNumber <= 27)
     MessageBox.Show("Is 'intNumber' less than or " +
                     "equal  to 27? Yes!", "If Demo");
else
     MessageBox.Show("Is 'intNumber' less than or " +
                     "equal to 27? No!", "If Demo");
```

۲) حال برنامه را اجرا کنید و عدد ۲۷ را در TextBox وارد کنید. روی دکمه Check Numbers کلیک کنید. کادر پیغامی مشابه شکل ۴-۵ نمایش داده خواهد شد.



شکل ۴-۵

چگونه کار می کند؟

تنها تفاوتی که این برنامه نسبت به برنامه قبلی دارد این است که شرط دستور if خود عدد ۲۷ را نیز شامل می شود. یعنی اگر در جعبه متنی عدد ۲۷ وارد کنید، کادر پیغامی مشابه شکل ۵-۴ را مشاهده خواهید کرد.
دو عملگر دیگر مشابه عملگر های قبلی می باشد که در بخش بعدی آنها را بررسی خواهیم کرد.

امتحان کنید: استفاده از عملگر بزرگتر و بزرگتر مساوی

۱) رویداد Click مربوط به کنترل Button را باز کرده و کد زیر را به آن اضافه کنید:

```
// Is intNumber less than or equal to 27?
if (intNumber <= 27)
     MessageBox.Show("Is 'intNumber' less than or" +
                     "equal to 27? Yes!", "If Demo");
else
     MessageBox.Show("Is 'intNumber' less than or" +
                     "equal to 27? No!", "If Demo");

// Is intNumber greater than 27?
if (intNumber > 27 )
     MessageBox.Show("Is 'intNumber' greater than" +
                     "27? Yes!", "If Demo");
else
     MessageBox.Show("Is 'intNumber' greater than" +
                     "27? No!", "If Demo");

// Is intNumber greater than or equal to 27?
if( intNumber >= 27)
```

```
            MessageBox.Show("Is 'intNumber' greater than" +
                            "or equal to 27? Yes!", "If
Demo");
        else
            MessageBox.Show("Is 'intNumber' greater than" +
                            "or equal to 27? No!", "If Demo");
```

۲) برنامه را اجرا کنید و مقدار ۹۹ را در جعبه متنی وارد کنید. سپس روی دکمه `Check Numbers` کلیک کنید. سه کادر پیغام متوالی را مشاهده خواهید کرد. در کادر پیغام اول گفته می شود که عدد کوچکتر یا مساوی ۲۷ نیست. در کادر پیغام دوم و سوم به ترتیب مشاهده می کنید که عدد بزرگتر و همچنین بزرگتر مساوی ۲۷ است.

چگونه کار می کند؟

عملگر های بزرگتر و بزرگتر مساوی دقیقا بر عکس عملگر کوچکتر و کوچکتر مساوی عمل می کند. سوالاتی که در این قسمت برای شرط `if` می پرسید به صورت "آیا `intNumber` بزرگتر از ۲۷ است؟" و " آیا `intNumber` بزرگتر یا مساوی ۲۷ است؟" خواهد بود که بر اساس پاسخ آنها کد مربوط به شرط `if` اجرا می شود.

عملگر های And و Or منطقی[1] :

در بعضی از شرایط ممکن است بخواهید به جای یک شرط، چند شرط را در دستور `if` بررسی کنید. برای مثال می خواهید بدانید آیا `intNumber` بزرگتر از ۲۷ و کوچکتر از ۱۰ است یا نه و یا میخواهید بدانید آیا `strName` برابر `"Sydney"` و یا `"Stephanie"` است یا نه. در این موارد میتوانید شرطهای درون دستور `if` را به وسیله عملگر های `And` و `Or` منطقی ترکیب کنید. در امتحان کنید بعد، روش استفاده از عملگر `Or` را خواهیم دید. نتیجه ترکیب چند شرط به وسیله این عملگر فقط هنگامی درست که حداقل یکی از شروط برابر با درست باشد.

امتحان کنید: استفاده از عملگر های And و Or

۱) برنامه ویندوزی جدیدی به نام `And Or Demo` ایجاد کنید.
۲) در قسمت طراحی فرم مربوط به `Form1` دو کنترل `TextBox` و یک کنترل `Button` اضافه کنید. خاصیت `Name` مربوط به کنترلهای `TextBox` را برابر `txtName1` و `txtName2` و خاصیت `Name` کنترل `Button` را برابر `btnOrCheck` قرار دهید.
۳) خاصیت `Text` مربوط به `TextBox` اول را برابر `Sydney` و `TextBox` دوم را برابر `Stephanie` قرار دهید. در آخر خاصیت `Text` کنترل `Button` را برابر `Or Check` قرار دهید. بعد از این تنظیمات فرم شما باید مشابه شکل ۴-۶ باشد.

[1] در مقابل عملگر های `And` و `Or` منطقی، عملگر های `And` و `Or` بیتی نیز وجود دارند که برای ایجاد تغییرات روی بیتها به کار می روند.



شکل ۴-۶

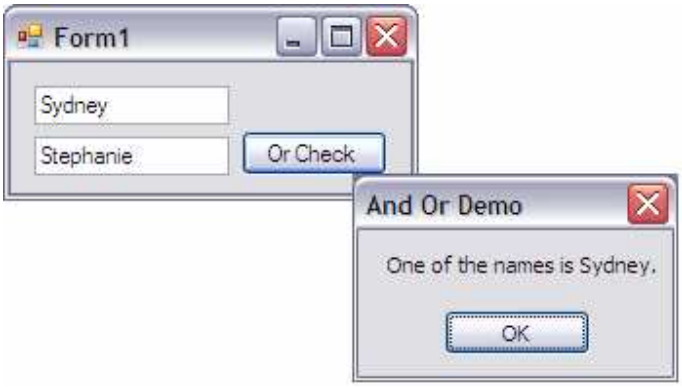
۴) روی دکمه Or Check دو بار کلیک کنید و کد زیر را در رویداد Click آن وارد کنید.

```
private void btnOrCheck_Click(object sender, EventArgs e)
{
     // Declare variables
     string strName1 , strName2;

     // Get the names
     strName1 = txtName1.Text;
     strName2 = txtName2.Text;

     // Is one of the names Sydney?
     if (strName1 == "Sydney" || strName2 == "Sydney")
          MessageBox.Show("One of the names is Sydney.",
                          "And Or Demo");
     else
          MessageBox.Show("Neither of the names is
Sydney.",                          "And Or Demo");
}
```

۵) برنامه را اجرا کنید و روی دکمه Or Check کلیک کنید. کادر پیغامی را مشابه شکل ۴-۷ مشاهده خواهید کرد.



شکل ۴-۷

۶) روی دکمه OK بر روی کادر پیغام کلیک کنید تا به فرم اصلی برگردید. حال متن داخل TextBox اول را به Stephanie و متن جعبه متنی دوم را به Sydney تغییر دهید. مجددا روی دکمه Or Check کلیک کنید. کادر پیغامی را مشاهده خواهید کرد که می گوید یکی از TextBoxها شامل Sydney است.

۷) مجددا روی OK کلیک کنید و در صفحه اصلی متن های داخل صفحه را به گونه ای تغییر دهید که هیچ یک از آنها شامل Sydney نباشد و روی دکمه Or Check کلیک کنید. کادر پیغامی را مشاهده خواهید کرد که می گوید هیچ یک از آنها شامل Sydney نیست.

چگونه کار می کند؟

عملگر || در C# به عنوان "یا" منطقی استفاده می شود و معمولاً در بررسی شرط ها، برای ترکیب دو شرط متفاوت از هم به کار می رود. در رویداد Click، ابتدا دو متغییر تعریف می کنیم و سپس مقادیری که کاربر در TextBox ها وارد کرده است را آنها قرار می دهیم.

```
// Declare variables
string strName1 , strName2;

// Get the names
strName1 = txtName1.Text;
strName2 = txtName2.Text;
```

همانطور که میبینید در این قسمت هر دوی متغییر ها در یک خط تعریف شده اند. در C# اگر بخواهید چند متغییر از یک نوع داشته باشید، می توانید همه آنها را همانند کد بالا در یک خط تعریف کنید و فقط باید نام آنها را با ویرگول از هم جدا کنید. این مورد باعث می شود که کد برنامه فشرده تر شود و تفاوتی با تعریف متغییر ها در خطهای جداگانه ندارد.
حال که هر دو نام را از داخل TextBoxها به دست آوردید، میتوانید آنها را در یک شرط if با استفاده از عملگر || ترکیب کنید. در این حالت سوالی که شما در بخش شرط if ایجاد می کنید به صورت "آیا مقدار strName1 برابر با Sydney است و یا مقدار strName2 برابر با Sydney است؟" خواهد بود. در این حالت مقدار هر یک از این متغییر ها که برابر با Sydney باشد موجب میشود که پاسخ سوال برابر با true و یا درست باشد.

```
// Is one of the names Sydney?
if (strName1 == "Sydney" || strName2 == "Sydney")
     MessageBox.Show("One of the names is Sydney.",
                     "And Or Demo");
else
     MessageBox.Show("Neither of the names is
Sydney.",                         "And Or Demo");
```

استفاده از عملگر And منطقی:

این عملگر هم مانند عملگر Or منطقی است به جز اینکه برای درست بودن شرط آن، باید تک تک شرط ها درست ارزیابی شوند. علامت این عملگر در C# به صورت && است.

امتحان کنید: استفاده از عملگر And منطقی

۱) کنترل Button دیگری به فرم اضافه کنید، خاصیت Name آن را برابر btnAndCheck و خاصیت Text آن را برابر And Check قرار دهید. سپس بر روی این کنترل دو بار کلیک کرده و کد مشخص شده در زیر را در آن وارد کنید:

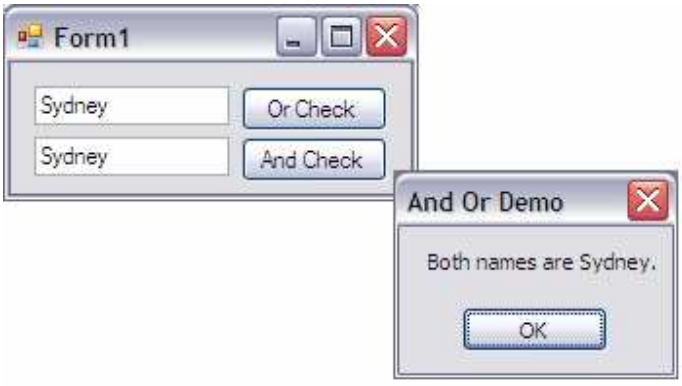
```
private void btnAndCheck_Click(object sender, EventArgs e)
{
     // Declare variables
     string strName1, strName2;

     // Get the names
     strName1 = txtName1.Text;
     strName2 = txtName2.Text;

     // Are both names Sydney?
     if( strName1 =="Sydney" &&  strName2 =="Sydney")
          MessageBox.Show("Both names are Sydney.",
                          "And Or Demo");
     else
          MessageBox.Show("One of the names is not" +
                          "Sydney.","And Or Demo");
}
```

۲) برنامه را اجرا کنید و روی دکمه And Check کلیک کنید. کادر پیغامی را خواهید دید که می گوید یکی از TextBoxها شامل Sydney نیست.

۳) البته اگر متن داخل هر دو TextBox را به Sydney تغییر دهید، نتیجه ای مشابه شکل ۴-۸ را خواهید دید.



شکل ۴-۸

چگونه کار می کند؟

بعد از اینکه نامهای موجود در TextBox را بدست آوردید، آنها را با هم مقایسه می کنید. در اینجا با استفاده از عملگر && می پرسید "آیا strName1 برابر با Sydney و strName2 برابر با Sydneyاست؟". واضح است جـواب ایـن سـوال هنگامی درست است که هر دو TextBox محتوی کلمه Sydney باشند.

```
// Are both names Sydney?
if( strName1 ="Sydney" &&  strName2 ="Sydney")
     MessageBox.Show("Both names are Sydney.",
                     "And Or Demo");
else
     MessageBox.Show("One of the names is not" +
                     "Sydney.","And Or Demo");
```

مطالب بیشتر در رابطه با عملگر های And و Or منطقی:

تاکنون با نحوه استفاده از عملگر های And و Or در رشته ها آشنا شده اید، اما میتوانید این عملگر ها را با اعداد نیز همانند زیر به کار ببرید:

```
     if( intX == 2 && dblY == 2.3)
          MessageBox.Show("Hello, the conditions has been"
+                         "satisfied!");
```

یا:

```
     if( intX == 2 || dblY == 2.3)
          MessageBox.Show("Hello, the conditions have been"
                          + "satisfied!");
```

همچنین در استفاده از عملگر های And و Or در یک دستور if هیچ محدودیتی نیست. به عبارت دیگر، میتوانید در برنامه خـود دستوری مشابه زیر داشته باشید:

```
if( intA == 1 && intB == 2 && intC == 3 && intD == 4 &&
    intE == 5 && intF == 6 && intG == 7 && intH == 1 &&
    intI == 2 && intJ == 3 && intK == 4 && intL == 5 &&
    intM == 6 && intN == 7 && intO == 1 && intP == 2 &&
    intQ == 3 && intR == 4 && intS == 5 && intT == 6 &&
    intU == 7 && intV == 1 && intW == 2 && intX == 3 &&
    intY == 4 && intZ == 5)
     MessageBox.Show("That's quite an If statement!");
```

البته باید توجه داشته باشید که استفاده زیاد از این عملگر ها از خوانایی برنامه می کاهد و درک آن را مشکل تر می کند. پس تا حد امکان باید از شرطهای کمتر در دستور `if` خود استفاده کنید.
همچنین میتوانید از چند عملگر `And` و `Or` در شرط خود استفاده کنید. در این مواقع می توانید با استفاده از پرانتزها این عملگر ها را دسته بندی کنید. برای مثال می خواهید اگر مقدار متغییری بین ۱۰ تا ۲۰ و یا بین ۲۵ تا ۳۰ باشد، دستورات داخل شرط اجرا شوند. در این صورت می توانید از دستور `if` زیر استفاده کنید:

```
if( (intX > 10 && intX < 20) || (intX > 25 && intX < 30))
```

حالتهای مختلفی برای ترکیب عملگرها در هر زبان برنامه نویسی وجود دارد و تعداد آنها بیشتر از آن است که بتوانیم آنها را در این کتاب عنوان کنیم. اما بدانید که هر شرطی را که بخواهید بررسی کنید یک ترکیب از این عمگرها وجود دارد که نیاز شما را برطرف کند.

مقایسه رشته ها:

معمولا هنگامی که در دستورات `if` رشته ها را با یکدیگر مقایسه می کنید، به علت حساسیت برنامه به حروف کوچک و بزرگ با مشکل مواجه می شوید. با وجود اینکه هر دو کاراکتر `"a"` و `"A"` برای انسانها یک معنی را دارند و یکسان تلقی می شوند، اما در کامپیوتر دو کاراکتر مجزا از یکدیگر هستند. این مورد به عنوان حساسیت به نوع حروف شناخته می شود. برای مثال اگر کد زیر را در برنامه خود اجرا کنید، کادر پیغام نمایش داده نخواهد شد:

```
string strName = "winston";
if (strName == "WINSTON")
    MessageBox.Show("Aha! you are Winston!");
```

همانطور که ممکن است حدس زده باشید، کلمه `WINSTON` که با حروف بزرگ است با مقدار متغییر `strName` که با حروف کوچک است تفاوت دارد و شرط اجرا نخواهد شد. اما در بیشتر مواقع شما نمی خواهید رشته ها را به این صورت بررسی کنید، پس باید راهی را پیدا کنید که آنها را به حالت عادی و بدون در نظر گرفتن نوع حروف مقایسه کنید. در امتحان کنید بعدی، روشی را برای این کار مشاهده خواهیم کرد:

امتحان کنید: مقایسه رشته ها بدون در نظر گرفتن نوع حروف

۱) قسمت طراحی فرم را برای `Form1` باز کنید و کنترل `Button` دیگری را به فرم اضافه کنید. خاصیت `Name` دکمه فرمان را برابر `btnStringCompare` و خاصیت `Text` آن را برابر `Compare String` قرار دهید.
۲) روی کنترل `Button` دو بار کلیک کنید و کد زیر را در متد مربوط به رویداد `Click` آن وارد کنید:

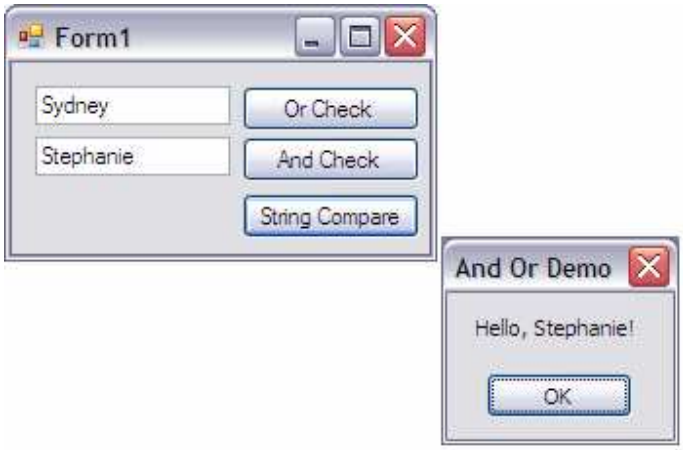
```
private void btnStringCompare_Click(object sender,
                                    EventArgs e)
```

```
{
      // Declare variable
      string strName;

      // Get the name
      strName = txtName2.Text;

      // Compare the name
      if (String.Compare(strName, "STEPHANIE", True) == 0)
           MessageBox.Show("Hello, Stephanie!",
                           "And Or Demo");
}
```

۳) برنامه را اجرا کنید و روی دکمه ای که جدیدا اضافه کردید کلیک کنید. نتیجه ای مشابه شکل ۴-۹ را مشاهده خواهید کرد.



شکل ۴-۹

۴) روی دکمه OK کلیک کنید و در TextBox عبارتی را مانند STePHaniE یا هر ترکیب دیگری از آن وارد کنید. سپس روی دکمه String Compare کلیک کنید. مشاهده میکنید که کادر پیغام قسمت قبل مجددا نمایش داده خواهد شد.

چگونه کار می کند؟

همانطور که در کد مشاهده می کنید بعد از اینکه اسم نوشته شده در TextBox را در متغییر قرار دادیم، به جای استفاده از عملگر == برای مقایسه آنها، از تابع Compare در System.String استفاده میکنیم. این تابع سه پارامتر را به عنوان ورودی دریافت می کند. پارامتر اول و دوم رشته هایی هستند که باید با یکدیگر مقایسه شوند. در این مثال برای پارامتر اول مقدار درون متغییر strName و برای پارامتر دوم ثابت رشته ای "STEPHANIE" را فرستاده ایم. پارامتر سوم هم برای تابع مشخص می کند که در هنگام مقایسه، نوع حروف را نادیده بگیرد یا نه. در اینجا مقدار true به این معنی است که بزرگی و یا

کوچکی حروف به وسیله تابع نادیده گرفته شوند. اگر مقدار این پارامتر را برابر با false قرار دهید، عملکرد این تابع دقیقا مـشابه استفاده از عملگر == خواهد بود.

```
// Compare the name
if (String.Compare(strName, "STEPHANIE", True) == 0)
     MessageBox.Show("Hello, Stephanie!",
                     "And Or Demo");
```

نتیجه ای که به وسیله این تابع برگردانده می شود مقداری عجیب است. این تابع به جای مقدار true و یا false، یک عـدد صحیح را به عنوان نتیجه برمی گرداند. در حقیقت این تابع علاوه بر مشخص کردن برابری و یا نابرابری دو رشته، می تواند مشخص کند که در صورت نابرابری آنها کدامیک بزرگتر می باشند. اگر مقدار صفر توسط تابع برگردانده شد به این معنی است که دو رشته بـا هم برابرند، اگر عدد منفی برگردانده شود یعنی رشته اول کوچکتر از رشته دوم است و عدد مثبت نیز به این معنی است که رشـته اول بزرگتر از رشته دوم است. از این اعداد میتوان برای نوشتن الگوریتم های مرتب سازی و یا جستجو استفاده کرد.

انتخاب بین حالتها با استفاده از switch:

در بعضی از مواقع باید شرط هایی را مشابه زیر بررسی کنید:

- آیا نام مشتری برابر با Bryan است؟ در این صورت عمل A را انجام بده.
- آیا نام مشتری برابر با Stephanie است؟ در این صورت عمل B را انجام بده.
- آیا نام مشتری برابر با Cathy است؟ در این صورت عمل C را انجام بده.
- آیا نام مشتری برابر با Betty است؟ در این صورت عمل D را انجام بده.
- آیا نام مشتری برابر با Edward است؟ در این صورت عمل E را انجام بده.

اگر بخواهید این کار را با استفاده از دستور if انجام دهید، باید از کدی مشابه زیر استفاده کنید:

```
if (Customer.Name == "Bryan")
     // Do A
else if(Customer.Name == "Stephanie")
     // Do B
else if(Customer.Name == "Cathy")
     // Do C
else if(Customer.Name == "Betty")
     // Do D
else if(Customer.Name == "Edward")
     // Do E
```

در این حالت اگر بخواهید بعد از مدتی کد را به صورتی تغیر دهید که به جای استفاده از نام مشتری از نام کوچک او استفاده کند، چه کـاری بایـد بکنیـد؟ در ایـن حالـت بایـد عبـارت Customer.Name را در تمـام دسـتورات if بـه

Customer.FirstName تغیر دهید که کار بسیار وقت گیری است. همچنین استفاده از این روش خوانایی برنامه را نیز کم می کند. در امتحان کنید بعد با روشی آشنا میشویم که بتوانیم بهتر این شرط ها را بررسی کنیم.

امتحان کنید: استفاده از دستور `switch`

۱) یک پروژه ویندوزی جدید به نام Switch Demo ایجاد کنید. سپس خاصیت Name مربوط به فرم را برابر Switch قرار دهید.

۲) با استفاده از جعبه ابزار یک ListBox به برنامه اضافه کنید. خاصیت Name آن را برابر lstData، خاصیت Dock را به Fill و خاصیت IntegralHeight را به False تغییر دهید.

۳) هنگامی که کنترل ListBox در فرم انتخاب شده است، به پنجره Properties بروید و خاصیت Items را انتخاب کنید و بر روی دکمه سمت راست آن کلیک کنید. پنجره ای به نام String Collection Editor نمایش داده خواهد شد. مطابق شکل ۴-۱۰، پنج نام مختلف را در آن وارد کنید.



شکل ۴-۱۰

۴) بر روی کلید OK کلیک کنید تا نامها به ListBox اضافه شوند. سپس بر روی آن دو بار کلیک کنید تا متد مربوط به رویداد SelectedIndexChanged ایجاد شود. کد زیر را در آن وارد کنید:

```
private void lstData_SelectedIndexChanged(object sender,
                                          EventArgs e)
{
    // Declare variables
    string strName;
    string strFavoriteColor = "";

    // Get the selected name
    strName =
    lstData.Items[lstData.SelectedIndex].ToString();

    // Use a Switch to get the favorite color
    // of the selected name
```

```
        switch(strName)
        {
            case "Bryan":
                strFavoriteColor = "Madras Yellow";
                break;
            case "Stephanie":
                strFavoriteColor = "Sea Blue";
                break;
            case "Cathy":
                strFavoriteColor = "Morning Mist";
                break;
            case "Betty":
                strFavoriteColor = "Passionate Purple";
                break;
            case "Edward":
                strFavoriteColor = "Battleship Gray";
                break;
        }
        // Display the favorite color of the selected name
        MessageBox.Show(strName + "'s favorite color is " +
                        strFavoriteColor, "Select Demo");
}
```

۵) برنامه را اجرا کنید. هر بار که روی نامی در ListBox کلیک کنید، کادر پیغامی مشابه شکل ۴-۱۱ نمایش داده خواهد شد.



شکل ۴-۱۱

چگونه کار می کند؟

اولین کاری که در رویداد SelectedIndexChanged باید انجام دهید این است که متغییر های مورد نیاز را تعریف کنید. سپس باید مشخص کنید که کدام نام در ListBox انتخاب شده است. برای این کار باید آیتمی که شماره آن برابر با

خاصیت SelectedIndex است را پیدا کنید و متن داخل آن آیتم را در متغییر قرار دهید. این کار به وسیله کد زیر صورت می گیرد:

```
    // Declare variables
    string strName;
    string strFavoriteColor = "";

    // Get the selected name
    strName =
    lstData.Items[lstData.SelectedIndex].ToString();
```

در این قسمت از کد به چند نکته توجه کنید. اول اینکه همانطور که مشاهده می کنید، هنگام تعریف متغییر strFavoriteColor به آن مقدار اولیه داده ایم. اگر این مقدار اولیه را از کد حذف کنید، برنامه هنگام کامپایل با خطا مواجه می شود.

در C# هر متغییری قبل از استفاده باید دارای مقدار اولیه باشد و رعایت این مورد در کد، هنگام کامپایل توسط کامپایلر بررسی می شود. مقدار دهی اولیه به متغییر ها حتما باید قبل از استفاده از آنها و همچنین در خارج از بخشهایی از کد که فقط در شرایط خاص اجرا می شوند (همانند بلاک دستور if)، انجام شود.

در این قسمت قبل از استفاده از این متغییر به آن مقدار اولیه داده ایم، ولی این کار را در بلاک دستورات switch قرار داده ایم. فرض کنید که هیچ یک از حالتهای دستور switch اجرا نشوند. در این حالت هنگامی که کامپایلر به خط بعد از دستور switch برسد و بخواهد مقدار متغییر strFavoriteColot را چاپ کند، این متغییر مقداری نخواهد داشت. بنابراین برنامه با خطا مواجه خواهد شد. به همین دلیل در هنگام تعریف متغییر به آن مقدار اولیه داده ایم تا در این موارد هم متغییر دارای مقدار باشد.

هنگامی که نام انتخاب شده در لیست را بدست آوردید، میتوانید حالتهای مختلف آن را با استفاده از دستور switch بررسی کنید. برای استفاده از این دستور، نام متغییری که میخواهید بررسی کنید را باید در داخل پرانتز مقابل دستور وارد کنید.

درون بلاک دستور switch باید برای هر حالت که میخواهید بررسی کنید یک دستور case مجزا قرار دهید. در این مثال، پنج دستور case دارید که هر کدام به یک نام مربوط هستند. هنگام اجرای این کد، اگر ویژوال C# به یکی از این حالتها برخورد کند، کد آن را اجرا می کند.

در نظر داشته باشید برای اینکه بعد از اجرای کد مربوط به یک بلاک، اجرای برنامه به خط بعد از دستور switch منتقل شود، باید در انتهای دستورات آن بلاک از دستور break به نحوی که در برنامه نشان داده شده است استفاده کنید.

```
    // Use a Switch to get the favorite color
    // of the selected name
    switch(strName)
    {
        case "Bryan":
            strFavoriteColor = "Madras Yellow";
            break;
        case "Stephanie":
            strFavoriteColor = "Sea Blue";
            break;
        case "Cathy":
            strFavoriteColor = "Morning Mist";
```

```
                break;
            case "Betty":
                strFavoriteColor = "Passionate Purple";
                break;
            case "Edward":
                strFavoriteColor = "Battleship Gray";
                break;
        }
        // Display the favorite color of the selected name
    MessageBox.Show(strName + "'s favorite color is " +
        strFavoriteColor, "Select Demo");
```

در زیر مراحلی که با انتخاب یک نام از لیست باکس توسط برنامه طی می شوند، را بررسی می کنیم:

- فرض کنید کاربر در لیست روی نام `Betty` کلیک می کند. رویداد `SelectedIndexChanged` فراخوانی می شود و مقدار `"Betty"` در متغییر `strName` ذخیره می شود.
- برنامه به دستور `switch` می رسد و مقدار درون متغییر `strName` را با تک تک مقادیری که به عنوان حالتهای مختلف این دستور وارد شده اند بررسی می کند.
- بعد از بررسی حالتهای وارد شده در دستور `switch`، برنامه متوجه می شود که حالت چهارم با مقدار متغییر برابر است. بنابراین دستورات این حالت را اجرا می کند (یعنی مقدار `strFavoriteColor` را برابر `"Passionate Purple"` قرار می دهد).
- برنامه از بلاک دستورات `switch` خارج شده و اولین خط بعد از آن را اجرا می کند و کادر پیغام مناسبی را به کاربر نمایش می دهد.

استفاده از switch با و بدون حساسیت به نوع حروف:

همانند دستور `if`، دستور `switch` هم به بزرگی و کوچکی حروف حساس است. به بخش امتحان کنید زیر توجه کنید:

امتحان کنید: استفاده از `switch` با حساسیت به نوع حروف

۱) قسمت طراحی فرم مربوط به `Form1` را باز کرده و کنترل `ListBox` را در فرم انتخاب کنید. از پنجره `Properties` مربوط به کنترل `ListBox`، گزینه `Items` را انتخاب کنید و سپس روی دکمه جلوی این خاصیت کلیک کنید تا پنجره `String Collection Editor` باز شود.

۲) اسامی موجود در این پنجره را به گونه ای تغییر دهید که همانند شکل ۴-۱۲ همه با حروف بزرگ نوشته شوند.



شکل ۴-۱۲

۳) روی دکمه OK کلیک کنید تا تغیرات این پنجره ذخیره شوند و سپس برنامه را اجرا کنید. مشاهده خواهید کرد که با کلیک بر روی هر یک از نامهای داخل `ListBox`، کادر پیغامی که ظاهر می شود نام رنگ مورد نظر را نمایش نمی دهد. (شکل ۴-۱۳)



شکل ۴-۱۳

چگونه کار می کند؟

دستور `switch` هم همانند دستور `if` نسبت به نوع حروف حساس است. بنابراین اگر در شرط دستور `switch` از نامهای BETTY و یا CATHY استفاده کنید، هیچ حالت مناسبی یافته نخواهد شد. این مورد مانند این است که به صورت زیر از دستور `if` استفاده کنید:

```
if ("BETTY" == "Betty")
```

یا:

```
if ("CATHY" == "Cathy")
```

در بخشهای قبلی دیدید که برای مقایسه دو رشته بدون در نظر گرفتن نوع حروف می توان از تابع `String.Compare` استفاده کرد. اما در دستور `switch` نمی توانید از این روش استفاده کنید. در بخش امتحان کنید بعد، روشی را خواهیم دید که به وسیله آن می توانید مقایسه های دستور `switch` را هم بدون حساسیت به نوع حروف انجام دهید.

امتحان کنید: استفاده از `switch` بدون حساسیت به نوع حروف

۱) بخش ویرایشگر کد را برای `Form1` باز کنید و در متد مربوط بـه رویـداد `SelectedIndexChanged` تغیرات زیر را انجام دهید. دقت کنید که حتما رشته هایی را که برای بررسی در جلوی دستور `case` قرار میدهید حتما با حروف کوچک نوشته شوند.

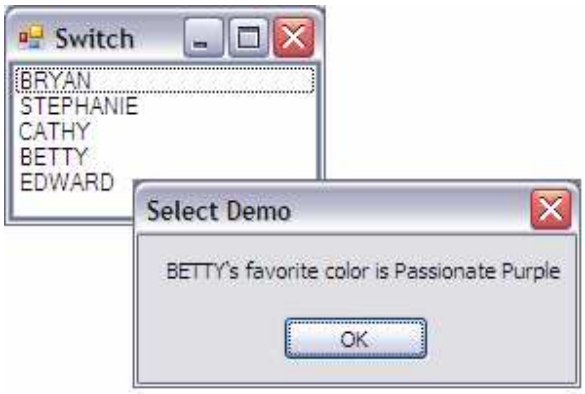
```
private void lstData_SelectedIndexChanged(object sender,
                                          EventArgs e)
{
     // Declare variables
     string strName;
     string strFavoriteColor = "";

     // Get the selected name
     strName =
     lstData.Items[lstData.SelectedIndex].ToString();

     // Use a Switch to get the favorite color
     // of the selected name
     switch(strName.ToLower())
     {
          case "bryan":
               strFavoriteColor = "Madras Yellow";
               break ;
          case "stephanie":
               strFavoriteColor = "Sea Blue";
               break;
          case "cathy":
               strFavoriteColor = "Morning Mist";
               break;
          case "betty":
               strFavoriteColor = "Passionate Purple";
               break;
          case "edward":
               strFavoriteColor = "Battleship Gray";
               break;
     }
     // Display the favorite color of the selected name
     MessageBox.Show(strName + "'s favorite color is " +
          strFavoriteColor, "Select Demo");
```

```
}
```

۲) برنامه را اجرا کنید و مجددا بر روی یکی از اسامی درون ListBox کلیک کنید. همانند شکل ۴-۱۴ کادر پیغامی را مشاهده خواهید کرد که نام فرد انتخاب شده و رنگ مورد نظر او را نمایش می دهد.



شکل ۴-۱۴

چگونه کار می کند؟

برای این که رشته ی داخل متغییر strName بدون حساسیت به نوع حروف بررسی شوند، می توانید با استفاده از تابع ToLower تمام حروف آن را به حروف کوچک تبدیل کنید و سپس آن را با مقادیر مختلف مقایسه کنید.

```
switch(strName.ToLower())
```

این دستور مقدار متغییر strName را ابتدا به حروف کوچک تبدیل می کند و سپس آن را با مقادیر موجود در دستور case بررسی می کند. در استفاده از این روش باید دقت کنید که تمام عبارتهای مقابل دستور case را با حروف کوچک بنویسید. در غیر این صورت، حالت مورد نظر اجرا نخواهد شد. برای مثال فرض کنید عبارت جلوی دستور case برابر با "Betty" باشد و کاربر نیز نام BETTY را از لیست باکس انتخاب می کند. در این حالت، این نام با استفاده از تابع ToLower به betty تبدیل میشود و سپس مقدار "Betty" با "betty" مقایسه می شود که به علت برابر نبودن این دو مقدار، کد مربوط به این بخش اجرا نخواهد شد.

```
case "bryan":
     strFavoriteColor = "Madras Yellow";
     break ;
case "stephanie":
     strFavoriteColor = "Sea Blue";
     break;
case "cathy":
     strFavoriteColor = "Morning Mist";
     break;
case "betty":
     strFavoriteColor = "Passionate Purple";
```

```
                break;
            case "edward":
                strFavoriteColor = "Battleship Gray";
                break;
        }
```

در انتها، هنگامی که نام رنگ مورد نظر را بدست آوردید، آن را با استفاده از یک کادر پیغام به کاربر نمایش می دهید.

نکته: به جای استفاده از حروف کوچک، میتوانید تمام عبارتهای مقابل دستورات `case` را با حروف بزرگ بنویسید و سپس با استفاده از تابع `ToUpper`، رشته مورد نظر خودتان را به حروف بزرگ تبدیل کنید.

انتخابهای چند گانه:

در بررسی حالتهای مختلف در شرط مقابل دستور `case` اجباری نیست که فقط یک حالت را بررسی کنید، بلکه میتوانید چند حالت را در مقابل یک `case` بررسی کنید، تا در صورتی که هر کدام از آنها رخ داد، کد مورد نظر اجرا شود. در بخش امتحان کنید زیر، برنامه بالا را به گونه ای تغییر می دهیم که با انتخاب نام از `ListBox`، جنسیت فرد انتخاب شده را نمایش دهد.

امتحان کنید: انتخابهای چند گانه

۱) محیط ویرایشگر کد را برای `Form1` باز کنید و تغیرات مشخص شده در زیر را در متد مربوط به رویداد `SelectedIndexChanged` اعمال کنید:

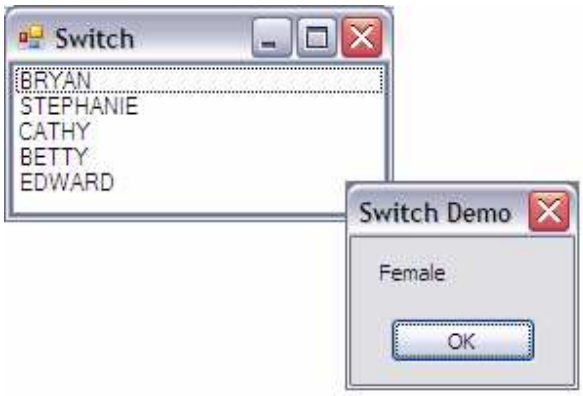
```
private void lstData_SelectedIndexChanged(object sender,
                                          EventArgs e)
{
    // Declare variables
    string strName;

    // Get the selected name
    strName =
        lstData.Items[lstData.SelectedIndex].ToString();

    // Use a Switch to display a person's gender
    switch (strName.ToLower())
    {
        case "bryan":
        case "edward":
            MessageBox.Show("Male", "Switch Demo");
            break;
        case "stephanie":
```

```
            case "cathy":
            case "betty":
                MessageBox.Show("Female", "Switch Demo");
                break;
        }
}
```

۲) برنامه را اجرا کنید و روی یکی از اسامی داخل `ListBox` کلیک کنید. مشاهده می کنید که کادر پیغامی نمایش داده می شود و جنسیت فرد انتخاب شده را نمایش می دهد (شکل ۴-۱۵).



شکل ۴-۱۵

چگونه کار می کند؟

کد مربوط به دریافت نام و مقدار دهی اولیه به متغییر برای استفاده در `switch`، همانند قبل است و تفاوتی ندارد. اگر بخواهیم برای چند شرط مختلف یک کد اجرا شود، باید برای هر شرط یک بار دستور `case` را بنویسیم، با این تفاوت که فقط برای مورد آخر، کد و دستور `break` را می نویسیم. در بقیه دستورات `case`، فقط شرط مورد نظر را وارد می کنیم. این مورد موجب می شود که این شرط ها همانند استفاده از عملگر `Or` در دستور `if` با یکدیگر ترکیب شوند. بنابراین هر کدام از آنها که درست باشد، کدی که بعد از آخرین دستور `case` آمده است اجرا می شود. برای مثال در مورد اول، اگر اسم انتخاب شده `bryan` و یا `edward` باشد، کدهای بعد از دو دستور `case` اجرا می شوند.

```
case "bryan":
case "edward":
    strFavoriteColor = "Madras Yellow";
    break ;
```

دقت داشته باشید که با این روش، در حقیقت در حال ترکیب این شرطها با عملگر "یا" هستید و می گویید "یا این مورد یا آن مورد" نه اینکه "این مورد و آن مورد".

دستور default:

تاکنون مشاهده کردیم که چگونه شرطهای مختلف را در دستور switch بررسی کنیم. اما سوال این است که اگـر هـیچ یـک از این شرط ها اجرا نشدند چه می شود؟ این مورد که هیچ یک از شرط ها اجرا نشوند را در بخش قبلی، هنگامی کـه در حـال بررسـی مقایسه با حساسیت به نوع حروف بودیم مشاهده کردیم. اما چگونه می شود کدی را به نحوی مشخص کرد که هر گاه هـیچ یـک از شرط ها برقرار نبودند اجرا شود؟ در بخش امتحان کنید بعد، این مورد را بررسی خواهیم کرد.

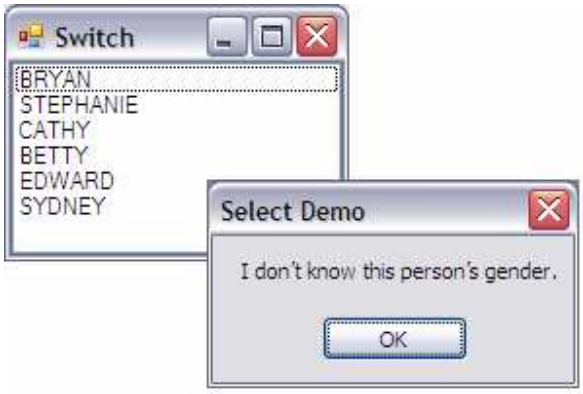
امتحان کنید: استفاده از دستور `default`

۱) قسمت طراحی فرم را برای Form1 باز کنید و کنترل ListBox را از روی فرم انتخاب کنیـد. سـپس بـه پنجـره Properties برویـد و بـا اسـتفاده از خاصـیت Items، پنجـره String Collection Editor را مجددا باز کنید. نام دیگری به لیست اضافه کنید و روی دکمه OK کلیک کنید.

۲) در متد lstData_SelectedIndexChanged بخش مربوط به دستور switch را بـه صـورت زیر تغییر دهید:

```
        switch(strName.ToLower())
        {
            case "bryan":
            case "edward":
                MessageBox.Show("Male", "Switch Demo");
                break;
            case "stephanie":
            case "cathy":
            case "betty":
                MessageBox.Show("Female", "Switch Demo");
                break;
            default:
                MessageBox.Show("I don't know this person's
"                                   + "gender.", "Select Demo");
                break;
        }
```

۳) برنامه را اجرا کنید و از ListBox نامی که جدیداً به برنامه لیست اضافه شده است را انتخاب کنید. نتیجه ای مـشابه شکل ۴-۱۶ مشاهده خواهید کرد.



شکل ۴-۱۶

چگونه کار می کند؟

دستوراتی که بعد از بخش `default` وارد می شوند، هنگامی اجرا می شوند که هیچ یک از شرایطی که در بخش `case` قید شده است برابر با مقدار متغییر نباشد. برای مثال در برنامه بالا هیچ دستور `case` برای عبارت `"Sydney"` وارد نشده است، بنابراین دستورات بخش `default` اجرا می شوند و کادر پیغامی نمایش داده می شود که می گویید جنسیت فرد انتخاب شده مشخص نیست.

نکته: دقت داشته باشید که همواره در انتهای دستورات بخش `case` و یا بخش `default` از دستور `break` استفاده کنید تا بعد از اجرای کد کنترل برنامه به خط بعد از دستور `switch` برگردد. در غیر این صورت با خطای زمان کامپایل مواجه خواهید شد.

استفاده از نوع های داده ای گوناگون در دستور switch:

در این درس، فقط از متغییر های رشته ای در دستور `switch` استفاده کردید. اما می توانید این دستور را با انواع متغییر های موجود در C# همانند اعداد صحیح (`int`)، اعداد اعشاری (`double` و `float`) و یا متغییر های `Boolean` هم استفاده کنید.
هر نوع متغییری که بتواند در دستور `if` با استفاده از عملگر `==` بررسی شود، میتواند در دستور `switch` نیز به کار رود. اما معمولا از متغییر هایی که دارای مقادیر پیوسته هستند مانند متغییر های رشته ای و یا اعداد صحیح برای دستور `switch` استفاده میکنند. زیرا اعداد اعشاری می توانند مقدارهای کسری مانند ۲,۲، ۲,۲۱، ۲,۲۲۱ و ... را داشته باشد و استفاده از آنها در دستور `switch` منطقی نیست.

حلقه ها:

هنگامی که در حال نوشتن یک برنامه کامپیوتری هستید، ممکن است بخواهید یک عمل مشخص را چندین بار متوالی انجام دهید تا نتیجه مطلوب خود را دریافت کنید. مثلا ممکن است بخواهید صورت حساب تلفن را برای تمام مشترکین بدست آورید و یا ۱۰ فایل را از روی کامپیوتر بخوانید.
در برنامه نویسی، برای انجام این امور معمولا از حلقه ها استفاده می شود. در این بخش با سه دسته کلی از حلقه ها که در C# وجود دارند آشنا خواهیم شد.

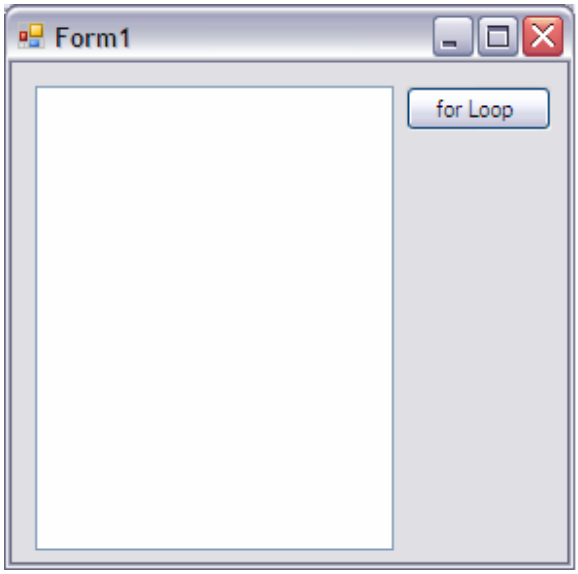
- حلقه های `for` – این حلقه ها معمولا به تعداد مرتبه مشخصی اجرا می شوند (برای مثال، دقیقا ۱۰ بار).
- حلقه های `while` – این حلقه ها معمولا تا هنگامی که نتیجه یک شرط درست شود ادامه پیدا می کنند.
- حلقه های `do` – عملکرد این حلقه ها نیز همانند حلقه های `while` است، با این تفاوت که شرط در حلقه های `while` در ابتدا بررسی می شود ولی در این حلقه ها، شرط در انتها بررسی میشود.

حلقه for:

حلقه `for` حلقه ای است که درک نحوه کارکرد آن بسیار راحت است. در امتحان کنید بعد، با این دستور آشنا خواهیم شد.

امتحان کنید: ایجاد یک حلقه `for`

۱) یک پروژه ویندوزی جدید به نام `Loops` ایجاد کنید.
۲) در فرمی که ظاهر می شود یک `ListBox` و یک کنترل `Button` اضافه کنید.
۳) خاصیت `Name` مربوط به `ListBox` را برابر `lstData` و خاصیت `IntegralHeight` آن را برابر `false` قرار دهید.
۴) خاصیت `Name` مربوط به دکمه فرمان `Button` را برابر `btnForLoop` و خاصیت `Text` آن را برابر `For Loop` قرار دهید. تاکنون فرم شما باید مشابه شکل ۴-۱۷ شده باشد.



شکل ۴-۱۷

۵) بر روی دکمه فرمان دو بار کلیک کنید تا متد مربوط به رویداد `Click` آن ایجاد شود. سپس کد زیر را در آن وارد کنید:

```
private void btnForLoop_Click(object sender, EventArgs e)
{
     // Declare variable
     int intCount;

     // Perform a loop
     for(intCount = 1;intCount <= 5;intCount += 1)
     {
          // Add the item to the list
          lstData.Items.Add("I'm item " + intCount +
                            " in the list!");
     }
}
```

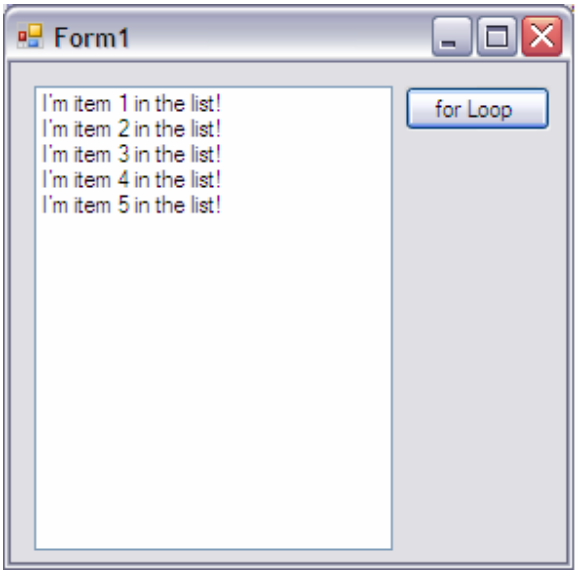
۶) برنامه را اجرا کنید و بر روی دکمه `for Loop` کلیک کنید. نتیجه ای مشابه شکل ۴-۱۸ مشاهده خواهید کرد.

چگونه کار می کند؟

در ابتدای متد مربوط به رویداد کلیک، متغییری را به صورت زیر تعریف می کنیم:

```
     // Declare variable
     int intCount;
```

برای ایجاد یک حلقه باید از کلمه کلیدی `for` استفاده کنید. این کلمه به کامپایلر ویژوال C# میگوید که می خواهید یک حلقه با تعداد دفعات تکرار مشخص ایجاد کنید. تمام کلمات و علامتهایی که بعد از این کلمه می آیند، برای مشخص کردن نحوه عملکرد این حلقه به کار می روند. برای تعیین نحوه کارکرد یک حلقه، سه مورد را باید در جلوی آن مشخص کنید. این سه مورد، همانطور که در کد مشاهده می کنید با کاراکتر `";"` از یکدیگر جدا می شوند.



شکل ۴-۱۸

در قسمت اول باید مشخص کنید که از چه متغییری میخواهید برای شمارش دفعات تکرار در این حلقه استفاده کنید. همچنین در این بخش مقدار اولیه متغییر را نیز تعیین کنید. به عبارت دیگر در این قسمت باید مشخص کنید که میخواهید شمارش در حلقه از چه عددی شروع شود. در این مثال برای شمارش حلقه از متغییر `intCount` که در خط قبل تعریف کردیم، استفاده می کنیم و مقدار اولیه آن را نیز ۱ تعیین میکنیم تا شمارش حلقه از عدد یک شروع شود.

در قسمت دوم باید تعیین کنیم که حلقه، شمارش را تا چه عددی ادامه دهد. در این مثال تا زمانی که متغییر `intCount` کوچکتر و یا مساوی ۵ است شمارش ادامه پیدا میکند. توجه داشته باشید که برای شرط این قسمت از هر یک از عملگرهایی که در بخش قبل معرفی کردیم همانند عملگر بزرگتر مساوی و یا عملگر کوچکتر و ... میتوانید استفاده کنید.

در قسمت آخر مشخص کنید که مقدار متغییر در هر مرحله باید چه تغییری کند. در این مثال میخواهیم مقدار متغییر را در هر مرحله از اجرای حلقه یک واحد افزایش یابد. بنابراین در این قسمت مقدار متغییر را با عدد یک جمع می کنیم.

از توضیحات قبلی مشخص است که این حلقه از عدد یک شروع به شمارش می کند و تا عدد ۵ شمارش را ادامه می دهد و در هر مرحله نیز یک واحد به مقدار متغییر اضافه می کند. بنابراین دستورات داخل حلقه پنج بار اجرا می شوند.

مراحلی که برای اجرای این حلقه به وسیله ویژوال C# بررسی می شود به این ترتیب است که ابتدا متغییر `intCount` برابر با مقدار مشخص شده در برنامه می شود. سپس شرط وارد شده در قسمت دوم حلقه بررسی میشود. در صورتی که این شرط برقرار نباشد دستورات حلقه اجرا نمی شوند و برنامه از خط بعد از حلقه ادامه پیدا می کند. در صورتی که شرط حلقه برقرار باشد (یعنی در این مثال مقدار متغییر `intCount` کوچکتر یا مساوی عدد ۵ باشد) دستورات داخل حلقه اجرا می شوند. بعد از اینکه دستورات حلقه برای مرتبه اول اجرا شدند، ویژوال C# تغییراتی را که در قسمت سوم مشخص شده است بر روی متغییر اعمال می کند (برای مثال در این برنامه، یک واحد به متغییر `intCount` اضافه می کند) و سپس شرط وارد شده در قسمت دوم حلقه `for` را

مجددا بررسی میکند و در صورت درست بودن این شرط دستورات داخل حلقه اجرا می شوند. این شرایط ادامه پیدا می کنند تا زمـانی که شرط قسمت دوم حلقه for نادرست باشد. در این هنگام کامپایلر ویژوال C# از حلقه خارج می شود و بـه اولـین خـط بعـد از حلقه می رود.

```
// Perform a loop
for(intCount = 1;intCount <= 5;intCount += 1)
{
     // Add the item to the list
     lstData1.Items.Add("I'm item " + intCount +
                        " in the list!");
}
```

نکته: در قسمت سوم حلقه ی بالا، به جای استفاده از عبارت intCount+=1 می توانستیم از دستور intCount++ نیز استفاده کنیم. این دستور باعث می شود که یک واحد به متغییر intCount اضـافه شـود و حاصـل در همـان متغییـر قـرار بگیرد.

```
for(intCount = 1;intCount <= 5;intCount++)
```

علاوه بر عملگر ++ که یک واحد به یک متغییر اضافه میکند، عملگر -- نیز وجود دارد که یک واحـد از مقـدار یـک متغییـر مـی کاهد.کاربرد این عملگر همانند عملگر ++ است.

همانطور که تاکنون متوجه شده اید، در حلقه for اجباری نیست که مقدار شروع حلقه را عدد یک در نظر بگیرید و یا شـمارنده در هر مرحله متغییر را فقط یک واحد افزایش دهد. در امتحان کنید زیر حلقه for جدیدی خواهیم ساخت کـه از ایـن مـوارد اسـتفاده کند.

امتحان کنید: انعطاف پذیری حلقه for

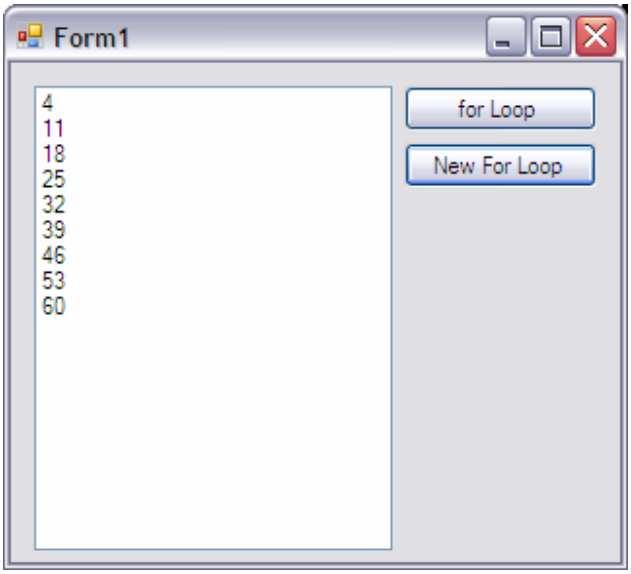
۱) اگر برنامه قبلی همچنان در حال اجرا است آن را ببندید و سـپس کنتـرل Button دیگـری روی فـرم اضـافه کنیـد. خاصیت Name آن را برابر btnNewForLoop و خاصیت Text آن را برابر New For Loop قرار دهید.

۲) روی این کنترل دو بار کلیک کنید و کد زیر را در متد مربوط به رویداد Click آن وارد کنید.

```
private void btnNewForLoop_Click(object sender, EventArgs
e)
{
     // Perform a loop
     for (int intCount = 4; intCount < 62; intCount += 7)
     {
          // add the item to the list
          lstData.Items.Add(intCount);
```

```
        }
}
```

۳) برنامه را اجرا کنید و روی دکمه `New For Loop` جدید کلیک کنید. نتیجه ای مشابه شکل ۴-۱۹ را دریافت خواهید کرد.



شکل ۴-۱۹

چگونه کار می کند؟

به تعریف حلقه `for` در این مثال توجه کنید:

```
// Perform a loop
for (int intCount = 4; intCount < 62; intCount += 7)
```

نکته اولی که در این حلقه وجود دارد، تعریف متغییر `intCount` در خود حلقه است. این مورد باعث می شود برنامه هنگامی که به حلقه رسید متغییری را به نام `intCount` تعریف کند و برای شمارش درون حلقه از آن استفاده کند. هنگامی که کار حلقه به پایان رسید، متغییر نیز از بین خواهد رفت و فضای اشغال شده توسط آن آزاد می شود. در مرحله بعد، به جای استفاده از عدد ۱ به عنوان مقدار شروع، از عدد ۴ استفاده کرده ایم. در حقیقت در اولین دوره اجرای حلقه مقدار `intCount` برابر با عدد ۴ است و بنابراین اولین مورد اضافه شده به لیست عدد ۴ خواهد بود. همچنین در هر مرحله از اجرای حلقه، ۷ واحد به مقدار `intCount` افزوده می شود. به همین دلیل، دومین موردی که به لیست اضافه می شود عدد ۱۱ است، نه عدد ۵.

با وجود اینکه حلقه باید در عدد ۶۲ به پایان برسد، اما مشاهده میکنید که حلقه در عدد ۶۰ به پایان می رسد. زیرا عدد بعد از آن، ۶۷ خواهد بود که از ۶۲ بزرگتر است. بنابراین حلقه برای مرتبه نهم اجرا نخواهد شد.

شمارش معکوس در حلقه:

اگر در هر مرحله از اجرای حلقه عددی را از شمارنده ی آن کم کنید، حلقه به صورت معکوس حرکت خواهد کرد. در امتحان کنید بخش بعد، این مورد را مشاهده خواهید کرد.

امتحان کنید: شمارش معکوس حلقه

۱) اگر هنوز برنامه قسمت قبل در حال اجرا است آن را ببندید و سپس کنترل `Button` دیگری به فرم اضافه کنید. خاصیت `Name` آن را برابر `btnBackwardsForLoop` و خاصیت `Text` آن را برابر `Backwards For Loop` قرار دهید.

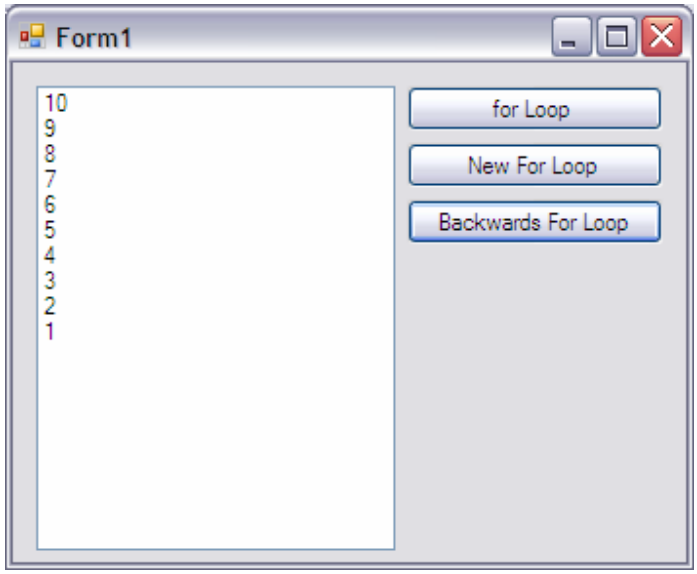
۲) روی این کنترل دو بار کلیک کنید و کد زیر را در متد مربوط به رویداد کلیک آن وارد کنید:

```
private void btnBackwardsForLoop_Click(object sender,
                                       EventArgs e)
{
     // Perform a loop
     for (int intCount = 10; intCount >= 1; intCount--)
     {
          // Add the item to the list
          lstData.Items.Add(intCount);
     }
}
```

۳) برنامه را اجرا کنید و روی دکمه `Backwards For Loop` کلیک کنید. نتیجه ای را مشابه شکل ۴-۲۰ مشاهده خواهید کرد.

چگونه کار می کند؟

همانطور که در قسمت سوم تعریف حلقه `for` مشاهده می کنید، در هر مرتبه اجرای حلقه، استفاده از عملگر `--` موجب می شود یک واحد از مقدار `intCount` کم شود. چون مقدار اولیه این متغییر برابر ۱۰ در نظر گرفته شده است، حلقه ۱۰ بار اجرا می شود و از عدد ۱۰ به صورت معکوس به عدد ۱ می رسد و اجرای حلقه تمام می شود.



شکل ۴-۲۰

حلقه های foreach:

در استفاده روزمره از حلقه `for` در برنامه ها، کمتر از این حلقه به نحوی که شرح داده شد استفاده می شود. به علت نحوه کارکردی که چارچوب `.NET` دارد، معمولا در برنامه ها با نوع خاصی از این حلقه که `foreach` نامیده می شود بیشتر کار خواهیم کرد.

در الگوریتم برنامه ها عموما هنگامی از یک حلقه استفاده می کنید که مجموعه ای از اشیا را در اختیار داشته باشید و بخواهید بین اعضای آن جا به جا شوید، که این مجموعه هم اغلب به صورت یک آرایه است. برای مثال ممکن است بخواهید بین تمام فایلهای درون یک فولدر بگردید و فایلی را پیدا کنید که اندازه آن بیش از حد مجاز است. هنگامی که از چارچوب `.NET` بخواهید که لیست تمام فایلها را به شما برگرداند، یک آرایه از اشیا را دریافت خواهید کرد که هر کدام از اعضای آن نشان دهنده ی یک فایل است. در امتحان کنید بعد، حلقه داخل برنامه خود را به نحوی تغییر خواهید داد که نام تمام فولدرهای داخل درایو `C` شما را برگرداند.

امتحان کنید: حلقه `foreach`

۱) کنترل `Button` جدیدی به برنامه اضافه کنید، خاصیت `Name` آن را برابر `btnForEachLoop` و خاصیت `Text` آن را برابر `ForEach Loop` قرار دهید.

۲) روی این کنترل دو بار کلیک کنید و کد زیر را در متد مربوط به رویداد `Click` آن وارد کنید:

```
private void btnForEachLoop_Click(object sender,
                                  EventArgs e)
{
    // List each folder at the root of your C Drive
    foreach (string strFolder
```

```
            in System.IO.Directory.GetDirectories("C:\\"))
        {
            // Add the item to the list
            lstData.Items.Add(strFolder);
        }
}
```

۳) برنامه را اجرا کنید و روی دکمه ی ForEach Loop کلیک کنید. در ListBox نام تمامی فولدرهای موجود در درایو C خود را مشاهده خواهید کرد.

چگونه کار می کند؟

برای بدست آوردن لیست تمام دایرکتوری های موجود در یک مسیر خاص در برنامه باید از تابع GetFirectories مربوط به کلاس Directory در فضای نام[1] System.IO استفاده کنیم. این تابع یک آرایه رشته ای از نام تمام دایرکتوری های موجود در مسیری که برای آن مشخص شده است را برمی گرداند. در این برنامه از این تابع برای دریافت نام تمامی دایرکتوری های موجود در درایو C استفاده کرده ایم.
اصل کار حلقه foreach به این صورت است که به وسیله ی آن شما می توانید در بین تمامی اشیای موجود در یک آرایه خاص (که تعداد آن را نیز نمی دانید) حرکت کنید. برای ایجاد این حلقه به منبعی از اشیا نیاز دارید (در این مثال یک آرایه از رشته ها) و یک متغییر کنترل کننده که در هر مرحله، شیئ مورد بررسی در آن قرار خواهد گرفت. در این مثال تابع GetDirectories، آرایه ای از اشیا را به عنوان منبع برای حلقه foreach فراهم می کند و متغییر strFolder به عنوان متغییر کنترل کننده به کار می رود:

```
foreach (string strFolder
        in System.IO.Directory.GetDirectories("C:\\"))
{
        // Add the item to the list
        lstData.Items.Add(strFolder);
}
```

این عبارت به این معنی است که در مرحله اول، strFolder برابر با اولین آیتم در آرایه رشته ای است (در این جا برابر با فولدر "C:\Documents and Settings") . شما میتوانید این فولدر را با استفاده از دستور زیر به ListBox اضافه کنید.

```
// Add the item to the list
lstData.Items.Add(strFolder);
```

همانند حلقه های for عادی در هر مرحله، یکی از عناصر این آرایه در متغییر strFolder قرار میگیرد و سپس مقدار آن به ListBox اضافه می شود.

[1] با مفهوم فضای نام در فصل ۹ آشنا خواهیم شد.

نکته: همانطور که مشاهده می کنید به جای استفاده از رشته "C:\" به عنوان پارامتر برای تابع GetDirectories، از رشته "C:\\" استفاده کرده ایم. در زبانهای برنامه نویسی خانواده C از جمله C#، کاراکتر \ به عنوان یک کاراکتر کنترلی در نظر گرفته می شود. فرض کنید میخواهید رشته Sign A " را در یک متغییر رشته ای ذخیره کنید. این رشته شامل کاراکتر " است که برای مشخص کردن انتهای رشته به کار می رود. پس نمیتوان به حالت عادی این رشته را در یک متغییر رشته ای قرار داد. برای اینکه به کامپایلر بگوییم کاراکتر " جزئی از رشته است، باید از کاراکتر \ قبل از آن استفاده کنیم. به همین ترتیب برای این که به کامپایلر بگوییم در رشته "C:\" کاراکتر \ جزئی از رشته است، باید از دو \ به صورت متوالی استفاده کنیم و رشته را به صورت "C:\\" بنویسیم. اگر این عبارت را به صورت "C:\" بنویسیم کامپایلر تصور میکند که شما انتهای رشته را مشخص نکرده اید و خطا ایجاد می کند، زیرا " را در انتهای رشته، به عنوان بخشی از عبارت محسوب می کند.

حلقه های do:

نوع دیگری از حلقه هایی که می توانید در برنامه های خود استفاده کنید، حلقه هایی هستند که تا زمان برقراری یک شرط مشخص اجرا می شوند. یکی از این حلقه ها، حلقه های do هستند.
در این حلقه ها، ابتدا دستورات داخل حلقه اجرا شده، سپس شرط حلقه بررسی می شود. در صورت درست بودن شرط حلقه، دستورات مربوط به حلقه بار دیگر نیز اجرا می شوند. در غیر این صورت، دستورات اجرا نخواهند شد و اجرای برنامه به خط بعد از حلقه منتقل می شود. در امتحان کنید زیر برنامه ای را ایجاد خواهیم کرد که تعدادی عدد تصادفی تولید کند. تولید این اعداد تصادفی به وسیله توابع درونی .NET انجام می شود و تا زمانی که عدد ۱۰ تولید نشده است ادامه پیدا می کند.

امتحان کنید: استفاده از حلقه do

۱) در قسمت طراحی فرم، کنترل Button دیگری به فرم اضافه کنید. خاصیت Name آن را برابر با btnDoLoop و خاصیت Text آن را برابر Do Loop قرار دهید.
۲) روی این کنترل دو بار کلیک کنید و کدهای مشخص شده در زیر را در متد مربوط به رویداد Click دکمه فرمان قرار دهید:

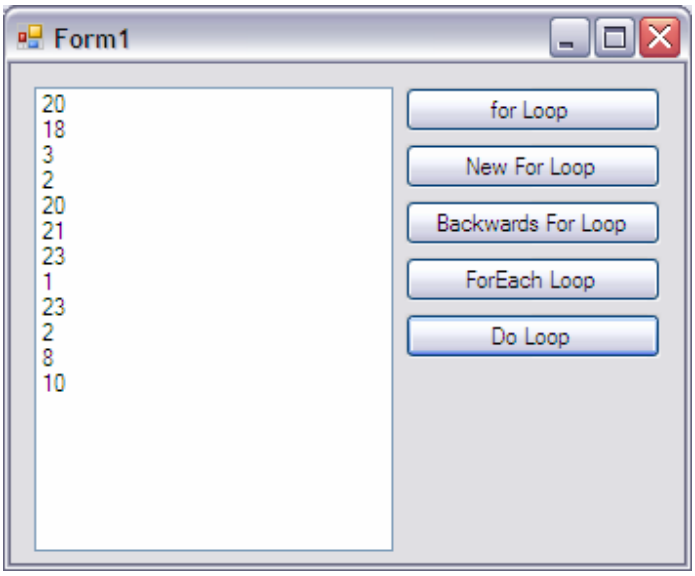
```
private void btnDoLoop_Click(object sender, EventArgs e)
{
     // Declare variable
     Random objRandom = new Random();
     int intRandomNumber = 0;

     // Clear the list
     lstData.Items.Clear();

     // Process the loop until intRandomNumber = 10
     do
     {
          // Get a random number between 0 and 24
          intRandomNumber = objRandom.Next(25);
```

```
            // Add the number to the list
            lstData.Items.Add(intRandomNumber);
        } while (intRandomNumber != 10);
}
```

۳) برنامه را اجرا کنید و روی دکمه Do Loop کلیک کنید. نتیجه ای را مشابه شکل ۴-۲۱ مشاهده خواهید کرد. بار دیگر روی این دکمه کلیک کنید. مشاهده می کنید که اعداد داخل ListBox با اعداد دیگری عوض می شوند و با هر بار کلیک کردن تعدادی عدد جدید در ListBox قرار می گیرد.



شکل ۴-۲۱

چگونه کار می کند؟

یک حلقه do تا زمانی که یک شرط خاص برقرار نشده است اجرا می شود. در این حلقه ها، هیچ متغییر کنترل کننده و یا شمارشگری وجود ندارد، بلکه باید خودتان مکان کنونی حلقه را نگهداری کنید. در این مثال ابتدا یک متغییر (به عبارت دیگر یک شیئ) از کلاس Random ایجاد می کنید. این متغییر توابع مورد نیاز برای تولید عدد تصادفی را در اختیار شما قرار می دهد.

نکته: همانطور که میدانید هنگامی که یک متغییر ایجاد شد، باید مقدار اولیه آن را مشخص کرد و سپس از آن استفاده کرد. برای بعضی از متغییر ها همانند اعداد صحیح، به راحتی می توان مقدار اولیه مشخص کرد. اما بسیاری از متغییر ها که با استفاده از کلاسها ایجاد می شوند را نمی توان همانند اعداد صحیح مقدار اولیه داد. برای مقدار دهی به این اشیا (همانند شیئ objRandom در مثال قبل) از کلمه کلیدی new استفاده می کنیم. در این حالت خود کلاس، برای شیئ یک مقدار اولیه ایجاد کرده و آن را به شیئ نسبت می دهد.

این شیئ با پیشوند obj نامگذاری شده است تا مشخص شود که یک شیئ است که از یک کلاس مشتق شده است. متغییر بعدی که تعریف می کنید، یک عدد صحیح به نام intRandomNumber خواهد بود که وظیفه نگهداری عدد تصادفی تولید شده توسط objRandom را بر عهده دارد:

```
// Declare variable
Random objRandom = new Random();
int intRandomNumber = 0;
```

در مرحله بعد، هر موردی را که قبلا به لیست اضافه شده بود از آن پاک می کنید:

```
// Clear the list
lstData.Items.Clear();
```

سپس حلقه do را به نحوی ایجاد می کنید که تا تولید عدد ۱۰ به اجرای خود ادامه دهد. در این حلقه، ابتدا برای مرتبه اول دستورات اجرا می شوند و سپس شرط بررسی می شود. با هر بار اجرای حلقه عدد تصادفی جدیدی تولید و آن را در متغییر intRandomNumber ذخیره می کنید. برای تولید عدد تصادفی از تابع Next در شیئ objRandom استفاده می کنید که یک پارامتر می گیرد. این پارامتر بزرگترین عددی که این تابع می تواند تولید کند را مشخص می کند. در اینجا عدد ۲۵ به تابع فرستاده شده است و مشخص میکند که تابع باید عدد تصادفی در بازه ۰ تا ۲۴ تولید کند. بعد از تولید عدد تصادفی، آن را به لیست خود اضافه می کنید:

```
// Get a random number between 0 and 24
intRandomNumber = objRandom.Next(25);
// Add the number to the list
lstData.Items.Add(intRandomNumber);
```

در انتها بررسی می کنید که آیا عدد تولید شده برابر با ۱۰ بوده است یا نه؟ اگر عدد تولید شده مخالف ۱۰ باشد، برنامه به ابتدای دستورات حلقه do برمیگردد و بار دیگر حلقه را اجرا می کند. در غیر این صورت از حلقه خارج شده و به اولین خط بعد از حلقه می رود.

نکته: به علت اینکه دستورات در این حلقه قبل از بررسی شرط حتما یک بار اجرا می شوند، نیازی نیست که به متغییر intRandomNumber مقدار اولیه اختصاص دهید. زیرا این متغییر در مرحله اولی که دستورات حلقه اجرا می شوند مقدار می گیرد.

حلقه while:

عملکرد این حلقه نیز همانند حلقه do است با این تفاوت که شرط این حلقه در ابتدا بررسی می شود و سپس، در صورت درست بودن آن دستورات داخل حلقه اجرا می شوند. در بخش بعد کاربرد این حلقه ها را در برنامه خواهیم دید.

امتحان کنید: استفاده از حلقه های while

۱) کنترل Button دیگری به فرم خود اضافه کنید، خاصیت Name آن را برابر btnDoWhileLoop و خاصیت Text آن را برابر Do While Loop قرار دهید.

۲) روی این کنترل دو بار کلیک کنید و کد مشخص شده در زیر را در متد مربوط به رویداد Click آن قرار دهید:

```
private void btnDoWhileLoop_Click(object sender,
                                  EventArgs e)
{
     // Declare variables
     Random objRandom = new Random();
     int intRandomNumber = 0;

     // Clear the list
     lstData.Items.Clear();

     // Process the loop while intRandomNumber < 15
     while (intRandomNumber < 15)
     {
          // Get a random number between 0 and 24
          intRandomNumber = objRandom.Next(25);
          // Add the number to the list
          lstData.Items.Add(intRandomNumber);
     }
}
```

۳) برنامه را اجرا کنید و روی دکمه ی Do While Loop کلیک کنید. نتیجه ای را مشابه شکل ۴-۲۲ مشاهده خواهید کرد.

۴) هر بار که روی دکمه فرمان کلیک کنید اعداد جدیدی درون لیست باکس قرار می گیرند. تولید این اعداد تا زمانی که عددی بزرگتر از ۱۵ تولید شود، ادامه پیدا می کند.

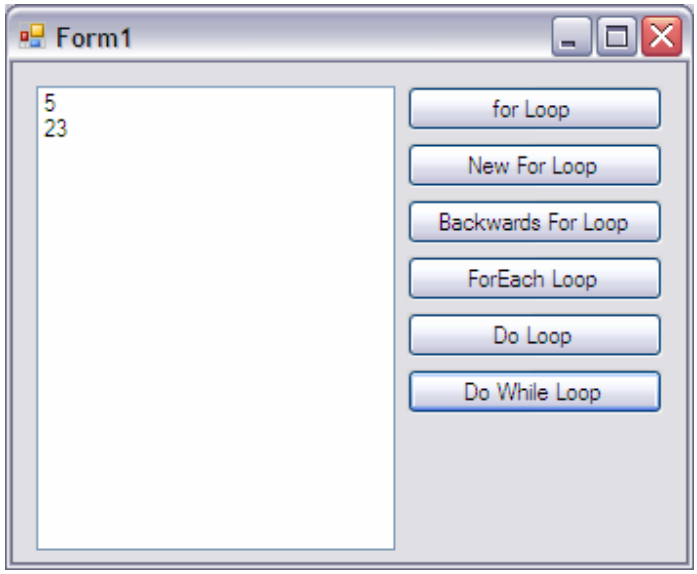
چگونه کار می کند؟

همانطور که گفتیم، عملکرد حلقه while همانند حلقه do است با این تفاوت که در این نوع حلقه، شرط در ابتدا بررسی می شود و در صورت درست بودن آن، دستورات حلقه ادامه پیدا می کند. در غیر این صورت برنامه اولین خط بعد از حلقه را اجرا می کند. در این مثال، هنگامی که حلقه شروع به کار می کند، در ابتدا بررسی می کند که آیا مقدار intRandomNumber از ۱۵ کوچکتر است یا نه؟ در صورتی که شرط درست باشد، دستورات داخل حلقه اجرا می شوند.

```
     // Process the loop while intRandomNumber < 15
     while (intRandomNumber < 15)
     {
```

```
            // Get a random number between 0 and 24
            intRandomNumber = objRandom.Next(25);
            // Add the number to the list
            lstData.Items.Add(intRandomNumber);
        }
```

بعد از اینکه حلقه برای مرتبه اول اجرا شد، عدد تصادفی جدید در متغییر `intRandomNumber` قرار می گیرد و شرط حلقه مجددا بررسی می شود. اگر عدد تولید شده از ۱۵ کوچکتر باشد دستورات حلقه دوباره اجرا می شوند. در غیر ایـن صـورت، برنامـه بـه اولین خط بعد از حلقه می شود و دستورات آن را اجرا میکند.



شکل ۴-۲۲

شرطهای قابل قبول برای حلقه های `do` و `while`:

ممکن است از عبارتهای شرطی که می توانید در این حلقه ها استفاده کنید تعجب کنید. ولی هر عبارتی را که بتوانید در دسـتور `if` استفاده کنید، می توانید آن را به عنوان شرط حلقه `do` و `while` نیز به کار ببرید. برای مثال به شرط های زیر توجه کنید:

```
while (intX > 10 && intX < 100)
```

یا:

```
do
{

} while ((intX > 10 && intX < 100) || intY == true);
```

یا:

```
while (String.Compare(strA, strB) > 0)
```

خلاصه، این حلقه ها قدرت و انعطاف پذیری بسیار زیادی دارند.

حلقه های تودرتو:

در بعضی از شرایط ممکن است نیاز داشته باشید در حین این که درون یک حلقه هستید، حلقه جدیدی را شروع کنید. به این نوع حلقه ها، حلقه های تودرتو می گویند و در اصل همانند دستورات `if` تودرتو هستند. در امتحان کنید بعد، مشاهده خواهید کرد که چگونه می توانید از این نوع حلقه ها استفاده کنید.

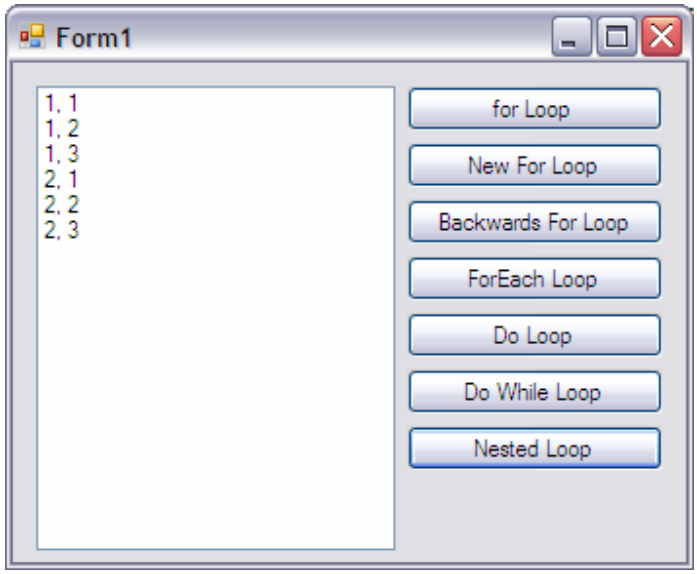
امتحان کنید: استفاده از حلقه های تودرتو

۱) در قسمت طراحی فرم، کنترل `Button` دیگری به فرم اضافه کنید، خاصیت `Name` آن را برابر با `btnNestedLoop` و خاصیت `Text` آن را برابر با `Nested Loop` قرار دهید.

۲) بر روی این کنترل دو بار کلیک کنید و کد مشخص شده در زیر را در متد مربوط به رویداد `Click` وارد کنید.

```
private void btnNestedLoops_Click(object sender,
                                  EventArgs e)
{
     // Process an outer loop
     for (int intLoop1 = 1; intLoop1 <= 2; intLoop1++)
     {
          // Process a nested (inner) loop
          for (int intLoop2 = 1; intLoop2 <= 3; intLoop2++)
          {
               lstData.Items.Add(intLoop1 + ", "
                                 + intLoop2);
          }
     }
}
```

۳) برنامه را اجرا کنید و روی دکمه `Nested Loop` کلیک کنید. نتیجه ای مشابه شکل ۴-۲۳ را مشاهده خواهید کرد.



شکل ۴-۲۳

چگونه کار می کند؟

کد این برنامه کاملا مشخص است و جای ابهامی ندارد. حلقه اول (حلقه بیرونی) با استفاده از شمارشگر `intLoop1` از عـدد ۱ تا ۲ و حلقه دوم یا حلقه درونی با استفاده از شمارشگر `intLoop2` از ۱ تا ۳ حرکت می کند. در حلقه درونی کدی برای نمایش مقدار متغییر های `intLoop1` و `intLoop2` وجود دارد:

```
// Process an outer loop
for (int intLoop1 = 1; intLoop1 <= 2; intLoop1++)
{
     // Process a nested (inner) loop
     for (int intLoop2 = 1; intLoop2 <= 3; intLoop2++)
     {
          lstData.Items.Add(intLoop1 + ", "
                            + intLoop2);
     }
}
```

در حلقه های تو در تو باید توجه داشته باشید، حلقه ای که در درون یک حلقه دیگر شروع می شود، در همان حلقه نیز تمام می شود. به عبارت دیگر کدهای مربوط به حلقه درونی نمی توانند در خارج از حلقه بیرونی قرار بگیرند. در اینجا اولین آکولادی که بسته شـده است مربوط به حلقه داخلی است. هنگامی که برنامه برای اولین بار وارد حلقه `intLoop1` می شود، حلقـه `intLoop2` را سه بار اجرا می کند. سپس یک واحد به شمارنده حلقه `intLoop1` اضافه می کند و بار دیگر سـه بـار دسـتورات داخـل حلقـه `intLoop2` را اجرا می کند. این تکرار ادامه پیدا می کند تا دفعات تکرار حلقه اول به پایان برسد. به این ترتیب برنامه از هر دو حلقه خارج می شود و به اولین خط بعد از آنها می رود.

خروج زودهنگام از حلقه:

بعضی مواقع در برنامه نیازی نیست که یک حلقه `for` تا انتها اجرا شود. مثلا در یک لیست به دنبال موردی خاص میگردید و میخواهید با پیدا شدن آن مورد از حلقه خارج شوید، زیرا گشتن بقیه عناصر لیست لازم نیست.
در امتحان کنید بعد، برنامه ای که تمام فولدرهای درایو C را نمایش میداد به نحوی تغییر میدهید که با رسیدن به فولدر `C:\Program Files` پیغامی را نشان دهد و از برنامه خارج شود.

امتحان کنید: خروج زودهنگام از حلقه

۱) کنترل `Button` جدیدی به فرم اضافه کنید، خاصیت `Name` آن را برابر `btnQuittingAForLoop` و خاصیت `Text` آن را برابر `Quitting a For Loop` قرار دهید.
۲) روی این کنترل دو بار کلیک کنید و کد مشخص شده در زیر را در متد مربوط به رویداد `Click` آن قرار دهید:

```
private void btnQuittingAForLoop_Click(object sender,
                                       EventArgs e)
{
     // List each folder at the root of your C Drive
     foreach (string strFolder in
               System.IO.Directory.GetDirectories("C:\\"))
     {
          // Add the item to the list
          lstData.Items.Add(strFolder);

          // Do you have the folder C:\Program Files?
          if(String.Compare(strFolder,
                            "c:\\program files",true) == 0)
          {
               // Tell the user
               MessageBox.Show("Found it, exiting the loop"
                               + " now.","Loops");

               // Quit the loop early
               break;
          }
     }
}
```

۳) برنامه را اجرا کنید و روی دکمه ی `Quitting A For Loop` کلیک کنید. نتیجه ای مشابه شکل ۴-۲۴ را مشاهده خواهید کرد.



شکل ۴-۲۴

چگونه کار می کند؟

هر بار که حلقه اجرا می شود، با استفاده از تابع `String.Compare` که نحوه کار آن قبلا توضیح داده شد، بررسی میکنید که آیا نام فولدر برابر با `C:\Program Files` است یا نه؟

```
// Do you have the folder C:\Program Files?
if(String.Compare(strFolder,"c:\\program files",true) == 0)
```

در صورتی که نام فولدر برابر با `"C:\Program Files"` بود ابتدا یک کادر پیغام را به کاربر نمایش میدهید.

```
// Tell the user
MessageBox.Show("Found it, exiting the loop now.","Loops");
```

بعد از آن با استفاده از دستور `break` از حلقه خارج می شوید. در این حالت هنگامی که ویژوال C# به این دستور برسد، به اولین خط بعد از حلقه می رود و آن را اجرا می کند. در این مثال از دستور `break` برای خروج از حلقه `for` استفاده کردیم اما از این دستور برای خروج زودهنگام از هر نوع حلقه ای در C# میتوانید استفاده کنید.

```
    // Quit the loop early
    break;
```

البته اگر نام فولدر برابر با نامی که به دنبال آن هستید نباشد، جستجو تا انتهای لیست ادامه پیدا میکند. در برنامه ها برای جستجوی یک مورد خاص معمولا از حلقه های `for` استفاده می کنند. استفاده از دستور `break` برای خروج از حلقه هنگام پیدا شدن آیتم مورد نظر باعث افزایش سرعت برنامه می شود.
تصور کنید می خواهید لیستی شامل ۱۰۰۰ آیتم را جستجو کنید و آیتم مورد نظر شما در مکان دهم قرار دارد. اگر بعد از پیدا کردن آن از حلقه خارج نشوید، از کامپیوتر می خواهید که ۹۹۰ آیتم را بی دلیل جستجو کند که باعث می شود سرعت برنامه به شدت کم شود.

نکته: به نحوه نوشتن آدرسها در عبارتهای رشته ای این برنامه دقت کنید. کاراکتر \ در تمام آنها به صورت \\ نوشته شده است. همانطور که در بخش قبلی شرح داده شد، برای قرار دادن کاراکتر \ در رشته باید از \\ استفاده کنید. برای مثال اگر در برنامه بالا به جای استفاده از رشته `"c:\\program files"` از `"c:\program files"` استفاده کنید با خطای `"Unrecognized scape sequence"` روبرو خواهید شد، زیرا بعد از کاراکتر \ در یک رشته، باید یک کاراکتر کنترلی با معنی برای کامپایلر قرار بگیرد و کاراکتر p جز این کاراکترها نیست.

نکته: به کد زیر برای تعریف یک ثابت رشته ای دقت کنید:

```
string FileName = @"C:\Test\Temp.txt";
```

در این کد بر خلاف قسمتهای قبلی از دو کاراکتر \ استفاده نکرده ایم. بلکه فقط یک \ به کار برده ایم و قبل از رشته هم از علامت @ استفاده کرده ایم. در ویژوال C# اگر قبل از شروع یک رشته از علامت @ استفاده شود، تمام کاراکترهای کنترلی در رشته به صورت کاراکترهای عادی در نظر گرفته می شوند و رشته تا علامت نقل قول بعدی ادامه پیدا می کند. در این رشته هم قبل از شروع از علامت @ استفاده کرده ایم. بنابراین تمام کاراکتر ها (حتی \) تا کاراکتر نقل قول پایانی جزئی از رشته به شمار می روند.

دستور `continue`:

در بعضی از مواقع ممکن است بخواهید دستورات حلقه در شرایطی خاص اجرا نشوند. فرض کنید میخواهید اعداد ۱ تا ۱۵ را که مضربی از ۳ نیستند در `ListBox` قرار دهید. برای این کار باید از یک حلقه `for` استفاده کنید که شمارنده آن از یک تا پانزده، یک واحد یک واحد افزایش یابد، اما در صورتی که عدد بر سه بخش پذیر بود نباید آن را در `ListBox` قرار دهید و به عدد بعدی بروید.
در امتحان کنید زیر نحوه نوشتن چنین برنامه ای را مشاهده خواهیم کرد:

امتحان کنید: دستور `continue`

۱) کنترل `Button` دیگری به فرم اضافه کنید، خاصیت `Name` آن را برابر با `btnContinue` و خاصیت `Text` آن را برابر با `Continue` قرار دهید.
۲) روی این کنترل دو بار کلیک کنید و کد مشخص شده در زیر را در متد مربوط به رویداد `Click` آن وارد کنید:

```
private void btnContinue_Click(object sender, EventArgs e)
{
     // Perform a loop
     for (int intCount = 1; intCount <= 15; intCount++)
     {
          // If the item is dividable by 3,
          // go to next number
          if ((intCount % 3) == 0)
               continue;

          // Add the item to the list
          lstData.Items.Add(intCount);
     }
}
```

۳) برنامه را اجرا کنید و روی دکمه ی Continue کلیک کنید. مشاهده می کنید کـه تمـام اعـداد از ۱ تـا ۱۵ بـه جـز مضارب سه در لیست باکس قرار گرفته اند.

چگونه کار می کند؟

دستور continue باعث خروج کامل از حلقه نمی شود، بلکه در هر قسمتی که استفاده شود باعث می شود بقیه دستورات حلقه که بعد از این دستور قرار گرفته اند در نظر گرفته نشوند و حلقه مجددا اجرا شود. در این مثال بـرای اعـداد ۱ و ۲، حلقـه بـه صـورت عادی کار می کند. با رسیدن به عدد سه، بـه علـت اینکـه باقیمانـده ایـن عـدد بـر سـه برابـر بـا صـفر اسـت، کامپـایلر بـه دسـتور continue می رسد. هنگامی که ویژوال C# به این دستور رسید، بقیه دستورات حلقه را اجرا نمی کند، بلکه شمارنده حلقـه را یک واحد افزایش میدهد و حلقه را برای عدد ۴ اجرا می کند. این مرتبه نیز همانند اعداد قبل از ۳، چون عدد ۴ مـضربی از ۳ نیـست پس باقیمانده برابر با صفر نمی شود و عبارت داخل if درست نیست. پس دستور continue اجرا نمی شود و بقیه دسـتورات حلقه اجرا می شوند.

نکته: عملگر % برای محاسبه باقیمانده تقسیم دو عدد به کار میرود. در این مثال استفاده از این عملگر موجب می شود در هر مرحله باقیمانده تقسیم intCount بر سه محاسبه شود و در شرط if بررسی شود.

حلقه های بی نهایت:

هنگام ایجاد حلقه ها، میتوانید حلقه هایی ایجاد کنید که بینهایت بار اجرا شوند. به عبارت دیگر هنگامی که شروع شوند، هـیچ وقـت تمام نشوند. به حلقه ایجاد شده در کد زیر نگاه کنید:

```
for (int intCount = 1; intCount > 0; intCount++)
{
```

```
            lstData.Items.Add(intCount);
        }
```

با شروع حلقه مقدار شمارنده برابر با عدد ۱ است و چون از صفر بزرگتر است حلقه برای بار اول اجرا می شود. بعد از اجرای دستورات حلقه، شمارنده تغییر می کند و برابر با دو می شود. در این مرحله نیز به علت اینکه عدد ۲ از صفر بزرگتر است، حلقه مجددا اجرا می شود. همانطور که مشاهده می کنید، شمارنده ی این حلقه هیچگاه به عددی کوچکتر از صفر نمی رسد، پس شرط حلقه همواره درست است. این حلقه بینهایت بار اجرا می شود. ممکن است برنامه توقف نکند اما به کلیک های بعدی پاسخی نخواهد داد و دائما دستورات حلقه را اجرا می کند.

اگر حس کردید که یک برنامه درون یک حلقه بی نهایت قرار گرفته است، بایستی آن را ببندید. اگر برنامه را در محیط ویژوال استودیو اجرا کرده اید، به محیط ویژوال استودیو برگردید و از منوی Debug گزینه Stop Debugging را انتخاب کنید. این گزینه اجرای برنامه را فورا متوقف می کند. اگر برنامه کامپایل شده را در محیط ویندوز اجرا کرده اید و درون یک حلقه بینهایت قرار گرفته است، برای بستن آن باید از Task Manager استفاده کنید. کلیدهای Ctrl + Alt + Del را فشار دهید، برنامه Task Manager باز خواهد شد. برنامه ای که در حلقه ی بینهایت قرار دارد به عنوان Not Responding نمایش داده می شود. برنامه را انتخاب کنید و بر روی End Task کلیک کنید. پنجره ای باز خواهد شد و می گوید که برنامه مورد نظر پاسخی نمی دهد، آیا می خواهید آن را به اجبار ببندید. در این قسمت نیز بر روی End Task کلیک کنید.

در بعضی از مواقع ممکن است حلقه آنقدر حافظه سیستم را مصرف کند که نتوانید پنجره Task Manager را باز کنید و یا به محیط ویژوال استودیو برگردید. در این مواقع می توانید مقداری صبر کنید و مجددا این روشها را امتحان کنید و یا کامپیوتر خود را مجددا راه اندازی کنید.

هر بار که در محیط ویژوال استودیو برنامه را اجرا می کنید، ابتدا برنامه توسط ویژوال استودیو در دیسک ذخیره می شود و سپس اجرا می شود، تا اگر به علتی مجبور به راه اندازی مجدد کامپیوتر شدید کدهای خود را از دست ندهید.

نتیجه:

در این فصل روشهای گوناگونی که میتوان به وسیله آنها در یک برنامه تصمیم گیری کرده و یا قسمتی از کد را چند بار اجرا کرد را بررسی کردیم. ابتدا عملگر هایی را که میتوانیم در دستور if استفاده کنیم بررسی کردیم، سپس دیدیم که چگونه میتوان با استفاده از عملگر || (OR) و یا && (AND) میتوان چند عملگر را ترکیب کرد. همچنین چگونگی مقایسه رشته ها بدون در نظر گرفتن نوع حروف را نیز مشاهده کردیم.

سپس با دستور switch آشنا شدیم و دیدیم که چگونه می توان یک حالت خاص را از بین تعداد زیادی حالت انتخاب کرد. با مفهوم کلی حلقه ها در برنامه نویسی آشنا شدیم و سه حلقه for، do و while را بررسی کردیم. حلقه های for برای تکرار یک سری از دستورات به تعداد معین به کار می روند و حلقه foreach نیز که از این حلقه مشتق می شود، برای حرکت در بین عناصر یک مجموعه بدون دانستن تعداد آن استفاده می شوند. حلقه های while تا هنگامی که یک شرط خاص برقرار است اجرا می شوند. حلقه های do نیز همانند حلقه های while هستند با این تفاوت که شرط حلقه های while در ابتدای حلقه بررسی می شود ولی شرط حلقه های do در انتهای آن.

بعد از پایان این فصل باید نحوه استفاده از موارد زیر را در برنامه بدانید:

- دستورات `if`، `else if` و `else` برای تست حالتهای مختلف.
- دستورات `if` تودرتو.
- عملگر های مقایسه ای و تابع `Sring.Compare`.
- دستور `switch` برای انتخاب یک حالت بین حالتهای موجود.
- حلقه های `for` و `foreach`
- حلقه های `do`
- حلقه های `while`

تمرین:

تمرین ۱:

یک برنامه تحت ویندوز با یک `TextBox` و یک `Button` ایجاد کنید. در رویداد `Click` مربوط به `Button`، عدد داخل `TextBox` را بدست آورید و با استفاده از دستور `switch` اگر عدد یکی از اعداد یک تا پنج بود آن را به وسیله یک کادر پیغام در خروجی نمایش دهید. در صورتی که عدد وارد شده در `TextBox` در بازه یک تا پنج نبود، پیغام مناسبی را به کاربر نمایش دهید.

تمرین ۲:

یک برنامه تحت ویندوز که شامل یک کنترل `ListBox` و یک کنترل `Button` باشد ایجاد کنید. در رویداد `Click` مربوط به `Button`، یک حلقه `for` ایجاد کنید که اعداد ۱ تا ۱۰ را در `ListBox` قرار دهد. سپس حلقه دیگری ایجاد کنید که اعداد ۱۰ تا ۱ را در `ListBox` نمایش دهد.

فصل پنجم: کارکردن با ساختارهای داده ای

در فصلهای قبلی نحوه استفاده از متغیرهای ساده ای مانند متغیرهای integer برای اعداد صحیح و یا string برای رشته ها را مشاهده کردید. با وجود اینکه این نوع های داده ای بسیار پر کاربرد هستند، اما برنامه های پیچیده تر نیاز دارند که با ساختارهای داده ای کار کنند. **ساختارهای داده ای** به یک گروه از اطلاعات مرتبط به هم گفته می شود که در یک واحد مجزا قرار می گیرند.در این فصل با انواع ساختارهای داده ای موجود در C# و نحوه استفاده از آنها برای نگهداری مجموعه های داده ای پیچیده آشنا خواهید شد.
در این فصل مطالب زیر را خواهید آموخت:

- آرایه ها
- شمارنده ها
- ثابت ها
- ساختارها

مفهوم آرایه:

یکی از عمومی ترین نیازها در برنامه نویسی قابلیت نگهداری لیستی از اطلاعات مشابه و یا مرتبط به هم است. برای این مورد بایستی از آرایه ها استفاده کنید. **آرایه** ها، لیستی از متغیر ها می باشند که همه از یک نوع هستند. برای مثال ممکن است بخواهید سن تمامی دوستان خود را در یک آرایه از اعداد صحیح و یا نام تمامی آنها را در یک آرایه از رشته ها قرار دهید.
در این بخش با نحوه ایجاد، پر کردن و استفاده از آرایه ها در برنامه آشنا خواهید شد.

تعریف و استفاده از آرایه ها:

هنگامی که در برنامه یک آرایه تعریف می کنید، در حقیقت متغیری ایجاد می کنید که بیش از یک عنصر را بتواند در خود نگهداری کند. برای مثال اگر یک متغیر رشته ای را به صورت زیر تعریف کنید فقط یک رشته را میتوانید در آن نگهداری کنید:

```
string strName;
```

اما آرایه ها باعث ایجاد نوعی حالت افزایشی در متغیر می شوند، به این ترتیب بیش از یک مقدار را می توانید در یک متغیر ذخیره کنید. اگر بخواهید یک متغیر را به صورت آرایه تعریف کنید باید در مقابل نوع متغیر از علامت [] استفاده کنید. برای مثال کد زیر متغیری ایجاد می کند که بتواند ۱۰ عنصر را در خود نگهداری کند:

```
string[] strName = new string[10];
```

هنگامی که یک آرایه را ایجاد کردید، می توانید به تک تک عناصر آن با استفاده از اندیس آن عنصر دسترسی پیدا کنید. اندیس عناصر یک آرایه همواره عددی بین صفر و شماره ی آخرین عنصر آرایه است. شماره ی آخرین عنصر هر آرایه هم برابر با یک واحد کمتر از طول آرایه است. بنابراین اندیس هر آرایه عددی بین صفر تا یک واحد کمتر از آخرین عنصر آرایه است.
برای مثال اگر بخواهید مقدار عنصر سوم آرایه قبلی را برابر با رشته ی `"Katie"` قرار دهید باید از کد زیر استفاده کنید:

```
strName[2] = "Katie";
```

برای دسترسی به مقدار یک عنصر از آرایه هم میتوانید از همین روش استفاده کنید:

```
MessageBox.Show(strName[2]);
```

نکته ی مهم در اینجا این است که اگر یکی از عناصر آرایه را تغییر دهید، مقدار بقیه عناصر تغیری نخواهد کرد. برای مثال اگر کد زیر را در برنامه به کار ببرید:

```
strName[3] = "Betty";
```

مقدار `strName[2]` همچنان برابر با `"Katie"` خواهد ماند.
احتمالا بهترین روش برای فهمیدن این که آرایه ها چیستند و چگونه کار می کنند نوشتن برنامه ای با استفاده از آنها است.

امتحان کنید: تعریف و استفاده از یک آرایه ساده

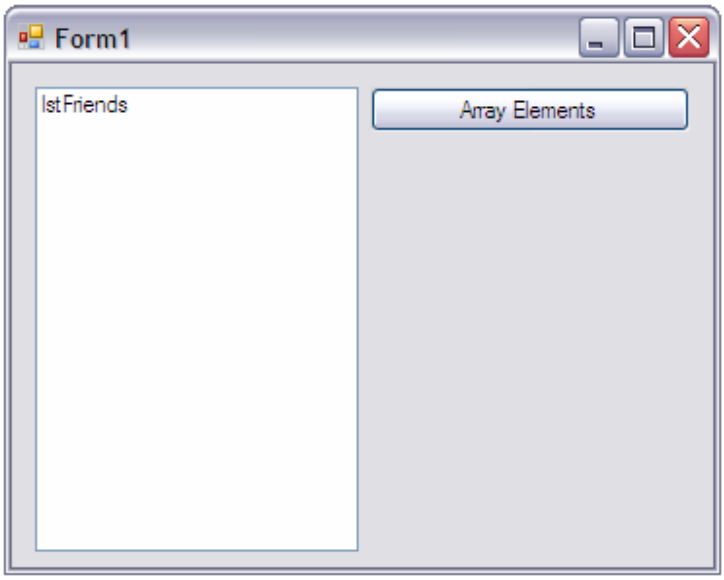
۱) با استفاده از ویژوال استودیو ۲۰۰۵ روی منوی `File` کلیک کرده و سپس `New` ➔ `Project` را انتخاب کنید. در پنجره `New Project` یک پروژه ویندوزی به نام `Array Demo` ایجاد کنید.
۲) هنگامی که قسمت طراحی `Form1` نمایش داده شد، یک کنترل `ListBox` به فرم اضافه کنید. خاصیت `Name` این کنترل را برابر با `lstFriends` و خاصیت `IntegralHeight` آن را برابر با `False` قرار دهید.
۳) اکنون یک کنترل `Button` به فرم اضافه کرده، خاصیت `Name` آن را برابر با `btnArrayElements` و خاصیت `Text` آن را برابر با `Array Elements` قرار دهید. فرم شما باید مشابه شکل ۵-۱ شده باشد.
۴) روی کنترل `Button` دو بار کلیک کنید و کد زیر را در متد مربوط به رویداد `Click` آن وارد کنید:

```
private void btnArrayElements_Click(object sender,
                                    EventArgs e)
{
    // Declare an array
    string[] strFriends = new string[5];

    // Populate the array
    strFriends[0] = "Robbin";
    strFriends[1] = "Bryan";
    strFriends[2] = "Stephanie";
    strFriends[3] = "Sydney";
```

```
        strFriends[4] = "Katie";

        // Add the first array item to the list
        lstFriends.Items.Add(strFriends[0]);
}
```



شکل ۵-۱

۵) برنامه را اجرا کنید و روی دکمه ی `Array Elements` کلیک کنید. کنترل `ListBox` روی فرم با نام `Robbin` پر می شود.

چگونه کار می کند؟

هنگامی که می خواهید یک آرایه را تعریف کنید، ابتدا باید نوع داده ای و اندازه آن را مشخص کنید. در این برنامه می خواهیم یک آرایه از نوع رشته ای و به طول ۵ عنصر ایجاد کنیم. پس از کد زیر استفاده می کنیم:

```
        // Declare an array
        string[] strFriends = new string[5];
```

به این ترتیب آرایه ای به طول ۵ ایجاد کرده اید. به عبارت دیگر میتوانیم بگوییم در این برنامه آرایه ای داریم که شامل ۵ عنصر است.

بعد از ایجاد آرایه، یک آرایه با پنج عنصر خواهید داشت که میتوانید به هر یک از عناصر آن با استفاده از اندیس آن دسترسی پیدا کنید.برای دسترسی به یک عنصر خاص باید اندیس آن عنصر را در داخل کروشه بعد از نام آرایه بیاورید. اندیس ها از شماره صفر شروع می شوند و تا یک واحد کمتر از طول آرایه ادامه پیدا می کنند. بنابراین عنصر اول آرایه دارای اندیس صفر، عنصر دوم آرایه

دارای اندیس یک و ... است. به همین دلیل است که در برنامه آرایه را به طول ۵ تعریف کرده، ولی برای پر کردن آن از اعداد ۰ تا ۴ استفاده کرده ایم.

```
// Populate the array
strFriends[0] = "Robbin";
strFriends[1] = "Bryan";
strFriends[2] = "Stephanie";
strFriends[3] = "Sydney";
strFriends[4] = "Katie";
```

همانطور که برای مقدار دهی به یک عنصر خاص باید از اندیس آن استفاده کنید، برای دسترسی به مقدار آن عنصر از آرایه نیز می توانید از اندیس آن استفاده کنید. در این برنامه، مقدار موجود در خانه صفرم آرایه که برابر با اولین مقدار آرایه (`"Robbin"`) است را به کاربر نمایش می دهید:

```
// Add the first array item to the list
lstFriends.Items.Add(strFriends[0]);
```

علت این که اندیس عناصر موجود در آرایه و طول آن آرایه مقداری گیج کننده به نظر می رسد این است که اندیسها در یک آرایه از صفر شروع می شوند، اما معمولا انسانها اولین عنصر هر مجموعه ای را با یک نمایش می دهند. هنگامی که می خواهید مقداری را در آرایه قرار دهید و یا مقدار یک عنصر از آرایه را بدست آورید، باید یک واحد از مکان عنصر مورد نظرتان کم کنید تا اندیس آن را بدست آورید. برای مثال عنصر پنجم در آرایه برابر با اندیس ۴ و اولین عنصر در آرایه برابر با اندیس ۰ است.

نکته: سوالی که ممکن است در این قسمت ایجاد شود این است که چرا اندیسها از صفر شروع میشوند؟ به خاطر دارید که برای کامپیوتر، یک متغیر آدرس مکانی از حافظه کامپیوتر است. هنگامی که برای دسترسی به یک عنصر از آرایه اندیس آن را مشخص می کنید، ویژوال C# این اندیس را در اندازه ی یکی از عناصر آرایه ضرب می کند[1] و حاصل را با آدرس آرایه جمع می کند تا به آدرس عنصر مشخص شده برسد. آدرس شروع یک آرایه هم در حقیقت آدرس اولین عنصر آن آرایه است. بنابراین اولین عنصر آرایه به اندازه صفر ضربدر اندازه عنصر، از نقطه شروع آرایه فاصله دارد، دومین عنصر به اندازه یک ضربدر اندازه عنصر از شروع آرایه فاصله دارد و به همین ترتیب ادامه پیدا می کند.

استفاده از foreach:

یکی از عمومی ترین روشهای استفاده از آرایه ها، به وسیله حلقه های `foreach` است.این نوع حلقه ها در فصل ۴، هنگامی که با آرایه رشته ای برگردانده شده توسط تابع `GetDirectories` کار می کردید، معرفی شدند. در بخش امتحان کنید بعد، ملاحظه خواهید کرد که چگونه می توانیم از این حلقه ها در آرایه ها استفاده کنیم.

[1] این مورد که اندازه کدام عنصر را در نظر بگیرد اهمیتی ندارد، زیرا همه عناصر یک آرایه از یک نوع هستند و اندازه آنها با هم برابر است. برای مثال اندازه هر یک از عناصر یک آرایه از نوع `int`، برابر با ۴ بایت است.

امتحان کنید: استفاده از حلقه های foreach با آرایه ها

۱) اگر برنامه ی Arrays Demo در حال اجرا است آن را ببندید. قسمت ویرایشگر کد را برای Form1 باز کنید و کد زیر را در بالاترین قسمت در بدنه کلاس خود وارد کنید:

```
public partial class Form1 : Form
{
     // Declare a form level array
     private string[] strFriends = new string[5];
```

۲) به قسمت طراحی فرم مربوط به Form1 برگردید و بر روی قسمت خالی فرم دو بار کلیک کنید. متد مربوط به رویداد Load فرم به صورت اتوماتیک ایجاد می شود. حال کد زیر را در این متد وارد کنید:

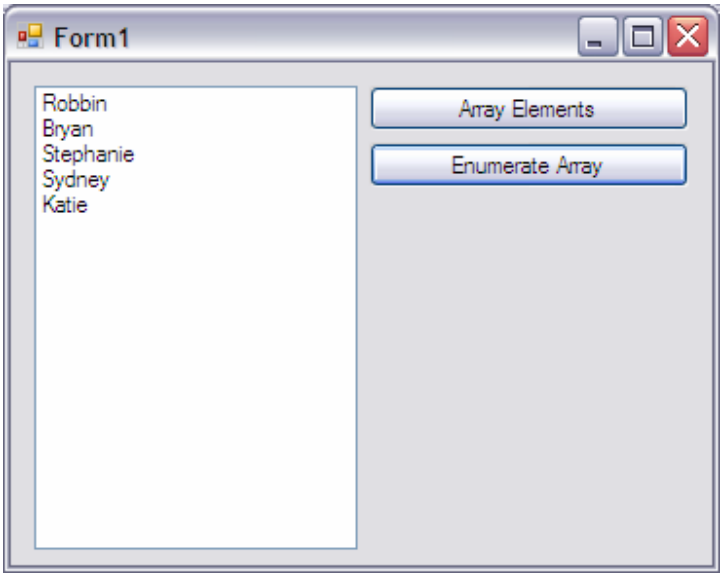
```
private void Form1_Load(object sender, EventArgs e)
{
     // Populate the array
     strFriends[0] = "Robbin";
     strFriends[1] = "Bryan";
     strFriends[2] = "Stephanie";
     strFriends[3] = "Sydney";
     strFriends[4] = "Katie";
}
```

۳) مجددا به قسمت طراحی فرم برگردید و یک کنترل Button به فرم اضافه کنید. خاصیت Name این کنترل را برابر با btnEnumerateArray و خاصیت Text آن را برابر با Enumerate Array قرار دهید.

۴) روی این کنترل دو بار کلیک کنید و کد زیر را در متد مربوط به رویداد Click آن وارد کنید:

```
private void btnEnumerateArray_Click(object sender,
                                     EventArgs e)
{
     // Enumerate the array
     foreach (string strName in strFriends)
     {
          // Add the array item to the list
          lstFriends.Items.Add(strName);
     }
}
```

۵) برنامه را اجرا کرده و بر روی دکمه ی Enumerate Array کلیک کنید. نتیجه ای را مشابه شکل ۵-۲ مشاهده خواهید کرد.



شکل ۵-۲

چگونه کار می کند؟

این تمرین را با تعریف یک متغیر که در تمام فرم قابل استفاده است شروع می کنیم. به عبارت دیگر این آرایه بـرای تمـام متـدهای موجود در کلاس `Form1` قابل استفاده است. هر گاه متغیری در خارج از متدها درون یک کلاس تعریف شود، در تمامی متـدهای آن کلاس قابل استفاده خواهد بود.

```
// Declare a form level array
private string[] strFriends = new string[5];
```

سپس متدی را برای رویداد `Load` مربوط به `Form1` ایجاد می کنید و کد مربوط به پر کردن آرایه را در آن قرار می دهید. این متد هنگامی اجرا می شود که این فرم از برنامه بخواهد در حافظه بار گذاری شده و نمایش داده شود. به این ترتیب مطمئن می شوید هنگامی که فرم نمایش داده می شود آرایه شما پر شده است.

```
private void Form1_Load(object sender, EventArgs e)
{
    // Populate the array
    strFriends[0] = "Robbin";
    strFriends[1] = "Bryan";
    strFriends[2] = "Stephanie";
    strFriends[3] = "Sydney";
    strFriends[4] = "Katie";
}
```

در فصل چهار مشاهده کردید که چگونه یک حلقه foreach در بین عناصر یک آرایه از رشته ها حرکت می کند. در این مثال هم همان مراحل را تکرار می کنیم. یک متغیر کنترل کننده، هم نوع با عناصر آرایه تعریف می کنیم و آن را برای دسترسی به تک تک عناصر آرایه مورد استفاده قرار می دهیم.
حلقه foreach از عنصر صفرم آرایه شروع می کند و تا رسیدن به آخرین عنصر در آرایه، بین تمام عناصر جا به جا می شود. در هر بار تکرار حلقه می توانید از عنصری که در متغیر کنترل کننده قرار گرفته است استفاده کنید. در این مثال این مقدار را به لیست اضافه می کنیم.

```
// Enumerate the array
foreach (string strName in strFriends)
{
     // Add the array item to the list
     lstFriends.Items.Add(strName);
}
```

همچنین توجه کنید که عناصر به همان ترتیبی که در آرایه وارد شده اند داخل لیست قرار می گیرند. این مورد به این دلیل است که حلقه foreach از عنصر صفرم تا آخرین عنصر آرایه را به همان ترتیبی که تعریف شده اند طی می کند.

انتقال آرایه ها به عنوان پارامتر:

در بسیاری از موارد ممکن است نیاز داشته باشید یک آرایه را که محتوی چندین عنصر است به عنوان پارامتر به یک متد بفرستید. در بخش امتحان کنید بعد، نحوه انجام این عمل را خواهیم دید.

امتحان کنید: انتقال آرایه ها به عنوان پارامتر

۱) به قسمت طراحی فرم برگردید و کنترل Button دیگری را به Form1 اضافه کنید. خاصیت Name این کنترل را برابر با btnArraysAsParameters و خاصیت Text آن را برابر با Arrays as Parameters قرار دهید.

۲) روی این کنترل دو بار کلیک کنید و کد زیر را در متد مربوط به رویداد Click آن وارد کنید. پیغامی را مشاهده خواهید کرد که می گوید زیربرنامه AddItemsToList تعریف نشده است. با کلیک بر روی این پیغام، ویژوال استودیو متد را به صورت اتوماتیک ایجاد می کند. همچنین می توانید این پیغام را نادیده بگیرید، زیرا این متد را در مرحله بعدی تعریف خواهیم کرد:

```
private void btnArraysAsParameters_Click(object sender,
                                         EventArgs e)
{
     // List your friends
     AddItemsToList(strFriends);
}
```

۳) هم اکنون متد `AddItemsToList` را به صورت زیر در کلاس خود تعریف کنید:

```
private void AddItemsToList(string[] arrayList)
{
     // Enumerate the array
     foreach (string strName in arrayList)
     {
          // Add the array item to the list
          lstFriends.Items.Add(strName);
     }
}
```

۴) برنامه را اجرا کرده و روی دکمه ی `Arrays as Parameters` کلیک کنید. نتیجه ای را مشابه شکل ۵-۲ مشاهده خواهید کرد.

چگونه کار می کند؟

نکته ای که در متد `AddItemsToList` وجود دارد این است که پارامتر مورد نیاز این متد، آرایه ای از نوع رشته است. برای این که پارامتر یک متد را از نوع آرایه تعریف کنید، باید در مقابل نوع داده ای آن، از یک کروشه خالی (`[]`) استفاده کنید:

```
private void AddItemsToList(string[] arrayList)
```

هنگامی که در تعریف پارامترهای یک متد، آرایه ای را تعریف می کنید اما طول آن را مشخص نمی کنید، در حقیقت به کامپایلر ویژوال C# می گویید که هر آرایه ای از این نوع می تواند به این متد فرستاده شود. به عبارت دیگر اندازه آرایه هایی که به این متد فرستاده می شوند مهم نیست و فقط نوع آرایه مهم است. در زیربرنامه `btnArraysAsParameters` می توانید آرایه اصلی خود را به عنوان پارامتر به این تابع منتقل کنید:

```
// List your friends
AddItemsToList(strFriends);
```

در بخش امتحان کنید بعد، آرایه ای با طول متفاوتی را تعریف می کنیم و برای اضافه کردن آن به لیست از متد قبلی استفاده می کنیم.

امتحان کنید: اضافه کردن دوستان بیشتر

۱) اگر برنامه در حال اجرا است آن را ببندید و به قسمت طراحی فرم مربوط به Form1 برگردید. کنترل Button دیگری به فرم اضافه کرده، خاصیت Name آن را برابر با btnMoreArrayParameters و خاصیت Text آن را برابر با More Array Parameters قرار دهید.

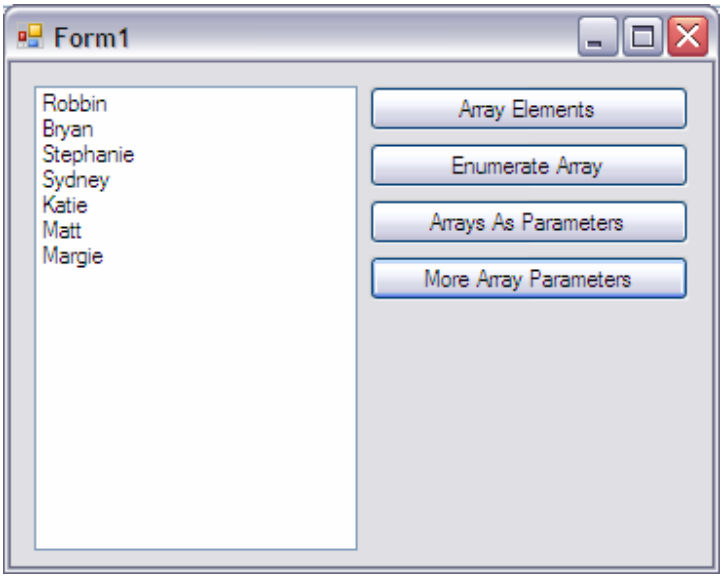
۲) بر روی این کنترل دو بار کلیک کنید و کد مشخص شده در زیر را در متد مربوط به رویداد Click وارد کنید:

```
private void btnMoreArrayParameters_Click(object sender,
                                          EventArgs e)
{
    // Declare an array
    string[] strMoreFriends = new string[2];

    // Populate the array
    strMoreFriends[0] = "Matt";
    strMoreFriends[1] = "Margie";

    // List your friends
    AddItemsToList(strFriends);
    AddItemsToList(strMoreFriends);
}
```

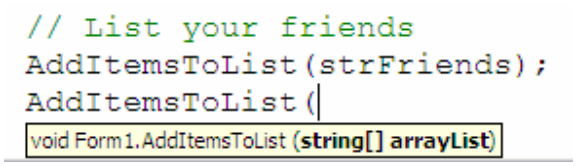
۳) برنامه را اجرا کنید و بر روی دکمه ای که جدیداً اضافه کرده اید کلیک کنید. نتیجه ای مشابه شکل ۵-۳ را مشاهده خواهید کرد.



شکل ۵-۳

چگونه کار می کند؟

در این مثال آرایه ای را به طول دو ایجاد کردیم و آن را به متد `AddItemsToList` فرستادیم تا آن را بـه لیـست اضـافه کند. همانطور که مشاهده کردید، طول آرایه ای که به عنوان پارامتر به این متد ارسال می کنید، اهمیتی ندارد.
هنگامی که در حال نوشتن برنامه هستید، اگر یک متد، پارامتر خود را به صورت آرایه دریافت کند، در پنجـره ای کـه بـرای تکمیـل هوشمندانه ی نام متد توسط ویژوال استودیو باز می شود، در مقابل نوع داده ای آرایه یک کروشه خالی قرار دارد. (شکل ۵-۴)



شکل ۵-۴

نکته: در پنجره باز شده نه تنها آرایه ای بودن یک پارامتر قابل تشخیص است، بلکه نوع این پارامتر نیز نمایش داده می شود.

مرتب سازی آرایه ها:

یکی از مواردی که همواره هنگام کار با آرایه ها مورد نیاز بوده است، مرتب کردن آرایه است. در بخش امتحـان کنیـد بعـد، مـشاهده خواهید کرد که چگونه می توان آرایه ها را مرتب کرد.

امتحان کنید: مرتب سازی آرایه ها

۱) در بخش طراحی فرم کنترل `Button` دیگری به `Form1` اضافه کـرده، سـپس خاصـیت `Name` آن را برابـر بـا `btnSortingArrays` و خاصیت `Text` آن را برابر `Sorting Arrays` قرار دهید.
۲) روی این کنترل دو بار کلیک کنید و کد مشخص شده در زیر را به متد مربوط به رویداد `Click` آن اضافه کنید:

```
private void btnSortingArrays_Click(object sender,
                                    EventArgs e)
{
    // Sort the array
    Array.Sort(strFriends);

    // List your friends
    AddItemsToList(strFriends);
}
```

۳) برنامه را اجرا کنید و روی این دکمه کلیک کنید. مشاهده خواهید کرد که لیست، اسامی موجود در آرایه را که بـه صـورت الفبایی مرتب شده اند نمایش می دهد.

چگونه کار می کند؟

تمام آرایه ها به صورت درونی از کلاسی به نام System.Array مشتق می شوند. در این برنامه از یکی از متدهای این کلاس به نام Sort استفاده می کنیم. این متد فقط یک پارامتر می گیرد و آن پارامتر نیز نام آرایه ای است که می خواهید آن را مرتب کنید. سپس این متد همانطور که از نام آن مشخص است، آرایه را بر اساس نوع داده ای عناصر آن، مرتب می کند و باز می گرداند.در این برنامه به علت این که نوع عناصر رشته ای است، آرایه به صورت الفبایی مرتب می شود. اگر از این متد برای مرتب سازی یک آرایه از اعداد صحیح و یا اعداد اعشاری استفاده کنید، آرایه به صورت عددی مرتب خواهد شد.

```
// Sort the array
Array.Sort(strFriends);
```

ای قابلیت که یک متد می تواند پارامترهایی از نوع های داده ای مختلف را دریافت کند، سپس بر اساس نوع پارامتر عملیات مناسبی را بر روی آن انجام دهد، سربار گذاری[1] متدها می نامند. در اینجا می گوییم که تابع Sort یک تابع سربار گذاری شده است. در فصل دهم بیشتر در مورد سربار گذاری متدها و نحوه ایجاد چنین متدهایی صحبت خواهیم کرد.

حرکت به عقب در آرایه ها:

حلقه های foreach فقط در یک جهت در بین عناصر یک آرایه حرکت می کنند. آنها از عنصر صفرم یک آرایه شروع می کنند و تا آخرین عنصر آرایه پیش می روند. اگر بخواهید بین عناصر یک آرایه به صورت برعکس حرکت کنید (یعنی از عنصر آخر به عنصر اول برگردید) دو راه در اختیار دارید.
راه اول این است که از حلقه های for معمولی استفاده کنید. در این صورت باید مقدار اولیه شمارنده حلقه را یک واحد کمتر از طول آرایه قرار دهید[2] و سپس حلقه را طوری تنظیم کنید که تا عدد یک برگردد. در قسمت زیر این روش را مشاهده می کنید:

```
for (int intIndex = strFriends.GetUpperBound(0);
          intIndex >= 0; intIndex--)
{
     // Add the array item to the list
     lstFriends.Items.Add(strFriends[intIndex]);
}
```

روش دوم این است که از متد Reverse در کلاس Array استفاده کنید. این متد عناصر یک آرایه را به صورت معکوس در آرایه قرار می دهد. بعد از استفاده از این متد، میتوانید از حلقه های foreach معمولی برای نمایش آرایه استفاده کنید. در امتحان کنید بعد، این روش را مشاهده خواهید کرد.

---

[1] Overloading

[2] همانطور که به خاطر دارید، اندیس آخرین عنصر هر آرایه یک واحد کمتر از طول آن آرایه است. بنابراین برای بازگشتن از آخرین عنصر آرایه باید از یک واحد کمتر از طول به عنوان عدد شروع استفاده کنید.

امتحان کنید: معکوس کردن یک آرایه

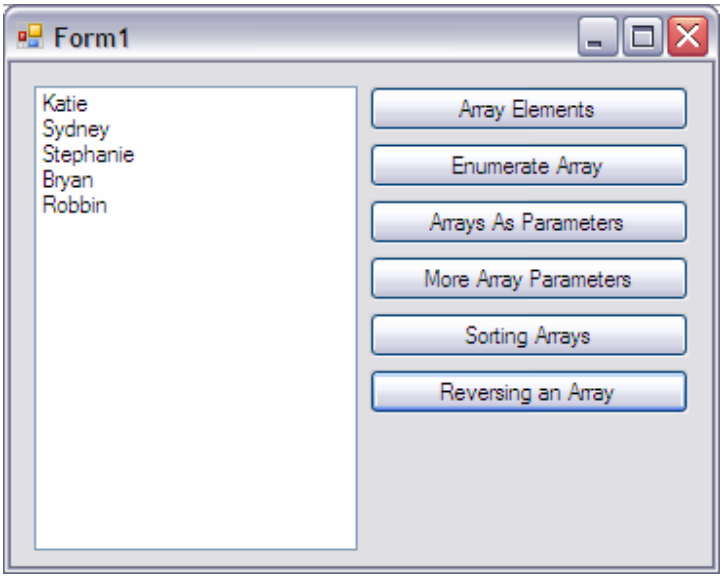
۱) کنترل Button دیگری به قسمت طراحی فرم اضافه کرده، خاصیت Name آن را برابر با btnReversingAnArray و خاصیت Text آن را برابر با Reversing an Array قرار دهید.

۲) روی این دکمه دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس کد زیر را در این متد وارد کنید:

```
private void btnReversingAnArray_Click(object sender,
                                       EventArgs e)
{
     // Reverse the order -
     // elements will be in descending order
     Array.Reverse(strFriends);

     // List your friends
     AddItemsToList(strFriends);
}
```

۳) برنامه را اجرا کنید و روی دکمه ی Reversing an Array کلیک کنید. مشاهده می کنید که همانند شکل ۵-۵ نامهای موجود در لیست به صورت معکوس نمایش داده شده اند.



شکل ۵-۵

چگونه کار می کند؟

تابع Reverse عناصر موجود در یک آرایه ی یک بعدی را به صورت معکوس در آن آرایه قرار می دهد. هنگامی کـه آرایـه ی strFriends را به این متد ارسال می کنید، در حقیقت از متد می خواهید که عناصر این آرایه را از آخر به اول قرار دهد:

```
// Reverse the order –
// elements will be in descending order
Array.Reverse(strFriends);
```

هنگامی که عناصر آرایه به صورت معکوس در آن قرار گرفتند، کافی است آرایه را به متد AddItemsToList بفرستید تـا آن را در لیست نمایش دهد.

```
// List your friends
AddItemsToList(strFriends);
```

نکته: اگر می خواهید آرایه به صورت نزولی الفبایی مرتب شود، کافی است آرایه را به وسیله تابع Sort به صورت صعودی الفبایی مرتب کنید، سپس با استفاده از تابع Reverse آن را به صورت معکوس درآورید. به این صورت آرایه به صورت الفبـایی نزولـی مرتب می شود.

مقدار دهی اولیه به آرایه ها:

در ویژوال C# می توانید آرایه ها را در همان خطی که تعریف می کنید، مقدار دهی کنید. به عبارت دیگر نیـازی نیـست کـه بعـد از تعریف آرایه از چندین خط کد همانند زیر برای پر کردن آرایه استفاده کنید:

```
// Declare an array
string[] strFriends = new string[4];

// Populate the array
strFriends[0] = "Robbin";
strFriends[1] = "Bryan";
strFriends[2] = "Stephanie";
strFriends[3] = "Sydney";
strFriends[4] = "Katie";
```

در بخش امتحان کنید زیر، با چگونگی مقداردهی آرایه ها در یک خط آشنا خواهید شد.

امتحان کنید: مقداردهی اولیه به آرایه ها

۱) در قـسمت طراحـی فـرم، کنتـرل Button دیگـری را بـه Form1 اضـافه کـرده، خاصـیت Name آن را برابـر btnInitializingArrayWithValues و خاصـــــیت Text آن را برابـــــر بـــــا Initializing Array With Values قرار دهید.

۲) روی این کنترل دو بار کلیک کنید و کد مشخص شده در زیر را در متد مربوط به رویداد Click آن وارد کنید:

```
private void btnInitializingArrayWithValues_Click(
                        object sender, EventArgs e)
{
     // Declare an populate an array
     String[] strMyFriends = new string[] {
          "Robbin","Bryan","Stephanie",
          "Sudney","Katie"};

     // List your friends
     AddItemsToList(strMyFriends);
}
```

۳) برنامه را اجرا کنید و بر روی دکمه ی جدید کلیک کنید. مشاهده خواهید کرد که لیست، با عناصر موجود در آرایه پر می شود.

چگونه کار می کند؟

هنگامی که یک آرایه را تعریف کردید، می توانید مقادیر هر یک از عناصر آن را به ترتیب در یک جفت آکولاد وارد کنید. در این حالت، ویژوال C# از خانه صفرم آرایه شروع می کند و مقادیر وارد شده را به ترتیب در خانه های آرایه قرار می دهد. در این مثال پنج رشته که به وسیله ویرگول از یکدیگر جدا شده اند را در آرایه قرار داده ایم. توجه کنید که در این قسمت طول آرایه را وارد نکرده ایم، بلکه طول آرایه بر اساس مقادیر وارد شده در داخل آکولاد به صورت اتوماتیک توسط کامپایلر محاسبه میشود.

```
// Declare an populate an array
String[] strMyFriends = new string[] {
     "Robbin","Bryan","Stephanie",
     "Sudney","Katie"};
```

البته همانطور که می بینید این روش برای مقدار دهی به آرایه های بزرگ مناسب نیست. اگر در برنامه می خواهید آرایه بزرگی را پر کنید، بهتر است از روشی که در بخش قبل گفته شد استفاده کنید. یعنی با استفاده از نام آرایه و اندیس خانه مورد نظر، مقدار آن عنصر را وارد کنید.

مفهوم شمارنده ها:

متغیرهایی که تاکنون ایجاد کرده ایم، هیچ محدودیتی در نوع اطلاعاتی که می توانستند در خود ذخیره کنند نداشتند. برای مثال اگر متغیری را از نوع `int` تعریف می کردید، می توانستید هر عدد صحیحی را در آن نگهداری کنید[1]. این مسئله برای متغیرهای `string` و `double` هم وجود داشت.
اما در بعضی از شرایط، ممکن است بخواهید متغیر شما، اعداد محدودی را در خود نگهداری کند. برای مثال فرض کنید می خواهید متغیری از نوع عدد صحیح تعریف کنید و تعداد درهای یک اتومبیل را در آن ذخیره کنید. قطعا نمی خواهید اجازه دهید که عدد ۱۶۳۲۷ در این متغیر ذخیره شود. برای رفع این مشکل می توانید از شمارنده ها استفاده کنید.

استفاده از شمارنده ها:

با استفاده از شمارنده ها میتوانید نوع های داده ای جدیدی بر اساس نوع های داده ای موجود از قبیل `int`، `long`، `short` و یا `byte` بسازید. متغیرهایی که از این نوع داده ی جدید ایجاد می شوند، فقط می توانند مقداری را داشته باشند که شما مشخص می کنید. به این ترتیب میتوانید در برنامه از وارد شدن اعداد غیر منطقی در متغیر ها جلوگیری کنید. همچنین استفاده از شمارنده ها در کد باعث افزایش خوانایی و وضوح می شود. در بخش امتحان کنید بعدی، مشاهده خواهید کرد که چگونه میتوان برنامه ای ساخت که بر اساس ساعت، یکی از اعمال قابل اجرا در یک روز را انتخاب کند.

- آماده شدن برای رفتن به محل کار.
- رفتن به محل کار.
- در محل کار بودن.
- رفتن برای نهار.
- برگشتن از محل کار.
- با دوستان بودن.
- آماده شدن برای خواب.
- خوابیدن.

امتحان کنید: استفاده از شمارنده ها

۱) با استفاده از ویژوال استودیو ۲۰۰۵ یک برنامه تحت ویندوز جدید ایجاد کنید و نام آن را برابر با `Enum Demo` قرار دهید.
۲) شمارنده ها معمولا به عنوان عضوی از کلاسی که در حال کد نویسی در آن هستید، تعریف می شوند. بعد از اینکه قسمت طراحی `Form1` باز شد، روی فرم کلیک راست کنید و گزینه ی `View Code` را انتخاب کنید. به این ترتیب قسمت ویرایشگر کد برای این فرم نمایش داده می شود. سپس کد مشخص شده در زیر را در بالای پنجره، بعد از نام کلاس وارد کنید:

[1] البته به خاطر محدود بودن فضای حافظه ای که به یک متغیر `int` اختصاص داده می شود، فقط اعداد در بازه خاصی را میتوان در این نوع متغییر ها نگهداری کرد. اما هر عددی که در این بازه وجود داشته باشد را میتوان در متغیرهای عدد صحیح ذخیره کرد و در بین اعداد موجود در این بازه محدودیتی وجود ندارد.

```
public partial class Form1 : Form
{
    private enum DayAction
    {
        GettingReadyForWork = 0,
        TravelingToWork,
        AtWork,
        AtLunch,
        TravelingFromWork,
        RelaxingForFriends,
        GettingReadyForBed,
        Asleep
    };
```

۳) هنگامی که شمارنده را تعریف کردید، میتوانید متغیری تعریف کنید که نوع داده ای آن برابر با این شمارنده باشد. متغیر زیر را در کلاس مربوط به `Form1` تعریف کنید:

```
    // Declare variable
    private DayAction CurrentState;
```

۴) به قسمت طراحی `Form1` برگردید و خاصیت `Text` فرم را برابر به `What's Matt Doing?` تغیر دهید.

۵) حال یک کنترل `DateTimePicker` به فرم اضافه کرده و خاصیتهای آن را برابر با مقادیر زیر قرار دهید:

- خاصیت `Name` آن را به `dtpHour` تغییر دهید.
- خاصیت `Format` آن را به `Time` تغییر دهید.
- خاصیت `ShowUpDown` را برابر با `true` قرار دهید.
- مقدار `Value` را `00:00 AM` وارد کنید.
- خاصیت `Size` را برابر با `91;20` وارد کنید.

۶) یک کنترل `Label` در فرم قرار داده، خاصیت `Name` آن را به `lblState` و خاصیت `Text` آن را به `State Not Initialized` تغییر دهید. اندازه فرم خود را نیز به گونه ای تغییر دهید که فرم شما مشابه شکل ۵-۶ شود.



شکل ۵-۶

۷) بر روی زمینه ی فرم کلیک کنید تا رویداد Load مربوط به Form1 ایجاد شود. سپس کدهای مشخص شده در زیر را در این متد وارد کنید.

```
private void Form1_Load(object sender, EventArgs e)
{
     // Set the hour property to the current hour
     this.Hour = DateTime.Now.Hour;
}
```

۸) حال کد مشخص شده در این قسمت را در پایین کدی که در قسمت سوم این بخش وارد کردید قرار دهید:

```
// Hour property
private int Hour
{
     get
     {
          // Return the current hour displayed
          return dtpHour.Value.Hour;
     }
     set
     {
          // Set the date using the hour
          // passed to this property
          dtpHour.Value = new DateTime(
               DateTime.Now.Year, DateTime.Now.Month,
               DateTime.Now.Day, value, 0, 0);
          // Set the display text
          lblState.Text = "At " + value + ":00 Matt is ";
     }
}
```

۹) به قسمت طراحی فرم برگردید و بر روی کنترل dtpHour دو بار کلیک کنید. متد مربوط به رویداد ValueChanged این کنترل به صورت اتوماتیک ایجاد خواهد شد. کد مشخص شده در زیر را به این متد اضافه کنید:

```
private void dtpHour_ValueChanged(object sender,EventArgs
e)
{
     // Update the hour property
     this.Hour = dtpHour.Value.Hour;
}
```

۱۰) برنامه را اجرا کنید و بر روی علامت های بالا و پایین کنار کنترل DateTimePicker کلیک کنید. مشاهده خواهید کرد که متن داخل کنترل لیبل تغییر می کند و ساعت انتخاب شده را نمایش می دهد (شکل ۵-۷).



شکل ۵-۷

چگونه کار می کند؟

در این برنامه، کاربر میتواند با استفاده از کنترل DateTimePicker یک ساعت از شبانه روز را انتخاب کند. سپس شما با توجه به ساعت انتخاب شده توسط کاربر مشخص می کنید که مت در آن ساعت مشغول انجام کدامیک از هشت کار تعریف شده است. برای این کار ابتدا باید ساعت مشخص شده توسط کاربر را در مکانی نگهداری کنید. برای این کار یک خاصیت جدید به فرمی که در حال کار با آن هستیم اضافه می کنیم. این خاصیت همانند دیگر خاصیتهای فرم مثل Name است. نام این خاصیت جدید Hour است و برای تنظیم ساعت در کنترل DateTimePicker و کنترل لیبل به کار می رود. برای تعریف یک خاصیت جدید کافی است که نوع داده ای و نام آن را مشخص کنید و سپس مانند زیر در بدنه آن خاصیت، دو بلاک به نامهای set و get تعریف کنید:

```
private int Hour
{
      get
      {
            …
            return dtpHour.Value.Hour;
      }
      set
      {
            …
      }
}
```

به بلاکهای get و set درون خاصیت توجه کنید. دستورات بلاک get که شامل دستور return هم هستند زمانی فراخوانی می شوند که بخواهیم مقدار ذخیره شده در خاصیت را بدست آوریم. در بلاک get لازم نیست نوع مقدار برگردانده شده توسط آن را مشخص کنیم، زیرا این نوع هم اکنون در خود تعریف خاصیت به صورت int مشخص شده است. پس دستورات بلاک get باید مقداری را از نوع عدد صحیح برگردانند. برای مثال هنگامی که از این خاصیت در سمت راست علامت مساوی استفاده کنیم، برنامه نیاز دارد که به مقدار آن دسترسی پیدا کند، به همین دلیل به صورت اتوماتیک دستورات بخش get را اجرا می کند.

دستورات بخش set زمانی اجرا می شوند که بخواهیم مقدار این خاصیت را تغییر دهیم. برای مثال اگر از این خاصیت در سمت چپ یک تساوی استفاده کنیم، برنامه با استفاده از متد set، حاصل عبارت سمت راست تساوی را در خاصیت قرار می دهد. در این قسمت نیز نیازی نیست نام و یا نوع داده ای مقداری که به بلاک set فرستاده می شود را مشخص کنیم. نوع داده ای این مقدار

که برابر با نوع داده ای خاصیت خواهد بود. مقدار فرستاده شده هم با کلمه کلیدی value به بلاک set ارسال می شود. بنابراین در بلاک set برای دسترسی به مقدار فرستاده شده، بایستی از کلمه کلیدی value استفاده کنید.
هنگامی که برنامه شروع می شود، مقدار خاصیت Hour را برابر با مقدار کنونی ساعت سیستم قرار می دهیم. برای دسترسی به مقدار کنونی ساعت سیستم، می توانیم از خاصیت Now در کلاس DateTime استفاده کنیم:

```
private void Form1_Load(object sender, EventArgs e)
{
     // Set the hour property to the current hour
     this.Hour = DateTime.Now.Hour;
}
```

یکی از مواقعی که باید خاصیت Hour را در فرم تنظیم کنید، زمانی است که خاصیت Value مربوط به کنترل DateTimePicker تغییر می کند، یا به عبارت دیگر کاربر زمان نمایش داده شده در این کنترل را تغییر می دهد. برای این کار از متد مربوط به رویداد ValueChanged کنترل DateTimePicker استفاده می کنیم:

```
private void dtpHour_ValueChanged(object sender,EventArgs
e)
{
     // Update the hour property
     this.Hour = dtpHour.Value.Hour;
}
```

همچنین هنگامی که خاصیت Hour در فرم تغییر کرد باید مقدار نمایش داده شده توسط کنترل DateTimePicker و نیز کنترل لیبل را برابر با مقدار جدید قرار دهید. کد مربوط به این مورد را می توانید در بلاک set در خاصیت وارد کنید تا با هر بار تغییر مقدار Hour، اجرا شود.
اولین موردی که باید در بلاک set تغییر دهید، مقدار کنترل DateTimePicker است. ممکن است تصور کنید همانطور که برای دسترسی به مقدار ساعت این کنترل از خاصیت Value.Hour استفاده می کردیم، می توانیم با تنظیم این خاصیت موجب شویم که مقدار جدیدی در این کنترل نمایش داده شود. اما خاصیت هایی مانند Hour از نوع فقط-خواندنی هستند و نمی توان آنها را به تنهایی تنظیم کرد. برای تنظیم مقدار ساعت در این کنترل، باید مقدار کل خاصیت Value را تغییر دهیم. خاصیت Value یک متغیر از نوع DateTime را دریافت می کند. بنابراین برای اینکه فقط مقدار Hour را تغییر دهیم، یک متغیر جدید از نوع DateTime تعریف کرده و تمام مقادیر آن به جز مقدار Hour را برابر با مقادیر قبلی قرار می دهیم.
برای ایجاد یک متغیر جدید از نوع DateTime از دستور new استفاده می کنیم (در فصل ۱۰ به طور کامل با این دستور آشنا می شوید). برای ایجاد یک متغیر جدید از نوع DateTime باید مقادیر مختلف زمان را برای آن فراهم کرد. مقدار Year، Month و Day را با استفاده از خاصیت Now کلاس DateTime بدست می آوریم. ساعت را برابر با ساعت مورد نظرمان (ساعت فرستاده شده به خاصیت Hour) قرار می دهیم و دقیقه و ثانیه را هم برابر با صفر در نظر می گیریم.

```
// Set the date using the hour passed to this property
dtpHour.Value = new DateTime(DateTime.Now.Year,
     DateTime.Now.Month, DateTime.Now.Day, value, 0, 0);
```

دومین موردی که باید در بلاک set تغییر کند، متن نشان داده شده در کنترل لیبل است که باید مقدار جدید ساعت را نمایش دهد:

```
// Set the display text
lblState.text = "At " + value + ":00 Matt is ";
```

در این بخش مشخص نکردیم که مت در آن ساعت مشغول به چه کاری است. در بخش امتحان کنید بعد، این مورد را نیز انجام می دهیم.

تعیین موقیت:

در بخش امتحان کنید بعد، مشاهده خواهیم کرد که چگونه می توان با تغییر ساعت، موقعیت مت را مشخص کرد. برای این کار ساعت را از کنترل DateTimePicker دریافت می کنید و سپس مشخص می کنید که کدامیک از مقادیر موجود در شمارنده DayAction برای این ساعت مناسب است. بعد از تعیین این مورد، آن را بر روی صفحه نمایش می دهید.

امتحان کنید: تعیین موقعیت

۱) قسمت ویرایشگر متن را برای Form1 باز کنید و کد موجود در بلاک set خاصیت Hour را به صورت زیر تغییر دهید:

```
set
{
     // Set the date using the hour passed to this property
     dtpHour.Value = new DateTime(DateTime.Now.Year,
     DateTime.Now.Month, DateTime.Now.Day, value, 0, 0);

     // Determine the state
     if (value >= 6 && value < 7)
          CurrentState = DayAction.GettingReadyForWork;
     else if (value > 7 && value < 8)
          CurrentState = DayAction.TravelingToWork;
     else if (value >= 8 && value < 13)
          CurrentState = DayAction.AtWork;
     else if (value >= 13 && value < 14)
          CurrentState = DayAction.AtLunch;
     else if (value >= 14 && value < 17)
          CurrentState = DayAction.AtWork;
     else if (value >= 17 && value < 18)
          CurrentState = DayAction.TravelingFromWork;
     else if (value >= 18 && value < 22)
```

```
            CurrentState = DayAction.RelaxingForFriends;
        else if (value >= 22 && value < 23)
            CurrentState = DayAction.GettingReadyForBed;
        else
            CurrentState = DayAction.Asleep;

        // Set the display text
        lblState.Text = "At " + value + ":00 Matt is " +
                        CurrentState;
}
```

۲) برنامه را اجرا کنید و روی علامت های بالا و پایین کنترل `DateTimePicker` کلیک کنید. نتیجه ای را مشابه شکل ۵–۸ مشاهده خواهید کرد.



شکل ۵–۸

چگونه کار می کند؟

در حال نوشتن این کدها، توجه کنید هنگامی که بخواهید مقدار متغیر `CurrentState` را تنظیم کنید، لیستی در ویژوال استودیو باز شده و مقادیر ممکن برای این متغیر را نمایش می دهد (شکل ۵–۹).

ویژوال استودیو ۲۰۰۵ می داند که متغیر `CurrentSate` از نوع `DayAction` تعریف شده است. همچنین ویژوال استودیو می داند که `DayAction` یک شمارنده است که هشت مقدار مختلف را می تواند در خود نگهداری کند، و هر کدام از این مقادیر در پنجره ای که هنگام نوشتن کد نمایش داده می شوند نشان داده می شود.

```
// Determine the state
if (value >= 6 && value < 7)
    CurrentState = DayAction.;
else if (value > 7 && value
    CurrentState = DayActic                rk;
else if (value >= 8 && valu
    CurrentState = DayActic
else if (value >= 13 && val
    CurrentState = DayActic
else if (value >= 14 && val
```

Asleep
AtLunch
AtWork
GettingReadyForBed
GettingReadyForWork
RelaxingForFriends
TravelingFromWork
TravelingToWork

شکل ۵–۹

اگر در هنگام نمایش مقدار CurrentState در لیبل متنهای دیگر را نمایش ندهید و فقط متغیر CurrentState را در لیبل قرار دهید مانند کد زیر، هنگام خطا با کامپایل روبرو خواهید شد.

```
lblState.Text = CurrentState;
```

این خطا به این دلیل است که نوع متغیر CurrentState در حقیقت از نوع عدد صحیح است و نمی تواند به صورت مستقیم در یک لیبل قرار بگیرد. برای این که این مقدار را در لیبل قرار دهید باید آن را به رشته تبدیل کنید. هنگامی که این متغیر را با دیگر رشته ها استفاده می کنید (همانند کد نوشته شده در برنامه)، ویژوال C# به صورت اتوماتیک مقدار CurrentState را به رشته تبدیل می کند و در لیبل قرار می دهد. در مواقعی که می خواهید مقدار CurrentState را به تنهایی نمایش دهید بایستی با استفاده از تابع ToString() آن را به رشته تبدیل کنید و سپس در لیبل قرار دهید.
استفاده از شمارنده ها، هنگامی که بخواهید یک عدد خاص را از بین یک سری مقادیر مشخص انتخاب کنید بسیار مناسب است. هنگامی که بیشتر با توابع و کلاسهای داخلی .NET کار کنید، متوجه خواهید شد که در قسمتهای بسیاری از NET.، از شمارنده ها استفاده شده است.

مفهوم ثابت ها:

یکی دیگر از تمرین های خوب برنامه نویسی که باید با آن آشنا شوید استفاده از ثابت ها است. تصور کنید در حال نوشتن یک برنامه برای محاسبه مالیات برای حقوق کارمندان یک شرکت هستید. در هنگام نوشتن برنامه، درصد مالیاتی که باید از حقوق هر کارمند کم شود فرضاً A درصد است. بعد از مدتی این مقدار به عددی مانند B تغییر پیدا می کند. در این حالت باید تمام قسمتهایی از برنامه که مالیات را A درصد وارد کرده اید پیدا کنید و عدد A را به B تغییر دهید. حال اگر حجم برنامه بزرگ باشد، کار بسیار خسته کننده و طولانی را باید انجام دهید. با استفاده از ثابت ها در برنامه می توانید از بروز چنین مشکلاتی جلوگیری کنید. همچنین ثابت ها باعث می شوند که خوانایی برنامه نیز افزایش پیدا کند.

استفاده از ثابت ها:

فرض کنید در برنامه ی زیر دو متد متفاوت مانند زیر دارید و در هر کدام می خواهید فایل مشخصی را باز کنید و روی آن عملیاتی انجام دهید:

```
public void DoSomething()
{
     // What's the file name?
     string FileName = @"C:\Temp\Demo.txt";

     // Open the file
     ...
}
public void DoSomethingElse()
```

```
{
      // What's the file name
      string FileName = @"C:\Temp\Demo.txt";

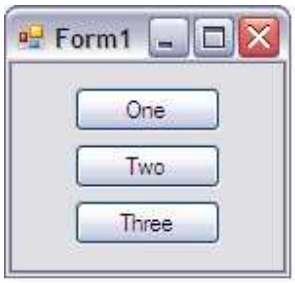
      // Open the file
      ...
}
```

در این کد دو ثابت رشته ای تعریف شده و نام فایل در آنها قرار گرفته است. این نوع برنامه نویسی ضعف های زیادی دارد، به این علت که نام فایل در دو متغیر مجزا تعریف شده است و اگر بخواهید نام فایل را تغیر دهید باید در هر دو متد تغییر ایجاد کنید.
در این مثال هر دو متد در کنار هم قرار دارند و حجم برنامه نیز کوچک است، اما تصور کنید که برنامه بسیار بزرگ باشد و از این رشته بخواهید در ۱۰، ۵۰ و یا بیش از ۱۰۰۰ قسمت استفاده کنید. در این صورت اگر بخواهید نام فایل را تغییر دهید، باید در تمام این قسمتها نام را عوض کنید. این مورد، دقیقا یکی از مواردی است که موجب به وجود آمدن خطاهای زیاد در نوشتن و نگهداری برنامه می شود.
برای رفع این مشکل می توانید یک نام کلی برای این ثابت رشته ای پیدا کنید و در برنامه از این نام به جای نام فایل استفاده کنید. به این کار تعریف یک **"ثابت"** گفته می شود. به عبارت دیگر، ثابت نوع خاصی از متغیر است که مقدار آن در طول برنامه قابل تغییر نیست. در امتحان کنید بخش بعد، نحوه تعریف و استفاده از آنها را مشاهده خواهید کرد.

امتحان کنید: استفاده از ثابتها

۱) با استفاده از ویژوال استودیو ۲۰۰۵ برنامه تحت ویندوز جدیدی ایجاد کرده و نام آن را `Constants Demo` قرار دهید.

۲) در محیط طراحی فرم، سه کنترل `Button` بر روی فرم قرار داده و خاصیت `Name` آنها را به ترتیب با مقادیر `btnOne`، `btnTwo` و `btnThree` تنظیم کنید. سپس خاصیت `Text` این کنترل ها را برابر با `One`، `Two` و `Three` قرار دهید. بعد از این موارد فرم شما باید مشابه شکل ۵-۱۰ باشد.



شکل ۵-۱۰

۳) به بخش ویرایشگر کد بروید و کد مشخص شده در زیر را در بالای کد بعد از تعریف کلاس `From1` قرار دهید:

```
public partial class Form1 : Form
```

```
{
    // File name constant
    private      const      string      strFileName      =
@"C:\Temp\Demo.txt";
```

۴) به قسمت طراحی فرم برگردید و بر روی دکمه btnOne دو بار کلیک کنید تا متد مربوط به رویداد Click آن ایجاد شود. سپس کد مشخص شده در زیر را در این متد وارد کنید:

```
private void btnOne_Click(object sender, EventArgs e)
{
    // Using a constant
    MessageBox.Show("1: " + strFileName, "Constants
Demo");
}
```

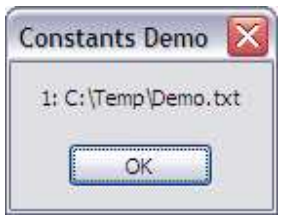
۵) مجددا به قسمت طراحی فرم برگردید و روی دکمه btnTwo دو بار کلیک کنید تا متد رویداد Click آن نیز ایجاد شود. سپس کد زیر را در آن متد وارد کنید:

```
private void btnTwo_Click(object sender, EventArgs e)
{
    // Using a constant
    MessageBox.Show("2: " + strFileName, "Constants
Demo");
}
```

۶) در آخر روی دکمه فرمان سوم روی فرم دو بار کلیک کنید و کد زیر را در قسمت مربوط به رویداد Click آن وارد کنید:

```
private void btnThree_Click(object sender, EventArgs e)
{
    // Using a constant
    MessageBox.Show("3: " + strFileName, "Constants
Demo");
}
```

۷) حال برنامه را اجرا کنید و بر روی دکمه فرمان اول کلیک کنید. نتیجه ای را مشابه شکل ۵-۱۱ مشاهده خواهید کرد.



شکل ۵-۱۱

اگر بر روی دکمه های فرمان دوم و سوم هم کلیک کنید همین نام فایل را مشاهده خواهید کرد.

چگونه کار می کند؟

همانطور که گفتم، ثابت ها همانند متغیر ها هستند با این تفاوت که مقدار آنها در طول اجرای برنامه نمی تواند تغییر کند. تعریف ثابت ها همانند تعریف متغییر ها است، با این تفاوت که قبل از تعریف نوع داده ای باید از یک عبارت `const` نیز استفاده کرد.

```
// File name constant
private const string strFileName = @"C:\Temp\Demo.txt";
```

توجه کنید که ثابتها هم مانند متغیرها دارای یک نوع داده ای هستند و باید دارای مقدار اولیه باشند که هنگام تعریف آن را مشخص می کنیم.
برای استفاده از ثابتها نیز همانند متغیرها، کافی است که نام آنها را در برنامه وارد کنیم:

```
private void btnOne_Click(object sender, EventArgs e)
{
      // Using a constant
      MessageBox.Show("1: " + strFileName, "Constants
Demo");
}
```

همانطور که قبلا گفتم، مزیت استفاده از ثابتها در این است که باعث می شود با تغییر قسمتی از کد، در بخشهای زیادی از برنامه تغیر ایجاد کنید. البته دقت داشته باشید که مقادیر ثابتها را فقط در هنگام نوشتن کد می توانید تغییر دهید و هنگامی که برنامه در حال اجرا باشد این مقادیر ثابت هستند.

نکته: ممکن است تعریف مفهوم ثابت در ابتدا کمی خنده دار به نظر برسد، "ثابت متغیری است که مقدار آن نمی تواند تغییر کند." اما این تعریف را فقط برای درک بهتر این مفهوم به کار برده ام. در حقیقت استفاده کردن و یا نکردن از ثابت ها هیچ تفاوتی در کد نهایی یک برنامه (کد محلی و یا کد MSIL) و یا در سرعت اجرای آن ندارد. به عبارت دیگر ثابت ها فقط برای افزایش خوانایی برنامه و همچنین سادگی تغییرات آتی آن به کار می روند. اگر در یک برنامه از ثابتی استفاده کنید، هنگام کامپایل کامپایلر تمام کد برنامه را جستجو کرده و هر جا به نام ثابت برخورد کند، مقدار آن را با مقدار مشخص شده برای ثابت عوض می کند. به عبارت دیگر تعریف یک ثابت بر خلاف متغیر هیچ فضایی از برنامه را در زمان اجرا اشغال نمی کند.

ثابتها با نوعهای داده ای گوناگون:

در این بخش، فقط از نوع داده ای رشته برای تعریف ثابت ها استفاده کردیم، اما در برنامه های واقعی از هر نوع داده ای که تاکنون با آن آشنا شده اید می توانید برای تعریف یک ثابت استفاده کنید. البته دقت داشته باشید اشیایی که از کلاسها به وجود می آیند (چه

کلاسهای موجود در .NET و چه کلاس هایی که برنامه نویس آنها را می سازد) نمی توانند در ثابت ها مورد استفاده قرار گیرند. در مورد کلاسها در فصل ۹ بیشتر صحبت خواهیم کرد.
برای مثال، اعداد صحیح یکی از نوع های داده ای هستند که در تعریف ثابت ها زیاد به کار می روند:

```
public const int intHourAsleepPerDay = 8;
```

ساختارها:

هنگام نوشتن نرم افزار در بیشتر مواقع نیاز خواهید داشت مقداری اطلاعات که هر کدام نوع داده ای متفاوتی دارند، ولی همگی به یک موضوع مرتبط می شوند را در یک گروه ذخیره کنید. برای مثال می خواهید اطلاعات مربوط به یک مشترک از قبیل نام و آدرس او (از نوع رشته ای) و نیز مقدار دارایی او (از نوع عددی) را در یک گروه اطلاعات ذخیره کنید. برای نگهداری چنین اطلاعاتی، یک کلاس ویژه طراحی می کنید و اطلاعات را در یک شیئ از آن کلاس قرار می دهید که این روش در فصل ۹ توضیح داده می شود. اما روش دیگر برای نگهداری این مجموعه اطلاعات، استفاده از ساختارها است. ساختارها نیز کاربردی مانند کلاسها دارند اما ساده تر هستند، بنابراین آنها را در این قسمت بررسی خواهیم کرد.
بعدها هنگام برنامه نویسی، نیاز خواهید داشت که تصمیم بگیرید برای یک حالت خاص باید از کلاسها استفاده کنید و یا از ساختارها. به عنوان یک قانون کلی همواره در نظر داشته باشید که اگر می خواهید متدها، توابع، خاصیت ها و متغیرهای زیادی را که همگی به هم مرتبط هستند در یک گروه قرار دهید، بهتر است که از کلاسها استفاده کنید. در غیر این صورت بهتر است که ساختارها را به کار ببرید. فقط در نظر داشته باشید که اگر یک سری اطلاعات را در یک ساختار قرار دهید و پس از مدتی بخواهید آن را به یک کلاس تبدیل کنید، مشکل زیادی را خواهید داشت. پس بهتر است قبل از تعریف کلاس و یا ساختار، به دقت بررسی کنید که کدام مورد برای استفاده شما مناسب تر است.

ایجاد ساختارها:

در امتحان کنید زیر، با نحوه ایجاد یک ساختار آشنا خواهیم شد.

امتحان کنید: ایجاد یک ساختار

۱) با استفاده از ویژوال استودیو ۲۰۰۵ برنامه تحت ویندوز جدیدی به نام Structure Demo ایجاد کنید.
۲) بعد از این که پروژه ایجاد شد، در پنجره Solution Explorer بر روی نام پروژه کلیک راست کرده، از منوی نمایش داده شده گزینه Add را انتخاب کنید. سپس از زیر منوی باز شده گزینه ...Class را انتخاب کنید. پنجره ای به نام Add New Item – Structure Demo نمایش داده خواهد شد. در قسمت نام، عبارت Customer را وارد کرده و سپس بر روی دکمه Add کلیک کنید تا آیتم جدیدی به برنامه اضافه شود.
۳) بعد از اینکه صفحه ای که جدیداً ایجاد کرده اید نمایش داده شد، کدهای داخل آن را پاک کنید و کدهای زیر را به جای آن وارد کنید:

```
public struct Customer
{
    // Public members
    public string FirstName;
    public string LastName;
    public string Email;
}
```

نکته: مطمئن شوید که بخش تعریف کلاس، با تعریف ساختار با کلمه ی کلیدی `struct` جایگزین شده است.

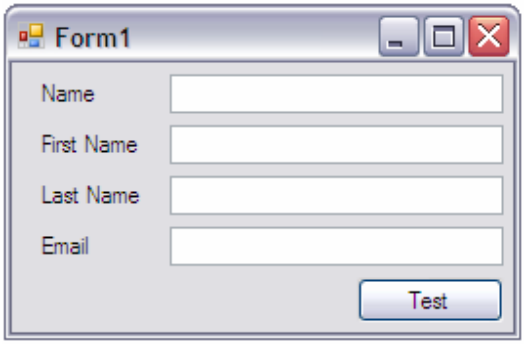
۴) به بخش طراحی فرم مربوط به `Form1` برگردید. چهار کنترل `Label`، چهار کنترل `TextBox` و یک کنترل `Button` بر روی فرم قرار دهید. این کنترل ها را به نحوی بر روی فرم قرار دهید که فرم شما مشابه شکل ۵-۱۲ شود.

۵) خاصیت `Name` کنترل ها را به صورت زیر تغییر دهید:

- `label1` را به **`lblName`** تغییر دهید.
- `textBox1` را به **`txtName`** تغییر دهید.
- `Label2` را به **`lblFirstName`** تغییر دهید.
- `textBox2` را به **`txtFirstName`** تغییر دهید.
- `Label3` را به **`lblLastName`** تغییر دهید.
- `textBox3` را به **`txtLastName`** تغییر دهید.
- `Label4` را به **`lblEmail`** تغییر دهید.
- `textBox4` را به **`txtEmail`** تغییر دهید.
- `Button1` را به تغییر دهید.

۶) خاصیت `Text` کنترلهای زیر را به مقادیر مشخص شده تغییر دهید:

- کنترل `lblName` را به `Name` تغییر دهید.
- کنترل `lblFirstName` را به `First Name` تغییر دهید.
- کنترل `lblLastName` را به `Last Name` تغییر دهید.
- کنترل `lblEmail` را به `Email` تغییر دهید.
- کنترل `btnTest` را به `Test` تغییر دهید.



شکل ۵-۱۲

۷) بر روی کنترل `Button` دو بار کلیک کنید تا متد رویداد `Click` این کنترل ایجاد شود، سپس کدهای مشخص شده در زیر را در این متد وارد کنید:

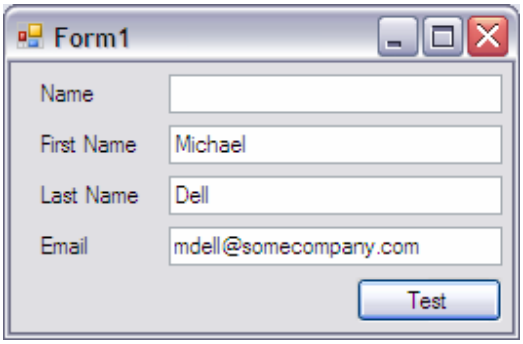
```
private void btnTest_Click(object sender, EventArgs e)
{
     // Create a new customer
     Customer objCustomer;
     objCustomer.FirstName = "Michael";
     objCustomer.LastName = "Dell";
     objCustomer.Email = "mdell@somecompany.com";

     // Display the customer
     DisplayCustomer(objCustomer);
}
```

۸) سپس زیربرنامه زیر را در کلاس `Form1` وارد کنید:

```
private void DisplayCustomer(Customer objCustomer)
{
     // Display the customer details on the form
     txtFirstName.Text = objCustomer.FirstName;
     txtLastName.Text = objCustomer.LastName;
     txtEmail.Text = objCustomer.Email;
}
```

۹) برنامه را اجرا کرده و بر روی دکمه ی `Test` کلیک کنید. نتیجه ای مشابه شکل ۵-۱۳ را مشاهده خواهید کرد.



شکل ۵-۱۳

چگونه کار می کند؟

برای تعریف یک ساختار باید از کلمه کلیدی `struct` در C# استفاده کنید. درون بلاک این ساختار باید متغیرهایی که می خواهید در این گروه باشند را به همراه اسم و نوع داده ای آنها مشخص کنید. به این متغییر ها عضو های این ساختار می گویند.

```
public struct Customer
{
    // Public members
    public string FirstName;
    public string LastName;
    public string Email;
}
```

به کلمه کلیدی `public` در مقابل هر یک از متغییر ها و همچنین قبل از خود تعریف ساختار توجه کنید. در برنامه های قبلی، کلمه `private` نیز که به همین ترتیب به کار می رفت را دیده بودید. کلمه `public` به این معنی است که شما در کدهای خارج از ساختار `Customer` هم میتوانید به متغیرهای آن (مانند `FirstName`) دسترسی داشته باشید و مقدار آن را تغییر دهید.
در داخل متد `btnTest_Click` متغیری را از نوع داده ای `Customer` تعریف می کنید. اگر ساختار `Customer` را از نوع کلاس تعریف می کردید، در این قسمت باید مقدار اولیه آن را نیز با استفاده از کلمه کلیدی `new` مشخص می کردید. نحوه این کار در فصل ۱۰ توضیح داده شده است.

```
private void btnTest_Click(object sender, EventArgs e)
{
    // Create a new customer
    Customer objCustomer;
```

بعد از تعریف این متغیر از نوع `Customer`، می توانید به هر یک از متغیرهای موجود در این ساختار با استفاده از `objCustomer` دسترسی پیدا کنید. برای این کار هنگامی که نام `objCustomer` را وارد کردید یک نقطه نیز قرار دهید. به این ترتیب تمام اعضای این ساختار به وسیله ویژوال استودیو نمایش داده خواهند شد و می توانید مقدار هر یک از آنها را تغییر دهید.

```
        objCustomer.FirstName = "Michael";
        objCustomer.LastName = "Dell";
        objCustomer.Email = "mdell@somecompany.com";

        // Display the customer
        DisplayCustomer(objCustomer);
}
```

در انتهای برنامه هم متد DisplayCustomer را می نویسیم. وظیفه این متد این است که یک ساختار از نوع Customer را به عنوان پارامتر ورودی دریافت کند و سپس با استفاده از مقادیر وارد شده در اعضای این ساختار خاصیت Text هر کدام از TextBoxهای روی فرم را تنظیم کند.

```
private void DisplayCustomer(Customer objCustomer)
{
        // Display the customer details on the form
        txtFirstName.Text = objCustomer.FirstName;
        txtLastName.Text = objCustomer.LastName;
        txtEmail.Text = objCustomer.Email;
}
```

اضافه کردن خاصیت به ساختارها:

هنگام تعریف یک ساختار، علاوه بر متغیر می توانید خاصیت نیز برای آن تعریف کنید. برای اضافه کردن خاصیت می توانید از همان روشی که در برنامه Enum Demo به کار بردیم استفاده کنید. در امتحان کنید زیر چگونگی این کار را مشاهده خواهیم کرد.

امتحان کنید: اضافه کردن خاصیت Name

۱) ویرایشگر کد را برای ساختار Customer باز کنید و سپس کد زیر را به این ساختار اضافه کنید تا یک خاصیت فقط-خواندنی به این ساختار اضافه شود:

```
// Public members
public string FirstName;
public string LastName;
public string Email;

// Name Property
public string Name
{
        get
        {
```

```
            return FirstName + " " + LastName;
        }
}
```

نکته: خاصیتهای یک کلاس و یا یک ساختار به سه صورت می توانند تعریف شوند: یا فقط–خواندنی باشند، یا فقط–نوشتنی باشند و یا خواندنی–نوشتنی باشند. برای این که بتوان مقدار یک خاصیت را تنظیم کرد باید بخش `set` را در آن قرار داد. برای این که بتوان مقدار کنونی آن را بدست آورد باید بخش `get` را در آن وارد کرد. اگر یک خاصیت دارای هر دو بخش باشد، خاصیت از نوع خواندنی–نوشتنی خواهد بود. در این قسمت برای این که خاصیت فقط–خواندنی باشد، قسمت `set` را برای آن قرار ندادیم و فقط بخش `get` در آن نوشتیم.

۲) اکنون ویرایشگر کد را برای `Form1` باز کنید و کد مشخص شده در زیر را به متد `DisplayCustomer` اضافه کنید:

```
private void DisplayCustomer(Customer objCustomer)
{
      // Display the customer details on the form
      txtName.Text = objCustomer.Name;
      txtFirstName.Text = objCustomer.FirstName;
      txtLastName.Text = objCustomer.LastName;
      txtEmail.Text = objCustomer.Email;
}
```

۳) برنامه را اجرا کرده و بر روی دکمه ی `Test` کلیک کنید. مشاهده خواهید کرد کادر `Name` که در قسمت قبل (شکل ۵–۱۳) خالی می ماند، حالا با نام و نام خانوادگی مشترک پر می شود.

کار با لیست های پیوندی:

فرض کنید که میخواهید لیستی از اطلاعات تمام مشترکین خود داشته باشید. در این حالت می توانید از آرایه ها استفاده کنید، اما همیشه هم کار با آرایه ها چندان ساده نیست.

- اگر نیاز داشته باشید که یک مشترک جدید را به آرایه اضافه کنید، باید اندازه آرایه را تغییر دهید و سپس مشترک را به انتهای آرایه اضافه کنید. برای این کار باید آرایه ای جدید که یک واحد بزرگتر از آرایه کنونی است ایجاد کنید و سپس تمام عناصر آرایه کنونی را به آرایه جدید منتقل کنید و مشترک جدید را نیز به آرایه اضافه کنید. در انتها نیز آرایه اول را از بین ببرید.
- اگر بخواهید یک مشترک را از آرایه حذف کنید باید به صورت خطی در تمام عناصر آرایه به دنبال مشترک بگردید و پس از پیدا کردن مکان آن، تمام عناصر این آرایه را به جز عنصری که می خواهید حذف کنید، در یک آرایه جدید قرار دهید و آرایه کنونی را از بین ببرید.
- برای این که یکی از مشترکین لیست را با مشترک دیگری تعویض کنید، باید مشترک اول را در آرایه پیدا کنید و سپس به صورت دستی مشترک اول را با مشترک دوم جا به جا کنید.

با استفاده از لیست های پیوندی در .NET که به وسیله کلاس ArrayList قابل دسترسی هستند می توانید بـه راحتـی در طول برنامه آرایه ها را کنترل کنید.

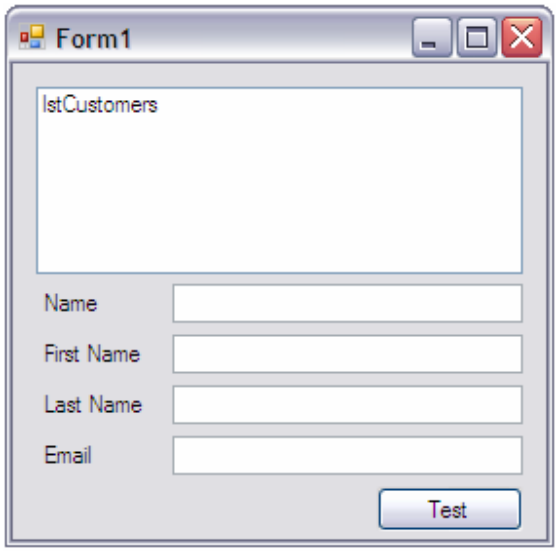
استفاده از لیست های پیوندی:

نحوه استفاده از لیست های پیوندی در امتحان کنید زیر شرح داده شده است.

امتحان کنید: استفاده از لیست های پیوندی

۱) به قسمت طراحی فرم برگردید و یک کنترل ListBox را به فرم اضافه کنید. مکـان کنتـرل هـای روی فـرم را بـه نحـوی تغییـر دهیـد کـه فـرم شـما مـشابه شـکل ۵-۱۴ شـود. خاصـیت Name ایـن ListBox را برابـر بـا lstCustomers و خاصیت IntegralHeight آن را برابر با False قرار دهید.

نکته: می توانید با استفاده از کلید Ctrl+A تمام کنترل های روی فرم را انتخاب کنید سپس آنها را به موقعیت جدیدشان ببرید.



شکل ۵-۱۴

۲) ویرایشگر کد را برای Form1 باز کرده و کد مشخص شده در زیر را به ابتدای کلاس Form1، بعد از تعریف کلاس اضافه کنید:

```
public partial class Form1 : Form
{
    // Form level members
```

```
    private ArrayList objCustomers = new ArrayList();
```

نکته: اگر هنگام نوشتن این کد، ویژوال استودیو نام کلاس `ArrayList` را کامل نکرد، به عبارت دیگر این کلاس را جز کلاسهای تعریف شده نداشت بایستی فضای نام آن را به برنامه خود اضافه کنید. این کلاس در فضای نام `System.Collection` قرار دارد. یک فضای نام با استفاده از کلمه کلیدی `using` به برنامه اضافه می شود. برای اضافه کردن فضای نام `System.Collection`، به بالاترین خط در قسمت کدهای مربوط به `Form1` بروید و کد زیر را وارد کنید:

```
using System.Collections;
```

در فصل نهم در رابطه با کلمه کلیدی `using` بیشتر صحبت خواهیم کرد.

۳) حال متد زیر را برای ایجاد یک مشترک جدید به برنامه اضافه کنید:

```
public void CreateCustomer(string FirstName,
                    string LastName, string Email)
{
    // Declare a customer object
    Customer objNewCustomer;

    // Create the new customer
    objNewCustomer.FirstName = FirstName;
    objNewCustomer.LastName = LastName;
    objNewCustomer.Email = Email;

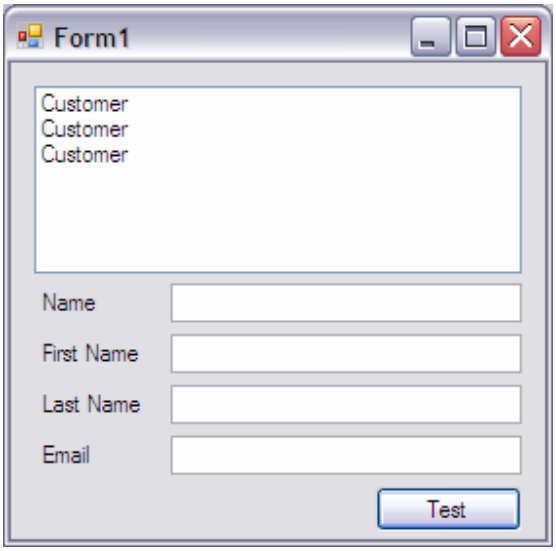
    // Add the new customer to the list
    objCustomers.Add(objNewCustomer);

    // Add the new customer to the ListBox control
    lstCustomers.Items.Add(objNewCustomer);
}
```

۴) سپس متد `btnTest_Click` را به صورت زیر تغییر دهید:

```
private void btnTest_Click(object sender, EventArgs e)
{
    // Create some customers
    CreateCustomer("Darrel", "Hilton",
                   "dhilton@somecompany.com");
    CreateCustomer("Frank", "Peoples",
                   "fpeoples@somecompany.com");
    CreateCustomer("Bill", "Scott",
                   "bscott@somecompany.com");
}
```

۵) برنامه را اجرا کرده و بر روی دکمه ی Test کلیک کنید نتیجه ای مشابه شکل ۵-۱۵ را مشاهده خواهید کرد.



شکل ۵-۱۵

شما چندین متغیر از ساختار Customer را به لیست اضافه کردید، اما چیزی که به وسیله کنترل ListBox نمایش داده می شود چندین عبارت Customer است. کنترل ListBox فقط می تواند مقادیر رشته ای را به لیست خود اضافه کند. هنگامی که یک متغیر از ساختار Customer را به این کنترل می فرستید، ویژوال C# متد ToString() را برای این متغییر ها فراخوانی می کند. به صورت پیش فرض، این متد نام ساختار و یا کلاس را برمی گرداند نه محتویاتی از آن ساختار را که شما می خواهید. کاری که در این مرحله باید انجام دهید این است که متد ToString() را به نحوی تغییر دهید که عبارت با معنی تری را برگرداند. چگونگی این کار را در امتحان کنید بعد مشاهده خواهیم کرد.

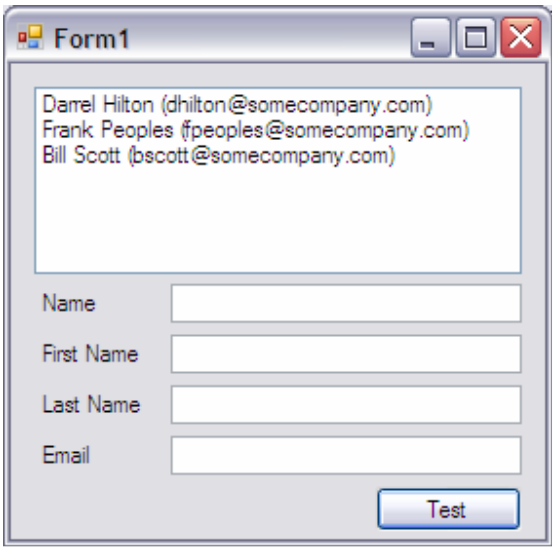
امتحان کنید: بازنویسی متد ToString()

۱) قسمت ویرایشگر کد برای ساختار Customer را باز کرده و کد زیر در این ساختار، بعد از تعریف متغیر ها قرار دهید. همانطور که در کد مشاهده می کنید، این تابع دارای XML Document Comment است. از فصل سه به خاطر دارید که برای اضافه کردن این نوع توضیحات به برنامه باید از سه کاراکتر / متوالی قبل از متد استفاده کنید.

```
/// <summary>
/// Overrides the default ToString method
/// </summary>
/// <returns>String</returns>
/// <remarks>Returns the customer name and email
address</remarks>
public override string ToString()
{
```

```
    return Name + " (" + Email + ") ";
}
```

۲) برنامه را اجرا کرده و بر روی دکمه ی Test کلیک کنید. نتیجه ای را مشابه شکل ۵-۱۶ مشاهده خواهید کرد.



شکل ۵-۱۶

چگونه کار می کند؟

هنگامی که یک متغیر از نوع داده ای Customer به لیست اضافه شود، کنترل ListBox تابع ToSring این متغیر را فراخوانی کرده و رشته ای که به وسیله این تابع برگردانده می شود را دریافت می کند. در این کد، متد ToString را به صورتی بازنویسی کردیم که به جای برگرداندن نام خود ساختار، یک عبارت با معنی را نمایش دهد.

```
/// <summary>
/// Overrides the default ToString method
/// </summary>
/// <returns>String</returns>
/// <remarks>Returns the customer name and email
address</remarks>
public override string ToString()
{
    return Name + " (" + Email + ") ";
}
```

نکته: به این عمل override کردن متدها گفته می شود که در فصل نهم بیشتر با آن آشنا می شویم.

یک لیست پیوندی که به وسیله ArrayList ایجاد می شود، می تواند لیستی از اشیا و یا ساختارها، از هر نوعی که باشند، را در خود نگهداری کند. به عبارت دیگر می توانید اشیایی از نوع های گوناگون را در این لیست ذخیره کنید (این مورد مقداری جلوتر در این فصل بیشتر توضیح داده شده). در این مثال متدی به نام CreateCustomer ایجاد کردیم که بر اساس پارامترهایی که به آن فرستاده شده بود، یک متغیر از نوع Customer را ایجاد می کرد:

```
public void CreateCustomer(string FirstName,
                    string LastName, string Email)
{
     // Declare a customer object
     Customer objNewCustomer;

     // Create the new customer
     objNewCustomer.FirstName = FirstName;
     objNewCustomer.LastName = LastName;
     objNewCustomer.Email = Email;
```

هنگامی که متغیر مورد نظرمان را ایجاد کردیم، آن را به لیست پیوندی که از کلاس ArrayList به نام objCustomers ایجاد کرده بودیم اضافه می کنیم.

```
     // Add the new customer to the list
     objCustomers.Add(objNewCustomer);
```

همچنین متغیر ایجاد شده را به کنترل لیست باکس نیز اضافه می کنیم تا آن را نمایش دهد:

```
     // Add the new customer to the ListBox control
     lstCustomers.Items.Add(objNewCustomer);
}
```

هنگامی که متد CreateCustomer را تعریف کردید، می توانید با فراخوانی آن یک مشترک جدید تعریف کرده و آن را به لیست باکس نیز اضافه کنید:

```
private void btnTest_Click(object sender, EventArgs e)
{
     // Create some customers
     CreateCustomer("Darrel", "Hilton",
                    "dhilton@somecompany.com");
     CreateCustomer("Frank", "Peoples",
                    "fpeoples@somecompany.com");
     CreateCustomer("Bill", "Scott",
                    "bscott@somecompany.com");
}
```

حذف یک عنصر از لیست های پیوندی:

تاکنون با مبانی کار با لیست های پیوندی آشنا شدید. در برنامه های قبلی با استفاده از این نوع لیست ها، کارهایی که انجام آنها با آرایه ها بسیار مشکل بود را به راحتی انجام دادیم. برای مثال توانستیم چندین عنصر جدید را به راحتی به این لیست اضافه کنیم. در این بخش چگونگی پاک کردن بعضی از عناصر یک لیست را بررسی خواهیم کرد.

امتحان کنید: پاک کردن مشترکین

۱) با استفاده از قسمت طراحی فرم، یک کنترل `Button` جدید به قسمت پایین فرم اضافه کنید. سپس خاصیت `Name` آن را به `btnDelete` و خاصیت `Text` آن را به `Delete` تغییر دهید.

۲) بر روی این کنترل دو بار کلیک کنید و کد مشخص شده در زیر را در متد مربوط به رویداد `Click` آن وارد کنید:

```
private void btnDelete_Click(object sender, EventArgs e)
{
     // If no customer is selected in the ListBox then...
     if (lstCustomers.SelectedIndex == -1)
     {
          // Display a message
          MessageBox.Show("You must select a customer to "
+                         "delete!", "Structure Demo");

          // Exit the method
          return;
     }

     // Prompt the user to delete the selected customer
     DialogResult result = MessageBox.Show(
          "Are you sure you want to delete " +
          SelectedCustomer.Name + "? ", "Structure Demo",
          MessageBoxButtons.YesNo,
MessageBoxIcon.Question);
     if (result == DialogResult.Yes)
     {
          // Get the customer to be deleted
          Customer objCustomerToDelete = SelectedCustomer;

          // Remove the customer from the ArrayList
          objCustomers.Remove(objCustomerToDelete);

          // Remove the customer from the ListBox
          lstCustomers.Items.Remove(objCustomerToDelete);
     }
}
```

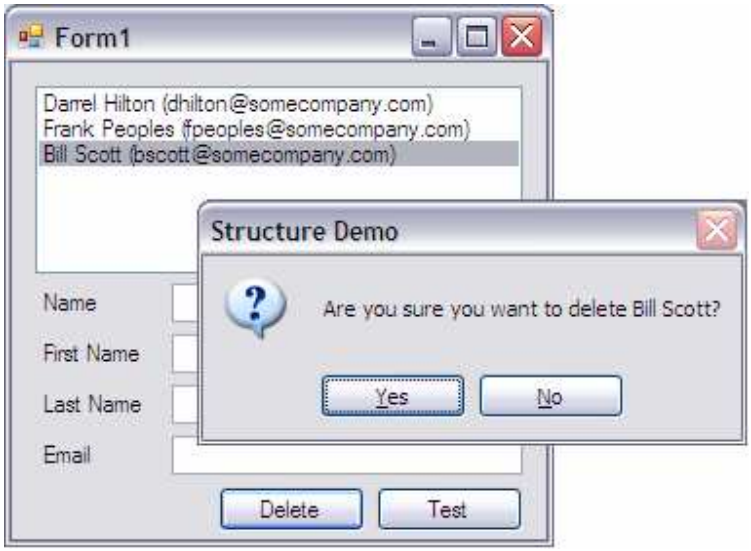
۳) سپس خاصیت `SelectedCustomer` که در کد بالا استفاده شده است را به صورت زیر به کلاس `Form1` اضافه کنید.

```
public Customer SelectedCustomer
{
      get
      {
            // Return the selected customer
            return (Customer)lstCustomers.Items[
                        lstCustomers.SelectedIndex];
      }
}
```

۴) برنامه را اجرا کرده و بر روی دکمه ی `Test` کلیک کنید. بدون اینکه مشترکی را از لیست انتخاب کنید، بر روی دکمه `Delete` کلیک کنید. کادر پیغامی را مشاهده خواهید کرد که می گوید باید یک مشترک را از لیست انتخاب کنید تا بتوانید آن را حذف کنید.

۵) حال یک مشترک را انتخاب کنید و بر روی دکمه ی `Delete` کلیک کنید. کادر پیغامی را مشاهده خواهید کرد که برای حذف مشترک از شما سوال می کند؟ (شکل ۵-۱۷)

۶) بر روی گزینه `Yes` کلیک کنید. مشاهده خواهید کرد که گزینه انتخابی شما از لیست پاک می شود.



شکل ۵-۱۷

چگونه کار می کند؟

یکی از نکته های این برنامه ایجاد خاصیتی بود کـه مـشترک انتخـاب شـده در کنتـرل ListBox را برمـی گردانـد. خاصـیت SelectedIndex در کنترل ListBox، اندیس عنصر انتخاب شده در لیست را برمی گرداند. اما اگر هیچ عنصری در لیست انتخاب نشده باشد، این تابع عدد ۱- را نتیجه می دهد.

```
public Customer SelectedCustomer
{
      get
      {
            // Return the selected customer
            return (Customer)lstCustomers.Items[
                        lstCustomers.SelectedIndex];
      }
}
```

همانند خاصیت Name در ساختار Customer، این خاصیت نیز از نوع فقط-خواندنی ایجاد شده است. همانطور کـه مـشاهده می کنید، این خاصیت فقط دارای بلاک get است و بلاک set ندارد. به همین دلیل در برنامه فقط می توانیم به مقـدار کنـونی این خاصیت دسترسی پیدا کنیم و نمی توانیم مقدار آن را تنظیم کنیم.
درون کنترل کننده ی رویداد Click برای دکمه Delete ابتدا بررسی می کنیم که آیا مشترکی از لیست انتخاب شده اسـت یا نه؟ در صورتی که هیچ فردی از لیست انتخاب نشده بود، با نمایش یک کادر پیغام به کاربر می گـوییم کـه بایـد یـک مـشترک را انتخاب کند تا بتواند آن را حذف کند. سپس از متد خارج می شویم و به کاربر اجازه می دهیم که فردی را از لیست انتخاب کند.

```
      // If no customer is selected in the ListBox then...
      if (lstCustomers.SelectedIndex == -1)
      {
            // Display a message
            MessageBox.Show("You must select a customer to "
+                           "delete!", "Structure Demo");

            // Exit the method
            return;
      }
```

اگر کاربر مشترکی را از لیست انتخاب کرده بود، نام او را در یک کادر پیغام نمایش می دهید تا مطمئن شوید کاربر می خواهـد آن را از لیست حذف کند.

```
      DialogResult result = MessageBox.Show(
            "Are you sure you want to delete " +
            SelectedCustomer.Name + "? ", "Structure Demo",
            MessageBoxButtons.YesNo,
MessageBoxIcon.Question);
      if (result == DialogResult.Yes)
      {
```

نحوه استفاده از تابع MessageBox.Show در این قسمت، مقداری با دفعات قبل تفاوت دارد. همانطور که پیش تر نیز گفتم[1] یک تابع می تواند در حالی که فقط یک نام دارد، پارامترهای مختلفی را به عنوان ورودی دریافت کند. اگر در هنگام کار با این تابع به راهنمایی که ویژوال استودیو درباره این تابع نمایش می دهد دقت کنید، متوجه خواهید شد که این تابع به چندین روش می تواند مورد استفاده قرار گیرد. یکی از این روشها که در قسمتهای قبلی نیز از آن استفاده می کردیم، یک رشته را به عنوان متن اصلی و رشته دیگری را نیز برای نمایش در عنوان پنجره دریافت می کرد و کادر پیغامی را بر اساس این اطلاعات نمایش می داد.
در این برنامه از یکی دیگر از حالتهای تابع MessageBox.Show استفاده کرده ایم. این حالت علاوه بر پارامترهای قبلی، دکمه های فرمان و آیکونی که باید در کادر نمایش دهد را نیز دریافت می کند. دکمه های فرمان باید از شمارنده ی MessageBoxButtons و آیکون باید از شمارنده ی MessageBoxIcon انتخاب شود. در اینجا علاوه بر مشخص کردن متنهایی که باید در کادر نمایش داده شود، به ویژوال C# گفته ایم که به جای دکمه OK، دکمه های Yes و No را به همراه یک علامت سوال بر روی کادر نمایش دهد. برای این که بفهمیم کاربر کدام گزینه را انتخاب کرده است، باید از نتیجه ای که توسط تابع برگردانده می شود استفاده کنیم. این تابع نتیجه خود را به صورت شیئ از کلاس DialogResult برمی گرداند. به همین دلیل ابتدا متغیری از این کلاس ایجاد می کنیم، سپس نتیجه برگردانده شده توسط تابع را در آن قرار می دهیم.
حال باید بررسی کنیم که کاربر دکمه Yes را فشرده است یا نه؟ برای این کار از دستور if استفاده می کنیم. اگر مقدار متغیری که از نوع DialogResult تعریف کردیم برابر با DialogResult.Yes باشد، به این معنی است که کاربر گزینه Yes را انتخاب کرده است. در غیر این صورت کاربر از حذف مشترک منصرف شده است و گزینه No را انتخاب کرده است[2].
برای حذف کاربر از لیست، متغیری را از نوع ساختار Customer تعریف می کنیم و مشترکی را که می خواهیم از لیست حذف کنیم در آن قرار می دهیم:

```
// Get the customer to be deleted
Customer objCustomerToDelete = SelectedCustomer;
```

با استفاده از متد Remove کلاس ArrayList می توانیم مشترک انتخاب شده را از لیست حذف کنیم. برای این کار باید متغیری را که در مرحله قبل، مشترک را در آن ذخیره کردیم به عنوان پارامتر به متد بفرستیم:

```
// Remove the customer from the ArrayList
objCustomers.Remove(objCustomerToDelete);
```

برای حذف کاربر از لیست باکس هم باید از روشی مشابه استفاده کنید:

```
// Remove the customer from the ListBox
lstCustomers.Items.Remove(objCustomerToDelete);
```

نمایش عناصر موجود در لیست پیوندی:

[1] رجوع کنید به بخش مرتب سازی آرایه ها در همین فصل
[2] در فصل هفتم در مورد استفاده از کادر پیغام به تفصیل بحث خواهیم کرد.

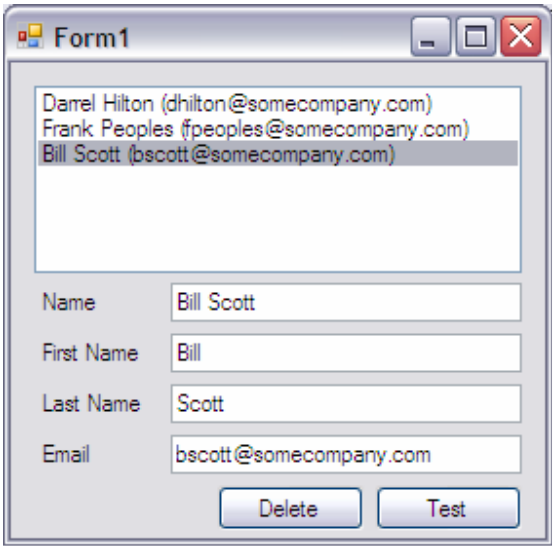
برای کامل شدن برنامه، باید یک سری از قابلیت ها را به آن اضافه کنید تا رابط کاربری برنامه بهبود پیدا کند. در بخش امتحان کنید بعدی، کد درون رویداد `SelectedIndexChanged` مربوط به `ListBox` را به گونه ای تغییر خواهید داد تا هر بار که مشترک جدیدی را از `ListBox` انتخاب کنید، اطلاعات او بر روی صفحه نمایش داده شود.

امتحان کنید: نمایش اطلاعات مشترک انتخاب شده در صفحه

۱) در بخش طراحی فرم بر روی لیست باکس دو بار کلیک کنید. به این ترتیب متد مربوط به رویداد `SelectedIndexChanged` به طور اتوماتیک ایجاد می شود. کد مشخص شده در زیر را به این متد اضافه کنید:

```
private void lstCustomers_SelectedIndexChanged(
                    object sender, EventArgs e)
{
     // Display the customer details
     DisplayCustomer(SelectedCustomer);
}
```

۲) برنامه را اجرا کنید و بر روی دکمه ی فرمان `Test` کلیک کنید تا آیتم های مورد نیاز در `ListBox` قرار گیرند. حال اگر بر روی نام یکی از مشترکین در لیست باکس کلیک کنید، همانند شکل ۵-۱۸ اطلاعات او در کادرهای متنی روی فرم نمایش داده می شوند.



شکل ۵-۱۸

چگونه کار می کند؟

ArrayList یکی از انواع کلکسیون های موجود در .NET است که از آن استفاده زیادی شده است. یک کلکسیون راهی است برای ساختن راحت گروه هایی از عناصر مرتبط به هم. اگر به برنامه ی Structure Demo نگاهی بیندازید و کد تابع CreateCustomer را مشاهده کنید، متوجه می شوید که برای اضافه کردن یک مشترک به لیست، همانند اضافه کردن یک مشترک به لیست پیوندی، از تابع Add استفاده می کردید.

```
    // Add the new customer to the list
    objCustomers.Add(objNewCustomer);

    // Add the new customer to the ListBox control
    lstCustomers.Items.Add(objNewCustomer);
```

برای حذف یک مشترک هم (چه در لیست باکس و چه در لیست پیوندی) از متد Remove استفاده کردید:

```
    // Remove the customer from the ArrayList
    objCustomers.Remove(objCustomerToDelete);

    // Remove the customer from the ListBox
    lstCustomers.Items.Remove(objCustomerToDelete);
```

پیشنهاد می شود که هنگام برنامه نویسی اگر می خواهید لیستی از عناصر مرتبط به هم را نگهداری کنید، از کلکسیون ها استفاده کنید. در زبانهای برنامه نویسی تحت .NET و همچنین در کلاسهای موجود در کتابخانه کلاس .NET، مایکروسافت سعی کرده است تمام کلکسیون ها صرفنظر از نوع مقداری که باید ذخیره کنند، به یک روش کار کنند. به همین دلیل است که در برنامه هر جا که بخواهید عنصری را حذف کنید از متد Remove و اگر بخواهید عنصری را اضافه کنید از متد Add استفاده می کنید، چه در حال کار با یک ArrayList باشید، چه در حال کار با عناصر موجود در یک لیست باکس.
این ثبات و یکسانی در توابع و کلاسها به برنامه نویس اجازه می دهد تا بتواند آموخته های خود را از یک موضوع، برای کار با موضوعات مشابه نیز به کار ببرد. پس هنگامی که می خواهید برای برنامه های خود ساختارهای داده ای تعریف کنید، بهتر است این قواعد را نیز رعایت کنید. برای مثال اگر در حال ایجاد یک کلاس همانند کلکسیون ها هستید و می خواهید که متدی برای حذف عناصر در آن تعریف کنید بهتر است نام آن را برابر با Remove قرار دهید، نه عبارتهای مشابه مانند Delete و یا .... به این ترتیب اگر برنامه نویسی بخواهد از کلاس شما استفاده کند، نام Remove برای او آشنا خواهد بود و می تواند عملکرد این متد را حدس بزند.

ایجاد جداول قابل جستجو با Hashtableها:

تاکنون اگر می خواستید عنصری را در یک آرایه یا یک لیست پیوندی پیدا کنید، باید اندیس عدد صحیح آن عنصر را که معرف مکان قرار گرفتن آن عنصر بود مشخص می کردید. اما اگر می خواستید برای دسترسی به آن عنصر از مقدار دیگری به جز اندیس استفاده کنید، نمی توانستید این روش را به کار ببرید. برای مثال فرض کنید در برنامه قبل می خواستید اطلاعات مشترکین را بر اساس آدرس پست الکترونیکی آنها بدست آورید.
در این قسمت نوع خاصی از کلکسیون ها به نام Hashtable را بررسی خواهیم کرد که روشهای بهتری را برای جستجو ارائه می دهند. این کلکسیون ها بر اساس یک مقدار کلیدی که برای آنها مشخص می شود، آرایه را جستجو می کنند.

استفاده از Hashtable:

Hashtable نوعی کلکسیون است که هر عنصر آن دارای یک کلید است. با استفاده از این کلید می توانید به مقدار عنصر در کلکسیون دسترسی پیدا کنید. برای مثال فرض کنید اطلاعات مشترکی با نام Darrel را که از نوع Customer است در یک Hashtable قرار می دهید و مقدار کلیدی این عنصر را نیز برابر با آدرس پست الکترونیکی او مشخص می کنید. در این صورت اگر برای دسترسی به اطلاعات این مشترک آدرس پست الکترونیکی او را وارد کنید می توانید به سرعت او را در لیست پیدا کنید.
هنگامی که دو عنصر کلید و مقدار را به Hashtable اضافه می کنید، عنصر کلید تابعی به نام System.Object.GetHashCode() را فراخوانی می کند. این تابع یک عدد صحیح منحصر به فرد را برای کلید برمی گرداند که به عنوان شناسه ی آن استفاده می شود. این عدد به صورتی است که اگر چند بار دیگر هم تابع برای این عنصر فراخوانی شود عدد یکسانی برمی گردد. به این ترتیب هنگامی که بخواهید به عنصری در یک Hashtable دسترسی پیدا کنید، کلید آن عنصر از شما گرفته شده و تابع GetHashCode مربوط به آن کلید اجرا می شود. شناسه ای که به وسیله ی این تابع به دست می آید با تمام شناسه های موجود در Hashtable مقایسه می شود. اگر آن شناسه در لیست وجود داشته باشد، مقدار مرتبط به آن کلید برمی گردد.
جستجو در یک Hashtable بسیار سریع انجام می شود. زیرا صرفنظر از نوع عنصری که در این جدول ذخیره می کنید، یک عدد صحیح کوچک به عنوان شناسه عنصر در نظر گرفته می شود. در امتحان کنید بعد، نحوه استفاده از Hashtable ها را مشاهده خواهیم کرد.

نکته: عدد صحیحی که برای ذخیره شناسه ی یک کلید در Hashtable به کار می رود فقط ۴ بایت از حافظه را اشغال می کند. بنابراین اگر رشته ای شامل ۱۰۰ کاراکتر را که ۲۰۰ بایت فضا اشغال می کند به عنوان کلید در نظر بگیرید، برای جستجو در جدول فقط اعداد ۴ بایتی با یکدیگر مقایسه می شوند که باعث افزایش سرعت می شود.

امتحان کنید: استفاده از Hashtableها

۱) ویرایشگر کد را برای فرم باز کرده و تغییر زیر را در تعریف متغیر objCustomers ایجاد کنید:

```
public partial class Form1 : Form
{
     // Form level members
     private Hashtable objCustomers = new Hashtable();
```

۲) به قسمت کدهای مربوط به تابع CreateCustomer بروید و کدهای آن را به صورت مشخص شده در زیر تغییر دهید:

```
public void CreateCustomer(string FirstName, string
                           LastName, string Email)
```

```
{
      // Declare a customer object
      Customer objNewCustomer;

      // Create the new customer
      objNewCustomer.FirstName = FirstName;
      objNewCustomer.LastName = LastName;
      objNewCustomer.Email = Email;

      // Add the new customer to the list
      objCustomers.Add(Email.ToLower(),objNewCustomer);

      // Add the new customer to the ListBox control
      lstCustomers.Items.Add(objNewCustomer);
```

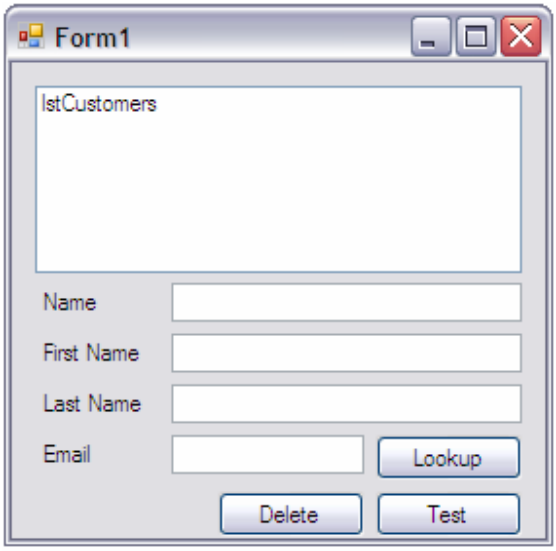
۳) به بخش کد مربوط به متد `btnDelete_Click` بروید و تغیرات مشخص شده در زیر را در آن ایجاد کنید:

```
// Prompt the user to delete the selected customer
DialogResult result = MessageBox.Show(
      "Are you sure you want to delete " +
      SelectedCustomer.Name    + "? ", "Structure Demo",
      MessageBoxButtons.YesNo,
      MessageBoxIcon.Question);
if (result == DialogResult.Yes)
{
      // Get the customer to be deleted
      Customer objCustomerToDelete = SelectedCustomer;

      // Remove the customer from the ArrayList
      objCustomers.Remove(txtEmail.Text.ToLower());

      // Remove the customer from the ListBox
      lstCustomers.Items.Remove(objCustomerToDelete);
}
```

۴) به قسمت طراحی فرم برگردید و کنترل `Button` جدیدی را به فرم اضافه کنید. خاصیت `Name` این کنترل را برابر با `btnLookup` و خاصیت `Text` آن را برابر با `Lookup` قرار دهید. فرم شما در این مرحله باید مشابه شکل ۵-۱۹ باشد.

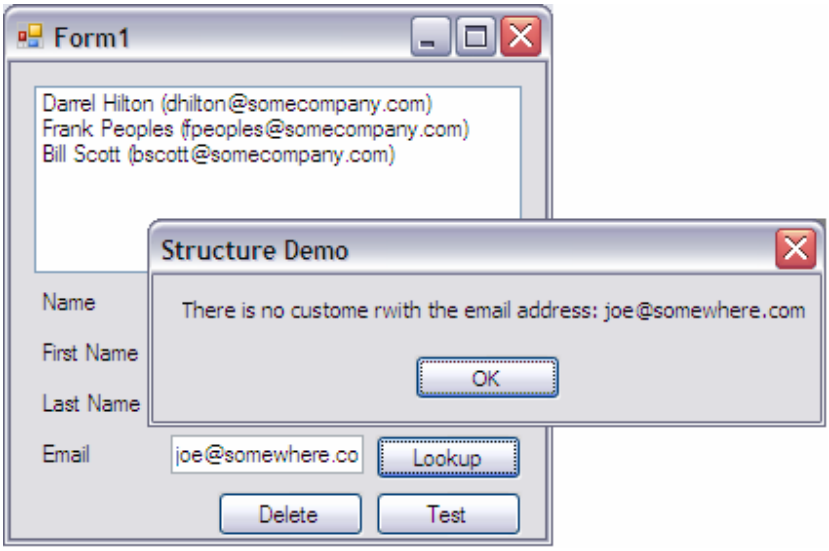


شکل ۵-۱۹

۵) بر روی دکمه ی Lookup دو بار کلیک کنید تا متد مربوط به رویداد Click آن ایجاد شود. سپس کدهای زیـر را به این متد اضافه کنید:

```
private void btnLookup_Click(object sender, EventArgs e)
{
     // If the customer found in the Hashtable
     if (objCustomers.Contains(txtEmail.Text.ToLower()) ==
          true)
          // Display the customer name
          MessageBox.Show("The customer name is: " +
     ((Customer)objCustomers[txtEmail.Text.ToLower()]).Name
,               "Structure Demo");
     else
          //Display an error
          MessageBox.Show("There is no custome rwith the "
+               "email address: " + txtEmail.Text,
               "Structure Demo");
}
```

۶) برنامه را اجرا کنید و بر روی دکمه Test کلیک کنیـد تـا لیـست از نـام مـشترکین پـر شـود. اگـر یـک آدرس پـست الکترونیکی که در لیست وجود ندارد را در بخش Email وارد کنید و بر روی دکمـه Lookup کلیـک کنیـد کـادر پیغامی را مشابه شکل ۵-۲۰ مشاهده خواهید کرد.



شکل ۵-۲۰

۷) اگر یک آدرس پست الکترونیکی که در لیست وجود دارد، برای مثال `dhilton@somecompany.com` را در قسمت `Email` وارد کنید و دکمه `Lookup` را فشار دهید مشاهده خواهید کرد که نام فرد در کادر پیغام نمایش داده می شود.

چگونه کار می کند؟

برای ایجاد یک `HashTable`، تعریف متغیر `objCustomers` را به صورت زیر تغییر می دهیم:

```
// Form level members
private Hashtable objCustomers = new Hashtable();
```

به این ترتیب نوع داده ای متغیر `objCustomers`، از نوع `HashTable` خواهد بود و عناصر ایجاد شده در متد `CreateCustomer` به جای ذخیره شدن در `ArrayList` در یک `HashTable` ذخیره می شوند.
بر خلاف متدهای `Add` که در برنامه های قبلی مشاهده کردید، متد `Add` مربوط به `HashTable` دو پارامتر دریافت می کند. اولین پارامتر عنصر کلید است، که در این برنامه از آدرس پست الکترونیکی به عنوان کلید استفاده کرده ایم. در اضافه کردن یک آیتم به `HashTable` از هر عنصری می توانید به عنوان کلید استفاده کنید، فقط باید دقت داشته باشید که این عنصر باید در آرایه منحصر به فرد باشد و نمی توانید از یک عنصر به عنوان کلید دو آیتم در `HashTable` استفاده کنید. در غیر این صورت برنامه هنگام اجرا با خطا مواجه شده و بسته می شود. پارامتر دوم این متد، آیتمی است که می خواهید همراه با این کلید در `HashTable` قرار دهید. به این ترتیب هر بار که این کلید را به `HashTable` بدهید، می توانید به این آیتم دسترسی پیدا کنید.

```
// Add the new customer to the list
objCustomers.Add(Email.ToLower(),objNewCustomer);
```

برای حذف یک آیتم از HashTable باید کلید آن را که همان آدرس پست الکترونیکی است مشخص کنید.

```
// Remove the customer from the ArrayList
objCustomers.Remove(txtEmail.Text.ToLower());
```

نکته: برای جلوگیری از حساسیت نسبت به نوع حروف هنگام اضافه و یا حذف کردن یک آیتم بـه HashTable، ابتـدا تمـام حروف آدرس پست الکترونیکی را با استفاده از تابع ToLower به حروف کوچک تبدیل می کنیم، سپس رشته جدید را به عنوان کلید آرایه در نظر می گیریم.

حال می خواهیم در متد مربوط به رویداد کلیک btnLookup کدی قرار دهیم کـه کـاربر بتوانـد بـا وارد کـردن آدرس پـست الکترونیکی، نام مشترک را مشاهده کند. برای این کار باید با استفاده از آدرس وارد شـده در HashTable بـه دنبـال مـشترک بگردیم. برای اطلاع از اینکه آیا آیتم مورد نظر ما در HashTable وجود دارد یـا نـه، بایـد از تـابع Contains اسـتفاده کنیم. این تابع یک کلید را به عنوان پارامتر می گیرد و مشخص می کند که آیتمی با کلید داده شده در لیست وجود دارد یا نـه؟ اگـر مقدار برگردانده شده توسط این تابع برابر با true بود، یعنی آیتم در لیست وجود دارد.

```
// If the customer found in the Hashtable
if (objCustomers.Contains(txtEmail.Text.ToLower()) ==
     true)
```

برای دسترسی به این آیتم می توانیم همانند دسترسی به عناصر در یک آرایه عمل کنیم و به جای استفاده از یـک عـدد صـحیح، از کلید که در اینجا همان آدرس پست الکترونیکی است به عنوان اندیس استفاده کنیم:

```
// Display the customer name
MessageBox.Show("The customer name is: " +
((Customer)objCustomers[txtEmail.Text.ToLower()]).Name,
          "Structure Demo");
```

تمام آیتم ها قبل از این که در HashTable ذخیره شوند، به شیئ از کلاس Object تبدیل می شوند. بنابراین بعد از این که با استفاده از کلید، مکان آنها را در آرایه پیدا کردیم، قبل از استفاده باید آنها را به نوع داده ای اصلی خود تبدیل کنیم. همانطور که در قبل گفتم[1] برای اینکه یک متغیر را به نوع داده ای دیگری تبدیل کنیم باید از عملگر ( ) استفاده کنـیم. در ایـن قـسمت بـرای تبدیل آیتم ذخیره شده در HashTable از نوع Object به نوع Customer به صورت زیر عمل می کنیم:

```
(Customer)objCustomers[txtEmail.Text.ToLower()]
```

حال می توانیم با استفاده از خاصیت Name نام مربوط به مشترک را در کادر پیغام نمایش دهیم.

[1] رجوع کنید به بخش تبدیل نوعهای داده ای

جلوگیری از وارد شدن عناصر تکراری:

همانطور که می دانید از یک کلید نمی توان دو بار در یک `HashTable` استفاده کرد. این کار باعث بـه وجـود آمـدن خطـا در زمان اجرای برنامه می شود. به همین دلیل باید قبل از اینکه عنصری را به یک `HashTable` اضافه کنیم، از منحصر بـه فـرد بودن آن مطمئن شویم. در بخش امتحان کنید بعد، چگونگی جلوگیری از این خطا در برنامه را خواهید دید.

امتحان کنید: جلوگیری از وارد شدن عناصر تکراری

۱) برای مشاهده این که در صورت وارد کردن کلید تکراری به `HashTable`، چگونه خطا ایجاد می شود برنامه را اجرا کنید و روی دکمه ی `Test` کلیک کنید تا لیست مشترکین پر شود. حال مجددا بر روی دکمه فرمان `Test` کلیـک کنید. پنجره خطایی را مشابه شکل ۵-۲۱ مشاهده خواهید کرد.



شکل ۵-۲۱

۲) بر روی دکمه ی `Stop Debugging` در نوار ابزار ویژوال استودیو ۲۰۰۵ کلیک کنید تا اجـرای برنامـه متوقـف شود.

۳) ویرایشگر کد را برای `Form1` باز کنید و به محل متد `CreateCustomer` بروید. کـد زیـر را بـه ایـن متـد اضافه کنید تا هر بار قبل از اینکه مشترک به لیست اضافه شود از عدم وجود آن در لیست مطمئن شویم:

```
public void CreateCustomer(string FirstName,
                           string LastName, string Email)
{
      // Declare a customer object
      Customer objNewCustomer;

      // Create the new customer
      objNewCustomer.FirstName = FirstName;
```

```
        objNewCustomer.LastName = LastName;
        objNewCustomer.Email = Email;

        // Check if the customer isn't currently in the list
        if (objCustomers.Contains(Email.ToLower()) == false)
        {
            // Add the new customer to the list
            objCustomers.Add(Email.ToLower(),
objNewCustomer);

            // Add the new customer to the ListBox control
            lstCustomers.Items.Add(objNewCustomer);
        }
        else
        {
            MessageBox.Show("The customer: " + FirstName + "
"               + LastName + " is currently in the list! ",
                "Structure Demo");
        }
}
```

۴) مجددا برنامه را اجرا کنید و بر روی دکمه ی Test کلیک کنید تا لیست مشترکین کامل شود.

۵) حال دوباره روی دکمه ی Test کلیک کنید. مشاهده خواهید کرد هنگامی که بخواهید مشترکی را برای بار دوم به لیست وارد کنید، در یک کادر پیغام نمایش داده می شود که مشترک هم اکنون در لیست وجود دارد و نمی توانید آن را مجددا به لیست اضافه کنید.

چگونه کار می کند؟

همانطور که مشاهده می کنید برای بار دوم که برنامه را اجرا می کنید با خطا مواجه نمی شوید. دلیل آن نیز مشخص است. زیرا در اجرای دوم برنامه فقط در صورتی یک آیتم به لیست اضافه می شود که کلید آن در جدول وجود نداشته باشد. در این صورت اگر کلید آیتم را به عنوان پارامتر به تابع Contains بفرستیم، تابع مقدار false را برگردانده و برنامه آیتم را به لیست اضافه می کند:

```
        // Check if the customer isn't currently in the list
        if (objCustomers.Contains(Email.ToLower()) == false)
        {
            // Add the new customer to the list
            objCustomers.Add(Email.ToLower(),
objNewCustomer);

            // Add the new customer to the ListBox control
            lstCustomers.Items.Add(objNewCustomer);
        }
```

اگر تابع `Contains` مقدار `true` را برگرداند، می توان نتیجه گرفت که آیتم در لیست وجود داشته است. پس با نمایش یک کادر پیغام این مورد را به اطلاع کاربر می رسانیم و از تابع خارج می شویم.

```
        else
        {
            MessageBox.Show("The customer: " + FirstName + "
"                   + LastName + " is currently in the list! ",
                    "Structure Demo");
        }
```

نتیجه:

در این فصل روشهایی را برای مدیریت و نگهداری گروه های پیچیده داده ای فرا گرفتیم. فصل را با آموختن مفاهیم آرایه شروع کرده و مشاهده کردیم که چگونه می توان متغیرهایی را ایجاد کرد که بیش از یک مقدار را در خود نگهداری کنند.
سپس به بررسی مفاهیم شمارنده ها و ثابت ها پرداختیم. دیدیم که چگونه این دو قابلیت باعث می شوند که کدهای خواناتری بنویسیم. با استفاده از شمارنده ها می توانیم به جای استفاده از متغیرهای ابتدایی، نوع های قابل فهم تری را استفاده کنیم، مثلا به جای استفاده از کد `"mode = 3"` می توانیم از کد `"mode = MyMode.Menu"` استفاده کنیم. ثابت ها این اجازه را می دهند که یک اسم خاص را به یک مقدار که در قسمتهای مختلف کد استفاده شده است نسبت دهیم و در کد از اسم مشخص شده استفاده کنیم.
سپس ساختارها را بررسی کردیم. دیدیم که ساختارها همانند کلاسها هستند و اجازه می دهند اطلاعات مربوط به یک فرد یا مورد خاص را در یک گروه قرار دهیم. سپس با انواع مختلف کلکسیون ها از قبیل `ArrayList` آشنا شدیم در آخر هم کلاس `HashTable` و فواید آن را بررسی کردیم و مشاهده کردیم که چگونه می توان با استفاده از آن کلکسیونی قویتر از `ArrayList`ها ایجاد کرد.

در پایان این فصل باید با موارد زیر آشنا شده باشید:

- تعریف آرایه ای از عناصر مرتبط به هم
- حرکت در بین عناصر آرایه و پیدا کردن آخرین عنصر آن
- تعریف یک شمارنده با استفاده از `enum`
- ایجاد و استفاده از ساختارها برای نگهداری داده های مرتبط به هم
- استفاده از `ArrayList` برای نگهداری انواع متغییر ها
- استفاده از کلکسیون ها برای نگهداری و مدیریت داده های مرتبط به هم

تمرین:

تمرین ۱:

یک برنامه تحت ویندوز ایجاد کنید که شامل سه کنترل `Button` باشد. یک شمارنده شامل سه نام را بـه برنامـه اضـافه کنیـد. برای متد مربوط به رویداد `Click` هر یک از دکمه ها کدی بنویسید که یک کادر پیغام محتوی یکی از نام ها و مقدار متناظر آن را در شمارنده نمایش دهد. (برای نمایش مقدار عددی متناظر با هر یک از آیتم های شمارنده، نوع آن را به عدد صحیح تبدیل کنید)

فصل ششم: ایجاد برنامه های ویندوزی

همانطور که در قسمتهای پیش نیز مشاهده کردید، در C# پنجره ها به نام **فرم** شناخته می شوند. در پنج فصل قبلی از این فرم ها در برنامه های خود استفاده می کردید، اما درک کاملی از آن نداشتید و بیشتر بر روی کدهایی که درون آن می نوشتید تمرکز می کردید.
در این فصل، بیشتر با جزئیات فرمهای ویندوزی آشنا خواهید شد و مشاهده خواهید کرد که چگونه می توان با استفاده از فرمهای ویندوزی در ویژوال C#، برنامه هایی با امکانات کامل نوشت. در این فصل به صورت کلی به بررسی مباحث زیر خواهیم پرداخت:

- اضافه کردن خصوصیات بیشتر به فرم با استفاده از کنترلهای Button، TextBox و RadioButton.
- ایجاد یک نوار ابزار ساده که دارای دکمه هایی برای پاسخ به رویدادها باشد.
- ایجاد برنامه های ویندوزی که دارای بیش از یک فرم باشند.

پاسخ به رویدادها:

ایجاد یک رابط گرافیکی کاربر[1] با استفاده از فرمهای ویندوزی تا حد زیادی به پاسخ دادن به رویدادها بستگی دارد. به همین علت برنامه نویسی برای ویندوز عموماً به عنوان **برنامه نویسی رویداد گرا** شناخته می شود. همانطور که در قسمتهای قبل مشاهده کردید، برای ایجاد یک فرم با استفاده از ماوس کنترل های مورد نظرتان را بر روی یک فرم خالی که در بخش Forms Designer است قرار می دهید. هر کدام از این کنترل ها می توانند به شما بگویند که چه زمانی یک رویداد اتفاق می افتد. برای مثال اگر یک برنامه را اجرا کنید و بر روی دکمه فرمانی که بر روی فرم قرار گرفته است کلیک کنید آن دکمه، رویداد کلیک را به برنامه اطلاع می دهد. بنابراین این فرصت به شما داده می شود که کدهایی را در برنامه مشخص کنید تا با کلیک شدن بر روی دکمه فرمان اجرا شوند. نحوه استفاده از این موارد را در قسمتهای قبلی مشاهده کردیم.

تنظیم یک رویداد برای کنترل Button:

یک روش خوب برای توضیح مفهوم رویداد، ایجاد یک رویداد برای دکمه فرمان است. برای مثال یک رویداد کلیک که کد مربوط به آن هنگام کلیک شدن بر روی دکمه فرمان اجرا شود. در برنامه های ویندوزی رویدادهای زیادی وجود دارند که هر یک در مواقع خاصی رخ می دهند. تاکنون استفاده از رویداد کلیک مربوط به دکمه فرمان را در عمل دیده اید.. در بخش امتحان کنید بعد، رویدادهای بیشتری از کنترل دکمه فرمان را که در قسمتهای قبلی مشاهده نکردید بررسی خواهیم کرد.

امتحان کنید: استفاده از رویدادهای دکمه فرمان

[1] رابط گرافیکی کاربر یا GUI، محیطی است که در آن کاربر می تواند با کلیک بر روی برنامه ها و یا دکمه های فرمان، به آنها دسترسی داشته باشد

۱) ویژوال استودیو را اجرا کنید و گزینه File ➔ New ➔ Project... را از منوی ویژوال استودیو انتخاب کنید. در پنجره New Project، گزینه Visual C# را از قسمت Project Types و گزینه Windows Application را از قسمت Templates انتخاب کنید. در قسمت Name، نام Hello World 2 را وارد کنید و سپس بر روی دکمه OK کلیک کنید تا پروژه ایجاد شود.

۲) بر روی فرم ایجاد شده کلیک کنید و سپس در پنجره Properties خاصیت Text را از Form1 به Hello, World! 2.0 تغییر دهید.

۳) با استفاده از جعبه ابزار یک کنترل Button بر روی فرم قرار داده، خاصیت Text آن را به Hello World! و خاصیت Name آن را به btnSayHello تغییر دهید. سپس اندازه فرم و کنترل Button را به گونه ای تغییر دهید که مشابه شکل ۱-۶ شود.

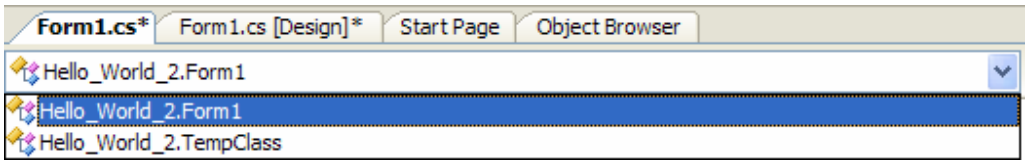


شکل ۶-۱

۴) بر روی کنترل Button کلیک کنید و کد مشخص شده در زیر را به متد مربوط به رویداد کلیک آن اضافه کنید.

```
private void btnSayHello_Click(object sender, EventArgs e)
{
    // Display a MessageBox
    MessageBox.Show("Hello World!", this.Text);
}
```

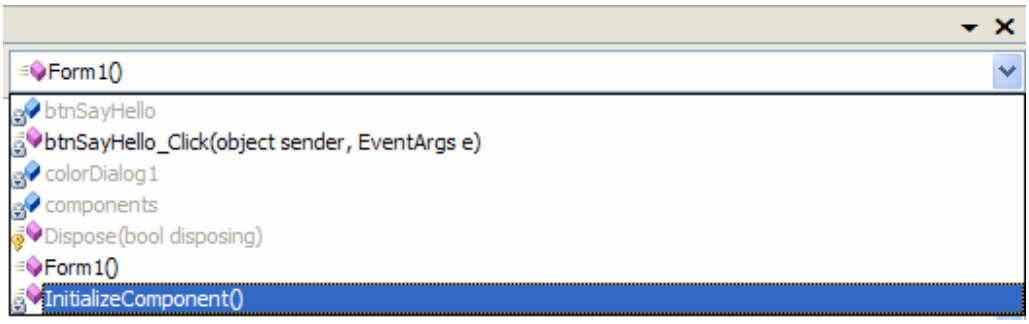
۵) در قسمت ویرایشگر کد، به دو کادر بالای صفحه توجه کنید. کادر سمت چپ مربوط به نمایش کلاسها است و نام کلاسهای تعریف شده در این قسمت، در این کادر قرار می گیرد. در حالت عادی فقط یک کلاس که مربوط به فرم برنامه است در این قسمت وجود دارد. اما اگر کلاسهای دیگری را در این قسمت تعریف کنید، نام آنها نیز به این لیست اضافه می شود. برای مثال اگر کلاسی به نام TempClass در قسمت پایین کلاس Form1 در این برنامه تعریف کنیم محتویات این کادر همانند شکل ۶-۲ خواهد بود.



شکل ۶-۲

۶) بعد از اینکه یکی از کلاسهای موجود در لیست سمت چپ را انتخاب کردید، اعضای آن کلاس مانند متدها، خاصیت ها و ... در لیست سمت راست نمایش داده می شوند. برای مثال اگر در لیست سمت چپ، کلاس مربوط به فرم Form1 را انتخاب کنید، اعضای آن همانند شکل ۶-۳ خواهند بود. گزینه های موجود در این لیست در زیر تعریف شده اند:

- همانطور که گفتم محتویات این لیست بر اساس نام کلاسی که در سمت چپ انتخاب می کنید تغییر می کند. با استفاده از این لیست می توانید بین اعضای موجود در یک کلاس جا به جا شوید. البته وظیفه اصلی این لیست نمایش تمام اعضای مرتبط با کلاسی است که در لیست سمت چپ انتخاب کرده اید.
- در ابتدای این برنامه، یک دکمه فرمان به نام `btnSayHello` به فرم اضافه کردیم. بنابراین همانطور که در لیست نیز مشاهده می کنید، فرم ما دارای یک عضو از نوع کنترل `Button` به نام `btnSayHello` است. با انتخاب این گزینه از لیست سمت چپ، ویژوال استودیو شما را به خطی از برنامه می برد که این کنترل در آن تعریف شده است.



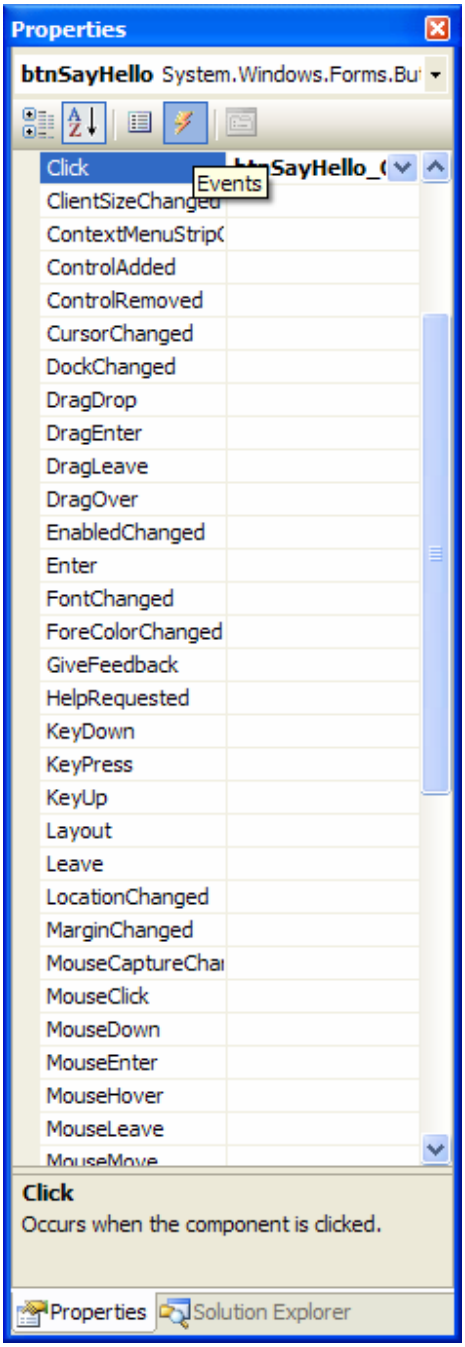
شکل ۶-۳

- بعد از اینکه کنترل `Button` را به فرم اضافه کردیم، متدی برای رویداد کلیک آن به نام `btnSayHello_Click` ایجاد کردیم. پس یکی دیگر از اعضای فرم ما نیز متدی به نام `btnSayHello_Click` خواهد بود. همانطور که در لیست مشاهده می کنید، این عضو از کلاس `Form1` نیز در لیست نمایش داده شده است که با انتخاب آن مکان نما به قسمتی از برنامه که این متد قرار دارد منتقل می شود.
- هر کلاس دارای متدی همنام با نام کلاس است. اگر به گزینه های موجود در لیست سمت راست دقت کنید، متدی را با نام `Form1` مشاهده خواهید کرد. این متد که به نام سازنده کلاس نیز شناخته می شود، شامل کدهایی است که در ابتدای اجرای کلاس، شرایط اولیه کنترلهای موجود در کلاس را تنظیم می کند. بنابراین اگر بخواهید کدی هنگام ایجاد شدن یک شیئ از یک کلاس اجرا شود، باید آن را در این قسمت قرار دهید.
- یکی دیگر از گزینه های این لیست، گزینه `Dispose` است. کلاس تشکیل دهنده فرم دارای تابعی به نام `Dispose` است که منابع گرفته شده به وسیله کنترلهای کلاس را در پایان اجرای این فرم از برنامه آزاد می کند.
- یکی دیگر از اعضای کلاس `Form1` که در لیست مشاهده می کنید، متد `InitializeComponent` است که مسئول مقدار دهی اولیه به کنترلهای موجود در کلاس است. برای مثال کد مربوط به تنظیم نام کنترلهای موجود در فرم، تنظیم خاصیت `Text` آنها و یا هر تغییر دیگری که با استفاده از قسمت `Form Designer` به وجود می آورید، در این قسمت به طور اتوماتیک توسط ویژوال استودیو نوشته می شود. توجه داشته باشید که تا حد امکان کدهای موجود در این متد را به صورت دستی تغییر ندهید و از بخش `Form Designer` برای تغییر خاصیتهای کنترل ها استفاده کنید.
- همانطور که مشاهده می کنید ویژوال C# ۲۰۰۵ آیکون کوچکی را در سمت چپ هر یک از گزینه های موجود در لیست قرار می دهد. به وسیله این آیکونها می توانید نوع آنها را تشخیص دهید. برای مثال، یک جعبه ی صورتی

رنگ کوچک مشخص کننده یک متد است، یک جعبه آبی رنگ کوچک مشخص کننده یک عضو از کلاس است و ....

نکته: در کنار این آیکونها که برای مشخص کردن نوع عضوهای یک کلاس به کار می رود، آیکونهای کوچک دیگری نیز قرار دارند که نوع تعریف شدن آن عضو از کلاس را مشخص می کنند. برای مثال، کلید زرد رنگ کوچک به این معنی است که آن عضو از نوع `Protected` است، و یا قفل خاکستری رنگ به این معنی است که عضو از نوع `private` تعریف شده است. در مورد مفهوم این کلمات در فصلهای بعد بیشتر صحبت خواهیم کرد.

۷) به قسمت طراحی فرم برای `Form1` برگردید و دکمه فرمان `btnSayHello` را در فرم انتخاب کنید. در پنجره `Properties` بر روی آیکون زرد رنگ بالای پنجره یعنی آیکون `Events` کلیک کنید (شکل ۶-۴). به این ترتیب لیستی طولانی از متدهای مربوط به کنترل دکمه فرمان را مشاهده خواهید کرد.



شکل ۶-۴

یکی از این رویداد ها، رویداد کلیک است که به صورت پررنگ نمایش داده شده است. علت این تفاوت در این است که شما در این قسمت برای رویداد کلیک این کنترل، یک متد را مشخص کرده اید. اگر بر روی رویداد کلیک در این قسمت دو بار کلیک کنید، به بخش تعریف متد مربوط به این رویداد در قسمت ویرایشگر کد خواهید رفت.

۸) حال رویداد دیگری را برای کنترل `Button` تعریف می کنیم. از لیست نمایش داده شده در پنجره `Properties` گزینه `MouseEnter` را پیدا کرده و بر روی آن دو بار کلیک کنید. رویداد جدیدی برای کنترل `Button` ایجاد می شود. کد مشخص شده در زیر را به این رویداد اضافه کنید:

```
private void btnSayHello_MouseEnter(object sender,
                                    EventArgs e)
{
     // Change the Button text
     btnSayHello.Text = "The Mouse is here!";
}
```

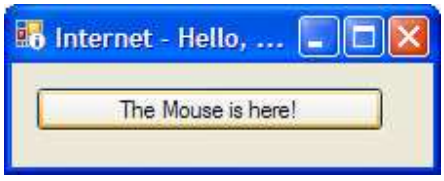
این رویداد زمانی اجرا می شود که اشاره گر ماوس وارد کنترل شود، به عبارت دیگر از محدوده ی خـارج از Button به لبه ی Button برسد.

۹) برای تکمیل این بخش، باید یک رویداد دیگر نیز ایجاد کنید. مجدداً به قسمت طراحی فرم مربوط به Form1 برگـشته و پس از انتخاب کنترل Button، در بالای پنجره ی Properties بر روی آیکون Events کلیک کنید تا لیست تمام رویدادهای این کنترل را مشاهده کنید. از لیست نمایش داده شـده گزینـه MouseLeave را انتخـاب کرده و بر روی آن دو بار کلیک کنید. به این ترتیب متد دیگری برای این رویداد ایجاد می شود. کد مشخص شده در زیـر را به این متد اضافه کنید.

```
private void btnSayHello_MouseLeave(object sender,
                                    EventArgs e)
{
     // Chenage the Button text
     btnSayHello.Text = "The mouse has gone!";
}
```

رویداد MouseLeave هنگامی فراخوانی می شود که اشاره گر ماوس از محدوده ی یک کنترل خارج شـود. بـرای مثال در اینجا هنگامی فراخوانی می شود که اشاره گر ماوس از روی کنترل Button به کنار برود.

۱۰) برنامه را اجرا کرده، ماوس را بر روی کنترل Button ببرید و از روی آن رد کنید. مشاهده خواهید کرد که مـتن روی کنترل همانند شکل ۶-۵ تغییر خواهد کرد.



شکل ۶-۵

چگونه کار می کند؟

اغلب کنترل هایی که از آنها استفاده می کنیم دارای تعداد زیادی رویداد هستند. البته در برنامه نویسی عادی، تعداد کمـی از آنهـا بـه طور ثابت مورد استفاده قرار می گیرد. برای مثال رویداد کلیک کنترل Button، یکی از رویدادهایی اسـت کـه بـه شـدت مـورد استفاده قرار می گیرد.

در ویژوال C# هر کنترل یک رویداد پیش فرض دارد که با دو بار کلیک بـر روی آن کنتـرل، متـد مربـوط بـه آن رویـداد بـه طـور اتوماتیک ایجاد می شود. این رویداد معمولاً پر کاربرد ترین رویداد آن کنترل است. برای مثال در کنترل Button، رویداد کلیـک به عنوان رویداد پیش فرض در نظر گرفته می شود.
یکی دیگر از رویدادهای کنترل Button، رویداد MouseEnter است که با وارد شدن اشاره گر ماوس به محدوده کنترل فعال می شود.

```
private void btnSayHello_MouseEnter(object sender,
                                    EventArgs e)
{
     // Change the Button text
     btnSayHello.Text = "The Mouse is here!";
}
```

در این هنگام می توانیم خاصیت Text کنترل را تغییر دهیم تا تغییر موقعیت اشاره گر ماوس را نـشان دهـد. در قـسمتهای قبلـی برای تغییر یکی از خاصیت های یک کنترل، از قسمت طراحی فرم در زمان طراحی استفاده می کردیم. اما همانطور که مشاهده مـی کنید برای این کار می توان از کدهای زمان اجرا نیز استفاده کرد.
نکته: زمان طراحی[1] در برنامه نویسی به زمانی اطلاق می شود که در حال طراحی رابط گرافیکی برنامـه و یـا حتـی نوشـتن کـد مربوط به آن هستید. در مقابل به زمانی که برنامه در حال اجرا است، **زمان اجرا**[2] گفته می شود.

ایجاد یک برنامه ساده:

ویژوال استودیو ۲۰۰۵ دارای مجموعه کاملی از کنترل ها است که می توانید برای طراحی برنامه های خود از آنهـا اسـتفاده کنیـد. در طراحی یک برنامه، اغلب با استفاده از این کنترل ها میتوان برنامه را طراحی کرد. اما در فصل ۱۳ خواهید دید که چگونه مـی توانیـد کنترل هایی مخصوص به خودتان بسازید.
در قسمت بعد مشاهده خواهیم کرد که چگونه می توان با ترکیب این کنترل ها، برنامه ساده ای را ایجاد کرد. در قسمت امتحان کنید بعد، یک برنامه ساده ویندوزی ایجاد خواهیم کرد که به کاربر اجازه دهد متنی را در یک کادر وارد کند. سـپس برنامـه تعـداد حـروف متن و تعداد کلمات آن را شمرده و آن را در صفحه نمایش می دهد.

ایجاد فرم:

برای نوشتن این برنامه، اولین کار ایجاد یک پروژه جدید و سـاختن یـک فـرم بـرای برنامـه اسـت. ایـن فـرم شـامل یـک کنتـرل TextBox چند خطی خواهد بود تا کاربر بتوانـد مـتن مـورد نظـر خـود را در آن وارد کنـد. همچنـین برنامـه شـامل دو کنتـرل RadioButton خواهد بود که به کاربر اجازه می دهد، بین شمردن کلمات متن و یا حروف آن یک مورد را انتخاب کند.

---

[1] Design Time
[2] Run Time

امتحان کنید: ایجاد فرم

۱) از منوی ویژوال استودیو ۲۰۰۵ گزینه File ➔ New ➔ Project... را انتخاب کرده و یک برنامه ویندوزی جدید ایجاد کنید. نام پروژه را Word Counter قرار دهید و سپس بر روی دکمه OK کلیک کنید تا پروژه ایجاد شود.

۲) بر روی فرم برنامه کلیک کنید تا انتخاب شود. سپس با استفاده از پنجره Properties خاصیت Size را برابر با 424;312، خاصیت StartPosition را برابر با CenterScreen و خاصیت Text آن را برابر با Word Counter قرار دهید.

۳) یک کنترل TextBox بر روی فرم قرار دهید و خاصیتهای آن را مطابق با لیست زیر تنظیم کنید.

خاصیت Name را برابر با txtWords قرار دهید.
خاصیت Location را برابر با 8,23 قرار دهید.
خاصیت Multiline را برابر با True قرار دهید.
خاصیت ScrollBars را برابر با Vertical قرار دهید.
خاصیت Size را برابر با 400,217 قرار دهید.

۴) برای این که کاربر را در استفاده از فرم راهنمایی کنید، باید یک برچسب نیز در فرم قرار دهید. برای این کار از جعبه ابزار کنترل Label را انتخاب کنید و آن را با استفاده از ماوس همانند قرار دادن کنترل TextBox، بر روی فرم قرار دهید. سپس خاصیت Text این کنترل را برابر با Enter some text into this box قرار دهید.
همانطور که ملاحظه می کنید، در این قسمت خاصیت Name مربوط به کنترل لیبل را تغییر ندادیم. دلیل این امر در این است که تا زمانی که نخواهید از یک کنترل در قسمت ویرایشگر کد استفاده کنید، نیازی نیست که برای آن یک نام تعیین کنید. برای مثال در این برنامه، نیاز داریم از متد ها و خاصیتهای کنترل TextBox در کد استفاده کنیم تا بتوانیم کلمات و حروف داخل آن را بشماریم. اما نیازی نداریم که از کنترل لیبل اطلاعاتی را دریافت کنیم و یا در طول اجرای برنامه خاصیتی از آن را تغییر دهیم. بنابراین نام آن را تغییر نمی دهیم و اجازه می دهیم همان مقدار اولیه باقی بماند.

۵) برنامه شما قادر خواهد بود تعداد کلمات و نیز تعداد حروف داخل یک متن را بشمارد. بنابراین باید به کاربر اجازه دهید که شمارش تعداد کلمات و یا تعداد حروف را انتخاب کند. برای این کار می توانید از دو کنترل RadioButton استفاده کنید. با استفاده از جعبه ابزار، دو کنترل RadioButton در کنار هم و در پایین TextBox بر روی فرم قرار دهید. سپس خاصیتهای این کنترل ها را برابر با مقادیر مشخص شده در زیر وارد کنید.
برای دکمه رادیویی اول:

- خاصیت Name را برابر با radCountChars قرار دهید.
- خاصیت Checked را برابر با True قرار دهید.
- خاصیت Text را برابر با Chars قرار دهید.

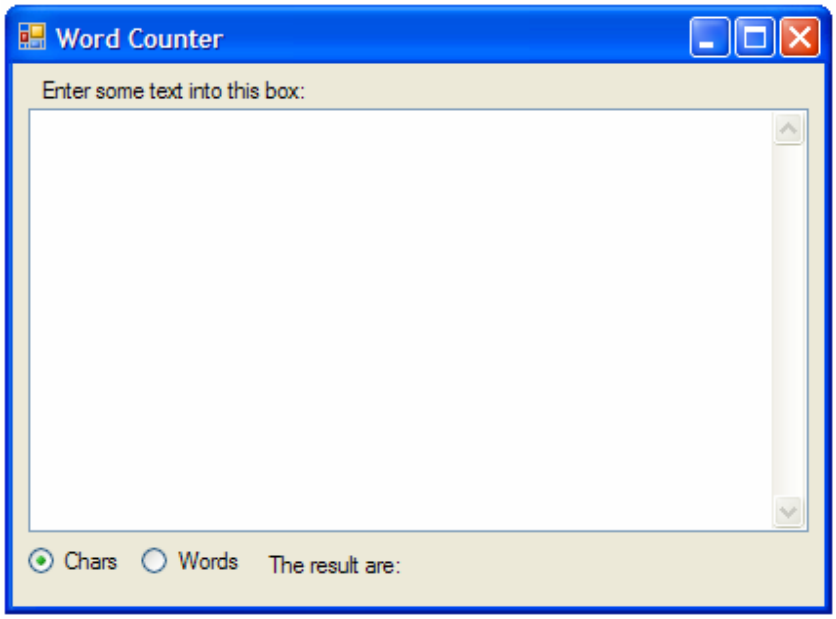
برای دکمه رادیویی دوم:

- خاصیت Name را برابر با radCountWords قرار دهید.
- خاصیت Text را برابر با Words قرار دهید.

۶) هنگامی که کاربر متنی را در کادر مشخص شده وارد کرد، برنامه تعداد کاراکتر ها و یا تعداد کلمات آن را خواهد شمرد. در مرحله بعد این تعداد باید به وسیله پیغام مناسب به کاربر نمایش داده شود. بنابراین، برای نمایش نتیجه دو کنترل لیبل را در کنار کنترلهای RadioButton در فرم قرار دهید.

۷) کنترل لیبل اول (که دارای نام label2 است) فقط برای نمایش یک متن ثابت در طول برنامه به کار می رود، بنابراین نیازی نیست که نام آن را تغییر دهیم. کافی است که خاصیت Text آن را با The result are: تنظیم کنیم. لیبل دوم برای نمایش نتیجه به کار می رود. بنابراین خاصیت Name آن را برابر با lblResults قرار می دهیم و متن داخل قسمت Text آن را نیز پاک می کنیم. بعد از انجام این موارد فرم برنامه شما باید مشابه شکل ۶-۶ باشد.

۸) حال که کنترل ها را در مکان مورد نظرتان بر روی فرم قرار دادید، بهتر است کاری کنید که در جای خود ثابت باقی بمانند و موقعیت شان به طور تصادفی تغییر نکند. برای این کار یکی از کنترلهای روی فرم را انتخاب کرده و سپس گزینه Format ➔ Lock Controls را از نوار منو انتخاب کنید. به این ترتیب خاصیت Locked همه کنترل ها برابر با True خواهد شد و احتمال این وجود نخواهد داشت که یکی از این کنترل ها به طور تصادفی پاک شود، اندازه اش تغییر داده شود و یا مکان آن در فرم تغییر کند.



شکل ۶-۶

شمارش کاراکتر ها:

حال که فرم برنامه طراحی شد، باید تعدادی متد برای رویدادهای مورد نیاز طراحی کنید تا هنگامی که کاربر متنی را در TextBox وارد کرد، تعداد کاراکترهای آن متن در پایین فرم نمایش داده شود. بر روی نام فرم در قسمت Solution

Explorer و یا بر روی خود فرم کلیک راست کنید و از منوی باز شده گزینه View Code را انتخاب کنید تا به قسمت ویرایشگر کد مربوط به Form1 بروید. برای اینکه می خواهید هم تعداد کاراکترهای یک متن و هم تعداد کلمات آن را بشمارید نیاز دارید که توابعی جداگانه برای این دو مورد بنویسید. در قسمت امتحان کنید بعدی، کدی خواهیم نوشت که تعداد کاراکترهای یک متن را بشمارد.

امتحان کنید: شمارش کاراکتر ها

۱) در قسمت ویرایشگر کد، کد زیر را درون کلاس مربوط به Form1 وارد کنید. به یاد دارید که برای قرار دادن بخشهای توضیحی از نوع XML Document Comment باید سه کاراکتر / را به طور متوالی قبل از تابع وارد کنید.

```
/// <summary>
/// Count the characters in a blick of text
/// </summary>
/// <param name="text">The string containing the text to
/// count characters in</param>
/// <returns>The number of characters in the
/// string</returns>
private int CountCharacters(string text)
{
    return text.Length;
}
```

۲) حال باید یک متد برای رویداد TextChanged کنترل TextBox ایجاد کنید. به بخش طراحی فرم مربوط به Form1 بازگردید و کنترل txtWords را بر روی فرم انتخاب کنید. در بالای پنجره Properties بر روی آیکون Events کلیک کنید تا لیستی از متدهای مربوط به کنترل TextBox نمایش داده شود. در این لیست گزینه TextChanged را انتخاب کرده و بر روی آن دو بار کلیک کنید. متد مربوط به این رویداد به طور اتوماتیک ایجاد خواهد شد. سپس کد مشخص شده در زیر را در این متد وارد کنید.

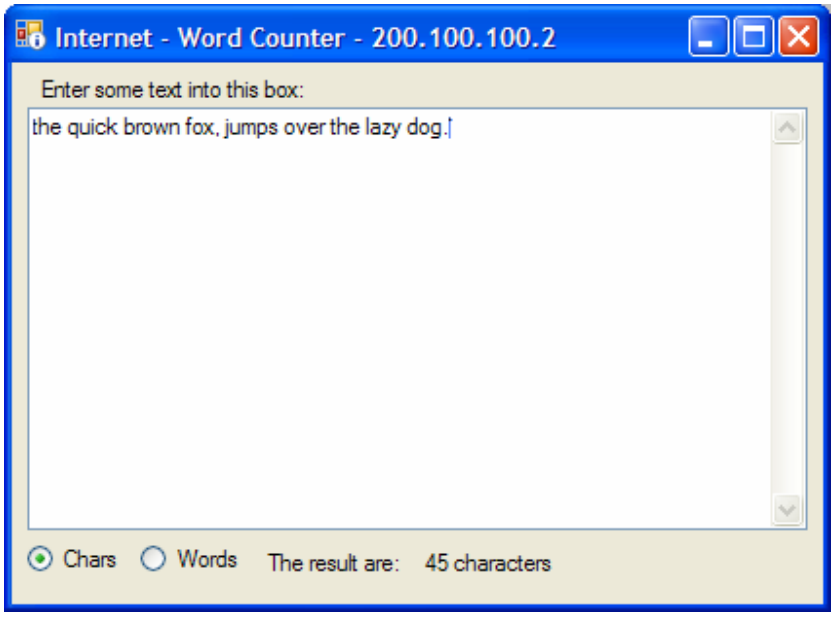
```
private void txtWords_TextChanged(object sender,
                                  EventArgs e)
{
    // Count the number of characters
    int intChars = CountCharacters(txtWords.Text);

    // Display the results
    lblResults.Text = intChars + " characters";
}
```

۳) برنامه را اجرا کنید و متنی را در TextBox وارد کنید. مشاهده خواهید کرد که تعداد کاراکترهای موجود در متن در پایین فرم نمایش داده می شود (شکل ۶-۷).

چگونه کار می کند؟

همانطور که ملاحظه می کنید هنگامی که کاراکتری را در `TextBox` وارد می کنید، کنترل لیبل پایین فـرم تعـداد کاراکترهـای موجود در متن را نمایش می دهد. زیرا هر بار که متن داخل `TextBox` تغییر می کند، رویداد `TextChanged` فراخوانی می شود. بنابراین هر بار که متن جدیدی وارد شود، قسمتی از متن قبلی حذف شود و در کل قسمتی از متن به هر نحوی تغییر کنـد، متد مربوط به این رویداد تابع `CountCharacters` را فراخوانی می کند و متن داخل `TextBox` را به عنوان پـارامتر به این متد می فرستد. این متد نیز تعداد کاراکترهای موجود در متن را شمرده و نتیجه را در متغیر `intChars` قرار می دهد.



شکل ۶-۷

```
// Count the number of characters
int intChars = CountCharacters(txtWords.Text);
```

سپس عدد به دست آمده با پیغام مناسب در کنترل لیبل قرار می گیرد تا تعداد کاراکترهای موجود در متن به کاربر اطلاع داده شود.

```
// Display the results
lblResults.Text = intChars + " characters";
```

شمارش کلمات:

اگرچه نوشتن یک برنامه با استفاده از ویژوال C# ۲۰۰۵ بسیار ساده به نظر می رسد، اما ارائه یک راه حل ظریف و کارآمد برای یک مسئله به ترکیبی از تجربه و استدلال نیاز دارد.

برای مثال همین برنامه را در نظر بگیرید. شما می خواهید هنگامی که دکمه رادیویی Words انتخاب شده بود، برنامه تعداد کلمات را بشمارد و هنگامی که دکمه رادیویی Chars انتخاب شده بود برنامه تعداد کاراکتر ها را بشمارد. در این مورد باید به دو مورد توجه کنید. اول اینکه زمانی که به رویداد TextChanged پاسخ می دهید، برای شمارش تعداد کلمات باید از یک تابع و برای شمارش تعداد کاراکتر ها باید از تابعی دیگر استفاده کنید. البته این مورد زیاد سخت نیست.
دوم اینکه هنگامی که کاربر بر روی یکی از کنترلهای RadioButton کلیک می کند، باید متن نمایش داده شده را از "Characters" به "Words" و یا برعکس تغییر دهید.
حال، تعداد بیشتری رویداد را به برنامه اضافه می کنیم و بعد از تمام شدن برنامه، منطقی که در پشت تکنیکهای آن به کار رفته است را بررسی خواهیم کرد.

امتحان کنید: شمارش کلمات

۱) اگر همچنان برنامه در حال اجرا است، آن را متوقف کنید. اولین کاری که باید انجام دهید این است که تابع دیگری بنویسید که تعداد کلمات موجود در یک متن مشخص را بشمارد. کد زیر را به برنامه اضافه کنید تا تابع CountWords ایجاد شود:

```
/// <summary>
/// Counts the number of words in a block of text
/// </summary>
/// <param name="text">The string containing the text to
/// count words in</param>
/// <returns>The number of words in the string</returns>
private int CountWords(string text)
{
     // Is the text box empty
     if (txtWords.Text == String.Empty)
          return 0;

     // Split the words
     string[] strWords = text.Split(' ');

     // Return the number of words
     return strWords.Length;
}
```

۲) در این مرحله زیربرنامه ای به نام UpdateDisplay ایجاد خواهیم کرد که متن را از TextBox دریافت کند و بعد از تشخیص اینکه تعداد کاراکتر ها را باید بشمارد و یا تعداد کلمات را، عدد مورد نظر را در صفحه نمایش دهد. برای اضافه کردن این زیربرنامه کد زیر را به برنامه اضافه کنید:

```
private void UpdateDisplay()
{
     // Do we want to count words?
```

```
    if (radCountWords.Checked == true)
    {
        // Update the results
        lblResults.Text =
            CountWords(txtWords.Text) + " words";
    }
    else
    {
        // Update the results
        lblResults.Text =
        CountCharacters(txtWords.Text) + " characters";
    }
}
```

۳) حال به جای اینکه در متد مربوط به رویداد کلیک، تابع `CountCharacters` را فراخوانی کنیم، تابع `UpdateDisplay` را فراخوانی می کنیم. بنابراین تغییرات زیر را در این متد ایجاد کنید.

```
private void txtWords_TextChanged(object sender,
                                  EventArgs e)
{
    // Something chenged, so display the results
    UpdateDisplay();
}
```

۴) در آخر باید برنامه را به گونه ای تغییر دهید تا هنگامی که دکمه رادیویی از `Chars` به `Words` و یا بر عکس تغییر کرد، عدد نمایش داده شده در صفحه نیز تغییر کند. برای این کار می توانید از رویداد `CheckedChanged` یکی از دکمه های رادیویی استفاده کنید. در قسمت طراحی فرم، کنترل `radCountWords` را انتخاب کرده و در پنجره `Properties` بر روی آیکون `Events` کلیک کنید تا رویدادهای مربوط به این کنترل نمایش داده شوند. از لیست رویدادها، رویداد `CheckedChanged` را انتخاب کرده و بر روی آن دو بار کلیک کنید. در متد ایجاد شده برای این رویداد، کد زیر را وارد کنید:

```
private void radCountWords_CheckedChanged(object sender,
                                          EventArgs e)
{
    // Something chenged, so display the results
    UpdateDisplay();
}
```

۵) مراحل قبل را برای کنترل `radCountChars` نیز تکرار کنید:

```
private void radCountChars_CheckedChanged(object sender,
                                          EventArgs e)
{
    // Something chenged, so display the results
```

```
        UpdateDisplay();
}
```

۶) برنامه را اجرا کرده و متنی را در قسمت مشخص شده وارد کنید. سپس بر روی دکمه رادیویی Words کلیک کنید. مشاهده می کنید که متن نمایش داده شده در فرم تغییر کرده و تعداد کلمات را نمایش می دهد (شکل ۶-۸).

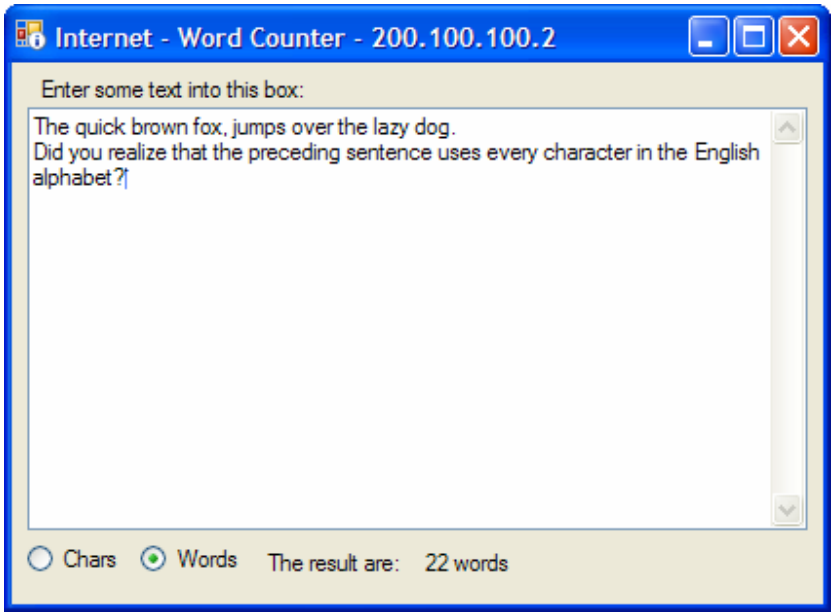
چگونه کار می کند؟

قبل از اینکه به بررسی قسمتهای مختلف برنامه بپردازیم، بهتر است که نحوه عملکرد تابع CountWords را بررسی کنیم:

```
/// <summary>
/// Counts the number of words in a block of text
/// </summary>
/// <param name="text">The string containing the text to
/// count words in</param>
/// <returns>The number of words in the string</returns>
private int CountWords(string text)
{
     // Is the text box empty
     if (txtWords.Text == String.Empty)
          return 0;

     // Split the words
     string[] strWords = text.Split(' ');

     // Return the number of words
     return strWords.Length;
}
```



شکل ۶-۸

در ابتدای تابع، مقدار خاصیت `Text` مربوط به `TextBox` را با عضو `Empty` از کلاس `String` بررسی می کنیم تا از وجود متن در آن مطمئن شویم. عضو `Empty` از کلاس `String` برابر با رشته ای به طول صفر (`""`) است، بنابراین اگر مقدار داخل خاصیت `Text` برابر با این عضو باشد می توان فهمید که متنی داخل `TextBox` وارد نشده است. در این حالت تابع مقدار صفر را برمی گرداند.
تابع `Split` از کلاس `String` یک رشته را دریافت می کند و آن را به آرایه ای از رشته ها تبدیل کرده و برمی گرداند. این تابع یک کاراکتر و یا آرایه ای از کاراکتر ها را به عنوان ورودی دریافت کرده و از کاراکتر ها به عنوان جدا کننده استفاده می کند تا رشته وارد شده را به چند زیر رشته تقسیم کند. برای مثال در این برنامه رشته داخل `TextBox` را به همراه کاراکتر فاصله (' ') به تابع می فرستیم. تابع نیز رشته را به چند زیر رشته که به وسیله فاصله از هم جدا شده اند تبدیل کرده و نتیجه را برمی گرداند. بنابراین آرایه ای که این تابع برمی گرداند حاوی کلمات رشته ورودی خواهد بود. بعد از آن می توانیم طول آرایه که در حقیقت تعداد کلمات رشته اصلی است را برگردانیم.

نکته: در این برنامه فرض کرده ایم که رشته به صورت استاندارد در `TextBox` وارد شده است، یعنی تمام کلمات فقط با یک کاراکتر فاصله از یکدیگر جدا شده اند. بنابراین اگر بین کلمات متنی که در برنامه وارد می کنید بیش از یک فاصله وجود داشته باشد، تعداد کلمات به صورت نادرست نمایش داده می شود.

نکته: همانطور که گفتم تابع `Split` یک کاراکتر و یا آرایه ای از کاراکتر ها را به عنوان ورودی دریافت می کند، بنابراین اگر بخواهید کاراکتر فاصله را به تابع بفرستید باید از ' ' استفاده کنید نه از " ". زیرا در C# علامت " برای مشخص کردن رشته به کار می رود و علامت ' برای مشخص کردن کاراکتر. به عبارت دیگر، عبارت " " به عنوان یک رشته با طول یک در نظر گرفته می شود و عبارت ' ' به عنوان یک کاراکتر.

یکی از عادت های خوب برنامه نویسی در این است که فقط به مقدار مورد نیاز در یک برنامه، کد نوشته شود. به عبارت دیگر اگر در موقعیتی مجبور شدید که از یک کد دوبار استفاده کنید، روشی را به کار ببرید که آن کد را فقط یک بار بنویسید. برای مثال در این

برنامه، در دو قسمت نیاز داشتید که متن داخل lblResults را تغییر دهید تا تعداد درست کلمات و یا کاراکتر ها را نمایش دهد. بهترین روش برای این کار در این است که کد مربوط به تغییر متن داخل lblResults را در داخل یک متد مجزا مانند UpdateDisplay قرار دهید. به این ترتیب به راحتی می توانید هنگام نوشتن کد مربوط به رویدادهای TextChanged و CheckedChaned فقط متد قبلی را فراخوانی کنید تا متن داخل lblResults تصحیح شود. با استفاده از این روش کد برنامه نه تنها کوتاهتر شده و نگهداری و تغییر آن در آینده راحت تر می شود، بلکه هنگام ایجاد خطا در برنامه تصحیح آن راحت تر صورت می گیرد.

```
private void UpdateDisplay()
{
     // Do we want to count words?
     if (radCountWords.Checked == true)
     {
          // Update the results
          lblResults.Text =
               CountWords(txtWords.Text) + " words";
     }
     else
     {
          // Update the results
          lblResults.Text =
          CountCharacters(txtWords.Text) + " characters";
     }
}
```

ایجاد برنامه های پیچیده تر:

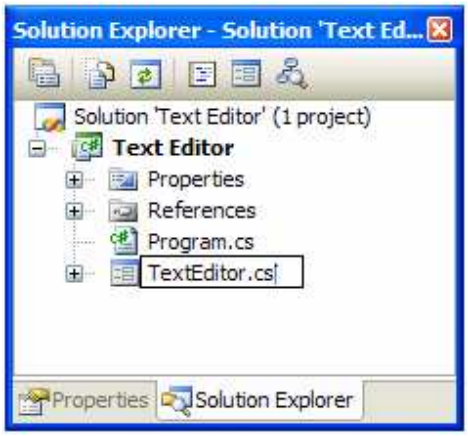
برنامه های عادی علاوه بر کنترلهایی مانند Button و یا TextBox و ...، عموما دارای بخشهایی مانند نوار ابزار و نوار وضعیت نیز هستند. ایجاد این قسمتها با استفاده از کنترلهای موجود در ویژوال C# کار بسیار ساده ای است.
در بخش بعد برنامه ای خواهیم نوشت که به وسیله آن بتوانیم متن وارد شده در برنامه را تغییر دهیم. برای مثال رنگ متن را تغییر دهیم و یا آن را به حروف بزرگ و یا کوچک تبدیل کنیم.

برنامه ویرایشگر متن:

در قسمت امتحان کنید زیر، برنامه ای خواهیم نوشت که به وسیله آن بتوان متن وارد شده در یک TextBox را ویرایش کرد. در این برنامه از یک نوار ابزار استفاده می کنیم تا به وسیله آن رنگ متن را تغییر دهیم و یا آن را به حروف بزرگ و یا کوچک تبدیل کنیم.
همچنین در این برنامه از یک نوار وضعیت نیز استفاده می کنیم تا بتوانیم هنگامی که بر روی یکی از کلیدهای نوار ابزار کلیک شد، وضعیت برنامه را بر اساس آن نمایش دهیم.
اولین مرحله در این برنامه ایجاد یک پروژه جدید است.

امتحان کنید: ایجاد برنامه ویرایشگر متن

۱) یک پروژه ویندوزی جدید ایجاد کرده و نام آن را `Text Editor` قرار دهید.
۲) در اغلب موارد نام `Form1` برای یک فرم مناسب نیست، زیرا در برنامه به وسیله ی این نام نمی توان فرم ها را از یکدیگر تشخیص داد. برای تغییر نام فرم این برنامه، در پنجره `Solution Explorer` بر روی نام فرم کلیک راست کرده و از منوی باز شده گزینه `Rename` را انتخاب کنید. سپس همانند شکل ۶-۹ نام فرم را به `TextEditor.cs` تغییر دهید.



شکل ۶-۹

۳) حال بر روی فرم در قسمت طراحی فرم کلیک کنید تا انتخاب شود. سپس در پنجره `Properties` خاصیت `Text` آن را به `Text Editor` و خاصیت `Size` فرم را به `600;460` تغییر دهید.

در قسمت بعد به طراحی رابط کاربری برنامه خواهیم پرداخت.

ایجاد نوار ابزار:

همانطور که می دانید یک نوار ابزار مجموعه ای از کنترل ها است که اغلب آنها `Button` هستند، همانند نوار ابزار موجود در محیط ویژوال استودیو ۲۰۰۵. در بخش امتحان کنید بعد، یک نوار ابزار برای برنامه ایجاد کرده و `Button` های مورد نیاز را نیز به آن اضافه خواهیم کرد.

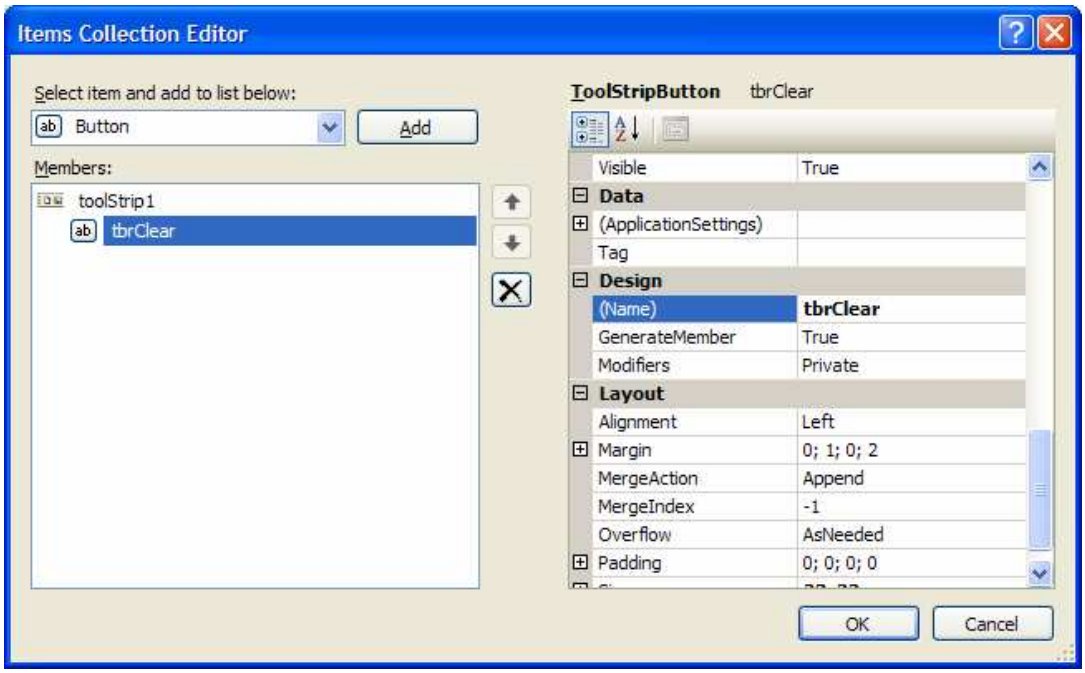
امتحان کنید: اضافه کردن نوار ابزار

۱) با استفاده از قسمت جعبه ابزار ویژوال استودیو، کنترل ToolStrip را انتخاب کرده و به وسیله ماوس آن را بر روی فرم قرار دهید. این کنترل به صورت اتوماتیک در بالای فرم قرار خواهد گرفت.

۲) برای این که بتوانیم تعدادی کنترل را به نوار ابزار اضافه کنیم، باید از پنجره Items Collection Editor استفاده کنیم. برای نمایش این پنجره بر روی نوار ابزار در بالای فرم کلیک راست کنید و از منوی باز شده گزینه Edit Items... را انتخاب کنید.

۳) در این مرحله باید شش کنترل Button به نوار ابزار اضافه کنیم تا در برنامه از آنها استفاده کنیم. دکمه های Clear، Red، Blue، LowerCase، UpperCase و دکمه About.

۴) با استفاده از پنجره ی Items Collection Editor می توانید کنترلهای مورد نیاز خود را به نوار ابزار اضافه کنید. برای انتخاب نوع کنترل باید از ComboBox ای که در گوشه بالای سمت چپ صفحه قرار دارد استفاده کنید. از لیست این قسمت گزینه Button را انتخاب کرده و بر روی دکمه Add کلیک کنید تا یک کنترل Button به فرم اضافه شود. به این ترتیب لیستی از خاصیتهای مربوط به این کنترل را، همانند خاصیتهای موجود در پنجره Properties مشاهده خواهید کرد. برای تمام کنترلهایی که در این قسمت اضافه می کنیم باید نام و نحوه نمایش آنها را تغییر دهید، یک آیکون مناسب برای آنها انتخاب کنید، متن داخل آنها را پاک کنید و یک متن راهنمای مناسب برای آنها قرار دهید. خاصیت Name کنترل Button جدید را همانند شکل ۶-۱۰ به tbrClear تغییر دهید. این کنترل برای پاک کردن متن وارد شده در داخل ویرایشگر به کار می رود.



شکل ۶-۱۰

۵) خاصیت DisplayStyle آن را نیز به Image تغییر دهید.

۶) در لیست خاصیتهای موجود برای این کنترل، خاصیت Image را انتخاب کرده و بر روی علامت ... روبروی آن کلیک کنید تا پنجره Select Resource باز شود. در این پنجره بر روی دکمه فرمان Import کلیک کنید. در پنجره Open به آدرس زیر بروید و فایل document.ico را انتخاب کنید. (اگر برنامه ویژوال استودیو را در فولدر دیگری نصب کرده اید به جای این آدرس از آدرسی که ویژوال استودیو در آن قرار دارد استفاده کنید)

C:\Program Files\Microsoft Visual Studio 8 \Common7\ VS2005ImageLibrary\icons\WinXP

سپس بر روی دکمه OK در پنجره Select Resource کلیک کنید تا به قسمت Items Collection Editor بازگردید.

۷) خاصیت ImageScaling را به none و خاصیت ToolTipText را برابر با New قرار دهید. همچنین مقدار وارد شده برای خاصیت Text را نیز پاک کنید. به این ترتیب کنترل Button اول ایجاد شده است.

۸) دکمه ی بعدی که اضافه خواهیم کرد، دکمه ی Red است. اما ابتدا باید یک خط جدا کننده بین دکمه Clear و دکمه Red قرار دهید. برای این کار در پنجره Item Collection Editor از لیست جعبه ترکیبی، گزینه Seperator را انتخاب کنید و سپس بر روی کلید Add کلیک کنید. نیازی نیست که خاصیتهای این کنترل را تغییر دهید و می توانید تمام خاصیتهای پیش فرض آن را قبول کنید.

۹) مراحل ۴ تا ۷ را برای ایجاد دکمه فرمان دوم تکرار کنید و خاصیتهای این دکمه فرمان را برابر با مقادیر زیر قرار دهید. از این دکمه فرمان برای تغییر رنگ متن به رنگ قرمز استفاده می کنیم. برای اضافه کردن دکمه فرمان قبل از اینکه بر روی دکمه Add کلیک کنید، توجه داشته باشید که گزینه Button در جعبه ترکیبی انتخاب شده باشد:

- خاصیت Name را برابر با tbrRed وارد کنید.
- خاصیت DisplayStyle را برابر با Image قرار دهید.
- برای خاصیت Image از آدرس \VS2005ImageLibrary\icons\Misc\servicestopped.ico استفاده کنید.
- خاصیت ImageScaling را برابر با None قرار دهید.
- مقدار موجود در خاصیت Text را پاک کنید.
- خاصیت ToolTipText را برابر با Red قرار دهید.

۱۰) مراحل ۴ تا ۷ را برای اضافه کردن دکمه فرمان Blue تکرار کنید و در این مرحله از خاصیتهای زیر استفاده کنید:

- خاصیت Name را برابر با tbrBlue وارد کنید.
- خاصیت DisplayStyle را برابر با Image قرار دهید.
- برای خاصیت Image از آدرس \VS2005ImageLibrary\icons\Misc\services.ico استفاده کنید.
- خاصیت ImageScaling را برابر با None قرار دهید.
- مقدار موجود در خاصیت Text را پاک کنید.
- خاصیت ToolTipText را برابر با Blue قرار دهید.

۱۱) در این مرحله باید بین دکمه های Blue و UpperCase نیز یک خط جدا کننده قرار دهیم. در جعبه ترکیبی صفحه ی Items Collection Editor گزینه Seperator را انتخاب کرده و بر روی دکمه Add کلیک کنید. در این قسمت نیز نیازی نیست خاصیتی از این کنترل را تغییر دهید.

۱۲) حال باید کنترل Button جدیدی اضافه کنیم تا متن داخل ویرایشگر را به حروف بزرگ تبدیل کند. برای این کار مراحل ۴ تا ۷ را برای ایجاد دکمه فرمان UpperCase تکرار کنید و در این مرحله از خاصیتهای زیر استفاده کنید:

- خاصیت Name را برابر با tbrUpperCase وارد کنید.
- خاصیت DisplayStyle را برابر با Image قرار دهید.
- برای خاصیت Image از آدرس \VS2005ImageLibrary\icons\WinXP\fonFile.ico استفاده کنید.
- خاصیت ImageScaling را برابر با None قرار دهید.
- مقدار موجود در خاصیت Text را پاک کنید.
- خاصیت ToolTipText را برابر با Upper Case قرار دهید.

۱۳) کنترل Button دیگری برای Lowercase ایجاد کنید تا به وسیله آن بتوانیم متن را به حروف کوچک تبدیل کنیم. برای این کنترل از خاصیتهای زیر استفاده کنید:

- خاصیت Name را برابر با tbrLowerCase وارد کنید.
- خاصیت DisplayStyle را برابر با Image قرار دهید.
- برای خاصیت Image از آدرس \VS2005ImageLibrary\icons\WinXP\fonfont.ico استفاده کنید.
- خاصیت ImageScaling را برابر با None قرار دهید.
- مقدار موجود در خاصیت Text را پاک کنید.
- خاصیت ToolTipText را برابر با Lower Case قرار دهید.

۱۴) با تغییر گزینه انتخاب شده در جعبه ترکیبی به Seperator و کلیک کردن بر روی دکمه Add جدا کننده دیگری بین دکمه های LowerCase و About قرار دهید.

۱۵) در این مرحله باید دکمه ای برای قسمت About برنامه اضافه کنیم. برای این کار مراحل ۴ تا ۷ را برای ایجاد یک دکمه فرمان جدید تکرار کنید و این بار از خاصیتهای زیر استفاده کنید:

- خاصیت Name را برابر با tbrHelpAbout وارد کنید.
- خاصیت DisplayStyle را برابر با Image قرار دهید.
- برای خاصیت Image از آدرس \VS2005ImageLibrary\icons\WinXP\help.ico استفاده کنید.
- خاصیت ImageScaling را برابر با None قرار دهید.
- مقدار موجود در خاصیت Text را پاک کنید.
- خاصیت ToolTipText را برابر با About قرار دهید.

۱۶) در پنجره Items Collection Editor بر روی دکمه OK کلیک کنید تا پنجره بسته شود و به قسمت طراحی فرم برگردید.

۱۷) با کلیک کردن بر روی دکمه Save All در نوار ابزار ویژوال استودیو، تغییرات ایجاد شده در برنامه را ذخیره کنید.

چگونه کار می کند؟

کنترل ToolStrip هنگامی که در فرم قرار داده شود به صورت اتوماتیک به یکی از کناره های فرم متصل می شود. در این برنامه، این کنترل به بالای فرم متصل شده است.
شش کنترل Button و سه جدا کننده ای که به برنامه اضافه شده اند، هر یک همانند کنترلهای معمولی که بر روی فرم قرار می گیرند، قابل انتخاب شدن هستند و می توانید با انتخاب آنها، خصوصیات آنها را در قسمت Properties مشاهده کنید و نیز به رویدادهای این کنترل ها دسترسی داشته باشید. در قسمتهای بعدی مشاهده خواهید کرد که چگونه می توانیم به رویدادهای این کنترل ها همانند رویداد کلیک پاسخ دهیم.
یک کنترل در نوار ابزار می تواند فقط دارای عکس باشد، فقط دارای نوشته باشد و یا دارای هم عکس و هم نوشته باشد. اما در بیشتر برنامه ها دکمه های روی نوار ابزار فقط دارای عکس هستند. به همین علت خاصیت DisplayStyle این کنترل ها را برابر با Image قرار می دهیم تا فقط یک عکس برای هر دکمه فرمان نمایش داده شود. همچنین برای اینکه اندازه عکسهای مشخص شده برای کنترل ها تغییر نکند و تمام عکسها در اندازه اولیه خود باقی بمانند، خاصیت ImageScaling آنها را برابر با None قرار می دهیم.
خاصیت ToolTipText به ویژوال C# این امکان را می دهد که یک راهنمای مختصر برای کنترلهای موجود در نوار ابزار ایجاد کند. به این ترتیب هنگامی که اشاره گر ماوس برای مدت کوتاهی بر روی این کنترل ماند، پنجره کوچکی باز می شود و متن نوشته شده در این خاصیت را به عنوان راهنما به کاربر نمایش می دهد.
در این مرحله نوار ابزار شما باید مشابه شکل ۶-۱۱ شده باشد.



شکل ۶-۱۱

ایجاد نوار وضعیت:

همانطور که در برنامه های ویندوزی دیگر مشاهده کرده اید، یک نوار وضعیت، پنل کوچکی است که در پاین صفحه قرار می گیرد و وضعیت کنونی برنامه را مشخص می کند. در بخش امتحان کنید بعد، نحوه ایجاد نوار وضعیت در یک برنامه را بررسی خواهیم کرد.

امتحان کنید: ایجاد یک نوار وضعیت

۱) با استفاده از جعبه ابزار ویژوال استودیو، یک کنترل StatusStrip را در فرم قرار دهید. مشاهده خواهید کرد که این کنترل به طور اتوماتیک در قسمت پاین فرم قرار گرفته و طول آن نیز به اندازه طول صفحه خواهد شد. خاصیت RenderMode این کنترل را به System تغییر دهید تا کنترل به صورت مسطح درآید.
۲) در این مرحله باید یک کنترل لیبل را به لیست کنترلهای موجود در StatusStrip اضافه کنید تا بتوانید متن های مورد نظرتان را در آن نمایش دهید. برای این کار بر روی نوار وضعیت اضافه شده به فرم کلیک راست کرده و از منوی باز شده گزینه ...Edit Items را انتخاب کنید تا پنجره Items Collection Editor

باز شود. این پنجره نیز مشابه پنجره ی Items Collection Editor برای کنترل نوار ابزار است. در جعبه ترکیبی سمت چپ این پنجره، گزینه StatusLabel را انتخاب کنید و سپس بر روی دکمه ی Add کلیک کنید.

۳) خاصیت های کنترل لیبل اضافه شده به نوار وضعیت را برابر با مقادیر زیر قرار دهید:

- خاصیت Name آن را برابر با sspStatus قرار دهید.
- خاصیت DisplayStyle آن را به Text تغییر دهید.
- خاصیت Text آن را به Ready تغییر دهید.

۴) بر روی دکمه OK در پنجره Items Collection Editor کلیک کنید.

۵) به قسمت ویرایشگر کد بروید و کد زیر را در به کلاس برنامه اضافه کنید:

```
// Get or set the text on the status bar
public string StatusText
{
      get
      {
            return sspStatus.Text;
      }
      set
      {
            sspStatus.Text = value;
      }
}
```

در این مرحله نیازی نیست که برنامه را اجرا کنید، زیرا هنوز کد اصلی آن را وارد نکرده ایم. بنابراین بهتر است مروری بر قسمتهای قبلی داشته باشیم.

چگونه کار می کند؟

ویژوال استودیو ۲۰۰۵ ویژگیهای زیادی برای راحت تر کردن طراحی فرم دارد. یکی از مواردی که همواره در نسخه های قبلی باعث دردسر برنامه نویسان می شد، تنظیم اندازه ی کنترل ها هنگام تغییر اندازه فرم توسط کاربر بود. برای مثال برنامه نویس فرم را برای اندازه ۴۰۰*۶۰۰ طراحی می کرد. اما هنگام اجرای برنامه، کاربر اندازه ی فرم را به ۷۰۰*۸۰۰ و یا هر اندازه دیگری تغییر می داد. به این ترتیب تمام کنترل ها به طور نامرتب بر روی فرم قرار می گرفتند.

در ویژوال استودیو ۲۰۰۵، کنترل ها می توانند به یکی از لبه های فرم متصل شوند. به صورت پیش فرض کنترل StatusStrip به پایین فرم متصل می شود. البته این خاصیت قابل تغییر است و می توانید در صورت لزوم با استفاده از

خاصیت Dock آن را تغییر دهید[1]. به این ترتیب هنگامی که اندازه فرم تغییر کند، چه در زمان طراحی و چه در زمان اجرا، نوار وضعیت (کنترل StatusStrip) موقعیت خود را با اندازه جدید فرم تنظیم خواهد کرد.
ممکن است بپرسید که چرا در این برنامه یک خاصیت به نام StatusText ایجاد کرده ام و وظیفه ی تنظیم کردن متن داخل نوار وضعیت را به آن سپرده ام. خوب این مورد به علت تجرد[2] برنامه است. فرض کنیم در آینده افرادی بخواهند از برنامه ی ویرایشگر متنی که شما نوشته اید، به عنوان یکی از فرمهای برنامه ی خود استفاده کنند. حال ممکن است این افراد در قسمتی از برنامه بخواهند که متن داخل نوار وضعیت این فرم را تغییر دهند. اگر در برنامه از یک خاصیت برای تغییر و یا دسترسی به متن نوار وضعیت استفاده کنیم، آن افراد در آینده صرفنظر از اینکه ما، از چه نوار ابزاری در ویرایشگر متن خود استفاده می کنیم و یا بدون توجه به اینکه آن نوار ابزار به چه صورت متن را نمایش می دهد، می توانند با استفاده از این خاصیت متن داخل نوار ابزار را تغییر دهند.
دلیل اینکه این خاصیت را از نوع public تعریف کرده ام نیز به همین مورد برمی گردد. در حقیقت عبارت public به این معنی است که کدهای دیگری که در خارج از این فایل هستند، می توانند به این خاصیت دسترسی داشته باشند و از آن استفاده کنند. اگر نخواهید برنامه های دیگر این خاصیت را تغییر دهند، باید این خاصیت را از نوع private تعریف کنید. در فصول ۹ تا ۱۳ بیشتر با این مفاهیم آشنا خواهیم شد.
در قسمتهای بعدی این برنامه، مشاهده خواهید کرد که بعضی از متدها و یا خاصیت ها به صورت private و برخی دیگر به صورت public تعریف شده اند. به این ترتیب متوجه می شوید که کدامیک به وسیله برنامه نویسان دیگر نیز قابل استفاده هستند و کدامیک نمی توانند توسط آنان مورد استفاده قرار بگیرند.

ایجاد قسمت ویرایش متن:

در بخش امتحان کنید بعد، TextBoxای را در فرم قرار می دهیم که کاربر بتواند متن مورد نظر خود را در آن وارد کند. هر کنترل TextBox دارای خاصیتی به نام Multiline است که به صورت پیش فرض برابر با False است. این خاصیت مشخص می کند که آیا کنترل می تواند بیش از یک خط متن را در خود جای دهد یا خیر. اگر مقدار این خاصیت را برابر با true قرار دهید، می توانید اندازه ی TextBox را به هر اندازه ای که بخواهید تغییر دهید و کنترل نیز می تواند هر چند خط متن که در آن وارد شود را نمایش دهد.

امتحان کنید: ایجاد قسمت ویرایش متن

۱) به قسمت مربوط به طراحی فرم بروید و با استفاده از جعبه ابزار ویژوال استودیو یک کنترل TextBox بر روی فرم قرار دهید.
۲) خاصیتهای کنترل را مطابق با مقادیر زیر تنظیم کنید:

- خاصیت Name آن را برابر با txtEdit قرار دهید.
- خاصیت Dock آن را برابر با Fill قرار دهید.

[1] یکی دیگر از خاصیتهای پر کاربرد همانند Dock، خاصیت Anchor است که فاصله یک کنترل را با بعضی از لبه های فرم به اندازه مشخصی نگه می دارد. برای اطلاعات بیشتر در مورد این خاصیت به راهنمای ویژوال استودیو مراجعه کنید.

[2] Abstraction

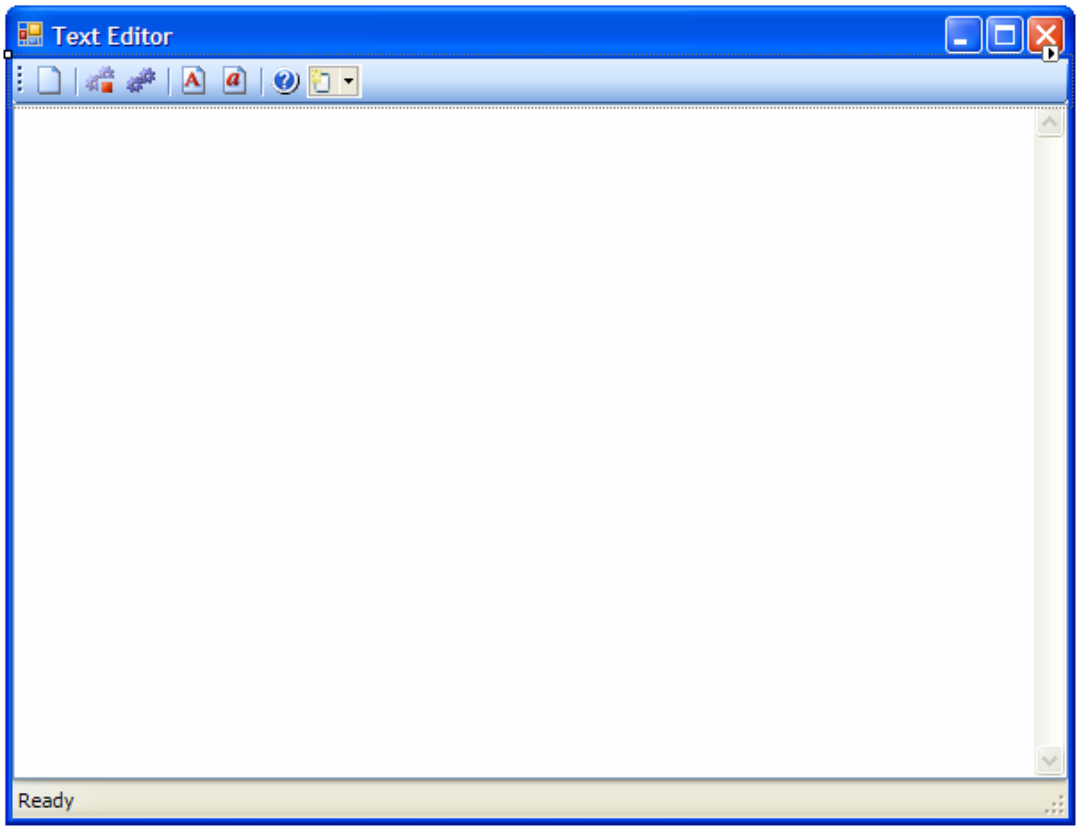
- خاصیت `MultiLine` را برابر با `True` قرار دهید.
- خاصیت `ScrollBars` را برابر با `Vertical` قرار دهید.

بعد از این موارد فرم برنامه شما باید مشابه شکل ۶-۱۲ باشد.

پاک کردن بخش ویرایشگر متن:

در بخش امتحان کنید بعدی، خاصیتی به نام `EditText` ایجاد خواهید کرد که به وسیله آن بتوانید متن موارد شده در `TextBox` را دریافت کرده و یا آن را تغییر دهید. به این ترتیب پاک کردن متن داخل `TextBox` بسیار ساده خواهد بود، کافی است که این خاصیت را برابر با یک رشته خالی مانند `String.Empty` قرار دهید.

امتحان کنید: پاک کردن متن داخل ویرایشگر



شکل ۶-۱۲

۱) به قسمت ویرایشگر کد برنامه بروید و کد زیر را در آن وارد کنید:

```
// Gets or sets the text that you are editing
public string EditText
{
     get
     {
          return txtEdit.Text;
     }
     set
     {
          txtEdit.Text = value;
     }
}
```

همانند قسمت قبل، با تعریف یک خاصیت برای تنظیم متن داخل ویرایشگر به برنامه نویسان دیگری که بخواهند از این برنامه استفاده کنند اجازه می دهیم صرفنظر از اینکه چگونه متن داخل برنامه تغییر می کند، از این خاصیت استفاده کرده و متن را تغییر دهند. البته این نکته در برنامه های ویندوزی کاربرد کمتری دارد. این نکته بیشتر در طراحی کلاسها، کنترل ها و یا کامپوننت ها به کار می رود، همانند کنترل هایی که هم اکنون از آن استفاده می کنید مانند Button. در فصول بعدی با این موارد بیشتر آشنا خواهید شد.

۲) حال می توانید متدی ایجاد کنید که با استفاده از این خاصیت، متن داخل ویرایشگر را پاک کند. برای این کار کد زیر را به برنامه اضافه کنید:

```
// Clears the txtEdit control
public void ClearEditBox()
{
     // Set the EditText property
     EditText = String.Empty;
     // Reset the font color
     txtEdit.ForeColor = Color.Black;
     // Set the status bar text
     StatusText = "Text box cleared!";
}
```

۳) به قسمت طراحی فرم برگردید و کنترل TextBox را در فرم انتخاب کنید. با دو بار کلیک بر روی این کنترل، متد مربوط به رویداد TextChanged را ایجاد کرده و کد زیر را در آن وارد کنید:

```
private void txtEdit_TextChanged(object sender, EventArgs
e)
{
     // Reset the status bar text
     StatusText = "Ready";
}
```

چگونه کار می کند؟

اولین کاری که باید انجام دهید، پاک کردن `TextBox` است. در بخش بعدی مشاهده خواهید کرد که چگونه می توان با استفاده از `ClearEditBox` در نوار ابزار، متن داخل `TextBox` را پاک کرد.
تنها کاری که این متد انجام می دهد این است که خاصیت `EditText` را برابر با یک رشته خالی مانند عضو `Empty` از کلاس `String` قرار دهد. سپس خاصیت `ForeColor` از `TextBox` را برابر با رنگ سیاه قرار می دهد. این خاصیت رنگ متن موجود در `TextBox` را مشخص می کند. در انتها نیز عبارت `Text box cleared` را در نوار ابزار می نویسد تا مشخص کند که متن داخل ویرایشگر پاک شده است.

```
// Clears the txtEdit control
public void ClearEditBox()
{
     // Set the EditText property
     EditText = String.Empty;
     // Reset the font color
     txtEdit.ForeColor = Color.Black;
     // Set the status bar text
     StatusText = "Text box cleared!";
}
```

همانطور که گفتم استفاده از خاصیت `EditText` باعث می شود تا برنامه نویسانی که بخواهند از این برنامه استفاده کنند، بتوانند بدون توجه به اینکه شما چگونه متن داخل برنامه را تغییر می دهید این کار را انجام دهند.

```
// Gets or sets the text that you are editing
public string EditText
{
     get
     {
          return txtEdit.Text;
     }
     set
     {
          txtEdit.Text = value;
     }
}
```

پاسخ به رویدادهای نوار ابزار:

در بخش امتحان کنید بعد، نوشتن کدهای مربوط به رویداد کلیک دکمه های موجود در نوار ابزار را شروع خواهیم کرد. در فصل هشتم هنگامی که با ایجاد منو در برنامه ها آشنا شدید، متوجه خواهید شد که در بیشتر برنامه ها دکمه های موجود در نوار ابزار همان

کاری را انجام می دهند که منو ها انجام می دهند. برای مثال عملکرد دکمه New در نوار ابـزار معـادل عملکـرد گزینـه New در منوی File خواهد بود. به این ترتیب خواهید توانست برای هر دوی این موارد از فراخوانی متدهای یکسان استفاده کنید.

امتحان کنید: پاسخ دادن به رویداد کلیک دکمه های فرمان موجود در نوار ابزار

۱) به قسمت طراحی فرم بروید و در نوار ابزار بالای فرم، بر روی دکمه ی tbrClear (اولین دکمه فرمان در نوار ابـزار) دو بار کلیک کنید تا رویداد مربوط به متد Click این کنترل ایجاد شود. سپس کد مشخص شده در زیر را به ایـن متـد اضافه کنید:

```
private void tbrClear_Click(object sender, EventArgs e)
{
     // Clear the edit box
     ClearEditBox();
}
```

۲) برای کنترل Button دوم باید زیربرنامه ای بنویسید که متن داخل TextBox را به رنگ قرمز درآورد و این مورد را در نوار وضعیت نیز اعلام کند. بنابراین متد زیر را در بخش ویرایشگر کد وارد کنید:

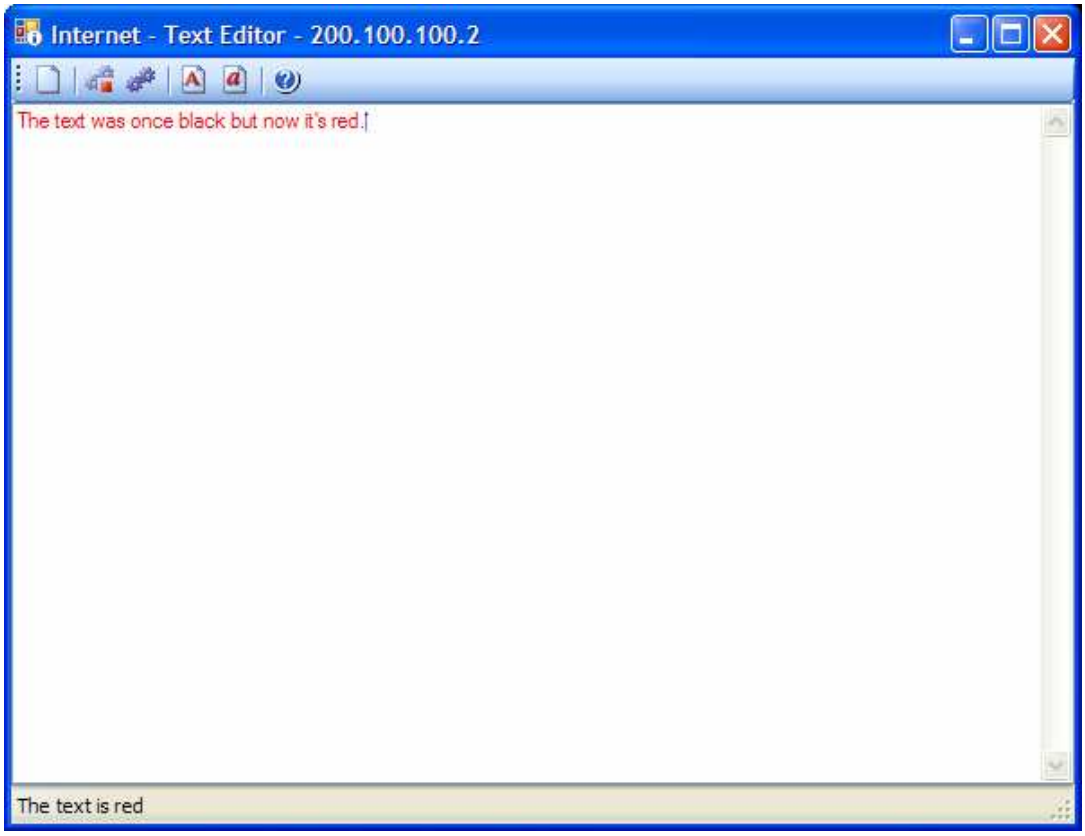
```
public void RedText()
{
     // Make the text red
     txtEdit.ForeColor = Color.Red;

     // Update the status bar text
     StatusText = "The text is red";
}
```

۳) حال به قسمت طراحی فرم برگشته، دکمه ی tbrRed را در نوار ابزار انتخاب کرده و بر روی آن دو بـار کلیـک کنیـد. سپس در متد ایجاد شده برای رویداد Click این کنترل، کد زیر را بنویسید:

```
private void tbrRed_Click(object sender, EventArgs e)
{
     // Make the text red
     RedText();
}
```

برنامه را اجرا کرده و متنی را در TextBox وارد کنید. سپس بر روی دکمه Red در نوار ابزار کلیک کنید. مـشاهده خواهید کرد که متن وارد شده همانند شکل ۶-۱۳ به رنگ قرمز درمی آید. اگر بعد از آن متن، متنی را بـه برنامـه اضـافه کنید، رنگ آن نیز قرمز خواهد بود.



شکل ۶-۱۳

۴) حال بر روی دکمه ی `Clear` کلیک کنید. مشاهده خواهید کرد که متن وارد شده از برنامه پاک می شود و رنـگ مـتن نیز به حالت اولیه بازمی گردد. به این ترتیب اگر متنی را در برنامه وارد کنید رنگ آن سیاه خواهد بود.

۵) برنامه را متوقف کنید و به قسمت ویرایشگر کد برگردید. کد زیر را به این قسمت اضافه کنید تا زیربرنامه ای ایجاد شود که رنگ متن را به آبی تغییر دهد:

```
public void BlueText()
{
     // Make the text blue
     txtEdit.ForeColor = Color.Blue;

     // Update the status bar text
     StatusText = "The text is blue";
}
```

۶) در قسمت طراحی فرم، دکمه ی `tbrBlue` را از نوار ابزار انتخاب کرده و بر روی آن دو بـار کلیـک کنیـد تـا رویـداد کلیک آن ایجاد شود. کد مشخص شده در زیر را به این متد اضافه کنید:

```
private void tbrBlue_Click(object sender, EventArgs e)
{
```

```
    // Make the text blue
    BlueText();
}
```

۷) حال به زیر برنامه ای نیاز داریم که متن داخل برنامه را به حروف بزرگ تبدیل کند. برای این کار کد زیر را به برنامه اضافه کنید:

```
public void UppercaseText()
{
    // Make the text uppercase
    EditText = EditText.ToUpper();

    // Update the status bar text
    StatusText = "The text is all uppercase";
}
```

۸) در قسمت طراحی فرم، بر روی دکمه ی tbrUpperCase در نوار ابزار دو بار کلیک کنید تا متد مربوط به رویـداد کلیک آن ایجاد شود. سپس کد زیر را در آن متد وارد کنید:

```
private void tbrUpperCase_Click(object sender, EventArgs e)
{
    // Make the text uppercase
    UppercaseText();
}
```

۹) برای تغییر متن برنامه به حروف کوچک نیز به متد زیر نیاز داریم:

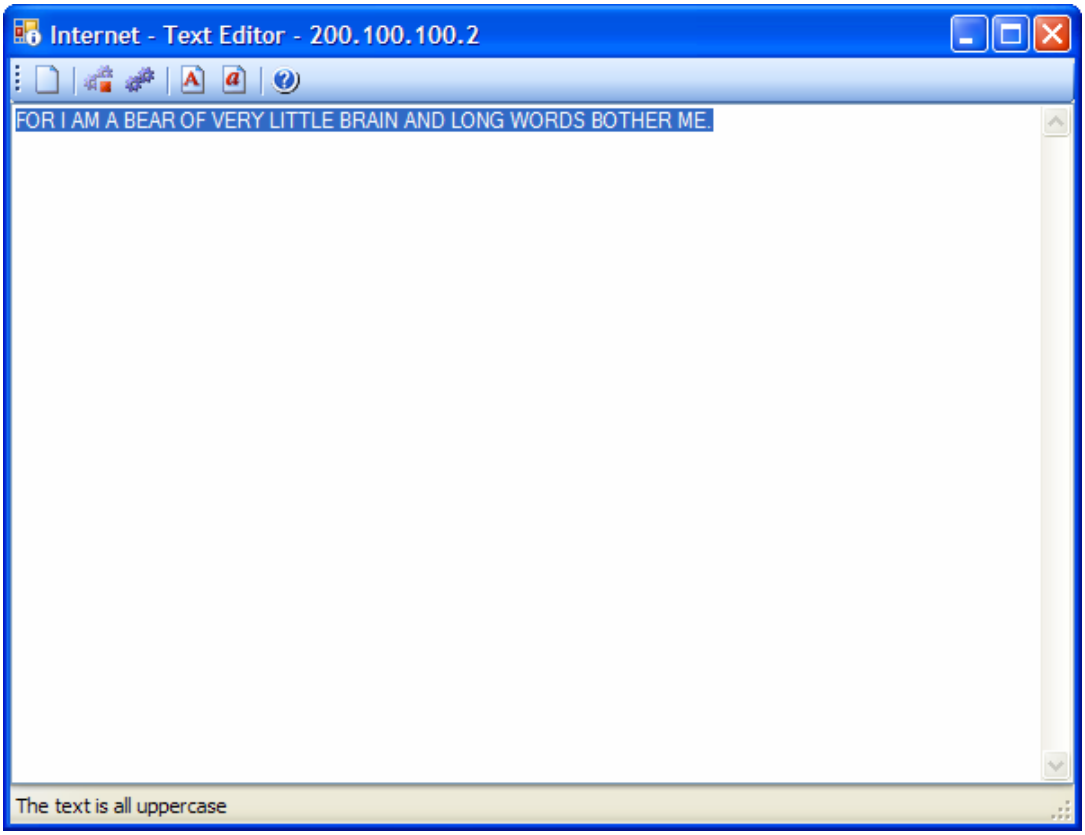
```
public void LowercaseText()
{
    // Make the text lowercase
    EditText = EditText.ToLower();

    // Update the status bar text
    StatusText = "The text is all lowercase";
}
```

۱۰) متد مربوط به رویداد کلیک دکمه ی tbrLowerCase را ایجاد کرده و کد زیر را در آن وارد کنید:

```
private void tbrLowerCase_Click(object sender, EventArgs e)
{
    // Make the text lowercase
    LowercaseText();
}
```

۱۱) برنامه را اجرا کرده و متنی را در آن وارد کنید که ترکیبی از حروف کوچک و بزرگ باشد. سپس بر روی دکمه ی Upper Case کلیک کنید. مشاهده می کنید که متن وارد شده همانند شکل ۶-۱۴ به حروف بزرگ تبدیل می شود. به همین ترتیب اگر بر روی دکمه Lower Case کلیک کنید، متن به حروف کوچک تبدیل می شود. کلیک کردن بر روی دکمه های Red و Blue نیز باعث تغییر رنگ برنامه می شود. در انتها می توانید بر روی دکمه Clear کلیک کنید تا متن موجود در برنامه پاک شده و برنامه به حالت اول برگردد.



شکل ۶-۱۴

چگونه کار می کند؟

این بخش از امتحان کنید بسیار ساده بود. در برنامه های قبلی چگونگی ایجاد متدی برای رویداد Click یک کنترل Button را مشاهده کرده بودید، ایجاد رویداد Click برای کنترلهای Button موجود در نوار ابزار نیز کاملاً مشابه موارد قبلی است. اولین کاری که انجام دادیم، ایجاد رویداد Click برای دکمه ی Clear و اضافه کردن کدی بود که زیربرنامه ClearEditBox را فراخوانی کند:

```
private void tbrClear_Click(object sender, EventArgs e)
{
     // Clear the edit box
     ClearEditBox();
```

```
}
```

در مرحله بعد، زیربرنامه ای نوشتیم که رنگ متن وارد شده در برنامه را به قرمز تغییر دهد و این تغییر را در نوار وضعیت نیز اعلام کند. برای تغییر رنگ متن وارد شده در TextBox باید از خاصیت ForeColor استفاده کنیم. برای تعیین رنگ مورد نظر نیز می توانیم از شمارنده ی Color استفاده کنیم. خاصیت ForeColor به رنگ قرمز باقی می ماند تا آن را مجدداً تغییر دهیم. بنابراین تا بر روی دکمه فرمان Clear کلیک نکنیم، رنگ متن برنامه به صورت قرمز خواهد بود. کلیک کردن بر روی دکمه Clear باعث می شود متنی که از آن به بعد در برنامه وارد می شود. به رنگ سیاه نمایش داده شود:

```
public void RedText()
{
      // Make the text red
      txtEdit.ForeColor = Color.Red;

      // Update the status bar text
      StatusText = "The text is red";
}
```

بعد از اینکه رنگ متن موجود در برنامه را به قرمز تغییر دادیم، این تغییر در نوار وضعیت نیز نشان داده می شود. هنگامی که کاربر متنی را در برنامه تایپ کند، متن نوار وضعیت نیز به Ready تغییر می کند. زیرا در این حالت متن داخل TextBox تغییر کرده و بنابراین رویداد TextChanged مربوط به کنترل TextBox فراخوانی می شود. این رویداد نیز متن نوار وضعیت را به Ready تغییر می دهد.
اگر کاربر بر روی دکمه فرمان Upper Case در نوار ابزار کلیک کند، برنامه متد UppercaseText را فراخوانی می کند، این متد نیز با استفاده از تابع ToUpper متن موجود در خاصیت EditText را به حروف بزرگ تبدیل می کند:

```
    // Make the text uppercase
    EditText = EditText.ToUpper();
```

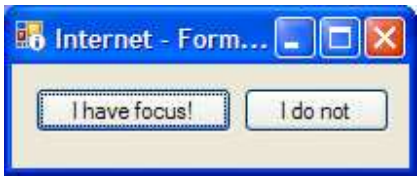
به همین ترتیب کلیک کردن بر روی دکمه ی Lower Case باعث می شود که متن موجود در برنامه به حروف کوچک تبدیل شود:

```
    // Make the text lowercase
    EditText = EditText.ToLower();
```

تمامی این زیربرنامه ها به وسیله رویداد کلیک مربوط به دکمه ی مربوط به خودشان در نوار ابزار فراخوانی می شوند و بعد از اینکه تغییر مورد نظر را در متن ایجاد کردند، متن موجود در نوار وضعیت را نیز به گونه ای تغییر می دهند که نشان دهنده ی تغییر ایجاد شده در برنامه باشد.

مفهوم فوکوس:

اگر هنگام اجرای برنامه دقت کنید، مشاهده می کنید که هنگامی که برنامه شما در حال اجرا است و شما به برنامه دیگری می روید و سپس به همین برنامه باز میگردید، تمام متن وارد شده در جعبه متنی به صورت انتخاب شده در می آید. این مورد به این دلیل است که **فوکوس**[1] برنامه بر روی کنترل TextBox قرار گرفته است. در اصطلاح، هنگامی که یک کنترل در یک برنامه انتخاب شود، گفته می شود که فوکوس بر روی آن کنترل قرار گرفته است. برای مثال به شکل ۶-۱۵ دقت کنید. در این شکل دو کنترل Button بر روی فرم قرار گرفته اند. همانطور که می بینید در این فرم یکی از کنترل ها به صورت انتخاب شده است. بنابراین اگر کلید Enter فشار داده شود، همانند این خواهد بود که با ماوس بر روی دکمه ی انتخاب شده کلیک شود و کد مربوط به رویداد Click آن اجرا می شود.



شکل ۶-۱۵

اگر چندین کنترل TextBox در یک فرم قرار داشته باشند، هنگامی که متنی را در برنامه تایپ کنید، متن در کنترلی نوشته می شود که دارای فوکوس است.

برای این که بین کنترل های موجود در یک فرم جا به جا شوید و کنترل دیگری را به صورت انتخاب شده در آورید می توانید از کلید Tab در برنامه استفاده کنید. برای مثال در فرم نمایش داده شده در شکل ۶-۱۵ اگر کاربر کلید Tab را فشار دهد، دکمه فرمان "I do not" به صورت انتخاب شده در می آید. با فشار مجدد کلید Tab فوکوس به دکمه فرمان "I Have Focus" برمی گردد.

این که در یک برنامه با فشار کلید Tab فوکوس چگونه بین کنترلهای موجود در فرم جا به جا شود به صورت اتفاقی انتخاب نمی شود. هنگامی که در طراحی فرم، کنترلی را بر روی فرم قرار می دهید عددی به خاصیت TabIndex آن کنترل نسبت داده می شود. برای اولین کنترل این عدد صفر خواهد بود، برای کنترل دوم یک، برای کنترل سوم عدد دو و .... بنابراین اگر هنگام اجرای برنامه کلید Tab را فشار دهید، فوکوس به ترتیب قرار گرفتن کنترل ها بر روی فرم بین آنها جا به جا می شود. البته هنگام طراحی فرم می توانید عدد موجود در خاصیت TabIndex را تغییر دهید تا فوکوس به شکلی که تمایل دارید بین کنترل ها حرکت کند.

نکته: با وجود اینکه کنترل لیبل دارای خاصیت TabIndex است، اما هنگام اجرای برنامه این کنترل نمی تواند دارای فوکوس باشد. بنابراین فوکوس به کنترل بعدی مانند TextBox و یا Button منتقل می شود.

تغییر عدد موجود در خاصیت TabIndex برای تنظیم تغییر فوکوس بین کنترل ها کار مشکلی است. ویژوال استودیو ۲۰۰۵ دارای ابزاری است که به وسیله آن می توان این کار را بسیار راحت تر انجام داد. گزینه View ➔ Tab Order را در محیط ویژوال استودیو انتخاب کنید، به این ترتیب فرم شما مشابه شکل ۶-۱۶ خواهد شد.

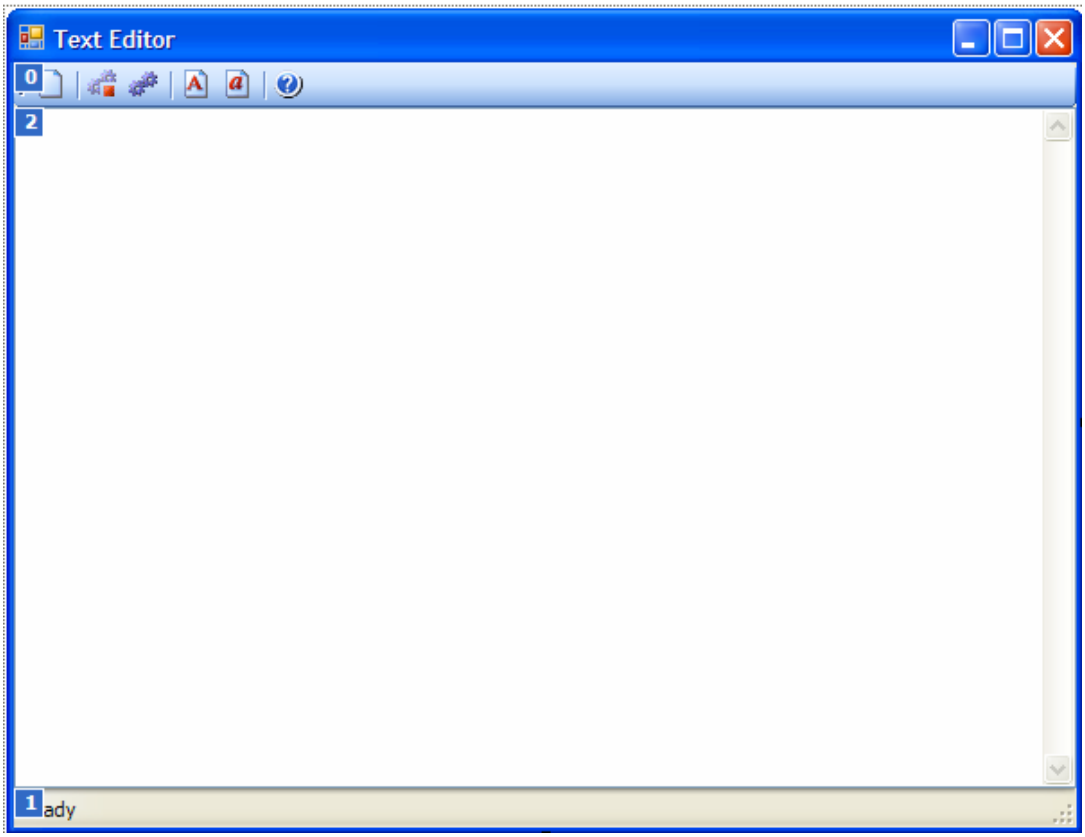
اعدادی که در کنار کنترل ها نمایش داده می شوند، ترتیب تغییر فوکوس بین کنترل ها را نمایش می دهند. این اعداد به ترتیب قرار گرفتن کنترل ها بر روی فرم تنظیم شده اند. برای اینکه این ترتیب را خودتان مشخص کنید، بر روی این اعداد به ترتیبی که می

[1] Focus

خواهید فوکوس تغییر کند کلیک کنید. بعد از مشخص کردن ترتیب تغییر فوکوس مجددا گزینه View ➔ Tab Order را از نوار منوی ویژوال استودیو انتخاب کنید تا اعداد نمایش داده شده از کنار کنترل ها حذف شوند.

استفاده از چندین فرم در برنامه:

تمام برنامه های ویندوزی دارای دو نوع پنجره هستند: پنجره های معمولی و کادرهای محاوره ای. یک پنجره معمولی، صفحه اصلی رابط کاربر را در برنامه نمایش می دهد. برای مثال در برنامه Word، کاربر از یک پنجره عادی که صفحه اصلی برنامه را تشکیل می دهد برای وارد کردن و یا ویرایش متن استفاده می کند.



شکل ۶-۱۶

هنگامی که بخواهید عمل خاصی را در برنامه انجام دهید، یک کادر محاوره ای توسط برنامه نمایش داده می شود. این نوع از پنجره ها تا زمانی که بسته نشده اند به کاربر اجازه نمی دهند که به صفحه اصلی برنامه دسترسی پیدا کند. برای مثال اگر در برنامه Word گزینه Print را انتخاب کنید، یک کادر محاوره ای نمایش داده می شود و تا زمانی که با کلیک کردن بر روی دکمه های OK و یا Cancel کادر را نبندید، نخواهید توانست متن داخل سند Word را تغییر دهید. کادرهایی که به این صورت در یک برنامه نمایش داده می شوند، به فرمهای مقید[1] و یا فرمهای مودال معروف هستند.

[1] Modal Forms

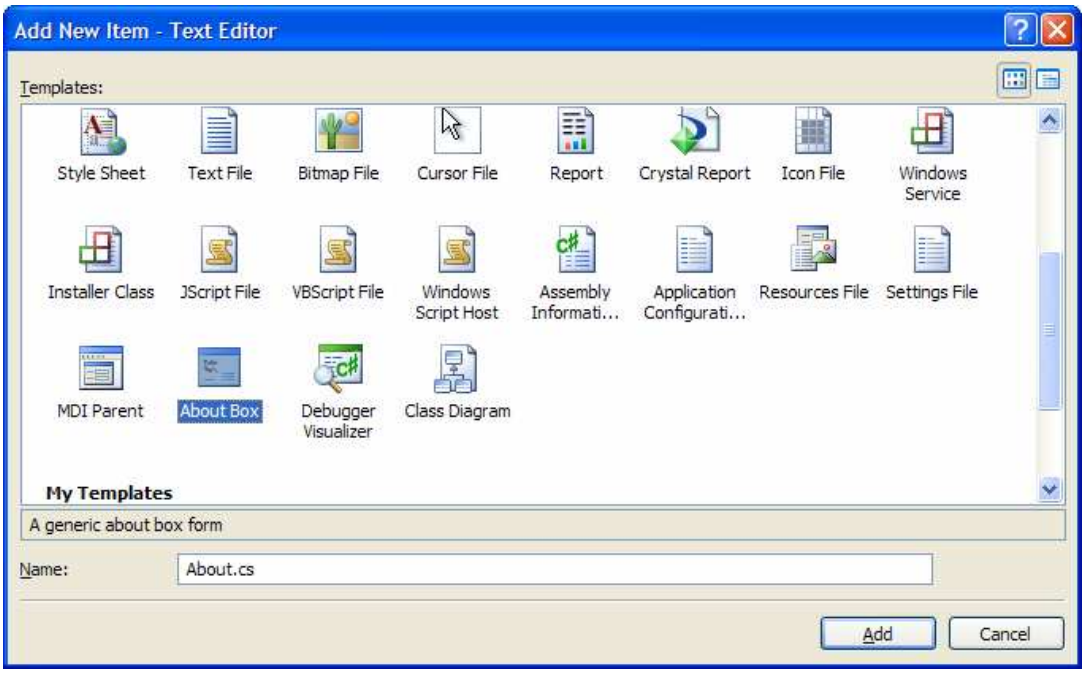
کادرهای محاوره ای همانند کادر مربوط به چاپ و یا کادرهای مشابه با آن، در فصل هفتم بررسی خواهند شد. در این بخش بر روی اضافه کردن فرمهای عادی به برنامه تمرکز خواهیم کرد و در تمرین بعد، یک فرم ساده را به صورت مودال نمایش خواهیم داد.

فرم About:

بیشتر برنامه ها دارای یک کادر محاوره ای About هستند که نام برنامه و همچنین حقوق مولف آن را شرح می دهد. در برنامه قبلی یک کنترل Button برای این قسمت در نوار ابزار قرار دادیم. حال فرم مربوط به آن را ایجاد خواهیم کرد.

امتحان کنید: اضافه کردن فرم About

۱) برای اضافه کردن یک فرم به برنامه می توانید از پنجره Solution Explorer استفاده کنید. برای این کار در این پنجره بر روی نام پروژه ی Text Editor کلیک راست کنید و از منوی باز شده گزینه Add ➔ Windows Forms را انتخاب کنید. در پنجره Add New Item – Text Editor همانند شکل ۶-۱۷ در قسمت Templates گزینه About Box را انتخاب کرده و سپس در کادر Name نام About.cs را وارد کنید. در انتها بر روی دکمه Add کلیک کنید تا فرم جدید به برنامه اضافه شود.

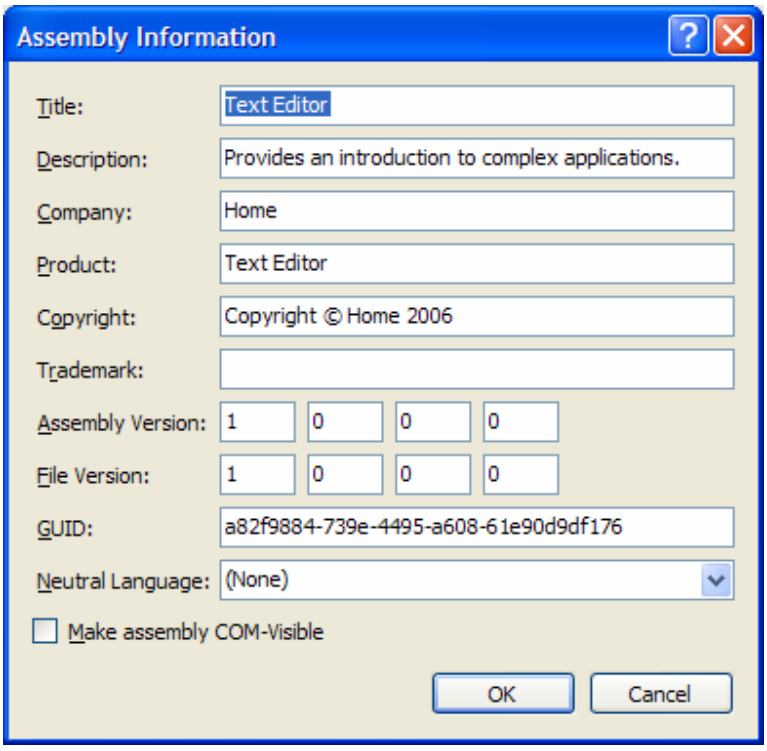


شکل ۶-۱۷

۲) هنگامی که قسمت طراحی فرم برای پنجره About نمایش داده شود، مشاهده خواهید کرد که تمام قسمتهایی که در یک کادر About معمولی وجود دارد، در این فرم نیز قرار داده شده است. برای مثال فرم، دارای قسمتهایی برای قرار دادن نام برنامه، نسخه ی برنامه، اطلاعات حقوق مولف و ... است.

۳) بر روی فرم کلیک راست کنید و گزینه View Code را انتخاب کنید. مشاهده خواهید کرد که تابع سازنده ی مربوط به کلاس این فرم دارای چندین خط کد است که هنگام نمایش داده شدن فرم، اطلاعات برنامه را در کنترلهای موجود در فرم نمایش می دهند. در ابتدای کدها توضیحاتی نوشته شده است تا به برنامه نویس اطلاع دهد که برای نمایش درست اطلاعات در فرم About، باید اطلاعات برنامه را در قسمت Assembly Information برنامه وارد کند.

۴) بر روی نام پروژه در پنجره Solution Explorer کلیک راست کرده و از منوی باز شده گزینه Properties را انتخاب کنید. صفحه ی Application از پنجره Properties مربوط به پروژه نمایش داده می شود. در این صفحه بر روی دکمه ی Assembly Information کلیک کنید تا کادر Assembly Information نمایش داده شود. اطلاعات نوشته شده در کادرهای موجود در این پنجره را همانند شکل ۶-۱۸ تغییر دهید و سپس بر روی دکمه OK کلیک کنید تا پنجره بسته شود.



شکل ۶-۱۸

۵) حال در فرم اصلی برنامه (فرم TextEditor.cs) به زیربرنامه ای نیاز دارید که کادر About را نمایش دهد. برای این کار قسمت ویرایشگر کد مربوط به فرم TextEditor را باز کرده و کد زیر را در آن وارد کنید:

```
public void ShowAboutBox()
{
     // Display the About dialog box
     About objAbout = new About();
     objAbout.ShowDialog(this);
}
```

۶) در انتها باید کدی بنویسید که به وسیله ی آن هنگام کلیک شدن بر روی دکمه About در نوار ابـزار، فـرم About نمـایش داده شـود. بـرای ایـن کـار در قـسمت طراحـی فـرم مربـوط بـه TextEditor بـر روی کنتـرل tbrHelpAbout در نوار ابزار دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس کد زیـر را در این متد وارد کنید:

```
private void tbrHelpAbout_Click(object sender, EventArgs e)
{
    // Display the about dialog box
    ShowAboutBox();
}
```

۷) برنامه را اجرا کرده و بر روی دکمه ی About در نوار ابزار کلیک کنید. مشاهده می کنید که کادری هماننـد شـکل ۶-۱۹ نمایش داده می شود.



شکل ۶-۱۹

چگونه کار می کند؟

در زبان C# هر فرم، از یک کلاس هم نام با فرم تشکیل شده است. برای نمایش فرم ابتدا باید یک شیئ از آن کلاس ایجـاد کـرد. همانطور که در متد ShowAboutBox مشاهده می کنید، ابتدا از کلاس مربوط به فرم About یـک شـیئ جدیـد ایجـاد کرده ایم و سپس با استفاده از متد ShowDialog از شیئ ایجاد شده، فرم را به صـورت مـودال در صـفحه نمـایش داده ایـم. هنگامی که عبارت this را به عنوان پارامتر به این متد می فرستید، در حقیقت مشخص می کنید که کادر محاوره ای About

مربوط به فرم TextEditor است و تا زمانی که این کادر بسته نشده است کاربر نمی تواند به فرم TextEditor دسترسی داشته باشد.

```
public void ShowAboutBox()
{
      // Display the About dialog box
      About objAbout = new About();
      objAbout.ShowDialog(this);
}
```

برای نمایش فرم About باید در رویداد Click مربوط به کنترل tbrHelpAbout زیر برنامه نوشته شده را فراخوانی کنیم:

```
private void tbrHelpAbout_Click(object sender, EventArgs e)
{
    // Display the about dialog box
    ShowAboutBox();
}
```

در ویژوال استودیو فرمهای از پیش طراحی شده زیادی وجود دارد که باعث تسریع در طراحی برنامه ها می شود. یکی از این فرم ها همانطور که در برنامه قبلی مشاهده کردید، پنجره About است که با اضافه کردن آن به برنامه می توانید کادر About را در برنامه نمایش دهید.
هنگامی که صفحه About بخواهد در حافظه قرار بگیرد، تابع سازنده آن اجرا می شود و این تابع نیز به طور اتوماتیک حاوی کدی است که قسمتهای مختلف فرم About را بر اساس اطلاعات برنامه تکمیل می کند.
همانطور که در قسمتهای قبل گفتم[1] یک برنامه چه تولید کننده ی یک فایل اجرایی (EXE) باشد، مانند برنامه هایی که تاکنون ایجاد کرده ایم و چه محتوی کلاسها و توابعی برای استفاده در برنامه های دیگر باشد مانند برنامه هایی که در فصول ۱۲ و ۱۳ ایجاد خواهیم کرد، بعد از کامپایل در فایلهایی که به اسمبلی معروف هستند ذخیره می شود. همچنین ذکر شد که این فایلها حاوی اطلاعاتی هستند که به اطلاعات متا معروف است و شامل مشخصات اسمبلی می شود. حال اگر به کدهای موجود در تابع سازنده این فرم نگاهی بیندازید، متوجه می شوید که این کدها با استفاده از کلاس Assembly در فضای نام System.Reflection به اطلاعات متای اسمبلی برنامه دسترسی پیدا کرده و مشخصات لازم برای صفحه About را از آنها استخراج می کنند. سپس این اطلاعات را در قسمتهای مناسب در صفحه About نمایش می دهند. همانطور که مشاهده کردید با استفاده از این فرم ها در ویژوال استودیو، می توان به راحتی قسمتهای مختلف برنامه را تکمیل کرد.

نتیجه:

[1] رجوع کنید به فصل دوم "چارچوب .NET و ارتباط آن با C#"

در این فصل به معرفی ویژگیهای پیشرفته تر فرم ها در برنامه و نیز کنترلهای موجود در ویژوال استودیو پرداختیم. طبیعت رویداد گرای ویندوز را بررسی کردیم و با سه رویداد پر کاربرد کنترل دکمه فرمان (Click، MouseEnter و MouseLeave) آشنا شدیم.
در ابتدا برنامه ساده ای ایجاد کردید که به شما اجازه می داد متنی را وارد کنید و سپس برنامه تعداد کلمات و یا تعداد کاراکترهای متن را نمایش می داد.
سپس توجه خود را بر روی برنامه ی پیچیده تری متمرکز کردیم و برنامه ای طراحی کردیم که به وسیله آن می توانستید رنگ و یا حالت حروف وارد شده در آن را تغییر دهید. در این پروژه مشاهده کردید که چگونه می توان با استفاده از نوار ابزار و نوار وضعیت، قابلیتهای برنامه را افزایش داد. همچنین فرم دیگری اضافه کردیم تا اطلاعات مختصری از قبیل نام برنامه، نسخه برنامه و حقوق مولف آن را نمایش دهد.
در پایان این فصل باید با موارد زیر آشنا شده باشید:

- نوشتن کدی که به رویدادهای مختلف یک کنترل پاسخ دهد.
- خاصیتهای یک کنترل را برای تنظیم ظاهر آن کنترل تغییر دهید.
- از کنترلهای ToolStrip و StatusStrip برای نمایش نوار ابزار و نوار وضعیت در برنامه استفاده کنید.
- در برنامه خود از بیش از یک فرم استفاده کنید.

تمرین:

تمرین ۱:

یک برنامه ویندوزی ایجاد کنید و به وسیله جعبه ابزار، دو دکمه فرمان بر روی فرم برنامه قرار دهید. متدی برای رویداد MouseUp دکمه فرمان اول به وجود آورید و در آن یک کادر پیغام نمایش دهید. همین کار را برای رویداد LostFocus دکمه فرمان دوم تکرار کنید. سپس برنامه را اجرا کنید و مشاهده کنید که این رویدادها چه هنگام رخ می دهند.

تمرین ۲:

یک برنامه ویندوزی ایجاد کنید و یک نوار ابزار و یک نوار وضعیت بر روی فرم برنامه قرار دهید. در قسمت پایین طراحی فرم بر روی کنترل ToolStrip کلیک راست کرده و گزینه Insert Standard Items را انتخاب کنید تا دکمه های فرمان استاندارد نوار ابزار به آن اضافه شوند. برای رویداد کلیک هر کدام از این دکمه های فرمان کدی را اضافه کنید که با نمایش پیغامی در نوار وضعیت مشخص کند کدام دکمه فرمان فشار داده شده است.

فصل هفتم: نمایش کادرهای محاوره ای

ویژوال C# ۲۰۰۵ دارای چندین کادر محاوره ای درونی است که می تواند در طراحی ظاهر برنامه، کمک زیادی کند. این کادرها، در حقیقت همان پنجره های عمومی هستند که در بیشتر برنامه های تحت ویندوز مشاهده کرده اید. به علاوه این کادرها دارای خاصیت ها و متدهای فراوانی هستند که به وسیله ی آنها می توانید این کادرها را با قسمتهای مختلف برنامه ی خود هماهنگ کنید.
در این فصل:

- با روشهای مختلف ایجاد یک کادر پیغام با آیکونها و یا دکمه های گوناگون آشنا خواهید شد.
- با نحوه ی ایجاد یک کادر `Open` که بتوانید برای دسترسی به فایلها از آن استفاده کنید آشنا خواهید شد.
- چگونگی ایجاد یک کادر `Save` که بتوانید از آن برای ذخیره اطلاعات برنامه استفاده کنید را مشاهده خواهید کرد.
- مشاهده خواهید کرد که چگونه می توان با استفاده از کادر `Font` به کاربر اجازه دهید فونت مورد نظر خود را انتخاب کند.
- با کادر `Color` و موارد استفاده از آن در برنامه آشنا خواهید شد.
- با استفاده از کادر `Print` قابلیتهای مربوط به امور چاپ را به برنامه اضافه خواهیم کرد.

در این فصل، این کادرهای محاوره ای را به تفصیل مورد بررسی قرار خواهیم داد و مشاهده خواهیم کرد که چگونه به وسیله آنها می توانیم برنامه هایی که دارای ظاهری حرفه ای تر هستند را طراحی کنیم.

کادر محاوره ای MessageBox:

همانطور که از ابتدای کتاب تا کنون متوجه شده اید، کادر `MessageBox` یکی از کادرهایی است که در اغلب برنامه ها مورد استفاده قرار می گیرد. از این کادر عموماً برای نمایش یک پیغام به کاربر و دریافت جواب کاربر به آن پیغام استفاده می شود. با وجود اینکه در برنامه های قبلی به صورت یکنواخت از این کادر استفاده می کردیم، اما این کادر می تواند بر اساس موقعیت برنامه دارای ظاهری متفاوت باشد. برای مثال می توانیم علاوه بر نمایش متن در آن، آیکون خاصی را نیز برای آن مشخص کنیم و یا دکمه های دیگری به جز دکمه `OK` در آن قرار دهیم.
در استفاده ی روزمره از برنامه های کامپیوتری، کادرهای پیغام گوناگونی را مشاهده می کنید که دارای آیکونهایی مانند آیکونهای شکل ۷-۱ هستند. در این بخش مشاهده خواهیم کرد که چگونه می توان از این آیکونها در کادرهای محاوره ای استفاده کرد.



شکل ۷-۱

هنگام ایجاد یک برنامه ویندوزی، در مواقعی نیاز دارید که موردی را به کاربر اطلاع دهید و یا به کاربر هشدار دهید که یک پیشامد غیر منتظره رخ داده است. برای مثال فرض کنید کاربر اطلاعاتی از برنامه را تغییر داده است و بدون ذخیره کردن تغییرات سعی در بستن برنامه دارد. در این حالت می توانید کادر پیغامی حاوی آیکون هشدار (سومین آیکون از چپ) و یا آیکون اطلاعات (اولین آیکون از چپ) و یک پیغام مناسب را به کاربر نمایش دهید و بگویید که در صورت بسته شده برنامه تمام اطلاعات ذخیره نشده از

بین می روند. همچنین می توانید دکمه های OK و Cancel را در کادر پیغام قرار دهید تا کاربر بتواند به بستن برنامه ادامه دهد و یا این عمل را لغو کند.
در مواردی مشابه مورد بالا، استفاده از کادر پیغام می تواند به تسریع طراحی برنامه کمک کند. زیرا به وسیله آن می توانید با نمایش یک کادر پیغام مناسب شامل آیکون و دکمه های مورد نظر، به کاربر اجازه دهید در مورد یک مسئله خاص تصمیم گیری کند. همچنین با استفاده از کادر پیغام در بخش مدیریت خطا در برنامه، می توانید خطاهای اتفاق افتاده در برنامه را با آیکون و دکمه های مناسب به کاربر اطلاع دهید.
قبل از اینکه استفاده از کادرهای پیغام گوناگون را در کد بررسی کنیم، ابتدا بهتر است با کلاس MessageBox آشنا شویم. همانطور که می دانید این کلاس دارای متدی به نام Show است که برای نمایش یک کادر پیغام به کار می رود. عنوان کادر پیغام، متن نمایش داده شده در آن، آیکونها و نیز دکمه های فرمان کادر پیغام همه به وسیله پارامترهای این متد مشخص می شوند. این مورد در ابتدا ممکن است مقداری پیچیده به نظر برسد، اما مشاهده خواهید کرد که استفاده از آن بسیار ساده است.

آیکونهای قابل استفاده در یک کادر پیغام:

آیکون های قابل استفاده در یک کادر پیغام را در شکل ۷–۱ مشاهده کردید. در جدول زیر چهار آیکون قابل استفاده در کادر پیغام آورده شده است. در حقیقت آیکون مورد استفاده در این قسمت از سیستم عامل دریافت می شود و فعلاً چهار آیکون برای این موارد در نظر گرفته شده است که برای هماهنگی بعضی از آنها دارای چند نام هستند:

نام عضو	توضیح
Asterisk	مشخص می کند که یک آیکون اطلاعات در کادر پیغام نمایش داده شود.
Information	مشخص می کند که یک آیکون اطلاعات در کادر پیغام نمایش داده شود.
Error	مشخص می کند که یک آیکون خطا در کادر پیغام نمایش داده شود.
Hand	مشخص می کند که یک آیکون خطا در کادر پیغام نمایش داده شود.
Stop	مشخص می کند که یک آیکون خطا در کادر پیغام نمایش داده شود.
Exclamation	مشخص می کند که یک آیکون هشدار در کادر پیغام نمایش داده شود.
Warning	مشخص می کند که یک آیکون هشدار در کادر پیغام نمایش داده شود.
Question	مشخص می کند که یک علامت سوال در کادر پیغام نمایش داده شود.
None	مشخص می کند که آیکونی در کادر پیغام نمایش داده نشود.

دکمه های موجود برای کادر پیغام:

در هر کادر پیغام می توانید یکی از چندین گروه دکمه ی موجود را نمایش دهید. در جدول زیر گزینه های قابل انتخاب برای این مورد شرح داده شده اند:

نام عضو	شرح
AbortRetryIgnore	مشخص می کند که کادر شامل دکمه هـای Abort، Retry و Cancel باشد.
OK	مشخص می کند که کادر شامل دکمه OK باشد.
OKCancel	مشخص می کند که کادر شامل دکمه های OK و Cancel باشد.
RetryCancel	مشخص می کند که کادر شامل دکمه های Retry و Cancel باشد.
YesNo	مشخص می کند که کادر شامل دکمه های Yes و No باشد.
YesNoCancel	مشخص می کند که کادر شامل دکمه های Yes و No و Cancel باشد.

تنظیم دکمه ی پیش فرض:

هنگام تنظیم ویژگیهای مختلف یک کادر پیغام برای نمایش، علاوه بر مشخص کردن دکمه های آن می توانید مـشخص کنیـد کـه کدام دکمه به عنوان پیش فرض در نظر گرفته شود. به عبارت دیگر با استفاده از این ویژگی مشخص می کنید که در بین دکمه های موجود در کادر، کدام دکمه باید دارای فوکوس باشد. با تنظیم این مورد می توانید به کاربر اجازه دهید که بعـد از خوانـدن مـتن کـادر پیغام، با فشار دادن کلید Enter و بدون حرکت ماوس، دکمه ی پیش فرض را انتخاب کند. برای تنظیم این مورد باید از شمارنده MessageBoxDefaultButton استفاده کنید که شرح گزینه های آن در جدول زیر آمده است:

نام عضو	شرح
Button 1	مشخص می کند که دکمه اول در کادر پیغام به عنوان دکمه پیش فرض در نظر گرفته شود.
Button 2	مشخص می کند که دکمه دوم در کادر پیغام به عنوان دکمه پیش فرض در نظر گرفته شود.
Button 3	مشخص می کند که دکمه سوم در کادر پیغام به عنوان دکمه پیش فرض در نظر گرفته شود.

ترتیب این دکمه ها از سمت چپ در نظر گرفته می شود. برای مثال اگر در کادر پیغام سه دکمه Yes و No و Cancel داشته باشید و دکمه سوم را به عنوان دکمه پیش فرض مشخص کنید، دکمه Cancel پیش فرض خواهد بود. همچنـین اگـر در کـادر پیغام دو دکمه Yes و No داشته باشید و دکمه سوم را به عنوان پیش فرض مشخص کنید، دکمه Yes پیش فرض خواهد بود.

گزینه های مختلف کادر پیغام:

هنگام کار با کادر پیغام علاوه بر گزینه های بالا، موارد دیگری نیز قابل تنظیم است که در شمارنده MessageBoxOptions قرار دارد. بعضی از موارد پر کاربرد که در این قسمت قابل تنظیم هستند، در جدول زیر توضیح داده شده اند:

نام عضو	شرح
RightAlign	مشخص می کند که متن داخل کادر پیغام باید از سمت راست نوشته شود. این حالت بر عکس حالت پیش فرض است که متن از سمت چپ نوشته می شود.
RTLReading	مشخص می کند که کادر پیغام باید برای نمایش متن راست به چپ، تنظیم شود. این حالت برای نمایش متن به زبانهایی مناسب است که از راست به چپ نوشته می شوند (مانند فارسی). برای مثال در این حالت آیکون کادر پیغام در سمت راست متن قرار می گیرد.

حالتهای مختلف استفاده از متد Show:

همانطور که می دانید برای نمایش یک کادر پیغام از متد Show کلاس MessageBox استفاده می کنیم. کدی که در زیر مشاهده می کنید متد Show را به گونه ای فراخوانی می کند که یک کادر پیغام مشابه شکل ۷-۲ را نمایش دهد. در این کد متنی که باید در کادر نمایش داده شود به عنوان پارامتر اول به متد فرستاده می شود و به دنبال آن نیز متنی که باید در نوار عنوان کادر قرار بگیرد وارد می شود. سپس دکمه هایی که باید به وسیله کادر نمایش داده شود و نیز آیکون کادر مشخص شده است. در انتها نیز مشخص شده است که کدام دکمه فرمان به عنوان پیش فرض در نظر گرفته شود.

```
MessageBox.Show("My Text", "My Caption",
     MessageBoxButtons.OKCancel,
MessageBoxIcon.Information,
     MessageBoxDefaultButton.Button1);
```



شکل ۷-۲

حال که با آیکونها و دکمه های قابل استفاده در کادر پیغام آشنا شدید، بهتر است به بررسی متد Show از کلاس MessageBox بپردازیم. این متد به چندین روش قابل استفاده است و برای فراخوانی آن می توانید پارامترهای گوناگونی را به آن ارسال کنید. برای مثال در برنامه هایی که از ابتدای کتاب تاکنون نوشته ایم، تنها متنی که باید نمایش داده می شد را به این متد می فرستادیم، اما در مثال قبل بیشتر جزئیات کادر پیغام را برای متد مشخص کردیم. پر کاربرد ترین نوعهای فراخوانی این متد در لیست زیر آمده است:

- `MessageBox.Show(Text)`
- `MessageBox.Show(Text,Caption)`
- `MessageBox.Show(Text,Caption,Button)`
- `MessageBox.Show(Text,Caption,Button,Icon)`
- `MessageBox.Show(Text,Caption,Button,Icon,DefaultButton)`

در این لیست پارامتر `Text` که یک پارامتر اجباری است، مشخص کننده متنی است که باید توسط کادر نمایش داده شود و می تواند یک مقدار ثابت و یا یک متغیر رشته ای باشد. بقیه پارامترهای این تابع به صورت اختیاری هستند:

- `Caption`: مشخص کننده متنی است که باید در نوار عنوان کادر نمایش داده شود. اگر این پارامتر به تابع فرستاده نشود، متنی در نوار عنوان نمایش داده نمی شود.
- `Button`: مشخص کننده مقداری از شمارنده `MessageBoxButtons` است. این پارامتر به شما این امکان را می دهد که تعیین کنید کدام دکمه ها در کادر نمایش داده شوند. اگر این پارامتر حذف شود، فقط دکمه `OK` در کادر نمایش داده می شود.
- `Icon`: مشخص کننده مقداری از شمارنده `MessageBoxIcon` است و برای تعیین آیکونی که باید در کادر نمایش داده شود به کار می رود. اگر این پارامتر حذف شود آیکونی در کارد پیغام نمایش داده نخواهد شد.
- `DefaultButton`: مشخص کننده مقداری از شمارنده `MessageBoxDefaultButton` است و برای تعیین دکمه فرمان پیش فرض در فرم به کار می رود. اگر این مقدار در فراخوانی تابع تعیین نشود، دکمه اول به عنوان دکمه فرمان پیش فرض در نظر گرفته می شود.

تمام حالتهای متد `Show` که در بالا ذکر شد مقداری را از نوع شمارنده `DialogResult` برمی گردانند. این مقدار مشخص می کند کدامیک از دکمه های کادر پیغام توسط کاربر انتخاب شده است. در جدول زیر تمام گزینه های شمارنده `DialogResult` مورد بررسی قرار گرفته است.

نام عضو	شرح
`Abort`	مقدار بازگشتی `Abort` است و مشخص می کند که کاربر بر روی دکمه `Abort` کلیک کرده است.
`Cancel`	مقدار بازگشتی `Cancel` است و مشخص می کند که کاربر بر روی دکمه `Cancel` کلیک کرده است.
`Ignore`	مقدار بازگشتی `Ignore` است و مشخص می کند که کاربر بر روی دکمه `Ignore` کلیک کرده است.
`No`	مقدار بازگشتی `No` است و مشخص می کند که کاربر بر روی دکمه `No` کلیک کرده است.
`None`	مقدار بازگشتی `None` است. به عبارت دیگر هنوز گزینه ای از کادر پیغام توسط کاربر انتخاب نشده است.
`OK`	مقدار بازگشتی `Cancel` است و مشخص می کند که کاربر بر روی دکمه `Cancel` کلیک کرده است.

Retry	مقدار بازگشتی Retry است و مشخص می کند که کاربر بر روی دکمه Retry کلیک کرده است.
Yes	مقدار بازگشتی Yes است و مشخص می کند که کاربر بر روی دکمه Yes کلیک کرده است.

کادرهای پیغام نمونه:

در مواردی که فقط یک دکمه در کادر پیغام به کار می رود نیازی به بررسی نتیجه کادر پیغام نداریم، اما اگر در کادر پیغام از بیش از یک دکمه استفاده کنیم بعد از نمایش باید نتیجه را نیز بررسی کنیم. در بخش امتحان کنید بعد، مشاهده خواهیم کرد که چگونه می توان یک کادر پیغام با بیش از یک دکمه نمایش داد و سپس مشخص کرد که کاربر کدام دکمه را انتخاب کرده است.

امتحان کنید: ایجاد کادر پیغام با دو دکمه

۱) ویژوال استودیو ۲۰۰۵ را اجرا کرده و از نوار منو، گزینه File ➔ New ➔ Project... را انتخاب کنید. در پنجره ی New Project از قسمت Templates گزینه Windows Application را انتخاب کنید و در بخش Name نام Simple MessageBox را وارد کنید. سپس روی دکمه OK کلیک کنید تا پروژه ایجاد شود.

۲) بر روی فرم برنامه در قسمت طراحی فرم کلیک کرده و سپس خاصیت Text آن را به Simple MessageBox تغییر دهید.

۳) با استفاده از جعبه ابزار، یک کنترل Button به فرم اضافه کرده، خاصیت Name آن را برابر با btnShow و خاصیت Text آن را برابر با Show قرار دهید.

۴) سپس یک کنترل Label در فرم قرار دهید. این کنترل برای نمایش گزینه ای به کار می رود که کاربر از کادر پیغام انتخاب کرده است. خاصیت Name این گزینه را به lblResult و خاصیت Text آن را به Nothing Clicked تغییر دهید. سپس اندازه فرم را به گونه ای تنظیم کنید که فرم شما مشابه شکل ۷-۳ شود.



شکل ۷-۳

۵) بر روی دکمه ی Show دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس کد مشخص شده در زیر را در آن وارد کنید:

```
private void btnShow_Click(object sender, EventArgs e)
{
    if (  MessageBox.Show(
          "Your Internet Connection will be closed now!",
          "Dial-Up Networking Notification",
```

```
            MessageBoxButtons.OKCancel, MessageBoxIcon.None,
            MessageBoxDefaultButton.Button1)
            ==   DialogResult.OK)
    {
        lblResult.Text = "OK Clicked!";
        // Call some method here...
    }
    else
    {
        lblResult.Text = "Cancel Clicked!";
        // Call some method here...
    }
}
```

۶) برنامه را اجرا کرده و بر روی دکمه ی Show کلیک کنید. کادر پیغامی مشابه شکل ۷-۴ مشاهده خواهید کرد.



شکل ۷-۴

۷) بر روی دکمه ی OK و یا دکمه Cancel کلیک کنید. مشاهده می کنید که نتیجه انتخاب شما در لیبل نمایش داده می شود.

چگونه کار می کند؟

در این کد ابتدا با استفاده از متد Show کادر پیغامی را به کاربر نمایش دادیم. سپس در یک دستور if بررسی کرده ایم که کاربر چه گزینه ای را انتخاب کرده است:

```
if (  MessageBox.Show(
      "Your Internet Connection will be closed now!",
      "Dial-Up Networking Notification",
      MessageBoxButtons.OKCancel, null,
      MessageBoxDefaultButton.Button1)
      ==   DialogResult.OK)
```

توجه کنید در فراخوانی تابع Show مشخص کرده ایم که دکمه های OK و Cancel بر روی فرم قرار بگیرند و همچنین دکمه OK به عنوان دکمه پیش فرض در نظر گرفته شود.

همچنین مشاهده می کنید که در دستور if، مقدار برگشت داده شده توسط تابع Show را با مقدار DialogResult.OK بررسی می کنیم. در صورتی که این دو مقدار با یکدیگر مساوی نبودند می توان فهمید کاربر گزینه ی Cancel را انتخاب کرده است. همچنین می توانستید در این مقایسه از گزینه ی DialogResult.Cancel نیز استفاده کنید.

استفاده از این روش مقایسه هنگامی که دو دکمه در کادر وجود دارد روش مناسبی است. اما برای شرایطی که سه دکمه در کادر قرار دارند می توان از روشهای کوتاه تری استفاده کرد. در امتحان کنید بعد، این مورد را مشاهده خواهیم کرد.

امتحان کنید: استفاده از سه دکمه فرمان در کادر پیغام

۱) اگر همچنان برنامه در حال اجرا است آن را متوقف کرده و به قسمت طراحی فرم مربوط به Form1 بروید.

۲) دکمه ی دیگری به فرم اضافه کرده، خاصیت Name آن را برابر با btn3Buttons و خاصیت Text آن را برابر با 3 Buttons قرار دهید. سپس بر روی آن دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. کد زیر را در این متد وارد کنید:

```
private void btn3Buttons_Click(object sender, EventArgs e)
{
    // Declare a variable
    DialogResult intResult;

    // Get the results of the button clicked
    intResult = MessageBox.Show(
          "The A Drive is not ready.\n\nPlease insert a"  +
          " diskette into drive.", "Device Not Ready",
          MessageBoxButtons.AbortRetryIgnore,
          MessageBoxIcon.Error,
          MessageBoxDefaultButton.Button2);

    // Process the results of the button clicked
    switch (intResult)
    {
        case DialogResult.Abort:
            // Do abort processing here...
            lblResult.Text = "Abort clicked!";
            break;
        case DialogResult.Retry:
            // Do retry processing here...
            lblResult.Text = "Retry clicked!";
            break;
        case DialogResult.Ignore:
            // Do ignore processing here...
            lblResult.Text = "Ignore clicked!";
            break;
    }
```

```
}
```

۳) برنامه را اجرا کرده و بر روی دکمه ی Buttons 3 کلیک کنید. به این ترتیب کادر پیغامی با سه دکمه مشابه شکل ۷-۵ را مشاهده خواهید کرد. توجه کنید که در این کادر پیغام، دکمه ی دوم به صورت پیش فرض انتخاب شده است.



شکل ۷-۵

چگونه کار می کند؟

در فصل پنجم که در مورد شمارنده ها صحبت می کردیم، مشاهده کردید که شمارنده ها در حقیقت نوعی عدد صحیح هستند که به هر یک از آنها عددی نسبت داده می شود. نوع DialogResult که به وسیله ی متد Show برگشته می شود نیز نوعی عدد صحیح است. در این مثال ابتدا متغیری از نوع شمارنده DialogResult تعریف کرده و مقدار برگشتی از متد Show را (مقداری که به وسیله ی کاربر انتخاب شده است) در آن قرار دهیم.

```
// Declare a variable
DialogResult intResult;

// Get the results of the button clicked
intResult = MessageBox.Show(
      "The A Drive is not ready.\n\nPlease insert a"  +
      " diskette into drive.", "Device Not Ready",
      MessageBoxButtons.AbortRetryIgnore,
      MessageBoxIcon.Error,
      MessageBoxDefaultButton.Button2);
```

همانطور که در فصلهای قبل گفتم، برای استفاده از کاراکتر های کنترلی در یک رشته از علامت \ استفاده می کنند[1]. یکی از کاراکتر های کنترلی، کاراکتر n است که باعث می شود ادامه متن در یک خط جدید نمایش داده شود. در اینجا برای اینکه ادامه متنی که می خواهیم در کادر پیغام نمایش دهیم در دو خط پایین تر قرار بگیرد از کاراکتر کنترلی n دو بار استفاده کرده ایم.
در انتها نیز نتیجه برگشت داده شده توسط کادر پیغام را توسط یک دستور switch بررسی می کنیم:

```
// Process the results of the button clicked
```

[1] رجوع کنید به فصل چهارم بخش حلقه های foreach

```
switch (intResult)
{
    case DialogResult.Abort:
        // Do abort processing here...
        lblResult.Text = "Abort clicked!";
        break;
    case DialogResult.Retry:
        // Do retry processing here...
        lblResult.Text = "Retry clicked!";
        break;
    case DialogResult.Ignore:
        // Do ignore processing here...
        lblResult.Text = "Ignore clicked!";
        break;
}
```

نکته: همواره دقت کنید که از کادر پیغام بیش از اندازه استفاده نکنید و سعی کنید برای استفاده از آن دلیل مناسبی داشته باشید، زیرا استفاده بیش از اندازه از آن باعث ناراحتی کاربر می شود. در مواقعی از کادر پیغام استفاده کنید که بخواهید کاربر را از رخ دادن خطایی در برنامه آگاه کنید و یا به کاربر در مورد یک مسئله مهم که ممکن است باعث ایجاد خطا و یا از دست رفتن اطلاعات شود هشدار دهید. یک مثال برای کار هنگامی است که کاربر بدون ذخیره تغیرات سعی در خارج شدن از برنامه را داشته باشد. در این مواقع، باید به کاربر اطلاع دهید که اگر ادامه دهد ممکن است تمام تغییراتی که در برنامه ایجاد کرده است از بین برود.

کنترل OpenFileDialog:

تقریباً در نوشتن بیشتر برنامه های ویندوزی شرایطی وجود دارد که بخواهید داده هایی را در فایل نوشته و یا از آن بخوانید، پس به کنترلی نیاز دارید تا به وسیله ی آن بتوانید فایلی را باز کنید و یا داده هایی را در یک فایل ذخیره کنید. در چارچوب .NET دو کنترل برای این موارد در نظر گرفته شده است: OpenFileDialog و SaveFileDialog. در این بخش به بررسی کنترل OpenFileDialog می پردازیم و در بخش بعد نیز کنترل SaveFileDialog را بررسی خواهیم کرد.

هنگامی که با برنامه های ویندوزی مانند Word و یا Paint کار می کنید، معمولا برای باز کردن یک فایل و یا ذخیره آن و یا ... با محیطی یکسان روبرو می شوید. این نوع کادرها، به صورت یک مجموعه ی استاندارد در ویندوز وجود دارد و برنامه نویسان می توانند از آنها در برنامه های خود استفاده کنند.

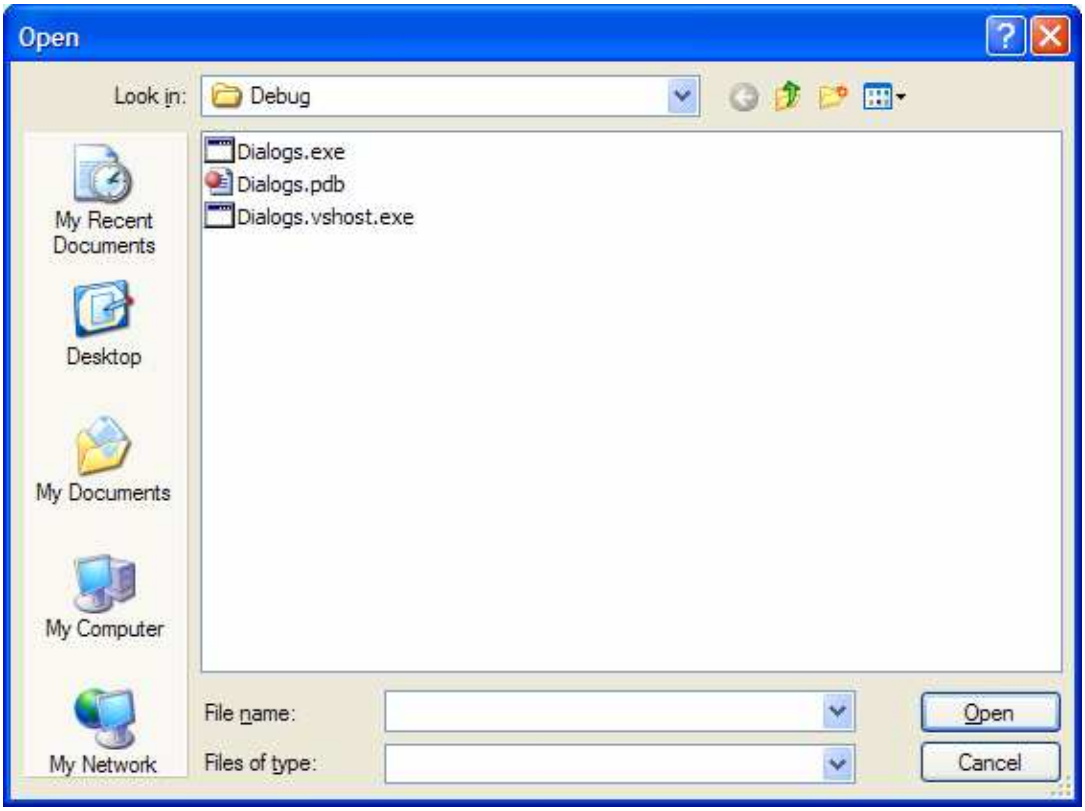
در .NET برای دسترسی به پنجره Open از این مجموعه باید از کلاس OpenFileDialog استفاده کرد. برای استفاده از این کلاس در .NET مانند هر کلاس دیگری باید یک متغیر از آن ایجاد و سپس خاصیتهای آن را به وسیله کد تنظیم کرد، و یا می توان با استفاده از جعبه ابزار هنگام طراحی فرم این کنترل را در برنامه قرار داده و از آن استفاده کرد. در هر دو حالت شیئ ایجاد شده دارای متدها، رویدادها و خاصیتهای یکسان خواهد بود.

برای دسترسی به این کنترل در جعبه ابزار، باید به بخش Dialogs آن مراجعه کنید و با استفاده از ماوس این کنترل را بر روی فرم قرار دهید. بعد از آن تنها کاری که باید انجام دهید، این است که خاصیتهای مورد نظرتان به وسیله پنجره Properties تنظیم کرده و سپس متد مربوط به نمایش آن را فراخوانی کنید.

برای استفاده از کنترل `OpenFileDialog` به صورت کلاس، ابتدا باید شیئ از نوع این کلاس ایجاد کنید. سپس در مواقعی که به این کنترل نیاز داشتید، به این شیئ مقدار بدهید و از آن استفاده کنید. پس از پایان استفاده نیز می توانید آن را نابود کنید تا منابع اشغال شده به وسیله آن آزاد شوند.

در این فصل با `OpenFileDialog` به صورت یک کنترل برخورد می کنیم. اما هنگامی که کاملاً مفهوم آن را درک کردید و توانستید به راحتی در برنامه از آن استفاده کنید، می توانید از این کنترل به صورت یک کلاس نیز استفاده کنید. مفهوم کلاسها و نحوه استفاده از آنها در فصل نهم به طور مفصل شرح داده شده اند.

برای نمایش پنجره ی `Open` کافی است متد `ShowDialog` آن را فراخوانی کنید، به این ترتیب پنجره ای مشابه شکل ۷-۶ نمایش داده خواهد شد.



شکل ۷-۶

خاصیتهای کنترل OpenFileDialog:

اگرچه کادر محاوره ای نمایش داده شده در شکل ۷-۶ یک صفحه `Open` استاندارد در ویندوز است و معمولاً هنگامی که این کادر را در برنامه ای مشاهده می کنید فقط فایلهای خاصی در آن نمایش داده شده است (برای مثال این کادر در برنامه ی `Notepad` فقط فایلهای متنی را نمایش می دهد)، اما در این کادر هیچ محدودیتی در نوع فایلهای قابل نمایش دیده نمی شود. در این پنجره تمام فایلهای موجود در فولدر جاری را مشاهده می کنید و نمی توانید مشخص کنید که چه نوع فایلهایی را نمایش دهد. برای فیلتر کردن نوع فایلهای نمایش داده شده در این کادر باید از خاصیتهای این کنترل استفاده کرده و آنها را به نحوی تنظیم کنید که کادر، فایلهای مورد نظر شما را نمایش دهد.

البته این یکی از مواردی است که با استفاده از خاصیتهای این کنترل قابل تنظیم است. در جدول زیر لیستی از خاصیتهای پر کـاربرد این کنترل را بررسی می کنیم:

خاصیت	شرح
AddExtension	این خاصیت مشخص می کند که اگر کاربر پسوندی را بـرای فایـل مـشخص نکـرد، برنامه به طور اتوماتیک پسوند را به فایل اضافه کند یا نه؟ این مورد بیشتر در پنجـره SaveFileDialog که در بخش بعد توضیح داده خواهد شد استفاده مـی شود.
CheckFileExist	مشخص می کند اگر کاربر آدرس فایلی را وارد کرد که وجـود نداشـت، برنامـه پیغـام خطایی را نمایش بدهد یا نه؟
CheckPathExist	مشخص می کند اگر کاربر آدرس مسیری را وارد کرد که وجود نداشت، برنامـه پیغـام خطایی را نمایش بدهد یا نه؟
DefaultExt	پسوند پیش فرض را برای فایل انتخاب شده مشخص می کند.
DereferenceLinks	با شورت کات ها استفاده می شود و مشخص می کند که اگر کاربر یک شورت کـات را انتخاب کرد، مسیر فایل اصلی برگشت داده شود (True) و یا مسیر خـود فایـل شورت کات به برنامه برگردد(False).
FileName	مشخص کننده نام فایلی است که در این پنجره انتخاب شده است.
FileNames	مشخص کننده نام فایلهایی است که در این پنجره انتخاب شده است. این خاصیت از نوع فقط–خواندنی است.
Filter	این خاصیت حاوی رشته ای است که برای فیلتر کردن فایلهایی کـه بایـد در پنجـره Open نمایش داده شوند به کار می رود. به وسیله این رشـته مـی توانیـد، چنـدین گروه فیلتر را برای این پنجره مشخص کنید تا در جعبه ترکیبی موجود در این پنجـره نمایش داده شوند و کاربر بتواند یکی از آنها را انتخاب کند.
FilterIndex	مشخص کننده شماره فیلتری است که کاربر هم اکنون در این صفحه انتخـاب شـده است.
InitialDirectory	مشخص کننده آدرس مسیری است که باید در ابتدا، در پنجره Open نمـایش داده شود.
Multiselect	مشخص می کند آیا کاربر می تواند چندین فایل را در این پنجره انتخاب کند و یا نه؟
ReadOnlyChecked	مشخص می کند آیا قسمت ReadOnly در پنجره Open انتخاب شده اسـت و یا نه؟
RestoreDirectory	تعیین می کند آیا کادر Open باید آدرس مسیری که قبل از بسته شدن در آن قرار داشت را برگرداند یا نه؟
ShowHelp	مشخص می کند آیا دکمه Help نیز در پنجره Open نمایش داده شود یا نه؟

ShowReadOnly	مشخص می کند آیا امکان تعیین این که فایل به صورت فقط–خواندنی باز شود برای کاربر وجود داشته باشد یا نه؟
Title	مشخص کننده متنی است که در نوار عنوان پنجره Open نمایش داده می شود.
ValidateNames	مشخص می کند که آیا پنجره فقط باید نام فایلهای معتبر ویندوزی را قبول کند و یا هر نامی را می تواند دریافت کند؟

متدهای OpenFileDialog:

اگرچه متدهای زیادی در کنترل OpenFileDialog وجود دارند، اما در مثال های این بخش بیشتر بر روی متد ShowDialog تمرکز خواهیم کرد. در لیست زیر، نام و شرح استفاده بعضی از توابع پر کاربرد این کنترل آمده است:

- **Dispose**: حافظه اشغال شده توسط این کنترل را آزاد می کند.
- **OpenFile**: فایلی را که به وسیله کاربر در پنجره Open انتخاب شده است به صورت فقط–خواندنی باز می کند. نام فایل به وسیله خاصیت FileName مشخص می شود.
- **Reset**: مقدار تمام خاصیتهای کنترل OpenFileDialog را به حالت اولیه برمی گرداند.
- **ShowDialog**: کادر محاوره ای پنجره Open را نمایش می دهد.

استفاده از تابع ShowDialog بسیار واضح است، زیرا این متد هیچ پارامتری را به عنوان ورودی دریافت نمی کند. بنابراین قبل از اینکه این تابع را فراخوانی کنید، باید تمام خاصیتهای موردنظر را در کنترل تنظیم کنید. بعد از اینکه پنجره Open نمایش داده شد می توانید با بررسی مقدار خاصیتهای کنترل مشخص کنید که چه فایل و یا چه فایلهایی، در چه مسیرهایی انتخاب شده اند. یک نمونه از فراخوانی این متد در قطعه کد زیر نمایش داده شده است:

```
openFileDialog1.ShowDialog();
```

این متد مقداری از نوع شمارنده ی DialogResult به صورت OK و یا Cancel برمی گرداند. مقدار OK به معنی کلیک کردن کاربر بر روی دکمه ی Open و مقدار Cancel برابر با کلیک کردن روی Cancel است. توجه داشته باشید که این کنترل هیچ فایلی را برای برنامه باز نمی کند و یا محتویات آن را نمی خواند. این کنترل فقط رابطی است که به کاربر اجازه می دهد یک یا چند فایل را برای باز شدن به وسیله ی برنامه مشخص کند. بعد از این مرحله، شما باید در برنامه نام و آدرس فایلهای مشخص شده توسط کاربر را به وسیله خاصیتهای کنترل OpenFileDialog بدست آورده و سپس آنها را باز کنید.

استفاده از کنترل OpenFileDialog:

حال که خاصیت ها و متدهای مهم کنترل OpenFileDialog را بررسی کردیم، بهتر است از این کنترل در یک برنامه استفاده کنیم تا با نحوه استفاده از آن در برنامه آشنا شویم.

در امتحان کنید بعد، با استفاده از کنترل OpenFileDialog برنامه ای خواهید نوشت که کاربر بتواند در آن یک فایل متنی را مشخص کند و برنامه محتویات آن فایل را در یک TextBox نمایش دهد.

امتحان کنید: کار با کنترل OpenFileDialog

۱) در محیط ویژوال استودیو یک پروژه ویندوزی جدید به نام Dialogs ایجاد کنید.

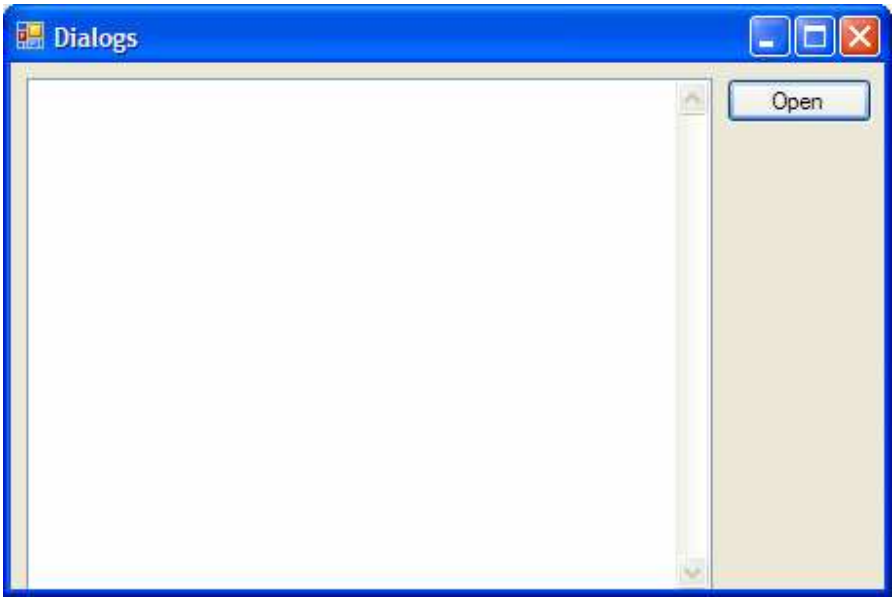
۲) برای تغییر نام فرم خود، در پنجره Solution Explorer بر روی نام فرم کلیک راست کرده و از منوی باز شده گزینه Rename را انتخاب کنید. سپس نام فرم را به Dialogs.cs تغییر دهید. با استفاده از پنجره Properties خاصیتهای فرم را به صورت زیر تغییر دهید:

- خاصیت Size آن را برابر با 456;304 قرار دهید.
- خاصیت StartPosition را برابر با CenterScreen قرار دهید.
- خاصیت Text را برابر با Dialogs قرار دهید.

۳) برای این که فایل مشخص شده توسط کاربر را در برنامه نمایش دهید، به یک کنترل TextBox نیاز دارید. همچنین باید یک کنترل Button نیز بر روی فرم قرار دهید تا کاربر به وسیله آن بتواند پنجره Open را نمایش دهد. بنابراین یک کنترل TextBox و یک کنترل Button به فرم خود اضافه کرده و خاصیتهای آن را بر اساس لیست زیر تنظیم کنید:

- خاصیت Name کنترل TextBox را برابر با txtFile، خاصیت Anchor را برابر با Top، Bottom،Left،Right خاصیت Location را برابر با 8,8 خاصیت MultiLine را برابر با True خاصیت ScrollBars را برابر با Vertical و خاصیت Size را برابر با 352;264 قرار دهید.
- خاصیت Name دکمه فرمان را برابر با btnOpen خاصیت Text آن را برابر با Open خاصیت Anchor را برابر با Top,Right و خاصیت Location را برابر با 367,8 قرار دهید.

۴) بعد از اینکه کنترل ها را در فرم قرار دادید و خاصیت آنها را طبق لیست قبلی تنظیم کردید، فرم برنامه شما باید مشابه شکل ۷-۷ شده باشد.



شکل ۷-۷

نکته: هنگامی که خاصیت Anchor را برای کنترلهای این فرم تنظیم کنیم، با تغییر اندازه فرم به وسیله کاربر اندازه کنترل ها نیز به صورت متناسب تغییر خواهد کرد.

۵) در نوار ابزار به قسمت Dialogs بروید، کنترل FileOpenDialog را انتخاب کرده و بر روی آن دو بار کلیک کنید. مشاهده می کنید که این کنترل به قسمت پایین محیط طراحی ویژوال استودیو اضافه خواهد شد. حال می توانید این کنترل را در این قسمت انتخاب کرده و خاصیتهای مختلف آن را به وسیله پنجره Properties تنظیم کنید. با وجود این فعلاً نام و خاصیتهای پیش فرض این کنترل را قبول کنید. در قسمتهای بعدی با استفاده از کد خاصیتهای مورد نظر را در این کنترل تغییر خواهیم داد.

۶) به قسمت ویرایشگر کد بروید و در ابتدای کلاس مربوط به فرم، یک متغیر رشته ای تعریف کنید تا نام فایل در آن ذخیره شود. در قسمتهای بعدی برنامه، نام فایلی که به وسیله کادر Open برگشته می شود را در این متغییر ذخیره خواهیم کرد:

```
public partial class Dialogs : Form
{
    // Declare variables
    private string strFileName;
```

۷) حال باید کد مربوط به باز کردن کادر Open را در رویداد کلیک کنترل btnOpen قرار دهیم. برای این کار به قسمت طراحی فرم بروید و بر روی این کنترل دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس کدهای مشخص شده در زیر را به آن اضافه کنید:

```
private void btnOpen_Click(object sender, EventArgs e)
{
    // Set the OpenFileDialog properties
```

```
        openFileDialog1.Filter = "Text files (*.txt) |*.txt|"
                                 + " All files (*.*) |*.*";
        openFileDialog1.FilterIndex = 1;
        openFileDialog1.Title = "Demo Open File Dialog";

        // Show the OpenFileDialog and if the user clicks the
        // Open button, load the file
        if (openFileDialog1.ShowDialog() == DialogResult.OK)
        {
            // Save the file name
            strFileName = openFileDialog1.FileName;
            // Read the contents of the file
            txtFile.Text =
                System.IO.File.ReadAllText(strFileName);
        }
}
```

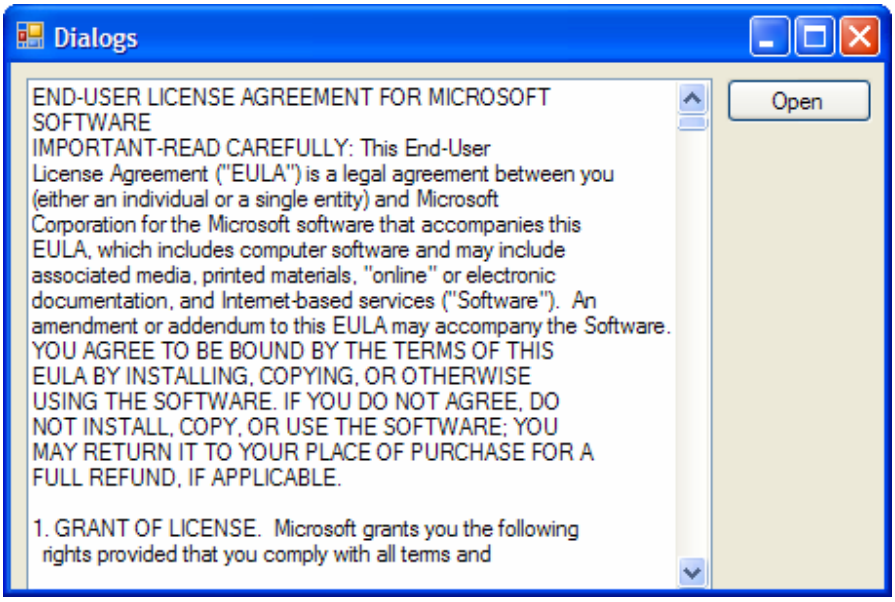
۸) حال برنامه را اجرا کرده و هنگامی که فرم اصلی برنامه نمایش داده شد، بر روی دکمه ی Open کلیک کنید تا کادر محاوره ای Open نمایش داده شود. توجه کنید که عنوان این کادر همانطور که در کد مشخص کرده بودید تغییر کرده است. در جعبه ترکیبی Files of type: در پایین کادر کلیک کنید. مشاهده می کنید که دو نوع فیلتر برای نمایش فایلها در نظر گرفته شده است.

۹) یکی از فیلترهای نمایش داده شده در این قسمت را انتخاب کنید . سپس یک فایل متنی را در دیسک مشخص کرده و بر روی دکمه Open کلیک کنید. مشاهده خواهید کرد که همانند شکل ۷-۸ محتویات فایل در فرم نمایش داده خواهد شد.

۱۰) برای امتحان، از برنامه خارج شوید و مجدداً آن را اجرا کنید. سپس بر روی دکمه ی Open کلیک کنید. مشاهده خواهید کرد که کادر Open دایرکتوری را نمایش می دهد که در اجرای قبلی برنامه فایل را از آن انتخاب کردید.

چگونه کار می کند؟

قبل از نمایش دادن کادر محاوره ای Open باید بعضی از خصوصیات آن را تنظیم کنید تا کادر به نحوی مناسب برای برنامه نمایش داده شود. اولین خاصیتی که باید تنظیم شود، خاصیت Filter است. این خاصیت به شما اجازه می دهد فیلترهایی که در جعبه ترکیبی Files of type: در کادر نمایش داده می شوند را مشخص کنید. هنگامی که بخواهید یک فیلتر برای پسوندی خاص ایجاد کنید، باید ابتدا توضیح آن فیلتر را وارد کرده، سپس یک خط عمودی (|) قرار دهید و در انتها نیز پسوند فایل را وارد کنید. اگر بخواهید چندین فیلتر در این کادر وجود داشته باشند، باید هر یک از آنها به وسیله یک خط عمودی از یکدیگر جدا کنید.



شکل ۷-۸

```
// Set the OpenFileDialog properties
openFileDialog1.Filter =
     "Text files (*.txt) |*.txt| All files (*.*) |*.*";
```

دومین خاصیتی که باید تنظیم شود، خاصیت `FilterIndex` است که مشخص می کند کدام فیلتر بایـد بـه صـورت پـیش فرض در فرم در نظر گرفته شود. مقدار ۱ برای این خاصیت مشخص می کند که فیلتر اول به عنوان فیلتر پیش فرض در نظر گرفتـه می شود.

```
openFileDialog1.FilterIndex = 1;
```

در انتها نیز با استفاده از خاصیت `Title` عنوان پنجره `Open` را تغییر می دهیم:

```
openFileDialog1.Title = "Demo Open File Dialog";
```

برای نمایش کادر `Open` باید از تابع `Show` استفاده کنیم. همانطور که گفتم این تابع مقداری از نـوع `DialogResult` را برمیگرداند که می تواند برابر با `DialogResult.OK` و یا `DialogResult.Cancel` باشد. اگر کاربر در پنجره `Open` بر روی دکمه `Open` کلیک کند مقدار `DialogResult.OK` توسط تابع برگردانده می شود. در صورتی که کاربر دکمه `Cancel` را انتخاب کند مقدار بازگشتی برابر با `DialogResult.Cancel` خواهد بود.

```
// Show the OpenFileDialog and if the user clicks the
// Open button, load the file
if (openFileDialog1.ShowDialog() == DialogResult.OK)
```

در چارچوب .NET، فضای نام System.IO دارای تمام توابع و کلاسهای مورد نیاز برای کنترل ورودی و خروجی می باشد. برای دریافت اطلاعات درون فایل متنی می توانیم از کلاس File در این فضای نام استفاده کنیم. اگر یک فضای نام با استفاده از دستور using در ابتدای کد مربوط به یک کلاس مشخص شود، در طول برنامه می توان بدون ذکر کردن فضای نام، از کلاسهای داخل آن استفاده کرد[1]. برای مثال کلاس MessageBox که برای نمایش کادر پیغام از آن استفاده می کنیم در فضای نام System.Windows.Forms قرار دارد. اما همانطور که در برنامه های قبلی نیز مشاهده کردید، بدون اینکه فضای نام این کلاس را مشخص کنیم از آن در برنامه استفاده کرده ایم. این مورد به این علت است که فضای نام System.Windows.Forms با استفاده از راهنمای using در ابتدای کد (قبل از تعریف کلاس) به برنامه اضافه شده است. بنابراین در طول برنامه برای استفاده از این کلاس نیازی به ذکر کردن فضای نام آن نداریم.
اما در مورد کلاس File اگر به ابتدای کد نگاه کنید مشاهده می کنید که فضای نام مربوط به این کلاس به برنامه اضافه نشده است، پس برای استفاده از آن باید نام کلاس را به همراه فضای نام آن یعنی به صورت System.IO.File به کار ببریم.

```
        // Read the contents of the file
        txtFile.Text =
System.IO.File.ReadAllText(strFileName);
```

با استفاده از تابع ReadAllText در کلاس System.IO.File می توانیم محتویات یک فایل متنی را از دیسک بخوانیم. دقت کنید که تابع ReadAllText به گونه ای نوشته شده است که برای استفاده از آن لازم نیست ابتدا یک شیئ از کلاس File ایجاد کنیم. این نوع توابع، توابع Static نامیده می شوند که در فصل دهم با مفهوم آنها آشنا خواهیم شد. این تابع پارامتری از نوع رشته که حاوی آدرس فایل مورد نظر است را دریافت کرده و متن داخل فایل را به صورت رشته برمی گرداند.
در این برنامه ابتدا آدرسی که به وسیله کنترل OpenFileDialog مشخص شده است را در متغییر strFileName قرار می دهیم. همانطور که در لیست خاصیتهای این کنترل مشاهده کردید، نام فایل انتخاب شده توسط کاربر در خاصیت FileName آن قرار می گیرد. بنابراین با استفاده از دستور زیر نام فایل را در متغییر strFileName قرار می دهیم.

```
// Save the file name
strFileName = openFileDialog1.FileName;
```

سپس با استفاده از تابع ReadAllText کلاس File و ارسال متغییر strFileName به عنوان پارامتر، محتویات فایل را خوانده و نتیجه را که به صورت متغیر رشته ای برگشت داده می شود در TextBox قرار می دهیم.

```
// Read the contents of the file
txtFile.Text = System.IO.File.ReadAllText(strFileName);
```

در این تمرین با قسمتی از خاصیت ها و متدهای کلاس OpenFileDialog آشنا شدیم. اما همچنان تعداد زیادی از آنها باقی مانده است که درباره آنها صحبتی نکرده ایم. می توانید با تمرین و استفاده از این کنترل در برنامه های گوناگون با توابع و قابلیتهای فراوانی که ارائه می دهد بیشتر آشنا شوید.

---

[1] راهنمای using در زبان C# با دستور using در C++ مقداری متفاوت است و بیشتر می توان آن را مشابه دستور پیش پردازنده include در C++ تلقی کرد. در مورد using در ادامه ی کتاب بیشتر صحبت خواهیم کرد.

کنترل SaveFileDialog:

حال که می توانید با استفاده از کنترل OpenFileDialog یک فایل را باز کرده و از اطلاعات آن در برنامه استفاده کنید، بهتر است به بررسی کنترل SaveFileDialog بپردازیم تا مشاهده کنید که چگونه می توان به وسیله آن اطلاعاتی را در دیسک ذخیره کرد. همانند کنترل OpenFileDialog، این کنترل نیز می تواند هم به صورت یک کنترل و هم به صورت یک کلاس مورد استفاده قرار گیرد. البته در این قسمت از SaveFileDialog به عنوان یک کنترل استفاده می کنیم، اما بعد از اینکه با این کنترل بیشتر آشنا شدید می توانید از آن به عنوان یک کلاس نیز استفاده کنید.
بعد از اینکه فایلی را در برنامه باز کردید، ممکن است بخواهید تغییراتی در آن ایجاد کرده و نتیجه را در دیسک ذخیره کنید. در این شرایط است که کنترل SaveFileDialog می تواند موثر واقع شود. کنترل SaveFileDialog نیز کارکردی مشابه کنترل OpenFileDialog دارد، البته در جهت عکس. این کنترل به شما اجازه می دهد یک نام و آدرس را برای ذخیره ی یک فایل در دیسک از کاربر دریافت کنید. مجدداً باید ذکر کنم که همانند کنترل OpenFileDialog این کنترل نیز فایلی را در دیسک ذخیره نمی کند، بلکه فقط یک رابط استاندارد را برای برنامه به وجود می آورد تا کاربر به وسیله آن بتواند محلی را برای ذخیره اطلاعات مشخص کند.

خاصیتهای کنترل SaveFileDialog:

در جدول زیر لیستی از خصوصیت های پر کاربرد کنترل SaveFileDialog به همراه کاربرد آنها آورده شده است. همانطور که مشاهده می کنید این کنترل (و یا این کلاس، بسته به نوعی که از آن استفاده می کنید)، خاصیتهای زیادی دارد که می توان به وسیله آنها، رفتار کنترل را در برنامه تغییر داد.

خاصیت	شرح
AddExtension	مشخص می کند اگر کاربر پسوند فایل را تعیین نکرد، برنامه به طور اتوماتیک پسوند را به فایل اضافه کند.
CheckFileExist	مشخص می کند اگر کاربر نام فایلی را مشخص کرد که در دیسک وجود داشت، پیغام هشداری نمایش داده شود یا نه؟ این مورد معمولا هنگامی که کاربر بخواهد فایل را بر روی یک فایل موجود بنویسد کاربرد دارد.
CheckPathExist	مشخص می کند اگر کاربر آدرس فایلی را مشخص کرد که در دیسک وجود نداشت، پیغام هشداری نمایش داده شود یا نه؟
CreatePrompt	مشخص می کند اگر کاربر فایلی را مشخص کرد که وجود نداشت، برای ایجاد آن فایل از کاربر سوال شود یا نه؟
DefaultExt	پسوند پیش فرض را در این کادر مشخص می کند.
DereferenceLinks	مشخص می کند اگر کاربر یک شورت کات را انتخاب کرد، آدرس فایل اصلی که شورت کات به آن اشاره می کند برگشته شود و یا آدرس خود فایل شورت کات به

برنامه برگردد.

FileName	نام فایلی که در کادر توسط کاربر مشخص شده است را برمی گرداند. این خاصیت به صورت فقط–خواندنی است.
FileNames	نام فایلهایی که در کادر توسط کاربر انتخاب شده است را برمی گرداند. این خاصیت که شامل یک آرایه رشته ای است نیز به صورت فقط–خواندنی است.
Filter	این خاصیت حاوی رشته ای است که برای فیلتر کردن فایلهایی که باید در پنجره Save نمایش داده شوند به کار می رود. به وسیله این رشته می توانید، چندین گروه فیلتر را برای این پنجره مشخص کنید تا در جعبه ترکیبی موجود در این پنجره نمایش داده شوند و کاربر بتواند یکی از آنها را انتخاب کند.
FilterIndex	مشخص کننده اندیس فیلتری است که هم اکنون در کادر محاوره ای انتخاب شده است.
InitialDirectory	مشخص کننده آدرس دایرکتوری است که باید در ابتدا، در پنجره Save نمایش داده شود.
OverwritePrompt	مشخص می کند اگر کاربر خواست فایل را بر روی فایل دیگری ذخیره کند، پیغام هشدار به کاربر نمایش داده شود یا نه؟
ResotreDirectory	تعیین می کند آیا کادر Save باید آدرس دایرکتوری را که قبل از بسته شدن در آن قرار داشت، برگرداند یا نه؟
ShowHelp	مشخص می کند که آیا دکمه Help نیز در پنجره Open نمایش داده شود یا نه؟
Title	مشخص کننده متنی است که در نوار عنوان پنجره Open نمایش داده می شود.
ValidateNames	مشخص می کند که آیا پنجره فقط باید نام فایلهای معتبر ویندوزی را قبول کند و یا هر نامی را بتواند دریافت کند؟

متدهای کنترل SaveFileDialog:

متدهای کنترل SaveFileDialog همانند متدهای OpenFileDialog هستند. برای مطالعه متدهای کنترل OpenFileDialog می توانید به بخش قبلی مراجعه کنید. در تمام مثال های بعدی نیز همانند کنترل OpenFileDialog از تابع ShowDialog برای نمایش کادر Save استفاده می کنیم.

استفاده از کنترل SaveFileDialog:

برای بررسی نحوه کارکرد کنترل SaveFileDialog، از پروژه Dialogs در قسمت قبلی استفاده می کنیم. در ایـن قسمت می خواهیم برنامه را به صورتی تغییر دهیم که متن داخل TextBox را در فایلی ذخیره کند.
در این قسمت، با استفاده از کنترل SaveFileDialog پنجره Save File را به کاربر نمایش داده و بـه او اجـازه می دهیم تا مکانی را برای ذخیره محتویات TextBox مشخص کند. سپس محتویات داخل آن را در فایلی در مـسیر مـشخص شده توسط کاربر ذخیره می کنیم.

امتحان کنید: کار با کنترل SaveFileDialog

۱) برنامه Dialogs را که در قسمت قبل ایجاد کرده بودیم، مجددا باز کنید.
۲) در فرم اصلی برنامه کنترل Button دیگری اضافه کرده و خاصیتهای آن را برابر با لیست زیر قرار دهید.

- خاصیت Name را برابر با btnSave قرار دهید.
- خاصیت Text را برابر با Save قرار دهید.
- خاصیت Anchor را برابر با Top,Right قرار دهید.
- خاصیت Location را برابر با 367;38 قرار دهید.

۳) در جعبه ابزار به قسمت Dialogs بروید و بر روی کنترل SaveFileDialog دو بار کلیک کنید. به ایـن ترتیب یک کنترل SaveFileDialog در قسمت پایین طراحی فرم قرار می گیرد.
۴) بر روی دکمه ی btnSave دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس کـد زیـر را در آن متد وارد کنید:

```
private void btnSave_Click(object sender, EventArgs e)
{
     // Set the save dialog properties
     saveFileDialog1.DefaultExt = "txt";
     saveFileDialog1.FileName = strFileName;
     saveFileDialog1.Filter =
     "Text files (*.txt)|*.txt|All files (*.*)|*.*";
     saveFileDialog1.FilterIndex = 1;
     saveFileDialog1.OverwritePrompt = true;
     saveFileDialog1.Title = "Demo Save File Dialog";

     // Show the Save file dialog and if the user clicks
the  // Save button, save the file
     if (saveFileDialog1.ShowDialog() == DialogResult.OK)
     {
          // Save the file name
          strFileName = saveFileDialog1.FileName;

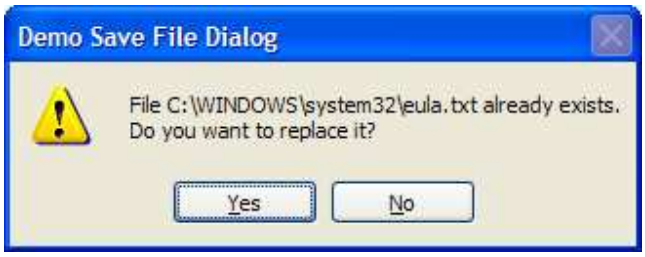
          // Write the contents of the text box in file
          System.IO.File.WriteAllText(
```

```
                    strFileName, txtFile.Text);
        }
}
```

۵) در این مرحله می توانید برنامه خود را تست کنید، بنابراین پروژه را اجرا کرده و متن ساده ای را داخل آن وارد کنید. سپس بر روی دکمه ی Save کلیک کنید. مشاهده خواهید کرد که کادر محاوره ای Save نمایش داده خواهد شد.

۶) نامی را برای فایل انتخاب کرده و بر روی دکمه ی OK کلیک کنید. به این ترتیب متن داخل TextBox در فایلی با نام و مسیری که مشخص کرده بودید ذخیره می شود. برای امتحان این مورد می توانید با کلیک کردن بر روی دکمه ی Open فایل ایجاد شده را مجدداً در برنامه باز کرده و مشاهده کنید.

۷) برای تست عملکرد خاصیت OverwritePrompt در کنترل SaveFileDialog متن دیگری را در TextBox وارد کرده و بر روی دکمه ی Save کلیک کنید. مجدداً مسیر و نام فایل قبلی را برای ذخیره فایل جدید وارد کنید. مشاهده خواهید کرد که پیغامی همانند شکل ۷-۹ نمایش داده می شود و می گوید که فایلی با این نام موجود است. آیا می خواهید آن را با این فایل تعویض کند؟ در صورتی که بر روی گزینه Yes کلیک کنید، فایل قبلی پاک می شود و فایل جدید به جای آن قرار می گیرد. اگر بر روی گزینه No کلیک کنید، به کادر Save برمی گردید تا نام دیگری را برای فایل انتخاب کنید.



شکل ۷-۹

نکته: هنگامی که صفحه Save و یا Open نمایش داده می شود، منویی که به وسیله کلیک راست نمایش داده اجازه می دهد کارهایی را از قبیل انتقال فایل به محلی دیگر، حذف فایل و یا تغییر نام آن را انجام دهید. همچنین بر حسب اینکه چه نرم افزارهایی بر روی سیستم شما نصب شده باشند گزینه های دیگری نیز در این منو نمایش داده می شوند. برای مثال اگر WinZip یا WinRar بر روی سیستم شما نصب شده باشد، در این پنجره می توانید فایلها را فشرده کنید.

چگونه کار می کند؟

قبل از نمایش کنترل SaveFileDialog باید بعضی از خاصیتهای آن را تنظیم کنید تا بتوانید از آن به صورت مناسب در برنامه ی خود استفاده کنید. در این برنامه اول خاصیت DefaultExt را تنظیم کرده ایم. اگر کاربر هنگام مشخص کردن نام فایل پسوندی برای آن مشخص نکرده باشد، پسوندی که در این خاصیت مشخص شده است به طور اتوماتیک به انتهای فایل اضافه می شود. برای مثال فرض کنید کاربر هنگام ذخیره فایل نام test را بدون هیچ پسوندی وارد کرده و بر روی دکمه Save کلیک می کند. در این حالت پسوندی که در این خاصیت مشخص شده است به انتهای فایل اضافه شده و سپس فایل با نام test.txt در دیسک ذخیره می شود.

```
saveFileDialog1.DefaultExt = "txt";
```

سپس خاصیت FileName از این کنترل را برابر با مقدار متغیر strFileName قرار می دهیم. هنگامی که یک فایل را با استفاده از دکمه ی Open باز کنید، نام آن در این متغیر ذخیره می شود. با قرار دادن این متغیر در خاصیت FileName کنترل SaveFileDialog باعث می شویم که کاربر بدون وارد کردن نام فایل بتواند یک فایل را باز کرده، آن را ویرایش کند و سپس آن را ذخیره کند. البته کاربر می تواند با تغییر این نام در کادر Save نام جدیدی برای فایل انتخاب کند و یا فایل را در مسیر دیگری ذخیره کند.

```
saveFileDialog1.FileName = strFileName;
```

در دو خط بعدی خاصیت Filter و FilterIndex کنترل را تنظیم می کنیم. به این ترتیب فایلهای خاصی در کادر نمایش داده خواهند شد:

```
saveFileDialog1.Filter =
"Text files (*.txt)|*.txt|All files (*.*)|*.*";
saveFileDialog1.FilterIndex = 1;
```

خاصیت OverwritePrompt مقداری را از نوع Boolean قبول می کند. اگر مقدار این خاصیت را برابر با True قرار دهید، در صورتی که کاربر بخواهد فایلی را بر روی فایل دیگری ذخیره کند به او هشدار داده می شود. اگر مقدار این خاصیت برابر با False باشد، در صورت رخ دادن چنین مشکلی، بدون اینکه موردی به کاربر اطلاع داده شود فایل قبلی پاک می شود و فایل جدید بر روی آن ذخیره می شود.

```
saveFileDialog1.OverwritePrompt = true;
```

در انتها نیز عنوان پنجره Save را تعیین می کنم تا با نام برنامه هماهنگ شود:

```
saveFileDialog1.Title = "Demo Save File Dialog";
```

بعد از اینکه خاصیتهای مختلف کادر Save را تنظیم کردیم، می توانیم کادر را نمایش دهیم. سپس با استفاده از دستور if مشخص می کنیم که کاربر بر روی دکمه Save کلیک کرده است و یا بر روی دکمه Cancel. در این کنترل هم همانند کنترل OpenFileDialog اگر کاربر بر روی دکمه Save کلیک کند مقدار DialogResult.OK و اگر کاربر بر روی دکمه فرمان Cancel کلیک کند مقدار DialogResult.Cancel برگشت داده می شود.

```
if (saveFileDialog1.ShowDialog() == DialogResult.OK)
```

اگر کاربر گزینه Save را انتخاب کرده بود، ابتدا باید نام فایل را در متغیر strFileName ذخیره کنیم. سپس مجدداً از کلاس File برای ذخیره محتویات فرم، درون فایل استفاده می کنیم. اما در این قسمت باید از متد WriteAllText در این کلاس استفاده کنیم. استفاده از این متد نیز به نمونه سازی از این کلاس نیازی ندارد. این متد آدرس یک فایل و یک متغییر

رشته ای که حاوی محتویات فایل است را به عنوان ورودی دریافت کرده و در آدرس تعیین شده، فایلی را با محتویاتی که به آن فرستاده شده است ایجاد می کند:

```
// Save the file name
strFileName = saveFileDialog1.FileName;

// Write the contents of the text box in file
System.IO.File.WriteAllText(
          strFileName, txtFile.Text);
```

کنترل FontDialog:

بعضی مواقع در یک برنامه ممکن است بخواهید به کاربر اجازه دهید که فونت خاصی را انتخاب کند تا اطلاعات او با آن فونت نمایش داده شوند، و یا ممکن است بخواهید لیستی از تمام فونت هایی که در سیستم کاربر نصب شده است را در برنامه استفاده کنید. در این مواقع می توانید از کنترل `FontDialog` استفاده کنید. این کنترل تمام فونتهای نصب شده در سیستم کاربر را در یک کادر استاندارد نمایش می دهد و به کاربر اجازه می دهد تا فونت خاصی را بین آنها انتخاب کند.
همانند کادرهای `OpenFileDialog` و `SaveFileDialog`، کادر محاوره ای `FontDialog` هم می تواند به صورت یک کنترل و هم می تواند به صورت یک کلاس مورد استفاده قرار بگیرد. استفاده از این کادر بسیار راحت است. کافی است که تعدادی از خاصیتهای آن را تنظیم کنید، کادر را نمایش دهید و سپس با استفاده از خاصیتهای کادر مشخص کنید که کدام فونت توسط کاربر انتخاب شده است.

خاصیتهای کنترل FontDialog:

جدول زیر لیستی از خاصیتهای پر کاربرد `FontDialog` را نمایش می دهد.

خاصیت	شرح
`AllowScriptChange`	مشخص می کند آیا کاربر می تواند با استفاده از قسمت `Script` کادر، مجموعه کاراکترهایی جدای از آنچه در قسمت `Script` مشخص شده است را انتخاب کند یا نه؟ در صورت اینکه مقدار این خاصیت برابر با `True` باشد، تمام مجموعه کاراکتر های موجود در قسمت `Script` نمایش داده می شود.
`Color`	رنگ فونت انتخاب شده را مشخص می کند.
`Font`	نام فونت انتخاب شده را مشخص می کند.
`FontMustExist`	مشخص می کند اگر کاربر نام فونتی را انتخاب کرد که وجود نداشت، کادر پیغامی برای خطا نمایش داده شود یا نه؟
`MaxSize`	حداکثر اندازه ای که کاربر می تواند برای فونت انتخاب کند را مشخص می کند.

MinSize	حداقل اندازه ای که کاربر می تواند برای فونت انتخاب کند را مشخص می کند.
ShowApply	مشخص می کند کادر محاوره ای که نمایش داده مـی شـود بایـد دارای دکمـه ی Apply نیز باشد یا نه؟
ShowColor	مشخص می کند در کادر فونت، امکان انتخاب رنگ نیز وجود داشته باشد یا نه؟
ShowEffects	مشخص می کند آیا کادر فونت باید دارای قسمتی برای تعیین خط دار بـودن، زیـر خط دار بودن و یا انتخاب رنگ متن توسط کاربر باشد یا نه؟
ShowHelp	مشخص می کند آیا کادر فونت دارای دکمه فرمان Help باشد یا نه؟

متدهای کنترل FontDialog:

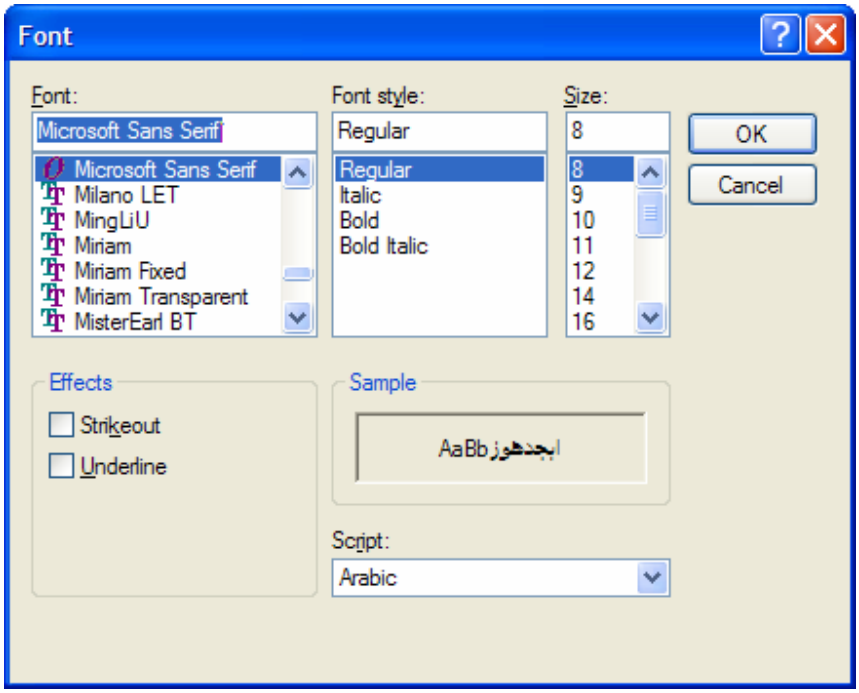
در مثال های بعدی فقط از یکی از متدهای کنترل FontDialog استفاده خواهیم کرد که آن نیز متـد ShowDialog برای نمایش کادر محاوره ای خواهد بود. علاوه بر این متد متدهای زیادی برای این کنترل وجود دارنـد، ماننـد متـد Reset کـه باعث می شود مقدار تمام خاصیتهای کنترل به حالت اول برگردد.

استفاده از کنترل FontDialog:

بـرای نمـایش کنتـرل FontDialog نیـاز بـه تنظـیم هـیچ مقـداری نیـست، بلکـه مـی تـوان بـه طـور مـستقیم متـد ShowDialog این کنترل را فراخوانی کرد تا کادر محاوره ای نمایش داده شود.

```
fontDialog1.ShowDialog();
```

در این صورت کادر محاوره ای همانند شکل ۷-۱۰ نمایش داده می شود.



شکل ۷-۱۰

مشاهده می کنید که کادر فونت دارای یک بخش `Effects` است که به کاربر اجازه می دهد تعیین کند یک فونت دارای خط و یا زیر خط نیز باشد. نمایش این بخش به این علت است که خاصیت `ShowEffects` به طور پیش فرض دارای مقدار `true` است. در این کادر قسمت `Color` برای انتخاب رنگ نمایش داده نشده است، زیرا مقدار پیش فرض `ShowColor` برابر با `False` است. برای نمایش قسمت رنگ، بایستی قبل از فراخوانی تابع `ShowDialog` مقدار این خاصیت را برابر با `True` قرار داد.

```
fontDialog1.ShowColor = true;
fontDialog1.ShowDialog();
```

متد `ShowDialog` از این کادر محاوره ای نیز، همانند تمام متدهای `ShowDialog` مقداری را از نوع `DialogResult` برمی گرداند. این مقدار می تواند برابر با `DialogResult.OK` و یا `DialogResult.Cancel` باشد.

هنگامی که کادر فونت نمایش داده شد و کاربر بر روی گزینه `OK` کلیک کرد، می توانید با استفاده از خاصیتهای `Color` و `Font` کنترل `FontDialog` بررسی کنید که کاربر چه نوع فونت و چه رنگی را انتخاب کرده است و سپس آن را در برنامه استفاده کنید و یا در متغیری قرار داده و در بخشهای بعدی استفاده کنید.

حال که با این کادر و نحوه کارکرد آن آشنا شدید، در امتحان کنید بعد از آن استفاده خواهیم کرد. در بخش بعد، از برنامه ای که در دو مثال قبلی ایجاد کرده بودیم استفاده می کنیم و آن را مقداری گسترش می دهیم. در قسمتهای قبلی کاربر می توانست در برنامه فایلی را باز کرده، تغییراتی را در آن انجام دهد و سپس فایل را ذخیره کند. در این قسمت بخشی را به برنامه اضافه می کنیم که کاربر به وسیله آن بتواند فونت متن درون `TextBox` را تغییر دهد.

امتحان کنید: کار با کنترل FontDialog

۱) مجددا پروژه Dialogs را باز کنید.
۲) در قسمت طراحی فرم، کنترل Button دیگری به فرم اضافه کرده و خاصیتهای آن را مطابق لیست زیر تعیین کنید:

- خاصیت Name را برابر با btnFont قرار دهید.
- خاصیت Anchor را برابر با Top, Right قرار دهید.
- خاصیت Anchor را برابر با 367;68 قرار دهید.
- خاصیت Text را برابر با Font قرار دهید.

۳) برای نمایش کادر فونت باید یک کنترل FontDialog بر روی فرم قرار دهید. برای این کار در جعبه ابزار به قسمت Dialogs بروید و در آنجا بر روی کنترل FontDialog دو بار کلیک کنید. به این صورت یک کنترل FontDialog در قسمت پاین محیط طراحی در ویژوال استودیو اضافه خواهد شد. تمام تنظیمات پیش فرض این کنترل را قبول کنید و خاصیتهای آن را تغییر ندهید.
۴) بر روی دکمه ی btnFont دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس کد زیر را به آن متد اضافه کنید:

```
private void btnFont_Click(object sender, EventArgs e)
{
     // Set the FontDialog control properties
     fontDialog1.ShowColor = true;

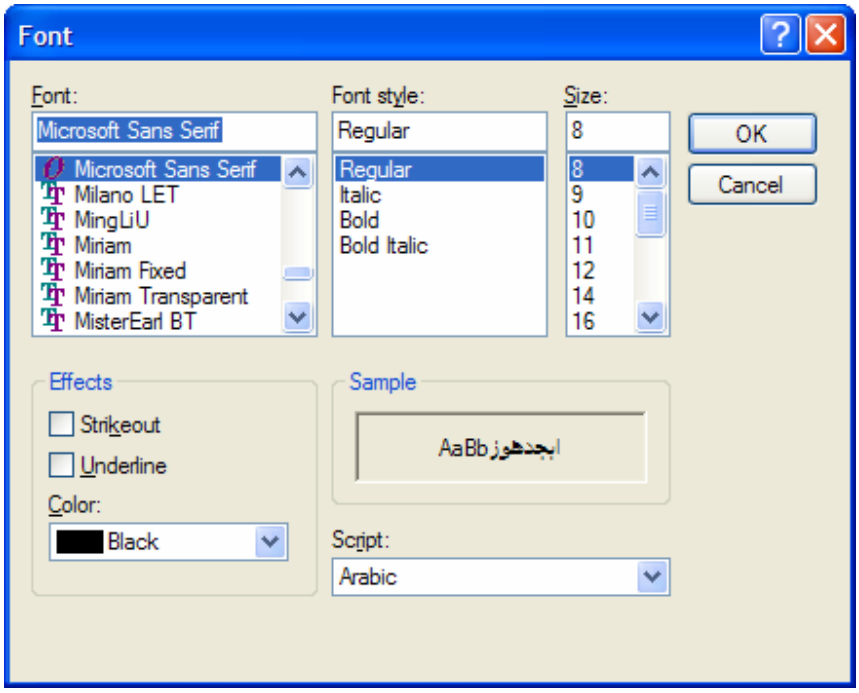
     // Show the Font dialog
     if (fontDialog1.ShowDialog() == DialogResult.OK)
     {
          // If the OK button was clicked set the font
          // in the text box on the form
          txtFile.Font = fontDialog1.Font;
          // Set the color of the font in the text box
          // on the form
          txtFile.ForeColor = fontDialog1.Color;
     }
}
```

۵) با کلیک بر روی دکمه Start در نوار ابزار برنامه را اجرا کنید. هنگامی که فرم برنامه نمایش داده شد بر روی دکمه ی Font کلیک کنید تا کادر محاوره ای Font همانند شکل ۷-۱۱ نمایش داده شود. فونت و رنگ جدیدی را برای TextBox انتخاب کرده و بر روی دکمه OK کلیک کنید.
۶) حال چندین خط متن را در فرم وارد کنید. مشاهده خواهید کرد که متن با فونت و رنگ جدید نوشته خواهد شد.
۷) همچنین اگر فایلی را با استفاده از دکمه ی Open باز کنید، رنگ و فونت جدید در آن اعمال می شود. برای تست این مورد روی دکمه ی Open کلیک کنید تا کادر Open نمایش داده شود، سپس یک فایل متنی را انتخاب کرده و آن را باز کنید. مشاهده می کنید که محتویات فایل با رنگ و فونت جدید نمایش داده می شود.

چگونه کار می کند؟

همانطور که می دانید، قسمت رنگ کادر `Font` به صورت پیش فرض نمایش داده نمی شود پس برنامه را با تنظیم مقدار این خاصیت با `True` آغاز می کنیم تا هنگام نمایش کادر `Font`، قسمت `Color` نیز نمایش داده شود.

```
// Set the FontDialog control properties
fontDialog1.ShowColor = true;
```



شکل ۷-۱۱

سپس کادر `Font` را نمایش می دهیم و با استفاده از یک دستور `if` بررسی می کنیم که کاربر بر روی دکمه `OK` کلیک کرده است و یا بر روی دکمه `Cancel`. اگر کاربر بر روی دکمه `OK` کلیک کرده باشد، همانند کادرهای دیگر، مقدار `DialogResult.OK` به وسیله تابع `ShowDialog` برگشته می شود.

```
// Show the Font dialog
if (fontDialog1.ShowDialog() == DialogResult.OK)
{
     // If the OK button was clicked set the font
     // in the text box on the form
     txtFile.Font = fontDialog1.Font;
     // Set the color of the font in the text box
     // on the form
     txtFile.ForeColor = fontDialog1.Color;
```

```
}
```

برای تغییر فونت TextBox ، خاصیت Font آن را برابر با فونتی قرار می دهیم کـه کـاربر در کنتـرل FontDialog انتخاب کـرده اسـت. بـرای ایـن کـار بایـد از خاصـیت Font کنتـرل FontDialog اسـتفاده کنـیم. همچنـین خاصـیت ForeColor کنترل TextBox را نیز برابر با خاصیت Color کنترل FontDialog قرار می دهـیم. بـه ایـن ترتیب رنگ متن داخل TextBox برابر با رنگی می شود که کاربر در کادر Font انتخاب کرده است.

کنترل ColorDialog.

در مواقعی ممکن است نیاز داشته باشید که به کاربر اجازه دهید رنگی را در برنامه انتخاب کند. برای مثال ممکن است بخواهید از این رنگ در تنظیم رنگ پس زمینه ی فرم، در تنظیم رنگ یک کنترل و یا برای تنظیم رنگ متن داخـل TextBox اسـتفاده کنیـد. ویژوال استودیو ۲۰۰۵ همانند کادر Font، یک کادر استاندارد نیز برای انتخاب رنگ در اختیار برنامـه نـویس قـرار مـی دهـد کـه ColorDialog نام دارد. همانند قسمتهای قبلی، کادر ColorDialog هم می تواند به عنوان یک کنترل و هم بـه عنوان یک کلاس مورد استفاده قرار گیرد.
کنترل ColorDialog که در شکل ۷-۱۲ نشان داده شده است، به کاربر اجازه مـی دهـد بـین ۴۸ رنـگ ابتـدایی رنگـی را انتخاب کند.



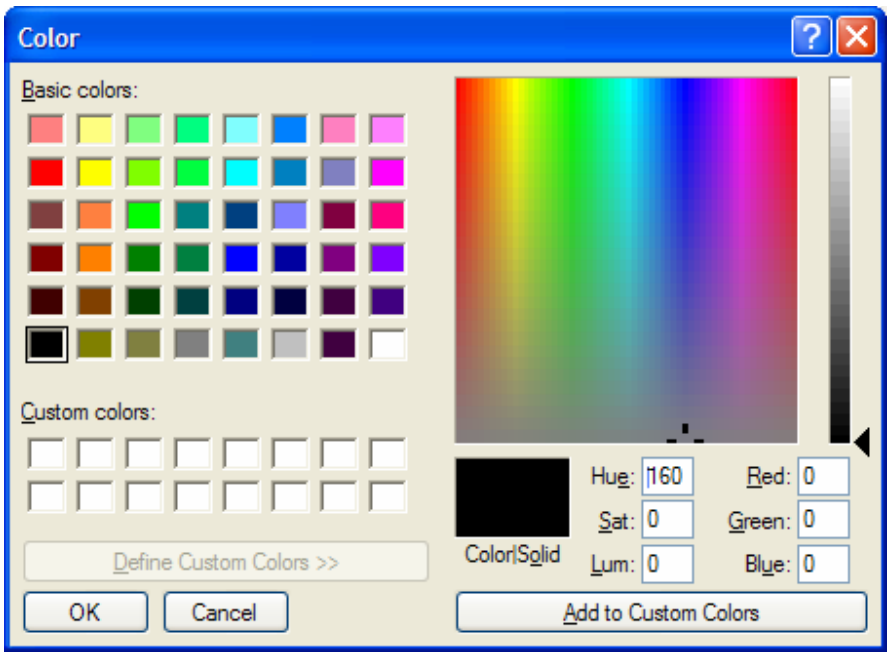
شکل ۷-۱۲

دقت کنید که علاوه بر این رنگهای ابتدایی کاربر می تواند بر روی دکمه ی Define Custom Color کلیک کـرده و با ترکیب رنگها، رنگ مورد نظر خود را ایجاد کند. این مورد باعث انعطاف پذیری بیشتر این کادر می شود و به کاربر اجازه می دهد رنگ مورد نظر خود را ایجاد کرده و در برنامه از آن استفاده کند (شکل ۷-۱۳).

خاصیتهای کنترل ColorDialog:

قبل از اینکه از این کنترل در برنامه استفاده کنیم، بهتر است بعضی از خاصیتهای پر کاربرد آن را بررسی کنیم. در جدول زیر نام تعدادی از آن خاصیت ها و نیز کاربرد آنها شرح داده شده است:

خاصیت	شرح
AllowFullOpen	مشخص می کند که آیا کاربر می تواند از قسمت Custom Color نیز برای تعریف رنگ جدید استفاده کند یا نه. در صورتی که مقدار این گزینه برابر با False باشد، دکمه فرمان Define Custom Colors غیر فعال خواهد بود.
AnyColor	مشخص می کند که آیا کادر محاوره ای تمام رنگهای موجود را به عنوان رنگهای ابتدایی نمایش دهد یا نه؟
Color	رنگی که در کادر به وسیله کاربر انتخاب شده است را مشخص می کند.
CustomColors	مجموعه رنگهایی را که در بخش Custom Color کادر نمایش داده می شود را مشخص می کند.
FullOpen	مشخص می کند که هنگام نمایش داده شدن کادر Color قسمت Custom Color هم به صورت پیش فرض دیده شود یا نه؟
ShowHelp	مشخص می کند که دکمه فرمان Help در کادر Color نمایش داده شود یا نه؟



شکل ۷-۱۳

همانطور که مشاهده می کنید خاصیتهای این کنترل نسبت به کنترل های قبلی کمتر است. همین مورد باعث می شود که استفاده از این کنترل حتی از کنترلهای قبلی نیز راحت تر باشد.
همانند کادرهای قبلی، کنترل `ColorDialog` نیز دارای تابع `ShowDialog` است که باعث نمایش آن می شود. نحوه کارکرد این تابع نیز همانند قسمتهای قبلی است. بنابراین در این قسمت از توضیح مجدد آن صرفنظر می کنیم.

استفاده از کنترل ColorDialog:

برای نمایش کادر `Color` تنها کافی است که متد `ShowDialog` آن را فراخوانی کنید:

```
colorDialog1.ShowDialog();
```

این تابع مقداری را از نوع `DialogResult` برمی گرداند که مشخص می کند کاربر در کادر بر روی دکمه `OK` کلیک کرده است و یا بر روی دکمه `Cancel`. همانند قسمت قبلی می توانید با استفاده از یک دستور `if` نتیجه کادر را بررسی کنید.
برای دسترسی به مقدار رنگی که توسط کاربر در این کادر انتخاب شده است باید از خاصیت `Color` این کنترل استفاده کنید. سپس می توانید این رنگ را به کنترلهایی که می توانید رنگ آنها را تعیین کنید نسبت دهید. برای مثال می توانید رنگ متن یک `TextBox` را برابر با رنگ انتخاب شده در این کادر قرار دهید:

```
txtFile.ForeColor = colorDialog1.Color;
```

در بخش امتحان کنید بعد، به پروژه قبلی امکانی را اضافه می کنیم که کاربر بتواند به وسیله آن رنگ زمینه فرم را تغییر دهد.

امتحان کنید: کار با کنترل `ColorDialog`

۱) پروژه `Dialogs` را باز کرده و به قسمت طراحی فرم مربوط به `Form1` بروید.
۲) با استفاده از جعبه ابزار یک کنترل `Button` بر روی فرم قرار داده و خاصیتهای آن را مطابق با مقادیر زیر تنظیم کنید:

- خاصیت `Name` آن را برابر با `btnColor` قرار دهید.
- خاصیت `Anchor` آن را برابر با `Top,Right` قرار دهید.
- خاصیت `Location` آن را برابر با `367;98` قرار دهید.
- خاصیت `Text` آن را برابر با `Color` قرار دهید.

۳) سپس با استفاده از قسمت `Dialogs` جعبه ابزار یک کنترل `ColorDialog` به برنامه اضافه کنید. این کنترل به قسمت پایین طراحی فرم اضافه خواهد شد.

۴) بر روی دکمه ی btnColor دو بار کلیک کرده تا متد مربوط به رویداد Click آن ایجاد شود. سپس کد زیر را به آن اضافه کنید:

```
private void btnColoe_Click(object sender, EventArgs e)
{
    // Show the Color dialog
    if (colorDialog1.ShowDialog() == DialogResult.OK)
    {
        // Set the BackColor property of the form
        this.BackColor = colorDialog1.Color;
    }
}
```

۵) تمام کدی که باید وارد می کردید همین بود. برای امتحان برنامه بر روی دکمه Start در نوار ابزار کلیک کنید.

۶) هنگامی که فرم برنامه نمایش داده شد، بر روی دکمه ی Color کلیک کنید تا کادر محاوره ای Color نمایش داده شود. در این کادر یکی از رنگهای ابتدایی را انتخاب کرده و یا روی دکمه Define Custom Color کلیک کنید و رنگی را از آن قسمت انتخاب کنید. سپس روی دکمه OK کلیک کنید تا کادر بسته شود.

۷) با کلیک روی دکمه OK در کادر Color رنگ زمینه فرم با رنگی که در کادر انتخاب کرده بودید تعویض می شود.

۸) همانند کادر Font نیازی نیست که قبل از نمایش فرم، خاصیت Color را برابر رنگ انتخاب شده در مرحله قبلی قرار دهید. زیرا کنترل ColorDialog خود مقدار رنگی که آخرین بار توسط کاربر انتخاب شده است را نگهداری می کند. به این ترتیب بعد از اینکه کاربر مجدداً وارد این کادر شد، مشاهده می کند رنگی که در مرحله قبل انتخاب کرده بود، همچنان به صورت انتخاب شده است.

چگونه کار می کند؟

این مرتبه برای فراخوانی تابع ShowDialog نیازی نیست که خاصیتهای کادر Color را تغییر دهیم. به همین دلیل به قسمت فراخوانی تابع درون دستور if می رویم. همانند بخشهای قبلی، اگر کاربر در کادر Color روی دکمه ی OK کلیک کند تابع ShowDialog مقدار DialogResult.OK را برمی گرداند. به همین علت در دستور if بررسی می کنیم که مقدار برگشتی از تابع برابر با DialogResult.OK هست یا نه؟

```
// Show the Color dialog
if (colorDialog1.ShowDialog() == DialogResult.OK)
```

اگر مقدار برگشتی برابر با DialogResult.OK بود باید رنگ پیش زمینه فرم را تغییر دهیم. همانطور که در قسمتهای قبلی نیز گفتیم برای دسترسی به خاصیتهای یک کلاس، درون خود کلاس باید از کلمه کلیدی this استفاده کنیم. برای دسترسی به خاصیتهای فرم نیز، به علت اینکه درون کلاس فرم هستیم از کلمه کلیدی this استفاده می کنیم. سپس خاصیت BackColor فرم را برابر با رنگی قرار می دهیم که توسط کاربر در کادر Color انتخاب شده است.

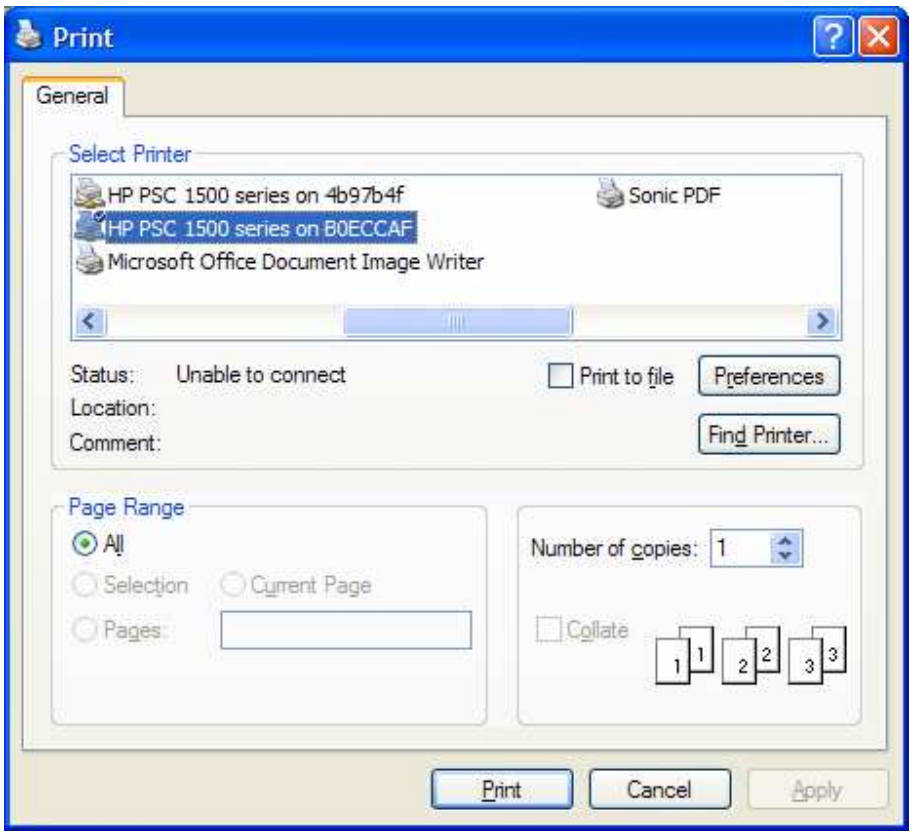
```
// Set the BackColor property of the form
this.BackColor = colorDialog1.Color;
```

کنترل PrintDialog:

هر برنامه ای معمولاً در قسمتی نیاز به امکان چاپ دارد. این نیاز می تواند به صورت نیاز بـه چـاپ سـاده ی یـک مـتن و یـا مـوارد پیشرفته تری مانند چاپ قسمتی از متن و یا صفحات مشخصی از آن باشد. در قسمت بعد به بررسی چگونگی چاپ یک مـتن سـاده خواهیم پرداخت و نحوه استفاده از کلاسهای مربوط به چاپ در `.NET` را مشاهده خواهیم کرد.

یکی از کنترل هایی که در ویژوال C# ۲۰۰۵ برای چاپ به کار می رود، کنترل `PrintDialog` است. این کنترل کار چـاپ را انجام نمی دهد، بلکه به کاربر اجازه می دهد که چاپگری را برای چاپ انتخاب کرده و تنظیمات قبل از چاپ را در آن چاپگر انجـام دهد. برای مثال کاربر می تواند در این کادر جهت صفحه، کیفیت چاپ و یا محدوده موردنظر برای چاپ را تعیین کنـد. شـما از ایـن ویژگی ها در مثال بعدی استفاده نخواهید کرد، اما همانطور که در شکل ۷-۱۴ مشاهده می کنید تمام این قابلیت ها به وسـیله کـادر `PrintDialog` قابل دسترسی است.

همانند تمام کادرهایی که در بخشهای قبلی مشاهده کردید، کادر `Print` نیز دارای دو دکمه `OK` و `Cancel` است. بنـابراین تـــابع `ShowDialog` مربـــوط بـــه ایـــن کـــادر هـــم مقـــدار `DialogResult.OK` و یـــا `DialogResult.Cancel` را برمی گرداند و می توانید از دستور `if` برای بررسی نتیجـه برگـشت داده شـده توسـط کادر استفاده کنید.



شکل ۷-۱۴

خاصیتهای کنترل PrintDialog:

در جدول زیر لیستی از خاصیتهای پر کاربرد کنترل PrintDialog و نیز توضیح آنها آمده است:

خاصیت	شرح
AllowPrintToFile	مشخص می کند آیا در کادر گزینه Print To File فعال باشد یا نه؟
AllowSelection	مشخص می کند در کادر، دکمه رادیویی Selectin فعال باشد یا نه؟
AllowSomePages	مشخص می کند در کادر، دکمه رادیویی Pages فعال باشد یا نه؟
Document	مشخص کننده سندی است که برای چاپ استفاده می شود.
PrinterSettings	تنظیماتی که در کادر، برای چاپگر انتخابی اعمال می شود را نگهداری می کند.
PrintToFile	مشخص می کند آیا گزینه Print to file انتخاب شده است یا نه؟
ShowHelp	مشخص می کند آیا دکمه فرمان Help در کادر نمایش داده شود یا نه؟
ShowNetwork	مشخص می کند دکمه فرمان Network در کادر Print نمایش داده شود یا نه؟

استفاده از کنترل PrintDialog:

برای نمایش کادر Print کافی است که تابع ShowDialog آن را فراخوانی کنید. به این صورت کادر Print همانند شکل ۷-۱۴ نشان داده خواهد شد. همانطور که پیشتر نیز گفتم کنترل PrintDialog فقط کادری را برای تنظیمات چاپ نمایش می دهد و هیچ متنی را نمی تواند چاپ کند. قطعه کد زیر برای نمایش کادر Print می تواند مورد استفاده قرار بگیرد:

```
printDialog1.ShowDialog();
```

کلاس PrintDocument:

قبل از اینکه تابع ShowDialog در کنترل PrintDialog را فراخوانی کنید، باید خاصیت Document کلاس PrintDialog را تنظیم کنید. این خاصیت مقداری را از نوع کلاس PrintDocument دریافت می کند. کلاس PrintDocument می تواند تنظیمات چاپگر را دریافت کرده و سپس با توجه به آن تنظیمات، خروجی خود (که در حقیقت همان اطلاعات مورد نظر ما است) را برای چاپ به چاپگر می فرستد. این کلاس در فضای نام System.Drawing.Printing قرار دارد. پس بهتر است که قبل از استفاده از آن برای اینکه هر بار نام کامل این فضای نام را وارد نکنیم، با استفاده از راهنمای using آن را به برنامه اضافه کنیم.

خصوصیات کلاس PrintDocument:

قبل از ادامه بهتر است نگاهی به بعضی از خصوصیات مهم کلاس PrintDocument که در جدول زیر آمده اند داشته باشیم.

خصوصیت	شرح
DefaultPageSettings	مشخص کننده ی تنظیمات پیش فرض چاپگر برای چاپ سند (اطلاعات) مورد نظر است.
DocumentName	مشخص کننده نامی است که هنگام چاپ سند نمایش داده می شود. همچنین این نام در کادر Print Status و در لیست اسناد موجود در صف چاپ برای مشخص کردن سند نوشته می شود.
PrintController	محتوی شیئ از کلاس PrintController است که پروسه چاپ را مدیریت می کند.
PrinterSettings	مشخص کننده چاپگری است که برای چاپ این سند استفاده می شود.

چاپ یک سند:

متد Print از کلاس PrintDocument سندی را به وسیله چاپگر مشخص شده در خاصیت PrinterSettings چاپ می کند. هنگامی که این تابع را در برنامه فراخوانی کنید، هر بار که صفحه ای بخواهد به وسیله این تابع چاپ شود متد مربوط به رویداد PrintPage از کلاس PrintDocument نیز فراخوانی می شود. تابع Print به وسیله این متد مشخص می کند که کدام بخش از فایل باید در صفحه جاری چاپ شود. بنابراین قبل از اینکه بتوانید متنی را چاپ کنید باید متدی را برای این رویداد ایجاد کنید. سپس در این متد باید یک صفحه از متن را به وسیله کلاس StreamReader از فایل خوانده و آن را به چاپگر بفرستید تا چاپ شود.
در بخش امتحان کنید زیر مشاهده خواهیم کرد که چگونه می توان محتویات یک فایل متنی را به وسیله کلاس PrintDocument چاپ کرد.

امتحان کنید: کار با کنترل PrintDialog

۱) در محیط ویژوال استودیو، پروژه Dialogs را باز کنید.
۲) با استفاده از جعبه ابزار کنترل Button دیگری را بر روی فرم قرار داده و خاصیتهای آن را مطابق لیست زیر تنظیم کنید:

- خاصیت Name را برابر با btnPrint قرار دهید.
- خاصیت Anchor را برابر با Top, Right قرار دهید.

- خاصیت Location را برابر با 367;128 قرار دهید.
- خاصیت Text را برابر با Print قرار دهید.

۳) در جعبه ابزار به قسمت Printing بروید و بر روی کنترل PrintDialog دو بار کلیک کنید تا بـر روی فرم قرار بگیرد. مشاهده خواهید کرد که این کنترل نیز همانند کادرهای قبلی، در پایین قـسمت طراحـی فـرم قـرار مـی گیرد.

۴) به قسمت ویرایشگر کد بروید و با استفاده از راهنمای using فضای نامهای زیر را به برنامه اضافه کنید:

```
using System.IO;
using System.Drawing.Printing;
```

۵) حال متغیرهای زیر را به صورت عمومی در ابتدای کلاس مربوط به فرم برنامه تعریف کنید:

```
// Declare variables
private string strFileName;

private StreamReader objStreamToPrint;
private Font objPrintFont;
```

۶) به قسمت طراحی فرم برگردید و بر روی دکمه ی btnPrint دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس کد زیر را در این متد وارد کنید:

```
private void btnPrint_Click(object sender, EventArgs e)
{
     // Declare an object for the PrintDocument class
     PrintDocument objPrintDocument = new PrintDocument();

     // Set the DocumentName property
     objPrintDocument.DocumentName = "Text File Print
Demo";

     // Set the PrintDialog properties
     printDialog1.AllowPrintToFile = false;
     printDialog1.AllowSelection = false;
     printDialog1.AllowSomePages = false;

     // Set the Document property for
     // the objPrintDocument object
     printDialog1.Document = objPrintDocument;

     // Show the Print dialog
     if (printDialog1.ShowDialog() == DialogResult.OK)
     {
          // If the user clicked on the OK button
```

```
            // then set the StreamReader object to
            // the file name in the strFileName variable
            objStreamToPrint = new StreamReader(strFileName);

            // Set the printer font
            objPrintFont = new Font("Arial", 10);

            // Set the PrinterSettings property of the
            // objPrintDocument Object to the
            // PrinterSettings property returned from the
            // PrintDialog control
            objPrintDocument.PrinterSettings =
                 printDialog1.PrinterSettings;

            // Add an event handler for the PrintPage event
of
            // the objPrintDocument object
            objPrintDocument.PrintPage +=
                 new PrintPageEventHandler(prtPage);

            // Print the text file
            objPrintDocument.Print();

            // Clean up
            objStreamToPrint.Close();
            objStreamToPrint = null;
       }
}
```

۷) سپس متد زیر را در قسمت ویرایشگر کد وارد کنید:

```
private void prtPage(object sender, PrintPageEventArgs e)
{
     // Declare variables
     float sngLinesPerpage = 0;
     float sngVerticalPosition = 0;
     int intLineCount = 0;
     float sngLeftMargin = e.MarginBounds.Left;
     float sngTopMargin = e.MarginBounds.Top;
     string strLine;

     // Work out the number of lines per page.
     // Use the MarginBounds on the event to do this
     sngLinesPerpage = e.MarginBounds.Height /

     objPrintFont.GetHeight(e.Graphics);
```

```
     // Now iterate through the file printing out each
line.
     // This assumes that a single line is not wider than
     // the page width. Check intLineCount first so that we
     // don't read a line that we won't print
     strLine = objStreamToPrint.ReadLine();
     while((intLineCount < sngLinesPerpage) &&
           (strLine != null))
     {
          // Calculate the vertical position on the page
          sngVerticalPosition = sngTopMargin +
     (intLineCount * objPrintFont.GetHeight(e.Graphics));

          // Pass a StringFormat to DrawString for the
          // Print Preview control
          e.Graphics.DrawString(strLine, objPrintFont,
                                Brushes.Black, sngLeftMargin,
                                sngVerticalPosition,
                                new StringFormat());

          // Increment the line count
          intLineCount = intLineCount + 1;

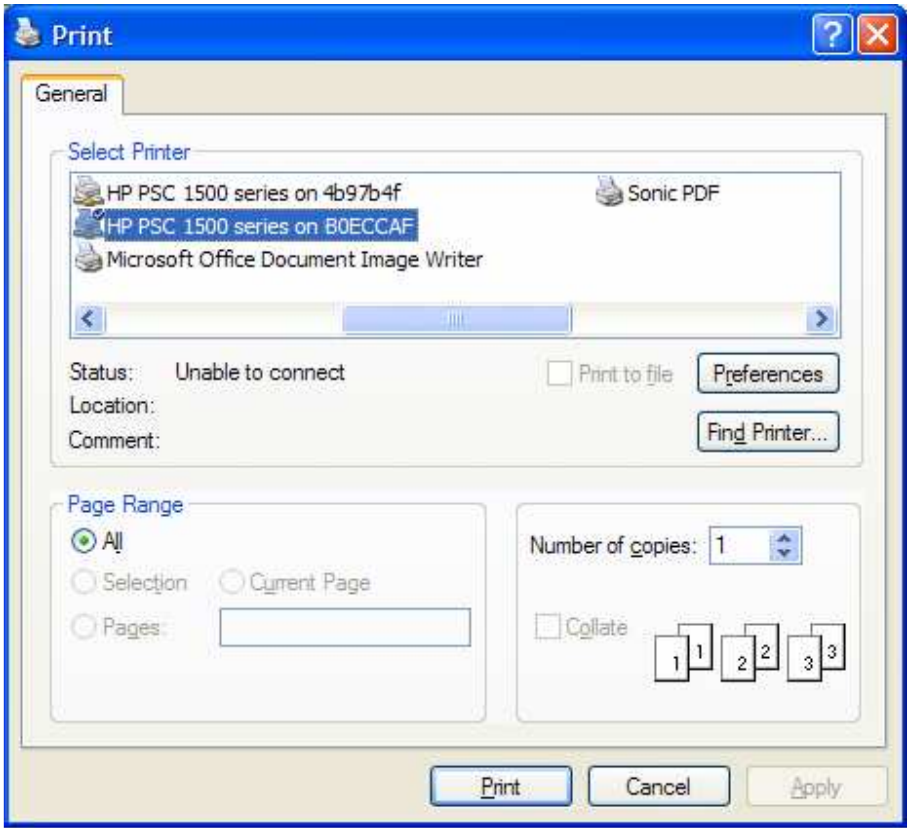
          // If the line count is less than the lines per
          // page then read another line of text
          if (intLineCount < sngLinesPerpage)
          {
               strLine = objStreamToPrint.ReadLine();
          }
     }

     // If we have more lines then print another page
     if (strLine != null)
     {
          e.HasMorePages = true;
     }
     else
     {
          e.HasMorePages = false;
     }
}
```

۸) حال می توانید عملکرد کدهای این قسمت را مشاهده کنید. بنابراین روی دکمه Start در نوار ابزار کلیک کنید تا برنامه اجرا شود.

۹) در فرم اصلی برنامه روی دکمه ی `Open` کلیک کنید و فایلی را باز کنید تا محتویات آن در فرم نمایش داده شود. سپس بر روی دکمه ی `Print` کلیک کنید تا کادر محاوره ای `Print` همانند شکل ۷-۱۵ نمایش داده شود. توجه کنید که در این کادر گزینه `Print to file` و همچنین قسمتهای `Selection` و `Pages` غیر فعال هستند. دلیل غیر فعال بودن این قسمتها به خاطر این است که قبل از فراخوانی تابع `ShowDialog` خاصیتهای `AllowPrintToFile`، `AllowSelection` و `AllowSomePages` را برابر با `False` قرار دادیم. اگر در سیستم خود بیش از یک چاپگر داشته باشید، همانند شکل ۷-۱۵ می توانید تعیین کنید که فایل باز شده به وسیله کدام چاپگر، چاپ شود.

۱۰) روی دکمه `Print` در کادر کلیک کنید تا محتویات فایل چاپ شوند.



شکل ۷-۱۵

چگونه کار می کند؟

یکی از مهمترین قسمتهای این برنامه، متد مربوط به رویداد کلیک دکمه ی `btnPrint` است. همانطور که مشاهده می کنید این متد را با تعریف یک شیئ از کلاس `PrintDocument` آغاز کردیم. عمل اصلی چاپ به وسیله این شیئ صورت می گیرد:

```
PrintDocument objPrintDocument = new PrintDocument();
```

سپس خاصیت DocumentName مربوط به این شیئ را تنظیم کردیم. اگر همزمان چند برنامه بخواهند از چاپگر استفاده کنند، سندهای آنها در یک صف چاپ قرار می گیرد. نامی که در این قسمت وارد می کنیم، برای مشخص کردن سند مربوط به برنامه ما در صف چاپ به کار می رود.

```
objPrintDocument.DocumentName = "Text File Print Demo";
```

در قسمت بعد، به تنظیم بعضی از خاصیتهای کنترل PrintDialog می پردازیم. در این بخش فقط می خواهیم یک عمل چاپ ساده را انجام دهیم، به همین دلیل بهتر است قسمتهای Print to file و همچنین Selection و Pages را در کادر Print غیر فعال کنیم. برای این کار کافی است خاصیتهای مربوط به آنها را برابر با false قرار دهیم:

```
printDialog1.AllowPrintToFile = false;
printDialog1.AllowSelection = false;
printDialog1.AllowSomePages = false;
```

قبل از نمایش کادر Print باید مشخص کنید تنظیماتی که کاربر در این کادر مشخص می کند، برای چاپ چه سندی به کار می روند. برای این کار باید خاصیت Document کنترل PrintDialog را برابر با شیئ PrintDocument ای قرار دهید که نشان دهنده ی سند مورد نظر است.

```
printDialog1.Document = objPrintDocument;
```

حال می توانیم کادر Print را نمایش دهیم. همانند کادرهای قبلی، برای این کار کافی است متد ShowDialog مربوط به این کنترل را فراخوانی کنیم. این متد نیز مقداری را از نوع DialogResult برمی گرداند. اگر کاربر در کادر روی دکمه ی Print کلیک کند، تابع DialogResult.OK و اگر کاربر روی دکمه ی Cancel کلیک کند، تابع DialogResult.Cancel را برمی گرداند. همانند قسمتهای قبل با استفاده از دستور if نتیجه را بررسی می کنیم.

```
if (printDialog1.ShowDialog() == DialogResult.OK)
```

اگر کاربر در کادر روی دکمه Print کلیک کند باید محتویات فایلی که آدرس آن در متغیر strFileName قرار دارد را چاپ کنیم. بنابراین ابتدا یک شیئ از نوع StreamReader تعریف می کنیم. این شیئ برای دسترسی به محتویات یک فایل مورد استفاده قرار می گیرد و هنگام تعریف آن باید آدرس فایل مورد نظر را به آن بفرستیم. پس متغییر strFileName که حاوی آدرس فایل است را به عنوان پارامتر به این شیئ ارسال می کنیم.

```
objStreamToPrint = new StreamReader(strFileName);
```

سپس باید فونت و اندازه متن را برای چاپ مشخص کنیم. به همین علت شیئ را ز نوع Font تعریف کرده و فونت Arial و اندازه 10 را برای آن تعریف می کنیم.

```
objPrintFont = new Font("Arial", 10);
```

همانطور که در قسمتهای قبل نیز گفتم، هنگامی که یک رویداد به وسیله یک کلاس رخ می دهد، تعدادی از توابع برای پاسخ دادن به آن رویداد اجرا می شوند. برای مثال در قسمتهای قبل مشاهده کردید با دو بار کلیک بر روی یک کنترل Button در زمان طراحی، متدی ایجاد می شد و این متد هنگامی که کاربر روی آن کنترل در طی اجرای برنامه کلیک می کرد، توسط برنامه فراخوانی می شد. برای بررسی دقیقتر این مورد باید بگویم که هر رویداد شامل لیستی از توابع است. هنگامی که رویداد رخ می دهد، کلاس مربوطه تمام توابع موجود در لیست مربوط به آن رویداد را فراخوانی می کند. برای مثال می توانید چندین متد تعریف کنید و آنها را به رویداد کلیک یک Button اضافه کنید. به این ترتیب هنگامی که بر روی آن دکمه کلیک شود تمام متدهایی که به آن اضافه کرده اید اجرا خواهند شد.

در قسمتهای قبل گفتم که کلاس PrintDocument برای اینکه تشخیص دهد چه متنی را باید چاپ کند، در هر صفحه رویداد PrintPage را فراخوانی می کند. به عبارت دقیق تر باید بگوییم که این کلاس در هر مرحله تمام توابعی که در لیست رویداد PrintPage هستند را اجرا می کند. پس باید متدی را ایجاد کنیم و آن را به لیست متدهای رویداد PrintPage اضافه کنیم. برای این کار متد prtPage را ایجاد کرده و آن را به وسیله دستور زیر به رویداد PrintPage اضافه می کنیم[1]:

```
objPrintDocument.PrintPage +=
    new PrintPageEventHandler(prtPage);
```

حال باید چاپگر مورد استفاده برای چاپ و همچنین تنظیم های آن را، برای شیئ PrintDocument مشخص کنیم. برای این کار کافی است تنظیمهایی که کاربر در کادر Print مشخص کرده است را به این شیئ بفرستیم. تنظیم های کادر Print در خاصیت PrinterSettings ذخیره می شوند. پس کافی است خاصیت objPrintDocument.PrinterSettings را برابر با آن قرار دهیم.

```
objPrintDocument.PrinterSettings =
        printDialog1.PrinterSettings;
```

حال باید متد Print را فراخوانی کنیم. این متد رویداد PrintPage را احضار می کند و احضار این رویداد نیز باعث می شود که کد درون متد prtPage اجرا شود.

```
objPrintDocument.Print();
```

نکته دیگری که در اضافه کردن یک متد به یک رویداد باید در نظر داشته باشید این است که متدهایی که می توانند به رویداد PrintPage اضافه شوند باید دارای ساختار خاصی باشند. این متدها نباید مقداری را برگردانند (مقدار برگشتی آنها باید به صورت void تعریف شود). همچنین باید دو پارامتر را از ورودی دریافت کنند: اولین پارامتر مشخص کننده شیئ است که این رویداد را فراخوانی کرده است. نام این پارامتر sender و نوع آن از نوع کلاس Object خواهد بود. دومین پارامتر باید شیئ از نوع کلاس PrintPageEventArgs باشد. این پارامتر حاوی اطلاعاتی در مورد صفحه ای که باید چاپ شود خواهد بود. بنابراین متد prtPage که باید به وسیله رویداد PrintPage فراخوانی شود مشابه زیر خواهد بود:

[1] در دو فصل مربوط به برنامه نویسی شیئ گرا و فصول بعد از آن، متدها را دقیق تر بررسی خواهیم کرد.

```
private void prtPage(object sender, PrintPageEventArgs e)
```

حال به بررسی کدهایی می پردازیم که در داخل این متد باید اجرا شوند. ابتدا باید تعدادی متغییر تعریف کنیم و مقادیر آنها را تنظیم کنیم. توجه کنید که مقادیر متغیرهای sngLeftMargin و sngTopMargin به وسیله مقادیر موجود در پارامتر PrintPageEventArgs که به متد ارسال می شود تنظیم خواهد شد.

```
float sngLinesPerpage = 0;
float sngVerticalPosition = 0;
int intLineCount = 0;
float sngLeftMargin = e.MarginBounds.Left;
float sngTopMargin = e.MarginBounds.Top;
string strLine;
```

حال باید مشخص کنیم که در هر صفحه چند خط می تواند چاپ شود. برای این کار باید ارتفاع قابل چاپ در صفحه را بر ارتفاع فونت (ارتفاع هر خط) تقسیم کنیم. برای دسترسی به ارتفاع قابل چاپ در صفحه می توانیم از خاصیت MarginBounds.Height در شیئ e از کلاس PrintPageEventArgs استفاده کنیم (این شیئ، به عنوان پارامتر به متد فرستاده شده است).
ارتفاع قابل چاپ در صفحه در کادر PrintDialog تنظیم شده و در خاصیت PrinterSettings قرار می گیرد. همانطور که مشاهده کردید در کدهای قبلی این خاصیت را در خاصیت PrinterSettings مربوط به شیئ objPrintDocument قرار دادیم. شیئ objPrintDoument هم هنگامی که بخواهد رویداد PrintPage را فراخوانی کند، این مقدار را به وسیله شیئ از کلاس PrintPageEventArgs به متدهای فراخوانی شده می فرستد.

```
sngLinesPerpage = e.MarginBounds.Height /

objPrintFont.GetHeight(e.Graphics);
```

پس به این ترتیب متغییر sngLinesPerPage حاوی تعداد خطوطی خواهد بود که در هر صفحه قرار می گیرد. حال باید محتویات فایل را خط به خط خوانده و در صفحه برای چاپ قرار دهیم. برای این کار با استفاده از یک حلقه، متن داخل فایل را خط به خط در صفحه وارد می کنیم. اجرای این حلقه تا زمانی ادامه پیدا می کند که یا متن داخل فایل تمام شود و به انتهای فایل برسیم و یا تعداد خطهایی که در صفحه قرار داده ایم برابر با حداکثر تعداد خطهایی شود که می توانیم در یک صفحه وارد کنیم، به عبارت دیگر صفحه پر شود. بنابراین ابتدا اولین خط را خوانده و در متغیر strLine قرار می دهیم و سپس حلقه را اجرا می کنیم:

```
strLine = objStreamToPrint.ReadLine();
while((intLineCount < sngLinesPerpage) && (strLine !=
null))
{
```

قبل از اینکه متنی را در صفحه قرار دهیم باید مشخص کنیم که موقعیت عمودی متن در صفحه چقدر است. به عبارت دیگر باید فاصله متن را از بالای صفحه مشخص کنیم. برای تعیین فاصله متن از بالای صفحه باید اندازه قسمت سفید بالای صفحه را با

ارتفاع تعداد خطهایی که تاکنون چاپ شده اند جمع کنیم. برای محاسبه ارتفاع تعداد خطهایی که تاکنون چاپ شده اند نیز باید حاصل ضرب تعداد خطهایی که در صفحه چاپ شده اند در ارتفاع هر خط را بدست آوریم:

```
sngVerticalPosition = sngTopMargin +
(intLineCount * objPrintFont.GetHeight(e.Graphics));
```

برای اینکه واقعاً متن را به چاپگر بفرستیم باید از تابع DrawString در کلاس Graphics استفاده کنیم. کلاس Graphics به صورت یکی از خاصیتهای کلاس PrintPageEventArgs به این متد فرستاده می شود. پارامترهایی که تابع DrawString دریافت می کند عبارتند از: متنی که باید چاپ شود، فونت متنی که باید چاپ شود، رنگ متنی که باید چاپ شود (این رنگ باید از شمارنده ی Brushes انتخاب شود)، فاصله متن از سمت چپ صفحه، فاصله متن از بالای صفحه، و قالب متن برای چاپ.در این قسمت قالبی برای متن مشخص نمی کنیم بلکه یک شیئ جدید از کلاس StringFormat ایجاد کرده و آن را به تابع می فرستیم.

```
e.Graphics.DrawString(strLine, objPrintFont,
                      Brushes.Black, sngLeftMargin,
                      sngVerticalPosition,
                      new StringFormat());
```

به این ترتیب یک خط از متن را چاپ کرده ایم، پس یک واحد به تعداد خطها اضافه می کنیم:

```
intLineCount = intLineCount + 1;
```

حال بررسی می کنیم که صفحه پر شده است یا نه. اگر صفحه پر نشده بود خط دیگری را از فایل خوانده و در متغییر strLine قرار میدهیم تا حلقه با خط جدید ادامه پیدا کند:

```
if (intLineCount < sngLinesPerpage)
{
     strLine = objStreamToPrint.ReadLine();
}
```

بعد از اینکه یک صفحه کاملاً پر شد، برنامه از حلقه خارج می شود. حال باید مشخص کنیم که صفحه دیگری هم باید چاپ شود و یا اینکه متن داخل فایل تمام شده است. اگر متن داخل فایل تمام شده بود باید خاصیت HasMorePages را برابر با false قرار دهیم که تابع Print بار دیگر باعث فراخوانی شدن رویداد PrintPage نشود. اما اگر متن تمام نشده بود، کافی است که برای چاپ ادامه ی متن، خاصیت HasMorePages را برابر با true قرار دهیم. به این ترتیب تابع Print بار دیگر رویداد PrintPage را فراخوانی می کند تا متد prtPage بتواند صفحه بعد را چاپ کند.

```
if (strLine != null)
{
     e.HasMorePages = true;
}
else
```

```
{
        e.HasMorePages = false;
}
```

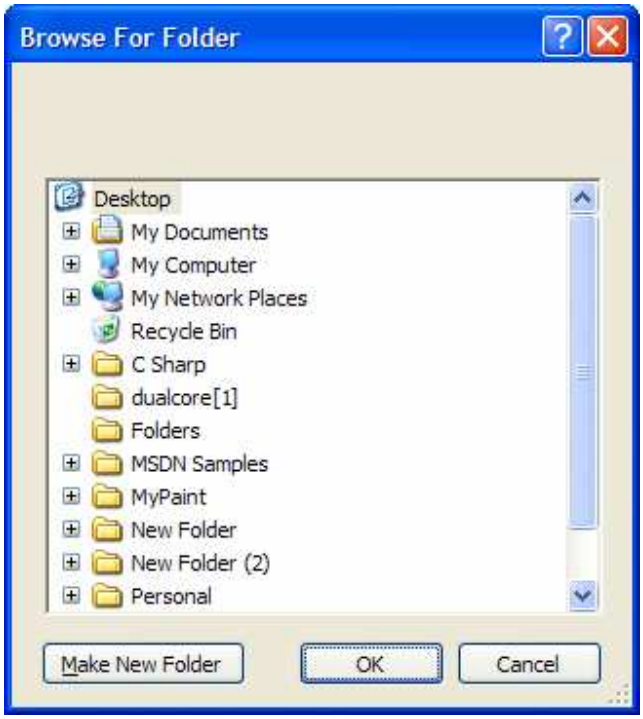
هنگامی که تمام متن داخل فایل به چاپگر فرستاده شد، وظیفه متد Print تمام شده است و برنامه به ادامه کدهای موجود در متد btnPrint_Click برمی گردد. تنها کاری که باید در ادامه انجام دهیم این است که فضای اشغال شده به وسیله اشیای مربوط به چاپ و نیز اشیای مربوط به خواندن از فایل را آزاد کنیم.

```
objStreamToPrint.Close();
objStreamToPrint = null;
```

کنترل FolderBrowserDialog:

بعضی از مواقع ممکن است در برنامه نیاز داشته باشید به کاربر اجازه دهید که به جای انتخاب یک فایل، یک فولدر را مشخص کند. برای مثال ممکن است بخواهید کاربر فولدری را برای ذخیره فایلهای پشتیبان[1] و یا فولدری را برای ذخیره فایلهای موقتی برنامه مشخص کند.در این مواقع می توانید با استفاده از کنترل FolderBrowserDialog، کادر استاندارد Browse For Folder را در برنامه نمایش دهید. همانطور که ممکن است در دیگر برنامه های ویندوزی نیز مشاهده کرده باشید، این کادر فقط فولدرهای موجود در کامپیوتر را نمایش می دهد و به واسطه آن کاربر می تواند فولدری را در برنامه مشخص کند.
همانند تمام کادرهای دیگر، کادر FolderBrowser نیز هم می تواند به صورت کنترل مورد استفاده قرار گیرد و هم به صورت یک کلاس. شکل ۷-۱۶ یک کادر FolderBrowser را بدون تنظیم خاصیتهای آن (با مقادیر پیش فرض خاصیت ها) نمایش می دهد. توجه کنید که در قسمت پایین این فرم یک دکمه فرمان Make New Folder وجود دارد که به کاربر اجازه می دهد فولدر جدیدی را ایجاد کند.

[1] Backup Files



شکل ۷-۱۶

خاصیتهای کنترل FolderBrowser:

قبل از اینکه نحوه استفاده از این کنترل را در کد مشاهده کنیم بهتر است به بررسی خاصیتهای مهم آن بپردازیم. در جدول زیر لیستی از نام و نحوه استفاده از خاصیتهای مهم این کنترل آورده شده است.

نام خاصیت	شرح
`Description`	مشخص کننده متنی است که به عنوان توضیح در کادر نمایش داده می شود.
`RootFolder`	مشخص کننده آدرس فولدری است که به صورت پیش فرض باید در کادر نمایش داده شود.
`SelectedPath`	مشخص کننده آدرس مسیری است که به وسیله کاربر انتخاب شده است.
`ShowNewFolderButton`	مشخص می کند آیا دکمه ی `Make New Folder` در کادر نمایش داده شود یا نه؟

کادر محاوره ای `FolderBrowser` اولین کادری است که تقریباً از تمام خاصیتهای آن استفاده خواهید کرد.
همانند تمام کادرهای دیگر، این کنترل نیز دارای متدی به نام `ShowDialog` است که باعث نمایش داده شدن کادر در برنامه می شود. نحوه استفاده از این متد در این کنترل همانند کادرهای دیگر است، بنابراین نیازی به توضیح مجدد آن نیست.

استفاده از کنترل FolderBrowser:

همانند تمام کادرهای محاوره ای دیگر، قبل از نمایش کادر Browse For Folder باید بعضی از خاصیتهای آن را تغییر دهیم. سه خاصیتی که عموماً قبل از نمایش این کادر تنظیم می شوند در قطعه کد زیر نشان داده شده اند. اولین خاصیت Description است که یک توضیح و یا دستورالعمل را برای کاربر در صفحه نمایش می دهد. متنی که در این خاصیت قرار داده شود، هنگام فراخوانی تابع ShowDialog در بالای کادر نوشته خواهد شد.
خاصیت بعدی خاصیت RootFolder است. این خاصیت مشخص می کند که هنگام نمایش کادر، چه فولدری به صورت پیش فرض نمایش داده شود. این خاصیت مقداری را از شمارنده Environment.SpecialFolder دریافت می کند و این شمارنده نیز خود حاوی آدرس فولدرهای مخصوص سیستم عامل ویندوز مانند فولدر My Documents است.
خاصیت دیگری که قبل از نمایش کادر تنظیم می شود، خاصیت ShowNewFolderButton است. اگر مقدار این خاصیت برابر با true باشد، دکمه ی Make New Folder در کادر نمایش داده می شود تا به کاربر اجازه داده شود فولدر جدیدی را ایجاد کند. در غیر این صورت این دکمه نمایش داده نخواهد شد.

```
folderBrowserDialog1.Description =
                "Select a folder for your backups:";
folderBrowserDialog1.RootFolder =
                Environment.SpecialFolder.MyComputer;
folderBrowserDialog1.ShowNewFolderButton = false;
```

بعد از تنظیم خاصیتهای لازم، می توانید با فراخوانی تابع ShowDialog کادر Browse For Folder را نمایش دهید:

```
folderBrowserDialog1.ShowDialog();
```

این تابع نیز همانند کادرهای قبلی مقداری را از نوع DialogResult برمی گرداند که می توانید با استفاده از یک دستور if به بررسی نتیجه آن بپردازید. برای دسترسی به آدرس فولدری که کاربر انتخاب کرده است می توانید از مقدار خاصیت SelectedPath استفاده کرده و آن را در متغیری ذخیره کنید. این خاصیت آدرس کامل فولدر انتخاب شده توسط کاربر را برمی گرداند. برای مثال اگر کاربر فولدر temp را درون درایو C انتخاب کند، مقدار این خاصیت به صورت C:\temp خواهد بود.

```
strFileName = folderBrowserDialog1.SelectedPath;
```

در امتحان کنید بعد، مجدداً از پروژه Dialogs استفاده کرده و کادر Browse For Folder را نمایش می دهیم. اگر کاربر فولدری را در این کادر انتخاب کرد، آدرس آن را در TextBox درون فرم نمایش خواهیم داد.

امتحان کنید: کار با کنترل FolderBrowser

۱) به قسمت طراحی فرم در پروژه Dialogs بروید.

۲) با استفاده از جعبه ابزار کنترل Button دیگری را به فرم برنامه اضافه کرده و خاصیتهای آن را بر طبق لیست زیر تنظیم کنید:

- خاصیت Name را برابر با btnBrowse قرار دهید.
- خاصیت Text را برابر با Browse قرار دهید.
- خاصیت Location را برابر با 367;158 قرار دهید.
- خاصیت Anchor را برابر با Top,Right قرار دهید.

۳) حال باید یک کنترل FolderBrowserDialog را به برنامه اضافه کنید. برای این کار در جعبه ابزار به قسمت Dialogs بروید و بر روی کنترل FolderBrowserDialog دو بار کلیک کنید. مشاهده خواهید کرد که این کنترل نیز همانند کنترلهای قبلی به قسمت پایین بخش طراحی فرم اضافه خواهد شد.

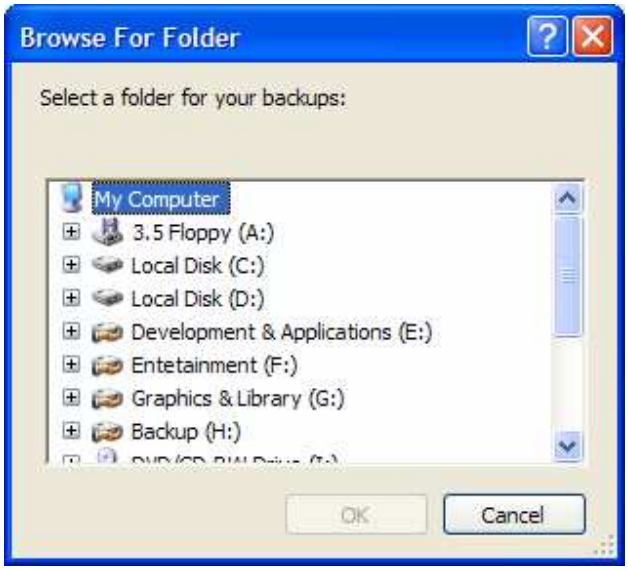
۴) بر روی دکمه ی btnBrowse دو بار کلیک کنید تا متد مربوط به رویداد Click آن ایجاد شود. سپس کد زیر را در آن متد وارد کنید:

```
private void btnBrowse_Click(object sender, EventArgs e)
{
    // Set the FolderBrowserDialog control properties
    folderBrowserDialog1.Description =
                    "Select a folder for your backups:";
    folderBrowserDialog1.RootFolder =
                    Environment.SpecialFolder.MyComputer;
    folderBrowserDialog1.ShowNewFolderButton = false;

    // Show the Browse For Folder dialog
    if (folderBrowserDialog1.ShowDialog() ==
                            DialogResult.OK)
    {
        // Display the selected folder
        txtFile.Text = folderBrowserDialog1.SelectedPath;
    }
}
```

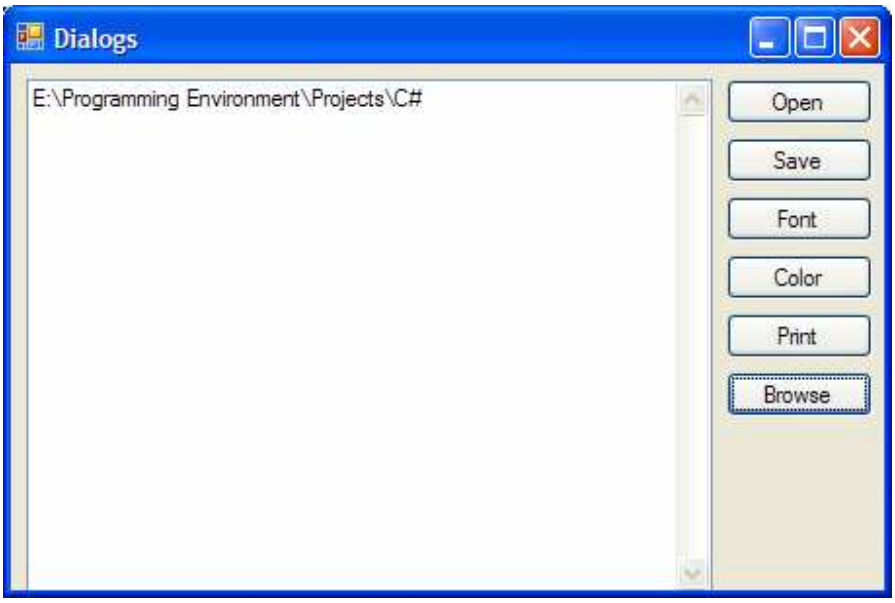
۵) تمام کد مورد نیاز برای این برنامه همین بود. برای امتحان عملکرد برنامه، در نوار ابزار روی دکمه Start کلیک کنید.

۶) هنگامی که فرم برنامه نمایش داده شد، روی دکمه ی Browse کلیک کنید. کادر Browse For Folder همانند شکل ۷-۱۷ نمایش داده خواهد شد.



شکل ۷-۱۷

۷) فولدری را در کامپیوتر خود مشخص کرده و روی دکمه فرمان OK کلیک کنید. مشاهده خواهید کرد که آدرس کامل فولدر مشخص شده، همانند شکل ۷-۱۸ در فرم نمایش داده خواهد شد.



شکل ۷-۱۸

چگونه کار می کند؟

قبل از اینکه کادر `Browse For Folder` را نمایش دهید، باید بعضی از خاصیتهای آن را تنظیم کنید تا ظاهر آن با برنامه هماهنگ شود. اولین خاصیت، خاصیت `Description` است که برای راهنمایی کاربر در مورد این کادر به کار می

رود. همانطور که مشاهده مـی کنیـد مقـدار ایـن متغییـر در ابتـدای ایـن کـادر نمـایش داده مـی شـود. خاصـیت بعـدی، خاصـیت RootFolder است که فولدر پیش فرض را مشخص می کند. در این برنامه My Computer را به عنوان فولدر پیش فرض در نظر گرفته ایم. همانطور که مشاهده می کنید، هنگام نمایش کادر درایوهـای موجـود در My Computer بـه طـور پیش فرض نمایش داده می شوند. در انتها نیز خاصـیت ShowNewFolderButton را برابـر بـا false قـرار مـی دهیم تا این دکمه در کادر نمایش داده نشود.

```
// Set the FolderBrowserDialog control properties
folderBrowserDialog1.Description =
     "Select a folder for your backups:";
folderBrowserDialog1.RootFolder =
     Environment.SpecialFolder.MyComputer;
folderBrowserDialog1.ShowNewFolderButton = false;
```

سپس کادر ای را با فراخوانی متد ShowDialog نمایش می دهیم و سپس با استفاده از دسـتور if بررسـی مـی کنـیم کـه کاربر روی دکمه OK کلیک کرده است و یا روی دکمه Cancel:

```
// Show the Browse For Folder dialog
if (folderBrowserDialog1.ShowDialog() ==
     DialogResult.OK)
{
```

اگر کاربر روی دکمه OK کلیک کرده بود، آدرس فولدر انتخاب شده توسط کاربر را که در خاصـیت SelectedPath قـرار گرفته است در فرم نمایش داده خواهد شد.

```
// Display the selected folder
txtFile.Text = folderBrowserDialog1.SelectedPath;
```

نتیجه:

در این فصل بعضی از کادرها را که در برنامه های ویژوال C# ۲۰۰۵ قابل استفاده است را بررسی کردیم. ایـن کادرهـا عبـارت انـد از:MessageBox، OpenFileDialog، SaveFileDialog، FontDialog، ColorDialog، PrintDialog و FolderBrowserDialog. همانطور که مـشاهده کردیـد، ایـن کادرها رابطهای کاربری استاندارد را برای برنامه فراهم می کنند و به واسطه آنها می توانید برنامه ای با ظاهر حرفه ای تـر و مـشابه دیگر برنامه های ویندوزی طراحی کنید.
اگرچه برای استفاده از این کادرها از کنترلهای متناظر آنها در جعبه ابزار استفاده کردید، اما به خاطر داشته باشید که تمام این کادرهـا می توانند همانند یک کلاس مورد استفاده قرار بگیرند. به عبارت دیگر کلاس متناظر با این کنترل ها نیز همین خاصیت ها و متـدها را ارائه می دهند و تفاوتی ندارد که در برنامه از آنها به عنوان کلاس استفاده کنید و یا به عنوان کنترل. برای استفاده از این کادرها به صورت کنترل می توانید متغیری را از نوع کلاس مرتبط با کادر مورد نظرتان تعریف کنید و در هر قسمتی از برنامه که خواستید از آن

کادر استفاده کنید با استفاده از دستور new کادر را ایجاد کنید. بعد از استفاده هم می توانید متغییر را از بین ببرید تـا حافظـه گرفتـه شده به وسیله آن آزاد شود. به این ترتیب حافظه کمتری در برنامه استفاده خواهید کرد و برنامه کارایی بیشتری خواهد داشت.

در پایان این فصل باید با موارد زیر آشنا شده باشید:

- با استفاده از کلاس MessageBox یک کادر پیغام را به کاربر نمایش دهید.
- در یک کادر پیغام از آیکون و دکمه های مورد نظر استفاده کنید.
- از کنترل OpenFileDialog استفاده کنید و محتویات یک فایل را بخوانید.
- از کنترل SaveFileDialog استفاده کرده و اطلاعاتی را در یک فایل بنویسید.
- با استفاده از کنترل FontDialog رنگ و فونت یک متن را تغییر دهید.
- با استفاده از کنترل ColorDialog رنگ پیش زمینه یک کنترل، مانند فرم را تغییر دهید.
- با استفاده از کنترل PrintDialog متنی را چاپ کنید.
- با استفاده از کنترل FolderBrowserDialog به کاربر اجازه دهید فولدری را در برنامه انتخاب کند.

تمرین:

تمرین ۱:

برنامه ویندوزی ساده ای ایجاد کرده و در فرم آن یک TextBox و دو Button قرار دهید. کنترل های Button را به گونه ای تنظیم کنید که فایلی را باز کرده، نمایش دهند و آن را ذخیره کننـد. بـرای بـاز کـردن و ذخیـره کـردن فایـل از کلاسـهای OpenFileDialog و SaveFileDialog استفاده کنید (نه از کنترلهای آنها).
راهنمایی: برای تعریف کلاسهای متناظر با کنترلهای OpenFileDialog و SaveFileDialog می توانید از دستورات زیر استفاده کنید:

```
OpenFileDialog openFileDialog1 = new OpenFileDialog();
SaveFileDialog saveFileDialog1 = new SaveFileDialog();
```

تمرین ۲:

یک برنامه ی ساده ی ویـندوزی ایجـاد کنیـد کـه شـامل یـک کنتـرل Label و یـک کنتـرل Button باشـد. در کنتـرل Button کدی را وارد کنید که یک کادر Browser For Folder را نمایش دهد. همچنین در این کادر دکمه ی Make New Folder را نیز فعال کنید. فولدر My Documents را در این کنترل به عنوان فولدر پیش فرض قرار

داده و سپس کادر را نمایش دهید. برای استفاده از کادر Browser For Folder از کلاس FolderBrowserDialog استفاده کنید. در انتها نیز آدرس انتخاب شده به وسیله ی کاربر را در کنترل Label قرار دهید.

فصل هشتم: منو ها

منو ها جز تفکیک ناپذیر هر برنامه ی خوب محسوب می شوند و راهی راحت و پر کاربرد برای دسترسی کاربر به تمام قسمتهای برنامه را فراهم می کنند. برای مثال برنامه ی ویژوال استودیو با استفاده از منو ها، به برنامه نویس این امکان را می دهد که به راحتی به ابزارهای این محیط دسترسی داشته باشد و بتواند از آنها استفاده کند. همچنین در محیط کد نویسی با استفاده از منو های فرعی[1] می توانید به راحتی اعمالی مانند کات کردن، کپی کردن، و یا جستجو در بین کد را انجام دهید.
در این فصل نحوه ایجاد منو در برنامه های ویژوال C# را بررسی خواهیم کرد. در طی فصل نحوه ی ایجاد و مدیریت منو ها و زیرمنو ها و همچنین نحوه ایجاد منو های فرعی در یک برنامه را مشاهده خواهید کرد. ویژوال C# ۲۰۰۵ دو نوع کنترل برای کار با منو ها در جعبه ابزار خود دارد که در این فصل هر دوی آنها را بررسی خواهیم کرد.
در این فصل:

- با نحوه ی ایجاد منو ها آشنا خواهید شد.
- با نحوه ی ایجاد زیر منو ها آشنا خواهید شد.
- چگونگی استفاده از منو های فرعی را مشاهده خواهید کرد.

درک ویژگیهای یک منو:

کنترلی که در ویژوال استودیو ۲۰۰۵ برای ایجاد منو در برنامه مورد استفاده قرار می گیرد، `MenuStrip` نام دارد. این کنترل دارای چندین ویژگی کلیدی است. اولین و مهمترین ویژگی این است که به وسیله آن می توانید به سرعت و به راحتی منو ها و زیرمنو های مورد نیاز در برنامه را ایجاد کنید.
منو هایی که به وسیله این کنترل ایجاد می کنید علاوه بر متنی که برای هر منو نمایش می دهد، می توانند شامل تصویر، کلید دسترسی، شورت کات و یا حتی به صورت گزینه ی قابل انتخاب باشند.

تصاویر:

حتماً تاکنون تصاویر کنار منو ها را در برنامه های ویندوزی مانند `Word` و یا حتی خود ویژوال استودیو مشاهده کرده اید. تا قبل از ویژوال استودیو ۲۰۰۵ برنامه نویسان برای نمایش تصویر در کنار منو های خود یا باید حجم زیادی از کد را در برنامه وارد کرده و یا از کنترلهای شخص–ثالث[2] استفاده می کردند. اما در ویژوال استودیو ۲۰۰۵ برای هر گزینه ی منو، یک خاصیت `Image` نیز در نظر گرفته شده است که می توانید به وسیله آن تصویری را معین کنید تا در کنار منو نمایش داده شود.

[1] منوی فرعی یا `Pop-Up Menu` منویی که با کلیک راست در یک محدوده نمایش داده می شود.

[2] `Third-Party Controls`

کلیدهای دسترسی:

یک کلید دسترسی به کاربر اجازه می دهد که با فشار کلید `Alt` و آن کلید در صفحه کلید، به منو دسترسی پیدا کند. کلیدهای دسترسی برای هر منو به صورت یک حرف هستند که در نام منو با یک زیر خط مشخص می شوند. هنگامی که کلید `Alt` همراه با کلید دسترسی منو در صفحه کلید فشار داده شد، منوی مربوطه در صفحه نمایش داده می شود و کاربر می تواند به وسیله کلیدهای مکان نما بین گزینه های منو جا به جا شود.

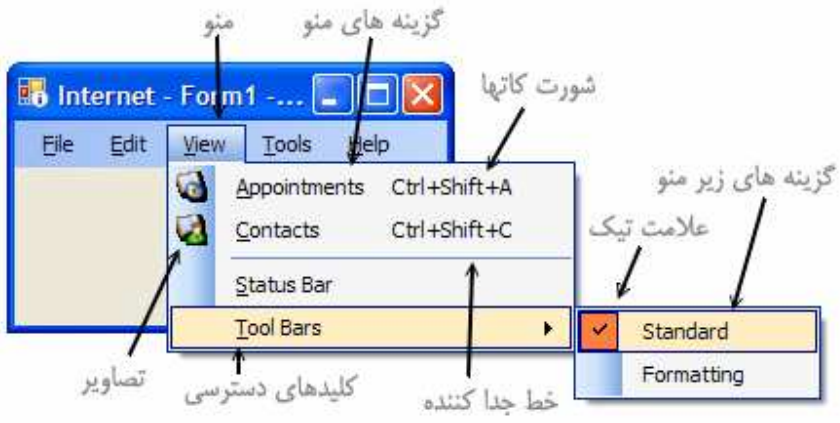
شورت کات ها:

به وسیله شورت کات ها می توانید بدون اینکه منوی مورد نظر را نمایش دهید، گزینه ای را از آن انتخاب کنید. شورت کات ها به صورت ترکیب یک کلید کنترلی همراه با یک کلید اصلی است، مانند شورت کات `Ctrl + X` برای کات کردن یک متن.

علامت تیک:

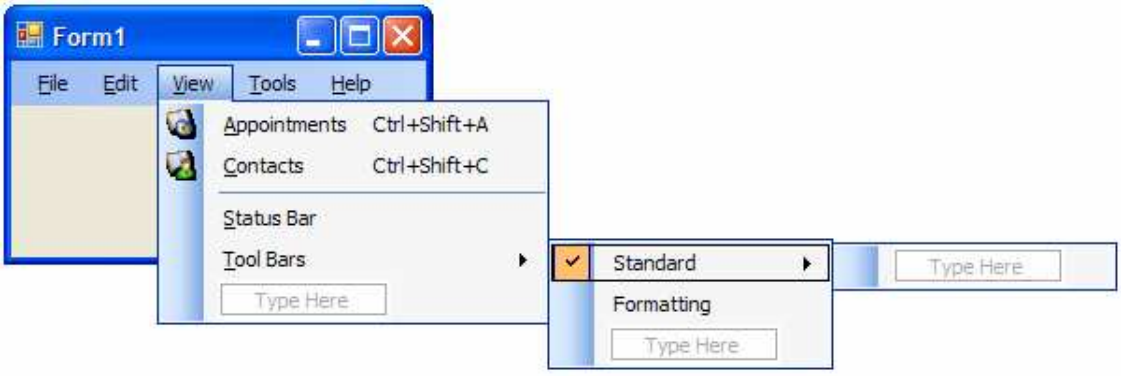
علامت تیک، علامتی است که در کنار بعضی از منو ها به جای عکس قرار می گیرد و مشخص می کند که این گزینه هم اکنون انتخاب شده و یا در حال استفاده است. برای مثال اگر منوی `View` را از ویژوال استودیو انتخاب کرده و سپس به زیر منوی `ToolBars` بروید، نام تمام نوار ابزارهای موجود در ویژوال استودیو را مشاهده خواهید کرد که کنار بعضی از آنها یک علامت تیک قرار گرفته است. وجود این علامت در کنار یک گزینه به این معنی است که نوار ابزار مربوط به آن گزینه هم اکنون در ویژوال استودیو نمایش داده شده است.

شکل ۸-۱ ویژگی هایی که یک منو می تواند داشته باشد را نمایش می دهد. همانطور که مشاهده می کنید، این منوی نمونه علاوه بر تمام ویژگیهایی که ذکر کردیم یک منوی جدا کننده نیز دارد. یک منوی جدا کننده به صورت یک خط افقی نمایش داده می شود و برای دسته بندی گزینه های مرتبط به هم در منو به کار می رود.



شکل ۸-۱

شکل ۸-۱ منوی برنامه را در زمان اجرای برنامه نمایش می دهد. این منو در زمان طراحی همانند شکل ۸-۲ خواهد بود.



شکل ۸-۲

در هنگام کار با کنترل MenuStrip اولین نکته ای که جلب توجه می کند، این است که علاوه بر منو های ایجاد شده در این کنترل، منویی هم وجود دارد که برای اضافه کردن یک گزینه ی جدید به منو و یا زیر منو به کار می رود. هر زمان که با استفاده از این گزینه در کنترل MenuStrip یک منوی جدید ایجاد کنید، گزینه دیگری مشابه قبلی در این قسمت ایجاد خواهد شد.

پنجره Properties:

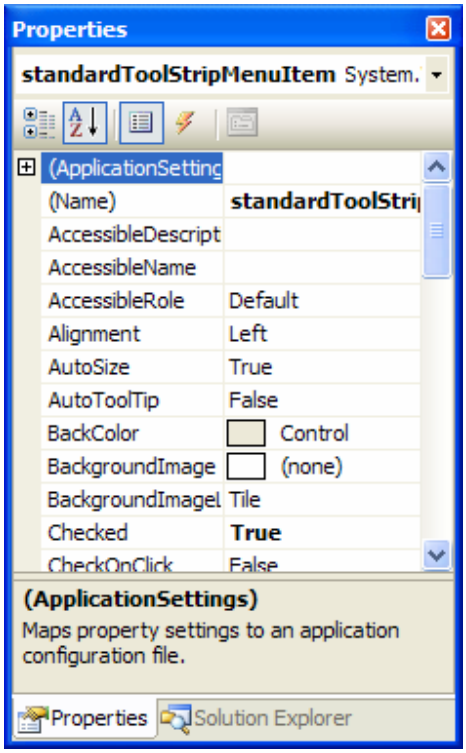
هنگامی که یک منو را به برنامه اضافه کردید و در حال تغییر ویژگی های آن بودید، پنجره Proiperties خاصیتهای آن منو را نمایش می دهد و مانند هر کنترل دیگری می توانید خاصیتهای آن را تنظیم کنید. شکل ۸-۳ پنجره Properties را برای یک کنترل منو نمایش می دهد.

در طراحی منو برای برنامه همواره باید در نظر داشته باشید که منوی برنامه خود را همانند منوی دیگر برنامه های ویندوزی ایجاد کنید. برای مثال می توانید به منو های برنامه هایی مانند ویژوال استودیو، Word و یا Outlook توجه کنید. در تمام این برنامه ها منویی به نام File وجود دارد و گزینه ای به نام Exit در این منو است که به وسیله آن می توان از برنامه خارج شد. بنابراین اگر در برنامه خود از منو استفاده کردید، بهتر است منویی به نام فایل ایجاد کرده و حداقل گزینه Exit را در آن قرار دهید. اگر برنامه شما به کاربر اجازه می دهد عملیاتی مانند کات کردن و یا کپی کردن را انجام دهد، بهتر است منویی به نام Edit ایجاد کرده و این گزینه ها را در آن قرار دهید.

نکته: در کتابخانه MSDN که همراه با ویژوال استودیو ۲۰۰۵ نصب می شود، بخشی به نام User Interface Design and Development وجود دارد که مسائل بسیاری را راجع به طراحی رابط کاربری استاندارد برای برنامه های ویندوزی بررسی می کند. برای اطلاع در مورد چگونگی طراحی استاندارد رابطهای کاربری می توانید به آن بخش مراجعه کنید.

دلیل اینکه منو های برنامه شما باید همانند منو های دیگر برنامه ویندوزی باشد در این است که به این صورت کاربران هنگام کار با برنامه احساس راحتی بیشتری می کنند و می توانند از آشنایی قبلی خود با منو های دیگر برنامه ها، در برنامه شما نیز استفاده کنند و نیازی ندارند که عملکرد تمامی منو ها را از ابتدا یاد بگیرند. البته همواره در منوی یک برنامه گزینه هایی وجود دارند که در برنامه

های دیگر نیستند. در مورد این گزینه ها میتوانید آنها را در چند منوی خاص در بین منو ها قرار دهید. امـا همـواره قـرار دادن گزینـه های عمومی در قسمتهای ثابت یک منو باعث می شود که کاربر زودتر با نحوه کارکرد برنامه آشنا شود.



شکل ۸-۳

ایجاد منو ها:

حال بهتر است که ببینیم چگونه می توان در یک برنامه منو ایجاد کرد. در بخش امتحان کنید بعد، برنامه ای ایجاد خواهیم کـرد کـه شامل یک نوار منو، دو نوار ابزار و دو `TextBox` باشد. نوار منـو شـامل منـو هـای `File`، `Edit`، `View`، `Tools` و `Help` و چندین گزینه و زیر منو در هر یک از آنها خواهد بود. به این ترتیب می توانید تمـام ویژگـی هـای منـو هـا را در برنامـه مشاهده کنید. به علت اینکه این برنامه از چند بخش تشکیل شده است، نوشتن آن را نیز به چند قسمت تقسیم می کنیم و در قسمت اول به طراحی منو ها می پردازیم.

طراحی منو ها:

امتحان کنید: ایجاد منو ها

۱) ویژوال استودیو ۲۰۰۵ را اجرا کرده و گزینه File ➔ New ➔ Project... را از نوار منو انتخاب کنید. در کادر New Project گزینه Windows Application را از قسمت Templates انتخاب کرده و در کادر Name گزینه Menus را انتخاب کنید. روی دکمه OK کلیک کنید تا پروژه جدید ایجاد شود.

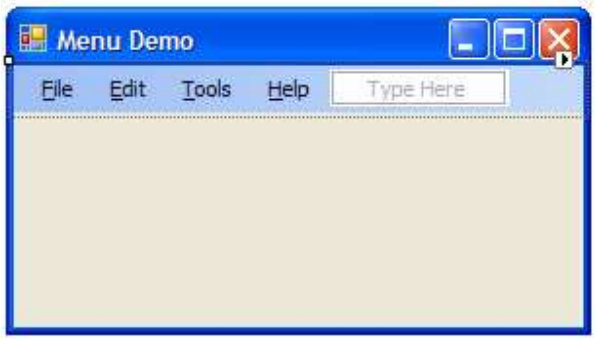
۲) در قسمت طراحی فرم، روی فرم برنامه کلیک کرده و خاصیتهای آن را مطابق لیست زیر تنظیم کنید:

- خاصیت Size را برابر با 300;168 قرار دهید.
- خاصیت StartPosition را برابر با CenterScreen قرار دهید.
- خاصیت Text را برابر با Menu Demo قرار دهید.

۳) با استفاده از جعبه ابزار، یک کنترل MenuStrip را در فرم برنامه قرار دهید. این کنترل به طور اتوماتیک در بالای فرم قرار خواهد گرفت. علاوه بر کنترلی که در بالای فرم قرار داده می شود، کنترلی هم همانند کنترلهای کادر محاوره ای که در فصل هفتم مشاهده کردید، به قسمت پایین بخش طراحی فرم اضافه می شود.

۴) در قسمت پایین طراحی روی کنترل menuStrip1 کلیک راست کنید و از منوی فرعی باز شده گزینه Insert Standard Items را انتخاب کنید تا گزینه های یک منوی استاندارد در نوار منوی بالای فرم قرار بگیرد.

۵) همانطور که در شکل ۸-۴ مشاهده می کنید، یک قسمت سفید رنگ در سمت راست منوی Help وجود دارد که به وسیله آن می توانید منو های جدید را ایجاد کنید. برای ایجاد منوی جدید علاوه بر استفاده از این قسمت، می توانید از پنجره Items Collection Editor نیز استفاده کنید.



شکل ۸-۴

نوار منوی بالای صفحه را انتخاب کرده و سپس در پنجره Properties روی دکمه ... در مقابل گزینه Items کلیک کنید. به این ترتیب کادر Items Collection Editor نمایش داده می شود. در این کادر روی دکمه فرمان Add کلیک کنید تا یک منوی جدید به نوار منو اضافه شود.

برای اینکه نام منوی جدید با نام منو هایی که در قسمت قبلی به نوار منو اضافه کردیم هماهنگ باشد، نام آن را به viewToolStripMenuItem تغییر دهید.

حال خاصیت Text منوی جدید را به &View تغییر دهید. کاراکتر & برای مشخص کردن کلید دسترسی به یک منو به کار می رود. این کاراکتر قبل از هر حرفی که قرار بگیرد، می توان با ترکیب آن حرف با کلید Alt به آن منو دسترسی داشت. در منویی که ایجاد کردیم کاراکتر & قبل از حرف V قرار گرفته است، بنابراین در طول اجرای برنامه کاربر برای دسترسی به این منو می تواند از ترکیب کلیدهای Alt + V استفاده کند.

حال باید منوی جدید را بین منو های Edit و Tools قرار دهیم. برای این کار منوی View را در لیست منو ها انتخاب کرده و روی فلش به سمت بالا در سمت راست لیست کلیک کنید تا منوی View در مکان درست خود قرار بگیرد.

۶) حال از بین خاصیتهای منوی View خاصیت DropDownItems را انتخاب کرده و روی دکمه ی ... در مقابل آن کلیک کنید. پنجره دیگری مشابه پنجره Items Collection Editor باز می شود که به وسیله آن می توانید، منو هایی که می خواهید تحت منوی View قرار گیرند را ایجاد کنید.
در این برنامه فقط یک منو تحت منوی View نمایش داده می شود و آن هم منوی ToolBars خواهد بود. بنابراین در پنجره Items Collection Editor دوم، روی دکمه Add کلیک کنید تا منوی دیگری اضافه شود.
مجددا برای سازگاری نام منوی جدید با دیگر منو ها، نام آن را به toolbarToolStripMenuItem تغییر دهید. همچنین خاصیت Text این منو را نیز برابر با &Toolbars قرار دهید.

۷) در این مرحله باید دو زیر منو به منوی Toolbars اضافه کنیم. بنابراین خاصیت DropDownItems را در بین خاصیتهای منوی Toolbars انتخاب کرده و روی دکمه ... مقابل آن کلیک کنید.
پنجره Items Collection Editor دیگری باز خواهد شد که به وسیله آن می توانید زیر منو های مربوط به منوی Toolbars را ایجاد کنید. در این پنجره روی دکمه Add کلیک کنید تا یک منوی جدید ایجاد شود. نام این منو را برابر با mainToolStripMenuItem و خاصیت Text آن را برابر با &Main قرار دهید.
هنگامی که برنامه اجرا می شود، نوار ابزار Main به صورت پیش فرض نمایش داده می شود. بنابراین کنار این منو باید علامتی قرار بگیرد تا مشخص کند که این نوار ابزار در حال حاضر نمایش داده شده است. خاصیت Checked این منو را برابر با True قرار دهید تا به صورت پیش فرض یک علامت تیک در کنار آن قرار بگیرد. همچنین خاصیت CheckOnClick این منو را نیز برابر با True قرار دهید تا با کلیک کاربر روی این منو، علامت تیک کنار آن برداشته شود و یا قرار داده شود.

۸) زیر منوی دیگری که باید در این قسمت اضافه کنیم، زیر منوی Formatting است. روی دکمه Add کلیک کنید تا زیر منوی جدیدی به این قسمت اضافه شود. سپس خاصیت Name آن را به formattingToolStripMenuItem و خاصیت Text آن را برابر با &Formatting قرار دهید.
به علت اینکه این زیر منو به صورت پیش فرض نمایش داده نمی شود، نیازی نیست که خاصیت Checked آن را برابر با True قرار دهید. اما باید خاصیت CheckedOnClick را برابر با True قرار دهید تا در صورتی که کاربر روی این گزینه کلیک کرد، علامت تیک در کنار آن نمایش داده شده و یا از کنار آن برداشته شود.
روی دکمه OK در تمام کادرهای Items Collection Editor کلیک کنید تا تمام آنها بسته شوند.

۹) حال اگر برنامه را اجرا کنید و در منوی View روی گزینه Toolbars کلیک کنید، زیر منویی همانند شکل ۸-۵ نمایش داده می شود.

چگونه کار می کند؟

ویژوال استودیو ۲۰۰۵ تمام سعی خود را کرده است تا برنامه نویس را از انجام کارهای تکراری و خسته کننده رها کند. یکی از این موارد ایجاد منو های استانداردی است که در همه برنامه ها یافت می شود. همانطور که مشاهده کردید با استفاده از گزینه

Insert Standard Items منو هایی که در بیشتر برنامه ها مورد استفاده قرار می گیرد با جزئیات کامل به برنامه اضافه شد. به این صورت می توانید تمرکز بیشتری روی گزینه های مخصوص برنامه خود داشته باشید. در این برنامه به جز منو های استاندارد، فقط کافی است منوی View را به برنامه اضافه کنیم که این منو خود نیز شامل منوی Toolbars و زیرمنو هـای Main و Formatting است.



شکل ۸–۵

اضافه کردن نوار ابزارها و کنترل ها:

حال که منو های مورد نیاز در برنامه را ایجاد کردیم، به سراغ نوار ابزارها و دکمه های آن می رویم. منو هـایی کـه در بخـش قبلـی ایجاد کردیم، نمایش داده شدن و یا نشدن این نوار ابزارها را کنترل خواهند کرد. همچنین دو TextBox نیـز بـه برنامـه اضـافه خواهیم کرد تا به وسیله دکمه های موجود در نوار ابزار و یا گزینه های منو ها بتوانیم متن داخل یکی از آن را کات و یا کپی کـرده و در دیگری قرار دهیم.

امتحان کنید: اضافه کردن نوار ابزارها و کنترل ها

۱) در این برنامه به دو نوار ابزار نیاز دارید. ابتدا با استفاده از جعبه ابزار یک کنترل ToolStrip بر روی فرم قرار دهید. مشاهده خواهید کرد که این کنترل به طور اتوماتیک در قسمت بالای فرم قرار می گیرد. خاصیت Name این نوار ابـزار را به tspFormatting تغییر دهید. همچنین به علت اینکه نمی خواهیم این نوار ابزار به طـور پـیش فـرض در فرم نمایش داده شود، خاصیت Visible آن را برابر false قرار دهید.

۲) در این نوار ابزار به سه دکمه نیاز داریم. بنابراین در بین خصوصیات آن، گزینه Items را انتخاب کرده و روی دکمه ی ... مقابل آن کلیک کنید.
در پنجـره Items Collection Editor روی دکمـه ی Add کلیـک کنیـد تـا یـک کنتـرل Button به این نوار ابزار اضافه شود. می توانید خاصیت نام و دیگر خاصیتهای این کنترل را تغییر ندهید، زیـرا نمـی خواهید از آن در کد استفاده کنید. تنها خاصیت DisplayStyle آن را برابـر بـا Image قـرار داده و سـپس روی دکمه ... در مقابل خاصیت Image کلیک کنید.
در پنجره Select Resource روی دکمه Imports کلیک کنید و به آدرس زیر بروید:

```
C:\Program Files\Microsoft Visual Studio 8\Common7\
```

VS2005ImageLibrary\Bitmaps\Commands\highColor\

در این فولدر، فایل AlignTableCellMiddleLeftJustHS.bmp را انتخاب کرده و روی دکمه Open کلیک کنید. سپس در پنجره Select Resource روی OK کلیک کنید تا بسته شود.

۳) در پنجره Items Collection Editor بار دیگر روی دکمه Add کلیک کنید تا کنترل Button دیگری به نوار ابزار اضافه شود. خاصیت DisplayStyle آن را برابر با Image قرار داده و سپس فایل AlignTableCellMiddleCenterHS.bmp از آدرس قبلی را برای خاصیت Image آن مشخص کنید.

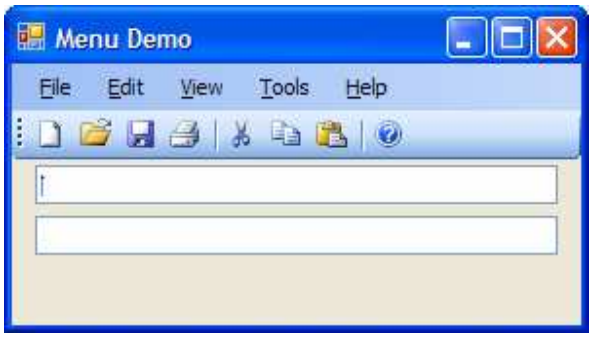
۴) کنترل Button دیگری به نوار ابزار اضافه کرده، خاصیت DisplayStyle آن را برابر با Image قرار دهید. برای این دکمه، فایل AlignTableCellMiddleRightHS.bmp را به عنوان تصویر مشخص کنید.

۵) در پنجره Items Collection Editor روی دکمه OK کلیک کنید تا بسته شود.

۶) حال با استفاده از جعبه ابزار، نوار ابزار دوم را به فرم اضافه کرده و خاصیت Name آن را برابر با tspMain قرار دهید. این نوار ابزار نیز همانند نوار ابزار قبلی در بالای فرم قرار می گیرد. در این برنامه می خواهیم این نوار ابزار شامل دکمه های استاندار باشد، بنابراین روی کنترل tspMain در پایین قسمت طراحی فرم کلیک راست کرده و از منوی فرعی که باز می شود گزینه Insert Standard Items را انتخاب کنید.

۷) یک کنترل Panel از جعبه ابزار روی فرم قرار داده و خاصیت Dock آن را برابر با Fill قرار دهید تا تمام فرم را دربر گیرد.

۸) دو کنترل TextBox روی فرم قرار دهید و مقدار پیش فرض خاصیتهای آن را تغیری ندهید. مکان و اندازه این کنترل ها مهم نیستند، اما باید به اندازه کافی بزرگ باشند تا بتوانید متنی را در آن بنویسید. فرم کامل شده برنامه باید مشابه شکل ۸-۶ باشد. توجه کنید که در این فرم نوار ابزار دوم مشاهده نمی شود، زیرا خاصیت Visible آن برابر با false قرار داده شده است.



شکل ۸-۶

چگونه کار می کند؟

با چگونگی استفاده از کنترل ToolStrip در فصل ششم آشنا شدید، برای مطالعه ی مجدد آن می توانید برنامه ی Text Editor را که در آن فصل تمرین کردیم مشاهده کنید. کنترل ToolStrip نیز همانند کنترل MenuStrip دارای گزینه Insert Standard Items است که با قرار دادن دکمه های عمومی در نوار ابزار، حجم زیادی از کارهای تکراری را در برنامه کم می کند. بدون شک استفاده از این گزینه بهترین راه برای اضافه کردن دکمه های استاندارد به یک

نوار ابـزار اسـت. البتـه بعـد از اینکـه ایـن دکمـه هـا در نـوار ابـزار قـرار داده شـد، مـی توانیـد بـا اسـتفاده از پنجـره Items Collection Editor مکان آنها را تغییر داده و یا دکمه های جدیدی به آن اضافه کنید.
همانطور که در فرم برنامه مشاهده می کنید، نوار ابزار Formatting دیده نمی شود زیرا خاصیت Visible آن برابر بـا False قرار داده شده است. برای مشاهده این نوار ابزار باید روی کنترل مربوط به آن در قسمت پایین طراحی فرم کلیک کنید.

نوشتن کد برای منو ها:

حال که تمام کنترلهای مورد نیاز را در فرم قرار دادیم، باید کد نویسی آنها را شروع کنیم. ابتدا کد مربوط به عملکـرد منـو هـا را مـی نویسیم. بعد از اتمام آن کد مربوط به بعضی از دکمه های موجود در نوار ابزار Main را کامل می کنیم.

امتحان کنید: نوشتن کد منوی File

۱) به قسمت طراحی فرم رفته و با استفاده از نوار منو، گزینه New را از منـوی File انتخـاب کنیـد. سـپس بـه پنجـره Properties بروید و روی آیکون Events کلیک کنید تا رویدادهای این گزینه از منو نمایش داده شوند. از لیست رویدادها، رویداد کلیک را انتخاب کرده و روی آن دو بار کلیک کنید تا متد مربوط بـه آن ایجـاد شـود. سـپس کـد مشخص شده در زیر را به این متد اضافه کنید:

```
private void newToolStripMenuItem_Click(object sender,
                                        EventArgs e)
{
    // Clear the text boxes
    textBox1.Text = string.Empty;
    textBox2.Text = string.Empty;

    // Set focus to the first text box
    textBox1.Focus();
}
```

۲) عملکرد دکمه New در نوار ابزار Main نیز مشابه عملکرد این گزینه از نوار منو است. برای نوشتن کد مربوط بـه ایـن دکمه، به قسمت طراحی فرم رفته و دکمه New را از نوار ابزار انتخاب کنید. سپس از لیست رویـدادهای ایـن کنتـرل در پنجره Properties رویداد کلیک را انتخاب کرده و روی آن دو بار کلیک کنید تا متد مربوط به آن ایجـاد شـود. به علت مشابه بودن عملکرد این متد با متد مربوط به رویداد کلیک گزینه New در منوی File، در این قسمت کافی است کد زیر را به متد اضافه کنید:

```
private void newToolStripButton_Click(object sender,
                                      EventArgs e)
{
    // Call the newToolStripMenuItem_Click method
    newToolStripButton_Click(sender, e);
```

```
}
```

۳) همانند قسمت یک، متد مربوط به رویداد کلیک گزینه exitToolStripmenuItem در منـوی File را ایجاد کرده، سپس کد زیر را به آن اضافه کنید:

```
private void exitToolStripMenuItem_Click(object sender,
                                         EventArgs e)
{
    // Colse the form and exit
    Application.Exit();
}
```

چگونه کار می کند؟

کد مربوط به رویداد کلیک گزینه New کاملاً واضح است. در این قسمت باید برنامه را به حالت اولیه برگردانیم. برای ایـن کـار نیـز کافی است که متن داخل TextBox ها را پاک کرده و فوکوس را به TextBox اول انتقال دهیم تا اگر کاربر متنـی را وارد کرد، در این قسمت نوشته شود. برای پاک کردن متن داخل TextBox ها نیز کـافی اسـت خاصـیت Text آنهـا را برابـر بـا خاصیت Empty از کلاس String قرار دهیم.

```
// Clear the text boxes
textBox1.Text = string.Empty;
textBox2.Text = string.Empty;

// Set focus to the first text box
textBox1.Focus();
```

دکمه ی New در نوار ابزار نیز باید همین عمل را انجام دهد، اما نیازی نیست که این کد را برای این کنترل نیز تکرار کنیم. یکی از کارهایی که در این قسمت می توانید انجام دهید این است که متـدی مجـزا بـرای ایـن مـورد بنویـسید و ایـن متـد را در متـدهای newToolStripButton_Click و newToolStripMenuItem_Click فراخوانی کنید. اما روش بهتری هم وجود دارد و این است که کد مربوط به این قسمت را در یکی از این زیربرنامه ها پیاده سازی کرد و در متد دیگر آن زیربرنامه را فراخوانی کرد. در اینجا کد مربوط به این کـار را در متـد newToolStripMenuItem_Click قـرار دادیم و در زیربرنامه ی newToolStripButton_Click آن را فراخوانی کردیم. به علت اینکـه پارامترهـای هـر دوی این متدها یکسان است، برای فراخوانی متد اول، می توانیم از پارامترهای متد دوم استفاده کنیم.

```
// Call the newToolStripMenuItem_Click method
newToolStripButton_Click(sender, e);
```

حال چه از دکمه New در نوار ابزار استفاده کنید و چه گزینه New از منوی فایل را انتخاب کنید، نتیجه مشابه ای دریافت خواهیـد کرد.

در گزینه ی Exit از منوی File باید کدی را قرار دهید تا برنامه را تمام کند. برای این کار می توانید از متد Exit در کلاس Application استفاده کنید. این متد فرم برنامه را می بندد، تمام منابع اشغال شده به وسیله آن را آزاد کرده و به اجرای برنامه خاتمه می دهد.

```
// Colse the form and exit
Application.Exit();
```

حال که بخش کد مربوط به منوی File و دکمه های متناظر آن در نوار ابزار به پایان رسید، به منوی Edit می رویم تا کد مربوط به گزینه های آن را بنویسیم.

امتحان کنید: نوشتن کد منو Edit

۱) اولین گزینه در منوی Edit، گزینه Undo است. برای نوشتن کد مربوط به این گزینه به قسمت طراحی فرم بروید و روی گزینه Undo در منوی Edit دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس کد زیر را در این متد وارد کنید:

```
private void undoToolStripMenuItem_Click(object sender,
                                         EventArgs e)
{
    // Declare a TextBox object and
    // set it to the ActiveControl
    TextBox objTextBox = (TextBox)this.ActiveControl;
    // Undo the last operation
    objTextBox.Undo();
}
```

۲) حال باید کد مربوط به گزینه Cut از این منو را بنویسیم. برای این کار در قسمت طراحی فرم گزینه Cut را از منوی Edit انتخاب کرده و روی آن دو بار کلیک کنید. به این ترتیب متد مربوط به رویداد کلیک آن ایجاد می شود. کد زیر را در این متد وارد کنید:

```
private void cutToolStripMenuItem_Click(object sender,
                                        EventArgs e)
{
    // Declare a TextBox object and
    // set it to the ActiveControl
    TextBox objTextBox = (TextBox)this.ActiveControl;
    // Copy the text to the clipboard and clear the field
    objTextBox.Cut();
}
```

۳) عملکرد دکمه Cut در نوار ابزار نیز مشابه عملکرد این گزینه در نوار منو است. بنابراین با استفاده از نوار ابزار در قسمت طراحی فرم، دکمه ی Cut را انتخاب کرده و روی آن دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس در این متد، زیر برنامه cutToolStripMenuItem_Click را فراخوانی کنید.

```
private void cutToolStripButton_Click(object sender,
                                      EventArgs e)
{
    // Call the cutToolStripMenuItem_Click procedure
    cutToolStripMenuItem_Click(sender, e);
}
```

۴) گزینه بعدی در منوی Edit، گزینه Copy است. کد این گزینه نیز به نسبت مشابه گزینه Cut است. با دو بار کلیک روی این گزینه در قسمت طراحی فرم، متد مربوط به رویداد کلیک آن را ایجاد کرده و کد زیر را در آن وارد کنید:

```
private void copyToolStripMenuItem_Click(object sender,
                                         EventArgs e)
{
     // Declare a TextBox object and
     // set it to the ActiveControl
     TextBox objTextBox = (TextBox)this.ActiveControl;
     // Copy the text to the clipboard
     objTextBox.Copy();
}
```

۵) متد مربوط به رویداد کلیک دکمه ی Copy در نوار ابزار را ایجاد کرده و در آن متد قبلی را فراخوانی کنید. برای این کار کافی است کد زیر را به این متد اضافه کنید:

```
private void copyToolStripButton_Click(object sender,
                                       EventArgs e)
{
     // Call the copyToolStripMenuItem_Click procedure
     copyToolStripMenuItem_Click(sender, e);
}
```

۶) حال به قسمت طراحی فرم برگردید و از منوی Edit گزینه ی Paste را انتخاب کرده، روی آن دو بار کلیک کنید. در متد مربوط به رویداد کلیک آن که به طور اتوماتیک ایجاد می شود، کد زیر را وارد کنید:

```
private void pasteToolStripMenuItem_Click(object sender,
                                          EventArgs e)
{
     // Declare a TextBox object and
     // set it to the ActiveControl
     TextBox objTextBox = (TextBox)this.ActiveControl;
```

```
        // Copy the data from the clipboard to the textbox
        objTextBox.Paste();
}
```

۷) برای کد مربوط به رویداد کلیک دکمه ی Paste در نوار ابزار نیز می توان از فراخوانی متد قبلی استفاده کرد. بنابراین با دو بار کلیک روی دکمه Paste در نوار ابزار، متد مربوط به رویداد کلیک آن را ایجاد کرده و کد زیر را در آن وارد کنید:

```
private void pasteToolStripButton_Click(object sender,
                                        EventArgs e)
{
        // Call the pasteToolStripButton_Click procedure
        pasteToolStripButton_Click(sender, e);
}
```

۸) آخرین گزینه در منوی Edit گزینه ی Select All است. برای نوشتن کد مربوط به این گزینه با دو بار کلیک روی آن، متد مربوط به رویداد کلیک آن را ایجاد کرده و کد زیر را در آن وارد کنید:

```
private void selectAllToolStripMenuItem_Click(object
sender,                                          EventArgs
e)
{
        // Declare a TextBox object and
        // set it to the ActiveControl
        TextBox objTextBox = (TextBox)this.ActiveControl;
        // Select all text
        objTextBox.SelectAll();
}
```

چگونه کار می کند؟

نوشتن کد منوی Edit را با گزینه Undo شروع کردیم. به علت اینکه در فرم خود دو TextBox داریم، یا باید به نحوی بفهمیم که کدام TextBox در فرم انتخاب شده است، یا از یک روش کلی استفاده کنیم که برای هر دو TextBox کاربرد داشته باشد. در این متد از روشی کلی استفاده کرده ایم که می تواند برای هر دو TextBox به کار رود.
برای اینکه به کنترلی که هم اکنون در فرم انتخاب شده است دسترسی داشته باشیم، می توانیم از خاصیت ActiveControl در فرم استفاده کنیم (همانطور که در قسمتهای قبل نیز ذکر شد برای دسترسی به خاصیتهای فرم از کلمه کلیدی this استفاده می کنیم). این خاصیت، شیئ ای را از نوع Control برمی گرداند و باید قبل از استفاده از آن در برنامه آن را به نوع کنترلی که مورد نظرمان است تبدیل کنیم. با توجه به اینکه در این فرم فقط دو کنترل وجود دارند که می توانند به عنوان کنترل فعال باشند و هر دوی آنها نیز از نوع TextBox هستند، پس کنترلی که به وسیله خاصیت ActiveControl مشخص می شود حتماً می تواند به یک شیئ از کلاس TextBox تبدیل شود. بنابراین شیئ

جدیدی از نوع TextBox تعریف می کنیم و کنترلی را که خاصیت ActiveControl برمی گرداند، بعد از تبدیل نوع، در آن قرار می دهیم:

```
// Declare a TextBox object and
// set it to the ActiveControl
TextBox objTextBox = (TextBox)this.ActiveControl;
```

حال که به TextBox که در فرم انتخاب شده است دسترسی داریم، برای لغو آخرین عمل آن کافی است متد Undo را در آن فراخوانی کنیم. این متد که عضوی از کلاس TextBox است، می تواند آخرین تغییری را که کاربر در آن TextBox ایجاد کرده است را لغو کند.

```
// Undo the last operation
objTextBox.Undo();
```

نکته: در این برنامه استفاده از خاصیت ActiveControl بسیار راحت بود، زیرا فقط دو کنترل TextBox در فرم قرار داشت. پس می توانستیم بدون نگرانی از به وجود آمدن خطا در برنامه، مقدار برگشتی از این خاصیت را به شیئ ای از کلاس TextBox تبدیل کنیم. اما در شرایطی که کنترل های گوناگونی در برنامه وجود دارند، ممکن است کاربر یک کنترل Button را انتخاب کرده و سپس روی گزینه ی Undo کلیک کند. در این صورت مقدار برگشتی توسط خاصیت ActiveControl از نوع TextBox نخواهد بود و تبدیل آن به TextBox نیز باعث ایجاد خطا در برنامه خواهد شد. در این شرایط برای بدست آوردن کنترل فعال در یک فرم باید از روشهای دیگری استفاده کنیم که در فصلهای بعدی با آنها بیشتر آشنا می شوید.

گزینه بعدی در منوی Edit، گزینه Cut است. در این گزینه نیز، برای کات کردن متن، ابتدا باید به کنترل فعال در فرم دسترسی داشته باشیم. به همین دلیل از روش قبلی برای دسترسی به این کنترل استفاده می کنیم. بعد از مشخص کردن کنترل فعال در فرم و تبدیل آن به شیئ از کلاس TextBox، کافی است متدCut مربوط به آن را فراخوانی کنیم. این متد یکی از عضو های کلاس TextBox است که متن داخل کادر را از آن پاک کرده و در داخل کلیپ برد قرار می دهد:

```
// Declare a TextBox object and
// set it to the ActiveControl
TextBox objTextBox = (TextBox)this.ActiveControl;
// Copy the text to the clipboard and clear the field
objTextBox.Cut();
```

عملکرد دکمه ی Cut در نوار ابزار نیز مشابه این گزینه در نوار منو است. بنابراین کافی است در متد مربوط به رویداد کلیک از این کنترل، متدی را که برای گزینه Cut در نوار منو نوشتیم فراخوانی کنیم:

```
private void cutToolStripButton_Click(object sender,
                                      EventArgs e)
{
    // Call the cutToolStripMenuItem_Click procedure
    cutToolStripMenuItem_Click(sender, e);
```

```
}
```

کد مربوط به گزینه Copy نیز مشابه گزینه Cut است. کافی است کنترل فعال در فرم را مشخص کرده و بعد از تبدیل آن به شیئ ای از نوع TextBox، متد Copy آن را فراخوانی کنیم. متد Copy که عضو کلاس TextBox است، یک کپی از متن داخل کادر را در کلیپ برد قرار می دهد.

```
// Declare a TextBox object and
// set it to the ActiveControl
TextBox objTextBox = (TextBox)this.ActiveControl;
// Copy the text to the clipboard
objTextBox.Copy();
```

برای رویداد کلیک دکمه ی Copy در نوار ابزار هم کافی است که متد قبلی را فراخوانی کنیم:

```
private void copyToolStripButton_Click(object sender,
                                       EventArgs e)
{
    // Call the copyToolStripMenuItem_Click procedure
    copyToolStripMenuItem_Click(sender, e);
}
```

برای کد مربوط به رویداد کلیک گزینه Paste از منوی Edit و همچنین دکمه ی Paste در نوار ابزار هم می توانیم از روشی مشابه موارد قبلی استفاده کنیم.
برای گزینه Select All در این منو می توانیم متد SelectAll که عضوی از کلاس TextBox است را فراخوانی کنیم. این متد باعث می شود که تمام متن موجود در TextBox انتخاب شود:

```
// Declare a TextBox object and
// set it to the ActiveControl
TextBox objTextBox = (TextBox)this.ActiveControl;
// Select all text
objTextBox.SelectAll();
```

کد نویسی منوی View و نوار ابزارها:

بعد از اتمام منو های File و Edit و دکمه های نوار ابزار مرتبط با آنها، نوبت به منوی View می رسد تا کد مربوط به آن را در برنامه وارد کنیم.

امتحان کنید: نوشتن کد مربوط به منوی View

۱) در قسمت طراحی فرم به منوی View بروید و گزینه Main را از زیر منوی Toolbars انتخاب کنید. سپس با استفاده از پنجره Properties، روی رویداد کلیک در لیست رویدادهای این گزینه دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. کد مشخص شده در زیر را به این متد اضافه کنید:

```
private void mainToolStripMenuItem_Click(object sender,
                                                EventArgs e)
{
    // Toggle the visibility of the Main toolbar
    // based on this menu item's Checked property
    if (mainToolStripMenuItem.Checked)
    {
        tspMain.Visible = true;
    }
    else
    {
        tspMain.Visible = false;
    }
}
```

۲) برای گزینه Formatting از این زیرمنو نیز باید از کدی مشابه قسمت قبلی استفاده کنید. برای این کار با دو بار کلیک روی این گزینه در قسمت طراحی فرم، متد مربوط به رویداد کلیک آن را ایجاد کرده و کد زیر را در آن وارد کنید:

```
private void formattingToolStripMenuItem_Click(
                          object sender, EventArgs e)
{
     // Toggle the visibility of the Main toolbar
     // based on this menu item's Checked property
     spFormatting.Visible =
          formattingToolStripMenuItem.Checked;
}
```

چگونه کار می کند؟

هنگامی که گزینه Main از زیر منوی Toolbars انتخاب شود، بر اساس اینکه خاصیت Checked این گزینه برابر با True و یا False است، یک علامت تیک در کنار آن نمایش داده می شود و یا این علامت از کنار آن برداشته می شود. بنابراین می توانیم در رویداد کلیک این گزینه کدی را وارد کنیم که اگر این گزینه علامت خورده بود (خاصیت Checked آن برابر با true بود) نوار ابزار Main نمایش داده شود (خاصیت Visible آن برابر با True شود). در غیر این صورت این نوار ابزار نمایش داده نخواهد شد.

```
    // Toggle the visibility of the Main toolbar
    // based on this menu item's Checked property
    if (mainToolStripMenuItem.Checked)
```

```
{
    tspMain.Visible = true;
}
else
{
    tspMain.Visible = false;
}
```

همانطور که مشاهده می کنید در شرط دستور `if` نیازی نیست از عبارت

```
mainToolStripMenuItem.Check == true
```

استفاده کنیم. همانطور که ذکر شد، هنگامی که عبارت داخل پرانتز مقابل `if` برابر با `true` باشد، دستورات داخل بلاک `if` اجرا می شوند. بنابراین می توانیم به طور مستقیم از خود خاصیت در مقابل دستور `if` استفاده کنیم. اگر مقدار خاصیت برابر با `true` بود، دستورات داخل بلاک `if` اجرا می شود و در غیر این صورت دستورات داخل بلاک `else` اجرا می شوند.
برای گزینه `Formatting` نیز می توانیم از همان روش استفاده کنیم تا مقدار خاصیت `Visible` نوار ابزار `Formatting` بر اساس انتخاب شدن و یا نشدن این گزینه مشخص شود. اما روش بهتری نیز برای نوشتن این کد وجود دارد و این است که مقدار خاصیت `Checked` این گزینه از نوار منو را به طور مستقیم در خاصیت `Visible` نوار ابزار قرار دهیم. به این ترتیب نیازی به بررسی شرط با استفاده از دستور `if` نیست و همان کد قبلی را می توانیم در یک خط پیاده سازی کنیم. این روش نسبت به روش قبلی علاوه بر کوتاه تر بودن، از سرعت اجرای بالاتری نیز برخوردار است. البته ممکن است تفاوت سرعت این روش در برنامه های کوچکی مانند این مثال چندان مورد توجه واقع نشود، اما هنگامی که حجم برنامه زیاد شود استفاده از این تکنیک ها در برنامه باعث افزایش چشمگیر سرعت برنامه می شود.

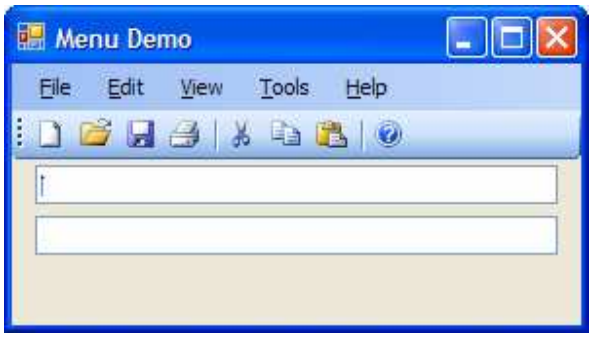
```
// Toggle the visibility of the Main toolbar
// based on this menu item's Checked property
spFormatting.Visible =
     formattingToolStripMenuItem.Checked;
```

امتحان برنامه:

هر چه برنامه ای که می نویسید پیچیده تر می شود، تست کردن مداوم برای اطمینان از نحوه عملکرد آن نیز مهمتر می شود. در یک برنامه هر چه بیشتر خطاهای آن را پیدا کرده و تصحیح کنید، بهتر می توانید آن را پیاده سازی کنید. در نتیجه می توانید برنامه ای پایدار تر و با قابلیت اطمینان بیشتری ایجاد کنید.
برای تست کردن یک برنامه، نه تنها باید عملکرد عادی آن را بررسی کنید، بلکه باید مقدارهای مختلفی که کاربر ممکن است در برنامه وارد کند را نیز به عنوان ورودی به برنامه بفرستید و رفتار برنامه را در آن شرایط بررسی کنید. برای مثال فرض کنید در حال طراحی یک برنامه بانک اطلاعاتی هستید که اطلاعات کاربر را به وسیله یک فرم دریافت می کند و در یک جدول در بانک اطلاعاتی قرار می دهد. یک برنامه خوب و پایدار، قبل از اینکه اطلاعات کاربر را برای ذخیره شدن به بانک اطلاعاتی بفرستد، صحت نوع داده ای تمام آنها را بررسی می کند. برای مثال قبل از اینکه سن کاربر را در جدول ذخیره کند، بررسی می کند که حتماً مقدار وارد شده به وسیله کاربر برای این قسمت به صورت عددی باشد و کاربر متنی را در این قسمت وارد نکرده باشد. به این ترتیب از ایجاد خطا در برنامه و توقف اجرا آن نیز می توان تا حد امکان جلوگیری کرد.

امتحان کنید: امتحان برنامه

۱) حال که رابط کاربری برنامه را طراحی کردیم و کد بیشتر قسمتهای آن را نیز وارد کردیم، باید عملکرد آن را بررسی کنیم. برای این کار روی دکمه Start در نوار ابزار کلیک کنید تا برنامه اجرا شود. هنگامی که فرم اصلی برنامه نمایش داده شد، مشابه شکل ۸-۷ فقط یکی از نوار ابزارها قابل مشاهده خواهد بود.



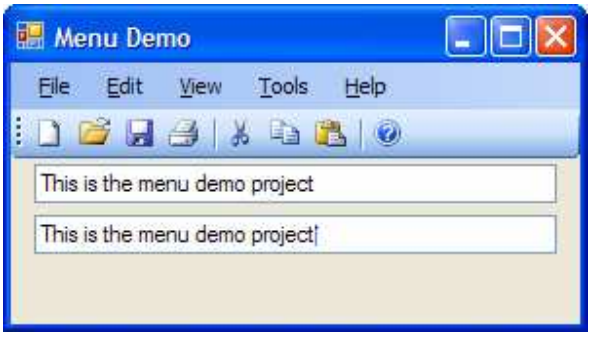
شکل ۸-۷

۲) از نوار منو، منوی View را انتخاب کرده و سپس به زیر منوی Toolbars بروید. مشاهده می کنید که در این زیرمنو دو گزینه Main و Formatting وجود دارند که در کنار گزینه اول یک علامت تیک قرار دارد. این علامت مشخص می کند که هم اکنون این نوار ابزار در فرم برنامه قابل مشاهده است. روی گزینه Formatting در این قسمت کلیک کنید تا نوار ابزار Formatting هم نمایش داده شود.
همانطور که مشاهده کردید با نمایش این نوار ابزار، تمام کنترل ها در فرم به اندازه لازم به سمت پایین حرکت کردند. دلیل این امر در این است که یک کنترل Panel در فرم قرار داده و بعد از تنظیم خاصیت Dock آن با مقدار Fill، دو TextBox را در آن قرار دادید. با انجام این کار، موقعیت کنترل ها می تواند در صورت لزوم در فرم تغییر کند. اگر کنترل Panel را از فرم حذف کنید و کنترلهای TextBox را در خود فرم قرار دهید، مشاهده خواهید کرد که با نمایش نوار ابزار Formatting، کنترل TextBox اول روی نوار ابزار Main قرار خواهد گرفت نوار ابزار Main قابل مشاهده نخواهد بود.

۳) حال اگر مجدداً به زیر منوی Toolbars در منوی View بروید، مشاهده خواهید کرد که در کنار هر دو گزینه های Main و Formatting یک علامت تیک قرار گرفته است که مشخص می کند در حال حاضر هر دوی این نوار ابزارها در حال نمایش است.

۴) حال عملکرد گزینه های منوی Edit را بررسی می کنیم. متنی را در کنترل TextBox اول وارد کرده و سپس گزینه Select All را از منوی Edit انتخاب کنید. مشاهده خواهید کرد که تمام متن نوشته شده در آن انتخاب خواهند شد.

۵) حال می خواهیم متن انتخاب شده در TextBox اول را در کلیپ برد کپی کنیم. برای این کار با ماوس روی دکمه ی Copy در نوار ابزار کلیک کنید و یا گزینه Copy را از منوی Edit انتخاب کنید.
مکان نما را در TextBox دوم قرار دهید و سپس روی دکمه ی Paste در نوار ابزار کلیک کنید و یا از نوار منو گزینه Edit ➔ Paste را انتخاب کنید. به این ترتیب متن نوشته شده در TextBox اول، همانند شکل ۸-۸ در TextBox دوم نیز قرار خواهد گرفت.



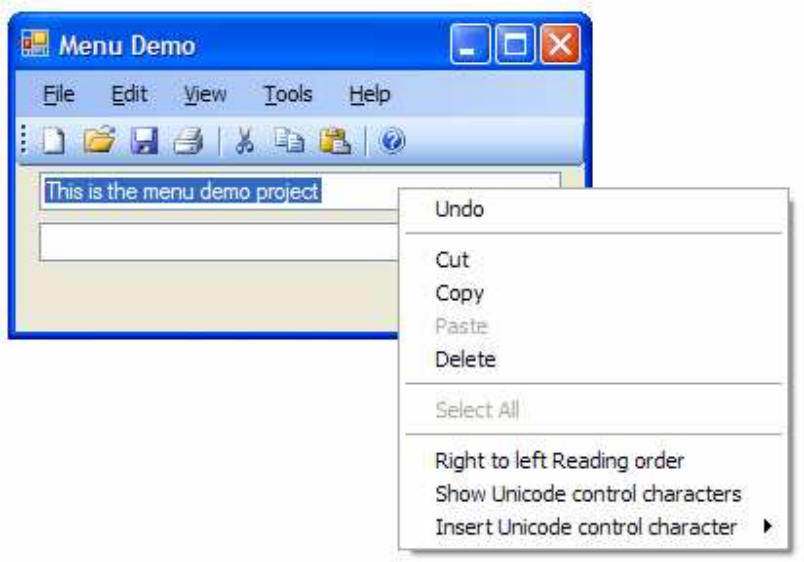
شکل ۸-۸

۶) کنترل TextBox اول را انتخاب کرده و گزینه Edit ➔ Undo را از نوار منو انتخاب کنید، مشاهده خواهید کرد که تغییراتی که در این کنترل ایجاد کرده بودید لغو می شود. ممکن است انتظار داشته باشید که با انتخاب این گزینه، آخرین تغییر، یعنی متنی که در TextBox دوم وارد کرده اید پاک شود. اما ویندوز تغییرات یک کنترل را برای لغو آنها به طور جداگانه ذخیره می کند. بنابراین اگر هنگامی که TextBox اول را انتخاب کرده اید روی گزینه Undo کلیک کنید، تغییرات مربوط به این کنترل لغو می شوند.

۷) آخرین گزینه ای که در منوی Edit باید مورد بررسی قرار بگیرد، گزینه Cut است. متنی را در TextBox اول وارد کرده و با انتخاب گزینه Edit ➔ Select All از نوار منو، تمام آن را انتخاب کنید. سپس با کلیک روی دکمه Cut در نوار ابزار و یا انتخاب گزینه Edit ➔ Cut از نوار منو، متن نوشته شده در این کنترل را از آن حذف کرده و در کلیپ برد قرار دهید. سپس مکان نما را در انتهای متن موجود در TextBox دوم برده و با فشار دادن شورت کات گزینه Paste (Ctrl+V)، متن داخل کلیپ برد را در این TextBox قرار دهید.
همانطور که مشاهده کردید با نوشتن مقدار بسیار کمی کد،گزینه های Cut، Copy و Paste را در این برنامه همانند دیگر برنامه های ویندوزی ایجاد کردیم.

۸) به منوی File بروید و گزینه New را انتخاب کنید. به این ترتیب متن داخل هر دو TextBox پاک خواهد شد، پس این گزینه نیز به درستی کار می کند. تنها گزینه باقی مانده برای بررسی کردن، گزینه Exit از منوی File است.

۹) قبل از خروج از برنامه، روی یکی از TextBox ها کلیک راست کنید. به این ترتیب یک منوی فرعی همانند شکل ۸-۹ نمایش داده می شود. توجه کنید که این منو به صورت اتوماتیک نمایش داده می شود و برای اضافه کردن آن به برنامه نیاز به نوشتن کد و یا استفاده از کنترل خاصی نیست. به عبارت دیگر این ویژگی توسط ویندوز به برنامه اضافه می شود. اما همانطور که در بخش بعد مشاهده خواهید کرد در ویژوال استودیو ۲۰۰۵ روشهایی برای حذف این منو و نمایش منوی موردنظر خودتان در این قسمت وجود دارد.



شکل ۸-۹

۱۰) برای بررسی عملکرد آخرین قسمت از برنامه، از نوار منو گزینه `File` ➔ `Exit` را انتخاب کنید تا از برنامه خارج شوید.

منو های فرعی:

منو های فرعی[1] منو هایی هستند که با کلیک راست کاربر روی یک کنترل و یا روی یک فرم نمایش داده می شوند. به وسیله این منو ها کاربر می تواند به سرعت به کارهای عمومی که در قسمتی از برنامه به شدت به آنها نیاز پیدا می کند دسترسی داشته باشد. برای مثال منوی فرعی که در برنامه قبلی به صورت اتوماتیک نمایش داده شد این امکان را به کاربر می دهد که به راحتی و به سرعت، به گزینه های پر کاربرد منوی `Edit` برای ویرایش متن درون یک `TextBox`، دسترسی داشته باشد.

منو های فرعی می تواند بر حسب کنترلی که کاربر انتخاب کرده است تغییر کنند. برای مثال همانطور که در دیگر برنامه های ویندوزی مشاهده کرده اید، اگر در فرم یک برنامه یک کنترل `TextBox` را انتخاب کرده و روی آن کلیک راست کنید منوی فرعی نمایش داده می شود که با منوی فرعی مربوط به خود فرم تفاوت دارد.

ویندوز به صورت پیش فرض یک منوی فرعی برای کنترلهای `TextBox` نمایش می دهد و تا کاربر بتواند به وسیله آن کارهای عمومی مانند `Cut`، `Copy` و یا `Paste` را انجام دهد. البته در صورت لزوم می توانید این منو را با هر منوی دیگری جایگزین کنید. برای مثال تصور کنید کاربر می تواند در برنامه شما متن داخل `TextBox` ها را کپی کند، اما نمی خواهید اجازه دهید که این متن کات شود. بنابراین می توانید منوی جدیدی ایجاد کنید و از آن به عنوان منوی فرعی در کنترل های `TextBox` استفاده کنید. سپس در این منو گزینه `Cut` را غیر فعال کنید.

برای ایجاد منو های فرعی در ویژوال استودیو می توانید از کنترل `ContextMenuStrip` استفاده کنید. عملکرد این کنترل و نحوه ایجاد منو در آن نیز همانند کنترل `MenuStrip` است. تفاوت این کنترل با کنترل `MenuStrip` در آن است که در کنترل `ContextMenuStrip` فقط می توانید یک منو در بالاترین سطح داشته باشید و دیگر منو ها باید به

[1] منو های فرعی یا `Pop-Up Menus`. نام دیگر این منو ها، منو های زمینه یا `Context Menus` است.

عنوان زیر مجموعه آن واقع شوند، در صورتی که در کنترل MenuStrip می توانید به تعداد دلخـواه منـو در بـالاترین سـطح داشته باشید.
بیشتر کنترلهایی که در جعبه ابزار وجود دارند دارای خاصیتی به نام ContextMenuStrip هستند که می توانند شیئ ای را از این نوع قبول کنند. با تنظیم این خاصیت برای هر یک از کنترل ها، هنگامی که کاربر روی آن کنترل کلیک راست کند منـوی فرعی که به آن نسبت داده شده است نمایش داده می شود.
بعضی از کنترل ها همانند ComboBox و یا ListBox دارای یک منوی فرعی پیش فرض نیستند. دلیل این مورد هم بـه این علت است که این کنترل ها بیش از یک آیتم را در خود نگه داری می کنند و مانند کنترل TextBox فقط یک آیـتم درون آنها وجود ندارد. البته این کنترل ها نیز دارای خاصیت ContextMenuStrip هستند که به وسـیله آن مـی توانیـد یـک منوی فرعی را در برنامه برای آنها ایجاد کنید.

ایجاد منو های فرعی:

حال که با مفهوم منو های فرعی آشنا شدید، به بررسی آنها در کد و چگونگی استفاده از آن در برنامه های ویـژوال C# ۲۰۰۵ مـی پردازیم. در بخش امتحان کنید بعد، برنامـه ی قـسمت قبلـی را بـا اضـافه کـردن یـک منـوی فرعـی مخـصوص بـرای کنترلهـای TextBox کامل می کنیم. در این برنامه یک منوی فرعی ایجـاد کـرده و بـرای هـر دو کنتـرل TextBox از آن اسـتفاده خواهیم کرد. البته می توانیم دو منوی فرعی ایجاد کرده و در هر یک منو های جداگانه قرار دهیم که کارهای متفاوتی را انجام دهند، سپس هر یک از آنها را به یکی از TextBoxها نسبت دهیم، ولی در این برنامه نیازی به این کار نیست.

امتحان کنید: ایجاد منو های فرعی

۱) به قسمت طراحی فرم بروید و با استفاده از جعبه ابزار، یک کنترل ContextMenuStrip در فرم قـرار دهیـد. این کنترل نیز همانند کنترل MenuStrip به قسمت پایین بخش طراحی فرم اضافه می شود.
۲) در پنجره Properties مربوط به این کنترل، روی دکمه ... مقابل خاصیت Items آن کلیک کنید تا پنجـره Items Collection Editor نمایش داده شود.
۳) در پنجره Items Collection Editor روی دکمه ی Add کلیک کنید تـا یـک منـو بـه لیـست اضافه شود. خاصیت Name این منو را برابر با contextUndoToolStripMenuItem و خاصیت Text آن را نیز برابر با Undo قرار دهید. سپس روی دکمه ... در مقابل خاصیت Image کلیک کنید و از پنجـره Select Resource آیکونی را برای این منو انتخاب کنید.
۴) حال باید یک خط جدا کننده بین گزینه Undo و دیگر گزینه ها قرار دهید. برای این کار از لیست کنـار دکمـه Add، گزینه Separator را انتخاب کرده و بر روی دکمه Add کلیک کنید تا یک جدا کننده در این قسمت واقع شود.
۵) مجدداً از لیست سمت راست دکمه Add گزینه MenuItem را انتخاب کـرده و سـپس روی دکمـه Add کلیـک کنیـد تـا گزینـه دیگـری بـه ایـن منـو اضـافه شـود. خاصـیت Name ایـن گزینـه را برابـر بـا contextCutToolStripMenuItem و خاصیت Text آن را Cut قرار دهید. سپس با تنظـیم خاصیت Image آن به وسیله پنجره Select Resource آیکونی را برای این منو انتخاب کنید.

۶) در پنجره Items Collection Editor گزینه دیگری را به منو اضافه کرده، خاصیت Name آن را برابر با contextCopyToolStripMenuItem و خاصیت Text آن را برابر با Copy قرار دهید. سپس آیکونی را برای خاصیت Image آن در نظر بگیرید تا در منو نمایش داده شود.

۷) مجدداً روی دکمه Add کلیک کنید تا گزینه دیگری به منو اضافه شود. خاصیت Name این گزینه را برابر با contextPasteToolStripMenuItem و خاصیت Text آن را برابر با Paste قرار دهید. همچنین آیکونی را برای آن در قسمت Image مشخص کنید.

۸) حال باید جدا کننده دیگری بین گزینه های این قسمت و قسمت بعدی منو قرار دهیم. بنابراین از لیست سمت راست دکمه Add گزینه Separator را انتخاب کرده و روی دکمه فرمان Add کلیک کنید تا به لیست منو ها اضافه شود. به دلیل اینکه از این منو در کد استفاده نمی کنیم، نیازی نیست که خاصیتهای آن را تغییر دهید و می توانید خاصیتهای پیش فرض آن را قبول کنید.

۹) مجدداً از لیست سمت راست دکمه Add، گزینه MenuItem را انتخاب کرده و روی دکمه Add کلیک کنید تا گزینه دیگری به لیست Members اضافه شود. خاصیت Name گزینه جدید را برابر با contextSelectAllToolStripMenuItem و خاصیت Text آن را به Select All تغییر دهید. برای این گزینه نیازی نیست که آیکونی را تعیین کنید، بنابراین می توانید در پنجره Items Collection Editor روی دکمه OK کلیک کنید تا بسته شود.

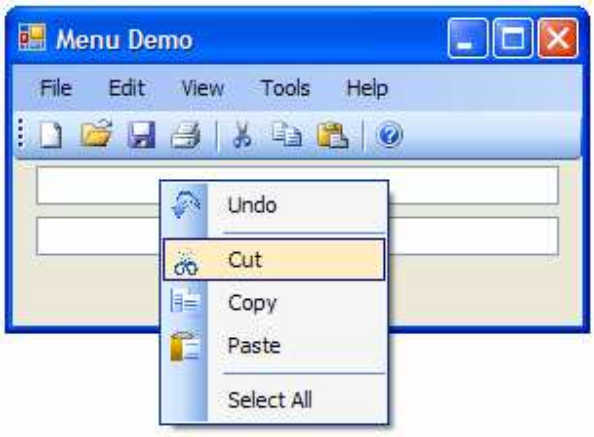
۱۰) هنگامی که به فرم برنامه برگردید مشاهده می کنید که منوی فرعی که ایجاد کرده بودید در بالای فرم نمایش داده می شود، برای حذف آن در قسمتی از فرم کلیک کنید. برای نمایش مجدد آن می توانید کنترل مربوط به آن را از پایین قسمت طراحی فرم انتخاب کنید.

۱۱) کنترل TextBox اول را در فرم انتخاب کرده و در قسمت Properties، خاصیت ContextMenuStrip آن را برابر با contextMenuStrip1 (یا هر نام دیگری که به کنترل مربوط به منوی فرعی نسبت داده اید) قرار دهید. این عمل را برای کنترل TextBox دوم نیز تکرار کنید.

۱۲) در این مرحله می توانید ظاهر منوی فرعی را که ایجاد کرده اید بررسی کنید. البته تاکنون هیچ کدی به این کنترل اضافه نکرده اید، پس هیچ یک از کنترلهای آن کار نمی کنند. برنامه را اجرا کنید و بعد از نمایش داده شدن فرم، روی کنترل TextBox اول کلیک راست کنید. این بار به جای منوی فرعی پیش فرض ویندوز، منوی فرعی که ساخته بودید همانند شکل ۸-۱۰ نمایش داده می شود. این کار را برای TextBox دوم نیز تکرار کنید. همانطور که مشاهده می کنید، منوی فرعی یکسانی برای هر دوی آنها نمایش داده می شود.

۱۳) از برنامه خارج شوید و به قسمت طراحی فرم برگردید. در این قسمت می خواهیم کد مربوط به گزینه های منوی فرعی را وارد کنیم. از پایین قسمت طراحی فرم، کنترل contextMenuStrip را انتخاب گرده تا منوی فرعی برنامه در بالای فرم نمایش داده شود. در این منو، روی گزینه Undo کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود، سپس کد زیر را در این متد وارد کنید:

```
private void contextUndoToolStripMenuItem_Click(
                        object sender, EventArgs e)
{
    // Call the undoToolStripMenuItem_Click procedure
    undoToolStripMenuItem_Click(sender, e);
}
```



شکل ۸-۱۰

۱۴) به قسمت طراحی فرم برگردید و در منوی فرعی روی گزینه Cut دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس کد زیر را در این متد وارد کنید:

```
private void contextCutToolStripMenuItem_Click(
                            object sender, EventArgs e)
{
     // Call the cutToolStripMenuItem_Click procedure
     cutToolStripMenuItem_Click(sender, e);
}
```

۱۵) همین مراحل را برای گزینه Copy در منوی فرعی نیز تکرار کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس کد زیر را در آن وارد کنید:

```
private void contextCopyToolStripMenuItem_Click(
                            object sender, EventArgs e)
{
    // Call the copyToolStripMenuItem_Click procedure
    copyToolStripMenuItem_Click(sender, e);
}
```

۱۶) با دو بار کلیک روی گزینه Paste در منوی فرعی برنامه، متد مربوط به رویداد کلیک آن را ایجاد کرده و کد زیر را در آن وارد کنید:

```
private void contextPasteToolStripMenuItem_Click(
                                object sender, EventArgs e)
{
    // Call the pasteToolStripMenuItem_Click procedure
    pasteToolStripMenuItem_Click(sender, e);
}
```

۱۷) آخرین گزینه ای که باید کد آن را بنویسید، گزینه `Select All` است. به قسمت طراحی فرم بروید و روی این گزینه در منوی فرعی نمایش داده شده در بالای فرم دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس کد زیر را در این متد وارد کنید:

```
private void contextSelectAllToolStripMenuItem_Click(
                                object sender, EventArgs e)
{
    // Call the selectAllToolStripMenuItem_Click procedure
    selectAllToolStripMenuItem_Click(sender, e);
}
```

۱۸) تمام کدی که باید برای منوی فرعی برنامه می نوشتید، همین بود. ساده بود، نه؟ حالا می توانید برنامه را اجرا کنید و عملکرد گزینه های مختلف منوی فرعی برنامه را مشاهده کنید. این گزینه ها نیز عملکردی مشابه دکمه های موجود در نوار ابزار و یا گزینه های نوار منو دارند. بنابراین تفاوتی ندارد که برای یک کار خاص از نوار ابزار، نوار منو و یا منوی فرعی استفاده کنید.

چگونه کار می کند؟

همانطور که در این تمرین مشاهده کردید، کنترل `ContextMenuStrip` نیز همانند کنترل `MenuStrip` کار می کند. به عبارت دیگر تمام ویژگی هایی که در کنترل `MenuStrip` وجود دارند در کنترل `ContextMenuStrip` نیز قابل دسترسی هستند که با استفاده از آنها می توانید به سادگی به طراحی منو های فرعی بپردازید. به نام گذاری منو های فرعی نیز توجه کنید. در ابتدای نام تمام آنها از `context` استفاده شده است تا بین گزینه های این منو و گزینه های منوی اصلی برنامه تفاوت ایجاد شود و تشخیص آنها از هم ساده تر باشد.
کدهایی که برای گزینه های این منو نوشتید نیز نکته تازه ای نداشت. متد مربوط به کارهایی مانند `Cut`، `Copy`، `Paste`، `Select All` و غیره را در قسمتهای قبل نوشته بودید و در این قسمت فقط آنها را فراخوانی کردید.

فعال و غیر فعال کردن گزینه های منو و دکمه های نوار ابزار:

همواره شرایطی در یک برنامه به وجود می آید که کاربر نمی تواند بعضی از کارهای خاص را انجام دهد. برای مثال هنگامی که متنی درون کلیپ برد قرار نگرفته است کاربر نمی تواند آن را در `TextBox` قرار دهد و یا تا زمانی که متنی در برنامه انتخاب نشده است کاربر نمی تواند آن را درون کلیپ برد قرار دهد. در چنین شرایطی بهتر است گزینه های مربوط به این امور در نوار منو و یا نوار ابزار غیر فعال باشند. نحوه فعال کردن و یا غیر فعال کردن دکمه های نوار ابزار و یا گزینه های نوار منو را در بخش امتحان کنید بعدی مشاهده خواهیم کرد.

امتحان کنید: فعال و غیر فعال کردن گزینه های منو و دکمه های نوار ابزار

۱) ابتدا زیربرنامه ای می نویسیم که با بررسی شرایط برنامه گزینه های نوار منو، دکمه های نوار ابزار و یا گزینه های منـوی فرعی را فعال و یا غیر فعال کند. بنابراین اگر برنامه در حال اجرا است آن را متوقف کرده و کد زیر را به کلاس حاوی فرم برنامه اضافه کنید:

```
private void ToggleMenus()
{
     // Declare a TextBox object and
     // set it to the ActiveControl
     TextBox objTextBox = (TextBox)this.ActiveControl;

     // Toggle the Undo menu items
     undoToolStripMenuItem.Enabled = objTextBox.CanUndo;
     contextUndoToolStripMenuItem.Enabled =
                                        objTextBox.CanUndo;

     // Toggle the Cut toolbar button and menu items
     if (objTextBox.SelectionLength == 0)
     {
          cutToolStripMenuItem.Enabled = false;
          contextCutToolStripMenuItem.Enabled = false;
          cutToolStripButton.Enabled = false;
     }
     else
     {
          cutToolStripMenuItem.Enabled = true;
          contextCutToolStripMenuItem.Enabled = true;
          cutToolStripButton.Enabled = true;
     }

     // Toggle the Copy toolbar button and menu items
     copyToolStripMenuItem.Enabled =
          Convert.ToBoolean(objTextBox.SelectionLength);
     contextCopyToolStripMenuItem.Enabled =
          Convert.ToBoolean(objTextBox.SelectionLength);
     copyToolStripButton.Enabled =
          Convert.ToBoolean(objTextBox.SelectionLength);

     // Toggle the Paste toolbar button and menu items
     pasteToolStripMenuItem.Enabled =
                               Clipboard.ContainsText();
     contextPasteToolStripMenuItem.Enabled =
                               Clipboard.ContainsText();
     pasteToolStripButton.Enabled =
                               Clipboard.ContainsText();

     // Toggle the Select All menu items
```

```
    selectAllToolStripMenuItem.Enabled =
    (objTextBox.SelectionLength < objTextBox.Text.Length);
    contextSelectAllToolStripMenuItem.Enabled =
    (objTextBox.SelectionLength < objTextBox.Text.Length);
}
```

۲) در فرم این برنامه فقط دو TextBox روی فرم در فعال بودن و یا نبودن دکمه های نوار ابزار و یا گزینه های منو ها تاثیر دارند. پس می توانیم هنگامی که کاربر ماوس خود را روی یکی از این کنترل ها بـرد، تـابع ToggleMenus را فراخوانی کنیم تا مشخص شود کدام کنترل ها باید فعال باشند و کدامیک نباید فعال باشند. برای ایـن کـار بـه قـسمت طراحی فرم بروید و کنتـرل TextBox اول را انتخـاب کنیـد. سـپس در پنجـره Properties روی آیکـون Events کلیک کنید تا لیست رویدادهای این کنترل نمایش داده شوند. از ایـن لیـست گزینـه MouseMove را انتخاب کرده و روی آن دو بار کلیک کنید تا متد مربوط به این رویداد ایجاد شود. سپس کد زیر را در این متد وارد کنید:

```
private void textBox1_MouseMove(object sender,
                                MouseEventArgs e)
{
    // Toggle the menu items and toolbar buttons
    ToggleMenus();
}
```

۳) مراحل قبلی را برای کنترل TextBox دوم نیز اجرا کنید و کد زیر را در متد مربوط به رویداد MouseMove آن وارد کنید:

```
private void textBox2_MouseMove(object sender,
                                MouseEventArgs e)
{
    // Toggle the menu items and toolbar buttons
    ToggleMenus();
}
```

۴) حال مجدداً برنامه را اجرا کرده و متنی را درون TextBox اول وارد کنید. سپس روی آن کلیک راست کنید تا منـوی فرعی آن نمایش داده شود. مشاهده می کنید که فقط گزینه های مناسب قابل انتخاب هستند و دیگر گزینه ها غیر فعـال شده اند (شکل ۸-۱۱).



شکل ۸-۱۱

چگونه کار می کند؟

اولین کاری که در زیربرنامه `ToggleMenus` انجام دهیم، این است که کنترل فعال را در فرم مشخص کنیم. برای این مورد می توانیم روشی را که در بخشهای قبلی استفاده کردیم به کار ببریم.

```
// Declare a TextBox object and
// set it to the ActiveControl
TextBox objTextBox = (TextBox)this.ActiveControl;
```

اولین گزینه ی منوی `Edit`، گزینه `Undo` است، بنابراین با این گزینه شروع می کنیم. کلاس `TextBox` دارای خاصیتی به نام `CanUndo` است که مشخص می کند آیا می توان آخرین عمل انجام شده در آن `TextBox` را لغو کرد یا نه؟ مقدار برگشتی این خاصیت برابر با `True` و یا `False` خواهد بود که می توانیم آن را به طور مستقیم در خاصیت `Enabled` گزینه `Undo` قرار دهیم. به این ترتیب اگر آخرین عمل انجام شده در `TextBox` قابل برگشت باشد، خاصیت `Enabled` آن برابر با `True` خواهد شد و در غیر این صورت برابر با `False` خواهد بود.

```
// Toggle the Undo menu items
undoToolStripMenuItem.Enabled = objTextBox.CanUndo;
contextUndoToolStripMenuItem.Enabled =
                                objTextBox.CanUndo;
```

گزینه بعدی در منوی `Edit`، گزینه `Cut` است. برای تعیین فعال یا غیر فعال بودن این گزینه می توانیم از خاصیت `SelectionLength` کلاس `TextBox` استفاده کنیم. این خاصیت مقداری از نوع عدد صحیح که مشخص کننده تعداد کاراکتر های انتخاب شده در `TextBox` است را برمی گرداند. اگر مقدار این خاصیت برابر با `0` بود یعنی متنی در `TextBox` انتخاب نشده است، پس نمی توان آن را `Cut` کرد. در این حالت باید گزینه `Cut` را در نوار منو، نوار ابزار و منوی فرعی غیر فعال کنیم:

```
// Toggle the Cut toolbar button and menu items
if (objTextBox.SelectionLength == 0)
{
     cutToolStripMenuItem.Enabled = false;
     contextCutToolStripMenuItem.Enabled = false;
     cutToolStripButton.Enabled = false;
}
else
{
     cutToolStripMenuItem.Enabled = true;
     contextCutToolStripMenuItem.Enabled = true;
     cutToolStripButton.Enabled = true;
}
```

بعد از این گزینه، گزینه Copy را از منوی Edit بررسی می کنیم. این گزینه نیز مانند Cut، هنگامی قابل انتخاب است که متنی در TextBox انتخاب شده باشد. بنابراین برای تعیین فعال یا غیر فعال بودن آن می توانیم همانند گزینه Cut ار بررسی خاصیت SelectionLength استفاده کنیم. روش دیگری که می توان در این قسمت به کار برد، تبدیل مقدار موجود در خاصیت SelectionLength به نوع Boolean است. برای تبدیل یک نوع داده ای به نوع داده ای دیگر می توانیم از توابع موجود در کلاس Convert استفاده کنیم. یکی از این توابع برای تبدیل یک مقدار int به Boolean به کار می رود. اگر عدد صحیحی که به این تابع فرستاده می شود تا تبدیل شود، برابر با صفر باشد مقدار برگشتی این تابع برابر با False خواهد بود در غیر این صورت این تابع مقدار True را برمی گرداند. بنابراین در اینجا نیز اگر مقدار خاصیت SelectionLength برابر با صفر باشد، تابع مقدار False را برگردانده و باعث غیرفعال شدن گزینه Copy می شود. در صورتی که کاربر متنی را در TextBox انتخاب کرده باشد، مقدار این خاصیت برابر با عددی به جز صفر خواهد بود. در نتیجه تابع مقدار True را برمی گرداند و باعث می شود که گزینه Copy فعال شود.

نکته: تابع ToBoolean از کلاس Convert، فقط بازای عدد صحیح صفر مقدار False برمی گرداند. هر پارامتری به جز صفر، چه مقدار مثبت باشد و چه منفی، باعث می شود که این تابع مقدار True را برگرداند.

نکته: از لحاظ سرعت، این روش با روشی که در مورد گزینه Cut استفاده کردیم تفاوت زیادی ندارد و تنها مزیت روش دوم در کوتاه تر بودن آن است. البته در روش دوم بهتر است که برای افزایش سرعت اجرای کد، ابتدا متغیری را از نوع Boolean تعریف کرده و مقدار برگشتی از تابع Convert.ToBoolean را در آن قرار دهیم. سپس مقدار این متغییر را به سه خاصیت مورد نظر نسبت دهیم. به این ترتیب فقط یک بار تابع Convert.ToBoolean را فراخوانی خواهیم کرد و از فراخوانی بی مورد این تابع نیز جلوگیری می شود.

```
// Toggle the Copy toolbar button and menu items
copyToolStripMenuItem.Enabled =

Convert.ToBoolean(objTextBox.SelectionLength);
contextCopyToolStripMenuItem.Enabled =
     Convert.ToBoolean(objTextBox.SelectionLength);
```

```
        copyToolStripButton.Enabled =

        Convert.ToBoolean(objTextBox.SelectionLength);
```

گزینه بعدی در منوی Edit، گزینه Paste است. روش تعیین فعال و یا غیر فعال بودن این گزینه با گزینه های دیگر مقداری متفاوت است. در این مورد باید بررسی کنیم که آیا متنی در داخل کلیپ بورد قرار دارد یا نه؟ برای تشخیص این مورد نیز می توانیم از متد ContainsText در کلاس Clipboard استفاده کنیم. کلاس Clipboard جزئی از فضای نام System.Windows.Forms است که به طور اتوماتیک به برنامه اضافه می شود. بنابراین می توانیم بدون مشخص کردن فضای نام آن، از این کلاس استفاده کنیم. تابع ContainsText مقداری را از نوع Boolean برمی گرداند که مشخص می کند آیا متنی در کلیپ بورد قرار دارد یا نه؟در صورت وجود متن در کلیپ بورد این تابع مقدار True را برمی گرداند، که با نسبت دادن آن به خاصیت Enabled از گزینه Paste باعث فعال شدن این گزینه می شویم. در صورتی که متنی وجود نداشته باشد هم این تابع با برگرداندن مقدار False باعث می شود که گزینه Paste غیر فعال شود.

```
        // Toggle the Paste toolbar button and menu items
        pasteToolStripMenuItem.Enabled =
Clipboard.ContainsText();
        contextPasteToolStripMenuItem.Enabled =
                                    Clipboard.ContainsText();
        pasteToolStripButton.Enabled =
                                    Clipboard.ContainsText();
```

گزینه آخری که در این قسمت باید مورد بررسی قرار گیرد، گزینه Select All است. این گزینه باید هنگامی فعال باشد که تمام متن داخل یک TextBox انتخاب نشده باشد. در غیر این صورت باید این گزینه غیر فعال باشد. برای بررسی اینکه تمام متن داخل یک TextBox انتخاب شده است یا نه مقدار موجود در خاصیت SelectionLength را با طول متن مقایسه می کنیم. اگر طول متن بزرگتر از مقدار انتخاب شده از متن بود، یعنی بخشی از متن انتخاب شده است. در این حالت می توانیم گزینه Select All را فعال کنیم.

```
        // Toggle the Select All menu items
        selectAllToolStripMenuItem.Enabled =
        (objTextBox.SelectionLength < objTextBox.Text.Length);
        contextSelectAllToolStripMenuItem.Enabled =
        (objTextBox.SelectionLength < objTextBox.Text.Length);
```

برای فعال و یا غیر فعال کردن گزینه های نوار منو و یا نوار ابزار، باید این زیربرنامه را فراخوانی کنیم. بهترین مکان برای فراخوانی این تابع، درون رویداد MouseMove کنترل TextBox است. این رویداد هنگامی رخ می دهد که ماوس از روی کنترل عبور کند، که در این حالت می توان حدس زد ممکن است کاربر بخواهد روی TextBox کلیک (و یا کلیک راست) کرده و گزینه ای را انتخاب کند.

```
private void textBox1_MouseMove(object sender,
                                    MouseEventArgs e)
{
        // Toggle the menu items and toolbar buttons
```

```
        ToggleMenus();
}
```

نتیجه:

در این فصل در طی چندین تمرین کاربردی و با استفاده از چند مثال با نحوه استفاده از منو ها، زیرمنو ها و گزینه های موجود در آن آشنا شدیم. همچنین مشاهده کردید که چگونه می توان از چند نوار ابزار در برنامه استفاده کرد، با وجود اینکه این مورد مدنظر این فصل نبوده است. علاوه بر این آموختید که چگونه برای منو ها کلیدهای دسترسی، شورت کات و یا آیکون تعیین کنید.
در طی ایجاد نوار ابزار `Edit`، با نحوه کاربرد تکنیکهای ابتدایی ویرایش متن به وسیله توابع درونی کلاس `TextBox` و کلاس `Clipboard` آشنا شدید و مشاهده کردید که به سادگی می توان این ویژگی ها را به برنامه اضافه کرد – ویژگیهایی که کاربر از هر برنامه ویندوزی انتظار دارد.
همچنین با منو های فرعی که ویندوز به طور پیش فرض برای کنترل ها قرار می دهد و نیز چگونگی ایجاد منو های فرعی مخصوص به برنامه خودتان آشنا شدید.

در پایان این فصل باید با موارد زیر آشنا شده باشید:

- با استفاده از کنترل `MenuStrip` به برنامه خود منو و زیرمنو اضافه کنید.
- در کنار گزینه های موجود در منوی برنامه علامت تیک قرار دهید.
- به منو های برنامه خود شورت کات و کلید دسترسی اختصاص دهید.
- با استفاده از کنترل `ContextMenuStrip` برای برنامه خود منو های فرعی ایجاد کنید.
- با استفاده از خاصیتهای موجود در کلاس `TextBox`، گزینه های منو ها را در صورت لزوم فعال و یا غیر فعال کنید.

تمرین:

برای اینکه ظاهر برنامه ای که در این فصل ایجاد کردید بیشتر مشابه برنامه های ویندوزی باشد، با استفاده از کنترل `StatusStrip` یک نوار ابزار به برنامه اضافه کنید و کدی را برای آن بنویسید که هنگام انجام دادن اعمالی مانند `Cut`، `Copy`، `Paste` و ... متن مناسبی برای کاربر نمایش داده شود.

فصل نهم: ساختن اشیا

از زمانی که با کامپیوتر آشنا شدید تاکنون ممکن است واژه **شیئ گرایی** را زیاد شنیده باشید. ممکن است شنیده باشید که این مبحث، یکی از مباحث سخت و غیر قابل درک برنامه نویسی است. در سالهای اول که این نوع برنامه نویسی به وجود آمده بود این گفته صحت داشت، اما زبانها و ابزارهای مدرن امروزی باعث شده اند برنامه نویسی شیئ گرا به مبحثی بسیار ساده تبدیل شود. برنامه نویسی شیئ گرا منافع زیادی را برای توسعه گران نرم افزاری[1] دارد، به همین دلیل زبانهای برنامه نویسی مانند C++، C# و ویژوال بیسیک و ... سعی کرده اند به نحوی رشد کنند که به ساده گی بتوان به وسیله آنها برنامه های شیئ گرا را طراحی و پیاده سازی کرد.

در بیشتر فصلهای این کتاب از اشیا و کلاسها استفاده کرده ایم، اما در این فصل بر مبحث شیئ گرایی تمرکز خواهیم کرد و آن را به تفصیل مورد بررسی قرار خواهیم داد. همچنین سعی می کنیم بر پایه تجربیات و آموخته هایی که در فصلهای قبل از برنامه نویسی بدست آورده اید و مواردی که در این فصل می آموزید، برنامه های جالبی را با ویژوال C# ایجاد کنیم.

در این فصل:

- با نحوه ی ساختن یک کلاس قابل استفاده با چندین متد و خاصیت گوناگون آشنا خواهیم شد.
- با مفهوم ارث بردن یک کلاس از کلاسی که قبلاً ایجاد کرده اید آشنا خواهید شد.
- چگونگی تغییر دادن متدهای کلاس پایه در کلاس مشتق شده را مشاهده خواهید کرد.
- چگونگی ایجاد فضای نام را مشاهده خواهید کرد.

مفهوم اشیا:

یک شیئ در دنیای واقعی، اغلب هر چیزی است که بتوانید تصور کنید. معمولاً در تمام طول روز در حال کار کردن با اشیای فیزیکی مانند تلویزیونها، اتومبیلها، مشترکین[2]، گزارشات، لامپ ها و یا هر چیز دیگری هستید. در کامپیوترها نیز، یک شیئ مشخص کننده هر چیزی است که در کامپیوتر می بینید و یا در برنامه های تان از آن استفاده می کنید. برای مثال فرم برنامه، دکمه های نوار ابزار، گزینه های موجود در منو ها و ... همه نمونه هایی از اشیا در کامپیوتر هستند. اشیای دیگری نیز در کامپیوتر وجود دارند که نمود ظاهری ندارند، اما در بخشهای مختلف برنامه به کار می روند مانند شیئ برای کنترل کاربران یک برنامه. کاربر برنامه چنین شیئ را در برنامه نمی بیند اما در طول اجرای برنامه این شیئ وجود دارد و وظایف خود را انجام می دهد.

فرض کنید می خواهید در قسمتی از برنامه برای یکی از کاربران، صورت حساب صادر کنید. در این صورت باید در برنامه ی خود یک شیئ برای نشان دادن صورت حساب و یک شیئ نیز برای نشان دادن کاربر داشته باشید. فرض کنید `Customer` شیئ مربوط به کاربر و `Bill` شیئ مربوط به صورت حساب باشد. شیئ `Customer` دارای خصوصیت ها ایی مانند نام، آدرس، تلفن و ... است که هر کاربر داراست. به علاوه، این شیئ باید بتواند یک صورت حساب برای خود ایجاد کند و می دانید که در این برنامه، صورت حساب ها را با شیئ ای از نوع `Bill` نشان می دهیم. پس در واقع می توان گفت شیئ `Customer` باید بتواند یک شیئ از نوع `Bill` ایجاد کند که صورت حساب کاربر را نشان دهد.

---

[1] منظور از توسعه گران نرم افزاری، افرادی است که در زمینه ی طراحی و توسعه ی نرم افزار فعالیت دارند مانند طراحان نرم افزار، برنامه نویسان و ....

[2] شاید با توجه به تعریف شیئ، یک مشترک را نتوان یک شیئ محسوب کرد. اما در اینجا فرض می کنیم که هر چیزی که در دنیای واقعی وجود دارد یک شیئ است.

شیئ `Bill` نیز می تواند جزئیات مربوط به یک صورت حساب را در خود نگهداری کند و نیز امور مربوط به آن را انجام دهد. برای مثال این شیئ باید بتواند صورت حساب ایجاد شده را چاپ کند.
اما نکته مهم و قابل توجه در اینجا این است که این اشیا باید هوشمندی لازم برای انجام تمام کارهای مربوط به خودشان را داشته باشند. در مثال قبلی، اگر شیئ ای از نوع `Customer` داشته باشید که مخصوص به یکی از کاربران برنامه است، می توانید به سادگی به آن شیئ بگویید که "یک صورت حساب برای خودت ایجاد کن". سپس این شیئ باید بتواند تمام کارهای طولانی و حتی سخت مربوط به ایجاد کردن یک صورت حساب را انجام داده و در انتها یک شیئ جدید از نوع `Bill` را به عنوان صورت حساب خود به شما برگرداند. همچنین اگر شیئ از نوع `Bill` به عنوان صورت حساب یکی از کاربران خود در اختیار داشتید، می توانید به آن بگویید که "جزئیات صورت حساب را چاپ کن" و شیئ `Bill` نیز باید قابلیت انجام این کار را داشته باشد.
اگر به خاطر داشته باشید، هنگام معرفی الگوریتم ها و چگونگی تبدیل آنها به برنامه های کامپیوتری گفتیم که برای حل یک مسئله باید بتوان آن را به قسمتهای کوچکتر تقسیم کرد. برنامه نویسی شیئ گرا نیز به این دلیل در مهندسی نرم افزار کاربرد بسیاری دارد که می توان به وسیله آن یک مسئله پیچیده را به بهترین نحو به چندین قسمت کوچکتر تقسیم کرد و سپس با قرار دادن آنها کنار هم، به راه حل مسئله اصلی رسید. ممکن است بگویید که این کار با استفاده از متد ها نیز امکان پذیر است، پس چه نیازی است که از شیئ ها استفاده کنیم؟ در توضیحات قبلی مشاهده کردید روابطی که برای `Customer` و `Bill` شرح داده شد بسیار مشابه دنیای واقعی است. در حقیقت در برنامه نویسی شیئ گرا سعی می شود تمام موارد شبیه به آنچه در واقعیت وجود دارد ایجاد شود و تمام روابط بین اشیا نیز بر این اساس صورت گیرد.
مورد دیگری که باعث قدرت فراوان برنامه نویسی شیئ گرا می شود در این است که، شما به عنوان یکی از افرادی که از یک شیئ استفاده می کنید نیازی به دانستن اینکه چگونه آن شیئ درخواستهای شما را انجام می دهد ندارید. این مورد در دنیای واقعی نیز صادق است. هنگامی که در حال استفاده از یک موبایل هستید، نیازی نیست که بدانید این دستگاه به صورت درونی چگونه کار می کند؟ حتی اگر نحوه عملکرد آن را بدانید و یا حتی آن را خودتان اختراع کرده باشید نیز، ساده تر است که برای کار با آن از رابط کاربری ساده ای که در اختیار شما قرار می گیرد استفاده کنید. به این صورت از خطاهای احتمالی که ممکن است هنگام کار با آن ایجاد شود نیز جلوگیری می کنید. این مورد در کامپیوترها هم صادق است. هنگامی که بخواهید با یک شیئ کار کنید، حتی اگر آن شیئ را خودتان ایجاد کرده باشید، بهتر است از رابط ساده و راحتی که آن شیئ فراهم می کند استفاده کنید و اجازه دهید که پیچیدگی های مربوط به وظیفه ی آن شیئ در پشت رابط ساده ی آن پنهان بماند.
برای توضیح بیشتر در مورد اشیا، بهتر است یک شیئ مانند تلویزیون را در دنیای واقعی بررسی کنیم. یک دستگاه تلویزیون را در نظر بگیرید. یک سری از کارها هستند که می دانید چگونه می توان با یک تلویزیون آنها را انجام داد. مانند:

- تماشای تصویر روی صفحه نمایش.
- تغییر دادن کانال تلویزیون.
- تغییر صدای تلویزیون.
- خاموش و یا روشن کردن آن.

هنگامی که در حال استفاده از یک تلویزیون هستید، نیازی نیست که بدانید اجزای تشکیل دهنده یک تلویزیون چگونه درخواستهای شما را انجام می دهند. احتمالاً اگر از شما بخواهند قسمتهای مختلفی را در کنار هم قرار دهید و یک تلویزیون مدرن ایجاد کنید، قادر به انجام چنین کاری نخواهید بود. البته ممکن است بعد از مدتها مطالعه و آزمایش بتوانید یک دستگاه تلویزیون ساده ایجاد کنید که باز هم به پیشرفتگی تلویزیون هایی که اکثر مردم از آن استفاده می کنند نخواهد بود. با وجود این، هم شما و هم اغلب مردم می دانند که چگونه باید از یک دستگاه تلویزیون استفاده کنند، چگونه کانال آن را تغییر دهند، چگونه صدای آن را تنظیم کنند، چگونه آن را خاموش و یا روشن کنند و غیره.

اشیا در مهندسی نرم افزار نیز اساساً به همین روش کار می کنند. هنگامی که یک شیئ در اختیار داشته باشید می توانید به سادگی از آن استفاده کنید و بخواهید که وظایف لازم را انجام دهد. برای این موارد لازم نیست بدانید که شیئ به صورت درونی چگونه کار می کند و یا لازم نیست حتی کوچکترین اطلاعاتی در مورد نحوه عملکرد آن داشته باشید.
اشیای نرم افزاری عموماً دارای مشخصات زیر هستند:

- **هویت:** یک شیئ همواره نوع خود را می داند. برای مثال یک تلویزیون همواره میداند که یک شیئ از نوع تلویزیون است.
- **حالت:** هر شیئ در هر زمانی وضعیت و حالت خود را میداند. برای مثال اگر در حال مشاهده کانال ۴ از یک تلویزیون باشید و بخواهید که تلویزیون شماره کانالی را که مشاهده می کنید روی صفحه نمایش دهد، عدد ۴ را نمایش خواهد داد.
- **رفتار:** به عکس العمل یک شیئ در مقابل درخواستهای کاربر، رفتار آن شیئ می گویند. برای مثال فرض کنید از تلویزیون بخواهید تا صدای آن زیاد شود. این افزایش صدای تلویزیون جزئی از رفتار آن محسوب می شود.

کپسولی بودن:

مفهوم اصلی که در پشت شیئ–گرایی قرار دارد **کپسولی بودن**[1] است. این مفهوم با وجود ساده بودن، از اهمیت زیادی برخوردار است. ایده کلی که کپسولی بودن ارائه می دهد به این صورت است که رفتار یک شیئ تا حد ممکن باید دور از دید کاربر باشد. به عبارت دیگر تا زمانی که لازم نباشد، کاربر نباید متوجه شود که یک شیئ چگونه درخواستهای او را انجام می دهد.
کنترل `OpenFileDialog` که در فصلهای قبل از آن استفاده می کردیم را به خاطر بیاورید. هنگامی که می خواستیم یکی از آنها را در صفحه نمایش دهیم، بعد از اینکه وضعیت آن را با استفاده از خاصیت ها تنظیم می کردیم، از آن می خواستیم که در صفحه نمایش داده شود (با استفاده از تابع `ShowDialog`). ولی همانطور که می دانید آن شیئ هیچ اطلاعاتی را در مورد نحوه نمایش داده شدنش در صفحه بروز نمی داد و ما نیز نیازی به دانستن این که چگونه این شیئ در صفحه نمایش داده می شود نداشتیم. این مورد که شیئ `OpenFileDialog`، نحوه انجام دادن وظایفش را از کاربر پنهان می کرد، جزئی از کپسولی بودن آن محسوب می شود.

متد ها و خاصیت ها:

برای ارتباط با یک شیئ از متد ها و خاصیتهای آن استفاده می کنند. در تعریف آنها می توانیم بگوییم:

- **متد ها:** روشهایی هستند که به وسیله آن می توان به یک شیئ گفت چگونه وظیفه خاصی را انجام دهد.
- **خاصیت ها:** اعضایی از یک شیئ هستند که ویژگیهای آن را شرح می دهند.

در قسمتهای قبل، متد ها به عنوان قطعه کد هایی معرفی شده بودند که می توانستند وظیفه مشخصی را انجام دهند. البته این تعریف درست است، اما تعریف بسیار ساده ای از یک متد است. تعریف کامل متد که فقط در برنامه نویسی شیئ گرا صادق است، عبارت است از کد هایی که به یک شیئ می گویند چگونه وظیفه مشخصی را انجام دهد.

[1] `Encapsulation`

بنابراین برای روشن کردن یک تلویزیون باید متدی را پیدا کنید که این کار را انجام دهد، زیرا همانطور که گفتم این متدها هستند که به یک شیئ می گویند چگونه یک وظیفه مشخص را انجام دهند. هنگامی که متد مربوط به این کار را فراخوانی می کنید، فرض می کنید که شیئ به واسطه این متد می داند که چگونه درخواست شما را انجام دهد.
خاصیت ها نیز برای تنظیم حالتها و ویژگیهای یک شیئ به کار می روند. در این مورد نیز وظیفه شیئ است که با توجه به حالتی که برای آن تعریف شده است عمل کند. برای مثال هنگامی که بخواهید کانال یک تلویزیون را از ۴ به ۳ تغییر دهید، کافی است مقدار خاصیت مربوط به کانال را برابر با ۳ قرار دهید و تلویزیون باید با توجه به مقدار این خاصیت، تصویر را در صفحه نمایش دهد.
همانطور که به خاطر دارید هنگام کار با شیئ `OpenFileDialog` در بخش کادرهای محاوره ای، کافی بود که خاصیت مربوط به متن نوار عنوان آن را برابر با مقدار دلخواه قرار دهید. بعد از آن، وظیفه شیئ `OpenFileDialog` محسوب می شد تا متن مشخص شده را در نوار عنوان نمایش دهد.

رویدادها:

رویدادها به عنوان عضوی از اشیا هستند که پیغامی را آماده کرده و به دیگر بخشهای برنامه می فرستند. زمانی که یک مورد قابل توجه برای یک شیئ رخ داد، آن شیئ رویدادی را فراخوانی می کند. به عبارت دیگر به وسیله یک پیغام، بخشهای دیگر برنامه را از این رویداد مطلع کرده و همچنین اطلاعات لازم را نیز در مورد این رویداد در اختیار این بخشها قرار می دهد. این پیغام فقط به قسمتهایی از برنامه فرستاده می شود که پیش از اتفاق افتادن رویداد، از برنامه درخواست کرده باشند تا این مورد را به آنها اطلاع دهد.
به عنوان مثال رویداد `Click` را که در بخشهای قبلی از آن استفاده می کردیم در نظر بگیرید. این رویداد زمانی رخ می دهد که کاربر ماوس را روی کنترل برده و در آنجا کلیک کند. به این ترتیب شیئ `Button` تمام اشیای لازم را از رخ دادن این رویداد مطلع می کند. برای اینکه شیئ ای از این رویداد مطلع شود، باید قبل از آن به شیئ `Button` بگوید در صورت رخ دادن این رویداد به آن نیز اطلاع داده شود.
روش این کار را در فصلهای قبل نیز مشاهده کردید. برای استفاده از این رویداد ابتدا تابعی را با قالبی خاص ایجاد می کردیم و سپس به شیئ `Button` اطلاع می دادیم که در صورت وقوع این رویداد، تابع مشخص شده را اجرا کند (البته بیشتر این کارها به صورت اتوماتیک توسط ویژوال استودیو انجام می شد). هنگامی که کاربر روی کنترل `Button` کلیک می کرد، این کنترل توابعی که مشخص شده بودند را فراخوانی می کرد و همچنین اطلاعات مورد نیاز آنها را نیز به عنوان پارامتر به آنها ارسال می کرد.

قابل رویت بودن:

یکی از ویژگیهای شیئ ای که خوب طراحی شده است، این است که استفاده از آن راحت باشد. برای مثال اگر بخواهید یک شیئ تلویزیون طراحی کنید باید بدانید که چه فرکانسی برای این شیئ مورد نیاز است. اما آیا فردی که از تلویزیون استفاده می کند نیز باید این مورد را بداند؟ و یا حتی مهمتر اینکه آیا می توان به کاربر اجازه داد این مقدار فرکانس را به صورت مستقیم تغییر دهد.
هنگام طراحی یک شیئ همواره بخشهایی از آن به صورت شخصی و بخشهای دیگری به صورت عمومی هستند. بخشهای عمومی می توانند توسط همه کاربران مورد استفاده قرار گیرند، اما بخشهای شخصی فقط توسط خود شیئ قابل دسترسی هستند. منطق و روشی که شیئ با آن کار می کند معمولا در بخشهای شخصی شیئ قرار می گیرند و می تواند شامل متدها و خاصیت هایی باشد که برای عملکرد شیئ مهم هستند ولی نباید توسط کاربران به طور مستقیم مورد استفاده قرار گیرد.

برای مثال یک تلویزیون ممکن است دارای متد هایی برای وصل کردن برق به اجزا، جریان دادن برق در مدارها و ... باشد. اما این متدها نباید توسط کاربر به طور مستقیم فراخوانی شوند، بلکه کاربر باید متد SwitchOn را برای روشن شدن دستگاه فراخوانی کند، سپس این متد باید به ترتیب متدهای گفته شده را استفاده کرده و به این ترتیب دستگاه را روشن کند. به عنوان مثالی دیگر، در شیئ تلویزیون یک خاصیت عمومی به نام کانال وجود دارد که با استفاده از یک خاصیت خصوصی به نام فرکانس کار می کند. تا زمانی که تلویزیون نداند برای هر کانال باید از چه فرکانسی استفاده کند، نمی تواند آنها را نمایش دهد. اما کاربر تلویزیون نباید به طور مستقیم بتواند فرکانس را تغییر دهد. بلکه باید به وسیله تغییر خاصیت کانال، موجب تغییر فرکانس شود.
حال که با مفاهیم ابتدایی شیئ گرایی آشنا شدید، بهتر است به بررسی استفاده از این موارد در برنامه بپردازیم. در اغلب کد هایی که در فصلهای قبل از آنها استفاده کرده ایم، خطی مانند کد زیر دیده می شده است:

```
lstData.Items.Add(strData);
```

این کد نمونه ای عادی از شیئ گرایی است. در این کد lstData در حقیقت یک شیئ و Items نیز خاصیتی از این شیئ است. خاصیت Items خود نیز یک شیئ است که دارای متدی به نام Add است. علامت نقطه (.) به ویژوال C# می گوید که کلمه سمت راست، نام عضوی از شیئ سمت چپ است. بنابراین در این کد، Items عضوی از lstData و Add عضوی از Items به شمار می رود.
lstData یک نمونه از کلاس System.Windows.Forms.ListBox است. این کلاس یکی از کلاسهای موجود در چارچوب .NET است که در فصل دوم با آن آشنا شدید. کلاس ListBox برای نمایش لیستی از عناصر در فرم به کار می رود و به کاربر اجازه می دهد که عناصری را از آن انتخاب کند. اگر دقت کنید نمونه های زیادی از کپسولی بودن را در این کلاس مشاهده خواهید کرد. شما به عنوان یکی از کاربران این کلاس، نیازی ندارید که بدانید برای نمایش لیستی از عناصر در صفحه نمایش و یا دریافت ورودی کاربر از چه تکنولوژیهایی باید استفاده کرد. حتی ممکن است تاکنون نام مواردی مانند GDI+، stdin، درایورهای کیبورد، درایورهای صفحه نمایش و یا بسیاری از تکنولوژیهای پیچیده دیگری که در این کلاس مورد استفاده قرار می گیرند را نیز نشنیده باشید. اما به راحتی می توانید از این کلاس استفاده کرده و عناصر مورد نظرتان را به صورت لیست در صفحه نمایش دهید.
البته همانطور که در ابتدای فصل نیز گفتم، ListBox یکی از اشیایی است که در یک برنامه استفاده شده است. عموماً یک برنامه از تعداد زیادی شیئ تشکیل می شود که فقط بعضی از آنها همانند شیئ ListBox قابل مشاهده هستند. اشیای بسیار دیگری نیز هستند که در ایجاد یک برنامه نقش دارند، اما از نظر کاربر پنهان هستند و در حافظه وظیفه خود را انجام می دهند.

یک کلاس چیست؟

کلاس تعریفی است که برای یک نوع خاص از شیئ به کار می رود. کلاسها را عموماً در دنیای واقعی نمی توان نشان داد. برای نمونه، مثال تلویزیون در بخش قبل را در نظر بگیرید. در دنیای واقعی هیچگاه نمی توانید شیئ ای را مشخص کنید و بگویید که "این تلویزیون است". بلکه همواره می توانید شیئ ای را مشخص کنید و بگویید "این یک نمونه از تلویزیون است". زیرا همانطور که می دانید، تلویزیون یک تعریف است و نمی توان در دنیای واقعی آن را نمایش داد. بلکه باید ابتدا آن را نمونه سازی کرد.
در دنیای نرم افزار نیز کلاسها به همین صورت هستند. یک کلاس از کدهای لازم برای ذخیره و نگهداری مقادیر خاصیت ها، انجام دادن متدها، تعیین زمان رخ دادن رویدادها و ... تشکیل شده است. اگر بخواهید یک شیئ نرم افزاری ایجاد کنید، باید بدانید که درون آن شیئ دقیقاً باید چگونه کار کند. سپس کد مربوط به این کارکرد را به وسیله یک زبان برنامه نویسی مانند C# در قالب یک کلاس می نویسید. بنابراین هنگامی که برنامه نویس دیگری بخواهد از شیئ که ایجاد کرده اید استفاده کند، فقط به این شیئ می

گوید که "صدا را زیاد کن". در این مرحله باید به عنوان کسی که این شیئ را ایجاد کرده است بدانید چگونه به آمپلی فایر دستور دهید که خروجی صدا را زیاد کند.

نکته: در مثال بالا توجه کنید که خود آمپلی فایر نیز یک شیئ است، بنابراین لازم نیست دقیقاً بدانید که آمپلی فایر به صورت درونی چگونه کار می کند. در برنامه نویسی شیئ گرا، یک شیئ معمولا از چندین شیئ دیگر به اضافه مقداری کد برای ایجاد ارتباط بین آنها تشکیل می شود، همانطور که در دنیای واقعی نیز یک تلویزیون از چندین قسمت عادی به اضافه مقداری مدار برای ایجاد ارتباط بین آن قسمتها تشکیل شده است.

هر شیئ ی که از یک کلاس ایجاد شود، به عنوان نمونه ای از آن کلاس در نظر گرفته می شود. بنابراین اگر ۵۰ عدد شیئ تلویزیون داشته باشید، در حقیقت ۵۰ نمونه از کلاس تلویزیون دارید. عمل ایجاد یک نمونه جدید از یک کلاس را **نمونه سازی**[1] می گویند. از این به بعد خواهیم گفت که "یک کلاس ایجاد کنید" اما در مورد اشیا می گوییم "یک شیئ را نمونه سازی کنید" (البته این تفاوت در گفتار فقط برای رفع ابهام است). ایجاد یک کلاس در زمان طراحی برنامه، زمانی که در حال نوشتن کد برنامه هستید صورت می گیرد. اما نمونه سازی اشیا از کلاس در زمان اجرای یک برنامه، هنگامی که برنامه بخواهد از آن شیئ استفاده کند صورت می گیرد.
یک مثال خوب برای این مورد، مثال قالبهای فلزی است. فرض کنید یک قالب فلزی دایره ای و مقداری خمیر در اختیار دارید. قالب همواره شکل ثابتی دارد و به تنهایی برای شما قابل استفاده نیست. بلکه باید مقداری خمیر در آن قرار دهید تا خمیر ها به شکل قالب درآیند. سپس از خمیر های شکل گرفته استفاده کنید. در ایجاد دایره ها از نظر قالب هیچ محدودیتی وجود ندارد، بلکه به هر اندازه که خمیر در اختیار داشته باشید می توانید با استفاده از قالب، دایره ایجاد کنید. توجه کنید که قالب هیچ موقع از بین نمی رود ولی دایره هایی که با آن ایجاد کرده اید، هنگامی که دیگر نیازی به آنها نداشته باشید و یا بخواهید به وسیله آنها شکل دیگری ایجاد کنید از بین می روند. کلاسها و اشیا نیز چنین رابطه ای دارند. کلاسها به صورت یک قالب هستند و حافظه یک کامپیوتر نیز به صورت خمیر ها. برای ایجاد شیئ از کلاس، خود کلاس محدودیتی در مورد تعداد اشیای ایجاد شده ندارد، تنها محدودیت از طرف مقدار حافظه موجود در کامپیوتر است. همچنین یک کلاس از ابتدا وجود دارد، اما یک شیئ فقط هنگامی وجود دارد که بخواهد استفاده شود. بعد از اتمام کار آن، شیئ از حافظه پاک می شود.
هنگامی که در زمان اجرای برنامه از یک کلاس نمونه سازی کردید، می توانید خاصیتهای نمونه ایجاد شده را تنظیم کنید، متدهای آن را فراخوانی کنید و .... برای مثال فرض کنید که در یک برنامه کلاسی برای تلویزیون ایجاد کرده اید. در زمان اجرای برنامه می توانید به هر تعداد که بخواهید از این کلاس نمونه سازی کنید. سپس برای مثال با فراخوانی متد `SwitchOn` از یکی از نمونه ها، آن نمونه از تلویزیون را روشن کنید و یا با تنظیم خاصیت `Channel` یکی از نمونه ها، کانال آن نمونه از تلویزیون را تغییر دهید.

ایجاد کلاسها:

در فصلهای قبلی به کرّات کلاس هایی ایجاد کرده و از آن در برنامه های نمونه استفاده کرده ایم. به طور کلی هنگامی که الگوریتم یک برنامه را طراحی کردید، اشیای زیادی در دنیای واقعی در آن دیده می شوند. برای نوشتن برنامه باید تمام این اشیای واقعی را، به اشیای در برنامه خودتان تبدیل کنید. به مثال زیر توجه کنید:

[1] `Instantiation`

- لیستی مشتمل بر ۱۰ کاربر را از بانک اطلاعاتی انتخاب کن.
- از اولین کاربر انتخاب شده شروع کن و برای هر یک صورت حساب او را آماده کن.
- زمانی که هر یک از صورت حساب ها آماده شدند، با استفاده از چاپگر آن را چاپ کن.

برای اینکه یک برنامه به طور کامل شیئ گرا باشد، باید هر شیئ که در دنیای واقعی آن برنامه وجود دارد، به شیئ در آن برنامه تبدیل شود. برای مثال:

- **Customer:** یک شیئ برای مشخص کردن کاربر.
- **Bill:** یک شیئ برای مشخص کردن صورت حساب ایجاد شده.
- **Printer:** یک شیئ که یک چاپگر سخت افزاری را مشخص می کند و می تواند برای چاپ صورت حساب مورد استفاده قرار بگیرد.

هنگامی که از ویژوال C# ۲۰۰۵ برای برنامه نویسی استفاده می کنید، مجموعه وسیعی از کلاسها را تحت عنوان **کتابخانه کلاس چارچوب .NET**[1] در اختیار دارید. این کلاسها هر چیزی برای محاسبات در محیطی که می خواهید برای آن برنامه بنویسید را پوشش می دهند. بنابراین برنامه نویسی برای .NET به سادگی استفاده از چند کلاس و ایجاد ارتباط درست بین آنها و یا ترکیب چند کلاس و ایجاد یک کلاس جدید است. معمولاً هنگام ساختن یک برنامه، بعضی از کلاسهای مورد نیازتان در چارچوب .NET وجود دارند و بعضی از آنها را نیز باید خودتان ایجاد کنید.
برای مثال در چارچوب .NET کلاس هایی برای چاپ و یا کلاس هایی برای دسترسی به بانک اطلاعاتی وجود دارد، بنابراین اگر در الگوریتم خود به دسترسی به بانک اطلاعاتی و یا چاپ اطلاعات نیاز داشتید، لازم نیست که کلاس آنها را مجدداً بنویسید. اگر بخواهید مطلبی را چاپ کنید کافی است شیئ از کلاس مربوط به چاپ را نمونه سازی کنید، به آن شیئ بگویید که چه مطلبی را می خواهید چاپ کنید و سپس آن شیئ عمل چاپ را برای شما انجام می دهد. نیازی نیست که بدانید شیئ مذکور چگونه سند شما را به دستورات PostScript تبدیل می کند و از طریق پورت مخصوص چاپگر آن را به دستگاه می فرستد، فقط کافی است نحوه کاربرد متدها و خاصیتهای آن را بدانید، بقیه موارد را خود شیئ انجام می دهد.

قابلیت استفاده مجدد:

یکی از زیباترین جنبه های برنامه نویسی شیئ گرا قابلیت استفاده مجدد[2] از یک کد است. برای درک بهتر این قابلیت بهتر است مثالی را بررسی کنیم. فرض کنید در یک شرکت خدمات تلفنی کار می کنید و به دو برنامه ی متفاوت نیاز دارید. برنامه اول برای کنترل امور مشترکین تلفن ثابت و برنامه دوم برای کنترل امور مشترکین تلفن همراه است. در هر کدام از این برنامه ها نیاز است که از کلاسی به نام Customer استفاده کنید.
هنگامی که بخواهید این دو برنامه را بنویسید، احتمالا هر دو را با هم شروع نخواهید کرد. از برنامه اول شروع می کنید، هنگامی که تمام شد برنامه دوم را شروع می کنید. حال بهتر است که برای هر کدام از آنها یک کلاس Customer جداگانه بنویسید و یا یک کلاس کلی برای Customer در برنامه اول بنویسید و در برنامه ی دوم از آن استفاده کنید؟ خوب مشخص است که روش دوم بهتر است.

---

[1] .NET Framework Class Library
[2] Reusability

استفاده مجدد از یک کلاس در حالت کلی مورد خوبی به شمار می رود اما چند نکته هم باید برای این مورد در نظر گرفته شود. به صورت ایده آل اگر یک کلاس `Customer` را برای یک برنامه ایجاد کنید و سپس در برنامه دیگری به استفاده از کلاسی مشابه کلاس `Customer` نیاز داشته باشید، باید بتوانید از کلاس `Customer` قبلی استفاده کنید. ممکن است در بعضی شرایط به دلایلی نتوانید از کلاس `Customer` در برنامه جدید استفاده کنید. البته نمی توان دلایل مشخص و قاطعی را بیان کرد و گفت که چه زمانی شرایطی پیش می آیند که نمی توان از یک کلاس در برنامه های دیگر استفاده کرد. ولی ممکن است برای اینکه بتوانید از کلاس `Customer` در چند برنامه استفاده کنید، مجبور شوید آن را به نحوی تغییر دهید که بسیار پیچیده شود. در این شرایط ایجاد چند کلاس ساده و استفاده از آنها در برنامه های مخصوص به خود، راحت تر از ایجاد یک کلاس پیچیده و استفاده از آن در چند برنامه است. فهمیدن اینکه یک کلاس را چگونه باید نوشت تا هم سادگی در آن رعایت شود و هم قابلیت استفاده مجدد، به تجربه در برنامه نویسی شیئ گرا بستگی دارد. هرچه برنامه های بیشتری بنویسید، بهتر می توانید کلاس هایی طراحی کنید که این دو فاکتور در آنها به بهترین نحو رعایت شوند.

طراحی یک شیئ:

بر خلاف مطالبی که از ابتدای فصل تاکنون بررسی کرده ایم، به عنوان اولین پروژه این فصل نمی خواهیم یک الگوریتم تعریف کرده و سپس اشیای مورد نیاز آن را ایجاد کنیم. بلکه سعی می کنیم با ارائه یک مثال، کاربرد مطالبی که در قسمتهای قبل به صورت تئوری با آنها آشنا شدید را در عمل مشاهده کنید. مثالی که در این قسمت بررسی خواهیم کرد، طراحی یک کلاس برای اتومبیل است.

در مورد چنین کلاسی، اصول مشخصی هستند که در ابتدا باید آنها را بدانید:

- **ظاهر آن چگونه است:** ظاهر یک اتومبیل شامل مواردی مانند مدل، رنگ، تعداد درها و مواردی مشابه است. این گونه ویژگیهای یک اتومبیل هنگام ایجاد شدن آن تنظیم می شود و معمولاً تا انتهای عمر شیئ نیز ثابت است.
- **توانایی های آن چیست:** قدرت و اندازه موتور، ترکیب و تعداد سیلندرها و غیره.
- **وضعیت کنونی آن چیست:** اتومبیل ثابت است، به جلو حرکت می کند و یا به عقب حرکت می کند، سرعت و جهت آن چقدر است؟
- **موقعیت مکانی آن چیست:** شیئ اتومبیل معمولا دارای شیئ دیگری از نوع GPS[1] است که موقعیت مکانی آن را نسبت به دیگر اشیا (مثلاً اتومبیل های دیگر) نمایش می دهد.

علاوه بر این می خواهید اشیایی که از این کلاس نمونه سازی می شوند، قابل کنترل نیز باشند. برای مثال:

- بتوانید سرعت آن را کاهش دهید.
- بتوانید سرعت آن را افزایش دهید.
- بتوانید جهت حرکت آن را به سمت چپ تغییر دهید.
- بتوانید جهت حرکت آن را به سمت راست تغییر دهید.
- بتوانید آن را متوقف کنید.

[1] Global Positioning System

همانطور که در ابتدای فصل گفتم، ابتدا باید سه مورد را در رابطه با این کلاس مشخص کنید: هویت، حالت و رفتار. فرض می کنیم که جنبه هویت در نظر گرفته شده است و اشیای این کلاس می دانند که از نوع اتومبیل هستند. بنابراین به بررسی دو جنبه دیگر، یعنی حالت و رفتار می پردازیم.

حالت:

حالت، وضعیت کنونی یک شیئ از کلاس را توصیف می کند. برای مثال، موقعیت و سرعت اتومبیل بخشی از حالت آن به شمار می روند. هنگام طراحی یک کلاس، ابتدا باید مشخص کنید که برای دانستن حالت یک شیئ چه مواردی را باید در نظر بگیرید. هنگام طراحی کلاس اتومبیل سرعت آن در تعیین حالت آن اهمیت زیادی دارد، اما برای طراحی کلاس `Customer`، سرعت مفهومی ندارد بلکه آدرس کاربر یکی از مهمترین عوامل در حالت کنونی آن محسوب می شود.
حالت یک شیئ، معمولا به صورت مقادیری درون آن نگهداری می شوند. بعضی از این مقادیر به صورت عمومی (`Public`) در اختیار کاربران قرار داده می شوند و کاربران می توانند به وسیله خصوصیات کلاس آنها را تغییر دهند، بعضی دیگر نیز به صورت خصوصی (`Private`) هستند و فقط به وسیله اشیای آن کلاس می توانند استفاده شوند. همچنین بعضی از حالتهای یک کلاس نیز، فقط قابل خواندن هستند و کاربر نمی تواند به صورت مستقیم آنها را تغییر دهد. برای مثال اتومبیل خاصیتی به نام سرعت سنج دارد که سرعت شیئ را در هر لحظه مشخص می کند. کاربر اتومبیل فقط می تواند این سرعت را بخواند، اما نمی تواند آن را به طور مستقیم تغییر دهد – برای تغییر آن باید به وسیله متدهایی مانند `Accelerate` و یا `Break` باعث افزایش و یا کاهش سرعت شود.

رفتار:

به عکس العمل یک شیئ از کلاس در مقابل درخواستهای کاربر، رفتار آن شیئ می گویند. زمانی که متدی را از یک شیئ فراخوانی می کنید، در حقیقت از شیئ می خواهید که وظیفه ای را برای شما انجام دهد و آن شیئ نیز در مقابل این درخواست عکس العملی را نشان می دهد. بنابراین رفتار یک شیئ معمولاً به متدها مربوط است. البته رفتار می تواند به خاصیتهای یک شیئ نیز مربوط باشد. در مثال تلویزیون که در قسمت قبلی مشاهده کردید، اگر خاصیت کانال آن را برابر با عدد خاصی قرار دهید، باعث می شوید که آن شیئ با تغییر تصویر صفحه نمایش، رفتار خود را نمایش دهد.
رفتار یک شیئ از کلاس معمولا به صورت چندین خط کد است که وظیفه خاصی را انجام می دهد. این کدها معمولا یک و یا هر دوی موارد زیر را دربر می گیرند:

- **تغییر حالت خود شیئ:** هنگامی که متد `Accelerate` را برای شیئ از کلاس اتومبیل فراخوانی می کنید، باعث می شوید که آن شیئ سریعتر حرکت کند. بنابراین حالت آن را تغییر می دهید.
- **اثر گذاشتن روی دنیای خارج از شیئ:** این مورد می تواند شامل تغییر دادن اشیای دیگر در برنامه، نمایش مطلبی روی صفحه نمایش، ذخیره اطلاعات روی دیسک، یا چاپ یک سند از برنامه باشد.

در بخش امتحان کنید بعد، یک پروژه جدید ایجاد کرده و کلاس `Car` را در آن به وجود می آوریم.

امتحان کنید: ایجاد پروژه جدید و کلاس Car

۱) ویژوال استودیو ۲۰۰۵ را اجرا کرده و از نوار منو گزینه File ➔ New ➔ Project... را انتخاب کنید.
۲) هنگامی که کادر New Project نمایش داده شد، از قسمت Templates گزینه Console Application را انتخاب کرده، نام Objects را در کادر Name وارد کنید. سپس روی دکمه OK کلیک کنید تا پروژه ایجاد شود.
۳) حال باید یک کلاس جدید به این پروژه اضافه کنید. در پنجره Solution Explorer روی نام پروژه کلیک راست کرده و گزینه Add ➔ Class را انتخاب کنید. در کادر Add New Item – Objects نام Car.cs را به عنوان نام کلاس مشخص کرده و روی دکمه Add کلیک کنید. به این ترتیب کلاس جدید ایجاد شده و به Solution Explorer نیز اضافه می شود.

نگهداری حالت:

تاکنون متوجه شدیم حالت یک شیئ مشخص می کند که آن شیئ در رابطه با خود چه چیزهایی را می داند. اما سوالی که در اینجا پیش می آید این است که چگونه حالت یک شیئ را در آن نگهداری کنیم؟ خوب، عموماً برای این کار، متغیرهایی را درون کلاس تعریف می کنند و سپس حالت شیئ را در آنها نگهداری می کنند. در برنامه نویسی شیئ گرا به این متغیر ها **فیلد**[1] گفته می شود.
معمولا متدها و خاصیت هایی که در یک کلاس استفاده می کنید، یا حالت شیئ را تغییر می دهند و یا از آن برای انجام وظیفه خود استفاده می کنند. فرض کنید می خواهید خاصیتی ایجاد کنید که رنگ یک شیئ از کلاس Car را مشخص کند. بنابراین هنگامی که بخواهید این خاصیت را تغییر دهید، فیلد این حالت تغییر کرده و مقدار جدید رنگ را در خود ذخیره می کند. همچنین اگر بخواهید مقدار این خاصیت را بدست آورید، فیلد این حالت خوانده خواهد شد و مقدار آن به عنوان رنگ شیئ به شما برگردانده می شود.
از یک جهت، می توان گفت که خاصیت ها نیز مانند متدها باعث بروز رفتار از شیئ می شوند. هر خاصیت، معمولاً از دو متد تشکیل شده است: متد get و متد set (که به وسیله بلاکهای { ... }get و {...}set مشخص می شوند). یک متد ساده get برای خاصیت Color در کلاس Car، فقط شامل کدی است که با توجه به مقدار ذخیره شده در فیلد مربوط به این حالت، رنگ شیئ را به کاربر اعلام می کند، همچنین یک متد ساده set برای خاصیت Color فقط شامل کدی است که مقدار این فیلد را با توجه به مقدار مورد نظر کاربر تنظیم می کند.
اما در برنامه های واقعی این متدها در خاصیت Color به این سادگی نیستند. برای مثال فرض کنید می خواهید از کلاس Car در یک بازی اتومبیل رانی استفاده کنید. در این صورت هنگامی که کاربر خاصیت Color یک شیئ از کلاس Car را تغییر داد، متد set از این خاصیت باید علاوه بر ذخیره رنگ در فیلد مربوطه، رنگ اتومبیلی که در صفحه نمایش داده شده است را نیز تغییر دهد.
در بخش امتحان کنید بعد، برای ایجاد خاصیت Color در کلاس Car، یک فیلد به نام Color و از نوع public (تا به وسیله کاربر نیز قابل دسترسی باشد) تعریف می کنیم. دقت کنید با توجه به این که در این مثال از فیلد به جای خاصیت استفاده کرده ایم، اما در برنامه های واقعی هیچگاه نباید به جای اینکه از یک خاصیت با متدهای get و set ایجاد کنید، از فیلد استفاده کنید.

[1] Field

امتحان کنید: نمونه سازی یک شیئ و اضافه کردن خاصیت Color

۱) کد زیر را به کلاس Car اضافه کنید:

```
public string Color;
```

۲) کد لازم برای اضافه کردن یک فیلد به کلاس، همین بود! حال باید به نحوی از کلاسی که ایجاد کرده ایم استفاده کنیم تا عملکرد آن را ببینیم. با استفاده از پنجره Solution Explorer، فایل Program.cs را باز کرده و کد زیر را به آن اضافه کنید:

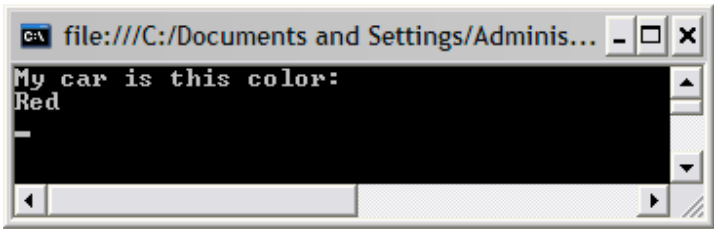
```
static void Main(string[] args)
{
    // Create a new Car
    Car objCar = new Car();

    // Set the Color property to Red
    objCar.Color = "Red";

    // Show what the value of the property is
    Console.WriteLine("My car is this color: ");
    Console.WriteLine(objCar.Color);

    // Wait for input from the user
    Console.ReadLine();
}
```

۳) برنامه را اجرا کنید. پنجره جدیدی مشابه شکل ۹-۱ نمایش داده خواهد شد.



شکل ۹-۱

۴) برای اتمام برنامه کلید Enter را از صفحه کلید فشار دهید.

چگونه کار می کند؟

تعریف یک فیلد بسیار راحت است. کد:

```
public string Color;
```

به کلاس می گوید که می خواهید یک فیلد به نام Color ایجاد کنید و در آن رشته ای از کاراکتر ها را نگهداری کنید. کلمه کلیدی public در ابتدای این فیلد به کلاس می گوید که می خواهید این فیلد به وسیله تمام برنامه نویسانی که بخواهند از این کلاس استفاده کنند قابل دسترسی باشد.

نکته: متغیرهایی که در بدنه یک کلاس تعریف می شوند (یعنی درون خود کلاس تعریف شده اند، نه در متدهای موجود در آن کلاس)، از دید خود کلاس، عضو داده ای و از دید کسانی که از کلاس استفاده می کنند فیلد نامیده می شوند.

همانطور که در کد نوشته شده در فایل Program.cs مشاهده می کنید، استفاده از یک کلاس بسیار راحت است. این پروسه از دو مرحله تشکیل می شود. ابتدا باید متغیری را تعریف کنید که بتواند شیئ از آن کلاس را در خود نگهداری کند؛ سپس آن شیئ را نمونه سازی می کنید. کد زیر متغیری را به نام objCar تعریف می کند و به آن می گوید که فقط باید اشیایی از کلاس Car را در خود نگهداری کند.

```
Car objCar;
```

بعد از تعریف objCar، این متغییر شامل هیچ شیئ از کلاس Car نیست، زیرا فقط نوع شیئ که باید در این متغییر نگهداری شود را مشخص کرده اید. این خط کد مانند این است که به کامپیوتر بگویید قلّابی به شما بدهد که به وسیله آن بتوانید یک شیئ از کلاس Car را آویزان کنید، و سپس نام آن قلّاب را objCar قرار دهید. بنابراین هنوز هیچ چیز به این قلّاب آویزان نکرده اید. برای این کار باید یک شیئ از کلاس Car را نمونه سازی کنید. این عمل به وسیله کلمه کلیدی new انجام می شود:

```
objCar = new Car();
```

اما همانطور که در فصلهای قبلی نیز دیدید، می توان این دو مرحله را در یک خط انجام داد:

```
Car objCar = new Car();
```

بنابراین در این خط به ویژوال C# می گویید که "objCar را به شیئ که جدیداً از کلاس Car نمونه سازی شده است ارجاع بده". به عبارت دیگر "یک شیئ جدید از کلاس Car نمونه سازی کرده و آن را از قلّابی به نام objCar آویزان کن".

نکته: توجه کنید که در برنامه نویسی شیئ گرا، یک شیئ می تواند در یک لحظه از چندین قلّاب آویزان شود، و بنابراین دارای چندین نام باشد. این مورد ممکن است کمی گیج کننده به نظر برسد، اما در بسیاری از موارد باعث راحتی کارها می شود. تصور کنید که می توانستید کلیدهای خود را در یک زمان از چند جا آویزان کنید – به این ترتیب پیدا کردن آنها بسیار راحت تر بود!

بعد از اینکه یک نمونه از شیئ ایجاد کردید، می توانید خاصیت های آن را تنظیم و یا متـدهای آن را فراخـوانی کنیـد. بـرای تنظـیم خاصیت `Color`[1] شیئ جدید می توانید از کد زیر استفاده کنید:

```
// Set the Color property to Red
objCar.Color = "Red";
```

هنگامی که مقدار یک خاصیت را تنظیم کردید، می توانید هر چند بار که نیاز داشته باشید به آن دسترسی پیدا کنید و یا مجدداً مقـدار آن را تغییر دهید. در اینجا، نحوه دسترسی به مقدار یک خاصـیت را بـا ارسـال خاصـیت `Color` بـه متـد `WriteLine` از کلاس `Console` نمایش داده ایم:

```
// Show what the value of the property is
Console.WriteLine("My car is this color: ");
Console.WriteLine(objCar.Color);
```

خط `Console.ReadLine` به این خاطر است که برنامه بعد از اتمام کار صبر کند تا کاربر کلید `Enter` را فشار دهد، سپس بسته شود. به این ترتیب پنجره کنسول قبل از بسته شدن صبر می کند تا کلید `Enter` را فشار دهید.

```
// Wait for input from the user
Console.ReadLine();
```

نکته: برنامه های `Console` در `.NET`، یک روش خوب برای تست عملکرد کلاس هایی هستند که نمود ظاهری ندارنـد و فقط در حافظه کار می کنند، زیرا با استفاده از آنها نیازی نیست که به ایجاد یک رابط گرافیکی توجه کنید. به این ترتیب می توانید با نمایش چند خط متن در مواقع مورد نیاز از وضعیت شیئ مطلع شوید.

خاصیت های فقط-خواندنی:

در قسمت قبل با نحوه ایجاد یک عضو داده ای (یا همان فیلد) برای کلاس آشنا شدید و مشاهده کردید که چگونه مـی تـوان از آن استفاده کرد. اما همانطور که گفتم هیچگاه نباید اجازه دهید که کاربر به طور مستقیم مقدار موجود در یک فیلـد را تغییـر دهـد، بلکـه همواره باید یک خاصیت ایجاد کنید تا کاربر به وسیله آن خاصیت مقدار فیلد را تغییر دهد و یا به آن دسترسی پیدا کند. با وجود اینکه خاصیت ها به وسیله دو متد تعریف می شوند، اما نحوه استفاده از آنها برای کاربر، دقیقاً مشابه فیلد ها اسـت. در حقیقـت مـی تـوانیم بگوییم خاصیت ها، متدهایی هستند که کاربر می تواند مشابه یک فیلد با آنها کار کند. به این ترتیب، انتخاب این که برای یک مورد خاص از متد استفاده کنید و یا از خاصیت، به این بستگی دارد که کاربر با کدامیک می تواند راحت تر کار کند.
یکی از مشکلاتی که در صورت دسترسی مستقیم کاربر به فیلد ممکن است به وجود آید، این است که در این صورت کاربر می تواند هم مقدار موجود در فیلد را بخواند و هم آن را تغییر دهد. در شرایطی ممکن است نخواهید اجازه دهید که کاربر فیلـد را تغییـر دهـد، بلکه می خواهید فیلد به صورت فقط-خواندنی باشد و کاربر فقط بتواند اطلاعات آن را بخواند.

---

[1] در کلاس `Car`، `Color` یک فیلد است نه یک خاصیت. البته استفاده از خاصیت نیز مانند استفاده از فیلد است و فقط نحوه تعریف آن در کـلاس تفـاوت دارد. در این کلاس برای ساده گی، `Color` را به صورت فیلد تعریف کرده ایم، ولی برای اینکه می خـواهیم خاصـیت هـا را بررسـی کنـیم، از آن بـه عنـوان خاصیت یاد می کنیم.

سرعت اتومبیل یک نمونه خوب برای این است که مشخص شود چگونه مدلی از یک شیئ واقعی در کامپیوتر، باید دقیقاً مشابه همان شیئ عمل کند. در یک اتومبیل واقعی اگر در حال حرکت با سرعت ۶۰ کیلومتر در ساعت هستید، نمی توانید به سادگی سرعت را به هر عددی که بخواهید تغییر دهید. با استفاده از سرعت سنج فقط می توانید عدد سرعت را بخوانید، اما نمی توانید با انگشت عقربه ی سرعت سنج را جا به جا کنید تا سرعت اتومبیل تغییر کند. برای تغییر سرعت باید از پدال گاز و یا پدال ترمز برای افزایش و یا کاهش سرعت اتومبیل استفاده کنید. برای مدل کردن عملکرد این پدالها در کلاس Car، باید متدهایی ایجاد کنید که سرعت را تغییر دهند (Accelerate، Decelerate)، همچنین یک خاصیت فقط–خواندنی نیز به نام Speed ایجاد کنید تا سرعت کنونی اتومبیل را نمایش دهد.

بنابراین بهتر است برای جلوگیری از این مشکل و همچنین مشکلات مشابه، از یک خاصیت برای دریافت مقدار این فیلد استفاده کنیم تا بتوانیم قبل از قرار دادن مقدار مورد نظر در فیلد، از درست بودن آن مطمئن شویم.

البته برای نگهداری مقدار سرعت مسلماً به یک عضو داده ای در کلاس نیاز دارید، اما این عضو داده ای باید فقط بتواند توسط اعضای کلاس مورد استفاده قرار گیرد و یا تغییر داده شود. به همین دلیل برای تعریف آن از کلمه کلیدی private استفاده می کنیم:

```
private int _speed;
```

متغییر _speed در این قسمت به صورت private تعریف شده است، بنابراین فقط می تواند توسط متدهایی که در داخل کلاس وجود دارند مورد استفاده قرار بگیرد. کاربران کلاس Car، حتی از وجود چنین عضوی نیز مطلع نخواهند بود. همچنین در کلاس خاصیتی به نام Speed تعریف می کنیم که کاربر به وسیله آن بتواند از مقدار این متغییر که نشان دهنده سرعت شیئ است مطلع شود. در این کلاس نام فیلد مربوط به سرعت و همچنین خاصیت آن یکی است، برای جلوگیری از این تشابه معمولاً در ابتدای نام فیلد، یک زیر خط اضافه می کنند. بنابراین نام فیلد، _speed و نام خاصیت، Speed خواهد شد.

امتحان کنید: اضافه کردن خاصیت Speed

۱) برای تعریف متغیری که فقط به وسیله اعضای کلاس قابل دسترسی باشد، به جای کلمه کلیدی public از private استفاده می کنیم. کد زیر را به کلاس Car اضافه کنید:

```
private int _speed;
```

۲) برای اینکه کاربر بتواند از اندازه سرعت مطلع شود، باید یک خاصیت فقط–خواندنی به کلاس اضافه کنیم. کد زیر را به کلاس Car اضافه کنید:

```
    // Speed - Read-Only property to return the speed
        public int Speed
            {
              get
                {
            return _speed;
                }
            }
```

۳) حال متدی را به نام Accelerate برای تنظیم سرعت اتومبیل ایجاد می کنیم. ایـن متـد مقـداری را بـر حـسب کیلومتر بر ساعت به عنوان پارامتر دریافت می کند و سرعت شیئ را برابر آن قرار مـی دهـد. کـد زیـر را بعـد از خاصـیت Speed وارد کنید:

```
        // Accelerate - Add kmph to the speed
    public void Accelerate(int accelerateBy)
                                            {
                    // Adjust the speed
                _speed += accelerateBy;
                                            }
```

۴) برای تست کلاس، باید تغییراتی را در زیر برنامه Main فایل Program.cs ایجاد کنید. فایل را باز کنید و کد آن را به صورت زیر تغییر دهید:

```
static void Main(string[] args)
                                    {
                    // Create a new Car
                Car objCar = new Car();

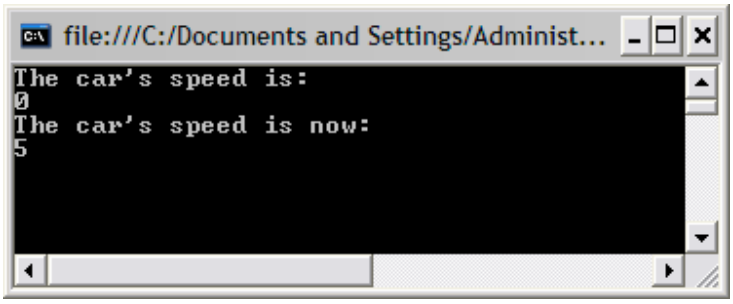
                    // Report the speed
        Console.WriteLine("The car's speed is: ");
                Console.WriteLine(objCar.Speed);

                        // Accelerate
                objCar.Accelerate(5);

                    // Report the new speed
        Console.WriteLine("The car's speed is now: ");
                Console.WriteLine(objCar.Speed);

            // Wait for input from the user
                    Console.ReadLine();
                                    }
```

۵) با کلیک روی دکمه Start در نوار ابزار برنامه را اجرا کنید. پنجره ای مشابه شکل ۹-۲ نمایش داده خواهد شد:



شکل ۹-۲

چگونه کار می کند؟

اولین کاری که باید انجام دهید این است که در کلاس `Car` یک عضو داده ای از نوع `private` به نـام `_speed` ایجـاد کنید:

```
private int _speed;
```

به صورت پیش فرض، هنگامی که شیئ از این کلاس ساخته شود، مقدار `_speed` در آن شیئ برابر با صفر خواهد بود زیرا مقدار پیش فرض متغییر های نوع داده ای `int` برابر با صفر است.
سپس خاصیتی برای برگرداندن مقدار سرعت تعریف می کنیم:

```
// Speed - Read-Only property to return the speed
public int Speed
{
get
{
return _speed;
}
}
```

هنگام تعریف یک خاصیت می توانید آن را به صورت فقط-خواندنی، فقط نوشتنی، و یا خواندنی-نوشتنی مشخص کنید. همانطور که می دانید هنگامی که کاربر بخواهد به مقدار یک خاصیت دسترسی داشته باشد، کدهای نوشته شده در بلاک `get`، و اگـر بخواهـد مقدار خاصیت را تغییر دهد کدهای بلاک `set` اجرا می شوند. پس اگر هنگام ایجاد یک خاصیت، بـلاک `get` را در آن ننویـسید آن خاصیت قابل خواندن نخواهد بود و به یک خاصیت فقط-نوشتنی تبدیل می شود. به همین ترتیـب اگـر بـلاک `set` را از یـک خاصیت حذف کنید، خاصیت قابل نوشتن نخواهد بود و به خاصیت فقط-خواندنی تبدیل می شود. در صـورتی کـه در خاصـیت هـم بلاک `get` و هم بلاک `set` وجود داشته باشد، خاصیت خواندنی-نوشتنی خواهد بود.
بعد از ایجاد خاصیت `Speed`، متدی به نام `Accelerate` ایجاد کرده ایم. این متد مقداری را برنمی گرداند، بنابراین نـوع برگشتی آن را `void` قرار می دهیم.

```
// Accelerate - Add kmph to the speed
```

```
public void Accelerate(int accelerateBy)
                                        {
                                            // Adjust the speed
                                        _speed += accelerateBy;
                                        }
```

این متد یک پارامتر به نام accelerateBy دریافت می کند که برای مشخص کردن مقدار افزایش سرعت به کار می رود. توجه کنید تنها کاری که این تابع انجام می دهد این است که مقدار دریافتی را به فیلد _speed اضافه می کند. در دنیای واقعی فشار روی پدال گاز، همراه با فاکتورهای دیگری مانند سرعت باد و یا اصطکاک با سطح زمین، سرعت جدید اتومبیل تعیین می شود. به عبارت دیگر سرعت نتیجه اثر چند فاکتور بر یکدیگر است، نه فقط تغییر دادن یک عدد. برای شبیه سازی واقعی این قسمت باید کدهای پیچیده تری نوشت. اما در اینجا برای اینکه مثال همچنان ساده باقی بماند، مقدار معین شده به وسیله کاربر را با سرعت کنونی جمع می کنیم.

سرعت بخشیدن به اتومبیل نمونه دیگری از کپسولی بودن است. برای شتاب دادن به یک اتومبیل در دنیای واقعی، سیستمهای زیادی با یکدیگر فعالیت می کنند تا اتومبیل به سرعت مورد نظر کاربر برسد. اما راننده به عنوان کسی که در حال استفاده از این شیئ است، هیچ اطلاعی از این سیستم ها ندارد. در این مثال هم به همین صورت است. فردی که در حال استفاده از متد Accelerate است، در مورد اینکه این متد چگونه سرعت را افزایش می دهد هیچ اطلاعی ندارد و فقط می داند که برای افزایش سرعت باید از آن استفاده کند.

استفاده از قسمتهای جدید بسیار ساده است. ابتدا شیئ از این کلاس را همانند قسمت قبل نمونه سازی می کنید:

```
// Create a new Car
Car objCar = new Car();
```

سپس سرعت اولیه را در صفحه نمایش می دهید:

```
// Report the speed
Console.WriteLine("The car's speed is: ");
Console.WriteLine(objCar.Speed);
```

سپس با استفاده از متد Accelerate، سرعت شیئ را افزایش می دهید:

```
// Accelerate
objCar.Accelerate(5);
```

در انتها نیز سرعت جدید را اعلام می کنید:

```
// Report the new speed
Console.WriteLine("The car's speed is now: ");
Console.WriteLine(objCar.Speed);
```

خاصیتهای خواندنی-نوشتنی:

تاکنون متوجه شدیم که یکی از دلایل برتری استفاده از خاصیت ها به جای فیلد ها، در این است که با استفاده از خاصیت ها می توان از تغییر دادن مستقیم فیلد توسط کاربر جلوگیری کرد، همانند خاصیت Speed که در قسمت قبل به صورت فقط-خواندنی تعریف شد. اما در اینجا ممکن است سوال کنید برای مواردی که کاربر هم می تواند مقدار یک فیلد را بخواند و هم آن را تغییر دهد، چرا باید به جای استفاده از فیلد، از خاصیت استفاده کنیم؟
خوب، اگر به جای استفاده از فیلد از خاصیت استفاده کنید، می توانید کدهایی را مشخص کنید تا هنگام خوانده شدن و یا نوشته شدن فیلد توسط کاربر اجرا شوند، و این مورد از اهمیت زیادی برخوردار است.
برای مثال با استفاده از خاصیت ها می توانید قبل از اینکه مقداری به یک فیلد اختصاص داده شود، از درست بودن آن مطمئن شوید. برای مثال تصور کنید که می خواهید فیلدی به نام NumberOfDoors، برای مشخص کردن تعداد درهای یک اتومبیل در کلاس Car قرار دهید. نوع داده ای این فیلد از کلاس باید از نوع عدد صحیح باشد تا بتواند تعداد درهای یک اتومبیل را در خود نگه دارد. اما مسلماً نمی خواهید که کاربر بتواند عدد ۰ و یا عدد ۶۵۵۰۰ را در این فیلد وارد کند. به عبارت دیگر می خواهید عدد وارد شده توسط کاربر در بازه ۲ تا ۶ باشد.

نکته: ممکن است بگویید که این کلاس توسط برنامه نویس دیگری مورد استفاده قرار می گیرد، بنابراین وظیفه اوست که هنگامی که بخواهد مقداری را در این فیلد قرار دهد از درست بودن آن مطمئن شود. در طراحی یک کلاس، وظیفه برنامه نویس است که تا حد ممکن کار را برای افرادی که می خواهند از کلاس استفاده کنند ساده کند. رسیدگی کردن به صحت داده های ورودی، یکی از مهمترین جنبه های طراحی یک کلاس محسوب می شود.

همچنین همواره خاصیت ها مقدارهای ذخیره شده در یک فیلد را برنمی گردانند، بلکه ممکن است مقداری را از جای دیگری بدست آورند و یا آن را بر اساس یک سری اطلاعات محاسبه کرده و برگردانند. برای مثال تصور کنید بخواهید تعداد کل سفارشات یک مشتری را به عنوان یک خاصیت در کلاس Customer قرار دهید. همانطور که می دانید هنگام طراحی یک کلاس برای Customer، فیلدی برای نگهداری این عدد مشخص نمی شود، بلکه یک شیئ باید در صورت لزوم، آن را محاسبه کند. در این صورت می توانید در قسمت get یک خاصیت، کدی را بنویسید که با توجه به لیست سفارشات یک مشتری در یک بانک اطلاعاتی، تعداد کل آنها را محاسبه کرده و برگرداند. این موارد در بخشهای بعدی مورد بررسی قرار می گیرند، بنابراین بهتر است به مسئله تعداد درهای اتومبیل برگردیم.

امتحان کنید: اضافه کردن خاصیت NumberOfDoors

۱) اولین کاری که باید انجام دهید این است که فیلدی برای نگهداری تعداد درهای یک شیئ از کلاس Car را، در آن ایجاد کنید. به صورت پیش فرض در نظر می گیریم که مقدار این خاصیت برابر با ۴ است. بنابراین کد مشخص شده در زیر را به کلاس اضافه کنید:

```
public string Color;
private int _speed;
private int _numberOfDoors = 4;
```

۲) حال می توانید خاصیتی برای تنظیم و یا دسترسی به تعداد در های اتومبیل ایجاد کنید و همواره بررسی کنید که عدد موجود برای این فیلد بین ۲ تا ۶ باشد. کد زیر را بعد از متد `Accelerate` در کلاس `Car` وارد کنید:

```
        // NumberOfDoors - get/set the number of doors
        public int NumberOfDoors
        {
            // Called when the property is read
            get
            {
                return _numberOfDoors;
            }
            // Called when the property is set
            set
            {
                // Is the new value between two and six
                if (value >= 2 && value <= 6)
                {
                    _numberOfDoors = value;
                }
            }
        }
```

نکته: در این فصل از ایجاد خطا هنگامی که عددی خارج از محدوده مورد نظر وارد شد صرفنظر می کنیم. اما اصولا هنگامی که یک عدد نا معتبر در کلاس وارد شد، باید یک خطا ایجاد کنید. به این ترتیب فردی که در حال استفاده از کلاس است متوجه رخ دادن خطا شده و در مورد چگونگی برخورد با این خطا تصمیم گیری می کند. در مورد چگونگی ایجاد این نوع خطاها در فصل ۱۱ صحبت خواهیم کرد.

۳) برای بررسی تغییراتی که در کلاس ایجاد کرده اید، باید زیر برنامه `Main` در `Program.cs` را به صورت زیر تغییر دهید:

```
        static void Main(string[] args)
        {
            // Create a new Car
            Car objCar = new Car();

            // Report the number of doors
            Console.WriteLine("The number of doors is: ");
            Console.WriteLine(objCar.NumberOfDoors);

            // Try Changing the number of doors to 1000
            objCar.NumberOfDoors = 1000;

            // Report the number of doors
            Console.WriteLine("The number of doors is: ");
```

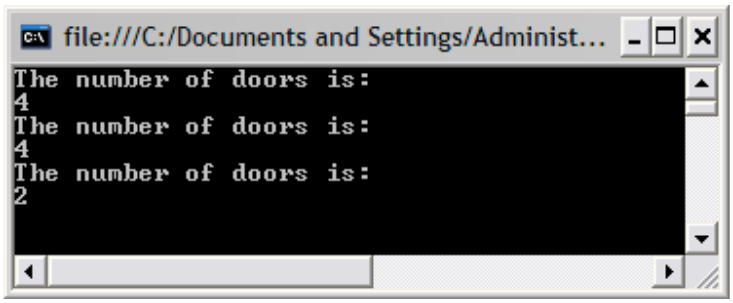
```
            Console.WriteLine(objCar.NumberOfDoors);

            // Now try changing the number of doors to 2
            objCar.NumberOfDoors = 2;

            // Report the number of doors
            Console.WriteLine("The number of doors is: ");
            Console.WriteLine(objCar.NumberOfDoors);

            // Wait for input from the user
            Console.ReadLine();
        }
```

حال برنامه را اجرا کنید. صفحه ای مشابه شکل ۹-۳ مشاهده خواهید کرد.



شکل ۹-۳

چگونه کار می کند؟

ابتدا یک فیلد برای نگهداری تعداد در ها به صورت `private` تعریف می کنیم. همچنین مقدار پیش فرض این فیلد را نیز ۴ در نظر می گیریم.

```
private int _numberOfDoors = 4;
```

دلیل اینکه در این مرحله به این فیلد عدد داده ایم نیز مشخص است. همانطور که گفتم مـی خـواهیم تعـداد درهـای یـک اتومبیـل همواره بین ۲ تا ۶ باشد. همچنین می دانیم که یک متغییر از نوع عدد صحیح به صورت پیش فرض دارای مقدار ۰ است. بنابراین اگر در این قسمت به این فیلد مقدار ندهیم، هنگامی که شیئ ایجاد می شود، تعداد در ها به صورت پیش فرض برابر با صفر خواهد بود.
بعد از تعریف فیلد نوبت به خود خاصیت می رسد. بخش `get` که همانند قسمت قبل است و نکته جدیدی ندارد – فقط کافی است مقدار فیلد `_numberOfDoors` را برگرداند. اما در بلاک `set` ابتدا باید بررسی کنیم عددی که کـاربر بـه ایـن خاصـیت فرستاده است معتبر باشد، سپس آن را در `_numberOfDoors` قرار دهیم (مقداری که به وسیله کاربر به خاصیت فرسـتاده می شود با کلمه کلیدی `value` قابل دسترسی است):

```
// NumberOfDoors - get/set the number of doors
```

```
        public int NumberOfDoors
        {
            // Called when the property is read
            get
            {
                return _numberOfDoors;
            }
            // Called when the property is set
            set
            {
                // Is the new value between two and six
                if (value >= 2 && value <= 6)
                {
                    _numberOfDoors = value;
                }
            }
        }
```

بقیه کدی هم که به فایل `Program.cs` اضافه کرده اید، مورد پیچیده ای نیست. ابتدا مقدار اولیه فیلد `_numberOfDoors` را نمایش می دهید، سپس سعی می کنید این مقدار را به ۱۰۰۰ تغییر دهید. در این هنگام کدی که برای تعیین صحت داده ها در خاصیت `NumberOfDoors` وارد شده است، اجازه نمی دهد که مقدار خاصیت به ۱۰۰۰ تغییر کند. بنابراین مقدار فیلد `_numberOfDoors` همچنان برابر با ۴ باقی می ماند. در انتها نیز مقدار خاصیت را برابر با یک مقدار منطقی مانند ۲ قرار می دهیم و مشاهده می کنیم که تعداد درها تغییر می کند.

نکته: اگرچه ممکن است روش کارکرد خاصیتهای خواندنی-نوشتنی و همچنین فیلد های `public` مانند هم به نظر رسد، اما با هم تفاوت زیادی دارند. زمانی که ویژوال C# ۲۰۰۵ بخواهد کد برنامه شما را کامپایل کند، با قسمتهایی که کاربر از خاصیت استفاده کرده است، همانند فراخوانی متد رفتار می کند. توجه داشته باشید که استفاده از خاصیت ها به جای فیلد های `public` باعث می شود که کد برنامه شما انعطاف پذیر تر شده و قابلیت گسترش بیشتری داشته باشد.

متد IsMoving:

هنگامی که در حال طراحی یک کلاس هستید، باید همواره این سوال را در نظر داشته باشید که "چگونه می توانم استفاده از این کلاس را ساده تر کنم؟". برای مثال اگر کاربر بخواهد تشخیص دهد آیا این اتومبیل در حال حرکت است یا نه، چگونه می تواند این کار را انجام دهد؟

یک راه برای انجام این کار، بررسی خاصیت `Speed` است. اگر مقدار برگشتی توسط این خاصیت برابر با صفر باشد، می توان فهمید که اتومبیل توقف کرده است. اگر بررسی این مورد را به کاربر واگذار کنیم، کاربر نیز بر اساس برداشت خود از نحوه کارکرد کلاس برای نتیجه گیری استفاده می کند و ممکن است او از این روش برای بررسی توقف اتومبیل استفاده نکند. البته در این مورد واضح است که همواره سرعت صفر برابر با توقف اتومبیل است، اما در مواردی مشابه حتی اگر ۹۹٪ افرادی که از کلاس استفاده می کنند در یک مورد اشتراک نظر داشته باشند، باید برای وضوح بیشتر کار با کلاس متدی طراحی کرد که به آن مورد پاسخ دهد.
بنابراین در مثال بالا، بهتر است متدی طراحی کنیم که مشخص کند آیا اتومبیل توقف کرده و یا در حال حرکت است.

امتحان کنید: اضافه کردن متد IsMoving

۱) تمام کاری که متد IsMoving باید انجام دهد این است که پس از بررسی سرعت اتومبیل، مقداری را از نوع Boolean برگرداند تا مشخص شود اتومبیل توقف کرده و یا در حال حرکت است. کد زیر را در کلاس Car، بعد از خاصیت NumberOfDoors وارد کنید:

```
        // IsMoving - is the car moving?
        public Boolean IsMoving()
        {
            // Is the car's speed zero?
            if (Speed == 0)
                return false;
            return true;
        }
```

۲) برای تست این متد، تغییرات مشخص شده در زیر را در زیر برنامه Main فایل Program.cs ایجاد کنید:

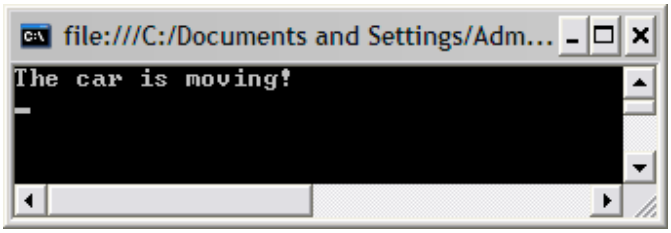
```
        static void Main(string[] args)
        {
            // Create a new Car
            Car objCar = new Car();

            // Accelerate the car to 25kmph
            objCar.Accelerate(25);

            // Report whether or not the car is moving
            if (objCar.IsMoving() == true)
            {
                Console.WriteLine("The car is moving!");
            }
            else
            {
                Console.WriteLine("The car is stopped!");
            }

            // Wait for input from the user
            Console.ReadLine();
        }
```

۳) حال برنامه را اجرا کنید. پنجره جدیدی را مشابه شکل ۹-۴ مشاهده خواهید کرد.



شکل ۹-۴

چگونه کار می کند؟

در این قسمت فقط متد ساده ای را به کلاس اضافه کرده ایم که با توجه به خاصیت Speed، در صورت غیر صفر بودن آن مقدار true و در صورت صفر بودن آن مقدار false را برمی گرداند.

```
// IsMoving - is the car moving?
        public Boolean IsMoving()
                                {
            // Is the car's speed zero?
                        if (Speed == 0)
                          return false;
                           return true;
                                }
```

ممکن است در ابتدا از حالت نوشته شدن این تابع تعجب کنید و این سوال پیش بیاید که چرا دستور return دوم در بخش else قرار داده نشده است. همانطور که می دانید دستور return برای برگرداندن مقداری توسط تابع، به کدی که تابع را فراخوانی کرده است به کار می رود. هنگامی که برنامه، در اجرای تابع به اولین return رسید، بقیه دستورات تابع را اجرا نمی کند و به متدی که تابع را فراخوانی کرده است برمی گردد. پس در اینجا اگر مقدار Speed برابر با صفر بود، برنامه مقدار false را برمی گرداند و دستور return true را نیز اجرا نمی کند. اما اگر مقدار Speed مخالف صفر بود، برنامه به خط بعد از if می آید که در این حالت باید مقدار true را برگرداند. البته این نوع نوشتن کد فقط باعث کوتاهی برنامه می شود و در سرعت اجرای آن هیچ تاثیری ندارد.

اگرچه متدی که در این قسمت تعریف کرده ایم ساده است، اما از گیج شدن کاربر در مورد اینکه برای تعیین متوقف بودن اتومبیل چه خاصیت هایی را باید بررسی کند جلوگیری می کند.

اما قبل از اینکه شروع کنید و برای هر مسئله ای در کلاس یک متد ایجاد کنید، توجه داشته باشید که هر چه تعداد متدها و خاصیت های یک کلاس بیشتر باشند، کار با آن مشکل تر خواهد بود. بنابراین با در نظر گرفتن این نکته و مورد قبلی در ایجاد متد و خاصیت برای یک کلاس تعادل را رعایت کنید.

ممکن است تصور کنید به علت اینکه متدی که در این قسمت تعریف کردیم باعث ایجاد هیچ رفتاری در برنامه نمی شود، بهتر بود از یک خاصیت در این مورد استفاده می کردیم. بله، می توانستیم برای این مورد از یک خاصیت استفاده کنیم. اما، استفاده از متد در این قسمت باعث می شود کاربر بفهمد که این نتیجه به صورت مستقیم از یک فیلد خوانده نمی شود، بلکه توسط شیئ محاسبه می شود.

متدهای سازنده:

یکی از مهمترین جنبه های طراحی کلاسها، مفهوم **متد سازنده**[1] در کلاس است. **این متدها شامل کدهایی هستند که هنگام نمونه سازی شدن یک شیئ اجرا می شوند**. این مورد هنگامی مفید است که بخواهید قبل از اینکه کاربر از یک شیئ استفاده کند، آن را به صورت خاصی تنظیم کنید. برای مثال بخواهید به بعضی از خاصیت های آن مقدار اولیه نسبت دهید، همانند خاصیت `NumberOfDoors` در کلاس `Car`.

متدهای سازنده در کلاس متدهایی هستند که نام آنها با نام کلاس یکی است. همچنین نباید برای این متدها هیچ مقدار بازگشتی مشخص کرد (حتی `void`). یک تابع سازنده برای کلاس `Car`، متدی مانند زیر خواهد بود:

```
    public Car()
        {
// Do some initialization here
        }
```

ایجاد یک متد سازنده:

در بخش امتحان کنید زیر، نحوه ایجاد یک متد سازنده ی ساده را مشاهده خواهید کرد.

امتحان کنید: ایجاد یک متد سازنده

۱) برای اینکه نحوه کارکرد متد سازنده را بررسی کنیم، باید مقدار اولیه ۴ را از مقابل تعریف `_numebrOfDoors` حذف کنیم. در کلاس `Car` تغییرات مشخص شده در زیر را وارد کنید:

```
public string Color;
 private int _speed;
private int _numberOfDoors;
```

۲) حال متد زیر را اضافه کنید تا تابع سازنده آن ایجاد شود. هر کدی که در این متد وارد کنید، هنگام نمونه سازی شیئ اجرا می شود.

```
    // Constructor
    public Car()
        {
// Set the default values
        Color = "White";
            _speed = 0;
        _numberOfDoors = 4;
```

---

[1] `Constructor`

```
    }
```

نکته: تنظیم مقدار speed_ با عدد صفر کاری بیهوده است، زیرا مقدار این متغیر به صورت پیش فرض برابر با صفر می شود. اما برای تکمیل شدن مثال، مقدار این متغییر را نیز در این مرحله تنظیم کرده ایم.

۳) برای تست متد سازنده کلاس Car، بهتر است زیر برنامه جدایی را در فایل Program.cs به صورت زیر اضافه کنید تا اطلاعات یک شیئ از کلاس را در صفحه نمایش دهد.

```
        // DisplayCarDetails -
    // procedure that displays the car's details
        static void DisplayCarDetails(Car theCar)
                                                {
            // Display the details of the car
    Console.WriteLine("Color: " + theCar.Color);
        Console.WriteLine("Number of doors: " +
    theCar.NumberOfDoors);
          Console.WriteLine("Current speed: " +
              theCar.Speed);
                                                }
```

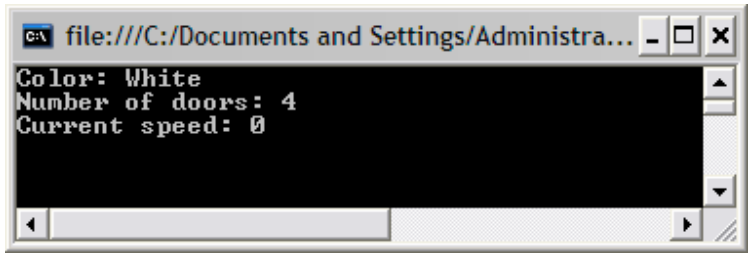
۴) حال زیر برنامه Main را به گونه ای تغییر دهید تا متد DisplayCarDetails را فراخوانی کند.

```
                    static void Main(string[] args)
                                                   {
                        // Create a new Car
                       Car objCar = new Car();

                // Display the details of the car
                        DisplayCarDetails(objCar);

                  // Wait for input from the user
                        Console.ReadLine();
                                                   }
```

۵) برنامه را اجرا کنید. پنجره ای مشابه شکل ۹-۵ نمایش داده می شود.



شکل ۹-۵

چگونه کار می کند؟

هنگامی که شیئ جدیدی از کلاس Car بخواهد ایجاد شود، کدی که درون تابع سازنده نوشته اید اجرا می شود. به این صورت می توانید خاصیت های کلاس را برابر با مقادیر مورد نظرتان قرار دهید.

```
// Constructor
  public Car()
                {
// Set the default values
          Color = "White";
              _speed = 0;
      _numberOfDoors = 4;
                }
```

نتیجه این مقدار دهی اولیه را هنگام اجرای برنامه، زمانی که اطلاعات کلاس در صفحه نمایش داده می شود مشاهده خواهید کرد. دقت کنید که اگر برای متد سازنده نوع داده برگشتی تعیین کنید و یا حتی از کلمه void استفاده کنید، با خطا مواجه خواهید شد. به عبارت دیگر متد سازنده یک کلاس همواره دارای ساختار مشخصی است.
برای تست شیئ، از یک متد جدا به نام DisplayCarDetails در فایل Program.cs استفاده می کنیم. به این ترتیب اگر بخواهیم مشخصات چندین شیئ از نوع Car را نمایش دهیم و یا مشخصات یک شیئ را چندین مرتبه نمایش دهیم، به کد کمتری نیاز خواهیم داشت.

نکته: دقت کنید که متد DisplayCarDetails برای اینکه بتواند در متد Main فراخوانی شود حتماً باید از نوع static تعریف شود. در مورد متدهای static و نحوه استفاده از آنها در فصل ۱۰ صحبت خواهیم کرد.

وراثت:

وراثت[1] یکی از مباحث پیشرفته و همچنین کاربردی برنامه نویسی شیئ گرا محسوب می شود. در چارچوب .NET از وراثت استفاده زیادی شده است و حتی خود شما تاکنون کلاس هایی ایجاد کرده اید که از کلاسهای دیگر ارث برده اند – هر فرم ویندوزی که در برنامه های خود ایجاد می کردید، در حقیقت یک کلاس جدید بود که از بسیاری از اطلاعات خود را از کلاس مربوط به یک فرم خالی به ارث می برد.
وراثت برای ایجاد اشیای به کار برده می شود که "تمام اعضای یک شیئ دیگر را داشته باشد و علاوه بر آنها، شامل چندین عضو جدید برای خودش باشد". هدف اصلی وراثت این است که بتوانید کارایی های یک کلاس را، بدون اینکه بدانید آن کلاس به صورت درونی چگونه کار می کند، افزایش دهید. به عبارت دیگر با استفاده از وراثت می توانید اشیایی را بر پایه اشیای دیگر که توسط برنامه نویسان دیگری نوشته شده است ایجاد کنید، بدون اینکه بدانید برنامه نویسان اصلی چگونه آن شیئ پایه را ایجاد کرده اند.

---

[1] Inheritance

به وسیله وراثت می توانید از یکی از کلاسهای موجود استفاده کرده، خاصیت ها و متدهای جدیدی به آن اضافه کنید و یـا بعـضی از متدها و خاصیت های آن را با متدها و خاصیت های مورد نظر خودتان عوض کنید و به این ترتیب کلاس جدیدی ایجـاد کنیـد کـه دقیقاً نیازهای تان را برطرف کند. برای مثال، با استفاده از کلاس Car که یک کلاس کلی است، می توانید کلاسهای خاصی تری مانند کلاسی برای اتومبیل های مسابقه ای، کلاسی برای وسایل نقلیه سنگین، کلاسی برای اتومبیل های سواری و ... ایجاد کنید.
فرض کنید می خواهید اتومبیل های مسابقه ای را در کامپیوتر به وسیله کلاسـی بـه نـام SportsCar مـدل کنیـد. کـلاس SportsCar مشابه کلاس Car خواهد بود اما در بعضی از قسمتها تفاوتهای جزئی دارد. برای مثال تعداد درهـا در اتومبیـل های مسابقه ای ممکن است بین ۲ تا ۶ نباشد و یا در این کلاس، علاوه بر متد ها و خاصیت های موجود در کلاس Car به متدها و خاصیت هایی نیاز دارید که اطلاعاتی را در مورد کارایی و عملکـرد اتومبیـل بـه کـاربر بدهـد، ماننـد متـدهای Weight و یـا PowerToWeightRatio که در شکل ۹-۶ نیز نشان داده شده اند.

نکته: به کلاسی که از کلاس دیگری به ارث گرفته شود (همانند کلاس SportsCar در مثال بالا)، **کلاس مشتق شده**[1] و به کلاسی که کلاسهای دیگر از آن مشتق می شوند (مانند کلاس Car در مثال بالا) **کلاس پایه**[2] می گویند.

یکی از مواردی که در مورد وراثت باید بدانید، نحوه دسترسی کـلاس مـشتق شـده بـه عـضو هـای public و private کلاس پایه است. هر عضو public از کلاس پایه به وسیله کلاس مشتق شده قابل دسترسی است، اما کلاس های مشتق شده به عضو های private کلاس پایه دسترسی ندارند. بنابراین اگر کلاس SportsCar بخواهد سرعت یک شیئ را تغییر دهد باید از خاصیت ها و متدهای موجود در کلاس Car استفاده کند و نمی تواند به صورت مستقیم به فیلد speed_ دسترسی داشته باشد.

شکل ۹-۶

اضافه کردن متدها و خاصیت های جدید:

برای درک بهتر وراثت، در بخش امتحان کنید بعد کلاس جدیدی به نام SportsCar ایجاد می کنـیم کـه از کـلاس Car مشتق شود و به وسیله آن بتوانید نسبت وزن اتومبیل به نیروی موتور آن را بدانید.

امتحان کنید: به ارث بردن از کلاس Car

۱) برای این مثال باید یک فیلد public به کلاس Car اضافه کنید تا قدرت اتومبیل را بر حسب اسب بخـار در خـود ذخیره کند. البته اگر می خواهید به صورت دقیق و درست کار کنید، باید یک خاصیت ایجاد کنید و به وسیله آن از درسـت بودن مقدار وارد شده توسط کاربر مطمئن شوید. اما در اینجا سادگی و سرعت برای مـا اهمیـت بیـشتری دارد، بنـابراین از فیلد به جای خاصیت استفاده می کنیم. فایل حاوی کلاس Car را باز کرده و کد زیر را به آن اضافه کنید:

```
public string Color;
```

[1] Derived Class
[2] Base Class

```
public int HorsePower;
private int _speed;
private int _numberOfDoors;
```

۲) پنجره Solution Explorer بروید و روی نام پروژه کلیک راست کنید. از منوی باز شده گزینه Add ➔ Class را انتخاب کرده و با استفاده از پنجره ای که نمایش داده می شود کلاس جدیدی به نام SportsCar.cs ایجاد کنید.

۳) حال باید به کلاس SportsCar بگوییم که از کلاس Car مشتق شود. برای این کار در مقابل نام کلاس SportsCar یک علامت : قرار داده و سپس نام کلاس پایه را ذکر می کنید (که در اینجا برابر با Car است). تعریف کلاس SportsCar را به صورت زیر تغییر دهید:

```
class SportsCar : Car
{
```

۴) به این ترتیب کلاس SportsCar دارای تمام متدها و خاصیت هایی است که کلاس Car شامل می شود. در این قسمت باید یک فیلد public که مخصوص کلاس SportsCar است را به آن اضافه کنیم. برای این کار کد زیر را به این کلاس اضافه کنید:

```
public int Weight;
```

۵) برای تست کلاس جدید باید یک زیر برنامه جدید به فایل Program.cs اضافه کنید. بنابراین کد زیر را به این فایل اضافه کنید:

```
// DisplaySportsCarDetails –
// procedure that displays a sports car's details
static void DisplaySportsCarDetails(SportsCar
theCar)
{
// Display the details of the sports car
Console.WriteLine();
Console.WriteLine("Sports Car Horsepower: " +
theCar.HorsePower);
Console.WriteLine("Sports Car Weight: " +
theCar.Weight);
}
```

۶) حال زیر برنامه Main در فایل Program.cs را تغییر دهیم. توجه کنید که در این مرحله برای اینکه بتوانیم به فیلد Weight دسترسی داشته باشیم، باید یک شیئ از کلاس SportsCar ایجاد کنیم نه یک شیئ از کلاس Car. تغییرات زیر را در متد Main ایجاد کنید:

```
static void Main(string[] args)
{
```

```
            // Create a new sport car object
            SportsCar objCar = new SportsCar();

            // Modify the number of doors
            objCar.NumberOfDoors = 2;

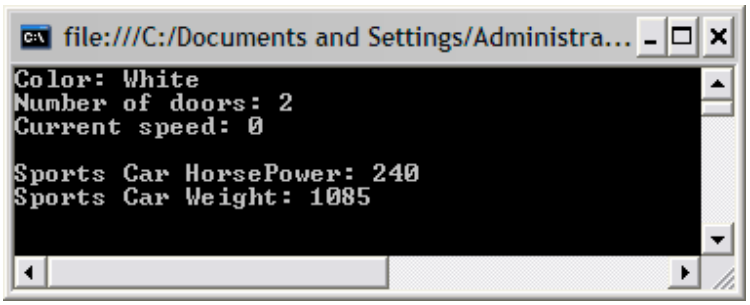
            // Set the horsepower and weight (KG)
            objCar.HorsePower = 240;
            objCar.Weight = 1085;

            // Display the details of the car
            DisplayCarDetails(objCar);

            DisplaySportsCarDetails(objCar);

            // Wait for input from the user
            Console.ReadLine();
        }
```

۷) برنامه را اجرا کنید. پنجره ای مشابه شکل ۹-۷ نمایش داده خواهد شد.



شکل ۹-۷

چگونه کار می کند؟

برای اینکه مشخص کنیم یک کلاس از کلاس دیگری مشتق شده است، باید هنگام تعریف کلاس، نام کلاس پایه را در مقابل نام کلاس مشتق شده بعد از علامت : قرار دهیم.

```
class SportsCar : Car
```

به این صورت کلاس SportsCar شامل تمام خاصیت ها و متدهای کلاس Car خواهد بود، اما با وجود این کلاس SportsCar نمی تواند به اعضای private کلاس Car دسترسی داشته باشد. هنگامی که یک فیلد جدید تعریف می کنید:

```
public int Weight;
```

این فیلد فقط در اشیای که از کلاس SportsCar نمونه سازی شوند وجود خواهند داشت و اشیای که از کلاس Car نمونه سازی شوند شامل چنین فیلدی نخواهند بود. به این مورد همواره توجه کنید – اگر شیئ که ایجاد می کنید از کلاس SportsCar نباشد و بخواهید به فیلد Weight در آن شیئ دسترسی پیدا کنید، هنگام کامپایل برنامه با خطا مواجه خواهید شد. فیلد Weight تحت هیچ شرایطی نمی تواند در اشیایی که از کلاس Car نمونه سازی شده اند وجود داشته باشد (برای روشن شدن بهتر این مطلب، شکل ۹-۶ را ببینید).
زیر برنامه DisplaySportsCarDetails خاصیت Horsepower از کلاس Car و همچنین خاصیت Weight از کلاس SportsCar را نمایش می دهد. توجه کنید به علت اینکه کلاس SportsCar از کلاس Car مشتق شده است شامل تمام خاصیت ها و متدهای موجود در این کلاس است.

```
// DisplaySportsCarDetails –
// procedure that displays a sports car’s details
static void DisplaySportsCarDetails(SportsCar
theCar)
{
// Display the details of the sports car
Console.WriteLine();
Console.WriteLine("Sports Car Horsepower: " +
theCar.HorsePower);
Console.WriteLine("Sports Car Weight: " +
theCar.Weight);
}
```

حال در زیر برنامه Main ابتدا باید یک شیئ از کلاس SportsCar ایجاد کنیم تا بتوانیم به مقدار فیلد Weight دسترسی داشته باشیم:

```
// Create a new sport car object
SportsCar objCar = new SportsCar();
```

همانطور که مشاهده می کنید هنگام استفاده از تابع DisplayCarDetails به جای اینکه یک شیئ از نوع Car را به آن بفرستیم می توانیم از یک شیئ SportsCar استفاده کنیم، زیرا کلاس SportsCar زیر مجموعه کلاس Car است. به عبارت دیگر هر شیئ از نوع SportsCar در حقیقت یک شیئ از نوع Car است (همانطور که در دنیای واقعی نیز هر اتومبیل مسابقه ای، در واقع یک اتومبیل است). بعد از فراخوانی متد DisplayCarDetails، متد DisplaySportsCarDetails را احظار می کنیم تا خاصیتهای مربوط به شیئ SportsCar نیز نمایش داده شود.

```
// Display the details of the car
DisplayCarDetails(objCar);
DisplaySportsCarDetails(objCar);
```

اضافه کردن متد GetPowerToWeightRatio:

متد GetPowerToWeightRatio برای مشخص کردن نسبت وزن اتومبیل به نیروی موتور آن به کار می رود و می تواند به صورت یک خاصیت فقط-خواندنی ایجاد شود (که در این صورت نام آن باید به PowerToWaightRatio تغییر کند)، اما برای تکمیل مثال این قسمت بهتر است که آن را به صورت یک متد تعریف کنیم.

امتحان کنید: اضافه کردن متد GetPowerToWeightRatio

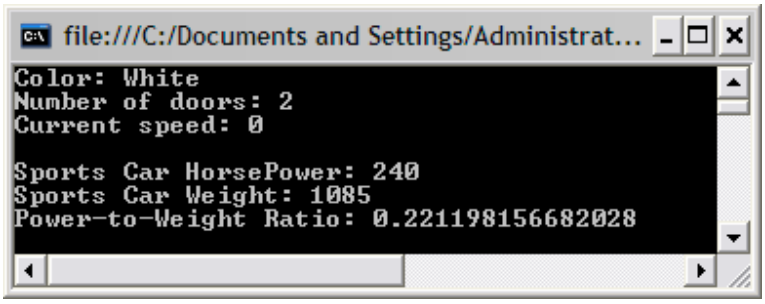
۱) برای محاسبه مقدار مورد نظر در این متد باید نیروی موتور را بر وزن اتومبیل تقسیم کنید. برای این کار کد زیر را به کلاس SportsCar اضافه کنید:

```
// GetPowerToWeightRatio - work out the power to
                                        // weight
                public double GetPowerToWeightRatio()
                                                    {
            // Calculate the power-to-weight ratio
                return (double)HorsePower / Weight;
                                                    }
```

۲) برای مشاهده نتیجه، کد زیر را به متد DisplaySportsCarDetails در فایل Program.cs اضافه کنید:

```
                        // DisplaySportsCarDetails –
      //procedure that displays a sports car's details
        static void DisplaySportsCarDetails(SportsCar
       theCar)
                                                    {
          // Display the details of the sports car
                                Console.WriteLine();
  Console.WriteLine("Sports Car Horsepower: " +
                theCar.HorsePower);
      Console.WriteLine("Sports Car Weight: " +
                    theCar.Weight);
    Console.WriteLine("Power-to-Weight Ratio: " +
theCar.GetPowerToWeightRatio());
                                                    }
```

۳) برنامه را اجرا کنید. پنجره ای مشابه شکل ۹-۸ مشاهده می کنید.



شکل ۹-۸

چگونه کار می کند؟

مجدداً تمام کاری که باید انجام دهید این است که متد جدیدی را به نـام `GetPowerToWeightRatio` بـه کـلاس `SportsCar` اضافه کنید. به این ترتیب همانطور که در شکل ۹-۹ مشاهده می کنید، این متد برای تمام اشیایی که از کلاس `SportsCar` نمونه سازی شده باشند قابل دسترسی خواهد بود.

شکل ۹-۹

توجه کنید که در این متد قبل از تقسیم نیروی موتور اتومبیل بر وزن آن، فیلد حاوی نیروی موتـور را بـه نـوع داده ای `double` تبدیل کرده ایم. همانطور که در قسمتهای قبلی گفتم، تقسیم یک متغیر از نوع عدد صحیح بر متغیـر دیگـری از نـوع عـدد صـحیح، باعث می شود که حاصل نیز از نوع عدد صحیح باشد. اما در این متد ما به قسمت اعشار حاصل تقسیم نیاز داریم.
با تبدیل یکی از طرفین تقسیم به نوع داده ای `double`، باعث می شویم که نوع داده ای دو موردی که می خواهند بـر یکـدیگر تقسیم شوند متفاوت شود. بنابراین هنگامی که کامپایلر بخواهد این دو مقدار را بر هم تقسیم کند نوع داده ای کوچکتر را (`int`) به نوع داده ای بزرگتر (`double`) تبدیل مـی کنـد و سـپس تقـسیم را انجـام مـی دهـد، همچنـین حاصـل تقـسیم را نیـز از نـوع `double` برمی گرداند. به این ترتیب بعد از تقسیم به مقدار اعشاری نیز دسترسی خواهیم داشت.

```
// Calculate the power-to-weight ratio
   return (double)HorsePower / Weight;
```

تغییر دادن پیش فرض ها:

علاوه بر اضافه کردن متدها و خاصیت های جدید به یک کلاس مشتق شده، در شرایطی ممکن است بخواهید یکـی از متـدها و یـا خاصیت های کلاس پایه، در کلاس مشتق شده به گونه ای دیگر عمل کند. برای این کار بایـد آن متـد و یـا خاصـیت را از اول در کلاس مشتق شده بنویسید.
همانطور که در بخش متدهای سازنده گفتیم، این متدها زمانی که یک شیئ از کلاس بخواهد نمونه سازی شود فراخوانی می شوند و اجازه می دهند که حالت اولیه اعضای یک شیئ را تعیین کنید. در متد سازنده کلاس `Car`، مقدار `_numberOfDoors` را برابر با ۴ در نظر گرفتیم در صورتی که به طور معمول تعداد درها برای یک اتومبیل مسابقه ای ۲ در است. در مثال قبلـی بـا تنظـیم

این خاصیت بعد از ایجاد شیئ، تعداد درها را برابر با ۲ قرار دادیم. اما سوال این است که چگونه می توانیم این مقدار را هنگـام نمونـه سازی شدن شیئ از کلاس SportsCar تنظیم کنیم؟

اگر بخواهید یکی از متدهای کلاس پایه را با یک متد جدید در کلاس مشتق شده که توسط خودتان نوشته شده است تعویض کنید، به این پروسه **override کردن** گفته می شود. به این ترتیب اگر آن متد را در یک شیئ از کلاس مشتق شده فراخوانی کنید، متد جدید اجرا می شود اما اگر این متد را در یک شیئ از کلاس پایه فراخوانی کنید، متد قدیمی مورد استفاده قرار خواهـد گرفـت. در بخش امتحان کنید بعد، مشاهده خواهید کرد که چگونـه مـی تـوان متـد سـازنده کـلاس Car را در کـلاس SportsCar، override کرد.

امتحان کنید: override کردن متد سازنده

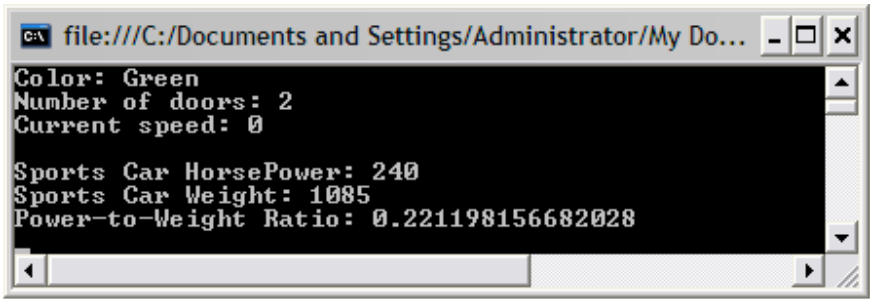
۱) برای override کردن متد سازنده کلاس Car در کلاس SportsCar، تنها کاری که باید انجـام دهیـد این است که یک متد سازنده جدید برای کلاس SportsCar ایجاد کنید. بـرای ایـن کـار کـد زیـر را بـه کـلاس SportsCar اضافه کنید:

```
        // Constructor
        public SportsCar()
        {
            // Change the default values
            Color = "Green";
            NumberOfDoors = 2;
        }
```

۲) خط زیر را از زیر برنامه Main در فایل Program.cs حذف کنید.

```
// Modify the number of doors
objCar.NumberOfDoors = 2;
```

۳) حال برای تست کردن متد سازنده کلاس SportsCar، برنامه را اجرا کنید. بعد از اجرای برنامه پنجـره ای مـشابه شکل ۹-۱۰ را مشاهده خواهید کرد.



شکل ۹-۱۰

چگونه کار می کند؟

متد سازنده ای که برای کلاس SportsCar ایجاد کرده اید، بعد از متد سازنده موجود در کلاس Car اجرا می شود. توجه داشته باشید هنگامی که یک متد سازنده برای کلاس مشتق شده ایجاد کنید، باز هم متد سازنده موجود در کلاس پایه احضار می شود. اما .NET، متد سازنده کلاس پایه را قبل از متد سازنده کلاس مشتق شده فراخوانی می کند. در حقیقت ترتیب اجرای متدها به این صورت است که ابتدا متد زیر اجرا می شود:

```
// Constructor
public Car()
{
// Set the default values
Color = "White";
_speed = 0;
_numberOfDoors = 4;
}
```

سپس متد زیر اجرا می شود:

```
// Constructor
public SportsCar()
{
// Change the default values
Color = "Green";
NumberOfDoors = 2;
}
```

همچنین توجه کنید که برای تغییر دادن تعداد درها در کلاس مشتق شده نمی توانیم به صورت مستقیم به فیلد _numberOfDoors دسترسی داشته باشیم، زیرا این فیلد در کلاس پایه از نوع private تعریف شده است و متدهای کلاس مشتق شده به آن دسترسی ندارند. به همین دلیل با استفاده از خاصیت NumberOfDoors تعداد درها را برابر با ۲ قرار می دهیم.

چند شکلی بودن: کلمه ای ترسناک، مفهومی ساده

یکی دیگر از اصول مهم برنامه نویسی شیئ گرا، مفهوم **چند شکلی**[1] بودن است. احتمالاً این مفهوم یکی از سخت ترین مفاهیم شیئ گرایی به نظر می آید، اما درک آن بسیار راحت است. در حقیقت تاکنون در برنامه های قبلی خود نیز از چند شکلی بودن استفاده کرده اید.

بار دیگر به کد زیر برنامه ی DisplayCarDetails توجه کنید:

```
static void DisplayCarDetails(Car theCar)
```

[1] Polymorphism

```
{
        // Display the details of the car
    Console.WriteLine("Color: " + theCar.Color);
        Console.WriteLine("Number of doors: " +
    theCar.NumberOfDoors);
        Console.WriteLine("Current speed: " +
            theCar.Speed);
}
```

همانطور که در خط اول مشاهده می کنید، این متد پارامتری از نوع Car به عنوان ورودی دریافت می کند. اما هنگام فراخـوانی آن پارامتری از نوع SportsCar به آن فرستاده می شود.

```
// Create a new sport car object
SportsCar objCar = new SportsCar();

// Display the details of the car
DisplayCarDetails(objCar);
```

سوالی که پیش می آید این است که چگونه می توان به متدی که پارامتری از نوع Car دارد، شـیئ از نـوع SportsCar را فرستاد؟

خوب، چند شکلی بودن به این معنی است که یک شیئ از کلاس مشتق شده بتواند در مواقعی که به یک شیئ از کـلاس پایـه نیـاز است، همانند شیئ ای از کلاس پایه عمل کند. در این مثال شما می توانید با شیئ ای از SportsCar هماننـد یـک شـیئ از کلاس Car رفتار کنید، زیرا کلاس SportsCar از کلاس Car مشتق شده است. چند شکلی بودن به این دلیل است که همانطور که در بخش وراثت گفتیم، هر شیئ از کلاس مشتق شده باید دارای تمام قابلیتهای یک شیئ از کلاس پایـه باشـد؛ اگرچـه می تواند متدها و یا خاصیتهای بیشتری نیز داشته باشد. به این ترتیب اگر بخواهیم متدی را از کلاس Car فراخوانی کنیم، کـلاس SportsCar نیز حتماً دارای این متد خواهد بود.

البته واضح است که عکس این مورد درست نیست. متد DisplaySportsCarDetails که به صورت زیر تعریف شده است:

```
static void DisplaySportsCarDetails(SportsCar
theCar)
```

نمی تواند یک شیئ از نوع Car را نیز به عنوان پارامتر دریافت کند. کلاس Car تضمین نمی کند که هر متد و یا خاصـیتی کـه در کلاس SportsCar وجود دارد را داشته باشد، زیرا متدها و یا خاصیت هایی در کلاس SportsCar وجود دارند کـه در کلاس Car تعریف نشده است. به عبارت دیگر SportsCar نوع خاصی از Car است.

Override کردن متدهای بیشتر:

اگرچه در قسمت قبلی متد سازنده کلاس Car را در کلاس SportsCar، override کردیم، اما بهتـر اسـت کـه بـا نحوه override کردن یک تابع معمولی نیز آشنا شویم.

همانطور که می دانید برای اینکه بتوانیم یک متد را `override` کنیم، باید آن متد در کلاس پایه وجود داشته باشد. متد `Accelerate` در کلاس `Car` و کلاس `SportsCar` نباید تفاوتی داشته باشد، زیرا عملکرد آن در هر دو نوع مشابه است. همچنین متد `IsMoving` هم فقط برای راحتی کاربر اضافه شده است و به عنوان رفتار شیئ به شمار نمی رود که آن را `override` کنیم. بنابراین نیاز داریم که متد جدیدی به نام `CalculateAccelerationRate` اضافه کنیم. فرض می کنیم که در یک اتومبیل عادی این مقدار یک عدد ثابت است، اما در اتومبیل های مسابقه ای باید محاسبه شود. در بخش امتحان کنید بعد، متد جدیدی برای `override` کردن اضافه می کنیم.

امتحان کنید: اضافه کردن و `override` کردن یک متد دیگر

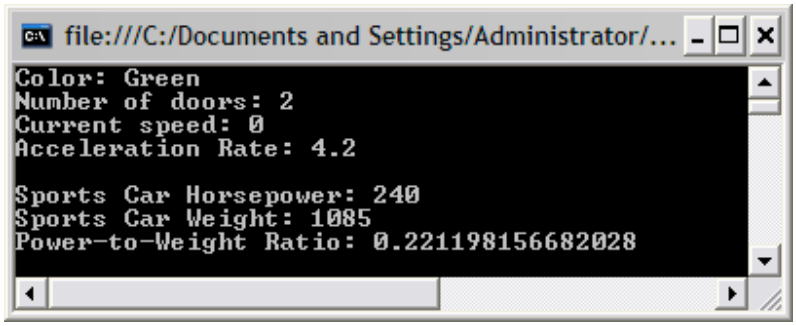
۱) متد زیر را به کلاس `Car` اضافه کنید:

```
        // CalculateAccelerationRate -
    // assume a constant for a normal car
public double CalculateAccelerationRate()
                                        {
// If we assume a normal car goes from 0-60 in
 // 14 seconds, that's an average of 4.2 kmph/s
                                  return 4.2;
                                        }
```

۲) حال برای تست این متد، زیر برنامه ی `DisplayCarDetails` را به صورت زیر تغییر دهید:

```
                        // DisplayCarDetails -
    // procedure that displays the car's details
       static void DisplayCarDetails(Car theCar)
                                              {
            // Display the details of the car
  Console.WriteLine("Color: " + theCar.Color);
      Console.WriteLine("Number of doors: " +
              theCar.NumberOfDoors);
        Console.WriteLine("Current speed: " +
                    theCar.Speed);
     Console.WriteLine("Acceleration Rate: " +
theCar.CalculateAccelerationRate());
                                              }
```

۳) برنامه را اجرا کنید، به این ترتیب خروجی مشابه شکل ۹–۱۱ مشاهده خواهید کرد.



شکل ۹-۱۱

۴) برای اینکه بتوانیم یکی از متدهای کلاس پایه را در کلاس مشتق شده `override` کنیم، آن متد در کلاس پایه باید با کلمه کلیدی `virtual` تعریف شده باشد. بنابراین در متد جدید کلاس `Car`، کلمه `virtual` را به صورت زیر اضافه کنید:

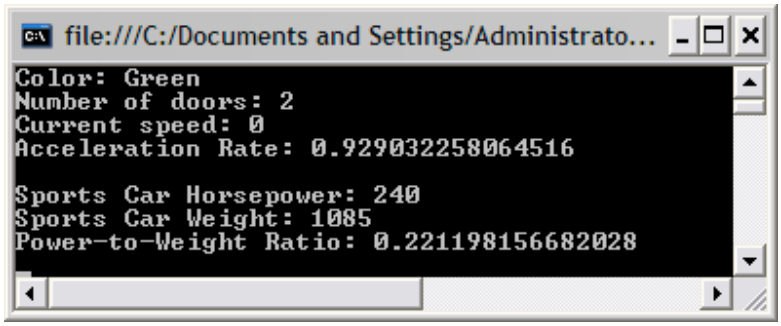
```
public virtual double CalculateAccelerationRate()
```

۵) حال می توانید متد `CalculateAccelerationRate` را در کلاس `SportsCar`، `override` کنید. برای این کار باید متدی در کلاس `SportsCar` ایجاد کنید که نام، تعداد پارامتر ها و نیز مقدار برگشتی آن دقیقاً مشابه این متد در کلاس `Car` باشد و همچنین در تعریف متد از کلمه کلیدی `override` استفاده کنید. متد زیر را به کلاس `SportsCar` اضافه کنید:

```
                    // CalculateAccelerationRate -
            // take the power/weight into consideration
    public override double CalculateAccelerationRate()
                                                    {
  // You'll assume the same 4.2 value, but you'll
            // multiply it by the power/weight ratio
                return 4.2 * GetPowerToWeightRatio();
                                                    }
```

نکته: همانطور که در بخش قبلی مشاهده کردید، برای `override` کردن متد سازنده یک کلاس، نیازی به استفاده از کلمه کلیدی `override` نیست. ویژوال C# این کار را به صورت اتوماتیک انجام می دهد.

۶) حال برنامه را اجرا کنید. همانطور که در شکل ۹-۱۲ مشاهده می کنید، این مرتبه از متد تعریف شده در کلاس `SportsCar` استفاده شده است.



شکل ۹-۱۲

چگونه کار می کند؟

به وسیله `override` کردن یک متد، می توانید پیاده سازی[1] متدهای موجود در کلاس پایه را در کلاس مشتق شده تغییر دهید. البته این کار برای تمام متدهای کلاس پایه ممکن نیست، بلکه متد مورد نظر باید با کلمه کلیدی `virtual` مشخص شود تا بتوانید آن را در کلاس مشتق شده `override` کنید.

مجدداً به مفهوم کپسولی بودن برمی گردیم. بعد از `override` کردن متد، فردی که از آن استفاده می کند اصلاً تفاوتی را در اجرای متد متوجه نمی شود – او فقط به همان صورتی که از این متد قبل از `override` شدن استفاده می کرد هم اکنون هم استفاده می کند. اما این مرتبه به علت اینکه در واقع در حال کار با یک شیئ از کلاس `SportsCar` است نتیجه متفاوتی را دریافت می کند.

`override` کردن یک متد از بسیاری از جهات با `override` کردن یک متد سازنده تفاوت دارد. هنگامی که یک متد سازنده را `override` می کنید، قبل از اینکه متد سازنده ی جدید فراخوانی شود متد سازنده قبلی فراخوانی خواهد شد. اما اگر یک متد عادی را `override` کنید، متد قبلی فقط در صورتی فراخوانی خواهد شد که از دستور `base.MethodName` استفاده کنید. کلمه کلیدی `base` در کلاس مشتق شده برای دسترسی به اعضای کلاس پایه به کار می رود.

فرض کنید متد `CalculateAccelerationRate` در کلاس `Car` یک مقدار ثابت را برنمی گرداند، بلکه بعد از انجام دادن یک سری محاسبات، مقداری را برمی گرداند و شما می خواهید این متد در کلاس `SportsCar`، علاوه بر انجام دادن آن محاسبات، مقدار نهایی را در مقدار تابع `GetPowerToWeightRatio` نیز ضرب کند. بنابراین می توانید از دستور زیر در متد `override` شده در کلاس `SportsCar` استفاده کنید:

```
return (base.CalculateAccelerationRate() *
        GetPowerToWeightRatio());
```

به این ترتیب ابتدا تابع `CalculateAccelerationRate` از کلاس پایه فراخوانی می شود. سپس مقدار برگشتی این تابع در مقدار بر گشتی تابع `GetPowerToWeightRatio` ضرب شده و نتیجه به عنوان مقدار جدید تابع `CalculateAccelerationRate` برمی گردد.

---

[1] `Implementation` – به یک سری دستوراتی که درون یک متد قرار می گیرند و نحوه عملکرد آن را مشخص می کنند، پیاده سازی آن متد می گویند.

به ارث بردن از کلاس Object:

آخرین مطلبی که در مورد وراثت باید بررسی کنیم این است که در .NET حتی اگر کلاسی را به صورت عادی و بدن تعیین کردن کلاس پایه برای آن تعریف کنید، آن کلاس از کلاسی به نام Object مشتق می شود. کلاس Object شامل چهار متد است که می توانید تضمین کنید در تمام کلاسهای نوشته شده به وسیله .NET وجود دارند. البته شرح وظیفه این متدها خارج از مباحث این کتاب است، با وجود این دو تابع پر استفاده از آنها را در این قسمت معرفی می کنیم:

- **ToString:** این متد رشته ای که حاوی متنی برای معرفی کردن شیئ است را برمی گرداند. به خاطر دارید که در ابتدای فصل گفتم هر شیئ باید بداند که از چه نوعی است. در حقیقت با استفاده از این متد، یک شیئ نوع خود را به شما اعلام می کند. البته این در صورتی است که این متد override نشده باشد. هنگام طراحی کلاس، می توانید این متد را override کنید تا یک رشته ی با معنی برای مشخص کردن شیئ برگرداند، برای مثال در کلاس Customer این متد را به صورتی override کنید که نام مشترک را برگرداند. اگر در یک کلاس این تابع را override نکنید، به صورت پیش فرض نام کلاس توسط این تابع برگشت داده می شود.
- **GetType:** این متد شیئ را از کلاس Type برمی گرداند که مشخص کننده نوع داده ای کلاس است.

به خاطر داشته باشید که لازم نیست یک کلاس را با استفاده از عملگر : از کلاس Object مشتق کنید، زیرا این کار به صورت اتوماتیک انجام می شود.

اشیا و ساختارها:

با نحوه کار با ساختارها در فصل پنجم آشنا شدید. ساختارها نیز همانند کلاسها راهی را فراهم می کنند که بتوانید چندین قطعه از اطلاعات مرتبط به هم را در یک گروه قرار دهید. یک ساختار هم می تواند همانند یک کلاس، علاوه بر فیلد شامل خاصیت و متد نیز باشد. در این قسمت با بعضی از تفاوتهای کلاسها و ساختارها آشنا می شویم.

در اصطلاح کامپیوتری، ساختارها به عنوان **نوع های مقداری**[1] و کلاسها به عنوان **نوع های ارجاعی**[2] شناخته می شوند. هر متد هنگامی که فراخوانی می شود، قسمتی از حافظه را اشغال می کند و متغییر هایی که در طول کار خود به آنها نیاز دارد را در این قسمت ایجاد می کند و معمولاً بعد از اجرای تابع، این قسمت از حافظه نیز آزاد می شود. همچنین پارامترهایی که به آن فرستاده می شوند نیز در حقیقت در این قسمت از حافظه کپی می شوند تا متد بتواند از آنها استفاده کند.

ساختارها معمولاً اندازه های کوچکی دارند، برای مثال شامل چندین متغیر از نوع عدد صحیح و یا چند رشته هستند که مقدار کمی از فضای حافظه را اشغال می کنند. بنابراین کامپیوتر می تواند به راحتی آنها را به فضای مخصوص یک تابع انتقال دهد تا تابع بتواند از آن استفاده کند. اما کلاسها بسیار بزرگتر از ساختارها هستند. یک کلاس معمولاً شامل چندین عضو از کلاسهای دیگر است. برای مثال کلاسی به نام Control در چارچوب .NET وجود دارد که تمام کنترلهای مورد استفاده از آن مشتق می شوند. این کلاس شامل ۲۰۵ فیلد (که بسیاری از آنها شامل شیئ ای از یک کلاس دیگر هستند)، ۷ متد سازنده، ۱۶۳ خاصیت و ۵۲۴ متد است. مسلم است که فضایی که چنین کلاسی از حافظه اشغال می کند بسیار بیشتر از یک ساختار است و کامپیوتر نمی تواند در صورت لزوم آن را به راحتی به فضای مخصوص به یک تابع انتقال دهد. برای حل این مشکل از آدرس کلاسها در حافظه استفاده

---

[1] Value Type
[2] Reference Type

می کنند. به این معنی که هنگامی که یک شیئ برای مرتبه اول در قسمتی از حافظه ایجاد می شود، اگر بخواهیم آن را به یک تابع به عنوان پارامتر انتقال دهیم کل شیئ را مجدداً در فضای مخصوص تابع کپی نمی کنیم، بلکه آدرس کنونی شیئ را در حافظه به تابع می فرستیم. به این ترتیب تابع در صورت نیاز می تواند با استفاده از آدرس شیئ، به آن دسترسی داشته باشد.
بنابراین به علت اینکه در صورت نیاز به انتقال ساختارها مقدار آنها منتقل می شود، به آنها نوع های مقداری و برای اینکه در صورت نیاز به انتقال کلاسها فقط آدرس قرارگیری آنها در حافظه منتقل می شود، به کلاسها نوع های ارجاعی می گویند.
در قسمتهای قبلی گفتم که یک شیئ از کلاس می تواند دارای چندین نام باشد (به عبارت دیگر از چندین قلاب آویزان شود). دلیل این مورد این است که یک متغیر که برای یک شیئ از کلاس تعریف می کنیم در واقع محتوی خود شیئ نیست، بلکه محتوی آدرس قرارگیری آن شیئ در حافظه است. بنابراین می توانیم چندین متغیر با نامهای متفاوت تعریف کنیم و آدرس شیئ را در آنها قرار دهیم. به این ترتیب برای دسترسی به شیئ از هر کدام از این متغییر ها می توانید استفاده کنید. همچنین اگر با استفاده از یکی از این متغییر ها شیئ را تغییر دهید، شیئ موردنظر برای تمام متغییر های دیگر نیز تغییر می کند.
بنابراین هنگام تصمیم گیری بین ساختار و کلاس همواره در نظر داشته باشید که اگر اطلاعاتی که می خواهید در یک گروه قرار دهید کوچک بودند بهتر است که از ساختارها استفاده کنید. در غیر این صورت استفاده از کلاسها بهتر از استفاده از ساختارها است. البته کلاسی که در بالا مثال زدیم نیز کلاس بسیار بزرگی است و حتماً نباید حجم اطلاعات به این اندازه باشد تا از کلاس استفاده کنید.
همچنین یکی دیگر از تفاوتهای ساختارها با کلاسها در این است که ساختارها دارای مبحث وراثت نیستند.

کلاسهای چارچوب .NET:

اگرچه چارچوب .NET را در فصل دوم به طور کلی بررسی کردیم، در این قسمت سعی می کنیم به قسمتهایی از ساختار این چارچوب که می تواند در طراحی کلاسها به شما کمک کند نگاهی بیندازیم. همچنین در این قسمت فضای نامها و نحوه ایجاد و استفاده از آنها در برنامه را مشاهده خواهیم کرد.

فضای نام:

یکی از قسمتهای مهم چارچوب .NET.، کلکسیون عظیم کلاسهای آن است. در چارچوب .NET حدود ۳۵۰۰ کلاس در رابطه با موارد مختلف وجود دارد، اما سوال این است که چگونه به عنوان یک برنامه نویس می توانید کلاس مورد نظرتان را در بین این کلاسها پیدا کنید؟
کلاسهای موجود در چارچوب .NET به چندین گروه مختلف به نام **فضای نام**[1] تقسیم می شود که هر کدام از آنها حاوی چندین کلاس مرتبط به هم است. به این ترتیب اگر به دنبال کلاسی برای یک کار خاص باشید می توانید فقط کلاسهای داخل فضای نام مربوط به آن کار را جستجو کنید.
خود این فضای نامها نیز به صورت سلسله مراتبی هستند، به این معنی که یک فضای نام خود نیز می تواند شامل چندین فضای نام دیگر باشد، که آنها نیز به نوبه خود کلاسهای داخل آن فضای نام را دسته بندی می کنند.
بیشتر کلاسهای موجود در چارچوب .NET در فضای نام System و یا فضای نامهای موجود در آن دسته بندی شده اند. برای مثال:

[1] NameSpace

- System.Data شامل کلاس هایی برای دسترسی به اطلاعات ذخیره شده در یک بانک اطلاعاتی است.
- System.Xml شامل کلاس هایی برای خواندن و نوشتن سندهای XML است.
- System.Windows.Forms شامل کلاس هایی برای ترسیم یک فرم و کنترلهای آن روی صفحه نمایش است.
- System.Net شامل کلاس هایی برای ایجاد ارتباطات شبکه ای در برنامه است.

هنگامی که یک کلاس در یک فضای نام قرار می گیرد، برای دسترسی به آن علاوه بر ذکر نام کلاس، باید فضای نام آن را نیز مشخص کنیم. البته تاکنون در برنامه ها از نام کوتاه شده آنها، یعنی فقط نام خود کلاس استفاده کرده و فضای نام آن را مشخص نمی کردید. به عبارت دیگر در قسمت قبلی وقتی گفتیم که تمام کلاسها از کلاس Object مشتق می شوند، در حقیقت نام کلاس را خلاصه کردیم. زیرا کلاس Object در فضای نام System قرار دارد و همه کلاسها در واقع از کلاس System.Object مشتق می شوند. همچنین کلاس Console در واقع نام خلاصه شده کلاس System.Console است و کد زیر:

```
Console.ReadLine();
```

با کد زیر معادل است:

```
System.Console.ReadLine();
```

هر کلاس فقط می تواند به یک فضای نام تعلق داشته باشد و همچنین نمی تواند عضو هیچ فضای نامی نباشد. به عبارت دیگر هر کلاس باید دقیقاً در یک فضای نام قرار داشته باشد. اما سوال این است که کلاس هایی که تاکنون می نوشتیم عضو چه فضای نامی بودند؟ خوب، اگر به قسمت ویرایشگر کد نگاه دقیقتری بیاندازید، خواهید دید که در بالاترین قسمت کد، بلاکی وجود دارد که با کلمه کلیدی namespace شروع می شود و در مقابل آن نام پروژه وارد شده است. برای مثال کلاس هایی که در این فصل در پروژه Objects ایجاد کردیم، همگی درون بلاکی به صورت زیر قرار گرفته بودند:

```
namespace Objects
{
    ...
}
```

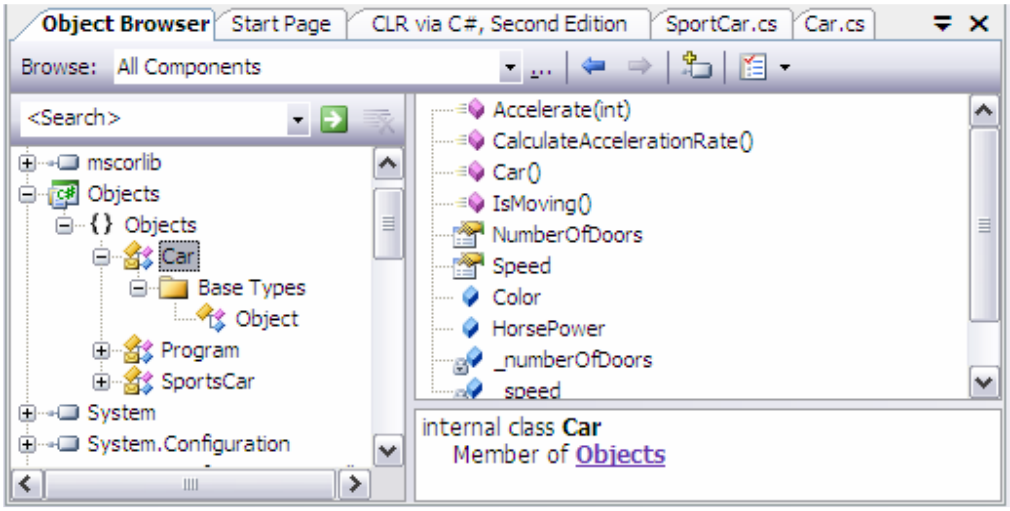
بنابراین نام اصلی کلاس Car به صورت Objects.Car و نام اصلی کلاس SportsCar به صورت Objects.SportsCar بوده است.
هدف اصلی ایجاد فضاهای نام در .NET ساده تر کردن استفاده از کلاسها برای کاربران آنها است. فرض کنید کلاسهای خود را بعد از تکمیل شدن در اختیار برنامه نویس دیگری قرار دهید تا از آنها در برنامه خود استفاده کند. اگر او نیز در برنامه ی خود کلاسی به نام Car ایجاد کرده بود، چگونه می تواند بین این دو کلاس تفاوت قائل شود؟
خوب، نام واقعی کلاس شما در حقیقت برابر با Objects.Car است و نام کلاس او نیز می تواند هر چیزی مانند MyOwnNameSpace.Car باشد. به این ترتیب احتمال اشتباه شدن این دو کلاس با یکدیگر از بین می رود.

نکته: البته دقت کنید که نام `Objects` که برای فضای نام این پروژه انتخاب کردیم، اصلاً نـام مناسـبی بـرای یـک دسـته از کلاسهای مرتبط به هم نیست، زیرا به این ترتیب نمی توان هیچ اطلاعاتی راجع به کلاسهای داخل این فضای نـام بدسـت آورد. در این برنامه فقط این نام را انتخاب کردیم که مشخص کننده برنامه های این فصل باشد.

برای آشنایی بیشتر با فضای نامهای موجود در `.NET` می توانید به ضمیمه ی ۲ مراجعه کنید.

راهنمای using:

همانطور که در فصل هفتم گفتم، برای استفاده از یک کلاس بدون ذکر فضای نام آن، باید فضای نـام آن کـلاس را بـا اسـتفاده از راهنمای `using`[1] به برنامه اضافه کنیم. اهمیت استفاده از این راهنما ممکن است هنگامی که در حال کار با برنامه های کوچـک هـستید، نمایـان نـشود، امـا فـرض کنیـد بخواهیـد بـه کلاسـی ماننـد `ServiceDescription` در فـضای نـام `System.Web.Services.Description` دسترسی پیدا کنید. واضح است که ذکر اسم کـلاس و فـضای نام آن در هر بار استفاده از این کلاس کار بسیار سختی است. بنابراین بهتر است که با اضافه کردن فضای نام آن، هر مرتبه فقط نام کلاس را ذکر کنید.
تمام فضای نامهایی که مـی خواهیـد بـه یـک برنامـه اضـافه کنیـد، بایـد در ابتـدای کـد و قبـل از هـر دسـتوری (قبـل از دسـتور `namespace` که برای مشخص کردن فضای نام برنامه به کار می رود)، با استفاده از `using` مشخص شوند.
تنها مشکلی که هنگام استفاده از این راهنما ممکن است رخ دهـد ایـن اسـت کـه در دو فـضای نـامی کـه بـا اسـتفاده از راهنمـای `using` به برنامه اضافه می کنید دو کلاس هم نام وجود داشته باشد. برای مثال فـرض کنیـد دو فـضای نـام `Objects` و `MyOwnNameSpace` که هر دو دارای کلاس `Car` هستند را با استفاده از راهنمای `using` به برنامه اضـافه کنیـد. در این صورت برای استفاده از این کلاس در برنامه باید نام کامل آن را مشخص کنید (برای مثال `Objects.Car`).
`Object Browser` یکی از ابزارهای خوب ویژوال استودیو برای مشاهده کلاس هایی که به برنامه شما اضافه شـده انـد، است. برای نمایش این ابزار می توانید از گزینه `View` ➔ `Object Browser` در نوار منو استفاده کنید (شـکل ۹-۱۳)



[1] `using directive` – توجه کنید که این کاربرد این کلمه در این حالت با کاربرد آن در برنامه به عنوان دستور `using` متفاوت است. کـاربرد دستور `using` در فصلهای بعدی توضیح داده خواهد شد.

شکل ۹-۱۳

توجه کنید که در این پنجره علاوه بر کلاسها، می توانید متدها، فیلد ها و خاصیتهای موجود در یک کلاس را نیز مشاهده کنید. همانطور که در شکل مشاهده می کنید، کلاس `Car` در فضای نامی که برابر با نام پروژه است قرار گرفته است (فضای نامها در شکل با علامت `{}` مشخص شده اند) و همچنین این کلاس از کلاس `Object` مشتق شده است. با وجود اینکه هنگام تعریف کلاس `Car` هیچ کلاس پایه ای برای آن مشخص نکردید، اما این کلاس به صورت اتوماتیک از کلاس `Object` مشتق شده است.

وراثت در چارچوب NET.:

همانطور که گفتم وراثت یکی از مباحث پیشرفته و مهم در برنامه نویسی شیئ گرا به شمار می رود. در این فصل به صورتی اجمالی با این مبحث آشنا شدیم، اما در چارچوب `.NET` به شدت از وراثت استفاده شده است. بنابراین بهتر است چند نکته اساسی دیگر را نیز در این رابطه ذکر کنیم.
یکی از مواردی که باید در این رابطه بدانید این است که در `.NET` هیچ کلاسی نمی تواند به صورت مستقیم از بیش از یک کلاس مشتق شود. همچنین ذکر شد که اگر برای یک کلاس، کلاس پایه مشخص نشود آن کلاس به صورت اتوماتیک از کلاس `Object` مشتق می شود. بنابراین می توانیم از این دو گفته نتیجه بگیریم که در `.NET` هر کلاس باید از دقیقاً یک کلاس مشتق شود.
البته اینکه می گوییم هر کلاس فقط می تواند از یک کلاس مشتق شود، منظور مشتق شدن مستقیم است نه مشتق شدن غیر مستقیم. برای مثال تصور کنید که در برنامه قبل کلاسی به نام `Porsche` ایجاد کنید که از کلاس `SportsCar` مشتق شود. این کلاس به صورت مستقیم از فقط یک کلاس به نام `SportsCar` و به صورت غیر مستقیم از کلاس `Car` مشتق شده است.

نتیجه:

در این فصل با مفاهیم ابتدایی برنامه نویسی شیئ گرا آشنا شدیم. فصل را با آموختن اینکه چگونه می توان یک کلاس با متدها و خاصیت های گوناگون ایجاد کرد شروع کردیم و سپس یک کلاس برای مدل کردن یک اتومبیل ایجاد کردیم. سپس متدها و خاصیتهای دیگری به این کلاس اضافه کردیم و آن را در یک برنامه به کار بردیم.
قبل از اینکه وارد بحث وراثت شویم، با بحث متدهای سازنده و نحوه کاربرد آنها آشنا شدیم. سپس در مبحث وراثت تعدادی از مباحث مهم طراحی شیئ گرا از قبیل چند شکلی بودن و `override` کردن را نیز بررسی کردیم.
در پایان این فصل باید با موارد زیر آشنا شده باشید:

- ایجاد خاصیت ها و متدهای مختلف در یک کلاس.
- ایجاد یک متد سازنده برای کلاس تا شرایط اولیه اشیای نمونه سازی شده از آن کلاس را تنظیم کند.
- ایجاد کلاس هایی که از کلاس دیگری مشتق شوند.
- `Override` کردن متدها و خاصیت های کلاس پایه در کلاس مشتق شده.
- مفهوم و چگونگی استفاده از فضای نام.

فصل دهم: مباحث پیشرفته برنامه نویسی شیئ گرا

در فصل نهم با مقدمات چگونگی ایجاد اشیا و نحوه طراحی یک کلاس آشنا شدید. قبل از آن فصل، از کلاسهای زیادی در چارچوب .NET برای ایجاد برنامه های خود استفاده می کردید. اما با اطلاعاتی که در فصل قبل بدست آوردیم نمی توان کلاس هایی کاربردی و قابل استفاده طراحی کرد. در این فصل با نحوه ایجاد کلاس هایی کارآمد بیشتر آشنا خواهیم شد.
در ابتدای این فصل بعد از معرفی تعدادی از مباحث باقی مانده برنامه نویسی شیئ گرا مانند interfaceها و نیز سربار گذاری متد ها، سعی می کنیم که به صورت عملی یک کلاس کاربردی ایجاد کنیم. برنامه ای که در این فصل بررسی می کنیم، همانند برنامه های قبلی به صورت یک لایه خواهد بود. ایده ایجاد برنامه های دو لایه در فصل سیزدهم بررسی خواهد شد. در طی ایجاد برنامه ی مثال در این فصل، با نحوه ایجاد خاصیت ها و متد هایی که در بین تمام اشیای یک کلاس مشترک باشند نیز آشنا خواهید شد.
در این فصل:

- چگونگی ایجاد چند تابع با نامهای یکسان و تعریفهای متفاوت را مشاهده خواهید کرد.
- با چگونگی ایجاد قالب خاص برای کلاس با استفاده از interfaceها آشنا خواهید شد.
- نحوه ی ایجاد کلاس های کاربردی در برنامه ها را بررسی خواهیم کرد.
- نحوه ایجاد خاصیت ها و متدهای مشترک در برنامه را مشاهده خواهیم کرد.

سربار گذاری متد ها:

یکی از کلاس هایی که در فصل هفتم با آن آشنا شدیم، کلاس MessageBox بود که به وسیله آن می توانستیم یک کادر پیغام را در صفحه نمایش دهیم. این کلاس شامل متد ها و خاصیت های زیادی بود، اما در برنامه ها فقط از یکی از متدهای آن استفاده می کردیم: متد Show. نکته جالبی که در این متد وجود داشت این بود که به هر صورتی که نیاز داشتیم، می توانستیم این متد را فراخوانی کنیم. برای مثال می توانستیم فقط یک متن به عنوان پارامتر به آن ارسال کنیم و تا آن متن در یک کادر نمایش داده شود، یا اینکه می توانستیم علاوه بر مشخص کردن متنی که باید نمایش داده شود، عنوان پنجره را نیز مشخص کنیم و یا .... اما سوال این است که معمولاً هنگام تعریف متد مشخص می کنیم که پارامترهای مورد نیاز آن چیست و به چه ترتیب باید به متد ارسال شود، پس چگونه در متد Show از کلاس MessageBox، تعداد پارامتر ها قابل تغییر بود؟
خوب، در هر برنامه می توانیم متد هایی ایجاد کنیم که نام یکسانی داشته باشند، اما پارامترهای مختلفی را دریافت کنند و کد های متفاوتی هم در آنها قرار دهیم. به این نوع تعریف متد، **سربار گذاری متد ها**[1] گفته می شود. برای مثال متد زیر را در نظر بگیرید. فرض می کنیم این متد عدد صحیحی را به عنوان پارامتر دریافت می کند و آن را در صفحه نمایش می دهد:

```
public void Dispaly(int DisplayValue)
{
    // Implementation omitted
}
```

[1] Method Overloading

فرض کنید می خواهید متدی داشته باشید که مانند این متد عمل کند ولی یک رشته ی متنی را به عنوان پارامتر دریافت کند و آن را نمایش دهد. در این صورت باید متدی جدید با نامی متفاوت ایجاد کنید. حال اگر بخواهید متدی با همین عملکرد داشته باشید ولی مدت زمان نمایش داده شدن عدد یا متن را نیز از کاربر دریافت کند و آن را در زمان مشخص شده نمایش دهد، در این صورت باید متد دیگری با نامی جدید ایجاد کنید. همانطور که می دانید، اگر تعداد حالتها زیاد شود این روش زیاد جالب نخواهد بود. همچنین ممکن است اسامی متد ها یا طولانی شود و یا بی معنی.

راه حل این مشکل استفاده از سربار گذاری در متد ها است[1]. متدهای سربار گذاری شده متد هایی هستند که نام یکسانی دارند اما می توانند نوع داده برگشتی از آنها و یا تعداد پارامترهای ورودی آنها مختلف باشد (حتی سطح دسترسی به آنها نیز مانند `public` و یا `private` بودن آنها نیز می تواند تفاوت داشته باشد). البته دقت کنید که متدهای سربار گذاری شده نمی توانند فقط از نظر نوع داده برگشتی تفاوت داشته باشند. به عبارت دیگر نمی توانید در یک کلاس، دو متد با یک نام و یک نوع پارامتر ورودی داشته باشید ولی نوع داده برگشتی آنها تفاوت داشته باشد، برای مثال یکی از آنها یک عدد صحیح برگرداند و دیگری یک رشته. تابع های سربار گذاری شده باید نام مشابه داشته باشند و تعداد و یا نوع پارامترهای آنها نیز حتماً متفاوت باشد. نوع داده برگشتی و سطح دسترسی آنها نیز هم می تواند متفاوت باشد و هم مشابه باشد.

در بخش امتحان کنید زیر نحوه استفاده از سربار گذاری در متد ها را مشاهده خواهیم کرد.

امتحان کنید: سربار گذاری توابع

۱) با استفاده از گزینه `File ➔ New ➔ Project...` در نوار منو یک پروژه جدید ایجاد کنید. در قسمت `Templates` از پنجره `New Project` گزینه `Console Application` را انتخاب کرده و نام پروژه را `Overloading Demo` وارد کنید. سپس روی دکمه `OK` کلیک کنید تا پروژه ایجاد شود.

۲) فایل `Program.cs` را باز کرده و متد زیر را در آن ایجاد کنید:

```
// A method for displaying an integer
static void Display(int I)
{
    Console.WriteLine("This is an integer: " +
                                I.ToString());
}
```

۳) حال می خواهیم متدی با همین نام داشته باشیم، ولی یک رشته را به عنوان ورودی دریافت کند. بنابراین متد زیر را به برنامه اضافه کنید:

```
// this method has the same name as the previous
// method, but is distinguishable by signature
static void Display(string str)
{
    Console.WriteLine("This is a string: " + str);
}
```

[1] به وسیله زبان C# می توانید عملگر هایی مانند + و یا – و ... را نیز سربار گذاری کنید. در این مورد در ادامه فصل توضیح داده شده است.

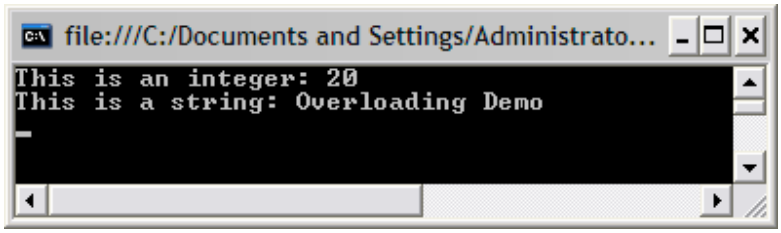
۴) حال متد Main را به صورت زیر تغییر دهید تا توابع ایجاد شده را بررسی کنیم:

```
static void Main(string[] args)
{
     // Displaying an integer
     Display(20);

     // Displaying a string
     Display("Overloading Demo");

     Console.ReadLine();
}
```

۵) برنامه را اجرا کنید. پنجره ای مشابه شکل ۱۰-۱ مشاهده خواهید کرد.



شکل ۱۰-۱

چگونه کار می کند؟

در ابتدای برنامه متدی برای نمایش یک عدد صحیح در خروجی ایجاد کردیم. این متد همانند متدهای عادی است و هیچ نکته خاصی در ایجاد کردن آن نباید در نظر گرفته شود. البته دقت داشته باشید، همانطور که در فصل قبل هم گفتم، این متد ها برای اینکه بتوانند توسط متد Main قابل دسترسی باشند باید از نوع static تعریف شوند.

```
// A method for displaying an integer
static void Display(int I)
{
     Console.WriteLine("This is an integer: " +
                                   I.ToString());
}
```

بعد از آن برای ایجاد متدی که بتواند یک رشته را نمایش دهد، از نام متد قبلی استفاده کردیم. ولی این بار پارامترهای ورودی آن را به صورت یک رشته تعریف کردیم.

```
     // this method has the same name as the previous
     // method, but is distinguishable by signature
     static void Display(string str)
```

```
    {
        Console.WriteLine("This is a string: " + str);
    }
```

به این صورت هنگام فراخوانی متد Display اگر یک عدد به آن ارسال کنیم، برنامه به صورت اتوماتیک متـد اول را فراخـوانی می کند و اگر هم یک رشته به آن ارسال کنیم برنامه از متد دوم استفاده می کند. این مورد را در هنگام اسـتفاده از ایـن متـد هـا در قسمت Main مشاهده می کنید. ابتدا عدد فرستاده شده به متد به همراه یک پیغام در صفحه چاپ می شود تا مـشخص شـود کـه برنامه از متد اول استفاده کرده است. در مرتبه دوم فراخوانی متد هم، رشته فرستاده شده به همراه پیغامی برای مـشخص شـدن متـد دوم در صفحه چاپ می شود.

```
static void Main(string[] args)
{
    // Displaying an integer
    Display(20);

    // Displaying a string
    Display("Overloading Demo");

    Console.ReadLine();
}
```

استفاده از خاصیت ها و متدهای Static

بعضی مواقع ممکن است بخواهید از توابع و متد هایی استفاده کنید که مختص به یک شیئ از کلاس نباشند، بلکه بـه کـل کـلاس اختصاص داشته باشند. تصور کنید می خواهید کلاسی طراحی کنید که نام کاربری و کلمه عبور را برای کاربران یـک برنامـه ذخیـره کند. برای این کار ممکن است از کدی مشابه زیر استفاده کنید:

```
    public class User
    {
        // Public members
        public string UserName;

        // Private members
        private string _password;
    }
```

حال تصور کنید که کلمه عبور هر کاربر نباید از تعداد کاراکتر های مشخصی (فرضاً ۶ کاراکتر) کمتر باشد. برای این کـار یـک فیلـد جدید در کلاس تعریف می کنید و حداقل تعداد کاراکتر های مجاز برای کلمه عبور را در آن ذخیره می کنید:

```
public class User
{
    // Public members
```

```
        public string UserName;

        // Private members
        private string _password;
        private string MinPasswordLength = 6;

        // Password property
        public string Password
        {
            get
            {
                return _password;
            }
            set
            {
                if (value.Length >= this.MinPasswordLength)
                    _password = value;
            }
        }
}
```

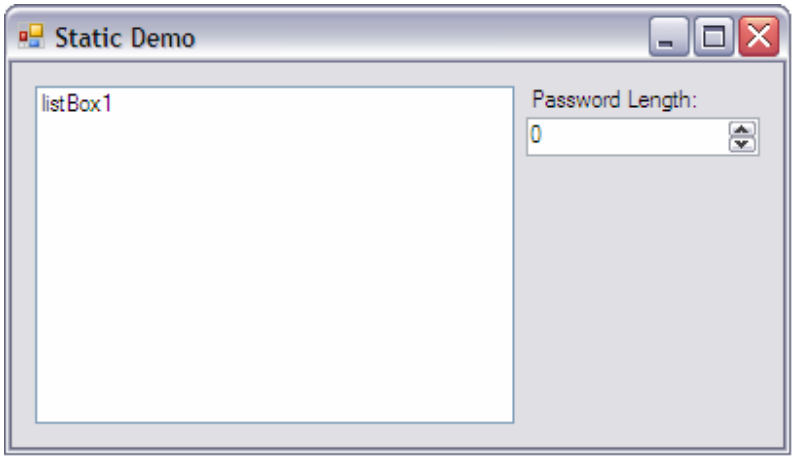
خوب، تاکنون همه چیز عادی و واضح به نظر می رسد. اما تصور کنید که برنامه شما ۵۰۰۰ کاربر داشته باشد، بنابراین ۵۰۰۰ شیئ از این کلاس باید در حافظه ایجاد شود و هر کدام از این شیئ ها نیز یک متغییر به نام `MinPasswordLength` دارند که ۴ بایت فضا را اشغال می کنند. به این صورت مشاهده می کنید که ۲۰ کیلو بایت از حافظه برای نگهداری یک عدد کوچک استفاده شده است. اگرچه در کامپیوترهای امروزی ۲۰ کیلو بایت فضای زیادی محسوب نمی شود، اما همانطور که در فصل سوم گفتم نباید بی دلیل حافظه را هدر داد. برای این کار روشهای بهتری نیز وجود دارد.

استفاده از خاصیت های Static:

در مثال قبلی ممکن است بخواهید مقدار حداقل کاراکتر های یک کلمه عبور را در یک فیلد از کلاس ذخیره کنید، سپس آن فیلد را در بین تمام اشیایی که از آن کلاس نمونه سازی می شوند، به اشتراک بگذارید. به این ترتیب بازای هر تعداد کاربری که در برنامه داشته باشید، فقط یک متغییر به نام `MinPasswordLength` ایجاد شده و ۴ بایت فضا اشغال می شود. در بخش امتحان کنید بعد، نحوه ی انجام این کار را مشاهده خواهیم کرد.

امتحان کنید: استفاده از خاصیت های `Static`

۱) ویژوال استودیو ۲۰۰۵ را باز کنید و پروژه ویندوزی جدیدی با ویژوال C# به نام `Static Demo` ایجاد کنید.
۲) هنگامی که قسمت طراحی `Form1` نمایش داده شد، عنوان فرم را به `Static Demo` تغییر دهید. سپس با استفاده از جعبه ابزار، یک کنترل `ListBox`، یک کنترل `Label` و یک کنترل `NumericUpDown` را در فرم قرار دهید. بعد از تغییر اندازه این کنترل ها، فرم شما باید مشابه شکل ۱۰-۲ باشد.



شکل ۱۰-۲

۳) خاصیت Name کنترل ListBox را به lstUsers تغییر دهید.

۴) خاصیت Name کنترل NumericUpDown را به nupMinPasswordLength، خاصیت Maximum آن را به 10 و خاصیت Value آن را به 6 تغییر دهید.

۵) با استفاده از پنجره Solution Explorer، کلاس جدیدی به نام User به برنامه اضافه کنید و کد مشخص شده در زیر را در آن قرار دهید:

```
class User
{
     // Public members
     public string Username;
     public static int MinPasswordLength = 6;

     // Private members
     private string _password;

     // Password property
     public string Password
     {
          get
          {
               return _password;
          }
          set
          {
               if (value.Length >= MinPasswordLength)
                    _password = value;
          }
     }
}
```

۶) به قسمت ویرایشگر کد مربوط به Form1 بروید و کد زیر را به آن اضافه کنید:

```
public partial class Form1 : Form
{
     // Private member
     private ArrayList arrUserList = new ArrayList();
```

همانطور که به خاطر دارید برای استفاده از کلاس ArrayList باید فضای نام System.Collections را با استفاده از using به برنامه ی خود اضافه کنید.

۷) حال، متد زیر را به کلاس مربوط به Form1 اضافه کنید:

```
private void UpdateDisplay()
{
     // Clear the list
     lstUsers.Items.Clear();

     // Add the users to the list box
     foreach (User objUsers in arrUserList)
     {
          lstUsers.Items.Add(objUsers.Username + ", " +
                             objUsers.Password + " (" +
                             User.MinPasswordLength + ")");
     }
}
```

۸) به قسمت طراحی مربوط به Form1 برگردید و بر روی فرم دو بار کلیک کنید. به این ترتیب متد مربوط به رویداد Load این فرم به صورت اتوماتیک ایجاد می شود. سپس کد مشخص شده در زیر را در این متد وارد کنید:

```
private void Form1_Load(object sender, EventArgs e)
{
     // Load 100 users
     for (int i = 0; i < 100; i++)
     {
          // Create a new user
          User objUser = new User();
          objUser.Username = "Robbin" + i;
          objUser.Password = "Password15";

          // Add the user to the array list
          arrUserList.Add(objUser);
     }
```

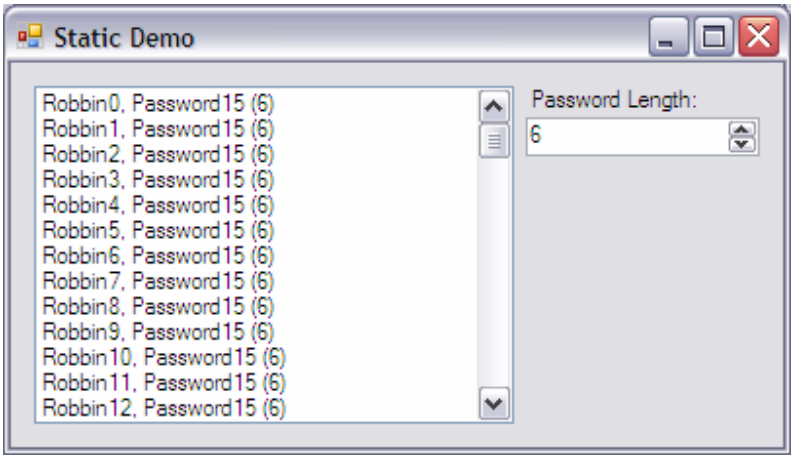
```
        // Update the display
        UpdateDisplay();
}
```

۹) مجدداً به قسمت طراحی `Form1` برگردید و روی کنترل `nupMinPasswordLength` دو بار کلیک کنید. به این ترتیب متد مربوط به رویداد `ValueChanged` این کنترل به صورت اتوماتیک ایجاد می شود. سپس کد زیر را در این متد وارد کنید:

```
private void nupMinPasswordLength_ValueChanged(
                            object sender, EventArgs e)
{
        // Set the minimum password length
        User.MinPasswordLength =
                        (int)nupMinPasswordLength.Value;

        // Update the display
        UpdateDisplay();
}
```

۱۰) حال می توانید برنامه را اجرا کنید. با اجرای برنامه باید با فرمی مشابه شکل ۱۰-۳ مواجه شوید.



شکل ۱۰-۳

۱۱) عدد موجود در کنترل `NumericUpDown` را کم و یا زیاد کنید. مشاهده خواهید کرد که اعداد داخل پرانتز در لیست نیز کم و یا زیاد می شوند.

چگونه کار می کند؟

برای اینکه یک فیلد، خاصیت و یا یک متد را در کلاس به عنوان عضو مشترک بین تمام اشیا مشخص کنیم، باید از کلمه کلیدی static استفاده کنیم.

```
public static int MinPasswordLength = 6;
```

به این ترتیب به ویژوال C# ۲۰۰۵ می گوییم که می خواهیم این عضو بین تمام اشیایی که از این کلاس نمونه سازی می شوند به اشتراک گذاشته شود.
اعضای static در یک کلاس می توانند به وسیله ی اعضای غیر static آن کلاس مورد استفاده قرار گیرند. برای مثال در برنامه ی بالا، عضو MinPasswordLength که یک عضو static است به وسیله ی خاصیت Password مورد استفاده قرار گرفت:

```
// Password property
public string Password
{
     get
     {
          return _password;
     }
     set
     {
          if (value.Length >= MinPasswordLength)
               _password = value;
     }
}
```

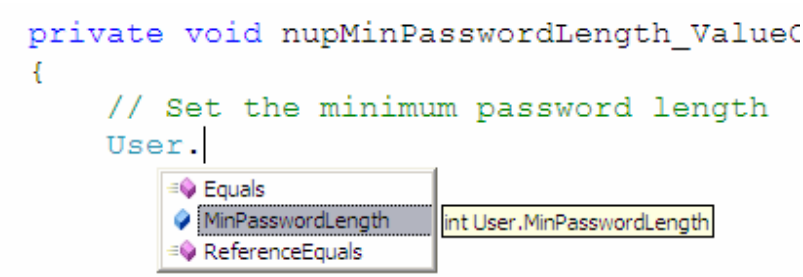
نکته مهمی که در اینجا همواره باید مدنظر داشته باشید این است که بر خلاف _password و Password که فقط به یک شیئ از کلاس User تعلق دارند (به عبارت دیگر بازای هر شیئ از این کلاس، یک فیلد _password و یک خاصیت Password وجود دارد)، اما MinPasswordLength به تمام اشیایی که از این کلاس ایجاد می شوند تعلق دارد، بنابراین اگر تغییری در این فیلد ایجاد شود، این تغییر بر تمام اشیای موجود اثر خواهد گذاشت.
در فرم برنامه از متد UpdateDisplay برای پر کردن لیست استفاده کرده ایم. کد های این متد هم کاملاً واضح هستند. فقط دقت داشته باشید که برای دسترسی به اعضای Static یک کلاس نمی توانید از اشیای نمونه سازی شده از آن کلاس استفاده کنید. برای مثال در متد UpdateDisplay، شیئ objUser حاوی فیلدی به نام MinPasswordLength نخواهد بود. برای دسترسی به این اعضا همانطور که مشاهده می کنید باید از نام خود کلاس استفاده کرد.

```
private void UpdateDisplay()
{
     // Clear the list
     lstUsers.Items.Clear();

     // Add the users to the list box
     foreach (User objUsers in arrUserList)
```

```
        {
            lstUsers.Items.Add(objUsers.Username + ", " +
                               objUsers.Password + " (" +
                               User.MinPasswordLength + ")");
        }
}
```

در شکل ۱۰-۴ نیز مشاهده می کنید که هنگام نوشتن کد، زمانی که نام کلاس را وارد می کنید ویژوال استودیو اعضای `static` کلاس را نمایش می دهد.



شکل ۱۰-۴

این موارد هنگامی که کاربر عدد موجود در کنترل `NumericUpDown` را تغییر می دهد جالب تر می شود. در این هنگام خاصیت `MinPasswordLength` تغییر می کند، اما چون این فیلد به شیئ خاصی تعلق ندارد نیازی نیست که این فیلد را برای تک تک اشیا تغییر دهید. بلکه کافی است با استفاده از نام کلاس، فقط یک بار مقدار این فیلد را عوض کنید تا کل اشیای نمونه سازی شده از کلاس تغییر کنند:

```
private void nupMinPasswordLength_ValueChanged(
                        object sender, EventArgs e)
{
    // Set the minimum password length
    User.MinPasswordLength =
                    (int)nupMinPasswordLength.Value;

    // Update the display
    UpdateDisplay();
}
```

البته مقدار خاصیت `Value` از کنترل `NumericUpDown` از نوع `Decimal` است و برای اینکه آن را در فیلد `MinPasswordLength` قرار دهیم باید آن را به یک عدد صحیح تبدیل کنیم. برای این کار همانطور که در فصل های قبلی گفتم باید از عملگر `()` به صورت زیر استفاده کنیم:

```
    // Set the minimum password length
    User.MinPasswordLength =
                    (int)nupMinPasswordLength.Value;
```

استفاده از متدهای Static:

در بخش قبلی مشاهده کردیم چگونه می توان یک فیلد `public` را به صورت اشتراکی معرفی کرد. در این قسمت هم به چگونگی ایجاد یک متد که بین تمام اشیا مشترک باشد خواهیم پرداخت. در بخش امتحان کنید بعدی از نوع `static` ایجاد خواهیم کرد که بتواند یک نمونه از کلاس `User` را ایجاد کند. تنها محدودیتی که این متد ها نسبت به متدهای همانند خود دارند در این است که این متد ها فقط می توانند به متد ها و خاصیت های `static` کلاس دسترسی داشته باشند.

نکته: مثال زیر فقط یک مثال تصنعی برای معرفی متدهای `static` است، زیرا این کار می تواند بدون استفاده از این نوع متد ها نیز به سادگی انجام شود.

امتحان کنید: استفاده از متدهای `static`

۱) قسمت ویرایشگر کد را برای کلاس `User` باز کنید و کد مشخص شده در زیر را به آن اضافه کنید

```
public static User CreateUser(string UserName,
                              string Password)
{
     // Declare a new User object
     User objUser = new User();

     // Set the user properties
     objUser.Username = UserName;
     objUser.Password = Password;

     // Return the new user
     return objUser;
}
```

۲) قسمت ویرایشگر کد را برای `Form1` باز کنید و به متد مربوط به رویداد `Load` فرم بروید. کد موجود در این متد را به صورتی که در زیر مشخص شده است تغییر دهید. دقت کنید هنگامی که نام کلاس `User` را وارد کنید، ویژوال استودیو متد `CreateUser` را نیز به عنوان یکی از گزینه های قابل انتخاب نمایش می دهد.

```
private void Form1_Load(object sender, EventArgs e)
{
     // Load 100 users
     for (int i = 0; i < 100; i++)
     {
          // Create a new user
          User objUser =
               User.CreateUser("Robbin" + i,"Password15");
```

```
            // Add the user to the array list
            arrUserList.Add(objUser);
        }

        // Update the display
        UpdateDisplay();
}
```

۳) برنامه را اجرا کنید. مشاهده خواهید کرد که در خروجی برنامه هیچ تغییری ایجاد نشده است.

چگونه کار می کند؟

نکته مهمی که در این مثال وجود دارد این است که با وارد کردن نام کلاس `User`، نام متد `CreateUser` نیز به عنوان یکی از گزینه های قابل انتخاب نمایش داده می شود. دلیل این مورد این است که این متد به عنوان یک متد `static` تعریف شده است و بین تمام اشیایی که از این کلاس نمونه سازی می شوند مشترک است. پس برای دسترسی به آن می توانید از نام خود کلاس استفاده کنید. برای تعریف یک متد `static`، می توانید در هنگام تعریف کلاس از کلمه کلیدی `static` استفاده کنید.

```
public static User CreateUser(string UserName,
                              string Password)
```

یکی از نکاتی که هنگام نوشتن متدهای `static` باید در نظر داشته باشید این است که این متد ها فقط می توانند به اعضای `static` کلاس دسترسی داشته باشند. دلیل این امر هم ساده است، زیرا این متد به شیئ خاصی وابسته نیست. بنابراین اگر در این متد از اعضای معمولی کلاس، که بازای هر شیئ یک نمونه از آنها وجود دارد استفاده شود، هنگام فراخوانی این متد نمی توان تشخیص داد که اعضای مربوط به کدام شیئ از کلاس مد نظر است.

سربار گذاری عملگر ها:

در برنامه هایی که تاکنون نوشته ایم از عملگر های استاندارد مانند + و یا – برای کار بر روی نوع های داده ای خاصی مانند اعداد صحیح، اعداد اعشاری و یا رشته ها استفاده می کردیم. این عملگر ها به صورت درونی برای کار با این نوع های داده ای برنامه ریزی شده اند. بعضی از زبانهای برنامه نویسی مانند C# به برنامه نویس اجازه می دهند که عملکرد این عملگر ها را به گونه ای تغییر دهند تا با ساختارها و یا کلاسهای نوشته شده توسط آنها نیز کار کنند.
برای مثال ممکن است بخواهیم ساختاری را ایجاد کنیم که بتواند اعداد مختلط را در خود نگهداری کند. همانطور که می دانید هر عدد مختلط از یک قسمت حقیقی و یک قسمت موهومی تشکیل شده است. فرض کنید در یک برنامه، کلاسی را برای نگهداری از اعداد مختلط به صورت زیر تعریف کرده اید:

```
public class ComplexNumber
{
```

```
        public double real;
        public double imaginary;
}
```

برای جمع دو شیئ از این کلاس، باید قسمتهای حقیقی هر کدام را با هم و قسمتهای موهومی را نیز با هم جمع کنیم. اما عملگر + به صورت عادی نمی تواند اشیایی که از این کلاس نمونه سازی می شوند را با یکدیگر جمع کند. در C# این قابلیت وجود دارد تا برای عملگر + تعریف کنیم که چگونه باید دو شیئ از این نوع را با یکدیگر جمع کند. بعد از این کار، این عملگر قادر خواهد بود همانطور که دو عدد عادی را با هم جمع می کند، دو شیئ از کلاس `ComplexNumber` را نیز با هم جمع کند. به این کار **سربار گذاری عملگر**[1] می گویند.

نکته: بسیاری از زبانهای برنامه نویسی مانند `Java` و یا `Visual Basic` اجازه نمی دهند که عملگر ها را سربار گذاری کنید. در این زبانها معمولاً از یک متد با نامی مشابه عملگر (همانند متد `Add`) برای انجام دادن آن عمل در کلاس استفاده می کنند.

قبل از اینکه بتوانیم با نحوه ی انجام این کار در C# آشنا شویم، باید عملگر ها را بهتر بشناسیم. پس ابتدا نکات دیگری که لازم است در مورد هر عملگر بدانید را بررسی کرده و سپس به سراغ نحوه ی سربار گذاری آنها می رویم.

درک عملگر ها:

هر عملگر در برنامه نویسی دارای اولویت خاصی است. برای مثال اولویت عملگر * بیشتر از عملگر + است. بنابراین برای ارزیابی مقدار عبارت `A + B * C`، ابتدا مقدار B در مقدار C ضرب شده و حاصل آن با مقدار A جمع می شود.
همچنین هر عملگر دارای شرکت پذیری خاصی است که مشخص می کند آن عملگر از سمت چپ به راست ارزیابی می شود و یا از سمت راست به چپ. برای مثال عملگر = دارای شرکت پذیری چپ است. به بیان دیگر برای ارزیابی مقدار عبارت `A = B = C` مقدار C در B و سپس در A قرار می گیرد.
عملگر **یگانی** عملگری است که فقط بر روی یک عملوند، کار می کند. برای مثال عملگر ++ که برای اضافه کردن یک واحد به متغیر به کار می رود، فقط به یک عملوند نیاز دارد.
عملگر **دوتایی** نیز عملگری است که بر روی دو عملوند کار می کند. برای مثال عملگر + برای کار خود به دو عملوند نیاز دارد.

هنگام سربار گذاری عملگر ها باید همواره نکات زیر را در نظر داشته باشید:

- هنگام سربار گذاری عملگر ها نمی توانید اولویت و یا ترتیب شرکت پذیری آنها را تغییر دهید. زیرا این موارد به مفهوم عملگر بستگی دارند و بر اساس نوع داده ای که با آنها مورد استفاده قرار می گیرد نباید تغییر کنند. برای مثال عبارت `A + B * C` صرف نظر از نوع داده ای متغییر های A ، B و C همواره به صورت `A + (B * C)` ارزیابی می شود.

[1] `Operator Overloading`

- هنگام سربار گذاری عملگر ها مجاز نیستید که تعداد عملوند های مورد نیاز آن را تغییر دهید. برای مثال عملگر * همواره دو عملوند دریافت می کند و نمی توانید عملکرد آن را برای یک کلاس به گونه ای تغییر دهید که به سه عملوند نیاز داشته باشد.
- هنگام نوشتن یک برنامه مجاز نیستید که عملگر های جدیدی را ایجاد کرده و مفهومی را به آن اختصاص دهید. برای مثال نمی توانید عملگری را به صورت ** تعریف کنید.

چگونگی سربار گذاری عملگر ها:

در بخش امتحان کنید زیر، کلاسی را برای نگهداری اعداد مختلط ایجاد کرده و عملگر های مربوط به جمع و تفریق را به نحوی تغییر می دهیم که بتوانند با اشیای ساخته شده از این کلاس نیز کار کنند. سپس برنامه ای ایجاد می کنیم که دو عدد مختلط را از کاربر دریافت کند و حاصل جمع و تفریق آنها را نمایش دهد.

نکته: یک عدد مختلط را به صورت `(R+Ii)` نمایش می دهند که `R` قسمت حقیقی آن و `I` قسمت موهومی آن است، مانند (`2+3.4i`) . اگر قسمت حقیقی عدد مختلط اول را با `R1` و قسمت موهومی آن را با `I1` نمایش دهیم، همچنین برای نمایش قسمتهای حقیقی و موهومی عدد دوم نیز از `R2` و `I2` استفاده کنیم، فرمول جمع و تفریق این اعداد به صورت زیر خواهد بود:

```
جمع:  (R1+R2) + (I1+I2)i
تفریق: (R1-R2) + (I1-I2)i
```

امتحان کنید: سربار گذاری عملگر ها

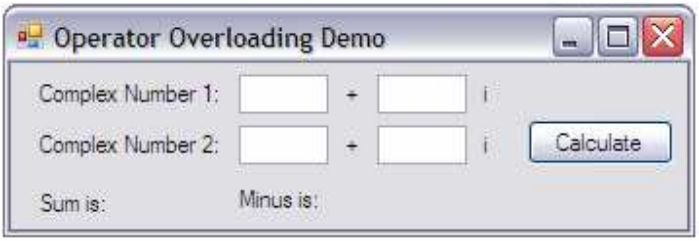
۱) با استفاده از گزینه ی ...`File ➔ New ➔ Project` در ویژوال استودیو، برنامه ی ویندوزی جدیدی به نام `Operator Overloading Demo` ایجاد کنید.

۲) بعد از اینکه فرم برنامه نمایش داده شد، خاصیت `Text` مربوط به آن را برابر با `Operator Overloading Demo` قرار دهید.

۳) با استفاده از جعبه ابزار، چهار کنترل `TextBox` بر روی فرم قرار داده و نام آنها را به ترتیب به `txtReal1`، `txtImg1`، `txtReal2` و `txtImg2` تغییر دهید. همچنین با استفاده از چند کنترل `Label` مشخص کنید که هر کنترل `TextBox` مربوط به نگهداری چه مقداری است.

۴) یک کنترل `Button` روی فرم قرار داده، خاصیت `Name` آن را به `btnCalculate` و خاصیت `Text` آن را به `Calculate` تغییر دهید.

۵) دو کنترل `Label` به نامهای `lblSum` و `lblMinus` در فرم قرار دهید تا نتیجه محاسبات را به وسیله آنها به کاربر اطلاع دهیم. خاصیت `Text` این دو کنترل را به ترتیب برابر با `Sum is:` و `Minus is:` قرار دهید. بعد از تنظیم اندازه کنترل ها فرم شما باشد مشابه شکل ۱۰-۵ باشد.



شکل ۱۰-۵

۶) با استفاده از پنجره Solution Explorer کلاس جدیدی به نـام ComplexNumber بـه برنامـه اضافه کرده، سپس کد مشخص شده در زیر را در بدنه ی آن کلاس وارد کنید.

```
class ComplexNumber
{
     public double Real = 0;
     public double Imaginary = 0;

     // Override ToString() to display a complex number
     // in the traditional format
     public override string ToString()
     {
          return (Real + " + " + Imaginary + "i");
     }
}
```

۷) برای اینکه عملگر جمع (+) را برای این کلاس سربار گذاری کنیم، متد زیـر را بـه کـلاس ComplexNumber اضافه کنید:

```
// Overloading '+' operator:
public static ComplexNumber operator +(ComplexNumber a,
                                        ComplexNumber b)
{
     // Create a new ComplexNumber object to store the sum
     ComplexNumber sum = new ComplexNumber();

     // Calculate the sum
     sum.Real = a.Real + b.Real;
     sum.Imaginary = a.Imaginary + b.Imaginary;

     return sum;
}
```

۸) برای سربار گذاری عملگر تفریق (-) نیز باید متد زیر را به کلاس ComplexNumber اضافه کنید:

```
// Overloading '-' operator:
```

```
public static ComplexNumber operator -(ComplexNumber a,
                                       ComplexNumber b)
{
     // Create a new ComplexNumber object to store the
minus
     ComplexNumber minus = new ComplexNumber();

     // Calculate the minus
     minus.Real = a.Real - b.Real;
     minus.Imaginary = a.Imaginary - b.Imaginary;

     return minus;
}
```

۹) برای محاسبه ی جمع و تفریق دو عدد مختلط ای که کاربر در برنامه وارد کرده است، به چهار شیئ از نوع `ComplexNumber` نیاز داریم. بر روی کنترل `btnCalculate` دو بار کلیک کنید تا متد مربوط به رویداد `Click` آن ایجاد شود. سپس کد زیر را در آن وارد کنید:

```
private void btnCalculate_Click(object sender, EventArgs e)
{
     ComplexNumber number1 = new ComplexNumber();
     ComplexNumber number2 = new ComplexNumber();

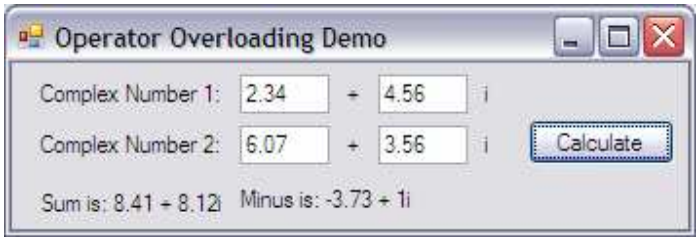
     number1.Real = double.Parse(txtReal1.Text);
     number1.Imaginary = double.Parse(txtImg1.Text);
     number2.Real = double.Parse(txtReal2.Text);
     number2.Imaginary = double.Parse(txtImg2.Text);

     ComplexNumber sum = new ComplexNumber();
     ComplexNumber minus = new ComplexNumber();

     sum = number1 + number2;
     minus = number1 - number2;

     lblSum.Text = "Sum is: " + sum.ToString();
     lblMinus.Text = "Minus is: " + minus.ToString();
}
```

۱۰) برنامه را اجرا کرده و دو عدد مختلط را در کادرهای متنی فرم وارد کنید. سپس بر روی دکمه ی `Calculate` کلیک کنید. نتیجه ای مشابه شکل ۱۰-۶ مشاهده خواهید کرد:



شکل ۱۰-۶

چگونه کار می کند؟

برنامه را با ایجاد کلاسی برای نگهداری اعداد مختلط آغاز می کنیم. برای نگهداری این اعداد به دو فیلد، یکی برای نگهداری قسمت حقیقی و دیگری برای نگهداری قسمت موهومی عدد نیاز داریم. بعد از ایجاد این دو فیلد، برای ایکه بتوانیم راحت تر این اعداد را نمایش دهیم، متد `ToString` را به گونه ای `override` می کنیم تا این اعداد را به فرم عادی نمایش دهد. برای این کار از روشی که در فصل قبلی مشاهده کردید استفاده می کنیم:

```
// Override ToString() to display a complex number
// in the traditional format
public override string ToString()
{
     return (Real + " + " + Imaginary + "i");
}
```

در مرحله ی بعد باید عملگر های جمع و تفریق را برای این کلاس سربار گذاری کنیم. برای سربار گذاری یک عملگر باید متدی با یک قالب خاص به صورت `static` و `public` تعریف کنیم. نام این متد باید با کلمه ی کلیدی `operator` شروع شده و سپس عملگر مورد نظر را وارد کرد. برای مثال نام متدی که برای سربار گذاری عملگر جمع به کار می رود باید برابر با `operator +` باشد.

پارامترهای این متد، عملوند هایی هستند که برای آن عملگر لازم است. مثلا عملگر جمع به دو عملوند نیاز دارد تا بتواند آن را با یکدیگر جمع کند، پس باید دو پارامتر را برای متد مربوط به سربار گذاری این عملگر مشخص کنیم. خروجی این متد هم باید هم نوع با نتیجه ی آن عملگر باشد. برای مثال در عملگر جمع، خروجی متد مربوط به سربار گذاری آن باید برابر با نوع متغیری باشد که به عنوان حاصل جمع مشخص می شود.

در برنامه قبل برای اینکه عملگر `+` را سربار گذاری کنیم، باید متدی به صورت زیر تعریف کنیم:

```
public static ComplexNumber operator +(ComplexNumber a,
                                       ComplexNumber b)
```

همانطور که گفتم این متد باید به صورت `static` و `public` تعریف شود. همچنین در نام آن نیز باید از کلمه ی کلیدی `operator` و عملگری که می خواهیم سربار گذاری شود استفاده کنیم (در اینجا `operator +`). عملوند هایی که در این قسمت به عملگر `+` فرستاده می شوند، هر دو از نوع عدد مختلط هستند. به بیان بهتر هر دوی عملوند ها، شیئ ای از نوع `ComplexNumber` هستند. پس پارامترهای ورودی متد نیز از نوع `ComplexNumber` خواهند بود.

توجه کنید که اگر می خواستیم عملگر + را به گونه ای سربار گذاری کنیم که بتوانیم یک شیئ از نوع ComplexNumber را با یک عدد صحیح جمع کنیم، باید در متد را به گونه ای تعریف می کردیم که یک شیئ از نوع ComplexNumber و یک عدد از نوع int را به عنوان ورودی دریافت کند. به این ترتیب می توانستیم از دستوری مانند زیر برای جمع یک عدد مختلط با یک عدد صحیح استفاده کنیم:

```
Sum = number1 + 20;
```

بعد از اینکه دو شیئ از نوع ComplexNumber را با یکدیگر جمع کردیم، نتیجه نیز از نوع ComplexNumber خواهد بود، پس باید خروجی (مقدار برگشتی متد) را از نوع ComplexNumber مشخص کنیم. سپس می توانیم نحوه جمع کردن دو شیئ از نوع مشخص شده را در بدنه ی متد مشخص کنیم. در این جا برای جمع دو مقدار فرستاده شده، شیئ جدیدی از نوع ComplexNumber تعریف کرده، قسمتهای حقیقی دو عدد را با یکدیگر و قسمتهای موهومی آن را نیز با هم جمع کرده حاصل را در شیئ ComplexNumber جدید قرار می دهیم. سپس این شیئ را به عنوان نتیجه برمی گردانیم.

```
// Overloading '+' operator:
public static ComplexNumber operator +(ComplexNumber a,
                                        ComplexNumber b)
{
      // Create a new ComplexNumber object to store the sum
      ComplexNumber sum = new ComplexNumber();

      // Calculate the sum
      sum.Real = a.Real + b.Real;
      sum.Imaginary = a.Imaginary + b.Imaginary;

      return sum;
}
```

برای سربار گذاری عملگر – نیز از همین روش استفاده می کنیم. این عملگر نیز مانند + دو شیئ از نوع ComplexNumber را دریافت کرده و حاصل را نیز به صورت ComplexNumber برمی گرداند.

```
// Overloading '-' operator:
public static ComplexNumber operator -(ComplexNumber a,
                                        ComplexNumber b)
{
      // Create a new ComplexNumber object to store the
      // minus
      ComplexNumber minus = new ComplexNumber();

      // Calculate the minus
      minus.Real = a.Real - b.Real;
      minus.Imaginary = a.Imaginary - b.Imaginary;

      return minus;
```

```
}
```

با اضافه کردن این دو متد به کلاس ComplexNumber، این کلاس دیگر کامل شده است و می توانیم از آن در برنامه استفاده کنیم. در صفحه ی اصلی برنامه چهار کادر متنی برای دریافت قسمتهای موهومی و حقیقی دو عدد مختلطی که باید با هم جمع و تفریق شوند قرار داده ایم. اما همانطور که می دانید این اعداد به صورت متن در این کادر ها وارد می شوند، پس ابتدا آنها را به عدد تبدیل کنیم. در فصل سوم گفتم که برای تبدیل یک رشته ی حاوی عدد به یک عدد باید از متد Parse در کلاس مربوط به نوع داده ای آن عدد استفاده کنیم. در این قسمت هم، به این علت که می خواهیم به کاربر اجازه دهیم بتواند اعداد اعشاری نیز وارد کند، نوع داده ای اعداد موجود در کادر ها را برابر با double در نظر گرفته و از تابع Parse مربوط به آن استفاده می کنیم تا متن وارد شده درون کادر به عددی از نوع Double تبدیل شود.

```
number1.Real = double.Parse(txtReal1.Text);
number1.Imaginary = double.Parse(txtImg1.Text);
number2.Real = double.Parse(txtReal2.Text);
number2.Imaginary = double.Parse(txtImg2.Text);
```

حال که اشیای مربوط به نگهداری اعداد مختلط وارد شده به وسیله کاربر را ایجاد کردیم، دو شیئ جدید از نوع ComplexNumber ایجاد می کنیم تا حاصل جمع و همچنین حاصل تفریق این اعداد را در آنها قرار دهیم.

```
ComplexNumber sum = new ComplexNumber();
ComplexNumber minus = new ComplexNumber();
```

بعد از تعریف این دو شیئ به قسمت جالب برنامه می رسیم. به نحوه استفاده از عملگر جمع برای جمع کردن دو شیئ از نوع ComplexNumber توجه کنید. هنگامی که برنامه به این دستور می رسد، متد operator + را فراخوانی کرده، متغییر سمت چپ علامت + را به عنوان پارامتر اول و متغییر سمت راست را به عنوان پارامتر دوم به متد می فرستد. مقدار برگشتی متد را نیز به عنوان حاصل عبارت در متغییر sum قرار می دهد. سپس همین عمل را برای قسمت تفریق نیز انجام می دهد و حاصل عبارت مربوط به آن را در متغییر minus قرار می دهد. به این ترتیب متغییر sum حاوی نتیجه ی متد operator + (یا همان حاصل جمع دو عدد مختلط) و متغییر minus حاوی نتیجه متد operator – (یا همان حاصل تفریق دو عدد مختلط) خواهد بود.

```
sum = number1 + number2;
minus = number1 - number2;
```

بعد از محاسبه ی این دو مقدار، باید نتیجه را در خروجی چاپ کنیم. اما برای این کار نیازی نیست که از فیلد های کلاس ComplexNumber استفاده کنیم. همانطور که خاطر دارید در ابتدای برنامه، متد ToString را به گونه ای override کردیم که عدد مختلط مربوط به شیئ را به صورت مطلوب نمایش دهد. بنابراین در این قسمت کافی است برای هر یک از اشیای sum و minus این متد را فراخوانی کرده و حاصل را در یک لیبل نمایش دهیم.

```
lblSum.Text = "Sum is: " + sum.ToString();
lblMinus.Text = "Minus is: " + minus.ToString();
```

در این مثال می توانستیم همین کار را با استفاده از متدهای عادی مانند یک متد Add نیز انجام دهیم، اما استفاده از این روش سادگی برنامه نویسی و نیز خوانایی کد را افزایش می دهد. برای مثال تصور کنید بخواهید چهار شیئ A، B، C و D را با یکدیگر جمع کنید. در این صورت اگر بخواهید از متدهای عادی استفاده کنید باید فرضاً چهار مرتبه متد Add را فراخوانی کنید، اما با استفاده از سربار گذاری عملگر جمع می توانید به راحتی از دستور A+B+C+D استفاده کنید.

کلاسهای Abstract:

همانطور که می دانید تمام کنترل هایی که در برنامه های خود از آنها استفاده می کنید در حقیقت یک شیئ از کلاسی مربوط به آن کنترل هستند. برای مثال هنگامی که یک کنترل Button روی فرم برنامه قرار می دهید، در حقیقت یک شیئ از کلاسی به نام Button ایجاد کرده اید و در طی برنامه نیز با آن شیئ کار می کنید.
فرض کنید بخواهید کلاس مربوط به چنین کنترل هایی را در برنامه بنویسید. برای این کار ابتدا باید یک کلاس پایه، برای مثال به نام Window ایجاد کرده و تمام خاصیت ها و متدهای مشترک بین کنترل ها را در این کلاس قرار دهید. سپس تمام کنترل هایی که می خواهید ایجاد کنید را، مانند کنترل Button و یا کنترل TextBox، از این کلاس به ارث ببرید. در این حالت مسلماً نمی خواهید بعدها کسی بتواند شیئ را از کلاس Window نمونه سازی کند و از آن شیئ در برنامه استفاده کند، چون این امر بی معنی است. کلاس Window نشان دهنده هیچ کنترل خاصی نیست، بلکه فقط به عنوان یک کلاس پایه برای تمام کنترل ها به کار می رود. بنابراین این کلاس را به عنوان یک کلاس abstract مشخص می کنید. کلاس های abstract به بیان ساده تر کلاس هایی هستند که نمی توان هیچ شیئ را از آنها نمونه سازی کرد و حتماً باید به عنوان کلاس های پایه برای دیگر کلاسها مورد استفاده قرار گیرند.
حال تصور کنید که می خواهید تمام کنترل هایی که از کلاس Window مشتق می شوند دارای متدی به نام DrawWindow باشند که آن کنترل را در صفحه رسم کند. برای این کار باید متدی به نام DrawWindow در کلاس Window ایجاد کنید. اما نحوه رسم هر کنترل به نوع آن بستگی دارد و هیچ نقطه ی اشتراکی در این مورد بین کنترل ها وجود ندارد که بخواهید آن را در بدنه ی این متد قرار دهید. به عبارت دیگر این متد نمی تواند شامل هیچ کدی باشد. همچنین می خواهید هر کنترلی که با مشتق شدن از کلاس Window ایجاد می شود، حتماً این متد را در خود override کند تا به این وسیله نحوه ی رسم آن کنترل در صفحه مشخص شود. برای این کار می توانید آن متد را از نوع abstract مشخص کنید. به این ترتیب هر کلاسی که از کلاس Window به عنوان کلاس پایه استفاده کند موظف است که تمام اعضای abstract آن کلاس را در خود override کند. البته دقت داشته باشید که یک عضو abstract فقط می تواند در کلاسهای abstract ایجاد شود. به عبارت دیگر نمی توانید در یک کلاس عادی، یک عضو abstract ایجاد کنید.
در بخش امتحان کنید بعد سعی می کنیم مثالی که در بالا عنوان شد را به گونه ای ساده پیاده سازی کنیم.

امتحان کنید: کلاسهای Abstract

۱) با استفاده از گزینه ی ...File ➔ New ➔ Project در نوار منوی ویژوال استودیو، یک پروژه تحت کنسول جدید با ویژوال C# به نام Abstract Demo ایجاد کنید.
۲) با استفاده از پنجره ی Solution Explorer یک کلاس جدید به نام Window به برنامه اضافه کنید.
۳) تعریف کلاس Window را به صورت زیر تغییر دهید:

```
namespace Abstract_Demo
{
     abstract public class Window
     {
```

۴) کد مشخص شده در زیر را به کلاس Window اضافه کنید. دقت کنید متدی که در این کلاس به صورت abstract تعریف شده است نباید حاوی کد باشد.

```
abstract public class Window
{
     public int Left;
     public int Top;

     // Constructor method to set
     // the initial value of fields
     public Window()
     {
          this.Top = 0;
          this.Left = 0;
     }

     // simulates drawing the window
     // notice: no implementation
     abstract public void DrawWindow();
}
```

۵) در داخل فضای نام Abstract_Demo و بعد از کلاس Window، کلاس جدیدی به نام ListBox به صورت زیر تعریف کنید. این کلاس از کلاس Window مشتق می شود و به صورت فرضی یک لیست باکس را در صفحه نمایش می دهد. در مورد تداخل نام این کلاس با کلاس ListBox در ویژوال استودیو نگران نباشید، زیرا کلاس ListBox که در این قسمت تعریف می کنید در فضای نام Abstract_Demo است ولی کلاس ListBox ویژوال استودیو در فضای نام System.Windows.Forms است.

```
// ListBox derives from Window
public class ListBox : Window
{
     private string listBoxContents; // new member variable

     // constructor adds a parameter
     public ListBox(int top,int left,string contents)
     {
          Top = top;
          Left = left;
          listBoxContents = contents;
     }
```

```
        // an overridden version implementing the
        // abstract method
        public override void DrawWindow()
        {
            Console.WriteLine("Writing string to the" +
                        " listbox: " + listBoxContents);
        }
}
```

۶) حال می خواهیم کنترل دیگری نیز مانند یک کنترل Button، با استفاده از کلاس Window ایجاد کنیم. برای این کار کلاس دیگری را بعد از کلاس ListBox در فضای نام Abstract_Demo به صورت زیر ایجاد کنید:

```
// Button control that derives from Window class
public class Button : Window
{
        // The new class's constructor
        public Button(int top, int left)
        {
            Top = top;
            Left = left;
        }

        // implement the abstract method
        public override void DrawWindow()
        {
            Console.WriteLine("Drawing a button at " + Top
                                    + ", " + Left);
        }
}
```

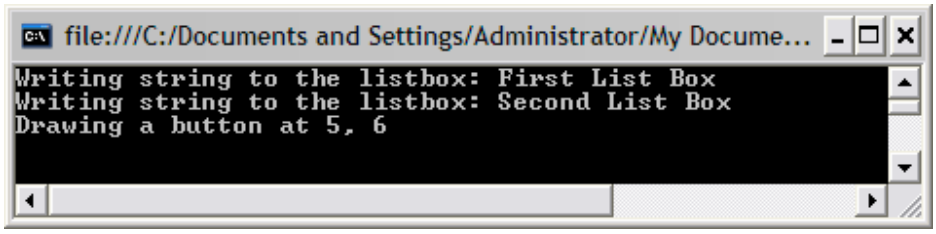
۷) ایجاد کلاسهای مورد نیاز برای برنامه به پایان رسید و حالا باید نحوه عملکرد آنها را بررسی کنیم. ویرایشگر کد برای فایل Program.cs را باز کرده و کد زیر را در متد Main وارد کنید.

```
static void Main(string[] args)
{
        // Create two list boxes and one button
        ListBox lstBox1 = new ListBox(1, 2, "First List Box");
        ListBox lstBox2 = new ListBox(3, 4,"Second List Box");
        Button newButton = new Button(5, 6);

        // Draw all objects on the form
        lstBox1.DrawWindow();
        lstBox2.DrawWindow();
        newButton.DrawWindow();
```

```
        Console.ReadLine();
}
```

۸) حال برنامه را اجرا کنید، نتیجه ای مشابه شکل ۱۰-۷ را مشاهده خواهید کرد.



شکل ۱۰-۷

چگونه کار می کند؟

برنامه را با تعریف یک کلاس پایه به نام `Window` شروع می کنیم. همانطور که گفتم این کلاس باید به گونه ای باشد که تمام خاصیت ها و متد هایی را که بین تمام کنترل ها مشترک است شامل شود. همچنین برای اینکه نتوان شیئ را از این کلاس ایجاد کرد، آن را به صورت `abstract` تعریف می کنیم. برای این کار کافی است هنگام تعریف کلاس، کلمه ی کلیدی `abstract` را به ابتدای آن اضافه کنیم.

```
abstract public class Window
```

تمام کنترل ها در یک فرم دارای موقعیت مکانی معینی هستند که به وسیله دو خاصیت `Top` (فاصله ی کنترل از بالای فرم) و `Left` (فاصله ی کنترل از سمت چپ فرم) مشخص می شوند. پس همه ی آنها باید شامل دو فیلد به نامهای `Top` و `Left` باشند. به همین دلیل این دو فیلد را در کلاس `Window` قرار می دهیم.

```
public int Left;
public int Top;
```

همچنین می خواهیم تمام کنترل هایی که از این کلاس مشتق می شوند، حتماً متدی به نام `DrawWindow` داشته باشند تا آنها را در فرم رسم کند. اما همانطور که گفتم این متد برای هر کنترل متفاوت است و نمی توانیم پیاده سازی آن را در کلاس `Window` قرار دهیم، بنابراین آن را در کلاس پایه به صورت `abstract` معرفی می کنیم تا تمام کلاس هایی که از کلاس `Window` مشتق می شوند چنین متدی را در خود ایجاد کنند. به این صورت می توانیم تضمین کنیم که تمام کنترل هایی که از کلاس `Window` مشتق می شوند دارای متدی به نام `DrawWindow` هستند که آنها را در صفحه رسم می کند. دقت کنید که اگر متدی در یک کلاس به عنوان `abstract` معرفی شود، آن متد نباید شامل هیچ دستوری باشد. به عبارت دیگر آن متد نباید دارای پیاده سازی باشد.

```
// Simulates drawing the window
// Notice: NO implementation
abstract public void DrawWindow();
```

در انتها نیز یک متد سازنده برای این کلاس ایجاد می کنیم تا مقدار اولیه فیلد های Top و Left را مشخص کند.

```
// Constructor method to set
// the initial value of fields
public Window()
{
     this.Top = 0;
     this.Left = 0;
}
```

حال باید با استفاده از کلاس Window، کنترلهای مورد نظرمان را ایجاد کنیم. ابتدا از کنترلی همانند ListBox شروع می کنیم. برای این کار کلاسی به نام ListBox ایجاد کرده و کلاس Window را به عنوان کلاس پایه ی آن مشخص می کنیم. همانطور که در بخش فضای نام گفتم، نام کامل این کلاس به صورت Abstract_Demo.ListBox است، بنابراین با کلاس ListBox مربوط به ویژوال استودیو که به نام System.Windows.Forms.ListBox است اشتباه نخواهد شد.
در این کلاس فیلد جدیدی به نام listBoxContents از نوع رشته ای ایجاد می کنیم تا علاوه بر موقعیت هر لیست باکس، محتویات آن را نیز بتوانیم نگهداری کنیم و در مواقع مورد نیاز نمایش دهیم.

```
// ListBox derives from Window
public class ListBox : Window
{
     private string listBoxContents; // new member variable
```

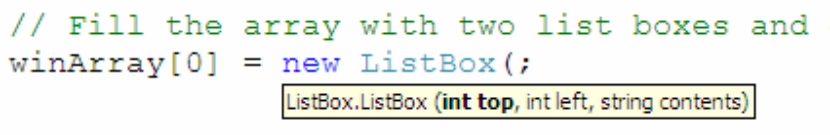
همچنین یک متد سازنده نیز در این کلاس قرار می دهیم تا مقدار فیلد ها را تنظیم کند. همانطور که مشاهده می کنید این متد سازنده همانند متدهای معمولی سه پارامتر می گیرد: پارامتر اول برای تعیین فاصله ی کنترل از سمت چپ (مقدار فیلد Top)، پارامتر دوم برای تعیین فاصله ی کنترل از بالای فرم (مقدار فیلد Left) و پارامتر سوم برای تعیین متنی که باید در لیست باکس قرار گیرد (مقدار خاصیت listBoxContents). به این ترتیب می توانیم در همان ابتدای ایجاد شیئ از کاربر بخواهیم که مقدار این موارد را تعیین کند.

```
     // constructor adds a parameter
     public ListBox(int top,int left,string contents)
     {
          Top = top;
          Left = left;
          listBoxContents = contents;
     }
```

نکته: اگر متد سازنده ی یک کلاس پارامتر داشته باشد، هنگام نمونه سازی یک شیئ از آن کلاس با استفاده از دستور new باید مقدار آن پارامتر ها را به کلاس فرستاد. برای مثال به نحوه نمونه سازی اشیای کلاس ListBox در متد Main توجه کنید. بعد از دستور new، در مقابل اسم کلاس مقدارهای لازم برای ایجاد کلاس نیز به ان فرستاده شده اند.

```
ListBox lstBox1 = new ListBox(1, 2, "First List Box");
```

البته هنگامی که نام کلاس را بعد از دستور new در این قسمت وارد کردید، ویژوال استودیو مشخص می کند که برای نمونه سازی یک شیئ از این کلاس، چه پارامترهایی را باید مشخص کنید (شکل ۱۰-۸)



شکل ۱۰-۸

هنوز کلاس ListBox کامل نشده است و اگر در این مرحله آن را کامپایل کنیم با خطا مواجه خواهیم شد. زیرا هنوز متد DrawWindow را در این کلاس override نکرده ایم و در کلاس Window هم این متد به صورت abstract مشخص شده است، پس باید در کلاس مشتق شده override شود. بنابراین همانند override کردن متدهای عادی که در قسمتهای قبل مشاهده کرده اید، متد DrawWindow از کلاس پایه را در این کلاس override می کنیم.

```
// an overridden version implementing the
// abstract method
public override void DrawWindow()
{
     Console.WriteLine("Writing string to the" +
               " listbox: " + listBoxContents);
}
```

البته در این قسمت درگیر نحوه ی پیاده سازی این متد نمی شویم، بلکه فقط می خواهیم نحوه override کردن متدهای abstract در کلاس مشتق شده را توضیح دهیم. بنابراین در این متد، فقط در خروجی چاپ می کنیم که یک لیست باکس با مشخصات معین شده رسم شد.

حال که کنترل اول را ایجاد کردیم، به سراغ کنترل بعدی که یک Button است می رویم. برای ایجاد این کنترل نیز همانند کنترل ListBox، یک کلاس به نام Button ایجاد کرده و کلاس Window را به عنوان کلاس پایه ی آن مشخص می کنیم.

```
// Button control that derives from Window class
public class Button : Window
{
```

بقیه موارد این کلاس نیز مشابه کلاس ListBox است و هیچ ابهامی در آن وجود ندارد. البته دقت کنید در این کلاس نیز همانند کلاس ListBox، اگر متد DrawWindow را override نکنیم، با خطای زمان کامپایل مواجه خواهیم شد.

```
// The new class's constructor
public Button(int top, int left)
{
     Top = top;
     Left = left;
}

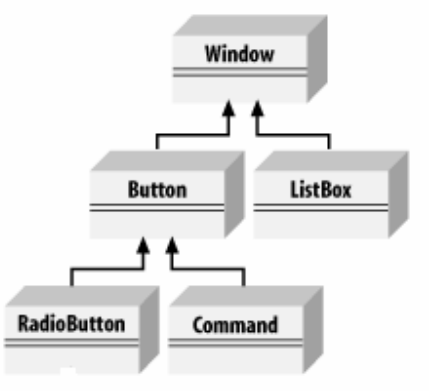
// implement the abstract method
public override void DrawWindow()
{
     Console.WriteLine("Drawing a button at " + Top
                       + ", " + Left);
}
```

حال که کنترلهای مورد نظرمان را ایجاد کردیم، باید آنها را در برنامه تست کنیم. برای این کار، ابتدا دو شیئ از نوع ListBox و یک شیئ نیز از نوع Button ایجاد می کنیم. سپس با فراخوانی متد DrawWindow در آنها، این کنترل ها را در صفحه نمایش می دهیم. همانطور که در شکل ۱۰-۷ نیز مشاهده کردید با فراخوانی متد DrawWindow در این اشیا، آنها در صفحه رسم می شوند.

```
// Create two list boxes and one button
ListBox lstBox1 = new ListBox(1, 2, "First List Box");
ListBox lstBox2 = new ListBox(3, 4,"Second List Box");
Button newButton = new Button(5, 6);

// Draw all objects on the form
lstBox1.DrawWindow();
lstBox2.DrawWindow();
newButton.DrawWindow();
```

نکته: از یک کلاس abstract، می توان یک کلاس abstract دیگر مشتق کرد. برای مثال تصور کنید می خواهید کلاسی به نام Button ایجاد کنید، اما اجازه ندهید کسی از این کلاس شیئ را ایجاد کند. بلکه کلاس هایی مانند Command، RadioButton و ... را از این کلاس مشتق کنید. بنابراین کلاس Button خود باید یک کلاس abstract باشد که از کلاس abstract دیگری به نام window مشتق می شود (شکل ۱۰-۹). در این حالت نیازی نیست که متدهای abstract کلاس پایه را در کلاس مشتق شده override کنید و می توانید از پیاده سازی متد DrawWindow در کلاس Button صرفنظر کنید.



شکل ۱۰-۹

کلاسهای sealed

در قسمت قبل با کلاسهای `abstract` آشنا شدید و مشاهده کردید که چگونه می توان کلاس هایی داشت که فقط به عنوان کلاس پایه استفاده شوند و نتوان شیئ ای از آنها نمونه سازی کرد.
کلاسهای `sealed` دقیقاً مفهومی عکس کلاسهای `abstract` دارند. به عبارت دیگر کلاسهای `sealed` کلاس هایی هستند که نمی توان آنها را به عنوان کلاس پایه مورد استفاده قرار داد و از آنها کلاس جدیدی را مشتق کرد، بلکه فقط می توان از آنها برای نمونه سازی اشیا استفاده کرد. برای تعریف یک کلاس `sealed` باید از کلمه ی کلیدی `sealed` قبل از تعریف کلاس همانند کد زیر استفاده کنید.

```
public sealed class SealedClass
{
      // Implementation here ...
}
```

Interfaceها:

به بیان ساده می توانم بگویم که `interface` یک قرارداد در نوع رفتار یک کلاس است. هر کلاسی که یک `interface` را به کار ببرد، در حقیقت تضمین می کند پیاده سازی تمام متد ها، خاصیت ها و ... که در آن `interface` وجود دارد را در خود ایجاد کند.
فرض کنید در حال نوشتن برنامه ای برای کنترل امور داخلی یک شرکت هستید. در نوشتن چنین برنامه ای مسلماً باید از کلاسهای زیادی استفاده کنید، برای مثال یک کلاس برای کنترل امور مالی شرکت به نام `Financial`، کلاسی برای کنترل امور کارمندان شرکت به نام `Employees`، کلاسی برای کنترل امور چاپ در برنامه به نام `Print` و ... حال ممکن است بخواهید متدی در برنامه ایجاد کنید تا اطلاعات مهمی که در بعضی از کلاسها قرار دارند را در دیسک ذخیره کند. خوب، واضح است که کلاسی مثل کلاس `Print` اطلاعاتی برای ذخیره شدن در دیسک ندارد، اما کلاسی مانند کلاس `Financial` و یا کلاس `Employees` دارای اطلاعات مهمی هستند که باید ذخیره شوند تا در صورت از دست رفتن آنها در برنامه، بتوان اطلاعات را بازیابی کرد. سوالی که در اینجا پیش می آید این است که این متد را باید چگونه نوشت تا بتواند این کار را انجام دهد؟

یک روش این است که در خارج از تعریف کلاسها، بازای هر کلاسی که می خواهیم اطلاعاتش در دیسک ذخیره شود یک نسخه از متد را ایجاد کنیم. برای مثال یک متد با نام Save ایجاد کنیم که پارامتری از نوع Financial دریافت کند سپس اطلاعات مهم شیئ ای که به این متد فرستاده می شود را در دیسک ذخیره کنیم[1]. همچنین متد Save را سربار گذاری کرده تا پارامتری از کلاس Employees دریافت کند و پیاده سازی آن را نیز به گونه ای تغییر دهیم تا اطلاعات مهم اشیای Employees را ذخیره کند و به همین ترتیب این کار را برای تمام کلاس هایی که می خواهیم اطلاعات شان در دیسک ذخیره شود تکرار کنیم.

اما این کار منطقی به نظر نمی رسد، زیرا به این ترتیب تعداد زیادی متد به نام Save که هر کدام دارای پیاده سازی مخصوص به خود هستند در برنامه ایجاد خواهد شد که ممکن است نگهداری و فهم برنامه را مشکل کند.

یک روش بهتر این است که در داخل هر کلاس متدی به نام RetrieveData ایجاد کنیم تا اطلاعات مهم مربوط به آن کلاس را که باید ذخیره شوند در قالب رشته برگرداند، سپس متد Save را در خارج از همه ی کلاسها، به گونه ای بنویسیم تا بتواند هر شیئ ای که دارای متد RetrieveData باشد را دریافت کند و اطلاعات برگشتی از این متد را در دیسک ذخیره کند. سپس هنگام استفاده کافی است تک تک اشیایی که دارای متد RetrieveData هستند را به عنوان پارامتر به متد Save فرستاده تا اطلاعات شان در دیسک ذخیره شود. اما سوال اینجاست که چگونه می توانیم به متد Save بگوییم که "فقط اشیایی را به عنوان پارامتر دریافت کن که دارای متد RetrieveData هستند."؟

در فصل نهم مشاهده کردید که اگر یک متد، به پارامتری از نوع یک کلاس پایه (مانند Car) نیاز داشت، اشیایی که از کلاسهای مشتق شده از آن کلاس نمونه سازی می شدند نیز (همانند اشیای نمونه سازی شده از کلاس SportsCar) می توانستند به عنوان پارامتر به آن فرستاده شوند. این مورد به این علت امکان پذیر بود که کلاس پایه تضمین می کرد کلاس مشتق شده تمام متد ها و خاصیت های مورد نیاز را داشته باشد. اما مشخص است که در متد Save نمی توانیم از این روش استفاده کنیم، زیرا در این برنامه کلاسهای مختلفی وجود دارند که وظایف گوناگونی را انجام می دهند و نمی توان یک کلاس پایه برای همه آنها مشخص کرد تا متد Save را در آن کلاس پایه قرار دهیم. برای مثال کلاس Financial ممکن است از کلاسی به نام FinanBase مشتق شود، اما کلاس Employees ممکن است از کلاس EmpBase مشتق شود و یا حتی اصلاً از هیچ کلاسی مشتق نشود.

در این گونه موارد بهترین راه، استفاده از Interfaceها است. همانطور که گفتم هنگامی که یک کلاس از یک interface استفاده می کند، در حقیقت تضمین می کند که تمام متد ها، خاصیت ها و ... که در آن interface تعریف شده است را در خود پیاده سازی کند. بنابراین می توانید متد RetrieveData (و هر متد و یا خاصیت دیگری که ممکن است برای ذخیره سازی در دیسک مورد نیاز باشد) را در یک Interface به نام IStorable[2] قرار دهید، سپس هر کلاسی که بخواهد قابلیت ذخیره شدن در دیسک را داشته باشد باید از این Interface استفاده کند.

در بخش امتحان کنید زیر سعی می کنیم برنامه ای که در بالا توضیح داده شد را به صورتی ساده انجام دهیم تا با نحوه استفاده از Interfaceها در برنامه بیشتر آشنا شوید. البته برنامه ای که در زیر ایجاد می کنیم کاملاً خلاصه شده است، برای مثال در کلاس Financial فقط از دو فیلد برای نگهداری درآمدها و هزینه ها استفاده می کنیم. در کلاس Employees نیز فرض می کنیم که فقط به ذخیره کردن اطلاعات مختصری از یک کارمند نیاز داریم. اما مسلماً در یک برنامه واقعی این کلاسها بسیار بزرگتر هستند و جزئیات بیشتری را شامل می شوند.

امتحان کنید: استفاده از Interfaceها در برنامه

[1] با فرض این که تمام خاصیت ها و فیلدهایی که باید در دیسک ذخیره شوند به صورت public تعریف شده باشند تا بتوانیم به آنها دسترسی پیدا کنیم.

[2] معمولا نام Interface ها با یک حرف I بزرگ شروع می شود تا به راحتی قابل تشخیص باشند.

۱) با استفاده از گزینه ی ...File ➔ New ➔ Project در نوار منوی ویـژوال اسـتودیو، پـروژه جدیـدی ایجاد کرده و نام آن را برابر با Interface Demo قرار دهید.

۲) با استفاده از پنجره ی Solution Explorer، روی نام پروژه کلیک راست کرده و از منوی باز شده گزینـه ی ...Add ➔ Class را انتخـاب کنیـد تـا پنجـره ی Add New Item – Interface Demo باز شود. سپس در کادر Name نام Financial.cs را برای کلاس انتخـاب کـرده و روی دکمـه ی OK کلیک کنید تا کلاس Financial ایجاد شود.

۳) ابتدا باید Interface مورد نیاز در برنامه را ایجاد کنیم. برای ایجاد یـک Interface از کلمـه ی کلیـدی interface استفاده می کنند. کد زیر را در داخل فضای نام Interface_Demo، قبل از تعریف کلاس Financial قرار دهید

```
namespace Interface_Demo
{
     // Define functions and properties needed for
     // an object to be storable
     public interface IStorable
     {
          // A method for retrieving important data
          string RetrieveData();

          // A property to set the path for saving data
          string SavePath
          {
               get;
               set;
          }
     }

     class Financial
     {
```

۴) حال باید فیلد های مورد نیاز در کلاس Financial را تعریف کنیم. برای این کار کد مشخص شده در زیـر را بـه داخل کلاس Financial اضافه کنید:

```
     class Financial
     {
          // Define some fields in class to store data
          public string _savePath =
                              @"C:\FinancialBackup.dat";
          public long Expenditures;
          public long Earnings;
```

۵) برای تنظیم مقادیر اولیه فیلد های Expenditure و Earnings می توانیم یک متد سازنده به کلاس اضافه کنیم. اما بهتر است به جای اینکه مقدارهای پیش فرضی را به این دو فیلد اختصاص دهیم، مقدار آنها را به صورت پارامترهای متد سازنده از کاربر دریافت کنیم. بنابراین متد سازنده ای همانند زیر در برنامه ایجاد کنید:

```
    // Class Constructor that takes two arguments
    public Financial(long expend, long earn)
    {
        this.Expenditures = expend;
        this.Earnings = earn;
    }
```

۶) حال باید مشخص کنیم که کلاس Financial از اینترفیس IStorable استفاده می کند. برای این کار باید همانند مشخص کردن یک کلاس پایه برای یک کلاس عمل کنیم. بنابراین تعریف کلاس Financial را به صورت زیر تغییر دهید:

```
class Financial : IStorable
{
```

۷) حال که مشخص کردیم این کلاس از اینترفیس IStorable استفاده می کند، باید متد ها و خاصیت هایی که در این Interface مشخص شده است را در کلاس ایجاد کنیم. در اینترفیس IStorable یک متد و یک خاصیت وجود دارد. ابتدا از متد RetrieveData شروع می کنیم. برای این کار کد زیر را به برنامه اضافه کنید. دقت کنید که تعریف این متد دقیقاً باید مشابه تعریفی باشد که در داخل Interface مشخص شده است.

```
    // Implementation of RetrieveData method
    // in IStorable interface
    public string RetrieveData()
    {
        string result;
        result = "Expenditure = " + this.Expenditures;
        result += " Earnings = " + this.Earnings;
        return result;
    }
```

۸) بعد از ایجاد متد مشخص شده، باید خاصیتی که در Interface تعیین شده است را نیز در داخل کلاس ایجاد کنیم. برای این کار کد زیر را نیز به کلاس اضافه کنید:

```
    // Implementation of SavePath property
    // in IStorable interface
    public string SavePath
    {
        get
        {
            return _savePath;
```

```
            }
            set
            {
                _savePath = value;
            }
        }
```

۹) بعد از اتمام کلاس Financial، کلاس Employees را آغاز می کنیم. برای این کار با استفاده از پنجره ی Solution Explorer، روی نام پروژه کلیک راست کنید و از منوی باز شده گزینه ی Add ➔ Class... را انتخاب کنید. سپس با استفاده از پنجره ی Add New Item، کلاس جدیدی به نام Employees به برنامه اضافه کنید.

۱۰) به علت اینکه فقط نحوه استفاده از Interface ها مد نظر ماست، سعی می کنیم کلاسهای برنامه را تا حد ممکن ساده ایجاد کنیم.به همین دلیل برای این کلاس فقط دو فیلد برای نگهداری اطلاعات یک کارمند، یک متد سازنده برای تنظیم مقدار اولیه این دو فیلد و یک فیلد نیز برای نگهداری آدرس فایلی که باید برای ذخیره اطلاعات استفاده شود ایجاد می کنیم. برای این کار کد زیر را به کلاس Employees اضافه کنید:

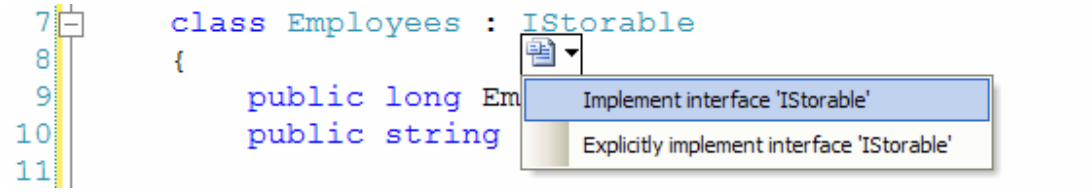
```
    public string _savePath = @"C:\EmployeesBackup.dat";
    public long EmployeeID;
    public string EmployeeName;

    public Employees(int ID, string Name)
    {
        this.EmployeeID = ID;
        this.EmployeeName = Name;
    }
```

۱۱) همانند کلاس Financial، این کلاس نیز نیاز دارد که اطلاعات خود را در دیسک ذخیره کند. پس اینترفیس IStorable را برای این کلاس نیز باید به کار ببریم. برای این کار تعریف کلاس را به صورت زیر تغییر دهید:

```
namespace Interface_Demo
{
    class Employees : IStorable
    {
```

۱۲) اگر برای چند لحظه، اشاره گر ماوس را بر روی نام اینترفیس قرار دهید، کادر کوچکی همانند شکل ۱۰-۱۰ نمایش داده خواهد شد. به وسیله این کادر می توانید از ویژوال استودیو بخواهید که تعریف تمام متد ها و خاصیت های این اینترفیس را به برنامه ی شما اضافه کند. به این ترتیب فقط کافی است که پیاده سازی مورد نظرتان را در متد ها و خاصیت های مربوط قرار دهید. برای این کار در کادری که در شکل ۱۰-۱۰ نیز نمایش داده شده است، روی گزینه ی Implement interface 'IStorable' کلیک کنید.



شکل ۱۰-۱۰

۱۳) مشاهده می کنید که با انتخاب این گزینه، تمام متد ها و خاصیت های موجود در اینترفیس `IStorable` به کلاس اضافه می شوند. در تمام این متد ها و خاصیت ها کدی مشابه زیر قرار دارد:

```
throw new Exception(
    "The method or operation is not implemented.");
```

۱۴) کد موجود در متد `RetrieveData` را به صورت زیر تغییر دهید:

```
public string RetrieveData()
{
    // Define a variable to store results
    string result;

    // Produce the result
    result = "Employee ID: " + this.EmployeeID;
    result += " Employee Name: " + this.EmployeeName;

    return result;
}
```

۱۵) کد درون بخشهای `get` و `set` خاصیت `SavePath` را نیز به صورت زیر تغییر دهید:

```
    public string SavePath
    {
        get
        {
            return _savePath;
        }
        set
        {
            _savePath = value;
        }
    }
```

۱۶) تا اینجا کلاس های مورد نیاز در برنامه را ایجاد کرده ایم، حال باید کدی برای تست کردن برنامه بنویسیم. با استفاده از پنجره ی `Solution Explorer` فایل `Program.cs` را باز کرده و کد زیر را به متد `Main`

اضافه کنید. هنگام اضافه کردن کد با پیغام خطایی مواجه می شوید که متد Save تعریف نشده است. می توانید از ایـن پیغام صرفنظر کنید، زیرا این متد را در مرحله بعد ایجاد خواهیم کرد.

```
static void Main(string[] args)
{
     // Instantiating the needed variables
     Financial finan = new Financial(264399, 368547);
     Employees emp = new Employees(1, "John Smith");

     // Writing information about created objects
     Console.WriteLine("A Financial object created!");
     Console.WriteLine("Expenditure: " +
                       finan.Expenditures);
     Console.WriteLine("Earning: " + finan.Earnings);

     Console.WriteLine("An Employees object created!");
     Console.WriteLine("Employee ID: " + emp.EmployeeID);
     Console.WriteLine("Employee Name: " +
                       emp.EmployeeName);
     Console.WriteLine("Press any key to save data" +
                       " on disk.");

     Console.Read();

     Save(finan);
     Console.WriteLine("Financial information saved at: " +
                       finan.SavePath);
     Save(emp);
     Console.WriteLine("Employees information saved at: " +
                       emp.SavePath);
     Console.Read();
}
```

۱۷) برای نوشتن متد Save به استفاده از کلاس File در فضای نام System.IO نیاز داریم، پس بـا اسـتفاده از راهنمای زیر این فضای نام را به برنامه اضافه کنید:

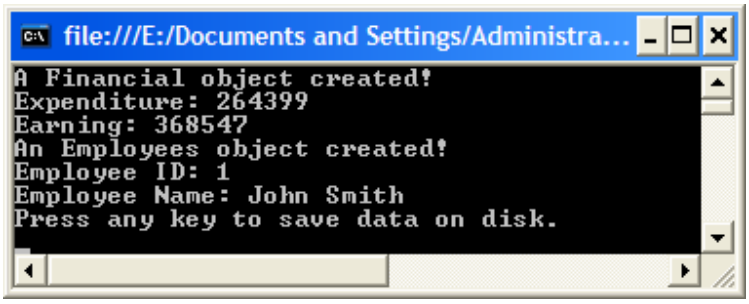
```
using System.IO;
```

۱۸) متد زیر را به برنامه اضافه کنید تا قسمت تست برنامه نیز تکمیل شود:

```
static void Save(IStorable obj)
{
     // Define a variable to store important data
     string data;
```

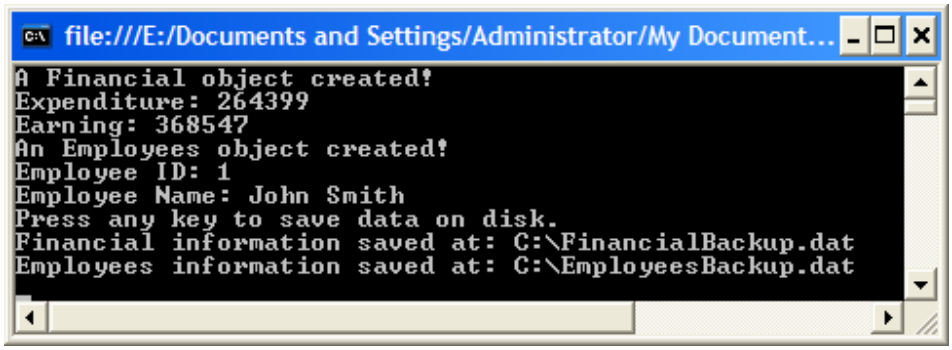
```
        // Retrieving important data to save on disk
        data = obj.RetrieveData();
        // Saving retrieved data on disk
        File.WriteAllText(obj.SavePath, data);
}
```

۱۹) حال برنامه را اجرا کنید. پنجره ای مشابه شکل ۱۰-۱۱ مشاهده خواهید کرد که اطلاعات اشیای ساخته شده را نمایش می دهد.



شکل ۱۰-۱۱

۲۰) کلیدی را در برنامه فشار دهید. به این ترتیب اطلاعات کلاسها در فایلهای مشخص شده ذخیره می شوند. برنامه نیز ذخیره شدن این اطلاعات را همانند شکل ۱۰-۱۲ اعلام می کند.



شکل ۱۰-۱۲

چگونه کار می کند؟

همانطور که گفتم، `interface` یک قرار داد است. در این برنامه ابتدا یک اینترفیس ایجاد می کنیم، به بیان دیگر قراردادی ایجاد می کنیم تا هر کلاسی که بخواهد اطلاعاتش را در دیسک ذخیره کند باید از آن قرار داد پیروی کند. در این قرارداد نیز ذکر می کنیم هر کلاسی که بخواهد اطلاعاتش را در دیسک ذخیره کند باید تابعی به نام `RetrieveData` داشته باشد که این تابع اطلاعات مهم کلاس را از نوع رشته برگرداند. همچنین کلاس مذکور باید دارای خاصیتی از نوع خواندنی-نوشتنی به نام `SavePath` باشد که مسیر ذخیره اطلاعات را برگرداند.

```
// Define functions and properties needed for
// an object to be storable
public interface IStorable
{
     // A method for retrieving important data
     string RetrieveData();

     // A property to set the path for saving data
     string SavePath
     {
          get;
          set;
     }
}
```

در طراحی یک اینترفیس دقت کنید که همانند کلاسهای Abstract، نباید هیچ گونه پیاده سازی ای برای متد ها و یا خاصیت ها تعیین کنید. بلکه فقط کافی است نام، نوع داده برگشتی و همچنین پارامترهای آنها را مشخص کنید. همچنین نباید هیچ سطح دسترسی از قبیل public و یا private برای یک اینترفیس مشخص کنید، بلکه سطح دسترسی تمام متد ها و خاصیتهای آن برابر با سطح دسترسی خود interface می باشد. برای مثال در اینترفیسی که در برنامه ی قبلی ایجاد کردیم مشاهده می کنید که هیچ سطح دسترسی ای برای متد ها و خاصیت ها تعیین نشده است، اما سطح دسترسی اینترفیس برابر با public مشخص شده است. بنابراین سطح دسترسی کلیه اعضا برابر با public خواهد بود.

حال هر کلاسی که از اینترفیس IStorable استفاده کند بر طبق این قرارداد موظف خواهد بود که متدی به نام RetrieveData از نوع public داشته باشد که هیچ پارامتری دریافت نکند و همچنین یک string را به عنوان نتیجه برگرداند. همچنین این کلاس باید یک خاصیت خواندنی-نوشتنی به نام SavePath از نوع public داشته باشد که یک رشته را برگرداند.

تعریف کلاس Financial کاملاً واضح است. همانطور که گفتم برای اینکه در این برنامه فقط بتوانیم بر نحوه کار اینترفیس ها تمرکز کنیم، کلاسها را تا حد ممکن ساده تعریف می کنیم. این کلاس شامل دو فیلد برای نگهداری هزینه ها و مخارج است. همچنین به فیلد دیگری نیز در این کلاس نیاز داریم تا مکان ذخیره شدن اطلاعات کلاس در دیسک را در آن نگهداری کنیم.

```
// Define some fields in class to store data
public string _savePath = @"C:\FinancialBackup.dat";
public long Expenditures;
public long Earnings;
```

همچنین یک متد سازنده برای این کلاس ایجاد می کنیم تا مقدار اولیه فیلد های شیئ را برابر با مقادیر مورد نظر کاربر قرار دهد.

```
// Class Constructor that takes two arguments
public Financial(long expend, long earn)
{
    this.Expenditures = expend;
    this.Earnings = earn;
}
```

برای اینکه مشخص کنیم یک کلاس از قراردادهای یک اینترفیس خاص پیروی می کند باید نام اینترفیس را بعد از کاراکتر : در مقابل نام کلاس ذکر کنیم، دقیقاً مشابه مشخص کردن یک کلاس پایه برای کلاس. البته دقت کنید که در وراثت هر کلاس فقط می توانست شامل یک کلاس پایه باشد، اما هر کلاس می توانند چندین اینترفیس را در خود به کار ببرد. به عبارت دیگر بر خلاف وراثت، یک کلاس می توانند از قواعد چندین اینترفیس پیروی کند و متدهای تعیین شده در آنها را در خود ایجاد کند. برای اینکه مشخص کنیم کلاس Financial از اینترفیس IStorable استفاده می کند، تعریف کلاس را به صورت زیر تغییر می دهیم:

```
class Financial : IStorable
{
```

حال باید یک متد به نام RetrieveData در کلاس ایجاد کنیم که اطلاعات مهم کلاس را در قالب رشته برگرداند. این متد باید از نوع Public باشد و همچنین نباید پارامتری را دریافت کند، در غیر این صورت هنگام کامپایل با خطا مواجه خواهید شد. بنابراین متدی را به صورت زیر به برنامه اضافه می کنیم:

```
// Implementation of RetrieveData method
// in IStorable interface
public string RetrieveData()
{
     string result;
     result = "Expenditure = " + this.Expenditures;
     result += " Earnings = " + this.Earnings;
     return result;
}
```

در این متد نیز مقادیر موجود در فیلد های کلاس را در داخل یک متغیر رشته ای قرار داده و آن را برمی گردانیم. حال باید خاصیت ای که در اینترفیس مشخص شده است را نیز در کلاس ایجاد کنیم. این خاصیت در اینترفیس دارای بخشهای get و set است، پس در کلاس نیز باید دارای این بخشها باشد، به عبارت دیگر باید خواندنی–نوشتنی باشد.

```
// Implementation of SavePath property
// in IStorable interface
public string SavePath
{
     get
     {
          return _savePath;
     }
     set
     {
          _savePath = value;
     }
}
```

با اضافه کردن این خاصیت به کلاس، تمام متد ها و خاصیت های مورد نیاز را به برنامه اضافه کرده ایم. حال به طراحی کلاس بعدی یعنی کلاس Employees می پردازیم. در این کلاس نیز همانند کلاس Financial دو فیلد برای نگهداری اطلاعات کارمند، یک فیلد برای نگهداری آدرسی که باید اطلاعات کلاس در آن ذخیره شود و یک متد سازنده برای مشخص کردن اولیه فیلد های کلاس ایجاد می کنیم:

```
public string _savePath = @"C:\EmployeesBackup.dat";
public long EmployeeID;
public string EmployeeName;

public Employees(int ID, string Name)
{
    this.EmployeeID = ID;
    this.EmployeeName = Name;
}
```

اکنون باید مشخص کنیم که این کلاس نیز از قواعد موجود در اینترفیس IStorable پیروی می کند. برای این کار همانند کلاس Financial، نام اینترفیس را در مقابل نام کلاس ذکر می کنیم. به این ترتیب باید متد RetrieveData و همچنین خاصیت SavePath را نیز در کلاس ایجاد کنیم.

```
public string RetrieveData()
{
     // Define a variable to store results
     string result;

     // Produce the result
     result = "Employee ID: " + this.EmployeeID;
     result += " Employee Name: " + this.EmployeeName;

     return result;
}

public string SavePath
{
     get
     {
          return _savePath;
     }
     set
     {
          _savePath = value;
     }
}
```

بعد از تمام شدن طراحی کلاسهای مورد نیاز در برنامه، باید قسمتی را برای بررسی نحوه عملکرد آن بنویسیم. در متد Main ابتدا دو شیئ از نوع Financial و Employees ایجاد کرده و اطلاعات آنها را با استفاده از متد WriteLine به کاربر نمایش می دهیم:

```
// Instantiating the needed variables
Financial finan = new Financial(264399, 368547);
Employees emp = new Employees(1, "John Smith");

// Writing information about created objects
Console.WriteLine("A Financial object created!");
Console.WriteLine("Expenditure: " +
                  finan.Expenditures);
Console.WriteLine("Earning: " + finan.Earnings);
Console.WriteLine("An Employees object created!");
Console.WriteLine("Employee ID: " + emp.EmployeeID);
Console.WriteLine("Employee Name: " +
                  emp.EmployeeName);
```

حال به قسمتی می رسیم که راحتی کار با اینترفیس ها را می توانید به وضوح درک کنید. همانطور که گفتم هنگام ایجاد متد Save، برای ما تفاوتی ندارد که چه شیئ ای به این متد فرستاده می شود بلکه فقط می خواهیم شیئ ای که فرستاده می شود دارای متدی به نام RetrieveData باشد تا اطلاعات مهم آن ر در قالب رشته برگرداند. همچنین می خواهیم این شیئ دارای خاصیتی به نام SavePath باشد که آدرس ذخیره شدن فایل را برگرداند. بنابراین به راحتی می توانیم به متد Save بگوییم پارامتری که دریافت می کند حتما باید از قواعد اینترفیس IStorable پیروی کرده باشد.

```
static void Save(IStorable obj)
```

نکته ای که باید به آن توجه کنید این است که در بدنه ی متد Save فقط به خاصیت ها و متد هایی از شیئ obj دسترسی داریم که در اینترفیس IStorable تعریف شده اند. برای مثال اگر شیئ ای از نوع Employees را به متد Save بفرستیم، درون این متد فقط به خاصیت SavePath و متد RetrieveData از آن شیئ دسترسی خواهیم داشت و نمی توانیم به مقدار فیلد EmployeeID دسترسی داشته باشیم (با وجود اینکه این فیلد از نوع public است). دلیل این امر هم کاملاً واضح است. در هنگام تعریف متد مشخص کرده ایم که اشیایی که به متد فرستاده می شوند فقط باید دارای متد ها و خاصیتهای تعریف شده در اینترفیس IStorable باشند، بنابراین نمی توانیم انتظار بیشتری از پارامترهای فرستاده شده به این متد داشته باشیم. در بدنه ی متد هم کافی است با استفاده از متد RetrieveData اطلاعات مهم پارامتر فرستاده شده را بدست آوریم، سپس با استفاده از متد WriteAllText در کلاس File، اطلاعات را در آدرسی که درون کلاس مشخص شده است قرار می دهیم:

```
// Define a variable to store important data
string data;

// Retrieving important data to save on disk
data = obj.RetrieveData();
// Saving retrieved data on disk
```

```
File.WriteAllText(obj.SavePath, data);
```

نکته: دقت کنید که در هنگام تعریف یک اینترفیس، نمی توانید فیلدی در آن ایجاد کنید. به عبارت دیگر نمی توانید کلاسهای استفاده کننده از یک اینترفیس را ملزم به تعریف یک فیلد خاص کنید. برای مثال استفاده از اینترفیس زیر در برنامه موجب بروز خطا خواهد شد:

```
public interface IInvalid
{
    int MakesAnError;
    // Other members of this interface goes here
}
```

حال که با بیشتر تکنیکهای برنامه نویسی شیئ گرا آشنا شدید، در ادامه سعی می کنیم با استفاده از این تکنیکها برنامه ای کاربردی ایجاد کرده تا با نحوه استفاده از آنها در عمل بیشتر آشنا شویم.

ایجاد یک برنامه ی کاربردی:

در این قسمت از فصل برنامه ی خواهیم نوشت که به وسیله آن بتوانید سایتهای موجود در بخش Favorites اینترنت اکسپلورر را مشاهده کنید. همچنین در این برنامه دکمه ای قرار می دهیم که به وسیله آن بتوانید آن سایتها را با استفاده از اینترنت اکسپلورر مشاهده کنید.

شورت کات های اینترنتی و Favoritesها:

احتمالاً تاکنون با قسمت Favorites در اینترنت اکسپلورر برخورد کرده اید و عملکرد آن را نیز می دانید. اما نکته ای که ممکن است در این مورد ندانید، این است که اینترنت اکسپلورر چگونه اطلاعات موجود در قسمت Favorites را ذخیره می کند؟ در حقیقت، گزینه های موجود در قسمت Favorites فقط مخصوص اینترنت اکسپلورر نیستند و هر برنامه ای می تواند به آنها دسترسی داشته باشد، فقط کافی است مکان ذخیره شده این فایلها را بداند.
همانطور که می دانید در ویندوز XP، برنامه ها می توانند اطلاعات مربوط به هر کاربر را در فولدر مخصوص به آن کاربر که در مسیر C:\Documents and Settings قرار دارد، بنویسند. در این مسیر بازای هر کاربر یک فولدر به نام او وجود دارد. برای مثال کامپیوتری که در شکل ۱۰-۱۳ نشان داده شده است یک کاربر به نام Mohammad وجود دارد.

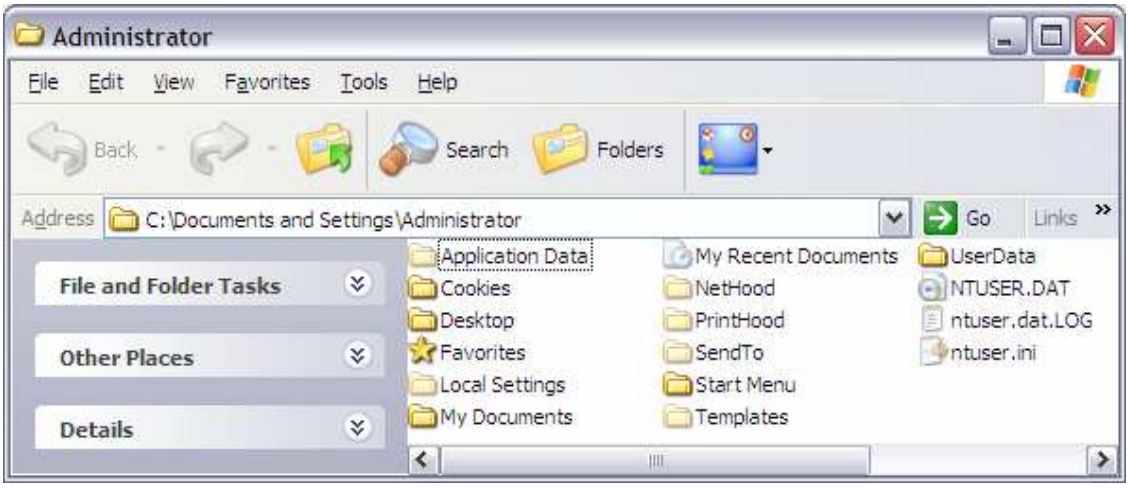
نکته: در این مسیر علاوه بر فولدر هایی مخصوص کاربران کامپیوتر، فولدر های دیگری نیز وجود دارد. فولدر Administrator برای مدیر پیش فرض سیستم است و در سیستم عامل های ویندوز ۲۰۰۰ و یا ویندوز XP Professional وجود دارد. فولدر All Users حاوی اطلاعاتی است که بین تمام کاربران مشترک است. فولدر Default User نیز یک فولدر مخصوص است که هنگامی که یک کاربر برای اولین بار از سیستم استفاده می کند، ویندوز تنظیمات مشخص شده در این فولدر را برای او به کار می برد.



شکل ۱۰-۱۳

بر اساس تنظیمات امنیتی کامپیوتری که از آن استفاده می کنید، ممکن است به محتویات این فولدر دسترسی نداشته باشید و یا اینکه فقط بتوانید محتویات فولدر مخصوص به کاربری که در حال استفاده از آن هستید را مشاهده کنید. بعد از اینکه وارد یکی از این فولدر ها شدید، یک دسته دیگر از فولدر ها را همانند شکل ۱۰-۱۴ مشاهده خواهید کرد (البته ممکن است محتویات آن فولدر بر اساس تنظیمات کامپیوتر شما با محتویات نمایش داده شده در شکل متفاوت باشد).
توجه کنید که در شکل ۱۰-۱۴ بعضی از فولدر ها کمرنگ نمایش داده شده اند زیرا این فولدر ها به صورت پیش فرض مخفی هستند. به همین دلیل بر حسب تنظیمات کامپیوتری که از آن استفاده می کنید، ممکن است این فولدرهای کمرنگ نمایش داده نشوند. البته در این قسمت از این فولدر ها استفاده نخواهیم کرد و فقط با فولدر `Favorites` کار می کنیم که در هر حال نمایش داده می شود.
همانطور که گفتم این فولدر و فولدرهای درون آن مکانی هستند که ویندوز، اطلاعات شخصی هر کاربر را در آنها ذخیره می کند. برای مثال:

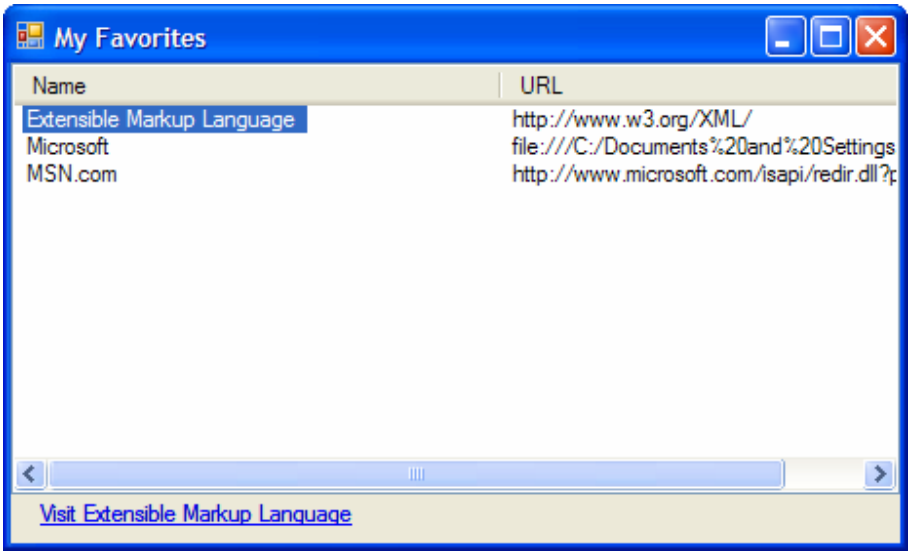
- **`Cookies`:** اطلاعات مربوط به صفحات وبی که مشاهده کرده اید را نگه داری می کند.
- **`Desktop`:** فایلها و فولدر هایی که در `Desktop` نمایش داده می شوند در این قسمت ذخیره می شوند.
- **`Favorites`:** آیتم های موجود در قسمت `Favorites` اینترنت اکسپلورر در این فولدر ذخیره می شوند.
- **`My Documents`:** مکانی برای ذخیره اطلاعات، عکسها و سندهای ایجاد شده به وسیله کاربر است.
- **`Start Menu`:** لیستی از آیکون ها و فولدر هایی که زمان استفاده از منوی `Start` نمایش داده می شوند را نگهداری می کند.
- **`User Data`:** اطلاعات مربوط به برنامه هایی که کاربر از آنها استفاده می کند را نگهداری می کند.



شکل ۱۰-۱۴

در این قسمت فقط از فولدر `Favorites` استفاده خواهیم کرد، بنابراین این فولدر را باز کنید. در این فولدر لیستی شامل تعدادی لینک به آدرس های اینترنتی مشاهده خواهید کرد. مشاهده می کنید که لینک های این قسمت با لینک های موجود در قسمت `Favorites` اینترنت اکسپلورر برابر هستند. اگر روی هر کدام از این لینک ها دو بار کلیک کنید، اینترنت اکسپلورر باز شده و به سایتی که لینک به آن اشاره می کند می رود.

حال که متوجه شدید گزینه های قسمت Favorites در اینترنت اکسپلورر در چه فولدری نگهداری می شوند، می توانید برنامه ای بنویسید که این فولدر را باز کرده و از لینک های درون آن استفاده کند، برای مثال آنها را به یک لیست اضافه کند، سایت مربوط به آنها را با استفاده از اینترنت اکسپلورر نمایش دهد و .... در این مثال از فولدر هایی که در قسمت Favorites وجود دارند صرف نظر می کنیم و فقط با فایلهای این قسمت کار خواهیم کرد.
این برنامه بعد از اتمام همانند شکل ۱۰-۱۵ خواهد بود.



شکل ۱۰-۱۵

استفاده از کلاسها:

تاکنون در مثال های قبلی این کتاب برنامه های ساده ای ایجاد می کردید که بیشتر وظایف شان را در فرم برنامه انجام می دادند. در این قسمت می خواهیم کلاسی ایجاد کنیم که لیستی از گزینه های موجود در Favorites را نمایش دهد. به این ترتیب می توانید از لیستی که این کلاس ایجاد می کند در هر برنامه و به هر نحوی که بخواهید استفاده کنید. برای مثال می توانید گزینه های موجود در این لیست را با استفاده از یک لیست باکس نمایش دهید و سپس به کاربر اجازه دهید که با استفاده از اینترنت اکسپلورر آنها را مشاهده کند.
بهترین راه برای ایجاد چنین برنامه ای این است که کلاس هایی را به صورت زیر ایجاد کنیم:

- **WebFavorite**: هر شیئ از این کلاس برای نگهداری یکی از گزینه های موجود در Favorites و همچنین ویژگی های آن مانند Name و URL به کار می رود.
- **Favorites**: این کلاس می تواند کامپیوتر کاربر را برای پیدا کردن Favorites جستجو کند، برای هر یک از گزینه های Favorite یک شیئ از کلاس WebFavorite ایجاد کرده و اشیای ایجاد شده را در یک آرایه قرار دهد.

این دو کلاس اصطلاحا قسمت ***back-end*** برنامه را تشکیل می دهند. به عبارت دیگر می توانیم بگوییم تمام کلاس هایی که وظیفه ی خاصی را در برنامه انجام می دهند اما هیچ رابط گرافیکی خاصی ندارند تا در برنامه به کاربر نمایش داده شود، قسمت back-end یک برنامه را تشکیل می دهند. جدا کردن این قسمتها از ظاهر برنامه باعث می شود تا بتوانید از این کلاسها در برنامه های دیگر خود نیز به راحتی استفاده کنید (استفاده مجدد از کد). همچنین برای تکمیل این برنامه به یک قسمت ***front-end*** نیز نیاز دارید که رابط کاربری برنامه را تشکیل می دهد. در این برنامه، این قسمت شامل یک فرم ویندوزی و چند کنترل عادی خواهد بود.
در چند بخش بعدی، به طراحی کلاسهای مورد نیاز و همچنین رابط گرافیکی برنامه ای که در شکل ۱۰-۱۶ مشاهده کردید خواهیم پرداخت.

امتحان کنید: ایجاد برنامه ی Favorites Viewer

۱) محیط ویژوال استودیو ۲۰۰۵ را باز کنید و یک پروژه ویندوزی جدید به نام Favorites Viewer ایجاد کنید.
۲) خاصیتهای فرم را بر طبق لیست زیر تنظیم کنید:

- خاصیت Size را برابر با 464;280 قرار دهید.
- خاصیت StartPosition را برابر با CenterScreen قرار دهید.
- خاصیت Text را برابر با My Favorites قرار دهید.

۳) یک کنترل ListView به فرم اضافه کنید و اندازه آن را به صورتی تغییر دهید که مشابه شکل ۱۰-۱۵ شود. سپس خاصیت های آن را به صورت زیر تغییر دهید:

- خاصیت Name آن را برابر با lstFavorites قرار دهید.
- خاصیت Anchor آن را برابر با Left,Right,Top,Bottom قرار دهید.
- خاصیت View را برابر با Details قرار دهید.

۴) خاصیت Columns کنترل lstFavorites را با استفاده از پنجره ی Properties انتخاب کرده و روی دکمه ی ... در مقابل آن کلیک کنید. به این ترتیب کادر ColumnHeader Collection Editor نمایش داده خواهد شد.
۵) در این کادر روی دکمه ی Add کلیک کنید. سپس خاصیت های ستون جدید را که در بخش Members اضافه شده است، برابر با مقادیر زیر قرار دهید:

- خاصیت Name را برابر با hdrName قرار دهید.
- خاصیت Text آن را برابر با Name قرار دهید.
- خاصیت Width آن را برابر با 250 قرار دهید.

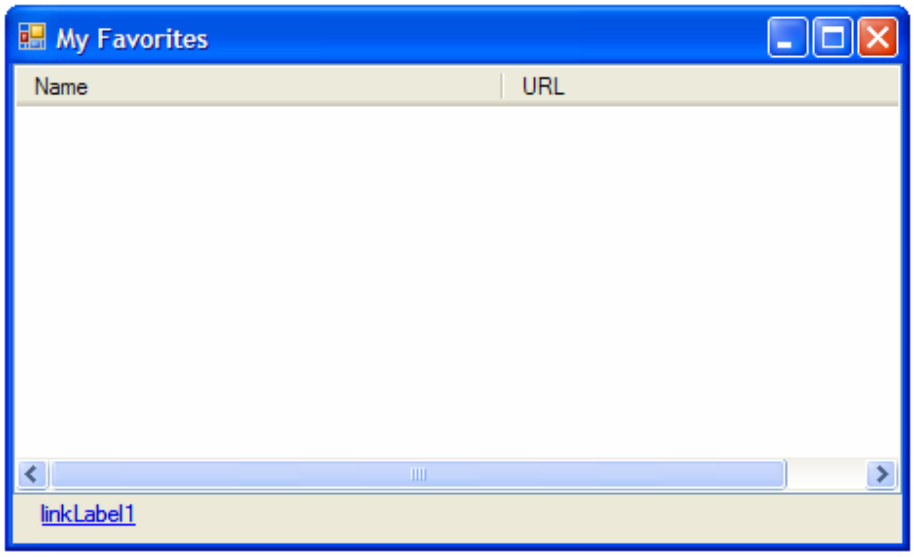
۶) مجدداً روی دکمه ی Add کلیک کنید تا ستون جدیدی به قسمت Members اضافه شود. سپس خاصیت های این ستون را برابر با مقادیر زیر قرار دهید:

- خاصیت `Name` آن را برابر با `hdrUrl` قرار دهید.
- خاصیت `Text` آن را برابر با `Url` قرار دهید.
- خاصیت `Width` آن را برابر با `250` قرار دهید.

۷) روی دکمه ی `OK` کلیک کنید تا کادر `ColumnHeader Collection Editor` بسته شود.
۸) یک کنترل `LinkLabel` به فرم اضافه کرده و خاصیت های آن را برابر با مقادیر زیر قرار دهید.

- خاصیت `Name` آن را برابر با `lnkUrl` قرار دهید.
- خاصیت `Anchor` آن را برابر با `Bottom, Left, Right` قرار دهید.
- خاصیت `TextAlign` را برابر با `MiddleLeft` قرار دهید.

۹) فرم کامل شده ی برنامه باید مشابه شکل ۱۰-۱۶ شده باشد.



شکل ۱۰-۱۶

چگونه کار می کند؟

تمام کاری که در این قسمت انجام دادید، طراحی ظاهر برنامه بود تا بتواند نتیجه ی بدست آمده را نمایش دهد. در این فرم کنترل `ListView` برای نمایش نام و آدرس هر یک از گزینه های موجود در فولدر `Favorites` به کار می رود. کنترل `LinkLabel` هم برای باز کردن برنامه ی مرورگر و نمایش سایت انتخاب شده در لیست استفاده می شود.
تاکنون مراحل اولیه کار را انجام داده ایم. در بخش امتحان کنید بعد، مشاهده خواهید کرد که چگونه می توان کلاسهای بخش `back-end` برنامه را ایجاد کرد.

امتحان کنید: ایجاد کلاس WebFavorite

۱) با استفاده از پنجره ی Solution Explorer روی نام پروژه (Favorites Viewer) کلیک راست کنید و سپس از منوی باز شده گزینه ی Add ➔ Class... را انتخاب کنید تا کادر Add New Item – Favorites Viewer نمایش داده شود. در قسمت Name عبارت WebFavorite.cs را وارد کرده و روی دکمه ی Add کلیک کنید.

۲) با استفاده از دستور زیر فضای نام System.IO را به برنامه اضافه کنید:

```
using System.IO;

namespace Favorites_Viewer
```

۳) حال طراحی خود کلاس را شروع می کنیم. برای این کار ابتدا دو فیلد که در زیر آمده است را به کلاس اضافه کنید:

```
    // Public Members
    public string Name;
    public string Url;
```

۴) همچنین یک متد به نام Load همانند زیر به کلاس اضافه کنید تا مقادیر مورد نیاز را در فیلد های کلاس قرار دهد:

```
public void Load(string FileName)
{
    // Declare variables
    string strData;
    string[] strLines;
    FileInfo objFileInfo = new FileInfo(FileName);

    // Set the Name member to the file name
    // minus the extension
    Name = objFileInfo.Name.Substring(0,
              objFileInfo.Name.Length -
              objFileInfo.Extension.Length);

    // Read the entire contents of the file
    strData = File.ReadAllText(FileName);

    // Split the lines of data in the file
    strLines = strData.Split('\n');

    // Process each line looking for the URL
    foreach(string strLine in strLines)
    {
        // Does the line of data start with URL=
```

```
            if (strLine.StartsWith("URL="))
            {
                // Yes, Set the Url member to the actual URL
                Url = strLine.Substring(4);
                // Exit the foreach loop
                break;
            }
        }
}
```

چگونه کار می کند؟

تنها نکته ی مهمی که در این کلاس قرار دارد این است که این کلاس چگونه هنگام فراخوانی متد `Load` اطلاعات داخل خود را پر می کند؟

در ابتدای این متد باید متغییر های مورد نیاز در متد را تعریف کنیم. یک متغیر به نام `strData` ایجاد کرده و تمام محتویات فایلی که در حال بررسی است را در آن قرار می دهیم (هر فایلی که مشخص کننده یکی از گزینه های منوی `Favorites` در اینترنت اکسپلورر است، در حقیقت یک فایل متنی است که محتویات آن با ویرایشگری مانند `notepad` نیز قابل مشاهده است. پس می توانیم در برنامه محتویات آن فایل را خوانده و در یک متغیر رشته ای قرار دهیم). همچنین یک آرایه رشته ای به نام `strLines` تعریف کرده و هر خط از اطلاعات متغیر `strData` را در یک عنصر از آن قرار می دهیم. در آخر هم یک شیئ از کلاس `FileInfo` ایجاد می کنیم تا به اطلاعات کامل فایلی که نام و آدرس آن به متد فرستاده می شود دسترسی داشته باشیم.

```
public void Load(string FileName)
{
      // Declare variables
      string strData;
      string[] strLines;
      FileInfo objFileInfo = new FileInfo(FileName);
```

همانطور که در فولدر `Favorites` مشاهده کردید، نام هر فایل برابر با نام گزینه ای است که در لیست `Favorites` اینترنت اکسپلورر نمایش داده می شود مانند `Radio Station Guide`. بنابراین می توانیم از نام فایل برای تنظیم خاصیت `Name` در کلاس استفاده کنیم. البته پارامتر `FileName` که به کلاس فرستاده می شود حاوی آدرس کامل فایل است که شامل آدرس فایل، نام فایل و پسوند فایل می شود (برای مثال `C:\Documents And Settings\Administrator\Favorites\Extensible Markup Language.url`). کاری که باید انجام دهیم این است که نام فایل را از این رشته جدا کرده و در فیلد `Name` قرار دهیم.

برای این کار می توانیم از کلاس `FileInfo` استفاده کنیم. ابتدا شیئ ای از این کلاس به نام `objFileInfo` ایجاد می کنیم و هنگام نمونه سازی آن، آدرس فایل مورد نظر را که در `FileName` قرار دارد به عنوان پارامتر به تابع سازنده کلاس `FileInfo` می فرستیم. حال به وسیله این شیئ می توانیم به اطلاعات مختلف فایل از قبیل نام و یا آدرس آن دسترسی داشته باشیم.

یکی از خاصیت های شیئ objFileInfo، خاصیت Name است که به وسیله آن می توانیم به نام و پسوند فایل دسترسی پیدا کنیم (برای مثال این خاصیت برای فایل بالا مقدار "Extensible Markup Language.url" را برمی گرداند). در این قسمت نیز از این خاصیت استفاده کرده و نام و پسوند فایل را بدست می آوریم، سپس با استفاده از متد SubString نام فایل را از پسوند آن جدا می کنیم.
برای این کار باید دو پارامتر را برای متد Substring مشخص کنیم. پارامتر اول از این متد اندیس شروع کاراکتر ها است. در اینجا نام فایل از کاراکتر اول شروع می شود، پس اندیس شروع را صفر در نظر می گیریم[1]. پارامتر دوم تعداد کاراکتر هایی است که باید از رشته جدا شوند. برای اینکه فقط نام فایل را بدست آوریم باید این پارامتر را به صورت "طول کل رشته منهای طول پسوند آن" وارد کنیم.
برای مثال در فایل بالا طول کل رشته ی "Extensible Markup Language.url" برابر با ۳۰ کاراکتر است و طول پسوند آن ۴ کاراکتر، پس متد Substring از کاراکتر اول یعنی حرف R شروع می کند و تا ۴-۳۰ یعنی ۲۶ کاراکتر را برمی گرداند. به این ترتیب عبارت .url از انتهای رشته حذف می شود.

```
// Set the Name member to the file name
// minus the extension
Name = objFileInfo.Name.Substring(0,
          objFileInfo.Name.Length-
          objFileInfo.Extension.Length);
```

حال باید محتویات فایل را خوانده و در متغییر strData قرار دهیم. برای این کار می توانیم از متد ReadAllText در کلاس System.File استفاده کنیم. این متد فایل مورد نظر را باز می کند، تمام محتویات داخل آن را درون متغیر مشخص شده قرار می دهد، فایل را می بندد و منابع گرفته شده به وسیله آن را نیز آزاد می کند.

```
// Read the entire contents of the file
strData = File.ReadAllText(FileName);
```

بعد از این که کد بالا توسط برنامه اجرا شد، متغیر strData دارای متنی مشابه متن زیر خواهد بود. این متن مربوط به محتویات فایل Extensible Markup Language.url است.

```
[DEFAULT]
BASEURL=http://www.w3.org/XML/
[InternetShortcut]
URL=http://www.w3.org/XML/
Modified=B0C9EC877EB3C401E2
```

اما کار با این اطلاعات به صورتی که در این متغییر قرار گرفته است کمی مشکل است. برای راحتی کار بهتر است آرایه ای رشته ای ایجاد کرده و هر خط از این اطلاعات را درون یکی از عناصر آن قرار دهیم. برای این کار می توانیم از متد Split در کلاس

[1] اگر یک رشته را به صورت آرایه ای از کاراکتر های متوالی در نظر بگیریم، همانطور که در آرایه ها اندیس عنصر اول برابر با صفر است، در رشته ها نیز اندیس کاراکتر اول برابر با صفر است.

String استفاده کنیم[1]. از متد Split چندین نسخه سربار گذاری شده است. نسخه ای که در این قسمت از آن استفاده می کنیم کاراکتر هایی که باید به عنوان جدا کننده در نظر گرفته شوند را به عنوان پارامتر دریافت می کند.
همانطور که در بخشهای قبلی نیز گفتم کاراکتر کنترلی '\n' در C# برای مشخص کردن خط جدید به کار می رود. در اینجا نیز برای این که مشخص کنیم متن داخل strData (یا محتویات فایلی که خوانده شده) باید بر اساس خط جدید جدا شوند این کاراکتر را به عنوان جدا کننده به متد Split می فرستیم.

```
// Split the lines of data in the file
strLines = strData.Split('\n');
```

بعد از اجرای این خط از برنامه، هر یک از خطوط فایل مورد بررسی، در یکی از عناصر آرایه ی strLines قرار می گیرد. سپس باید با استفاده از یک حلقه ی foreach عناصر آرایه ی strLines را بررسی کنیم تا به خطی برسیم که با عبارت "URL=" شروع شود. برای تشخیص اینکه یک متغیر رشته ای با عبارت خاصی شروع می شود یا نه می توانیم از متد StartsWith در کلاس String استفاده کنیم. این متد یک عبارت را دریافت می کند و یک مقدار Boolean را برمی گرداند. اگر این مقدار برابر با true بود به این معنی است که رشته ی مورد نظر با عبارت مشخص شده شروع می شود. پس می توانیم نتیجه ی برگشتی از این متد را با استفاده از یک دستور if بررسی کنیم.

```
// Process each line looking for the URL
foreach(string strLine in strLines)
{
      // Does the line of data start with URL=
      if (strLine.StartsWith("URL="))
      {
```

اگر خطی که در حال بررسی آن هستیم با عبارت "URL=" شروع شود، یعنی آدرس مورد نظر ما که باید در فیلد Url قرار گیرد در این خط قرار دارد. پس با استفاده از متد Substring، آن را بدست آورده و در فیلد url قرار می دهیم. به علت اینکه چهار کاراکتر اولی خط (کاراکتر های ۰ تا ۳) محتوی عبارت "URL=" هستند، پس به تابع Substring می گوییم که از کاراکتر پنجم تا انتهای رشته را برگرداند و در فیلد Url قرار دهد.

```
      // Yes, Set the Url member to the actual URL
      Url = strLine.Substring(4);
      // Exit the foreach loop
      break;
```

بررسی بقیه عناصر آرایه نیز کار بیهوده ای است، زیرا خط مورد نظرمان را پیدا کرده ایم. پس با استفاده از دستور break از حلقه ی foreach خارج می شویم.

---

[1] اگر به کد دقت کنید مشاهده می کنید که برای دسترسی به متد Split از متغییر strData استفاده کرده ایم. دلیلی این امر نیز این است که strData را می توان به صورت شیئ ای در نظر گرفت که از کلاس String نمونه سازی شده است.

پیدا کردن گزینه های Favorites:

برای تکمیل شدن قسمت back-end برنامه، به کلاس دیگری نیاز دارید که بتواند آرایه ای از اشیای WebFavorite را در خود نگهداری کند. این کلاس باید بتواند فولدر Favorites را بررسی کرده و بازای هر فایلی که پیدا می کند یک شیئ جدید از نوع WebFavorite متناسب با اطلاعات فایل ایجاد کرده و به آرایه اضافه کند.
در بخش امتحان کنید زیر چنین کلاسی را ایجاد خواهیم کرد.

امتحان کنید: ایجاد کلاس Favorites

۱) با استفاده از پنجره ی Solution Explorer، کلاس جدیدی به نام Favorites ایجاد کنید.
۲) حال باید فیلدی از نوع ArrayList در برنامه ایجاد کنیم تا اشیایی که از نوع WebFavorite ایجاد می شوند را در آن قرار دهیم. برای این کار کد زیر را به کلاس اضافه کنید:

```
class Favorites
{
     // Public member
     public  System.Collections.ArrayList  FavoriteCollection
                                   = new ArrayList();
```

۳) در این کلاس به خاصیتی نیاز داریم تا آدرس فولدر Favorites را در کامپیوتر کاربر برگرداند بنابراین این خاصیت باید از نوع فقط–خواندنی باشد. برای این کار کد مشخص شده در زیر را به کلاس اضافه کنید:

```
public string FavoritesFolder
{
     get
     {
          // Return the path to the user's Favorites folder
          return Environment.GetFolderPath(
                    Environment.SpecialFolder.Favorites);
     }
}
```

۴) در آخر نیز باید متدی به کلاس اضافه کنید که فایلهای درون فولدر Favorites را بررسی کند. هنگامی که فایلی را پیدا کرد، یک شیئ WebFavorite برای آن ایجاد کرده و به ArrayList اضافه کند. در این کلاس دو نسخه از این متد را ایجاد می کنیم؛ یکی از آنها برای بدست آوردن آدرس فولدر Favorites، از خاصیت FavoritesFolder در کلاس استفاده می کند، دیگری این آدرس را به عنوان پارامتر دریافت می کند. برای اضافه کردن دو متد به کلاس، کد زیر را در کلاس قرار دهید:

```
public void ScanFavorites()
{
```

```
     // Scan the favorites folder
     ScanFavorites(this.FavoritesFolder);
}

public void ScanFavorites(string FolderName)
{
     // Process each file in the Favorites folder
     foreach (string strFile in
               System.IO.Directory.GetFiles(FolderName))
     {
          // If the file has a url extension...
          if (strFile.EndsWith(".url",true,null))
          {
               // Create and use a new instance
               // of the WebFavorite class
               WebFavorite objWebFavorite =
                              new WebFavorite();
               // Load the file information
               objWebFavorite.Load(strFile);
               // Add the object to the collection
               FavoriteCollection.Add(objWebFavorite);
          }
     }
}
```

برای ایجاد برنامه ی اصلی، باید شیئ ای از نوع Favorites ایجاد کرده، فولدر Favorites را جستجو کنید و عناصری که در آن فولدر پیدا می شود را به لیست اضافه کنید. این کار در امتحان کنید بعد انجام خواهد شد.

چگونه کار می کند؟

همانطور که در فصل پنجم مشاهده کردید، برای ایجاد یک آرایه که طول مشخصی نداشته باشد و بتوانید به راحتی اشیایی را به آن اضافه کرده و یا حذف کنید، باید از کلاس ArrayList استفاده کنید. در این قسمت نیز شیئ ای از این نوع را ایجاد می کنیم تا بتوانیم اشیای ایجاد شده از نوع WebFavorite (که تعداد آنها نیز مشخص نیست) را در آن قرار دهیم.
اما سوال اینجاست که چگونه این لیست را پر کنیم؟ خوب، در کلاس Favorites متدی سربار گذاری شده به نام ScanFavorites ایجاد می کنیم. نسخه ی دوم این متد آدرس فولدری را به عنوان پارامتر دریافت می کند و درون آن فولدر به دنبال فایلهایی با پسوند .url می گردد. امام بهتر است قبل از بررسی نحوه کار این متد، نحوه کارکرد خاصیت FavoritesFolder را بررسی کنیم.
به دلیل اینکه بر حسب کاربری که هم اکنون در حال استفاده از کامپیوتر است ممکن است آدرس فولدر Favorites تغییر کند، بهترین راه برای بدست آوردن آدرس این فولدر، پرسیدن آن از ویندوز است. برای این کار می توانیم از متد GetFolderPath که در کلاس System.Environment قرار دارد استفاده کنیم:

```
public string FavoritesFolder
```

```
{
      get
      {
            // Return the path to the user's Favorites folder
            return Environment.GetFolderPath(
                        Environment.SpecialFolder.Favorites);
      }
}
```

این متد یکی از گزینه های شمارنده ی `Environment.SpecialFolder` را به عنوان پارامتر دریافت می کند و آدرس آن فولدر را برمی گرداند. شمارنده ی `SpecialFolder` حاوی نام تعدادی از فولدرهای خاص است که معمولاً هنگام برنامه نویسی زیاد مورد استفاده قرار می گیرد (مانند فولدر `My Documents` و یا `Program Files`).
برای اینکه در برنامه ی اصلی از کلاس `Favorites` بخواهیم تا با جستجوی فولدر `Favorites`، لیست خود را پر کند باید نسخه ی اول از متد `ScanFavorites` را فراخوانی کنیم. این نسخه با استفاده از خاصیت `FavoritesFolder`، آدرس فولدر `Favorites` را بدست آورده و آن را به عنوان پارامتر به نسخه ی دوم از این متد می فرستد.

```
public void ScanFavorites()
{
      // Scan the favorites folder
      ScanFavorites(this.FavoritesFolder);
}
```

کاری که نسخه ی دوم از متد باید انجام دهد این است که فایلهای موجود در آدرسی که به آن فرستاده شده است را بدست آورده و آنها را پردازش کند. برای بدست آوردن لیست فایلهای موجود در آن فولدر می توانیم از متد `GetFiles` در کلاس `System.IO.Directory` استفاده کنیم. این متد آدرس فولدری را که باید بررسی کند را به عنوان پارامتر دریافت می کند و آرایه ای از نام فایلهای موجود در آن را برمی گرداند. به این ترتیب می توانیم با استفاده از یک حلقه ی `foreach` آنها را بررسی کنیم. متغیری را از نوع رشته ای و به نام `strFiles` در حلقه تعریف می کنیم تا با هر بار گردش حلقه نام یکی از فایل ها در آن قرار گیرد.

```
      // Process each file in the Favorites folder
      foreach (string strFile in
                  System.IO.Directory.GetFiles(FolderName))
      {
```

درون حلقه ی `foreach` ابتدا بررسی می کنیم که آیا پسوند فایل برابر با `.url` است یا نه؟ برای این کار می توانیم از متد `EndsWith` در کلاس `String` استفاده کنیم. متد `EndsWith` یک متد سربار گذاری شده است و نسخه ای که در این برنامه از آن استفاده می کنیم سه پارامتر دریافت می کند. پارامتر اول عبارتی است که باید با انتهای رشته مقایسه شود، در اینجا عبارت `".url"` را برای این پارامتر استفاده می کنیم. پارامتر دوم یک مقدار `Boolean` است که مشخص می کند آیا متد `EndsWith` در هنگام مقایسه حالت حروف را نیز در نظر بگیرد یا نه؟ در این قسمت نمی خواهیم مقایسه ی متن به صورت حساس به بزرگی و یا کوچکی حروف انجام شود پس مقدار `true` را برای این پارامتر استفاده می کنیم. پارامتر سوم هم شیئ ای

را از نوع `System.Globalization.CultureInfo` دریافت می کند، این شیئ مشخص کننده تنظیمات محلی است که باید هنگام مقایسه در نظر گرفته شوند، اما در اینجا برای اینکه تنظیمات محلی پیش فرض به کار گرفته شود از عبارت `null` برای این پارامتر استفاده می کنیم.

```
// If the file has a url extension...
if (strFile.EndsWith(".url",true,null))
{
```

اگر فایلی که در حال بررسی است دارای پسوند `.url` باشد باید شیئ جدیدی از نوع `WebFavorite` ایجاد کنید، اطلاعات آن فایل را در داخل شیئ قرار دهید و سپس شیئ را به آرایه ی `Favorites` اضافه کنید. برای این کار ابتدا باید شیئ ای را از نوع `WebFavorite` ایجاد کرده و سپس متد `Load` را در آن شیئ فراخوانی کنید و آدرس فایل را به آن متد بفرستید، در انتها نیز شیئ را در آرایه قرار دهید.

```
// Create and use a new instance
// of the WebFavorite class
WebFavorite objWebFavorite = new WebFavorite();
// Load the file information
objWebFavorite.Load(strFile);
// Add the object to the collection
FavoriteCollection.Add(objWebFavorite);
```

در بخش امتحان کنید بعد، با استفاده از کلاس هایی که در این بخش ایجاد کردیم لیست داخل فرم را با فایلهای موجود در فولدر `Favorites` کاربر کامل می کنیم.

امتحان کنید: ایجاد شیئ ای از کلاس `Favorites`

۱) در فرم اصلی برنامه، روی قسمتی خالی از فرم دو بار کلیک کنید تا متد مربوط به رویداد `Load` آن ایجاد شود. سپس کد مشخص شده در زیر را در این متد وارد کنید.

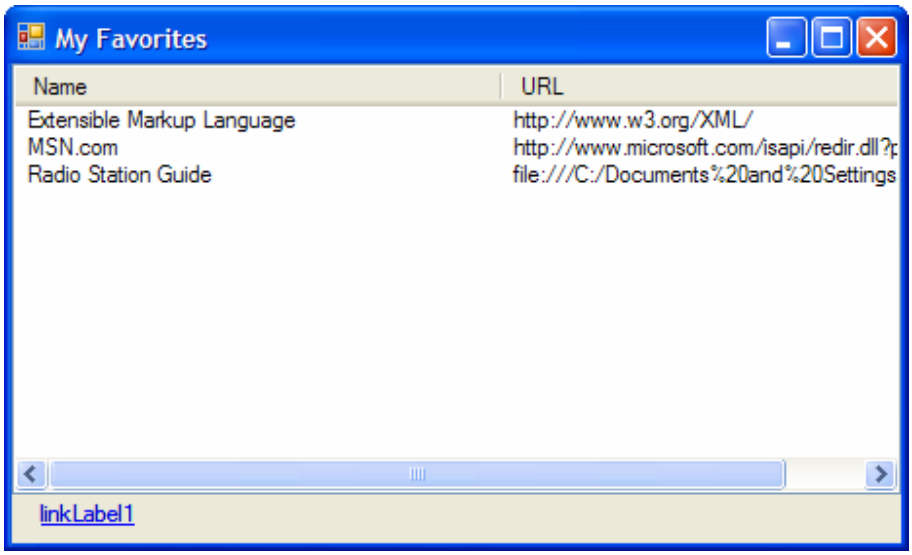
```
private void Form1_Load(object sender, EventArgs e)
{
    // Create a new instance of the Favorites class
    Favorites objFavorites = new Favorites();

    // Scan the Favorites folder
    objFavorites.ScanFavorites();

    // Process each objWebFavorite object
    // in the Favorites collection
    foreach (WebFavorite objWebFavorite in
                    objFavorites.FavoriteCollection)
```

```
    {
        // Declare a ListViewItem object
        ListViewItem objListViewItem = new
                                        ListViewItem();
        // Set the properties of ListViewItem object
        objListViewItem.Text = objWebFavorite.Name;
        objListViewItem.SubItems.Add(objWebFavorite.Url);
        // Add the ListViewItem object to the ListView
        lstFavorites.Items.Add(objListViewItem);
    }
}
```

۲) برنامه را اجرا کنید، پنجره ای مشابه شکل ۱۰-۱۷ مشاهده خواهید کرد.



شکل ۱۰-۱۷

چگونه کار می کند؟

در متد مربوط به رویداد `Load` فرم، ابتدا شیئ را از نوع `Favorites` ایجاد کرده و سپس نسخه ی بدون پارامتر از متد `ScanFavorites` را فراخوانی می کنیم. به این ترتیب از کلاس می خواهیم که هنگام `load` شدن فرم، آرایه ی جدیدی به نام `FavoritesCollection` ایجاد کرده و بازای هر یک از گزینه هایی که در فولدر `Favorites` دارد یک شیئ از نوع `WebFavorite` تشکیل دهد و آن را به آرایه اضافه کند.

```
// Create a new instance of the Favorites class
Favorites objFavorites = new Favorites();

// Scan the Favorites folder
objFavorites.ScanFavorites();
```

بعد از اتمام متد ScanFavorites، آرایه ی FavoritesCollection با عناصری از نوع WebFavorite پر شده است. حال باید با استفاده از عناصر این آرایه در کلاس Favorites، آیتم های درون لیست را کامل کنیم. برای این کار مانند قسمتهای قبل از یک حلقه ی foreach استفاده می کنیم تا بتوانیم تمام عناصر آرایه را پیمایش کنیم.
قبل از ادامه بهتر است به این نکته توجه کنید که آیتم های درون کنترل ListView، اشیای از کلاس ListViewItem هستند که در یک آرایه قرار دارند. بنابراین برای اینکه یک آیتم به این کنترل اضافه کنیم، باید یک شیئ از نوع ListViewItem ایجاد کرده و آن را به کنترل اضافه کنیم.
درون حلقه ی foreach ابتدا شیئ ای را از کلاس ListViewItem نمونه سازی کرده و خاصیت Text آن را با توجه به فیلد Name در کلاس WebFavorite کامل می کنیم. سپس آدرس لینک مورد نظر را نیز که در فیلد Url قرار دارد به خاصیت SubItems از ListViewItem اضافه می کنیم.

```
// Declare a ListViewItem object
ListViewItem objListViewItem = new ListViewItem();
// Set the properties of ListViewItem object
objListViewItem.Text = objWebFavorite.Name;
objListViewItem.SubItems.Add(objWebFavorite.Url);
```

در انتها نیز شیئ ListViewItem را به خاصیت Items از کنترل ListView اضافه می کنیم تا در فرم نمایش داده شود.

```
// Add the ListViewItem object to the ListView
lstFavorites.Items.Add(objListViewItem);
```

خوب، تا اینجا برنامه ی ما قادر است گزینه های موجود در منوی Favorites اینترنت اکسپلورر را به صورت یک لیست در فرم نمایش دهد، اما هنوز نمی تواند سایتی که این لینک ها به آن اشاره می کنند را باز کند. نحوه ی انجام این کار را نیز در بخش بعدی بررسی خواهیم کرد.

مشاهده ی لینک ها:

در بخش امتحان کنید بعد می خواهیم قابلیتی را به برنامه اضافه کنیم تا کاربر بتواند با انتخاب یک لینک از لیست، محتویات آن لینک را درون اینترنت اکسپلورر مشاهده کند.

امتحان کنید: مشاهده ی لینک ها

۱) به قسمت طراحی مربوط به Form1 بروید و کنترل lstFavorites را از فرم انتخاب کنید. سپس در پنجره ی Properties روی آیکون Events کلیک کنید تا لیستی از رویداد ای این کنترل نمایش داده شود. در

این لیست رویداد Click را پیدا کرده و روی آن دو بار کلیک کنید تا متد مربوط به آن ایجاد شود، سپس کد زیر را به این متد اضافه کنید:

```
private void lstFavorites_Click(object sender, EventArgs e)
{
     // Update the link label control Text property
     lnkUrl.Text = "Visit " +
                    lstFavorites.SelectedItems[0].Text;

     // Clear the default hyperlink
     lnkUrl.Links.Clear();

     // Add the selected hyperlink to the LinkCollection
     lnkUrl.Links.Add(6,
          lstFavorites.SelectedItems[0].Text.Length,
          lstFavorites.SelectedItems[0].SubItems[1].Text);
}
```

۲) مجدداً به قسمت طراحی فرم برگردید و روی کنترل LinkLabel دو بار کلیک کنید تا متد مربوط به رویداد LinkClicked آن ایجاد شود. سپس کد زیر را به این متد اضافه کنید:

```
private void lnkUrl_LinkClicked(object sender,
                          LinkLabelLinkClickedEventArgs e)
{
     System.Diagnostics.Process.Start(
                          e.Link.LinkData.ToString() );
}
```

۳) برنامه را اجرا کنید. مشاهده می کنید که با کلیک کردن روی هر کدام از آیتم های درون لیست، کنترل پایین فرم تغییر کرده تا نام آن آیتم را نمایش دهد (شکل ۱۰-۱۵ را مشاهده کنید). اگر روی این نام کلیک کنید، اینترنت اکسپلورر باز شده و سایت مربوط به آن را نمایش می دهد.

چگونه کار می کند؟

هنگامی که روی یکی از آیتم های درون کنترل ListView کلیک می کنید، متد مربوط به رویداد Click این کنترل فراخوانی می شود. در این متد کدی را قرار می دهیم تا بر اساس آیتمی که در لیست انتخاب شده است متن مناسبی را در کنترل LinkLabel نمایش دهد.

همانطور که گفتم هر یک از آیتم های درون کنترل ListView یک شیئ از کلاس ListViewItem است. برای دسترسی به آیتمی که هم اکنون در لیست انتخاب شده است می توانیم از خاصیت SelectedItems استفاده کنیم که به صورت آرایه ای از نوع ListViewItem است. دلیل آرایه ای بودن این خاصیت این است که در کنترل ListView کاربر می تواند بیش از یک آیتم را از لیست انتخاب کند، بنابراین خاصیت SelectedItems باید به صورت یک آرایه باشد

تا بتوانیم به تمام آیتم های انتخاب شده دسترسی داشته باشیم. البته در این برنامه، ما فقط آیتم اول را در کنترل LinkLabel نمایش می دهیم بنابراین در آرایه ی SelectedItems فقط به عنصر اول (با اندیس صفر) نیاز داریم. برای دسترسی به اطلاعات دیگر ستون های فیلد انتخاب شده در لیست نیز می توانیم از خاصیت SubItems استفاده کنیم. مثلاً برای دسترسی به آدرس لینک انتخاب شده که در ستون اول قرار دارد، از SubItems[1] استفاده می کنیم.
برای تنظیم متنی که در کنترل LinkLabel نمایش داده می شود، خاصیت Text آن را برابر با ثابت رشته ای "Visit " به اضافه ی نام آیتم انتخاب شده در لیست قرار می دهیم. برای دسترسی به نام آیتم انتخاب شده نیز می توانیم از خاصیت Text آن استفاده کنیم:

```
// Update the link label control Text property
lnkUrl.Text = "Visit " +
              lstFavorites.SelectedItems[0].Text;
```

خاصیت Links از کنترل LinkLabel، به صورت آرایه ای از نوع LinkCollection است و شامل لینک هایی اینترنتی می شود که این کنترل به آنها اشاره می کند. بنابراین باید Url آیتمی که در لیست lstFavorites انتخاب شده است را به این خاصیت اضافه کنیم. اما ابتدا بهتر است که محتویات آن را با استفاده از متد Clear پاک کنیم.

```
// Clear the default hyperlink
lnkUrl.Links.Clear();
```

حال می توانیم با استفاده از متد Add، لینک مربوط به آیتم انتخاب شده در لیست را به آن اضافه کنیم. متد Add یک متد سربار گذاری شده است و نسخه ای از این متد که از آن استفاده می کنیم، سه پارامتر دریافت می کند: Start، Length و LinkData. پارامتر Start مشخص می کند که از کجای رشته ای که به این متد فرستاده می شود باید به عنوان لینک در نظر گرفته شود. پارامتر Length مشخص کننده طول رشته ای است که باید به عنوان لینک مشخص شود. پارامتر LinkData نیز رشته ی حاوی لینک را مشخص می کند. در اینجا برای اینکه عبارت "Visit " حذف شود، نقطه شروع را ۶ در نظر گرفته ایم، همچنین طول لینک را نیز برابر با طول لینکی که در لیست انتخاب شده است قرار داده ایم.

```
// Add the selected hyperlink to the LinkCollection
lnkUrl.Links.Add(6,
     lstFavorites.SelectedItems[0].Text.Length,
     lstFavorites.SelectedItems[0].SubItems[1].Text);
```

هنگامی که کاربر روی کنترل LinkLabel کلیک کند متد مربوط به رویداد Click این کنترل فراخوانی می شود، بنابراین در این متد باید کدی را قرار دهیم تا محتویات لینک را نمایش دهد. به این متد پارامتری به نام e از نوع LinkLabelLinkClickedEventArgs فرستاده می شود که حاوی اطلاعاتی مانند آدرس لینک مورد نظر است. برای نمایش لینک کافی است این آدرس را بدست آوریم و آن را به عنوان پارامتر به متد Start از کلاس System.Diagnostics.Process بفرستیم. به این ترتیب، مرورگر اینترنت باز می شود و محتویات لینک انتخاب شده را نمایش می دهد.

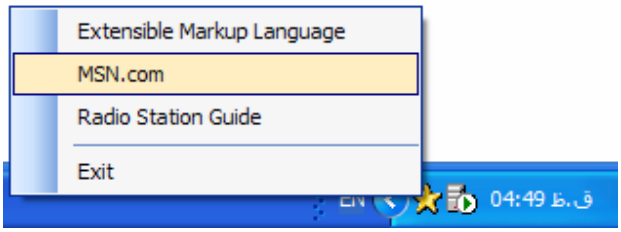
```
System.Diagnostics.Process.Start(
                    e.Link.LinkData.ToString() );
```

ایجاد نمونه ای دیگر از برنامه ی Favorite Viewer:

همانطور که گفتم استفاده از کلاسها برای جدا سازی قسمتهای مختلف برنامه، این امکان را می دهد تا بتوانیم از آن کدها در قسمتهای دیگر نیز استفاده کنیم. برای اثبات این مطلب در این قسمت برنامه ی دیگری مشابه برنامه ی Favorite Viewer قسمت قبل، ایجاد می کنیم و از کلاسهای آن برنامه، مانند کلاس Favorites و یا WebFavorite در این برنامه نیز استفاده می کنیم.

ایجاد برنامه ی Favorites Tray:

در این قسمت برنامه ای ایجاد خواهیم کرد که به صورت یک آیکون در کنار ساعت سیستم قرار گیرد و با کلیک کردن کاربر بر روی این آیکون لیستی از گزینه های Favorites، به صورت یک منو نمایش داده شود (شکل ۱۰-۱۸). به این ترتیب کاربر می تواند با کلیک کردن روی هر کدام از گزینه های این منو محتویات آن لینک را در اینترنت اکسپلورر مشاهده کند.



شکل ۱۰-۱۸

امتحان کنید: ایجاد برنامه ی Favorites Tray

۱) در محیط ویژوال استودیو، گزینه ی File ➔ New ➔ Project... را از نوار منو انتخاب کنید و به وسیله ی کادر نمایش داده شده، یک پروژه ی ویندوزی با ویژوال C# به نام Favorites Tray ایجاد کنید.

۲) هنگامی که فرم برنامه نمایش داده شد، مقدار خاصیت WindowState آن را به Minimized و مقدار خاصیت ShowInTaskBar را به False تغییر دهید. به این ترتیب از نمایش داده شدن فرم جلوگیری خواهید کرد.

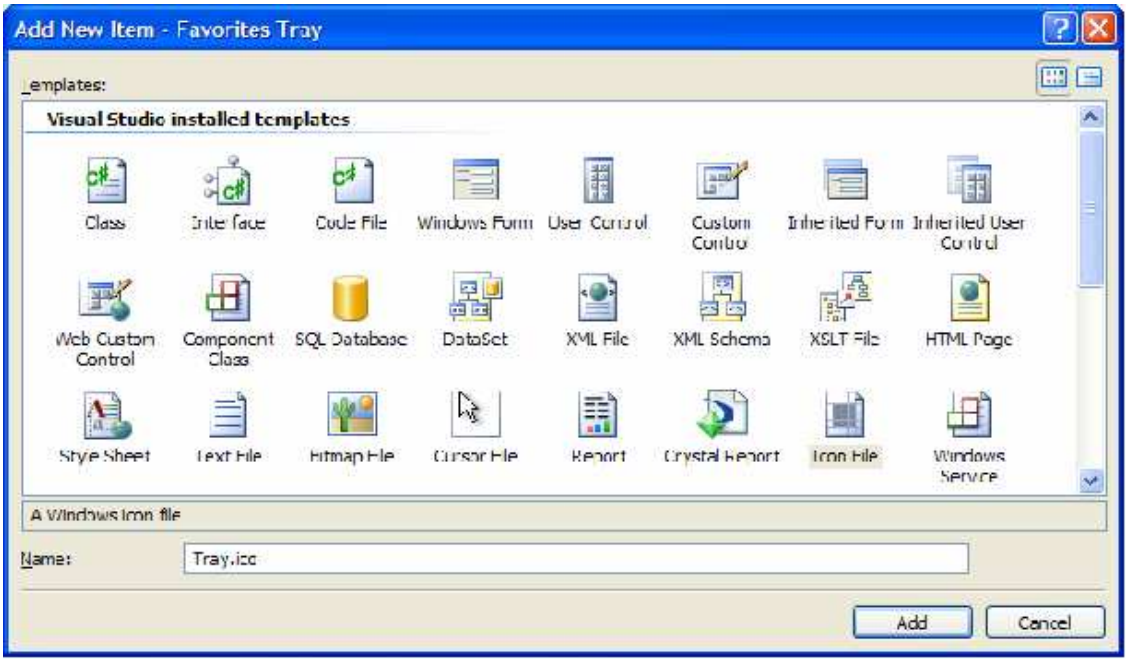
۳) با استفاده از جعبه ابزار یک کنترل NotifyIcon در فرم قرار داده، خاصیت Name این کنترل را برابر با icnNotify و خاصیت Text آن را به Right-click me to view Favorites تغییر دهید.

۴) حال فرم برنامه را انتخاب کرده و سپس در پنجره ی Properties روی آیکون Events کلیک کنید تا لیست رویداد های مربوط به فرم نمایش داده شوند. در این لیست رویداد VisibleChanged را انتخاب کرده و روی آن دو بار کلیک کنید تا متد مربوط به این رویداد ایجاد شود. سپس کد مشخص شده در زیر را در این متد قرار دهید.

```
private void Form1_VisibleChanged(object sender,
```

```
                                        EventArgs e)
{
     // If the user can see us, hide us
     if (this.Visible)
          this.Visible = false;
}
```

۵) حال باید آیکونی را برای برنامه ایجاد کنیم، در غیر این صورت هیچ چیز که مشخص کننده برنامه باشد در سیستم نمایش داده نمی شود و به این ترتیب کاربر نمی تواند از برنامه بخواهد تا کاری را انجام دهد. با استفاده از پنجره ی Solution Explorer بر روی نام برنامه (Favorites Tray) کلیک راست کنید و از منوی باز شده گزینه ی Add ➔ New Item را انتخاب کنید. در قسمت Tamplates همانند شکل ۱۰-۱۹، گزینه ی Icon File را انتخاب کنید و در کادر Name هم نام Tray.ico را وارد کنید. سپس روی کلید Add کلیک کنید تا این فایل به برنامه اضافه شود.



شکل ۱۰-۱۹

۶) به این ترتیب ابزار Image Editor در ویژوال استودیو نمایش داده می شود. از این ابزار می توانید برای طراحی آیکون ها، اشاره گرها و یا فایلهای تصویری جدید در برنامه استفاده کنید. در نوار ابزار این قسمت، همانند شکل ۱۰-۲۰ ابزارهایی که می توانید برای این کار از آنها استفاده کنید را مشاهده می کنید.



شکل ۱۰-۲۰

۷) اگر جعبه رنگ همانند شکل ۱۰-۲۱ در صفحه دیده نمی شود، با انتخاب گزینه ی Image ➔ Show Color Window این پنجره را به صفحه اضافه کنید.



شکل ۱۰-۲۱

۸) قبل از اینکه طراحی آیکون را شروع کنیم، بهتر است نوع آیکون را تنظیم کنیم. ویژوال استودیو به صورت پیش فرض یک آیکون 32*32 ایجاد می کند، اما این آیکون بزرگتر از چیزی است که نیاز داریم. با استفاده از نوار منو گزینه ی Image ➔ New Image Type را انتخاب کنید تا کادر New Icon Image Type نمایش داده شود. سپس در این کادر گزینه ی 16*16, 256 Color را انتخاب کرده و روی OK کلیک کنید.

۹) به این ترتیب یک آیکون 16*16 نیز به برنامه اضافه می شود، اما آیکون قبلی از بین نمی رود. اگر آیکون قبلی همچنان در برنامه باقی بماند ممکن است بعدها با مشکل مواجه شویم، پس بهتر است آن را حذف کنیم. برای این کار گزینه ی Image ➔ Current Image Type ➔ 32*32, 16 Color را از نوار منو انتخاب کنید. سپس بلافاصله گزینه ی Image ➔ Delete Image Type را انتخاب کنید. همین مراحل را برای گزینه ی 16*16, 16 Color نیز تکرار کنید. به این ترتیب فقط یک آیکون به صورت 16*16, 256 Color در این قسمت باقی می ماند.

۱۰) اگر حس می کنید که فرد خلاقی هستید می توانید آیکون مورد نظرتان را برای این برنامه طراحی کنید، در غیر این صورت می توانید کاری که ما در این قسمت انجام می دهیم را انجام دهید. یعنی به نحوی از آیکون اینترنت اکسپلورر برای این قسمت استفاده کنید.

۱۱) با انتخاب گزینه ی File ➔ Save Tray.ico از نوار منو آیکون برنامه را در دیسک ذخیره کنید.

۱۲) به قسمت طراحی فرم برگردید و کنترل icnNotify را انتخاب کنید. سپس با استفاده از پنجره ی Properties، خاصیت Icon این کنترل را برابر با آیکونی که در مرحله ی قبل ایجاد کردید قرار دهید.

۱۳) حال برنامه را اجرا کنید. مشاهده ه می کنید آیکونی که طراحی کرده اید در کنار ساعت سیستم قرار می گیرد، اما هیچ پنجره ای از برنامه در صفحه نمایش داده نمی شود (شکل ۱۰-۲۲). همچنین اگر اشاره گر ماوس را برای لحظاتی روی این آیکون قرار دهید، متنی را که در خاصیت Text کنترل Notify Icon وارد کرده بودید مشاهده می کنید.



شکل ۱۰-۲۲

۱۴) همانطور که مشاهده می کنید در این حالت هیچ راهی برای بستن برنامه نیز وجود ندارد. به محیط ویژوال استودیو برگردید و گزینه ی Debug ➔ Stop Debugging را انتخاب کنید تا اجرای برنامه متوقف شود.

۱۵) با انجام این کار اجرای برنامه متوقف می شود اما آیکون آن همچنان در صفحه باقی می ماند. برای حذف آیکون نیز، اشاره گر ماوس را بر روی آن قرار دهید تا ناپدید شود.

چگونه کار می کند؟

معین کردن این که فرم در Taskbar نمایش داده نشود (ShowInTaskBar = False) و همچنین به صورت Minimize اجرا شود (WindowState = Minimized) موجب می شود که فرم برنامه قابل مشاهده نباشد. در این برنامه هم فقط می خواهیم آیکون برنامه در کنار ساعت نمایش داده شود و به نمایش داده شدن فرم برنامه نیازی نداریم. اما تاکنون فقط نیمی از کار را انجام داده اید زیرا باز هم فرم برنامه با استفاده از کلیدهای Alt + Tab در صفحه نمایش داده می شود. برای جلوگیری از این کار می توانیم از کد زیر استفاده کنیم.

```
private void Form1_VisibleChanged(object sender,
                                  EventArgs e)
{
     // If the user can see us, hide us
     if (this.Visible)
          this.Visible = false;
}
```

به این ترتیب کاربر با استفاده از کلیدهای Alt + Tab هم نمی تواند فرم برنامه را ببیند.

نمایش گزینه های Favorites:

در بخش امتحان کنید بعدی، قابلیت نمایش گزینه های موجود در فولدر Favorites را به این برنامه اضافه خواهیم کرد. اما برای این کار لازم است که کلاس هایی که در برنامه Favorites Viewer ایجاد کرده بودیم را به این برنامه اضافه کنیم.

امتحان کنید: نمایش گزینه های Favorites

۱) برای کامل کردن برنامه ی Favorites Tray باید اشیایی را از کلاسهای WebFavorite و Favorites ایجاد کنیم. پس ابتدا باید این کلاسها را به برنامه اضافه کنیم. با استفاده از پنجره ی Solution Explorer روی نام پروژه ی Favorites Tray کلیک راست کنید و از منوی باز شده گزینه ی ... Add ➔ Existing Item را انتخاب کنید. با استفاده از کادری که باز می شود فایل مربوط به کلاس Favorites را پیدا کنید (این فایل در فولدر مربوط به برنامه ی Favorites Viewer قرار دارد) و سپس روی کلید Add کلیک کنید. به این ترتیب فایل مربوط به این کلاس به برنامه اضافه شده و در پنجره ی Solution Explorer قابل مشاده است.

۲) همین مراحل را تکرار کنید تا فایل مربوط به کلاس WebFavorite نیز به برنامه اضافه شود.

۳) به قسمت طراحی فرم بروید و کنترل NotifyIcon را انتخاب کنید. سپس به پنجره ی Properties بروید و روی آیکون Events کلیک کنید تا لیستی از رویداد های این کنترل نمایش داده شود. در این لیست رویداد Click را انتخاب کرده و روی آن دو بار کلیک کنید تا متد مربوط به این رویداد ایجاد شود. سپس کد زیر را به این متد اضافه کنید:

```
private void icnNotify_Click(object sender, EventArgs e)
{
     // Create a new instance of the Favorites class
     Favorites_Viewer.Favorites objFavorites =
                    new Favorites_Viewer.Favorites();

     // Scan the Favorites folder
     objFavorites.ScanFavorites();

     // Clear current menu items
     FavoritesMenu.Items.Clear();

     // Process each objWebFavorite object
     // in the Favorites collection
     foreach (Favorites_Viewer.WebFavorite objWebFavorite
                    in objFavorites.FavoriteCollection)
     {
          // Declare a ToolStripMenuItem object
          ToolStripMenuItem objMenuItem =
                              new ToolStripMenuItem();

          // Set the properties of ToolStripMenuItem object
          objMenuItem.Text = objWebFavorite.Name;
          objMenuItem.Tag = objWebFavorite.Url;

          // Add a handler to Click event of new menu item
          objMenuItem.Click +=
                    new EventHandler(MenuItems_Click);

          // Add the ToolStripMenuItem object
          // to the ContextMenu
          FavoritesMenu.Items.Add(objMenuItem);
     }

     // Create a Seperator item and adding it
     // to context menu
     ToolStripSeparator objSeperatorItem =
                         new ToolStripSeparator();
     FavoritesMenu.Items.Add(objSeperatorItem);

     // Create an Exit menu item and set it's properties
     ToolStripMenuItem objExitItem =
                         new ToolStripMenuItem();
     objExitItem.Text = "Exit";
     objExitItem.Click +=
                    new EventHandler(ExitMenuItem_Click);
```

```
        // Add Exit menu item to context menu
        FavoritesMenu.Items.Add(objExitItem);
}
```

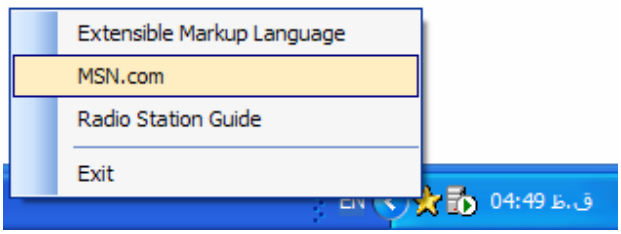
۴) حال باید دو متد برای کنترل رویداد کلیک منوی برنامه ایجاد کنیم. متد اول هنگامی فراخوانی می شود که کاربر روی یکی از لینک ها کلیک کند، پس این متد باید بتواند اینترنت اکسپلورر را باز کند و لینک را نمایش دهد. برای این کار متد زیر را به کلاس `Form1` اضافه کنید:

```
private void MenuItems_Click(object sender,
                             System.EventArgs e)
{
        // Create a ToolStripMenuItem
        // and fill it with sender parameter
        ToolStripMenuItem s = (ToolStripMenuItem)sender;
        // Open the internet explorer to view selected
        // favorite
        System.Diagnostics.Process.Start(s.Tag.ToString());
}
```

۵) متد دوم نیز هنگامی احضار می شود که کاربر روی گزینه ی `Exit` کلیک می کند و می خواهد از برنامه خارج شود. برای اضافه شدن این متد، کد زیر را به کلاس `Form1` اضافه کنید:

```
private void ExitMenuItem_Click(object sender,
                                System.EventArgs e)
{
        Application.Exit();
}
```

۶) با اجرای برنامه، آیکونی مانند مراحل قبل در کنار ساعت سیستم نمایش داده می شود. با کلیک راست بر روی این آیکون، لیستی همانند شکل ۱۰-۲۳ نمایش داده می شود. هر کدام از لینک های موجود در این لیست را که انتخاب کنید، اینترنت اکسپلورر باز می شود و محتویات آن را نمایش می دهد.



شکل ۱۰-۲۳

چگونه کار می کند؟

همانطور که مشاهده می کنید این برنامه نیز تقریبا مشابه برنامه ی Favorites Viewer است، فقط تغییرات کوچکی در آن ایجاد شده است. دلیل این تغییرات نیز این بوده است که ظاهر برنامه را تغییر داده ایم، در غیر این صورت قسمت اصلی برنامه که از کلاسهای Favorites و WebFavorite تشکیل شده است، با برنامه قبلی برابر است.
پروسه ی مربوط به ایجاد منوی برنامه را این بار به جای آنکه در رویداد Load فرم قرار دهیم، در رویداد Click مربوط به کنترل NotifyIcon قرار می دهیم. به این ترتیب هر بار که کاربر روی آیکون کلیک کند، ابتدا منو ایجاد شده و سپس نمایش داده می شود.
در این متد همانند برنامه ی قبلی، ابتدا شیئ ای از کلاس Favorites ایجاد کرده و سپس متد ScanFavorites را در آن فراخوانی می کنیم تا آرایه ی FavoriteCollection را کامل کند. البته دقت کنید به علت اینکه فضای نام کلاس Favorites برابر با Favorite_Viewer است، یا باید این فضای نام را با استفاده از راهنمای using به برنامه اضافه کنیم و یا همانند بالا، نام کامل کلاس را ذکر کنیم.

```
// Create a new instance of the Favorites class
Favorites_Viewer.Favorites objFavorites =
                new Favorites_Viewer.Favorites();

// Scan the Favorites folder
objFavorites.ScanFavorites();
```

قبل از این که در تک تک عناصر آرایه بگردیم و آنها را در منو قرار دهیم، بهتر است منو را خالی کنیم.

```
// Clear current menu items
FavoritesMenu.Items.Clear();
```

همانند کنترل ListView، گزینه های موجود در کنترل ContextMenuStrip نیز در حقیقت اشیایی از کلاس ToolStripMenuItem هستند. بنابراین در حلقه ی foreach بازای هر عنصری که در آرایه ی FavoriteCollection قرار دارد، یک شیئ از کلاس ToolStripMenuItem ایجاد کرده و خاصیت های آن را تنظیم می کنیم.
اولین خاصیتی که باید تنظیم شود متنی است که برای این گزینه نمایش داده می شود. با تنظیم خاصیت Text این شیئ، متن مناسبی را برای آن قرار می دهیم. همچنین باید به نحوی لینک گزینه ی مربوطه را نیز در آن قرار دهیم تا هنگامی که کاربر روی آن گزینه کلیک کرد، منو بداند چه لینکی را باید به وسیله ی اینترنت اکسپلورر نمایش دهد. برای این کار لینک را با استفاده از خاصیت Url بدست آورده و در خاصیت Tag قرار می دهیم.

```
// Declare a ToolStripMenuItem object
ToolStripMenuItem objMenuItem =
                    new ToolStripMenuItem();

// Set the properties of ToolStripMenuItem object
objMenuItem.Text = objWebFavorite.Name;
objMenuItem.Tag = objWebFavorite.Url;
```

حال باید در این مرحله مشخص کنیم هنگامی که کاربر روی این گزینه کلیک می کند، چه متدی باید فراخوانی شود. چون این متد مربوط به رویداد کلیک است، پس باید بر طبق ساختار خاصی باشد. به عبارت دیگر این متد حتماً باید از نوع `void` باشد و دو پارامتر، یکی از نوع `System.EventArgs` و یکی از نوع `object` دریافت کند. بنابراین متدی به نام `MenuItems_Click` با این مشخصات ایجاد می کنیم، سپس با اضافه کردن این متد به رویداد `Click` شیئ ای که از نوع `ToolStripMenuItem` ایجاد کرده ایم، مشخص می کنیم که این متد باید هنگام کلیک شدن روی این گزینه فراخوانی شود.

```
// Add a handler to Click event of new menu item
objMenuItem.Click += new
                     EventHandler(MenuItems_Click);
```

به این ترتیب تمام خاصیت های مورد نیاز برای شیئ `ToolStripMenuItem` را تنظیم کرده ایم و می توانیم آن را به منوی برنامه اضافه کنیم.

```
// Add the ToolStripMenuItem object to the ContextMenu
FavoritesMenu.Items.Add(objMenuItem);
```

بعد از اتمام این حلقه، گزینه های موجود در فولدر `Favorites` به منو اضافه شده اند، اما هنوز منو کامل نشده است و باید دو گزینه ی دیگر نیز به آن اضافه شود: یک خط جدا کننده و یک گزینه برای خروج از برنامه. برای اضافه کردن خط جدا کننده به منو باید شیئ ای را از نوع `ToolStripSeperator` ایجاد کرده و آن را به منو اضافه کنیم.

```
// Create a Seperator item and adding it
// to context menu
ToolStripSeparator objSeperatorItem =
                new ToolStripSeparator();
FavoritesMenu.Items.Add(objSeperatorItem);
```

برای ایجاد گزینه ی خروج از برنامه نیز کافی است مانند قسمتهای قبل عمل کنیم. اما دقت داشته باشید دی که در رویداد `Click` این گزینه قرار می گیرد با کد رویداد `Click` دیگر گزینه ها مقداری تفاوت دارد. پس برای آن، متدی جداگانه (اما با همان ساختار و با همان پارامترهای ورودی و خروجی) به نام `ExitMenuItem_Click` ایجاد کرده و آن را به رویداد `Click` شیئ اضافه می کنیم. در آنها نیز شیئ را به منو اضافه می کنیم.

```
// Create an Exit menu item and set it's properties
ToolStripMenuItem objExitItem =
                    new ToolStripMenuItem();
objExitItem.Text = "Exit";
objExitItem.Click +=
                new EventHandler(ExitMenuItem_Click);

// Add Exit menu item to context menu
FavoritesMenu.Items.Add(objExitItem);
```

کد درون متد ExitMenuItem_Click ساده است و تاکنون به مراتب از آن استفاده کرده اید، بنابراین نیازی به توضیح ندارد. پس به سراغ کد درون متد MenuItems_Click می رویم. همانطور که گفتم متد هایی که برای رویداد کلیک فراخوانی می شوند، باید دو پارامتر دریافت کنند. یکی از این پارامتر ها شیئ ای از کلاس Object به نام sender است. هنگامی که در برنامه رویدادی برای یک شیئ رخ بدهد، برای مثال روی یکی از گزینه های منو کلیک شود درون متد با استفاده از پارامتر sender می توانیم به این شیئ دسترسی پیدا کنیم. پس درون متد MenuItems_Click هم می توانیم با استفاده از پارامتر sender به شیئ ای که روی آن کلیک شده است (و مسلماً از نوع ToolStripMenuItem است) دسترسی داشته باشیم.
در این متد می خواهیم به اطلاعاتی که درون خاصیت Tag مربوط به شیئ قرار دارد دسترسی پیدا کنیم، پس ابتدا شیئ را از نوع Object به نوع ToolStripMenuItem تبدیل کرده و متن داخل Tag را بدست می آوریم. همانطور که مشاهده کردید قبلا لینک مربوط به هر گزینه را در این خاصیت قرار دادیم، پس حالا هم با به دست آوردن مقدار خاصیت Tag، لینک مربوط به آن گزینه را بدست آورده ایم. حال فقط کافی است این لینک را به متد Start از کلاس System.Diagnostics.Process بفرستیم تا آن را در اینترنت اکسپلورر نمایش دهد.

```
// Create a ToolStripMenuItem
// and fill it with sender parameter
ToolStripMenuItem s = (ToolStripMenuItem)sender;
// Open the internet explorer to view selected
// favorite
System.Diagnostics.Process.Start(s.Tag.ToString());
```

نکته ی مهمی که در این برنامه وجود دارد نحوه ی نوشتن شدن آن نیست، بلکه استفاده از کلاس هایی است که در برنامه ای دیگر ایجاد کرده بودید. بنابراین مشاهده کردید که هر بار برای ایجاد یک اتومبیل جدید نیاز ندارید که ابتدا چرخ را اختراع کنید.
مشکل این روش این است که با اضافه کردن این کلاسها به برنامه در حقیقت علاوه بر نسخه ای که در فولدر برنامه ی Favorite Viewer قرار داشت یک کپی از آنها را نیز در برنامه خود ایجاد کردیم. این روش راه مناسبی نیست، زیرا به این ترتیب دو نسخه از کلاس در دیسک وجود دارد و فضای بیشتری را اشغال می کند. البته حجم این کلاسها کوچک است و ممکن است این اشغال فضا در مقابل فوایدی که ارایه می دهند ناچیز به نظر برسد.
یک روش دیگر برای انجام این کار استفاده از کتابخانه های کلاس است. کتابخانه های کلاس، پروژه های مجزایی هستند که فقط شامل کلاسهای گوناگون می شوند و می توانند توسط چندین برنامه مورد استفاده قرار بگیرند. در فصل دوازدهم با این نوع پروژه ها بیشتر آشنا می شویم.

نتیجه:

در طی این فصل سعی کردیم با مباحث پیشرفته ی برنامه نویسی شیئ گرا آشنا شویم. در ابتدا ی فصل مشاهده کردیم که چگونه می توان چندین نسخه ی گوناگون از یک متد ایجاد کرد که هر یک، پارامترهای خاص خود را دریافت کنند و پیاده سازی خاص خود را نیز داشته باشند. همچنین مشاهده کردیم که چگونه می توان عملگر های موجود در C# را تغییر داد تا با کلاس هایی که ایجاد می کنیم نیز کار کنند، برای مثال عملگر + بتواند دو متغییر از نوع ComplexNumber را با یکدیگر جمع کند.
سپس به معرفی متد ها و خاصیت هایی پرداختیم که می توانند در بین تمام اشیایی که از کلاس ساخته می شوند مشترک باشند، و فواید استفاده از این نوع متد ها و خاصیت ها را نیز مشاهده کردیم. در ادامه ی بخش نیز به انواع ویژه ای از وراثت، یعنی کلاسهای

Sealed و کلاسهای Abstract آشنا شدیم و موارد استفاده از آنها را در برنامه بررسی کردیم. سپس بـه عنـوان آخـرین مبحث تئوری در ای فصل به معرفی و مرور اجمالی اینترفیس ها پرداختیم. اینترفیس ها کـاربرد زیـادی در برنامـه نویـسی .NET دارند و در فصل های بعد بیشتر از آنها استفاده خواهیم کرد.
در پایان فصل نیز یک برنامه ی عملی با استفاده از کلاسها ایجاد کردیم تا به مزایای برنامه نویسی شیئ گـرا و مهمتـرین آن یعنـی قابلیت استفاده مجدد از کد بیشتر پی ببریم.
در این فصل مباحث تئوری زیادی را بررسی کردیم، اما این مباحث از مهمترین مباحث برنامه نویسی شیئ گرا به شـمار مـی رونـد و درک آنها از اهمیت زیادی برخوردار است. بهتر است با بررسی کلاسهای موجود در کتابخانـه ی کـلاس .NET و یـا هـر کـلاس دیگری سعی کنید بر این مباحث مسلط شوید، زیرا این مطالب در برنامه های کاربردی نقش زیادی را ایفا می کنند.
در پایان این فصل باید با موارد زیر آشنا شده باشید:

- سربار گذاری متدهای مختلف و ایجاد چند نسخه از یک متد.
- تعریف نحوه ی عملکرد جدید برای یک عملگر و سربار گذاری آن.
- ایجاد خاصیت ها و متد های static و چگونگی استفاده از آنها در برنامه.
- مفهوم و نحوه ی کاربرد کلاسهای abstract در برنامه.
- مفهوم و نحوه ی کاربرد کلاسهای sealed در برنامه.
- چگونگی تعریف یک interface و تفاوت آن با کلاس های عادی و یا کلاسهای abstract.

فصل یازدهم: اشکال زدایی و کنترل خطا در برنامه

اشکال زدایی یکی از قسمتهای مهم هر پروژه محسوب می شود که به وسیله آن می توانید خطاهای موجود در کد برنامه و یا منطق آن را متوجه شده و رفع کنید. ویژوال استودیو ۲۰۰۵ ابزارهای پیشرفته ای برای این کار در اختیار برنامه نویسان قرار می دهد که این ابزارها می توانند به وسیله تمام زبانهایی که توسط ویژوال استودیو پشتیبانی می شوند مورد استفاده قرار گیرند. بنابراین هنگامی که نحوه ی اشکال زدایی با استفاده از این ابزارها را در یکی از زبانهای ویژوال استودیو ۲۰۰۵ یاد گرفتید، می توانید از آن در تمام زبانهای دیگر موجود در ویژوال استودیو نیز استفاده کنید.

صرفنظر از این که کد برنامه شما تا چه اندازه خوب و کارآمد نوشته شده است، همواره ممکن است برنامه با شرایط پیش بینی نشده ای مواجه شود که باعث توقف اجرای آن شود. اگر در کد برنامه این گونه شرایط را کنترل نکنید، هنگامی که برنامه در حالت اجرا با این شرایط مواجه شود پیغام خطای پیش فرض CLR را نمایش می دهد . این پیغام خطا به کاربر اعلام می کند که یک خطای کنترل نشده در برنامه رخ داده است و شامل اطلاعاتی فنی در مورد خطای اتفاق افتاده است که کاربر با توجه به آن نمی تواند متوجه شود علت رخداد خطا چه بوده و چگونه می تواند آن را تصحیح کند.

در این مواقع است که اهمیت کنترل خطا در برنامه مشخص می شود. ویژوال استودیو ۲۰۰۵ دارای توابع و ساختارهایی عمومی برای کنترل خطا در برنامه است که بین تمام زبانهای آن مشترک است. به وسیله این توابع و ساختارها می توانید یک قسمت از کد برنامه را بررسی کرده و هر خطایی که ممکن است در آن قسمت رخ دهد را مشخص کنید. سپس کدی بنویسید که اگر خطایی در آن قسمت از کد اتفاق افتاد، کادر پیغامی به کاربر نمایش دهد و رخ دادن آن خطا و نحوه تصحیح آن را اعلام کند. همچنین می توانید مشخص کنید که برنامه بدون توجه به خطای اتفاق افتاده به اجرای خود ادامه دهد.

در این فصل بعضی از ویژگیهای اشکال زدایی موجود در ویژوال استودیو ۲۰۰۵ را بررسی کرده و نحوه اشکال زدایی از یک برنامه نمونه را مشاهده خواهیم کرد. همچنین با نحوه استفاده از Breakpointها در برنامه برای توقف اجرای برنامه در خط مشخصی از کد و بررسی وضعیت برنامه در آن خط آشنا خواهیم شد. به وسیله این امکان و امکانات نظیر آن می توانید متوجه شوید که در هر لحظه برنامه چه کاری انجام می دهد.

در این فصل:

- انواع خطاهایی که ممکن است در یک برنامه ایجاد شوند و نحوه تصحیح هر کدام از آنها را فرا می گیرید.
- چگونگی یافتن خطاهای موجود در یک برنامه و تصحیح آنها را بررسی می کنید.
- نحوه کنترل خطاها و شرایط پیش بینی نشده در یک برنامه را مشاهده خواهید کرد.

انواع مختلف خطاها:

خطاهایی که در یک برنامه رخ می دهند به سه دسته کلی تقسیم می شوند: خطاهای دستوری، خطاهای زمان اجرا و خطاهای منطقی. در این بخش به بررسی این سه نوع خطا می پردازیم و تفاوتهای آنها را بیان می کنیم.

خطاهای دستوری:

خطاهای دستوری[1] ساده ترین نوع خطاها از نظر پیدا کردن و رفع کردن هستند. این گونه خطاها معمولاً زمانی رخ می دهند که کد نوشته شده به وسیله شما از نظر کامپایلر خطا داشته باشد و کامپایلر نتواند آن را تفسیر کند. ممکن است دستوری را ناقص وارد کرده باشید، ترتیب نوشتن دستورات را رعایت نکرده باشید و یا حتی عمومی تر از همه آنها اینکه در وارد کردن دستورات خطای تایپی داشته باشید. برای مثال فرض کنید که متغیری را در برنامه تعریف می کنید و هنگامی که می خواهید از آن استفاده کنید، نام آن را اشتباه وارد می کنید.

محیط طراحی و توسعه ویژوال استودیو .NET ۲۰۰۵ دارای ابزاری قوی برای بررسی درستی دستورات وارد شده توسط برنامه نویس است . به وسیله این ابزار احتمال ایجاد خطای دستوری به شدت کاهش پیدا می کند، اما باز هم این احتمال به صفر نمی رسد و ممکن است چنین خطاهایی در برنامه ایجاد شوند.

برای مثال تصور کنید که متغیری را درون یک زیر برنامه از نوع private تعریف کرده اید. بلافاصله بعد از نوشتن چنین کدی، ویژوال استودیو زیر آن خط قرمزی قرار می دهد که مشخص می کند این نوع تعریف از نظر کامپایلر نادرست است. حال اگر با ماوس روی عبارت مشخص شده بروید، کادر کوچکی نمایش داده می شود و علت نادرست بودن آن را بیان می کند. به این ترتیب می توانید آن را اصلاح کنید (شکل ۱۱-۱).

```
18        private void button1_Click(object sender, EventArgs e)
19        {
20            private string strFileName;
21        }   Invalid expression term 'private'
```

شکل ۱۱-۱

روش دیگری برای مشاهده تمام خطاهای دستوری موجود در برنامه استفاده از پنجره Error List است. در این پنجره جدولی نمایش داده می شود که در آن خطاهای موجود در برنامه لیست شده اند. در مقابل هر خطا نیز توضیحی درباره ی آن خطا، نام فایل حاوی خطا، شماره سطر و ستون خطا و همچنین نام پروژه ای که خطا در آن رخ داده، آورده شده است.

پنجره Error List در پایین محیط طراحی ویژوال استودیو قرار دارد. هنگامی که این پنجره نمایش داده شد، با دو بار کلیک کردن روی هر خطا مکان نما به خط شامل خطا منتقل می شود و می توانید خطا را بررسی کرده و تصحیح کنید.

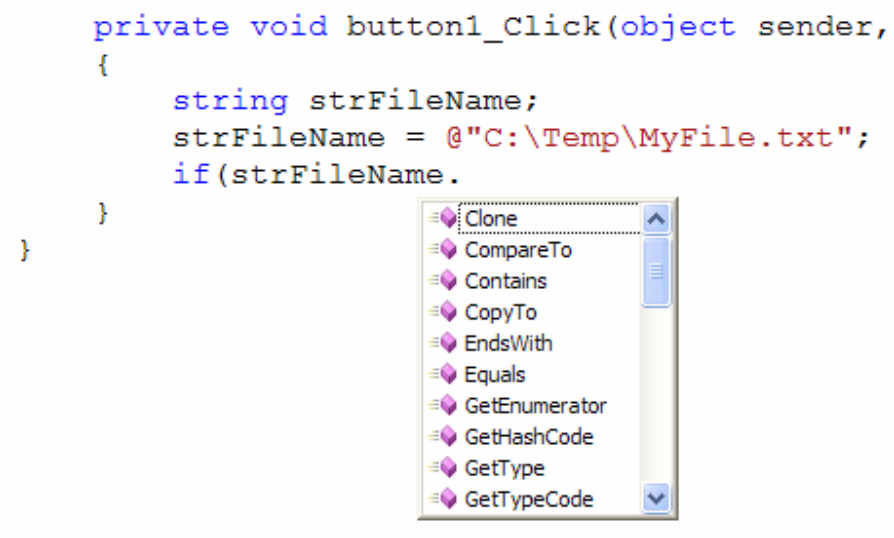
بعضی مواقع زیر قسمتهایی از کد خط سبز کشیده می شود. این خطها مشخص کننده هشدارهایی در کد هستند و از کامپایل شدن کد جلوگیری نمی کنند. البته بهتر است تا حد ممکن قبل از اجرای برنامه این هشدارها را نیز تصحیح کنید، زیرا ممکن است در هنگام اجرای برنامه موجب به وجود آمدن شرایط نا مطلوبی شوند.

به عنوان مثال فرض کنید که در یک زیر برنامه متغیری را تعریف می کنید، اما تا پایان آن زیر برنامه از متغیر تعریف شده استفاده ای نمی کنید. در این شرایط ویژوال استودیو زیر این متغیر خط سبزی قرار می دهد تا هشداری را در این قسمت مشخص کند. برای تصحیح این هشدار اگر به این متغیر در زیر برنامه نیاز دارید از آن استفاده کنید، در غیر این صورت خط مربوط به تعریف آن را پاک کنید.

یکی از ویژگیهای دیگر ویژوال استودیو که باعث می شود خطاهای دستوری در یک برنامه به حداقل برسند، ویژگی تکمیل خودکار متن یا IntelliSense است. به وسیله این ویژگی هنگامی که بخواهید کدی را بنویسید کادری باز می شود و بر اساس چندین فاکتور مختلف از قبیل حروفی که وارد کرده اید، دستورات قبلی و یک سری اطلاعات دیگر، کلمه ای را برای قرار دادن در آن مکان پیشنهاد می کند (شکل ۱۱-۲). به این وسیله می توانید به سرعت و بدون اینکه چیزی را به خاطر بسپارید، نام اعضای کلاسها، نام ساختارها و یا فضای نامهایی که با آنها کار می کنید را در برنامه وارد کنید.

---

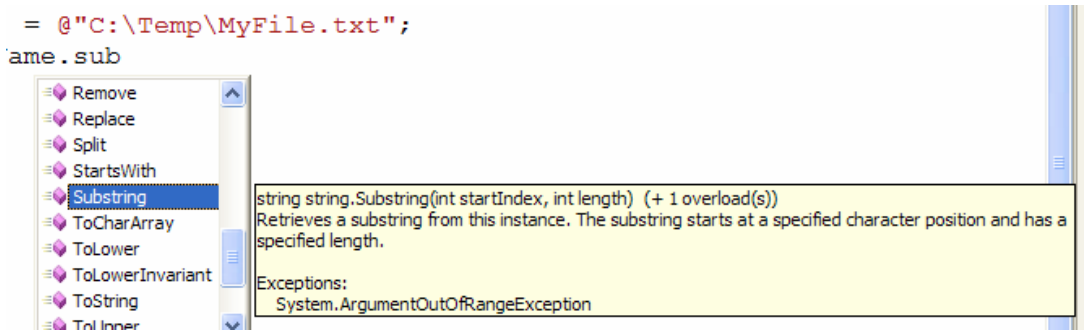
[1] Syntax Errors



شکل ۱۱-۲

همچنین اگر در این لیست برای مدت کوتاهی روی یک گزینه صبر کنید، کادر کوچکی که حاوی متن راهنمایی در مورد گزینه ی انتخاب شده است، نمایش داده می شود. متن راهنمایی که در این کادر نمایش داده می شود اطلاعات مفیدی را برای هر متد و یا عضو کلاس نمایش می دهد. برای مثال در این پنجره نام و نوع تمام پارامترهایی که برای فراخوانی یک متد لازم است نمایش داده می شود. به این وسیله نیازی نیست که پارامترهای یک متد را به خاطر بسپارید و احتمال ایجاد خطا را نیز در فراخوانی آن کاهش می دهد. همچنین در این کادر تعداد نسخه هایی که از یک متد موجود است نیز نوشته می شود. به عبارت دیگر در این کادر گفته می شود که این تابع به چند صورت سربار گذاری شده است. علاوه بر این موارد، خطاهایی که ممکن است هنگام اجرای این تابع رخ دهد نیز در پایین این کادر نوشته می شود.

به عنوان مثال کادر راهنمایی که در شکل ۱۱-۳ نمایش داده شده است مشخص می کند که ورودی متد `SubString` مقداری از نوع عدد صحیح بوده و خروجی آن یک متغییر رشته ای است. عبارت `"(+1 Overloads)"` مشخص می کند که دو نسخه ی متفاوت از این متد وجود دارد. همچنین در پایین این کادر مشخص می شود که این تابع ممکن است خطایی را از نوع `System.ArgumentOutOfRangeException` ایجاد کند که باید هنگام نوشتن برنامه در نظر گرفت.



شکل ۱۱-۳

علاوه بر این مورد، هنگامی که نام یک متد را در برنامه بنویسید و بخواهید پارامترهای آن را وارد کنید، کادر راهنمای دیگری نمایش داده می شود و اطلاعات مربوط به پارامترهای یکی از نسخه های متد را نمایش می دهد. اگر از این متد بیش از یک نسخه وجود داشته باشد، با فشار دادن کلیدهای مکان نما می توانید اطلاعات مربوط به نسخه های دیگر را نیز مشاهده کنید. همچنین در این کادر نام و نوع پارامترهای تابع و توضیح هر کدام از پارامتر ها نیز آورده شده است (شکل ۱۱-۴).

```
private void button1_Click(object sender, EventArgs e)
{
    string strFileName;
    strFileName = @"C:\Temp\MyFile.txt";
    if(strFileName.Substring(
}
        ▲1 of 2▼ string string.Substring (int startIndex)
        startIndex: The starting character position of a substring in this instance.
```

شکل ۱۱-۴

علاوه بر این موارد، ویژگیهای بسیار دیگری در محیط توسعه ویژوال استودیو وجود دارد که باعث کمتر شدن خطاهای دستوری می شود. تنها کاری که باید انجام دهید، این است که با شناخت این ویژگیها و استفاده صحیح از آنها از به وجود آمدن خطاهای دستوری در برنامه جلوگیری کنید.

خطاهای اجرایی:

خطاهای اجرایی[1] و یا **خطاهای زمان اجرا**[2] خطاهایی هستند که در زمان اجرای یک برنامه رخ می دهند. این خطاها عموماً به این علت رخ می دهند که بعضی از عوامل خارج از برنامه مانند کاربر، بانک اطلاعاتی، دیسک سخت موجود در کامپیوتر و یا ... رفتاری غیر قابل پیش بینی از خود بروز می دهند.

در هنگام نوشتن یک برنامه همواره این نوع مسائل را نیز باید مدنظر قرار داد و کد مناسبی برای کنترل رخ دادن این خطاها نوشت. البته نمی توان از رخ دادن چنین خطاهایی در برنامه جلوگیری کرد، اما می توان با نوشتن کد های مناسب برای کنترل آنها، هنگام بروز چنین خطاهایی یا با نمایش پیغام مناسب از برنامه خارج شد و یا بدون در نظر گرفتن خطا از اجرای بقیه کد صرفنظر کرد و به کاربر اطلاع داد که چگونه از بروز مجدد این خطا جلوگیری کند. نحوه کنترل خطاهای زمان اجرا در قسمتهای بعدی این فصل به تفصیل شرح داده شده است.

خطاهای اجرایی معمولاً هنگام تست قسمتهای مختلف در زمان نوشتن برنامه مشخص می شوند. به این ترتیب بعد از تشخیص آنها می توانید کدی بنویسید که رفتار برنامه را در آن شرایط کنترل کند و از توقف ناگهانی برنامه جلوگیری کنید. نحوه این کار در بخش اشکال زدایی در ادامه ی فصل کاملاً توضیح داده شده است.

خطاهای منطقی:

[1] Execution Errors
[2] Run-Time Errors

خطاهای منطقی[1] یا خطاهای مفهومی، خطاهایی هستند که باعث می شوند برنامه نتایج نا مطلوبی را تولید کند. احتمالاً بیشترین نوع خطاهای منطقی که در یک برنامه به وجود می آیند، حلقه های بی نهایت هستند. برای مثال کد زیر را در نظر بگیرید:

```
private void PerformLoopExample()
{
    int intIndex;
    while (intIndex < 10)
    {
        // Some logic here
    }
}
```

اگر در کد داخل این حلقه مقدار `intIndex` به نحوی تغییر نکند که به عددی بزرگتر از ۱۰ برسد، برنامه در یک حلقه بینهایت قرار می گیرد. این یک مثال ساده از خطاهای منطقی در برنامه بود، اما حتی برنامه نویسان با تجربه نیز ممکن است در هنگام نوشتن کد دچار چنین خطاهایی در برنامه خود شوند.
پیدا کردن خطاهای منطقی و رفع آنها در یک برنامه معمولاً از دیگر خطاها مشکل تر است، زیرا برای تشخیص آنها باید عملکرد تمام قسمتهای برنامه را در مقابل مقادیر مختلف ورودی بررسی کرده و مشاهده کرد که آیا نتیجه مطلوب توسط برنامه تولید می شود یا نه؟
خطای منطقی دیگری که ممکن است در برنامه ایجاد شود، خطا در مقایسه ها است. برای مثال فرض کنید می خواهید مقدار یک متغیر را با ورودی کاربر مقایسه کرده و در صورت برابر بودن این دو مقدار، عمل خاصی را در برنامه انجام دهید. در این شرایط معمولاً نمی خواهید که مقایسه نسبت به بزرگی و کوچکی حروف حساس باشد. فرض کنید برای این مقایسه از کد زیر استفاده کنید:

```
    if (strFileName == txtInput.Text)
    {
        // Perform some logic here
    }
```

در این شرایط برای مثال اگر مقدار متغییر `strFileName` برابر با `Index.Html` و مقدار موجود در `TextBox` برابر با `index.html` باشد، نتیجه مقایسه نادرست خواهد بود و کد داخل دستور `if` اجرا نخواهد شد. یکی از روشهای جلوگیری از این خطاها در این است که ابتدا، هر دو مقداری که می خواهید با هم مقایسه کنید را به حروف بزرگ و یا حروف کوچک تبدیل کنید (برای این کار می توانید از توابع `ToUpper` و یا `ToLower` در کلاس `String` استفاده کنید). به این ترتیب اگر متنی که کاربر در `TextBox` وارد کرده است با متن موجود در متغییر `strFileName` برابر باشد و فقط از نظر بزرگی و یا کوچکی کاراکتر ها با هم تفاوت داشته باشند حاصل مقایسه درست خواهد بود.

نکته: به علت اینکه خطاهای منطقی در یک برنامه سخت ترین نوع خطاها از نظر تشخیص و رفع هستند و همچنین ممکن است باعث توقف اجرای برنامه و یا تولید نتیجه نامطلوب توسط آن شوند باید هنگام نوشتن کد از درست بودن منطق آن اطمینان حاصل کنید و مطمئن شوید که روشی که برای اجرای این قسمت از برنامه به کار می برید درست است. همچنین باید تمام خطاهایی که ممکن است به وسیله کاربر در برنامه ایجاد شود را نیز بررسی کرده و کنترل کنید. هر چه که بیشتر در برنامه نویسی تجربه کسب کنید، بیشتر با خطاهای عمومی و خطاهایی که ممکن است توسط کاربر ایجاد شوند آشنا خواهید شد.

---

[1] Logic Errors

یکی از بهترین راه ها برای مشخص کردن و از بین بردن خطاهای منطقی در یک برنامه استفاده از امکانات اشکال زدایی برنامه است که در ویژوال استودیو ۲۰۰۵ قرار دارد. با استفاده از این امکانات و ویژگیها می توانید به راحتی حلقه های بی نهایت را تشخیص داده و یا به تصحیح مقایسه های نادرست در برنامه بپردازید.

اشکال زدایی:

اشکال زدایی از یک برنامه، همواره بخشی جدا نشدنی از مراحل نوشتن یک نرم افزار است، زیرا حتی برنامه نویسان با تجربه نیز ممکن است در برنامه های خود خطا هایی داشته باشند. دانستن این که چگونه می توان به راحتی اشکالات یک برنامه را مشخص کرده و آنها را رفع کرد باعث می شود که بتوانید به سرعت و با حداقل خطا برنامه های مورد نظرتان را پیاده سازی کنید.
در بخش بعد یک برنامه ی نمونه ایجاد خواهیم کرد و به اشکال زدایی آن خواهیم پرداخت. در طول کار بر روی این برنامه سعی خواهیم کرد که با `breakpoint`ها، چگونگی اجرای خط به خط کد، بررسی موقعیت برنامه بعد از اجرای هر خط و... آشنا شویم.

ایجاد یک برنامه نمونه:

در تمرین های مختلف بخش "امتحان کنید" این فصل ، برنامه ی ساده ای خواهیم نوشت و سعی خواهیم کرد در طی این برنامه، نگاهی به ویژگیهای عمومی و پر کاربرد ویژوال استودیو ۲۰۰۵ برای اشکال زدایی برنامه ها داشته باشیم. در این برنامه آدرس یک فایل متنی را به وسیله یک کنترل `TextBox` از کاربر دریافت می کنیم و محتویات آن را نمایش می دهیم و یا آدرسی را دریافت کرده و محتویات داخل `TextBox` را در فایلی در آن آدرس ذخیره می کنیم. البته همانطور که می دانید و در فصول قبلی نیز مشاهده کردید بهترین راه برای این کار استفاده از کنترل های `OpenFileDialog` و `SaveFileDialog` است، اما در این قسمت برای اینکه بتوانیم بهتر روی خطا یابی در برنامه تمرکز کنیم، آدرس را به صورت مستقیم از کاربر دریافت می کنیم.

امتحان کنید: ایجاد یک برنامه ی نمونه برای اشکال زدایی

۱) با استفاده از ویژوال استودیو یک برنامه ویندوزی جدید به نام `ErrorHandling` ایجاد کنید.
۲) روی فرم برنامه کلیک کنید تا انتخاب شود، سپس با استفاده از پنجره `Properties` خاصیتهای آن را برابر با مقادیر زیر قرار دهید:

- خاصیت `Size` را برابر `445;315` قرار دهید.
- خاصیت `StartPosition` را برابر با `CenterScreen` قرار دهید.
- خاصیت `Text` را برابر با `Error Handling` قرار دهید.

۳) حال باید تعدادی کنترل روی فرم قرار داده و خاصیتهای آنها را تنظیم کنید. ابتدا یک کنترل `TextBox` ایجاد کرده و خاصیتهای آن را طبق لیست زیر تنظیم کنید.

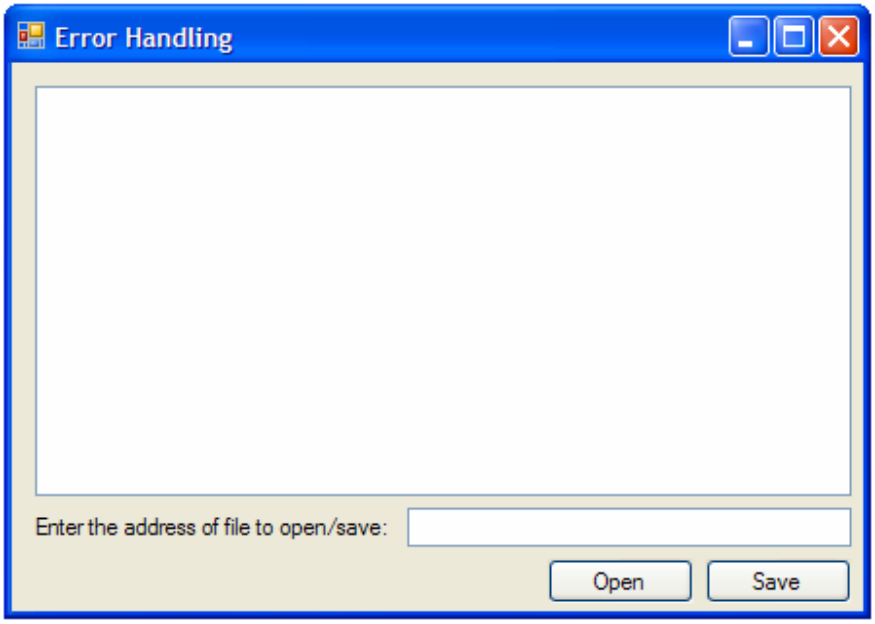
- خاصیت `Multiline` آن را برابر با `true` قرار دهید.
- خاصیت `Size` را برابر با `419;210` قرار دهید.
- خاصیت `Name` را برابر با `txtBody` قرار دهید.

یک کنترل `TextBox` دیگر در فرم قرار داده و خاصیتهای آن را مطابق با لیست زیر تغییر دهید:

- خاصیت `Name` را برابر با `txtAddress` قرار دهید.
- خاصیت `Location` را برابر با `13;43` قرار دهید.
- خاصیت `Size` را برابر با `448;254` قرار دهید.

یک کنترل `Button` بر روی فرم قرار داده، خاصیت `Name` آن را برابر با `btnOpen` و خاصیت `Text` آن را برابر با `Open` قرار دهید. کنترل `Button` دیگری با نام `btnSave` بر روی فرم قرار دهید و خاصیت `Text` آن را با `Save` تنظیم کنید. در انتها نیز یک کنترل `Label` که حاوی متن `"Enter the address of file to open/save:"` باشد را به گونه ای در فرم قرار دهید که پنجره ی برنامه ی شما مشابه شکل ۱۱-۵ شود.

۴) بهتر است قبل از اینکه طراحی قسمتهای مختلف برنامه را شروع کنیم ، با یکی از مهمترین (و البته ساده ترین) ابزارهای خطا یابی در ویژوال استودیو یعنی پنجره ی `Error List` آشنا شویم. برای نمایش این پنجره، از نوار منو گزینه ی `View` ➔ `Error List` را انتخاب کنید.

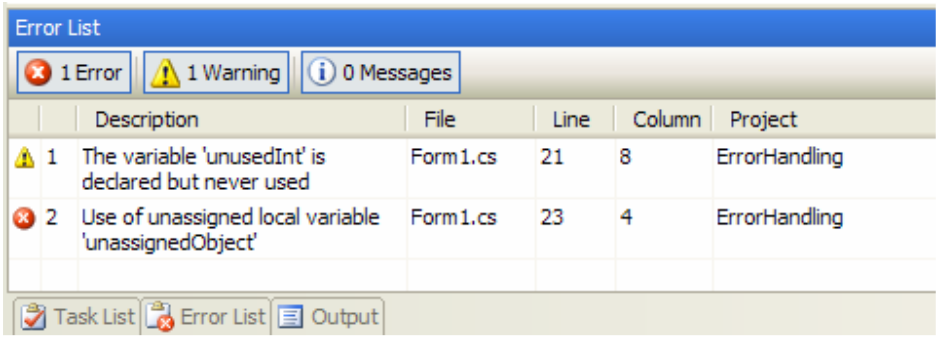


شکل ۱۱-۵

۵) در قسمت طراحی فرم، روی دکمه ی Open دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس کد زیر را در این متد وارد کنید:

```
private void btnOpen_Click(object sender, EventArgs e)
{
      int unusedInt;
      DateTime unassignedObject;
      unassignedObject.AddDays(1);
}
```

۶) همانطور که می دانید در این کد یک خطا وجود دارد و این است که بدون اینکه با استفاده از دستور new به متغییر unassignedObject مقدار اولیه دهیم، از آن در برنامه استفاده کرده ایم. خوب، با فشار دادن کلید F5 سعی کنید که برنامه را اجرا کنید[1]. مشاهده می کنید که در زیر متغییر unsedInt خطی سبز رنگ و در زیر متغییر unassignedObject خطی آبی رنگ کشیده می شود. همچنین پنجره ی Error List نیز گزینه هایی را همانند شکل ۱۱-۶ نمایش می دهد.



شکل۱۱-۶

۷) همانطور که در شکل مشاهده می کنید در این پنجره عنوان شده است که در هنگام کامپایل برنامه، کامپایلر با یک خطا و یک هشدار موجه شده است. هشدار که در گزینه ی اول و با یک علامت زرد رنگ مشخص شده است مربوط به خط 21 از فایل Form1.cs در پروژه ی ErrorHandling است. این هشدار همانطور که در قسمت Description آمده است به علت تعریف یک متغییر در برنامه و استفاده نکردن از آن است. ویژوال استودیو هشدار ها را در کد با زیر خط سبز رنگ نمایش می دهد.

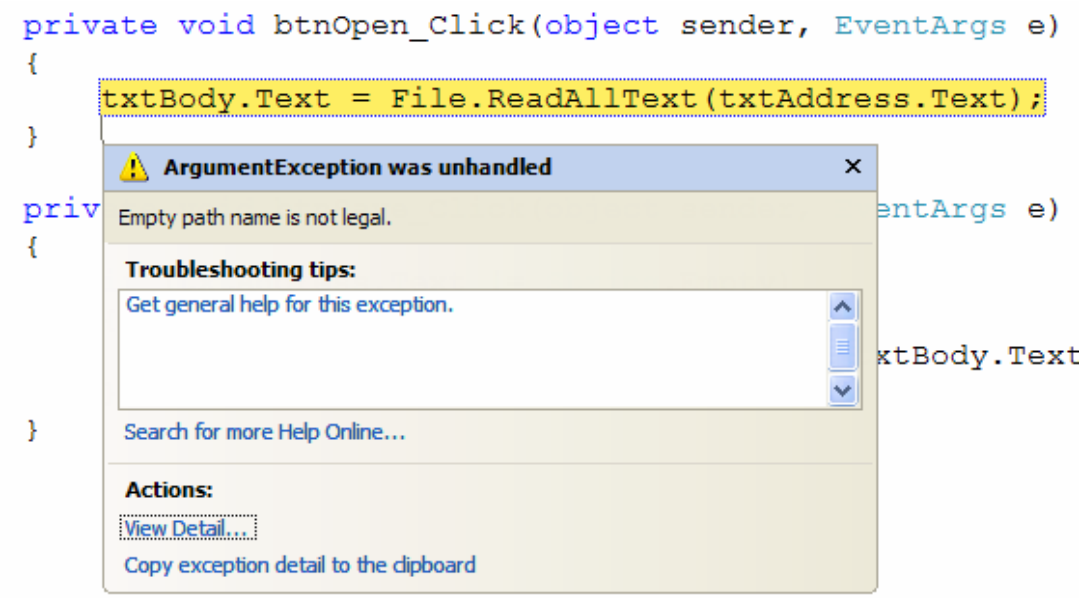
۸) گزینه ی دوم در این پنجره نیز که با علامت قرمز مشخص شده است، مربوط به یک خطا در خط ۲۳ از فایل Form1.cs در پروژه ی ErrorHandling است. علت به وجود آمدن این خطا نیز همانطور که در بخش Description توضیح داده شده است، استفاده از یک متغییر بدون مقدار دهی اولیه به آن است. خطا ها در ویژوال استودیو با زیر خط آبی رنگ در کد مشخص می شوند.

۹) حال کد درون متد btnOpen_Click را به صورت مشخص شده در زیر تغییر دهید.

[1] کلید ها و یا ترکیبات کلیدی که در ویژوال استودیو به کار می روند، بسته به تنظیم های کاربر ممکن است تفاوت داشته باشند. بنابراین ممکن است با فشار کلید F5 برنامه اجرا نشود. برای اطلاع از کلید مربوط به شروع اجرای برنامه، عبارت نوشته شده در مقابل گزینه ی Start Debugging در منوی Debug را مشاهده کنید.

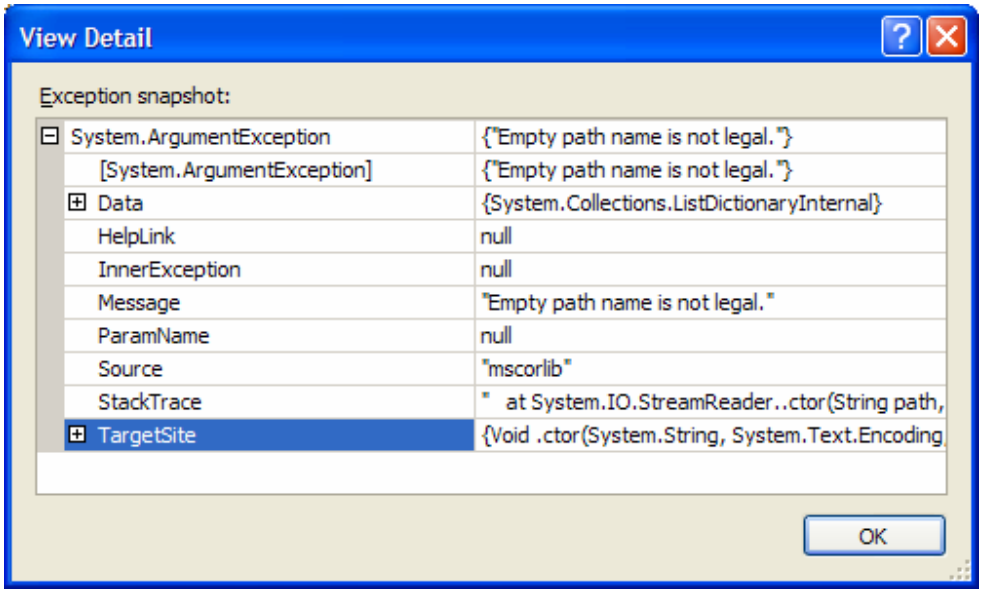
```
private void btnOpen_Click(object sender, EventArgs e)
{
    txtBody.Text = File.ReadAllText(txtAddress.Text);
}
```

۱۰) برنامه را اجرا کنید و بدون اینکه در کنترل txtAddress آدرس فایل را وارد کنید روی دکمه ی Open کلیک کنید. در این حالت برنامه با یک خطای زمان اجرا مواجه می شود و کادری را مشابه شکل ۱۱-۷ نمایش می دهد.
همانطور که مشاهده می کنید، نوع خطایی که برنامه با آن مواجه شده است در نوار عنوان این کادر نمایش داده شده است. در این قسمت برنامه با خطایی از نوع ArgumentException مواجه شده است. در پایان این فصل با این نوع خطاها و مفهوم آنها بیشتر آشنا خواهیم شد. همچنین در قسمت Troubleshooting tips این کادر، لینک هایی در مورد چگونگی رفع این خطا آورده شده است. در پایین کادر نیز قسمتی به نام View Details... وجود دارد که به وسیله ی آن می توانید از جزئیات خطای به وجود آمده مطلع شوید.



شکل ۱۱-۷

۱۱) در کادر نمایش داده شده، روی گزینه ی View Details... کلیک کنید. پنجره ی View Details مشابه شکل ۱۱-۸ نمایش داده می شود. با استفاده از این پنجره می توانید به اطلاعات دیگری در مورد خطای به وجود آمده از قبیل پیغام آن خطا، لینکی برای دسترسی به اطلاعات بیشتر در مورد خطا، متدی که باعث به وجود آمدن خطا شده است و ... دسترسی داشته باشید.



شکل ۱۱-۸

۱۲) در این پنجره روی دکمه ی OK کلیک کنید تا به برنامه برگردید و سپس با استفاده از نوار منو، گزینه ی Debug ➔ Stop Debugging را انتخاب کنید تا اجرای برنامه متوقف شود.

۱۳) حال کد متد btnOpen_Click را به صورت زیر تغییر دهید:

```
private void btnOpen_Click(object sender, EventArgs e)
{
     if (txtAddress.Text != String.Empty)
     {
          txtBody.Text =
               File.ReadAllText(txtAddress.Text);
     }
}
```

۱۴) به قسمت طراحی فرم برگردید و روی دکمه ی Save دو بار کلیک کنید تا متد مربوط به رویداد کلیک این کنترل نیز ایجاد شود. سپس کد زیر را در این متد وارد کنید:

```
private void btnSave_Click(object sender, EventArgs e)
{
     if (txtAddress.Text != String.Empty)
     {
          File.WriteAllText(txtAddress.Text,
                                    txtBody.Text);
     }
}
```

۱۵) برنامه را اجرا کرده و بعد از وارد کردن آدرس یک فایل متنی در کنترل `txtAddress`، روی دکمه ی `Open` کلیک کنید. مشاهده می کنید که محتویات فایل در برنامه نمایش داده می شوند.

مشاهده کردید که چگونه در این قسمت از ایجاد خطا جلوگیری کردیم، اما همانطور که می دانید این روش نمی تواند به صورت کلی مورد استفاده قرار بگیرد. زیرا برای مثال اگر کاربر آدرس فایل را به صورت نادرست وارد کند، برنامه مجدداً با خطا مواجه خواهد شد. روش صحیح کنترل این نوع خطاها در برنامه استفاده از دستورات `try` و `catch` است که در بخشهای بعدی این فصل توضیح داده شده است. اما قبل از این که با دستورات کنترل خطا در کد برنامه آشنا شویم، بهتر است نحوه کار ابزارهای دیگر اشکال زدایی در ویژوال استودیو را نیز بررسی کنیم.

کنترل اجرای برنامه با استفاده از Breakpointها:

هنگامی که در حال طراحی یک برنامه ی بزرگ باشید، ممکن است برای فهمیدن اشکالات قسمتی از کد نیاز داشته باشید که برنامه تا آن قسمت اجرا شود و سپس در خطی خاص متوقف شده تا وضعیت آن را بررسی کنید. در این شرایط می توانید از `breakpoint`ها استفاده کنید. هنگامی که در یک خط از برنامه یک `breakpoint` قرار می دهید، برنامه تا آن خط اجرا می شود اما خط شامل `breakpoint` را اجرا نکرده و قبل از آن متوقف می شود.
برای تعیین `breakpoint` در یک خط، هم در زمان اجرای برنامه و هم در زمان نوشتن کد آن می توانید اقدام کنید. برای این کار کافی است در ناحیه خاکستری رنگ سمت چپ خط کلیک کنید. به این ترتیب در آن ناحیه یک دایره ی قرمز رنگ قرار می گیرد. برای حذف یک `breakpoint` نیز کافی است مجدداً بر روی دایره ی قرمز رنگ کنار خط کلیک کنید. به این ترتیب `breakpoint` به وجود آمده در آن خط حذف خواهد شد.
هنگامی که کامپایلر در اجرای یک برنامه به خطی شامل یک `breakpoint` می رسد، اجرای برنامه را متوقف می کند و به قسمت کد نویسی برنامه برمیگردد. البته دقت کنید که اجرای برنامه کاملاً متوقف نمی شود، بلکه وقفه ای در آن ایجاد می شود تا بتوانید وضعیت برنامه را در آن قسمت بررسی کنید، سپس می توانید اجرای برنامه را ادامه دهید. به این حالت در اصطلاح، حالت `break` گفته می شود.
در بخشهای "امتحان کنید" بعدی، با `breakpoint`ها و نیز امکانات و ویژگیهایی که برای اشکال زدایی از کد در حالت `break` وجود دارند بیشتر آشنا می شویم. اما قبل از ادامه برای سادگی کار بهتر است نوار ابزار `Debug` را به محیط ویژوال استودیو اضافه کنید. برای این کار با استفاده از نوار منو، گزینه ی `Debug` ➔ `Toolbars` ➔ `View` را انتخاب کنید.

امتحان کنید: کار با `breakpoint`ها

۱) برنامه ی `ErrorHandling` که در قسمت قبل ایجاد کردیم را مجدداً باز کرده و به قسمت طراحی فرم بروید. سپس با استفاده از جعبه ابزار کنترل `Button` جدیدی بر روی فرم قرار داده، خاصیت `Name` آن را برابر با `btnTest` و خاصیت `Text` آن را برابر با `Test` قرار دهید.
۲) بر روی این کنترل دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس کد زیر را در این متد وارد کنید.

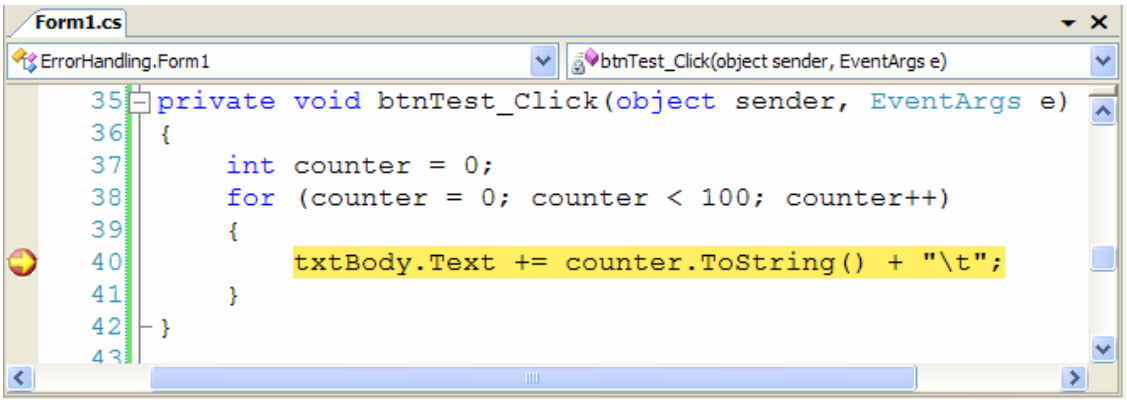
```
private void btnTest_Click(object sender, EventArgs e)
```

```
{
      int counter = 0;
      for (counter = 0; counter < 100; counter++)
      {
            txtBody.Text += counter.ToString() + "\t";
      }
}
```

۳) در نوار خاکستری سمت چپ خطی که شامل دستور داخل حلقه `for` است، کلیک کنید تا دایره ای قرمز رنگ در آن نمایش داده شود. به این ترتیب برای آن خط از برنامه یک `breakpoint` ایجاد کرده اید. حال برنامه را اجرا کنید.

۴) هنگامی که فرم برنامه نمایش داده شد، بر روی دکمه ی `Test` کلیک کنید. مشاهده می کنید کامپایلر خط اول از متد `btnTest_Click` را اجرا می کند و هنگامی که به ابتدای دستور `for` می رسد، پنجره ی مربوط به محیط کد نویسی ویژوال استودیو را نمایش می دهد. در این هنگام دستور داخل حلقه ی `for` با رنگ زرد مشخص می شود و همچنین در داخل دایره قرمز رنگ کنار دستور نیز یک فلش زرد رنگ قرار خواهد گرفت (شکل ۱۱-۹)

همچنین اگر به پایین محیط ویژوال استودیو نگاه کنید، مشاهده خواهید کرد که پنجره های بیشتری نسبت به حالت عادی در حال نمایش هستند و با انتخاب هر یک از این پنجره ها اطلاعات خاصی در مورد آن لحظه از اجرای برنامه نمایش داده می شود (البته ممکن است تعداد این پنجره ها در کامپیوتر شما نسبت به شکل بیشتر و یا کمتر باشد).



شکل ۱۱-۹

بهتر است قبل از اینکه به ادامه ی این بخش از "امتحان کنید" بپردازیم، با بعضی از گزینه های موجود در نوار ابزار `Debug` آشنا بشویم تا بتوانیم از آنها در اشکال زدایی برنامه استفاده کنیم.

گزینه های پر کاربرد در نوار ابزار Debug:

در نوار ابزار `Debug` سه آیکون وجود دارند که معمولا کاربرد زیادی در اشکال زدایی برنامه ها دارند (شکل ۱۱-۱۰). با استفاده از این گزینه ها می توانید هنگامی که برنامه در حالت `break` قرار گرفت، آن را خط به خط اجرا کنید و روند تغییر متغییر ها و اشیای موجود در برنامه را بررسی کنید.

- آیکون اول مربوط به گزینه ی **Step Into** است. هنگامی که روی این آیکون کلیک کنید، کامپایلر یک خط از برنامه را اجرا کرده و مجدداً توقف می کند و به حالت break برمی گردد. اگر خطی که کامپایلر آن را اجرا کرده شامل فراخوانی متدی باشد، کامپایلر به اولین دستور درون این متد می رود و در آنجا متوقف می شود.
- آیکون دوم مربوط به گزینه ی **Step Over** است. با کلیک بر روی این آیکون، همانند آیکون Step Into کامپایلر یک خط از برنامه را اجرا می کند و در ابتدای خط بعد توقف می کند. تفاوت این گزینه با گزینه Step Into در این است که اگر خطی که باید اجرا شود شامل فراخوانی متدی باشد، کامپایلر آن متد را کاملاً اجرا می کند و در ابتدای خط بعد در متد جاری متوقف می شود.
- آخرین آیکون نیز مربوط به **Step Out** است. با استفاده از این آیکون برنامه تا انتهای متد جاری اجرا می شود و در اولین خط، بعد از خطی که این متد را فراخوانی کرده بود متوقف می شود. عموماً هر متد، حتی متدهای مربوط به رویدادها نیز توسط متد دیگری فراخوانی می شوند. بنابراین هنگامی که از این گزینه استفاده کنیم، برنامه اجرای متد جاری را به پایان می رساند و به متدی که این متد را فراخوانی کرده بود برمی گردد. سپس در اولین خط بعد از فراخوانی این متد متوقف می شود.



شکل ۱۱-۱۰

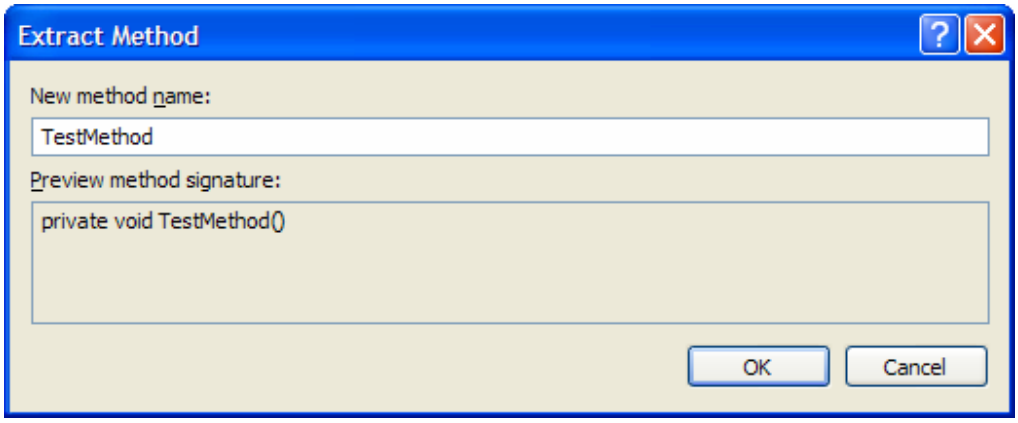
یکی دیگری از ویژگی های مهمی که بهتر است قبل از ادامه ی فصل با آن آشنا شویم، خاصیت Run To Cursor است. هنگامی که برنامه به یک breakpoint برخورد می کند و متوقف می شود، اگر به نوار خاکستری رنگ سمت چپ آن نگاه کنید یک فلش زرد رنگ مشاهده خواهید کرد که به کدی که در حال اجرا بوده است اشاره می کند. اگر با ماوس روی این فلش کلیک کنید می توانید مکان آن را تغییر دهید (برای مثال کمی بالاتر و یا کمی پایین تر ببرید) تا به کد دیگری اشاره کند. حال اگر بر روی دکمه ی Start در نوار منو کلیک کنید تا اجرای برنامه ادامه پیدا کند، اجرای برنامه از خطی که فلش در حال اشاره کردن به آن است ادامه پیدا خواهد کرد.

حال به ادامه ی بخش "امتحان کنید" قبل برمی گردیم.

امتحان کنید: کار با breakpointها (ادامه)

۱) اگر برنامه ی ErrorHandling هنوز در حال اجرا است آن را ببندید. همچنین با کلیک کردن روی تمام breakpoint هایی که در برنامه وجود دارند، آنها را حذف کنید.

۲) کد نوشته شده در متد btnTest_Click را انتخاب کرده و سپس از نوار منو گزینه ی Refactor ➔ Extract Method... را انتخاب کنید تا پنجره ی Extract Method همانند شکل ۱۱-۱۱ نمایش داده شود.



شکل ۱۱-۱۱

۳) در قسمت `New method name` همانند شکل، نام `TestMethod` را وارد کرده و روی دکمه ی `OK` کلیک کنید. به این ترتیب محتویات متد `btnTest_Click` در متد جدیدی به نام `TestMethod` قرار خواهند گرفت. کد درون متد `btnTest_Click` هم به کدی برای فراخوانی متد `TestMethod` تغییر خواهد کرد.

۴) حال در نوار خاکستری رنگ کنار دستور داخلی حلقه ی `for` (دستوری که متن داخل کنترل `txtBody` را تغییر می دهد) کلیک کنید تا یک `breakpoint` برای این خط ایجاد شود.

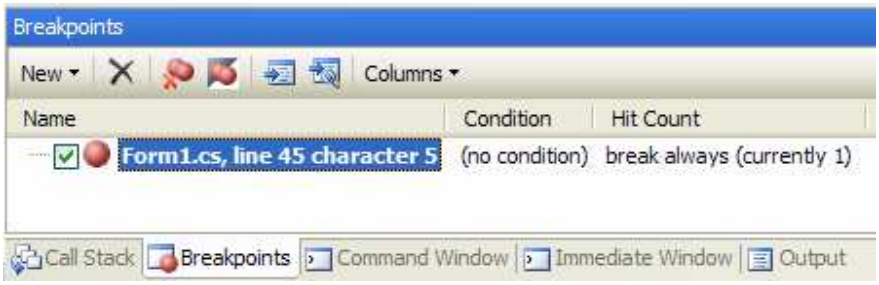
۵) برنامه را اجرا کرده و روی دکمه ی `Test` کلیک کنید. هنگامی که برنامه برای اولین بار بخواهد دستور داخل حلقه را اجرا کند، `breakpoint` قرار داده شده در این قسمت باعث می شود که در اجرای برنامه وقفه ایجاد شود. بعد از توقف برنامه چندین بار روی آیکون `Step Into` در نوار ابزار `Debug` کلیک کنید. مشاهده می کنید که با کلیک روی این آیکون کد برنامه خط به خط اجرا می شود.

۶) حال `breakpoint` در کنار این دستور را حذف کنید. سپس در نوار ابزار `Debug` روی آیکون `Step Out` کلیک کنید. مشاهده خواهید کرد که برنامه تا انتهای متد `TestMethod` را اجرا می کند و در اولین خط درون متد `btnTest_Click` توقف می کند (همانطور که در کد برنامه دیدید، متد `TestMethod` توسط متد `btnTest_Click` فراخوانی می شود. پس متد `btnTest_Click` در لیست متدهای در حال اجرا، قبل از متد `TestMethod` قرار می گیرد).

در این قسمت با نحوه ی استفاده از `breakpoint` ها به صورت ساده آشنا شدیم. اما انعطاف پذیری `breakpoint` ها بیش از آنچه است که در این قسمت مشاهده کردید. برای مثال می توانید برای یکی از دستورات درون حلقه ی `while` یک `breakpoint` قرار دهید. سپس تعیین کنید که این `breakpoint` بعد از بیست بار اجرای حلقه باعث ایجاد وقفه در برنامه شود. و یا شرط خاصی را تعیین کنید و بگویید که در صورت درست بودن این شرط، `breakpoint` باعث ایجاد وقفه در اجرای برنامه شود و در غیر این صورت `breakpoint` غیر فعال باشد. برای تعیین این شرایط باید ابتدا با نحوه کارکرد پنجره ی `Breakpoints` آشنا شویم. در بخش بعدی به این مطلب خواهیم پرداخت.

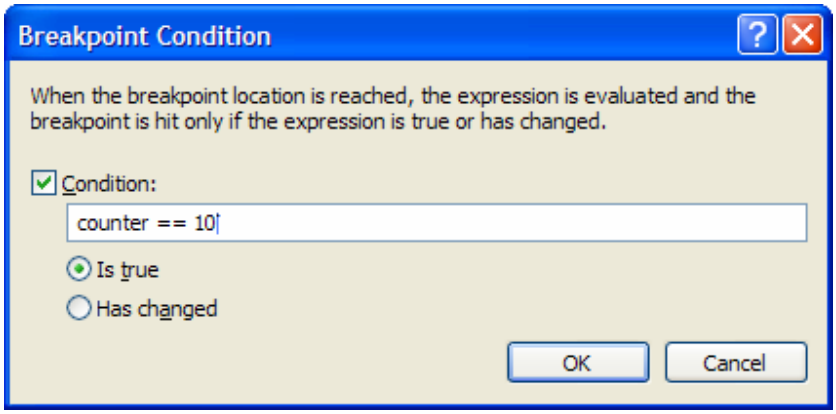
پنجره ی Breakpoints:

با استفاده از این پنجره می توانید تمامی breakpoint هایی که در برنامه وجود دارند را در یک لیست مشاهده کنید. هنگامی که اجرای برنامه به یکی از این breakpoint ها برسد، گزینه ی مربوط به آن breakpoint در لیست برجسته خواهد شد. برای نمایش پنجره ی Breakpoints می توانید در نوار ابزار Debug روی سمت راست ترین آیکون کلیک کنید و یا با استفاده از نوار منو گزینه ی Debug ➔ Windows ➔ Breakpoints را انتخاب کنید. به این ترتیب پنجره ی Breakpoints مشابه شکل ۱۱-۱۲ نمایش داده خواهد شد.



شکل ۱۱-۱۲

در این پنجره علاوه بر نمایش مکان هر breakpoint، شرط اجرای آن و تعداد دفعاتی که تاکنون برنامه با این breakpoint برخورد کرده است نیز نمایش داده می شود. همچنین در این پنجره می توانید یک breakpoint جدید تعریف کرده، یکی از breakpoint های کنونی را حذف کرده، آن را غیر فعال کنید و یا ویژگی های آن را تغییر دهید. یکی از این ویژگی هایی که در این پنجره قابل تنظیم است این است که هر breakpoint فقط در شرایط خاصی باعث ایجاد وقفه در اجرای برنامه شود. برای مثال تصور کنید که در برنامه ی قبل می خواهید فقط هنگامی اجرای برنامه متوقف شود که مقدار شمارنده ی counter برابر با ۱۰ باشد. برای اینکه برای یک breakpoint شرط خاصی تعیین کنید، روی نام آن در پنجره ی Breakpoints کلیک راست کنید و از منویی که باز می شود گزینه ی ...Condition را انتخاب کنید. در این صورت پنجره ای مشابه شکل ۱۱-۱۳ نمایش داده خواهد شد.

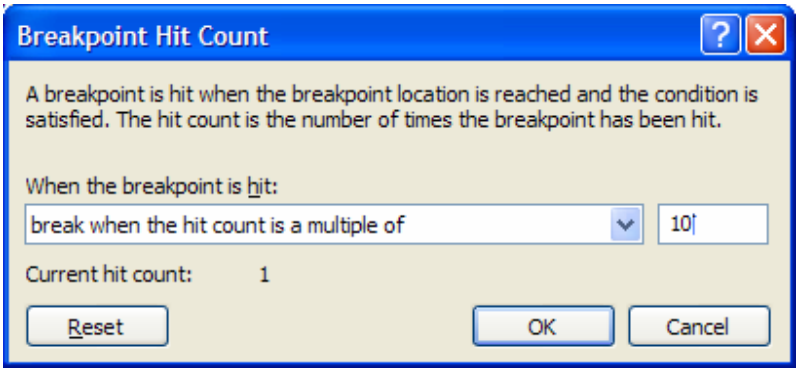


شکل ۱۱-۱۳

برای مثال در این پنجره مشخص شده است فقط هنگامی این breakpoint باید موجب ایجاد وقفه در برنامه شود که مقدار متغییر counter برابر با ۱۰ باشد.

یکی دیگر از ویژگی های breakpoint ها که در این پنجره قابل تنظیم است این است که breakpoint فقط در تعداد دفعات تکرار مشخصی فعال شود. این ویژگی بیشتر برای breakpoint هایی به کار می رود که درون یک حلقه قرار

می گیرند و برنامه در طول اجرا چندین بار با آنها مواجه می شود. برای مثال فرض کنید در برنامه قبلی می خواهید هنگامی که عدد درون متغیر `counter` برابر با ضریبی از ۱۰ بود (اعداد ۱۰، ۲۰، ۳۰ و ...)، `breakpoint` فعال و در غیر این صورت غیر فعال باشد. برای این کار روی `breakpoint` مورد نظرتان در پنجره ی `Breakpoints` کلیک راست کنید و از منوی باز شده گزینه ی ...`Hit Count` را انتخاب کنید. به این ترتیب پنجره ی `Breakpoint Hit Count` همانند شکل ۱۱-۱۴ نمایش داده خواهد شد.



شکل ۱۱-۱۴

در این پنجره می توانید مشخص کنید که `breakpoint` بعد از دفعات تکرار مشخصی اجرا شود (برای مثال بعد از ۱۰ بار)، همواره اجرا شود، فقط وقتی مرتبه اجرا ضریب عدد خاصی بود اجرا شود و یا فقط در `n`امین مرتبه اجرا شود.
ابزارها و پنجره هایی که در این قسمت با آنها آشنا شدیم، فقط قسمتی از ابزارهای موجود در ویژوال استودیو برای اشکال زدایی برنامه ها بودند. پنجره های بسیار دیگری مانند `Immediate Window`، `Command Window`، `Locals`، `Autos`، `Watch` و ... نیز وجود دارند که البته توضیح تمام آنها خارج از اهداف این کتاب است. برای آشنایی با این پنجره ها و نیز نحوه ی استفاده از آنها می توانید به سیستم راهنمای ویژوال استودیو (MSDN) مراجعه کنید. واضح است که هر چه با نحوه ی کارکرد این ابزارها و پنجره ها بیشتر آشنا باشید، راحت تر می توانید خطاهای موجود در برنامه را پیدا کرده و آنها را برطرف کنید.

کنترل استثنا ها در برنامه:

همانطور که در ابتدای فصل گفتم یک برنامه ممکن است در طول اجرای خود با شرایط پیش بینی نشده ای مواجه شود. برای مثال در برنامه ی قبلی مشاهده کردید که اگر کاربر نام فایل را در کنترل `txtAddress` به صورت نادرست وارد کند، برنامه با یک حالت پیش بینی نشده مواجه می شود. اما با هیچ کدام از ابزارها و روشهایی که در قسمتهای قبلی فصل با آنها آشنا شدیم نمی توان از رخ دادن چنین حالتی جلوگیری کرد و یا آنها را در طی برنامه کنترل کرد.
هنگامی که برنامه با یک چنین حالت استثنایی مواجه می شود، برای مثال یک ارتباط شبکه ای در برنامه قطع می شود و یا فایل مورد نظر برای امری خاص در کامپیوتر پیدا نمی شود، در اصطلاح گفته می شود که یک **استثنا**[1] رخ داده است. رخ دادن یک استثنا هم می تواند به وسیله کلاسهای تعریف شده در `.NET` اعلام شود (برای مثال سعی کنید با استفاده از کلاس `System.IO.File`، فایلی را باز کنید که وجود ندارد) هم می تواند به وسیله ی متدی درون کلاسهای تعریف شده در برنامه اعلام شود (برای مثال پارامترهایی که به یکی از متدهای کلاس فرستاده اند دارای مقدار اشتباهی است).

[1] `Exception`

برای اعلام کردن اینکه یک استثنا رخ داده است، ابتدا باید یک شیئ از کلاس Exception (و یا کلاسهای مشتق شده از این کلاس) ایجاد شود و اطلاعات مربوط به استثنای به وجود آمده در این شیئ قرار گیرد. سپس این شیئ در اصطلاح باید توسط برنامه ای که با این استثنا مواجه شده است، **پرتاب**[1] شود.

ممکن است این عبارت که "شیئ حاوی اطلاعات خطا باید از طرف برنامه پرتاب شود" در ابتدا کمی گیج کننده و یا نا مفهوم به نظر برسد و یا ممکن است این سوال به وجود بیاید که بعد از پرتاب شدن این شیئ چگونه می توان استثنای به وجود آمده را کنترل کرد؟ خوب، برای کنترل این نوع حالتها در کد معمولاً از بلاک های try/catch استفاده می کنند. به عبارت دیگر کدی که ممکن است با یک استثنا مواجه شود را درون یک بلاک try قرار می دهند. اگر در هنگام اجرای این کد استثنایی رخ داد، شیئ ای از نوع Exception به وجود آمده و پرتاب می شود. در این حالت اولین بلاک catch **که شرایط دریافت شیئ پرتاب شده را داشته باشد**، آن را دریافت می کند و به کنترل استثنای به وجود آمده می پردازد.

بعد از اینکه یکی از بلاک های catch موجود در برنامه توانست شیئ پرتاب شده را دریافت کند، به بررسی حالت استثنای به وجود آمده می پردازد. در این حالت اگر این بلاک بتواند، استثنای به وجود آمده را تصحیح کرده و یا بدون در نظر گرفتن آن به اجرای برنامه ادامه می دهد. در غیر این صورت پیغام مناسبی را به کاربر نمایش دهد و برنامه را به اتمام می رساند.

بلاک های try/catch با ساختاری به صورت زیر در برنامه ایجاد می شوند.

```
private void MethodA()
{
      // Some usual statements
      try
      {
            // Some dangerous codes that may encounter
            // with unavoidable exceptions goes here
      }
      catch
      {
            // Some codes to handle exceptions
      }
}
```

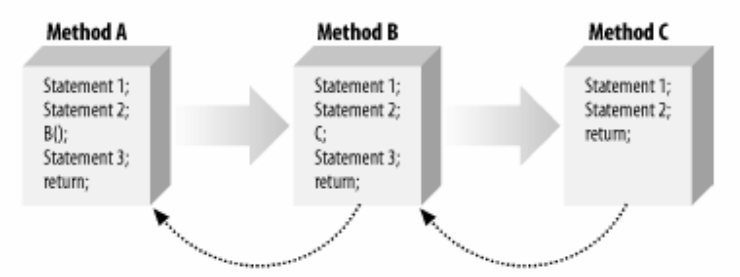
نکته: درک تفاوت بین خطاهای موجود در کد برنامه با حالتهای استثنایی که ممکن است برنامه با آنها مواجه شود از اهمیت زیادی برخوردار است. خطاهای موجود در کد برنامه معمولا به علت خطای برنامه نویس در هنگام پیاده سازی کد برنامه است. اما استثنا ها معمولاً به علت رخ دادن شرایط قابل پیش بینی اما اجتناب ناپذیر است. برای مثال تصور کنید که برنامه در شرایطی خاص در یک حلقه ی بینهایت قرار می گیرد و کاربر ناچار به بستن برنامه می شود. چنین رخ دادی به علت اشتباه برنامه نویس است و خطای برنامه نویس محسوب می شود. اما تصور کنید در حال خواند اطلاعات یک فایل در شبکه هستید که ناگهان شبکه قطع می شود. وقوع چنین شرایطی در برنامه قابل پیش بینی است اما واضح است که نمی توان از آنها جلوگیری کرد. این دسته موارد جزئی از استثنا ها به شمار می روند و باید به درستی در برنامه کنترل شوند.

چگونگی یافتن بلاک Catch برای یک استثنا:

[1] Throw

همانطور که گفتم هنگامی که قسمتی از برنامه با یک استثنا مواجه شد، شیئ ای از نوع Exception را ایجاد کـرده و بعـد از اینکه اطلاعات مربوط به استثنای به وجود آمده را در آن قرار داد، آن را پرتاب می کند. سپس این شیئ پرتـاب شـده مـی توانـد بـه وسیله ی یکی از بلاک های catch دریافت شود. اما باید به این نکته توجه کنید بلاک catch ای که این شیئ پرتاب شده را دریافت می کند، لزوماً نباید در همان متدی باشد که حالت استثنا به وجود آمده است.

برای مثال تصور کنید که در یک برنامه سه متد به نامهای A و B و C وجود دارند. متد A متد B را فراخوانی می کند و متد B نیـز متد C را فراخوانی می کند (شکل ۱۱-۱۵). حال فرض کنید که دستور دوم در متد C با یک استثنا مواجه شود. در این حالت متـد C شیئ ای از نوع Exception را ایجاد کرده و اطلاعات مربوط به این استثنا را در آن قرار می دهد و سـپس آن را پرتـاب مـی کند. برنامه ابتدا متد C را بررسی می کند تا ببیند آیا بلاک catch در این متد وجود دارد که بتوانـد اسـتثنای بـه وجـود آمـده را کنترل کند؟ در صورتی که چنین بلاکی در متد C یافته شد، کنترل برنامه به این بلاک می رود. در غیر این صورت شیئ پرتاب شده توسط متد C به متد B خواهد رفت. درون متد B برنامه مجدداً به دنبال یک بلاک catch می گردد تا بتواند استثنای بـه وجـود آمده را دریافت و کنترل کند. همین مراحل برای متد A نیز تکرار می شود. اگر در هیچ کدام از متـدهای A و B و C بلاکـی بـرای کنترل این استثنا یافته نشد، شیئ پرتاب شده به وسیله ی CLR دریافت می شود و پیغام خطای پیش فرض بـرای آن نمـایش داده شده و برنامه بسته می شود.



شکل ۱۱-۱۵

کلاس Exception:

برای این که رخ دادن یک استثنا اعلام شود، باید اطلاعات مربوط به این استثنا در شیئ ای خاص قـرار بگیـرد و سـپس ایـن شـیئ پرتاب شود. این شیئ باید از کلاس Exception و یا یکی از کلاسهای مشتق شده از آن، نمونه سازی شده باشد.

هنگامی که یکی از کلاسهای .NET با یک استثنا مواجه شود، بسته به نوع آن استثنا یک شیئ مناسب را ایجاد کرده و پرتاب می کند. برای مثال اگر پارامتر نادرستی به این متد فرستاده شود شیئ از کلاس ArgumentException تولید و پرتاب می شـود. اگـر آدرس فایـل و یـا دایرکتـوری کـه بـه ایـن متـد فرسـتاده شـده اسـت وجـود نداشـته باشـد، شـیئ ای از کـلاس DirectoryNotFoundException و یا FileNotFoundException ایجـاد شـده و پرتـاب می شود. تمام کـلاس هـایی کـه بـرای اعـلام یـک اسـتثنا توسـط .NET مـورد اسـتفاده قـرار مـی گیرنـد، از کلاسـی بـه نـام SystemException مشتق شده اند[1].

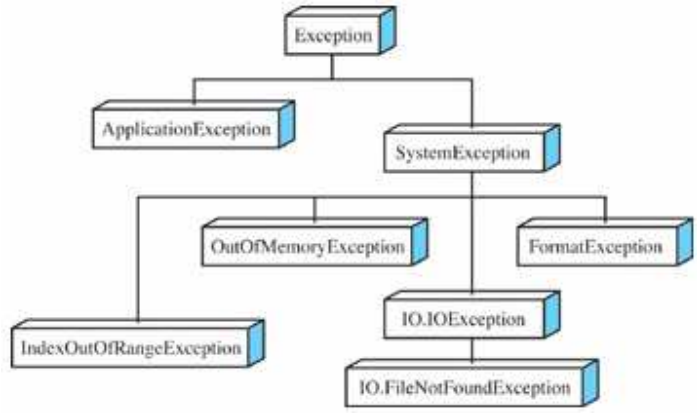
اما همانطور که گفتم فقط .NET نیست که می تواند رخ دادن یک استثنا را اعلام کند. بلکه برنامه نویس نیز اگر در کلاس هـایی که طراحی می کند با یک استثنا مواجه شد، می تواند از این روش استفاده کند. برای مثال طراحی کلاس Car در فصل نهم را بـه خطر بیاورید. خاصیت NumberOFDoors در این کلاس فقط می تواند اعداد در بازه ی ۱ تا ۶ را قبول کنـد. بنـابراین مـی

[1] کلاس SystemException خود نیز از کلاس Exception مشتق می شود.

توانید عدد فرستاده شده به این خاصیت را بررسی کرده و اگر خارج از این محدوده بود، شیئ ای را از کلاس ArgumentOutOfRangeException ایجاد کرده و آن را پرتاب کنید و به این وسیله رخ دادن یک استثنا در این کلاس را اعلام کنید.

اگر هم هیچ یک از کلاسهای موجود در .NET برای استثنای به وجود آمده در کلاس شما مناسب نبود، می توانید خودتان یک کلاس برای این استثنا ایجاد کنید و سپس شیئ ای را از آن نمونه سازی کرده و پرتاب کنید. البته این کلاس حتماً باید از کلاس ApplicationException مشتق شده باشد.

اگر بخواهم مطالب بالا را دسته بندی کنم، باید بگویم که اشیایی که هنگام رخ دادن یک استثنا به وجود آمده و پرتاب می شوند، حتماً باید از کلاس Exception و یا یکی از کلاس های مشتق شده از آن نمونه سازی شوند. از کلاس Exception دو کلاس کلی به نامهای SystemException و ApplicationException مشتق می شوند. حال اگر یکی از کلاسهای درونی .NET با استثنا مواجه شد، آن کلاس برای اعلام رخ دادن استثنا از یکی از کلاسهای مشتق شده از کلاس SystemException مانند کلاس ArgumentException و یا FileNotFoundException استفاده می کند. اما اگر کلاسی که در برنامه طراحی کرده ایم با استثنا مواجه شد، هم می توانیم مانند .NET، از کلاسهای مشتق شده از SystemException استفاده کنیم و هم می توانیم کلاس جدیدی را از کلاس ApplicationException مشتق کرده و از آن برای اعلام رخ دادن استثنا استفاده کنیم (شکل ۱۱-۱۶).



شکل ۱۱-۱۶

دستور throw:

همانطور که گفتم برای اعلام رخ دادن یک استثنا در برنامه باید شیئ ای را از نوع Exception ایجاد کرده و آن را پرتاب کنید. برای پرتاب شیئ ایجاد شده می توانید از دستور throw استفاده کنید. در بخش امتحان کنید بعد، یک کلاس آزمایشی ایجاد کرده و در یکی از متدهای این کلاس با استفاده از این دستور اعلام خواهیم کرد که یک استثنا رخ داده است. همچنین در این برنامه مشاهده خواهید کرد که اگر برای یک استثنای به وجود آمده هیچ بلاک catch ای یافت نشود، چه اتفاقی رخ خواهد داد.

امتحان کنید: استفاده از دستور throw

۱) با استفاده از نوار منوی ویژوال استودیو گزینه ی File ➔ New ➔ Project... را انتخاب کرده و سپس یک برنامه ی تحت کنسول (Console Application) به نام ThrowCommand ایجاد کنید.

۲) در فایل Program.cs، از کلاس Program کلاس جدیدی به نام Tester ایجاد کرده و کد زیر را درون این کلاس قرار دهید.

```
class Tester
{
      public void Run()
      {
            Console.WriteLine("Enter Run...");
            Func1();
            Console.WriteLine("Exit Run...");
      }

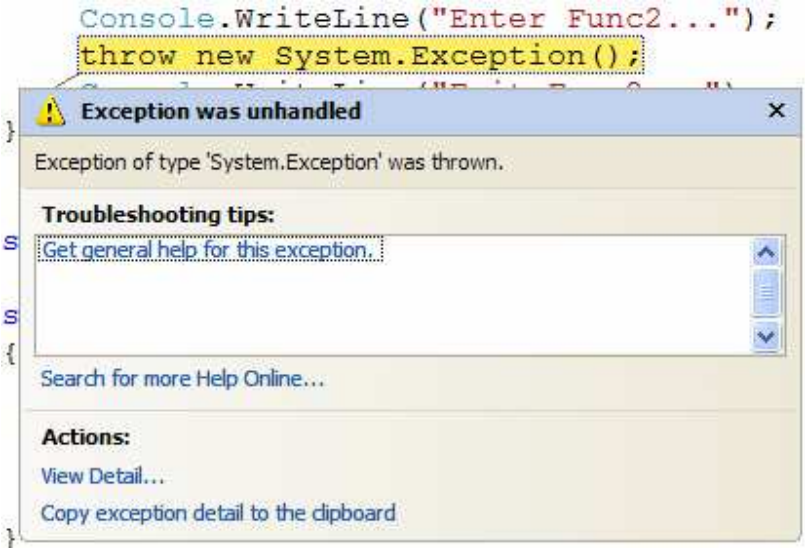
      public void Func1()
      {
            Console.WriteLine("Enter Func1...");
            Func2();
            Console.WriteLine("Exit Func1...");
      }

      public void Func2()
      {
            Console.WriteLine("Enter Func2...");
            throw new System.Exception();
            Console.WriteLine("Exit Func2...");
      }
}
```

۳) در کلاس Program به قسمت مربوط به متد Main بروید و کد مشخص شده در زیر را به بدنه ی این متد اضافه کنید.

```
static void Main(string[] args)
{
      Console.WriteLine("Enter Main...");
      Tester t = new Tester();
      t.Run();
      Console.WriteLine("Exit Main...");
}
```

۴) حال برنامه را اجرا کنید. مشاهده می کنید پنجره ای مشابه شکل ۱۱-۱۷ نمایش داده می شود. این پنجره را قبلاً نیز هنگامی که در حال کار با پروژه ی ErrorHandling بودید مشاهده کرده اید. اما این مرتبه اطلاعات درون این کادر بر اساس استثنایی است که با استفاده از دستور throw ایجاد کرده ایم.



شکل ۱۱-۱۷

۵) با انتخاب گزینه ی Debug ➔ Stop Debugging از نوار منوی ویژوال استودیو برنامه را ببندید.

چگونه کار می کند؟

همانطور که مشاهده می کنید در این برنامه هیچ کار خاصی انجام نمی شود، فقط چند تابع فراخوانی می شوند و وارد و خارج شدن از تابع در خروجی نوشته می شود. در ابتدا برنامه یک شیئ از کلاس Tester ایجاد کرده و متد Run را از این شیئ فراخوانی می کند. متد Run بعد از نوشتن عبارتی در خروجی متد Func1 را فراخوانی کرده و متد Func1 نیز متد Func2 را فراخوانی می کند. درون متد Func2، با استفاده از دستور new یک شیئ جدید از کلاس System.Exception ایجاد کرده و سپس با استفاده از دستور throw آن را پرتاب می کنیم.
با این کار برنامه بلافاصله درون متد Func2 به دنبال یک بلاک catch می گردد تا بتواند این خطا را کنترل کند. اما هیچ بلاکی را پیدا نمی کند. بنابراین به سراغ متد Func1 (که احضار کننده ی متد Func2 محسوب می شود) خواهد رفت. در این متد هم هیچ بلاک catch ای یافت نمی شود. به همین ترتیب متد های Run و Main نیز برای یافتن یک بلاک catch مناسب بررسی می شوند. اما همانطور که در کد هم مشاهده می کنید، چنین بلاکی یافت نمی شود. بنابراین CLR اجرای برنامه را به حالت تعلیق در آورده و پیغام خطای پیش فرض را همانند شکل ۱۱-۱۷ نمایش می دهد.

نکته: دقت کنید در اینجا به علت اینکه برنامه به وسیله ی ویژوال استودیو اجرا شده است چنین پیغام خطایی نمایش داده شده و خطی که باعث به وجود آمدن خطا شده است نمایش داده می شود. اگر فایل اجرایی این برنامه را بدون استفاده از ویژوال استودیو اجرا کنید، پیغامی مشابه شکل ۱۱-۱۸ مشاهده خواهید کرد. برای دسترسی به فایل اجرایی برنامه به فولدری که برنامه را در آن ایجاد کرده اید بروید، فایل اجرایی برنامه در فولدر bin\debug قرار دارد. همچنین مشاهده می کنید که محتویات این صفحه به زبان فارسی نمایش داده شده اند. دلیل این مورد این است که تنظیمات محلی این کامپیوتر برابر با کشور ایران و زبان فارسی است. اگر تنظیمات کامپیوتر شما برابر با زبان انگلیسی است، محتویات این صفحه به زبان انگلیسی نمایش داده خواهد شد.



شکل ۱۱-۱۸

در اینجا به علت اینکه هیچ حالت خاصی برای اعلام یک استثنا رخ نداده است (نه ارتباط شبکه ای قطع شده است، نه تقسیم بر صفر صورت گرفته است و نه ...) صرفا یک شیئ از کلاس `Exception` را ایجاد کرده و پرتاب می کنیم. اما در برنامه های واقعی بهتر است با بررسی علت رخ دادن استثنا، از یکی از کلاسهای دیگر `.NET` متناسب با حالت به وجود آمده استفاده کنیم و یا کلاس مخصوصی را طراحی کنیم تا بهتر مشخص شود که علت به وجود آمدن خطا چه بوده است.
استفاده از این روش مزایای دیگری را نیز شامل می شود که بعد از آشنایی با بلاک های `try` و `catch`، به اهمیت آن پی خواهید برد.

دستورات try و catch:

برای اینکه بتوانیم یک استثنای رخ داده را کنترل کنیم، باید با استفاده از دستور `catch` بلاکی ایجاد کنیم که بتواند شیئ پرتاب شده برای آن استثنا را دریافت کند. اما بلاک `catch` به تنهایی نمی تواند ایجاد شود باید همراه با بلاک `try` تعریف شود. برای تعریف بلاک `try` هم می توانید از دستور `try` استفاده کنیم. به این ترتیب کدی که ممکن است موجب ایجاد استثنا شود را در یک بلاک `try` قرار می دهیم و سپس یک بلاک `catch` بعد از بلاک `try` نیز می نویسیم تا در صورت بروز استثنا کنترل برنامه به این بلاک فرستاده شود.
در بخش امتحان کنید بعد، نحوه ی استفاده از این دستورات را در یک برنامه مشاهده خواهیم کرد.

امتحان کنید: استفاده از دستورات `try` و `catch`

۱) با استفاده از ویژوال استودیو یک برنامه ی تحت کنسول جدید (`Console Application`) به نام `TryCatchDemo` ایجاد کنید.

۲) کد زیر را در داخل فضای نام TryCatchDemo قبل از تعریف کلاس Program وارد کنید تا کلاسی به نام Tester به برنامه اضافه شود:

```
class Tester
{
      public void Run()
      {
            Console.WriteLine("Enter Run...");
            Func1();
            Console.WriteLine("Exit Run...");
      }

      public void Func1()
      {
            Console.WriteLine("Enter Func1...");
            Func2();
            Console.WriteLine("Exit Func1...");
      }

      public void Func2()
      {
            Console.WriteLine("Enter Func2...");
            try
            {
                  Console.WriteLine("Entering try block...");
                  throw new System.Exception();
                  Console.WriteLine("Exiting try block...");
            }
            catch
            {
                  Console.WriteLine(
                              "Exception caught and handled!");
            }
            Console.WriteLine("Exit Func2...");
      }
}
```

۳) دستورات مشخص شده در زیر را نیز در بدنه ی متد Main از کلاس Program قرار دهید.

```
static void Main(string[] args)
{
      Console.WriteLine("Enter Main...");
      Tester t = new Tester();
      t.Run();
      Console.WriteLine("Exit Main...");
      Console.ReadLine();
```

```
}
```

۴) برنامه را اجرا کنید. مشاهده خواهید کرد با وجود اینکه با استفاده از دستور `throw` رخ دادن یک استثنا در برنامه اعلام شده است، اما اجرای برنامه متوقف نمی شود.

چگونه کار می کند؟

در این برنامه نیز تمام قسمت ها واضح است. قسمتی از دستور که ممکن است باعث رخ دادن استثنا شود (البته در اینجا همواره استثنا رخ خواهد داد) درون یک بلاک `try` قرار داده شده است. برنامه وارد متد `Func2` می شود و این مورد را در خروجی چاپ می کند. سپس وارد بلاک `try` می شود و بعد از چاپ عبارت در خروجی با رخ دادن یک استثنا روبرو می شود. با رخ دادن استثنا برنامه به دنبال یک بلاک `catch` می گردد تا بتواند این استثنا را کنترل کند. همانطور که مشاهده می کنید نزدیکترین بلاک درون متد `Func2`، بعد از بلاک `try` قرار دارد. بنابراین کنترل برنامه به این قسمت منتقل می شود و عبارت `"Exception caught and handled!"` در خروجی نمایش داده می شود.

```
public void Func2()
{
      Console.WriteLine("Enter Func2...");
      try
      {
            Console.WriteLine("Entering try block...");
            throw new System.Exception();
            Console.WriteLine("Exiting try block...");
      }
      catch
      {
            Console.WriteLine(
                  "Exception caught and handled!");
      }
      Console.WriteLine("Exit Func2...");
}
```

دقت کنید بعد از اینکه دستورات درون بلاک `catch` تمام شد، کنترل برنامه به داخل بلاک `try` بر نمی گردد. بلکه از بلاک `catch` خارج شده و ادامه ی دستورات بعد از این بلاک در متد را اجرا می کند (عبارت `"Exit Func2..."` را در خروجی نمایش می دهد). بنابراین تمام دستوراتی که بعد از رخ دادن یک استثنا در بلاک `try` وجود دارند اجرا نخواهند شد (مشاهده می کنید که عبارت `"Exiting try block..."` در خروجی نمایش داده نمی شود).
این مورد برای حالتهایی که استثنا توسط یک بلاک `catch` خارج از این متد دریافت می شد نیز صادق است. برای مثال تصور کنید که دستورات متد `Func2` درون بلاک `try` قرار نگرفته باشند، بلکه دستورات متد `Run` درون بلاک `try` باشند. در این صورت هنگامی که برنامه در متد `Func2` با استثنا مواجه می شد، هیچ بلاکی برای کنترل این استثنا در این متد پیدا نمی کرد. بنابراین از اجرای بقیه ی دستورات این متد صرفنظر می کرد و به درون متد `Func1` می رفت. درون متد `Func1` نیز بلاک مناسبی پیدا نمی کرد، بنابراین دستورات باقیمانده در متد `Func1` را نیز اجرا نمی کرد و به متد `Run` می رفت. در این متد

کنترل برنامه را به بلاک catch موجود می سپرد و اجرای برنامه از دستورات این متد به بعد ادامه پیدا می کرد (ایـن تغییـرات را در کد ایجاد کنید و بعد از اجرای برنامه، خروجی آن را بررسی کنید تا بهتر با نحوه ی انجام این موارد آشنا شوید).

ایجاد بلاک های catch اختصاصی:

بلاک catch ای که در برنامه ی قبلی ایجاد کردیم، استثنا هایی را که شیئ مربوط به آنها از نوع Exception بـود مـی توانست کنترل کند. همانطور که می دانید هنگامی که یک کلاس از کلاس دیگری مشتق شود، اشیایی که از کلاس نمونـه سـازی شده ایجاد می شوند می توانند به صورت اشیایی از کلاس پایه رفتار کنند. بنابراین در اینجا هـم اگـر بـرای مثـال اسـتثنایی از نـوع ArgumentException بـه وجـود مـی آمـد، شـیئ مربـوط بـه ایـن اسـتثنا مـی توانسـت بـه شـیئ ای از نـوع Exception تبدیل شود و سپس توسط این بلاک catch کنترل شود. بنابراین می تـوانیم بگـوییم بـلاک catchای که در برنامه ی قبلی ایجاد کردیم هر نوع استثنایی را می توانست کنترل کند.
اما در شرایطی ممکن است بخواهیم یک بلاک catch را به نحوی ایجاد کنیم که فقط بتواند نوع خاصی از استثنا هـا را کنتـرل کند. برای مثال تصور کنید در قسمتی از یک متد در حال خواندن یک فایل از شبکه هستید و می خواهید اگر در طی این مدت شبکه قطع شد از ادامه ی خواندن فایل صرفنظر کرده و بقیه ی دستورات متد را اجرا کنید. امـا اگـر اسـتثنای دیگـری در متـد رخ داد مـی خواهید آن استثنا به متد بالاتر منتقل شود (به عبارت دیگر نمی خواهید استثنا های دیگر، درون این متد کنترل شوند). در این حالـت باید بلاک catch را به گونه ای تغییر دهید که فقط استثنا های نوع خاصی را بتواند دریافت کند. در بخش امتحان کنیـد بعـد در طی یک برنامه با نحوه ی انجام این کار بیشتر آشنا خواهیم شد.

امتحان کنید: ایجاد بلاک های catch اختصاصی

۱) با استفاده از ویژوال اسـتودیو یـک برنامـه ی تحـت کنسـول جدیـد (Console Application) بـه نـام DedicatedCatchDemo ایجاد کنید.
۲) درون فضای نام مربوط به این برنامه و خارج از کلاس Program کد زیر را وارد کنید تا کـلاس جدیـدی در برنامـه ایجاد شود.

```
class Tester
{
     public void Run()
     {
          try
          {
               double a = 5;
               double b = 0;
               Console.WriteLine("Dividing " + a +
                                 " by " + b + "...");
               Console.WriteLine(a + " / " + b +
                                 " = " + DoDivide(a, b));
          }
          // most derived exception type first
```

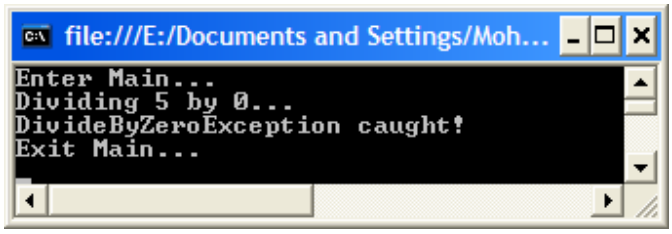
```
            catch (System.DivideByZeroException)
            {
                Console.WriteLine(
                    "DivideByZeroException caught!");
            }
            catch (System.ArithmeticException)
            {
                Console.WriteLine(
                    "ArithmeticException caught!");
            }
            // generic exception type last
            catch
            {
                Console.WriteLine(
                    "Unknown exception caught");
            }
        }

        // do the division if legal
        public double DoDivide(double a, double b)
        {
            if (b == 0)
                throw new System.DivideByZeroException();
            if (a == 0)
                throw new System.ArithmeticException();
            return a / b;
        }
}
```

۳) کد مشخص شده در زیر را به بدنه ی متد `Main` در کلاس `Program` اضافه کنید تا برنامه کامل شود.

```
static void Main(string[] args)
{
    Console.WriteLine("Enter Main...");
    Tester t = new Tester();
    t.Run();
    Console.WriteLine("Exit Main...");
    Console.ReadLine();
}
```

۴) برنامه را اجرا کنید. مشاهده خواهید کرد که استثنای به وجود آمده توسط بلاک `catch` کلی دریافت نخواهد شد (عبارت `"Unknown exception caught"` در خروجی چاپ نمی شود)، بلکه به علت اینکه استثنای به وجود آمده از نوع `DivideByZeroException` است کنترل برنامه به اولین بلاک `catch` فرستاده می شود و عبارت `"DivideByZeroException caught!"` در خروجی نمایش داده می شود (شکل ۱۱-۱۹).



شکل ۱۱-۱۹

چگونه کار می کند؟

در کلاس Tester در برنامه ی بالا، متد Run از متد DoDivide برای تقسیم دو عدد استفاده می کند. این متد ابتدا پارامتر ها را بررسی می کند، اگر پارامتر دوم صفر بود شیئ ای از نوع DivedieByZeroException ایجاد کرده و پرتاب می کند، زیرا انجام چنین تقسیمی در برنامه باعث به وجود آمدن خطا می شود. همچنین اگر پارامتر اول صفر بود این متد استثنایی را از نوع ArithmeticException تولید می کند. البته در این حالت نباید استثنایی تولید شود زیرا تقسیم صفر بر هر عددی یک تقسیم منطقی است. اما برای اینکه مثال این قسمت کامل شود متد DoDivide در صورتی که پارامتر اول صفر باشد استثنایی را از نوع ArithmeticException تولید می کند.

```
// do the division if legal
public double DoDivide(double a, double b)
{
     if (b == 0)
          throw new System.DivideByZeroException();
     if (a == 0)
          throw new System.ArithmeticException();
     return a / b;
}
```

در بدنه ی متد Run دو متغییر به نامهای a و b تعریف کرده و مقادیر ۵ و ۰ را در آنها قرار می دهیم. سپس سعی می کنیم با استفاده از متد DoDivide آنها را بر یکدیگر تقسیم کنیم. همانطور که می دانید در این حالت متد DoDivide استثنای DivideByZeroException را ایجاد کرده و پرتاب می کند و چون در متد DoDivide قسمتی برای کنترل این استثنا وجود ندارد، برنامه به متد Run برمی گردد. در متد Run سه بلاک catch مختلف وجود دارد. بلاک اول فقط می تواند استثنا هایی از نوع DivideByZeroException را دریافت کند. بلاک دوم فقط می تواند استثنا هایی از نوع ArithmeticException را دریافت کند و بلاک سوم هر نوع استثنایی را می تواند کنترل کند. بنابراین استثنای تولید شده در متد DoDivide توسط بلاک اول دریافت می شود و برنامه از درون این بلاک ادامه پیدا می کند.

```
public void Run()
{
     try
     {
          double a = 5;
```

```
            double b = 0;
            Console.WriteLine("Dividing " + a +
                              " by " + b + "...");
            Console.WriteLine(a + " / " + b +
                              " = " + DoDivide(a, b));
        }
        // most derived exception type first
        catch (System.DivideByZeroException)
        {
            Console.WriteLine(
                "DivideByZeroException caught!");
        }
        catch (System.ArithmeticException)
        {
            Console.WriteLine(
                "ArithmeticException caught!");
        }
        // generic exception type last
        catch
        {
            Console.WriteLine(
                "Unknown exception caught");
        }
    }
```

توجه کنید که کلاس مربوط به استثنای DivideByZeroException از کلاس مربوط به استثنای ArithmeticException، و خود این کلاس نیز از کلاس Exception مشتق می شود. پس اشیای نمونه سازی شده از کلاس DivideByZeroException می توانند به کلاسهای ArithmeticException و یا Exception نیز تبدیل شوند.
بنابراین اگر در متد Run ترتیب قرارگیری بلاک های catch را تغییر دهیم، برای مثال بلاک ArithmeticException را قبل از بلاک DivideByZeroException قرار دهیم برنامه هیچگاه وارد بلاک DivideByZeroException نخواهد شد. زیرا اگر شیئ مربوط به یک استثنا از نوع DivideByZeroException باشد، وقتی با بلاکی از نوع ArithmeticException برخورد کند به این کلاس تبدیل شده و کنترل برنامه وارد این بلاک خواهد شد. بنابراین دقت کنید که همواره بلاک های catch مربوط به دریافت استثنا های خاص را قبل از بلاک های catch مربوط به دریافت استثنا های عمومی قرار دهید.

خاصیت ها و متدهای کلاس Exception:

در قسمتهای قبلی همواره گفته شده است که برای اعلام رخ دادن یک استثنا باید اطلاعات مربوط به آن استثنا را در یک شیئ از نوع Exception قرار داد و سپس آن را پرتاب کرد تا به وسیله ی یک بلاک دریافت شود. نحوه ی انجام این موارد را نیز در مثال های قبلی مشاهده کردیم. اما در اینجا این سوال پیش می آید که چگونه می توان درون یک بلاک catch به اطلاعاتی در مورد استثنای به وجود آمده دسترسی پیدا کرد.

برای دسترسی به اطلاعات یک استثنا در بلاک catch می توانیم از کلاس Exception و یا کلاسهای مشتق شده از آن استفاده کنیم. برای مثال اگر یک بلاک catch برای دریافت استثنا هایی از نوع ArithmeticException در متد قرار دادیم، درون این بلاک می توانیم به تمام خاصیت ها و متدهای کلاس ArithmeticException دسترسی داشته باشیم. قبل از اینکه نحوه ی این کار را در برنامه مشاهده کنیم، بهتر است ابتدا با یک سری از خاصیت ها و متدهای مهم کلاس Exception (که مسلماً در تمام کلاسهای مشتق شده از آن نیز وجود دارند) آشنا شویم.

اولین و مهمترین خاصیت این کلاس، خاصیت Message می باشد که حاوی اطلاعاتی در مورد دلیل رخ دادن این استثنا است. کدی که استثنا در آن رخ داده است، مقدار این خاصیت را تنظیم کرده و سپس شیئ را پرتاب می کند. اما بعد از تنظیم شدن دیگر نمی توان مقدار آن را تغییر داد. به عبارت دیگر این خاصیت یک خاصیت فقط-خواندنی است.

خاصیت مهم دیگر خاصیت HelpLink است و حاوی لینکی است که به راهنمای استثنای به وجود آمده اشاره می کند. این خاصیت به صورت خواندنی-نوشتنی است و مقدار آن در هر قسمتی از برنامه می تواند تغییر کند.

در قسمت امتحان کنید بعد، برنامه ای خواهیم نوشت و در آن قبل از اعلام رخ دادن یک استثنا مقدار بعضی از خاصیت های آن را تنظیم می کنیم. همچنین در بلاک catch از مقدار این خاصیتهای تنظیم شده استفاده خواهیم کرد تا به اطلاعات استثنای به وجود آمده دسترسی داشته باشیم.

امتحان کنید: خاصیتهای کلاس Exception

۱) با استفاده از ویژوال استودیو برنامه ای تحت کنسول به نام ExceptionProperties ایجاد کنید.

۲) درون فضای نام برنامه و قبل از کلاس Program کد زیر را وارد کنید تا کلاس Tester به برنامه اضافه شود.

```
class Tester
{
     public void Run()
     {
          try
          {
               Console.WriteLine("Open file here");
               double a = 12;
               double b = 0;
               Console.WriteLine(a + " / " + b + " = "
                                   + DoDivide(a, b));
               Console.WriteLine(
                         "This line may or may not print");
          }
          // most derived exception type first
          catch (System.DivideByZeroException e)
          {
               Console.WriteLine(
                         "\nDivideByZeroException! Msg: "
                         + e.Message);
               Console.WriteLine("\nHelpLink: "
                                   + e.HelpLink);
```

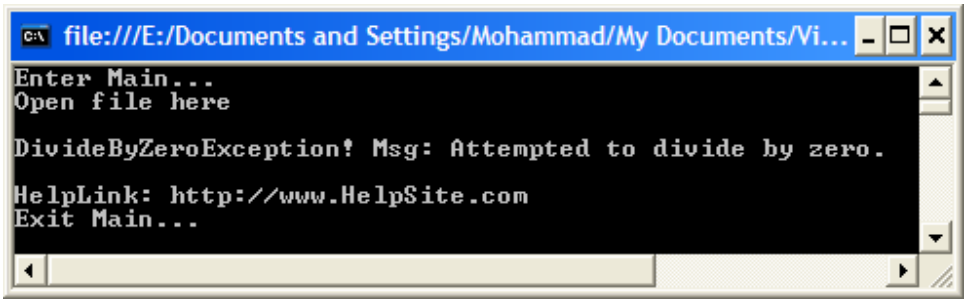
```
            }
            catch
            {
                Console.WriteLine(
                    "Unknown exception caught");
            }
        }

        // do the division if legal
        public double DoDivide(double a, double b)
        {
            if (b == 0)
            {
                DivideByZeroException e =
                    new DivideByZeroException();
                e.HelpLink =
                            "http://www.HelpSite.com";
                throw e;
            }
            if (a == 0)
                throw new ArithmeticException();
            return a / b;
        }
}
```

۳) کد زیر را نیز به بدنه ی متد `Main` درون کلاس `Program` اضافه کنید.

```
        static void Main(string[] args)
        {
            Console.WriteLine("Enter Main...");
            Tester t = new Tester();
            t.Run();
            Console.WriteLine("Exit Main...");
            Console.ReadLine();
        }
```

۴) برنامه را اجرا کنید. پنجره ای مشابه شکل ۱۱-۲۰ نمایش داده خواهد شد و اطلاعات تنظیم شده مربوط به خطایی که به وجود آمده است در آن نوشته می شود.



شکل ۱۱-۲۰

چگونه کار می کند؟

نحوه ی عملکرد این برنامه نیز مشابه برنامه ی قبل است. فقط در متد `DoDivide` در این برنامه اگر پارامتر دوم برابر با صفر باشد، ابتدا شیئ ای از نوع `DivideByZeroException` ایجاد شده و خاصیت `HelpLink` آن برابر با آدرس سایتی قرار داده می شود که درباره ی این استثنا اطلاعاتی دارد. سپس شیئ ایجاد شده با استفاده از دستور `throw` پرتاب می شود.
در متد `Run` هم اگر استثنایی از نوع `DivideByZeroException` رخ دهد، پیغام مربوط به آن استثنا (مقدار خاصیت `Message`) و آدرس سایت حاوی اطلاعات درباره ی آن (مقدار خاصیت `HelpLink`) در صفحه نمایش داده می شود.

نتیجه:

در این فصل با مقدمات کنترل خطا ها و استثنا ها در یک برنامه آشنا شدیم. در ابتدای فصل به دسته بندی انواع خطا ها پرداختیم و مشاهده کردیم که سه نوع خطای کلی وجود دارد که روش کنترل هر کدام از آنها نیز متفاوت است. اولین نوع خطا، خطاهای دستوری هستند که با استفاده از ابزارهایی مانند پنجره ی `Error List` و یا ویرایشگر کد ویژوال استودیو به سادگی تشخیص داده شده و قبل از اینکه برنامه کامپایل و اجرا شود رفع می شوند.
نوع دوم خطاها، خطاهای منطقی هستند و همانطور که گفتم سخت ترین نوع خطاها از نظر تشخیص و رفع به شمار می روند. در ویژوال استودیو `.NET` ابزارهای زیادی برای تشخیص این خطا ها وجود دارد که هر چه بیشتر با این ابزارها آشنا باشید، ساده تر می توانید با این نوع خطاها مقابله کنید. در این فصل با مهمترین ابزار در این زمینه یعنی `Breakpoint`ها آشنا شدیم. اما ابزارهای بسیار دیگری نیز وجود دارند که بهتر است نحوه ی کار با آنها را نیز یاد بگیرید، مانند پنجره ی `Watch` که برای مشاهده ی مقدار هر شیئ و یا متغییر در برنامه مورد استفاده قرار می گیرد، پنجره ی `Immediate` که برای اجرای یک دستور خارج از دستورات موجود در برنامه، زمانی که برنامه در حالت `Break` است به کار می رود و ...
در انتهای فصل نیز روشهای مختلف برخورد با خطاهای زمان اجرا را مشاهده کردیم و دیدیم که می توانیم در این شرایط با استفاده از بلاک `try/catch`، یا به کاربر اطلاع دهیم که خطایی رخ داده است و سپس از برنامه خارج شویم، یا خطا را رفع کرده و برنامه را ادامه دهیم. همچنین با نحوه ی تولید خطاهای زمان اجرا در مواقع مورد نیاز نیز آشنا شدیم.

در پایان این فصل باید با موارد زیر آشنا شده باشید:

- انواع مختلف خطاهایی که در یک برنامه ممکن است به وجود آید را نام برده و توضیح دهید.
- بتوانید از ابزارهای موجود در ویژوال استودیو استفاده کرده و خطاهای دستوری ایجاد شده را بر طرف کنید.
- با استفاده از ابزارهای ویژوال استودیو قسمتهای مختلف یک برنامه را کنترل کرده و خطاهای منطقی موجود در آن را رفع کنید.
- بتوانید با استفاده از بلاک `try/catch` در برنامه، خطاهای زمان اجرا را کنترل کنید.
- در مواقع مورد نیاز با استفاده از دستور `throw` رخ دادن خطایی را در برنامه اعلام کنید.

فصل دوازدهم: ایجاد کتابخانه های کلاس

در این فصل می خواهیم به طراحی کتابخانه های کلاس بپردازیم، کتابخانه هایی که حاوی چندین کلاس باشند. برای این کار از تمام مطالبی که تاکنون در این کتاب آموخته اید استفاده خواهیم کرد، پس بهتر است قبل از شروع فصل مروری بر این مطالب داشته باشیم. بیشتر برنامه هایی که تاکنون طراحی کرده ایم به این صورت بوده اند که با استفاده از جعبه ابزار تعدادی کنترل بر روی فرم قرار می دادیم، با استفاده از پنجره ی Properties خاصیتهای این کنترل ها را تنظیم کرده و در انتها نیز کد های مورد نیاز برای عملکرد آنها را می نوشتیم. تمام این موارد درون یک فرم در قسمت طراحی فرم ویژوال استودیو انجام می شد. اولین نکته ی قابل توجه در این قسمت این است که هنگامی که فرمی را با استفاده از قسمت طراحی فرم ویژوال استودیو طراحی می کردیم (برای مثل کنترلهای مورد نیاز را در آن قرار می دادیم)، در حقیقت در حال ایجاد کلاس جدیدی بودیم که این کلاس از کلاس System.Windows.Forms.Form مشتق شده بود.

وقتی در هنگام طراحی فرم تغییری در فرم برنامه ایجاد می کردید، ویژوال استودیو کد لازم برای آن تغییر را تشخیص می داده و آن را به این کلاس جدید اضافه می کرد. برای مشاهده ی کد هایی که ویژوال استودیو برای ایجاد کلاس مورد نیاز شما می نویسد می توانید در پنجره ی Solution Explorer روی علامت مثبت کنار فایل مربوط به فرم برنامه کلیک کنید، برای مثال فایل Form1.cs، سپس فایل Form1.Designer.cs را باز کنید. هنگامی که برنامه را ایجاد می کنید، یک شیئ جدید از این کلاس نمونه سازی می شود. بنابراین تمام مواردی که در فصل نهم و دهم در مورد اشیا گفتیم، در مورد فرم برنامه ی شما نیز صادق است. به عبارت دیگر مانند هر شیئ دیگری، فرم برنامه ی شما می تواند دارای حالت و یا رفتار باشد. شما می توانید در فرم برنامه ی خود متغییر ها و یا کنترل هایی را ایجاد کنید (حالت شیئ) و یا می توانید عمل خاصی را، برای مثال هنگامی که کاربر روی یک کنترل کلیک می کند در فرم انجام دهید (رفتار شیئ). به صورت تئوری برای نوشتن یک برنامه می توانید از یک ویرایشگر متنی ساده نیز استفاده کنید. البته در این حالت باید تمام کد هایی که ویژوال استودیو برای تکمیل کلاس مربوط به برنامه شما می نویسد، توسط خود شما نوشته شود. این مورد در برنامه های بزرگ و حتی متوسط به امری غیر عملی تبدیل می شود، بنابراین بیشتر برنامه نویسان برای طراحی برنامه های ویندوزی از یک محیط طراحی استفاده می کنند.

پس می توانیم بگوییم که از ابتدای این کتاب تاکنون در حال طراحی کلاسها بوده اید، در فصل نهم نیز با نحوه ایجاد کلاسها از پایه آشنا شدید. برنامه ی Objects را در این فصل به خاطر بیاورید، این برنامه شامل دو کلاس به نامهای Car و SportsCar بود. در آن فصل این کلاسها را در یک برنامه ی تحت کنسول ایجاد کردیم زیرا تست کردن نحوه ی عملکرد کلاسها در این نوع برنامه ها ساده تر بود. اما همانطور که می دانید این کلاسها بدون هیچ تغییری می توانستند در برنامه های تحت ویندوز و یا حتی برنامه های تحت وب و وب سرویس ها نیز مورد استفاده قرار بگیرند. در حقیقت یکی از مهمترین مزایای برنامه نویسی به صورت شیئ گرا این است که هنگامی که یک کلاس مناسب را طراحی می کنید، می توانید از آن در تمام قسمتهای مورد نیاز در برنامه های دیگر نیز استفاده کنید.

در این فصل:

- با چگونگی ایجاد کتابخانه های کلاس آشنا خواهید شد و خواهید دید که چگونه می توان در مورد کتابخانه های کلاسی که عضو چارچوب .NET نیستند اطلاعاتی را بدست آورد.
- با چگونگی نامگذاری قوی در اسمبلی ها (فایلهای کامپایل شده) آشنا خواهید شد تا به این وسیله بتوانید از منحصر به فرد بودن اسمبلی ها مطمئن شوید.
- مشاهده خواهید کرد که چگونه می توان یک اسمبلی را در مکانی عمومی به نام Global Assembly Cache یا GAC قرار داد تا به وسیله ی تمام برنامه های در حال اجرا در آن کامپیوتر قابل دسترسی باشند.

مفهوم کتابخانه های کلاس:

همانطور که در فصل دهم مشاهده کردید، برای اینکه بتوانیم به لیستی از گزینه های Favoritesدسترسی داشته باشیم و بتوانیم از آنها استفاده کنیم دو کلاس مختلف را طراحی کردیم. سپس با استفاده از این کلاسها دو برنامه ی مختلف ایجاد کرده تا گزینه های Favorites را نمایش دهند، یکی از آنها گزینه های موجود را به صورت عادی در یک برنامه ی ویندوزی و دیگری به وسیله ی یک منو در کنار ساعت سیستم نمایش می داد. در آنجا برای این که بتوانیم از کلاسهای طراحی شده در هر دو برنامه استفاده کنیم، فایل سورس آنها را از برنامه ی اول در برنامه ی دوم کپی کردیم. این روش، روش سریع و ساده ای برای استفاده ی مجدد از یک کد است اما مشکلاتی نیز دارد:

- برای استفاده از این روش باید به سورس کلاسهای نوشته شده دسترسی داشته باشید. اما همانطور که گفتم یکی از مزیت های استفاده از برنامه نویسی شیئ گرا در این است که می توانید کلاسهای طراحی شده را به صورت یک "جعبه ی سیاه" در اختیار برنامه نویسان دیگر قرار دهید. برنامه نویسان دیگر، برای استفاده از کلاس شما نیازی ندارند که بدانند آن کلاس به صورت درونی چگونه کار می کند. همچنین ممکن است اگر کلاسی را طراحی کنید بخواهید کد آن کلاس به صورت سری باقی بماند. ممکن است بخواهید کلاس خود را برای استفاده در اختیار برنامه نویسان دیگر قرار دهید، اما اجازه ندهید که آن را تغییر دهند و یا ادعا کنند که آن کلاس به وسیله آنها نوشته شده است.
- هر بار که بخواهید برنامه را کامپایل کنید، کلاس هایی که به این صورت به برنامه اضافه شده اند نیز باید کامپایل شوند. ممکن است این مورد در برنامه های کوچک که فقط از چند کلاس ساده استفاده می کنند زیاد چشمگیر نباشد، اما در برنامه های بزرگ که کلاسهای زیادی را به کار می برند، این مورد موجب کند شدن سرعت کامپایل می شود. همچنین به این صورت حجم برنامه ی تولید شده نیز بسیار زیاد خواهد بود، زیرا فایل اجرایی که در مرحله ی کامپایل تولید می شود شامل تمامی این کدها خواهد بود.
- اگر در این کلاس خطایی را مشاهده کردید و خواستید آن را تصحیح کنید و یا تصمیم گرفتید قسمتی از برنامه را به نحوی تغییر دهید تا عملکرد برنامه بهتر و کارآمدتر شود، باید این تغیرات را در تمام برنامه هایی که از این کلاس استفاده می کنند اعمال کنید.

راه حلی که برای رفع این مشکلات در این قسمت می توان به کار برد استفاده از کتابخانه های کلاس است. یک **کتابخانه ی کلاس**[1]، شامل مجموعه ای از کلاس ها است که هنگام کامپایل درون یک فایل مجزا با پسوند .DLL[2] قرار می گیرند. این فایلها به تنهایی قابل اجرا نیستند، بلکه کلاسهای موجود در آنها می توانند در برنامه های دیگر مورد استفاده قرار گیرند. با استفاده از کتابخانه های کلاس می توانید بدون دسترسی به کد، از آنها در چند برنامه استفاده کنید. همچنین با استفاده از این کتابخانه ها هر بار که برنامه کامپایل می شود نیازی نیست که کلاسهای موجود در این فایلها نیز کامپایل شوند. اگر هم یکی از کلاسهای موجود در این کتابخانه ها تغییر کند، تمام برنامه ها به صورت اتوماتیک از کد تغییر داده شده استفاده خواهند کرد.

ایجاد یک کتابخانه ی کلاس:

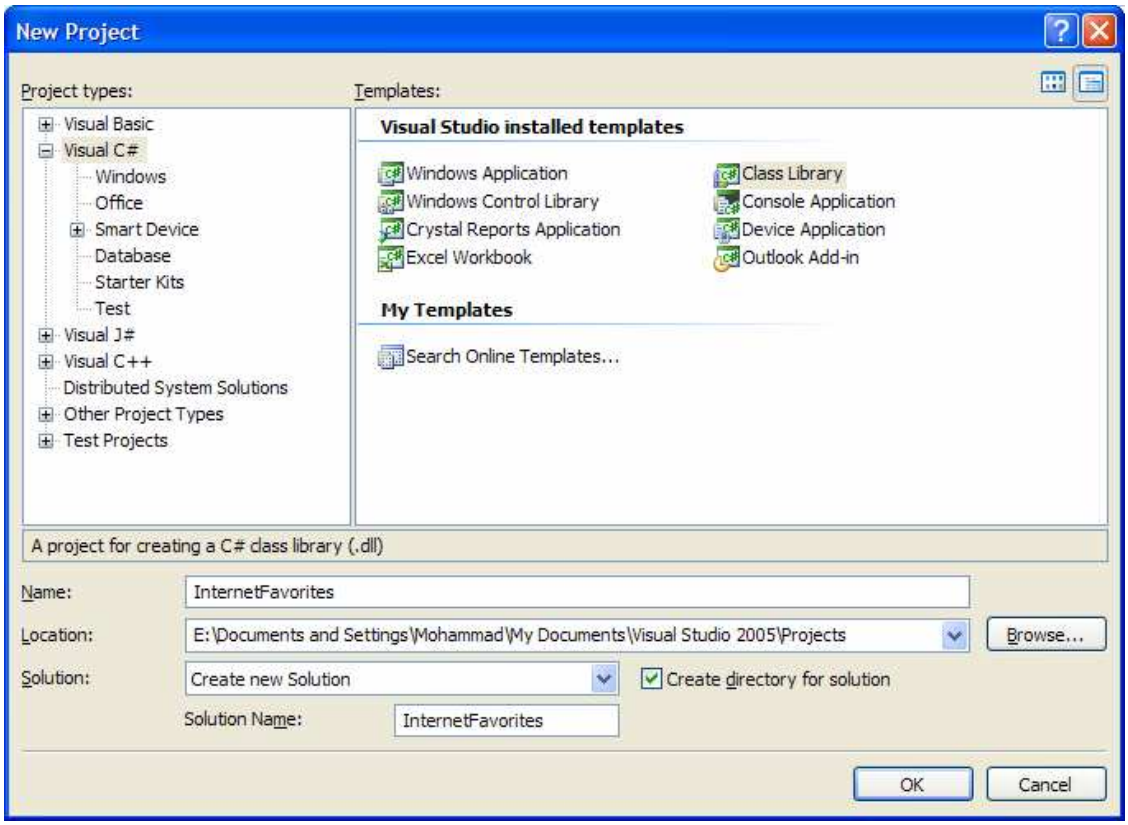
بخش امتحان کنید زیر، حاوی دستورالعمل هایی برای ایجاد یک کتابخانه ی کلاس در ویژوال استودیو .NET ۲۰۰۵ نسخه ی Standard و یا نسخه های بالاتر است.

---

[1] Class Library
[2] Dynamic Link Library

امتحان کنید: ایجاد کتابخانه ی کلاس

۱) با استفاده از نوای منوی ویژوال استودیو، گزینه ی File ➔ New Project... را انتخاب کنید.
۲) از لیست Project Type گزینه ی Visual C# و سپس از قسمت Templates آیکون Class Library را انتخاب کنید (شکل ۱۲-۱). در کادر Name نیز نام InternetFavorites را وارد کرده و روی دکمه ی OK کلیک کنید.



شکل ۱-۱۲

۳) یک پروژه ی جدید از نوع Class Library ایجاد شده و یک کلاس نیز به صورت پیش فرض به نام Class1.cs به این پروژه اضافه می شود. در پنجره ی Solution Explorer روی نام Class1.cs کلیک راست کنید و از منوی باز شده گزینه ی Delete را انتخاب کنید.

چگونه کار می کند؟

همانطور که مشاهده کردید ایجاد این نوع پروژه بسیار راحت بود. اما اجازه دهید کارهایی که ویژوال استودیو در این چند مرحله انجام می دهد را بررسی کنیم. در ابتدا تعیین می کنید که قالب پروژه ای که می خواهید ایجاد کنید از نوع Class Library

است. قالب یک پروژه تعیین می کند که ویژوال استودیو چگونه قسمتهای مختلف برنامه را تنظیم کند. بنابراین با تغییر دادن قالب این پروژه مشاهده خواهید کرد که این برنامه با برنامه های قبلی تفاوتهای زیادی دارد. اولین تفاوت در این است که در برنامه های ویندوزی قبلی، ابتدا یک فرم ویندوزی خالی به نام Form1.cs در محیط طراحی فرم در اختیار شما قرار می گرفت. اما هنگامی که پروژه ای از نوع Class Library ایجاد می کنید، هیچ فرمی در اختیار شما قرار داده نمی شود، بلکه یک کلاس خالی به نام Class1.cs نمایش داده می شود.

تفاوت اساسی دیگری که در این قسمت وجود دارد این است که هنگامی که یک برنامه ی ویندوزی ایجاد می کنید، ویژوال استودیو می داند که باید هنگام کامپایل برنامه یک فایل قابل اجرا تولید کند. اما هنگامی که پروژه ای از نوع Class Library ایجاد می کنید، ویژوال استودیو هنگام کامپایل فایلی را ایجاد می کند که به تنهایی قابل اجرا نخواهد بود. بنابراین نوع پروژه ای که انتخاب می کنید در نوع فایلی که به وسیله ی ویژوال استودیو تولید می شود تاثیر خواهد گذاشت. اگر پروژه ای را از نوع Class Library ایجاد کنید، ویژوال استودیو فایلی با پسوند dll و در غیر این صورت فایلی با پسوند exe به عنوان خروجی برنامه تولید خواهد کرد.

بعد از اینکه پروژه ی مورد نظر را ایجاد کردیم، فایلی که به صورت پیش فرض توسط ویژوال استودیو به وجود آمده است را حذف می کنیم. داشتن یک کلاس با نام Class1 در برنامه کاربردی ندارد، بنابراین بهتر است از ابتدا فایلها و کلاس هایی با نامهای معنی دار در برنامه ایجاد کنیم.

در فصل دهم کلاس هایی را طراحی کرده و سپس از آنها در دو برنامه ی گوناگون استفاده کردیم: برنامه ی Favorites Viewer و برنامه ی Favorites Tray. در این بخش می خواهیم این کلاسها را از این برنامه جدا کرده و برنامه ها را نیز به گونه ای تغییر دهیم تا هر دوی آنها از یک نسخه ی کامپایل شده از این کلاسها استفاده کنند. البته میدانید که این حالت یک حالت غیر واقعی است، زیرا در برنامه ها عموماً ابتدا کلاسها را در یک پروژه طراحی می کنند و سپس به طراحی برنامه ی اصلی می پردازند، نه اینکه مانند اینجا ابتدا برنامه ی اصلی را طراحی کنند سپس قسمتهای مختلف آن را تفکیک کرده و در کتابخانه های کلاس قرار دهند. هدف ما از این بخش فقط این است که با چگونگی ایجاد یک کتابخانه ی کلاس و نحوه ی استفاده از آن در چند پروژه از پایه آشنا شوید. برای شروع در یک پنجره ی ویژوال استودیو ی جدید پروژه ی Favorites Viewer را باز کنید. به خاطر دارید که این پروژه شامل فایلهای زیر بوده است.

- Favorites.cs که حاوی کلاس Favorites بود.
- WebFavorite.cs که حاوی کلاس WebFavorite بود.
- Form1.cs که حاوی کلاس Form1 بود. این کلاس فرم اصلی برنامه را تشکیل می داد.

از این لیست، دو فایل اول را در یک کتابخانه ی کلاس مجزا قرار می دهیم. فایل سوم فقط به این پروژه مربوط است و به آن نیازی نداریم. به عبارت دیگر می خواهیم یک کتابخانه ی کلاس حاوی کلاسهای Favorites و WebFavorite ایجاد کنیم.

ایجاد یک کتابخانه ی کلاس برای Favorites Viewer:

در فصل دوم با مفهوم راه حل ها و دلیل استفاده از آنها آشنا شدید. همانطور که به خاطر دارید در آن فصل ذکر شد که یک راه حل می تواند شامل بیش از یک پروژه باشد. در راه حل برنامه ی Favorites Viewer فقط یک پروژه وجود دارد و آن هم پروژه ی Favorites Viewer است. در بخش امتحان کنید بعد، پروژه ی دیگری از نوع Class Library را به این راه حل اضافه کرده و سپس کلاسهای برنامه را به این پروژه منتقل می کنیم.

امتحان کنید: اضافه کردن یک پروژه ی Class Library به راه حل

۱) به برنامه ی InternetFavorites بروید و بعد از ذخیره ی آن، ویژوال استودیو را ببندید.
۲) پروژه ی Favorites Viewer را در محیط ویژوال استودیو باز کنید.
۳) با استفاده از نوار منوی ویژوال استودیو گزینه ی ...File ➔ Add ➔ Existing Project را انتخاب کنید.
۴) در پنجره ی باز شده به فولدری بروی که برنامه ی InternetFavorites را در آن ایجاد کرده اید و سپس در آنجا فایل InternetFavorites.csproj را انتخاب کرده و روی دکمه ی OK کلیک کنید.
۵) در پنجره ی Solution Explorer روی نام پروژه ی Favorites Viewer کلیک راست کرده و از منوی باز شده گزینه ی Set As Startup Project را انتخاب کنید.

چگونه کار می کند؟

با انجام این کارها در راه حل خود دو پروژه خواهید داشت: یک پروژه ی ویندوزی و یک کتابخانه ی کلاس. البته فعلاً پروژه کتابخانه ی کلاس خالی است و تمام کلاس هایی که باید در آن قرار بگیرند درون پروژه ی ویندوزی هستند.
در برنامه های قبلی نحوه اضافه کردن یک کلاس به برنامه را مشاهده کرده اید. در این جا هم می توانید از همان روش استفاده کنید، یعنی روی نام پروژه کلیک راست کنید و از منوی نمایش داده شده گزینه ی Add ➔ Class را انتخاب کنید. اما کلاس هایی که می خواهیم در این قسمت به برنامه ی InternetFavorites اضافه کنیم قبلاً طراحی شده اند. پس نیازی به استفاده از این روش نیست و به راحتی می توانیم با ماوس این کلاس ها را از برنامه ی Favorites Viewer به برنامه ی InternetFavorites منتقل کنیم. در بخش امتحان کنید بعد این روش را مشاهده خواهید کرد.

امتحان کنید: انتقال کلاسها بین دو پروژه

۱) با استفاده از پنجره ی Solution Explorer همانند شکل ۱۲-۲ فایل Favorites.cs را انتخاب کرده و با ماوس آن را به پروژه ی InternetFavorites منتقل کنید. به این ترتیب یک کپی از این فایل به داخل فولدر این پروژه فرستاده خواهد شد.



شکل ۱۲-۲

۲) همین کار را برای فایل `WebFavorite.cs` نیز تکرار کنید تا به پروژه ی `InternetFavorites` منتقل شود.

۳) حال فایلهای `Favorites.cs` و `WebFavorite.cs` را از پروژه ی `Favorites Viewer` انتخاب کرده و روی آنها کلیک راست کنید. سپس با انتخاب گزینه ی `Delete` آنها را از این پروژه حذف کنید.

۴) در انتها باید فضای نام دو فایلی که به پروژه ی `InternetFavorites` اضافه شده اند را تصحیح کنید. بر روی فایل `Favorites.cs` در پنجره ی `Solution Explorer` دو بار کلیک کنید تا کد درون آن نمایش داده شود. مشاهده خواهید کرد که کلاس `Favorites` در فضای نام `Favorites_Viewer` قرار گرفته است. نام مقابل دستور `namespace` را به `InternetFavorites` تغییر دهید تا فضای نام آن تنظیم شود. به این ترتیب کد درون این فایل باید مشابه زیر باشد.

```
namespace InternetFavorites
{
      public class Favorites
      {
```

۵) همین مراحل را برای فایل `WebFavorite.cs` نیز تکرار کرده و فضای نام آن را نیز تغییر دهید تا هر دو کلاس `Favorites` و `WebFavorite` در یک فضای نام قرار بگیرند.

به این ترتیب برنامه ی شما شامل دو پروژه خواهد بود که هر یک دارای فایل های مرتبط به خود است. دقت کنید با وجود اینکه این دو پروژه در یک راه حل هستند، اما به فایلهای هم دسترسی ندارند و نمی توانند فایلهای یکدیگر را ببینند. بنابراین اگر سعی کنید که برنامه را اجرا کنید، پیغام خطایی نمایش داده می شود و اعلام می کند که کلاس `Favorites` و `WebFavorite` توسط برنامه پیدا نشده است.

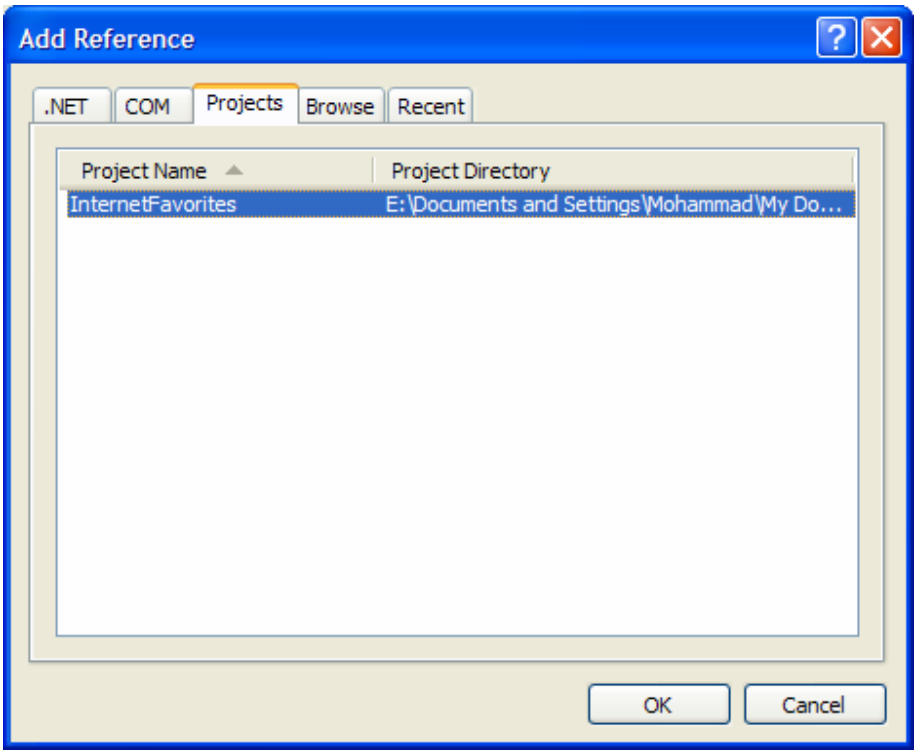
این خطا به این دلیل رخ می دهد که کد های درون فایل `Form1.cs` نمی توانند کلاسهای درون برنامه ی `InternetFavorites` را مشاهده کنند و به آنها دسترسی داشته باشند. رفع این مشکل شامل دو مرحله می شود.

- پروژه ی کتابخانه ی کلاس را به صورت یک ارجاع، به برنامه ی ویندوزی اضافه کنید. به این صورت برنامه ی ویندوزی می تواند برای پیدا کردن کلاسهای مورد نیاز خود، خروجی پروژه ی `InternetFavorites`، یعنی فایل `InternetFavorites.dll` که حاوی کلاسهای مورد نیاز است را نیز جستجو کند.
- همانطور که می دانید کلاسها در فضای نامی هم نام با پروژه ی خود قرار می گیرند. بنابراین برای این که بتوانید فقط با ذکر نام کلاسهای موجود در فایل `InternetFavorites.dll` به آنها دسترسی داشته باشید، باید با استفاده از دستور `using` فضای نام آنها را به برنامه اضافه کنید. البته بدون انجام این کار نیز می توانید از کلاسها استفاده کنید، اما در این صورت باید نام کامل کلاسها (یعنی نام خود کلاس همراه با فضای نام آن) را ذکر کنید.

در بخش امتحان کنید بعد، نحوی انجام این دو مورد را مشاهده خواهیم کرد.

امتحان کنید: اضافه کردن یک ارجاع به برنامه ی ویندوزی

۱) روی نام پروژه ی `Favorites Viewer` در پنجره ی `Solution Explorer` کلیک راست کرده و از منوی باز شده گزینه ی `Add Reference...` را انتخاب کنید.
۲) در کادر `Add Reference` به قسمت `Projects` بروید. مشاهده خواهید کرد که همانند شکل ۱۲-۳ پروژه ی `InternetFavorites` در لیست این قسمت وجود دارد. روی دکمه ی `OK` کلیک کنید تا این پروژه به عنوان یک مرجع به پروژه ی `Favorites Viewer` اضافه شود.



شکل ۱۲-۳

۳) در ابتدای فایل Form1.cs با استفاده از راهنمای using فضای نام InternetFavorites را مانند زیر به برنامه اضافه کنید.

```
using InternetFavorites;
```

چگونه کار می کند؟

با اضافه کردن یک ارجاع به پروژه ی InternetFavorites در برنامه ی ویندوزی در مراحل ۱و ۲، در حقیقت به ویژوال استودیو گفته اید که فایل FavoritesViewer.exe برای اجرا شدن به فایل InternetFavorites.dll و کلاسهای داخل آن نیاز دارد. به این ترتیب ویژوال استودیو به شما اجازه می دهد در برنامه ی خود از کلاس هایی که در فایل InternetFavorites.dll وجود دارد استفاده کنید.

نکته: هر زمان که بخواهید از یک کتابخانه ی کلاس در برنامه ی خود استفاده کنید باید یک ارجاع به آن کتابخانه را در برنامه ایجاد کنید. اگر آن کتابخانه به صورت یک پروژه از نوع Class Library در برنامه بود می توانید از روش قبل استفاده کنید. اگر هم به صورت یک فایل dll کامپایل شده بود می توانید با استفاده از قسمت Browse در کادر Add Reference آدرس فایل را مشخص کنید تا ویژوال استودیو آن را به برنامه اضافه کند.

تا قبل از مرحله ی سوم با وجود اینکه مشخص کرده اید که برنامه باید از کلاسهای درون پروژه ی InternetFavorites استفاده کند، اما اگر سعی کنید برنامه را اجرا کنید با پیغام خطا مواجه خواهید شد. دلیل این خطا نیز این است که سعی کرده اید بدون ذکر نام کامل کلاسهای Favorites و WebFavorite از آنها در برنامه استفاده کنید. تمام این کلاسها در فضای نام InternetFavorites هستند، پس برای استفاده از آنها یا باید فضای نام آن را با استفاده از راهنمای using به برنامه اضافه کرد و یا نام کلاس را به نام کامل آن تغییر داد. مسلماً استفاده از راهنمای using ساده تر است، بنابراین با استفاده از این راهنما فضای نام InternetFavorites را به برنامه اضافه می کنیم.
خوب، تمام مراحل مورد نیاز برای این قسمت همین بود. به این ترتیب برنامه ی شما به دو قسمت تقسیم شد: یکی شامل یک برنامه ی ویندوزی کوچک برای قسمت کاربری برنامه و دیگری نیز شامل یک کتابخانه ی کلاس حاوی کلاس های مورد نیاز در قسمت کاربری برنامه. حال می توانید برنامه را اجرا کنید، مشاهده خواهید کرد که برنامه همانند قبل به درستی کار می کند.
دقت کنید که برای کامپایل برنامه ی Favorites Viewer، ویژوال استودیو ابتدا برنامه ی InternetFavorites را کامپایل کرده و فایل dll آن را تولید می کند. سپس برنامه ی ویندوزی را کامپایل کرده و فایل exe را تولید می کند. دلیل این مورد هم مشخص است، زیرا کارکرد برنامه ی Favorites Viewer به کلاسهای موجود در InternetFavorites بستگی دارد.

برنامه های چند لایه:

در برنامه ی قبل با ایجاد یک کتابخانه ی کلاس در حقیقت برنامه را به دو **لایه**[1] تقسیم کردیم. یکی از این لایه ها کلاسهای موجود در کتابخانه ی کلاس بود. وظیفه این لایه این بود که کامپیوتر کاربر را جستجو کرده و تمام گزینه های موجود در بخش Favorites پیدا کرده و در یک آرایه قرار دهد. لایه ی دیگر نیز برنامه ی ویندوزی ایجاد شده بود. وظیفه ی این لایه این بود که گزینه های Favorites که به وسیله ی لایه ی قبلی در آرایه قرار داده شده بود را به نحو مناسبی به کاربر نمایش دهد و همچنین به کاربر اجازه دهد تا آنها را به وسیله ی اینترنت اکسپلورر مشاهده کند.
کتابخانه های کلاس معمولاً ابزار بسیار مناسبی برای ایجاد برنامه های چند لایه به شمار می روند، زیرا به وسیله آنها می توانید قسمت های مختلف برنامه را در لایه های جداگانه تقسیم کنید. این برنامه یک برنامه ی ساده به شمار می رفت به همین دلیل فقط از دو لایه تشکیل شده بود. اما ممکن است در برنامه های بزرگ عبارت "برنامه نویسی چند لایه" را زیاد بشنوید. در دنیای واقعی برنامه های چند لایه حداقل از سه لایه ی مختلف و مجزا تشکیل می شوند و این سه لایه نیز به صورت زیر تعریف می شوند:

- **لایه ی داده ها**[2] فقط بر روی دریافت اطلاعات خام از یک منبع اطلاعاتی و فرستادن اطلاعات پردازش شده به این منبع تمرکز می کند. این منبع اطلاعاتی می تواند شامل یک بانک اطلاعاتی، یک فایل متنی و یا هر نوع منبع اطلاعاتی دیگر باشد. این لایه در نوع و یا مفهوم اطلاعات فرستاده شده و یا دریافت شده هیچ دخالتی ندارد و فقط موظف است که وظیفه ی خواندن و نوشتن در منبع اطلاعاتی را انجام دهد. در برنامه ی قبلی فولدر Favorites در کامپیوتر کاربر به عنوان لایه ی اطلاعات به شمار می رفت.
- **لایه ی تجاری**[3] در یک برنامه، فقط مقررات و قوانین خاصی را بر داده هایی که از بانک اطلاعاتی دریافت می شوند و یا به بانک اطلاعاتی فرستاده می شوند اعمال می کند. به عبارت دیگر بیشتر توجه این لایه بر این است که برای مثال داده هایی که به بانک اطلاعاتی فرستاده می شوند حتماً دارای شرایط خاصی باشند و قبل از نوشته شدن در بانک اطلاعاتی صحت آنها بررسی شود. برای مثال در برنامه ی قبل ممکن است بخواهید قبل از اینکه اطلاعات در برنامه نمایش داده شوند، از درست بودن لینک مطمئن شوید و یا از نمایش داده شدن لینک های خاصی در برنامه جلوگیری کنید. البته ممکن است کد هایی نیز در این قسمت قرار داده شوند تا داده ها را تغییر دهند و یا کارهایی روی آنها انجام دهند. برای مثال ممکن است کد مورد نیاز برای نمایش دادن یک لینک در برنامه در این لایه قرار داده شود.
- **لایه ی ارائه دهنده**[4] داده های رسیده را به کاربر نمایش می دهد و به او اجازه می دهد تا با این داده ها کار کند. در این مثال، برای این لایه یک برنامه ی ویندوزی ایجاد کردید که لینک ها را به صورت یک لیست نمایش می دهد و همچنین به کاربر اجازه می دهد تا با استفاده از اینترنت اکسپلورر آنها را مشاهده کند.

برنامه ای که در قسمت قبل طراحی کردیم بسیار کوچک بود، بنابراین نیازی نبود که لایه ی داده را از لایه ی تجاری مجزا کنیم. اما در یک برنامه ی بزرگ، تقسیم برنامه به چند لایه باعث می شود که مدیریت قسمتهای مختلف آن بسیار ساده تر انجام شود، حتی اگر با این روش طراحی برنامه زمان بیشتری را نیاز داشته باشد.
یکی دیگر از مزایای تقسیم یک برنامه به چند لایه در این است که به این صورت می توانید قسمتهای مختلف برنامه را هنگام نیاز به راحتی تغییر دهید، بدون اینکه دیگر قسمتها نیازی به تغییر کردن داشته باشند. برای مثال تصور کنید که بعد از مدتی استفاده از نوع خاصی از مرورگر ها به جز اینترنت اکسپلورر فراگیر شود و شما بخواهید برنامه ی خود را به صورتی تغییر دهید که اطلاعات مورد نیاز خود را از منوی Favorites این مرورگر دریافت کند. در این حالت فقط لازم است که کد مربوط به دریافت اطلاعات را در لایه ی داده ها تغییر دهید. به این ترتیب قسمتها و لایه های دیگر برنامه بدون تغییر باقی می مانند و مانند قبل به درستی با لایه ی داده ای جدید کار می کنند.

---

[1] Layer
[2] Data Layer
[3] Business Layer
[4] Presentation Layer

نکته: هدف از برنامه نویسی چند لایه، پیچیده تر از آن است که در این کتاب مورد بررسی قرار گیرد. اما برای آشنایی مقدماتی با این نوع برنامه نویسی و نقش .NET در آن می توانید به ضمیمه ی ۲ مراجعه کنید.

در ادامه ی فصل مشاهده خواهید کرد که چگونه می توان از کتابخانه کلاس InternetFavorites که در حقیقت ترکیبی از دو لایه ی داده ای و لایه ی تجاری این برنامه است در برنامه ای دیگر به نام Favorites Tray استفاده کرد.

نکته: در این فصل فقط با پروژه ی انجام شده در فصل دهم کار می کنیم. به این ترتیب می توانیم به جای نوشتن کد بیشتر بر مفهوم و نحوه ی کارکرد کتابخانه های کلاس تمرکز کنیم.

استفاده از نامگذاری قوی

برنامه ای که در مرحله ی قبل ایجاد کردید هنگام کامپایل دو فایل مجزا تولید می کند: یک فایل exe و یک فایل dll که هر دوی آنها به وسیله ی شما طراحی و نوشته شده است. مسلماً مطمئن هستید که فرد دیگری برنامه ی خود را بر اساس کلاسهای موجود در این فایل dll نخواهد نوشت و یا فرد دیگری به جز شما کد های درون این فایل dll را تغییر نخواهد داد. اما در برنامه های واقعی معمولاً چنین شرایطی به وجود نمی آید. در برنامه های واقعی اغلب از dll هایی استفاده می کنید که به وسیله ی یک گروه برنامه نویس نوشته شده و به صورت گسترده در بین دیگر برنامه نویسان توزیع شده اند و یا ممکن است عضو یک گروه برنامه نویسی باشید که در آن افرادی روی یک dll و افراد دیگری روی یک فایل exe کار می کنند.
برای مثال تصور کنید که فرد A در حال کار روی فایل InternetFavorites.dll است و فرد B نیز روی فایل FavoritesViewer.exe کار می کند. فرد A احساس می کند که نام ScanFavorites نام مناسبی نیست و آن را به LoadFavorites تغییر می دهد. سپس برنامه را کامپایل کرده و فایل dll جدیدی را تولید می کند. فرد B بدون اینکه این مورد را بداند برنامه ی FavoritesViewer.exe را اجرا کرده و این برنامه نیز سعی می کند که متد ScanFavorites را فراخوانی کند. اما این متد دیگر وجود ندارد بنابراین برنامه با خطا مواجه شده و متوقف می شود.
البته ممکن است بگویید که در این مورد فرد A نباید نام متد را در فایل dll تغییر می داد و باید متوجه می بود که برنامه هایی وجود دارند که اجرای آنها به متد ScanFavorites در این کلاس بستگی دارد. اما همواره برنامه نویسانی هستند که بدون توجه به مشکلات ناشی از این کار، به این صورت عمل می کند و موجب از کار افتادن برنامه ها می شوند.
مشکل دیگری که ممکن است در این حالت رخ دهد این است که همزمان با فرد A، فرد C نیز یک کتابخانه ی کلاس به نام InternetFavorites ایجاد کرده و بخواهد از آن استفاده کند. این کتابخانه ی کلاس با فایل نوشته شده به وسیله ی فرد A تفاوت دارد و اگر هر دوی آنها برای کار روی یک کامپیوتر قرار بگیرند این دو فایل با هم اشتباه خواهند شد و مجدداً برنامه ها به درستی کار نخواهند کرد.
مشکلاتی که به این صورت برای مدیریت فایلهای DLL رخ می دهد از ابتدای برنامه نویسی ویندوز وجود داشته است و به کابوسی برای برنامه نویسان تبدیل شده بود، به صورتی که معمولا از آنها به عنوان "جهنم DLLها"[1] یاد می کنند. در محیط .NET تلاش زیادی شده است تا این مشکلات تا حد ممکن برطرف شود. عمده ی مشکلاتی که به این صورت هستند، دو دلیل کلی دارند:

[1] DLL Hell

- از یک فایل DLL ممکن است چندین نسخه وجود داشته باشد و هر کدام نیز ممکن است به نحوی متفاوت عمل کنند. همچنین با استفاده از نام فایل نمی توان نسخه ی آن را تشخیص داد.
- افراد و شرکتهای مختلف می توانند فایلهای DLLای با نام مشابه ایجاد کنند.

هنگامی که یک اسمبلی به صورت قوی نامگذاری شود[1] اطلاعات مربوط به شماره ی نسخه و نویسنده ی آن اسمبلی نیز در آن ذخیره می شود. به این ترتیب هنگام استفاده از یک اسمبلی می توان بین نسخه ی مورد استفاده در برنامه و نسخه های جدیدتر و یا قدیمی تر تفاوت قائل شد. همچنین به این وسیله می توان فایل `InternetFavorites.dll` که به وسیله ی فرد A نوشته شده است را از فایل `InternetFavorites.dll` که توسط فرد C نوشته شده است تشخیص داد. با نامگذاری قوی یک اسمبلی می توان اطلاعات دیگری را نیز در مورد آن اسمبلی در آن ذخیره کرد تا به این ترتیب از منحصر به فرد بودن یک اسمبلی مطمئن شد (برای مثال فرهنگ و زبانی که برای نوشتن آن اسمبلی به کار رفته است[2]) اما در این قسمت فقط بر مشخص کردن نویسنده و نسخه ی یک اسمبلی تمرکز خواهیم کرد.

امضا کردن اسمبلی ها:

یک راه برای مشخص کردن این موضوع که یک اسمبلی به وسیله ی چه کسی نوشته شده است این است که آن اسمبلی **امضا** شود. برای انجام این کار می توانید یک جفت-کلید ایجاد کرده و سپس با این کلید ها اسمبلی را امضا کنید. این کلید ها هنگامی که تولید می شوند به صورت منحصر به فرد هستند. بنابراین وقتی توسط فرد یا شرکتی برای امضا کردن یک اسمبلی به کار روند، می توان از نوشته شدن آن اسمبلی توسط آن فرد و یا شرکت مطمئن شد. اصولی که در پشت امضا کردن این اسمبلی ها به کار می روند مباحث کاملاً پیچیده ای هستند، اما نحوه ی انجام این کار بسیار ساده است.

نکته: یک اسمبلی که به صورت قوی نامگذاری می شود نمی تواند از یک اسمبلی که به صورت عادی نامگذاری شده است استفاده کند، زیرا به این ترتیب ممکن است کنترل نسخه ها در این اسمبلی از بین برود.

نامگذاری یک اسمبلی به صورت قوی شامل دو مرحله می شود:

- ایجاد یک جفت-کلید که برای نامگذاری اسمبلی به کار می رود. نحوه ی انجام این کار را در بخش امتحان کنید بعد مشاهده خواهید کرد.
- کلید ایجاد شده را به اسمبلی اضافه کنید. به این ترتیب هنگام کامپایل اسمبلی این کلید برای نامگذاری قوی آن به کار می رود.

امتحان کنید: ایجاد یک جفت-کلید

[1] به این نوع اسمبلی ها `Strongly Named Assemblies` نیز گفته می شود.

[2] `Assembly Culture`

۱) با استفاده از منوی Start در ویندوز گزینه ی All Programs ➔ Microsoft Visual Studio 2005 ➔ Visual Studio 2005 Tools ➔ Visual Studio 2005 Command Prompt را انتخاب کنید.
۲) در خط فرمانی که نمایش داده می شود، دستور زیر را تایپ کنید:

```
sn -k InternetFavoritesKey.snk
```

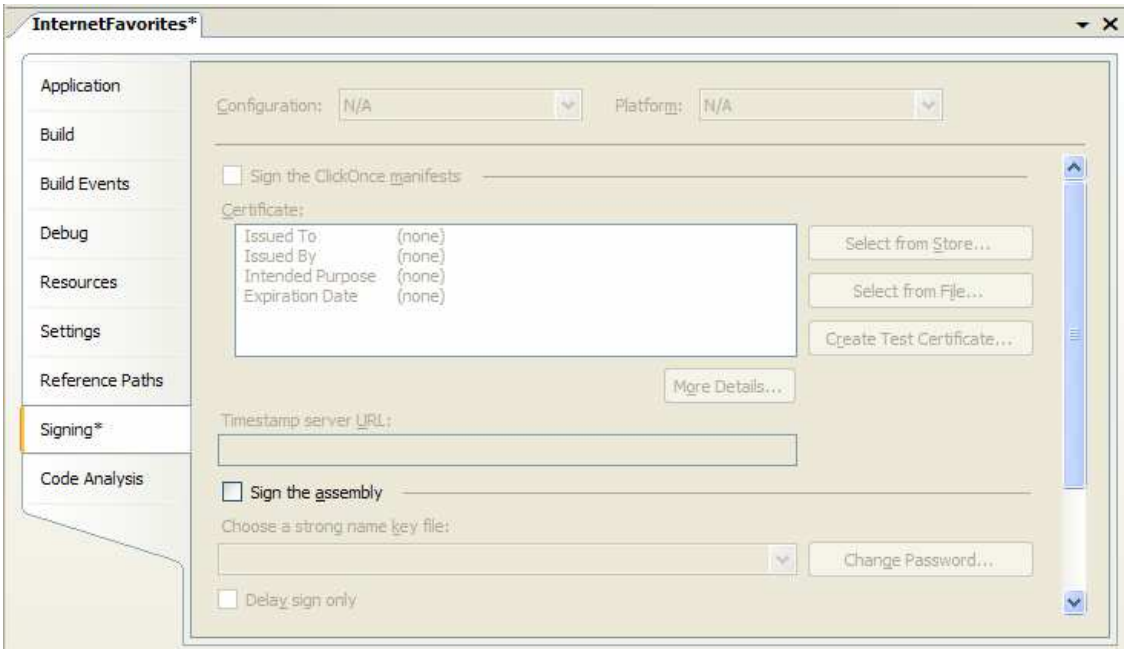
به این ترتیب فایلی مربوط به یک جفت-کلید در دایرکتوری که در آن قرار دارید ایجاد می شود (در ای حالت C:\Program Files\Microsoft Visual Studio 8\VC).

چگونه کار می کند؟

اجرای خط فرمان ویژوال استودیو ۲۰۰۵ باعث می شود محیطی همانند محیط DOS نمایش داده شود که این محیط برای کار با ابزارهای ویژوال استودیو تنظیم شده است. یکی از این ابزارها که برای ایجاد جفت-کلید به کار می رود، فایل sn.exe است و به صورت دستور sn مورد استفاده قرار می گیرد. هنگام اجرای این دستور از سوییچ k استفاده می کنیم تا تعیین کنیم که می خواهیم یک جفت-کلید جدید در فایل مشخص شده ایجاد کنیم.
بعد از اجرای این دستور یک جفت کلید در فایل InternetFavoritesKey.snk به آدرس C:\Program Files\Microsoft Visual Studio 8\VC ایجاد می شود. اگر بخواهید می توانید این فایل را به مکان مناسب تری مانند فولدر پروژه انتقال دهید. در بخش امتحان کنید بعد نحوه ی استفاده از فایل را برای نامگذاری قوی یک اسمبلی مشاهده خواهیم کرد.

امتحان کنید: نامگذاری قوی اسمبلی ***InternetFavorites***

۱) در پنجره ی Solution Explorer روی گزینه ی Properties در قسمت مربوط به پروژه ی InternetFavorites دو بار کلیک کنید.
۲) در نوار سمت چپ پنجره ای که نمایش داده می شود روی عبارت Signing کلیک کنید (شکل ۱۲-۴).
۳) در ای قسمت گزینه ی "Sign the assembly" را انتخاب کنید.
۴) در قسمت Choose a strong name key file کلیک کرده و گزینه ی ...Browse را انتخاب کنید. سپس در پنجره ی نمایش داده شده به آدرس فایل جفت-کلیدی که ایجاد کرده اید بروید و آن فایل را انتخاب کنید.
۵) برنامه را مجدداً کامپایل کنید. به این ترتیب فایل dll تولید شده به صورت قوی نامگذاری خواهد شد.



شکل ۱۲-۴

چگونه کار می کند؟

هنگامی که یک جفت کلید را به این صورت به برنامه اضافه می کنید، کامپایلر کلید عمومی این جفت کلید را به اسمبلی اضافه می کند. همچنین با استفاده از کلید خصوصی، محتویات اسمبلی را به نحوی خاص کد گذاری کرده و متن کد شده را در اسمبلی قرار می دهد.

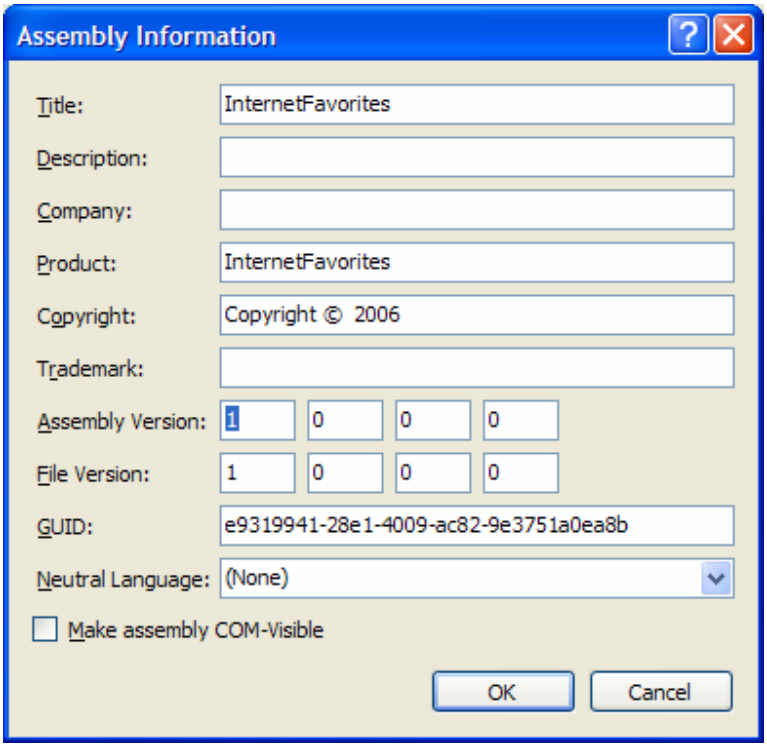
در سیستم رمزنگاری با استفاده از کلیدهای عمومی-خصوصی، یک پیغام با استفاده از یکی از این کلید ها رمز گذاری می شود اما برای باز کردن رمز آن باید از کلید دیگر استفاده کرد. به عبارت دیگر با استفاده از یک کلید نمی توانید هم عمل قفل کردن و هم عمل باز کردن پیغام را انجام دهید. بنابراین اگر در این روش یک متن را با استفاده از کلید اول قفل کنید، تمام افرادی که کلید دوم را در اختیار دارند می توانند متن قفل شده را بخوانند، اما نمی توانند آن را تغییر داده، مجدداً قفل کرده و توزیع کنند.

برای امضای اسمبلی ها با استفاده از این روش به این صورت عمل می شود که بعد از ایجاد فایل اسمبلی، محتویات آن را با استفاده از کلید خصوصی قفل کرده، سپس فایل قفل شده را به همراه کلید عمومی توزیع کنید. به این ترتیب تمام افراد می توانند با استفاده از کلید عمومی این فایل را باز کرده و از آن استفاده کنند. اما نمی توانند آن را تغییر داده و مجدداً قفل کنند. بنابراین با استفاده از این روش می توان مطمئن شد که فردی به جز دارنده ی کلید خصوصی نمی تواند فایل را تغییر دهد.

بررسی این موارد توسط ویژوال استودیو انجام می شود. به این صورت که اگر فایلی که به صورت قوی نامگذاری شده است توسط افراد دیگری تغییر داده شود، ویژوال استودیو از اجرای این فایل خودداری خواهد کرد. البته به علت اینکه حجم یک اسمبلی معمولاً زیاد است و قفل گذاری و باز کردن آن ممکن است باعث کاهش سرعت شود، در این روش قسمت خاصی از فایل اسمبلی کد گذاری می شود که البته این قسمت نیز قابل تشخیص نیست و به صورت پراکنده انتخاب می شود.

نسخه های یک اسمبلی:

نسخه ی یک فایل اغلب ترتیب تولید آن را نمایش می دهد. هنگام ایجاد اسمبلی در ویژوال استودیو نیازی نیست که نگـران نـسخه ی آن باشید، زیرا ویژوال C# آن را به صورت اتوماتیک تنظیم می کند. هر بار که یک فایل اسمبلی را کامپایل می کنید عددی کـه به عنوان نسخه ی فایل به کار می رود توسط ویژوال C# به روز رسانده می شود تا شماره ی نسخه ی جدید را اعلام کند.
عددی که به عنوان نسخه ی یک اسمبلی به کار می رود از چهار بخش تشکیل شـده اسـت: Major، Minor، Build و Revision. بـرای مـشاهده ی مقـدار ایـن اعـداد در پنجـره ی Solution Explorer روی فایـل Properties مربوط به پروژه دو بار کلیک کنید تا پنجره ی نمایش داده شده در شکل ۱۲-۴ مجدداً ظاهر شود. سپس روی قـسمت Application کلیـک کـرده و در پنجـره ی نمـایش داده شـده روی دکمـه ی Assembly Information کلیک کنید. به این ترتیب کـادر Assembly Information مـشابه شـکل ۱۲-۵ نمـایش داده می شود.



شکل ۱۲-۵

همانطور که در تصویر نمایش داده شده است، با هر بار کامپایل کردن شماره ی مربوط به قسمت Major برابر با 1 و شـماره ی مربوط به قسمت Minor برابر با ۰ است. بقیه شماره ها نیز به وسیله ی ویژوال استودیو تنظیم می شوند تا در هر بار، فـایلی کـه تولید می شود به صورت منحصر به فرد باشد. البته می توانید اعداد موجود در این قسمت را خودتان وارد کنید. به این صورت ویـژوال استودیو آنها را هنگام کامپایل تغییر نخواهد داد و این اعداد ثابت می مانند تا مجدداً آنها تغییر دهید.

نکته: توصیه می شود که همواره نسخه های مربوط به برنامه را به صورت دستی تنظیم کنید، به خصوص اگر می خواهید برنامه را توزیع کنید. به این ترتیب می توانید بر شماره ی نسخه ها کنترل کامل داشته باشید و از بروز اشکالاتی که ممکن است با تنظیم اتوماتیک شماره نسخه ی برنامه به وجود آید هم جلوگیری خواهید کرد.

ثبت کردن یک اسمبلی:

هر برنامه ای که تحت .NET نوشته می شود، نمی تواند به تمام کتابخانه های کلاسی که در یک کامپیوتر وجود دارد دسترسی داشته باشد. برای اینکه بتوانید از کلاسهای درون یک کتابخانه ی کلاس در برنامه ی خود استفاده کنید ابتدا باید فایل dll مربوط به آن کتابخانه را در فولدر مربوط به برنامه قرار داده و سپس از آن استفاده کنید. این مورد در رابطه با کتابخانه های کلاسی که در چند برنامه مورد استفاده قرار می گیرند مشکلاتی را ایجاد می کند. برای مثال تصور کنید که می خواهید کتابخانه ی کلاس InternetFavorites.dll را که در قسمت قبل ایجاد کردیم در دو برنامه استفاده کنید. برای این کار مجبور خواهید بود که فایل dll این کتابخانه را در فولدر هر دو برنامه کپی کنید.
در این حالت فرض کنید خطایی در این کتابخانه ایجاد شود و برای تصحیح آن بخواهید آن را تغییر دهید، و یا حتی به هر دلیل دیگری مجبور به تغییر کلاسهای این کتابخانه شوید. به این ترتیب مجبور خواهید بود بعد از اینکه فایل dll جدید را ایجاد کردید، آن را در فولدر تمام برنامه هایی که از این کتابخانه استفاده می کنند قرار دهید که مسلماً این کار منطقی به نظر نمی رسد.
برای رفع این مشکل بهتر است کتابخانه های کلاسی که به صورت عمومی توسط برنامه ها مورد استفاده قرار می گیرند را در فولدری خاص قرار دهیم و سپس تمام برنامه ها بتوانند به کتابخانه های کلاس موجود در آن فولدر دسترسی داشته باشند و از آن استفاده کنند. این فولدر همانطور که در فصل دوم هم مقداری با آن آشنا شدیم، GAC[1] نام دارد. هر برنامه در .NET می تواند علاوه بر کتابخانه های کلاس موجود در فولدر خود برنامه، به کتابخانه های کلاس موجود در GAC نیز دسترسی داشته باشد. همچنین اسمبلی هایی که دارای نامهای یکسان باشند ولی از نظر نسخه و یا نویسنده و یا ... تفاوت داشته باشند نیز در این فولدر به صورت درست نگه داری شده و با یکدیگر اشتباه نخواهند شد. این فولدر در ویندوز XP در آدرس C:\Windows\Assembly و در ویندوز ۲۰۰۰ در آدرس C:\WinNT\Assembly قرار دارد.
اما برای قرار دادن یک کتابخانه ی کلاس در این قسمت، نباید آن را همانند فایل های عادی در این فولدر کپی کرد. بلکه باید از ابزار خاصی به نام gacutil استفاده کرد که نحوه کار آن در قسمت بعدی توضیح داده شده است.

ابزار GacUtil:

Gacutil برنامه ای است که همراه با چارچوب .NET ارائه شده است و برای قرار دادن و یا حذف کردن یک فایل در GAC مورد استفاده قرار می گیرد. این ابزار نیز مانند ابزار sn توسط خط فرمان قابل دسترسی است.
برای کار با این ابزار مجدداً خط فرمان Visual Studio 2005 Command Prompt را با استفاده از منوی Start باز کرده و سپس به فولدر bin در فولدری که برنامه ی InternetFavorites در آن قرار دارد بروید[2]. برای نصب کتابخانه ی کلاس InternetFavorites.dll در GAC دستور زیر را وارد کنید:

```
Gacutil –i InternetFavorites.dll
```

[1] Global Assembly Cache

[2] برای حرکت در بین فولدر ها در خط فرمان می توانید از دستور cd استفاده کنید.

با استفاده از سوییچ I می توانید یک اسمبلی را در GAC قرار دهید. برای حذف اسمبلی نیز می توانید از سوییچ u به صورت زیر استفاده کنید:

```
Gacutil –u InternetFavorites.dll
```

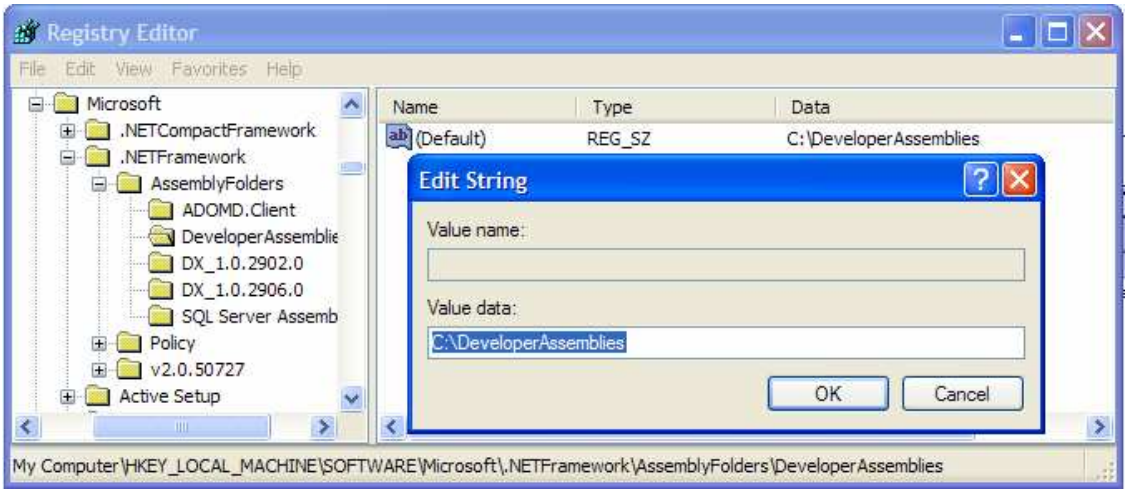
خوب، به این ترتیب کتابخانه ی کلاس InternetFavorites.dll در GAC نصب شده است و می توانید بعد از اضافه کردن ارجاعی از این کتابخانه ی کلاس به برنامه ی خودتان از آن استفاده کنید. همانطور که در قسمتهای قبل مشاهده کردید برای اضافه کردن یک ارجاع به یک کتابخانه ی کلاس باید از قسمت .NET در پنجره ی Add Reference استفاده کنیم (شکل ۳-۱۲). اما نکته ای که در این قسمت وجود دارد این است که فقط با اضافه کردن یک کتابخانه ی کلاس به GAC نمی توان آن را در لیست موجود در پنجره ی Add Reference نیز مشاهده کرد و از آن در برنامه استفاده کرد، بلکه برای نمایش داده شدن فایل مربوط به کتابخانه ی کلاس در این لیست باید آن را در رجیستری ویندوز نیز ثبت کنیم.
دلیل این مورد هم این است که ویژوال استودیو برای گردآوری نام اسمبلی های موجود در این لیست علاوه بر جستجوی GAC، بعضی از کلید های رجیستری را نیز جستجو می کند تا مسیر واقعی اسمبلی های مورد نظر را پیدا کند. پس باید کلیدی را در رجیستری تنظیم کنیم تا ویژوال استودیو بتواند اسمبلی InternetFavorites را نیز ببیند و در این لیست قرار دهد. در بخش امتحان کنید بعد نحوه ی انجام این کار را بررسی خواهیم کرد.

امتحان کنید: قرار دادن نام اسمبلی در لیست موجود در کادر Add Reference

۱) بر روی منوی Start کلیک کرده و گزینه ی Run را انتخاب کنید.
۲) در پنجره ی Run عبارت regedit را وارد کرده و کلید Enterرا فشار دهید تا پنجره ی Registry Editor نمایش داده شود.
۳) در این پنجره با استفاده از قسمت سمت چپ به کلید زیر بروید:

```
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\.NETFramework\AssemblyFolders\
```

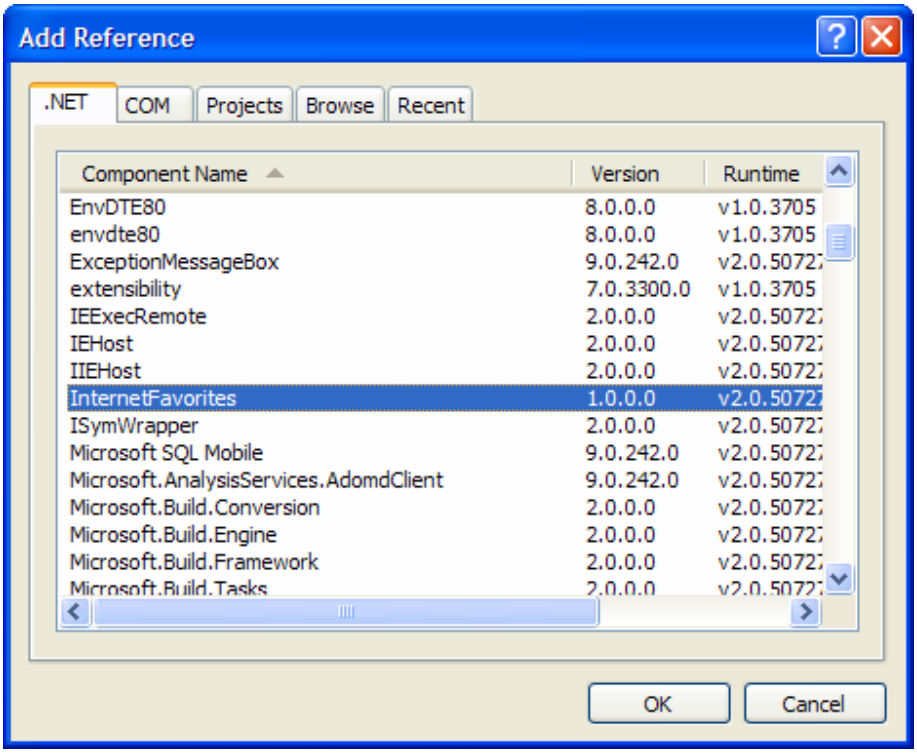
۴) بر روی فولدر AssemblyFolders کلیک راست کنید و گزینه ی New ➔ Key را انتخاب کنید.
۵) کلیدی با نام دلخواه ایجاد کنید. در این قسمت ما نام Developer Assemblies را وارد می کنیم.
۶) در قسمت سمت راست پنجره روی گزینه ی (Default) دو بار کلیک کرده و آدرس یک فولدر را در پنجره ای که باز می شود وارد کنید. در این قسمت ما آدرس C:\Developer Assemblies را وارد کردیم (شکل ۱۲-۶).



شکل ۱۲-۶

۷) حال پنجره ی `Windows Explorer` را باز کرده و آدرسی که در مرحله ی قبل وارد کردید را ایجاد کنید (البته اگر وجود ندارد). سپس فایل `InternetFavorites.dll` را در این آدرس کپی کنید.

۸) برای اینکه تنظیماتی که در این قسمت ایجاد کردیم اثر کنند، مکن است مجبور باشید که ویژوال استودیو را ببندید و مجدداً باز کنید. بعد از این کار بر روی نام پروژه در پنجره ی `Solution Explorer` کلیک کرده و گزینه ی `Add Reference` را انتخاب کنید. به این ترتیب مشاهده خواهید کرد که نام اسمبلی `InternetFavorites` نیز در لیست نمایش داده می شود (شکل ۱۲-۷).



شکل ۱۲-۷

طراحی کتابخانه های کلاس:

تاکنون متوجه شده اید که کتابخانه های کلاس چه فوایدی دارند و چگونه و در چه قسمتهایی مورد استفاده قرار می گیرند. همچنین مفاهیم کلاسها، اشیا و کتابخانه های کلاس را نیز درک کرده اید.
قبل از طراحی یک برنامه باید به دقت بررسی کنید تا متوجه شوید که دقیقاً می خواهید چه چیزی را طراحی کنید. همانند یک معمار که می خواهد یک خانه را طراحی کند، باید ابتدا بدانید که می خواهید هر قسمت از برنامه چه کاری را و به چه نحو انجام دهد تا بتوانید طرح درستی از برنامه را ایجاد کنید.
معمولاً هنگامی که طراحان نرم افزار می خواهند یک برنامه را طراحی کرده، تحلیل کنند و یا یک قالب کلی برای کد های آن برنامه ایجاد کنند از ابزارهایی مانند Microsoft Visio برای ترسیم این موارد استفاده می کنند که معمولاً برای کار با ویژوال استودیو ۲۰۰۵ نیز تنظیم شده اند. Visio دارای انواع مختلفی از سمبل ها است که با استفاده از آنها می توان بر اساس قواعد یکی از مدهای ترسیم طرح برنامه، طرح کلی، فلو چارت و یا انواع دیگر دیاگرام ها را برای یک برنامه ایجاد کرد. یکی از معروفترین مدلهای ترسیم طرح و مدل برای یک برنامه، UML[1] است که بیشترین کاربرد را در طراحی و معماری یک نرم افزار دارد. UML شامل چندین سمبل و علامت است که هر کدام در ترسیم طرح کلی یک برنامه مفهوم و معنی خاصی را می رسانند.
نحوه استفاده از UML در طراحی برنامه ها موردی است که بیشتر در مباحث مهندسی نرم افزار مورد بررسی قرار می گیرد و در این کتاب نمی خواهیم در این مورد صحبت کنیم. اما بهتر است بدانید که هنگام طراحی یک کلاس اگر به سوالاتی مانند "این کلاس یابد دارای چه متد ها و یا چه پارامترهایی باشد؟" و یا "آیا کلاس باید علاوه بر متد دارای خاصیت نیز باشد؟" پاسخ درستی داده نشود ممکن است که باز هم کلاس بتواند مورد استفاده قرار بگیرد، اما مطمئنا به صورت کارآمد عمل نخواهد کرد.
برای مثال تصور کنید که یک کتابخانه ی کلاس شامل ۲۰ کلاس است که هر یک ۴۰متد و یا خاصیت دارند و هر متد نیز حداقل ۱۵ پارامتر دریافت می کند. چنین کتابخانه ی کلاسی مسلماً نمی تواند به سادگی مورد استفاده قرار گیرد. در حقیقت باید بگویم که یک کتابخانه ی کلاس هیچگاه نباید به این صورت طراحی شود.
در عوض هنگام طراحی یک کلاس همواره بهتر است که قانون زیر را رعایت کنید: سادگی. تقریبا می توان گفت که سادگی یکی از مهمترین فاکتور هایی است که در طراحی یک کلاس باید مد نظر قرار داده شود. برای مثال همواره استفاده از چند کلاس کوچک که هر یک وظیفه ی خاصی را انجام می دهند بسیار بهتر از استفاده از یک کلاس بزرگ است که همه ی کارها را انجام می دهد.
هنگامی که در حال کار با یک کلاس بزرگ و بسیار پیچیده در یک سیستم نرم افزاری هستید، درک کد درون چنین برنامه ای بسیار پیچیده و مشکل خواهد بود و اغلب خطا یابی و تغییر چنین کد هایی به کابوس شبیه می شود. در بیشتر مواقع در چنین شرایطی نیازی نیست که کلاسهای مورد استفاده در برنامه به این صورت طراحی شوند. بلکه می توان با تقسیم این کلاسها به کلاسهای کوچکتر، هم نگه داری و خطا یابی را در آنها ساده تر کرد و هم در مواقع مورد نیاز از این کلاسها در برنامه های دیگر نیز استفاده کرد.
در طراحی یک کلاس همواره سعی کنید به نحوی عمل کنید که استفاده کنند گان از آن بتوانند به راحتی و بدون نیاز به مطالعه ی مقدار زیادی مستندات، آن کلاس را مورد استفاده قرار دهند. برای این کار باید به موارد زیر همواره به صورت یک قانون عمل کنید:

- سعی کنید که تعداد پارامترهای متد ها را حداکثر پنج و یا شش پارامتر قرار دهید، مگر در شرایطی که واقعاً به پارامترهای بیشتری نیاز داشته باشید. به این ترتیب استفاده از یک کلاس بسیار ساده تر خواهد بود.

[1] Unified Modeling Language

- مطمئن شوید که متد و تمام پارامترهای آن دارای نام معنی داری هستند. همواره بهتر است از اسامی ای که به راحتی خوانده می شوند به جای اسامی کوتاه استفاده کنید. برای مثال استفاده از `stdNo` به جای `StudentNumber` روش مناسبی نیست.
- هیچ گاه لازم نیست که در یک کلاس از تمام توابع و خاصیتهای ممکن استفاده کنید. به این ترتیب موجب می شوید که کاربر هنگام استفاده از آن کلاس برای کار با یک متد با انتخاب های زیادی روبرو باشد که این امر استفاده از کلاس را برای کاربر مشکل تر و پیچیده تر می کند. حتی ممکن است بسیاری از این کلاسها اصلاً مورد استفاده ی عمومی قرار نگیرند و وجود آنها در کلاس بیهوده باشد.
- سعی کنید در یک کتابخانه حداقل تعداد کلاس را قرار دهید. زیرا به این ترتیب استفاده از آن کتابخانه و درک عملکرد کلاسهای آن بسیار ساده تر خواهد بود.
- استفاده از خاصیت ها در یک کلاس بسیار پر کاربرد است و موجب راحتی استفاده از آن کلاس می شود.

استفاده از یک کتابخانه ی کلاس شخص ثالث[1]:

مشاهده کردید که یک کتابخانه ی کلاس به یک فایل `dll` کامپایل می شود. برای استفاده از کلاسهای موجود در یک کتابخانه ی کلاس فقط به این فایل `dll` نیاز است و لازم نیست که کاربر به سورس اصلی برنامه دسترسی داشته باشد. به این ترتیب می توانید کتابخانه ی کلاس خود را در یک فایل `Dll` کامپایل کرده و آن را برای استفاده توسط افراد دیگر توزیع کنید و یا فایلهای `dll` که حاوی کتابخانه های کلاس هستند را دریافت کرده و از آن در برنامه های خود استفاده کنید. با نحوه ی ایجاد یک کتابخانه ی کلاس در قسمت قبل آشنا شدیم و یک کتابخانه ی کلاس به نام `InternetFavorites.dll` نیز طراحی کردیم. در این قسمت با نحوه ی استفاده از این فایلهای `dll` در برنامه آشنا خواهیم شد و سعی خواهیم کرد که از فایلی که در قسمت قبل ایجاد کردیم در یک برنامه استفاده کنیم.

استفاده از فایل InetrnetFavorites.dll:

در قسمتهای قبل نحوه ی ایجاد یک ارجاع به یک پروژه ی کتابخانه ی کلاس را در یک برنامه مشاهده کردیم. این مورد به خصوص در مواردی که بخواهیم یک کتابخانه ی کلاس را در یک برنامه تست کنیم بسیار مورد استفاده قرار می گیرد. اما در این مثال می خواهیم وانمود کنیم که کتابخانه ی کلاس `InternetFavorites.dll` توسط ما طراحی نشده است، بلکه فایل آن را از فرد دیگری دریافت کرده ایم و حال می خواهیم برنامه ی `Favorites Tray` را به گونه ای تغییر دهیم تا از این فایل استفاده کند. این یک روش بسیار ساده و سریع برای نمایش نحوه ی استفاده از یک فایل `dll` در برنامه است. اما به خاطر داشته باشید که در برنامه های واقعی باید ابتدا ارجاعی از یک کتابخانه ی کلاس به برنامه اضافه کرده و سپس از آن استفاده کنید.

امتحان کنید: استفاده از `InternetFavorites.dll` در برنامه ی `Favorites Tray`

[1] کتابخانه ی کلاسی که توسط افراد و گروهای برنامه نویسی دیگری نوشته شده اند و به صورت جداگانه در بازار به فروش می رسند.

۱) پروژه ی Favorites Tray را در محیط ویژوال استودیو باز کنید.
۲) فایل های WebFavorite.cs و Favorites.cs را از پروژه حذف کنید.
۳) حال باید ارجاعی را به فایل InternetFavorites.dll در برنامه اضافه کنیم. برای این کار در پنجره ی Solution Explorer روی نام پروژه ی Favorites Tray کلیک راست کرده و از منوی باز شده گزینه ی Add Reference را انتخاب کنید. در قسمت .NET در این پنجره، لیست را حرکت داده تا کتابخانه ی کلاس InternetFavorites.dll را پیدا کنید. سپس آن را انتخاب کرده و روی دکمه ی OK کلیک کرده تا پنجره ی Add Reference بسته شود.
۴) همانطور که به خاطر دارید، فضای نام مورد استفاده در این کتابخانه ی کلاس InternetFavorites است. بنابراین باید همانند دیگر فضای نامها، این فضای نام را نیز به برنامه ی خود اضافه کنید. برای این کار با استفاده از راهنمای using مانند زیر، این فضای نام را نیز به برنامه ی خود اضافه کنید.

```
using InternetFavorites;
```

۵) برنامه را اجرا کنید. مشاهده خواهید کرد که برنامه همانند قبل به درستی کار می کند، اما این بار به جای استفاده از کلاسهای موجود در فایل اجرایی برنامه، از کلاسهای موجود در فایل dll استفاده می کند.

چگونه کار می کند؟

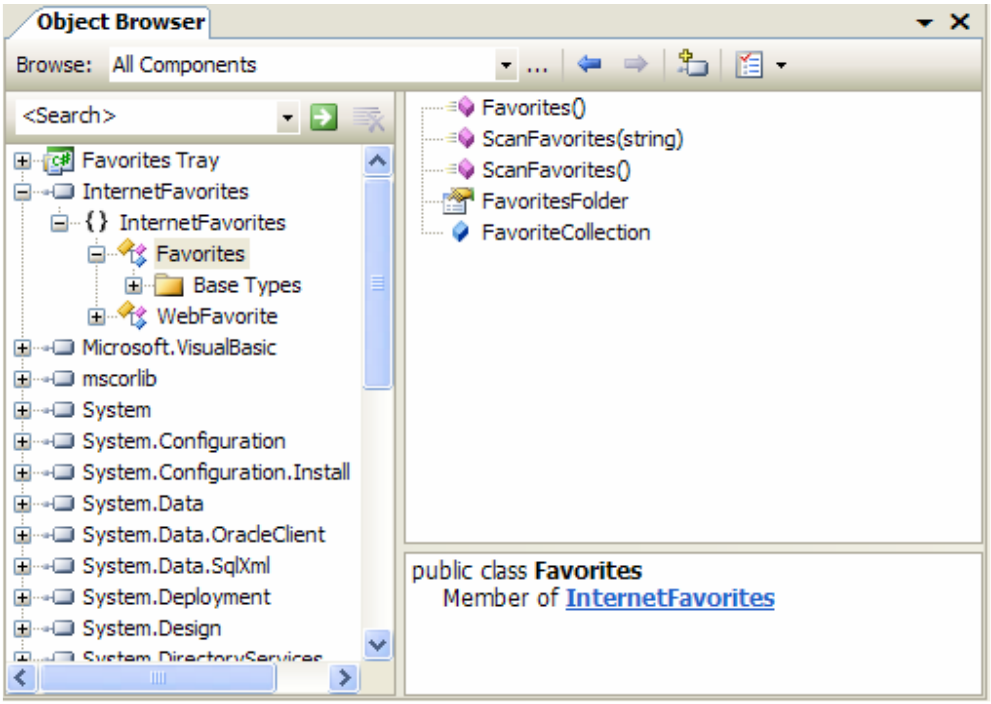
همانطور که مشاهده کردی اضافه کردن یک ارجاع به فایل dll حتی از اضافه کردن یک ارجاع به یک پروژه ی کتابخانه ی کلاس نیز ساده تر بود. همچنین دیدید که چه یک کتابخانه ی کلاس به صورت یک فایل dll به برنامه اضافه شود و چه به صورت یک پروژه، در هر دو حالت به یک صورت مورد استفاده قرار می گیرد. تنها تفاوت در این روش نسبت به استفاده از کتابخانه ی کلاس به صورت قبلی در این است که در این حالت نمی توانید به کد برنامه دسترسی داشته باشید.
البته با وجود اینکه نمی توانید به کد یک کتابخانه ی کلاس دسترسی داشته باشید، باز هم می توانید اطلاعات زیادی را در رابطه با کلاس های موجود در آن به دست آورید. برای مثال می توانید بفهمید که هر کلاس دارای چند متد، خاصیت، فیلد و یا ... است و یا اینکه هر متد چند پارامتر و از چه نوع هایی دریافت می کند. برای فهمیدن این موارد در مورد یک کتابخانه ی کلاس می توانید از ابزار Object Browser استفاده کنید.

استفاده از Object Browser:

برای مشاهده ی کلاس هایی که در ویژوال استودیو به کار رفته اند می توانید از ابزاری سریع و ساده به نام Object Browser استفاده کنید. با استفاده از این ابزار می توانید کلاسها و همچنین متد ها و خاصیت هایی که درون هر کلاس به کار رفته اند را مشاهده کنید و اطلاعات لازم در مورد آنها را نیز بدست آورید. برای مشاهده ی پنجره ی Object Browser در ویژوال استودیو کافی است کلید F2 را فشار دهید. البته می توانید از گزینه ی View ➔ Object Browser و یا آیکون Object Browser در نوار ابزار نیز برای نمایش این پنجره استفاده کنید.

این ابزار عموماً برای دریافت اطلاعات مختصر و سریع در مورد کلاسهای مورد استفاده در یک برنامه به کار مـی رود. همـانطور کـه مشاهده می کنید تمام اسمبلی ها و فضای نامهایی که در یک برنامه به کار رفته اند در پنجره ی Object Browser بـه صورت یک لیست نمایش داده شده اند.

در این پنجره همانطور که مشاهده می کنید تمام اعضای یک کلاس از قبیل متد ها، شمارنده ها، ثابت ها و ... نمایش داده می شوند و برای هر عضو نیز بر اساس نوع آن آیکون خاصی نمایش داده می شود. در شکل ۱۲-۸ کلاس Favorites و اعـضای آن نمایش داده شده اند. برای نمایش این کلاس ابتدا باید اسمبلی InternetFavorites را انتخاب کرده و سپس در این اسمبلی فضای نام InternetFavorites را انتخاب کنید. در انتها نیز روی کلاس Favorites در این فضای نام کلیک کنید.



شکل ۱۲-۸

نکته: دقت کنید که در یک اسمبلی می تواند بیش از یک فضای نام وجود داشته باشد و یک فضای نام نیز می تواند به بیش از یک اسمبلی تقسیم شود.

کتابخانه ی MSDN دارای مستندات و اطلاعات بسیار زیادی در مورد کلاسهای نصب شده همراه با ویـژوال اسـتودیو در چـارچوب .NET است. بنابراین اگر در برنامه ی خود بیشتر از کلاسهای درونی .NET استفاده می کنید، کمتر بـه کـار بـا Object Browser نیاز خواهید داشت. این ابزار بیشتر هنگامی کارآمد خواهد بود که بخواهید از یـک کتابخانـه ی کـلاس نوشـته شـده توسط شخص ثالث استفاده کنید که راهنمایی هم برای استفاده از آن کتابخانه ی کلاس در دسترس ندارید. در این موارد می توانیـد با مشاهده ی نام متد ها و خاصیت ها عملکرد آنها را متوجه شوید. البته این شرایط وقتی صادق خواهنـد بـود کـه برنامـه نـویس آن کتابخانه ی کلاس نام مناسبی برای متد ها و خاصیت های آن کلاس انتخاب کرده باشد.

در بعضی شرایط یک فایل DLL می تواند در مورد کلاسهای موجود در خود و نیز هر یک از متد ها و خاصیت های آن کلاس توضیح مختصری ارائه دهد. در این حال باید هنگام طراحی کلاس برای مشخص کردن توضیحات مربوط به هر عضو از Attribute ها استفاده کرد که صحبت در رابطه با این موضوع از مباحث این کتاب خارج است.

نتیجه:

کتابخانه های کلاس بخشی جدا نشدنی در ویژوال استودیو ۲۰۰۵ به شمار می روند و در حقیقت از اهمیت زیادی در تمامی زبانهای چارچوب .NET برخوردار هستند. با استفاده از آنها می توانید به راحتی کلاسهای مورد استفاده در برنامه های خود را دسته بندی کرده و در برنامه های دیگر نیز از آنها استفاده کنید.
در این فصل با نحوه ی ایجاد کتابخانه های کلاس و نیز نحوه ی استفاده از کتابخانه های کلاس ایجاد شده توسط افراد دیگر آشنا شدیم. همچنین مشاهده کردیم که چگونه می توان یک اسمبلی را به صورت قوی نامگذاری کرد و به این ترتیب از قرار گرفتن در جهنم DLLها جلوگیری کرد.
در فصل سیزدهم مشاهده خواهید کرد که چگونه می توان کنترل هایی را طراحی کرد که بتوانند در برنامه های ویندوزی و فرمهای این برنامه مورد استفاده قرار گیرند. به عبارت دیگر چگونه می توان به جای طراحی کلاس هایی که فقط شامل کد هایی برای انجام یک سری امور خاص در پشت برنامه می باشند، کلاس هایی را طراحی کرد که دارای رابط کاربری مشخصی باشند و بتوانند در فرم های ویندوزی مورد استفاده قرار بگیرند. در آن فصل نیز مجدداً با اهمیت استفاده ی مجدد از کد آشنا خواهید شد.

تمرین:

برنامه ی Favorites Viewer را به صورتی تغییر دهید که به جای استفاده از پروژه ی InternetFavorites از فایل dll کامپایل شده برای این کتابخانه ی کلاس استفاده کند.

فصل سیزدهم: ایجاد کنترلهای سفارشی

تاکنون در برنامه هایی که در طول این کتاب ایجاد کرده ایم، از کنترلهای زیادی که همراه با .NET ارائه شده بودند استفاده کردیم، از کنترل Button گرفته تا کنترل های TextBox و ListBox. حتی ممکن است سعی کرده باشید که از کنترلهای پیشرفته تر دیگری مانند DataGrid و TreeView در برنامه های خود استفاده کنید و با آنها آشنا شوید. البته کار با این کنترل ها در ابتدا کمی مشکل به نظر می رسد، به علت اینکه معمولاً دارای خاصیت ها و متدهای زیادی هستند. با استفاده از این کنترل ها می توانید رابط های کاربری بهتری و قوی تری ایجاد کنید. به عبارت دیگر هر چه بیشتر در استفاده از این کنترل ها مهارت داشته باشید و بیشتر با نحوه ی کاربرد آنها آشنا باشید، به سرعت می توانید رابط های کاربری کارآمدی را ایجاد کنید. یکی دیگر از جنبه های مهم این کنترل ها قابلیت استفاده ی مجدد از آنها است. هر بار که برنامه ی ویندوزی جدیدی ایجاد کنید می توانید به راحتی یک کنترل Button را از جعبه ابزار در فرم برنامه قرار دهید و سپس این کنترل مانند کنترل Button در برنامه های دیگر کار می کند. استفاده ی مجدد از کنترل ها و سادگی انجام آن در ویژوال C# (و کلاً در زبانهای تحت .NET) یکی از مهمترین فاکتور های موفقیت این زبان برنامه نویسی و این محیط طراحی و توسعه به شمار می رود.
در این فصل:

- در مورد کنترلهای ویندوزی و نحوه ی کارکرد آنها در برنامه ها مطالبی را خواهید آموخت.
- با ایجاد و استفاده از کنترلهای ویندوزی آشنا خواهید شد.
- با نحوه ی اضافه کردن متد و رویداد به این کنترل ها آشنا خواهید شد.
- مشاهده خواهید کرد که چگونه می توان با استفاده از کد، نحوه ی عملکرد این کنترل ها را در زمان طراحی و زمان اجرا تعیین کرد.

نکته: کنترل هایی که در این فصل ایجاد خواهند شد برای استفاده در برنامه های تحت ویندوز مناسب هستند نه برنامه های تحت وب. برای نحوه ی ایجاد کنترلهای سفارشی تحت وب به فصل هجدهم مراجعه کنید. در این فصل فقط روی کنترل های تحت ویندوز تمرکز خواهیم کرد.

کنترلهای ویندوزی:

ممکن است ابتدا از خود سوال کنید ایجاد کنترلهای سفارشی چه دلیلی ممکن است داشته باشد. خوب، دلایل زیادی برای ایجاد کنترلهای سفارشی تحت ویندوز وجود دارند:

- با استفاده از کنترلهای سفارشی می توانید از یک کنترل در چند قسمت برنامه و یا حتی در چندین برنامه ی مختلف استفاده کنید، بنابراین مقدار کد مورد نیاز به شدت کاهش می یابد (استفاده ی مجدد از کد).
- می توانید کد های مربوط به یک کنترل را در کلاس همان کنترل قرار دهید و به این ترتیب باعث شوید که کد برنامه بسیار واضح تر شود و ساده تر بتوان آن را درک کرد. برای مثال می توانید کنترل Button را به صورتی ایجاد کنید که بتواند رویداد Click خود را کنترل کند، به این ترتیب دیگر نیازی ندارید این رویداد را در کد مربوط به فرم برنامه کنترل کنید.

برای استفاده مجدد از کنترل ها در برنامه ها دو روش کلی وجود دارند. روش اول این است که سورس اصلی کنترل را به هر برنامه ای می خواهید از کنترل در آن استفاده کنید اضافه کرده و سپس برنامه را کامپایل کنید. به این ترتیب کد مربوط به کنترل در فایل اجرایی برنامه قرار می گیرد. در طی این فصل به علت سادگی این روش از آن استفاده خواهیم کرد تا بتوانیم بیشتر تمرکز خود را بر نحوه ی عملکرد کنترل ها متمرکز کنیم.
روش دوم ایجاد کتابخانه ی کنترل است. کتابخانه های کنترل همانند کتابخانه های کلاس هستند که در فصل قبل با آنها آشنا شدیم. در حقیقت کتابخانه های کنترل، کتابخانه های کلاسی هستند که دارای یک رابط کاربری نیز می باشند. همانند کتابخانه های کلاس، کتابخانه ای کنترل نیز در فایل اسمبلی جداگانه ی مربوط خودشان قرار می گیرند و می توانید بعد از ثبت آنها در GAC با استفاده از روشهایی که در فصل قبل مشاهده کردید، از آنها در چندین برنامه استفاده کنید. این روش بسیار جالبتر از روش قبلی است، زیرا به این ترتیب می توانید فایل DLL مربوط به این کنترل ها را در اختیار دیگر برنامه نویسان قرار دهید تا بتوانند از آن در برنامه های خود استفاده کنند. می توانید به راحتی اسمبلی مربوط به این برنامه ها را تغییر دهید و به شکل جدیدی تبدیل کنید. به این ترتیب تمام برنامه ها این تغییرات را دریافت کرده و با فایل جدید کار می کنند، بدون اینکه نیاز باشد مجدداً آنها را کامپایل کنید. تکنیک هایی که برای طراحی کنترل به کار می رود در هر دو مورد مشابه است، چه کنترل را به صورت یک پروژه ی مجزا ایجاد کرده و از آن در برنامه استفاده کنیم، چه کنترل را در برنامه طراحی کرده و در همان قسمت نیز از آن استفاده کنیم.

ایجاد و تست کردن کنترل های سفارشی:

هنگام طراحی یک برنامه ممکن است به کنترلی نیاز داشته باشید که به یک بانک اطلاعاتی وصل شود و اطلاعات خاصی مانند نام کاربری و کلمه ی عبور یک کاربر را استخراج کند. اگر بخواهید یک کنترل قوی و کارآمد برای این کار ایجاد کنید، باید به نحوی آن را طراحی کنید که آن کنترل در بیشتر موارد قابل استفاده باشد و همچنین کاربر بتواند به راحتی آن را تنظیم کند تا وظیفه ی مدنظر او را انجام دهد. در این موارد بهتر است اموری را مانند متصل شدن به بانک اطلاعاتی، دریافت نتایج مورد نیاز و قرار دادن نتایج بدست آمده در کنترل های دیگر را دور از چشم کاربر انجام دهید، به صورتی که کاربرانی که از کنترل شما استفاده می کنند در رابطه با این موارد هیچ اطلاعی نداشته باشند. به این ترتیب می توانید از درگیر شدن آنها در این گونه موارد جلوگیری کنید و به آنها اجازه دهید که روی مسائل و وظایف مربوط به خودشان تمرکز کنند.
طراحی یک کنترل سفارشی از پایه، کار سختی نیست. از جهتی انجام چنین کاری مشابه طراحی یک فرم در برنامه های ویندوزی است. در این قسمت می خواهیم یک برنامه ی تحت ویندوز ایجاد کنیم که از یک کنترل سفارشی استفاده می کند. در بخش امتحان کنید بعد، ابتدا یک کنترل سفارشی ایجاد خواهیم کرد که شامل سه کنترل Button باشد و با فشار هر یک از این دکمه ها یک پیغام در صفحه نمایش داده شود. سپس نحوه ی استفاده از این کنترل در برنامه را مشاهده خواهیم کرد.

نکته: کنترلهای سفارشی ای که در طراحی آنها از چندین کنترل دیگر استفاده شده است عموماً به عنوان **کنترل های متراکم**[1] شناخته می شوند.

امتحان کنید: ایجاد اولین کنترل سفارشی

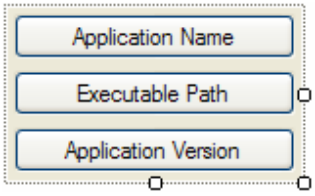
۱) برنامه ی ویژوال استودیو ۲۰۰۵ را باز کرده با استفاده از نوا منو گزینه ی File ➔ New ➔ Project... انتخاب کنید تا کادر New Project نمایش داده شود. در قسمت Project Type

[1] Aggregate Controls

این کادر، گزینه ی Visual C# را انتخاب کرده و از قسمت Templates گزینه ی Windows Control Library را انتخاب کنید. در قسمت Name عبارت MyNamespaceControl را وارد کرده و سپس روی دکمه ی OK کلیک کنید.

۲) حال روی UserControl1.cs در پنجره ی Solution Explorer کلیک کرده و با استفاده از پنجره ی Properties خاصیت File Name آن را به MyNamespace.cs تغییر دهید. مشاهده می کنید که محیطی مشابه محیط طراحی فرم در این قسمت نیز به چشم می خورد، ولی در این محیط بخشهایی مانند نوار عنوان و یا حاشیه های فرم وجود ندارد. عمدتاً هنگام طراحی یک کنترل، کنترل های موجود را در این قسمت قرار می دهیم و سپس کد هایی که قرار است در چندین قسمت از برنامه مورد استفاده قرار داده شوند را در این کنترل ها وارد می کنیم.

۳) با استفاده از جعبه ابزار سه کنترل Button روی فرم قرار داده و خاصیت Text آنها را به صورتی تنظیم کنید که مشابه شکل ۱۳-۱ شوند. همچنین با تنظیم اندازه ی هر یک از این کنترل ها، فرم خود را مشابه فرم شکل ۱۳-۱ ایجاد کنید.



شکل ۱۳-۱

۴) خاصیت Name این کنترل ها را به ترتیب برابر با btnApplicatinName، btnExecutablePath و btnApplicationVersion قرار دهید.

۵) تا اینجا با کلیک کردن روی Buttonهایی که در این قسمت ایجاد کرده ایم هیچ اتفاق خاصی رخ نمی دهد – پس نیاز داریم که متدی را ایجاد کرده و هنگام رخ دادن رویداد کلیک Button، آن متد را فراخوانی کنیم. روی کنترل btnApplicationName دو بار کلیک کنید و کد مشخص شده در زیر را در متد ایجاد شده وارد کنید.

```
private void btnApplicationName_Click(object sender,
                                      EventArgs e)
{
    MessageBox.Show("Application Name is: " +
                    Application.ProductName);
}
```

۶) مجدداً به قسمت طراحی کنترل برگشته و روی کنترل btnExecutablePath دو بار کلیک کنید تا متد مربوط به رویداد Click ایجاد شود. سپس کد مشخص شده در زیر را در آن متد وارد کنید.

```
private void btnExecutablePath_Click(object sender,
                                     EventArgs e)
{
    MessageBox.Show("Executable Path is: " +
                    Application.ExecutablePath);
```

```
}
```

۷) در آخر نیز مجدداً به قسمت طراحی فرم برگردید و روی کنترل btnApplicationVersion دو بار کلیک کنید تا متد مربوط به رویداد کلیک آن ایجاد شود. سپس کد مشخص شده در زیر را در آن وارد کنید:

```
private void btnApplicationVersion_Click(object sender,
                                                EventArgs e)
{
     MessageBox.Show("Application Version is: " +
                                Application.ProductVersion);
}
```

۸) حال برنامه را اجرا کنید. کنترلی که طراحی کرده اید همانند شکل ۱۳-۲ در کادر TestContainer نمایش داده می شود. با استفاده از این کادر می توانید نحوه ی عملکرد کنترل خود را بررسی کنید. برای مثال با کلیک روی هر یک از دکمه های موجود مشاهده خواهید کرد که متن مناسبی در یک کادر پیغام نمایش داده می شود. بعد از اتمام کار با این پنجره روی دکمه ی Close کلیک کنید تا بسته شود.

چگونه کار می کند؟

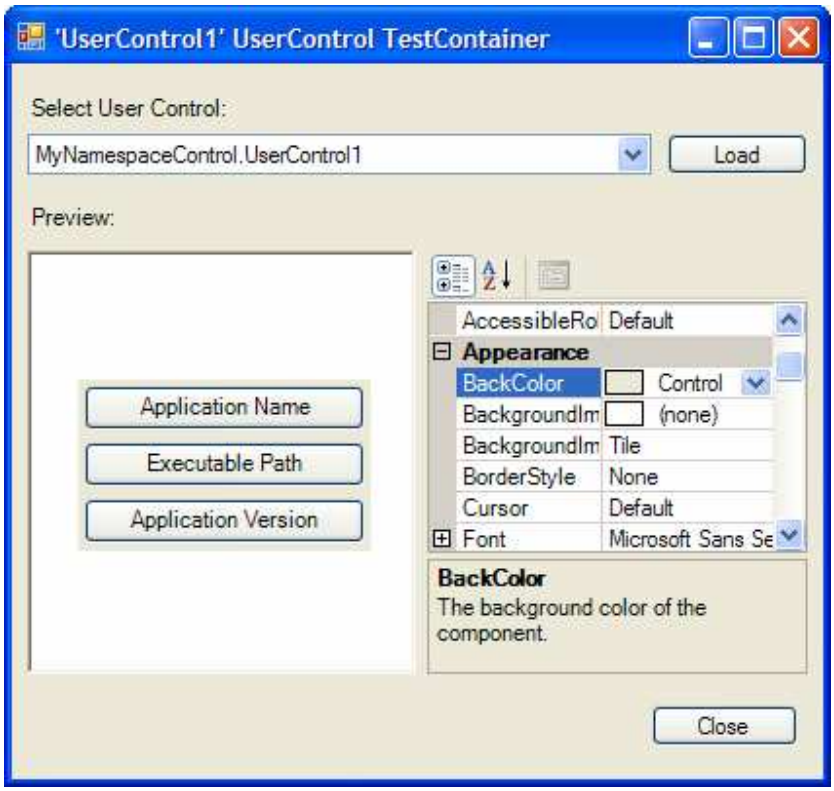
همانطور که مشاهده کردید ایجاد رابط کاربری برای یک کنترل تفاوت زیادی با ایجاد رابط کاربری در یک برنامه ی ویندوزی ندارد. کافی است کنترلهای مورد نیاز خود را با استفاده از جعبه ی ابزار در محیط طراحی کنترل قرار دهیم. سپس کد لازم برای عملکرد این کنترل ها را همانند روشهایی که در طراحی برنامه های ویندوزی استفاده می کردیم در متد های مربوط به هر کنترل وارد می کنیم. کدی که در متد مربوط به رویداد Click کنترل btnApplicationName وارد کردیم با استفاده از توابع استاتیک موجود در کلاس Application، نام برنامه ای که در حال اجرا است را در یک کادر پیغام نمایش می دهد.

```
private void btnApplicationName_Click(object sender,
                                             EventArgs e)
{
     MessageBox.Show("Application Name is: " +
                           Application.ProductName);
}
```

کد وارد شده در متد مربوط به رویداد Click کنترل btnExecutablePath نیز عملکردی مشابه دارد، با این تفاوت که در این قسمت با استفاده از خاصیت ExecutablePath از کلاس Application مسیر برنامه ی در حال اجرا را بدست آورده و آن را در یک کادر پیغام نمایش می دهیم.

```
private void btnExecutablePath_Click(object sender,
                                             EventArgs e)
{
     MessageBox.Show("Executable Path is: " +
                                Application.ExecutablePath);
```

```
}
```



شکل ۱۳-۲

در انتها نیز کد مورد نیاز برای متد مربوط بـه رویـداد `Click` کنتـرل `btnApplicationVersion` را وارد مـی کنیم تا نسخه ی برنامه ی مورد استفاده را نمایش دهد.

```
private void btnApplicationVersion_Click(object sender,
                                         EventArgs e)
{
     MessageBox.Show("Application Version is: " +
                              Application.ProductVersion);
}
```

هنگامی که برنامـه را کامپایـل کنیـد، ایـن کنتـرل بـه صـورت اتوماتیـک بـه قـسمت `MyNamespace Control` در جعبه ابزار اضافه خواهد شد. به این ترتیب می توانید این کنترل را مانند هر کنترل دیگری در برنامـه هـای ویندوزی خود مورد استفاده قرار دهید. البته تا زمانی که یک برنامه ی ویندوزی به راه حل موجود در این قسمت اضـافه نکنیـد ایـن کنترل در جعبه ابزار دیده نخواهد شد.

برای بررسی عملکرد این کنترل فقط کار کردن با آن در پنجره ی `TestContainer` کافی نیست، بلکه بهتر است کنتـرل ایجاد شده را در یک فرم ویندوزی قرار دهیم که این کار را نیز در بخش امتحان کنید بعد انجام خواهیم داد.

امتحان کنید: اضافه کردن کنترل سفارشی ایجاد شده به فرم برنامه

۱) روی منوی File کلیک کنید و گزینه ی Add ➔ New Project را انتخاب کنید.
۲) در پنجره ی Add New Project مطمئن شوید کـه گزینـه ی Windows Application در قسمت Templates انتخاب شده اسـت. سـپس عبـارت Controls را در فیلـد Name وارد کـرده و روی دکمه ی OK کلیک کنید.
۳) در جعبه ابزار قسمت MyNamespaceControl Components را انتخـاب کـرده و روی گزینـه ی UserControl1 دو بار کلیک کنید تا یک نمونه از این کنترل در Form1 قرار داده شود.
۴) روی پروژه ی Controls در پنجره ی Solution Explorer کلیک راست کرده و عبارت گزینه ی Set as Startup Project را از منوی باز شده انتخاب کنید.
۵) حال برنامه را اجرا کنید. مشاهده می کنید که سه دکمه ای که در کنترل سفارشی قرار داده بودیم در این قـسمت نمـایش داده می شـوند و بـا کلیـک روی هـر کـدام از آنهـا کـادر پیغـامی مـشابه کـادر پیغـام نمـایش داده شـده در پنجـره ی TestContainer دیده خواهد شد.

چگونه کار می کند؟

کنترل های سفارشی که توسط برنامه نویس ایجاد می شوند همانند کنترل هایی که تاکنون از آنها در برنامه ها استفاده کرده بـودیم کار می کنند. برای استفاده از این کنترل ها کافی است آنها را از جعبه ابزار انتخاب کرده، بر روی فرم قـرار دهیـد و سـپس برنامـه را اجرا کنید. لازم نیست هیچ کدی را برای رویداد Click دکمه های موجود در این کنترل بنویسید، زیرا کد لازم برای عملکرد این دکمه ها در خود کنترل قرار داده شده است.

ایجاد کردن خاصیت برای کنترل های سفارشی:

یک کنترل سفارشی همانند یک کلاس ایجاد می شود. بنابراین می توانید تمام عضو هایی را که برای یک کلاس ایجاد می کردیـد برای یک کنترل سفارشی نیز ایجاد کنید. به عبارت دیگر می توانید خاصیت ها، متد ها و یا رویداد هایی را به یک کنتـرل سفارشـی اضافه کنید تا افرادی که آن کنترل را در برنامه ی خود به کار می برند بتوانند از آنها استفاده کنند. ابتدا نحوه ی اضـافه کـردن یـک خاصیت را به کنترل سفارشی مشاهده خواهیم کرد.
کنترلهای سفارشی می توانند دارای دو نوع خاصیت باشند: خاصیت هایی که می توانند در زمـان طراحـی و بـا اسـتفاده از پنجـره ی Properties تغییر داده شوند و خاصیت هایی که باید با استفاده از کد نویسی و در زمان اجرا تغییر داده شوند[1]. بـرای مثـال

[1] به خاطر دارید که هنگام طراحی یک برنامه ی ویندوزی، برنامه را به دو زمان مختلف تقسیم می کردیم: زمان طراحی و زمان اجـرا. زمـان طراحـی بـه زمـانی اطلاق می شد که در حل طراحی رابط کاربری برنامه و یا کد نویسی بودیم وم زمان اجرا نیز به زمانی گفته می شد که برنامه در حال اجرا بـود. امـا پـروژه هـای مربوط به کنترلهای سفارشی به سه بازه ی زمانی تقسیم می شوند: زمانی که در حال طراحی و کد نویسی خود کنترل هستیم، زمانی که کار طراحـی کنتـرل بـه پایان رسیده است و در حال استفاده از کنترل طراحی شده در یک برنامه هستیم ولی خود برنامه در حالت طراحی است، زمانی که برنامه ای کـه از کنتـرل در آن استفاده کرده ایم در حال اجرا است.
در این قسمت منظور از زمان طراحی، بازه ی زمانی دوم یعنی زمانی که کنترل را طراحی کرده و در حال استفاده از آن در یک برنامه هستیم می باشـد و منظـور از زمان اجرا، بازه ی زمانی سوم یعنی زمانی که برنامه ای که از کنترل در آن استفاده کرده ایم در حال اجرا است می باشد.

ممکن است بخواهید در زمان طراحی با استفاده از خاصیتهای یک کنترل، رنگ مورد استفاده در آن و یا فونت نوشته های آن را تغییر دهید. اما بخواهید در زمان اجرای برنامه با استفاده از خاصیت ها برای مثال به آدرس بانک اطلاعاتی که به آن متصل شده اید دسترسی داشته باشید.

اضافه کردن خاصیت ها:

در بخش امتحان کنید بعد، خاصیتی به کنترل اضافه خواهیم کرد تا بتوانید آن را چه در زمان اجرا و چه در زمان طراحی تغییر دهید. نام این خاصیت `ApplicationName` است و نام برنامه را نگهداری می کند. هنگامی که این خاصیت در برنامه تغییر کرد، متن موجود در نوار عنوان کادرهای پیغام را تغییر خواهیم داد.

امتحان کنید: اضافه کردن خاصیت جدید به کنترل `MyNamespace`

۱) برای اینکه بتوانید یک خاصیت را به کنترل اضافه کنید، ابتدا باید یک فیلد در کلاس مربوط به آن کنترل ایجاد کنید تا مقدار آن خاصیت را نگهداری کند. به قسمت ویرایشگر کد مربوط به فضای نام `MyNamespace` بروید و کد مشخص شده در زیر را در ابتدای کلاس `UserControl1` اضافه کنید:

```
public partial class UserControl1 : UserControl
{
    // Private members
    private string strApplicationName = "";
```

۲) بعد از ایجاد این فیلد، باید یک خاصیت ایجاد کنید که مقدار بتوان به وسیله ی آن مقدار موجود در این فیلد را تغییر داد. برای این کار کد زیر را بعد از کدی که در قسمت اول وارد کردید بنویسید:

```
    public string ApplicationName
    {
        get
        {
            return strApplicationName;
        }
        set
        {
            strApplicationName = value;
        }
    }
```

۳) برای اینکه مقدار موجود در این خاصیت در نوار عنوان کادرهای پیغام نیز نمایش داده شود، باید پارامتر `Caption` از متد `Show` را با مقدار این خاصیت تنظیم کنیم. برای این کار کد موجود در رویداد کلیک هر سه کنترل `Button` موجود در فرم را به صورت زیر تغییر دهید:

```
private void btnApplicationName_Click(object sender,
                                      EventArgs e)
{
     MessageBox.Show("Application Name is: " +
                     Application.ProductName,
                     this.ApplicationName);
}

private void btnExecutablePath_Click(object sender,
                                      EventArgs e)
{
     MessageBox.Show("Executable Path is: " +
                     Application.ExecutablePath,
                     this.ApplicationName);
}

private void btnApplicationVersion_Click(object sender,
                                      EventArgs e)
{
     MessageBox.Show("Application Version is: " +
                     Application.ProductVersion,
                     this.ApplicationName);
}
```

۴) برای اینکه بتوانید از خاصیت اضافه شده در کنترل استفاده کنید، کافی است یک بار پروژه ی مربوط به کنترل را کامپایل کنید. روی نام پروژه ی MyNamespace Control در پنجره ی Solution Explorer کلیک راست کرده و گزینه ی Build را انتخاب کنید تا برنامه مجدداً کامپایل شود. به این ترتیب خاصیت ای که در این قسمت اضافه کردید قابل استفاده خواهد بود.

۵) به قسمت طراحی فرم مربوط به Form1 برگشته و کنترل UserControl11 را در فرم انتخاب کنید. مشاهده خواهید کرد که خاصیت ApplicationName در قسمت Misc در پنجره ی Properties نمایش داده می شود (اگر خاصیت ها را بر اساس حروف الفبایی مرتب کرده باشید، این خاصیت در مکان مناسب خود نمایش داده می شود).

۶) مقدار این خاصیت را برابر با My Windows Application قرار دهید.

۷) حال برنامه را اجرا کنید و روی یکی از دکمه ها به دلخواه کلیک کنید. مشاهده خواهید کرد که عبارت My Windows Application در نوار عنوان کادر پیغام نمایش داده می شود.

چگونه کار می کند؟

همانطور که مشاهده می کنید به علت اینکه مقدار اولیه ی خاصیت ApplicationName برابر با رشته ی تهی در نظر گرفته شده است، همین مقدار نیز به پنجره ی Properties فرستاده می شود و این خاصیت به صورت اولیه مقداری

نخواهد داشت. حال اگر در زمان طراحی مقداری را برای آن تعیین کنید، هنگام اجرای برنامه این مقدار در نوار عنوان هر یک از کادرهای پیغام نمایش داده می شود.
هنگامی که محیط طراحی ویژوال استودیو بخواهد با استفاده از پنجره ی Properties خاصیت های مربوط به یک کنترل را نمایش دهد، به درون شیئ مربوط به آن کنترل رفته و مقدار هر یک از خاصیت را دریافت می کند و در این پنجره نمایش می دهد. همچنین هنگامی که مقدار یکی از این خاصیت ها را با استفاده از پنجره ی Properties تغییر دهید، محیط طراحی ویژوال استودیو به شیئ مربوط به آن کنترل رفته و مقدار آن خاصیت را برابر با مقدار جدید قرار می دهد.

اضافه کردن متد به کنترل های سفارشی:

همانطور که ممکن است تاکنون حدس زده باشید، وقتی می توانید یک خاصیت را برای یک کنترل سفارشی ایجاد کنید، مسلماً می توانید یک متد را نیز به آن اضافه کنید. تمام کاری که برای اضافه کردن یک متد به کنترل باید انجام دهید این است که یک تابع و یا زیر برنامه از نوع public را به کنترل اضافه کنید. به این ترتیب می توانید آن متد را در زمان کار با کنترل فراخوانی کنید. نحوه ی انجام این کار را در بخش امتحان کنید بعد مشاهده خواهیم کرد.

امتحان کنید: اضافه کردن یک متد به کنترل ایجاد شده

۱) به قسمت ویرایشگر کد مربوط به فایل UserControl1.cs بروید و متد زیر را به کلاس UserControl1 اضافه کنید.

```
public string FormCaption()
{
    return Application.OpenForms[0].Text;
}
```

۲) حال به قسمت طراحی فرم مربوط به Form1 برگردید و با استفاده از جعبه ابزار یک کنترل Button جدید روی فرم قرار دهید. خاصیت Name این کنترل را با مقدار btnFormName و خاصیت Text آن را با مقدار Form Name تنظیم کنید.

۳) روی این کنترل دو بار کلیک کنید و کد مشخص شده در زیر را در متد مربوط به رویداد Click این کنترل اضافه کنید:

```
private void btnFormName_Click(object sender,
                                    EventArgs e)
{
    MessageBox.Show(userControl11.FormCaption(),
                            "Form1");
}
```

۴) برنامه را اجرا کرده و روی دکمه ی Form Name کلیک کنید. مشاهده می کنید که کارد پیغامی نمایش داده شده و نام اولین فرم باز برنامه را اعلام می کند.

چگونه کار می کند؟

ایجاد یک تابع و یا زیر برنامه برای یک کنترل تفاوت چندانی با ایجاد یک تابع و یا زیر برنامه برای یک کلاس ندارد. البته توجه کنید متدی که در این قسمت ایجاد می کنید باید از نوع Public باشد تا بتواند توسط افرادی که از این کنترل استفاده می کنند مورد استفاده قرار بگیرد.
متدی که در این قسمت ایجاد می کنیم عمل خاصی را انجام نمی دهید. فقط با استفاده از خاصیت OpenForms در کلاس Application لیستی از تمام فرمهایی از برنامه که هم اکنون باز هستند را دریافت می کند و نام اولین فرم را در یک کادر پیغام نمایش می دهد.

```
public string FormCaption()
{
      return Application.OpenForms[0].Text;
}
```

برای استفاده از این متد در برنامه، مانند استفاده از متد های مربوط به دیگر کنترل ها عمل می کنیم. یعنی ابتدا نام کنترل را وارد می کنیم سپس یک "." قرار می دهیم به این ترتیب لیستی نمایش داده می شود که نام متد مورد نظر ما نیز در آن وجود دارد.

```
private void btnFormName_Click(object sender,
                                        EventArgs e)
{
      MessageBox.Show(userControl11.FormCaption(),
                                              "Form1");
}
```

برای استفاده از متدی که در این قسمت اضافه کرده ایم حتی نیازی به کامپایل برنامه نیز وجود ندارد. بعد از اضافه کردن متد به کنترل، به راحتی می توانید از متد در برنامه استفاده کنید.

اضافه کردن رویداد به کنترل:

تا اینجا با اضافه کردن خاصیت و متد در یک کنترل آشنا شدیم. در این قسمت می خواهیم نحوه ی اضافه کردن رویداد به یک کنترل را بررسی کنیم. با اضافه کردن یک رویداد به کنترل، افرادی که از آن کنترل استفاده می کنند می توانند هنگام رخ دادن آن رویداد متدهای مشخصی را اجرا کنند.
در بخش امتحان کنید بعد، سه رویداد برای هر یک از کلید های موجود در کنترل ایجاد می کنید. به این ترتیب هنگامی که کاربر روی هر کدام از این کلید ها کلیک کند رویداد مربوط به آن فراخوانی می شود.

نکته: توجه به این نکته ضروری است که هنگامی که از یک کنترل در برنامه ی خود استفاده می کنید و برای مثال متـدی را بـرای رویداد `Click` آن مشخص می کنید، زمان رخ دادن رویداد `Click` به وسیله ی خود کنترل مشخص می شود. پـس در ایـن قسمت که خودمان در حال طراحی کنترل هستیم، اگر رویدادی را به این کنترل اضافه کنیم زمان رخ دادن این رویداد را نیز خودمان باید مشخص کنیم.

امتحان کنید: ایجاد و فراخوانی یک رویداد

۱) همانطور که می دانید هنگامی که یک رویداد رخ می دهد، فقط توابع با ساختارهای خاصی می توانند فراخوانی شوند. پس برای ایجاد یک رویداد نیز، ابتدا باید ساختار توابعی که این رویداد می تواند فراخوانی کند را مشخص کنیم. برای ایـن کـار باید با استفاده از کلمه ی کلیدی `delegate` ساختار توابع مورد نیاز را تعریف کنیم. پس ابتدا کد زیر را بـه کـلاس مربوط به کنترل اضافه کنید:

```
// Private members
private string strApplicationName = "";

// Public Delegates
public delegate void _ApplicationNameChanged(
                                        string AppName);
```

۲) حال که نوع توابع مورد نیاز برای رویداد را مشخص کردیم، باید خود رویداد را تعریف کنیم. برای این کار نیز باید از کلمـه ی کلیدی `event` به صورت زیر استفاده کنیم. پس کد مشخص شده در زیر را به کلاس اضافه کنید تا رویـداد مـورد نظرمان ایجاد شود:

```
// Public Delegates
public delegate void _ApplicationNameChanged(
                                        string AppName);

// Public Events
public event _ApplicationNameChanged
                                    ApplicationNameChanged;
```

۳) در انتها نیز باید مشخص کنیم که این رویداد در چه مواقعی باید فراخوانی شود. در این برنامه می خواهیم رویداد هنگـامی فراخوانی شود که کاربر روی دکمه ی `btnApplicationName` کلیک کند. بنابراین کد مربوط به فراخوانی رویداد را به صورت زیر به متد `btnApplicationName_Click` اضافه می کنیم.

```
private void btnApplicationName_Click(object sender,
                                        EventArgs e)
{
    if (this.ApplicationNameChanged != null)
    {
        this.ApplicationNameChanged(
```

```
                                    Application.ProductName);
            }

            MessageBox.Show("Application Name is: " +
                                    Application.ProductName,
                                    this.ApplicationName);
        }
```

۴) برنامه را کامپایل کنید تا مطمئن شوید خطای دستوری در آن وجود ندارد.

چگونه کار می کند؟

همانطور که در قسمتهای قبلی نیز ممکن است متوجه شده باشید، یک رویداد را در حقیقت می توان مانند یک آرایه از توابع در نظر گرفت[1]. به این ترتیب می توانیم بگوییم هنگامی که یک رویداد فراخوانی می شود، در حقیقت توابع موجود در این آرایه فراخوانی می شوند. در قسمت آرایه ها مشاهده کردید که تمام عناصر یک آرایه باید از یک نوع باشند. برای مثال نمی توان آرایه ای تعریف کرد که بتواند اعضایی از نوع `int` و نیز اعضایی از نوع `string` را در خود نگهداری کند. در رویداد ها نیز به همین صورت است. یک رویداد نمی تواند شامل توابع با پارامتر ها و خروجی های مختلف باشد. بلکه تمام توابعی که در آرایه ی مربوط به یک رویداد قرار می گیرند باید از یک نوع باشند.

در تعریف یک رویداد در مرحله ی اول باید نوع توابعی که می توانند در آن رویداد قرار بگیرند را مشخص کنیم. این کار با کلمه ی کلیدی `delegate` انجام می شود. در این قسمت می خواهیم توابعی که در این رویداد قرار می گیرند یک پارامتر از نوع `string` دریافت کنند و خروجی نیز نداشته باشند (خروجی آنها از نوع `void` باشد). بنابراین از کد زیر برای تعریف نوع توابع مورد نیاز (یا همان `delegate`) استفاده می کنیم:

```
// Public Delegates
public delegate void _ApplicationNameChanged(
                                    string AppName);
```

بعد از اینکه نوع توابع را تعیین کردیم، باید با استفاده از آن یک رویداد تعریف کنیم. در اینجا می توانید تعریف کردن یک رویداد را همانند تعریف کردن یک آرایه از نوع `delegate` ایجاد شده در مرحله ی قبل در نظر بگیرید. برای تعریف یک رویداد از کلمه ی کلیدی `event` به صورت زیر استفاده می کنیم:

```
// Public Events
public event _ApplicationNameChanged
                                    ApplicationNameChanged;
```

[1] در زبان C++ که نسل قبلی C# به شما می رود، برای برنامه نویسی به صورت رویداد گرا بایستی آرایه هایی ایجاد می کردیم که هر یک از اعضای آن یک تابع می بود. سپس تمام عناصر این آرایه را در مواقع مورد نیاز فراخوانی می کردیم. در C# این مورد به صورتی که در این مثال مشاهده کردید پیاده سازی شده است، اما مفهوم آن همچنان مانند قبل است.

به عبارت دیگر می توانید فرض کنید که در این مرحله یک آرایه به نام ApplicationNameChanged ایجاد می کنیم که می تواند توابعی از نوع ApplicationNameChanged_ را در خود نگهداری کند.
همانطور که قبلا گفتم، هنگامی که یک کنترل را طراحی می کنیم و می خواهیم یک رویداد را به آن اضافه کنیم، باید زمان فراخوانی توابع داخل آن رویداد را نیز مشخص کنیم. در این کنترل می خواهیم این رویداد زمانی فراخوانی شود که کاربر روی دکمه ی btnApplicationName کلیک می کند، بنابراین کد مورد نیاز برای فراخوانی رویداد را در متد btnApplicationName_Click قرار می دهیم.
قبل از فراخوانی یک رویداد باید بررسی کنیم تا ببینیم آیا متدی در لیست رویداد برای فراخوانی قرار دارد یا نه. زیرا اگر متدی در این لیست قرار نداشته باشد نمی توان آن را فراخوانی کرد. برای بررسی این مورد در یک دستور if مقدار رویداد را با عبارت null بررسی می کنیم. اگر مقدار رویداد مخالف با null (تهی) بود، به این معنی است که متد هایی در لیست این رویداد قرار دارند و می توانیم آن متد ها را فراخوانی کنیم. برای فراخوانی یک رویداد باید همانند یک متد عادی با آن رفتار کرد. برای مثال رویدادی که در این قسمت تعریف کردیم شامل متد هایی است که یک پارامتر از نوع string دریافت می کنند و مقداری هم برنمی گردانند. پس در اینجا نام رویداد را با رشته ای که می خواهیم به عنوان پارامتر به تمام متد های موجود در این رویداد فرستاده شود احضار می کنیم. به این ترتیب برنامه به صورت اتوماتیک تمام متد هایی که در لیست این رویداد وجود دارند را فراخوانی کرده و رشته ی مشخص شده را به عنوان پارامتر به آنها ارسال می کند.

```
if (this.ApplicationNameChanged != null)
{
     this.ApplicationNameChanged(
                    Application.ProductName);
}
```

خوب، به این ترتیب تمام مراحل لازم برای ایجاد یک رویداد در این کنترل طی شده است و می توانیم از رویداد ایجاد شده همانند رویداد های دیگر کنترل ها استفاده کنیم. در بخش امتحان کنید بعد از رویداد ایجاد شده در برنامه استفاده خواهیم کرد.

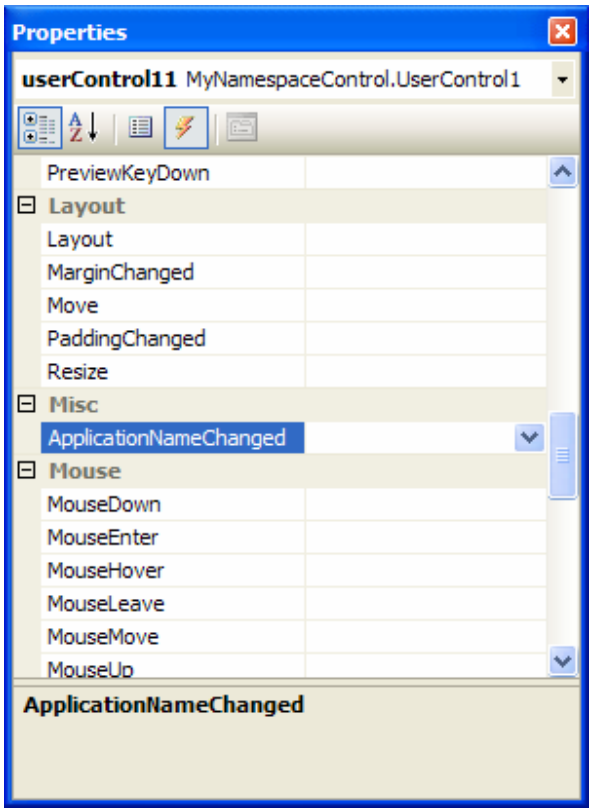
امتحان کنید: استفاده از رویداد ایجاد شده

۱) به قسمت طراحی فرم مربوط به Form1 بروید و با استفاده از جعبه ابزار، یک کنترل TextBox همانند شکل ۱۳-۳ در فرم قرار دهید. خاصیت Text این کنترل را نیز با مقدار txtApplicationName تنظیم کنید.



شکل ۱۳-۳

۲) حال کنترل UserControl11 را از فرم برنامه انتخاب کرده و در پنجره ی Properties روی آیکون Events کلیک کنید تا رویدادهای مربوط به این کنترل نمایش داده شود. مشاهده می کنید تمام رویداد هایی که برای یک کنترل معمولی وجود دارد در این قسمت نیز لیست شده اند (شکل ۱۳-۴) . در لیست رویداد ها به گروه Misc بروید و بر روی رویداد ApplicationNameChanged دو بار کلیک کنید تا یک متد برای این رویداد ایجاد شود.



شکل ۱۳-۴

۳) کد مشخص شده در زیر را در متد ایجاد شده وارد کنید:

```
private void userControl11_ApplicationNameChanged(
                                        string AppName)
{
      txtApplicationName.Text = AppName;
}
```

۴) حال برنامه را اجرا کنید. مشاهده می کنید هر زمان که روی دکمه ی Application Name کلیک کنید، ابتدا کادر متنی داخل فرم با استفاده از رشته ی برگردانده شده به وسیله ی رویداد پر شده و سپس کادر پیغام نمایش داده می شود.

چگونه کار می کند؟

استفاده از رویداد های یک کنترل بسیار واضح است و کاری است که از ابتدای کتاب تاکنون با کنترل هایی مانند TextBox و یا Button انجام می داده ایم. برای این کار کافی است که کنترل مورد نظر خود را در فرم برنامه انتخاب کرده و با کلیک روی آیکون Events در پنجره ی Properties لیست رویداد های آن را مشاهده کنیم. سپس در این لیست رویداد مورد نظر خود را انتخاب کرده و روی آن دو بار کلیک کنیم. با این کار ویژوال استودیو به صورت اتوماتیک یک متد با همان ساختاری که رویداد نیاز دارد ایجاد کرده و آن را به لیست متدهای مربوط به آن رویداد اضافه می کند. بنابراین هر بار که آن رویداد رخ دهد، متد ایجاد شده نیز فراخوانی می شود.
اغلب متد هایی که برای رویداد ها در قسمتهای قبل به صورت اتوماتیک ایجاد می شد دو پارامتر به نامهای e از نوع EventArgs و نیز sender از نوع object را دریافت می کردند، اما متدی که در این مرحله به صورت اتوماتیک برای رویداد ApplicationNameChanged ایجاد شد، فقط یک پارامتر به نام AppName از نوع string را دریافت می کند. این مورد نیز به این دلیل است که ویژوال استودیو قالب متد هایی را که در این قسمت ایجاد می کند از قالب تعریف شده در کلاس برای ایجاد رویداد دریافت می کند. برای مثال قالب متدی که در این قسمت برای این رویداد ایجاد شده است از delegate تعریف شده در داخل کنترل گرفته شده است.
بعد از ایجاد متدی برای این رویداد کافی است کد مورد نظر خود را در آن وارد کنیم تا هر بار که رویداد رخ می دهد، این کد نیز فراخوانی شود. کدی که در اینجا وارد می کنیم، فقط نام برنامه که به وسیله ی پارامتر AppName به متد فرستاده می شود را در کادر متنی موجود در فرم نمایش می دهد.

زمان اجرا یا زمان طراحی:

در شرایط خاصی ممکن است نیاز داشته باشید که بدانید در حال حاضر کنترل شما در زمان طراحی قرار دارد و یا در زمان اجرا. یک کنترل هنگامی در زمان طراحی است که کاربر آن را در فرم قرار داده است و می تواند خاصیت های آن را با استفاده از پنجره ی Properties تغییر دهد. همچنین زمانی که برنامه ی حاوی کنترل اجرا شود و کاربر بتواند از متد ها و یا رویداد های آن کنترل استفاده کند گفته می شود که کنترل در زمان اجرا قرار دارد.
برای مثال ممکن است کنترل شما بخواهد هنگامی که خاصیت مشخصی از آن تنظیم شد، به یک بانک اطلاعاتی متصل شود. اما اگر مقدار آن خاصیت در زمان طراحی تنظیم شود متصل شدن به بانک اطلاعاتی برای کنترل ممکن نخواهد بود، و این مورد فقط زمانی امکان پذیر خواهد بود که خاصیت در زمان اجرا و به وسیله ی کد تنظیم شود.
عموماً هر کنترل خود دارای یک خاصیت به نام DesignMode از نوع بولین است که مشخص می کند آیا کنترل در حالت طراحی قرار دارد یا نه؟ اگر این خاصیت مقدار True را برگرداند به این معنی است که کنترل در حالت طراحی است و اگر مقدار False را برگرداند به این معنی است که کنترل در حالت اجرا است.
در بخش امتحان کنید بعد با اضافه کردن یک کنترل Label و یک کنترل Timer به UserControl1 مقداری آن را تغییر خواهیم داد. به این ترتیب هنگامی که کنترل در زمان طراحی باشد عبارت "Design Mode" در لیبل نمایش داده می شود و زمانی که کنترل در حالت اجرا قرار بگیرد، ساعت سیستم در برنامه نمایش داده می شود.

امتحان کنید: ایجاد یک کنترل که "زمان طراحی" را متوجه شود!

۱) به قسمت طراحی مربوط به UserControl1 بروید و سپس اندازه ی فضای طراحی را مقداری بزرگ کنید تا بتوانید یک کنترل لیبل را در زیر کنترلهای Button موجود در این قسمت قرار دهید.

۲) با استفاده از جعبه ابزار یک کنترل Label را در زیر کنترل های Button قرار دهید و خاصیت Name آن را با مقدار lblTime تنظیم کنید.

۳) حال با استفاده از جعبه ابزار روی کنترل Timer دو بار کلیک کنید تا این کنترل به قسمت پایین بخش طراحی کنترل اضافه شود. مقادیر پیش فرض را برای خاصیتهای این کنترل قبول کنید، فقط خاصیت Enabled آن را برابر با False و خاصیت Interval آن را برابر با ۱۰۰ قرار دهید.

۴) حال باید در زمان خاصی بررسی کنیم که آیا کنترل در زمان طراحی است و یا در زمان اجرا. بهترین قسمت برای این کار متد InitLayout است که در کلاس System.Windows.Forms.Contorl تعریف شده است. این متد هم در زمان طراحی و هم در زمان اجرا به وسیله ی برنامه فراخوانی می شود، بنابراین بهترین مکان برای بررسی این مورد است که آیا کنترل در زمان طراحی است و یا در زمان اجرا، و اگر در زمان طراحی بود کنترل Timer را فعال کنیم. بنابراین به قسمت کد نویسی مربوط به UserControl1 رفته و کد زیر را به این کلاس اضافه کنید:

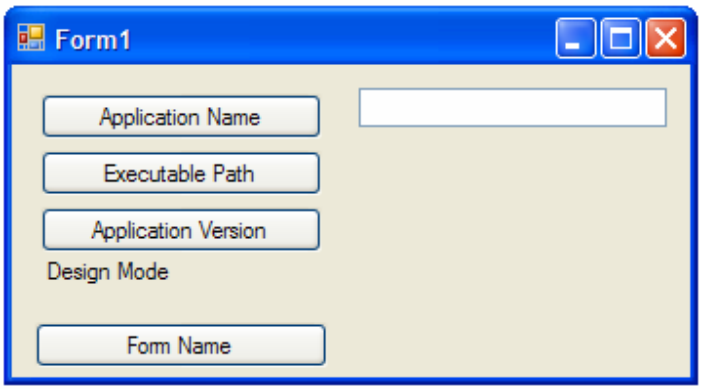
```
protected override void InitLayout()
{
     // Are we in design mode
     if (this.DesignMode)
     {
          lblTime.Text = "Design Mode";
     }
     else
     {
          timer1.Enabled = true;
     }
}
```

۵) در آخر نیز باید کد مربوط به رویداد Tick کنترل Timer را به برنامه اضافه کنیم. این رویداد زمانی که کنترل Timer فعال باشد، در فاصله ی زمانی مشخص شده فراخوانی می شود. بر روی کنترل Timer در قسمت طراحی کنترل دو بار کلیک کنید تا متد مربوط به رویداد Tick آن به صورت اتوماتیک ایجاد شود. سپس کد زیر را در این متد وارد کنید:

```
private void timer1_Tick(object sender, EventArgs e)
{
     // Display the time
     lblTime.Text = DateTime.Now.ToLongTimeString();
}
```

۶) قبل از اینکه بتوانید تغییرات اعمال شده در کنترل را در برنامه ی ویندوزی Controls نیز دریافت کنید، باید پروژه ی مربوط به کنترل را به صورت کامل کامپایل کنید. برای این کار با استفاده از پنجره ی Solution Explorer روی پروژه ی MyNamespaceControl کلیک راست کرده و گزینه ی Build را انتخاب کنید.

۷) به قسمت طراحی فرم مربوط به Form1 برگردید و کنترل userControl1 را از فرم حذف کنید. سپس با استفاده از جعبه ابزار یک کنترل دیگر از این نوع را در فرم قرار دهید. مشاهده خواهید کرد کنترل Label اضافه شده که حاوی متن Design Mode است، در این قسمت نمایش داده شده است (شکل ۱۳-۵)



شکل ۱۳-۵

۸) حال برنامه را اجرا کنید. به این ترتیب متن موجود در Label با ساعت کنونی سیستم تعویض خواهد شد.

چگونه کار می کند؟

متد InitLayout هنگامی که یک کنترل بخواهد به صورت اولیه توسط برنامه مقدار دهی شود فراخوانی خواهد شد، حال چه کنترل در زمان طراحی باشد و چه در زمان اجرا. خاصیت DesignMode نیز از این کنترل یک مقدار از نوع بولین برمی گرداند، که اگر این مقدار برابر با true باشد به این معنی است که کنترل در زمان طراحی است و اگر برابر با False باشد به این معنی است که کنترل در زمان اجرا است.
اگر کنترل در زمان طراحی باشد، کافی است که متن "Design Mode" را در لیبل نمایش دهید، اما اگر کنترل در زمان اجرا باشد باید کنترل Timer را فعال کنید تا در فاصله های زمانی متوالی ساعت سیستم را در پنجره نمایش دهد.

```
protected override void InitLayout()
{
      // Are we in design mode
      if (this.DesignMode)
      {
            lblTime.Text = "Design Mode";
      }
      else
      {
            timer1.Enabled = true;
      }
}
```

مسلماً موارد زیادی وجود دارد که می خواهید رفتار کنترلی که طراحی می کنید در هنگام طراحی و در هنگام اجرا متفاوت باشد. در تمام این موارد می توانید با استفاده از خاصیت DesignMode تشخیص دهید که کنترل در چه زمانی قرار دارد و سپس کد مناسبی را اجرا کنید.
رویداد Tick مربوط به کنترل Timer در فاصله های زمانی متناوب که به وسیله ی خاصیت Interval مشخص می شود فراخوانی شده و کد درون آن به وسیله ی برنامه اجرا می شود. در اینجا مشخص می کنیم که این رویداد در هر ۱۰۰ میلی ثانیه، یک بار فراخوانی شود. به این ترتیب هر زمانی که این کنترل فعال شود (مقدار خاصیت Enabled برابر با true شود) در هر ۱۰۰ میلی ثانیه یک بار زمان سیستم با استفاده از متد ها و خاصیتهای موجود در کلاس DateTime دریافت شده و در صفحه نمایش داده می شود.

```
private void timer1_Tick(object sender, EventArgs e)
{
      // Display the time
      lblTime.Text = DateTime.Now.ToLongTimeString();
}
```

در این قسمت به علت اینکه در ظاهر کنترل تغییر ایجاد کرده ایم، باید بعد از کامپایل مجدد پروژه ی مربوط به آن کنترل فعلی را از داخل فرم حذف کرده و یک نمونه ی جدید از آن را در فرم قرار دهیم. اما اگر صرفاً در کد مربوط به کنترل تغییراتی ایجاد کرده بودیم این تغییرات به صورت اتوماتیک توسط کنترلهای مورد استفاده در برنامه نیز دریافت می شدند.

ایجاد یک کتابخانه ی فرم:

در قسمت قبل مشاهده کردیم که چگونه می توان چند وظیفه ی مرتبط به یکدیگر را در قالب یک کتابخانه ی کنترل دسته بندی کرده و در قسمتهای مختلف از آن استفاده کرد. اما همواره نیز لازم نیست که کارهای مورد نیاز را به صورت یک کنترل دسته بندی کنیم و سپس در یک فرم قرار دهیم، بلکه می توانیم فرمهای عمومی ایجاد کرده و از آن در موارد مورد نیاز استفاده کنیم. این کار، همان منطق ای است که برای نمایش کادر هایی مانند Open File و یا Print در برنامه به کار می رود.
فرض کنید بخواهید یک فرم را در چندین قسمت از برنامه ی خود استفاده کنید. برای مثال ممکن است بخواهید اغلب برنامه های خود از فرمی برای جستجوی مشترکین و یا از فرمی برای ورود کاربران و تعیین هویت آنها استفاده کنید. در این موارد می توانید این فرم ها را طراحی کرده و در یک کتابخانه ی فرم قرار دهید. سپس هر زمان که به استفاده از این فرم ها نیاز داشتید می توانید آنها را نمونه سازی کرده و در برنامه مورد استفاده قرار دهید. انجام چنین کاری در .NET زیاد مشکل نیست. به عبارت دیگر انجام این کار تفاوت چندانی با ایجاد کتابخانه های کلاس و یا کتابخانه های کنترل ندارد. فقط باید خاصیت ها و متد هایی را به این فرم اضافه کنید تا کاربر بتواند آن را بر اساس نیاز خود تغییر داده سپس فراخوانی کند و همچنین بتواند نتیجه ی برگشت داده شده از آن فرم را بدست آورد.

ایجاد یک کتابخانه ی فرم حاوی فرم ورود:

در بخش امتحان کنید بعد، یک فرم ورود ساده ایجاد خواهیم کرد. البته در طراحی این فرم، زیاد روی تعیین هویت کاربر تمرکز نمی کنیم و سعی خواهیم کرد که بیشتر توجه خود را روی ایجاد و نمایش فرم به کاربر متمرکز کنیم.

امتحان کنید: ایجاد پروژه ی کتابخانه ی فرم

۱) پروژه هایی که هم اکنون در محیط ویژوال استودیو باز هستند را ببندید و یک پروژه ی جدید به نام FormsLibrary از نوع Class Library ایجاد کنید.

۲) با استفاده از پنجره ی Solution Explorer روی نام پروژه کلیک راست کرده و گزینه ی Add Reference را انتخاب کنید. سپس از میان کامپوننت های .NET موجود، System.Windows.Forms را انتخاب کرده و روی دکمه ی OKکلیک کنید.

۳) حال بر روی نام پروژه در پنجره ی Solution Explorer کلیک راست کرده و گزینه ی Add ➔ Windows Form را انتخاب کنید تا یک فرم ویندوزی جدید به پروژه اضافه شود. نام این فرم را Login.cs قرار داده و روی دکمه ی OK کلیک کنید.

۴) برای اینکه فرم قابل استفاده شود باید تعدادی از خاصیت های آن را تغییر دهید. برای این کار خاصیت های فرم را بر اساس لیست زیر تنظیم کنید:

- خاصیت FormBorderStyle را برابر با FixedDialog قرار دهید.
- خاصیت MaximizeBox را برابر با False قرار دهید.
- خاصیت MinimizeBox را برابر با False قرار دهید.
- خاصیت StartPosition را برابر با CenterScreen قرار دهید.

۵) حال دو کنترل لیبل به فرم اضافه کرده و خاصیت های Text آنها را به ترتیب برابر با User Name و Password قرار دهید.

۶) دو کنترل TextBox روی فرم قرار دهید و خاصیت Name آن ها را به ترتیب با مقادیر txtUserName و txtPassword تنظیم کنید. سپس خاصیت PasswordChar کنترل txtPassword را برابر با * قرار دهید تا کلمه ی عبوری که در این کادر وارد می شود در صفحه نمایش داده نشود.

۷) یک کنترل Button به فرم اضافه کرده و خاصیت های آن را بر طبق لیست زیر تنظیم کنید:

- خاصیت Name را برابر با btnOK قرار دهید.
- خاصیت Text را برابر با OK قرار دهید.
- خاصیت DialogResult را برابر با OK قرار دهید.

۸) کنترل Button دیگری روی فرم قرار داده و خاصیت های آن را بر طبق لیست زیر تنظیم کنید:

- خاصیت Name آن را برابر با btnCancel قرار دهید.
- خاصیت Text آن را برابر با Cancel قرار دهید.
- خاصیت DialogResult آن را برابر با Cancel قرار دهید.

۹) فرم تکمیل شده ی شما باید مشابه شکل ۱۳-۶ شده باشد.



شکل ۱۳-۶

۱۰) در پنجره ی Solution Explorer روی فایل Class1.cs کلیک راست کرده و نام آن را به LoginEventArgs.cs تغییر دهید. سپس کد مشخص شده در زیر را به آن کلاس اضافه کنید:

```
public class LoginEventArgs : EventArgs
{
     // Public member
     int UserID;

     // Constructor
     public LoginEventArgs(int userIdentifier)
     {
          UserID = userIdentifier;
     }
}
```

۱۱) به قسمت ویرایشگر کد مربوط به فرم Login بروید و کد مشخص شده در زیر را در کلاس Login وارد کنید:

```
public partial class Login : Form
{
     // Private members
     private int intAttemptCount = 0;
     private bool blnAllowClosing = false;
     private int intUserID;

     // Public delegates
     public delegate void _LoginFailed(Object sender,
                                       EventArgs e);
     public delegate void _LoginSucceeded(Object sender,
                                       LoginEventArgs e);
     public delegate void _LoginCancelled(Object sender,
                                       EventArgs e);

     // Public events
     public event _LoginFailed LoginFailed;
     public event _LoginSucceeded LoginSucceeded;
     public event _LoginCancelled LoginCancelled;
```

۱۲) حال باید یک خاصیت فقط خواندنی به کلاس این فرم اضافه کنیم تا افرادی که از این فرم استفاده می کنند به وسیله ی این خاصیت بتواند شناسه ی مربوط به کاربری که توانسته است با موفقیت وارد سیستم شود را دریافت کنند. برای این کار کد زیر را به کلاس اضافه کنید:

```
    public int UserID
    {
        get
        {
            return intUserID;
        }
    }
```

۱۳) برای سادگی بیشتر کار با این کنترل بهتر است هنگامی که فرم فعال می شود، کادر UserName را با نام ای که کاربر به وسیله ی آن وارد ویندوز شده است پر کنیم. برای دسترسی به نامی که کاربر با آن وارد ویندوز شده است می توانیم از خاصیت UserName از کلاس Environment استفاده کنیم. برای اینکه هنگام فعال شدن فرم این کار انجام شود، از رویداد Activated در فرم استفاده می کنیم.
فرم Login را در قسمت طراحی فرم انتخاب کرده و در پنجره ی Properties روی آیکون Events کلیک کنید تا لیست رویداد های آن نمایش داده شود. سپس در این لیست رویداد Activated را انتخاب کرده و روی آن دو بار کلیک کنید تا متد مربوط به این رویداد ایجاد شود. سپس کد زیر را در این متد وارد کنید:

```
private void Login_Activated(object sender, EventArgs e)
{
    // Populate the UserName text box
    txtUserName.Text = Environment.UserName;

    // Set the focus to the password text box
    txtPassword.Focus();
}
```

۱۴) در این فرم نیاز داریم که بسته شدن فرم را کنترل کنیم تا در شرایط خاصی از آن جلوگیری کنیم و به کاربر اجازه ندهیم که فرم را ببندد. برای این کار می توانیم از رویداد FormClosing مربوط به فرم Login استفاده کنیم. مجدداً به قسمت طراحی فرم مربوط به فرم Login بروید و فرم را انتخاب کنید. در لیست رویداد های آن در پنجره ی Properties، روی گزینه ی FormClosing دو بار کلیک کنید تا متد مربوط به این رویداد به صورت اتوماتیک ایجاد شود. سپس کد زیر را به این متد اضافه کنید:

```
private void Login_FormClosing(object sender,
                               FormClosingEventArgs e)
{
    // If we are not allowing the form to close...
    if (!blnAllowClosing)
    {
        // Set the cancel flag to true
```

```
            e.Cancel = true;
        }
}
```

۱۵) هنگامی که کاربر در این فرم روی دکمه ی OK کلیک کرد باید کدی را اجرا کنیم تا نام کاربری و کلمه ی عبور او را کنترل کند. برای این کار به قسمت طراحی فرم برگشته و روی کنترل btnOK دو بار کلیک کنید تا متد مربوط به رویداد Click آن ایجاد شود. سپس کد زیر را در این متد وارد کنید:

```
private void btnOK_Click(object sender, EventArgs e)
{
     // Was a user name entered?
     if(txtUserName.Text.Trim().Length > 0)
     {
          // Was the password correct?
          if(txtPassword.Text == "secret")
          {
               // Successful login, set the User ID
               intUserID = 27;

               // Raise the LoginSucceeded event
               if(this.LoginSucceeded != null)
                    this.LoginSucceeded(this,
                         new LoginEventArgs(intUserID));

               // Turn on the allow closing flag
               blnAllowClosing = true;
          }
          else
          {
               // Inform the user
               // that the password was invalid
               MessageBox.Show("The password you entered" +
                         " was invalid.","Login");

               // Increment the attempt count
               intAttemptCount += 1;

               // Check the attempt count
               if (intAttemptCount == 3)
               {
                    // Raise the LoginFailed event
                    if(this.LoginFailed != null)
                         this.LoginFailed(this,
                                   new EventArgs());

                    // Set the Cancel dialog result
```

```
                    this.DialogResult =
                                DialogResult.Cancel;

                    // Turn on the allow closing flag
                    blnAllowClosing = true;
                }
            }
        }
        else
        {
            // Inform the user
            // that they must supply a user name
            MessageBox.Show("You must supply a User Name.",
                                    "Login");
        }
}
```

۱۶) در انتها نیز باید کدی را برای کنترل `btnCancel` بنویسیم تا هنگامی که کاربر روی این کنترل کلیک کرد، عمل مناسبی رخ دهد. برای این کار به قسمت طراحی فرم برگردید و روی کنترل `btnCancel` دو بار کلیک کنید تا متد مربوط به رویداد `Click` آن ایجاد شود. سپس کد زیر را در این متد وارد کنید:

```
private void btnCancel_Click(object sender, EventArgs e)
{
    // Raise the LoginCancelled event
    if(this.LoginCancelled != null)
        this.LoginCancelled(this, new EventArgs());
    // Turn on the allow closing flag
    blnAllowClosing = true;
}
```

چگونه کار می کند؟

در ابتدای این برنامه بعد از ساختن پروژه، یک ارجاع به فایل `System.Windows.Forms` در برنامه ایجاد می کنیم. علت این کار نیز این است که طی این برنامه می خواهیم از کلاسهای موجود در این فضای نام استفاده کنیم و ویژوال استودیو به صورت پیش فرض این فضای نام را به برنامه هایی که از نوع `Class Library` ایجاد می شوند اضافه نمی کند.
بعد از انجام این کار یک فرم ویندوزی به برنامه اضافه کرده، کنترلهای مورد نیاز را در آن قرار می دهیم و خاصیتهای آن را نیز تنظیم می کنیم. یکی از مهمترین خاصیت هایی که در این قسمت تغییر می دهیم، خاصیت `DialogResult` در کنترل `Button` است. با مقدار دادن به این خاصیت سبب می شویم هنگامی که کاربر روی یکی از دکمه های `OK` و یا `Cancel` کلیک کند، این دکمه ها باعث شوند که فرم بسته شده و مقدار `DialogResult.OK` و یا `DialogResult.Cancel` به احضار کننده ی فرم برگشت داده شود.
در قسمت قبل هنگامی که در رابطه با رویداد ها و نحوه ی ایجاد آنها صحبت می کردیم، مشاهده کردیم که ساختار توابعی که در لیست یک رویداد قرار می گیرند باید مشابه هم باشد. همچنین دیدیم معمولاً توابعی که هنگام رخ دادن یک رویداد فراخوانی می

شوند دو پارامتر دریافت می کنند که یکی از آنها از نوع Object است و دیگری از نوع EventArgs و یا یکی از کلاس های مشتق شده از آن. البته هیچ الزامی در رعایت این مورد وجود ندارد و می توان رویداد هایی را ایجاد کرد که متد های آن هیچ پارامتری دریافت نکنند و یا بیش از دو پارامتر دریافت کنند (همانند برنامه ی قبل).
اما معمولاً به صورت قرارداد هنگام ایجاد یک رویداد، متدهای مربوط به آن را به صورتی تعریف می کنند که دو پارامتر دریافت کند:

- پارامتر اول که از نوع Object است مشخص کننده ی شیئ ای است که این رویداد را فراخوانی کرده است.
- پارامتر دوم که از نوع EventArgs و یا یکی از کلاس های مشتق شده از آن است، حاوی اطلاعات لازم در مورد رویداد رخ داده است. معمولاً اگر هنگام رخ دادن یک رویداد نیاز نباشد که اطلاعات خاصی به متد ها ارسال شود از کلاس EventArgs استفاده می کنند. اما اگر بخواهیم پارامترهایی را به متد ها ارسال کنیم، می توانیم یک کلاس از EventArgs مشتق کرده و کلاس را به گونه ای تغییر دهیم تا بتواند این پارامتر ها را در خود نگهداری کند. سپس پارامتر دوم متد مربوط به رویداد را، از نوع کلاس ایجاد شده در نظر بگیریم.

در تعریف رویداد های مربوط به این کلاس می خواهیم این قاعده را رعایت کنیم، بنابراین ابتدا کلاسی را از کلاس EventArgs مشتق می کنیم تا بتوانیم اطلاعات لازم برای رخ دادن یک رویداد را در آن قرار دهیم و به متد های فراخوانی شده ارسال کنیم. البته استفاده از این کلاس فقط زمانی لازم است که رویداد LoginSucceeded رخ دهد. در این حالت لازم است که شناسه ی کاربری که با موفقیت وارد سیستم شده است را به متد ها ارسال کنیم. در بقیه ی موارد می توانیم از کلاس EventArgs استفاده کنیم. زیرا هنگام رخ دادن دیگر رویداد ها لازم نیست هیچ اطلاعات خاصی را به آنها ارسال کنیم.

```
public class LoginEventArgs : EventArgs
{
      // Public member
      int UserID;

      // Constructor
      public LoginEventArgs(int userIdentifier)
      {
            UserID = userIdentifier;
      }
}
```

بعد از انجام دادن این موارد به طراحی فرم و متد های مورد نیاز در آن می پردازیم. در ابتدا سه متغییر از نوع private تعریف می کنیم که به وسیله ی متد های داخل فرم مورد استفاده قرار خواهند گرفت. متغییر intAttemptCount برای شمارش تعداد مراتبی است که کاربر سعی کرده است وارد سیستم شود و با شکست مواجه شده است. هنگامی که مقدار این متغییر به عدد سه برسد برنامه به صورت اتوماتیک بسته خواهد شد. متغییر blnAllowClosing برای مشخص کردن این مورد به کار می رود که آیا فرم می تواند بسته شود یا خیر. این متغییر در طی برنامه و در شرایط مختلف مقادیر مختلفی را قبول خواهد کرد. متغییر intUserID نیز برای ذخیره ی شناسه ی کاربری به کار می رود که توانسته است با موفقیت وارد سیستم شود (به عبارت دیگر نام کاربری و کلمه ی عبور را درست وارد کرده است).

```
public partial class Login : Form
{
      // Private members
```

```
private int intAttemptCount = 0;
private bool blnAllowClosing = false;
private int intUserID;
```

بعد از تعریف متغییر های مورد نیاز، باید سه رویداد برای فرم ایجاد کنیم. رویداد اول LoginFailed است و زمانی فراخوانی می شود که کاربر سه بار کلمه ی کاربری را به صورت نادرست وارد کرده باشد. در این حالت نیاز نیست اطلاعات خاصی به متد هایی که قرار است فراخوانی شوند فرستاده شود، بنابراین بر طبق قراردادی که گفتم متد هایی که یک پارامتر از نوع Object و یک پارامتر از نوع EventArgs دارند می توانند در لیست مربوط به این رویداد قرار بگیرند.
رویداد دوم LogionSucceeded است و هنگامی فراخوانی می شود که کاربر بتواند به درستی وارد سیستم شود. به عبارت دیگر این رویداد زمانی فراخوانی می شود که کاربر نام و کلمه ی عبور را به درستی وارد کرده باشد. متد هایی که هنگام رخ دادن این رویداد فراخوانی می شوند، نیاز دارند شناسه ی کاربری که توانسته است وارد شود را دریافت کنند. پس این متد ها باید یک پارامتر از نوع Object و یک پارامتر از نوع LoginEventArgs (که خود از کلاس EventArgs مشتق شده است) دریافت کنند تا بتوانند در لیست رویداد LoginSucceeded قرار بگیرند.
رویداد آخر نیز رویداد LoginCanceled است و زمانی فراخوانی می شود که کاربر روی دکمه ی Cancel کلیک کند. این رویداد نیز مشابه رویداد LoginFailed است و متد هایی که پارامتری را از نوع Object و پارامتر دیگری را از نوع EventArgs دریافت کنند می توانند در لیست این رویداد قرار بگیرند.

```
// Public delegates
public delegate void _LoginFailed(Object sender,
                                  EventArgs e);
public delegate void _LoginSucceeded(Object sender,
                                  LoginEventArgs e);
public delegate void _LoginCancelled(Object sender,
                                  EventArgs e);

// Public events
public event _LoginFailed LoginFailed;
public event _LoginSucceeded LoginSucceeded;
public event _LoginCancelled LoginCancelled;
```

بعد از تعریف رویداد های مورد نیاز در فرم، یک خاصیت فقط–خواندنی ایجاد می کنیم تا افرادی که از این فرم در برنامه های خود استفاده می کنند به وسیله ی این خاصیت بتوانند به شناسه ی کاربری که وارد سیستم شده است دسترسی داشته باشند.

```
public int UserID
{
     get
     {
          return intUserID;
     }
}
```

بعد از اینکه فرم برنامه در حافظه بار گذاری شد، رویداد Activated آن فراخوانی می شود. در این زمان بـرای اینکـه کـار را برای کاربر ساده تر کنیم ، نامی که کاربر با آن وارد ویندوز شده است را بدست آورده و در کـادر User Name قـرار مـی دهیم. برای بدست آوردن نام کاربر می توانیم از خاصیت UserName در کلاس Environment استفاده کنیم[1]. بعـد از قرار دادن نام کاربر در فیلد User Name، فوکوس را به کادر Password اختصاص می دهیم تا عباراتی که بعـد از ایـن تایپ می شوند در کادر Password قرار گیرند.

```
private void Login_Activated(object sender, EventArgs e)
{
     // Populate the UserName text box
     txtUserName.Text = Environment.UserName;

     // Set the focus to the password text box
     txtPassword.Focus();
}
```

به خاطر دارید هنگامی که در ابتدای برنامه کنترلهای Button را به فرم اضافه کردیم، خاصیت DialogResult آنها را برابر با مقدار OK و یا Cancel قرار دادیم. به این ترتیب هنگامی که کاربر روی یکی از این کنترل ها کلیک کند فرم برنامـه بسته شده و مقدار DialogResult.OK و یا DialogResult.Cancel به عنـوان نتیجـه بـه برنامـه ی احضار کننده ی فرم برگشت داده می شود. اما در این فرم نمی خواهیم همواره این اتفاق رخ دهد، برای مثال نمـی خـواهیم همـواره زمانی که کاربر روی دکمه ی OK کلیک کرد (چه کلمه ی عبور درست وارد شده باشد و چه غلط) فرم برنامـه بـسته شـده و مقـدار DialogResult.OK به عنوان نتیجه برگردد. پس نیاز داریم که هنگام بسته شدن فرم بررسی کنیم که آیا فرم می تواند بسته شود یا نه؟ برای این کار می توانیم از رویداد FormClosing استفاده کنیم. این رویداد زمانی فراخوانی می شود که فرم در حال بسته شدن باشد. در این رویداد مقدار فیلد blnAllowClose را بررسی می کنـیم. اگـر مقـدار ایـن فیلـد برابـر بـا false بود به این معنی است که فرم برنامه نباید بسته شود. بنابراین مقدار Cancel را در پارامتر e برابر بـا true قـرار می دهیم تا از بسته شدن فرم جلوگیری شود.

```
private void Login_FormClosing(object sender,
                               FormClosingEventArgs e)
{
     // If we are not allowing the form to close...
     if (!blnAllowClosing)
     {
          // Set the cancel flag to true
          e.Cancel = true;
     }
}
```

[1] در .NET تعداد کلاس هایی که مانند کلاس Application و یا کلاس Environment اطلاعات لازم و مورد نیاز را در اختیار قرار مـی دهند بسیار زیاد است، به صورتی که صحبت در رابطه با آنها از یک یا چند کتاب بیشتر خواهد شد. برای آشنایی با این کلاسها و خاصیت هـای آنهـا بهتـر اسـت که سیستم راهنمای ویژوال استودیو یا MSDN را مطالعه کنید.

بعد از اتمام این قسمت ها به کنترل های Button موجود در فرم می پردازیم. هنگامی که کاربر روی دکمه ی OK کلیک کرد، ابتدا باید مطمئن شویم که کادر User Name خالی نیست و نام کاربری در آن وارد شده است. برای این کار ابتدا متد Trim را برای مقدار خاصیت Text فراخوانی می کنیم تا تمام فضاهای خالی اطراف آن را حذف کند، سپس طول متن باقی مانده را بررسی می کنیم که برابر صفر است یا نه؟

اگر نام کاربری وارد شده بود، مقدار فیلد Password را با کلمه ی عبور کاربر مقایسه می کنیم (در اینجا برای سادگی کار، از کلمه ی عبور ثابت "secret" استفاده شده است). در صورتی که کلمه ی عبور درست وارد شده بود یک شناسه برای این کاربر در نظر گرفته و مقدار آن را در فیلد intUserID قرار می دهیم (در اینجا از مقدار ثابت ۲۷ استفاده کرده ایم)، رویداد LoginSucceeded را فراخوانی کرده و مقدار ۲۷ را به آن ارسال می کنیم. در انتها نیز مقدار blnAllowClose را برابر true قرار می دهیم تا فرم Login بتواند بسته شود.

البته در برنامه های واقعی می توانید در این قسمت به بانک اطلاعاتی متصل شوید و کلمه ی عبور کاربر را استخراج کنید. سپس آن را با مقدار وارد شده توسط کاربر بررسی کنید و در صورت درست بودن آن به کاربر اجازه دهید وارد برنامه شود.

```
private void btnOK_Click(object sender, EventArgs e)
{
     // Was a user name entered?
     if(txtUserName.Text.Trim().Length > 0)
     {
          // Was the password correct?
          if(txtPassword.Text == "secret")
          {
               // Successful login, set the User ID
               intUserID = 27;

               // Raise the LoginSucceeded event
               if(this.LoginSucceeded != null)
                    this.LoginSucceeded(this,
                         new LoginEventArgs(intUserID));

               // Turn on the allow closing flag
               blnAllowClosing = true;
          }
```

اگر کلمه ی عبور وارد شده به وسیله کاربر اشتباه باشد، کادر پیغام مناسبی را نمایش داده و مقدار متغییر intAttemptCount را نیز یک واحد اضافه می کنیم. سپس مقدار این متغییر را با عدد ۳ مقایسه می کنیم. اگر این مرتبه ی سومی بود که کاربر کلمه ی عبور را اشتباه وارد می کرد، رویداد LoginFailed را فراخوانی کرده و خاصیت DialogResult فرم را برابر با Cancel قرار می دهیم. در انتها نیز مقدار خاصیت blnAllowClosing را برابر با true قرار می دهیم تا فرم بتواند بسته شود و مقدار Cancel را به عنوان نتیجه برگرداند.

```
          else
          {
               // Inform the user
               // that the password was invalid
```

```
                MessageBox.Show("The password you entered" +
                          " was invalid.","Login");

                // Increment the attempt count
                intAttemptCount += 1;

                // Check the attempt count
                if (intAttemptCount == 3)
                {
                     // Raise the LoginFailed event
                     if(this.LoginFailed != null)
                          this.LoginFailed(this,
                                        new EventArgs());

                     // Set the Cancel dialog result
                     this.DialogResult =
                               DialogResult.Cancel;

                     // Turn on the allow closing flag
                     blnAllowClosing = true;
                }
           }
      }
```

اگر هیچ نامی در کادر User Name وارد نشده بود، کادر پیغامی را نمایش می دهیم که مشخص می کند پر کردن این کادر الزامی است.

```
      else
      {
           // Inform the user
           // that they must supply a user name
           MessageBox.Show("You must supply a User Name.",
                                   "Login");
      }
```

کدی که در متد مربوط به رویداد Click کنترل btnCancel وارد شده است نیز کاملاً واضح است. اگر کاربر روی دکمه Cancel کلیک کند، کافی است رویداد LoginCanceled را فراخوانی کرده و مقدار فیلد blnAllowClosing را نیز برابر با true قرار دهید تا فرم برنامه بتواند بسته شود.

```
private void btnCancel_Click(object sender, EventArgs e)
{
      // Raise the LoginCancelled event
      if(this.LoginCancelled != null)
           this.LoginCancelled(this, new EventArgs());
      // Turn on the allow closing flag
```

```
        blnAllowClosing = true;
}
```

استفاده از کتابخانه ی فرم ایجاد شده:

حال که فرم Login را طراحی کردیم، می توانیم از آن در هر قسمت از هر برنامه ای که لازم بود استفاده کنیم. در این بخش برای بررسی عملکرد این فرم، یک برنامه ی ویندوزی جدید را به این Solution اضافه می کنیم، دقیقاً مانند کاری که برای تست کنترل ایجاد شده در قسمت های قبل انجام می دادیم.

امتحان کنید: استفاده از فرم Login در یک برنامه

۱) با استفاده از نوار منوی ویژوال استودیو گزینه ی File ➔ Add ➔ New Project... را انتخاب کنید تا کادر Add New Project نمایش داده شود. سپس با استفاده از این کادر یک برنامه ی تحت ویندوز جدید به نام Secure Login به برنامه ی قبلی اضافه کنید.

۲) در پنجره ی Solution Explorer روی نام پروژه ی Secure Login کلیک راست کرده و از منوی باز شده گزینه ی Set as Startup Project را انتخاب کنید. به این ترتیب ویژوال استودیو این پروژه را به عنوان پروژه ی آغازین در نظر می گیرد.

۳) در این مرحله نیاز داریم که یک ارجاع از پروژه ی FormsLibrary به پروژه ی Secure Login اضافه کنیم تا بتوانیم از فرم داخل پروژه ی FormsLibrary استفاده کنیم. در پنجره ی Solution Explorer روی پروژه ی Secure Login کلیک راست کرده و از منوی باز شده گزینه ی Add Reference را انتخاب کنید. در کادر Add Reference روی قسمت Projects کلیک کنید. در لیست نمایش داده شده در این قسمت، پروژه ی FormsLibrary را انتخاب کرده و روی دکمه ی OK کلیک کنید.

۴) با استفاده از جعبه ابزار یک کنترل Label به فرم اضافه کرده و خاصیت Name آن را برابر با lblUserID قرار دهید.

۵) حال به قسمت ویرایشگر کد مربوط به کلاس Form1 بروید. برای نوشتن کد های این قسمت لازم است که از فضای نام FormsLibrary استفاده کنیم، پس دستور زیر را به ابتدای کدها اضافه کنید:

```
using FormsLibrary;
```

۶) به قسمت طراحی فرم Form1 برگشته و روی قسمت خالی فرم دو بار کلیک کنید تا متد مربوط به رویداد Load آن ایجاد شود. سپس کد زیر را در این متد قرار دهید:

```
private void Form1_Load(object sender, EventArgs e)
{
    using (Login objLogin = new Login())
    {
        if (objLogin.ShowDialog(this) ==
```

```
                    System.Windows.Forms.DialogResult.OK)
            {
                // Update the label with the User ID
                lblUserID.Text = "User ID = " +
                                    objLogin.UserID;
            }
            else
            {
                // Inform the user that the login failed
                MessageBox.Show("Login Failed");
                // Close this form since the login failed
                this.Close();
            }
        }
}
```

۷) برنامه را اجرا کنید. مشاهده خواهید کرد که فرم Login نمایش داده شده و در کادر User Name این فرم، نام کاربری که با آن وارد ویندوز شده اید نیز قرار دارد. در کادر Password کلمه ی عبوری به جز secret را وارد کرده و روی دکمه ی OK کلیک کنید. مشاهده می کنید که یک کادر پیغام نمایش داده می شود و می گوید که کلمه ی عبور وارد شده اشتباه است. این کار را دو بار دیگر تکرار کنید. به این ترتیب یک کادر پیغام نمایش داده شده و رخ دادن رویداد LoginFailed را عنوان می کند. در انتها نیز برنامه بسته می شود.

۸) بار دیگر برنامه را اجرا کرده و زمانی که فرم Login نمایش داده شد، روی دکمه ی Cancel کلیک کنید. مجدداً یک کادر پیغام نمایش داده شده و برنامه بسته خواهد شد.

۹) حال برنامه را برای آخرین بار اجرا کرده و کلمه ی secret را در کادر Password وارد کنید، البته دقت کنید که در این قسمت کلمه ی عبور به اندازه ی حروف حساس است. حال روی دکمه ی OK کلیک کنید. مشاهده خواهید کرد که فرم Login بسته شده و فرم Form1 نمایش داده می شود. عدد ۲۷ نیز به عنوان شناسه ی کاربر در کنترل Label فرم نمایش داده می شود.

چگونه کار می کند؟

در این برنامه با یک روش خوب و کارآمد برای اضافه کردن فرم های ورود امن به برنامه های خود آشنا شدید. اگر کاربر نتواند کلمه ی عبور و نام کاربری را وارد کند، برنامه بدون اینکه حتی فرم اصلی را به کاربر نمایش دهد بسته خواهد شد.
اولین کاری که در این قسمت انجام می دهیم این است که یک ارجاع از پروژه ی FormsLibrary در پروژه ی Secure Login ایجاد کنیم. به این ترتیب می توانیم از کلاسهای موجود در پروژه ی FormsLibrary نیز در برنامه استفاده کنیم. سپس با استفاده از راهنمای using، فضای نام FormsLibrary را به برنامه اضافه می کنیم. البته این کار ضروری نیست، ولی برای اینکه نخواهیم در طی برنامه نام کامل کلاسها را وارد کنیم بهتر است این فضای نام را اضافه کنیم.

```
using FormsLibrary;
```

کدی که در متد مربوط به رویداد Load فرم قرار داده ایم کاملاً واضح و ساده است. در ابتدا تمام کد را درون یک بلاک که با استفاده از دستور using ایجاد شده است قرار داده ایم. در فصلهای قبلی گفتم که کلمه ی using به دو صورت می تواند مورد استفاده قرار بگیرد: در حالت اول این کلمه می تواند به صورت راهنمای using برای اضافه کردن یک فضای نام به برنامه مورد استفاده قرار گیرد، که این حالت را تاکنون به مراتب مشاهده کرده اید.

در حالت دوم این کلمه به صورت دستور using به کار می رود. اگر بخواهیم از یک شیئ سنگین که فضای زیادی از حافظه را اشغال می کند استفاده کنیم، بهتر است موقع نیاز آن را ایجاد کرده و بعد از استفاده نیز بلافاصله آن را از بین ببریم. برای اطمینان از این که این شیئ فقط در زمان مورد نیاز وجود دارد، از دستور using به این صورت استفاده می کنیم که در پرانتز جلوی این دستور شیئ را ایجاد کرده و سپس یک بلاک بعد از دستور ایجاد می کنیم. در داخل این بلاک می توانیم از آن شیئ استفاده کنیم، اما هنگامی که برنامه به انتهای بلاک برسد شیئ را نابود می کند و این شیئ در خارج بلاک قابل دسترسی نخواهد بود.

فرم Login نیز یک شیئ سنگین به شما می رود. بنابراین با استفاده از دستور using یک بلاک ایجاد کرده و در داخل آن از این فرم استفاده می کنیم. به محض اینکه برنامه به انتهای بلاک برسد، این شیئ را نابود کرده و حافظه ی اشغال شده به وسیله ی آن را نیز آزاد می کند.

```
using (Login objLogin = new Login())
{

}
```

در داخل دستور using از یک دستور if استفاده می کنیم تا فرم Login ایجاد شده را نمایش داده و نتیجه ی برگشت داده شده از آن را بررسی کنیم. برای نمایش یک فرم می توانیم از متد Show و یا متد ShowDialog استفاده کنیم. در اینجا از متد ShowDialog استفاده کرده و پارامتر this را به آن ارسال می کنیم. این پارامتر مشخص می کند که کلاس جاری یعنی کلاس Form1، فرم Login را نمایش داده است. به عبارت دیگر این پارامتر مشخص می کند که Form1 به عنوان مالک فرم Login به شمار می رود. در انتها نیز نتیجه ی برگشت داده شده از فرم را با ثابت OK از شمارنده ی DialogResult مقایسه می کنیم.

اگر متد ShowDialog نتیجه ی DialogResult.OK را برگرداند، به این معنی است که کاربر توانسته است نام کاربری و کلمه ی عبور را به درستی وارد کند. پس متن کنترل lblUserID را برابر با شناسه ی موجود در فرم Login قرار داده و اجازه می دهیم که Form1 در حافظه بارگذاری شده و نمایش داده شود.

اگر فرم Login مقدار OK را برگشت ندهد، پس حتماً مقدار Cancel را برمی گرداند. در این صورت برنامه وارد قسمت else می شود. در این قسمت نیز یک کادر پیغام حاوی عبارت Login Failed نمایش داده می شود. سپس متد Close مربوط به فرم جاری (که با کلمه ی کلیدی this قابل دسترسی است)، فراخوانی شده و برنامه بسته می شود.

```
using (Login objLogin = new Login())
{
     if (objLogin.ShowDialog(this) ==
               System.Windows.Forms.DialogResult.OK)
     {
          // Update the label with the User ID
          lblUserID.Text = "User ID = " +
                              objLogin.UserID;
     }
     else
```

```
        {
            // Inform the user that the login failed
            MessageBox.Show("Login Failed");
            // Close this form since the login failed
            this.Close();
        }
    }
```

استفاده از رویداد های موجود در کتابخانه ی فرم:

حال که یک روش ساده استفاده از فرم Login را مشاهده کردید، بهتر است از این فرم به صورت کامل تـری اسـتفاده کنـیم. در بخش امتحان کنید بعد برنامه ای ایجاد خواهیم کرد که از فرم Login استفاده کند و در طول این برنامه رویداد هـای موجـود در فرم Login را نیز به کار خواهیم برد.

امتحان کنید: استفاده از رویداد های موجود در فرم Login

۱) در محیط ویژوال استودیو و در همان Solutionای که دو پـروژه ی قبلـی را در آن ایجـاد کردیـد، بـه پنجـره ی Solution Explorer بروید و روی نام solution کلیک راست کرده و از منوی باز شده گزینه ی Add ➔ New Project… را انتخـاب کنیـد. در کـادر Add New Project گزینـه ی Windows Application را از قـسمت Templates انتخـاب کـرده و عبـارت Access Control را در کادر Name وارد کنید. در انتها نیز روی دکمه ی OK کلیک کنید تا پروژه ی جدید ایجاد شود.

۲) حال باید این پروژه را به عنوان پروژه ی آغازین در این solution مـشخص کنـیم. بنـابراین روی نـام پـروژه ی Access Control در پنجره ی Solution Explorer کلیک راست کرده و گزینه ی Set as Startup Project را انتخاب کنید.

۳) مجدداً باید یک ارجاع از پروژه ی FormsLibrary را به این پروژه نیز اضافه کنیم. برای این کار روی پروژه ی Access Control در پنجره ی Solution Explorer کلیک راست کرده و از منوی باز شـده گزینـه ی Add Reference… را انتخـاب کنیـد. در کـادر Add Reference روی قـسمت Projects کلیک کنید. مشاهده خواهید کرد که دو پروژه ی دیگر ایـن solution در لیـست ایـن قـسمت نمایش داده شده اند. از این لیست پروژه ی FormsLibrary را انتخاب کرده و روی دکمه ی OK کلیک کنید.

۴) با استفاده از جعبه ابزار یک کنترل Button را به فرم برنامه ی Access Control اضافه کرده، خاصـیت Name آن را برابر با btnLogin و خاصیت Text آن را برابر با Login قرار دهید.

۵) یک کنترل Label نیز به فرم اضافه کرده و آن را زیر کنترل Button قرار دهید. سپس خاصـیت Name ایـن کنترل را با مقدار lblMessage تنظیم کنید. فرم تکمیل شده ی این برنامه باید مشابه شکل ۱۳-۷ باشد.



شکل ۱۳-۷

۶) حال به قسمت ویرایشگر کد Form1 بروید و با استفاده از دستور زیر، فضای نام FormsLibrary را به فرم این برنامه نیز اضافه کنید:

```
using FormsLibrary;
```

۷) بعد از اضافه کردن فضای نام FormsLibrary، یک شیئ از کلاس Login را در کلاس Form1 ایجاد می کنیم تا در این فرم بتوانیم از فرم Login استفاده کنیم. البته فعلاً به این شیئ مقدار اولیه نمی دهیم. برای این کار کد زیر را به ابتدای کلاس Form1 اضافه کنید:

```
public partial class Form1 : Form
{
    Login objLogin;
```

۸) برای استفاده از رویدادهای موجود در فرم Login ابتدا باید متد هایی را با ساختارهایی که تعیین کردیم ایجاد کنیم. ابتدا متدی برای رویداد LoginCanceled ایجاد می کنیم. برای این کار متد زیر را به کلاس Form1 اضافه کنید:

```
private void objLogin_LoginCancelled(Object sender,
                                     EventArgs e)
{
    lblMessage.Text = "Login Cancelled!";
}
```

۹) سپس باید متدی را برای رویداد LoginFailed و نیز رویداد LoginSucceeded ایجاد کنیم. این متد ها نیز باید دارای ساختار تعیین شده باشند. بنابراین کد زیر را به کلاس Form1 اضافه کنید تا این دو متد ایجاد شوند:

```
private void objLogin_LoginFailed(Object sender,
                                  EventArgs e)
{
    lblMessage.Text = "Login Failed!";
}

private void objLogin_LoginSucceeded(Object sender,
                                     LoginEventArgs e)
{
```

```
        lblMessage.Text = "The Login was successful for " +
                          "the UserID: " + e.UserID;
}
```

۱۰) یک قسمت دیگر از کد باقی مانده است تا برنامه ی این قسمت تکمیل شود. به بخش طراحی فرم Form1 برگردید و روی کنترل btnLogin دو بار کلیک کنید تا متد مربوط به رویداد Click آن ایجاد شود، سپس کد زیر را در این متد وارد کنید:

```
private void btnLogin_Click(object sender, EventArgs e)
{
        objLogin = new Login();

        objLogin.LoginCancelled += new Login._LoginCancelled(
                                   objLogin_LoginCancelled);
        objLogin.LoginFailed += new Login._LoginFailed(
                                   objLogin_LoginFailed);
        objLogin.LoginSucceeded += new Login._LoginSucceeded(
                                   objLogin_LoginSucceeded);

        objLogin.ShowDialog(this);
}
```

۱۱) حال برنامه را اجرا کرده و هنگامی که Form1 نمایش داده شد، روی دکمه ی Login کلیک کنید. در فرم Login روی دکمه ی Cancel کلیک کنید تا این فرم بسته شود. مشاهده می کنید که کنترل Label در Form1 متن Login Cancelled را نمایش می دهد.

۱۲) مجدداً روی دکمه ی Login کلیک کرده و در کادر Password فرم Login کلمه ای به جز secret را وارد کنید. سه بار روی دکمه ی OK کلیک کنید. مشاهده خواهید کرد که کنترل Label در فرم Form1 عبارت Login Failed را نمایش می دهد.

۱۳) برای آخرین بار روی دکمه ی Login کلیک کرده و زمانی که فرم Login نمایش داده شد، عبارت secret را در کادر Password وارد کنید. با کلیک روی دکمه ی OK مشاهده خواهید کرد که کنترل Label در Form1 وارد شدن یک کاربر با شناسه ی ۲۷ را عنوان می کند.

چگونه کار می کند؟

در این برنامه روشی معرفی شد که با استفاده از آن می توانید دسترسی افراد مختلف را به بعضی از قسمتهای برنامه محدود کنید. معمولاً همه ی کاربران به اغلب قسمتهای یک برنامه دسترسی دارند، اما ممکن است بخواهید که فقط افراد خاصی به قسمتی از برنامه دسترسی داشته باشند. در این صورت می توانید هنگامی که یک کاربر سعی کرد به آن قسمت از برنامه وارد شود، کادر Login را نمایش دهید. سپس با استفاده از رویداد های مختلفی که در این فرم ایجاد کرده ایم تعیین کنید که آیا کاربر می تواند وارد آن قسمت شود یا نه؟

این پروژه نیز تقریباً مانند برنامه ی قبل است. ابتدا یک ارجاع به برنامه ی FormsLibrary را در آن ایجاد کرده و فضای نام FormsLibrary را نیز به آن اضافه می کنیم. سپس یک شیئ از نوع Login را در ابتدای کلاس ایجاد می کنیم تا تمام متد ها بتوانند به آن دسترسی داشته باشند.
برای اینکه بتوانیم رویداد های موجود در فرم Login را کنترل کنیم، باید متد هایی که می توان در لیست این رویداد ها اضافه کرد را ایجاد کنیم. همانطور که می دانید این متد ها باید دارای ساختار خاصی باشند و ساختار آنها نیز در کلاس Login تعریف شده است.
بنابراین سه متد به نام های objLofin_LoginCancelled ،objLogin_LoginFailed و objLogin_LoginSucceeded ایجاد می کنیم تا آنها را به لیست رویداد های فرم Login اضافه کنیم. نام این متد ها را می توانیم به صورت دلخواه انتخاب کنیم. معمولاً در ویژوال استودیو نام متد هایی که برای رویداد خاصی هستند شامل نام کنترل به همراه با نام رویداد است. در این قسمت نیز از این قاعده پیروی می کنیم. در این قسمت نام هر متد مشخص می کند که مربوط به چه رویدادی است، بنابراین ساختار آن متد را نیز برابر با ساختار مورد نیاز برای آن رویداد در نظر می گیریم.

```
private void objLogin_LoginCancelled(Object sender,
                                     EventArgs e)
{
     lblMessage.Text = "Login Cancelled!";
}

private void objLogin_LoginFailed(Object sender,
                                     EventArgs e)
{
     lblMessage.Text = "Login Failed!";
}

private void objLogin_LoginSucceeded(Object sender,
                                     LoginEventArgs e)
{
     lblMessage.Text = "The Login was successful for " +
                       "the UserID: " + e.UserID;
}
```

درون هر یک از این رویداد ها نیز کد مناسبی قرار می دهیم تا در صورتی که رویداد مربوطه رخ داد و این متد ها اجرا شدند، متن مناسبی در فرم نمایش داده شود.
هنگامی که کاربر روی دکمه ی Login کلیک کرد، باید فرم Login را نمایش دهیم. برای این کار ابتدا باید به شیئ objLogin که در ابتدای کلاس ایجاد کردیم مقدار اولیه دهیم.

```
     objLogin = new Login();
```

بعد از انجام این کار باید متد هایی را که در مرحله ی قبل ایجاد کردیم به رویداد های موجود در فرم Login اضافه کنیم. برای اضافه کردن یک متد به یک رویداد، ابتدا باید یک شیئ جدید از نوع delegateای که ساختار متدهای مورد نیاز برای آن رویداد را مشخص می کرد ایجاد کرده و نام متد را به عنوان پارامتر به این شیئ ارسال کنیم. سپس این شیئ را با استفاده از عملگر =+ به رویداد اضافه کنیم.

```
objLogin.LoginCancelled += new Login._LoginCancelled(
                           objLogin_LoginCancelled);
objLogin.LoginFailed += new Login._LoginFailed(
                           objLogin_LoginFailed);
objLogin.LoginSucceeded += new Login._LoginSucceeded(
                           objLogin_LoginSucceeded);
```

به این ترتیب متد `objLogin_LoginCancelled` بـه رویـداد `LoginCancelled` اضـافه شـده، متـد `objLogin_LoginFailed` به رویداد `LoginFailed` اضافه شده و ....
در انتها نیز کافی است با استفاده از متد `ShowDialog` فرم `Login` را نمایش دهیم.

```
objLogin.ShowDialog(this);
```

در این قسمت دو روش برای استفاده از فرم `Login` را مشاهده کردید و همانطور که در ابتدای هر کدام از ایـن روش هـا گفـتم، این فرم می تواند در هر کدام از این شرایط (که کاملاً متفاوت از هم نیز هستند) مورد استفاده قرار گیرد. تنها کاری کـه کـافی اسـت انجام دهید این است که توابع و متدهای داخل فرم را به گونه ای تغییر دهید که نام کاربری و کلمه ی عبـور را بـه درسـتی بررسـی کرده و نتیجه را به فرم اصلی برگرداند.

نتیجه:

در این فصل مشاهده کردید که چگونه می توان یک رابط کاربری را در چند قسمت از برنامه و یا حتی در چند برنامه ی مختلف مورد استفاده قرار داد. این کار به دو روش امکان پذیر است. اول اینکه یک کنترل سفارشی با استفاده از کنترل های موجود ایجاد کـرده و سپس کارایی های مورد نیاز خود را با اضافه کردن متد ها، خاصیت ها و یا رویداد های جدید در این کنترل قرار دهید. سپس از ایـن کنترل مانند کنترلهای دیگر از قبیل `TextBox` و یا `Button` در برنامه استفاده کنید.
روش دیگر این است که فرمی را طراحی کرده و تمام متد ها و خاصیتهای مورد نیاز را به این فرم اضافه کنید. سپس در هر قسمت از هر برنامه ای که به این فرم نیاز داشتید، آن را فراخوانی کرده و از آن استفاده کنید. به این ترتیب می توانید ظاهری پیوسته و پایـدار برای برنامه های خود ایجاد کنید.
در پایان این فصل باید با موارد زیر آشنا شده باشید:

- یک کنترل ویندوزی چیست و چگونه کار می کند؟
- چگونه یک کنترل ویندوزی ایجاد کنیم؟
- چگونه متد ها، خاصیت ها و یا رویداد هایی را به کنترل خود اضافه کنیم؟
- زمان اجرا و زمان طراحی چیستند و چه تفاوت هایی دارند؟
- چگونه یک کتابخانه ی کلاس ایجاد کنیم که دارای فرم های عمومی پر کاربرد باشند؟

تمرین:

خاصیتی به نام `SuppressMsgBox` از نوع `Boolean` را به کنترل `MyNamespace` اضافه کنید. سپس متد مربوط به رویداد `Click` کنترلهای `Button` موجود در فرم را به گونه ای تغییر دهید تا در صورتی که مقدار این خاصیت برابر با `False` بود کادر پیغامی را نمایش دهند و در غیر این صورت از نمایش کادر پیغام خودداری کنند.

فصل چهاردهم: ایجاد برنامه های گرافیکی

در برنامه هایی که تاکنون در ط.ل این کتاب انجام داده اید، رابط گرافیکی برنامه را از ابتدا با استفاده از کنترلهای موجـود در ویـژوال استودیو ایجاد می کردید. اما خوب است بدانید زمانی که از ویژوال استودیو برای برنامه نویسی استفاده می کنید، هیچ اجباری نیـست که از کنترل هایی با شکل و شمایل موجود استفاده کنید. بلکه می توانید خودتان محیط برنامه را به صورت دلخواه رسـم کنیـد و بـه این ترتیب قادر خواهید بود که ظاهر برنامه را به هر شکلی که تمایل دارید طراحی کنید.
در طی این فصل نگاهی کلی بر توابع و قابلیتهای موجود در ویژوال استودیو در رابطه با گرافیک و رسم دو بعـدی خـواهیم داشـت و سعی می کنیم که با مفاهیم نوشتن برنامه های گرافیکی پیچیده آشنا شویم. علاوه بر این سعی می کنـیم کـه بعـضی از قابلیتهـای ویژوال استودیو در رابطه با برنامه های چند رسانه ای را بررسی کرده و با استفاده از فایلهای متداولی مانند `.jpg` ، `.png` و یـا `.gif` در برنامه ها آشنا شویم.
در این فصل:

- در رابطه با فضای نام `System.Drawing` مطالبی را خواهید آموخت.
- نحوه ی استفاده از کلاسهای `Pen` و `Brush` را مشاهده خواهید کرد.
- با چگونگی انتخاب و استفاده از رنگهای مختلف آشنا خواهید شد.
- با نحوه ی تغییر اندازه ی تصویر آشنا خواهید شد.
- یک برنامه مشابه `Paint` ایجاد خواهید کرد.

ایجاد یک برنامه ی Paint ساده:

در این قسمت می خواهیم یک برنامه ی `Paint` ساده ایجاد کنیم. برای این کار لازم است که چند کنترل سفارشی ایجاد کرده و سپس با استفاده از آنها در یک پروژه ی ویندوزی، برنامه ی `Paint` را ایجاد کنیم.

ایجاد یک پروژه همراه با کنترل های سفارشی:

انگیزه ی استفاده از کنترلهای سفارشی برای ایجاد این برنامه بسیار ساده است: همانطور که می دانید همواره بهتر است یک برنامه را به مؤلفه ها و کامپوننت های کوچک تقسیم کنیم. با استفاده از این تکنیک اگر بعدها بخواهید در قسمتی از برنامه ی خـود از برنامـه ای مشابه این مورد استفاده کنید، به سادگی می توانید این کار را انجام دهید.
کنترل هایی که برای استفاده در این برنامه ایجاد خواهیم کرد از یک جنبه با کنترل هایی که در قسمتهای قبلی با آنهـا آشـنا شـدید تفاوت دارند. این کنترل ها خود وظیفه دارند که ظاهر و شکل خود را در یک فرم ترسیم کنند و مانند کنترلهای قبلی که ایجـاد مـی کردیم به وسیله ی کلاس پایه در فرم ترسیم نمی شوند[1].

[1] به چنین کنترل هایی `Owner-Draw User Control` گفته می شود.

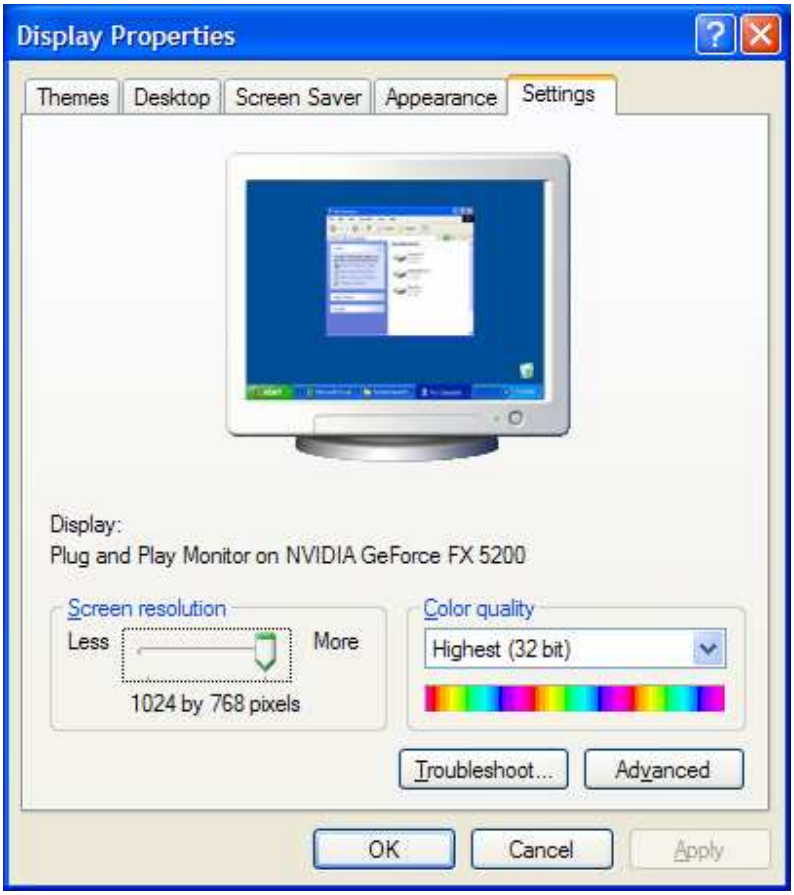
امتحان کنید: ایجاد پروژه

۱) با استفاده از ویژوال استودیو یک برنامه ی تحت ویندوز جدید به نام `MyPaint` ایجاد کنید.
۲) در پنجره ی `Solution Explorer` روی نام پروژه ی `MyPaint` کلیک راست کرده و از منوی باز شده گزینه ی ...`Add ➔ User Control` را انتخاب کنید. سپس نام `PaintCanvas.cs` را در قسمت `Name` وارد کرده و روی دکمه ی `OK` کلیک کنید.
۳) به قسمت طراحی کنترل `PaintCanvas` بروید و روی پس زمینه ی کنترل کلیک کنید تا انتخاب شود. سپس با استفاده از پنجره ی `Properties` خاصیت `BackColor` آن را به `White` تغییر دهید. برای این کار روی خاصیت `BackColor` کلیک کرده و از قسمت `Custom` رنگ سفید را انتخاب کنید.
۴) قبل از این که بتوانید از این کنترل در برنامه استفاده کنید، باید یک بار برنامه را کامپایل کنید. برای این کار با استفاده از نوار منوی ویژوال استودیو گزینه ی `Build ➔ Build MyPaint` را انتخاب کنید. به این ترتیب کنترل `PaintCanvas` در جعبه ابزار قرار می گیرد و می توانید از آن استفاده کنید.
۵) حال به قسمت طراحی فرم `Form1` برگشته و با استفاده از جعبه ابزار کنترل `PaintCanvas` را انتخاب کنید و بر روی فرم قرار دهید. همچنین خاصیت `Dock` این کنترل را نیز به `Fill` تغییر دهید.
۶) برای مرتب تر شدن برنامه خاصیت `Text` فرم را نیز به `My Paint` تغییر دهید.

برنامه های گرافیکی چگونه کار می کنند؟

صفحات کامپیوتر از صدها هزار نقطه ی ریز به نام **پیکسل**[1] تشکیل شده اند. این نقاط بسیار ریز هستند و به صورت عادی نمی توان آنها را در صفحه نمایش از یکدیگر تفکیک کرد، اما هنگامی که در کنار یکدیگر قرار می گیرند می توانند یک تصویر را در صفحه ی کامپیوتر نمایش دهند. به علت اینکه در هر مانیتوری پیکسل ها اندازه ی مشخص و ثابتی دارند، در برنامه های کامپیوتری از آنها به عنوان واحد اندازه گیری استفاده می کنند.
اگر می خواهید بدانید که صفحه نمایش کامپیوتر شما از چند پیکسل تشکیل شده است، تمام برنامه های موجود را کوچک کنید تا بتوانید محیط دسک تاپ را مشاهده کنید. در دسک تاپ کلیک راست کرده و از منوی باز شده گزینه ی `Properties` را انتخاب کنید. در پنجره ی `Display Properties` به قسمت `Settings` بروید. در قسمت چپ پایین این پنجره تعداد پیکسل های تشکیل دهنده ی صفحه نمایش نوشته شده است. برای مثال در شکل ۱۴-۱ صفحه نمایش از ۱۰۲۴ پیکسل افقی و ۷۶۸ پیکسل عمودی تشکیل شده است.
معمولاً در برنامه های گرافیکی از دو روش کلی استفاده می کنند: روش **ترسیم بیت مپی**[2] و روش **ترسیم برداری**[3]. بنابراین بهتر است قبل از هر چیز با مفهوم این دو روش و تفاوتهای آنها آشنا شویم.

[1] Pixel
[2] Bitmap Graphics
[3] Vector Graphics



شکل ۱۴-۱

ترسیم بیت مپی:

در این روش یک فضا برای ترسیم در اختیار شما قرار داده می شود و می توانید با استفاده از ابزارهایی مانند مدادها و یـا قلـم موهـا، تصاویری را در این فضا رسم کنید. در روش ترسیم بیت مپی، فضای رسم به پیکسل ها تقسیم می شود و هر پیکسل نیز می توانـد رنگ مشخصی داشته باشد. برنامه ای گرافیکی که با آن در حال ترسیم در محیط هستید، وظیفه دارد بر اساس رنـگ انتخـابی شـما، نحوه ی حرکت ماوس در صفحه و ابزاری که انتخاب کرده اید، رنگ هر پیکسل را تنظیم کند.
سپس برنامه ی گرافیکی فایل ترسیمی شما را در قالب یک بیت مپ ذخیره می کند، به عبارت دیگر مختصات هر پیکسل و رنگ آن را در فایل قرار می دهد. یک تصویر بیت مپ در حقیقت یک آرایه ی دو بعدی از پیکسل ها است که هر عنصر آن که با اسـتفاده از x و y قابل دسترسی است رنگ آن پیکسل را در خود نگهداری می کند.

نکته: نام بیت مپ یا "Bitmap" از زمانی گرفته شده است که صفحه نمایش کامپیوترها به صورت تک رنـگ بـود. بـه ایـن صورت هر پیکسل به وسیله ی یک بیت نمایش داده می شد که مشخص کننده سیاه و یا سفید بودن آن پیکسل بود.

برای مثال اگر با استفاده از ابزارهای ترسیم بیت مپی یک مستطیل رسم کنید، هنگام ذخیره این مستطیل به صورت یک مجموعه از پیکسل ها و رنگ آنها ذخیره می شود. به این ترتیب بعد از رسم مستطیل و ذخیره ی آن دیگر قـادر نخواهیـد بـود کـه آن را تغییـر

دهید، برای مثال طول و یا عرض مستطیل رسم شده را زیاد کنید. زیرا هنگامی که این مستطیل به صورت یک مجموعه پیکسل در کامپیوتر ذخیره شود، دیگر برنامه ها نمی توانند مستطیل بودن آن را تشخیص دهند. در این حالت برای تغییر آن باید مستطیل جدیدی را بر روی آن رسم کنید.

نکته: تصاویری با پسوند `jpg.`، `gif.` و یا `png.` همه روشهایی برای ذخیره ی تصویر در قالب بیت مپی هستند. البته این فرمت ها با استفاده از روشهای خاص ذخیره سازی باعث کاهش حجم تصاویر و افزایش سرعت دانلود آنها در اینترنت می شوند.

ترسیم برداری:

ترسیم برداری با استفاده از روش کاملاً متفاوتی تصاویر را رسم می کند. برای مثال اگر با استفاده از روش برداری یک مستطیل رسم کنید، برنامه ایجاد شدن یک مستطیل در مکان آن را ذخیره می کند، نه یک مجموعه از پیکسل ها و رنگ آنها را. به عبارت دیگر در روش برداری نحوه ی ترسیم یک تصویر با استفاده از معادلات ریاضی ذخیره می شود. به این ترتیب هنگامی که یک تصویر را با استفاده از این روش رسم کنید، برنامه می تواند جزء جزء این تصویر را تشخیص داده و از هم تفکیک کند. به این ترتیب بعد از ذخیره سازی این تصاویر نیز می توانید هر جزء از آن را انتخاب کرده و تغییر دهید.
با وجود اینکه تصاویر در روش برداری به صورت متفاوتی ذخیره می شوند، اما برای نمایش آنها در صفحه باید این تصاویر را به بیت مپ تبدیل کرد و سپس آنها را نمایش داد. زیرا روشهایی که در رسم برداری استفاده می شوند برای صفحه نمایش قابل فهم نیستند و تصویر ابتدا باید به صورت مجموعه ای از پیکسل ها و رنگ آنها تبدیل شده (بیت مپی) و سپس در صفحه نمایش داده شود. این پروسه معمولاً به نام **رندر کردن**[1] شناخته می شود.
برنامه ی گرافیکی که در این قسمت ایجاد خواهیم کرد از روش ترسیم برداری استفاده می کند. البته هیچ دلیل خاصی برای انتخاب این روش وجود ندارد، فقط با استفاده از این روش بهتر می توان نحوه ی عملکرد ابزارهای ترسیم را در چارچوب `.NET` درک کرد.

ایجاد کلاس GraphicsItem:

در این برنامه می خواهیم دو ابزار ابتدایی برای رسم ایجاد کنیم: دایره و مستطیل. هر کدام از این ابزارها نیاز دارند که بدانند در چه ناحیه ای از محیط رسم (و همچنین در چه قسمتی از صفحه نمایش) باید قرار بگیرند، چه رنگی باید داشته باشند و ... بنابراین یک کلاس پایه ایجاد می کنیم و تمام این موارد را در آن قرار می دهیم. سپس کلاسهای مربوط به دایره و مستطیل را از آن مشتق خواهیم کرد.

امتحان کنید: ایجاد کلاسهای `GraphicsItem` و `GraphicsCircle`

۱) ابتدا باید یک کلاس جدید به برنامه اضافه کنیم. بنابراین در پنجره ی `Solution Explorer` روی نام پروژه کلیک راست کرده و از منوی باز شده گزینه ی `...Class` ➔ `Add` را انتخاب کنید. سپس عبارت `GraphicsItem.cs` را در کادر `Name` وارد کرده و روی دکمه ی `OK` کلیک کنید.

[1] `Rendering`

۲) در ایجاد این کلاس باید از کلاسهای موجود در فـضای نـام System.Drawing اسـتفاده کنـیم. بنـابراین بـا استفاده از دستور زیر، این فضای نام را به برنامه اضافه کنید:

```
using System.Drawing;
```

۳) کد زیر را به GraphicsItem.cs اضافه کنید تا کلاس GraphicsItem ایجاد شود. دقت کنید کـه نباید اجازه دهیم شیئ ای از این کلاس مشتق شود و فقط باید به عنوان کلاس پایه برای دیگر کلاسها مورد استفاده قرار گیرد، بنابراین آن را از نوع abstract تعریف می کنیم[1].

```
abstract class GraphicsItem
{
     // Public members
     public Color color;
     public Boolean IsFilled;
     public Rectangle rectangle;

     // Public methods
     public abstract void Draw(Graphics graphics);

     // Add an item at the given point
     public void SetPoint(int x, int y, int graphicSize,
                          Color graphicColor,
                          Boolean graphicIsFilled)
     {
          // Set the rectangle depending
          // on the graphic and the size
          rectangle = new Rectangle(
                               x – (graphicSize / 2),
                               y - (graphicSize / 2),
                               graphicSize, graphicSize);
          // Set the Color and IsFilled members
          color = graphicColor;
          IsFilled = graphicIsFilled;
     }
}
```

۴) با استفاده از پنجره ی Solution Explorer کلاس دیگری به نـام GraphicsCircle.cs ایجاد کرده و کد زیر را به آن اضافه کنید تا کلاس GraphicsCricle ایجاد شود. دقت کنید که این کـلاس باید از کلاس GraphicsItem مشتق شده و متد Draw نیز در آن override شود.

```
public class GraphicsCircle : GraphicsItem
{
```

[1] برای آشنایی با کلاسهای abstract و دلیل استفاده از آنها به فصل دهم "مباحث پیشرفته در برنامه نویسی شیئ گرا" مراجعه کنید.

```
        public override void Draw(Graphics graphics)
        {
            // Create a new pen
            SolidBrush objSolidBrush = new
                                        SolidBrush(this.Color);

            // Draw the circle
            graphics.FillEllipse(objSolidBrush,
                                        this.Rectangle);
        }
}
```

چگونه کار می کند؟

کلاس `GraphicsItem` در این قسمت شامل تمامی اعضایی است که باید در ابزارهای ابتدایی ترسیم وجود داشته باشند. بنابراین تمام کلاسهای مربوط به ابزارهای ترسیم مانند `GraphicsCircle` باید از این کلاس مشتق شوند، اما نباید اجازه دهیم که شیئ ای به صورت مستقیم از این کلاس نمونه سازی شود. پس این کلاس را از نوع `abstract` معرفی می کنیم.
همچنین در تمام ابزارهای ترسیم باید متدی به نام `Draw` داشته باشیم تا به وسیله ی آن بتوانیم شکل را رسم کنیم. اما پیاده سازی این متد در هر ابزاری متفاوت است. پس این متد را به صورت `abstract` معرفی می کنیم تا تمام کلاس هایی که از `GraphicsItem` مشتق می شوند، این متد را `override` کرده و پیاده سازی مربوط به خود را در آن قرار دهند.
با استفاده از متد `SetPoint` می توانیم با استفاده از موقعیت کنونی ماوس و نیز رنگ و اندازه ی مشخص شده توسط پارامتر ها، به فیلد های موجود در شیئ مقدار دهیم. در این متد ابتدا یک مستطیل ایجاد می کنیم.
معمولاً برای رسم یک دایره از مرکز و شعاع آن استفاده می کنند، اما در `.NET` برای رسم یک دایره باید مستطیل ای که آن دایره را دربر می گیرد را مشخص کنید. بنابراین با استفاده از موقعیت ماوس و نیز اندازه ی شکلی که باید ترسیم شود (که با استفاده از پارامتر `graphicSize` به متد ارسال می شود)، موقعیت گوشه ی بالا و سمت چپ مستطیل را بدست می آوریم و مستطیل را ایجاد می کنیم.
متد سازنده ی کلاس `Rectangle` برای ایجاد یک مستطیل علاوه بر این دو پارامتر به طول و عرض مستطیل نیز نیاز دارد که برای آنها از مقدار `graphicSize` استفاده می کنیم. در انتها نیز رنگ شکل و اینکه آیا باید به صورت توپر رسم شود یا نه را نیز در فیلد های مرتبط در کلاس ذخیره می کنیم.

```
        // Add an item at the given point
        public void SetPoint(int x, int y, int graphicSize,
                             Color graphicColor,
                             Boolean graphicIsFilled)
        {
            // Set the rectangle depending
            // on the graphic and the size
            rectangle = new Rectangle(
                                 x - (graphicSize / 2),
                                 y - (graphicSize / 2),
                                 graphicSize, graphicSize);
```

```
            // Set the Color and IsFilled members
            color = graphicColor;
            IsFilled = graphicIsFilled;
      }
```

بعد از اتمام کلاس GraphicsItem، کلاس GraphicsCircle را از آن مشتق می کنیم. در این کلاس لازم نیست متد و یا فیلد خاصی را ایجاد کنیم و فقط باید متد Draw را که در کلاس پایه به صورت abstract مشخص شده است override کنیم. این متد همانطور که از نام آن مشخص است، وظیفه ی رسم شکل را بر عهده دارد. رسم یک شکل نیز معمولاً به صورت فراخوانی چند متد ساده از کلاس Graphics صورت می گیرد. در این متد احتیاج داریم که یک دایره را در صفحه رسم کنیم، بنابراین از متد FillEllipse استفاده می کنیم. این متد بر حسب پارامترهایی که به آن فرستاده می شود می تواند دایره ها و یا بیضی های توپر را در صفحه رسم کند. برای رسم دایره و یا بیضی معمولی باید از متد DrawEllipse استفاده کنیم.

قبل از اینکه بتوانیم دایره ی مورد نظر را رسم کنیم باید مشخص کنیم که از چه نوع قلم مویی می خواهیم برای رنگ آمیزی شکل استفاده کنیم. برای این کار باید شیئ ای از کلاس Brush و یا کلاسهای مشتق شده از آن استفاده کنیم. در اینجا برای تعیین نوع قلم مو، یک شیئ از کلاس SolidBrush با رنگ مشخص شده در فیلد color ایجاد می کنیم.

در حقیقت با فراخوانی متد FillEllipse به این صورت، به این متد می گوییم که با استفاده از قلم موی مشخص شده در شیئ objSolidBrush، دایره ای را در محدوده ی تعیین شده به وسیله ی rectangle رسم کند.

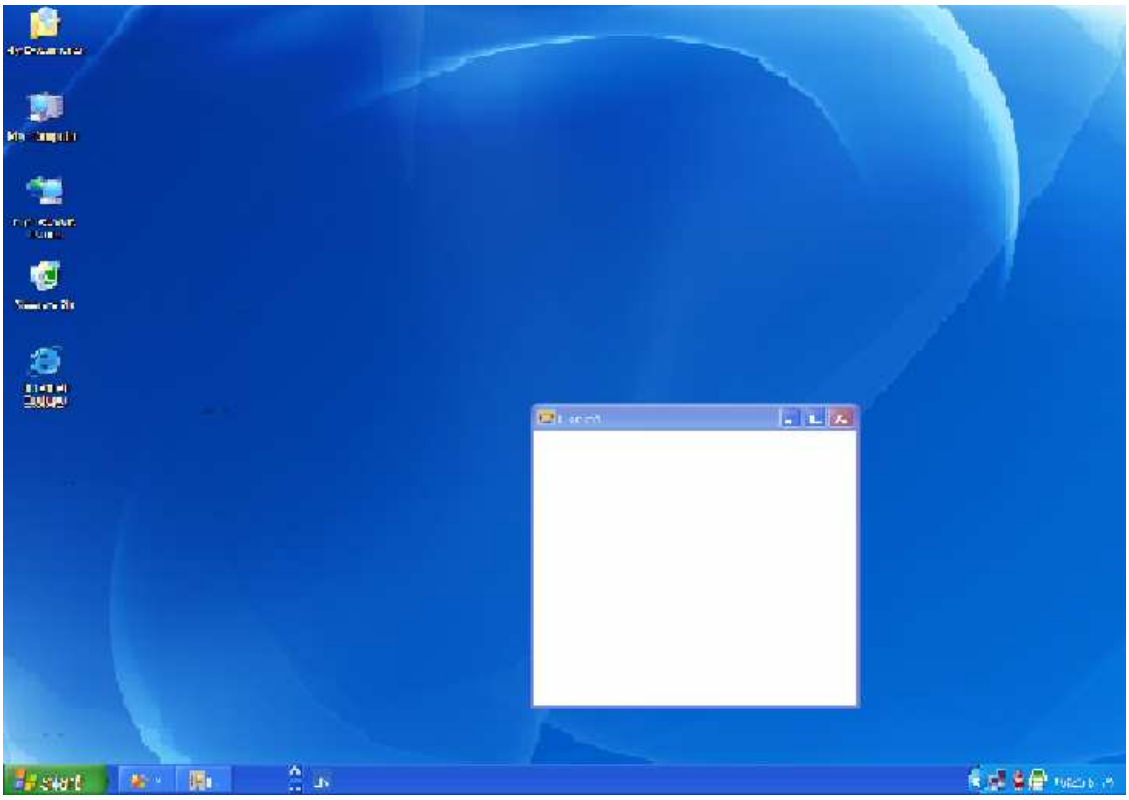
```
public class GraphicsCircle : GraphicsItem
{
      public override void Draw(Graphics graphics)
      {
            // Create a new pen
            SolidBrush objSolidBrush = new
                        SolidBrush(this.Color);

            // Draw the circle
            graphics.FillEllipse(objSolidBrush,
                                    this.Rectangle);
      }
}
```

مختصات صفحه و مختصات برنامه:

هنگامی که بخواهید در یک برنامه رابط کاربری را خودتان رسم کنید، معمولاً به دانستن موقعیت ماوس زیاد احتیاج پیدا خواهید کرد. قبل از این گفتیم که واحد اندازه گیری برای رسم اشکال در .NET پیکسل است. به عبارت دیگر وقتی می خواهیم موقعیت ماوس را بدست آوریم (برای مثال می خواهیم بدانیم ماوس روی کنترل خاصی قرار دارد یا نه)، نتیجه ای که دریافت می کنیم مشخص می کند که ماوس در کدام پیکسل از صفحه قرار دارد. معمولاً اگر ماوس در گوشه سمت چپ بالای فرم قرار داشته باشد، مقدار 0,0 و اگر در گوشه سمت راست در پایین فرم قرار داشته باشد، مقدار 1024*768 به عنوان موقعیت ماوس برگشت داده می شود.

ممکن است این مورد بسیار واضح به نظر برسد، اما باید دقت کنید هنگامی که ماوس در فرم یک برنامه قرار دارد موقعیت ماوس نسبت به گوشه های فرم سنجیده می شود نه نسبت به گوشه های صفحه نمایش.
برای مثال به شکل ۱۴-۲ دقت کنید. در این شکل برنامه ی `MyPaint` که تقریباً در گوشه ی سمت راست پایین فرم قرار دارد نشان داده شده است. صفحه نمایشی که در این شکل نشان داده شده است دارای ۱۰۲۴ پیکسل افقی و ۷۶۸ پیکسل عمودی است، بنابراین فرم برنامه ی `MyPaint` نسبت به گوشه ی بالا سمت چپ صفحه نمایش تقریباً در موقعیت `(500,300)` قرار دارد.



شکل ۱۴-۲

هر فرم دارای یک محدوده است که معمولاً کنترلهای مورد استفاده در برنامه، در آن محدوده نمایش داده می شود. این محدوده شامل تمام محیط طراحی فرم به جز قسمتهای اضافی مانند حاشیه ی فرم، نوار عنوان، نوار منو و نوار ابزار می شود. معمولاً هنگامی که در حال ترسیم یک کنترل در فرم هستید فقط می توانید از این ناحیه استفاده کنید، به این ترتیب نیازی ندارید که موقعیت کنترل ها را نسبت به صفحه نمایش بدانید. بنابراین در فرم برنامه موقعیت ماوس و دیگر کنترل های موجود در این ناحیه نسبت به گوشه ی بالا سمت چپ فرم گزارش می شوند.
در این مثال اگر ماوس خود را در گوشه ی بالا سمت چپ فرم در شکل ۱۴-۲ قرار دهید، دو نوع مختصات می توانید برای آن دریافت کنید:

- مختصات اول ماوس (510,330) است و همانطور که مشاهده می کنید مقداری از مختصات لبه ی فرم نسبت به گوشه ی صفحه نمایش پایین تر است. این مختصات معمولاً به عنوان **مختصات مطلق**[1] هم شناخته می شود.
- مختصات دوم فرم چیزی در حدود (0,0) است که نسبت به لبه ی محیط طراحی فرم سنجیده می شود. این مختصات به مکان فرم در صفحه نمایش بستگی ندارد و فرم در هر نقطه ای از صفحه نمایش که قرار داشته باشد، مختصات این نقطه تقریباً (0,0) خواهد بود. این مختصات معمولاً به عنوان **مختصات نسبی**[2] فرم نیز شناخته می شود.

بررسی حرکات ماوس و رسم اشیای GraphicsCircle:

برای نوشتن این برنامه ی گرافیکی، حرکات ماوس به وسیله ی کاربر را دنبال می کنیم و سپس اشیایی را از یکی از کلاسهای مشتق شده از GraphicsItem ایجاد کرده و در یک لیست نگهداری می کنیم. زمانی که بخواهیم شکل ترسیم شده به وسیله ی کاربر را در صفحه نمایش دهیم، در بین اشیای موجود در این لیست حرکت کرده و متد Draw را در هر کدام از آنها فراخوانی می کنیم. در بخش امتحان کنید بعد، کد این موارد را مشاهده خواهید کرد.

امتحان کنید: رسم اشکال ایجاد شده به وسیله ی کاربر در صفحه

۱) در پنجره ی Solution Explorer روی کنترل PaintCanvas کلیک راست کرده و گزینه ی View Code را انتخاب کنید. سپس شمارنده های مشخص شده در زیر را به کلاس PaintCanvas اضافه کنید. شمارنده ی اول برای مشخص کردن نوع ابزاری که برای رسم به کار رفته است استفاده می شود (دایره و یا مستطیل)، شمارنده ی دوم نیز برای تعیین ضخامت آن ابزار به کار می رود.

```
public partial class PaintCanvas : UserControl
{
    public enum GraphicTools
    {
        CirclePen = 0
    }

    public enum GraphicSizes
    {
        Samll = 4,
        Medium = 10,
        Large = 20
    }
```

۲) فیلد های زیر را نیز به کلاس PaintCanvas اضافه کنید:

---

[1] Absolute Position
[2] Relative Position

```
// Public members
public ArrayList GraphicsItems = new ArrayList();
public GraphicTools GraphicTool = GraphicTools.CirclePen;
public GraphicSizes GraphicSize = GraphicSizes.Medium;
public Color GraphicColor = Color.Black;
```

در لیست زیر کاربرد هر یک از این فیلد ها شرح داده شده است. توجه کنید که برای این فیلد ها مقدار پیش فرض در نظر گرفته ایم تا مقدار دهی اولیه به آنها ساده تر شود.

- **GraphicsItems** شامل لیستی از اشیای مشتق شده از کلاس GraphicsItem است که شکل رسم شده در فرم را تشکیل می دهند.
- **GraphicTool** حاوی ابزاری است که هم اکنون برای ترسیم در فرم مورد استفاده قرار گرفته است.
- **GraphicSize** حاوی اندازه ی ضخامت ابزاری است که هم اکنون برای ترسیم در فرم مورد استفاده قرار گرفته است.
- **GraphicColor** حاوی رنگ ابزاری است که برای ترسیم در فرم به کار می رود.

۳) رسم یک شکل در فرم با استفاده از ماوس از دو مرحله تشکیل می شود. هنگامی که کاربر روی ماوس کلیک می کند و سپس در حالی که کلید ماوس را نگه داشته است، موقعیت ماوس را تغییر می دهد. در این حالت باید هنگامی که کاربر ماوس را حرکت می دهد، به صورت مداوم اشیایی را از نوع GraphicsCircle ایجاد کرده و به لیست GraphicsItems اضافه کنید. با این کار لیست GraphicsItems حاوی اشیایی خواهد بود که کاربر با پایین نگه داشتن کلید ماوس و حرکت آن در فرم رسم کرده است. به این ترتیب هنگامی که ویندوز از شما خواست تا کنترل را رسم کنید، کافی است در بین اشیای موجود در فرم حرکت کرده و متد Draw را در هر یک از آنها فراخوانی کرده تا در صفحه رسم شوند. متد زیر را به کلاس PaintCanvas اضافه کنید:

```
private void DoMousePaint(MouseEventArgs e)
{
     // Store the new item somewhere
     GraphicsItem objGraphicsItem = null;

     // What tool are you using?
     switch(GraphicTool)
     {
          // CirclePen
          case GraphicTools.CirclePen:
          {
               // Create a new graphics circle
               GraphicsCircle objGraphicsCircle =
                                   new GraphicsCircle();

               // Set the point for drawing
               objGraphicsCircle.SetPoint(e.X, e.Y,
                                   (int)GraphicSize,
```

```
                                        GraphicColor, true);

                // Store this for addition
                objGraphicsItem = objGraphicsCircle;
                break;
            }
        }

        // Were you given an item?
        if( objGraphicsItem != null)
        {
            // Add it to the list
            GraphicsItems.Add(objGraphicsItem);

            // Invalidate the control
            this.Invalidate();
        }
}
```

۴) به قسمت طراحی کنترل PaintCanvas بروید و بر روی قسمتی از کنترل کلیک کنید تا انتخاب شود. سپس در پنجره ی Properties روی آیکون Events کلیک کنید تا لیستی از رویدادهای این کنترل را مشاهده کنید. در این لیست رویداد MouseDown را انتخاب کرده و روی آن دو بار کلیک کنید. سپس کد زیر را در متد مربوط به این رویداد وارد کنید:

```
private void PaintCanvas_MouseDown(object sender,
                                   MouseEventArgs e)
{
    // Is the left mouse button down?
    if( e.Button == MouseButtons.Left)
        DoMousePaint(e);
}
```

۵) مجدداً به قسمت طراحی کنترل برگشته و در لیست رویدادها در پنجره ی Properties رویداد MouseMove را انتخاب کرده و روی آن دو بار کلیک کنید. سپس کد مشخص شده در زیر را در متد مربوط به این رویداد وارد کنید:

```
private void PaintCanvas_MouseMove(object sender,
                                   MouseEventArgs e)
{
    // Is the left mouse button down?
    if (e.Button == MouseButtons.Left)
        DoMousePaint(e);
}
```

۶) در آخر نیز همین مراحل را تکرار کرده تا متد مربوط به رویداد Paint را نیز ایجاد کنید. سپس کد زیر را در این متد وارد کنید:

```
private void PaintCanvas_Paint(object sender,
                                    PaintEventArgs e)
{
     // Go through the list
     foreach (GraphicsItem objGraphicsItem in
                                          GraphicsItems)
     {
          // Ask each item to draw itself
          objGraphicsItem.Draw(e.Graphics);
     }
}
```

۷) برنامه را اجرا کنید و با کلیک کردن ماوس روی فرم و حرکت دادن آن شکلی را در فرم رسم کنید.

به این ترتیب برنامه ای مشابه Paint ایجاد کرده اید. البته مشاهده می کنید که اگر خطهای رسم شده در فرم زیاد شوند، برنامه بیش از حد کند خواهد شد. در رابطه با این مورد بعدها بیشتر صحبت خواهیم کرد، اما ابتدا بهتر است خود برنامه و نحوه ی عملکرد آن را بررسی کنیم.

چگونه کار می کند؟

زمانی که کاربر ماوس را روی یک شیئ از کنترل PaintCanvas حرکت می دهد، رویداد MouseMove آن فراخوانی می شود. در متد مربوط به این رویداد بررسی می کنیم که آیا کلید چپ ماوس فشار داده شده است یا نه. اگر این کلید فشار داده شده بود، متد DoMousePaint را فراخوانی کرده و شیئ MouseEventArgs را به عنوان پارامتر به آن می فرستیم.

```
private void PaintCanvas_MouseMove(object sender,
                                        MouseEventArgs e)
{
     // Is the left mouse button down?
     if (e.Button == MouseButtons.Left)
          DoMousePaint(e);
}
```

پروسه ی اصلی ترسیم شکل در متد DoMousePaint انجام می شود. همانطور که مشاهده کردید این متد زمانی فراخوانی می شود که کاربر کلید چپ ماوس را فشار داده و سپس ماوس را روی کنترل حرکت دهد. در این حالت اطلاعات مربوط به موقعیت ماوس به وسیله ی شیئ ای از کلاس MouseEventArgs به این متد ارسال می شود، و باید با استفاده از این اطلاعات و اطلاعاتی دیگر از قبیل نوع ابزار، ضخامت ابزار و ... شکل جدید را ترسیم کنیم.

همانطور که می دانید برای رسم یک شکل با استفاده از ماوس، باید هنگامی که کاربر در نقطه ای از کنترل کلیک می کند (و یـا در حالی که کلید ماوس را نگه داشته است، ماوس را از روی نقطه ای می گذراند)، بر اساس ابزاری که انتخاب کرده است یا یک دایـره ی کوچک و یا یک مستطیل کوچک در آن نقطه رسم می شود. این دایره ها و یا مستطیل های کوچک در حقیقت مانند نقطه عمل می کنند که اگر در کنار یکدیگر قرار بگیرند، شکلی را تشکیل می دهند.
پس در این قسمت باید آرایه ای ایجاد کنیم و نقطه هایی را که شکل از آنها تشکیل می شود را در آن آرایه قرار دهـیم. بـه عبـارت دیگر در این قسمت باید آرایه ای ایجاد کنیم و اشیایی را که از کلاس دایره و یا مستطیل ایجاد می شوند را در آن قرار دهیم. شکلی که در این قسمت باید به این آرایه اضافه شود چه یک دایره باشد (از کلاس `GraphicsCircle` نمونه سازی شده باشد)و چـه یـک مـستطیل، از کـلاس پایـه `GraphicsItem` مـشتق شـده اسـت. پـس مـی توانـد در آرایـه ای از نـوع `GraphicsItem` قرار بگیرد. بنابراین ابتدا یک شیئ از کلاس `GraphicsItem` ایجاد می کنیم تا زمانی که نقطه را ایجاد کردیم، بتوانیم آن را در این شیئ قرار داده و سپس به آرایه اضافه کنیم.

نکته: ممکن است در اینجا این سوال به وجود آید که کلاس `GraphicsItem` از نوع `Abstract` است و نمی تـوان شیئ را از آن ایجاد کرد، پس چگونه در این قسمت این کار را انجام داده ایم؟ دقت کنید تا زمانی که یک شیئ نمونـه سـازی نـشده باشد (با استفاده از دستور `new`)، شیئ ایجاد نشده است. به عبارت دیگر یک شیئ زمانی ایجاد می شود که نمونـه سـازی شـود. در اینجا نیز ما فقط متغیری را ایجاد کرده ایم که بتوانیم شیئ ای از نوع `GraphicsItem` و **یا یکی از کلاسهای مـشتق شده از آن را** در این متغییر ذخیره کنیم. اما به علت اینکه نمی توانیم شیئ از کـلاس `GraphicsItem` را نمونـه سـازی کنیم، پس در این متغییر فقط می توانیم اشیایی را نگهداری کنیم که از کلاس های مشتق شده از `GraphicsItem` (ماننـد `GraphicsCircle`) نمونه سازی شده باشند.
در حقیقت هدف ما نیز در اینجا این است که متغیـری داشـته باشـیم کـه بتوانـد اشـیایی کـه از کلاسـهای مـشتق شـده از کـلاس `GraphicsItem` ایجـاد مـی شـوند را در آن ذخیـره کنـیم. بنـابراین آن متغییـر را بایـد از نـوع کـلاس پایـه یعنـی `GraphicsItem` تعریف کنیم.

```
private void DoMousePaint(MouseEventArgs e)
{
      // Store the new item somewhere
      GraphicsItem objGraphicsItem = null;
```

حال باید تشخیص دهیم که کاربر می خواهد برای رسم نقطه بر روی یک شکل، از یک دایره ی کوچک استفاده کنـد و یـا از یـک مستطیل کوچک و یا ...، این مورد نیز بستگی به ابزاری دارد که کاربر در برنامه برای رسم انتخاب می کند. این ابزار می تواند یکی از ثابت های ی موجود در شمارنده ی `GraphicTools` باشد، که البته فعلاً فقط شامل یک عـضو یعنـی `CirclePen` است.
اگر ابزار انتخاب شده از نوع `CirclePen` باشند به این معنی است که کاربر می خواهد نقطه های ایجاد کننـده ی شـکل بـه صورت یک دایره باشند. بنابراین برای تشکیل این نقطه ها باید از کلاس `GraphicsCircle` استفاده کنیم. پس در ایـن قسمت شیئ ای از نوع `GraphicsCircle` ایجاد می کنیم.

```
switch(GraphicTool)
{
      // CirclePen
      case GraphicTools.CirclePen:
      {
```

```
// Create a new graphics circle
GraphicsCircle objGraphicsCircle =
                    new GraphicsCircle();
```

سپس باید با استفاده از متد SetPoint در این شیئ، اطلاعات لازم در مورد محل قرار گیری این نقطه و نیز اندازه ی آن را مشخص کنیم. محل قرار گیری این نقطه که باید در محل کنونی ماوس باشد، پس دو پارامتر اول را برابر با محل کنونی ماوس که در شیئ e وجود دارد قرار می دهیم. ضخامت نقطه نیز باید با استفاده از یکی از ثابت های موجود در شمارنده ی GraphicSizes تعیین شود اما نوع آن باید عدد صحیح باشد، پس مقدار انتخابی از این شمارنده را به یک عدد صحیح تبدیل کرده و به عنوان پارامتر سوم به متد SetPoint می فرستیم. پارامتر چهارم رنگ نقطه را مشخص می کند، پس مقدار فیلد GraphicColor را به عنوان پارامتر چهارم به این متد می فرستیم. پارامتر آخر نیز مشخص می کند که آیا باید شکل به صورت توپر رسم شود یا نه. در این جا تعیین می کنیم که شکل به صورت توپر رسم شود.

```
// Set the point for drawing
objGraphicsCircle.SetPoint(e.X, e.Y,
                    (int)GraphicSize,
                         GraphicColor, true);
```

بعد از تنظیم قسمت های مختلف نقطه ای که باید در فرم قرار گیرد، نقطه را در شیئ ای از نوع GraphicsItem قرار می دهیم تا بعدا بتوانیم آن را به لیست نقطه های تشکیل دهنده ی فرم اضافه کنیم.

```
// Store this for addition
objGraphicsItem = objGraphicsCircle;
break;
```

به این ترتیب نقطه ی مورد نظر ایجاد شده است. حال بررسی می کنیم که اگر مقداری در شیئ objGraphicsItem قرار گرفته بود (مقدار آن مخالف null بود)، آن را به آرایه ی نقاط تشکیل دهنده ی فرم اضافه می کنیم.

```
// Were you given an item?
if( objGraphicsItem != null)
{
     // Add it to the list
     GraphicsItems.Add(objGraphicsItem);
```

در انتها نیز متد Invalidate را در فرم برنامه (this) فراخوانی می کنیم. با فراخوانی این متد در حقیقت به برنامه می گوییم که قسمتی از ظاهر فرم برنامه تغییر کرده است و ویندوز باید آن را مجدداً رسم کند. هنگامی که ویندوز یک فرم را در صفحه نمایش داد، تا زمانی که لازم نباشد آن را مجدداً رسم نمی کند. این که چه هنگام باید یک کنترل دوباره رسم شود توسط خود ویندوز به صورت اتوماتیک مشخص می شود، اما اگر لازم باشد می توانیم با فراخوانی متد Invalidate به ویندوز بگوییم که فرم را مجدداً رسم کند. به عبارت دیگر با استفاده از این روش به ویندوز می گوییم که ظاهر یکی از کنترل های موجود در فرم برنامه تغییر کرده است و باید فرم را مجدداً رسم کند.

```
     // Invalidate the control
```

```
        this.Invalidate();
    }
```

نکته: اگر با استفاده از کد، فقط ظاهر یکی از کنترلهای موجود در فرم را تغییر دادید، می توانید متد `Invalidate` را در آن کنترل فراخوانی کنید تا ویندوز فقط آن کنترل را مجدداً رسم کند.

خوب، تاکنون زمان حرکت ماوس بر روی فرم را تشخیص داده ایم و در این زمان بر اساس ابزاری که کاربر انتخاب کرده بود، شیئ مناسبی را ایجاد کرده و آن را به لیست نقطه های نمایش دهنده ی فرم اضافه کردیم. همچنین به ویندوز نیز اعلام کردیم که طاهر فرم مجدداً باید در صفحه رسم شود تا تغییرات ایجاد شده نمایش داده شوند. خوب فکر می کنید که چه کارهای دیگری را باید انجام دهیم تا برنامه کامل شود؟

هنگامی که یک فرم و یا یک کنترل درون فرم به ترسیم مجدد نیاز داشته باشد، باید اجازه دهیم که ویندوز خودش تصمیم بگیرد که چه زمانی باید فرم را رسم کند. در سیستم عامل ویندوز برای افزایش سرعت و کارایی، رسم کنترل ها در صفحه از اهمیت کمی برخوردارند، بنابراین ویندوز زمانی این کارها را انجام می دهد که به اندازه ی کافی وقت آزاد داشته باشد. رسم یک کنترل و یا یک فرم در صفحه نمایش از نظر ویندوز یک مورد حیاتی و ضروری تلقی نمی شود، بنابراین نمی توانید از سیستم عامل انتظار داشته باشید هنگامی که یک فرم را به صورت نا معتبر مشخص کردید، ویندوز به سرعت آن را مجدداً در صفحه نمایش رسم کند. احتمالاً این مورد را در هنگام کار با برنامه ها زیاد مشاهده کرده اید که اگر ویندوز در حال انجام فعالیت سنگینی در فرم باشد، پنجره ی فرم به صورت یخ زده شده و هیچ چیز نمایش داده نمی شوند.

بنابراین نباید هیچ موقع ویندوز را مجبور کنید که فرم شما را به سرعت در صفحه رسم کند. در این سیستم عامل بیش از چندین هزار خط کد وجود دارند که بهترین زمان برای رسم مجدد یک کنترل در صفحه نمایش را مشخص می کنند. بنابراین فقط کافی است کنترلی که نیاز به ترسیم مجدد دارد را با استفاده از متد `Invalidate` مشخص کنید و سپس اجازه دهید که ویندوز در اولین فرصت ممکن آن را مجدداً رسم کند.

هنگامی که ویندوز بخواهد یک فرم و یا یک کنترل را مجدداً در صفحه نمایش رسم کند، رویداد `Paint` را در آن کنترل فراخوانی می کند. بنابراین باید متدی را ایجاد کنیم تا هنگام رخ دادن این رویداد این متد اجرا شده و کنترل مورد نظر ما را در فرم رسم کند. در این متد نیز کافی است که با استفاده از متد `Draw` از تک تک نقطه های موجود در فرم بخواهیم که خود را در صفحه رسم کنند. پس با استفاده از یک حلقه ی `foreach` بین تمام نقطه های موجود در لیست حرکت کرده و متد `Draw` را فراخوانی می کنیم.

```
private void PaintCanvas_Paint(object sender,
                                    PaintEventArgs e)
{
    // Go through the list
    foreach (GraphicsItem objGraphicsItem in
                                        GraphicsItems)
    {
        // Ask each item to draw itself
        objGraphicsItem.Draw(e.Graphics);
    }
}
```

هنگامی که ویندوز رویداد `Paint` را فراخوانی کرد، اطلاعات لازم در رابطه با رخ دادن این رویداد را با استفاده از شیئ ای از کلاس `PaintEventArgs` به متد ها می فرستد. این شیئ خاصیت ها و متدهای فراوانی دارد و یکی از آنها خاصیت

Graphics است که شیئ ای را از نوع System.Drawing.Graphics برمی گرداند. این شیئ شامل متد ها و توابع زیادی است که می تواند برای ترسیم مورد استفاده قرار گیرد، بنابراین این شیئ را به عنوان پارامتر به متد Draw می فرستیم تا در صورت نیاز بتواند از آن استفاده کند.

حال که تقریباً با نحوه ی کارکرد این قسمت از برنامه آشنا شدیم، بهتر است علت کند شدن برنامه و چشمک زدن آن هنگام رسم یک شکل در فرم را بررسی کرده و آن را رفع کنیم.

نا معتبر سازی:

کدی که در قسمت قبل وارد کردیم کاملاً درست کار می کند، اما اگر مقداری شکل در صفحه رسم کنیم سرعت برنامه کاهش پیدا کرده و برنامه شروع به چشمک زدن خواهد کرد. دلیل آن نیز این است که هنگام نوشتن برنامه یکی از مهمترین نکات برنامه نویسی را در نظر نگرفتیم: حجم کاری که باید توسط برنامه انجام شود را تا حد ممکن کاهش دهیم. رسم مجدد یک کنترل در صفحه نمایش کاری است که به کندی انجام می شود. هر چه محدوده ای که می خواهیم ترسیم کنیم کوچکتر باشد، سرعت ترسیم آن ناحیه بیشتر خواهد بود و ظاهر برنامه سریعتر ترسیم می شود.
علت چشمک زدن فرم برنامه در این است که ترسیم کنترل توسط ویندوز طی دو مرحله انجام می شود. ابتدا ویندوز تمام محدوده ای که باید دوباره رسم شوند را پاک می کند، به این ترتیب آن ناحیه به رنگ سفید درخواهد آمد. سپس به شما اجازه داده می شود که آن قسمت را مجدّداً ترسیم کنید.
در این قسمت با فراخوانی متد Invalidate در فرم، از ویندوز می خواهید که تمام فرم را مجدداً رسم کند، اما لازم به این کار نیست و می توانید با فراخوانی این متد در قسمتی که نقطه ی جدید قرار گرفته است، از ویندوز بخواهید که فقط آن محدوده را مجدداً رسم کند. به این ترتیب ناحیه ی کوچکتری نیاز به ترسیم مجدد خواهد داشت و سرعت اجرای برنامه افزایش پیدا خواهد کرد.
در بخش امتحان کنید بعد نحوه ی انجام این کار را با هم بررسی خواهیم کرد.

امتحان کنید: نا معتبر سازی یک محدوده ی مشخص

۱) در کلاس PaintCanvas متد DoMousePaint را پیدا کرده و فراخوانی متد Invalidate را به صورت زیر تغییر دهید تا فقط ناحیه ی مشخص شده توسط مستطیل موجود در شیئ objGraphicsItem مجدد رسم شود:

```
    // Invalidate the control
    this.Invalidate(objGraphicsItem.rectangle);
```

۲) برنامه را اجرا کنید. مشاهده خواهید کرد که هنگام رسم شکل، هر چه قدر هم که شکل بزرگ شود فرم چشمک نخواهد زد.

چگونه کار می کند؟

هنگامی که متد SetPoint در شیئ ایجاد شده از کلاس GraphicsCircle را فراخوانی می کنید، این متد مقدار فیلد rectangle را به گونه ای تنظیم می کند که حاوی محدوده ی قرارگیری نقطه ی جدید ایجاد شده باشد.
در این قسمت هنگام فراخوانی متد this.Invalidate مستطیل موجود در این فیلد را به این متد می فرستید و به این ترتیب برای متد مشخص می کنید که فقط این قسمت از فرم مجدداً باید ترسیم شود. بنابراین ویندوز هم فقط این قسمت از فرم را پاک می کند و به شما اجازه می دهد تا آن را رسم کنید.

بهینه سازی کردن رسم:

اگر هنگام کار با برنامه دقت کرده باشید متوجه می شوید که بعد از اینکه مقداری از شکل را در فرم رسم کردید، نقطه هایی که در فرم قرار می دهید به صورت ناهموار رسم می شوند. برای علت این مشکل باید بگوییم هنگامی که یک شکل بزرگ می شود، در حقیقت نقاطی که آن را تشکیل می دهند زیادتر می شوند بنابراین رسم تمام این نقاط در فرم مدت زمان بیشتری طول می کشد. هرچه قدر هم که مدت زمان رسم این نقاط در فرم طولانی تر شود، هنگام حرکت ماوس در صفحه تعدادی از رویدادهای MouseMove نمی توانند فراخوانی شوند و بنابراین هنگام عبور ماوس از آن قسمت از فرم نقطه ای در صفحه رسم نمی شود و شکل به صورت ناهموار در می آید. در بخش امتحان کنید بعد نحوه ی حل این مشکل را با هم مشاهده خواهیم کرد.

امتحان کنید: بهینه سازی رسم

۱) متد PaintCanvas_Paint در کلاس PaintCanvas را پیدا کرده و کد مشخص شده در زیر را به آن اضافه کنید:

```
private void PaintCanvas_Paint(object sender,
                                    PaintEventArgs e)
{
     // Go through the list
     foreach (GraphicsItem objGraphicsItem
                          in GraphicsItems)
          if (e.ClipRectangle.IntersectsWith(
                          objGraphicsItem.rectangle) )
               // Ask each item to draw itself
               objGraphicsItem.Draw(e.Graphics);
}
```

۲) برنامه را اجرا کرده و شکلی را در فرم رسم کنید. مشاهده خواهید کرد که شکل بسیار نرمتر رسم خواهد شد.

چگونه کار می کند؟

کلاس PaintEventArgs حاوی خاصیتی به نام ClipRectangle است که اطلاعات یک مستطیل را در خود نگهداری می کند. این مستطیل محدوده ای از فرم که از نوع نا معتبر مشخص شده است را معین می کند، که این محدوده به نام **مستطیل چیده شده**[1] شناخته می شود. کلاس Rectangle نیز دارای متدی به نام IntersectsWith است که یک مقدار بولین را برمی گرداند و این مقدار مشخص می کند که آیا دو مستطیل همپوشانی دارند یا نه (رو هم قرار می گیرند یا نه)؟

به این ترتیب می توانیم در این حلقه، مستطیل مشخص کننده ی محدوده ی تک تک عناصر موجود در لیست را با مستطیل مشخص کننده ی محدوده ی عنصر کنونی که باید رسم شود، مقایسه کرده و ببینیم که همپوشانی دارند یا نه. در صورتی که این دو مستطیل همپوشانی داشته باشد لازم است که آن را در فرم رسم کنیم و در غیر این صورت می توانیم نقطه ی بعدی در لیست را بررسی کنیم.

دو تکنیکی که برای بهبود برنامه در این قسمت استفاده کردیم (رسم ناحیه ای که لازم است ترسیم شود به جای رسم تمام فرم و نیز رسم نقطه ای که جدیداً ایجاد شده است به جای رسم تمام نقاط) دو نکته ی بسیار مهم در نوشتن یک برنامه ی گرافیکی محسوب می شوند وبا حذف هر یک از آنها از برنامه، نمی توانید به راحتی شکلی را در فرم برنامه ایجاد کنید.

انتخاب رنگ:

حال که یک برنامه ی ساده ی نقاشی ایجاد کردیم، بهتر است کنترلی بسازیم که به وسیله ی آن بتوانیم رنگ ابزار مورد استفاده برای ترسیم در فرم را نیز مشخص کنیم. مانند تمام برنامه های گرافیکی یک پالت رنگ ایجاد خواهیم کرد و به وسیله ی آن پالت، در هر لحظه به کاربر اجازه می دهیم دو رنگ را انتخاب کند: یک رنگ برای کلیک چپ ماوس و رنگ دیگر برای کلیک راست ماوس.

برای ایجاد این کنترل روشهای زیادی وجود دارند، اما منطقی ترین و ساده ترین روش این است که کنترلی حاوی چندین کنترل شبیه Button ایجاد کنیم. سپس رنگ پس زمینه ی هر یک از این کنترل ها را برابر با یکی از رنگهای مورد نظر در پالت رنگ قرار دهیم. به این ترتیب کاربر می تواند با کلیک روی هر کدام از این کنترل ها یک رنگ را انتخاب کرده و از آن برای رسم شکل استفاده کند. در قسمت بعد، نحوه ی ایجاد چنین کنترلی را از پایه بررسی خواهیم کرد.

ایجاد کنترل ColorPalette:

برای اینکه بتوانیم یک کنترل برای پالت رنگ در برنامه بسازیم، باید دو کلاس ایجاد کنیم. کلاس اول، کلاسی است به نام ColorPalette که از کلاس UserControl مشتق می شود. این کلاس برای ترسیم رابط گرافیکی کنترل پالت رنگ مورد استفاده قرار می گیرد. کلاس دوم نیز کلاسی به نام ColorPaletteButton است که برای نمایش رنگهای موجود در پالت به کار می رود.

در ایجاد پالت رنگ، ظاهر و مکان قرارگیری دکمه هایی که برای انتخاب رنگ در صفحه به کار می روند را باید خودمان مشخص کنیم و همچنین باید این ظاهر را هنگام تغییر اندازه ی کنترل نیز حفظ کنیم. بنابراین متدی ایجاد می کنیم تا هنگام رخ دادن رویداد Resize فراخوانی شود و مکان قرارگیری رنگها در کنترل پالت رنگ را تنظیم کند.

---

[1] Clipping Rectangle

هنگامی که رویداد Resize فراخوانی شد، باید مکان قرارگیری هر رنگ را در پالت مشخص کنید. برای این کار از بالا سمت چپ کنترل شروع می کنید و یکی یکی رنگها را در پالت قرار می دهید. سپس هنگامی که به انتهای یک سطر رسیدید، به سطر بعد رفته و رنگها را در سطر جدید قرار می دهید.

امتحان کنید: ایجاد کنترل ColorPalette

۱) با استفاده از پنجره ی SolutionExplorer کلاس جدیدی به نام ColorPaletteButton.cs به برنامه اضافه کرده و سپس با اضافه کردن کد زیر به این کلاس، فضای نام System.Drawing را به برنامه اضافه کنید:

```
using System.Drawing;
```

۲) سپس کد مشخص شده در زیر را در این کلاس وارد کنید:

```
public class ColorPaletteButton
{
     // Publiuc members
     public Color color = System.Drawing.Color.Black;
     public Rectangle rectangle;

     // Constructor
     public ColorPaletteButton(Color objColor)
     {
          color = objColor;
     }

     // Move the button to the given position
     public void SetPosition(int x, int y, int buttonSize)
     {
          // Update the members
          rectangle = new
               Rectangle(x, y, buttonSize, buttonSize);
     }

     // Draw the button
     public void Draw(Graphics graphics)
     {
          // Draw the color block
          SolidBrush objSolidBrush = new SolidBrush(color);
          graphics.FillRectangle(objSolidBrush, rectangle);

          // Draw an edge around the control
          Pen objPen = new Pen(Color.Black);
```

```
            graphics.DrawRectangle(objPen, rectangle);
        }
}
```

۳) حال یک کنترل سفارشی به نام ColorPalette به پروژه ی MyPaint اضافه کنید. در قسمت طراحی کنترل، روی این کنترل کلیک راست کرده و گزینه ی View Code را انتخاب کنید تا کد مربوط به این کنترل نمایش داده شود. سپس فیلد های زیر را به کلاس این کنترل اضافه کنید:

```
    // Public members
    public ArrayList Buttons = new ArrayList();
    public int ButtonSize = 15;
    public int ButtonSpacing = 5;
    public Color LeftColor = Color.Black;
    public Color RightColor = Color.White;
```

این فیلد ها برای موارد زیر مورد استفاده قرار می گیرند:

- Buttons لیستی از دکمه های مربوط به رنگهای درون پالت را نگهداری می کند.
- ButtonSize اندازه ی هر دکمه را در پالت رنگ مشخص می کند.
- ButtonSpacing فضای خالی بین هر یک از دکمه ها را در پالت رنگ تعیین می کند.
- LeftColor رنگی که هم اکنون برای کلید چپ ماوس در نظر گرفته شده است را نگهداری می کند.
- RightColor رنگی که هم اکنون برای کلید راست ماوس در نظر گرفته شده است را نگهداری می کند.

۴) حال متد زیر را نیز به کلاس اضافه کنید:

```
Add a new color button to the control
public void AddColor(Color newColor)
{
    // Create the button
    ColorPaletteButton objColorPaletteButton = new
                        ColorPaletteButton(newColor);

    // Add it to the list
    Buttons.Add(objColorPaletteButton);
}
```

۵) می خواهیم هنگامی که کنترل ایجاد می شود چند رنگ به صورت پیش فرض در آن وجود داشته باشد، بنابراین کدی را درون متد سازنده ی آن قرار می دهیم تا هنگام ایجاد کنترل ده رنگ ابتدایی را به آن اضافه کند. برای این کار کد زیر را به متد سازنده ی کلاس ColorPalette اضافه کنید:

```
public ColorPalette()
{
```

```
        InitializeComponent();

        // Add the colors
        AddColor(Color.Black);
        AddColor(Color.White);
        AddColor(Color.Red);
        AddColor(Color.Blue);
        AddColor(Color.Green);
        AddColor(Color.Gray);
        AddColor(Color.DarkRed);
        AddColor(Color.DarkBlue);
        AddColor(Color.DarkGreen);
        AddColor(Color.DarkGray);
    }
```

۶) به قسمت طراحی کنترل `ColorPalette` برگشته و روی خود کنترل کلیک کنید تا انتخاب شود. سپس با کلیک روی آیکون `Events` در پنجره ی `Properties`، لیست رویدادهای کنترل را نمایش دهید. از این لیست رویداد `Resize` را انتخاب کرده و روی آن دوبار کلیک کنید تا متد مربوط به این رویداد ایجاد شود، سپس کد زیر را به این متد اضافه کنید:

```
    private void ColorPalette_Resize(object sender,
                                     EventArgs e)
    {
        // Declare variables to hold the position
        int intX = 0;
        int intY = 0;

        // Go through the array and position the buttons
        foreach (ColorPaletteButton objColorPaletteButton
                                                in Buttons)
        {
            // Position the button
            objColorPaletteButton.SetPosition(intX, intY,
                                              ButtonSize);

            // Move to the next one
            intX += (ButtonSize + ButtonSpacing);

            // Do we need to go down to the next row
            if (intX + ButtonSize > Width)
            {
                // Move y
                intY += (ButtonSize + ButtonSpacing);

                // Reset x
                intX = 0;
```

```
            }
        }

        // Redraw
        this.Invalidate();
}
```

۷) مجدداً به لیست رویدادها در پنجره ی Properties برگردید و رویداد Paint را انتخاب کرده روی آن دو بار کلیک کنید تا متد مربوط به این رویداد ایجاد شود. سپس کد زیر را به این متد اضافه کنید:

```
private void ColorPalette_Paint(object sender,
                                        PaintEventArgs e)
{
        // Loop through the buttons
        foreach (ColorPaletteButton objColorPaletteButton
                                                in Buttons)
            // Do we need to draw?
            if (e.ClipRectangle.IntersectsWith(
                    objColorPaletteButton.rectangle))
                objColorPaletteButton.Draw(e.Graphics);
}
```

۸) قبل از اینکه بتوانید از کنترل ایجاد شده در فرم اصلی برنامه استفاده کنید باید یک بار پروژه را کامپایل کنید. با استفاده از نوار منوی ویژوال استودیو گزینه ی Build ➔ Build را انتخاب کنید تا برنامه کامپایل شود.

۹) بعد از اینکه برنامه کامپایل شد، به قسمت طراحی فرم مربوط به Form1 بروید. در این فرم کنترل PaintCanvas را انتخاب کرده و با استفاده از پنجره ی Properties مقدار خاصیت Dock آن را به None تغییر دهید. حال پنجره ی فرم را مقداری بزرگتر کنید تا فضای کمی در پایین آن ایجاد شود.

۱۰) با استفاده از قسمت MyPaint Components در جعبه ابزار یک کنترل ColorPalette را در پایین فرم قرار داده و خاصیت Name آن را برابر با paletteColor قرار دهید. همچنین خاصیت Dock این کنترل را نیز با مقدار Bottom تنظیم کنید.

۱۱) حال کنترل PaintCanvas را انتخاب کرده، اگر لازم است اندازه ی آن را در فرم تنظیم کنید و سپس خاصیت Anchor آن را برابر با مقدار Top, Right, Left, Bottom قرار دهید. به این ترتیب فرم شما باید مشابه شکل ۱۴-۳ شده باشد.

۱۲) اندازه ی فرم را مقداری تغییر دهید. مشاهده خواهید کرد که رنگهای موجود در پالت رنگ به صورت مناسب در مکان خود قرار خواهند گرفت.

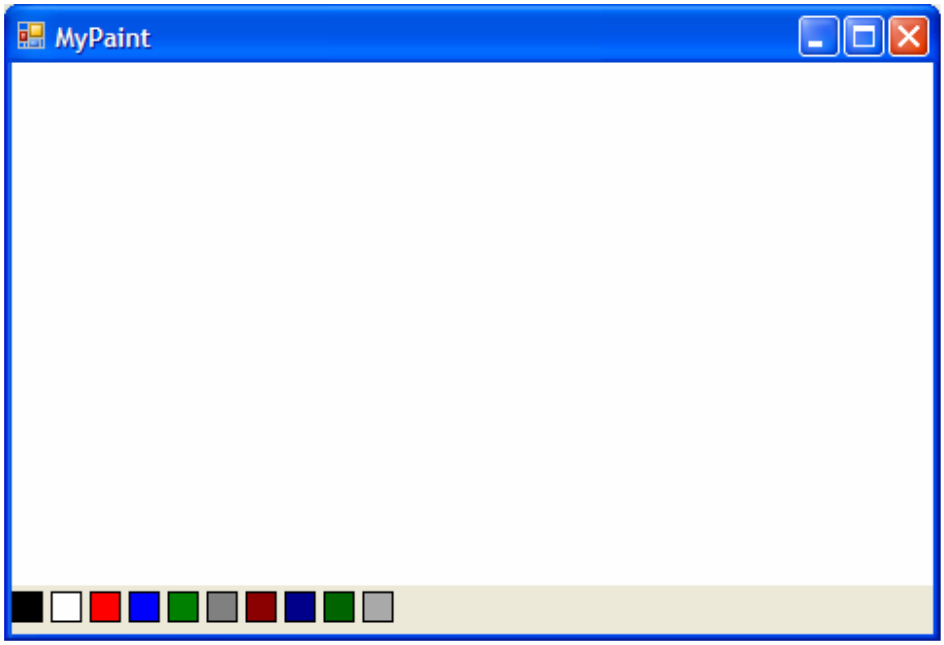
چگونه کار می کند؟

خوشبختانه کلاس ColorPaletteButton قسمت پیچیده ای ندارد که درک آن مشکل باشد و یا نیاز به توضیح زیادی داشته باشد. این کلاس از دو فیلد برای نگهداری رنگ دکمه و نیز مستطیلی که برای آن دکمه کشیده می شود تشکیل شده

است. همچنین در این کلاس یک متد سازنده وجود دارد که مقدار فیلد نگهدارنده ی رنگ را به صورت اتوماتیـک بـر اسـاس ورودی متد تنظیم می کند.

```
public class ColorPaletteButton
{
      // Publiuc members
      public Color color = System.Drawing.Color.Black;
      public Rectangle rectangle;

      // Constructor
      public ColorPaletteButton(Color objColor)
      {
            color = objColor;
      }
```



شکل ۱۴-۳

این کلاس دارای متدی به نام `Draw` است و زمانی فراخوانی می شود که بخواهیم این دکمه را در پالت رنگ قرار دهیم. برای این کار یک مربع توپر با استفاده از رنگی که این دکمه معرف آن است رسم می شود. سپس یک مربع توخالی به رنگ سیاه در اطراف مربع اول رسم می شود که همانند یک حاشیه برای آن محسوب می شود.

```
      // Draw the button
      public void Draw(Graphics graphics)
      {
            // Draw the color block
            SolidBrush objSolidBrush = new
                              SolidBrush(color);
```

```
            graphics.FillRectangle(objSolidBrush,
                                   rectangle);

            // Draw an edge around the control
            Pen objPen = new Pen(Color.Black);
            graphics.DrawRectangle(objPen, rectangle);
        }
```

همچنین این کلاس دارای متدی به نام SetPosition است که به وسیله ی پارامترهایی که دریافت می کند، مکان رسم دکمه را در پالت رنگ تعیین می کند. به این ترتیب هنگامی که اندازه ی فرم تغییر کرد می توانیم موقعیت جدید دکمه را (یعنی مکان گوشه ی بالا سمت چپ آن را) به همراه اندازه ی دکمه به این متد فرستاده و سپس با فراخوانی متد Draw دکمه را در این مکان جدید رسم کنیم.

```
        // Move the button to the given position
        public void SetPosition(int x, int y,
                                int buttonSize)
        {
            // Update the members
            rectangle = new
                Rectangle(x, y, buttonSize, buttonSize);
        }
```

احتمالاً جالب ترین قسمت این برنامه متد ColorPalette_Resize است. الگوریتمی که در این قسمت برای تعیین موقعیت دکمه ها استفاده می کنید، الگوریتمی است که معمولاً برای تعیین مکان کنترل ها و یا شیئ های گرافیکی دیگر به کار می رود. برای رسم کنترل اندازه ی هر کدام از آنها (که در این قسمت شامل اندازه ی خود دکمه و اندازه ی فضای بین دکمه ها است) و نیز محدوده ی هر یک را نیز می دانیم. تنها کاری که باید انجام دهیم این است که از گوشه ی بالا سمت چپ شروع کنیم و یکی یکی دکمه ها را در صفحه رسم کنیم تا دیگر فضایی در آن خط باقی نمانده باشد، سپس به خط بعدی برویم. برای شروع ابتدا باید حلقه ای ایجاد کنیم تا بین تمام دکمه های موجود حرکت کند.

```
private void ColorPalette_Resize(object sender,
                                 EventArgs e)
{
    // Declare variables to hold the position
    int intX = 0;
    int intY = 0;

    // Go through the array and position the buttons
    foreach (ColorPaletteButton objColorPaletteButton
                                        in Buttons)
    {
```

متغییر intX مکان افقی و متغییر intY مکان عمودی را نگهداری می کند و مقدار اولیه ی آنها نیز (0,0) است. به این معنی که اولین دکمه در موقعیت (0,0) در کنترل قرار خواهد گرفت.

```
// Position the button
objColorPaletteButton.SetPosition(intX, intY, ButtonSize);
```

بعد از قرار دادن این دکمه، موقعیت افقی را به اندازه ی طول یک دکمه و نیز فاصله ی بین دو دکمه به سمت راست منتقل می کنیم. سپس بررسی می کنیم که آیا اگر دکمه ی دیگری را در این خط قرار دهیم، طول آن از طول کنترل پالت رنگ بیشتر خواهد شد یا نه؟ در صورتی که فضایی در آن قسمت وجود نداشت، مقدار عمودی را (`intY`) به اندازه ی طول یک دکمه و نیز فاصله ی بین دو دکمه به پایین منتقل کرده و مقدار افقی را برابر با صفر قرار می دهیم تا مجدداً از ابتدای خط برای رسم دکمه ها استفاده کند.

```
            // Move to the next one
            intX += (ButtonSize + ButtonSpacing);

            // Do we need to go down to the next row
            if (intX + ButtonSize > Width)
            {
                // Move y
                intY += (ButtonSize + ButtonSpacing);

                // Reset x
                intX = 0;
            }
        }
```

در آخر نیز متد `Invalidate` در کنترل را فراخوانی می کنیم تا مشخص کنیم که ظاهر این کنترل نا معتبر است و باید توسط ویندوز مجدداً رسم شود.

```
        // Redraw
        this.Invalidate();
}
```

پاسخ دادن به کلیک ها:

کنترلی که در قسمت قبل برای استفاده به عنوان یک پالت رنگ ایجاد کردیم، باید بتواند هنگامی که کاربر روی یکی از رنگها کلیک کرد، رویداد مناسبی را فراخوانی کند. به همین منظور در بخش امتحان کنید بعد چند رویداد را به این کلاس اضافه خواهیم کـرد. بـه این ترتیب برنامه ای که از این کنترل استفاده می کند می تواند متد هایی را ایجاد کرده و تعیین کند که هنگام رخ دادن هـر یـک از این رویدادها این متد ها فراخوانی شوند.

امتحان کنید: پاسخ دادن به کلیک ها

۱) به قسمت ویرایشگر کد برای کلاس ColorPalette بروید و با اضافه کردن کد زیر به این کلاس دو متد مورد نظر را ایجاد کنید:

```
    // Public Delegates
    public delegate void _leftClick(Object sender,
                                    EventArgs e);
    public delegate void _rightClick(Object sender,
                                    EventArgs e);

    // Public Events
    public event _leftClick LeftClick;
    public event _rightClick RightClick;
```

۲) در این قسمت از برنامه به متدی نیاز داریم که موقعیت ماوس را دریافت کرده و اعلام کند که اشاره گر ماوس روی کدامیک از کلید های موجود در پالت رنگ قرار دارد. برای این کار متد زیر را به کلاس اضافه کنید:

```
public ColorPaletteButton GetButtonAt(int X, int Y)
{
    // Go through each button in the collection
    foreach (ColorPaletteButton objColorPaletteButton
                                            in Buttons)
        // Is this button in the rectangle?
        if (objColorPaletteButton.rectangle.Contains(
                                                X, Y))
            return objColorPaletteButton;

    // If no button found, return null value
    return null;
}
```

۳) حال در پنجره ی Properties به لیست رویدادهای کنترل ColorPalette رفته، از این لیست رویداد MouseUp را انتخاب کرده و روی آن دو بار کلیک کنید تا متد مربوط به این رویداد ایجاد شود. سپس کد زیر را به این متد اضافه کنید:

```
private void ColorPalette_MouseUp(object sender,
                                  MouseEventArgs e)
{
    // Find the button that we clicked
    ColorPaletteButton objColorPaletteButton =
                            GetButtonAt(e.X, e.Y);

    if (objColorPaletteButton != null)
    {
```

```
            // Was the left button was clicked?
            if (e.Button == MouseButtons.Left)
            {
                // Set the color
                LeftColor = objColorPaletteButton.color;

                // Raise the event
                LeftClick(this, new EventArgs());
            }
            else if (e.Button == MouseButtons.Right)
            {
                // Set the color
                RightColor = objColorPaletteButton.color;

                // Raise the event
                RightClick(this, new EventArgs());
            }
        }
}
```

۴) برای تست کنترل جدید به قسمت طراحی فرم مربوط به `Form1` رفته و کنترل `PaintCanvas` را انتخاب کنید. سپس خاصیت `Name` این کنترل را به `Canvas` تغییر دهید.

۵) برنامه را یک بار کامپایل کنید تا تغییراتی که در کنترل `ColorPalette` ایجاد کرده ایم اعمال شوند.

۶) سپس کنترل `paletteColor` را از فرم `Form1` انتخاب کرده و با استفاده از لیست رویدادهای این کنترل در پنجره ی `Properties`، رویداد `LeftClick` را انتخاب کرده و روی آن دوبار کلیک کنید. به این ترتیب متد مربوط به این رویداد ایجاد خواهد شد. سپس کد زیر را در این متد وارد کنید:

```
private void paletteControl_LeftClick(object sender,
                                      EventArgs e)
{
    Canvas.GraphicColor = paletteControl.LeftColor;
}
```

۷) برنامه را اجرا کنید. این مرتبه می توانید با کلیک روی هر کدام از رنگ هایی که در پایین فرم نمایش داده می شوند، رنگ مورد استفاده برای ترسیم در فرم را تغییر دهید.

چگونه کار می کند؟

کنترل هایی که در این قسمت به عنوان دکمه در پالت رنگ استفاده کرده ایم مقداری با کنترلهای `Button` که در قسمتهای قبل از آنها استفاده می کردیم تفاوت دارند. کنترلهای `Button` که در قسمت قبل از آنها استفاده می کردیم به اندازه ی کافی هوشمند بودند تا زمانی که روی آنها کلیک شود را تشخیص داده و با فراخوانی رویدادی در آن زمان کار خاصی را انجام دهند. اما

دکمه هایی که در این قسمت در پالت رنگ ایجاد کردیم، در حقیقت فقط قسمتی از محدوده ی پالت رنگ هستند. بنابراین زمان کلیک شدن روی آنها را باید خودمان تشخیص دهیم.
برای این کار از متد GetButtonAt استفاده می کنیم. این متد موقعیت کنونی اشاره گر ماوس را می گیرد و سپس بررسی می کند که کدامیک از دکمه های موجود در پالت رنگ شامل این موقعیت می شوند. به عبارت دیگر مشخص می کند که اشاره گر ماوس روی کدامیک از دکمه های موجود در پالت رنگ قرار دارد.
در اینجا برای تشخیص اینکه اشاره گر ماوس در محدوده ی مربع شکل کدامیک از دکمه ها قرار دارد می توانیم از متد Contains در کلاس Rectangle استفاده کنیم.

```
public ColorPaletteButton GetButtonAt(int X, int Y)
{
     // Go through each button in the collection
     foreach (ColorPaletteButton objColorPaletteButton
                                              in Buttons)
          // Is this button in the rectangle?
          if (objColorPaletteButton.rectangle.Contains(
                                                X, Y))
               return objColorPaletteButton;

     // If no button found, return null value
     return null;
}
```

البته مشخص است که کاربر می تواند در ناحیه ای از این کنترل کلیک کند که هیچ دکمه ای در آن قسمت وجود نداشته باشد. بنابراین اگر تا انتهای حلقه هیچ مقداری از تابع برگشت داده نشد، مقدار null را برمی گردانیم تا مشخص شود که در ناحیه ای که کلیک شده دکمه ای وجود نداشته است.
بعد از اتمام این تابع، متدی را ایجاد می کنیم تا هنگامی که رویداد MouseUp این کنترل رخ داد، این متد فراخوانی شود. در این متد ابتدا با استفاده از متد GetButtonAt مشخص می کنیم که کاربر روی کدامیک از دکمه های موجود در پالت رنگ کلیک کرده است. سپس اگر کاربر روی دکمه ای کلیک کرده بود (مقدار برگشتی از تابع مخالف null بود)، بر حسب اینکه کاربر برای کلیک کردن از کلید چپ و یا راست ماوس استفاده کرده است مقدار LeftColor و یا RightColor را برابر با رنگ انتخابی قرار داده و رویداد متناظر آن را نیز فراخوانی می کنیم.

```
private void ColorPalette_MouseUp(object sender,
                                  MouseEventArgs e)
{
     // Find the button that we clicked
     ColorPaletteButton objColorPaletteButton =
                              GetButtonAt(e.X, e.Y);

     if (objColorPaletteButton != null)
     {
          // Was the left button was clicked?
          if (e.Button == MouseButtons.Left)
          {
```

```
                // Set the color
                LeftColor = objColorPaletteButton.color;

                // Raise the event
                LeftClick(this, new EventArgs());
            }
            else if (e.Button == MouseButtons.Right)
            {
                // Set the color
                RightColor = objColorPaletteButton.color;

                // Raise the event
                RightClick(this, new EventArgs());
            }
        }
}
```

اما هنوز کنترل `PaintCanvas` در فرم اصلی برنامه فقط می تواند با یک رنگ کار کند. بنابراین متدی را برای رویداد `LeftClick` کنترل `paletteColor` در نظر می گیریم تا با فراخوانی این متد رنگ مورد استفاده در کنترل `PaintCanvas` بر اساس رنگ انتخابی در فرم تغییر کند. برای تنظیم رنگ کنترل نیز باید از خاصیت `GraphicColor` استفاده کنیم.

```
private void paletteControl_LeftClick(object sender,
                                      EventArgs e)
{
    Canvas.GraphicColor = paletteControl.LeftColor;
}
```

استفاده از دو رنگ در برنامه:

در این قسمت می خواهیم برنامه را به گونه ای تغییر دهیم که بتوانیم با دو رنگ نیز در آن کار کنیم. برای این کار باید دو فیلد `public` به کلاس `PaintCanvas` اضافه کنیم تا رنگهای مربوط به کلید چپ و راست ماوس در آنها نگهداری شوند. همچنین باید کد های دیگر را نیز به گونه ای تغییر دهیم تا تشخیص دهند که کلید چپ ماوس فشار داده شده است و یا کلید راست ماوس؟

امتحان کنید: استفاده از دو رنگ در برنامه

۱) ابتدا باید در کلاس `PaintCanvas` دو متغییر ایجاد کنیم تا بتوانیم رنگهای مربوط به کلید چپ و راست ماوس را در آنها نگهداری کنیم. بنابراین ویرایشگر کد را برای کلاس `PaintCanvas` باز کرده و تغییرات زیر را در کد آن کلاس وارد کنید:

```
// Public members
public ArrayList GraphicsItems = new ArrayList();
public GraphicTools GraphicTool = GraphicTools.CirclePen;
public GraphicSizes GraphicSize = GraphicSizes.Medium;

public Color GraphicLeftColor = Color.Black;
public Color GraphicRightColor = Color.White;
```

۲) در متد DoMousePaint باید از خاصیت Button در کلاس MouseEventArgs استفاده کنیم تا تشخیص دهیم که کدام رنگ باید برای ایجاد شکل به کار گرفته شود. کد نوشته شده در این متد را به صورت زیر تغییر دهید:

```
private void DoMousePaint(MouseEventArgs e)
{
     // Store the new item somewhere
     GraphicsItem objGraphicsItem = null;

     // What color do we want to use?
     Color objColor = GraphicLeftColor;

     if (e.Button == MouseButtons.Right)
          objColor = GraphicRightColor;

     // What tool are you using?
     switch(GraphicTool)
     {
          // CirclePen
          case GraphicTools.CirclePen:
          {
               // Create a new graphics circle
               GraphicsCircle objGraphicsCircle = new
                         GraphicsCircle();

               // Set the point for drawing
               objGraphicsCircle.SetPoint(e.X, e.Y,
                         (int)GraphicSize, objColor,
                    true);

               // Store this for addition
               objGraphicsItem = objGraphicsCircle;
               break;
          }
     }

     // Were you given an item?
```

```
        if( objGraphicsItem != null)
        {
            // Add it to the list
            GraphicsItems.Add(objGraphicsItem);

            // Invalidate the control
            this.Invalidate(objGraphicsItem.rectangle) ;
        }
}
```

۳) در ایـن کـلاس رویـدادهای MouseDown و MouseMove فقـط زمـانی متـد DoMousePaint را فراخوانی می کنند که کلید چپ ماوس فشار داده شده باشد. بنابراین باید کد موجود در این متد ها را نیز به گونه ای تغییر دهیم که در صورت فشار داده شدن کلید راست ماوس نیز، این متد را فراخوانی کنند. پس تغییرات زیـر را در ایـن دو متـد وارد کنید:

```
private void PaintCanvas_MouseDown(object sender,
                                   MouseEventArgs e)
{
    // Is the left mouse button down?
    if( e.Button == MouseButtons.Left ||
        e.Button == MouseButtons.Right)
        DoMousePaint(e);
}

private void PaintCanvas_MouseMove(object sender,
                                   MouseEventArgs e)
{
    // Is the left mouse button down?
    if (e.Button == MouseButtons.Left ||
        e.Button == MouseButtons.Right)
        DoMousePaint(e);
}
```

۴) سپس باید متد paletteColor_LeftClick در کلاس Form1 را نیز به گونه ای تغییر دهیم تا به جای تنظیم خاصیت GraphicColor، خاصیت GraphicLeftColor را تنظیم کنـد. بنـابراین بـه قسمت ویرایشگر کد مربوط به کلاس Form1 بروید و تغییرات زیر را در این متد ایجاد کنید:

```
private void paletteControl_LeftClick(object sender,
                                      EventArgs e)
{
    Canvas.GraphicLeftColor = paletteControl.LeftColor;
}
```

۵) در آخر نیز باید متد مربوط به رویداد RightClick را ایجاد کنیم. برای این کار روی رویداد RightClick در قسمت Events پنجره ی Properties دو بار کلیک کنید تا متد مربوط به کنترل این رویداد ایجاد شود. سپس کد زیر را در این متد وارد کنید:

```
private void paletteControl_RightClick(object sender,
                                       EventArgs e)
{
     Canvas.GraphicRightColor =
                            paletteControl.RightColor;
}
```

حال اگر برنامه را اجرا کنید مشاهده خواهید کرد که می توانید با استفاده از کلید راست و چپ ماوس دو رنگ را انتخاب کرده و از آنها برای ترسیم فرم استفاده کنید.

مشخص کردن رنگهای مورد استفاده:

مطمئناً تاکنون متوجه شده اید که استفاده از برنامه ی MyPaint مقداری گیج کننده است، زیرا با انتخاب یک رنگ از پالت رنگ هیچ روشی برای فهمیدن این که چه رنگی برای کلید راست و چه رنگی برای کلید چپ در نظر گرفته شده است وجود ندارد. برای رفع این مشکل در بخش امتحان کنید بعد برنامه را به گونه ای تغییر می دهیم تا بر روی دکمه ای که رنگ آن برای کلید راست مورد استفاده قرار گرفته است حرف R و بر روی دکمه ای که رنگ آن برای کلید چپ مورد استفاده قرار گرفته است، حرف L نمایش داده شود.

امتحان کنید: مشخص کردن رنگهای مورد استفاده

۱) ابتدا باید برنامه را به گونه ای تغییر دهیم که اشیای ایجاد شده از کلاس ColorPaletteButton بدانند به چه کلیدی تعلق دارند. برای این کار قسمت ویرایشگر کد برای کلاس ColorPaletteButton را باز کرده و کد زیر را به ابتدای کلاس اضافه کنید تا بتوانیم از این شمارنده در کلاس استفاده کنیم:

```
     public class ColorPaletteButton
     {
          // Public enumerations
          public enum ButtonAssignments
          {
               None = 0,
               LeftButton = 1,
               RightButton = 2
          }
```

۲) سپس فیلد زیر را نیز به این کلاس اضافه کنید تا بتوانیم توسط آن مشخص کنیم که رنگ هر دکمه به چه کلیدی از ماوس نسبت داده شده است.

```
// Publiuc members
public Color color = System.Drawing.Color.Black;
public Rectangle rectangle;

public ButtonAssignments ButtonAssignment =
                         ButtonAssignments.None;
```

۳) حال باید متد Draw را به گونه ای تغییر دهیم که حرف L و یا حرف R را روی دکمه های مربوط به آنها در پالت رنگ نمایش دهد. برای این کار کد زیر را به متد Draw اضافه کنید:

```
// Draw the button
public void Draw(Graphics graphics)
{
     // Draw the color block
     SolidBrush objSolidBrush = new SolidBrush(color);
     graphics.FillRectangle(objSolidBrush, rectangle);

     // Draw an edge around the control
     Pen objPen = new Pen(Color.Black);
     graphics.DrawRectangle(objPen, rectangle);

     // Are you selected?
     if (ButtonAssignment != ButtonAssignments.None)
     {
          // Create a Font
          Font objFont = new Font("verdana", 80,
                                  FontStyle.Bold);

          // Set the default button assignment
          String strButtonText = "L";

          // Update the button assignment if necessary
          if (ButtonAssignment ==
                         ButtonAssignments.RightButton)
          {
               strButtonText = "R";
          }

          // What brush do you want?
          if (color.R < 100 || color.B < 100 ||
                                  color.G < 100)
          {
               objSolidBrush = new
```

```
                              SolidBrush(Color.White);
            }
            else
            {
                  objSolidBrush = new
                              SolidBrush(Color.Black);
            }

            // Draw the text "L" or "R"
            graphics.DrawString(strButtonText, objFont,
                              objSolidBrush, rectangle.Left,
                              rectangle.Right);
      }
}
```

۴) به قسمت ویرایشگر کد مربوط به کلاس PaletteColor بروید و کد زیر را به این کلاس اضافه کنید به وسیله ی این دو فیلد می توانیم در هر لحظه مشخص کنیم که رنگ چه دکمه ای از پالت رنگ باید به عنوان رنگ کلید چپ ماوس و رنگ چه دکمه ای باید به عنوان رنگ کلید راست ماوس مورد استفاده قرار گیرد.

```
      // Private members
      private ColorPaletteButton LeftButton;
      private ColorPaletteButton RightButton;
```

۵) کدی که در این قسمت وارد می کنیم ممکن است مقداری طولانی باشد، اما کار ساده ای را انجام می دهد. در این قسمت باید تغییراتی ایجاد کنیم تا مطمئن شویم یک رنگ نمی تواند به هر دو کلید ماوس نسبت داده شود. همچنین باید حرف L (و یا حرف R) را از روی دکمه ی قبلی در پالت رنگ پاک کنیم و آن را روی دکمه ی جدید قرار دهیم. بنابراین این دو دکمه در پالت رنگ را نیز باید از نوع نا معتبر مشخص کنیم تا مجدداً توسط ویندوز رسم شوند. تغییرات زیر را در متد ColorPalette_MouseUp ایجاد کنید:

```
private void ColorPalette_MouseUp(object sender,
                                    MouseEventArgs e)
{
      // Find the button that we clicked
      ColorPaletteButton objColorPaletteButton =
                                    GetButtonAt(e.X, e.Y);

      if (objColorPaletteButton != null)
      {
            // Was the left button was clicked?
            if (e.Button == MouseButtons.Left)
            {
                  // Make sure that this button is not
                  // the current right button
                  if (objColorPaletteButton != RightButton)
                  {
```

```
                    // Set the color
                    LeftColor =
                         objColorPaletteButton.color;

                    // Clear the existing selection
                    if (LeftButton != null)
                    {
                         LeftButton.ButtonAssignment =
               ColorPaletteButton.ButtonAssignments.None;

                         this.Invalidate(
                              LeftButton.rectangle);
                    }

                    // Mark the button

          objColorPaletteButton.ButtonAssignment  =
     ColorPaletteButton.ButtonAssignments.LeftButton;

                    this.Invalidate(
               objColorPaletteButton.rectangle);

                    LeftButton = objColorPaletteButton;

                    // Raise the event
                    LeftClick(this, new EventArgs());
               }
          }
          else if (e.Button == MouseButtons.Right)
          {
               // Make sure that this button is not
               // the current left button
               if (objColorPaletteButton != LeftButton)
               {
                    // Set the color
                    RightColor =
                         objColorPaletteButton.color;

                    // Clear the existing selection
                    if (RightButton != null)
                    {
                         RightButton.ButtonAssignment =
               ColorPaletteButton.ButtonAssignments.None;

                         this.Invalidate(
                              RightButton.rectangle);
                    }
```

```
                    // Mark the button
                    objColorPaletteButton.ButtonAssignment
     = ColorPaletteButton.ButtonAssignments.RightButton;

                    this.Invalidate(
                         objColorPaletteButton.rectangle);

                    RightButton = objColorPaletteButton;

                    // Raise the event
                    RightClick(this, new EventArgs());
               }
          }
     }
}
```

۶) در آخر نیز باید کدی را وارد کنیم تا هنگامی که برنامه شروع به کار می کند، یکی از دکمه های موجود در پالت رنگ به کلید چپ ماوس و رنگ دیگر به کلید راست ماوس نسبت داده شود. برای این کار متد AddColor را به نحوی تغییر می دهیم تا اگر هیچ رنگی به دو کلید ماوس نسبت داده نشده بود، رنگ اول را برای کلید چپ و رنگ دوم را برای کلید راست در نظر بگیرد. بنابراین تغییرات زیر را در این متد ایجاد کنید:

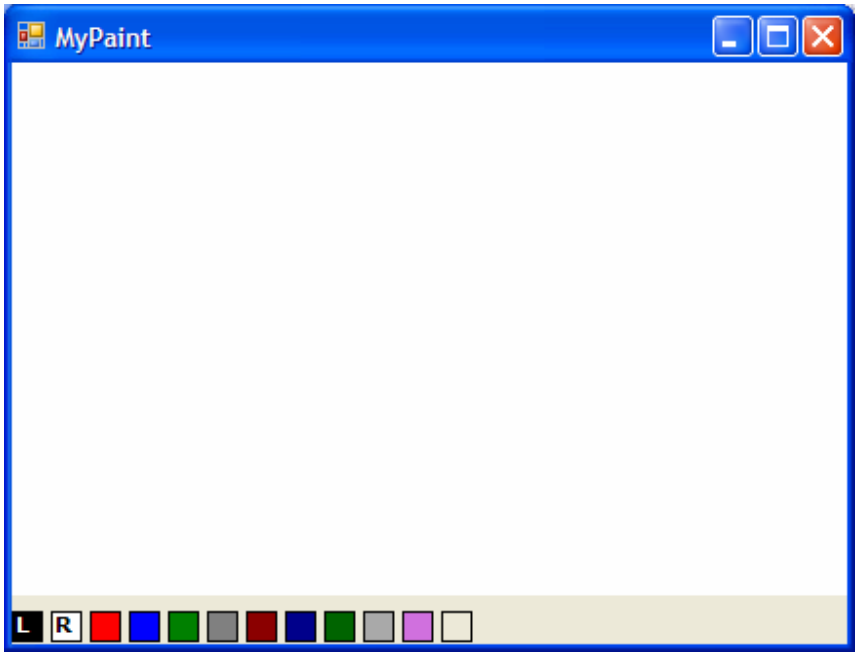
```
// Add a new color button to the control
public void AddColor(Color newColor)
{
     // Create the button
     ColorPaletteButton objColorPaletteButton = new
                              ColorPaletteButton(newColor);

     // Add it to the list
     Buttons.Add(objColorPaletteButton);

     // do we have a button assigned
     // to the left button yet?
     if (LeftButton != null)
     {
          objColorPaletteButton.ButtonAssignment =
          ColorPaletteButton.ButtonAssignments.LeftButton;
          LeftButton = objColorPaletteButton;
     }
     else if(RightButton != null)
     // How about the right button
     {
          objColorPaletteButton.ButtonAssignment =
          ColorPaletteButton.ButtonAssignments.RightButton;
          RightButton = objColorPaletteButton;
```

```
        }
}
```

۷) برنامه را اجرا کنید. مشاهده خواهید کرد که رنگ انتخاب شده برای کلیدهای چپ و راست ماوس در این قسمت به وسیله ی حروف L و یا R نمایش داده می شوند و با تغییر رنگ انتخابی در پالت رنگ مکان این حروف نیز تغییر می کند (شکل ۱۴-۴).



شکل ۱۴-۴

چگونه کار می کند؟

اولین کاری که در این قسمت از برنامه انجام می دهیم این است که یک شمارنده در کلاس `ColorPaletteButton` ایجاد می کنیم تا به وسیله ی آن بتوانیم وضعیت هر دکمه را تعیین کنیم:

```
// Public enumerations
public enum ButtonAssignments
{
      None = 0,
      LeftButton = 1,
      RightButton = 2
}
```

همانطور که در اعضای این شمارنده نیز مشخص می کنید، رنگ هر دکمه از پالت رنگ می تواند سه حالت داشته باشد: یا به هیچ یک از کلیدهای ماوس نسبت داده نشده باشد، یا برای کلید چپ ماوس در نظر گرفته شده باشد و یا برای کلید راست ماوس در نظر

گرفته شده باشد. علاوه بر این لازم است که دو فیلد نیز به کلاس ColorPalette اضافه کنیم تا بتوانیم به وسیله ی آنها در این کلاس تشخیص دهیم که کدامیک از دکمه های موجود در پالت رنگ به کلید چپ و راست ماوس نسبت داده شده اند. وظیفه ی این دو فیلد این است که در هر لحظه مشخص کنند که چه دکمه ای برای کلید چپ ماوس و چه دکمه ای برای کلید راست ماوس در نظر گرفته شده است.

```
// Private members
private ColorPaletteButton LeftButton;
private ColorPaletteButton RightButton;
```

استفاده از این دو فیلد باعث می شود که راحت تر بتوانیم دکمه ی انتخاب شده در پالت رنگ را تغییر دهیم. به این ترتیب دیگر لازم نیست هنگامی که کاربر رنگ مورد نظر خود را تغییر داد در بین تمام دکمه های موجود در پالت رنگ حرکت کرده و بررسی کنیم که حرف L و یا R از روی کدامیک از این دکمه ها باید حذف شود. برای مثال هنگامی که کاربر با استفاده از کلید چپ ماوس روی یکی از دکمه های موجود در پالت کلیک می کند، کافی است که خاصیت ButtonAssignment دکمه ای که به آن اشاره می کند را برابر با ButtonAssignment.None قرار داده و رنگ انتخابی کاربر را در فیلد LeftButton قرار دهیم. سپس خاصیت ButtonAssignment در دکمه ای که جدیداً انتخاب شده است را نیز برابر با ButtonAssignment.LeftButton قرار دهیم.
تنها متدی که در این قسمت ممکن است مقداری گیج کننده به نظر برسد، متد ColorPalette_MouseUp است. در این متد قبل از اینکه دکمه ای از پالت رنگ را برای کلید چپ ماوس در نظر بگیریم، باید بررسی کنیم که آن دکمه برای کلید راست مشخص نشده باشد.

```
private void ColorPalette_MouseUp(object sender,
                                  MouseEventArgs e)
{
     // Find the button that we clicked
     ColorPaletteButton objColorPaletteButton =
                                  GetButtonAt(e.X, e.Y);

     if (objColorPaletteButton != null)
     {
          // Was the left button was clicked?
          if (e.Button == MouseButtons.Left)
          {
               // Make sure that this button is not
               // the current right button
               if (objColorPaletteButton != RightButton)
               {
```

در این صورت می توانیم خاصیت LeftColor را برابر با رنگ دکمه ای که توسط کاربر انتخاب شده است قرار دهیم:

```
// Set the color
LeftColor = objColorPaletteButton.color;
```

سپس باید بررسی کنیم که آیا قبل از این نیز رنگی برای دکمه ی چپ ماوس در نظر گرفته شده بود، که در این صورت باید خاصیت ButtonAssignment آن را به None برگردانیم و متد Invalidate را برای آن فراخوانی کنیم تا حرف L از روی این دکمه از پالت رنگ حذف شود:

```
// Clear the existing selection
if (RightButton != null)
{
     RightButton.ButtonAssignment =
     ColorPaletteButton.ButtonAssignments.None;

     this.Invalidate(RightButton.rectangle);
}
```

علاوه بر این باید خاصیت ButtonAssignment دکمه ای که جدیداً انتخاب شده است را نیز برابر با LeftButton قرار دهیم و متد Invalidate را برای آن فراخوانی کنیم. به این ترتیب هنگام ترسیم مجدد فرم، حرف L بر روی این دکمه از پالت رنگ نمایش داده می شود. همچنین باید خاصیت LeftButton را نیز برابر با دکمه ای قرار دهید که انتخاب شده است تا در مواقع نیاز بتوانیم از آن استفاده کنیم:

```
// Mark the button
objColorPaletteButton.ButtonAssignment  =
ColorPaletteButton.ButtonAssignments.LeftButton;

this.Invalidate(objColorPaletteButton.rectangle);

LeftButton = objColorPaletteButton;
```

در آخر نیز همانند قسمتهای قبل رویداد LeftClick را فراخوانی می کنیم:

```
// Raise the event
LeftClick(this, new EventArgs());
```

بقیه ی متد ColorPalette_MouseUp نیز همانند این قسمت است، با این تفاوت که دستورهای آن برای کلید رایت ماوس انجام می شود. به این ترتیب که ابتدا بررسی می کنیم دکمه ای که برای کلید راست ماوس انتخاب شده است، به کلید چپ تعلق نداشته باشد، در این صورت رنگ این دکمه را به خاصیت RightColor نسبت می دهیم و ....
بعد از اتمام این متد، به متد Draw در کلاس ColorPaletteButton می رویم. در این متد بعد از رسم یک دکمه در پالت رنگ، بررسی می کنیم که آیا این دکمه به کلیدی از ماوس نسبت داده شده است و در این صورت حرفی متناسب با کلید ماوس را روی این دکمه می نویسیم (در مورد ترسیم متن در این قسمت به اختصار توضیح داده شده است، اما در قسمتهای بعدی این فصل در این مورد بیشتر صحبت خواهیم کرد). برای رسم یک متن ابتدا باید با ایجاد شیئ ای از کلاس System.Drawing.Font فونت متن را مشخص کنیم. در اینجا فونتی از نوع verdana با اندازه ی 8 و Bold ایجاد می کنیم:

```
// Draw the button
```

```
public void Draw(Graphics graphics)
{
     // Draw the color block
     SolidBrush objSolidBrush = new SolidBrush(color);
     graphics.FillRectangle(objSolidBrush, rectangle);

     // Draw an edge around the control
     Pen objPen = new Pen(Color.Black);
     graphics.DrawRectangle(objPen, rectangle);

     // Are you selected?
     if (ButtonAssignment != ButtonAssignments.None)
     {
          // Create a Font
          Font objFont = new Font("verdana", 80,
                                  FontStyle.Bold);
```

سپس مشخص می کنیم که چه حرفی باید روی این دکمه از پالت رنگ نمایش داده شود:

```
     // Set the default button assignment
     String strButtonText = "L";

     // Update the button assignment if necessary
     if (ButtonAssignment ==
                    ButtonAssignments.RightButton)
     {
          strButtonText = "R";
     }
```

حالا باید مشخص کنیم که می خواهیم متن با چه رنگی نوشته شود. در این قسمت نمی توانیم از یک رنگ ثابت برای نوشتن متن استفاده کنیم. زیرا برای مثال اگر همواره متن را با استفاده از رنگ سفید بنویسیم، این متن روی رنگهای روشن به درستی دیده نمی شود. بنابراین باید دقت کنیم که می خواهیم متن را روی چه رنگی نمایش دهیم. اگر رنگ دکمه ای که می خواهیم روی آن بنویسیم روشن بود باید از رنگ سیاه استفاده کنیم، در غیر این صورت بهتر است رنگ سفید را به کار ببریم:

```
     // What brush do you want?
     if (color.R < 100 || color.B < 100 || color.G < 100)
     {
          objSolidBrush = new SolidBrush(Color.White);
     }
     else
     {
          objSolidBrush = new SolidBrush(Color.Black);
     }
```

در آخر نیز متن را با استفاده از تنظیمات مشخص شده در صفحه رسم می کنیم:

```
            // Draw the text "L" or "R"
            graphics.DrawString(strButtonText, objFont,
                        objSolidBrush, rectangle.Left,
                        rectangle.Right);
      }
}
```

استفاده از رنگهای بیشتر در برنامه:

تاکنون تمام رنگهایی که در این برنامه استفاده کرده ایم، رنگهایی بودند که در چارچوب .NET تعریف شده بودند برای مثال Color.Black و یا Color.Blue. البته تعداد رنگهایی که به این صورت در .NET وجود دارند بسیار زیاد است و شاید در ابتدا کافی به نظر برسد، اما در شرایطی ممکن است بخواهیم که رنگی را خودمان به برنامه اضافه کنیم که در این لیست وجود ندارد.

نکته: برای مشاهده لیست تمام رنگهایی که در .NET تعریف شده است می توانید از قسمت "All Members" مربوط به "Color Structure" در سیستم راهنمای MSDN مراجعه کنید. همچنین هنگامی که در حال کد نویسی در ویرایشگر کد ویژوال استودیو هستید، با وارد کردن نام شمارنده ی Color سیستم هوشمند ویژوال استودیو لیست رنگهای موجود در این شمارنده را نمایش می دهد.

ویندوز برای نمایش یک رنگ از یک عدد ۲۴ بیتی استفاده می کند. این عدد همانطور که می دانید شامل ۳ بایت می شود و هر بایت آن برای نمایش مقدار یکی از سه رنگ اصلی (قرمز، سبز،آبی)در رنگ مورد نظر به کار می رود – این روش معمولاً به نام "مشخص کردن رنگ به صورت RGB[1]" شناخته شده است. به این ترتیب می توانید با مشخص کردن سه عدد در بازه ی صفر تا ۲۵۵ مقدار هر یک از سه رنگ اصلی را در رنگ نهایی تعیین کنید و با ترکیب این مقدار ها به یکی از ۱۶,۷ میلیون رنگی که می توان با استفاده از این روش نمایش داد دسترسی پیدا کنید. برای مثال با قرار دادن عدد ۲۵۵ برای Red و ۰ برای Green و Blue می توانید رنگ قرمز را ایجاد کنید. قرار دادن هر سه مقدار با عدد صفر رنگ سیاه و تنظیم آنها با عدد ۲۵۵ رنگ سفید را نتیجه می دهد.
برای درک بهتر این موضوع در قسمت بعد مشاهده خواهیم کرد که چگونه می توانیم رنگ دلخواه خود را به صورت دستی ایجاد کرده و آن را در پالت رنگ قرار دهیم.

امتحان کنید: اضافه کردن رنگهای دلخواه

۱) به قسمت طراحی کنترل ColorPalette بروید و بعد از کلیک کردن در قسمت خالی از این کنترل، خاصیت BackColor را در پنجره ی Properties انتخاب کنید.

[1] Red-Green-Blue

۲) در لیست مربوط به این خاصیت به قسمت Custom بروید و روی یکی از ۱۶ مربع سفید موجود در پایین این قسمت کلید راست ماوس را فشار دهید.

۳) به این ترتیب کادر Color نمایش داده خواهد شد. در این کادر با استفاده از ماوس رنگ مورد نظر خود را انتخاب کنید. در قسمت سمت راست پایین این فرم سه کادر متنی وجود دارند که با عبارات Red، Green و Blue همانند شکل ۱۴-۵ مشخص شده اند. اعدادی که در این سه کادر نمایش داده می شوند را یاداشت کنید.

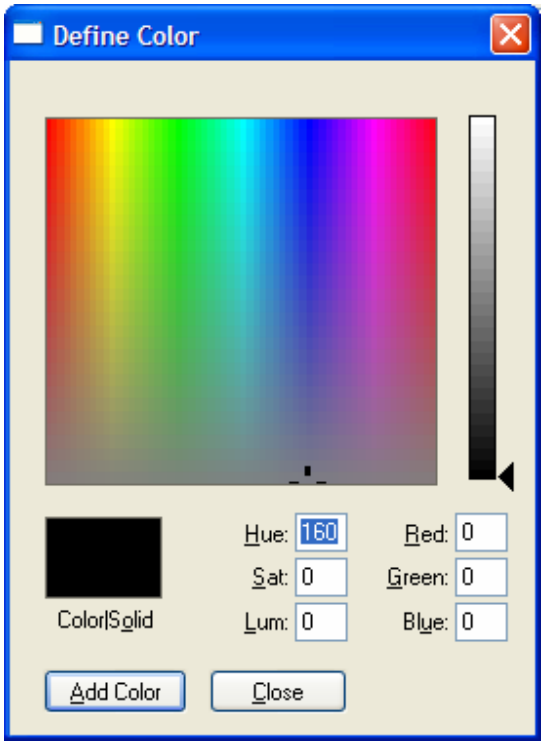
۴) کادر Define Color را ببندید.

۵) با استفاده از ویرایشگر کد مربوط به کلاس ColorPalette به قسمت متد سازنده ی این کلاس بروید. در متد سازنده با استفاده از متد AddColor دکمه ی دیگری را همانند کدی که در قسمت زیر مشخص شده است به پالت رنگ اضافه کنید. البته سه عددی که من در این قسمت استفاده کرده ام را با سه عدد مربوط به رنگ انتخابی خود تغییر دهید (دقت کنید که اعداد را به ترتیب وارد کنید: عدد اول مربوط به رنگ قرمز است، عدد دوم مربوط به رنگ سبز و عدد سوم نیز مربوط به رنگ آبی است).

```
public ColorPalette()
{
      InitializeComponent();

      // Add the colors
      AddColor(Color.Black);
      AddColor(Color.White);
      AddColor(Color.Red);
      AddColor(Color.Blue);
      AddColor(Color.Green);
      AddColor(Color.Gray);
      AddColor(Color.DarkRed);
      AddColor(Color.DarkBlue);
      AddColor(Color.DarkGreen);
      AddColor(Color.DarkGray);

      AddColor(Color.FromArgb(208, 112, 222));
}
```



شکل ۱۴-۵

۶) حال برنامه را اجرا کنید. مشاهده خواهید کرد که رنگ انتخابی شما نیز به پالت رنگ اضافه شده است.

متد `FromArgb` یکی از متدهای استاتیک موجود در کلاس `Color` است که می توانید از آن برای تعریف رنگهای مورد نظر خود استفاده کنید.

استفاده از کادر `Color`:

هر مقدار که رنگهای بیشتری را در ابتدای برنامه به صورت اتوماتیک با پالت رنگ اضافه کنید، باز هم ممکن است کاربر بخواهد از رنگی استفاده کند که در پالت وجود نداشته باشد. بهترین روش برای اینکه به کاربر اجازه دهیم از هر رنگی که تمایل دارد استفاده کند، این است که در برنامه کادر `Color` (که در فصل هفتم با آن آشنا شدیم) را به کار ببریم.
در بخش امتحان کنید بعد، برنامه را به گونه ای تغییر خواهیم داد که کاربر بتواند رنگ مورد نظر خود را به وسیله ی کادر محاوره ای `Color` انتخاب کند.

انتخاب کنید: استفاده از کادر `Color` در برنامه

۱) به قسمت طراحی فرم مربوط به کنترل ColorPalette بروید و با استفاده از جعبه ابزار، یک کنترل ColorDialog را به این کنترل اضافه کنید. سپس خاصیت Name کنترل را برابر با dlgColor قرار دهید.

۲) حال به قسمت ویرایشگر کد کلاس ColorPalette بروید و متد ColorPalette_MouseUp در این قسمت را پیدا کنید. می خواهیم این متد را به گونه ای تغییر دهیم که اگر کاربر در ناحیه ای از این کنترل کلیک کرد که هیچ دکمه ای از پالت وجود نداشت (به عبارت دیگر روی هیچ دکمه ای از پالت کلیک نکرده بود)، کادر Color را نمایش دهیم تا بتواند رنگ دلخواه خود را به وسیله ی این کادر به پالت رنگ اضافه کند. به قسمت انتهای این متد بروید و یک عبارت else به همراه کدی که در زیر مشخص شده است را به متد اضافه کنید:

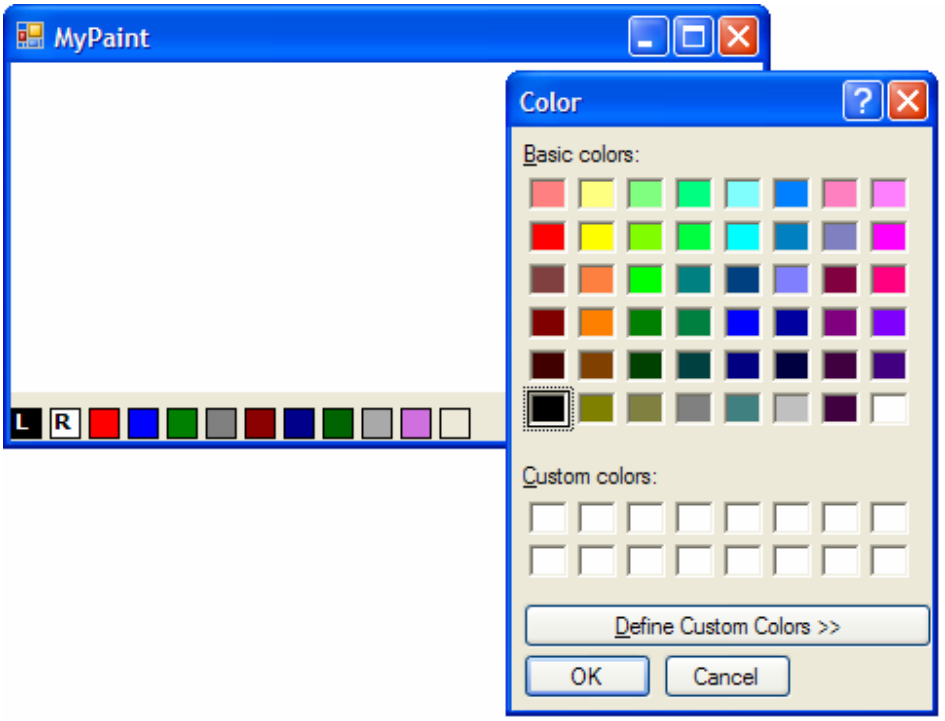
```
private void ColorPalette_MouseUp(object sender,
                                        MouseEventArgs e)
{
     // Find the button that we clicked
     ColorPaletteButton objColorPaletteButton =
                                    GetButtonAt(e.X, e.Y);

     if (objColorPaletteButton != null)
     {
                    …

                    // Raise the event
                    RightClick(this, new EventArgs());
               }
          }
     }
     else
     {
          // Display the Color dialog
          if (dlgColor.ShowDialog() == DialogResult.OK)
          {
               // Add the new color
               AddColor(dlgColor.Color);

               // Resize the palette to show the color
               OnResize(new EventArgs());
          }
     }
}
```

۳) برنامه را اجرا کرده و روی قسمتی خالی از پالت رنگ کلیک کنید. مشاهده می کنید که همانند شکل ۱۴-۶ کادر Color نمایش داده می شود و می توانید رنگ مورد نظر خود را به پالت اضافه کنید.



شکل ۱۴-۶

استفاده از رنگهای سیستمی:

تاکنون نحوه ی استفاده از رنگهای تعریف شده در .NET و همچنین استفاده از رنگهای مورد نظر خودتان را در برنامه مشاهده کردید. مورد آخری که باید در مورد رنگها بدانید، استفاده از رنگهای سیستمی در برنامه است.

در سیستم عامل ویندوز این امکان برای کاربر وجود دارد که رنگ هر قسمتی که مشاهده می کند را به گونه ای دلخواه تغییر دهد، برای مثال رنگ دکمه ها، رنگ منو ها، رنگ نوار عنوان و یا ... را به رنگ دلخواه خود در آورد. بنابراین هنگام نوشتن یک برنامه اگر می خواهید یک رابط گرافیکی برای کنترل خود طراحی کنید، بهتر است این نکته را در نظر داشته باشید که از یک سیستم به سیستم دیگر ممکن است رنگ کنترل ها کاملاً تغییر کند (برای مثال ممکن است کاربری ویندوز را به گونه ای تنظیم کند تا رنگ دکمه ها به رنگ آبی نمایش داده شوند)، بنابراین باید ظاهر کنترل خود را به گونه ای طراحی کنید که بر اساس ظاهر سیستمی که در آن مورد استفاده قرار می گیرد تغییر کند.

برای این کار می توانید از رنگهای سیستمی که در .NET تعریف شده اند استفاده کنید. برای دسترسی به رنگهای سیستمی باید کلاس `System.Drawing.SystemColors` را به کار ببرید. برای مشاهده ی لیستی از تمام رنگهای سیستمی موجود در این کلاس می توانید از قسمت مربوط به این کلاس در سیستم راهنمای MSDN استفاده کنید. همچنین با استفاده از ابزار `Object Browser` و یا ویژگی هوشمندی ویژوال استودیو هنگام کد نویسی نیز می توانید لیست رنگهای موجود در این قسمت را مشاهده کنید.

در بخش امتحان کنید بعد، رنگی را به پالت رنگ اضافه می کنیم که هم رنگ نوار منوی کامپیوتر باشد.

امتحان کنید: استفاده از رنگهای سیستمی

۱) ویرایشگر کد مربوط به کلاس `ColorPalette` را باز کرده و به متد سازنده ی این کلاس بروید. سپس کد مشخص شده در زیر را به این متد اضافه کنید:

```
public ColorPalette()
{
      InitializeComponent();

      // Add the colors
      AddColor(Color.Black);
      AddColor(Color.White);
      AddColor(Color.Red);
      AddColor(Color.Blue);
      AddColor(Color.Green);
      AddColor(Color.Gray);
      AddColor(Color.DarkRed);
      AddColor(Color.DarkBlue);
      AddColor(Color.DarkGreen);
      AddColor(Color.DarkGray);
      AddColor(Color.FromArgb(208, 112, 222));

      AddColor(SystemColors.MenuBar);
}
```

۲) با اجرای برنامه مشاهده خواهید کرد که رنگ جدیدی، همرنگ با نوار منوی ویندوز به پالت رنگ اضافه شده است.

استفاده از ابزارهای متفاوت:

حالا که توانستیم با موفقیت استفاده از ابزار دایره ی توپر را به همراه تمام امکانات لازم به برنامه اضافه کنیم، بهتر است توجه خود را روی اضافه کردن ابزارهای دیگر به برنامه قرار دهیم. در بخش امتحان کنید بعد نوار منویی به برنامه اضافه خواهیم کرد که به وسیله ی آن می توانید ابزار مورد استفاده در برنامه را انتخاب کنید.

نکته: برای تکمیل این بخش از برنامه لازم است که یک نوار منو به برنامه اضافه کنیم. اگر احساس می کنید در استفاده از کنترلهای مربوط به نوار منو مشکلی دارید، به فصل هشتم مراجعه کنید.

امتحان کنید: اضافه کردن منو به برنامه

۱) به قسمت طراحی فرم مربوط به `Form1` بروید و خاصیت `Anchor` کنترل `PaintCanvas` را به `Left, Right, Bottom` تغییر دهید.

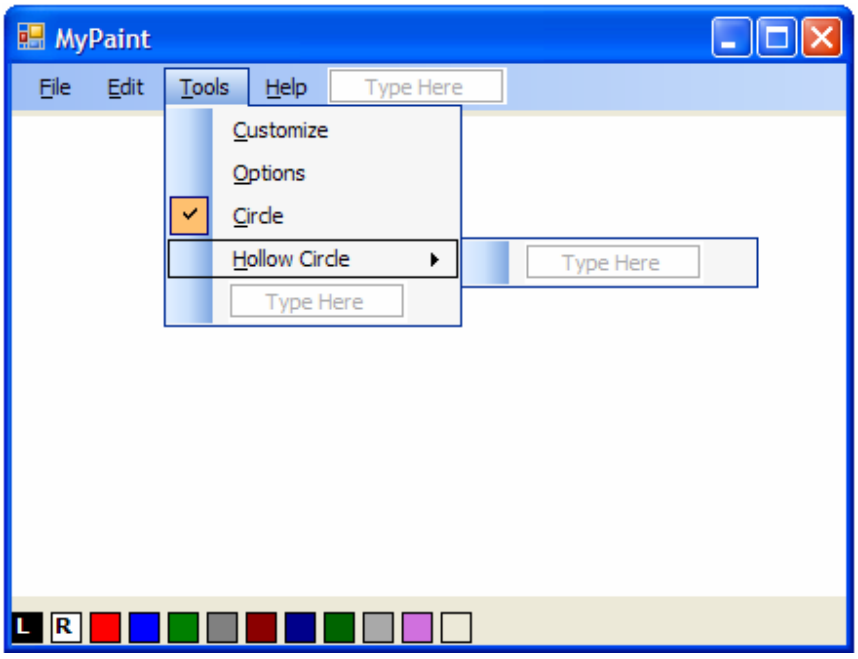
۲) حال بر روی نوار عنوان فرم کلیک کرده تا فرم انتخاب شود، سپس اندازه ی آن را به گونه ای تغییر دهید که بتوانید یک کنترل `MenuStrip` را در بالای آن قرار دهید.

۳) با استفاده از جعبه ابزار به قسمت `Menus & Toolbars` بروید و روی کنترل `MenuStrip` دو بار کلیک کنید. به این ترتیب یک نمونه از این کنترل به نام `MenuStrip1` در پایین قسمت طراحی فرم قرار خواهد گرفت. روی این کنترل کلیک راست کرده و در منویی که باز می شود گزینه ی `Insert Standard Items` را انتخاب کنید.

۴) در صورت لزوم اندازه ی فرم را به گونه ای تغییر دهید تا کنترل `PaintCanvas` دقیقاً پایین نوار منو قرار بگیرد. سپس کنترل `PaintCanvas` را انتخاب کرده و خاصیت `Anchor` آن را مجدداً به `Top, Left,` `Bottom, Right` برگردانید.

۵) حال به منوی `Tools` در کنترل `MenuStrip1` بروید و در این منو، در کادر سفید پایین منو که عبارت `Type Here` در آن نوشته شده است کلیک کرده و عبارت `&Circle` را در این قسمت وارد کنید تا گزینه ی جدیدی به منوی `Tools` اضافه شود. سپس با استفاده از پنجره ی `Properties` خاصیت `Checked` آن را به `True` و خاصیت `CheckOnClick` را نیز به `True` تغییر دهید.

۶) در کادر `Type Here` بعدی در پایین گزینه ی `Circle` کلیک کرده و عبارت `&Hollow Circle` را در آن وارد کنید. سپس با استفاده از پنجره ی `Properties` خاصیت `CheckOnClick` این گزینه را برابر با `True` قرار دهید. به این ترتیب فرم برنامه باید مشابه شکل ۱۴-۷ شده باشد.



شکل ۱۴-۷

استفاده از دایره های توخالی:

در قسمتهای قبلی برای ترسیم فقط می توانستیم از دایره های توپر استفاده کنیم. در بخش امتحان کنید بعد برنامه را به گونه ای تغییر خواهیم داد که بتوانیم ابزار مورد استفاده برای ترسیم یک شکل را تعیین کنیم. برای مثال از قلمی به شکل دایره های توپر استفاده کنیم و یا قلمی به شکل دایره های توخالی را به کار ببریم.

امتحان کنید: استفاده از دایره های توخالی

۱) اولین کاری که در این قسمت باید انجام دهیم این است که شمارنده ی `GraphicTools` که در کلاس `PaintCanvas` تعریف شده است را به گونه ای تغییر دهیم که شامل ابزاری برای دایره ی توخالی نیز بشود. بنابراین به قسمت ویرایشگر کد مربوط به کلاس `PaintCanvas` بروید و تغییر زیر را در شمارنده ی `PaintCanvas` ایجاد کنید:

```
public partial class PaintCanvas : UserControl
{
     public enum GraphicTools
     {
          CirclePen = 0,
          HollowCirclePen = 1
     }
```

۲) به قسمت طراحی فرم مربوط به `Form1` بروید و در نوار منوی بالای فرم، روی کنترل `circleToolStripMenuItem` دو بار کلیک کنید تا متد مربوط به رویداد `Click` این کنترل ایجاد شود. سپس کد مشخص شده در زیر را به این متد اضافه کنید:

```
private void circleToolStripMenuItem_Click(object sender,
                                           EventArgs e)
{
     // Set the tool
     Canvas.GraphicTool =
               PaintCanvas.GraphicTools.CirclePen;

     // Uncheck the Hollow Circle menu item
     hollowCircleToolStripMenuItem.Checked = false;
}
```

۳) حال مجدداً به قسمت طراحی فرم مربوط به `Form1` برگشته و روی کنترل `hollowCircleToolStripMenuItem` در منوی `Tools` در نوار منوی بالای فرم دو بار کلیک کنید تا متد مربوط به رویداد `Click` این کنترل ایجاد شود. سپس کد مشخص شده در زیر را در این متد وارد کنید:

```
private void hollowCircleToolStripMenuItem_Click(
                                object sender, EventArgs e)
```

```
{
     // Set the tool
     Canvas.GraphicTool =
               PaintCanvas.GraphicTools.HollowCirclePen;

     // Uncheck the Circle menu item
     circleToolStripMenuItem.Checked = false;
}
```

۴) نیازی نیست که گزینه های دیگر نوار منو را نیز تکمیل کنیم، اما بهتر است که کد مربوط به گزینه ی Exit در منـوی File را نیز وارد کنیم تا در برنامه بتوانیم از این منو استفاده کنیم. پس در قسمت طراحی فرم روی این گزینـه دو بـار کلیک کرده تا متد مربوط به رویداد Click آن ایجاد شود. سپس کد مشخص شده در زیر را به این منو اضافه کنید:

```
private void exitToolStripMenuItem_Click(object sender,
                                                EventArgs e)
{
     // Close the application
     this.Close();
}
```

۵) ویرایـشگر کـد کـلاس PaintCanvas را مجـدداً بـاز کـرده و دسـتورات موجـود در بـلاک switch(GraphicTool) در متد DoMousePaint ا به صورت مشخص شده در زیر تغییر دهید.

```
// What tool are you using?
switch(GraphicTool)
{
     // CirclePen
     case GraphicTools.CirclePen:
     }
          // Create a new graphics circle
          GraphicsCircle objGraphicsCircle = new
                                        GraphicsCircle();

          // Set the point for drawing
          objGraphicsCircle.SetPoint(e.X, e.Y,
                         (int)GraphicSize, objColor,
                                             true);

          // Store this for addition
          objGraphicsItem = objGraphicsCircle;
          break;
     }
     // HollowCirclePen
     case GraphicTools.HollowCirclePen:
     {
          // Create a new graphics circle
```

```
            GraphicsCircle objGraphicsCircle = new
                                            GraphicsCircle();

            // Set the point for drawing
            objGraphicsCircle.SetPoint(e.X, e.Y,
                        (int)GraphicSize, objColor,
                                                false);

            // Store this for addition
            objGraphicsItem = objGraphicsCircle;
            break;
        }
}
```

۶) بعد از این باید متد `Draw` در کلاس `GraphicsCircle` را به گونه ای تغییر دهیم که تشخیص دهد چه مواقعی باید دایره ی توخالی رسم کند و چه مواقعی باید دایره ی توپر رسم کند. بنابراین ویرایشگر کد کلاس `GraphicsCircle` را باز کرده و تغییرات زیر را در متد `Draw` ایجاد کنید:

```
public override void Draw(Graphics graphics)
{
     if (IsFilled)
     {
          // Create a new pen
          SolidBrush objSolidBrush = new
                                SolidBrush(this.color);

          // Draw the circle
          graphics.FillEllipse(objSolidBrush,
                                     this.rectangle);
     }
     else
     {
          // Create a pen
          Pen pen = new Pen(this.color);

          // Use DrawEllipse instead
          Rectangle objRectangle = this.rectangle;
          objRectangle.Inflate(-1, -1);
          graphics.DrawEllipse(pen, objRectangle);
     }
}
```

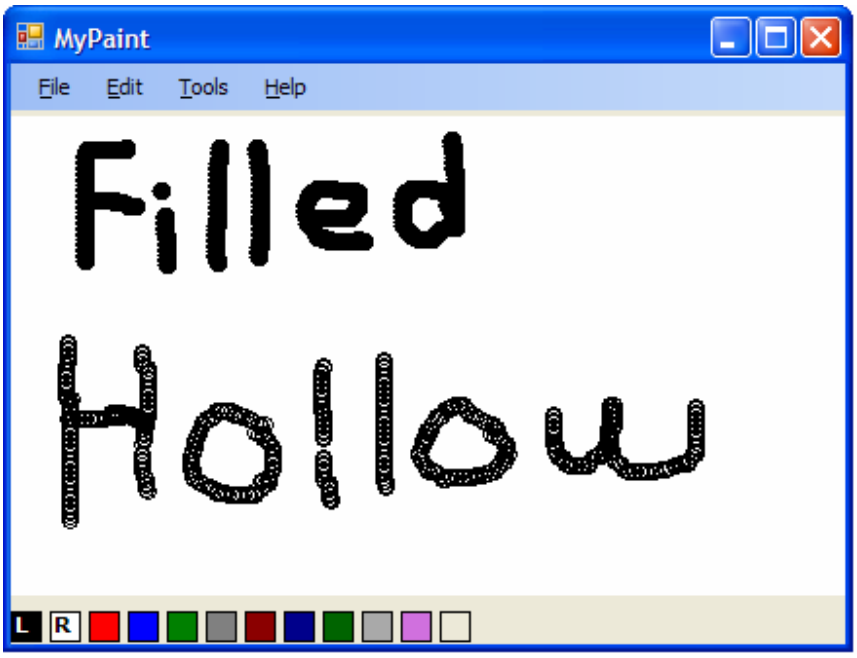
۷) حالا برنامه را اجرا کنید. به این ترتیب می توانید با استفاده از منوی `Tools` ابزار گرافیکی که می خواهید از آن استفاده کنید را انتخاب کرده و رسم شکل را با آن ابزار انجام دهید (شکل ۱۴-۸).

چگونه کار می کند؟

هنگامی که کاربر روی کی از ابزارهای موجود در منوی `Tools` کلیک می کند، رویداد `Click` آن ابزار فراخوانی می شود. هنگام فراخوانی این رویداد ابتدا باید خاصیت `GraphicTool` در کنترل `PaintCanvas` را برابر با ابزار انتخاب شده قرار دهیم، سپس باید علامت تیک کنار این گزینه ها در نوار منو را تصحیح کنیم. هنگامی که یک گزینه انتخاب شود، علامت تیک به صورت اتوماتیک در کنار آن قرار می گیرد اما علامت تیک که در کنار گزینه ی قبلی قرار داشته است برداشته نمی شود. بنابراین در متد مربوط به رویداد `Click` هر گزینه، باید این علامت را به صورت دستی از کنار گزینه ی دیگر حذف کنیم:

```
private void hollowCircleToolStripMenuItem_Click(
                                object sender, EventArgs e)
{
      // Set the tool
      Canvas.GraphicTool =
                PaintCanvas.GraphicTools.HollowCirclePen;

      // Uncheck the Circle menu item
      circleToolStripMenuItem.Checked = false;
}
```



شکل ۸-۱۴

صرفنظر از نوع ابزاری که برای ترسیم انتخاب شده است، هنگامی که اشاره گر ماوس بخواهد با حرکت روی صفحه شکلی را ترسیم کند متد `DoMousePaint` از کلاس `PaintCanvas` فراخوانی می شود. بنابراین قسمت `switch` این متد را باید به گونه ای تغییر دهیم که بسته به ابزاری که انتخاب شده است، شکل مناسبی را ایجاد کرده و به آرایه ی اجزای تشکیل دهنده

ی فرم اضافه کند. اگر بخواهیم با هر بار کلیک کردن روی فرم، یک دایره ی توخالی رسم شود باید پارامتر true و اگر بخواهیم یک دایره ی توپر رسم شود باید پارامتر false را به متد SetPoint بفرستیم.

```
switch(GraphicTool)
{
      // CirclePen
      case GraphicTools.CirclePen:
      }
            // Create a new graphics circle
            GraphicsCircle objGraphicsCircle = new
                                          GraphicsCircle();

            // Set the point for drawing
            objGraphicsCircle.SetPoint(e.X, e.Y,
                        (int)GraphicSize, objColor,
                                             true);

            // Store this for addition
            objGraphicsItem = objGraphicsCircle;
            break;
      }
      // HollowCirclePen
      case GraphicTools.HollowCirclePen:
      {
            // Create a new graphics circle
            GraphicsCircle objGraphicsCircle = new
                                          GraphicsCircle();

            // Set the point for drawing
            objGraphicsCircle.SetPoint(e.X, e.Y,
                        (int)GraphicSize, objColor,
                                             false);

            // Store this for addition
            objGraphicsItem = objGraphicsCircle;
            break;
      }
}
```

به این ترتیب پارامتر graphicIsFilled که به متد SetPoint فرستاده می شود مشخص می کند که باید دایره ی توخالی رسم کنیم و یا دایره ی توپر. بنابراین در متد Draw در کلاس GraphicsCircle شرطی را قرار می دهیم تا اگر خاصیت IsFilled برابر با true بود، این متد با استفاده از متد FillEllipse یک دایره ی توپر رسم کند. در غیر این صورت متد Draw با فراخوانی متد DrawEllipse یک دایره ی توپر رسم می کند.

در این متد برای اینکه دایره های توخالی بهتر نمایش داده شوند، از یک حقه ی کوچک استفاده کرده ایم، به این صورت که هنگام رسم این دایره طول و عرض آن را یک واحد کاهش می دهیم. به این ترتیب دایره ای که در صفحه رسم می شود واضح تر خواهد

بود. دلیل این مورد به علت طبیعت خاصی است که سیستم گرافیکی ویندوز بر اساس آن کار می کند. هر چه بیشتر در این زمینه فعالیت کنید، در این موارد نیز تجربه ی بیشتری بدست خواهید آورد.

```
public override void Draw(Graphics graphics)
{
      if (IsFilled)
      {
            // Create a new pen
            SolidBrush objSolidBrush = new
                                    SolidBrush(this.color);

            // Draw the circle
            graphics.FillEllipse(objSolidBrush,
                                          this.rectangle);
      }
      else
      {
            // Create a pen
            Pen pen = new Pen(this.color);

            // Use DrawEllipse instead
            Rectangle objRectangle = this.rectangle;
            objRectangle.Inflate(-1, -1);
            graphics.DrawEllipse(pen, objRectangle);
      }
}
```

خوب، به این ترتیب تمام قسمتهای مورد نیاز برای یک برنامه ی گرافیکی ساده را به برنامه ی خودمان اضافه کردیم و تقریباً تمام نکات ابتدایی که لازم بود برای برنامه نویسی گرافیکی دو بعدی در ویندوز بدانیم را نیز با هم بررسی کردیم. در ادامه ی فصل سعی می کنیم که مقداری هم به مبحث کار با عکسها در ویژوال C# ۲۰۰۵ بپردازیم.

کار با عکسها:

در چارچوب .NET برای سادگی کار با عکس، تمام امکانات لازم برای ذخیره کردن و همچنین نمایش دادن قالبهای عمومی، به صورت درونی ایجاد شده است. برای مثال می توانید به راحتی عکسهایی را با قالبهای زیر ذخیره کرده و یا نمایش دهید:

- .bmp: قالب استاندار ویندوز برای تصاویری که به صورت بیت مپی ذخیره می شوند.
- .gif: قالب استاندارد تصاویر کوچک اینترنتی که معمولاً حجم کم و کیفیت پایینی دارند.
- .jpeg و یا .jpg: قالب مربوط به عکسهایی که می خواهیم کیفیت مطلوب و نیز حجم کمی داشته باشند تا بتوانیم در اینترنت از آنها استفاده کنیم.
- .png: کاربردی مشابه فایلهایی با قالب .gif دارد.
- .tiff: فرمت فایل استاندارد برای ذخیره و یا تغییر در فایلهای اسکن شده است.

- .wmf و یا .emf: فرمت استاندارد برای فایلهایی از نوع Windows Metafile
- .ico: قالب استاندارد برای ذخیره ی آیکون برنامه ها
- .exif: قالب فایل استاندارد که معمولاً به صورت درونی بین دوربین های دیجیتالی مورد استفاده قرار می گیرد.

قبل از .NET برنامه نویسانی که می خواستند با تصاویر مورد استفاده در اینترنت کار کنند و یا از آنها در برنامه های خود استفاده کنند (معمولاً تصاویری با پسوند های .gif و یا .jpg) مجبور بودند کنترلهای سفارشی نوشته شده به وسیله ی شرکت های دیگر را در برنامه ی خود به کار ببرند. اما اکنون کار با این نوع تصاویر به صورت درونی در .NET پشتیبانی می شود و می توانید به سادگی از آنها در برنامه های خود استفاده کنید.
اما جالب اینجاست که .NET نه تنها استفاده از این فایلها را پشتیبانی می کند، بلکه اجازه ی ذخیره ی یک تصویر در هر یک از این قالب ها نیز می دهد. بنابراین برای مثال می توانید یک عکس با پسوند .gif را باز کرده و سپس آن را با پسوند دیگری مانند .png و یا .bmp ذخیره کنید. برای کار با تصاویر در .NET دو روش کلی وجود دارد. اول این است که از کنترل PictureBox در جعبه ابزار استفاده کنید. برای کار با این کنترل می توانید یک نمونه از آن را در فرم برنامه قرار داده و سپس چه در زمان اجرا و چه در زمان طراحی آدرس یک عکس را برای آن مشخص کنید. به این ترتیب این کنترل آن عکس را در فرم برنامه نمایش می دهد. البته این روش فقط برای نمایش عکسهای ثابت در برنامه مناسب است. روش دیگر این است که عکسها را در داخل کنترل هایی که خودتان ایجاد می کنید نمایش دهید. در ادامه ی فصل برنامه ی MyPaint را به گونه ای تغییر خواهیم داد که بتوانیم به جای ترسیم روی یک فرم سفید و خالی، روی یکی از عکسهایی که از دیسک کامپیوتر مشخص می کنیم، تصویری را رسم کنیم.

نمایش تصاویر:

برای نمایش یک تصویر باید از شیئ ای از کلاس System.Drawing.Image استفاده کنیم. در بخش امتحان کنید بعد، برنامه را به گونه ای تغییر خواهیم داد تا بتوانیم یک عکس خاص را در پس زمینه ی فرمی که در آن نقاشی می کنیم نمایش دهیم.

امتحان کنید: تنظیم عکس پس زمینه

۱) به قسمت طراحی فرم مربوط به Form1 بروید و با استفاده از جعبه ابزار یک کنترل OpenFileDialog را روی فرم قرار دهید. سپس خاصیت Name این کنترل را برابر با dlgFileOpenBackground قرار دهید.
۲) حال باید کاری کنیم تا زمانی که کاربر روی گزینه ی Open در منوی File کلیک کرد، پنجره ی Open File نمایش داده شود. بنابراین در منوی File موجود در فرم برنامه، روی گزینه ی Open دو بار کلیک کنید تا متد مربوط به رویداد Click این کنترل ایجاد شود. سپس کد زیر را در این متد وارد کنید:

```
private void openToolStripMenuItem_Click(object sender,
                                         EventArgs e)
{
    // Set the open file dialog properties
```

```
        dlgFileOpenBackground.Filter = "Image files" +
            " (*.gif,*.jpg,*.jpeg,*.bmp,*.wmf,*.png)" +
            "|*.gif;*.jpg;*.jpeg;*.bmp;*.wmf;*.png|All" +
            " files (*.*)|*.*";
        dlgFileOpenBackground.FilterIndex = 1;
        dlgFileOpenBackground.Title = "Open Picture Files";

        // Show the dialog
        if (dlgFileOpenBackground.ShowDialog() ==
                                        DialogResult.OK)
        {
            // Create a new image that references the file
            Image backgroundImage = Image.FromFile(
                            dlgFileOpenBackground.FileName);

            // Set the background of the canvas
            Canvas.BackgroundImage = backgroundImage;
        }
}
```

۳) برنامه را اجرا کرده و روی گزینه ی Open در منوی File کلیک کنید تا کادر محاوره ای Open نمایش داده شود. در این کادر آدرس یک فایل تصویر با یکی از فرمت های مشخص شده را معین کنید. مشاهده خواهید کرد که تصویر انتخابی شما همانند شکل ۱۴-۹ در برنامه نمایش داده می شود.



شکل ۱۴-۹

چگونه کار می کند؟

ممکن است با خود بگویید "من که برای نمایش تصویر کاری انجام ندادم؟"، خوب حق با شماست. برای اینکه بتوانید تصویری را در برنامه نمایش دهید نیازی ندارید که از کد زیادی استفاده کنید. کنترلهای سفارشی که ایجاد می کنیم همه از کلاس `UserControl` مشتق می شوند و این کلاس نیز خود به صورت غیر مستقیم از کلاس `Control` مشتق می شود. در کلاس `Control` خاصیتی به نام `BackgroundImage` تعریف شده است که می تواند شیئ ای از نوع `Image` را در خود نگهداری کرده و آن را به عنوان تصویر پس زمینه نمایش دهد. بنابراین کافی است که تصویر مورد نظر خود را انتخاب کرده و شیئ `Image` مربوط به آن را در این خاصیت قرار دهید. بقیه ی کارهای مربوط به ترسیم تصویر به عهده ی کلاس پایه خواهد بود.

بعد از اینکه آدرس فایل تصویر را به وسیله ی کادر `Open` بدست آوردیم می توانیم با استفاده از متد استاتیک `FromFile` از کلاس `Image` آن را در یک شیئ از کلاس `Image` قرار دهیم. استفاده از این کلاس ساده ترین روش برای نمایش یک فایل تصویر موجود در کامپیوتر است.

```
// Show the dialog
if (dlgFileOpenBackground.ShowDialog() ==
                                DialogResult.OK)
{
     // Create a new image that references the file
     Image backgroundImage = Image.FromFile(
                    dlgFileOpenBackground.FileName);

     // Set the background of the canvas
     Canvas.BackgroundImage = backgroundImage;
}
```

ممکن است حس کنید با قرار دادن یک عکس در پس زمینه ی برنامه، سرعت برنامه کاهش پیدا می کند. علت این امر در این است که ترسیم یک عکس در فرم کار سنگینی است و انجام آن زمان زیادی را صرف می کند. برای رفع این مشکل می توانید از عکسهای کوچکتر برای نمایش در فرم استفاده کنید.

تغییر اندازه ی تصاویر:

در قسمت قبل اگر عکسی که برای فرم در نظر می گرفتید کوچکتر از اندازه ی فرم بود، و یا اندازه ی فرم را به گونه ای تغییر می دادید که بزرگتر از اندازه ی عکس شود متوجه می شدید که چندین کپی از عکس انتخابی شما در کنار هم چیده می شود تا فرم برنامه پر شود. همچنین اگر اندازه ی فرم از اندازه ی عکس انتخابی شما کوچکتر باشد، مقداری از عکس در برنامه نمایش داده نمی شود. بهتر است برنامه را به گونه ای تغییر دهیم که با تغییر اندازه ی عکس آن را در فرم برنامه نمایش دهد. در قسمت امتحان کنید بعدی، برای اینکه بتوانیم با تغییر اندازه ی عکس آن را به اندازه ی فرم در آورده و نمایش دهیم وظیفه ی نمایش تصویر را از کلاس `Control` خواهیم گرفت و در خودمان آن را انجام خواهیم داد.

امتحان کنید: نمایش تصویر در کلاس `PaintCanvas`

۱) قسمت ویرایشگر کد مربوط به کلاس PaintCanvas را باز کنید.

۲) در این قسمت به جای استفاده از متد مربوط به رویداد Paint از متد دیگری به نام OnPaintBackground استفاده می کنیم که قبل از فراخوانی رویداد Paint فراخوانی می شود. کد مشخص شده در زیر را به کلاس PaintCanvas اضافه کنید:

```
protected override void OnPaintBackground(
                                        PaintEventArgs e)
{
     // Paint the invalid region
     // with the background brush
     SolidBrush backgroundBrush = new
                                     SolidBrush(BackColor);
     e.Graphics.FillRectangle(backgroundBrush,
                                     e.ClipRectangle);

     // Paint the image
     if (BackgroundImage != null)
     {
          // Find our client rectangle
          Rectangle clientRectangle = new Rectangle(0, 0,
                                        Width, Height);

          // Draw the image
          e.Graphics.DrawImage(BackgroundImage,
                                     clientRectangle);
     }
}
```

۳) حال به قسمت طراحی فرم کنترل PaintCanvas برگشته و روی قسمتی از این کنترل کلیک کنید تا انتخاب شود. سپس با استفاده از قسمت Events در پنجره ی Properties رویداد Resize را انتخاب کرده و روی آن دو بار کلیک کنید تا متد مربوط به این رویداد ایجاد شود. کد مشخص شده در زیر را در این رویداد وارد کنید:

```
private void PaintCanvas_Resize(object sender,
                                     EventArgs e)
{
     // Invalidate the control
     this.Invalidate();
}
```

۴) حال برنامه را اجرا کرده و عکسی را که به اندازه ی فرم نباشد را برای نمایش به عنوان پس زمینه انتخاب کنید. مشاهده خواهید کرد که اندازه ی تصویر تغییر می کند تا به اندازه ی فرم در آید (شکل ۱۴-۱۰).



شکل ۱۴-۱۰

چگونه کار می کند؟

همانطور که در قسمتهای قبلی نیز گفتم هر ترسیم از دو مرحله تشکیل شده است. در مرحله ی اول سطح کنترل پاک می شود، در این قسمت رویداد `PaintBackground` فراخوانی می شود. در مرحله ی دوم نیز به کنترل فرصت داده می شود تا خود را ترسیم کند، در این مرحله نیز رویداد `Paint` فراخوانی می شود. به عبارت دیگر برای هر ترسیم ویندوز با فراخوانی رویداد `PaintBackground` پس زمینه ی صفحه را رسم می کند، سپس با فراخوانی رویداد `Paint` در کنترل، به کنترل اجازه می دهد تا خود را روی پس زمینه ی ایجاد شده رسم کند.

در متد `OnPaintBackground` ابتدا باید فرم تا پاک کنیم در غیر این صورت ممکن است تصاویر درست نمایش داده نشوند. برای این کار یک قلم موی جدید از کلاس `SolidBrush` به رنگ پس زمینه ی کنترل ایجاد کرده و با استفاده از آن یک مستطیل توپر به اندازه و رنگ قسمتی از فرم که از نوع نا معتبر مشخص شده است (`ClipRectangle`) رسم می کنیم.

```
protected override void OnPaintBackground(
                                        PaintEventArgs e)
{
      // Paint the invalid region
      // with the background brush
      SolidBrush backgroundBrush = new
                                     SolidBrush(BackColor);
      e.Graphics.FillRectangle(backgroundBrush,
                                     e.ClipRectangle);
```

بعد از این کار باید تصویر مورد نظر را در پس زمینه ی فرم رسم کنیم. برای رسم یک تصویر می توانیم به سادگی از متد DrawImage در کلاس Graphics استفاده کنیم. اما در اینجا قبل از اینکه تصویر را رسم کنیم باید محدوده ی رسم آن را نیز مشخص کنیم. برای این کار هنگام فراخوانی متد DrawImage برای رسم تصویر، محدوده ی آن (شیئ ای از کلاس Rectangle) و نیز خود تصویر را به این متد می فرستیم و سپس تصویر را رسم می کنیم:

```
        // Paint the image
        if (BackgroundImage != null)
        {
            // Find our client rectangle
            Rectangle clientRectangle = new Rectangle(0, 0,
                                            Width, Height);

            // Draw the image
            e.Graphics.DrawImage(BackgroundImage,
                                        clientRectangle);
        }
}
```

متد های دیگر کلاس Graphics:

در این فصل سعی کردیم که با بعضی از ویژگیهای گرافیکی که در .NET وجود دارند آشنا شویم و مهمترین آنها را بررسی کنیم. در طول فصل همچنین با یکی از مهمترین کلاس هایی که برای کارهای گرافیکی در .NET وجود دارد به نام کلاس Graphics نیز آشنا شدیم و از بعضی از متدهای پر کاربرد آن استفاده کردیم. در زیر تعدادی از متدهای کلاس Graphics که استفاده ی زیادی در برنامه ها دارند را به اختصار معرفی می کنیم.

- **DrawLine** بین دو نقطه ی مشخص خطی را رسم می کند.
- **DrawCurve** و **DrawClosedCurve** یک مجموعه از نقاط را دریافت کرده و بین این نقاط یک منحنی رسم می کند.
- **DrawArc** یک کمان (قسمتی از یک دایره) را در صفحه رسم می کند.
- **DrawPie** قسمتی از یک دایره را رسم می کند (همانند یک نمودار دایره ای).
- **DrawPolygon** با استفاده از یک مجموعه نقاط، یک چند ضلعی را در صفحه رسم می کند.
- **DrawIcon** یک آیکون را در صفحه نمایش می دهد.

تمام این متد ها اشیایی را از نوع Brush، Pen، Rectangle و یا Point که در طی فصل با آنها آشنا شدید را به عنوان ورودی دریافت می کنند. همچنین این توابع، توابع متناظری نیز دارند که با نام Fill شروع می شوند. وظیفه ی این توابع این است که اشکال را به صورت توپر رسم کنند.

نتیجه:

در این فصل مشاهده کردید که چگونه می توانید در فرمهای برنامه و یا در کنترلهای سفارشی که ایجاد می کنید، رابط کاربری را خودتان طراحی کنید. در قسمتهای قبلی، برای ایجاد یک کنترل سفارشی مجبور بودیم که برای طراحی رابط کاربری آن از کنترلهای موجود استفاده کنیم. اما در این بخش مشاهده کردید که چگونه می توان کنترل هایی با ظاهر دلخواه طراحی کرد. در طی فصل نیز بیشتر بر روی نوشتن کنترلهای سفارشی که از کلاس `System.Windows.Forms.UserControl` مشتق می شوند تمرکز کردیم تا با اصول برنامه نویسی مبتنی بر کامپوننت ها بیشتر آشنا شویم.
بعد از توضیح تصاویر برداری و تصاویر بیت مپی و تفاوتهای آنها، برنامه ای ایجاد کردیم که به کاربر اجازه می داد با قرار دادن تعدادی نقطه با استفاده از ماوس در کنار یکدیگر شکلی را رسم کند. سپس کنترلی ایجاد کردیم که کاربر به وسیله ی آن می توانست رنگ ابزار مورد استفاده خود را نیز تغییر دهد. همچنین مشاهده کردیم که چگونه می توان با استفاده از کادر `Color` رنگهای جدیدی را به برنامه اضافه کرد و یا با استفاده از سیستم نمایش رنگ به صورت RGB، رنگهای جدیدی را ایجاد کرد.
سپس نگاه مختصری به کلاس `Image` انداختیم و مشاهده کردیم که چگونه می توان با استفاده از این کلاس تصاویر با پسوندهای مختلف از قبیل `.gif` و یا `.bmp` را در صفحه نمایش داد.
در پایان این فصل باید با موارد زیر آشنا شده باشید:

- در رویداد هایی که به ماوس مربوط می شود، بتوانید مختصات X و Y اشاره گر ماوس در صفحه را بدست آورید.
- با نا معتبر مشخص کردن قسمتهای ضروری فرم، از کند شدن برنامه و چشمک زدن آن جلوگیری کنید.
- بتوانید از رنگهای سیستمی همانند رنگهای عادی و یا مقادیر RGB استفاده کنید.
- ابزارهای گرافیکی مختلفی همانند `Circle` و یا `HollowCircle` ایجاد کنید.
- تصاویری را در صفحه نمایش دهید و یا تغییراتی در اندازه ی آنها ایجاد کنید.

فصل پانزدهم: استفاده از بانکهای اطلاعاتی

اغلب برنامه های کامپیوتری که امروزه نوشته می شوند به نحوی با داده ها و اطلاعات مختلف کار می کنند. در ویـژوال C# ۲۰۰۵ بیشتر این برنامه ها، داده های مورد نیاز خود را در بانکهای اطلاعاتی رابطه ای نگهداری می کنند. بنابراین در هنگام نوشتن این نوع برنامه ها نیاز دارید که بتوانید در برنامه ی خود با نرم افزارهای مربوط به این نوع بانکهای اطلاعـاتی، ماننـد SQL Server، Access، Oracle و یا Sybase کار کنید.

در ویژوال C# ۲۰۰۵ ابزارها و ویزاردهای زیادی برای متصل شدن به انواع بانکهای اطلاعاتی وجود دارد. به وسـیله ی ایـن ابزارهـا می توانید اطلاعات خود را در درون این بانکهای اطلاعاتی قرار دهید و یا آن را از بانکهای اطلاعاتی دریافت کرده و تغییـرات مـورد نظر خود را روی آنها انجام دهید. در طی این فصل سعی می کنیم که با این ابزارها و نحوه ی کـارکرد آنهـا در برنامـه بیـشتر آشـنا شویم.

در فصل شانزدهم تمرکز خود را روی استفاده از بانکهای اطلاعاتی از طریق کد نویسی قرار خواهیم داد و مشاهده خـواهیم کـرد کـه چگونه می توان از طریق برنامه نویسی به صورت مستقیم به این بانکهای اطلاعاتی دسترسی پیدا کرد. بعد از اینکه مقـداری در کـد نویسی بانکهای اطلاعاتی تمرین کردید، خواهید دید که استفاده از کد نسبت به استفاده از ویزاردها و ابزار ها زمان بـسیار کمتـری را اشغال می کند.

در این فصل:

- با مفهوم بانکهای اطلاعاتی آشنا خواهید شد.
- با دستور SELECT در زبان SQL آشنا خواهید شد و از آن استفاده خواهید کرد.
- کامپوننت های دسترسی به داده های درون بانک اطلاعاتی را بررسی خواهید کرد.
- با نحوه ی استفاده از داده ها در برنامه های ویندوز آشنا خواهید شد.
- از ویزاردهای دسترسی به اطلاعات در ویژوال استودیو ۲۰۰۵ استفاده خواهید کرد.

نکته: برای انجام تمرینات و مثال های این فصل لازم است که نسخه ی 2000 (و یـا بـالاتر) برنامـه ی Microsoft Access که جزئی از برنامه ی Microsoft Office به شمار می رود را نصب کنید.

بانک اطلاعاتی چیست؟

اصولا هر **بانک اطلاعاتی**[1] شامل یک و یا چند فایل بزرگ و پیچیده است که داده ها در آن در یک قالب و فرمت سـاخت یافتـه ذخیره می شوند. **موتور بانک اطلاعاتی**[2] معمولا به برنامه ای اطلاق می شود که این فایل و یا فایلها و نیز داده های درون آنهـا را مدیریت می کند. در طی این فصل از برنامه ی Microsoft Access بـه عنـوان موتـور بانـک اطلاعـاتی اسـتفاده خواهیم کرد.

[1] Database
[2] Database Engine

اشیای موجود در Access:

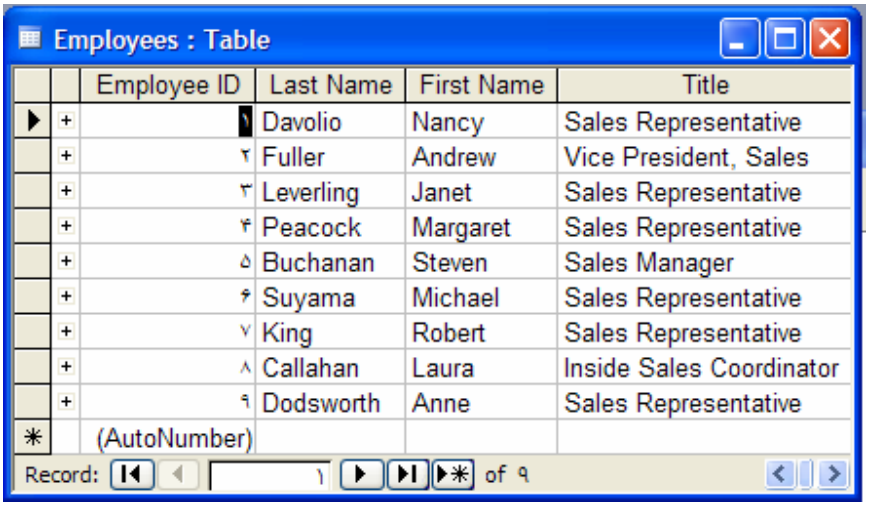
یک فایل بانک اطلاعاتی مربوط به برنامه ی Access (که پسوند آن نیز .mdb است) معمولاً از قسمتهای مختلفی مانند جدولها، پرس وجو ها، فرم ها، گزارشات، ماکرو ها و ماژول ها تشکیل شده است. به این قسمتهای تشکیل دهنده ی یک بانک اطلاعاتی، **اشیای بانک اطلاعاتی**[1] گفته می شود. در یک فایل مربوط به بانک اطلاعاتی عموماً داده های زیادی وجود دارند و به همین دلیل موتور های بانک اطلاعاتی مانند Access سعی می کنند با ارائه دادن امکانات اضافی، به کاربران اجازه دهند با این اطلاعات کار کنند. در بین اشیایی که در یک بانک اطلاعاتی Access وجود دارند، جدولها و پرس وجو ها برای نگهداری داده ها و یا دسترسی به آنها به کار می روند. دیگر اشیای یک بانک اطلاعاتی مانند فرم ها و یا گزارشات برای این است که کاربران بتوانند به سادگی با داده های موجود در جداول کار کنند.
اما به هر حال به علت پیچیده بودن ساختار موتور های بانک اطلاعاتی، کاربران معمولی حتی با استفاده از این قسمتها نیز نمی توانند به درستی از اطلاعات درون بانک اطلاعاتی استفاده کنند. هدف ما از نوشتن یک برنامه ی بانک اطلاعاتی با استفاده از ویژوال C# ۲۰۰۵ و یا هر زبان برنامه نویسی دیگر این است که به کاربر اجازه دهیم به سادگی از اطلاعات درون بانکها استفاده کند. پس در این برنامه ها فقط به اطلاعات درون یک بانک اطلاعاتی نیاز خواهیم داشت، نه به قسمتهایی مانند فرم ها و یا گزارشات. بنابراین در طی این فصل بیشتر تمرکز خود را روی دو قسمت اصلی بانکهای اطلاعاتی یعنی جدولها و پرس وجو ها قرار می دهیم.

جدولها:

یک **جدول**[2] شامل یک مجموعه از اطلاعات است که معمولاً حاوی یک و یا چند ستون و نیز یک و یا چند ردیف از داده ها است. در Access (و نیز بیشتر بانک های اطلاعاتی) به هر یک از این ستونها یک **فیلد**[3] گفته می شود. همچنین هر ردیف از این اطلاعات نیز یک **رکورد**[4] نامیده می شوند. هر فیلد در یک جدول از بانک اطلاعاتی، یکی از مشخصه های داده ای که در آن جدول ذخیره شده است را نگهداری می کند. برای مثال فیلدی به نام FirstName در یک جدول، مشخص کننده ی نام مشترک و یا کارمندی است که اطلاعات او در آن جدول ذخیره شده است. بنابراین این فیلد یکی از مشخصه های آن کارمند و یا مشترک را نمایش می دهد. در هر جدول، یک رکورد شامل یک مجموعه از فیلد ها است که اطلاعات و مشخصه های مربوط به یک نمونه از داده هایی که در آن جدول ذخیره شده است را نشان می دهد. برای مثال جدولی را در نظر بگیرید که دارای دو فیلد (دو ستون اطلاعات) به نامهای FirstName و LastName است و برای نگهداری اسامی کارمندان استفاده می شود. به مجموعه ی نام و نام خانوادگی هر کارمندی که اطلاعات او در این جدول وجود داشته باشد یک رکورد گفته می شود.
برای مثال در شکل ۱۵-۱، EmployeeID، FirstName و ... فیلد های این جدول و هر ردیف از اطلاعات نیز رکورد های آن را مشخص می کند.

---

[1] Database Objects
[2] Table
[3] Field
[4] Record



شکل ۱۵-۱

پرس وجو ها:

در هر بانک اطلاعاتی عموماً به یک سری از دستورات که زبان SQL[1] (به صورت "اس-کیو-ال" و یا "سی کو ال" تلفظ می شود) نوشته شده است و برای دریافت اطلاعات از بانک اطلاعاتی و یا ایجاد تغییراتی در اطلاعات موجود در بانک به کار می رود، یک **پرس وجو**[2] گفته می شود. با استفاده از پرس وجو ها می توانیم داده هایی را درون جدول های بانک اطلاعاتی وارد کنیم، آنها را از یک و یا چند جدول بدست آورده و یا تغییراتی را در آنها ایجاد کنیم.

در یک موتور بانک اطلاعاتی به دو روش می توانیم از پرس وجو ها استفاده کنیم. اول این است که یک دستور SQL را بنویسیم و سپس آن را اجرا کرده و نتیجه ی آن را مشاهده کنیم. روش دوم این است که همانند دیگر زبانهای برنامه نویسی یک زیر برنامه با استفاده از دستورات SQL ایجاد کنیم و سپس با فراخوانی آن زیر برنامه اطلاعات لازم را از بانک اطلاعاتی بدست آوریم[3]. در برنامه هایی که به زبان ویژوال C# می نویسیم نیز، هم می توانیم از یک زیر برنامه برای دسترسی به اطلاعات مورد نیاز استفاده کنیم و هم می توانیم دستور SQL مورد نیاز را با استفاده از برنامه به موتور بانک اطلاعاتی بفرستیم و نتایج حاصل را دریافت کرده و نمایش دهیم.

البته استفاده از زیر برنامه های موجود در یک موتور بانک اطلاعاتی نسبت به دستورات معمولی از سرعت بیشتری برخوردار است. زیرا موتور بانک اطلاعاتی می تواند دستورات درون آن زیر برنامه را تحلیل کرده و یک روش کلی برای سریعتر اجرا کردن آن ایجاد کند (به عبارت دیگر می تواند آنها را کامپایل کند). اما دستوراتی که به صورت عادی به موتور بانک اطلاعاتی می دهیم تا آنها را اجرا کند و داده های مربوط را برگرداند هر مرتبه لازم است که تفسیر شده و سپس اجرا شوند و این مورد باعث می شود که سرعت اجرای کمتری نسبت به زیر برنامه ها داشته باشند.

برای درک بهتر مفهوم پرس وجو ها بهتر است ابتدا مقداری با زبان SQL و دستورات آن آشنا شویم. خوشبختانه، زبان SQL نسبت به زبانهای برنامه نویسی دیگر بسیار ساده تر است و به سرعت می توان نحوه ی استفاده از آن را یاد گرفت.

---

[1] Structured Query Language

[2] Query

[3] به زیر برنامه هایی که برای پرس وجو از یک بانک اطلاعاتی نوشته می شود، زیر برنامه های ذخیره شده و یا Stored Procedures گفته می شود.

دستور SELECT در زبان SQL:

زبان SQL بر خلاف چیزی که ممکن است تصور کنید، زیاد مشابه زبانهای برنامه نویسی که تاکنون دیده اید نیست. دستورات این زبان به وسیله ی موسسه ی استاندارد ملی آمریکا (ANSI) به صورت استاندارد در آمده است. نسخه ی استاندارد این زبان که SQL ANSI نیز نامیده می شود، به وسیله ی تمام موتور های بانک اطلاعاتی پشتیبانی می شود. اما هر یک از این موتور های بانک اطلاعاتی امکانات مخصوص بیشتری را نیز به این زبان اضافه کرده اند که معمولاً فقط در همان موتور بانک اطلاعاتی قابل استفاده است.

مزایای یادگیری SQL ANSI در این است که، به این وسیله هنگامی که اصول دستورات زبان SQL را آموختید می توانید از آنها برای برنامه نویسی SQL در تمام موتور های بانک اطلاعاتی استفاده کنید. به این ترتیب برای این که بتوانید موتور بانک اطلاعاتی خود را تغییر دهید، فقط کافی است نحوه ی کارکرد با رابط گرافیکی آن را یاد بگیرید و سپس می توانید از دستورات SQL استاندارد در آن محیط نیز استفاده کنید. البته همانطور که گفتم هر موتور بانک اطلاعاتی دارای دستورات خاص خود است ک باعث افزایش کارایی و بهینه ساختن اجرای دستورات می شود. اما تا حد ممکن بهتر است از این دستورات در برنامه ای خود استفاده نکنید و دستورات استاندارد SQL را به کار ببرید. به این ترتیب می توانید هر زمان که لازم باشد به سادگی موتور بانک اطلاعاتی خود را تغییر دهید.

زبان SQL از تعداد کمی دستور تشکیل شده است که هر یک کار خاصی را انجام می دهند. یکی از پر کاربرد ترین و مهمترین این دستورات، دستور SELECT است که به وسیله ی آن می توانید یک یا چند فیلد اطلاعات مربوط به یک یا چند رکورد در جداول بانک اطلاعاتی خود را بدست آورید. البته دقت داشته باشید که به وسیله ی دستور SELECT فقط می توانید داده ها را از جداول بدست آورید، اما نمی توانید هیچ تغییری در آنها ایجاد کنید.

ساده ترین دستور SELECT در زبان SQL مشابه دستور زیر است:

```
SELECT * FROM Employees;
```

این دستور همانطور که مفهوم کلمات آن نیز مشخص می کنند، به این معنی است که " اطلاعات موجود در تمام فیلد های مربوط به همه ی رکورد های جدول Employees را انتخاب کن ". علامت * در دستور SELECT به معنی "تمام فیلد ها" است. کلمه ی Employees نیز نام جدولی در بانک اطلاعاتی است که این دستور باید بر روی آن اجرا شود.

اگر بخواهید فقط فیلد های مربوط به نام و نام خانوادگی افرادی که اطلاعات آنها در جدول Employees وارد شده است را بدست آورید، کافی است که علامت * را با نام فیلد های مورد نظر خود به صورت زیر عوض کنید:

```
SELECT [First Name], [Last Name] FROM Employees;
```

دقت کنید که هنگام وارد کردن این دستور حتماً باید از علامت بریس ([]) در ابتدای نام فیلد ها استفاده کنید. زیرا نام این فیلد ها حاوی فضای خالی (Space) است و باعث می شود که به برنامه در تفسیر نام First Name با مشکل مواجه شود. استفاده از بریس به موتور بانک اطلاعاتی می گوید که تا بسته شدن بریس را به عنوان یک نام در نظر بگیرد. البته اگر نام این فیلد حاوی کاراکتر فضای خالی نبود می توانستید از این بریسها استفاده نکنید.

همانطور که مشاهده می کنید دستورات SQL همانند زبان انگلیسی عادی و روزمره هستند و حتی فردی که برنامه نویس نیست نیز می تواند آن را خوانده و مفهوم آن را درک کند. برای مثال اگر بخواهیم فقط داده هایی که دارای شرط خاصی هستند از جدول انتخاب شده و نمایش داده شوند، کافی است از عبارت WHERE در پایان دستور SELECT استفاده کنیم. مثلاً اگر بخواهیم در دستور قبل فقط افرادی که نام خانوادگی آنها با حرف D شروع می شوند انتخاب شوند، باید از دستور زیر استفاده کنیم:

```
SELECT [First Name], [Last Name] FROM Employees
              WHERE [Last Name] LIKE 'D*';
```

عبارت WHERE باعث می شود فقط داده هایی از جدول انتخاب شوند که در شرط مقابل عبارت WHERE صدق می کنند. بنابراین دستور SELECT قبلی باعث می شود که موتور بانک اطلاعاتی به داخل جدول Employees برود و فیلد First Name و Last Name تمام رکورد هایی که Last Name آنها با حرف D شروع می شود را انتخاب کند. عبارت 'D*' نیز به این معنی است که "هر عبارتی که با حرف D شروع شده است". برای مثال عبارت '*D*' به این معنی است که "هر عبارتی که در آن حرف D وجود داشته باشد".
در آخر نیز بعد از اینکه داده های مورد نظر خود را انتخاب کردید می توانید آنها را به نحوی که تمایل دارید به صورت صعودی و یا نزولی مرتب کنید، برای مثال بر اساس فیلد First Name. برای این کار باید در انتهای دستور SELECT از عبارت ORDER BY استفاده کنید:

```
SELECT [First Name], [Last Name] FROM Employees
     WHERE [Last Name] LIKE 'D*' ORDER BY [First Name];
```

اجرای این دستور باعث می شود اطلاعاتی که از جدول انتخاب می شوند، قبل از نمایش داده شدن بر اساس فیلد First Name و به صورت صعودی مرتب شوند. برای مثال خروجی این دستور می تواند مانند زیر باشد:

```
Angela     Dunn
David      Dunstan
Zebedee    Dean
```

همانطور که مشاهده می کنید در این قسمت از یک دستور تقریباً کامل استفاده کردیم، اما درک آن نیز بسیار ساده بود و تقریباً بسیار مشابه چیزی بود که در زبان انگلیسی برای بیان منظور خود باید عنوان کنید. معمولاً هنگامی که اطلاعات را بر اساس فیلد های رشته ای مرتب می کنید، داده ها به صورت صعودی مرتب می شوند. به این صورت که اطلاعات با حرف A ابتدا و اطلاعات با حرف Z در انتها نمایش داده می شوند. اما هنگامی که بخواهید اطلاعات را بر اساس یک فیلد عددی مرتب کنید، ممکن است تمایل داشته باشید که داده های بزرگتر ابتدا نمایش داده شوند. برای مثال ممکن است بخواهید اطلاعاتی که انتخاب می شوند، بر اساس قیمت کالا مرتب شده و کالاهای گرانتر نیز در بالای جدول قرار بگیرند. بنابراین لازم است که اطلاعات را به صورت نزولی مرتب کنید. برای این کار کافی است در پایان دستور ORDER BY از عبارت DESC[1] استفاده کنید. به این ترتیب داده ها به صورت نزولی مرتب خواهند شد.

```
SELECT [First Name], [Last Name] FROM Employees
WHERE [Last Name] LIKE 'D*' ORDER BY [First Name] DESC;
```

اجرای دستور بالا نتایجی را مشابه زیر برمی گرداند:

```
Zebedee    Dean
David      Dunstan
```

[1] Descending

```
Angela     Dunn
```

نکته: اگر می خواهید در دستور خود مشخصاً قید کنید که اطلاعات باید بر اساس صعودی مرتب شوند، می توانید در انتهای دستور ORDER BY از عبارت ASC استفاده کنید. البته استفاده از این عبارت الزامی نیست زیرا به صورت پیش فرض اطلاعات به صورت صعودی مرتب می شوند.

به طور خلاصه می توان گفت که دستور SELECT می تواند با ساختاری مشابه زیر مورد استفاده قرار بگیرد:

```
SELECT select-list
      FROM table-name
      [WHERE search-condition]
      [ORDER BY order-by-expression [ASC | DESC]]
```

این عبارت به این معنی است که در قسمت *select-list* حتماً باید لیستی از نام فیلد های مورد نظر و یا علامت * برای انتخاب تمام فیلد ها را ذکر کنید. همچنین در قسمت *table-list* نیز باید نام جدول مورد نظر را بیاورید. می توانید از عبارت WHERE در دستور SELECT خود استفاده کنید. به این ترتیب فقط داده هایی که در شرط *search-condition* صدق می کنند انتخاب خواهند شد. با استفاده از قسمت ORDER BY نیز می توانید داده ها را مرتب کنید. برای این کار باید در قسمت *order-by-expression* فیلدی که می خواهید داده ها بر اساس آن مرتب شوند را ذکر کنید. برای صعودی و یا نزولی بدون مرتب سازی نیز می توانید از عبارت ASC و یا DESC در انتهای دستور استفاده کنید.
البته اگر بخواهید داده ها را از چندین جدول یک بانک اطلاعاتی استخراج کنید و یا بر اساس رابطه ی خاصی به داده ها دسترسی پیدا کنید، دستورات SQL به مقدار قابل ملاحظه ای پیچیده خواهند شد که توضیح این گونه دستورات از اهداف این کتاب خارج است و در برنامه های این فصل و فصل بعد نیز به آنها نیازی نخواهیم داشت.
در هر حال بهترین روش برای یادگیری نحوه ی استفاده از دستورات SQL، تمرین و کار کردن با این دستورات است. قبل از این که به ادامه ی فصل بپردازیم بهتر است به سوالات زیر به صورت ذهنی پاسخ دهید.

- چگونه می توانیم یک دستور SELECT بنویسیم که داده های موجود در فیلد های Name، Description و Price را از یک جدول به نام Products استخراج کند؟
- چگونه می توان دستور بالا را به گونه ای تغییر داد تا فقط داده هایی را برگرداند که در فیلد Description آنها عبارت DVD وجود داشته باشد؟
- چگونه می توان اطلاعات بالا را بر اساس قسمت به گونه ای مرتب کرد که اجناس گرانتر در ابتدای جدول قرار بگیرند؟

پرس وجو ها در Access:

در کار با بانکهای اطلاعاتی، زبان SQL از اهمیت خاصی برخوردار است. به گونه ای که اگر بخواهید برنامه ای بنویسید که از بانکهای اطلاعاتی و داده های درون آن استفاده کند، در مواقع زیادی به استفاده از این زبان نیاز پیدا خواهید کرد. در برنامه ی Access ابزارها و ویزاردهای زیادی وجود دارد که به برنامه نویسان تازه کار کمک می کند بتوانند دستورات SQL مورد نظر خود را ایجاد کنند. البته این ابزارها در بعضی مواقع به برنامه نویسان حرفه ای نیز در نوشتن دستورات SQL کمک زیادی می کنند. در ادامه ی این بخش با نحوه ی استفاده از این ابزارها در محیط Access آشنا خواهیم شد. این ابزارها در انتها، یک سری

دستورات SQL تولید می کنند که می توانید آنها را مشاهده کرده و تغییرات مورد نظر خود را در آنها ایجاد کنید. بررسی و مرور این دستورات و نتایج اجرای آنها، می تواند کمک زیادی به یادگیری دستورات SQL بکند.

ایجاد یک پرس وجو:

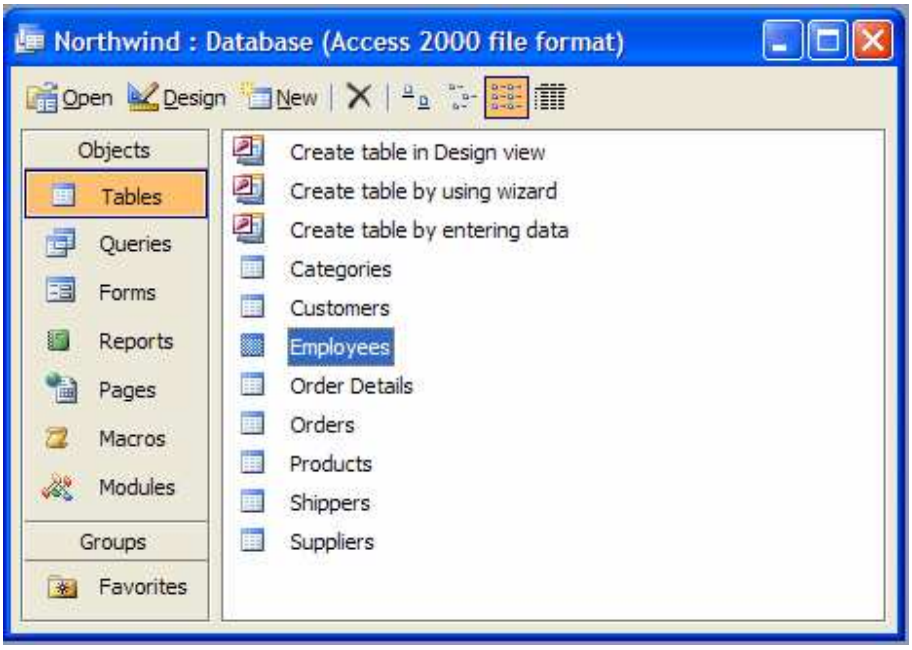
در بخش امتحان کنید بعد، با استفاده از برنامه ی Access یک پرس وجوی ساده ایجاد خواهیم کرد تا بتواند اطلاعات مربوط به مشترکین که در جدول Customers در بانک اطلاعاتی Northwind.mdb ذخیره شده است را بدست آورده و نمایش دهد. برای اجرای این تمرین لازم است بانک اطلاعاتی نمونه ای که همراه با Microsoft Office نصب می شود در سیستم شما وجود داشته باشد. به این ترتیب می توانیم یک پرس وجوی نمونه ایجاد کرده و دستور SQL تولید شده به وسیله ی Access را با هم مشاهده کنیم.

امتحان کنید: ایجاد یک پرس وجو

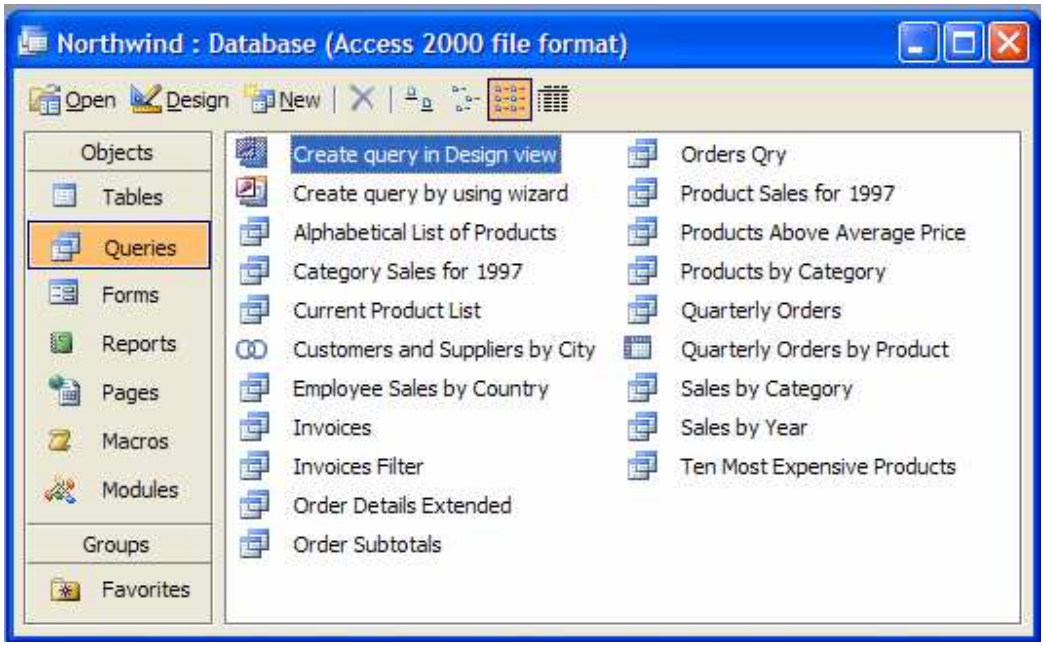
۱) برنامه ی Microsoft Access را باز کرده و در نوار ابزار این برنامه، روی آیکون Open کلیک کنید. سپس در کارد محاوره ای Open به آدرس C:\Program Files\Microsoft Office\Office11\Samples بروید و فایل Northwind.mdb را انتخاب کرده و روی دکمه ی OK کلیک کنید.

نکته: آدرس نصب برنامه ی Office بر اساس نسخه ای از Office که از آن استفاده می کنید و نیز مسیری که هنگام نصب انتخاب کرده اید ممکن است تفاوت داشته باشد.

۲) با باز شدن فایل مربوط به بانک اطلاعاتی Northwind.mdb، پنجره ای در برنامه نمایش داده می شود که در نوار سمت چپ آن دو قسمت به نامهای Objects و Groups وجود دارد. در بخش Objects اشیای موجود در بانک اطلاعاتی (که پیشتر در مورد آنها صحبت کردیم) نمایش داده می شوند. در بخش Groups نیز می توانید اشیایی از هر نوع را که به هم ارتباط دارند در یک گروه قرار دهید (شکل ۱۵-۲).
۳) برای اینکه ببینید چگونه برنامه ی Access می تواند به صورت اتوماتیک دستور SELECT مورد نیاز شما را ایجاد کند، باید در قسمت Objects روی آیکون Queries کلیک کنید.
۴) در این قسمت باید یک پرس وجوی جدید ایجاد کنیم، بنابراین در قسمت سمت راست روی عبارت "Create query in Design view" کلیک کنید (شکل ۱۵-۳)



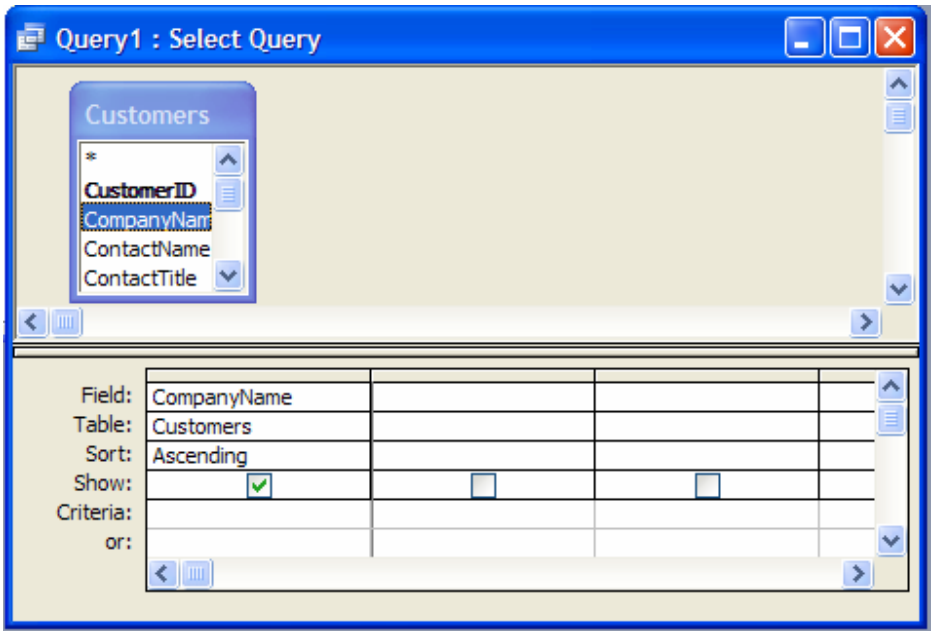
شکل ۱۵-۲



شکل ۱۵-۳

۵) به این ترتیب پنجره ی `Show Table` نمایش داده می شود تا به وسیله ی آن بتوانید جدول و یا جداولی که می خواهید از آنها در پرس وجوی خود استفاده کنید را انتخاب کنید. در این قسمت فقط به یک جدول نیاز داریم: `Customers`. بنابراین در این پنجره جدول `Customers` را انتخاب کرده و روی دکمه ی `Add` کلیک کنید تا این جدول به قسمت `Query Designer` اضافه شود. سپس روی دکمه ی `Close` کلیک کنید تا پنجره ی `Show Table` بسته شود.

۶) به این ترتیب تمام فیلد های موجود در جدول Customer به همراه یک علامت ستاره نمایش داده می شود. به وسیله ی این لیست می توانید فیلد های مورد نیاز خود را انتخاب کنید و یا با انتخاب علامت * مشخص کنید که تمام فیلد ها باید انتخاب شوند. در این قسمت فقط به فیلد های محدودی نیاز داریم، بنابراین لازم نیست که همه ی آنها را انتخاب کنیم. در لیست فیلد های موجود در برنامه، روی فیلد CompanyName در جدول Customers کلیک کنید تا به اولین ستون در جدول پایین پنجره اضافه شود. مشاهده خواهید کرد که قسمت Field و Table در این ستون به صورت اتوماتیک پر خواهند شد. همچنین می توانید قسمت Sort را نیز تنظیم کنید تا داده ها بر اساس این فیلد، به صورت نزولی و یا صعودی مرتب شوند. در این قسمت می خواهیم داده ها بر اساس صعودی مرتب شوند، بنابراین روی قسمت Sort کلیک کرده و گزینه ی Ascending را انتخاب کنید. به این ترتیب پنجره ی شما باید مشابه شکل ۱۵-۴ شده باشد.

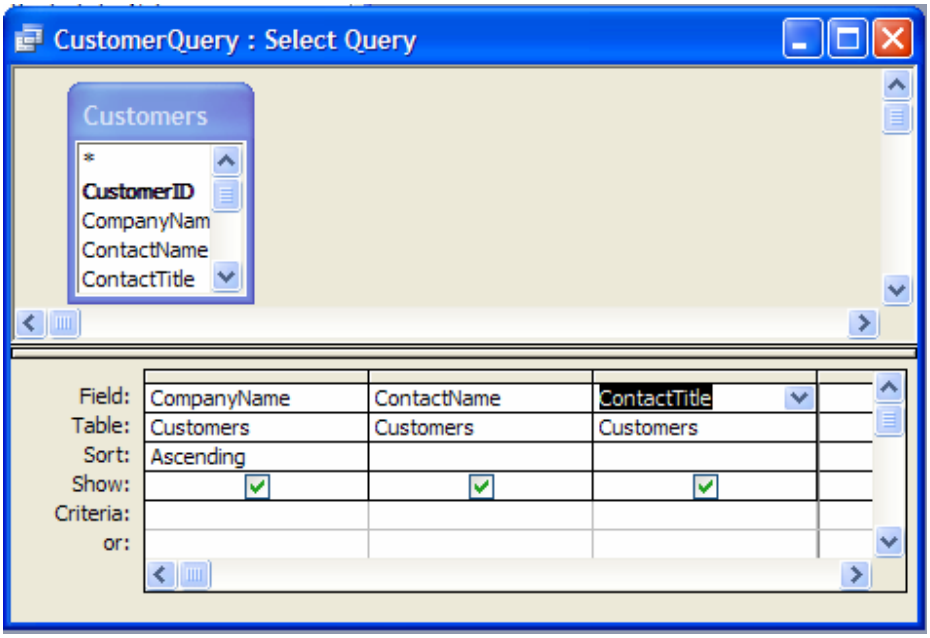


شکل ۱۵-۴

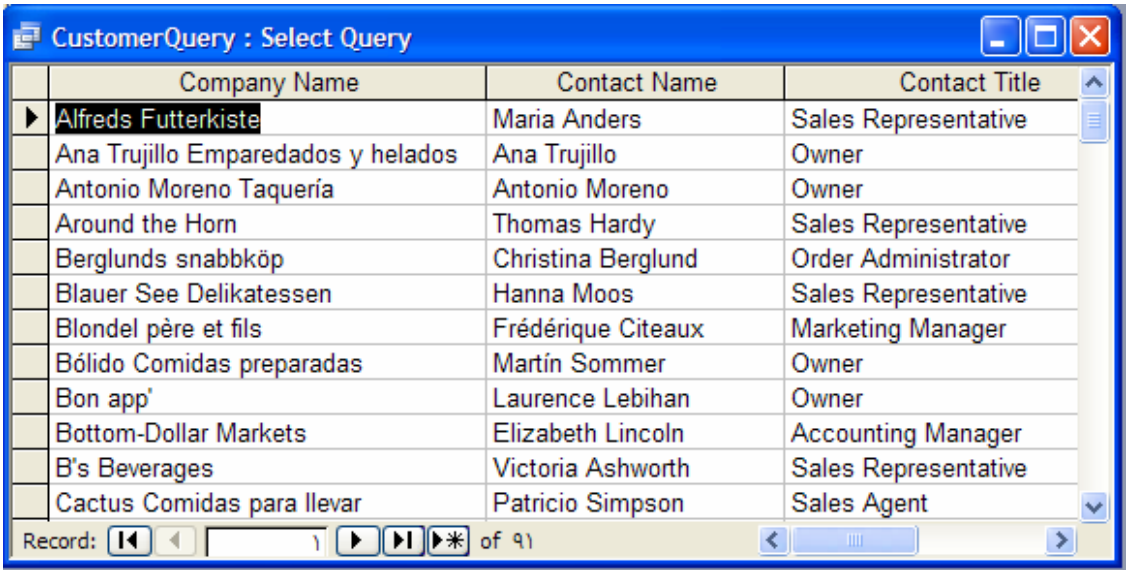
۷) علاوه بر فیلد CompanyName لازم است که فیلد ContactName را نیز به پرس وجو اضافه کنیم. بنابراین روی فیلد ContactName دو بار کلیک کنید تا به صورت اتوماتیک در ستون بعدی در جدول پایین صفحه قرار بگیرد. سپس به همین روش فیلد ContactTitle را نیز اضافه کنید. در این مرحله پنجره ی برنامه ی شما باید مشابه شکل ۱۵-۵ شده باشد.

۸) روی دکمه ی Save در نوار ابزار کلیک کنید و عبارت CustomerQuery را در پنجره ی Save As وارد کرده و کلید OK را فشار دهید. به این ترتیب پرس وجوی شما به این نام در بانک اطلاعاتی ذخیره می شود.

۹) در نوار ابزار روی آیکون Run که به وسیله ی یک علامت ! مشخص شده است کلیک کنید تا این پرس وجو اجرا شده و نتیجه ی آن نمایش داده شود. به این ترتیب نتیجه ای را همانند شکل ۱۵-۶ مشاهده خواهید کرد. دقت کنید که نتایج نمایش داده شده بر اساس فیلد CompanyName به صورت صعودی مرتب شده اند.



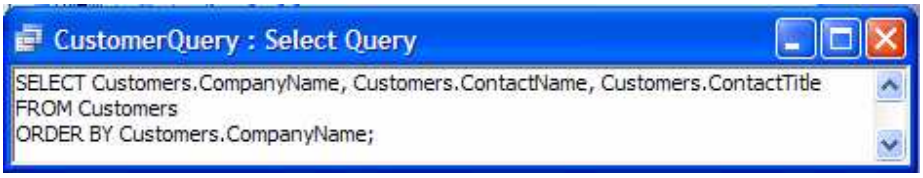
شکل ۱۵-۵



شکل ۱۵-۶

چگونه کار می کند؟

بر اساس تنظیمات و انتخاب هایی که در قسمت قبل انجام دادید، Access دستور SQL معادل را نوشته و آن را اجرا می کند و سپس نتیجه را نمایش می دهد. برای مشاهده ی دستور تولید شده، از منوی View گزینه ی SQL View را انتخاب کنید. به این ترتیب پنجره ی مشابه شکل ۱۵-۷ که حاوی دستور SELECT مورد نظر است نمایش داده می شود.



شکل ۱۵-۷

توجه کنید که در این قسمت یک دستور SELECT به همراه نام فیلد های مورد نیاز، مشابه مواردی که در قسمت قبل بررسی کردیم ایجاد شده است. البته Access برای مشخص تر شدن دستور، قبل از نام فیلد ها نام جدول حاوی آنها را نیز قرار می دهد. این مورد هنگامی که بخواهیم فیلد هایی را از چند جدول انتخاب کرده و نمایش دهیم، از پیش آمدن اشتباه بین فیلد ها با نام برابر (اما در جدولهای متفاوت) جلوگیری می کند. دقت کنید که برای استفاده از فیلد ها در این قسمت از بریس استفاده نشده است. دلیل این مورد هم در این است که استفاده از بریس فقط وقتی ضروری است که در نام فیلد ها فضای خالی وجود داشته باشد.
بعد از این قسمت، عبارت FROM به همراه نام جدولی که می خواهیم اطلاعات از آن استخراج شود آمده است (در این حال، جدول Customers). عبارت ORDER BY نیز در انتهای دستور مشخص می کند که اطلاعات باید بر اساس فیلد Customers.CompanyName مرتب شوند.
خوب، حال که دستور ایجاد شده در این قسمت را مشاهده کردیم، بهتر است بررسی کنیم که این دستورات چگونه به وسیله ی Access ایجاد می شوند؟ در ابتدای ایجاد یک پرس وجو، هنگامی که جدول مورد نظر خود مانند Customers را انتخاب می کنید، Access دستوری به صورت زیر ایجاد می کند:

```
SELECT
FROM Customers;
```

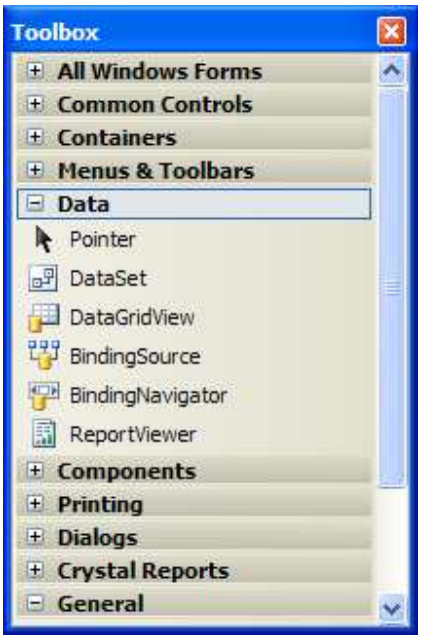
البته مشخص است که این دستور، یک دستور قابل استفاده نیست، زیرا در این دستور مشخص نیست که چه فیلد هایی باید انتخاب شوند. هنگامی که اولین فیلدی که باید در این پرس وجو وجود داشته باشد و نوع مرتب شدن آن را مشخص کردید، دستور SQL قبلی به صورت زیر تغییر داده می شود که یک دستور قابل اجرا است. به عبارت دیگر نام فیلد مورد استفاده در مکان مناسب قرار داده می شود و نیز عبارت مربوط به مرتب سازی به صورت صعودی نیز به دستور اضافه می شود.

```
SELECT Customers.CompanyName
FROM Customers
ORDER BY Customers.CompanyName;
```

به این ترتیب هنگامی که فیلد های دیگری را به این پرس وجو اضافه می کنید، نام آن فیلد ها بعد از نام فیلد CompanyName در خط اول دستور قرار خواهد گرفت و در انتها دستور به آنچه که در شکل ۱۵-۷ مشاهده می کنید تبدیل می شود.
خوب، بهتر است مجدداً به محیط ویژوال استودیو برگشته و مقداری با کامپوننت های دسترسی به داده ها که برای کار با بانکهای اطلاعاتی در برنامه های ویندوزی لازم هستند آشنا شویم. به علت اینکه در این فصل از Access به عنوان موتور بانک اطلاعاتی استفاده خواهیم کرد، در این قسمت نیز فقط کامپوننت هایی را بررسی خواهیم کرد که برای دسترسی به بانکهای اطلاعاتی Access مورد نیاز است.

کامپوننت های دسترسی اطلاعات:

در ویژوال C# ۲۰۰۵ برای دسترسی به اطلاعات و نمایش آنها سه کامپوننت مهم و اصلی وجود دارند که عبارتند از : BindingSource، TableAdapter و DataSet. دو کامپونـــــــــــت BindingSource و DataSet همانطور که در شکل ۱۵-۸ مشاهده می کنید در قسمت Data در جعبه ابزار وجود دارند. کامپوننت TableAdapter نیز بر اساس مسیری که برای دسترسی به اطلاعات درون بانک اطلاعاتی و نمایش آنها طی می کنید به صورت اتوماتیک ایجاد خواهد شد.



شکل ۱۵-۸

نکته: این کامپوننت ها که عموماً به عنوان کامپوننت های داده ای شناخته می شوند، خود فقط چندین کلاس هستند، مانند تمام کلاس های دیگر .NET که در قسمتهای قبلی از آنها استفاده کردیم. در این فصل فقط با نحوه ی استفاده از این کلاسها در برنامه های ویندوزی آشنا می شویم. در فصل بعد سعی می کنیم که کلاسهای مربوط به این کامپوننت ها را با جزئیات بیشتری بررسی کنیم.

:DataSet

کامپوننت DataSet در حقیقت همانند یک مخزن است که داده های مورد نیاز را در حافظه ی کامپیوتر نگهداری می کند. این کامپوننت همانند یک موتور بانک اطلاعاتی کوچک عمل می کند که داده های مورد نیاز خود را در حافظه نگهداری می کنند. با استفاده از این کامپوننت می توانید داده ها را درون جدولهایی نگهداری کرده و سپس با استفاده از کامپوننت DataView که در فصل شانزدهم توضیح داده خواهد شد، به چندین روش پرس وجو هایی را روی این داده ها اجرا کنید.

کامپوننت DataSet از قدرت و امکانات زیادی برخوردار است. این کامپوننت علاوه بر این توانایی ذخیره ی داده ها در جداول، حجم زیادی از **متادیتا**[1] ، یا "اطلاعاتی درباره ی داده های موجود"، را نیز نگه داری می کند. این اطلاعات شامل مواردی مانند نام جدول ها و یا فیلد ها، نوع داده های موجود اطلاعات مورد نیاز برای مدیریت داده ها و یا اطلاعاتی در رابطه با لغو کردن تغییرات اعمال شده در داده ها می باشد.
تمام این اطلاعات در قالب XML در حافظه ذخیره می شوند. به علاوه یک کامپوننت DataSet می تواند به سادگی در قالب XML در دیسک ذخیره شده و یا از قالب XML از دیسک در حافظه قرار داده شود. همچنین این کنترل می تواند به صورت XML از طریق شبکه های مختلف مانند اینترنت به برنامه های دیگر فرستاده شود و مورد استفاده قرار گیرد.
به علت اینکه داده های یک کامپوننت DataSet در حافظه قرار دارند، بنابراین می توانید به سادگی در بین آنها به جلو و یا عقب حرکت کنید و یا در آنها تغییراتی را ایجاد کنید. البته این تغییرات در داده های موجود در حافظه اعمال می شوند و تا زمانی که مشخص نکنید به داده ای موجود در بانک اطلاعاتی منعکس نخواهند شد. در مورد این کامپوننت در فصل بعد بیشتر صحبت خواهیم کرد، اما در این فصل فقط داده هایی را در آن قرار داده و سپس به وسیله ی کنترلهای دیگری آن داده ها را در برنامه نمایش خواهیم داد.

DataGridView:

این کنترل برای نمایش داده های موجود در یک بانک اطلاعاتی در فرم برنامه به کار می رود. برای کار با آن کافی است آن را به منبع داده های خود، برای مثال یکی از جدولهای موجود در بانک اطلاعاتی، متصل کرده و سپس این کنترل را تنظیم کنید تا آن داده ها را همانند یک جدول، یعنی ستونها را به صورت عمودی و ردیفها را به صورت افقی نمایش دهد.
همچنین این کنترل دارای خاصیتهای زیادی است که به وسیله ی آنها می توانید ظاهر آن را تنظیم کنید و تا به شکلی که مد نظر شماست تبدیل شود. علاوه بر این به وسیله ی این کنترل می توانید عنوان ستونهای داده ها و یا روش نمایش آنها نیز را تعیین کنید.

BindingSource:

این کامپوننت همانند پلی برای ایجاد ارتباط بین داده های موجود در منبع داده ای شما (DataSet) و نیز کنترل هایی که برای نمایش داده ها مورد استفاده قرار می گیرند به کار می رود. بنابراین هنگامی که بخواهید به وسیله ی کنترل هایی داده های موجود در برنامه ی خود را نمایش دهید، و یا به هر دلیل دیگری بخواهید به آنها در منبع اطلاعاتی دسترسی داشته باشید، این ارتباط باید از طریق این کامپوننت صورت بگیرد.
برای مثال تصور کنید که داده های موجود در یک DataSet را به وسیله ی یک کنترل DataGridView در فرم برنامه نمایش داده اید و حال می خواهید این داده ها بر اساس یکی از ستونها مرتب شده و سپس نمایش داده شوند. برای این کار کنترل DataGridView این تقاضا را به کامپوننت BindingSource می فرستد و سپس این کامپوننت آن را به کامپوننت DataSet اعلام می کند.
در ادامه ی این فصل با نحوه ی استفاده از این کامپوننت بیشتر آشنا خواهیم شد.

---

[1] Metadata

BindingNavigator:

کنترل BindingNavigator یک رابط گرافیکی استاندارد را برای حرکت بین رکورد های موجود در یک بانک اطلاعاتی ایجاد می کند. این کنترل بسیار مشابه کنترلی است که در پایین جدول نمایش داده شده در شکل ۱۵-۶ مشاهده می کنید. این کامپوننت نیز همانند کامپوننت DataGridView می تواند به کنترل BindingSource متصل شده و از طریق آن به داده های موجود در برنامه دسترسی داشته باشد. به این ترتیب برای مثال هنگامی که روی کلید Next در این کامپوننت کلیک کردید تا به رکورد بعدی اطلاعات بروید، درخواست شما به وسیله ی BindingNavigator به کامپوننت BindingSource فرستاده شده و سپس از کامپوننت BindingSource به کامپوننت DataSet (و یا هر منبع اطلاعاتی دیگر که در برنامه از آن استفاده می کنید) اعلام می شود.

TableAdapter:

تنها یک کامپوننت داده ای دیگر مانده است که باید در مورد آن صحبت کنیم: کامپوننت DataAdapter. این کامپوننت در جعبه ابزار وجود ندارد که بتوانید آن را همانند کامپوننت های قبلی بر روی فرم قرار دهید. بلکه بسته به روشی که کامپوننت های داده ای دیگر را در برنامه قرار داده و آنها را تنظیم می کنید، این کامپوننت به صورت اتوماتیک ایجاد می شود.
این کامپوننت حاوی پرس وجو هایی برای انتخاب داده های موجود در بانک اطلاعاتی و نیز اطلاعاتی در مورد نحوه ی اتصال برنامه به بانک است. همچنین این کامپوننت حاوی متد هایی است که به وسیله ی آنها می توان داده ها را از جداول بانک اطلاعاتی بدست آورد و در کامپوننت هایی مانند DataSet قرار داد و سپس در برنامه از آن داده ها استفاده کرد.
این کامپوننت این قابلیت را دارد که بر اساس دستور SELECTای که برای انتخاب داده ها از بانک اطلاعاتی برای آن وارد می کنید، دستورات INSERT، UPDATE و نیز DELETE[1] مناسب برای تغییر داده های انتخاب شده در بانک اطلاعاتی ایجاد کند.
در فصل بعد بیشتر با این کامپوننت آشنا خواهیم شد.

اتصال داده ها:

اتصال داده ها[2] به این معنی است که داده هایی که به وسیله ی کامپوننت BindingSource به آنها دسترسی دارید را به یک کنترل خاص نسبت دهید. به عبارت دیگر یک کنترل را بتوانید به نحوی تنظیم کنید که داده های مورد نیاز خود را به وسیله ی کامپوننت های دسترسی داده ها در برنامه دریافت کند و سپس آنها را به صورت اتوماتیک به کاربر نمایش دهد. به این ترتیب کاربر می تواند آنها را مشاهده کرده و یا تغییرات مورد نظر خود را در آنها اعمال کند. در ویژوال C# تقریباً تمام کنترل ها تا حدی اتصال به داده ها را پشتیبانی می کنند، اما بعضی از کنترل ها نیز وجود دارند که مخصوص این کار طراحی شده اند، مانند کنترل DataGridView و یا TextBox. در بخش امتحان کنید بعد اطلاعاتی که به وسیله ی کامپوننت

[1] این دستورات نیز مانند دستور SELECT از دستورات زبان SQL به شمار می روند و برای آشنایی با آنها می توانید به کتابهای آموزشی زبان SQL مراجعه کنید.

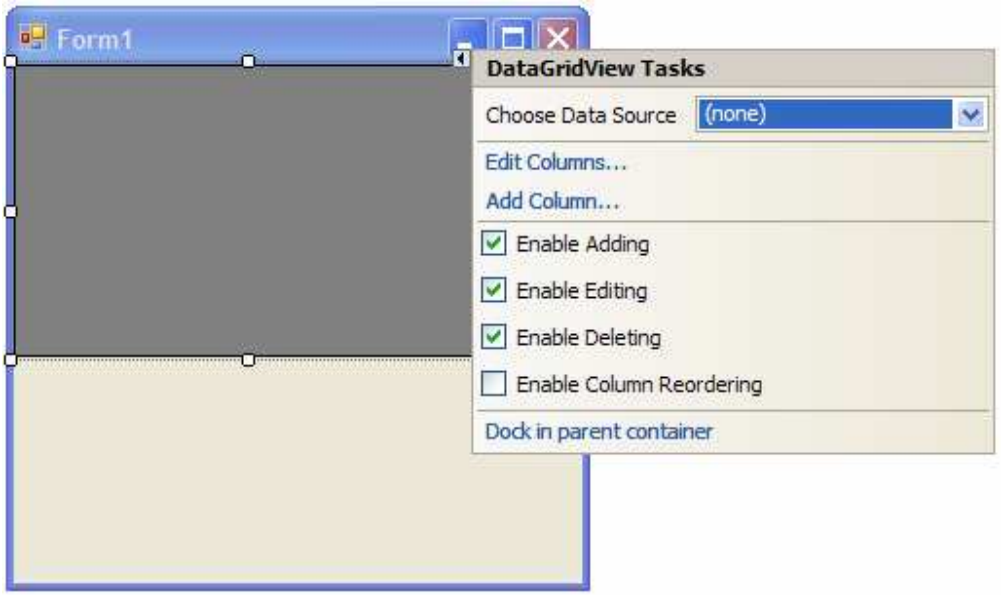
[2] Data Binding

BindingSource به آنها دسترسی داریم را به کنترل DataGridView متصل کرده و به وسیله ی این کنترل نمایش می دهیم. در بخش بعد نیز این اطلاعات را به وسیله ی کنترل TextBox در برنامه نمایش خواهیم داد.

امتحان کنید: متصل کردن داده ها به کنترل DataGridView

۱) با استفاده از ویژوال استودیو ۲۰۰۵ یک برنامه ی ویندوزی جدید به نام Northwind Customers DataGridView ایجاد کنید.

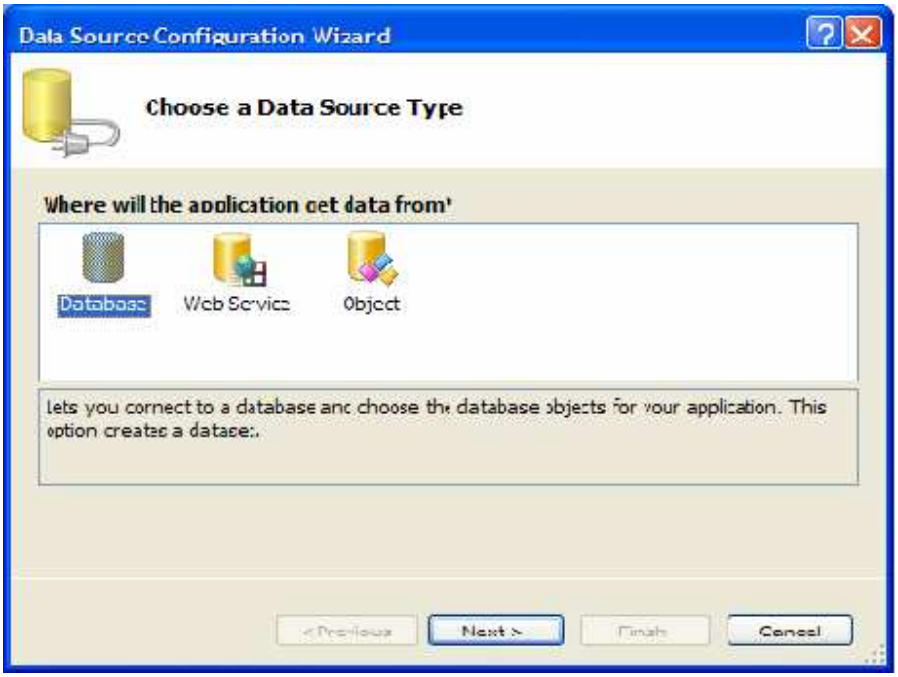
۲) با استفاده از جعبه ابزار به قسمت Data بروید و سپس روی کنترل DataGridView دو بار کلیک کرده تا یک نمونه از این کنترل روی فرم برنامه قرار بگیرد. به این ترتیب کادر DataGridView Tasks به صورت اتوماتیک همانند شکل ۱۵-۹ نمایش داده خواهد شد.



شکل ۱۵-۹

۳) در این کادر، در لیست روبروی عبارت Choose Data Source کلیک کرده و سپس در این لیست روی لینک Add Project Data Source کلیک کنید. به این ترتیب ویزارد Data Source Configuration Wizard نمایش داده خواهد شد.

۴) در صفحه ی اول این ویزارد، یعنی پنجره ی Choose a Data Source Type می توانید منبع داده ای مورد نظر خودتان را انتخاب کنید. همانطور که در شکل ۱۵-۱۰ نیز مشاهده می کنید در این قسمت می توانید انواع مختلفی از منبع های داده ای را مشخص کرده و به آنها متصل شوید. برای مثال اگر می خواهید به یک بانک اطلاعاتی که توسط نرم افزارهای مختلفی مانند SQL Server، Oracle، Access و یا ... ایجاد می شود دسترسی داشته باشید، روی آیکون Database کلیک کنید. اگر می خواهید از طریق یک وب سرویس به بانک اطلاعاتی خود متصل شوید روی آیکون Web Service کلیک کنید. آیکون Objects نیز برای دسترسی به کامپوننت های داده ای در لایه ی منطق تجاری به کار می رود.
در این قسمت آیکون Database را انتخاب کرده و سپس روی دکمه ی Next کلیک کنید.



شکل ۱۵-۱۰

۵) در پنجره ی Choose Your Data Connection روی دکمه ی New Connection کلیک کنید.

۶) به این ترتیب پنجره ی Choose Data Source نمایش داده خواهد شد. در این پنجره گزینه ی Microsoft Access Database File را از لیست Data Source انتخاب کرده و روی دکمه ی Continue کلیک کنید.

۷) در کادر Add Connection روی دکمه ی Browse کلیک کرده و سپس به فولدر Samples در مکان نصب برنامه ی Office بروید. این فولدر به صورت پیش فرض برای Office 2003 در آدرس C:\Program Files\Microsoft Office\Office11\Samples قرار دارد.
در این آدرس فایل Northwind.mdb را انتخاب کرده و سپس روی دکمه ی OK کلیک کنید تا نام و مسیر فایل انتخابی به کادر متنی موجود در پنجره ی Add Connection اضافه شوند. سپس در این پنجره نیز روی دکمه ی OK کلیک کنید تا کادر Add Connection بسته شود و به پنجره ی Choose Your Data Connection برگردید. در این پنجره نیز روی دکمه ی Next کلیک کنید.
به این ترتیب کادر پیغامی نمایش داده خواهد شد و از شما می پرسد که فایل بانک اطلاعاتی که انتخاب کرده اید جزئی از پروژه نیست. آیا می خواهید این فایل به فولدر پروژه کپی شده و از نسخه ی کپی آن استفاده شود؟ در این کادر روی دکمه ی Yes کلیک کنید.

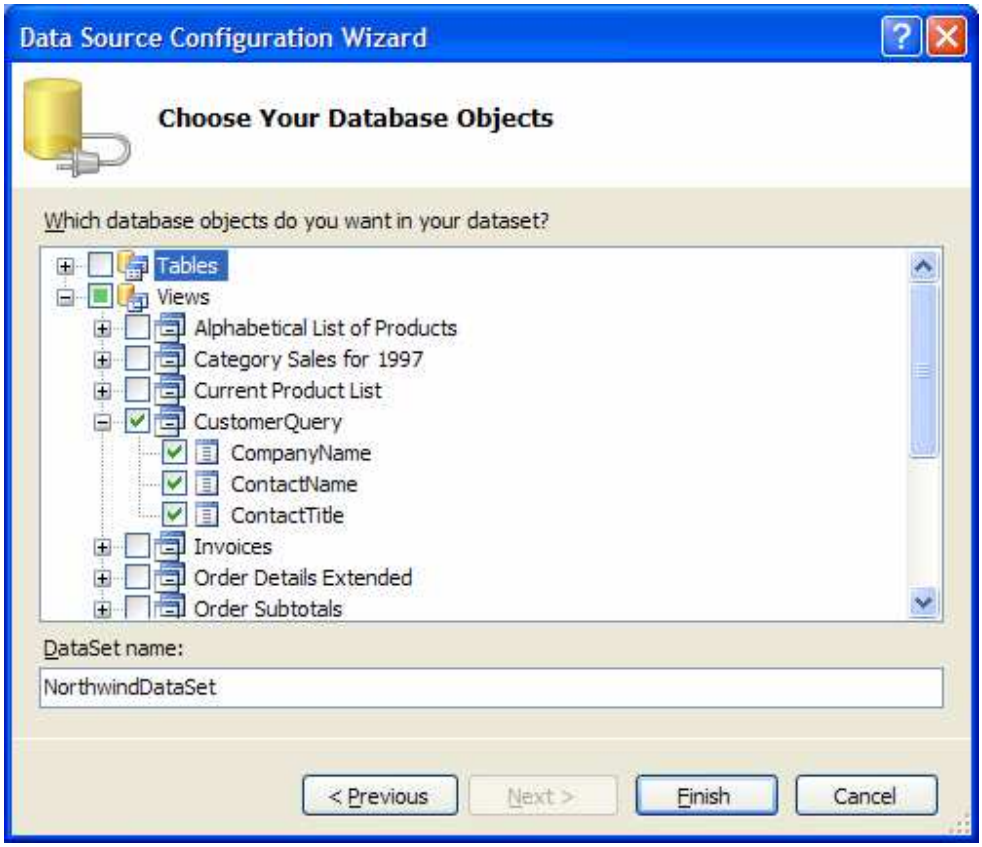
۸) به این ترتیب پنجره ی Save the Connection String on the Application Configuration File نمایش داده می شود. در این پنجره نیز روی دکمه Next کلیک کنید.

۹) بعد از طی این مراحل پنجره ی Choose Your Data Objects نمایش داده می شود و به شما اجازه می دهد تا داده های مورد نیاز در برنامه را انتخاب کنید. در این قسمت می توانید انتخاب کنید که داده های مورد

نیاز شما از یک جدول درون بانک اطلاعاتی وارد برنامه شوند، با اجرای پروسیجر های ذخیره شده در بانک اطلاعاتی ایجاد شده و در اختیار برنامه قرار بگیرند و یا از روشهای دیگر موجود دیگر برای گردآوری داده های مورد نیاز استفاده شود.

در ای قسمت از پرس وجویی که در بخش امتحان کنید قبل ایجاد کرده ایم استفاده خواهیم کرد. بنابراین در لیست نمایش دهنده ی اشیای موجود در بانک اطلاعاتی روی علامت مثبت کنار `Views` کلیک کرده و سپس از لیست باز شده همانند شکل ۱۵-۱۰ گزینه ی `CustomerQuery` را انتخاب کنید. اگر روی علامت مثبت کنار `CustomerQuery` کلیک کنید لیست تمام فیلد هایی که به وسیله ی این پرس وجو برگشته می شود نمایش داده خواهند شد. بعد از مشاهده ی این صفحه روی دکمه ی `Finish` کلیک کنید تا کار در این قسمت به اتمام برسد.

در این لحظه، ویزارد یک شیئ از نوع `DataSet` به نام `northwindDataSet`، یک شیئ از نوع `BindingSource` به نام `customerQueryBindingSource` و نیز یک شیئ از نوع `TableAdapter` به نام `customerQueryTableAdapter` ایجاد می کند.



شکل ۱۵-۱۱

۱۰) در فرم اصلی برنامه روی مثلث کوچک کنار کنترل `DataGridView` کلیک کرده تا کادر `DataGridView Tasks` نمایش داده شود. به علت اینکه در این قسمت نمی خواهیم داده های موجود را حذف کرده، اضافه کنیم و یا تغییر دهیم، با کلیک روی گزینه های `Enable Adding`، `Enable Editing` و `Enable Deleting` علامت تیک کنار آنها را حذف کنید. اما می خواهیم که بتوانیم داده

ها را بر اساس ستونهای مورد نظر مرتب کنیم. پس روی گزینه ی Enable Column Reordering کلیک کنید تا انتخاب شود. سپس در قسمتی از نوار عنوان فرم برنامه کلیک کنید تا این پنجره محو شود.

۱۱) روی کنترل DataGridView کلیک کرده و سپس با استفاده از پنجره ی Properties خاصیت Dock آن را به Fill تغییر دهید.

۱۲) حال برنامه را اجرا کنید. مشاهده خواهید کرد که کنترل DataGrid به وسیله ی اطلاعات موجود در بانک اطلاعاتی پر خواهد شد.

با کلیک روی نام هر یک از ستونهای موجود در جدول، می توانید اطلاعات را بر اساس آن ستون به صورت صعودی مرتب کنید. کلیک مجدد روی هر ستون باعث می شود که اطلاعات بر اساس آن ستون به صورت نزولی مرتب شوند. برای تشخیص نحوه ی مرتب شدن اطلاعات نیز می توانید از جهت مثلث کوچکی که در کنار نام ستون نمایش داده می شود استفاده کنید.

همانطور که مشاهده می کنید در این قسمت توانستید بدون اینکه حتی یک خط کد در برنامه وارد کنید داده هایی را از یک بانک اطلاعاتی بدست آورده و آنها را نمایش دهید. تمام کد های مورد نیاز برای این موارد به صورت اتوماتیک توسط این ویزارد نوشته شده اند[1]. این مورد ثابت می کند که ویزاردها در ویژوال استودیو تا چه اندازه قدرتمند عمل می کنند.

چگونه کار می کند؟

شیوه ای که در این بخش امتحان کنید، برای ایجاد یک برنامه که بتواند اطلاعات موجود در یک بانک اطلاعاتی را نمایش دهد استفاده کردیم بسیار ساده و آسان بود. برنامه را با اضافه کردن یک کنترل DataGridView به فرم شروع کردیم، و همین مورد نیز باعث شد که کادر Tasks مربوط به کنترل DataGridView نمایش داده شود.

در این کادر می توانیم با استفاده از ویزارد Data Source Configuration Wizard و طی کردن یک سری مراحل ساده، یک Data Source جدید ایجاد کنیم. در این ویزارد ابتدا باید نوع منبع اطلاعاتی که می خواهیم به آن متصل شویم را مشخص کنیم. به این ترتیب اشیای موجود در بانک اطلاعاتی که مشخص کرده ایم نمایش داده خواهند شد و می توانیم آن اشیایی که به اطلاعات آن در برنامه نیاز داریم را انتخاب کنیم.

هنگامی که در این ویزارد روی کلید Finish کلیک می کنیم، چندین کامپوننت به صورت اتوماتیک ایجاد شده و به برنامه اضافه می شوند. این کامپوننت ها شامل TableAdapter، DataSet و نیز BindingSource می باشند. از بین این کامپوننت ها فقط کامپوننت BindingSource است که با نسبت داده شدن به خاصیت DataSource در کنترل DataGridView، باعث برقراری ارتباط بین کنترل DataGridView و داده های موجود در برنامه می شود.

به خاطر داشته باشید که روش کارکرد این کنترل ها با یکدیگر به این صورت است که هنگام نمایش فرم برنامه، ابتدا داده های مورد نیاز به وسیله ی کامپوننت TableAdapter از بانک اطلاعاتی دریافت شده و در اختیار کامپوننت DataSet قرار می گیرد. سپس تمام کنترل های موجود در فرم که به این داده ها نیاز دارند می توانند با استفاده از کامپوننت BindingSource به این اطلاعات دسترسی داشته باشند، زیرا کامپوننت BindingSource تنها کامپوننتی است که می تواند اطلاعات درون DataSet را بدست آورد.

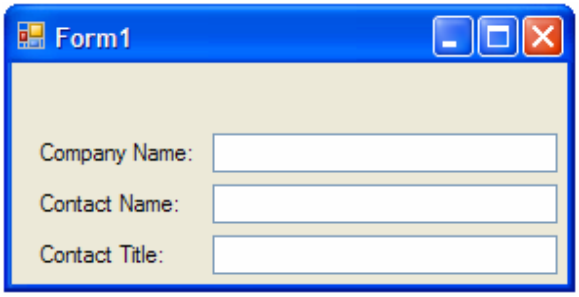
تنها هدفی که در این تمرین دنبال می کردیم این بود که متوجه شوید ایجاد یک برنامه با قابلیت دسترسی به داده های یک بانک اطلاعاتی تا چه اندازه ساده است و می توان حتی بدون وارد کردن یک خط کد در برنامه این کار را انجام داد. فقط لازم است که نوع

[1] در این قسمت، ویزاردی که استفاده کردیم در حدود ۹۰۰ خط کد را به برنامه اضافه کرد.

و مکان داده های خود را برای ویژوال استودیو مشخص کنید و سپس نحوه ی نمایش آنها را نیز تعیین کنید، بقیه ی کارها به صورت اتوماتیک انجام خواهند شد.
در بخش امتحان کنید بعد، برنامه ی جدیدی ایجاد خواهیم کرد که داده های مورد نیاز را به وسیله ی چند کنترل TextBox نمایش دهد. در این برنامه هر کنترل TextBox را به یکی از فیلد های جدول مورد نظرمان در بانک اطلاعاتی نسبت می دهیم، سپس با استفاده از کنترل BindingNavigator به کاربر اجازه می دهیم تا بین داده های موجود در برنامه حرکت کند.

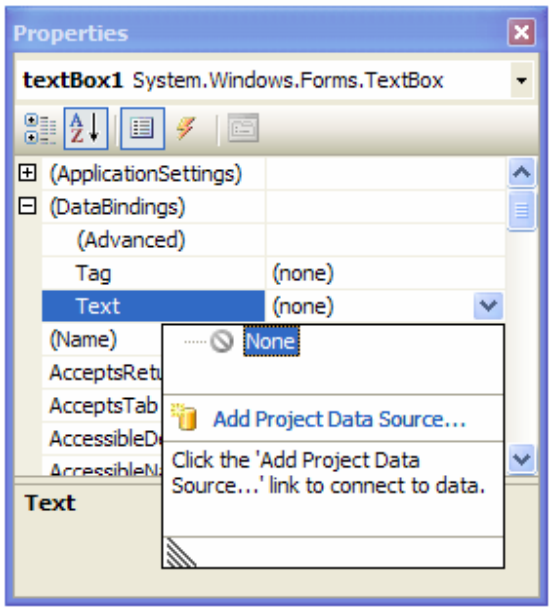
امتحان کنید: متصل کردن داده ها به کنترل TextBox

۱) با استفاده از ویژوال استودیو یک برنامه ی ویندوزی جدید به نام Northwind Customers ایجاد کنید.
۲) با استفاده از جعبه ابزار سه کنترل Label و سه کنترل TextBox به برنامه ی خود اضافه کنید. سپس خاصیت ها و مکان این کنترل ها را به گونه ای تغییر داده و تنظیم کنید که فرم برنامه مشابه شکل ۱۵-۱۲ شود.



شکل ۱۵-۱۲

۳) در فرم برنامه روی TextBox اول کلیک کنید تا انتخاب شود. سپس با استفاده از پنجره ی Properties روی علامت مثبت کنار خاصیت (DataBindings) این کنترل کلیک کنید. در لیستی که نمایش داده می شود، خاصیت Text را انتخاب کرده و روی علامت مثلث کوچک مقابل آن کلیک کنید.
به این ترتیب پنجره ی Data Source همانند شکل ۱۵-۱۳ نمایش داده می شود در این پنجره روی لینک Add Project Data Source... کلیک کنید تا ویزارد Data Source Configuration Wizard، همانند آنچه در بخش امتحان کنید قبل مشاهده کرده بودید نمایش داده شود.



شکل ۱۵-۱۳

۴) در پنجره ی Choose a Data Source Type آیکون Database را انتخاب کرده و روی کلید Next کلیک کنید.

۵) در پنجره ی Choose Your Data Connection روی دکمه ی New Connection کلیک کنید تا کادر Add Connection نمایش داده شود.

۶) در کادر Add Connection روی دکمه ی Browse کلیک کرده و سپس به فولدر Samples در مکان نصب برنامه ی Office بروید. این فولدر به صورت پیش فرض برای Office 2003 در آدرس C:\Program Files\Microsoft Office\Office11\Samples قرار دارد. در این آدرس فایل Northwind.mdb را انتخاب کرده و سپس روی دکمه ی OK کلیک کنید تا نام و مسیر فایل انتخابی به کادر متنی موجود در پنجره ی Add Connection اضافه شوند. سپس در این پنجره نیز روی دکمه ی OK کلیک کنید تا کادر Add Connection بسته شود و به پنجره ی Choose Your Data Connection برگردید. در این پنجره نیز روی دکمه ی Next کلیک کنید.

به این ترتیب کادر پیغامی نمایش داده خواهد شد و به از شما می پرسد که فایل بانک اطلاعاتی که انتخاب کرده اید جزئی از پروژه نیست. آیا می خواهید این فایل به فولدر پروژه کپی شده و از نسخه ی کپی آن استفاده شود؟ در این کادر نیز روی دکمه ی Yes کلیک کنید.

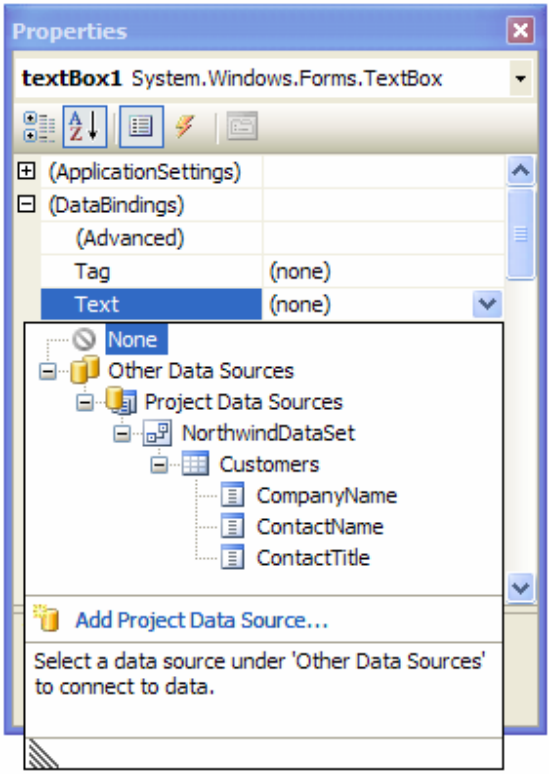
۷) در پنجره ی Save the Connection String to the Application Configuration File نیز روی دکمه ی Next کلیک کنید.

۸) در پنجره ی Choose Your Database Objects، روی علامت مثبت که در سمت چپ Tables قرار دارد، در لیست Database Objects کلیک کنید. سپس در لیستی که برای Tables نمایش داده می شود روی گزینه ی Customers کلیک کرده تا فیلد های این جدول نیز نمایش داده شوند. در انتها نیز با کلیک کردن روی فیلد های CompanyName، ContactName و ContactTitle آنها را انتخاب کرده و سپس روی کلید Finish کلیک کنید.

۹) مجدداً در پنجره ی Properties روی علامت مثلث کوچک که در مقابل خاصیت Text قرار دارد کلیک کنید. این بار پنجره ی Data Source همانند شکل ۱۵-۱۴ نمایش داده می شود. به ترتیب روی علامت مثبت کنار گزینه های Other Data Sources، Project Data Sources، NorthwindDataSet و در آخر نیز Customers کلیک کنید.

حال روی فیلد CompanyName در این قسمت کلیک کنید. به این ترتیب کادر بسته شده و خاصیت Text در این TextBox به فیلد CompanyName در کامپوننت DataSet متصل خواهد شد.

در این برنامه نیز اگر به پایین محیط ویژوال استودیو نگاه کنید مشاهده خواهید کرد که سه کامپوننت مورد نیاز برای دسترسی به داده ها به صورت اتوماتیک به برنامه اضافه شده اند.



شکل ۱۵-۱۴

۱۰) کنترل TextBox دوم را از فرم برنامه انتخاب کرده و سپس با استفاده از قسمت (DataBindings) در پنجره ی Properties روی خاصیت Text کلیک کنید و مشابه قسمت قبل، فیلد ContactName را به این کادر اختصاص دهید.

۱۱) مراحل قبل را برای TextBox سوم نیز انتخاب کرده و فیلد سوم، یعنی فیلد ContactTitle را نیز به این کنترل متصل کنید.

۱۲) حال در جعبه ابزار به قسمت Data بروید و روی کنترل BindingNavigator دو بار کلیک کرده تا یک نمونه از آن در فرم قرار بگیرد. این کنترل به صورت اتوماتیک به بالای فرم برنامه متصل خواهد شد.

۱۳) با استفاده از پنجره ی Properties خاصیت BindingSource کنترل BindingNavigator را برابر با customersBindingSource قرار دهید.

۱۴) حال برنامه را اجرا کنید. مشاهده خواهید کرد که فرم برنامه همانند شکل ۱۵–۱۵ نمایش داده خواهد شد. به وسیله ی این فرم می توانید بین رکورد های موجود در بانک اطلاعاتی جا به جا شوید. برای مثال به رکورد بعدی و یا رکورد قبلی بروید. همچنین کلید هایی نیز وجود دارند که به وسیله ی آنها می توانید به اولین و یا آخرین رکورد موجود منتقل شوید.
با کلیک کردن روی دکمه ی `Delete` یکی از رکورد های موجود در `DataSet` حذف خواهد شد، اما توجه کنید که این رکورد از `DataSet` حذف می شود نه از بانک اطلاعاتی. همچنین کلیک کردن روی دکمه ی `New` نیز باعث می شود که یک رکورد جدید ایجاد شود. این رکورد نیز در `DataSet` ایجاد خواهد شد نه در بانک اطلاعاتی. برای اینکه تغییرات به وجود آمده در این قسمت به بانک اطلاعاتی اعمال شوند لازم است مقداری کد در برنامه وارد کنید.
همانطور که مشاهده می کنید در طراحی این برنامه نیز برای دسترسی به داده های موجود در بانک اطلاعاتی حتی به وارد کردن یک خط کد هم نیاز نبود.



شکل ۱۵–۱۵

چگونه کار می کند؟

این قسمت را با اضافه کردن سه کنترل `TextBox` و نیز سه `Label` به فرم برنامه آغاز کردیم. سپس سعی کردیم که خاصیت `DataBindings` کنترلهای `TextBox` را تنظیم کنیم. در حقیقت با این کار می خواستیم که خاصیت `Text` این کنترل ها به فیلد های مشخصی از داده های موجود در برنامه متصل شوند. اما ابتدا باید یک منبع اطلاعاتی را برای برنامه ایجاد می کردیم. این کار را با استفاده از ویزارد `Data Source Configuration Wizard` همانند تمرین قبل انجام دادیم.
بعد از اتمام ویزارد، سه کامپوننت مورد نیاز یعنی `TableAdapter`، `BindingSource` و نیز `DataSet` به صورت اتوماتیک ایجاد شده و به برنامه اضافه شدند. سپس توانستیم با استفاده از این کنترل ها خاصیت `Text` هر یک از `TextBox` ها را به یکی از فیلد های موجود در `DataSet` متصل کنیم.
بعد از اضافه کردن کنترل `BindingNavigator`، برای تنظیم آن کافی بود که با استفاده از پنجره ی `Properties` خاصیت `BindingSource` آن را برابر با کامپوننت `BindingSource` ای قرار دهیم که در مرحله ی قبل به صورت اتوماتیک ایجاد شده بود.

نتیجه:

فصل را با تعریف مفهوم بانک اطلاعاتی شروع کردیم و آموختیم که بانک های اطلاعاتی به چه مواردی گفته می شود. سپس به آشنایی با زبان SQL و معرفی یکی از مهمترین دستورات آن یعنی دستور SELECT پرداختیم. بعد از اتمام این قسمت نیز با استفاده از این موارد، یک پرس وجو در بانک اطلاعاتی Northwind.mdb ایجاد کرده و مشاهده کردیم که Access چگونه به صورت اتوماتیک دستورات SQL مورد نیاز را تولید کرده و اجرا می کند.
سپس به بررسی نحوه ی استفاده از داده های موجود در یک بانک اطلاعاتی در یک برنامه ی ویندوزی پرداختیم و مشاهده کردیم که چگونه می توان اطلاعات را به کنترل هایی مانند DataGridView و یا TextBox متصل کرد. با تعدادی از کامپوننت های مهم و ضروری برای دسترسی به داده های موجود در بانکهای اطلاعاتی Access آشنا شدیم و مشاهده کردیم که چگونه می توان آنها را با استفاده از جعبه ابزار بر روی فرم قرار داد و یا با استفاده از ویزاردها آنها را به صورت اتوماتیک ایجاد کرده و به برنامه اضافه کرد.
در پایان این فصل باید با موارد زیر آشنا شده باشید:

- توضیح دهید که بانک اطلاعاتی چیست و از چه اشیایی تشکیل شده است؟
- با استفاده از دستور SELECT در زبان SQL داده های مورد نیاز خود را از یک بانک اطلاعاتی بدست آورید.
- با استفاده از ویزارد Data Source Configuration Wizard کامپوننت های لازم برای متصل شدن به یک بانک اطلاعاتی را ایجاد کنید.
- یک کنترل DataGridView را به داده های موجود در برنامه متصل کنید.
- کنترلهای TextBox را به فیلد های مشخصی در DataSet متصل کرده و از کنترل BindingNavigator استفاده کنید.

در این فصل مشاهده کردید که چگونه می توان با استفاده از ویزاردهای موجود در ویژوال استودیو ۲۰۰۵ داده های موجود در یک بانک اطلاعاتی را به سرعت به کنترلهای موجود در فرم متصل کرد. اما در شرایطی ممکن است نیاز به کنترل بیشتر روی نحوه ی ارتباط با بانک اطلاعاتی و یا نحوه ی متصل شدن کنترل ها به داده ها داشته باشید. در فصل شانزدهم سعی خواهیم کرد که خط مشی متفاوتی را نسبت به این فصل پیش گرفته و سعی کنیم با استفاده از برنامه نویسی، کنترلهای یک فرم را به داده های موجود در یک بانک اطلاعاتی متصل کنیم. همچنین با کامپوننت های دسترسی به داده ها بهتر آشنا خواهیم شد و مشاهده خواهیم کرد که چگونه می توان از خاصیت ها و یا متدهای آنها در برنامه استفاده کرد.

تمرین:

تمرین ۱:

پرس وجوی جدیدی را در بانک اطلاعاتی Northwind ایجاد کنید که بتواند اطلاعات درون فیلد های FirstName، LastName و Title را از بانک اطلاعاتی Employees بدست آورد. همچنین دستور را به گونه ای بنویسید که نتایج را بر اساس ستون LastName مرتب کند، سپس آن را به نام EmployeeQuery ذخیره کنید. برنامه ی ویندوزی ایجاد کنید که داده های حاصل از اجرای این پرس وجو را در یک کنترل DataGridView نمایش دهد.

تمرین ۲:

با استفاده از پرس وجویی که در تمرین قبل ایجاد کردید، برنامه ی ویندوزی جدیدی ایجاد کنید که بتواند با استفاده از چندین TextBox و نیز یک کنترل BindingNavigator داده های حاصل از اجرای پرس وجو را به کنترلهای TextBox متصل کند.

فصل شانزدهم: برنامه نویسی بانک اطلاعاتی با SQL Server و ADO.NET

در فصل پانزدهم با بانکهای اطلاعاتی و برنامه نویسی بانک اطلاعاتی به صورت مقدماتی آشنا شدیم. توانستیم اطلاعات داخل یک جدول از بانک اطلاعاتی را در یک برنامه ی تحت ویندوز بدست آورده و آن را در جدولی در فرم نمایش دهیم. همچنین بدون اینکه لازم باشید کدی وارد کنیم، توانستیم امکاناتی مانند مرتب کردن داده ها را نیز به برنامه اضافه کنیم.
برای انجام تمام این موارد از یک ویزارد استفاده کردیم و آن ویزارد نیز کد های زیادی را برای انجام موارد مورد نظر ما نوشت، کد هایی مانند ایجاد یک اتصال به بانک اطلاعاتی، تنظیم کردن آداپتور های داده ای و نیز ایجاد یک `DataSet` مخصوص برای جداول مورد نظر ما. استفاده از این روش برای دسترسی ساده به بانک اطلاعاتی و انجام کارهای معمولی مانند دریافت و مشاهده ی اطلاعات از یک یا چند جدول روش مناسبی است، اما برای نوشتن برنامه های بزرگتر لازم است که کنترل بیشتری بر داده ها و یا کامپوننت های موجود در برنامه داشته باشیم و این کار نیز فقط از طریق کد نویسی میسر است.
در این فصل سعی خواهیم کرد که نگاه عمیقتری به مبحث دسترسی به بانک اطلاعاتی داشته باشیم. تکنولوژیهایی که در فصل قبل برای دسترسی به داده ها و یا تغییر در آنها استفاده کردیم، از قبیل کامپوننت هایی برای دریافت اطلاعات از بانک اطلاعاتی، کامپوننت هایی برای ذخیره ی آنها در حافظه و نیز کامپوننت هایی برای متصل کردن این داده ها به کنترلهای موجود در فرم، همه مجموعاً به نام **`ADO.NET`** شناخته می شوند. در این فصل سعی خواهیم کرد با توانایی ها و قابلیتهای درونی `ADO.NET` برای دسترسی به داده های درون یک بانک اطلاعاتی و نیز ایجاد تغییرات در آنها آشنا شویم. همچنین مشاهده خواهیم کرد که چگونه می توان داده هایی که به وسیله ی یک `DataSet` درون حافظه ذخیره شده است را تغییر دهیم، فیلتر کرده و یا ویرایش کنیم.
داده هایی که از یک بانک اطلاعاتی استخراج می شوند، برای نمایش داده شدن باید به یکی از کنترل های موجود در فرم متصل شوند. بنابراین لازم است که اتصال داده ها به کنترل ها را نیز دقیق تر بررسی کنیم. به عبارت دیگر در این فصل مشاهده خواهیم کرد که چگونه می توان کنترلهای موجود در فرم را به گونه ای تنظیم کرد که در هر لحظه فقط داده های مربوط به یک رکورد را نمایش دهند (برای مثال، مانند `TextBox`ها) و یا چگونه می توان با استفاده از اشیایی مانند `CurrencyManager` بین رکورد ها حرکت کرد.
در این فصل:

- خواهیم آموخت که اشیای `ADO.NET` چیستند؟
- مشاهده خواهیم کرد که چگونه می توان کنترل ها را به داده ها متصل کرد.
- روشهای جستجو و یا مرتب سازی داده های داده های درون حافظه را با استفاده از اشیای `Data View` در `ADO.NET` بررسی خواهیم کرد.
- با نحوه ی انتخاب ، درج، ویرایش و یا حذف داده ها درون یک بانک اطلاعاتی به وسیله ی `ADO.NET` آشنا خواهیم شد.

در طی این فصل با نحوه ی استفاده از بانکهای اطلاعاتی ایجاد شده به وسیله ی موتور بانک اطلاعاتی `SQL Server` نیز آشنا خواهیم شد و خواهیم دید که چگونه می توان به وسیله ی سرویس دهنده ی اطلاعاتی `SqlClient` به آنها دسترسی پیدا کرد. سرویس دهنده ی اطلاعاتی `SqlClient`، نسبت به سرویس دهنده ی اطلاعاتی `OleDb` (که برای کار با بانک اطلاعاتی ایجاد شده با `Access` استفاده می شود) از سرعت بیشتری برخوردار است، اما فقط می تواند با بانکهای اطلاعاتی تحت `SQL Server` کار کند.

برای انجام تمرینات این فصل لازم است که به یکی از نرم افزارهای MSDE، SQL Server 7، SQL Server 2000 و یا SQL Server 2005 دسترسی داشته باشید. زیرا در برنامه های این فصل از بانک اطلاعاتی نمونه ای که در این نرم افزارها وجود دارد (بانک اطلاعاتی Pubs) استفاده شده است.

ADO.NET

همانطور که در ابتدای فصل نیز ذکر شد، به مجموعه کامپوننت هایی که برای دسترسی به داده های یک بانک اطلاعاتی در .NET استفاده می شود **ADO.NET**[1] گفته می شود. ADO.NET برای دسترسی به داده ها از **معماری غیر متصل**[2] استفاده می کند. معماری غیر متصل به این معنی است که ابتدا برنامه به موتور بانک اطلاعاتی مورد نظر خود متصل شده، داده های مورد نیاز خود را از بانک اطلاعاتی دریافت کرده و آنها را در حافظه ی کامپیوتر ذخیره می کند. سپس برنامه از بانک اطلاعاتی قطع می شود و تغییرات مورد نظر خود را در داده های موجود در حافظه انجام می دهد. هر زمان که لازم باشد تغییرات ایجاد شده در بانک اطلاعاتی ذخیره شوند، برنامه یک اتصال جدید را به بانک اطلاعاتی ایجاد کرده و از طریق این اتصال تغییراتی را که در داده ها اعمال کرده بود را در جداول اصلی نیز ایجاد می کند. کامپوننت اصلی که داده های دریافتی از بانک اطلاعاتی را در حافظه نگهداری می کند، کامپوننت DataSet است. این کامپوننت خود از چند کامپوننت دیگر مانند اشیایی از نوع DataTable تشکیل شده است. بعد از اینکه داده ها در حافظه قرار گرفتند می توانید بین آنها جستجو کنید، دستورات SELECT مورد نظر خود را روی آنها اجرا کرده و آنها را به این وسیله فیلتر کنید و یا تغییراتی را در این داده ها ایجاد کنید که در طی این فصل با نحوه ی انجام این موارد آشنا خواهیم شد.

استفاده از معماری غیر متصل مزایای زیادی دارد که مهمترین آن افزایش توانایی برنامه در سرویس دادن به چندین کاربر به صورت همزمان است. به عبارت دیگر می توانیم بگوییم با استفاده از این روش می توانیم تعداد افرادی که می توانند به صورت همزمان از برنامه استفاده کنند را از ده ها نفر به صدها نفر افزایش دهیم. دلیل این مورد نیز در این است که در این روش برنامه ها فقط در مواقع مورد نیاز به بانک اطلاعاتی متصل می شوند و بعد از اجرای وظایف لازم اتصال خود را قطع می کنند، به این ترتیب منابع استفاده شده برای اتصال آنها به بانک اطلاعاتی نیز آزاد شده و در اختیار کاربران دیگر قرار می گیرد.

فضای نام Data:

کلاسهای اصلی ADO.NET در فضای نام System.Data قرار دارند. این فضای نام خود نیز شامل چند فضای نام دیگر است که مهمترین آنها عبارتند از System.Data.OleDb و System.Data.SqlClient. فضای نام System.Data.SqlClient شامل کلاس هایی است که برای دسترسی به بانک های اطلاعاتی ایجاد شده به وسیله ی SQL Server به کار می رود. فضای نام System.Data.OleDb نیز حاوی کلاس هایی است که برای دسترسی به بانک های اطلاعاتی از نوع OLE[3] (مانند بانک های اطلاعاتی Access) مورد استفاده قرار می گیرد. برای مثال در فصل قبل برای اتصال به بانک اطلاعاتی از بعضی از کلاسهای فضای نام +++

در فضای نام System.Data دو فضای نام دیگر نیز وجود دارند که عبارتند از System.Data.OracleClient و System.Data.Odbc. فضای نام OracleClient برای دسترسی به بانک های اطلاعاتی ایجاد شده به وسیله ی موتور بانک اطلاعاتی Oracle مورد استفاده قرار می گیرد.

---

[1] ActiveX Data Object
[2] Disconnected Architecture
[3] Object Linking and Embedding

کلاسهای موجود در این فضای نام نیز، همانند کلاسهای موجود در فضای نام SqlClient برای دسترسی به بانک های اطلاعاتی از نوع Oracle بهینه سازی شده اند. فضای نام Odbc نیز حاوی کلاس هایی است که برای دسترسی به بانک های اطلاعاتی قدیمی از نوع ODBC[1] که تکنولوژی OleDB را پشتیبانی نمی کنند ایجاد شده است.
فضای نامهای SqlClient، OleDb، OracleClient و نیز Odbc در ADO.NET به عنوان سرویس دهنده های اطلاعاتی شناخته می شوند. در .NET سرویس دهنده های اطلاعاتی دیگری نیز وجود دارند، اما در طی این کتاب فقط بر روی دو سرویس دهنده ی اول تمرکز خواهیم کرد.
در طی این فصل با استفاده از فضای نام SqlClient به بانکهای اطلاعاتی از نوع SQL Server دسترسی خواهیم داشت. البته، در ADO.NET استفاده از دیگر سرویس دهنده های اطلاعاتی نیز بسیار مشابه استفاده از این سرویس دهنده است. بنابراین به راحتی می توانید از تکنیک هایی که با استفاده از کلاسهای موجود در این فضای نام خواهید آموخت در سرویس دهنده های دیگر نیز از قبیل OleDB استفاده کنید و یا تکنیکهای سرویس دهنده هایی مانند OleDB را در این قسمت نیز به کار ببرید. در ADO.NET بر اساس نوع موتور بانک اطلاعاتی که داده های شما به وسیله ی آن ایجاد شده اند، یکی از سرویس دهنده های موجود را انتخاب کرده و از آن استفاده می کنید. اما لازم نیست که مجدداً نحوه ی استفاده از آن سرویس دهنده را مطالعه کنید زیرا تمامی این سرویس دهنده ها بسیار مشابه یکدیگر کار می کنند و اگر نحوه ی استفاده از یکی از آنها را بیاموزید می توانید به راحتی از دیگر سرویس دهنده ها نیز استفاده کنید.
کلاسهای موجود در این سرویس دهنده های بانک اطلاعاتی به حدی زیاد هستند که نمی توانیم تمام آنها را در این قسمت معرفی کنیم. با این وجود در این قسمت ابتدا با تعدادی از مهمترین آنها که در طی مثال های این فصل نیز به کار رفته اند آشنا می شویم. این کلاسها عبارتند از:

- SqlConnection
- SqlCommand
- SqlDataAdapter
- SqlParameter

به خاطر داشته باشید که این کلاسها فقط برای ارتباط با بانکهای اطلاعاتی SQL Server مورد استفاده قرار می گیرند. برای استفاده از بانک های اطلاعاتی OLEDB می توانید از کلاسهای متناظر اینها در فضای نام System.Data.OleDb استفاده کنید. در این فضای نام نیز همین کلاسها وجود دارند که البته با پیشوند OleDb آغاز می شوند.
همچنین همانطور که می دانید برای استفاده از این کلاسها باید فضای نام System.Data.SqlClient را با استفاده از راهنمای using به برنامه اضافه کرد تا لازم نباشد هر بار نام کامل آنها را وارد کنیم. بنابراین به خاطر داشته باشید که در ابتدای برنامه های این قسمت دستور زیر را نیز اضافه کنید:

```
using System.Data.SqlClient;
```

همچنین برای استفاده از کلاسهای پایه ای ADO.NET مانند DataSet و یا DataView باید فضای نام System.Data را نیز به برنامه اضافه کنیم. بنابراین در ابتدای برنامه های خود دستور زیر را نیز وارد کنید.

```
using System.Data;
```

[1] Open Database Connectivity

خوب بهتر است که ابتدا نگاهی به کلاسهای اصلی موجود در فضای نام SqlClient داشته باشیم و نحوه ی کـاربرد آنهـا را بررسی کنیم.

کلاس SqlConnection:

تقریباً می توانیم بگوییم که کلاس SqlConnection در قلب کلاس هایی قرار دارد که در این قسمت مورد استفاده قـرار می دهیم، زیرا این کلاس وظیفه ی برقراری ارتباط بین برنامه و بانک اطلاعاتی را بر عهده دارد. هنگامی که بخواهید یک نمونـه از این کلاس را ایجاد کنید، باید پارامتری را به نام ConnectionString به آن ارسال کنید. این پـارامتر متغیـری از نـوع رشته ای است که شامل تمام داده های مورد نیاز برای برقراری اتصال به یک بانک اطلاعاتی می شود. البته بعد از ایجاد شیئ ای از این کلاس نیز می توانم با استفاده از خاصیت ConnectionString در این کلاس، مقدار آن را تغییـر داده و رشـته ی جدیدی را برای این پارامتر مشخص کنیم. در برنامه هایی که در فصل قبل ایجاد کردیم، ویژوال استودیو با استفاده از اطلاعاتی کـه در کادر Add Connection دریافت می کرد، چنین متنی را ایجاد کرده و در اختیار SqlConnection قرار می داد. اما اغلب بهتر است که متن لازم برای ConnectionString را خودمان بنویسیم. برای این کار ابتدا باید بدانیم کـه این ساختار این متنها باید چگونه باشد.

ایجاد بخشهای مختلف ConnectionString:

ساختار متنی که برای ConnectionString باید مورد استفاده قرار گیرد بستگی به سرویس دهنـده ی اطلاعـاتی دارد که مورد استفاده قرار می دهیم. برای مثال اگر بخواهیم از SQL Server به عنوان موتـور بانـک اطلاعـاتی برنامـه ی خـود استفاده کنیم (به این ترتیب لازم است که از سـرویس دهنـده ی SqlClient در برنامـه اسـتفاده کنـیم)، بایـد پارامترهـای Server و Database . مقدار آنها را همانطور که در جدول زیر نمایش داده شده است در این متن مشخص کنیم.

پارامتر	توضیح
Server	نام سرور بانک اطلاعاتی که می خواهید از آن استفاده کنید. این پارامتر معمولا حاوی نـام کـامپیوتری اسـت کـه موتـور بانـک اطلاعـاتی SQL Server در آن نـصب شـده اسـت. اگـر SQL Server بر روی همان کامپیوتری که برنامه را اجـرا مـی کنـد نـصب شـده اسـت، مـی توانیـد از مقادیری مانند local و یا localhost برای این پارامتر استفاده کنید. امـا اگـر از SQL Server ای که در کامپیوتر دیگری در شبکه نصب شده است استفاده مـی کنیـد، لازم اسـت کـه مقدار این پارامتر را برابر با نام آن کامپیوتر در شبکه قرار دهید. همچنین اگـر در آن کـامپیوتر بـیش از یک SQL Server قرار داشته باشد، باید بعد از نام کامپیوتر یک علامت \ قـرار داده و سـپس نام SQL Server ای که می خواهید مورد استفاده قرار دهید را ذکر کنید.
Database	نام بانک اطلاعاتی که می خواهید مورد استفاده قرار دهید، در این پارامتر قرار می گیـرد (بـرای مثـال، بانک اطلاعاتی Pubs).

برای ایجاد امنیت در بانکهای اطلاعاتی ایجاد شده به وسیله ی SQL Server، باید هنگام دسترسی به آنها ابتدا هویت استفاده کننده توسط SQL Server مشخص شود. بنابراین اگر بخواهیم توسط یک برنامه به داده های موجود در یک بانک اطلاعاتی دسترسی داشته باشیم، باید اطلاعات لازم برای این تعیین هویت را همراه با دیگر اطلاعات در متن ConnectionString مشخص کنیم. این تعیین هویت به دو روش می تواند توسط SQL Server انجام شود. روش اول استفاده از نام کاربری و کلمه ی عبور لازم برای دسترسی به بانک اطلاعاتی است که در این صورت باید این دو پارامتر را به صورت مستقیم در متن ConnectionString قرار دهیم. روش دوم استفاده از اکانت کاربری ای است که در ویندوز از آن استفاده می کنیم.
برای اینکه بتوانیم با استفاده از روش اول توسط SQL Server تعیین هویت شویم، باید پارامترهای نام کاربری و کلمه ی عبور را همانطور که در جدول زیر شرح داده شده است، در متن ConnectionString قرار دهیم.

پارامتر	توضیح
User ID	این پارامتر باید حاوی نام کاربری باشد که برای اتصال به بانک اطلاعاتی می خواهیم از آن استفاده کنیم. برای اینکه بتوانیم با استفاده از این روش از بانک اطلاعاتی استفاده کنیم، باید یک اکانت کاربری به این نام در SQL Server ایجاد شده و اجازه ی دسترسی به داده های مورد نیاز نیز به آن داده شود.
Password	کلمه ی عبوری که برای این نام کاربری مورد استفاده قرار می گیرد.

علاوه بر این SQL Server می تواند به گونه ای تنظیم شود که برای تعیین هویت کاربرانی که به آن دسترسی پیدا می کنند، از اکانت ویندوزی که با آن وارد کامپیوتر شده اند استفاده کند. در این صورت دیگر نیازی نیست که در متن ConnectionString مقادیر نام کاربری و کلمه ی عبور را وارد کنید، بلکه فقط باید مشخص کنید که از Integrated Security استفاده می کنید (این سیستم به این علت Integrated Security نامیده می شود که با استفاده از آن ویندوز و SQL Server با یکدیگر سعی خواهند که به حداکثر امنیت ممکن در ایجاد یک ارتباط دسترسی پیدا کنند و دیگر نیازی نباشد که نام کاربری و کلمه ی عبور را به صورت مستقیم در متن ConnectionString قرار دهیم). برای استفاده از این سیستم باید مقدار پارامتر IntegratedSecurity را در متن ConnectionString برابر با true قرار دهید.
البته توجه داشته باشید که در این روش نیز باید در محیط SQL Server به اکانت کاربری ای که برای این فرد در ویندوز ایجاد شده است، اجازه ی دسترسی به اطلاعات موجود داده شود، در غیر این صورت کاربر نمی تواند به بانک اطلاعاتی متصل شود.
برای اینکه بهتر متوجه شوید که چگونه پارامترهای لازم در ConnetionString تنظیم می شوند، به قطعه کد زیر نگاه کنید. این کد برای ایجاد یک شیئ جدید از نوع SqlConnection به کار می رود.

```
SqlConnection objConnection = new
     SqlConnection("Server=localhost;Database=Pubs;User" +
               " ID=sa;Password=csdotnet;");
```

همانطور که مشاهده می کنید این ConnectionString برای استفاده از یک SQL Server به کار می رود. مقدار localhost در پارامتر مشخص می کند که سرور SQLکه می خواهیم از آن استفاده کنیم در کامپیوتری قرار دارد که برنامه در آن اجرا شده است. مقدار پارامتر Database نیز نام بانک اطلاعاتی که می خواهیم به آن متصل شویم را تعیین می کند، در این جا ایم ConnectionString برای دسترسی به بانک اطلاعاتی Pubs به کار می رود. در آخر نیز

مقدار پارامتر های User ID و Password مشخص می شوند که برای تعیین نام کاربری و کلمه ی عبور لازم برای دسترسی به اطلاعات به کار می روند. دقت کنید که برای تعیین مقدار هر پارامتر از علامت = و برای جدا کردن پارامترهای مختلف از یکدیگر از علامت ; استفاده شده است.

متصل شدن و قطع کردن اتصال به یک بانک اطلاعاتی:

بعد از این که با ایجاد ConnectionString نحوه ی برقراری ارتباط با بانک اطلاعاتی را مشخص کردیم می توانیم با استفاده از متد های Open و Close در کلاس SqlConnection به بانک اطلاعاتی متصل شده و یا اتصال خود را قطع کنیم. یک نمونه از این کار در قطعه کد زیر نشان داده شده است:

```
// Open the database connection
objConnection.Open();
// ... Use the connection
objConnection.Close();
// Close the database connection
```

البته متد ها و خاصیتهای فراوان دیگری در کلاس SqlConnection وجود دارند که می توانیم در برنامه از آنها استفاده کنیم، اما مواردی که در این جا با آنها آشنا شدیم، پر کاربرد ترین آنها به شمار می روند و فکر می کنم که برای شروع فقط آشنایی با این موارد کافی باشد.

کلاس SqlCommand:

کلاس SqlCommand حاوی یک دستور SQL برای اجرا روی داده های دریافت شده از بانک اطلاعاتی است. این دستور می تواند یک دستور SELECT برای انتخاب داده هایی خاص، یک دستور INSERT برای درج داده های جدید در بانک اطلاعاتی، یک دستور DELETE برای حذف داده ها از بانک اطلاعات و یا حتی فراخوانی یک پروسیجر ذخیره شده در بانک اطلاعاتی باشد. دستور SQL ای که در این کلاس نگه داری می شود می تواند شامل پارامتر ها نیز باشد.
از متد سازنده ی کلاس SqlCommand چندین نسخه سربار گذاری شده است، اما ساده ترین آنها برای ایجاد یک شیئ از کلاس SqlCommand هیچ پارامتری را دریافت نمی کند. بنابراین می توانید بعد از ایجاد شیئ، با استفاده از خاصیت ها و یا متدهای موجود، آن شیئ را تنظیم کنید. قطعه کد زیر نحوه ی ایجاد یک شیئ از نوع SqlCommand را نمایش می دهد:

```
SqlCommand objCommand = new SqlCommand();
```

در برنامه های بانک اطلاعاتی معمولا از اشیای ایجاد شده از کلاس SqlCommand به تنهایی استفاده نمی کنند، بلکه آنها را همراه با DataSet ها و DataAdapter ها به کار می برند. به این ترتیب می توانند از دستور SELECT، INSERT و یا ... که در آن نگهداری می شود برای مقاصد مورد نیاز استفاده کنند. همچنین اشیای SqlCommand می توانند به همراه اشیای ایجاد شده از کلاس DataReader مورد استفاده قرار گیرند. کلاس DataReader کاربردی

همانند DataSet دارد، اما منابع سیستم (مانند حافظه و ...) را کمتر مصرف می کند و نیز انعطاف پـذیری کمتـری نیـز دارد. در ادامه ی این فصل بیشتر تمرکز خود را روی نحوه ی استفاده از این اشیا با کلاس DataSet قرار خواهیم داد.

خاصیت Connection:

قبل از اینکه بتوانی از یک شیئ از کلاس SqlCommand استفاده کنیم باید بعضی از خاصیت های آن را تنظیم کنـیم. اولـین خاصیتی که باید تنظیم شود، خاصیت Connection است. این خاصیت همانطور که در قطعه کد زیر نمایش داده شده اسـت، می تواند یک مقدار از نوع SqlConnection را دریافت کند:

```
objCommand.Connection = objConnection;
```

برای اینکه بتوانیم دستور SQL ای که در این شیئ نگهداری می شود را با موفقیت اجرا کرده و نتیجه ی آن را دریافت کنـیم، بایـد ابتدا با استفاده از متد Open در SqlConnection اتصال با بانک اطلاعاتی را برقرار کرده و سپس دستور را اجرا کنیم.

خاصیت CommandText:

خاصیت بعدی که باید تنظیم کنیم، خاصیت CommandText است. این خاصیت متنی را دریافت می کند که می تواند حاوی یک دستور SQL و یا فراخوانی یک پروسیجر ذخیره شده در بانک اطلاعاتی باشد که باید روی داده ها اجرا شود. برای مثـال قطعـه کد زیر یک نمونه از دستور SQL که در این خاصیت قرار داده شده است را نمایش می دهد:

```
SqlConnection objConnection = new
SqlConnection("Server=localhost;Database=Pubs;User " +
              "ID=sa;Password=csdotnet;");
SqlCommand objCommand = new SqlCommand();
objCommand.Connection = objConnection;
objCommand.CommandText = "INSERT INTO authors " +
          "(au_id, au_lname, au_fname, contract) " +
          "VALUES('123-45-6789', 'Barnes', 'David', 1)";
```

دستور INSERT یکی از دستورات ساده ی SQL است که برای درج یک رکورد از اطلاعات در یک جدول به کار مـی رود. ایـن دستور در این قسمت بیان می کند که "یک رکورد جدید از اطلاعات در جدول authors ایجاد کن. سپس فیلد au_id در این رکورد را برابر با '123-45-6789' قـرار بـده، فیلـد au_lname را برابـر بـا 'Barnes' قـرار بـده، فیلـد au_fname را برابر با 'David' قرار بده و فیلد contract را نیز برابر با 1 قرار بده".
روش استفاده از دستور INSERT برای درج یک ردیف از اطلاعـات در یـک جـدول بـه ایـن صـورت اسـت کـه بعـد از دسـتور INSERT INTO نام جدولی که می خواهیم اطلاعات در آن قرار بگیرد را ذکر می کنیم. سپس نام فیلد هایی را که باید کامل کنیم را در داخل یک پرانتز می آوریم و هر یک را نیز با یک ویرگول از هم جدا می کنیم. سـپس عبـارت VALUES نوشـته و در یک پرانتز دیگر، مقدار مورد نظر برای آن فیلد ها را به ترتیب وارد می کنیم.

در اینجا فرض کردیم که هنگام نوشتن برنامه، مقداری که باید در هر یک از فیلد ها قرار گیرد مشخص است. اما همانطور کـه مـی دانید در اغلب موارد چنین شرایطی رخ نمی دهد و مقدار هر یک از این فیلد ها در طول اجرای برنامه توسط کاربر تعیین مـی شـوند. خوشبختانه می توانیم دستورات SQL را به گونه ای ایجاد کنیم که همانند یک متد، پارامتر دریافت کنند. سپس هنگام اجرای برنامه این پارامتر ها را از کاربر دریافت کرده و آنها را در دستور قرار می دهـیم و دسـتور را اجـرا مـی کنـیم. بهتـر اسـت مقـداری هـم بـا پارامترهایی که می توانند در شیئ SqlCommand قرار گیرند آشنا شویم.

خاصیت Parameters:

قبل از اینکه بتوانیم نحوه ی استفاده از پارامتر ها در یک دستور SQL را مشاهده کنیم، باید با مفهـوم Placeholder هـا آشنا شویم. Placeholder ها متغیرهایی هستند که در یک دستور SQL قرار می گیرند و می توانند در زمان اجرای برنامه جای خود را عبارتی خاص عوض کنند. این متغیر ها با علامت @ در یک دستور مشخص می شوند و هنگـامی کـه از آنهـا در یـک دستور SQL استفاده کنیم، قبل از اجرای دستور باید تمامی آنها را با مقادیر مناسب تعویض کنیم. برای مثال اگر بخواهیم در دسـتور قبل مقادیر لازم برای قسمت VALUES از دستور INSERT را در زمان اجرای برنامه مشخص کنیم، باید جای آنها را با چهـار Placeholder به صورت زیر عوض کنیم:

```
SqlConnection objConnection = new
SqlConnection("Server=localhost;Database=Pubs;User " +
              "ID=sa;Password=csdotnet;");
SqlCommand objCommand = new SqlCommand();
objCommand.Connection = objConnection;
objCommand.CommandText = "INSERT INTO authors " +
     "(au_id, au_lname, au_fname, contract) " +
     "VALUES(@au_id, @au_lname, @au_fname, @au_contract)";
```

همانطور که مشاهده می کنید در اینجا به جای اینکه از چند مقدار ثابـت در دسـتور اسـتفاده کنـیم، از چنـد placeholder استفاده کرده ایم. همچنین تمام placeholder ها در دستور با استفاده از @ مشخص شده اند. البته هیچ ضرورتی ندارد که نام یک placeholder همنام با فیلدی باشد که قرار است مقدار placeholder در آن قرار بگیـرد. امـا ایـن کـار باعث می شود که برنامه خواناتر شده و درک آن ساده تر شود.

خـوب، بعـد از اینکـه بـا اسـتفاده از ایـن روش پارامترهـایی را در دسـتور ایجـاد کـردیم، بایـد قبـل از اجـرای دسـتور SQL ایـن placeholder ها را با مقادیر مناسب تعویض کنیم. این کار به صورت اتوماتیک توسط برنامه در زمان اجرای دستور انجـام مـی شـود. امـا ابتـدا بایـد پارامترهـایی را ایجـاد کـرده و آن را در لیـست Parameters در شـیئ ایجـاد شـده از کـلاس SqlCommand قرار دهیم تا برنامه بداند هنگام اجرای دستور هـر placeholder را بایـد بـا مقـدار چـه متغیـری در برنامه عوض کند. توجه کنید ه اصطلاح پارامتر در این قسمت به پارامترهایی اشاره می کند که برای اجرای یک دسـتور SQL و یـا یک پروسیجر ذخیره شده در بانک اطلاعاتی لازم است، نه به پارامترهایی که در ویژوال C# به متد ها فرستاده می شود.

بـرای دسترسـی بـه لیـست پارامترهـایی کـه در یـک شـیئ از کـلاس SqlCommand وجـود دارد مـی تـوانیم از خاصـیت Parameters در این کلاس استفاده کنیم.این خاصیت حاوی لیـستی از placeholder هـا بـه همـراه متغیرهـای وابـسته بـه آنهـا اسـت. بنـابراین در برنامـه قبـل از اجـرای دسـتور، بایـد بـه وسـیله ی ایـن لیـست مـشخص کنـیم کـه هـر placeholder با مقدار چه متغیری باید تعویض شود. ساده ترین روش انجام این کار در کد زیر نشان داده شده است.

```
SqlConnection objConnection = new
SqlConnection("Server=localhost;Database=Pubs;User
ID=sa;Password=csdotnet;");
SqlCommand objCommand = new SqlCommand();
objCommand.Connection = objConnection;
objCommand.CommandText = "INSERT INTO authors " +
     "(au_id, au_lname, au_fname, contract) " +
     "VALUES(@au_id, @au_lname, @au_fname, @au_contract)";
objCommand.Parameters.AddWithValue("@au_id",
                                   txtAuId.Text);
objCommand.Parameters.AddWithValue("@au_lname",
                                   txtLastName.Text);
objCommand.Parameters.AddWithValue("@au_fname",
                                   txtFirstName.Text);
objCommand.Parameters.AddWithValue("@au_contract",
                                   chkContract.Checked);
```

متد AddWithValue نام یک placeholder و متغیری که مقدار مربوط به آن را در زمان اجرای برنامه نگهداری می کند را به عنوان پارامتر دریافت کرده و آن را به لیست Parameters اضافه می کنید. برای مثال در این قسمت مشخص کرده ایم که هنگام اجرای دستور، مکان placeholder با نام @au_id باید با مقدار خاصیت Text کنترل txtAuId عوض شود. همچنین مکان placeholder با نام @au_lname نیز باید با مقدار خاصیت Text مربوط به کنترل txtLastName عوض شود و ....

متد ExecuteNonQuery:

بعد از انجام تمام این مراحل می توانید دستور موجود در این شیئ را روی داده های بانک اطلاعاتی اجرا کنید. برای این کار ابتدا باید اتصال خود را به بانک اطلاعاتی بر قرار کنید. سپس با فراخوانی متد ExecuteNonQuery دستور موجود در شیئ SqlCommand را اجرا کنید. البته این متد همانطور که نام آن نیز مشخص می کند فقط زمانی کاربرد دارد که بخواهیم دستوری را روی بانک اطلاعاتی اجرا کنیم که داده ای را برنمی گرداند. برای مثال اگر دستور موجود در شیئ SqlCommand یک دستور SELECT باشد که اطلاعاتی را از جداول استخراج کرده و آنها را به برنامه می دهد، نمی توانیم برای اجرای آن از این متد استفاده کنیم. اما اگر دستور مورد استفاده فقط تغییراتی را در داده های بانک اطلاعاتی ایجاد کند (برای مثال، مانند اینجا یک رکورد از اطلاعات را به جدول اضافه کند)می توانیم با فراخوانی آن متد دستور را در بانک اطلاعاتی اجرا کنیم.
این متد بعد از اجرا عددی را به عنوان خروجی برمی گرداند که مشخص کننده ی تعداد رکورد هایی است که با اجرای این دستور SQL تغییر کرده اند. این عدد معمولا برای بررسی صحت اجرای دستور مورد استفاده قرار می گیرد. قطعه کد زیر نحوه ی استفاده از دستور ExecuteNonQuery را در برنامه نشان می دهد.

```
SqlConnection objConnection = new
SqlConnection("Server=localhost;Database=Pubs;User
ID=sa;Password=csdotnet;");
SqlCommand objCommand = new SqlCommand();
objCommand.Connection = objConnection;
```

```
objCommand.CommandText = "INSERT INTO authors " +
     "(au_id, au_lname, au_fname, contract) " +
     "VALUES(@au_id, @au_lname, @au_fname, @au_contract)";
objCommand.Parameters.AddWithValue("@au_id",
                                   txtAuId.Text);
objCommand.Parameters.AddWithValue("@au_lname",
                                   txtLastName.Text);
objCommand.Parameters.AddWithValue("@au_fname",
                                   txtFirstName.Text);
objCommand.Parameters.AddWithValue("@au_contract",
                                   chkContract.Checked);
objConnection.Open();
objCommand.ExecuteNonQuery();
objConnection.Close();
```

کلاس SqlDataAdapter:

کلاس DataAdapter در برنامه های بانک اطلاعاتی، همانند پلی بین جداول بانک اطلاعاتی و نیز داده های موجود در حافظه که به وسیله ی DataSet نگهداری می شوند، عمل می کنند. این کلاس برای دسترسی به بانک اطلاعاتی از شیئ ایجاد شده از کلاس SqlCommand ای که به آن نسبت داده می شود استفاده می کند و همانطور که می دانید، هر شیئ از کلاس SqlCommand حاوی شیئ ای از کلاس SqlConnection است که ارتباط آن را با بانک اطلاعاتی برقرار می کند. بنابراین می توانیم بگوییم که کلاس DataAdapter برای دسترسی به بانک اطلاعاتی از کلاس SqlCommand و SqlConnection استفاده می کند.

کلاس DataAdapter دارای خاصیتی به نام SelectCommand است. این خاصیت حاوی شیئ ای از نوع SqlCommand است که از دستور موجود در آن شیئ، برای دریافت داده های مورد نیاز در برنامه از بانک اطلاعاتی به کار می رود. به عبارت دیگر DataAdapter دستوری که در این خاصیت نگهداری می شود را روی بانک اطلاعاتی اجرا کرده و نتایج آن را در کلاس هایی مانند DataSet و یا DataTable قرار می دهد تا در برنامه مورد استفاده قرار گیرند. علاوه بر این کلاس DataAdapter دارای خاصیت هایی به نام DeleteCommand، InsertCommand و UpdateCommand است که هر یک شیئ ای از نوع SqlCommand را قبول می کنند و DataAdapter از دستور ذخیره شده در هر یک از آنها به ترتیب برای حذف، درج و یا ویرایش داده ها در بانک اطلاعاتی استفاده می کند. در حقیقت، هنگامی که ما در طی برنامه تغییراتی را درون داده های موجود در حافظه نگه داری می کنیم، DataAdapter با استفاده از دستورات موجود در این خاصیت ها تغییرات ما را از داده های موجود در حافظه به داده های موجود در بانک اطلاعاتی منتقل می کند. ممکن است ابتدا این موارد کمی پیچیده به نظر برسند، اما استفاده از آنها مانند تمام قسمتهای دیگری که تا کنون مشاهده کرده ایم ساده هستند. در قسمتهای قبل مشاهده کردید که چگونه می توان دستورات SELECT مورد نیاز را برای انتخاب داده ها از بانک اطلاعاتی ایجاد کرد. برای تکمیل دستورات مورد نیاز برای کلاس DataAdapter نیز فقط کافی است که دستور SELECT مورد نظر خود را وارد کنید. در ویژوال استودیو کلاسی به نام Command Builder وجود دارد که می تواند بر اساس دستور SELECT وارد شده، دستورات INSERT، UPDATE و یا DELETE مناسب تولید کند.

بنابراین بهتر است که ابتدا با هم خاصیت SelectCommand و نحوه ی استفاده از آن را بررسی کنیم. سپس مشاهده خواهیم کرد که چگونه می توان با استفاده از Command Builder، دستورات دیگر را نیز تولید کرد.

خاصیت SelectCommand:

همانطور که در شکل ۱۶-۱ نیز نشان داده شده است، خاصیت SelectCommand در کلاس DataAdapter برای دریافت داده های مورد نیاز در برنامه از بانک اطلاعاتی و قرار دادن آنها در DataSet به کار می رود.

شکل ۱۶-۱

هنگامی که بخواهید با استفاده از کلاس DataAdapter اطلاعات مورد نیاز را از یک بانک اطلاعاتی دریافت کنید، ابتدا باید خاصیت SelectCommand را در DataAdapter تنظیم کنید. این خاصیت شیئ ای از نوع SqlCommand دریافت کرده که این شیئ مشخص می کند داده ها چگونه باید از بانک اطلاعاتی انتخاب شده و نیز چه داده هایی باید انتخاب شوند. اشیایی که از کلاس SqlCommand ایجاد می شوند، همانطور که در قسمت قبلی مشاهده کردید خود نیز خاصیت هایی دارند که قبل از استفاده باید آنها را تنظیم کرد. این خاصیت ها عبارتند از:

- Connection: یک شیئ از کلاس SqlConnection در این قسمت قرار گرفته و نحوه ی اتصال به بانک اطلاعاتی را مشخص می کند.
- CommandText: دستور SQL و یا نام پروسیجر ذخیره شده در بانک اطلاعاتی که باید توسط این شیئ اجرا شود، در این قسمت ذخیره می شود.

در قسمت قبل از یک دستور SQL مشخص در خاصیت CommandText از کلاس SqlCommand استفاده کردیم. اما اگر بخواهیم نام یک پروسیجر ذخیره شده در بانک اطلاعاتی را در این خاصیت قرار دهیم تا اجرا شود، باید خاصیت دیگری به نام CommandType را نیز در کلاس SqlCommand تنظیم کرده و مقدار آن را برابر با StoredProcedure قرار دهیم تا مشخص شود که متن درون CommandText نام یک پروسیجر ذخیره شده است، نه یک دستور SQL. البته در این فصل فقط از دستورات SQL در خاصیت CommandText استفاده می کنیم، بنابراین نیازی نیست که خاصیت CommandType را تغییر داده و برابر با مقدار خاصی قرار دهیم.

تنظیم خاصیت SelectCommand با استفاده از دستور SQL:

قطعه کدی که در زیر آورده شده است نحوه ی تنظیم خاصیت های مورد نیاز برای اجرای یک دستور SQL به وسیله ی کلاس DataAdapter را نمایش می دهد:

```
// Declare a SqlDataAdapter object...
SqlDataAdapter objDataAdapter = new SqlDataAdapter();

// Assign a new SqlCommand to the SelectCommand property
objDataAdapter.SelectCommand = new SqlCommand();

// Set the SelectCommand properties...
objDataAdapter.SelectCommand.Connection = objConnection;
```

```
objDataAdapter.SelectCommand.CommandText =
     "SELECT au_lname, au_fname FROM authors " +
     "ORDER BY au_lname, au_fname";
```

اولین کاری که باید در این قسمت انجام دهیم این است که شیئ را از نوع SqlDataAdapter ایجاد کنیم. سپس باید خاصیت SelectCommand آن را تنظیم کنیم. برای تنظیم این خاصیت باید یک شیئ از کلاس SqlCommandرا به آن نسبت دهیم، بنابراین شیئ ای را از این کلاس ایجاد کرده و تنظیمات مربوط به Connection آن را نیز انجام می دهیم ا بتواند به یک بانک اطلاعاتی متصل شود. در آخر نیز خاصیت CommandText آن را برابر با یک دستور SQL قرار می دهیم تا آن را روی بانک اطلاعاتی اجرا کرده و نتیجه را دریافت کند.

تنظیم خاصیت SelectCommand با استفاده از پروسیجر ذخیره شده:

در این قسمت مشاهده خواهیم کرد که برای استفاده از یک پروسیجر ذخیره شده در برنامه خاصیتهای لازم را باید چگونه تنظیم کرد. همانطور که در قسمت قبلی نیز گفتن، یک پروسیجر ذخیره شده، یک مجموعه از دستورات SQL است که تحت یک نام مشخص و به صورت یک واحد در بانک اطلاعاتی ایجاد شده و نگهداری می شود. در این قسمت فرض می کنیم پروسیجری به نام usp_select در بانک اطلاعاتی وجود دارد که می خواهیم به جای استفاده از دستور SQL، آن را فراخوانی کرده و نتایج اجرای آن را دریافت کنیم. قطعه کد زیر نحوه ی انجام این کار را نمایش می دهد:

```
// Declare a SqlDataAdapter object...
SqlDataAdapter objDataAdapter = new SqlDataAdapter();

// Assign a new SqlCommand to the SelectCommand property
objDataAdapter.SelectCommand = new SqlCommand();

// Set the SelectCommand properties...
objDataAdapter.SelectCommand.Connection = objConnection;
objDataAdapter.SelectCommand.CommandText = "usp_select";
objDataAdapter.SelectCommand.CommandType =
                        CommandType.StoredProcedure;
```

همانطور که مشاهد ه می کنید در این قطعه برنامه، بر خلاف قسمت قبل که یک دستور SQL را در خاصیت CommandText قرار می دادیم، از نام یک پروسیجر ذخیره شده استفاده کرده ایم. پس باید به نحوی مشخص کنیم که متن داخل CommandText نام یک پروسیجر ذخیره شده است، نه یک دستور SQL. برای تعیین نوع متن موجود در این خاصیت، باید از خاصیت CommandType استفاده کنیم. مقدار پیش فرض این خاصیت برابر با CommandType.Text است که مشخص می کند متن موجود یک دستور SQL است. در این قطعه کد، این مقدار را تغییر داده و برابر با CommandType.StoredProcedure قرار می دهیم تا مشخص شود مقدار موجود در خاصیت CommandText نام یک پروسیجر ذخیره شده است.

استفاده از Command Builder برای ایجاد دستورات SQL دیگر:

با استفاده از خاصیت SelectCommand موجود در کلاس DataAdapter می توانیم داده های مورد نیاز در برنامه را از بانک اطلاعاتی استخراج کرده و در یک DataSet در حافظه قرار دهیم. سپس در طول برنامه می توانیم بـه کـاربر اجـازه دهیم تا تغییرات مورد نظر خود را در داده های موجود در حافظه ایجاد کرده و بعد از اتمام آنها، این تغییرات را به داده های موجود در بانک اطلاعاتی منعکس کنیم. برای این کار لازم است که دستورات SQL مـورد نیـاز بـرای درج، حـذف و یـا ویـرایش داده هـای دریافتی را به کلاس DataAdapter اضافه کنیم تا این کلاس بتواند با استفاده از این دستورات، تغییـرات ایجـاد شـده را در بانک اطلاعاتی وارد کند.
اما برای ایجاد این دستورات لازم است که به زبان SQL تسلط بیشتری داشته باشیم. خوشبختانه روش ساده تری هم بـرای انجـام این کار وجود دارد و آن استفاده از کلاس CommandBuilder است که می تواند با توجه به دسـتور SELECT ای کـه برای DataAdapter وارد کرده ایم، دستورات INSERT، UPDATE و نیز DELETE مناسب تولید کند. قطعه کـد زیر نحوه ی استفاده از این دستور را نمایش می دهد.

```
// Declare a SqlDataAdapter object...
SqlDataAdapter objDataAdapter = new SqlDataAdapter();

// Assign a new SqlCommand to the SelectCommand property
objDataAdapter.SelectCommand = new SqlCommand();

// Set the SelectCommand properties...
objDataAdapter.SelectCommand.Connection = objConnection;
objDataAdapter.SelectCommand.CommandText = "usp_select";
objDataAdapter.SelectCommand.CommandType =
CommandType.StoredProcedure;

// automatically create update/delete/insert commands
SqlCommandBuilder objCommandBuilder =
                new SqlCommandBuilder(objDataAdapter);
```

با استفاده از این کلاس، دستورات لازم برای منعکس کردن تغییرات ایجاد شده از DataSet بـه بانـک اطلاعـاتی بـه صـورت اتوماتیک نوشته می شود. در ادامه ی فصل با نحوه ی انجام این کار بیشتر آشنا خواهیم شد، اما فعلا بهتر است ببینیم که چگونه می توان داده ها را از یک بانک اطلاعاتی استخراج کرده و در یک DataSet در حافظه قرار داد.

متد Fill:

بـا اسـتفاده از متـد Fill در کـلاس DataAdapter مـی توانیـد دسـتور SQL موجـود در خاصـیت SelectCommand را در بانک اطلاعـاتی اجـرا کـرده، و سـپس داده هـای برگـشتی از اجـرای ایـن دسـتور را درون یـک DataSet در حافظه قرار دهید. البته قبل از استفاده از این متد، باید شیئ ای از نوع DataSet ایجاد کنیم.

```
// Declare a SqlDataAdapter object...
```

```
SqlDataAdapter objDataAdapter = new SqlDataAdapter();

// Assign a new SqlCommand to the SelectCommand property
objDataAdapter.SelectCommand = new SqlCommand();

// Set the SelectCommand properties...
objDataAdapter.SelectCommand.Connection = objConnection;
objDataAdapter.SelectCommand.CommandText = "usp_select";
objDataAdapter.SelectCommand.CommandType =
CommandType.StoredProcedure;
DataSet objDataSet = new DataSet();
```

حال که شیئ `DataSet` و نیز `DataAdapter` مورد نیاز را ایجاد کردیم، می توانیم با استفاده از متـد `Fill` داده هـا را از بانک اطلاعاتی در `DataSet` قرار دهیم. متد `Fill` نیز همانند بسیاری از متد های دیگر دارای نـسخه هـای گونـاگونی است، اما یکی از پر کاربرد ترین آنها به صورت زیر مورد استفاده قرار می گیرد:

```
SqlDataAdapter.Fill(DataSet, String);
```

پارامتر `DataSet` در این متد، مشخص کننده ی نام `DataSet` ای اسـت کـه بایـد داده هـا در آن قـرار بگیرنـد. پـارامتر `String` نیز نام جدولی را مشخص می کند که داده ها در `DataSet` درون آن جدول قرار می گیرنـد. `DataSet` هـا نیز می توانند همانند بانکهای اطلاعاتی شامل چندین جدول مختلف از اطلاعات باشند. بنابراین هنگامی که مـی خـواهیم داده ای را در آن قرار دهیم باید مشخص کنیم که نام جدولی که داده ها در آن قرار می گیرند چه باید باشد؟ در این جا می توانیم هـر نـام کـه تمایل داشته باشیم برای جدول انتخاب کنیم، اما بهتر است همواره از اسامی جداولی استفاده کنیم که داده ها از آن گرفته شده اند. به این ترتیب درک برنامه بسیار راحت تر خواهد بود.

قطعه کد زیر یک پروسیجر ذخیره شده در بانک اطلاعاتی را اجرا کرده و نتایج برگشتی از آن را به وسیله ی متد `Fill` در جدولی به نام `authors` در `objDataSet` قرار می دهد:

```
// Declare a SqlDataAdapter object...
SqlDataAdapter objDataAdapter = new SqlDataAdapter();

// Create an instance of a new select command object
objDataAdapter.SelectCommand = new SqlCommand();

// Set the SelectCommand properties...
objDataAdapter.SelectCommand.Connection = objConnection;
objDataAdapter.SelectCommand.CommandText = "usp_select";
objDataAdapter.SelectCommand.CommandType =
CommandType.StoredProcedure;
DataSet objDataSet = new DataSet();
// Fill the DataSet object with data...
objDataAdapter.Fill(objDataSet, "authors");
```

متد Fill برای اتصال به بانک اطلاعاتی از شیئ Connection ای که در خاصیت SelectCommand قرار دارد استفاده می کند. این متد ابتدا بررسی می کند که اتصال این Connection به بانک اطلاعاتی برقرار است یا نه. در صورتی که اتصال برقرار باشد، متد Fill داده های مورد نیاز را از بانک اطلاعاتی بدست آورده، اما اتصال Connection با بانک اطلاعاتی را قطع نمی کند. اگر هم ارتباط شیئ Connection با بانک اطلاعاتی قطع باشد، متد Fill با فراخوانی متد Open ارتباط را برقرار کرده و پس از بدست آوردن اطلاعات مورد نیاز، متد Close را فراخوانی می کند تا اتصال به بانک اطلاعاتی مجدداً قطع شود.

به این ترتیب داده ها از بانک اطلاعاتی درون حافظه قرار می گیرند و می توانید به صورت مستقل آنها را تغییر دهید. دقت کنید که ابتدای کلاس DataSet کلمه ی Sql وجود ندارد. دلیل این مورد هم این است که این کلاس متعلق به فضای نام System.Data.SqlClient نیست بلکه در فضای نام System.Data قرار دارد. به عبارت دیگر کلاس DataSet به سرویس دهنده ی اطلاعاتی خاصی از قبیل SqlClient و یا OleDb تعلق ندارد و وظیفه ی آن نگهداری اطلاعات بدست آمده (به هر نحوی) در حافظه است. هنگامی که اطلاعات را در حافظه قرار دادیم دیگر نیازی نیست بدانیم که این اطلاعات از کجا بدست آمده اند (تا زمانی که بخواهیم آنها را دوباره در بانک اطلاعاتی قرار دهیم).

کلاس DataSet:

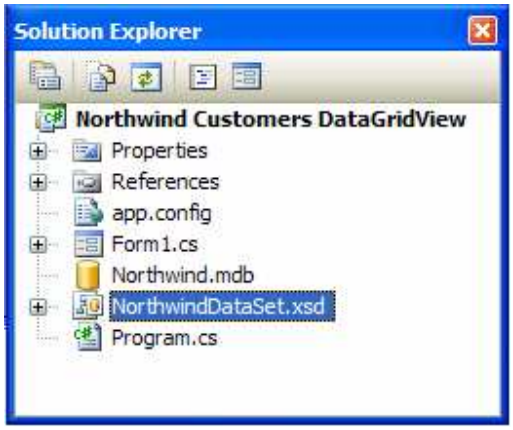
همانطور که گفتم کلاس DataSet برای نگهداری اطلاعات بدست آمده از بانک اطلاعاتی در حافظه به کار می رود. این کلاس شامل مجموعه ای از جداول، رابطه ها، قید و شرط ها و دیگر مواردی است که از بانک اطلاعاتی خوانده شده است. این کلاس خود همانند یک موتور بانک اطلاعاتی کوچک عمل می کند که می تواند داده ها را درون خود در جداولی مجزا نگهداری کرده و به کاربر اجازه دهد که آنها را ویرایش کند. همچنین می توان با استفاده از کلاس DataView پرس وجو هایی را روی داده های موجود در آن اجرا کرد.

داده هایی که در این کنترل قرار دارند از بانک اطلاعاتی قطع هستند و ارتباطی با بانک ندارند. در طول برنامه می توانیم داده های موجود در آن را حذف کرده، ویرایش و یا اضافه کنیم و بعد از اتمام تمام تغییرات مورد نظر، مجدداً با استفاده از DataAdapter به بانک اطلاعاتی متصل شده و تغییرات را در ذخیره کنیم.

کلاس DataSet از ساختار XML برای ذخیره ی داده ها استفاده می کند (در فصل نوزدهم با این ساختار بیشتر آشنا خواهیم شد)، به این ترتیب می توانید داده های موجود در یک شیئ از کلاس DataSet را به سادگی در یک فایل ذخیره کرده و یا آن را با استفاده از شبکه به کامپیوتر دیگری منتقل کنید. البته هنگام برنامه نویسی و کار با DataSet لازم نیست با آنها در قالب XML رفتار کنید. بلکه کافی است تمام کارهای مورد نظر خود را با استفاده از خاصیت ها و یا متدهای موجود در DataSet انجام دهید، بقیه ی امور را کلاس DataSet کنترل خواهد کرد.

مانند هر سند XML دیگری، یک DataSet نیز دارای یک **الگو**[1] است (فایلی که ساختار داده های درون یک یا چند فایل XML را شرح می دهد). در فصل قبل هنگامی که با استفاده از ویزارد یک DataSet را به برنامه اضافه کردیم، فایلی با پسوند XSD[2] ایجاد شد و الگوی DataSet در آن قرار گرفت (شکل ۱۶-۲).

[1] Schema
[2] XML Schema Definition



شکل ۱۶-۲

این فایل حاوی الگوی XML اطلاعاتی بود که به وسیله ی customerDataSet نگهداری می شد. بـه وسـیله ی ایـن فایل ویژوال استودیو کلاسی را از کلاس DataSet مشتق می کد تا بتواند داده های دریافت شده از بانک اطلاعاتی را در شیئ ای از آن کلاس نگهداری کند. البته تمام این موارد نیز از دید برنامه نویس دور می ماند و به صورت درونـی توسـط DataSet انجام می شود.

همچنین می توانیم فیلد های درون یک جدول از DataSet را به کنترلهای درون فرم متصل کنیم، تا آن کنترل ها داده هـای خود را به وسیله ی آن فیلد بدست آورند. در فصل قبل مقداری با انجام این کار آشنا شدید، در ادامه ی فصل نیز بیشتر در این مـورد صحبت خواهیم کرد.

کلاس DataView:

کلاس DataView عموماً برای جستجو، مرتب کردن، فیلتر کردن، ویرایش کردن و یا حرکت کـردن در بـین داده هـای درون یک DataSet مورد استفاده قرار می گیرد. کنترل DataView یک کنترل قابل اتصال است، به این معنـی کـه همـانطور که می توان کنترل ها را به یک DataSet متصل کرد، می توان آنها را به یک DataView نیز متصل کرد. در این مـورد نیز در ادامه ی فصل بیشتر صحبت خواهیم کرد.

همانطور که گفتم یک کنترل DataSet می تواند شامل چندین جدول باشد کـه هـر یـک از آنهـا بـه وسـیله ی یـک کنتـرل DataTable مشخص می شـود. در حقیقـت هنگـامی کـه بـا اسـتفاده از DataAdapter داده هـایی را درون یـک DataSet قرار می دهید، ابتدا یک جدول جدید ایجاد کرده (یک شیئ جدیـد از نـوع DataTable) و سـپس داده هـا را درون آن قرار می دهید و به DataSet اضافه می کنید. کاری که کنترل DataView انجام می دهد این است که به شما اجازه می دهد به صورتی که تمایل دارید به داده های درون یکی از جداول DataSet نگاه کنید. برای مثال آنها را بـه صـورت مرتب شده مشاهده کنید و یا همانند یک بانک اطلاعاتی، دستورات SQL خاصی را روی این جداول اجرا کرده و نتایج آنها را ببینید. می توانید یک شیئ از کلاس DataView را به گونه ای ایجاد کنید کـه شـامل تمـامی داده هـای موجـود در یـک جـدول از DataSet باشد و فقط نحوه ی نمایش آنها را تغییر دهید. برای مثـال اگـر جـدولی بـه نـام authors در DataSet وجود داشته باشد که بر اساس LastName و سپس FirstName مرتب شده است، می توانید یـک DataView را به گونه ای ایجاد کنید که حاوی همان اطلاعات باشد، اما آنهـا را ابتـدا بـر اسـاس FirstName و سـپس LastName مرتب کند. و یا حتی می توانید یک DataView ایجاد کنید که فقط فیلد LastName از جدول authors را نمایش دهد و یا فقط فیلد FirstName را نمایش دهد و ....

البته با وجود اینکه می توانید به وسیله ی کلاس DataView، اطلاعات درون یک DataTable را بـه گونـه ای دیگـر مشاهده کنید، باید دقت داشته باشید که اطلاعـات درون DataView در حقیقـت همـان اطلاعـات درون DataTable هستند. بنابراین هر تغییری که در اطلاعات DataView ایجاد شود، در اطلاعات DataTable نیز منعکس خواهد شـد و بر عکس.
برای ایجاد یک شیئ از کلاس DataView باید نام جدولی که می خواهیم به آن متصل شود را در متد سازنده ی آن مشخص کنیم. در قطعه کد زیر، یک شیئ از کلاس DataView ایجاد شده و به جدول authors از objDataSet متصل می شود. دقت کنید که برای دسترسی به یک جدول خاص از DataSet از خاصیت Tables در کلاس DataSet به همراه نام جدول مورد نظر استفاده کرده ایم.

```
// Set the DataView object to the DataSet object...
DataView objDataView = new
                DataView(objDataSet.Tables("authors"));
```

خاصیت Sort:

هنگامی که یک شیئ از نوع DataView ایجاد کرده و آن را به یک جدول درون DataSet متـصل کردیـد، مـی توانیـد نحوه ی نمایش داده ها را تغییر دهید. برای مثال تصور کنید که می خواهید داده های درون جـدول را بـه گونـه ای متفـاوت مرتـب کنید. برای این کار می توانید از خاصیت Sort در کلاس DataView استفاده کرده و مقدار آن را برابر بـا نـام سـتون و یـا ستون هایی قرار دهید که می خواهید داده ها بر اساس آنها مرتـب شـوند. قطعـه کـد زیـر جـدول authors را بـه وسـیله ی DataView ای که در قسمت قبل ایجاد کردیم، بر اساس FirstName و LastName مرتب می کند:

```
objDataView.Sort = "au_fname, au_lname";
```

همانطور که مشاهده می کنید عبارتی که به این خاصیت نسبت داده می شود، همانند عبارتی است که در مقابل ORDER BY در دستور SELECT زبان SQL وارد می کردیم. در این قسمت نیز همانند دستور SELECT، تمام مرتب سازی ها به طور پـیش فرض به صورت صعودی انجام می شوند و برای اینکه بتوانیم ترتیب مرتب شدن آنها را به صورت نزولی تغییر دهیم، بایـد در مقابـل نام ستون از عبارت DESC استفاده کنیم. برای مثال قطعه کد زیر، داده های موجود در جـدول authors را بـر اسـاس فیلـد FirstName به صورت صعودی و فیلد LastName به صورت نزولی مرتب می کند:

```
objDataView.Sort = "au_fname,au_lname DESC";
```

خاصیت RowFilter:

علاوه بر مرتب کردن داده ها، با استفاده از DataView می توانید داده های موجود در یک جدول را فیلتر کنید، به گونه ای که فقط داده هایی که دارای شرایط خاصی هستند نمایش داده شوند. این امکان هماننـد قـسمت WHERE از دسـتور SELECT در زبان SQL است که شرط خاصی را برای نمایش داده شدن داده ها ایجاد می کرد. برای فیلتـر کـردن اطلاعـات نیـز مـی توانیـد از خاصیت RowFilter استفاده کرده و شرط مورد نظر خود را در آن قرار دهید. نحوه ی وارد کردن دستورات در این قسمت نیز

همانند وارد کردن شرط ها در قسمت WHERE از دستور SELECT است. فقط توجه داشته باشید، به علت اینکه کل عبارت شرط باید درون علامت " قرار بگیرند، پس اگر بخواهید رشته ای را در شرط مشخص کنید باید آن را درون علامت ' قرار دهید. برای مثال قطعه کد زیر در جدول authors فقط افرادی را که LastName آنها برابر با Green است نمایش می دهد:

```
// Set the DataView object to the DataSet object...
DataView objDataView = new
                DataView(objDataSet.Tables("authors"));
objDataView.RowFilter = "au_lname = 'Green'";
```

و یا قطعه کد زیر در جدول authors افرادی که LastName آنها مخالف Green است را برمی گرداند:

```
// Set the DataView object to the DataSet object...
DataView objDataView = new
                DataView(objDataSet.Tables("authors"));
objDataView.RowFilter = "au_lname <> 'Green'";
```

به علاوه در شرطی که در این قسمت وارد می کنید می توانید با استفاده از عبارات AND و یا OR چندین شرط را با یکدیگر ترکیب کرده و سپس داده ها را بر اساس شرط نهایی نمایش دهید. برای مثال قطعه کد زیر در جدول authors افرادی را نمایش می دهد که FirstName آنها با حرف D شروع شده و LastName آنها نیز برابر با Green باشد:

```
objDataView.RowFilter =
          "au_lname <> 'Green' AND au_fname LIKE 'D*'";
```

متد Find:

برای پیدا کردن یک رکورد خاص از اطلاعات در بانک اطلاعاتی می توانید از متد Find در کلاس DataView استفاده کنید. البته قبل از فراخوانی این متد، باید داده های جدول را بر حسب فیلدی که می خواهید جستجو را بر اساس آن انجام دهید مرتب کنید. به عبارت دیگر قبل از فراخوانی متد Find، باید داده های موجود در جدول را بر اساس ستونی که حاوی کلید مورد نظر شماست مرتب کنید.
برای مثال تصور کنید که می خواهید با استفاده از objDataView که در قسمت قبل ایجاد کردیم، در جدول authors به دنبال رکوردی بگردید که FirstName آن برابر با Ann باشد. برای این کار ابتدا باید جدول را بر اساس فیلد au_fname مرتب کنید، سپس با استفاده از متد Find به دنبال Ann بگردید. قطعه کد زیر روش انجام این کار را نمایش می دهد:

```
int intPosition;
objDataView.Sort = "au_fname";
intPosition = objDataView.Find("Ann");
```

به این ترتیب DataView در جدول به دنبال فردی می گردد که FirstName آن برابر با Ann باشد و شماره ی مکان آن را برمی گرداند. اگر چنین فردی در جدول پیدا نشد، این متد مقدار تهی را برمی گرداند. دقت کنید به محض اینکه متـد Find اولین گزینه را پیدا کرد، مکان آن را برگردانده و از جستجوی ادامه ی جدول صرفنظر می کند، بنابراین اگر می دانید که بیش از یـک فرم با این نام در جدول وجود دارد برای مشاهدهی تمام آنها می توانید از روش فیلتر کردن که توضیح داده شد استفاده کنید.
همچنین این متد به کوچکی و یا بزرگی حروف حساس نیست و دستور بالا هر فردی که نام او Ann و یـا ANN و یـا ... باشـد را برمی گرداند.
البته دقت کنید که این متد دقیقاً به دنبال متنی که وارد شده است می گردد بنابراین باید تمام کلمه و یـا کلمـاتی کـه مـی خواهیـد جستجو بر اساس آن صورت گیرد را به صورت دقیق در این قسمت وارد کنید. برای مثال اگر می خواهید در جدول به دنبال فردی با نام خانوادگی Del Castillo بگردید، نمی توانید Del را به عنوان پارامتر به متد Find بفرستید و انتظار داشته باشید که این نام را برای شما برگرداند. بلکه باید نام کامل او را در این قسمت وارد کنید همانند دستور زیر:

```
objDataView.Sort = "au_lname";
intPosition = objDataView.Find("del castillo");
```

در قسمت قبل مشاهده کردید که با استفاده از DataView می توان یک جدول را بر اساس چند فیلد مرتب کرد. همـین مـورد برای جستجو کردن نیز صادق است. به عبارت دیگر می توانید بر اساس چند فیلد به جستجوی داده ها بپردازید. برای این کار بعـد از مرتب کردن جدول، آرایه ای از نوع Object ایجاد می کنید و سپس مقدار مورد نشر برای هر سـتون را در آن قـرار مـی دهیـد. سپس این آرایه را به عنوان پارامتر به متد Find می فرستید. برای مثال اگر بخواهیم ببینیم که آیا فردی با نـام Simon و نـام خانوادگی Watts در جدول وجود دارد یا نه می توانیم از قطعه کد زیر استفاده کنیم:

```
int intPosition;
Object[] arrValues = new Object[1];
objDataView.Sort = "au_fname, au_lname";

// Find the author named “Simon Watts”.
arrValues[0]= "Simon";
arrValues[1] = "Watts";
intPosition = objDataView.Find(arrValues);
```

نکته: دقت کنید که در این قسمت حتماً باید آرایه ای از نوع Object به متد Find فرستاده شود. دلیل این امر هـم در ایـن است که در .NET تمام نوع های داده ای از کلاس Object مشتق می شوند. بنابراین اگر بخواهیم آرایه ای داشته باشیم که هر متغیری را بتوانی در آن قرار دهیم، باید آن را از نوع Object تعریف کنیم. در این جا نیز لازم است آرایه ای داشته باشیم که بتوانیم متغیری از هر نوع داده ای را در آن قرار دهیم. برای مثال فرض کنید بخواهید جستجو در جدول authors را بـه گونـه ای تغییر دهید که افرادی که سن آنها برابر با ۲۵ و نیز نام آنها برابر با Ann است را پیدا کنید. در این صورت باید یک متغییر از نوع عددی و یک متغییر از نوع رشته ای را در آرایه قرار دهید.

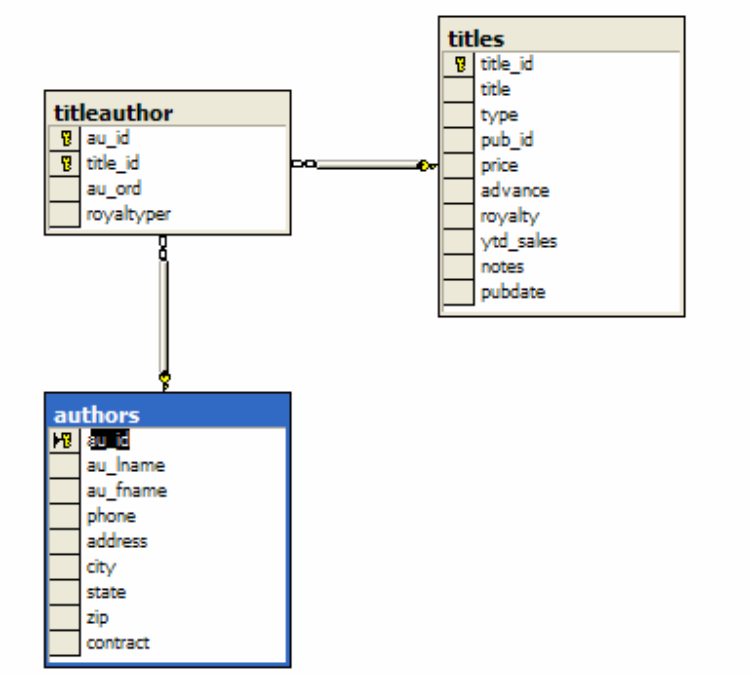
استفاده از کلاسهای ADO.NET در عمل:

تاکنون با اصول کار کلاسهای موجود در ADO.NET آشنا شدیم و مشاهده کردیم که چگونه می توان داده هایی را به وسیله ی این کلاسها از بانک اطلاعاتی SQL Server بدست آورده و یا در آنها وارد کرد. اما تا این قسمت از فصل فقط ذهن خود را با یک سری از مطالب تئوری درگیر کرده بودیم، و برای اینکه مطمئن شویم نحوه ی استفاده از این کلاسها، متد ها، خاصیت ها و ... را درست درک کرده ایم، بهترین راه این است که از آنها در یک مثال عملی استفاده کنیم. در دو بخش امتحان کنید بعد با استفاده از قدرت DataSet ها داده ها را از بانک اطلاعاتی استخراج کرده و به کاربر نمایش خواهیم داد. ممکن است بعد از اتمام این دو بخش امتحان کنید، لازم باشد که به اول فصل برگردید و مجدداً تمام مطالبی را که در مورد کلاسهای ADO.NET عنوان شد را مرور کنید. به این وسیله می توانید اطمینان حاصل کنید که این مطالب به طور کامل در ذهن شما قرار خواهند گرفت.
در بخش امتحان کنید اول از کلاسهای SqlCommand، SqlConnection، SqlDataAdapter و نیز DataSet استفاده کرده و به وسیله ی آنها یک برنامه ی ساده ایجاد می کنیم که داده ها را از یک بانک اطلاعاتی بدست آورد و در یک کنترل DataGrid نمایش دهد. در واقع برنامه ای که در این قسمت خواهیم نوشت، عملکردی بسیار مشابه برنامه ی فصل قبل خواهد داشت. البته با این تفاوت عمده که در این قسمت به جای استفاده از ویزارد، از کد نویسی استفاده خواهیم کرد.

نکته: هنگام نوشتن برنامه های واقعی، معمولا از ویزاردها و کد نویسی به صورت همزمان استفاده می کنند تا بتوانند به سرعت و به راحتی برنامه هایی با انعطاف پذیری بالا ایجاد کنند. کامپوننت هایی که در قسمت قبل با استفاده از جعبه ابزار به فرم اضافه کردیم را در این قسمت با استفاده از کد ایجاد خواهیم کرد. البته نحوه ی استفاده از آنها در هر دو روش یکسان خواهد بود. همچنین در فصل قبل اغلب از ویزاردها استفاده می کردیم، در صورتی که در این فصل بیشتر بر کد نویسی تمرکز خواهیم کرد.

کاربرد DataSet در برنامه:

قبل از اینکه نوشتن برنامه ی این قسمت رو شروع کنیم، بهتر است که به بررسی داده های که می خواهیم در این برنامه نمایش دهیم ونیز رابطه ی بین آنها بپردازیم. اطلاعات این برنامه از بانک اطلاعاتی pubs در SQL Server 2000 استخراج می شوند. البته اگر از نسخه های 2005، 7 و یا MSDE به جای SQL Server 2000 استفاده می کنید نیز باید همین اطلاعات را در بانک اطلاعاتی pubs مشاهده کنید.
این بانک اطلاعاتی مربوط به یک انتشارات فرضی است. در این برنامه می خواهیم لیستی از نویسندگان، کتابهایی که تاکنون چاپ کرده اند و قیمت هر کدام را نمایش دهیم. در شکل ۱۶-۳ این جدولها را به همراه فیلد های موجود در هر کدام و نیز رابطه های بین آنها را نمایش می دهد.
همانطور که در شکل مشاهده می کنید در این برنامه می خواهیم نام و نام خانوادگی نویسنده را از جدول authors بدست آورده و به همراه عنوان و قیمت کتاب او که در جدول titles قرار دارد، در برنامه نمایش دهیم. به علت اینکه یک کتاب می تواند بیش از یک نویسنده داشته باشد و نیز یک نویسنده نیز می تواند بیش از یک کتاب نوشته باشد، اطلاعات این دو جدول در جدول دیگری به نام titleauthor به یکدیگر متصل شده اند.



شکل ۱۶-۳

با توجه به رابطه ی موجود بین جداول و نیز اطلاعاتی که می خواهیم از آنها استخراج کنیم دستور SELECT ای که باید در این مورد استفاده کنیم مشابه زیر خواهد بود:

```
SELECT au_lname, au_fname, title, price
FROM authors
JOIN titleauthor ON authors.au_id = titleauthor.au_id
JOIN titles ON titleauthor.title_id = titles.title_id
ORDER BY au_lname, au_fname
```

خط اول این دستور نام فیلد هایی را که می خواهیم از جداول استخراج کنیم را نمایش می دهد و خط دوم هم مشخص کننده ی نام جدول اصلی است که داده ها از آن استخراج می شوند. در این قسمت داده ها از دو جدول `authors` و نیز `titles` استخراج می شوند، اما جدول `authors` را به عنوان جدول اصلی در نظر می گیریم.
خط سوم بین ستون `au_id` در جدول `authors` و نیز همین ستون در جدول `titleauthor` رابطه بر قرار می کند. به این ترتیب هر زمان که یک رکود از جدول `authors` انتخاب شود، تمام رکورد های موجود در جدول `authors` که مقدار ستون `au_id` آنها برابر با مقدار این ستون در رکورد انتخاب شده از جدول `titleauthor` باشد نیز انتخاب خواهند شد.
خط چهارم نیز مانند خط سوم، بین جدول `titles` و جدول `titleauthor` از طریق ستون `title_id` رابطه بر قرار می کند. به این ترتیب هر زمان که یک رکورد از جدول `titleauthor` انتخاب شود، رکورد های متناظر آن در جدول `titles` نیز انتخاب خواهند شد. خط آخر نیز اطلاعات را بر اساس نام خانوادگی و سپس نام، به صورت صعودی مرتب می کند.

نکته: ممکن است که توضیحات این دستور SELECT برای درک آن کافی نباشد، اما در هر صورت برای اتمام این بخش امتحان کنید بسنده می کند. مسلماً هنگامی که بخواهید برنامه های بانک اطلاعاتی واقعی بنویسید، نیاز خواهید داشت که مفهوم بانکهای اطلاعاتی رابطه ای را درک کرده باشید و نیز بتوانید دستورات SELECT پیچیده ای برای انتخاب داده ها از چندین جدول بنویسید. بنابراین در این صورت بهتر است که کتابهایی را در این زمینه نیز مطالعه کنید.

امتحان کنید: مثال `DataSet`

۱) با استفاده از محیط ویژوال استودیو یک برنامه ی ویندوزی جدید به نام `DataSetExample` ایجاد کنید.
۲) با استفاده از پنجره ی `Properties`، خاصیتهای فرم را به صورت زیر تغییر دهید:

- خاصیت `Size` را برابر با `600; 230` قرار دهید.
- خاصیت `StartPosition` را برابر با `CenterScreen` قرار دهید.
- خاصیت `Text` را برابر با `Bound DataSet` قرار دهید.

۳) با استفاده از قسمت `Data` در جعبه ابزار، یک کنترل `DataGridView` به فرم برنامه اضافه کرده و خاصیتهای آن را به صورت زیر تغییر دهید:

- خاصیت `Name` را برابر با `grdAuthorTitles` قرار دهید.
- خاصیت `Anchor` را برابر با `Top, Left, Bottom, Right` قرار دهید.
- خاصیت `Location` را برابر با `0;0` قرار دهید.
- خاصیت `Size` را برابر با `592;203` قرار دهید.

۴) ویرایشگر کد مربوط به کلاس `Form1` را باز کرده و ابتدا فضای نامهایی که در طول برنامه به آنها نیاز خواهیم داشت را به برنامه اضافه کنید. برای این کار دستور زیر را به بالای تعریف کلاس `Form1` اضافه کنید:

```
// Using Data and SqlClient namespaces...
using System.Data;
using System.Data.SqlClient;

public partial class Form1 : Form
{

}
```

۵) در مرحله ی بعد لازم است که اشیای لازم برای دسترسی به بانک اطلاعاتی و دریافت داده ها را ایجاد کنیم. بنابراین کد های مشخص شده در زیر را به برنامه ی خود اضافه کنید. مطمئن شوید که اطلاعات مربوط به نام کاربری و نیز کلمه ی عبور در `ConnectionString` به درستی وارد شده است.

```
public partial class Form1 : Form
{
```

```
    SqlConnection objConnection = new SqlConnection(
              "server=localhost;database=pubs;" +
              "user id=sa;password=");
    SqlDataAdapter objDataAdapter = new SqlDataAdapter();
    DataSet objDataSet = new DataSet();

    public Form1()
    {
```

نکته: دقت کنید که اگر سرور بانک اطلاعاتی که از استفاده می کنید در کامپیوتر دیگری به جز کامپیوتری که در حال استفاده از آن هستید قرار دارد، باید مقدار پارامتر server را به نام کامپیوتر حاوی SQL Server تغییر دهید. همچنین باید مقدار پارامترهای User ID و Password را نیز به گونه ای تنظیم کنید که به یک نام کاربری و کلمه ی عبور مناسب در سرور اشاره کنند. در غیر این صورت برنامه نخواهد توانست به داده های لازم در بانک اطلاعاتی دسترسی پیدا کند. اگر نام کاربری که در سرور تعریف کرده اید کلمه عبور ندارد، باید قسمت Password را در ConnectionString ذکر کنید اما در مقابل آن چیزی ننویسید. برای مثال Password=;

۶) به قسمت طراحی فرم مربوط به Form1 برگردید و روی نوار عنوان آن دو بار کلیک کنید تا متد مربوط به رویداد Load فرم به صورت اتوماتیک ایجاد شود. سپس کد مشخص شده در زیر را به این متد اضافه کنید:

```
private void Form1_Load(object sender, EventArgs e)
{
    // Set the SelectCommand properties...
    objDataAdapter.SelectCommand = new SqlCommand();
    objDataAdapter.SelectCommand.Connection =
                                        objConnection;
    objDataAdapter.SelectCommand.CommandText =
        "SELECT au_lname, au_fname, title, price " +
        "FROM authors " +
        "JOIN titleauthor ON authors.au_id = " +
        "titleauthor.au_id " +
        "JOIN titles ON titleauthor.title_id = " +
        "titles.title_id "  +
        "ORDER BY au_lname, au_fname";
    objDataAdapter.SelectCommand.CommandType =
                                CommandType.Text;

    // Open the database connection...
    objConnection.Open();

    // Fill the DataSet object with data...
    objDataAdapter.Fill(objDataSet, "authors");

    // Close the database connection...
    objConnection.Close();
```

```
        // Set the DataGridView properties
        // to bind it to our data...
        grdAuthorTitles.AutoGenerateColumns = true;
        grdAuthorTitles.DataSource = objDataSet;
        grdAuthorTitles.DataMember = "authors";

        // Clean up
        objDataAdapter = null;
        objConnection = null;
}
```

۷) با اجرای برنامه نتیجه ای مشابه شکل ۱۶-۴ مشاهده خواهید کرد.



شکل ۱۶-۴

۸) دقت کنید که کنترل `DataGridView` دارای خاصیت درونی مرتب کردن داده ها است. بنابراین اگر روی یکی از نامهای ستونها کلیک کنید، داده های موجود در فرم بر اساس آن ستون مرتب می شوند. همچنین کلیک دوباره بر روی نام یک ستون باعث می شود که داده ها بر حسب آن ستون به صورت نزولی مرتب شوند.

نکته: در این برنامه به علت کمبود جا کد های مربوط به مدیریت خطاها و استثنا های احتمالی حذف شده است، اما بهتر است در برنامه ای که می نویسید این کدها را نیز اضافه کنید. برای مشاهده ی نحوه ی انجام این کار می توانید به فصل یازدهم "اشکال زدایی و کنترل خطا در برنامه" مراجعه کنید.

چگونه کار می کند؟

برای شروع کار ابتدا باید دو فضای نام زیر را به برنامه اضافه کنیم:

```
using System.Data;
using System.Data.SqlClient;
```

فضای نام System.Data برای استفاده از کلاسهای DataSet و DataView و فضای نام System.Data.SqlClient نیز برای استفاده از کلاسهای SqlDataAdapter، SqlConnection و SqlCommand به برنامه اضافه شده اند.
سپس باید اشیایی که در طول برنامه به آنها نیاز داریم را ایجاد کنیم. ممکن است بعدها هنگام تغییر این برنامه بخواهیم از این اشیا در چندین متد از کلاس استفاده نیم. بنابراین آنها را به صورت سراسری در کلاس تعریف می کنیم:

```
public partial class Form1 : Form
{
     SqlConnection objConnection = new
          Connection("server=localhost;database=pubs;" +
          "User ID=sa;Password=");
     SqlDataAdapter objDataAdapter = new SqlDataAdapter();
     DataSet objDataSet = new DataSet();
```

اولین شیئ ای که در این قسمت باید ایجاد کنیم یک شیئ از کلاس SqlConnection است تا به وسیله ی آن بتوانیم به موتور بانک اطلاعاتی متصل شویم. در اینجا موتور بانک اطلاعاتی از نوع SQL Server است و در همان کامپیوتری قرار دارد که در حال اجرای برنامه روی آن هستیم. بنابراین خاصیت Server را برابر با localhost قرار داده و نیز مشخص می کنیم که می خواهیم از داده های موجود در بانک اطلاعاتی pubs استفاده کنیم.
بعد از ایجاد SqlConnection برای برقرار ارتباط با بانک اطلاعاتی، باید یک شیئ از کلاس SqlDataAdapter ایجاد کنیم تا به وسیله ی آن بتوانیم داده ا را از بانک اطلاعاتی استخراج کرده و در یک DataSet در حافظه قرار دهیم.
در آخر نیز باید یک شیئ از نوع DataSet ایجاد کنیم تا به وسیله ی آن بتوانیم داده های دریافت شده از بانک اطلاعاتی را در حافظه نگهداری کنیم. به خاطر داشته باشید که این شیئ به بانک اطلاعاتی متصل نیست و بعد از دریافت داده های مورد نیاز، ارتباط خود را قطع می کند.

نکته: در بعضی از برنامه ها نیازی نیست که این اشیا را به صورت سراسری در کلاس تعریف کنیم. بلکه می توانیم آنها را در یک متد ایجاد کرده، از آنها استفاده کنیم و سپس آنها را نابود کنیم تا فضای اشغال شده به وسیله ی آنها نیز آزاد شود. اما در برنامه هایی که می خواهیم به کاربر اجازه دهیم که داده های موجود در برنامه را تغییر دهد و سپس بخواهیم این تغییرات را در بانک اطلاعاتی وارد کنیم، بهتر است که این اشیا را به صورت سراسری تعریف کنیم تا هم در توابع مربوط به دریافت اطلاعات و هم در توابع مربوط به نوشتن اطلاعات در بانک بتوانیم از آنها استفاده کنیم.

بعد از ایجاد اشیای مورد نیاز، باید کدی را قبل از نمایش داده شدن فرم اجرا کنیم تا داده ها را از بانک اطلاعاتی دریافت کرده و آنها را در صفحه نمایش دهد. شیئ SqlDataAdapter مسئول دریافت اطلاعات از بانک اطلاعاتی و نمایش آن روی فرم برنامه است و این کار را با استفاده از شیئ SqlCommand ای انجام می دهد که در خاصیت SelectCommand آن قرار دارد. بنابراین ابتدا باید یک شیئ از نوع SqlCommand ایجاد کرده و بعد از تنظیم خاصیت های آن، آن را در SelectCommand قرار دهیم.

```
// Set the SelectCommand properties...
objDataAdapter.SelectCommand = new SqlCommand();
```

```
objDataAdapter.SelectCommand.Connection = objConnection;
objDataAdapter.SelectCommand.CommandText =
"SELECT au_lname, au_fname, title, price " +
"FROM authors " +
"JOIN titleauthor ON authors.au_id = titleauthor.au_id " +
"JOIN titles ON titleauthor.title_id = titles.title_id "  +
"ORDER BY au_lname, au_fname";
objDataAdapter.SelectCommand.CommandType =
                                        CommandType.Text;
```

برای این کار نیز ابتدا با استفاده از دستور new یک شیئ جدید از نوع SqlCommand ایجاد کرده و آن را در خاصیت SelectCommand قرار می دهیم. سپس خاصیت Connection این شیئ جدید را تنظیم می کنیم. این خاصیت برای ایجاد اتصال به بانک اطلاعاتی به کار می رود.
بعد از این باید دستور SQL ای که برای دریافت داده ا از بانک اطلاعاتی به کار می رود را مشخص کنیم. برای این کار متن حاوی دستور را در خاصیت CommandText قرار داده و با تنظیم خاصیت CommandType نیز مشخص می کنیم که این متن شامل یک دستور SQL است.
بعد از اتمام تمام این امور می توانیم به بانک اطلاعاتی متصل شده، داده های مورد نیاز را دریافت کرده و سپس اتصال را قطع کنیم.
برای متصل شدن به بانک اطلاعاتی از متد Open در کلاس SqlConnection استفاده می کنیم:

```
// Open the database connection...
objConnection.Open();
```

سپس با استفاده از متد Fill در کلاس SqlDataAdapter داده های مورد نیاز را از بانک اطلاعاتی دریافت کرده و در شیئ ایجاد شده از کلاس DataSet در حافظه قرار می دهیم. هنگام فراخوانی متد Fill نیز، نام شیئ DataSet و نیز نام جدولی که می خواهیم داده ها در آن قرار بگیرند را به عنوان پارامتر به متد ارسال می کنیم. در اینجا با وجود اینکه داده ها از چندین جدول مختلف دریافت شده اند برای نام جدول از نام authors استفاده کرده ایم.

```
// Fill the DataSet object with data...
objDataAdapter.Fill(objDataSet, "authors");
```

بعد از این کار نیز با فراخوانی متد Close از کلاس SqlConnection، اتصال ایجاد شده به بانک اطلاعاتی را قطع می کنیم.

```
// Close the database connection...
objConnection.Close();
```

البته همانطور که در قسمتهای قبلی نیز گفتم، می توانیم بدون اینکه با استفاده از متد Open به بانک اطلاعاتی متصل شده و سپس با استفاده از متد Close اتصال را قطع کنیم از متد Fill برای پر کردن DataSet استفاده کنیم. در این حالت هنگامی که متد Fill متوجه شود که به بانک اطلاعاتی متصل نیست، به صورت اتوماتیک متد Fill را فراخوانی کرده و بعد از اتمام کار نیز اتصال را قطع می کند. بنابراین در این قسمت می توانیم بدون هیچ مشکلی، خطهای مربوط فراخوانی متدهای Open و Close را از برنامه حذف کنیم.

بعد از دریافت داده ها باید خاصیتهای کنترل DataGridView را به گونه ای تنظیم کنیم تا بتوانـد داده هـای موجـود را در برنامه نمایش دهد. اولین خاصیت، خاصیت AutoGenerateColumns اسـت. بـا قـرار دادن مقـدار true در ایـن خاصیت در حقیقت به DataGridView اجازه می دهیم که هنگام نمایش داده ها بر روی فرم، سـتونهای مـورد نیـاز را بـه صورت اتوماتیک ایجاد کند. بعد از این نیز باید خاصیت DataSource را برابر با نام DataSet ای قرار دهـیم کـه مـی خواهیم کنترل DataGridView داده های مورد نیاز خود را از آن دریافت کند:

```
//Set the DataGridView properties to bind it to our data...
grdAuthorTitles.AutoGenerateColumns = true;
grdAuthorTitles.DataSource = objDataSet;
grdAuthorTitles.DataMember = "authors";
```

خاصـیت DataMember را نیـز بایـد برابـر بـا نـام جـدولی از DataSet قـرار دهـیم کـه مـی خـواهیم داده ای آن در DataGridView نمایش داده شود، که در اینجا برابر با جدول authors است.
در آخر نیز برای اینکه حافظه ی اشغال شده به وسیله ی این کنترل ها را آزاد کنیم، آنها را برابر با مقدار null قرار می دهیم:

```
// Clean up
objDataAdapter = null;
objConnection = null;
```

به این ترتیب هنگام اجرای برنامه، ابتدا DataGridView با استفاده از الگوی داده هـایی کـه در DataSet قـرار دارد ستون های لازم برای نمایش داده ها در فرم را ایجاد می کند (این الگو هنگام پر شدن DataSet به وسـیله ی متـد Fill ایجاد می شود). سپس تمام داد ها از DataSet دریافت شده و در DataGridView نمایش داده خواهند شد.
در بخش امتحان کنید بعد، با بعضی از خاصیت ها و متدهای DataGridView که به وسیله ی آنهـا مـی تـوانیم نحـوه ی نمایش داده ها در صفحه را تغییر دهیم آشنا خواهیم شد.

امتحان کنید: تغییر خاصیت های DataGridView

۱) در زیر لیستی از تغییراتی که می توانید در یک DataGridView انجام دهید تا داده ها بهتر نمـایش داده شـوند، آورده شده است:

- عنوان ستونها را برابر با نام مناسبی قرار دهید.
- اندازه ی هر ستون را به گونه ای تغییر دهید تا بتوان به راحتی داده های آن را مطالعه کرد.
- رنگ هر ردیف از اطلاعات را به گونه ای تغییر دهید که به صورت متمایز نمایش داده شوند.
- داده ها را در ستونها به صورت راست–چین قرار دهید (برای نمایش داده های عددی).

برای انجام این موارد، در متد Form_Load تغییرات مشخص شده در زیر را اعمال کنید:

```
private void Form1_Load(object sender, EventArgs e)
{
```

```
        // Set the SelectCommand properties...
        objDataAdapter.SelectCommand = new SqlCommand();
        objDataAdapter.SelectCommand.Connection =
                                        objConnection;
        objDataAdapter.SelectCommand.CommandText =
                  "SELECT au_lname, au_fname, title, price " +
                  "FROM authors " +
                  "JOIN titleauthor ON authors.au_id = " +
                  "titleauthor.au_id " +
                  "JOIN titles ON titleauthor.title_id = " +
                  "titles.title_id "  +
                  "ORDER BY au_lname, au_fname";
        objDataAdapter.SelectCommand.CommandType =
                                        CommandType.Text;

        // Open the database connection...
        objConnection.Open();

        // Fill the DataSet object with data...
        objDataAdapter.Fill(objDataSet, "authors");

        // Close the database connection...
        objConnection.Close();

        // Set the DataGridView properties
        // to bind it to our data...
        grdAuthorTitles.AutoGenerateColumns = true;
        grdAuthorTitles.DataSource = objDataSet;
        grdAuthorTitles.DataMember = "authors";

        // Declare and set
        // the currency header alignment property...
        DataGridViewCellStyle objAlignRightCellStyle = new
                                        DataGridViewCellStyle();
        objAlignRightCellStyle.Alignment =
                  DataGridViewContentAlignment.MiddleRight;

        // Declare and set the alternating rows style...
        DataGridViewCellStyle objAlternatingCellStyle = new
                                        DataGridViewCellStyle();
        objAlternatingCellStyle.BackColor = Color.WhiteSmoke;
        grdAuthorTitles.AlternatingRowsDefaultCellStyle =
                                        objAlternatingCellStyle;

        // Declare and set the style for currency cells ...
        DataGridViewCellStyle objCurrencyCellStyle = new
                                        DataGridViewCellStyle();
```

```
        objCurrencyCellStyle.Format = "c";
        objCurrencyCellStyle.Alignment =
                  DataGridViewContentAlignment.MiddleRight;

        // Change column names
        // and styles using the column name
        grdAuthorTitles.Columns["price"].HeaderCell.Value =
                                        "Retail Price";
        grdAuthorTitles.Columns["price"].HeaderCell.Style =
                                        objAlignRightCellStyle;
        grdAuthorTitles.Columns["price"].DefaultCellStyle =
                                        objCurrencyCellStyle;

        // Change column names
        // and styles using the column index
        grdAuthorTitles.Columns[0].HeaderText = "Last Name";
        grdAuthorTitles.Columns[1].HeaderText = "First Name";
        grdAuthorTitles.Columns[2].HeaderText = "Book Title";
        grdAuthorTitles.Columns[2].Width = 225;

        // Clean up
        objDataAdapter = null;
        objConnection = null;
        objCurrencyCellStyle = null;
        objAlternatingCellStyle = null;
        objAlignRightCellStyle = null;
}
```

۲) مجدداً برنامه را اجرا کنید. مشاهده خواهید کرد که داده ها در جدولی مشابه شکل ۱۶-۵ نمایش داده می شوند. با مقایسه ی این شکل با شکل ۱۶-۴ متوجه تفاوت های ایجاد شده در برنامه خواهید شد.



شکل ۱۶-۵

چگونه کار می کند؟

استیل هر سلول در کنترل DataGridView به صورت وراثتی تعیین می شود. به عبارت دیگر در هر کنترل DataGridView می توانید یک استیل کلی مشخص کنید. به این ترتیب این استیل به تمام سلولها به ارث می رسد و تمام سلولهای این کنترل دارای همین استیل خواهند بود مگر اینکه به صورت مشخص آن را تغییر دهیم. در این کنترل تمام مجموعه های عضو نیز دارای خاصیتی به همین صورت هستند. برای مثال اگر استیل پیش فرض یک DataGridView را برابر با استیل A قرار دهیم، تمام سلولهای این کنترل دارای استیل A خواهند بود. اما اگر استیل پیش فرض یک ستون از سلولها در DataGridView را برابر با استیل B قرار دهیم، تمام سلولهایی که در آن ستون قرار دارند دارای استیل B خواهند شد.
استیل هر سلول به وسیله ی شیئ ای از نوع DataGridViewCellStyle تعیین می شود. بنابراین برای شروع شیئ ای از نوع این کلاس ایجاد کرده و خاصیت Alignment آن را برابر با MiddleRight قرار می دهیم.

```
// Declare and set
// the currency header alignment property...
DataGridViewCellStyle objAlignRightCellStyle = new
          DataGridViewCellStyle();
objAlignRightCellStyle.Alignment =
          DataGridViewContentAlignment.MiddleRight;
```

قبل از هر چیز بهتر است جدول اطلاعات در برنامه را به گونه ای تغییر دهیم که ردیفها به صورت یکی در میان رنگ متفاوتی داشته باشند، به این ترتیب خواندن اطلاعات ساده تر می شود. برای این کار کافی است رنگ ردیفهای فرد را به صورت قبلی قرار داده و رنگ ردیفهای زوج را تغییر دهیم. برای ایجاد تغییر در ردیفهای فرد کافی است شیئ ای از نوع DataGridViewCellStyle ایجاد کرده و بعد از تنظیم قسمتهای مورد نظر در این شیئ، آن را در خاصیت AlternatingRowsDefaultCellStyle قرار دهیم. در اینجا نیز با استفاده از این روش رنگ ردیفهای زوج را به صورت WhiteSmoke در می آوریم:

```
// Declare and set the alternating rows style...
DataGridViewCellStyle objAlternatingCellStyle = new
                                   DataGridViewCellStyle();
objAlternatingCellStyle.BackColor = Color.WhiteSmoke;
grdAuthorTitles.AlternatingRowsDefaultCellStyle =
                                   objAlternatingCellStyle;
```

در مرحله ی بعد نیز با ایجاد یک شیئ جدید از کلاس DataGridViewCellStyle، آن را به گونه ای تنظیم می کنیم تا بتواند اعداد مالی را با قالب صحیح و نیز از راست به چپ نمایش دهد.

```
// Declare and set the style for currency cells ...
DataGridViewCellStyle objCurrencyCellStyle = new
DataGridViewCellStyle();
objCurrencyCellStyle.Format = "c";
objCurrencyCellStyle.Alignment =
          DataGridViewContentAlignment.MiddleRight;
```

سپس باید عنوان ستون price را به نامی با معنی تر تغییر داده و دو استیلی که در مرحله ی قبل ایجاد کرده بودیم را (یکی برای نمایش اعداد مالی و دیگری برای تنظیم نحوه ی نمایش تیتر یکی از ستونها) به سلولهای مورد نظر نسبت دهیم.

```
// Change column names and styles using the column name
grdAuthorTitles.Columns["price"].HeaderCell.Value =
                                    "Retail Price";
grdAuthorTitles.Columns["price"].HeaderCell.Style =
                                    objAlignRightCellStyle;
grdAuthorTitles.Columns["price"].DefaultCellStyle =
                                    objCurrencyCellStyle;
```

در انتها نیز عنوان دیگر ستونها را با استفاده از خاصیت HeaderText و یا HeaderText.Value به نام با معنی تری تغییر خواهیم داد. همچنین برای اینکه عنوان کتابها به سادگی قابل خواند باشد، طول سلولهای آن را افزایش می دهیم:

```
// Change column names and styles using the column index
grdAuthorTitles.Columns[0].HeaderText = "Last Name";
grdAuthorTitles.Columns[1].HeaderText = "First Name";
grdAuthorTitles.Columns[2].HeaderText = "Book Title";
grdAuthorTitles.Columns[2].Width = 225;
```

در این قسمت مشاهده کردید که چگونه می توان داده های درون یک DataSet را به یک کنترل مانند DataGridView متصل کرد. در بخش امتحان کنید بعد، این مثال را مقداری بسط داده و سعی خواهیم کرد داده های درون یک کنترل DataView را به چندین کنترل متصل کرده و سپس با استفاده از شیئ CurrencyManager بین آنها حرکت کنیم. اما قبل از اینکه این برنامه را شروع کنیم، بهتر است مقداری در رابطه با اتصال داده، نحوه ی انجام این کار با کنترلهای ساده ای مانند TextBox و چگونگی جا به جا شدن بین رکورد ها صحبت کنیم.

اتصال داده ها:

کنترل DataGridView بهترین کنترل برای نمایش تمام داده ها در فرم برنامه است. این کنترل علاوه بر این قابلیت، می تواند به راحتی به کاربر اجازه دهد که داده ها را حذف و یا ویرایش کند و یا داده های جدیدی را در جدول وارد کند. اما با این وجود ممکن است در شرایط بخواهید در هر لحظه فقط یک سطر از داده ها را در برنامه نمایش دهید. در این مواقع تنها راه این است که تعدادی کنترل ساده مانند TextBox بر روی فرم قرار داده و هر یک از آنها را به یکی از فیلد های جدول در برنامه متصل کنیم، سپس در هر لحظه اطلاعات مربوط به یک سطر از اطلاعات را در این کنترل ها قرار دهیم. با استفاده از این روش می توانید کنترل بیشتری روی داده ها داشته باشید، اما کدی که برای این کار باید در برنامه وارد کنید نیز مشکل تر خواهد بود، زیرا باید برنامه ای بنویسید که هر یک از کنترلهای روی فرم را به فیلد مربوط متصل کند. سپس قسمتی را در فرم برنامه طراحی کنید که به وسیله آن بتوان در بین سطر های اطلاعات حرکت کرد. بنابراین برای انجام این کار لازم است که درگیر اموری مانند اتصال کنترل های ساده به داده ها و نیز مدیریت این اتصالات شویم.

در بحث راجع به اتصال داده ها، منظور از **کنترل های ساده** کنترل هایی است که در هر لحظه فقط می توانند مقدار یک داده را در خود نگهداری کنند، برای مثال مانند TextBox، CheckBox، RadioButton و یا کنترل هایی از این قبیل. کنترل هایی مانند comboBox، ListBox و یا DataGridView که در هر لحظه می توانند بیش از یک آیتم از داده های موجود در برنامه را نمایش دهند به عنوان کنترلهای ساده در نظر گرفته نمی شوند.

BindingContext و CurrencyManager:

هر فرم دارای شیئ ای از نوع BindingContext است که اتصالات کنترل های درون فرم را مدیریت می کند. بنابراین به علت اینکه فرم برنامه ی شما به صورت درونی دارای چنین شیئ ای است، نیازی نیست که آن را در کد ایجاد کنید.
شیئ BindingContext در حقیقت یک مجموعه از اشیا از نوع CurrencyManager را مدیریت می کند. وظیفه ی CurrencyManager نیز این است که بین کنترل هایی که به منبع داده ای (مثلا DataSet) متصل هستند و منبع داده ای، و نیز این کنترل ها با دیگر کنترل هایی که در فرم به همان منبع داده ای متصل هستند هماهنگی برقرار کند. به این ترتیب می توان مطمئن شد که تمام این کنترل ها در فرم در حال نمایش داده های موجود در یک سطر هستند. شیئ CurrencyManager می تواند این هماهنگی را بین کنترل ها و منابع داده ای مختلفی مانند DataSet، DataTable، DataView و یا DataSetView ایجاد کند. هر زمان که یک منبع داده ای جدید به فرم برنامه اضافه کنید، یک شیئ CurrencyManager جدید نیز به صورت اتوماتیک ایجاد می شود. به این ترتیب کار با کنترل های متصل به یک منبع داده ای در فرم برنامه بسیار ساده خواهد شد.
اگر در برنامه ی خود از چندین منبع داده ای مختلف استفاده می کنید، می توانید یک متغییر از نوع CurrencyManager ایجاد کرده و آن را به BindingContext مربوط به منبع داده ای مورد نظر خود در ارجاع دهید. به این ترتیب به وسیله ی این متغییر می توانید به CurrencyManager مورد نظر خود در BindingContext دسترسی داشته باشید و به وسیله ی آن نمایش داده ها را بر روی فرم کنترل کنید.
قطعه کد زیر با استفاده از شیئ DataSet ای که در برنامه ی قبل ایجاد کرده بودیم، یک ارجاع به شیئ CurrencyManager ای که منبع داده ای مربوط به جدول authors را کنترل می کرد ایجاد می کند. برای این کار ابتدا یک متغییر از کلاس CurrencyManager ایجاد می کنیم. سپس مقدار این متغییر را برابر با CurrenyManager مربوط به منبع داده ای objDataSet در BindinContext قرار می دهیم. البته دقت کنید شیئ ای که در BindingContext ذخیره می شود از نوع CurrencyManager نیست و باید با استفاده از عملگر ( ) آن را به صورت صریح به CurrencyManager تبدیل کنیم:

```
CurrencyManager objCurrencyManager;
objCurrencyManager =
     (CurrencyManager)(this.BindingContext[objDataSet]);
```

بعد از اینکه با استفاده از این کد، ارجاعی به این شیئ ایجاد کردیم می توانیم با استفاده از خاصیت Position موقعیت رکورد جاری[1] را کنترل کنیم. برای مثال کد زیر موقعیت رکورد جاری را یک واحد افزایش می دهد:

[1] منظور از رکورد جاری در CurrencyManager یک منبع داده ای، رکوردی است که تمام کنترل های ساده ای از فرم که به این منبع داده ای متصل شده اند باید اطلاعات آن را رکورد را نمایش دهند.

```
objCurrencyManager.Position += 1;
```

و یا دستور زیر باعث می شود که تمام کنترل های ساده ای که به objDataSet متصل شده اند، اطلاعات رکورد قبلی را نمایش دهند:

```
objCurrencyManager.Position -= 1;
```

همچنین برای نمایش اطلاعات مربوط به اولین رکورد در objDataSet می توانیم از کد زیر استفاده کنیم:

```
objCurrencyManager.Position = 0;
```

خاصیت Count در کلاس CurrencyManager حاوی تعداد رکورد هایی است که در منبع داده ای که به وسیله ی CurrencyManager مدیریت می شود وجود دارد. بنابراین برای نمایش اطلاعات مربوط به رکورد آخر در فرم برنامه می توانید از کد زیر استفاده کنید:

```
objCurrencyManager.Position = objCurrencyManager.Count - 1;
```

دقت کنید که در این کد از تعداد رکورد های موجود منهای یک برای رفتن به آخرین رکورد استفاده کرده ایم. دلیل این کار این است که اندیس رکورد ها در این شیئ از صفر شروع می شود، بنابراین اندیس رکورد آخر برابر با تعداد کل رکورد ها منهای یک خواهد بود.

اتصال کنترل ها:

برای اتصال یک کنترل به یک منبع داده ای، باید از خاصیت DataBindings در آن کنترل استفاده کنیم. این خاصیت از کلاس DatabindingsCollection است و خود نیز دارای چندین خاصیت و متد مختلف است. اما در این قسمت از متد Add آن استفاده خواهیم کرد.
این متد سه پارامتر دریافت کرده و به صورت زیر فراخوانی می شود:

```
object.DataBindings.Add(propertyName,
                        dataSource, dataMember);
```

این پارامتر ها برای موارد زیر مورد استفاده قرار می گیرند:

- *object* نام کنترلی است که می خواهیم یک اتصال جدید برای آن ایجاد کنیم.
- *propertyName* حاوی نام خاصیتی است که می خواهیم در طول برنامه مقدار خود را از منبع داده ای دریافت کند.
- *dataSource* نام منبع داده ای که است که می خواهیم اطلاعات مورد نیاز برای این کنترل را از آن دریافت کنیم و می تواند شامل یک DataSet، DataView، DataTable و یا هر منبع داده ای دیگری باشد.
- *dataMember* مشخص کننده ی نام فیلدی از منبع داده ای است که می خواهیم آن را به خاصیت propertyName از کنترل متصل کنیم.

مثـالی از نحـوه ی اسـتفاده از متـد Add در قطعـه کـد زیـر آورده شـده اسـت. کـد زیـر خاصـیت Text در کنتـرل txtFirstName را به فیلد au_fname از شیئ objDataView متصل می کند:

```
txtFirstName.DataBindings.Add("Text",
                                objDataView, "au_fname");
```

در مواقعی ممکن است بعد از ایجاد اتصال در یک کنترل بخواهید تمام اتصالات آن با منابع داده ای را حذف کنید. برای این کار می توانید از متد Clear در کلاس ControlBindingsCollection استفاده کنید. این متد تمام اتصالاتی کـه برای خاصیتهای مختلف یک کنترل تعریف شده بود را حذف می کند. نحوه ی استفاده از این متد در کد زیر نشان داده شده است:

```
txtFirstName.DataBindings.Clear();
```

حـــال کـــه بـــا اشـــیای BindingContext، ControlBindingsCollection و نیـــز CurrencyManager آشنا شدیم، بهتر است نحوه ی استفاده از آنها در یک برنامه را نیز مشاهده کنیم.

مثال ایجاد اتصال:

برای مثالی که در بخش امتحان کنید بعد بررسی خواهیم کرد، فقط از کلاس هایی که در قسمت قبل معرفی شد اسـتفاده نخـواهیم کرد، بلکه کلاس های SqlCommand، DataView و Sqlparameter را نیز به کار خواهیم برد.

نکته: در این برنامه نیز از پرس و جویی که در برنامه ی قبل ایجاد کردیم، استفاده کرده و نام و نام خانوادگی هر نویـسنده، عنـاوین کتابهای چاپ شده از او و نیز قیمت هر کدام را در فرم برنامه نمایش خواهیم داد. این مثال با مثال قبلی فقط از این لحاظ تفاوت دارد که در این برنامه در هر لحظه فقط اطلاعات مربوط به یک رکورد را نمایش می دهیم.

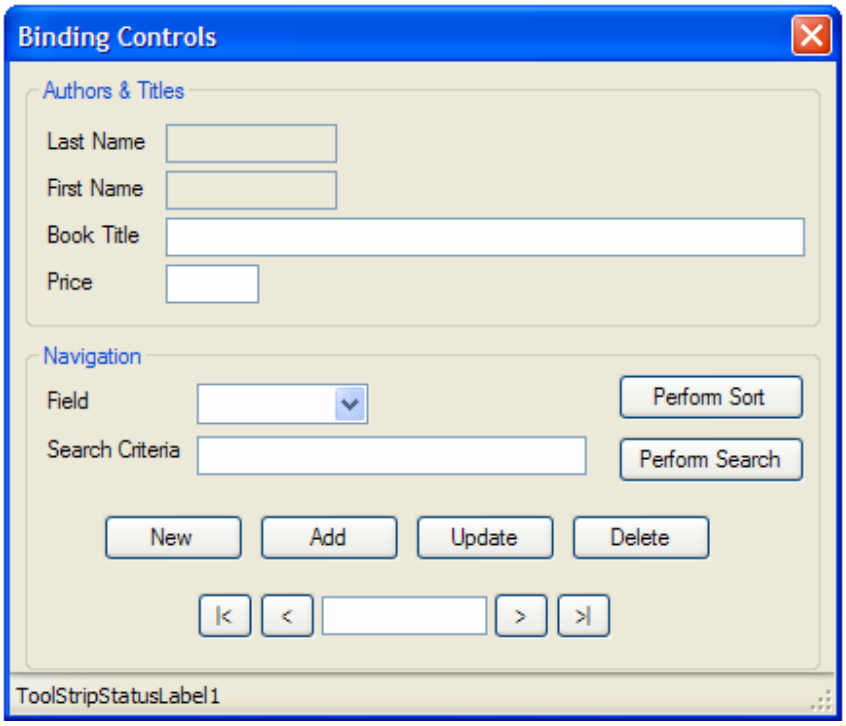
امتحان کنید: اتصال کنترلهای ساده به منبع داده ای

۱) با استفاده از ویژوال استودیو یک برنامه ویندوزی جدید به نام BindingExample ایجاد کنید.
۲) با استفاده از جعبه ابزار یک کنترل ToolTip به برنامه اضافه کنید. این کامپوننت نیز مانند تمام کامپوننت های دیگر به قسمت پایین بخش طراحی فرم مربوط به Form1 اضافه خواهد شد.
۳) بر روی فرم برنامه کلیک کنید تا انتخاب شود. سپس با استفاده از پنجـره ی Properties خاصـیتهای آن را بـه صورت زیر تغییر دهید:

- خاصیت FormBorderStyle را برابر با FixedDialog قرار دهید.
- خاصیت MaximizeBox را برابر با False قرار دهید.
- خاصیت MinimizeBox را برابر با False قرار دهید.
- خاصیت Size را برابر با 430;360 قرار دهید.

- خاصیت `StartPosition` را برابر با `CenterScreen` قرار دهید.
- خاصیت `Text` را برابر با `Binding Controls` قرار دهید.

۴) در این قسمت باید کنترل هایی را به فرم برنامه اضافه کرده و سپس خاصیتهای مختلف آنها را تنظیم کنیم تا فرم برنامه مشابه شکل ۱۶-۶ شود.
این مراحل به این دلیل طی می شوند تا ظاهر برنامه همانند یک برنامه ی واقعی شود. با این وجود، این مراحل اهمیت زیادی ندارند و در صورت لزوم می توانید از آنها صرفنظر کرده و خودتان فرمی را مشابه شکل ۱۶-۶ ایجاد کنید. البته در این صورت دقت کنید که اسامی کنترل ها باید مشابه آنچه باشند که در این قسمت عنوان شده است. در غیر این صورت ممکن است در اجرای برنامه با مشکل مواجه شوید.



شکل ۱۶-۶

۵) یک کنترل `GroupBox` به فرم برنامه اضافه کرده و خاصیتهای آن را به صورت زیر تغییر دهید:

- خاصیت `Location` را برابر با `8;8` قرار دهید.
- خاصیت `Size` را برابر با `408;128` قرار دهید.
- خاصیت `Text` را برابر با `Authors && Titles` قرار دهید.

نکته: برای نمایش علامت & در عنوان یک کنترل `GroupBox` باید از علامت && استفاده کنید. استفاده از یک & در عنوان باعث می شود که کاراکتر بعد از آن با زیرخط نمایش داده شود.

۶) چهار کنترل `Label` با خاصیت هایی که در جدول زیر عنوان شده است را به کنترل `GroupBox1` اضافه کنید:

Name	Location	Size	Text	AutoSize
Label1	8;26	64;16	Last Name	False
Label2	8;50	64;16	First Name	False
Label3	8;74	56;16	Book Title	False
Label4	8;98	64;16	Price	False

۷) با استفاده از جعبه ابزار چهار کنترل TextBox نیز به GroupBox1 در برنامه اضافه کـرده و خاصـیتهای آن را بر اساس جدول زیر تنظیم کنید:

Name	Location	Size	ReadOnly
txtLastName	72;24	88;20	True
txtFirstName	72;48	88;20	True
txtBookTitle	72;72	328;20	False
txtPrice	72;96	48;20	False

۸) با استفاده از جعبه ابزار کنترل GroupBox دیگری به فرم اضافه کرده و خاصیتهای آن را طبـق لیـست زیـر تنظـیم کنید:

- خاصیت Location را برابر با 8;144 قرار دهید.
- خاصیت Size را برابر با 408;168 قرار دهید.
- خاصیت Text را برابر با Navigation قرار دهید.

۹) دو کنترل Label به GroupBox2 اضافه کرده و بر اساس جدول زیر آن را تنظیم کنید:

Name	Location	Size	Text	AutoSize
Label5	8;23	64;16	Field	False
Label6	8;48	80;16	Search Criteria	False

۱۰) با استفاده از جعبه ابزار یک کنترل ComboBox به GroupBox2 اضافه کنید. خاصیت Name آن را برابر بـا cboField، خاصیت Location را برابـر بـا 88;21، خاصـیت Size را برابـر 88;21 و خاصـیت DropDownStyle را برابر با DropDownList قرار دهید.

۱۱) دو کنترل TextBox به GroupBox2 اضافه کرده و خاصیتهای آن را بر اساس جدول زیر تغییر دهید:

Name	Location	Size	TabStop	TextAlign
txtSearchCriteria	88;48	200;20	-	-
txtRecordPosition	152;130	85;20	False	Center

۱۲) ده کنترل Button به GroupBox2 اضافه کرده و خاصیتهای آنها را به صورت زیر تغییر دهید:

Name	Location	Size	Text	ToolTipon ToolTip1
btnPerformSort	304;16	96;24	Perform Sort	-
btnPerformSearch	304;48	96;24	Perform Search	-
btnNew	40;88	72;24	New	-
btnAdd	120;88	72;24	Add	-
btnUpdate	200;88	72;24	Update	-
btnDelete	280;88	72;24	Delete	-
btnMoveFirst	88;128	29;24	|<	Move First
btnMovePrevious	120;128	29;24	<	Move Previous
btnMoveNext	200;128	29;24	>	Move Next
btnMoveLast	272;128	29;24	|>	Move Last

۱۳) در آخر نیز یک کنترل StatusStrip به برنامه اضافه کنید. نیازی به تغییر خاصیتهای Name، Location و یا Size این کنترل نیست. بعد از انتخاب این کنترل، با استفاده از منوی کنار آن یک کنترل StatusLabel را به آن اضافه کنید.

۱۴) بعد از اتمام تمام این مراحل، فرم کامل شده ی برنامه باید مشابه شکل ۱۶-۶ باشد.

۱۵) حال قسمت کد نویسی برنامه را شروع می کنیم. به قسمت ویرایشگر کد مربوط به کلاس Form1 رفته[1] و با قرار دادن کد زیر در بالای کدها، فضای نام System.Data و System.Data.SqlClient را به برنامه اضافه کنید:

```
// Import Data and SqlClient namespaces
using System.Data;
using System.Data.SqlClient;
```

۱۶) سپس اشیایی که باید به صورت سراسری در برنامه وجود داشته باشند را در ابتدای کلاس تعریف کنیم. همچنین یک رشته ی ثابت تعریف کرده و دستور SQL ای که می خواهیم در طول برنامه به کار ببریم را در آن قرار می دهیم. بنابراین کد زیر را به ابتدای کلاس Form1 اضافه کنید:

```
public partial class Form1 : Form
{
    // Constant strings
    private const string _CommandText =
        "SELECT authors.au_id, au_lname, au_fname, " +
        "titles.title_id, title, price " +
        "FROM authors " +
        "JOIN titleauthor ON authors.au_id = " +
        "titleauthor.au_id " +
        "JOIN titles ON titleauthor.title_id = " +
        "titles.title_id " +
        "ORDER BY au_lname, au_fname";
    private const string _ConnectionString =
            "server=localhost;database=pubs;" +
            "user id=sa;password=;";

    // Declare global objects...
    SqlConnection objConnection;
    SqlDataAdapter objDataAdapter;
    DataSet objDataSet;
    DataView objDataView;
    CurrencyManager objCurrencyManager;
```

نکته: قبل از وارد کردن قطعه کد بالا در برنامه، ConnectionString را بر اساس تنظیمات سرور بانک اطلاعاتی خود تغییر دهید. User ID و Password مربوط به اکانت کاربری خود را وارد کرده و همچنین اگر سرور روی کامپیوتر دیگری قرار دارد، به جای استفاده از localhost نام کامپیوتر سرور در شبکه را وارد کنید.

۱۷) کد درون متد سازنده ی فرم را به صورت زیر تغییر دهید:

```
public Form1()
```

[1] برای جا به جا شدن بین قسمت طراحی و قسمت کد نویسی در یک فرم، می توانید از کلید F7 استفاده کنید.

```
{
     objConnection = new SqlConnection(_ConnectionString);
     objDataAdapter = new SqlDataAdapter(
                    _CommandText, objConnection);
     InitializeComponent();
}
```

۱۸) اولین زیر برنامه ای که باید ایجاد کنیم، زیر برنامه ای به نام FillDataSetAndView است. این زیر برنامه به همراه چند زیر برنامه ی دیگر در ابتدای برنامه فراخوانی می شوند. کد زیر را به کلاس Form1، بعد از تعریف متغییر ها اضافه کنید:

```
private void FillDataSetAndView()
{
     // Initialize a new instance of the DataSet object...
     objDataSet = new DataSet();

     // Fill the DataSet object with data...
     objDataAdapter.Fill(objDataSet, "authors");

     // Set the DataView object to the DataSet object...
     objDataView = new DataView(
                     objDataSet.Tables["authors"]);

     // Set our CurrencyManager object
     // to the DataView object...
     objCurrencyManager = (CurrencyManager)(
                this.BindingContext[objDataView]);
}
```

۱۹) در این قسمت باید زیر برنامه ای به فرم اضافه کنیم تا کنترلهای موجود در فرم را به فیلد های مربوط به آنها در DataView اضافه کند:

```
private void BindFields()
{
     // Clear any previous bindings...
     txtLastName.DataBindings.Clear();
     txtFirstName.DataBindings.Clear();
     txtBookTitle.DataBindings.Clear();
     txtPrice.DataBindings.Clear();

     // Add new bindings to the DataView object...
     txtLastName.DataBindings.Add("Text",
                          objDataView, "au_lname");
     txtFirstName.DataBindings.Add("Text",
                          objDataView, "au_fname");
```

```
     txtBookTitle.DataBindings.Add("Text",
                          objDataView, "title");
     txtPrice.DataBindings.Add("Text",
                          objDataView, "price");

     // Display a ready status...
     ToolStripStatusLabel1.Text = "Ready";
}
```

۲۰) سپس زیر برنامه ای به کلاس اضافه می کنیم که موقعیت رکورد جاری را در فرم برنامه نمایش دهد:

```
private void ShowPosition()
{
     // Always format the number
     // in the txtPrice field to include cents
     try
     {
          txtPrice.Text =
          Decimal.Parse(txtPrice.Text).ToString("##0.00");
     }
     catch(System.Exception e)
     {
          txtPrice.Text = "0";
          txtPrice.Text =
          Decimal.Parse(txtPrice.Text).ToString("##0.00");
     }

     // Display the current position
     // and the number of records
     txtRecordPosition.Text =
               (objCurrencyManager.Position + 1) +
               " of " + objCurrencyManager.Count;
}
```

۲۱) تا اینجا زیر برنامه ها ی لازم را به برنامه اضافه کرده ایم، اما در هیچ قسمت از کد از این زیر برنامه های ایجاد شده استفاده نکرده ایم. این زیر برنامه ها لازم است که قبل از نمایش داده شدن فرم و هنگام لود شدن آن فراخوانی شوند. بنابراین به قسمت طراحی فرم بروید و روی قسمت خالی از فرم دو بار کلیک کنید تا متد مربوط به رویداد Load فرم ایجاد شود (دقت کنید که باید در یک قسمت خالی از فرم دو بار کلیک کنید، نه در قسمتی خالی از کنترل GroupBox). سپس کد مشخص شده در زیر را به این متد اضافه کنید:

```
private void Form1_Load(object sender, EventArgs e)
{
     // Add items to the combo box...
     cboField.Items.Add("Last Name");
     cboField.Items.Add("First Name");
```

```
        cboField.Items.Add("Book Title");
        cboField.Items.Add("Price");

        // Make the first item selected...
        cboField.SelectedIndex = 0;

        // Fill the DataSet and bind the fields...
        FillDataSetAndView();
        BindFields();

        // Show the current record position...
        ShowPosition();
}
```

۲۲) حال باید کد کلیدهای مربوط به حرکت بین رکورد ها را در برنامه وارد کنیم. برای این کار لازم است که چهار بار به قسمت طراحی فرم بروید و روی هر کدام از دکمه های btnMoveFirst، btnMoveLast، btnMovePrevious و btnMoveNext کلیک کنید تا متد مربوط به رویداد کلیک هر یک از آنها ایجاد شود. کد مشخص شده در زیر را به متد مربوط به رویداد Click کنترل btnMoveFirst اضافه کنید:

```
private void btnMoveFirst_Click(object sender, EventArgs e)
{
        // Set the record position to the first record...
        objCurrencyManager.Position = 0;

        // Show the current record position...
        ShowPosition();
}
```

۲۳) کد زیر را به متد مربوط به رویداد Click کنترل btnMovePrevious اضافه کنید:

```
private void btnMovePrevious_Click(object sender,
                                   EventArgs e)
{
        // Move to the previous record...
        objCurrencyManager.Position -= 1;

        // Show the current record position...
        ShowPosition();
}
```

۲۴) کد زیر را به متد btnMoveNext_Click اضافه کنید:

```
private void btnMoveNext_Click(object sender, EventArgs e)
{
        // Move to the next record...
```

```
        objCurrencyManager.Position += 1;

        //Show the current record position...
        ShowPosition();
}
```

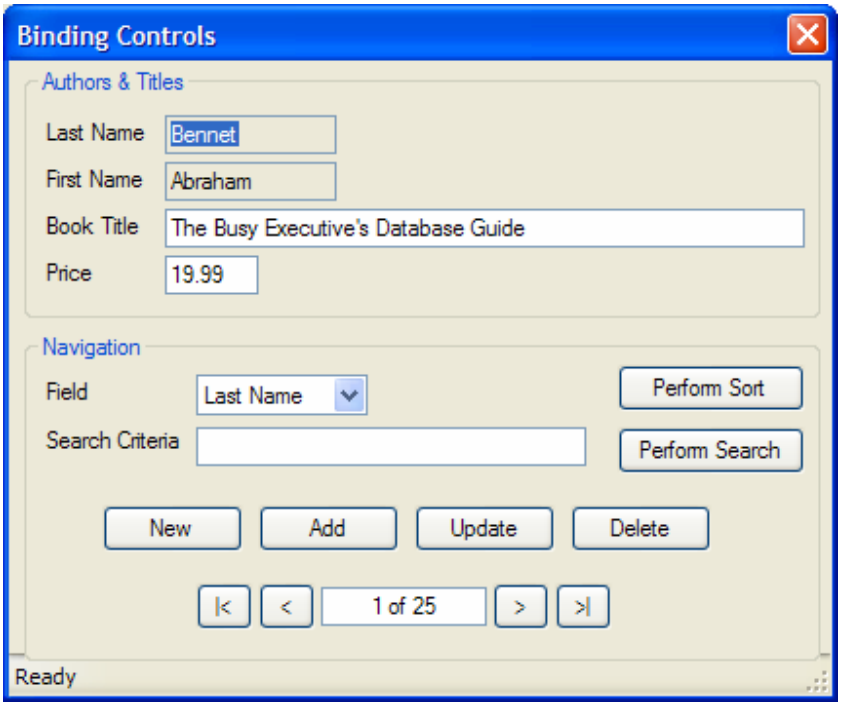
۲۵) در آخر نیز برای تکمیل این قسمت لازم است که کد زیر را در متد `btnMoveLast_Click` قرار دهید:

```
private void btnMoveLast_Click(object sender, EventArgs e)
{
        // Set the record position to the last record...
        objCurrencyManager.Position =
                        objCurrencyManager.Count - 1;

        // Show the current record position...
        ShowPosition();
}
```

۲۶) تا این قسمت کد زیادی را در برنامه وارد کرده ایم و احتمالا مشتاق هستید که نتیجه ی آن را مشاهده کنید. برنامه را اجرا کنید. مشاهده خواهید کرد که کنترلهای فرم هر یک به فیلد مربوط به خود در `DataView` متصل شده اند. روی کلیدهای مربوط به ابتدا و یا انتهای رکورد ها کلیک کنید تا نحوه ی عملکرد کلاس `CurrencyManager` برای ایجاد هماهنگی بین رکوردی که کنترل ها نمایش می دهند را مشاهده کنید.

با اجرای برنامه باید فرمی را مشابه شکل ۱۶-۷ مشاهده کنید. تا اینجا در فرم برنامه فقط کلیدهای مربوط به جا به جا شدن بین رکورد ها عمل می کنند. با کلیک کردن روی هر یک از کلیدهای بعدی و قبلی و یا روی کلیدهای مربوط به ابتدا و انتها، بین رکورد های موجود در فرم جا به جا شوید. مشاهده خواهید کرد هر بار که با استفاده از این کلید ها رکورد جاری را تغییر دهید، عدد نمایش داده شده در کادر در فرم برنامه نیز تغییر کرده و شماره رکورد جاری را نمایش می دهد.



شکل ۱۶-۷

اگر در رکورد ابتدا باشید، می توانید روی کلید مربوط به رکورد قبلی کلیک کنید، اما هیچ اتفاقی رخ نخواهد داد زیرا هم اکنون در رکورد قبلی هستید. همچنین می توانید به آخرین رکورد بروید و روی کلید مربوط به رکورد بعدی کلیک کنید. اما باز هم هیچ تغییری را مشاهده نخواهید کرد، زیرا در آخرین رکورد هستید.
همچنین اگر ماوس را روی هر یک از این دکمه ها ببرید، توضیحی را مشاهده خواهید کرد که عملکرد کلید را توضیح می دهد. این مورد فقط برای ایجاد رابط کاربری بهتر اضافه شده است.

نکته: قسمتهای مربوط به مدیریت خطاها و استثنا های احتمالی از کد این قسمت حذف شده اند تا مکان کمتری گرفته شود. در هنگام وارد کردن این کد بهتر است که این قسمت را نیز اضافه کنید. برای مرور اطلاعات مربوط به کنترل و مدیریت خطاها و استثنا ها می توانید به فصل یازدهم رجوع کنید.

چگونه کار می کند – فضای نامها و تعاریف

مانند قبل ابتدا فضای نامهای `System.Data` و `System.Data.SqlClient` را به برنامه اضافه می کنیم. سپس یک ثابت رشته ای ایجاد کرده و دستور SQL ای که می خواهیم برای دریافت داده ها از بانک اطلاعاتی، از آن استفاده کنیم را در این ثابت قرار می دهیم. همچنین ثابت رشته ای دیگری تعریف کرده و متن مربوط به اتصال به بانک اطلاعاتی را در آن قرار می دهیم. به این ترتیب در طول برنامه اگر بخواهیم به سرور دیگری متصل شویم و یا اطلاعات مربوط به اکانتی که برای اتصال به بانک از آن استفاده می کنیم را تغییر دهیم، فقط لازم است که تغییرات را در این ثابت رشته ای وارد کنیم. بعد از این کار به تعریف اشیای مورد نیاز در برنامه می پردازیم. با سه شیئ اول که از قسمت های قبل آشنا هستید و نیازی به توضیح ندارند.

موارد استفاده از دو شیئ آخر هم در قسمتهای قبلی توضیح داده شده اند. از شیئ DataView برای تغییر نحوه ی نمـایش داده هایی که از بانک اطلاعاتی دریافت و در DataSet نگهداری شده اند استفاده می کنیم. شیئ CurrencyManager نیز برای ایجاد هماهنگی در نمایش داده های یک رکورد به وسیلهی چند کنترل ساده به کار می رود.

```
// Constant strings
private const string _CommandText =
     "SELECT authors.au_id, au_lname, au_fname, " +
     "titles.title_id, title, price " +
     "FROM authors " +
     "JOIN titleauthor ON authors.au_id = " +
     "titleauthor.au_id " +
     "JOIN titles ON titleauthor.title_id = " +
     "titles.title_id " +
     "ORDER BY au_lname, au_fname";
private const string _ConnectionString =
          "server=localhost;database=pubs;" +
          "user id=sa;password=;";

// Declare global objects...
SqlConnection objConnection;
SqlDataAdapter objDataAdapter;
DataSet objDataSet;
DataView objDataView;
CurrencyManager objCurrencyManager;
```

اما همانطور که می دانید تا اینجا فقط متغیرهایی برای نگهداری این اشیا ایجاد کرده ایم و خود اشیا هنوز ایجاد نشده اند. از این پـنج متغیری که در این مرحله تعریف کرده ایم، دو متغییر اول را باید در ابتدای برنامه نمونه سازی کنیم تا بتوانیم آنها را از ابتدا در برنامـه به کار ببریم. بنابراین در متد سازنده ی کلاس Form1، با استفاده از ثابت های رشته ای که ایجاد کـرده بـودیم و بـه وسـیله ی دستور new، دو نمونه از کلاس SqlDataAdapter و SqlConnection ایجاد کرده و در این متغییر ها قرار می دهیم.

نمونه سازی شیئ objConnection که همانند برنامه ی قبل است و نیازی به توضیح نـدارد. امـا بهتـر اسـت کـه نمونـه سازی شیئ objDataAdapter را دقیق تر بررسی کنیم. هنگام نمونه سـازی ایـن شـیئ، یـک مـتن بـه عنـوان دسـتور SQLای که می خواهیم مورد استفاده قرار دهیم و نیز یک شیئ از نوع SqlConnection را به متد سازنده ی این کلاس ارسال کردیم. به این وسیله، متد سازنده ی کلاس SqlDataAdapter بـا اسـتفاده از دسـتور SQL، یـک شـیئ از نـوع SqlCommand ایجـاد کـرده و آن را در خاصـیت SelectCommand قـرار مـی دهـد. همچنـین شـیئ SqlConnection ای که به آن فرستاده ایم را نیز برای اتصال به بانک اطلاعاتی استفاده می کند. به ایـن ترتیـب نیـازی نیست که این اشیا را خودمان ایجاد کرده و در خاصیتهای مربوط به آنها قرار دهیم.

```
public Form1()
{
     objConnection = new SqlConnection(_ConnectionString);
     objDataAdapter = new SqlDataAdapter(
                    _CommandText, objConnection);
```

```
        InitializeComponent();
}
```

نکته: میدانید که هر فرم، خود نیز یک کلاس است و مانند هر کلاس دیگری دارای یک متد سازنده است که هنگام ایجاد یک شیئ از آن فراخوانی می شود. متد سازنده ی `Form1` نیز که در این قسمت مشاهده می کنید، هر زمان که نمونه ی جدیدی از این فرم ایجاد شود فراخوانی خواهد شد. بنابراین هر کدی که بخواهیم در ابتدای ایجاد یک فرم، زمانی که فرم در حال نمونه سازی شدن است اجرا شود را باید در این متد وارد کنیم. این متد به صورت پیش فرض دارای فراخوانی متدی به نام `InitializeComponent` است که در فایل `Form1.Designer.cs` قرار دارد (`Form1` نام فرم برنامه است و ممکن است تغییر کند). این متد در همان ابتدای نمونه سازی شدن فرم فراخوانی شده و وظیفه دارد که کنترل ها و اشیای موجود در فرم را ایجاد کرده و خاصیتهای آنها را تنظیم کند[1].

دستور `SELECT` ای که در این قسمت استفاده کرده ایم، در اصل همان دستوری است که در برنامه ی قبل به کار برده ایم، با این تفاوت که در این قسمت نام چند ستون دیگر را نیز در مقابل عبارت `SELECT` اضافه کرده ایم تا اطلاعات موجود در این ستون ها نیز از بانک اطلاعاتی دریافت شوند.
قبل از ستون `au_id` نام جدول `authors` را آورده ایم، زیرا این ستون در جدول `authortitle` هم وجود دارد. بنابراین باید برای بانک اطلاعاتی مشخص کنیم که می خواهیم این ستون از کدامیک از این دو جدول استخراج شود. همین روش را برای `title_id` نیز به کار برده ایم، زیرا این ستون نیز در جدول های `titles` و `authortitle` وجود دارد.

چگونه کار می کند – `FillDataSetAndView`

اولین زیر برنامه ای که در این برنامه ایجاد می کنیم، زیر برنامه ی `FillDataSetAndView` است. این زیر برنامه در چندین قسمت از برنامه فراخوانی شده و وظیفه دارد که اطلاعات را از بانک اطلاعاتی دریافت کرده و در `DataSet` قرار دهد.
در ابتدای این زیر برنامه، یک نمونه از شیئ `DataSet` را ایجاد کرده و آن را در متغییر `objDataSet` قرار می دهیم. این کار را به این دلیل در متد سازنده ی کلاس انجام ندادیم، زیرا ممکن است این متد در چندین قسمت از برنامه فراخوانی شود و می خواهیم هر بار که این متد فراخوانی شد، `DataSet` قبلی از بین برود و اطلاعاتی که از بانک اطلاعاتی به دست می آید در یک `DataSet` جدید قرار بگیرد:

```
private void FillDataSetAndView()
{
        // Initialize a new instance of the DataSet object...
        objDataSet = new DataSet();
```

سپس متد `Fill` از کلاس `SqlDataAdapter` را فراخوانی می کنیم تا داده ها را از بانک اطلاعاتی دریافت کرده و در شیئ `objDataSet` قرار دهد. بعد از آن نیز شیئ `DataView` را به گونه ای ایجاد می کنیم تا بتواند داده های موجود در جدول `authors` از `DataSet` را نمایش دهد. به این ترتیب می توانیم به سادگی بین داده های موجود در این جدول حرکت کرده، جستجو کنیم و یا ترتیب قرار گرفتن آنها را تغییر داده و آنها به صورت دلخواه مرتب کنیم.

[1] نحوه ی تنظیم خاصیتهای کنترل ها در این قسمت، بر اساس تغییراتی که با استفاده از پنجره ی `Properties` ایجاد می کنیم مشخص می شود.

```
        // Fill the DataSet object with data...
        objDataAdapter.Fill(objDataSet, "authors");

        // Set the DataView object to the DataSet object...
        objDataView = new DataView(
                        objDataSet.Tables["authors"]);
```

بعد از مقدار دهی اولیه به شیئ objDataView، باید شیئ objCurrencyManager را مقدار دهی کنیم. به خاطر دارید که شیئ BindingContext به صورت درونی در هر فرم ویندوزی وجود داشته و شامل یک مجموعه از اشیا از نوع CurrencyManager است که هر یک برای یکی از منابع داده ای موجود در فرم ایجاد شده است. بنابراین در این قسمت از میان آنها شیئ CurrencyManager مربوط به منبع داده ای objDataSet را انتخاب می کنیم:

```
        // Set our CurrencyManager object
        // to the DataView object...
        objCurrencyManager = (CurrencyManager)(
                    this.BindingContext[objDataView]);
}
```

چگونه کار می کند – BindFields

زیر برنامه ی بعدی که ایجاد می کنیم (زیر برنامه ی BindFields) برای اتصال کنترلهای ساده ی موجود در فرم به فیلد های موجود در DataView به کار می رود. این زیر برنامه ابتدا تمام اتصالات موجود برای هر کنترل را از بین می برد، سپس آنها را به فیلد های مربوط به آن در شیئ objDataView متصل می کند.
برای حذف اتصال های موجود در کنترل هایی که در فرم برنامه قرار دارند، کافی است متد Clear از خاصیت DataBindings (که شامل شیئ ای از نوع ControlBindingsColllection است) را فراخوانی کنیم.

```
private void BindFields()
{
        // Clear any previous bindings...
        txtLastName.DataBindings.Clear();
        txtFirstName.DataBindings.Clear();
        txtBookTitle.DataBindings.Clear();
        txtPrice.DataBindings.Clear();
```

بعد از این کار می توانی مجدداً کنترل ها را به فیلد های مورد نظر در منبع داده ای خود یعنی DataView متصل کنیم. برای این کار نیز از متد Add در شیئ ای از کلاس ControlBindingsCollection، که به وسیله ی DataBindings قابل دسترسی است استفاده می کنیم. همانطور که در قسمت قبل نیز گفتم، متد Add سه پارامتر دریافت می کند:

- پارامتر اول `propertyName` است و شامل نام خاصیتی است که می خواهیم به فیلدی از بانک اطلاعاتی متصل شود. در این قسمت برای اینکه می خواهیم خاصیت `Text` از این کنترل ها، مقدار خود را از فیلد های موجود در `DataView` دریافت کنند، عبارت `"Text"` را در این پارامتر قرار می دهیم.
- پارامتر بعدی `dataSource` است و نام منبع داده ای که می خواهیم به آن متصل شویم را تعیین می کند. توجه داشته باشید که این پارامتر می تواند هر شیئ ای که حاوی داده است را دریافت کند، مانند `DataSet`، `DataView` و یا `DataTable`. برای این مثال یک `DataView` را به عنوان منبع داده ای تعیین می کنیم.
- پارامتر آخر نیز `dataMember` است. این پارامتر حاوی نام یک فیلد در منبع داده ای مشخص شده است که می خواهیم خاصیت مورد نظر از کنترل به آن فیلد متصل شود. در این قسمت می توانید نام یکی از ستونهای اطلاعاتی که با استفاده از دستور `SELECT` از بانک اطلاعاتی استخراج کردید را به کار ببرید.

```
        // Add new bindings to the DataView object...
        txtLastName.DataBindings.Add("Text",
                                objDataView, "au_lname");
        txtFirstName.DataBindings.Add("Text",
                                objDataView, "au_fname");
        txtBookTitle.DataBindings.Add("Text",
                                objDataView, "title");
        txtPrice.DataBindings.Add("Text",
                                objDataView, "price");
```

در انتهای این زیر برنامه نیز پیغامی را در کنترل `Label` موجود در نوار وضعیت پایین فرم نمایش می دهیم تا به کاربر اطلاع دهیم که برنامه آماده شده است:

```
        // Display a ready status...
        ToolStripStatusLabel1.Text = "Ready";
}
```

چگونه کار می کند – `ShowPosition`

شیئ `objCurrencyManager` مسئول نگهداری موقعیت رکورد جاری در `objDataView` است. بنابراین در این متد باید از این شیئ استفاده کرده و به کاربر اطلاع دهیم که در حال مشاهده ی اطلاعات مربوط به کدام ردیف از داده های موجود در بانک اطلاعاتی است.

اما قبل از انجام این کار، بهتر است نحوه ی نمایش عددی که در کادر متنی متصل به ستون `price` وجود دارد را تغییر دهیم. این عدد نمایش دهنده ی قیمت کتابی است که هم اکنون در حال مشاهده ی اطلاعات آن هستیم و بر حسب دلار می باشد. بنابراین اگر قیمت کتابی برای مثال ۴۰ دلار باشد، عدد `40` در کادر نمایش داده می شود. اما بهتر است که بعد از این عدد، دو رقم اعشار نیز نمایش داده شوند تا قیمت یک کتاب بهتر مشخص شود.

برای تغییر در نحوه ی نمایش این عدد، کافی است که متن موجود در کادر `Price` را با استفاده از متد `Parse` از کلاس `Decimal` به یک عدد تبدیل کرده، سپس این عدد را با استفاده از متد `ToString` و مشخص کردن نحوه ی فرمت این عدد، آن را به متن تبدیل کنید و مجدداً در کادر `price` قرار دهید. البته اگر کادر `Price` خالی باشد، یعنی قیمتی برای آن

کتاب مشخص نشده باشد، خطایی در این قسمت رخ می دهد که موجب می شود اجرای برنامه متوقف شود. در این قسمت می توانیم از یک بلاک try...catch استفاده کرده تا در صورتی که خطایی به وجود آمد (یعنی اگر عددی در کادر price وجود نداشت) عدد صفر قالب بندی شود و در کادر قرار بگیرد.

```
private void ShowPosition()
{
      // Always format the number
      // in the txtPrice field to include cents
      try
      {
            txtPrice.Text =
            Decimal.Parse(txtPrice.Text).ToString("##0.00");
      }
      catch(System.Exception e)
      {
            txtPrice.Text = "0";
            txtPrice.Text =
            Decimal.Parse(txtPrice.Text).ToString("##0.00");
      }

      // Display the current position
      // and the number of records
      txtRecordPosition.Text =
                  (objCurrencyManager.Position + 1) +
                  " of " + objCurrencyManager.Count;
}
```

خط آخر این متد نیز موقعیت رکورد جاری به همراه تعداد رکورد هایی که در برنامه وجود دارند را نمایش می دهد. با استفاده از خاصیت Position می توانیم مکان رکورد جاری را بدست آوریم. اما همانطور که می دانید عددی که از این خاصیت دریافت می کنیم، اندیس رکورد جاری از شماره ی صفر است. بنابراین باید آن را با عدد یک جمع کنیم تا موقعیت واقعی رکورد جاری را بدست آوریم.
برای بدست آوردن تعداد رکورد ها نیز از خاصیت Count استفاده می کنیم که تعداد رکورد های موجود در برنامه را نمایش می دهد.

چگونه کار می کند – Form_Load

بعد از اتمام بررسی زیر برنامه های موجود در کلاس Form1، بهتر است متدی که در آن از زیر برنامه های قبلی استفاده شده است را بررسی کنیم.
در فرم برنامه یک کنترل ComboBox وجود دارد که هنگام جستجو و یا مرتب کردن داده ها مورد استفاده قرار می گیرد. این ComboBox باید با استفاده از نام ستونهای داده ای که در DataView قرار دارند تکمیل شود. بنابراین با استفاده از متد

Add از خاصیت Items این کنترل، نام ستونهای موجود را به آن اضافه می کنیم. همچنین نام ستونها را به همان ترتیبی که در DataView قرار گرفته اند به لیست اضافه خواهیم کرد:

```
private void Form1_Load(object sender, EventArgs e)
{
      // Add items to the combo box...
      cboField.Items.Add("Last Name");
      cboField.Items.Add("First Name");
      cboField.Items.Add("Book Title");
      cboField.Items.Add("Price");
```

بعد از اضافه کردن تمام آیتم های مورد نیاز به این کنترل، می خواهیم که اولین آیتمی که اضافه کردیم به صورت انتخاب شده قرار بگیرد. بنابراین خاصیت SelectedIndex را برابر با صفر قرار می دهیم (زیرا در این قسمت نیز اندیس آیتم ها با صفر شروع می شود)

```
      // Make the first item selected...
      cboField.SelectedIndex = 0;
```

سپس با فراخوانی متد FillDataSetAndView داده ها را از بانک اطلاعاتی بدست آورده و در کنترل DataSet در برنامه قرار می دهیم. همچنین کنترل DataView را نیز تنظیم می کنیم تا بتوانیم از آن استفاده کنیم. بعد از آن نیز متد BindFields را فراخوانی می کنیم تا کنترل های موجود در فرم را به فیلد های داده ای موجود در DataView متصل کند. در آخر نیز با فراخوانی متد ShowPosition موقعیت رکورد جاری و نیز تعداد رکورد های موجود در برنامه را در فرم نمایش می دهیم:

```
      // Fill the DataSet and bind the fields...
      FillDataSetAndView();
      BindFields();

      // Show the current record position...
      ShowPosition();
}
```

چگونه کار می کند – جا به جا شدن بین رکورد ها

زیر برنامه ی مربوط به رویداد Click کنترل btnMoveFirst باعث می شود که اولین رکورد از داده ها در فرم نمایش داده شود. برای این کار از خاصیت Position در کلاس CurrencyManager استفاده می کنیم. به عبارت دیگر با قرار دادن مقدار صفر در خاصیت Position مشخص می کنیم که CurrencyManager باید به اولین رکورد موجود حرکت کند:

```
      // Set the record position to the first record...
      objCurrencyManager.Position = 0;
```

به علت اینکه کنترلهای موجود در فرم برنامه به DataView متصل شده اند، بنابراین با تغییر موقعیت رکورد جاری به اولین رکورد، تمامی کنترل های موجود در فرم اطلاعات مربوط به رکورد اول را نمایش خواهند داد.
بعد از تعیین موقعیت رکورد جاری در برنامه، باید متد ShowPosition را فراخوانی کنیم تا اطلاعات مربوط به موقعیت رکورد جاری در فرم به درستی نمایش داده شوند:

```
// Show the current record position...
ShowPosition();
```

سپس متد مربوط به کنترل btnMovePrevious را می نویسیم. در این متد باید به رکورد قبلی حرکت کنیم و برای این کار نیز باید یک واحد از خاصیت Position کم کنیم. شیئ CurrencyManager متوجه این تغییر خواهد شد و در نتیجه تمام کنترلهای موجود در فرم اطلاعات مربوط به رکورد قبلی را نمایش می دهند.

```
// Move to the previous record...
objCurrencyManager.Position -= 1;
```

مجدداً در این قسمت نیز بعد از تغییر رکورد جاری، باید متد ShowPosition را فراخوانی کنیم تا اطلاعات نمایش داده شده در فرم برنامه تصحیح شوند.
در متد مربوط به کنترل btnMoveNext نیز کافی است که شماره ی رکورد جاری را که در خاصیت Position قرار دارد یک واحد افزایش دهیم. به این ترتیب CurrencyManager اطلاعات مربوط به رکورد بعدی را در کادرهای موجود در فرم نمایش خواهد داد.

```
// Move to the next record...
objCurrencyManager.Position += 1;
```

در آخر نیز باید کدی را متد مربوط به کلید btnMoveLast قرار دهیم تا زمانی که کاربر روی این دکمه کلیک کرد، به آخرین رکورد داده ها منتقل شود. برای این کار باید خاصیت Position را برابر با اندیس آخرین رکورد موجود، یعنی تعداد رکورد ها منهای یک قرار دهیم و سپس متد ShowPosition را فراخوانی کنیم تا اطلاعات موجود در فرم را تصحیح کند:

```
// Set the record position to the last record...
objCurrencyManager.Position =
                objCurrencyManager.Count - 1;

// Show the current record position...
ShowPosition();
```

نکته: در این قسمت نیز، همانند هنگامی که در اولین رکورد داده ها بودیم با کلیک بر روی دکمه ی بعدی اتفاق خاصی رخ نخواهد داد. زیرا بعد از آخرین رکورد داده ها دیگر رکوردی وجود ندارد که به آن برویم. در این شرایط می توانید با تنظیم خاصیت Enabled دکمه ی btnMoveNext و یا btnMovePrevious به مقدار False، از کلیک کردن روی آنها جلوگیری کنید. به این ترتیب برنامه ظاهر بهتری پیدا خواهد کرد. می توانید این مورد را خودتان به برنامه اضافه کنید.

بعد از اتمام این متد، این بخش از برنامه به پایان می رسد. در بخش امتحان کنید بعد به موارد مربوط به مرتب سازی داده ها خواهیم پرداخت.

امتحان کنید: اضافه کردن قابلیت مرتب سازی به برنامه

۱) به قسمت طراحی فرم مربوط به `Form1` بروید و روی دکمه ی `Perform Sort` دو بار کلیک کنید تا متد مربوط به رویداد `Click` این کنترل ایجاد شود. سپس کد مشخص شده در زیر را به این متد اضافه کنید:

```
private void btnPerformSort_Click(object sender,
                                  EventArgs e)
{
     // Determine the appropriate item selected and set the
     // Sort property of the DataView object...
     switch(cboField.SelectedIndex)
     {
          case 0: // Last Name
               objDataView.Sort = "au_lname";
               break;
          case 1: // First Name
               objDataView.Sort = "au_fname";
               break;
          case 2: // Book Title
               objDataView.Sort = "title";
               break;
          case 3: // Price
               objDataView.Sort = "price";
               break;
     }

     // Call the click event for the MoveFirst button...
     btnMoveFirst_Click(null, null);

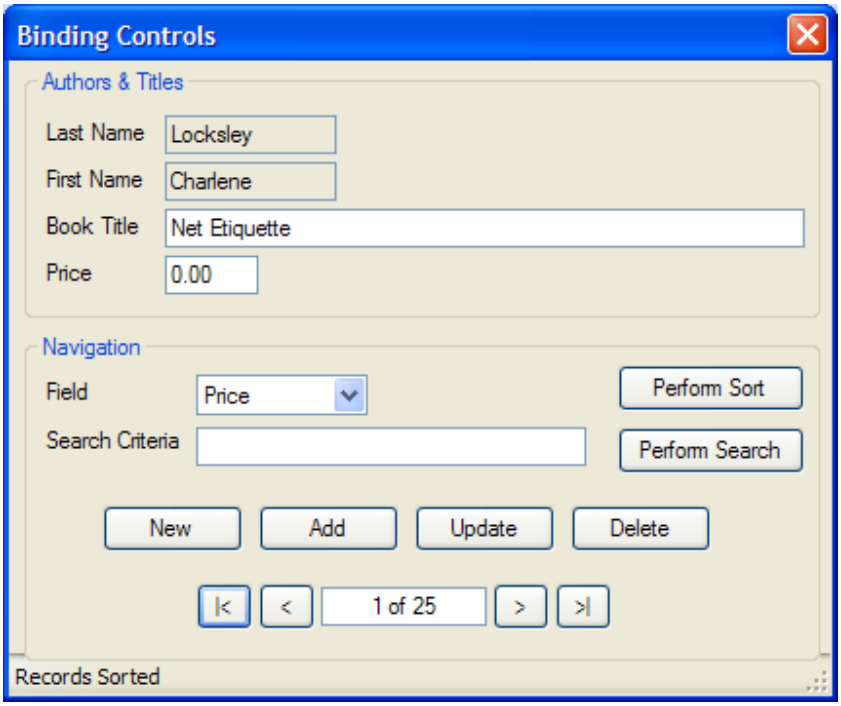
     // Display a message
     // that the records have been sorted...
     ToolStripStatusLabel1.Text = "Records Sorted";
}
```

۲) برنامه را اجرا کنید تا قابلیتی را که در این قسمت به برنامه اضافه کردیم را امتحان کنیم. در کنترل `ComboBox` موجود در فرم یک ستون را انتخاب کرده و سپس روی دکمه ی `Perform Sort` کلیک کنید تا داده ها بر اساس آن ستون مرتب شوند. شکل ۱۶-۸ فرم برنامه را در حالتی نمایش می دهد که داده های موجود در آن بر اساس ستون `Price` مرتب شده اند:

چگونه کار می کند؟

اولین کاری که در این قسمت انجام می دهیم این است که با توجه به مقدار انتخاب شده در کنترل cboField مشخص کنیم داده ها باید بر اساس کدام ستون مرتب شوند.

```
// Determine the appropriate item selected and set the
// Sort property of the DataView object...
switch(cboField.SelectedIndex)
{
      case 0: // Last Name
            objDataView.Sort = "au_lname";
            break;
      case 1: // First Name
            objDataView.Sort = "au_fname";
            break;
      case 2: // Book Title
            objDataView.Sort = "title";
            break;
      case 3: // Price
            objDataView.Sort = "price";
            break;
}
```



شکل ۱۶-۸

در اینجـا بـا اسـتفاده از دسـتور switch مـی تـوانیم حالتهـای مختلـف مقـدار خاصـیت SelectedIndex از ComboBox را بررسی کرده و به این وسیله مشخص کنیم که کاربر می خواهد داده ها را بر اساس چه ستونی مرتب کند. بعد از مشخص شدن نام ستون مورد نظر می توانیم خاصیت Sort را برابر با نام آن ستون قرار دهیم تا داده ها مرتب شوند.
بعد از مرتب شدن داده ها بهتر است که به اولین رکورد داده ها برویم. برای این کار چندین روش وجود دارد. روش اول این است که خاصیت Position را برابر با صفر قرار داده و سپس متد ShowPosition را فراخوانی کنیم، در واقع همان کـاری را انجام دهیم که در متد btnMoveFirst_Click انجام داده ایم. پس به جای استفاده از این روش بهتر است کـه متـد btnMoveFirst_Click را فراخوانی کرده و به جای پارامترهای آن مقدار null را ارسال کنـیم. انجـام ایـن کـار دقیقاً مشابه این است که کاربر روی دکمه ی btnMoveFirst کلیک کرده باشد.
متـد btnMoveFirst_Click دو پـارامتر دریافـت مـی کنـد، یکـی از نـوع object و دیگـری از نـوع EventArgs. هنگام فراخوانی متد باید این دو پارامتر را به آن ارسال کنیم. برای این کار مـی تـوانیم دو شـیئ، یکـی از نـوع EventArgs و دیگـری از نـوع Object ایجـاد کـرده و آنهـا را بـه متـد ارسـال کنـیم. امـا همـانطور کـه در کـد متـد btnMoveFirst مشاهده می کنید از این دو پارامتر هیچ استفاده ای نکرده ایم، بنابراین لزومی ندارد که مقداری حافظـه را اختصاص دهیم تا دو شیئ ایجاد کرده و آنها را به متد بفرستیم. بلکه می توانیم به جای این دو شیئ، دو مقـدار null را بـه متـد ارسال کنیم. البته لازم است به این نکته دقت کنید که استفاده از این روش هنگامی که از این پارامتر ها در متد استفاده شـده باشـد، باعث ایجاد خطایی از نوع System.NullReferenceException می شود.

```
// Call the click event for the MoveFirst button...
btnMoveFirst_Click(null, null);
```

بعد از اینکه به رکورد اول رفتیم لازم است که متنی را در نوار وضعیت نمایش دهیم تا مشخص شود که داده ها مرتب شده اند. برای این کار از خاصیت Text در کنترل ToolStripStatusLabel1 استفاده می کنیم.

```
// Display a message
// that the records have been sorted...
ToolStripStatusLabel1.Text = "Records Sorted";
```

در بخش امتحان کنید بعد، قابلیت جستجو کردن را نیز به برنامه اضافه خواهیم کرد.

امتحان کنید: اضافه کردن قابلیت جستجو به برنامه

۱) به قسمت طراحی فرم بروید و روی دکمه ی Perform Search دو بار کلیک کنید تا متد مربـوط بـه رویـداد Click آن ایجاد شود. سپس کد مشخص شده در زیر را در این متد وارد کنید:

```
private void btnPerformSearch_Click(object sender,
                                    EventArgs e)
{
    // Declare local variables...
    int intPosition;
```

```
     // Determine the appropriate item selected and set the
     // Sort property of the DataView object...
     switch(cboField.SelectedIndex)
     {
          case 0: // Last Name
               objDataView.Sort = "au_lname";
               break;
          case 1: // First Name
               objDataView.Sort = "au_fname";
               break;
          case 2: // Book Title
               objDataView.Sort = "title";
               break;
          case 3: // Price
               objDataView.Sort = "price";
               break;
     }

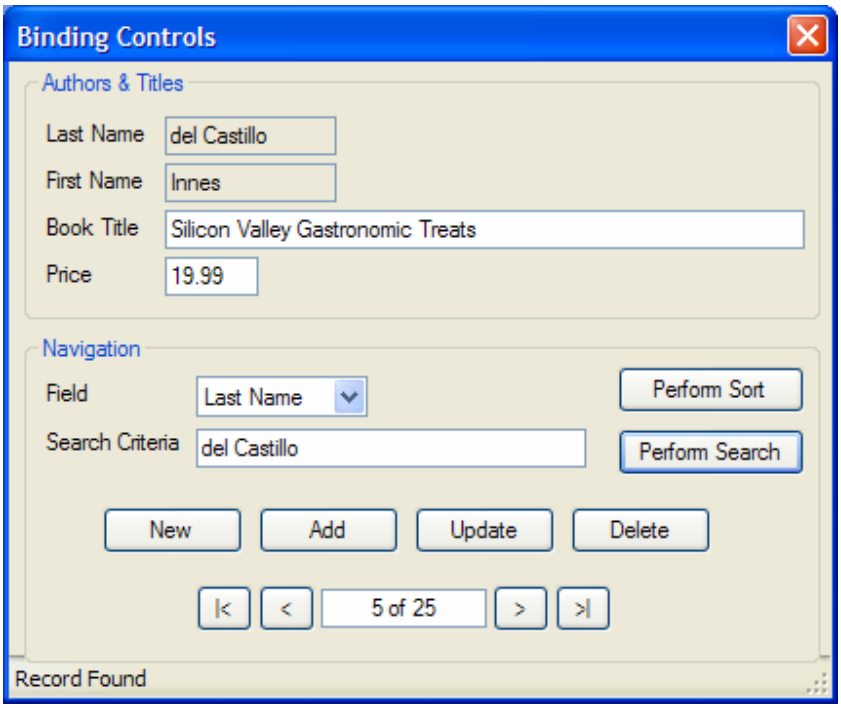
     // If the search field is not price then...
     if (cboField.SelectedIndex < 3)
     {
          // Find the last name, first name, or title...
          intPosition =
               objDataView.Find(txtSearchCriteria.Text);
     }
     else
     {
          // otherwise find the price...
          intPosition = objDataView.Find(
                    Decimal.Parse(txtSearchCriteria.Text));
     }
     if (intPosition == -1)
     {
          // Display a message
          // that the record was not found...
          ToolStripStatusLabel1.Text = "Record Not Found";
     }
     else
     {
          // Otherwise display a message that the record
          // was found and reposition the CurrencyManager
          // to that record...
          ToolStripStatusLabel1.Text = "Record Found";
          objCurrencyManager.Position = intPosition;
     }

     // Show the current record position...
```

```
        ShowPosition();
}
```

۲) برنامه را اجرا کنید تا قابلیت جدید آن را نیز امتحان کنیم. فیلدی که می خواهید جستجو بر اساس آن صورت گیرد را از داخل `ComboBox` انتخاب کرده و سپس عبارت مورد جستجو را در داخل فیلد `Search Criteria` وارد کنید. در آخر نیز روی دکمه ی `Perform Search` کلیک کنید.

اگر رکورد مورد نظر شما در بین داده ها پیدا شود، مشاهده خواهید کرد که اطلاعات آن رکورد در فرم نمایش داده می شود و موقعیت رکورد جاری به رکورد پیدا شده تغییر می کند. همچنین پیغامی در نوار ابزار نوشته می شود و مشخص می کند که رکورد مورد نظر پیدا شده است (شکل ۱۶-۹). همچنین اگر هیچ رکوردی پیدا نشود، متنی در نوار وضعیت نوشته خواهد شد و مشخص می کند که داده ی مورد نظر پیدا نشده است.



شکل ۱۶-۹

چگونه کار می کند؟

این قسمت ممکن است کمی پیچیده تر از قسمت قبلی به نظر برسد، زیرا برای این قسمت موارد بیشتری را باید بررسی کرده و تصمیم گیری کنیم، برای مثال شرایطی مانند زمانی که هیچ رکوردی پیدا نشود. اولین کاری که در این متد انجام می دهیم ایجاد یک متغییر از نوع `int` است تا بتوانیم مکان داده ای که پیدا شده است (و یا پیدا نشدن آن را) در آن نگهداری کنیم.

```
// Declare local variables...
int intPosition;
```

سپس باید داده های موجود در برنامه را بر اساس ستونی که می خواهیم در آن جستجو کنیم مرتب کنیم. زیرا متد Find فقط می تواند در ستونهای مرتب شده به دنبال داده ی مورد نظر بگردد. بنابراین همانند قسمت قبل با استفاده از دستور switch مشخص می کنیم که جستجو در کدام ستون می خواهد انجام شود، سپس مقدار خاصیت Sort را برابر با نام آن ستون قرار می دهیم تا داده ها بر اساس آن ستون مرتب شوند:

```
// Determine the appropriate item selected and set the
// Sort property of the DataView object...
switch(cboField.SelectedIndex)
{
     case 0: // Last Name
          objDataView.Sort = "au_lname";
          break;
     case 1: // First Name
          objDataView.Sort = "au_fname";
          break;
     case 2: // Book Title
          objDataView.Sort = "title";
          break;
     case 3: // Price
          objDataView.Sort = "price";
          break;
}
```

نکته: در این قسمت برای مرتب کردن داده ها به جای استفاده از این روش، می توانستیم متد btnPerformSort_Click را فراخوانی کنیم تا آن متد عمل مرتب سازی را انجام دهد. سپس با استفاده از متد btnPerformSort_Click به دنبال داده ی مورد نظر می گشتیم. در این صورت برای پارامترهای متد Find هم می توانستیم از مقدار null استفاده کنیم.

ستونهای FirstName، LastName و همچنین ستون Title دارای مقادیر رشته ای هستند، در صورتی که ستون Price دارای مقدار عددی است. بنابراین ابتدا باید مشخص کنیم که جستجو در کدامیک از این ستونها صورت خواهد گرفت و سپس اگر بخواهیم که در ستون Price جستجو کنیم، مقدار وارد شده در کادر txtSearchCriteria را به عدد از نوع Decimal تبدیل کنیم.
مجددا برای تشخیص این مورد نیز باید از خاصیت SelectedIndex در کنترل cboField استفاده کنیم. اگر مقدار این خاصیت از ۳ کمتر بود به این معنی است که یکی از سه ستون FirstName، LastName و یا Title انتخاب شده است.
در این شرایط متد Find را فراخوانی کرده و مقدار وارد شده در کادر txtSearchCriteria را نیز به عنوان پارامتر به آن ارسال می کنیم تا در ستون مشخص شده، به دنبال آن داده بگردد و سپس نتیجه را در متغییر intPosition قرار دهد.
اما اگر مقدار این خاصیت برابر با ۳ بود به این معنی است که جستجو باید بر اساس فیلد Price صورت گیرد. بنابراین ابتدا با استفاده از متد Parse در کلاس Decimal متن وارد شده در کادر txtSearchCriteria را به یک عدد از نوع Decimal تبدیل می کنیم. سپس آن را به عنوان پارامتر به متد Find ارسال می کنیم.

```
// If the search field is not price then...
if (cboField.SelectedIndex < 3)
{
     // Find the last name, first name, or title...
     intPosition =
          objDataView.Find(txtSearchCriteria.Text);
}
else
{
     // otherwise find the price...
     intPosition = objDataView.Find(
               Decimal.Parse(txtSearchCriteria.Text));
}
```

بعد از اینکه متد Find از کلاس DataView را اجرا کرده و نتیجه ی آن را در متغییر intPosition قرار دادیم، باید این نتیجه را بررسی کنیم تا مشخص شود داده ی مورد نظر پیدا شده است یا نه؟ اگر این متغییر حاوی مقدار -1 بود به این معنی است که داده ی مورد نظر پیدا نشده است. در غیر این صورت هر عددی به جز -1، نشان دهنده ی شماره رکورد ای است که حاوی داده ی مورد نظر است. بنابراین اگر مقدار متغییر intPosition برابر با -1 بود باید پیغامی را در نوار وضعیت نمایش دهیم که مشخص کند داده ی مورد نظر یافته نشده است. اما اگر عددی به جز -1 در intPosition قرار داشت، باید با استفاده از خاصیت Position از کلاس CurrencyManager موقعیت رکورد جاری در برنامه را به رکورد یافته شده تغییر دهیم و همچنین متنی را در نوار وضعیت نمایش دهیم که مشخص کند داده ی مورد نظر پیدا شده است.

```
if (intPosition == -1)
{
     // Display a message
     // that the record was not found...
     ToolStripStatusLabel1.Text = "Record Not Found";
}
else
{
     // Otherwise display a message that the record
     // was found and reposition the CurrencyManager
     // to that record...
     ToolStripStatusLabel1.Text = "Record Found";
     objCurrencyManager.Position = intPosition;
}
```

نکته: هنگام استفاده از متد Find دقت کنید که این متد دقیقاً عبارتی را که به آن فرستاده شده است جستجو می کند و فقط زمانی شماره رکورد داده ای برمی گرداند که آن داده برابر با عبارتی باشد که به متد فرستاده شده است. البته این متد بزرگی و یا کوچکی حروف را در نظر نمی گیرد. برای مثال عبارت Ann و ANN از نظر این متد یکسان هستند و نیازی نیست که هنگام وارد کردن متن برای جستجو به اندازه ی حروف نیز دقت کنید.

آخرین کاری که در این متد باید انجام دهیم این است که موقعیت رکورد جدید را در صفحه نمایش دهیم و برای این کار می تـوانیم از متد ShowPosition استفاده کنیم.
حال برای تکمیل برنامه، باید قابلیت های اضافه کردن رکورد جدید، حذف کردن یک رکورد و یا ویرایش کردن آن را نیز بـه برنامـه اضافه کنیم. در قسمت امتحان کنید بعدی، نحوه ی اضافه کردن یک رکورد جدید به داده های موجود را بررسی خواهیم کرد.

امتحان کنید: اضافه کردن رکورد جدید

۱) ابتدا به قسمت طراحی فرم Form1 بروید و روی دکمه ی btnNew دو بار کلیک کنید تا متد مربـوط بـه رویـداد Click آن ایجاد شود. سپس کد مشخص شده در زیر را در این متد وارد کنید:

```
private void btnNew_Click(object sender, EventArgs e)
{
     // Clear the book title and price fields...
     txtBookTitle.Text = "";
     txtPrice.Text = "";
}
```

۲) حال باید کد مربوط به متد btnAdd_Click را وارد کنیم. این متد مـسئول اضـافه کـردن یـک رکـورد داده ای جدید به برنامه است. این زیر برنامه، طولانی ترین زیر برنامه ای است که در این پروژه وجود دارد و کد زیـادی را شـامل می شود. دلیل آن نیز رابطه ی بین عنوان کتابها و نیز نویسندگان آنها و نیز کلید اصلی که برای عنوان کتابها استفاده می شود است. به قسمت طراحی فرم بروید و روی دکمه ی Add دو بار کلیک کنید تا متد مربوط به رویداد Click این کنترل ایجاد شود. سپس کد زیر را در این متد وارد کنید:

```
private void btnAdd_Click(object sender, EventArgs e)
{
     // Declare local variables and objects...
     int intPosition, intMaxID;
     String strID;
     SqlCommand objCommand = new SqlCommand();

     // Save the current record position...
     intPosition = objCurrencyManager.Position;
     // Create a new SqlCommand object...
     SqlCommand maxIdCommand = new SqlCommand(
          "SELECT MAX(title_id)" +
          "FROM titles WHERE title_id LIKE 'DM%'",
          objConnection);

     // Open the connection, execute the command
     objConnection.Open();
     Object maxId = maxIdCommand.ExecuteScalar();
```

```
    // If the MaxID column is null...
    if (maxId == DBNull.Value)
    {
        // Set a default value of 1000...
        intMaxID = 1000;
    }
    else
    {
        // otherwise set the strID variable
        // to the value in MaxID...
        strID = (String)maxId;

        // Get the integer part of the string...
        intMaxID = int.Parse(strID.Remove(0, 2));

        // Increment the value...
        intMaxID += 1;
    }

    // Finally, set the new ID...
    strID = "DM" + intMaxID.ToString();

    // Set the SqlCommand object properties...
    objCommand.Connection = objConnection;
    objCommand.CommandText = "INSERT INTO titles " +
    "(title_id, title, type, price, pubdate) " +
    "VALUES(@title_id,@title,@type,@price,@pubdate);" +
    "INSERT INTO titleauthor (au_id, title_id) " +
    "VALUES(@au_id,@title_id)";

    // Add parameters for the placeholders in the SQL in
    // the CommandText property...

    // Parameter for the title_id column...
    objCommand.Parameters.AddWithValue("@title_id",
                                       strID);

    // Parameter for the title column...
    objCommand.Parameters.AddWithValue("@title",
                                      txtBookTitle.Text);

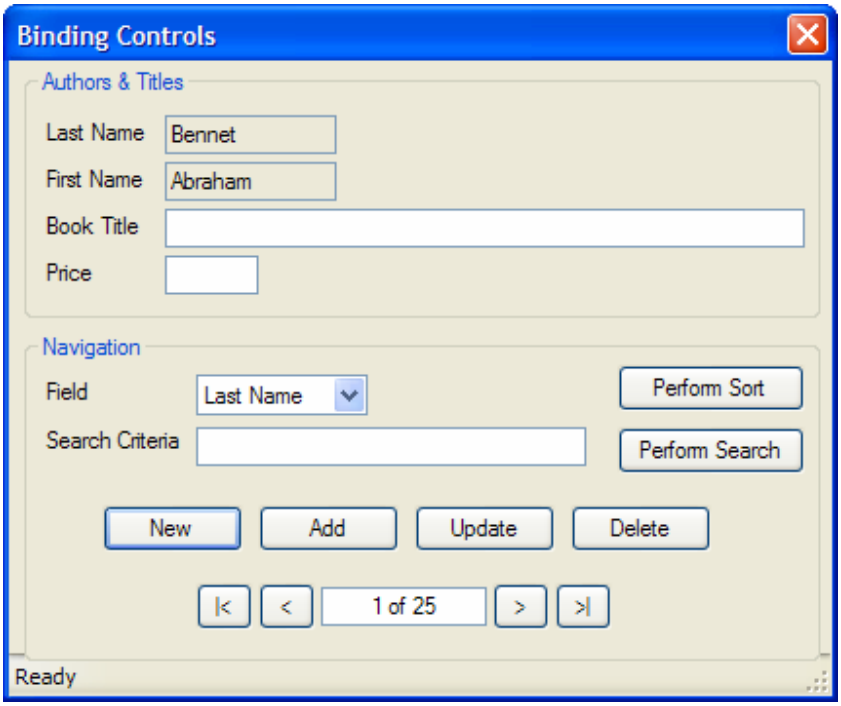
    // Parameter for the type column
    objCommand.Parameters.AddWithValue("@type", "Demo");
    // Parameter for the price column...
    objCommand.Parameters.AddWithValue("@price",
              txtPrice.Text).DbType = DbType.Currency;
```

```
        // Parameter for the pubdate column
        objCommand.Parameters.AddWithValue("@pubdate",
                                           DateTime.Now);

        // Parameter for the au_id column...
        objCommand.Parameters.AddWithValue("@au_id",
        this.BindingContext[objDataView,"au_id"].Current);

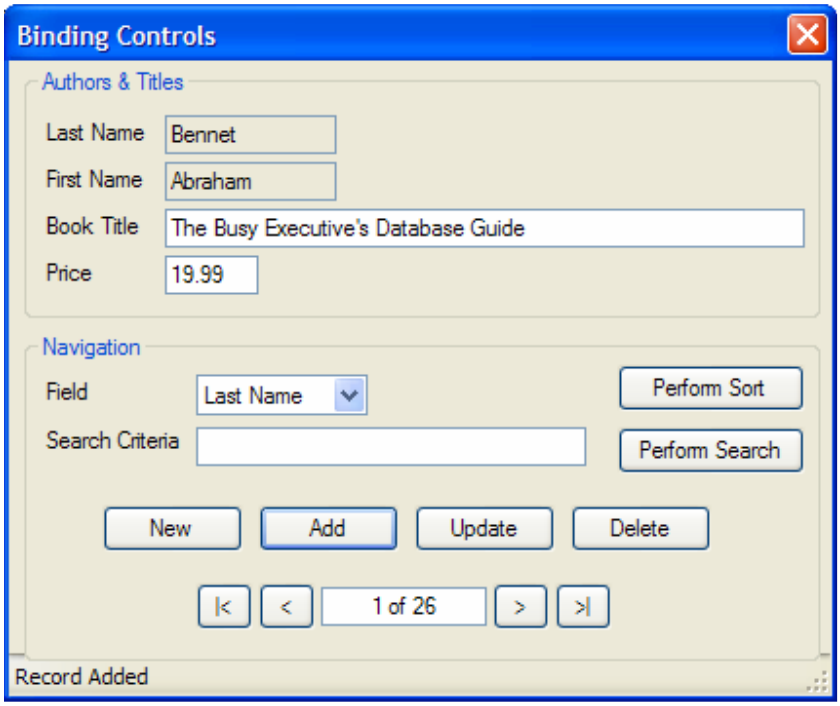
        // Execute the SqlCommand object
        // to insert the new data...
        try
        {
            objCommand.ExecuteNonQuery();
        }
        catch(SqlException SqlExceptionErr)
        {
            MessageBox.Show(SqlExceptionErr.Message);
        }
        // Close the connection...
        objConnection.Close();
        // Fill the dataset and bind the fields...
        FillDataSetAndView();
        BindFields();
        // Set the record position
        // to the one that you saved...
        objCurrencyManager.Position = intPosition;
        // Show the current record position...
        ShowPosition();
        // Display a message that the record was added...
        ToolStripStatusLabel1.Text = "Record Added";
}
```

۳) برنامه را اجرا کرده و کاربری را که می خواهید عنوان کتاب جدیدی را برای او ثبت کنید، انتخاب کنید، سپس روی دکمه ی Add کلیک کنید. به این ترتیب کادرهای Book Title و Price خالی خواهند شد و می توانید داده های مربوط به کتاب جدید را همانند شکل ۱۶-۱۰ وارد کنید. در برنامه به تعداد رکورد هایی که هم اکنون وجود دارند توجه کنید (۲۵ رکورد در شکل ۱۶-۱۰).



شکل ۱۶-۱۰

۴) حال نام کتاب و قسمت آن را در فیلد های مربوطه وارد کرده و روی دکمه ی Add کلیک کنید. به این ترتیب پیغامی در نوار وضعیت نمایش داده می شود و بیان می کند که رکورد جدید با موفقیت اضافه شده است. همچنین همانطور که در شکل ۱۶-۱۱ مشخص است تعداد رکورد ها در برنامه یک واحد افزایش پیدا می کند (در این شکل ۲۶ رکورد وجود دارد).



شکل ۱۶-۱۱

حال بهتر است نحوه ی عملکرد برنامه را بررسی کنیم.

چگونه کار می کند؟

در این برنامه تنها اطلاعاتی که می توانیم اضافه کنیم، نام و قیمت کتاب جدید است. بنابراین به جای اینکه اطلاعات داخل کادر های Price و Book Title را انتخاب کرده، پاک کنیم و سپس اطلاعات جدید را در آنها وارد کنیم، می توانیم به سادگی روی کلید New در فرم کلیک کنیم. وظیفه ی این کلید این است که داده های درون کادرهای Price و نیز Book Title را پاک کند تا بتوانیم داده های جدید را در آن وارد کنیم. بنابراین در متد مربوط به این دکمه، خاصیت Text این دو کنترل TextBox را برابر با رشته ی خالی قرار می دهیم:

```
private void btnNew_Click(object sender, EventArgs e)
{
     // Clear the book title and price fields...
     txtBookTitle.Text = "";
     txtPrice.Text = "";
}
```

هنگامی که روی دکمه ی New کلیک کنید این دو کادر خالی خواهند شد، بنابراین اگر کاربر در حال ویرایش کردن اطلاعات باشد تمام تغییرات وارد شده از دست خواهند رفت. معمولا در برنامه ها قسمتی را به کد اضافه می کنند که از این نوع مشکلات جلوگیری کند، اما در این برنامه فعلا نیازی به این کار نداریم.

بهتر است قبل از توضیح ادامه ی متد، مفهوم کلید اصلی در بانکهای اطلاعاتی را توضیح دهیم. در هر جدول از بانک اطلاعاتی که بخواهیم از منحصر به فرد بودن داده های موجود در هر رکورد از جدول مطمئن شویم، معمولا یک ستون از جدول را به عنوان **کلید اصلی**[1] در نظر می گیریم، سپس آن را در بانک اطلاعاتی به عنوان PrimaryKey مشخص می کنیم. به این ترتیب هنگامی که بخواهیم داده ای را در بانک اطلاعاتی وارد کنیم، موتور بانک اطلاعاتی ابتدا بررسی می کند که رکوردی با کلید اصلی که برای رکورد جدید مشخص شده است وجود نداشته باشد، سپس رکورد جدید را به جدول مورد نظر اضافه می کند.

در جدول titles نیز ستون title_id که به عنوان PrimaryKey مشخص شده است، از یک پیشوند مشخص کننده ی گروه کتاب و یک عدد ترتیبی تشکیل شده است. بنابراین برای مشخص کردن مقداری که باید در ستون کلید اصلی این جدول قرار بگیرد، ابتدا باید مشخص کنیم که تا کنون چند عنوان کتاب از این گروه در جدول ثبت شده اند. سپس یک واحد به عدد بدست آمده اضافه کرده و آن را به عنوان مقدار ستون کلید اصلی این رکورد قرار دهیم.

در زیر برنامه ی btnAdd_Click ابتدا متغییر ها و اشیایی که می خواهیم در طول برنامه از آنها استفاده کنیم را تعریف می کنیم. متغییر intPosition برای نگهداری موقعیت رکورد جاری به کار می رود و متغییر intMaxID نیز برای نگهداری تعداد عناوین کتابی که از یک گروه در جدول به ثبت رسیده اند استفاده می شود. متغیر strID برای نگهداری کلید اصلی رکوردی که قرار است در جدول titles ثبت شود به کار می رود. در آخر، شیئ objCommand نیز برای نگهداری دستورات SQL مربوط به قرار دادن یک رکورد جدید از اطلاعات در جدول های titles و authortitle به کار می رود.

---

[1] Primary Key

بهتر است قبل از هر چیز موقعیت رکورد جاری را در متغییر intPosition قرار دهیم، تا بعد از اینکه رکـورد داده ای جدیـد را به جدول در بانک اطلاعاتی اضافه کردیم و کنترل DataView را پر کردیم، بتوانیم مجدداً به این رکورد برگردیم:

```
// Save the current record position...
intPosition = objCurrencyManager.Position;
```

حال باید یک دستور SQL را در بانک اطلاعاتی اجرا کنیم تا بدانیم چه شناسه ای را باید به عنوان کلیـد اصـلی بـرای ثبـت رکـورد جدید در جدول titles به کار ببریم. این دستور SQL را در یک شیئ از کلاس SqlCommand قرار می دهیم و سپس آن را در بانک اطلاعاتی اجرا می کنیم. وظیفه ی این دستور این است که در بین شناسه هایی که با پیشوند DM شـروع مـی شـوند، شناسه ای که دارای بزرگترین عدد است را برگرداند.

نکته: در جدول titles هیچ کتابی در گروه Demo قرار ندارد، بنابراین تمام کتابهایی که در طول برنامه ثبت می کنـیم را در گروه Demo قرار داده و پیشوند DM را برای آنها در نظر می گیریم. به این ترتیب بعد از اتمام برنامه، هنگامی که بخواهیم داده ها را در نرم افزار مربوط به بانک اطلاعاتی بررسی کنیم، به راحتی می توانیم رکورد هایی که خود اضافه کـرده ایـم را تـشخیص داده و آنها را حذف کنیم.

تابع MAX که در این دستور SQL از آن استفاده کرده ایم، یک ستون از داده ها را دریافت کرده و بزرگترین آن را به عنوان نتیجـه برمی گرداند. همچنین در این دستور از عبارت LIKE استفاده می کنیم تا فقط داده هایی بررسی شوند که با DM (مشخص کننده ی گروه کتابهای Demo) شروع می شوند. به عبارت دیگر به وسیله ی این دستور به موتور بانک اطلاعاتی می گوییم که از ستون title_id در جدول Titles، داده هایی را که با DM شروع می شوند را انتخاب کـرده و سـپس از بـین ایـن داده هـا، بزرگترین آنها را به عنوان نتیجه برگردان:

```
// Create a new SqlCommand object...
SqlCommand maxIdCommand = new SqlCommand(
     "SELECT MAX(title_id)" +
     "FROM titles WHERE title_id LIKE 'DM%'",
     objConnection);
```

همانطور که احتمالا تا کنون متوجه شده اید، این دستور فقط یک داده را به عنوان نتیجه برمـی گردانـد: بزرگتـرین مقـدار از سـتون title_id که با عبارت DM شروع می شود. در چنین شرایطی برای اجرای دستور، می توانیم بعد از برقراری اتصال به بانـک اطلاعـاتی بـا اسـتفاده از متـد Open در کـلاس SqlConnection، متـد ExecuteScalar از کـلاس SqlCommand را فراخوانی کنیم. این متد زمانی به کار می رود که دستور SQL مورد اسـتفاده فقـط یـک داده را برگردانـد[1]. مقدار برگشتی این متد از نوع Object است و باید با توجه به نتیجه ای که انتظار می رود دستور SQL مورد استفاده برگردانـد، آن را به نوع داده ای مورد نظر تبدیل کنیم.

```
// Open the connection, execute the command
objConnection.Open();
Object maxId = maxIdCommand.ExecuteScalar();
```

[1] البته این متد در حقیقت مقدار موجود در اولین ردیف از اولین ستون را در جدول نتیجه بر می گرداند. اما در دستوری که در این قسمت به کار برده ایـم سـتونی وجود ندارد و فقط یک ردیف وجود دارد، بنابراین دقیقاً داده ای را که می خواهیم استفاده کنیم برمی گرداند.

بعد از اجرای دستور SQL در بانک اطلاعاتی، باید نتیجه را با مقدار Null بررسی کنیم:

```
// If the MaxID column is null...
if (maxId == DBNull.Value)
```

اگر این شرط برابر با true ارزیابی شود، به این معنی است که در جدول titles هیچ کلید اصلی که با مقدار DM شروع شود وجود ندارد. بنابراین مقدار اولیه متغیر intMaxID را برابر با ۱۰۰۰ قرار می دهیم:

```
// Set a default value of 1000...
intMaxID = 1000;
```

اما اگر مقدار شرط برابر با false باشد، به این معنی است که حداقل یک کلید اصلی در جدول وجود دارد که با عبارت DM شروع شود. در این حالت باید عدد صحیح این شناسه را بدست آورده تا بتوانیم مشخص کنیم که از چه عددی باید برای شناسه ی خود استفاده کنیم. برای این کار ابتدا باید شیئ maxID را به رشته تبدیل کرده و آن را در متغییر strID قرار دهیم:

```
else
{
     // otherwise set the strID variable
     // to the value in MaxID...
     strID = (String)maxId;
```

سپس با استفاده از متد Remove در کلاس String، دو حرف اول این رشته را حذف کرده تا عبارت DM از ابتدای آن پاک شود و فقط عدد مشخص کننده ی شناسه باقی بماند. متد Remove تعدادی کاراکتر مشخص را از یک رشته حذف می کند و باقیمانده آن را برمی گرداند. یکی از نسخه های این متد مکان شروع و نیز تعداد کاراکتر هایی که باید حذف شوند را به عنوان پارامتر دریافت کرده و متن بدون کاراکتر های مشخص شده را برمی گرداند. در اینجا با استفاده از متد Remove از کاراکتر صفرم شروع کرده و دو کاراکتر را حذف می کنیم. به این ترتیب عبارت DM از ابتدای رشته حذف می شود. سپس با استفاده از متد Parse از کلاس int، متن باقیمانده که شامل شماره ی شناسه است را به عدد تبدیل می کنیم. در انتها نیز برای مشخص شدن شماره شناسه ی مورد نیاز، یک واحد به عدد بدست آمده اضافه می کنیم.

```
     // Get the integer part of the string...
     intMaxID = int.Parse(strID.Remove(0, 2));

     // Increment the value...
     intMaxID += 1;
}
```

بعد از مشخص شدن شماره ی شناسه، با استفاده از این شماره و اضافه کردن پیشوند DM به ابتدای آن یک شماره شناسه ی جدید ایجاد می کنیم.

```
// Finally, set the new ID...
```

```
strID = "DM" + intMaxID.ToString();
```

بعد از مشخص شدن شناسه ی مورد نیاز، باید یک دستور SQL ایجاد کنیم تا بتواند داده های مورد نظر را در جدول titles و authortitle قرار دهد. اگر با دقت بیشتری کد قبل را بررسی کنید متوجه خواهید شد که در خاصیت CommandText از شیئ objCommand، دو دستور INSERT مجزا قرار داده شده است که با ; از یکدیگر جدا شده اند. در زبان SQL نیز برای جدا سازی دستورات و نوشتن متوالی آنها می توانید از کاراکتر ; استفاده کنید. این دستور SQL دارای چندین PlaceHolder است که می توانیم به وسیله ی اشیایی از نوع SqlParameter آنها را با مقادیر مناسب جایگزین کنیم.

نکته: به علت رابطه ی خاصی که بین دو جدول titles و authortitle وجود دارد، هنگام وارد کردن داده ای در این جداول ابتدا باید داده ها را در جدول titles وارد کرده و سپس آنها را در جدول authortitle قرار دهیم. در دستور SQL قبل هم اگر توجه کنید خواهید دید که دستور INSERT مربوط به قرار دادن داده ها در جدول titles قبل از دستور INSERT مربوط به قرار دادن داده ها در جدول authortitle آمده است.

بنابراین مطابق معمول Connectionای که باید برای اتصال به بانک اطلاعاتی مورد استفاده قرار بگیرد و همچنین دستور SQL مورد نظر را در خاصیتهای Connection و CommandText قرار می دهیم:

```
// Set the SqlCommand object properties...
objCommand.Connection = objConnection;
objCommand.CommandText = "INSERT INTO titles " +
"(title_id, title, type, price, pubdate) " +
"VALUES(@title_id,@title,@type,@price,@pubdate);" +
"INSERT INTO titleauthor (au_id, title_id) " +
"VALUES(@au_id,@title_id)";
```

بعد از این، باید بازای هر یک placeholderای که در دستور SQL قرار داده ایم، یک مقدار به خاصیت Parameters اضافه کنیم. البته در این قسمت با وجود اینکه placeholder مربوط به title_id دو بار استفاده شده است، فقط یک عضو به خاصیت Parameters اضافه می کنیم:

```
// Add parameters for the placeholders in the SQL in
// the CommandText property...

// Parameter for the title_id column...
objCommand.Parameters.AddWithValue("@title_id",
                                   strID);

// Parameter for the title column...
objCommand.Parameters.AddWithValue("@title",
                                  txtBookTitle.Text);

// Parameter for the type column
objCommand.Parameters.AddWithValue("@type", "Demo");
```

```
// Parameter for the price column...
objCommand.Parameters.AddWithValue("@price",
          txtPrice.Text).DbType = DbType.Currency;

// Parameter for the pubdate column
objCommand.Parameters.AddWithValue("@pubdate",
                                   DateTime.Now);

// Parameter for the au_id column...
objCommand.Parameters.AddWithValue("@au_id",
this.BindingContext[objDataView,"au_id"].Current);
```

برای مقدار پارامتر @title_id، از مقدار موجود در متغییر strID که پیشتر ایجاد کردیم استفاده می کنیم. برای پارامتر @title نیز از مقداری که به وسیله ی کاربر در کادر Book Title وارد شده است استفاده می کنیم. برای پارامتر @price مقدار موجود در کادر Price را به کار می بریم. البته مقدار موجود در این کادر یک String است و SQL Server نمی تواند آن را به طور اتوماتیک به مقدار عددی تبدیل کند. بنابراین لازم است که خاصیت DbType آن را برابر با DbType.Currency قرار دهیم تا مشخص کنیم که این مقدار، یک مقدار عددی است. به این ترتیب زمان اجرای دستور SQL، هنگامی که برنامه بخواهد این پارامتر را با مقدار آن جایگزین کند، با آن به صورت یک عدد برخورد می کند.
برای پارامتر au_id باید شناسه ی مربوط به نویسنده ای که هم اکنون انتخاب شده است را بدست آوریم. در این برنامه هیچ کنترلی به ستون au_id متصل نشده است تا بتوانیم شناسه را به وسیله ی آن بدست آوریم. بنابراین برای انجام این کار باید از مقداری کد در برنامه استفاده کنیم. بهتر است ابتدا کد مورد استفاده را کمی دقیقتر نگاه کنیم:

```
this.BindingContext[objDataView,"au_id"].Current
```

همانطور که می دانید خاصیت BindingContext از یک فرم، برای دسترسی به منابع داده ای موجود در آن فرم به کار می رود. در حقیقت این خاصیت را می توانیم همانند یک آرایه ی سه بعدی در نظر بگیریم که در یک بعد آن منابع داده ای قرار دارند، و دو بعد دیگر آن را ستونها و ردیف های اطلاعاتی موجود در هر منبع داده ای تشکیل می دهند. در اینجا با ذکر کردن نام objDataView و نیز "au_id" مشخص کرده ایم که می خواهیم به ستون au_id از منبع داده ای objDataView دسترسی داشته باشیم. سپس با استفاده از خاصیت Current در این قسمت، مشخص می کنیم که می خواهیم داده ی موجود در رکورد جاری از ستون تعیین شده را بدست آوریم.
دو پارامتر دیگر نیز نوع و زمان انتشار کتاب را مشخص می کنند که در این قسمت از رشته ی ثابت Demo برای نوع کتاب، و از خاصیت Now در کلاس DateTime که مشخص کننده ی زمان کنونی است برای زمان انتشار آن استفاده کرده ایم:

```
// Parameter for the type column
objCommand.Parameters.AddWithValue("@type", "Demo");

// Parameter for the pubdate column
objCommand.Parameters.AddWithValue("@pubdate",
                                   DateTime.Now);
```

بعد از این که تمام پارامترهای مورد نیاز را مشخص کردیم، می توانیم دستور SQL ایجاد شده را اجرا کنیم. برای این کار از متد ExecuteNonQuery در کلاس SqlCommand استفاده می کنیم. این متد زمانی استفاده می شود که دستور

SQL مورد استفاده، شامل پرس و جو نباشد (حاوی دستور SELECT نباشد) و فقط برای ایجاد تغییراتی در بانک اطلاعاتی به کار برود. در اینجا می خواهیم با استفاده از دستور INSERT داده هایی را در بانک اطلاعاتی قرار دهیم، پس می توانیم با استفاده از متد ExecuteNonQuery این دستور را در بانک اطلاعاتی اجرا کنیم. بعد از آن نیز باید اتصال به بانک اطلاعاتی را قطع کنیم.
این قسمت از برنامه یکی از قسمتهایی است که احتمال رخ دادن خطا در آن بسیار زیاد است. بنابراین یک بلاک خیلی ساده ی try...catch در این قسمت ایجاد می کنیم تا بتوانیم رخ دادن خطاهای احتمالی را کنترل کنیم. در قسمت کنترل خطا، فقط با استفاده از یک کادر پیغام متن خطای به وجود آمده را به کاربر نمایش می دهیم.

```
try
{
      objCommand.ExecuteNonQuery();
}
catch(SqlException SqlExceptionErr)
{
      MessageBox.Show(SqlExceptionErr.Message);
}
```

سپس، بعد از قطع اتصال به بانک اطلاعاتی، متدهای FillDataSetAndView و BindFields را مجدداً فراخوانی می کنیم تا به این وسیله اطلاعات جدید را از بانک اطلاعاتی دریافت کرده و در DataSet قرار دهیم، زیرا ممکن است در طول این زمانی که از بانک اطلاعاتی قطع بوده ایم فرد دیگری با استفاده از یک برنامه ی دیگر وارد این سرور شده و داده های جدول را تغییر داده باشد. به این ترتیب می توانیم مطمئن شویم که تمام تغییراتی که در این مدت در این جدول از بانک اطلاعاتی به وجود آمده است را دریافت کرده ایم.

نکته: جداول و بانکهای اطلاعاتی که در SQL Server ایجاد می شوند، می توانند در یک لحظه به وسیله ی چند برنامه، و در حقیقت به وسیله ی چند کاربر مختلف مورد استفاده قرار بگیرند و هر یک از این کاربران نیز ممکن است تغییرات مورد نظر خود را در داده های جدول وارد کنند. همانطور که می دانید، بعد از اینکه برنامه به بانک اطلاعاتی متصل شد و داده های مورد نیاز خود را دریافت کرد، اتصال را قطع می کند و به این ترتیب، تغییراتی که در این مدت توسط افراد دیگر در داده های بانک اطلاعاتی ایجاد می شود به وسیله ی برنامه دریافت نمی شود. بنابراین با دریافت مجدد اطلاعات در قسمت قبل می توانیم مطمئن شویم که تمام تغییراتی که در این مدت در داده ها به وجود آمده است را دریافت کرده ایم.

بعد از این کار، با استفاده از خاصیت Position در شیئ objCurrencyManager، رکورد جاری را به همان مکانی برمی گردانیم که قبل از فراخوانی متد قرار داشت. برای این کار از مکان رکورد که قبل از فراخوانی متد آن را در متغییر intPosition قرار داده بودیم استفاده می کنیم.
بعد از تغییر موقعیت رکوردی که در حال نمایش داده شدن است، متد ShowPosition را نیز فراخوانی می کنیم تا شماره رکورد جاری نیز در صفحه نمایش داده شود.در آخر پیغامی را در نوار وضعیت نمایش می دهیم تا مشخص شود رکورد جدید با موفقیت به بانک اطلاعاتی اضافه شده است.
در قسمت امتحان کنید بعد، کد مربوط به دکمه ی btnUpdate را وارد خواهیم کرد. کد مربوط به این دکمه مقداری ساده تر از قسمت قبل است، زیرا در این قسمت فقط می خواهیم بعضی از اطلاعات مربوط به یکی از رکورد های جدول titles را تغییر دهیم.

امتحان کنید: ویرایش داده ها

۱) به قسمت طراحی فرم بروی و روی دکمه ی btnUpdate دو بار کلیک کنید تا متد مربوط بـه رویـداد Click آن ایجاد شود. سپس کد مشخص شده در زیر را در این متد وارد کنید:

```
private void btnUpdate_Click(object sender, EventArgs e)
{
     // Declare local variables and objects...
     int intPosition;
     SqlCommand objCommand = new SqlCommand();

     // Save the current record position...
     intPosition = objCurrencyManager.Position;

     // Set the SqlCommand object properties...
     objCommand.Connection = objConnection;
     objCommand.CommandText = "UPDATE titles " +
          "SET title = @title, price = @price " +
          "WHERE title_id = @title_id";
     objCommand.CommandType = CommandType.Text;

     // Add parameters for the placeholders in the SQL in
     // the CommandText property...

     // Parameter for the title field...
     objCommand.Parameters.AddWithValue("@title",
                                        txtBookTitle.Text);

     // Parameter for the price field...
     objCommand.Parameters.AddWithValue("@price",
               txtPrice.Text).DbType = DbType.Currency;

     // Parameter for the title_id field...
     objCommand.Parameters.AddWithValue("@title_id",
     this.BindingContext[objDataView,"title_id"].Current);

     // Open the connection...
     objConnection.Open();

     // Execute the SqlCommand object to update the data...
     objCommand.ExecuteNonQuery();

     // Close the connection...
     objConnection.Close();

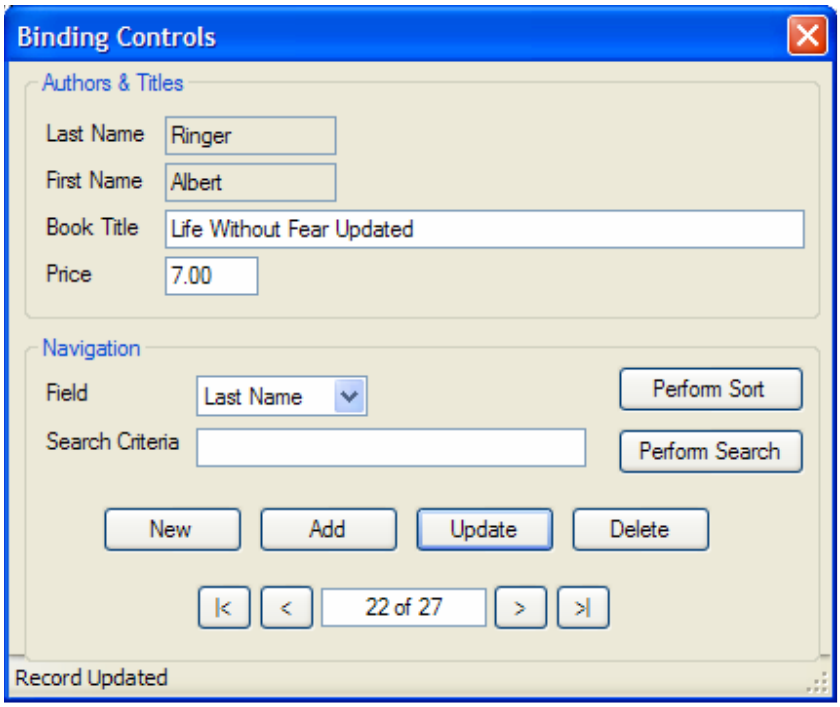
     // Fill the DataSet and bind the fields...
```

```
        FillDataSetAndView();
        BindFields();
        // Set the record position
        // to the one that you saved...
        objCurrencyManager.Position = intPosition;

        // Show the current record position...
        ShowPosition();

        // Display a message that the record was updated...
        ToolStripStatusLabel1.Text = "Record Updated";
}
```

۲) برنامه را اجرا کنید. حال می توانید اطلاعات مربوط به کتابی که اضافه کرده بودید را تغییر دهید و یا تغییراتی را در اطلاعات مربوط به دیگر کتابها ایجاد کنید. یک کتاب را انتخاب کرده و با استفاده از کادر `Price` قیمت آن را تغییر دهید. سپس روی دکمه ی `Update` کلیک کنید. به این ترتیب تغییرات مورد نظر شما در بانک اطلاعاتی ذخیره می شود و پیغامی نیز در نوار وضعیت نمایش داده می شود و ثبت تغییرات را اعلام می کند (شکل ۱۶-۱۲).



شکل ۱۶-۱۲

چگونه کار می کند؟

مانند قسمتهای قبل، اولین کاری که باید انجام دهیم تعریف متغییر ها و اشیای مورد نیاز است. در این قسمت نیز به یک متغییر برای نگهداری رکورد جاری و نیز یک شیئ از نوع SqlCommandنیاز داریم. بعد از تعریف این دو، همانند زیر برنامه ی قبلی مکان رکورد جاری را در متغیری که ایجاد کرده بودیم قرار می دهیم.

با اضافه کردن کد زیر، شیئ objConnection را در خاصیت Connection از شیئ SqlCommand قرار می دهیم. همچنین دستور SQL مورد نظر خود را نیز در خاصیت CommandText وارد می کنیم. دستور SQLای که در این قسمت استفاده می کنیم یک دستور UPDATE است که برای ویرایش ستونهای Price و Title از یکی از رکورد های جدول titles به کار می رود. توجه کنید که در این دستور از سه placeholder استفاده کرده ایم: دو placeholder برای مقادیر جدیدی که باید در ستون Price و Title قرار بگیرند و یک placeholder نیز برای title_id که در قسمت WHERE به کار گرفته شده است:

```
// Set the SqlCommand object properties...
objCommand.Connection = objConnection;
objCommand.CommandText = "UPDATE titles " +
     "SET title = @title, price = @price " +
     "WHERE title_id = @title_id";
objCommand.CommandType = CommandType.Text;
```

بعد از قرار دادن دستور SQL در خاصیت CommandText، مقدار CommandType را نیز برابر با CommandType.Text قرار می دهیم تا مشخص کنیم که عبارت موجود در خاصیت CommandText یک دستور SQL است.

حال باید پارامترهای لازم را به خاصیت Parameters اضافه کنیم. اولین پارامتر، @title است و باید شامل عنوان جدیدی باشد که برای کتاب در نظر گرفته شده است. این عنوان را کاربر در فرم برنامه وارد کرده است و برای دسترسی به آن می توانیم از خاصیت Text در کنترل txtBookTitle استفاده کنیم. بنابراین مقدار پارامتر @title را برابر با txtBookTitle.Text قرار می دهیم تا هنگامی که برنامه بخواهد این دستور SQL را برای اجرا به بانک اطلاعاتی بفرستد، عبارت @title در دستور را با مقدار موجود در خاصیت Text جایگزین کند.

پارامتر دوم مربوط به قسمت Price در دستور UPDATE است. این پارامتر برای تغییر قیمت کتاب به کار می رود و مقدار آن به وسیله ی کاربر در خاصیت Text کنترل txtPrice در فرم برنامه وارد شده است. در این قسمت نیز همانند بخش قبلی، باید خاصیت DbType این پارامتر را تعیین کنیم تا رشته ی وارد شده در کنترل txtPrice، قبل از جایگزین شدن در دستور به عدد تبدیل شود.

پارامتر آخر نیز شامل شناسه ی کتابی است که باید اطلاعات آن تغییر کند و در قسمت WHERE از دستور UPDATE وارد شده است. مقدار این پارامتر برابر با شناسه ی title_id از رکورد جاری است که آن را همانند زیر برنامه ی قبلی بدست آورده و در دستور UPDATE جایگزین می کنیم.

بقیه ی کد نیز همانند متد btnAdd_Click است و نیازی به توضیح ندارد. کد مربوط به دکمه ی آخر یعنی دکمه ی Delete نیز در قسمت امتحان کنید بعدی توضیح داده شده است.

امتحان کنید: حذف کردن یک رکورد

۱) به قسمت طراحی فرم رفته و روی دکمه ی btnDelete دو بار کلیک کنید تا متد مربوط به رویداد Click آن ایجاد شود. سپس کد مشخص شده در زیر را در این متد وارد کنید:

```
private void btnDelete_Click(object sender, EventArgs e)
{
     // Declare local variables and objects...
     int intPosition;
     SqlCommand objCommand = new SqlCommand();

     // Save the current record position - 1 for the one to
     // be deleted...
     intPosition =
          this.BindingContext[objDataView].Position - 1;

     // If the position is less than 0 set it to 0...
     if( intPosition < 0 )
          intPosition = 0;

     // Set the Command object properties...
     objCommand.Connection = objConnection;
     objCommand.CommandText = "DELETE FROM titleauthor " +
          "WHERE title_id = @title_id;" +
          "DELETE FROM titles WHERE title_id = @title_id";
     // Parameter for the title_id field...
     objCommand.Parameters.AddWithValue("@title_id",
     this.BindingContext[objDataView,"title_id"].Current);

     // Open the database connection...
     objConnection.Open();

     // Execute the SqlCommand object to update the data...
     objCommand.ExecuteNonQuery();

     // Close the connection...
     objConnection.Close();

     // Fill the DataSet and bind the fields...
     FillDataSetAndView();
     BindFields();

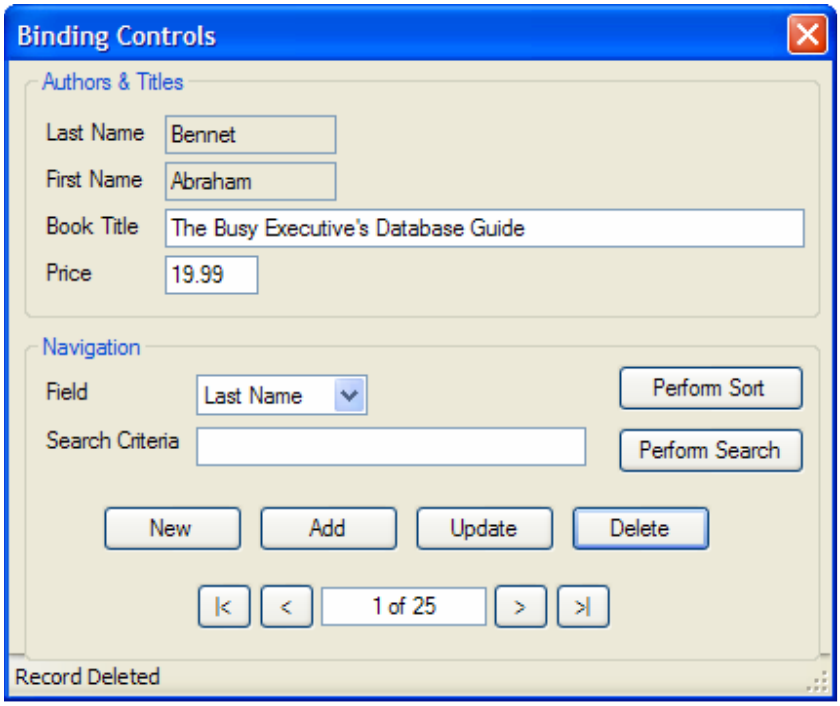
     // Set the record position
     // to the one that you saved...
     this.BindingContext[objDataView].Position =
                                          intPosition;

     // Show the current record position...
```

```
        ShowPosition();

        // Display a message that the record was deleted...
        ToolStripStatusLabel1.Text = "Record Deleted";
}
```

۲) خوب، به این ترتیب این پروژه نیز به پایان رسید. اما بهتر است قبل از اینکه از تمام شدن آن خوشحال شویم ابتدا قابلیت جدیدی که اضافه کرده ایم را امتحان کنیم. برنامه را اجرا کرده و هر کتابی که می خواهید حذف کنید را انتخاب کنید، سپس روی دکمه ی `Delete` کلیک کنید. به خاطر داشته باشید که بامک اطلاعاتی `pubs` که در این برنامه از آن استفاده کرده ایم یک بانک اطلاعاتی نمونه است و ممکن است افراد دیگری نیز به این `SQL Server` متصل شوند و بخواهند از آن برای تمرینات خود استفاده کنند. بنابراین بهتر است داده هایی را حذف کنیم که در قسمت قبل ایجاد کرده بودیم. قبل از اینکه یک کتاب را حذف کنید، به شماره ی رکورد ها در برنامه توجه کنید، سپس کتاب مورد نظر خود را حذف کنید (شکل ۱۶-۱۳). البته در این قسمت ممکن است با خطا مواجه شوید، زیرا کتابی که برای حذف انتخاب کرده اید ممکن است با داده های جدول `sales` در بانک اطلاعاتی رابطه داشته باشند. بنابراین باید کتاب دیگری را انتخاب کرده و حذف کنید.



شکل ۱۶-۱۳

چگونه کار می کند؟

این زیر برنامه ممکن است کمی پیچیده تر از زیر برنامه ی `btnUpdate_Click` به نظر برسد و دلیل آن نیز رابطه ی بین داده های موجود در جدول `titles` و جدول `authors` است. همانطور که به خاطر دارید، جدولی به نام

authortitle در بانک اطلاعاتی pubs وجود دارد کـه رابطـه ی بـین داده هـای موجـود در جـدول authors و جدول titles را تعیین می کند. بنابراین قبل از اینکه رکوردی را از جدول titles پاک کنیم، بانک رکورد مربوط بـه آن را از جدول authortitle حذف کنیم. بنابراین در این زیر برنامه نیز به دو دستور DELETE نیاز داریم.
دقت کنید که در این زیر برنامه هنگامی که متغییر های مورد نیاز را تعریف کردیم، مکان رکورد جاری را برابر با مکان رکورد انتخاب شده منهای ۱ قرار می دهیم. به این ترتیب کاربر می تواند به آخرین رکورد برود و آن را حذف کند. به این ترتیب رکورد قبلی رکـورد حذف شده در برنامه نمایش داده می شود. همچنین قبل از حذف رکورد بررسی می کنیم که اگر کاربر در اولـین رکـورد قـرار داشـته باشد (شماره رکورد جاری منهای یک از صفر کمتر شود)، شماره رکورد را برابر با صفر قرار دهیم تا بعد از حذف اولین رکورد، رکوردی که به با اندیس صفر مشخص می شود نمایش داده شود.
همچنین توجه کنید که در این زیر برنامه دیگر از شیئ CurrencyManager استفاده نکرده ایم، بلکه به صورت مـستقیم از شیئ BindingContext در فرم استفاده کرده و منبع داده ای objDataView را برای آن مشخص کرده ایـم. همانطور که به خاطر دارید شیئ BindingContext قسمتی از فرم به شمار می رود و نیازی نیست که برای اضافه کردن آن به برنامه کار خاصی را انجام دهیم. دلیل این که در این قسمت از BindingContext استفاده کرده ایم این است کـه نشان دهیم چگونه می توان از این شیئ در برنامه استفاده کرد و ضرورتی ندارد که حتماً در یک برنامه برای جـا بـه جـا شـدن بـین رکورد ها از شیئ CurrencyManager استفاده کنیم.

```
// Declare local variables and objects...
int intPosition;
SqlCommand objCommand = new SqlCommand();

// Save the current record position - 1 for the one to
// be deleted...
intPosition =
     this.BindingContext[objDataView].Position - 1;

// If the position is less than 0 set it to 0...
if( intPosition < 0 )
     intPosition = 0;
```

در این زیر برنامه برای تنظیم خاصیت CommandText از شیئ SqlCommand، دو دستور DELETE ایجاد کرده و آنها را به وسیله ی یک ; از هم جدا می کنیم. سپس عبارت مربوط به این دو دستور را در خاصیت CommandText قرار می دهیم. دستور DELETE اول رکـورد مربـوط بـه رابطـه ی کتـابی کـه مـی خـواهیم حـذف کنـیم و نویـسنده ی آن را از جـدول authortitle حذف می کند. دستور DELETE دوم نیز رکورد مربوط به خود کتاب را از جدول titles حذف می کند.

```
// Set the Command object properties...
objCommand.Connection = objConnection;
objCommand.CommandText = "DELETE FROM titleauthor " +
     "WHERE title_id = @title_id;" +
     "DELETE FROM titles WHERE title_id = @title_id";
```

در این قسمت نیز برای کلید اصلی که در قسمت WHERE دو دستور DELETE مورد استفاده قرار می گیرد، یک placeholder ایجاد می کنیم. سپس مقدار این placeholder را برابر با شناسه ی مربوط به کتابی که می خواهیم حذف کنیم قرار می دهیم (در این قسمت نیز با توجه به این که از placeholder در دو قسمت استفاده کرده ایم، اما فقط یک عضو را به خاصیت Parameters اضافه می کنیم):

```
// Parameter for the title_id field...
objCommand.Parameters.AddWithValue("@title_id",
this.BindingContext[objDataView,"title_id"].Current);
```

بقیه س کد نیز همانند زیر برنامه های قبلی است و باید تاکنون به اندازه ی کافی با آنها آشنا شده باشید. با اتمام این متد، این پروژه و این فصل نیز به پایان می رسد. امیدوارم که با اتمام این فصل اطلاعات خوبی در رابطه با اتصال به داده ها، وارد کردن داده های جدید، ویرایش و یا حذف آنها با استفاده از SQL در یک بانک اطلاعاتی بدست آورده باشید.
قبل از اینکه فصل را به پایان برسانیم، بهتر است یادآور شوم که کنترل خطاها بخش مهمی از یک برنامه را تشکیل می دهند. در پروژه ی قبلی به جز یک قسمت، در بقیه ی موارد از کنترل خطاها صرفنظر کردیم تا مکان کمتری به وسیله ی کد برنامه ها گرفته شود. همچنین هنگام وارد کردن داده ها در بانک اطلاعاتی از صحیح بودن و معتبر بودن آنها نیز صرفنظر کردیم. بنابراین ممکن است کاربر سعی کند رکوردی را بدون وارد کردن داده ای در بانک اطلاعاتی ایجاد کند و این خود موجب بروز خطا در برنامه می شود.

نتیجه:

در این فصل به بعضی از کلاسهای مهم ADO.NET از قبیل SqlConnection، SqlCommand، SqlDataAdapter و SqlParametre نگاهی انداختیم و مشاهده کردیم که این کلاسها چگونه می توانند هنگام دریافت اطلاعات، وارد کردن اطلاعات جدید، حذف اطلاعات جاری و یا ویرایش آنها کمک کنند. البته تمام این کلاسها برای بانک های اطلاعاتی مورد استفاده قرار می گیرد که به وسیله ی موتور بانک اطلاعاتی SQL Server ایجاد شده باشند. این کلاسها دارای کلاسهای متناظری هستند که با پیشوند OleDb شروع می شوند و در فضای نام System.Data.OleDb قرار دارند.
همچنین در این فصل کلاسهای DataSet و DataView از فضای نام System.Data را بررسی کرده و نحوه ی استفاده از آنها در یک برنامه را مشاهده کردیم و دیدیم که چگونه می توان با استفاده از این کلاسها اشیایی را ایجاد کرد و داده های موجود در آنها را به کنترلهای ساده ی موجود در فرم متصل کرد. در طول برنامه های این فصل بیشتر با کلاس DataView کار کردیم . دیدیم که چگونه می توان با استفاده از این کلاس در بین داده های موجود در فرم جستجو انجام داد و یا آنها را مرتب کرد.
مشاهده کردید که چگونه می توان داده های موجود در برنامه را به کنترلهای موجود در فرم متصل کرد و سپس با استفاده از کلاس CurrencyManager بین این داده ها جا به جا شده و آنها را کنترل کرد. کلاس CurrencyManager یک روش سریع و ساده برای کنترل حرکت بین داده های موجود در فرم را در برنامه فراهم می کند.
در این فصل برای دسترسی به داده ها، ایجاد داده های جدید، حذف داده های موجود و یا ویرایش آنها از روشهای دستی استفاده کرده و کد مربوط به تمام این موارد را خودمان در برنامه وارد کردیم. البته در مواردی که نیاز به سرعت بیشتر در نوشتن برنامه ها دارید، می توانید از ویزارد هایی که در فصل قبل با آنها آشنا شدیم استفاده کرده و فقط در موارد ضروری کد تولید شده به وسیله ی این ویزاردها را به صورت دستی تغییر دهید. استفاده از روشهایی که در این فصل با آنها آشنا شدیم در مواقعی مفید است که بخواهید

کنترل کاملی روی داده ها داشته باشید و یا مانند برنامه ای که در این فصل مشاهده کردیم، با جدولهایی پیچیده از داده های وابسته به هم در ارتباط باشید.
در پایان این فصل باید با موارد زیر آشنا شده باشید:

- به سادگی بتوانید از کلاسهای ADO.NET که در این فصل معرفی شد در برنامه های خود استفاده کنید.
- بدانید که چه مواقعی لازم است که از DataSet و چه مواقعی لازم است که از DataView استفاده کنید.
- بتوانید به صورت دستی و به وسیله ی کد خاصیت های موجود در کنترلهای برنامه را بـه داده هـای DataSet و یـا DataView متصل کنید.
- بتوانیـد بـا اسـتفاده از کـلاس CurrencyManager بـین داده هـای موجـود در DataSet و یـا DataView حرکت کنید.
- بتوانید با استفاده از کلاس DataView داده های موجود در برنامه را مرتب کرده و یا در بین آنها جستجو کنید.

تمرین:

تمرین ۱:

یک برنامه ی ویندوزی ایجاد کنید که داده های موجود در جدول Authors از بانک اطلاعـاتی Pubs را بـه کـاربر نمـایش دهد. برای نمایش داده ها از یک کنترل DataGridView در فرم برنامه استفاده کرده و برای دریافـت داده هـا نیـز دسـتور SQL ساده ی زیر را به کار ببرید:

```
SELECT * FROM Authors
```

فصل هفدهم: برنامه های مبتنی بر وب

امروزه، اینترنت یکی از بخشهای بسیار مهم و حیاتی در زندگی کاری و حتی شخصی بسیاری از افراد به شمار می رود. در چند سال اخیر، بانکها و فروشگاه های الکترونیکی باعث شده اند که در زمان و پول بسیاری از افراد صرفه جویی شود و دیگر نیازی نباشد که افراد برای خرید و یا تجارت با صرف وقت و زمان زیاد، به مکان های شلوغ رفت و آمد کنند. با وجود مشکلات امنیتی که در این نوع تجارت وجود دارد، اما هنوز بیشتر مردم به دلیل سادگی و راحتی آن و نیز صرفه جویی زیادی که در زمان و سرمایه ایجاد می کند، تمایل دارند که از این روش تجارت استفاده کنند. البته در مورد امنیت نیز در تمام موارد برنامه ها در حال بهبود هستند و روز به روز سعی می کنند که محیط ایمن تری را برای خرید و فروش و تجارت الکترونیکی ایجاد کنند. اما باز هم نمی توان مطمئن شد که یک محیط از امنیتی صد در صد برخوردار است.
با نگاهی به آینده می توان مطمئن شد که تجارت در اینترنت گسترش زیادی پیدا خواهد کرد، بنابراین برنامه نویسان باید بتوانند سایتهای دینامیک و قوی را برای این موارد ایجاد کنند. در این فصل سعی می کنیم با چگونگی ایجاد برنامه های مبتنی بر وب آشنا شویم. ابتدا سعی می کنیم مفهوم برنامه نویسی تحت وب را بیاموزیم، سپس به سمت برنامه نویسی بانک اطلاعاتی در وب حرکت کنیم. قبل از اینکه وارد نوشتن کد برای برنامه های مبتنی بر وب شویم، بهتر است کمی با اجزای تشکیل دهنده ی این نوع برنامه ها که برنامه نویسان از آنها استفاده می کنند آشنا شویم.
در این فصل:

- با مبانی برنامه نویسی مبتنی بر وب آشنا می شویم.
- مزایای فرمهای تحت وب بر فرمهای ویندوزی را مشاهده می کنیم.
- پردازش سمت سرور و پردازش سمت کلاینت را بررسی خواهیم کرد.
- با کنترلهای مربوط به بررسی صحت داده ها آشنا خواهیم شد.
- با کنترلهای مربوط به حرکت در سایت، تعیین هویت و نیز فرمهای اصلی آشنا خواهیم شد.
- از کنترل `GridView` استفاده کرده و یک فرم تحت وب برای نمایش داده ها ایجاد خواهیم کرد.

نکته: قسمتهای مربوط به کنترل خطا ها و استثنا ها از تمام قسمتهای امتحان کنید این فصل حذف شده اند تا مکان کمتری اشغال شود. اما هنگام نوشت این برنامه ها حتماً کد مربوط به کنترل خطا را نیز وارد کنید. برای مرور مطالب مربوط به کنترل خطا ها و استثنا ها می توانید به فصل یازدهم مراجعه کنید.

معماری برنامه های تحت وب:

در فصلهای قبلی با برنامه های مبتنی بر ویندوز آشنا شدیم و نحوه ی عملکرد آنها را مشاهده کردیم. در این برنامه ها بیشتر پردازش مربوط به برنامه، توسط کامپیوتر کاربری انجام می شد که برنامه را اجرا کرده است و به جز برنامه هایی که در چند فصل اخیر ایجاد کردیم، بیشتر این برنامه ها به صورت مستقل عمل می کردند و در طول زمان اجرا به برنامه و یا سرور دیگری نیاز نداشتند. اما در برنامه نویسی مبتنی بر وب شرایط کاملاً تفاوت دارد. در این نوع برنامه نویسی بیشتر پردازش توسط سرور انجام می شود و سپس نتیجه به کامپیوتر کاربر (کلاینت) برمی گردد و در مرورگر کاربر (مثلا اینترنت اکسپلورر) نمایش داده می شود.
هنگامی که یک برنامه ی تحت وب ایجاد می کنید، نیازی نیست که چیزی را به کاربران آن برنامه بدهید تا در سیستم خود نصب کنند و به واسطه ی آن بتوانند از برنامه ی شما استفاده کنند. بلکه فقط کافی است که این برنامه را روی یک سرور در شبکه ی وب

نصب کنید، سپس هر فردی که بتواند به این سرور وب شما متصل شود و همچنین در کامپیوتر خود یک مرورگر وب داشته باشد، به عنوان یکی از کاربران برنامه ی شما محسوب می شود و می تواند از آن برنامه استفاده کند. برای مثال یک موتور جستجو در اینترنت می تواند نمونه ی یک برنامه ی اینترنتی باشد. برای استفاده از این برنامه نیازی نیست که چیزی را در کامپیوتر خود نصب کنید، بلکه کافی است به شبکه ی اینترنت متصل باشید و بتوانید صفحات مربوط به آن موتور جستجو را در مرورگر خود مشاهده کنید. به این ترتیب شما نیز یکی از کاربران آن برنامه ی اینترنتی، و یا به عبارت دیگر آن برنامه ی تحت وب محسوب می شوید. همانطور که در این مثال نیز مشخص است در این نوع برنامه ها بیشتر پردازش در کامپیوتر سرور انجام می شود و سپس نتیجه ی آن به کامپیوتر کلاینت فرستاده می شود.

در این نوع برنامه ها اجباری نیست که تمام پردازش در سرور انجام شود و می توان مقداری را نیز در کامپیوتر کلاینت انجام داد. البته در این شرایط باید حجم پردازشی که به سمت کلاینت فرستاده می شود را نیز در نظر بگیریم. زیرا در این نوع برنامه ها، افراد و یا کاربران برنامه، از کامپیوتر ها و سیستمهای گوناگونی برای دسترسی به برنامه استفاده می کنند. بنابراین اگر بخواهیم حجم زیادی از کارهای برنامه را به وسیله ی کامپیوتر کاربر انجام دهیم، ممکن است بعضی از کاربران در استفاده از برنامه دچار مشکل شوند. این مورد یکی از تفاوتهای اصلی بین برنامه های مبتنی بر ویندوز و برنامه های مبتنی بر وب است. خوب، بهتر است مقداری با تفاوتهای این دو نوع برنامه در ویژوال استودیو NET. ۲۰۰۵ آشنا شویم.

هنگامی که از برنامه های مبتنی بر ویندوز صحبت می کنیم، معمولا یک برنامه ی کامپایل شده در اختیار داریم که فایل آن باید در اختیار کاربران برنامه قرار گیرد تا آنها بتوانند بعد از نصب برنامه در کامپیوتر خود، از آن استفاده کنند. بر حسب نوع برنامه ممکن است لازم باشد که یک یا چند فایل DLL و یا فایل EXE دیگر نیز همراه با برنامه توزیع شود تا افراد بتوانند از برنامه استفاده کنند.

در برنامه های مبتنی بر وب، معمولا نیازی نیست که هیچ فایل اجرایی و یا کتابخانه ی کلاسی بین کاربران آن برنامه توزیع شود. کاربران کافی است که بعد از اتصال به اینترنت، مرورگر خود را باز کرده و آدرس وب سایتی که برنامه در آن قرار دارد را وارد کنند. سروری که این برنامه ی تحت وب بر روی آن قرار داده شده است، مسئولیت دارد که تمام منابع مورد نیاز برای اجرای برنامه را در اختیار آن قرار دهد. به این ترتیب مرورگر کاربر، فقط وسیله ای است که فرد به وسیله ی آن می تواند بین صفحات برنامه جا به جا شود و همچنین نتایجی که از سرور برنامه دریافت می شود را مشاهده کند.

همچنین در این نوع برنامه ها تمام کد مربوط به اجرای برنامه در یک واحد متمرکز قرار می گیرد: کامپیوتر سروری که برنامه در آن قرار دارد. بنابراین می توانیم در صورت لزوم هر تغییری که مد نظر داشته باشیم را در کد موجود در این کامپیوتر اعمال کنیم. به این ترتیب مرتبه ی بعد که کاربر برای استفاده از برنامه به این سرور آمد از کد جدید استفاده خواهد کرد.

برنامه های مبتنی بر وب چندین مزیت کلیدی و اصلی دارند. اولین و مهمترین مزیت این نوع برنامه ها هزینه ی اولیه توزیع این نوع برنامه ها در بین کاربران است، زیرا تقریباً در این برنامه ها این کار هیچ هزینه ی ندارد. در مدل برنامه نویسی کلاینت/سرور قبلی، یک برنامه باید بین تمام کامپیوتر های کلاینت توزیع می شد تا کاربرانی که از این کامپیوتر ها استفاده می کنند بتوانند به برنامه ی سرور دسترسی داشته باشند. این کار، مخصوصا در مواردی که برنامه ی موجود در سرور می بایست به وسیله ی چندین کلاینت در دفاتر کار مختلف در چند نقطه از دنیا مورد استفاده قرار می گرفت کار بسیار پر هزینه ای بود.

یکی دیگر از مزایای عمده ی این نوع برنامه ها، هزینه های مربوط به توزیع نسخه های بروز رسانی[1] در بین این برنامه ها بود، زیرا این کار نیز هیچ هزینه ای دربر نداشت. کافی بود تمام فایلهای مربوط به به روز رسانی برنامه و قسمتهای مختلف آن، در کامپیوتر سرور که برنامه در آن قرار گرفته بود توزیع شود. به محض انجام این کار، نسخه ی جدید برنامه در دسترس تمام افرادی که از آن استفاده می کردند قرار می گرفت و مرتبه ی بعد که این افراد به سرور متصل می شدند از نسخه ی جدید برنامه استفاده می کردند. در مدل قدیمی کلاینت/سرور، برای به روز رسانی یک برنامه لازم بود که بسته های حاوی فایلهای به روز رسانی بین تمامی کلاینت ها توزیع شود و آنها نیز این بسته ها را در کامپیوتر خود نصب کنند. به این ترتیب این کار ممکن بود چندین روز و یا حتی چندین هفته زمان ببرد. اما در مدل برنامه های مبتنی بر وب، می توان یک برنامه را به روز کرد بدون اینکه حتی یکی از کاربران نیز متوجه اختلالی در برنامه شود.

---

[1] Update Patches

یکی دیگر از مزایای این برنامه ها این است که می توانید در معماری درونی برنامه ها تغییرات اساسی ایجاد کنید، بدون اینکه در مورد کاربرانی که از برنامه استفاده می کنند نگران باشید. برای مثال تصور کنید که می خواهید داده های برنامه را از یک سرور ضعیف، به یک سرور قوی و جدی منتقل کنید. قطعا سرور جدید دارای نام مخصوص به خود است. در مدل کلاینت/سرور قدیمی نام سروری که بانک اطلاعاتی برنامه در آن قرار داشت در کد موجود در سمت کلاینت نگهداری می شد. بنابراین برای تغییر سرور بانک اطلاعاتی، لازم بود که کد موجود در تمام کامپیوتر های کلاینت را تغییر دهید. اما در برنامه های مبتنی بر وب برای تغییر دادن سرور بانک اطلاعاتی مورد استفاده در برنامه، کافی است که تنظیمات سروری که برنامه روی آن قرار دارد را به گونه ای تغییر دهیم که از سرور بانک اطلاعاتی جدید استفاده کند. در این مدت نیز برنامه و تمام کاربرانی که در حال استفاده از آن هستند می توانند به راحتی به کار خود ادامه دهند.

همانطور که متوجه شدید در برنامه های مبتنی بر وب، هر فردی که یک اتصال به سروری که برنامه در آن قرار دارد و همچنین یک مرورگر وب داشته باشد می تواند یکی از کاربران برنامه به شمار آید و به آخرین نسخه ی برنامه دسترسی داشته باشد. حال که با اصول برنامه های مبتنی بر وب آشنا شدیم، بهتر است بررسی کنیم که این نوع برنامه ها در ویژوال استودیو چگونه کار می کنند؟

برنامه های تحت وب در مقایسه با برنامه های تحت ویندوز:

در این قسمت به بررسی مزایا و معایب برنامه های تحت ویندوز و نیز برنامه های تحت وب خواهیم پرداخت. به این ترتیب می توانید تشخیص دهید که در چه مواقعی باید از برنامه های تحت ویندوز و در چه مواقعی باید از برنامه ها تحت وب استفاده کرد. در اغلب موارد قبل از نوشتن یک برنامه باید تصمیم بگیرید که می خواهید از برنامه های مبتنی بر وب استفاده کنید و یا برنامه های تحت ویندوز را به کار ببرید. بنابراین بهتر است قبل از هر چیز درک درستی در مورد مزایا و نیز معایب این دو نوع برنامه بدست آورید تا بدانید که در چه شرایطی باید از برنامه های تحت وب و در چه شرایطی باید از برنامه های تحت ویندوز استفاده کنید.

مزایای برنامه های تحت ویندوز:

برنامه های تحت ویندوز دارای مزایای خاصی هستند. عموماً برنامه هایی که نیاز دارند به سرعت به در خواست کاربر پاسخ دهند، برای مثال برنامه هایی که در کامپیوتر های فروشگاه ها اجرا می شود، لازم است که از نوع برنامه های تحت ویندوز باشند. همچنین در بیشتر موارد برنامه هایی که پردازش زیادی را از پردازشگر درخواست می کنند، مانند بازی های کامپیوتری و برنامه های گرافیکی، لازم است که به صورت برنامه های تحت ویندوز نوشته شوند.

یکی از مهمترین مزیت های برنامه های تحت ویندوز، داشتن اعتبار کافی در آنها است. هنگامی که کاربری یک برنامه ی تحت ویندوز را نصب می کند، به این معنی است که به اندازه ی کافی به آن برنامه اعتماد دارد، بنابراین آن برنامه اجازه دارد که داده های مورد نیاز خود را در قسمتهای دلخواه از کامپیوتر ذخیره کرده و یا به تعیین وضعیت قسمتهای مختلف کامپیوتر بپردازد. در این موارد هنگامی که کاربر برنامه را اجرا کند، برنامه می تواند به رجیستری سیستم و یا فایلهای موجود در دیسک دسترسی داشته باشد. اما در برنامه های تحت وب، اعتباری که به برنامه داده می شود بسیار محدودتر است.

مزیت دیگر برنامه های تحت ویندوز کنترل کامل بر برنامه ی کلاینت است. به این ترتیب می توانید یک برنامه با رابط گرافیکی زیبا و قدرتمند طراحی کنید. در ادامه ی فصل مشاهده خواهید کرد که تعداد زیادی از کنترل هایی که در برنامه های تحت ویندوز وجود داشتند در برنامه های تحت وب وجود ندارند. به عبارت دیگر در برنامه های تحت ویندوز می توان رابط کاربری بسیار بهتری را ایجاد کرد.

همچنین سرعت عمل برنامه های تحت ویندوز یکی دیگر از مزایای آنها به شمار می رود. در این نوع برنامه ها به علت اینکه تمام پردازش در سمت کلاینت انجام می شود، بنابراین نیازی نیست که داده ها از طریق شبکه به سرور منتقل شوند و به این ترتیب از تاخیری که ممکن است به این علت در برنامه ها به وجود آید جلوگیری می شود. برای برنامه هایی که در کامپیوتر کلاینت اجرا می شوند، معمولا رویداد هایی که رخ می دهد سریعتر دریافت شده و متد مربوط به آنها اجرا می شود.

برنامه های تحت وب:

مزایای برنامه های تحت وب احتمالا بیشتر از مزایای برنامه های تحت ویندوز به نظر می رسند. اما این مورد نباید باعث شود که تمام تمرکز خود را روی برنامه های تحت وب متمرکز کنید و به یک برنامه نویس تحت وب تمام وقت تبدیل شوید. همیشه شرایطی وجود دارند که استفاده از برنامه های تحت ویندوز در آنها راه حل بهتری است.
مهمترین مزیت برنامه های تحت وب سادگی توزیع این برنامه ها بین کاربران آن است. برای توزیع این برنامه ها در بین کاربران، کافی است که آن را روی کامپیوتر سرور نصب کنید، همین. نیازی نیست که برای این برنامه ها، یک برنامه ی نصب ایجاد کرده و سپس آن را به وسیله ی CD و یا هر وسیله ی دیگری بین کاربران توزیع کنید. همچنین زمانی که بخواهید تغییری در برنامه ایجاد کنید، کافی است آن تغییر را در برنامه ی موجود در سرور اعمال کنید. به این ترتیب مرتبه ی بعد که کاربر بخواهد از برنامه ی شما در سرور استفاده کند، به آخرین نسخه از برنامه دسترسی خواهد داشت.
یکی دیگر از مزایای برنامه های تحت وب، کنترل نسخه ی برنامه است. به دلیل اینکه تمام کاربران برنامه از نسخه ی که در سرور وجود دارد استفاده می کنند، ایجاد تغییر در قسمتهای مختلف برنامه بسیار ساده خواهد بود. زیرا دیگر لازم نیست نگران این باشید که بعضی از افراد از نسخه ی ۸ و بعضی دیگر از نسخه ی ۱۰ برنامه ی شما استفاده می کنند. زیرا تمام افراد از نسخه ای از برنامه که در وب سرور قرار دارد استفاده می کنند.
آیا تاکنون واژه ی **مستقل از پلت فرم**[1] را شنیده اید؟ برنامه های تحت وب دارای چنین خاصیتی هستند. دیگر این موضوع که کاربر از چه نوع کامپیوتری استفاده می کند اهمیتی ندارد. همین که فردی یک اتصال به شبکه و نیز یک مرورگر وب داشته باشد، می تواند به عنوان کاربر برنامه ی شما محسوب شود. به عبارت دیگر مهم نیست که کاربر از سیستم عامل ویندوز برای دسترسی به برنامه استفاده می کند و یا از سیستم عامل لینوکس و یا ....
این مزایا می توانند باعث صرفه جویی بیش از میلیونها دلار هزینه نسبت به برنامه های تحت ویندوز شوند. توانایی ایجاد سریع تغییرات و نیز نگهداری و کنترل ساده ی کد این برنامه ها، از مهمترین مزایای آنها به شمار می روند. البته همانطور که گفتم در شرایطی برنامه های تحت وب، اجازه ی طراحی رابط کاربری مناسب را به برنامه نویس نمی دهند. بنابراین قبل از طراحی یک برنامه باید در مورد نوع آن تصمیم صحیحی گرفته شود.

اجزای اصلی برنامه های تحت وب:

در ساده ترین حالت، یک برنامه ی تحت وب شامل چندین صفحه ی وب است. همچنین برای اینکه کاربران بتوانند به این صفحات دسترسی داشته باشند، به یک سرور و یک مرورگر اینترنتی نیز نیاز داریم. معمولا مرورگر درخواستی را ایجاد می کند که داده های یکی از صفحات وب موجود در سرور را دریافت کند. سپس سرور داده هایی که باید در آن صفحه باشند را ایجاد کرده و نتیجه را به سمت مرورگر می فرستد. بنابراین کاربر می تواند داده های صفحه ی درخواستی را در پنجره ی مرورگر مشاهده کند. صفحه ای

[1] Platform Independence

کاربر مشاهده می کند معمولا شامل کد های HTML، CSS و یا اسکریپت های سمت کلاینت است. در این قسمت سعی می کنیم که با هر یک از این قسمتهای موجود در یک برنامه ی تحت وب آشنا شویم.

سرور وب:

سرور وب[1]، برنامه ای است که روی یکی از کامپیوتر های قوی که دائما به اینترنت متصل است نصب می شود و مسئول کنترل و مدیریت برنامه های تحت وبی است که در آن کامپیوتر قرار دارند. امروزه سرور های وب مختلفی در بازار وجود دارند که از معروفترین آنها می توان IIS[2] و یا Apache را نام برد. در این کتاب فقط بر روی وب سرور IIS تمرکز خواهیم کرد.

مرورگر:

هر کاربر یک برنامه ی تحت وب بایستی یک مرورگر داشته باشد. چهار مرورگری که امروزه کاربرد بیشتری دارند عبارتند از: Microsoft Internet Explorer، Mozilla FireFox، Netscape و Opera.
هنگامی که در حال طراحی یک برنامه ی وب عمومی هستید باید در نظر داشته باشید که صفحات این برنامه ممکن است در مرورگرهای مختلف به شکلهای متفاوت نمایش داده شود. در تمرینات این کتاب بیشتر بر روی مرورگر IE نسخه ی 6 تمرکز خواهیم کرد.

HTML:

HTML یا HyperText Markup Language، کدی است که قالب نمایش صفحات وب را مشخص می کند. کد های HTML همانند یک زبان برنامه نویسی هستند که با استفاده از تگ های مختلف می توانند نحوه ی نمایش داده ها در یک صفحه ی وب را مشخص کنند. برای مثال در HTML برای اینکه متنی را در یک صفحه ی وب به صورت Bold نمایش دهیم، کافی است از تگ استفاده کنیم. برای مثال متن زیر یک نمونه از کد HTML است:

```
This is <b>bold</b> in HTML.
```

اگر کد HTML قبل به وسیله ی یک مرورگر نمایش داده شود، متنی مشابه زیر دیده خواهد شد:

This is **bold** in HTML.

مرورگر ها باید کد HTML را تفسیر کرده و نتیجه ی آن را به کاربر نمایش دهند و نیز برای دستورات مورد استفاده ی خود نیز باید از استاندارد W3C[3] استفاده کنند. W3C در سال ۱۹۹۰ برای ایجاد پروتکلهای عمومی برای استفاده در وب به وجود آمد. برای اطلاعات بیشتر در مورد این سازمان می توانید به سایت آن به آدرس www.w3.org مراجعه کنید.

---

[1] Web Server
[2] Microsoft Internet Information Server
[3] World Wide Web Consortium

با وجود اینکه هنگام طراحی برنامه های تحت وب با استفاده از ویژوال استودیو ۲۰۰۵ نیازی به دانستن HTML نیست، اما در طول تمرینات این فصل با کد های آن بیشتر آشنا خواهید شد.

VBScript و JavaScript:

یکی از بخشهای عمده ی برنامه های تحت وب، اسکریپت های سمت کلاینت در این برنامه ها است. همانطور که گفتم در برنامه های تحت وب لزومی ندارد که تمام پردازش در سمت سرور انجام گیرد و می توان بعضی از آنها را در سمت کلاینت انجام داد. برای اجرای پردازشی در سمت کلاینت باید از اسکریپت ها استفاده کنیم. دو زبان اسکریپت نویسی عمومی که در بیشتر برنامه های تحت وب به کار می روند عبارتند از VBScript و JavaScript. اگر بخواهیم کد اسکریپت موجود در یک برنامه بتواند توسط همه ی مرورگر ها اجرا شود باید از زبان JavaScript استفاده کنیم. VBScript زبانی است که ساختاری مشابه Visual Basic دارد و فقط به وسیله ی مرورگر Internet Explorer به صورت کامل پشتیبانی می شود.
کد های اسکریپتی که در سمت کلاینت اجرا می شوند، بیشتر برای تایید داده های وارد شده به وسیله ی کلاینت و یا تولید کد های HTML دینامیک[1] به کار می رود. اسکریپت های مربوط به تایید صحت داده ها این امکان را به ما می دهند که بتوانیم کاربر را مجبور کنیم قبل از ادامه، قسمتهای خاصی از صفحه را تکمیل کند. برای مثال اگر کاربر در حال کامل کردن فرمی در برنامه باشد، می توانیم او را مجبور کنیم قبل از فشار داد دکمه ی Submit، کادرهای لازم در فرم را کامل کند. اسکریپت های مربوط به تولید دینامیک کد HTML نیز باعث می شوند کد HTML موجود در صفحه، در زمان نمایش داده شدن آن تغییر کند.
یکی از مهمترین ویژگی طراحی برنامه های تحت وب با استفاده از ویژوال استودیو، کنترلهای موجود برای تایید صحت داده های وارد شده در فرم و یا کنترلهای مربوط به حرکت در بین صفحات برنامه است. به این ترتیب می توانید این کنترل ها را در فرم برنامه قرار داده و بدون اینکه حتی یک خط نیز اسکریپت بنویسید، صحت داده های ورودی در برنامه را در سمت کلاینت تعیین کنید. با وجود این در ادامه ی این فصل مقداری نیز با نوشتن اسکریپت های سمت کلاینت آشنا خواهیم شد.

CSS:

CSS یا Cascading Style Sheets این اجازه را در برنامه می دهند که بتوانیم استیل و قالب صفحات را از محتویات آنها جدا کنیم. با استفاده از CSS می توانیم به سادگی فونت، رنگ، نحوه ی قرار گیری متنها و یا بسیاری از ویژگیهای دیگر محتویات صفحات وب را تغییر دهیم. بهترین خاصیت CSS این است که به وسیله ی آن می توانیم یک قالب کلی برای صفحات طراحی کرده و آن را در سرتاسر برنامه مورد استفاده قرار دهیم. به این وسیله می توانیم به سادگی با تغییر کد CSS موجود، ظاهر تمام قسمتهای برنامه را تغییر دهیم. در مورد CSS در ادامه ی این فصل بیشتر صحبت خواهیم کرد.

ASP:

[1] Dynamic HTML - DHTML

ASP یک پلت فرم و قالب کاری برای ایجاد برنامه های مبتنی بر وب است. همراه با ویژوال استودیو ۲۰۰۵ یک نسخه ی جدید از ASP به نام ASP.NET 2.0 معرفی شد. به وسیله ی این نسخه ی جدید به سادگی می توان سایتهای دینامیک ایجاد کرد. در این قسمت مفهوم ASPX و یا همان فرمهای تحت وب را توضیح دهیم.

مزایا:

برای ایجاد برنامه های تحت وب، از چند روش مختلف می توانید استفاده کنید. معروفترین روشهای انجام این کار عبارت است از ایجاد فایلهایی از یکی از انواع زیر:

- Active Server Pages (فایلهای asp. و یا aspx.)
- Java Server Pages (فایلهای jsp.)
- ColdFusion Pages (فایلهای cfm.)
- فایلهایی حاوی دستورات HTML ثابت (فایلهای html. و یا htm.)

در این فصل بر روی ایجاد فایلهایی از نوع aspx. تمرکز خواهیم کرد. این نوع فایلها مربوط به برنامه های تحت وبی هستند که با استفاده از ASP.NET 2 ایجاد می شوند. البته در برنامه ها مقداری هم از دستورات HTML استفاده می کنیم.
یکی از مهمترین مزایای فایلهایی که با استفاده از ASP.NET 2 ایجاد می شوند نسبت به فایلهای دیگری که معرفی شد در این است که، سرعت اجرای برنامه های ASP.NET 2 نسبت به موارد مشابه آن بالاتر است. دلیل این امر نیز به علت نحوه ی کامپایل شدن دستورات آن در سرور است.
همچنین با استفاده از امکانات ویژوال استودیو ۲۰۰۵ (که تاکنون با بعضی از آنها در برنامه نویسی ویندوز آشنا شده اید) برای ایجاد برنامه های تحت وب، باعث افزایش سرعت در طراحی برنامه ها و نیز افزایش کارایی آنها می شود. با استفاده از توابع و کلاسهای زیادی که در فضای نامهای مختلف .NET وجود دارند می توان به سادگی و به سرعت برنامه های کاربردی قدرتمندی طراحی کرد.

فایلهای خاص در یک برنامه ی تحت وب:

هنگامی که در ویژوال استودیو بخواهید با استفاده از ASP.NET 2 یک برنامه ی تحت وب ایجاد کنید، فایلهای مخصوص زیادی را مشاهده خواهید کرد. این فایلها از اهمیت زیادی برخوردارند و توضیح هر یک از آنها به یک فصل جداگانه نیاز دارد. دو فایلی که در زیر با آنها آشنا می شویم، فایلهایی هستند که می توانید با ایجاد تغییراتی در آنها در سرتاسر برنامه تاثیر بگذارید. برای اطلاعات بیشتر در مورد این فایلها می توانید به سیستم راهنمای ویژوال استودیو (MSDN) و یا سایت اینترنتی http://msdn2.microsoft.com مراجعه کنید.

فایل Global.asax:

به وسیله ی این فایل می توانید کدی را در رویدادهای مربوط به کل برنامه اجرا کنید. مهمترین این رویدادها شامل Application_Start، Application_End، Session_Start، Session_End و نیز Application_Error می شوند. رویدادهای Application_Start و نیز Application_End زمانی رخ می دهند که برنامه به وسیله ی IIS شروع شود و یا بسته شود. یک برنامه ی تحت وب زمانی شروع می شود که اولین کاربر به سرور متصل شده و بخواهد از آن استفاده کند. همچنین این برنامه زمانی بسته می شود که هیچ کاربری در حال استفاده از آن نباشد. رویدادهای Session_Start و Session_End زمانی رخ می دهند که یک کاربر جدید بخواهد از برنامه استفاده کند. به عبارت دیگر زمانی که یک کاربر به سرور متصل شده و بخواهد از برنامه استفاده کند، رویداد Session_Start و هنگامی که کاربر بخواهد از برنامه خارج شود رویداد Session_End فراخوانی می شود. بنابراین این رویدادها در طول اجرای برنامه می توانند چند بار فراخوانی شوند. البته هنگام استفاده از این رویداد ها باید در نظر داشته باشید به علت اینکه بازای هر کاربر یک بار این رویداد فراخوانی می شود، ممکن است فشار کاری زیادی را بر سرور متحمل کند. رویداد آخر نیز رویداد Application_Error است و زمانی فراخوانی می شود که خطایی در برنامه رخ دهد، اما هیچ قسمتی برای کنترل آن در برنامه وجود نداشته باشد. به این وسیله می توانید به صورت متمرکز، خطاهای عمومی را کنترل کرده و کاربر را به صفحات خطای مناسب بفرستید.

فایل web.config:

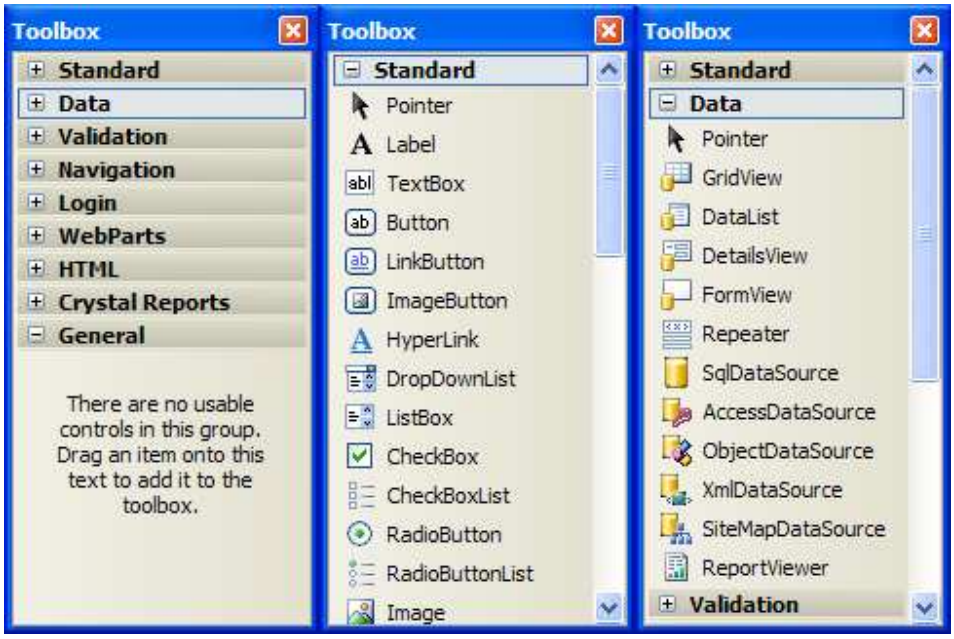
فایل web.config محلی برای ذخیره تنظیمات و پیکر بندی برنامه است که به صورت یک سند XML ایجاد می شود. در این قسمت می توانید تنظیمات مربوط به امنیت برنامه، خطاهای احتمالی در برنامه و بسیاری دیگر را مشخص کنید. در طی این فصل، فقط اطلاعات مربوط به اتصال به بانک اطلاعاتی (ConnectionString) را در این فایل نگهداری می کنیم.

استفاده از محیط ویژوال استودیو:

برای طراحی فرم در برنامه های تحت وب، می توانیم از محیط طراحی ویژوال استودیو که در هنگام طراحی برنامه های تحت ویندوز با آن آشنا شدیم استفاده کنیم. هنگام ایجاد این فرم ها، می توانیم صفحات را به گونه ای که با نام code-behind شناخته می شوند ایجاد کنیم. به این ترتیب کد HTML تشکیل دهنده ظاهر فرم (که به سمت کلاینت فرستاده می شود)، از کدی که برای عملکرد قسمتهای فرم نوشته می شود (که معمولا از زبانهای برنامه نویسی مانند C# و یا VB استفاده می کند و در سمت سرور اجرا می شوند) جدا خواهد شد. در هنگام طراحی یک فرم وب، سه نمای مختلف در محیط ویژوال استودیو وجود دارد: Design، Source و Code View. اطلاعات مربوط به نماهای Design و Source در فایلی با پسوند aspx. ذخیره شده و شامل کد HTML تشکیل دهنده ی ظاهر فرم و یا کد اسکریپت لازم برای تایید صحت داده ها است. نمای Code View نیز کد اصلی فرم را که در فایل cs. ذخیره شده و به زبان C# است را نمایش می دهد. این فایل حاوی کد هایی از فرم است که سمت سرور اجرا می شوند و پردازش اصلی صفحه را بر عهده دارند، به عبارت دیگر کد مربوط به وظیفه ی اصلی فرم در این فایل وارد می شود.

کنترلهای موجود در جعبه ابزار:

هنگامی که بخواهید یک برنامه ی تحت وب با استفاده از ویژوال استودیو ایجاد کنید، کنترلهای عمومی که برای این کار مورد استفاده قرار می گیرند در جعبه ابزار نشان داده خواهند شد. اگر جعبه ابزار در محیط ویژوال استودیو دیده نمی شود، کلیدهای Ctrl+Alt+X را فشار دهید. مشاهده خواهید کرد که کنترل ها در گروهای مختلفی دسته بندی شده اند. در شکل ۱۷-۱ نام این گروه ها به همراه تعدادی از کنترل ها نمایش داده شده است. در سمت چپ گروه های کنترل ها دیده می شود، در وسط کنترلهای موجود در گروه Standard و در سمت راست نیز کنترلهای موجود در گروه Data نمایش داده شده اند.



شکل ۱۷-۱

جعبه ابزار ویژوال استودیو به طور کامل قابل تنظیم است و می توانید با استفاده از منویی که با کلیک راست روی آن نمایش داده می شود، کنترلهای مورد نظر خود را به آن اضافه کرده و یا از آن حذف کنید. همچنین می توانید قطعه کدی که زیاد آن را در برنامه استفاده می کنید، در جعبه ابزار قرار داده و سپس با کلیک کردن روی آن، قطعه کد مورد نظر را در برنامه قرار دهید. برای اضافه کردن کد به جعبه ابزار می توانید آن را انتخاب کرده و سپس با ماوس آن را بکشید و در جعبه ابزار رها کنید. سپس روی آن کلیک راست کرده و Rename را انتخاب کنید تا بتوانید نام آن را به نام با مفهوم تری تغییر دهید. برای استفاده از این کد در برنامه نیز کافی است که مکان نما را به جایی که می خواهید کد در آن قرار گیرد ببرید و سپس با ماوس روی کد موجود در جعبه ابزار دو بار کلیک کنید. در طی این فصل کنترلهای موجود در اغلب این گروه ها را، به جز گروه های Crystal Reports، Login و Web Parts در برنامه به کار خواهیم برد.

ایجاد برنامه ها تحت وب:

در این قسمت ابتدا یک برنامه ی تحت وب ساده ایجاد خواهیم کرد تا با جنبه های مختلف این کار آشنا شویم.

ایجاد یک فرم وب برای پردازش سمت سرور و سمت کلاینت:

برنامه ای که در بخش امتحان کنید بعد ایجاد خواهیم کرد، دارای یک فرم وب خواهد بود که در آن از کنترلهای HTML و نیز کنترلهای سمت سرور استفاده شده است. کنترلهای HTML پردازش را در سمت کلاینت انجام می دهند، اما کنترلهای سمت سرور، پردازش خود را به سرور منتقل می کنند.

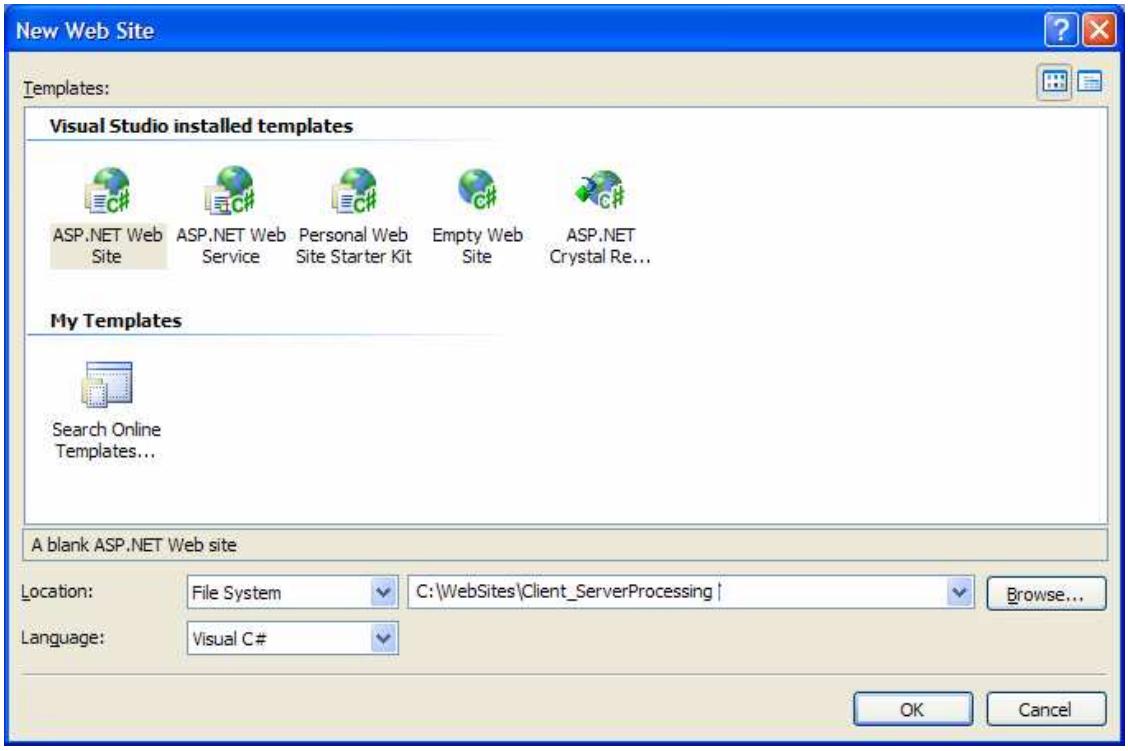
امتحان کنید: پردازش سمت سرور و سمت کلاینت

۱) با انتخاب گزینه ی File ➔ New Web Site... یک پروژه ی جدید را ایجاد کنید. در کادر New Web Site ویژوال #C را به عنوان زبان انتخاب کرده و از قسمت Templates نیز گزینه ی ASP.NET Web Site را انتخاب کنید. در قسمت Location گزینه ی File System را انتخاب کرده و در کادر مقابل آدرس C:\WebSites\Client_ServerProcessing را انتخاب کنید. در این صورت با کلیک کردن روی دکمه ی OK یک وب سایت در کامپیوتر محلی ایجاد خواهد شد که از وب سرور درونی ویژوال استودیو برای تست برنامه استفاده می کند. فرم ایجاد وب سایت جدید همانند شکل ۱۷-۲ خواهد بود.

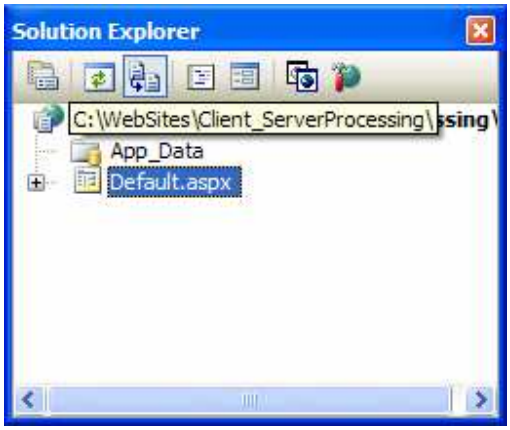
۲) ویژوال استودیو فایلها و فولدر های پیش فرض را برای یک وب سایت ایجاد خواهد کرد. پنجره ی Solution Explorer را در این پروژه مشاهده کنید. این پنجره همانند شکل ۱۷-۳ خواهد بود. همچنین صفحه ی مربوط به فایل default.aspx نیز در محیط طراحی باز شده است.

۳) حال باید کنترلهای لازم را به فرم اضافه کنیم. به حالت Design بروید و کنترلهای زیر را به فرم Default.aspx اضافه کنید (برای رفتن به حالت Design در حالی که فایل Default.aspx را مشاهده می کنید، یا از پایین محیط طراحی روی عبارت Design کلیک کنید و یا کلیدهای Shift+F7 را فشار دهید). فعلا لازم نیست که در مورد مکان کنترل ها نگران باشید، اما حتماً کنترل ها را از گروه های Standard و HTML در جعبه ابزار انتخاب کنید.

- از گروه Standard یک کنترل Button و دو کنترل Label را به فرم برنامه اضافه کنید.
- از گروه HTML نیز یک کنترل Input(Button) به برنامه اضافه کنید.



شکل ۱۷-۲



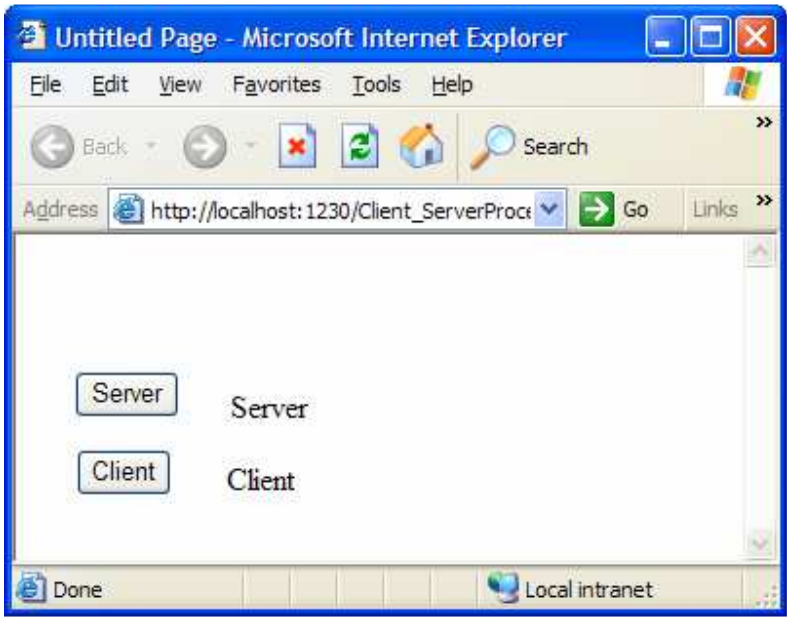
شکل ۱۷-۳

۴) حال خاصیتهای کنترل ها را به گونه ای تغییر دهید که فرم برنامه همانند شکل ۱۷-۴ شود.

- خاصیت `ID` از کنترل `Standard:Button` را به `btnServer` و خاصیت `Text` آن را به `Server` تغییر دهید.
- خاصیت `ID` از کنترل `HTML:Input(Button)` را به `btnClient` و خاصیت `Value` آن را به `Client` تغییر دهید.
- خاصیت `ID` از کنترل `Standard:Label` که در بالا قرار گرفته است را برابر با `lblServer` و خاصیت `Text` آن را برابر با `Server` قرار دهید.

- خاصیت ID از کنترل Standard:Label که در پایین قرار گرفته است را برابر با lblClient و خاصیت Text آن را برابر با Client قرار دهید.

۵) حال با پایین نگه داشتن کلید Ctrl، چهار کنترل موجود در فرم را انتخاب کرده، سپس گزینه ی Layout ➔ Position ➔ Absolute را از نور منوی ویژوال استودیو انتخاب کنید. به این ترتیب می توانید مکان کنترل ها را همانند فرم های تحت ویندوز در فرم برنامه مشخص کنید. حال موقعیت کنترل ها در فرم را همانند شکل ۴-۱۷ تنظیم کنید. بعد از اتمام این کار کلیدهای Ctrl+F5 را فشار دهید تا برنامه بدون دیباگ کردن اجرا شود و فرم برنامه را در مرورگر خود مشاهده کنید.



شکل ۴-۱۷

۶) مرورگر را ببندید تا مجدداً به ویژوال استودیو برگردید. بر روی دکمه ی btnServer دو بار کلیک کنید تا متد مربوط به رویداد Click این کنترل ایجاد شود. بسته به تنظیماتی که انجام داده اید، این متد یا در یک فایل کد مجزا ایجاد خواهد شد (روش Code-Behind) و یا در همان فایل .aspx و در کنار کد های HTML مربوط به ظاهر برنامه ایجاد می شود. کد مشخص شده در زیر را به این رویداد اضافه کنید:

```
protected void btnServer_Click(object sender, EventArgs e)
{
    lblServer.Text = "Changed";
}
```

با فشار دادن کلیدهای Ctrl+F5 مجدداً برنامه را اجرا کرده و سپس روی دکمه ی Server کلیک کنید. مشاهده خواهید کرد که متن درون کنترل Label به عبارت Changed تغییر می کند.

۷) خوب، حالا همین مراحل را برای کنترل HTML:Input(Button) انجام می دهیم و همچنین عنوان فرم را نیز عوض می کنیم. در حالی که در حالت Design هستید، از کادر بالای پنجره ی Properties گزینه ی Document را انتخاب کرده و سپس خاصیت Title را به My First Page تغییر دهید. حال باید کد مربوط به دکمه ی Client را وارد کنیم. همانطور که در قسمت قبل نیز گفتم به علت اینکه این کد باید در سمت کلاینت انجام شود، پس باید به یکی از زبانهای اسکریپت نویسی نوشته شود. در اینجا از زبان اسکریپت نویسی JavaScript استفاده می کنیم، زیرا هم شباهت زیادی به C# دارد و هم توسط همه ی مرورگر ها پشتیبانی می شود. پس به حالت Source بروید و از کادر سمت چپ بالای قسمت کد نویسی (کادر Client Object & Events) گزینه ی btnClient و از کادر سمت راست گزینه ی onclick را انتخاب کنید. به این ترتیب متد مربوط به رویداد onclick کنترل btnClient ایجاد می شود. کد زیر را در این رویداد وارد کنید:

```
function btnClient_onclick() {
    document.getElementById("lblClient").innerText =
                                                "Changed";
    document.getElementById("lblServer").innerText =
                                                "Server";
}
```

۸) برنامه را با فشار دادن کلید Ctrl+F5 اجرا کرده و عملکرد هر دو دکمه را تست کنید.

چگونه کار می کند؟

خوب، مشاهده کردید که ایجاد برنامه های تحت وب شباهت زیادی به ایجاد برنامه های تحت ویندوز دارد. این یکی از مزایای ویژوال استودیو .NET به شمار می رود که یک برنامه نویس تحت وب، می تواند با خواندن مطالب بسیار کمی به برنامه نویسی تحت ویندوز بپردازد و یا بر عکس.
بهتر است برای بررسی برنامه ابتدا از کد HTML نوشته شده برای آن شروع کنیم. اولین خط این کد، راهنمای Page[1] نامیده می شود:

```
<%@ Page Language="C#" AutoEventWireup="true"
     CodeFile="Default.aspx.cs" Inherits="_Default" %>
```

بر حسب تنظیماتی که برای نوشتن برنامه انجام می دهید، ویژوال استودیو خصیصه های مختلفی را در این قسمت تنظیم می کند. راهنمای Page که در این قسمت مشاهده می کنید، دارای بیش از ۳۰ خصیصه ی مختلف است که می توان آنها را با مقادیر گوناگونی تنظیم کرد. در این قسمت فقط درباره ی خصیصه هایی که به صورت پیش فرض در این راهنما وجود دارد صحبت خواهیم کرد. برای مطالعه در مورد بقیه ی این موارد می توانید در سیستم راهنمای ویژوال استودیو (MSDN) و یا در سایت http://msdn2.microsoft.com عبارت @Page را جستجو کنید.
اولین خصیصه ای که در راهنمای @Page وجود دارد، خصیصه ی Language است. در این قسمت زبان برنامه نویسی که کد های سمت سرور این فایل با آن نوشته شده اند را مشخص می کنید. دومین خصیصه AutoEventWireup است و

[1] Page Directive

مقدار پیش فرض آن برابر با False است. مقدار پیش فرض این خصیصه برابر با true است و اگر آن را برابـر بـا false قرار ندهید، متدهای مربوط به رویداد های خاصی ممکن است دو بار فراخوانی شوند. مایکروسافت توصیه می کنـد کـه همـواره ایـن خصیصه را برابر با true قرار دهید. خصیصه ی بعدی CodeFile است. اگر این فایل به صـورت Code-Behind نوشته شود، یعنی کد هایی از آن که باید در سمت سرور اجرا شوند در یک فایل جداگانـه ذخیـره شـده اسـت. در ایـن صـورت ایـن خصیصه نام فایل حاوی کد سمت سرور را نگهداری می کند. آخرین خصیصه نیز، خصیصه ی Inherits است و شـامل نـام کلاسی است که صفحه ی وب جاری از آن مشتق می شود.

خط بعدی، خط DOCTYPE ! است. این خط به مرورگر IE نسخه ی ۶ و بالاتر اعلام می کند که ایـن فایـل بـا XHTML 1.1 که به وسیله ی W3C برای زبان انگلیسی مشخص شده است هماهنگی دارد.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional
//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
```

خط بعدی اولین خطی است که شامل دستور HTML است. در بیشتر مواقع خصیصه ی خاصی درون این دستور تنظیم نمی شود. در اینجا ویژوال استودیو درون این دستور خصیصه ای را تنظیم می کند تا مشخص کند که فضای نام مربوط به تگهای ایجاد شـده بـه وسیله ی کاربر برابر با "http://www.w3.org/1999/xhtml" است. اگر به این سایت بروید، فـضای نـام تعریف شده ی XHTML به وسیله ی W3C را مشاهده خواهید کرد.

```
<html xmlns="http://www.w3.org/1999/xhtml" >
```

دستور بعدی که در فایل قرار دارد، دستور HEAD است. عناصری که درون این تگ قرار می گیرنـد، در صـفحه نمـایش داده نمـی شوند اما ممکن است در ظاهر صفحه تاثیر داشته باشـند. در ایـن تـگ مـی تـوانیم از تگهـای SCRIPT، META، TITLE، LINK، STYLE و یا چندین تگ دیگر استفاده کرده و ظاهر برنامه را تنظیم کنیم. تگهای LINK و STYLE در این بخش، هر دو برای اضافه کردن کد های CSS به کار می روند. در ادامه ی این فصل بیشتر با کد های CSS آشنا خواهیم شد.

اولین تگی که در قسمت HEAD قرار دارد TITLE است. این تگ عنوان پنجره ی مرورگری را مشخص می کند که صـفحه در آن نمایش داده می شود. بعد از آن نیز یک تگ SCRIPT قرار دارد و دستورات JavaScript کـه بـرای اجـرا شـدن در سمت کلاینت نوشته بودیم در این قسمت قرار گرفته اند.

تگ script قسمتی را تعریف می کند تا بتوانیم دستوراتی را که می خواهیم در سمت کلاینت اجرا شوند را در آن قـرار دهـیم. در این برنامه تنها متدی که ایجاد کرده ایم، متد مربوط به رویـداد onclick از کنتـرل btnClient اسـت. زمـانی کـه کاربر روی این دکمه کلیک کند، دستورات نوشته شده در ایـن متـد اجـرا مـی شـوند. در خـط اول ایـن متـد نیـز بـا فراخـوانی متـد getElementById، کنترلی را در فرم پیدا می کنیم که شناسه ی آن برابر با lblClient باشد. بعد از پیدا کـردن این کنترل در فرم، خاصیت innerText آن را برابر با Changed قرار می دهیم. در خط بعد نیز همین روش را برای پیدا کردن کنترل lblServer به کار برده و خاصیت innerText آن را برابر با Server قرار می دهیم.

```
<head runat="server">
    <title>My First Page</title>
<script language="javascript" type="text/javascript">
// <!CDATA[

function btnClient_onclick() {
```

```
    document.getElementById("lblClient").innerText =
                                                "Changed";
    document.getElementById("lblServer").innerText =
                                                "Server";
}

// ]]>
</script>
</head>
```

نکته ای که ممکن است به آن توجه نکرده باشید، روشی است که دو کنترل Button در فرم برای فراخوانی رویداد های خود استفاده می کنند. البته متوجه شدن این مورد هنگامی که سایت ایجاد شده را در کامپیوتر خودتان اجرا می کنید کار سختی است، اما مجدداً به صفحه ی وب در مرورگر برگردید و هنگامی که روی این دو دکمه کلیک می کنید نوار وضعیت را مشاهده کنید. مشاهده خواهید کرد هنگامی که روی دکمه ی Server کلیک می کنید، نوار سبز رنگی در پایین مرورگر پر می شود (البته با سرعت). این مورد نشان می دهد که هنگامی که روی دکمه ی Server کلیک می کنید، صفحه به سمت سرور فرستاده شده و پردازش مربوط به این رویداد در سرور انجام می شود. اما زمانی که روی دکمه ی Client کلیک کنید، پردازش مربوط به آن در سمت کلاینت انجام می شود و لازم نیست که هیچ داده ای به سمت سرور فرستاده شود. تفاوت این دو مورد زمانی به شدت مشخص می شود که بخواهید از طریق یک اتصال Dial-Up به اینترنت متصل شده و از این سایت استفاده کنید، بنابراین باید زمانی که کد های خود را به صورت runat="server" می نویسید این نکته را نیز در نظر داشته باشید.
بعد از تگ HEAD وارد تگ BODY خواهیم شد. این تگ جایی است که ویژوال استودیو کنترلهای موجود در فرم را قرار می دهد.

```
<body>
    <form id="form1" runat="server">
```

مشاهده می کنید که در تگ فرم، خصیصه ی runat به وسیله ی ویژوال استودیو برابر با server قرار گرفته است. این خصیصه مشخص می کند که در این فرم کنترل هایی قرار گرفته است که باید در سرور پردازش شوند.
بهتر است قبل از ادامه ی بررسی برنامه، مقداری درباره ی کد HTMLای که در حال مشاهده ی آن هستید و تفاوت آن با کد HTMLای که به وسیله ی مرورگرها درک شده و قابل نمایش داده شدن است صحبت کنیم. کدی که در این قسمت توسط ویژوال استودیو تولید می شود، کد HTML استاندارد نیست. بنابراین اگر این کد را در یک فایل متنی وارد کرده و سپس آن را با پسوند .htm و یا .html ذخیره کنید و بخواهید که فایل ایجاد شده را به وسیله ی یک مرورگر مشاهده کنید، نتیجه ی مطلوب را دریافت نخواهید کرد (می توانید این کد را در برنامه ای مانند notepad وارد کرده و سپس در فایل با پسوند .html ذخیره کنید. سپس آن را با مرورگر اینترنت خود باز کنید تا ببینید که چه چیزهایی نمایش داده می شوند).
در توضیح این مورد می توان گفت که زبان HTML دارای کنترلهای محدودی است که طراحی صفحات اینترنتی پیچیده با استفاده از این کنترل ها بسیار مشکل است (کنترل هایی که در گروه HTML در جعبه ابزار وجود دارند، کنترلهای موجود در زبان HTML هستند). ویژوال استودیو برای اینکه کار طراحی صفحات وب را ساده تر کند، با استفاده از کنترلهای HTML استاندارد، کنترلهای جدید دیگری را ایجاد کرده است که به کنترلهای سرور معروف هستند و باید در سمت سرور پردازش شوند. برای مثال ویژوال استودیو کنترل GridView را ایجاد کرده است و تا حد ممکن سعی کرده است که این کنترل مشابه کنترل DataGridView در برنامه های ویندوزی باشد تا برنامه نویس بتواند به سادگی از آن در برنامه های خود استفاده کند. اما زبان HTML استاندارد چنین کنترلی ندارد و ویژوال استودیو وسیله ی کنترلهای دیگری مانند Table و ...، کنترل GridView مورد نظر را ایجاد می کند.

بنابراین تمام تگ هایی که با ASP شروع می شوند در حقیقت کنترلهای مخصوص ویـژوال اسـتودیو هـستند. هنگـامی کـه کـاربر درخواست مشاهده ی یکی از این صفحات را به سرور می فرستد، ASP.NET کد آن صـفحه را اسـتخراج کـرده و کـد HTML معادل آن را تولید می کند، به عبارت دیگر به جای کنترلهای مخصوص ویژوال استودیو کنترلهـای اسـتاندارد HTML را قـرار مـی دهد. سپس کد HTML ایجاد شده را به سمت کلاینت می فرستد تا در مرورگر نمایش داده شود.

نکته: برای بررسی این مورد می توانید یک صفحه ی وب شامل چندین کنترل سرور مختلف ایجاد کرده و سـپس آن صـفحه را بـه وسیله ی ویژوال استودیو اجرا کنید. بعد از اینکه صفحه در مرورگر نمایش داده شد، در اینترنت اکسپلورر از منوی View گزینـه ی Source را انتخاب کنید. کدی که در این قسمت نمایش داده می شود، کد HTML تولیـد شـده بـه وسـیله ی ASP.NET است که برای کنترلهای مورد نظر شما ایجاد شده است.

خوب، با این توضیح می توانیم به ادامه ی بررسی برنامه برگشته و آن را با دقت بیشتری مرور کنیم. هنگامی کـه در صـفحه ی وب روی دکمه ی Server کلیک می کنید، باید فرم به سمت سرور فرستاده شود تا پردازش مربوط به این دکمـه در سـمت سـرور انجام شده و نتیجه ی آن به کلاینت برگردد. برای این کار ASP.NET کد HTMLای مشابه زیر را تولید می کند:

```
<form name="form1" method="post" action="Default.aspx"
id="form1">
     <input type="submit" name="btnServer" value="Server"
     style="z-index: 100; left: 30px; position: absolute;
     top: 70px" id="btnServer" />
```

بنابراین مرورگر متوجه می شود که کنترل btnServer، یک دکمـه از نـوع Submit اسـت. در کـد HTML وظیفـه ی دکمه ی Submit این است که اطلاعات موجود در فـرم را بـه سـمت سـرور وب برگردانـد. اطلاعـات مـشخص شـده در تـگ Form نیز مشخص می کند که این فرم باید با استفاده از متد post اطلاعات خود را به صـفحه ی Default.aspx در سرور بفرستد.
خوب، بهتر است که به ادامه ی کد موجود در ویژوال استودیو بپردازیم. قسمت آخر کدی که در نمای Source دیـده مـی شـود، تگ های مربوط به کنترل هایی است که در فرم قرار دادیم:

```
<div>
    <asp:Button ID="btnServer" runat="server"
      OnClick="btnServer_Click" Style="z-index: 100;
        left: 30px; position: absolute; top: 70px"
      Text="Server" />
    <asp:Label ID="lblServer" runat="server" Style="z-
      index: 101; left: 110px; position: absolute;
      top: 79px" Text="Server"></asp:Label>
    <asp:Label ID="lblClient" runat="server" Style="z-
      index: 102; left: 108px; position: absolute;
      top: 116px" Text="Client"></asp:Label>
    <input id="btnClient" style="z-index: 103; left:
      31px; position: absolute; top: 110px"
      type="button" value="Client" onclick="return
      btnClient_onclick()" />
```

```
    </div>
    </form>
</body>
</html>
```

کد وارد شده در متد مربوط به رویداد `Click` کنترل `btnServer` نیز کاملاً واضح است، زیرا شباهت زیادی به کد های برنامه های ویندوزی دارد. در این کد فقط خاصیت `Text` کنترل `lblServer` را برابر با `Changed` قرار می دهیم:

```
protected void btnServer_Click(object sender, EventArgs e)
{
     lblServer.Text = "Changed";
}
```

به این ترتیب اولین صفحه ی وب خود را با ASP.NET 2 ایجاد کردید. در این تمرین با تعدادی از کنترلهای ابتدایی آشنا شدیم و همچنین تفاوت بین کنترلهای سرور و نیز کنترلهای HTML را بررسی کردیم. در قسمت بعد، در مورد چگونگی دریافت داده ها و نیز نحوه ی تایید صحت داده های ورودی صحبت خواهیم کرد.

دریافت اطلاعات و اعتبار سنجی آنها:

یکی از کارهایی که در اغلب برنامه های تحت وب لازم است انجام شود، این است که داده هایی را از کاربر جمع آوری کرده و سپس آنها را به سرور بفرستد تا پردازش شوند. برای مثال حتماً تا کنون با صفحات مربوط به ایجاد یک اکانت کاربری جدید و یا ثبت یک در خواست در سایتها مواجه شده اید. این صفحات دارای چندین کادر متنی[1] هستند که داده های مورد نیاز را از شما دریافت کرده و پس از تایید صحت آنها[2]، آن را برای پردازش به سمت سرور می فرستند. در بخش امتحان کنید بعد، با کنترلهای ابتدایی مربوط به تایید صحت داده های ورودی و نیز نحوه ی دسترسی به اطلاعاتی که کاربر در یک فرم وب وارد کرده است آشنا خواهیم شد.

امتحان کنید: ورود اطلاعات و تایید صحت آنها

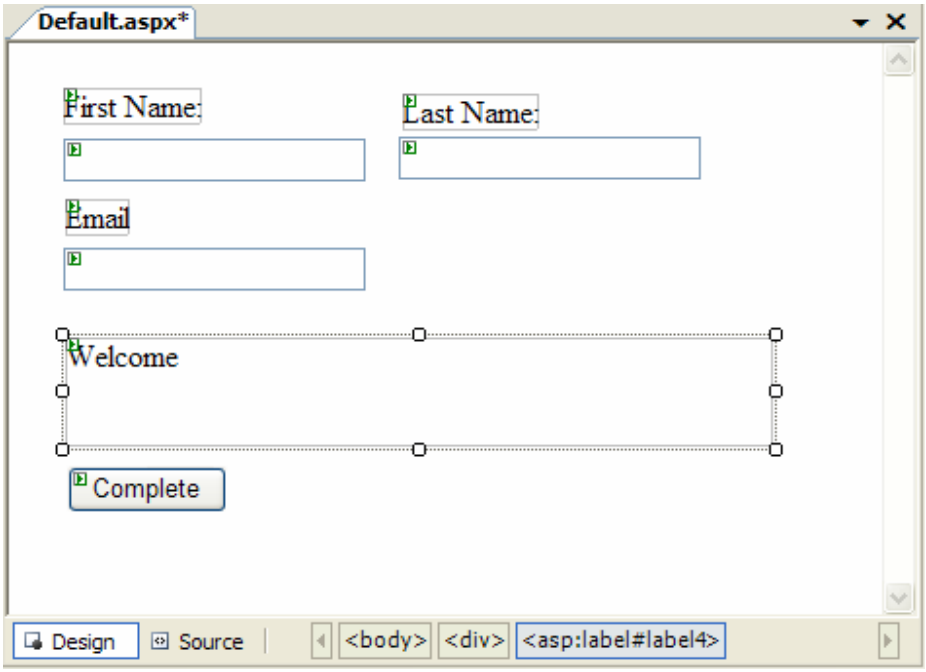
۱) با استفاده از گزینه ی `New Web Site...` ➔ `File` در نوار منوی ویژوال استودیو، یک سایت وب جدید به نام `DataEntry` ایجاد کنید.

۲) چهار کنترل `Label`، سه کنترل `TextBox` و یک کنترل `Button` را به صفحه ی `Default.aspx` اضافه کنید. برای اضافه کردن این کنترل ها از کنترلهای سرور موجود در گروه `Standard` استفاده کنید. با استفاده از منوی `Layout`، نحوه ی قرار گیری کنترل ها را به `Absolute` تغییر داده و سپس کنترل ها را همانند شکل ۱۷–۵ در فرم قرار دهید.

---

[1] مانند کنترلهای `TextBox` در برنامه های ویندوزی

[2] برای مثال اطمینان از اینکه شماره تلفن وارد شده به وسیله ی شما شامل حرف و یا کاراکتر غیر مجاز نیست و یا قبل از فشار دادن کلید ثبت، کادرهای لازم را کامل کرده باشید.



شکل ۱۷-۵

۳) حال خاصیتهای هشت کنترل موجود در فرم و نیز DOCUMENT را به صورت زیر تغییر دهید:

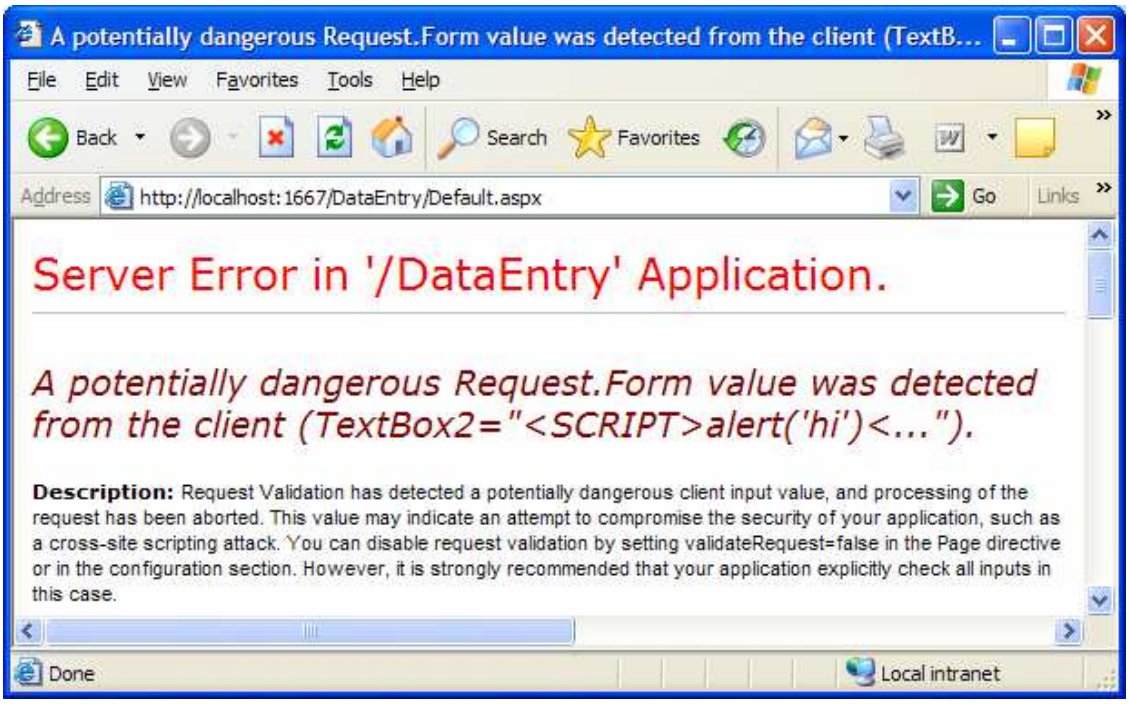
- خاصیت Title در Document را به Data Entry And Validation تغییر دهید.
- خاصیت ID کنترل Button را به btnComplete و خاصیت Text آن را به Complete تغییر دهید.
- خاصیت ID اولین کنترل TextBox را به txtFirstName تغییر دهید.
- خاصیت ID مربوط به دومین TextBox را به txtLastName تغییر دهید.
- خاصیت ID سومین TextBox را به txtEmail تغییر دهید.
- خاصیت ID اولین Label را به lblFirstName و خاصیت Text آن را به First Name تغییر دهید.
- خاصیت ID دومین Label را به lblLastName و خاصیت Text آن را به Last Name تغییر دهید.
- خاصیت ID سومین Label را به lblEmail و خاصیت Text آن را به Email تغییر دهید.
- خاصیت ID مربوط به چهارمین Label را به lblWelcome و خاصیت Text آن را به Welcome تغییر دهید.

۴) با فشار دادن کلید های Ctrl+F5 برنامه را اجرا کنید. هنگامی که صفحه نمایش داده شد، سه کادر متنی را در فرم مشاهده خواهید کرد. برای بررسی عملکرد این کنترل ها، اطلاعات خود را در این کادر ها وارد کرده و سپس روی دکمه ی Complete کلیک کنید. مشاهده می کنید که داده های شما به سمت سرور فرستاده شده و سپس فرم به همان

صورت قبلی نمایش داده می شود (بدون اینکه داده های وارد شده در فرم تغییری کنند). این مرتبه کد اسکریپت مشخص شده در زیر را در کادر First Name وارد کرده و سپس روی دکمه ی Complete کلیک کنید:

```
<SCRIPT>alert('hi')</SCRIPT>
```

مشاهده خواهید کرد که صفحه ی خطایی همانند شکل ۱۷-۶ به وسیله ی ویژوال استودیو نمایش داده می شود. ASP.NET 2 دارای خاصیتی است که Request Validation نامیده می شود. به وسیله ی این خاصیت داده هایی که به وسیله ی کاربر در فرم برنامه وارد شده است قبل از پردازش شدن به وسیله ی سرور، بررسی می شوند که شامل کد های خطرناکی نباشند. در آخر نیز می توانید ترتیب حرکت بین کنترل ها را با استفاده از کلید Tab بررسی کنید. همانند برنامه های ویندوزی، برای تغییر این مورد می توانید از خاصیت TabIndex کنترل ها استفاده کرده و یا آنها را با ترتیب مورد نظر در فرم قرار دهید.



شکل ۱۷-۶

۵) حال می خواهیم از داده هایی که کاربر وارد کرده است در برنامه استفاده کنیم. ابتدا باید به قسمت کد سرور مربوط به این فایل برویم. راحت ترین روش برای انجام این کار فشار دادن کلید F7 است. اما در این قسمت می خواهیم که در متد مربوط به رویداد Form_Load کدی را وارد کنیم. پس در نمای Design روی فرم برنامه دو بار کلیک کنید تا متد مربوط به این رویداد به صورت اتوماتیک ایجاد شود. سپس کد مشخص شده در زیر را به این متد اضافه کنید:

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Page.IsPostBack)
    {
```

```
            // If this is a postback and not the initial page
            // load Display the data to the user
            this.lblWelcome.Text = "Hello " +
                 this.txtFirstName.Text + " " +
                 this.txtLastName.Text + "<BR>" +
                 "Your email address is " +
                 this.txtEmail.Text;
        }
}
```

۶) حال باید از صحت داده های ورودی به وسیله ی کاربر مطمئن شویم. در ویژوال استودیو کنترل هایی برای این کار وجود دارد. برای مشاهده ی این کنترل ها با فشار دادن مجدد کلید F7 به نمای Design برگردید و سپس در جعبه ابزار روی گروه Validation کلیک کنید. کنترل هایی که در این قسمت قرار دارند برای کنترل صحت داده های وارد شده به وسیله ی کاربر در نظر گرفته شده اند. با استفاده از این قسمت از جعبه ابزار، دو کنترل RequiredFieldValidator و یک کنترل ValidationSummary را به فرم اضافه کرده و سپس با استفاده از منوی Layout نحوه ی قرار گیری آنها را در فرم به Absolute تغییر دهید.

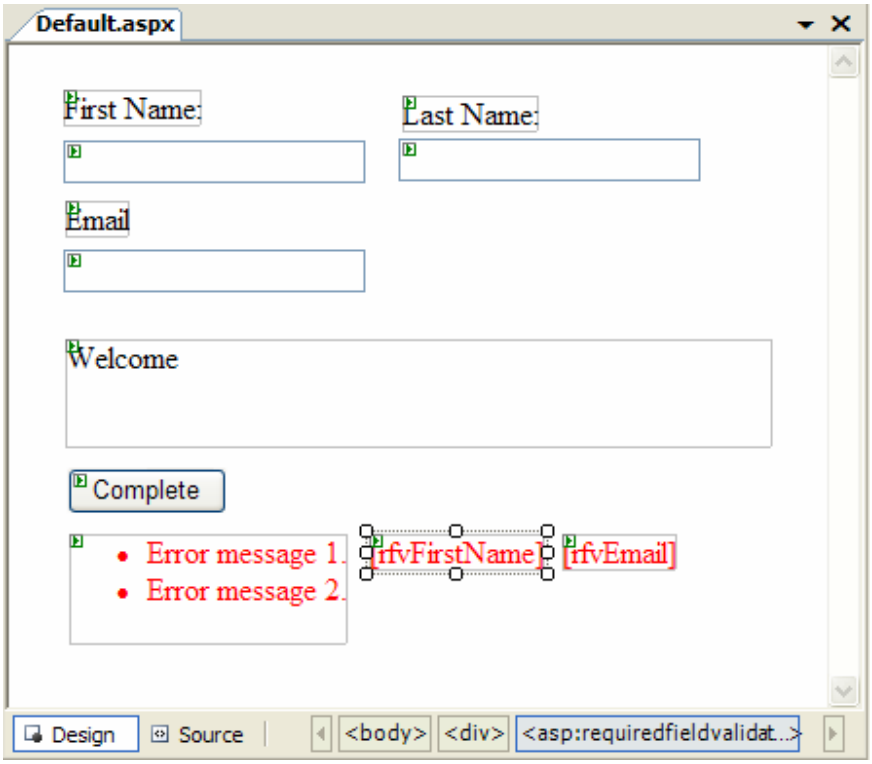
خاصیت کنترل RequiredFieldValidator اول را به صورت زیر تغییر دهید.

- خاصیت ID را برابر با rfvFirstName قرار دهید.
- خاصیت Display را به None تغییر دهید.
- خاصیت ControlToValidate را به txtFirstName تغییر دهید.
- خاصیت ErrorMessage را به First name is required تغییر دهید.

خاصیت دومین کنترل RequiredFieldValidator را نیز به صورت زیر تغییر دهید:

- خاصیت ID را به rfvEmail تغییر دهید.
- خاصیت Display را به None تغییر دهید.
- خاصیت ControlToValidate را به txtLastName تغییر دهید.
- خاصیت ErrorMessage را به Email is required تغییر دهید.

خاصیت ID کنترل ValidationSummary را نیز به ValidationSummary تغییر دهید.
فرم تکمیل شده ی برنامه باید مشابه شکل ۱۷-۷ باشد.



شکل ۱۷-۷

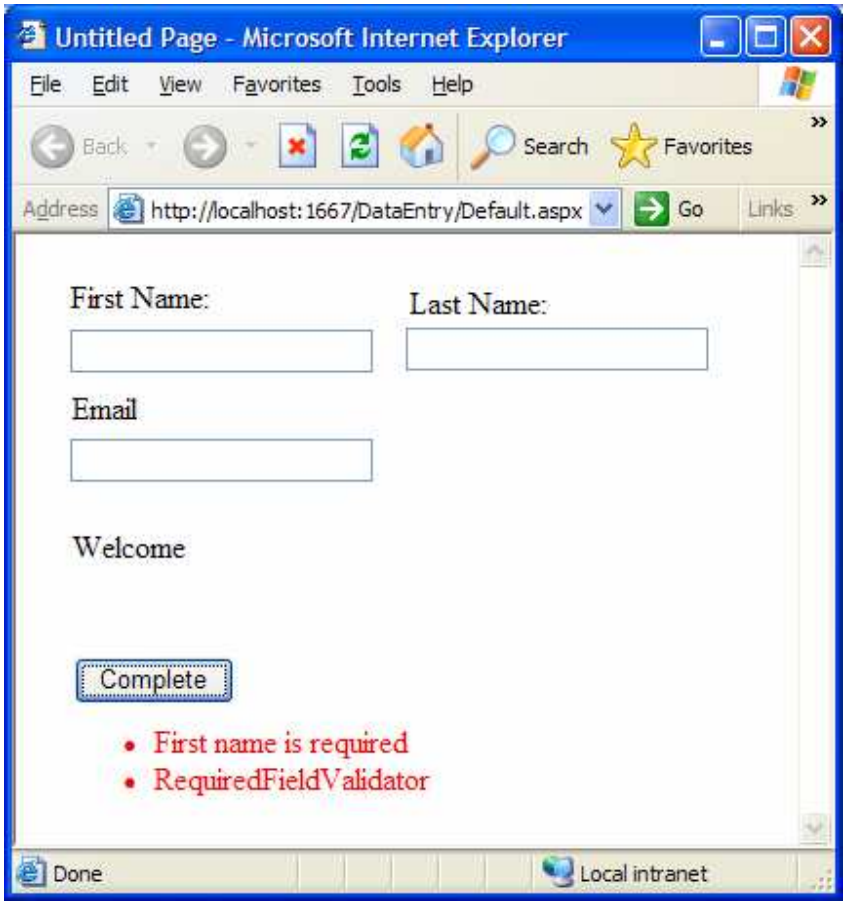
۷) برنامه را اجرا کرده و بدون اینکه در کادرهای `FirstName` و `Email` عبارتی را وارد کنید، روی دکمه ی `Complete` کلیک کنید. پیغام خطایی مشابه شکل ۱۷-۸ نمایش داده خواهد شد.

نکته: این مثال نشان می دهد که تایید صحت داده های ورودی در `ASP.NET 2` چقدر ساده و سریع می تواند انجام شود. همانطور که مشاهده کردید در گروه `Validation` در جعبه ابزار کنترلهای دیگری نیز برای این کار وجود دارند. برای مثال کنترل `CompareValidator` برای بررسی این مورد است که عبارت وارد شده در یک کادر حتماً با عبارت خاصی برابر باشد. این عبارت خاص می تواند شامل یک مقدار ثابت، مقدار وارد شده در یک کنترل دیگر و یا حتی مقداری از یک بانک اطلاعاتی باشد. کنترل `RangeValidator` برای بررسی این مورد به کار می رود که داده ی وارد شده حتماً در بازه ی خاصی باشد. برای مثال حتماً سنی که کاربر وارد کرده است بین ۱۸ تا ۳۵ باشد.

چگونه کار می کند؟

مشاهده کردید بدون اینکه کدی را وارد کنید، می توانید از کاربر بخواهید که قبل از ارسال فرم به سرور کادرهای لازم را تکمیل کند. برای اطمینان از این که کاربر کادری را تکمیل کرده است یا نه می توانید از کنترل `RequiredFieldValidator` استفاده کنید. تنظیم این کنترل هم بسیار ساده است. کافی است که خاصیت `ErrorMessage` آن را برابر با پیغام خطایی قرار دهیم که می خواهیم نمایش داده شود. این پیغام خطا در دو مکان مختلف می تواند نمایش داده شود: یکی در همان مکانی که کنترل قرار گرفته است و دیگری در کنترل `ValidationSummary`. در این قسمت با تنظیم خاصیت `Display` با `None` از نمایش داده شدن پیغام خطا به وسیله ی `RequiredFieldValidator` جلوگیری کرده و فقط آن را

در ValidationSummary نمایش داده ایم. خاصیت ControlToValidate نیز باید برابر با نام کنترلی قرار گیرد که می خواهیم حتماً کاربر آن را کامل کند.



شکل ۱۷-۸

```
<asp:RequiredFieldValidator ID="rfvFirstName"
     runat="server" ControlToValidate="txtFirstName"
     Display="None" ErrorMessage="First name is required"
     Style="z-index: 108; left: 185px;position: absolute;
     top: 251px">
</asp:RequiredFieldValidator>
```

کنترل ValidationSummary به عنوان یک محل مرکزی برای نمایش تمام خطاهایی که در فرم به وجود آمده است به کار می رود. اگر نخواهید از این کنترل در فرم خود استفاده کنید، می توانید خاصیت Display هر یک از کنترلهای اعتبار سنجی داده ها را برابر با Static. قرار دهید تا پیغام خطا در محدوده ی همان کنترل نمایش داده شود. همانطور که مشاهده کردید برای نمایش پیغام های خطا در کنترل ValidationSummary به هیچ تنظیمات اضافی در فرم نیاز نیست. فقط کافی است این کنترل را در محلی از فرم قرار دهید که می خواهید پیغام های خطا در آن قسمت نمایش داده شوند.

```
<asp:ValidationSummary ID="ValidationSummary"
```

```
Style="z-index: 111; left: 31px; position: absolute;
top: 251px" runat="server" />
```

تنها کدی که در این قسمت وارد کردیم، کد مربوط به متد Form_Load بود. در این کد ابتدا با استفاده از خاصیت PostBack بررسی می کنیم که آیا فرم برای اولین بار است که در این مرورگر نمایش داده می شود یا نه؟ متد Load در فرم های وب با فرمهای ویندوزی از چند جهت تفاوت دارد. یکی از این موارد این است که این متد در فرمهای وب معمولا چندین بار فراخوانی می شود، در صورتی که در فرمهای ویندوزی این متد فقط یک بار فراخوانی می شد و آن نیز زمانی بود که فرم در حال Load شدن در حافظه بود.

در فرمهای وب هنگامی که کاربر تقاضای مشاهده ی فرمی را می کند، متد Load فرم برای اولین بار فراخوانی شده و فرم در مرورگر کاربر نمایش داده می شود. سپس هنگامی که کاربر به هر روشی (برای مثال با کلیک کردن روی یکی از دکمه های فرم)، فرم را به سمت سرور فرستاد، متد Load مجدداً فراخوانی می شود. اما این بار خاصیت IsPostBack برابر با True شده است. بنابراین اگر بخواهیم عملی فقط در اولین بار رخ دادن رویداد Load اجرا شود، می توانیم آن را در شرط IsPostBack == False قرار دهیم. اما در اینجا می خواهیم هنگامی متن داخل کنترل lblWelcome تغییر کند که کاربر روی دکمه ی Complete کلیک کرده باشد. به عبارت دیگر می خواهیم زمانی این متن تغییر کند که متد Load به علت کلیک کردن کاربر روی یکی از دکمه های فرم فراخوانی شود. پس آن را در شرط IsPostBack == True قرار می دهیم:

```
if (Page.IsPostBack)
{
     // If this is a postback and not the initial page
     // load Display the data to the user
     this.lblWelcome.Text = "Hello " +
          this.txtFirstName.Text + " " +
          this.txtLastName.Text + "<BR>" +
          "Your email address is " +
          this.txtEmail.Text;
}
```

طراحی ظاهر سایت:

در گذشته یکی از عمده ترین مشکلات طراحی سایت، ایجاد ظاهری ثابت و پایدار برای آن در تمام صفحات تشکیل دهنده ی سایت بود، به گونه ای که بتوان به سادگی آن را کنترل کرد و یا تغییر داد. طراحان سایت با ایجاد کنترلهای سفارشی و قرار دادن آنها در تمامی صفحات و یا استفاده از چندین روش دیگر سعی می کردند به چنین هدفی دست پیدا کنند. استفاده از این روشها در اغلب موارد جوابگو بود، اما مشکل عمده ای که ایجاد می کرد این بود که طراح سایت نمی توانست مطمئن شود که تمام کنترل های سفارشی که ایجاد شده اند در مکان خود قرار دارند. برای انجام این کار زمان زیادی باید صرف می شد و با کوچکترین تغییری که در محتوی سایت به وجود می آمد، یک نفر باید مسئول می شد تا تمام قسمتهای سایت را بررسی کرده و از درست بودن طاهر آنها مطمئن شود.

در ویژوال استودیو ۲۰۰۵ با استفاده از ابزارهایی مانند Themeها، فرمهای اصلی و یا بسیار ی از ابزارهای دیگر می توان ظاهری ثابت . پایدار را برای تمام قسمتهای سایت ایجاد کرد. در بخش امتحان کنید بعد، با تعدادی از این ابزارها آشنا خواهیم شد.

امتحان کنید: ایجاد اولین وب سایت

۱) با استفاده از ویژوال استودیو یک وب سایت جدید به نام SiteLookAndFeel ایجاد کنید.

۲) برای شروع برنامه، باید چندین فایل و یا فولدر را به برنامه اضافه کنید. ابتدا روی نام پروژه در پنجره ی Solution Explorer کلیک راست کرده و از منوی باز شده گزینه ی Add New Item را انتخاب کنید. از کادری که نمایش داده می شود گزینه ی Master Page را انتخاب کرده و روی گزینه ی Add کلیک کنید.

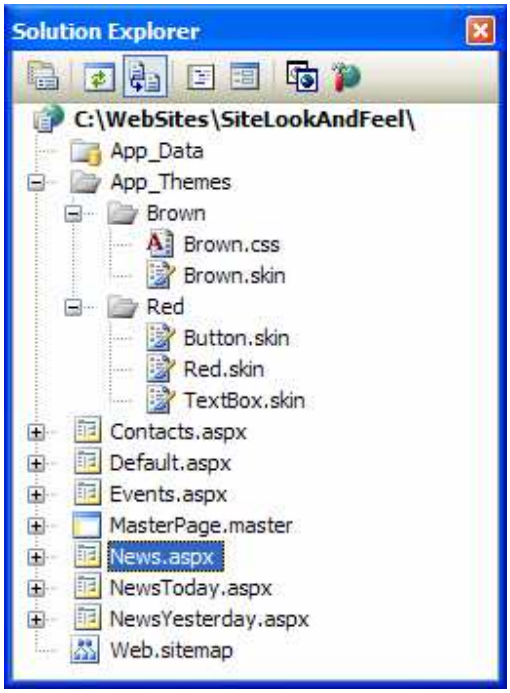
۳) راهنمای Page در فرم Default.aspx را به صورت زیر تغییر دهید تا از MasterPage ایجاد شده استفاده کند:

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master"
    AutoEventWireup="true"  CodeFile="Default.aspx.cs"
    Inherits="_Default" %>
```

۴) فایلها و فولدرهای مشخص شده در زیر را به برنامه اضافه کنید.

- یک فولدر Theme به فولدر اصلی سایت اضافه کرده و نام آن را برابر با Red قرار دهید. برای این کار در پنجره ی Solution Explorer روی نام برنامه کلیک راست کرده و از منوی باز شده، گزینه ی Add ASP.NET Folder ➔ Theme را انتخاب کنید. به این ترتیب یک فولدر اصلی به نام App_Theme به سایت اضافه شده و یک زیر فولدر نیز به در آن ایجاد می شود. نام فولدری که درون فولدر App_Theme ایجاد شده است را به Red تغییر دهید. سپس فولدر دیگری به نام Brown به App_Theme اضافه کنید و بعد از این کار یک فایل به نام brown.skin در فولدر Brown ایجاد کنید. روی فولدر Brown کلیک راست کرده و گزینه ی Add New Item... را انتخاب کنید. سپس با استفاده از کادر باز شده یک فایل از نوع Style Sheet به نام Brown.css را به فولدر Brown اضافه کنید. در فولدر Red نیز سه فایل به نامهای Red.skin، Button.skin و TextBox.skin اضافه کنید (برای اضافه کردن این فایلها می توانید روی فولدر کلیک راست کرده و از منوی باز شده گزینه ی Add New Item... را انتخاب کنید. سپس با استفاده از کادر باز شده یک فایل از نوع Skin File با نامهای مشخص شده را به فولدر اضافه کنید).
- با کلیک کردن روی نام برنامه در پنجره ی Solution Explorer و انتخاب گزینه ی Add New Item... پنج فرم وب جدید به نامهای News.aspx، NewsYesterday.aspx، NewsToday.aspx، Events.aspx و Contacts.aspx را به سایت خود اضافه کنید. هنگام اضافه کردن این فرم ها، در پنجره ی Add New Item گزینه ی Select master page را انتخاب کنید. به این ترتیب قبل از ایجاد فرم کادر Select a Master Page نمایش داده می شود و به شما این اجازه را می دهد تا یک Master Page برای فرم خود انتخاب کنید. در این کادر نیز Master Page ای که در ابتدای این قسمت ایجاد کرده بودیم را انتخاب کرده و روی دکمه ی OK کلیک کنید تا فرم وب مورد نظر ایجاد شود.

- در آخر نیز باید یک فایل از نوع Site Map به برنامه اضافه کنیم. برای این کار روی نام پروژه در پنجره ی Solution Explorer کلیک راست کرده و سپس گزینه ی ...Add New Item را انتخاب کنید. از کادر Add New Item گزینه ی Site Map را انتخاب کرده و روی دکمه ی OK کلیک کنید. بعد از اتمام این کار پنجره ی Solution Explorer باید مشابه شکل ۱۷-۹ باشد.



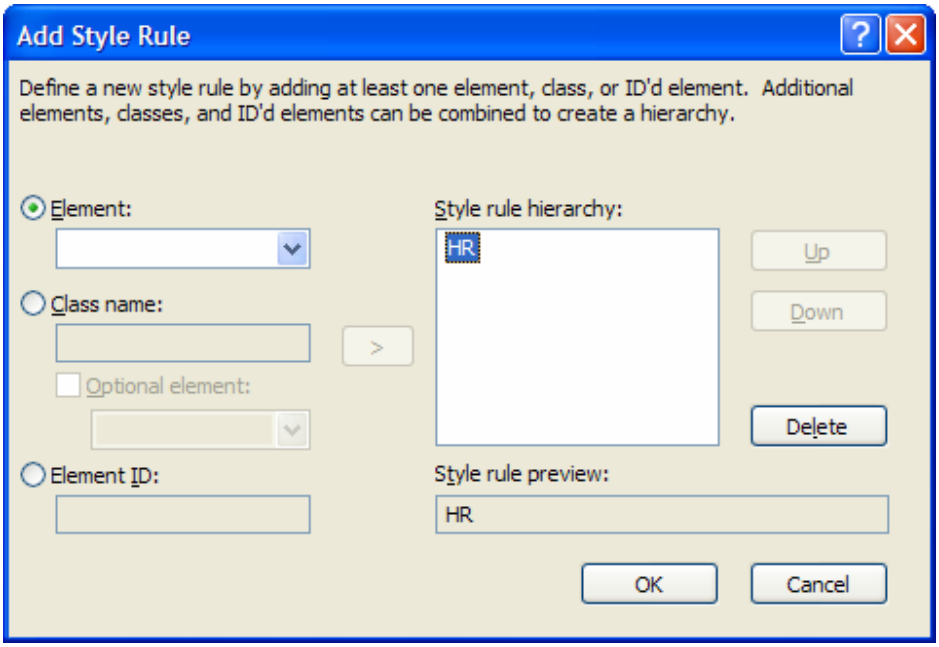
شکل ۱۷-۹

۵) فایل Web.sitemap را باز کرده و کد درون آن را به صورت مشخص شده در زیر تغییر دهید:

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns=
  "http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
  <siteMapNode url="Default.aspx" title="Home"
        description="Back to the main page" roles="" >
    <siteMapNode url="News.aspx" title="News"
         description="Your front page news." roles="">
      <siteMapNode url="NewsToday.aspx"
          title="Today's News"
          description="Today's top stories" roles="" />
      <siteMapNode url="NewsYesterday.aspx"
          title="Yesterday's News"
          description="Yesterday's top stories" roles="" />
    </siteMapNode>
    <siteMapNode url="Events.aspx" title="Upcoming Events"
          description="Today's top stories" roles="" />
```

```
    <siteMapNode url="Contact.aspx" title="Contact Us"
          description="Today's top stories" roles="" />
  </siteMapNode>
</siteMap>
```

۶) روی فایل Brown.css در پنجره ی Solution Explorer دو بار کلیک کنید تا باز شود. به صورت پیش فرض یک قسمت مشخصات خالی برای عنصر BODY در این فایل وجود دارد. در این فایل می توانید با استفاده از زبان CSS استیل و قالب قسمتهای مختلف یک صفحه ی وب را تعیین کنید. سپس با قرار دادن لینکی از این فایل در ابتدای صفحات وب مورد نظر، ظاهر آن صفحات به صورت مشخص شده در فایل در خواهند آمد. می توانید بعد از اینکه با قواعد کد نویسی به زبان CSS آشنا شدید به صورت دستی کد مربوط به ظاهر مورد نظر خود را در این فایل وارد کنید. اما در این قسمت از ابزارهای موجود برای تولید این کد استفاده خواهیم کرد. در قسمتی از فایل کلیک راست کرده و گزینه ی Add Style Rule را انتخاب کنید. به این ترتیب کادر Add Style Rule همانند شکل ۱۷-۱۰ نمایش داده می شود. از قسمت Element گزینه ی HR را انتخاب کرده و سپس با کلیک کردن روی دکمه ی > آن را به قسمت Style Rule Hierarchy اضافه کنید. حال اگر روی دکمه ی OK کلیک کنید، یک قسمت مشخصات خالی برای عنصر HR به فایل اضافه شده است.



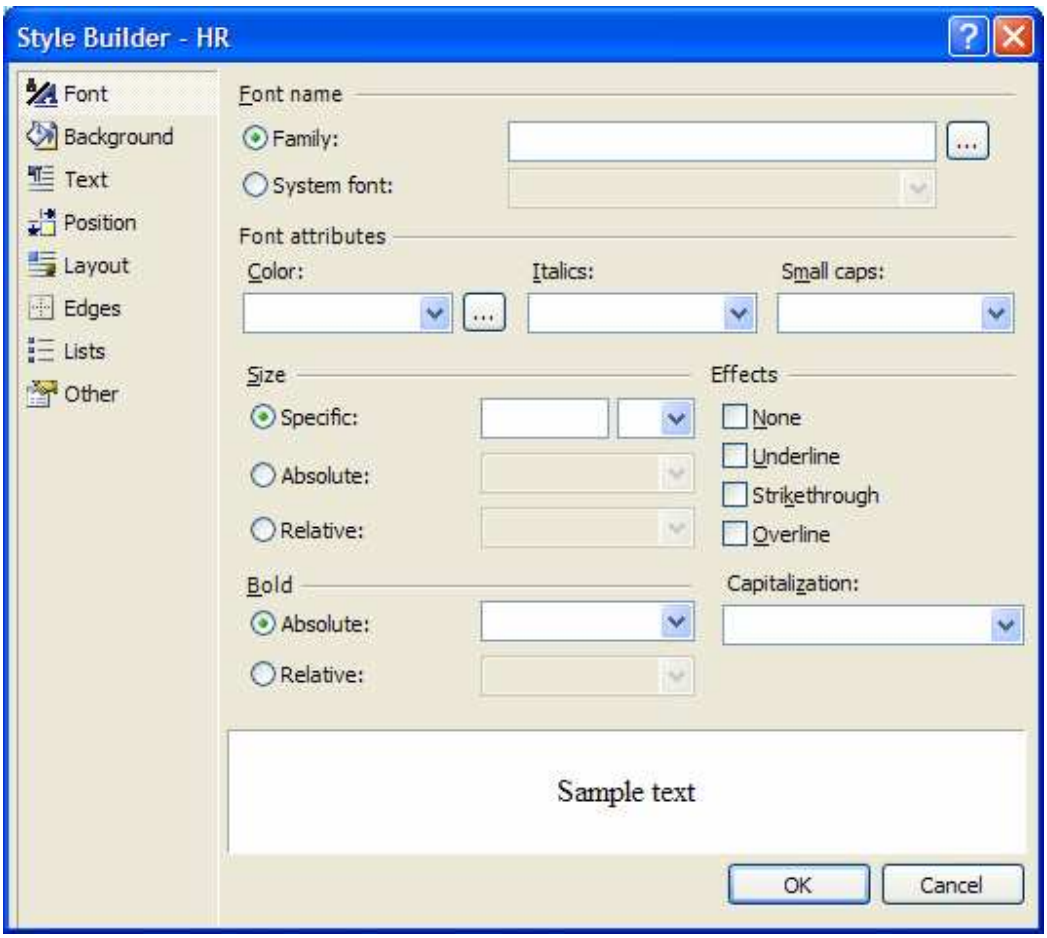
شکل ۱۷-۱۰

با اضافه کردن این قسمت، در حقیقت مشخص کرده اید که می خواهید ظاهر تمام تگهای HR[1] در فایل وبی که از این فایل CSS استفاده می کنند به صورتی که در این قسمت مشخص شده است باشد. خوب، پس به این ترتیب باید در این قسمت ظاهر تگهای HR را مشخص کنیم. برای این کار هم می توانیم از قسمت طراحی که برای این قسمت وجود دارد استفاده کنیم هم می توانیم کد آن را وارد کنیم. برای استفاده از ابزار طراحی، مکان نما را به بین دو علامت { بعد از HR برده و در آنجا کلیک راست کنید. سپس از منویی که باز می شود گزینه ی Build Style را انتخاب کنید تا کادر Build Style – HR همانند شکل ۱۷-۱۱ نمایش داده شود. برای وارد کردن کد نیز می توانید ظاهر مورد نظر خود را در بین دو علامت آکولاد وارد کنید.

[1] این تگ در زبان HTML برای ایجاد خطوط افقی به کار می رود.

همچنین برای این کار می توانید از سیستم تکمیل هوشمند ویژوال استودیو نیز استفاده کنید. در اینجا کد مشخص شده در زیر را بـه بلاک HR اضافه کنید:

```
HR
{
    color:#CC0000;
    height:12px;
}
```



شکل ۱۷-۱۱

۷) حال باید ظاهر MasterPage را تعیین کنیم. به این ترتیب قالب تمام صفحاتی که از ایـن MasterPage اسـتفاده مـی کننـد بـه ایـن شـکل در خواهـد آمـد. در پنجـره ی Solution Explorer روی فایـل MasterPage.master دو بار کلیک کنید تا باز شود. سپس به نمای Source بروید و کد HTML آن را به صورت مشخص شده در زیر تغییر دهید:

```
<%@ Master Language="C#" AutoEventWireup="true"
CodeFile="MasterPage.master.cs" Inherits="MasterPage" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head runat="server">
    <title>Untitled Page</title>
      <Style type="text/css">
        .TableLayout
        {
          width: 700px; background-color:#ffcc66;
        }
        .border
        {
          border-style:solid; border-color:black;
          border-width:thin;
        }
      </Style>
  </head>
  <body bgcolor="#cc0000">
    <form id="form1" runat="server">
      <div>
        <table id="tblMasterLayoutHeader"
          class="TableLayout" cellpadding="0"
          cellspacing="0" align="center" height="450">
          <tr>
            <td style="width: 100px" rowspan=2
              class="border">
              <!-- Add the menu to the page -->
              <asp:Menu ID="Menu1" Runat="server" >
              <Items>
                <asp:MenuItem Value="Home" Text="Home"
                  NavigateUrl="Default.aspx">
                </asp:MenuItem>
                <asp:MenuItem Value="News" Text="News"
                  NavigateUrl="News.aspx">
                  <asp:MenuItem Value="Today" Text="Today"
                    NavigateUrl="NewsToday.aspx">
                  </asp:MenuItem>
                  <asp:MenuItem Value="Yesterday"
                    Text ="Yesterday"
                    NavigateUrl="NewsYesterday.aspx">
                  </asp:MenuItem>
                </asp:MenuItem>
                <asp:MenuItem Value="Events" Text="Events"
                  NavigateUrl="Events.aspx">
                </asp:MenuItem>
                <asp:MenuItem Value="Contact Us"
                  Text="Contact Us"
```

```
                    NavigateUrl="Contact.aspx">
                  </asp:MenuItem>
                </Items>
                </asp:Menu>
              </td>
              <td bgcolor="#000000" class="border" >
              <!-- Main title -->
                <asp:Label ID="Label1" Runat="server"
                  Text="Beginning Visual C# 2005"
                  Font-Names="Arial" Font-Bold="true"
                  ForeColor="#ffcc33" Font-Size="28pt" />
              </td>
            </tr>
            <tr>
              <td class="border">
                <!-- Site map path under Title -->
                <asp:SiteMapPath ID="smpMain"
                  Runat="server"></asp:SiteMapPath>
              </td>
            </tr>
            <tr>
              <td class="border" colspan="2" height="100%"
                valign="top" align="center">
                <!-- All site content will go here -->
                <asp:contentplaceholder id="cphPageContent"
                  runat="server"></asp:contentplaceholder>
                <br />
              </td>
            </tr>
            <tr>
              <td class="border" align="center" colspan="2">
                <!-- Footer -->
                <asp:Label ID="Label2" Runat="server"
                  Text="(c)2004, All rights reserved."
                  Font-Names="Arial" Font-Bold="true"
                  ForeColor="black" Font-Size="10pt" >
                </asp:Label>
              </td>
            </tr>
          </table>
        </div>
      </form>
    </body>
</html>
```

۸) فایل `Default.aspx` را باز کرده و به نمای `Source` در آن بروید. سپس کد `HTML` موجود در آن را با کد زیر جایگزین کنید:

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master"
    AutoEventWireup="true"  CodeFile="Default.aspx.cs"
    Inherits="_Default" Theme="Red" %>
<asp:Content ID="Content1"
ContentPlaceHolderID="cphPageContent" Runat="Server">
  <asp:TextBox ID="txtTest" Runat="server">
       Just some text
  </asp:TextBox>
  <hr />
  <br />
  <asp:Button ID="btnTest" Runat="server" Text="Button" />
</asp:Content>
```

۹) حال کد HTML موجود در فایل News.aspx را به صورت زیر تغییر دهید:

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master"
    AutoEventWireup="true" CodeFile="News.aspx.cs"
    Inherits="News" Title="Untitled Page" Theme="Brown"%>
<asp:Content ID="Content1"
  ContentPlaceHolderID="cphPageContent" Runat="Server">
  <asp:TextBox ID="txtTest" Runat="server">
       Just some text
  </asp:TextBox>
  <asp:Button ID="btnTest" Runat="server" Text="Button" />
</asp:Content>
```

۱۰) با استفاده از Solution Explorer فایل Button.skin را باز کرده و کد زیر را به آن اضافه کنید:

```
<asp:Button runat="server" ForeColor="Red"
            Font- Name="Arial" Font-Size="28px"
            Font-Weight="Bold" />
```

۱۱) فایل TextBox.skin در فولدر Red را باز کرده و کد مشخص شده در زیر را به آن اضافه کنید:

```
<asp:TextBox runat="server" ForeColor="Red"
             Font-Name="Arial" Font-Size="28px"
             Font-Weight="Bold" />
```

۱۲) کد مشخص شده در زیر را به فایل Brown.skin در فولدر Brown اضافه کنید:

```
<asp:Button runat="server" ForeColor="Brown"
            Font-Name="Arial" Font-Size="28px"
            Font-Weight="Bold" />
```

```
<asp:TextBox runat="server" ForeColor="Brown"
             Font-Name="Arial" Font-Size="28px"
             Font-Weight="Bold" />
```

۱۳) در فرمهای دیگر، کافی است که عبارت ContentPlaceHolderID را با عبارت cphPageCpntent تعویض کنید. دلیل این کار نیز در این است که بعد از اضافه کردن این فرم ها به برنامه، فایل MasterPage را تغییر دادیم. بنابراین خط دوم فایلهای Events.aspx، Contacts.aspx، NewsToday.aspx و NewsYesterday.aspx به صورت زیر خواهد بود:

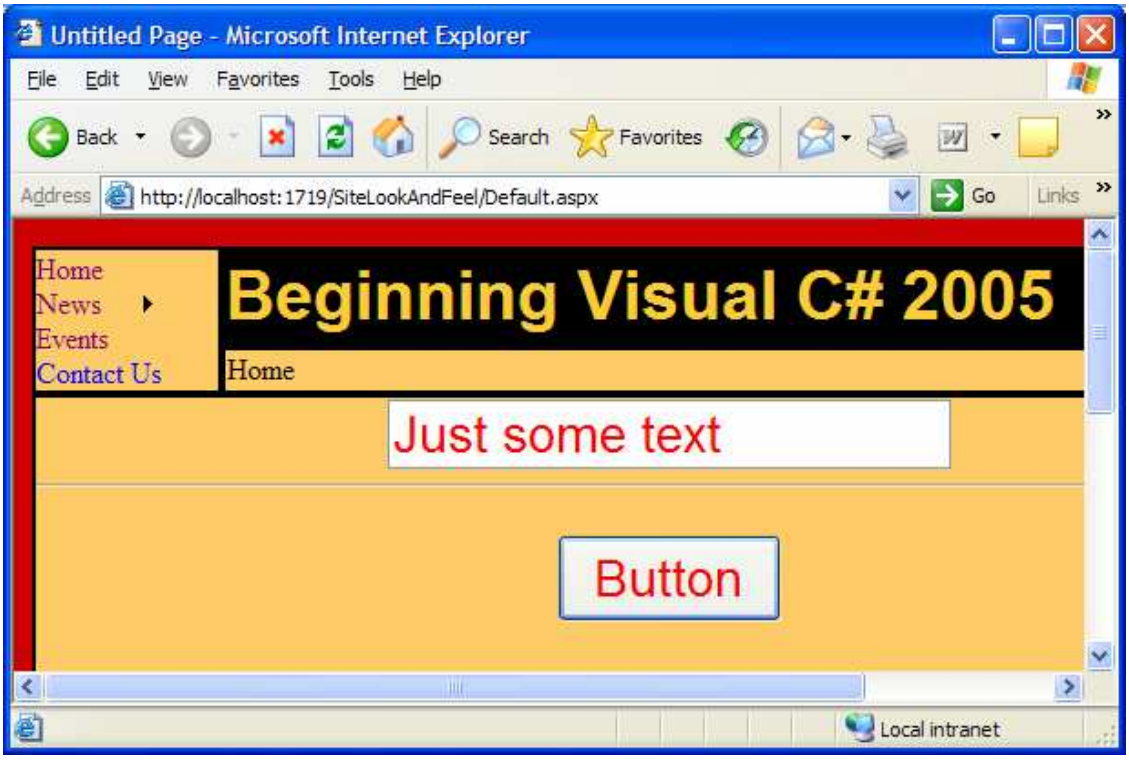
```
<asp:Content ID="Content1" Runat="Server"
             ContentPlaceHolderID="cphPageContent">
```

۱۴) برنامه را اجرا کنید و لینک های مختلفی که در سایت ایجاد شده اند را امتحان کنید. به کنترل هایی که برای جا به جایی در بین فرمهای برنامه ایجاد شده اند توجه کنید. سایت شما همانند شکل ۱۷-۱۲ خواهد بود.

چگونه کار می کند؟

در بخش امتحان کنید این قسمت، از بعضی از جدیدترین کنترل هایی که به ASP.NET 2 اضافه شده بودند استفاده کردیم. ترکیب این کنترل ها این امکان را به ما داد تا بتوانیم به سادگی سایتهایی با ظاهر قدرتمند و زیبا طراحی کنیم. همانطور که در طول این مثال مشاهده کردید با استفاده از MasterPage ها می توانیم که ظاهری پایدار را برای تمام صفحات سایت ایجاد کنیم. به این ترتیب قسمتهایی را که باید در چندین صفحه از سایت وجود داشته باشند را در یک MasterPage قرار داده و سپس از آن در صفحات مورد نظر استفاده می کنیم. برای مثال می توانیم بخشهای مربوط به جا ب جا شدن بین صفحات مختلف سایت و یا نقشه ی سایت را در این قسمتها قرار دهیم. به این ترتیب در صورت لزوم اگر بخواهیم که یکی از این قسمتها را تغییر دهیم، کافی است که این تغییر را در MasterPage اعمال کنیم تا تمام صفحات به صورت اتوماتیک آن را دریافت کنند.

کدی که در MasterPage وارد کردیم با وجود اینکه بسیار طولانی بود، اما هیچ قسمت پیچیده و مبهمی نداشت و تمام آن را با استفاده از جعبه ابزار نیز می توانستیم ایجاد کنیم. می توانید به MasterPage بروید و نمای آن را به Design تغییر دهید تا کنترل هایی را که به وسیله ی کد HTML در فرم قرار داده ایم مشاهده کنید. در فایل MasterPage یک کنترل به نام ContentPalceHolder قرار داده ایم. این کنترل قسمتی را مشخص می کند که اطلاعات صفحه های دیگر می توانند در آن قرار بگیرند. بنابراین اگر از این MasterPage در صفحات سایت خود استفاده کنید، فقط می توانید قسمت مشخص شده به وسیله ی ContentPlaceHolder را تغییر دهید.



شکل ۱۷-۱۲

همچنین در MasterPage از یک جدول استفاده کرده ایم تا عناصری که می خواهیم در تمام فرمهای برنامه قرار بگیرند را به وسیله ی آن مرتب کنیم. عناصری هم که در آن قرار داده ایم، شامل چند کنترل ساده و نیز چند کنترل دیگر است که احتمالا تا کنون از آنها استفاده نکرده اید، مانند Menu و یا SiteMapPath.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head runat="server">
    <title>Untitled Page</title>
      <Style type="text/css">
        .TableLayout
        {width: 700px; background-color:#ffcc66;}
        .border
        {border-style:solid; border-color:black;
          border-width:thin;}
      </Style>
  </head>
  <body bgcolor="#cc0000">
    <form id="form1" runat="server">
      <div>
        <table id="tblMasterLayoutHeader"
          class="TableLayout" cellpadding="0"
          cellspacing="0" align="center" height="450">
```

```
<tr>
  <td style="width: 100px" rowspan=2
    class="border">
    <!-- Add the menu to the page -->
    <asp:Menu ID="Menu1" Runat="server" >
    <Items>
      <asp:MenuItem Value="Home" Text="Home"
        NavigateUrl="Default.aspx">
      </asp:MenuItem>
      <asp:MenuItem Value="News" Text="News"
        NavigateUrl="News.aspx">
        <asp:MenuItem Value="Today" Text="Today"
          NavigateUrl="NewsToday.aspx">
        </asp:MenuItem>
        <asp:MenuItem Value="Yesterday"
          Text ="Yesterday"
          NavigateUrl="NewsYesterday.aspx">
        </asp:MenuItem>
      </asp:MenuItem>
      <asp:MenuItem Value="Events" Text="Events"
        NavigateUrl="Events.aspx">
      </asp:MenuItem>
      <asp:MenuItem Value="Contact Us"
        Text="Contact Us"
        NavigateUrl="Contact.aspx">
      </asp:MenuItem>
    </Items>
    </asp:Menu>
  </td>
  <td bgcolor="#000000" class="border" >
  <!-- Main title -->
    <asp:Label ID="Label1" Runat="server"
      Text="Beginning Visual C# 2005"
      Font-Names="Arial" Font-Bold="true"
      ForeColor="#ffcc33" Font-Size="28pt" />
  </td>
</tr>
<tr>
  <td class="border">
    <!-- Site map path under Title -->
    <asp:SiteMapPath ID="smpMain"
      Runat="server"></asp:SiteMapPath>
  </td>
</tr>
<tr>
  <td class="border" colspan="2" height="100%"
    valign="top" align="center">
```

```
            <!-- All site content will go here -->
            <asp:contentplaceholder id="cphPageContent"
              runat="server"></asp:contentplaceholder>
            <br />
          </td>
        </tr>
        <tr>
          <td class="border" align="center" colspan="2">
            <!-- Footer -->
            <asp:Label ID="Label2" Runat="server"
              Text="(c)2004, All rights reserved."
              Font-Names="Arial" Font-Bold="true"
              ForeColor="black" Font-Size="10pt" >
            </asp:Label>
          </td>
        </tr>
      </table>
    </div>
  </form>
 </body>
</html>
```

اگر چه در این مثال به صورت بسیار ساده از کنترل Menu استفاده کرده ایم، اما این کنترل از انعطاف پذیری بالایی برخوردار است. برای مثال می توانید به جای اینکه گزینه های منو را به صورت ثابت و در کد ایجاد کنید، آن را به یک بانک اطلاعاتی متصل کنید. همچنین می توانید جهت قرار گرفتن منو را نیز عوض کنید. در این سایت از منو به صورت عمودی استفاده کردیم اما می توانید با تنظیم خاصیت Orientation آن را به صورت افقی نمایش دهید. ویژگی دیگری از منو که می توانید آن را تنظیم کنید، ظاهر آن است. با استفاده از خاصیت هایی مانند خاصیت Style می توانید ظاهر منو را به گونه ای تغییر دهید تا با دیگر قسمتهای فرم مشابه شود.

در این برنامه فایل Red.Skin را بدون استفاده گذاشتیم، اما در برنامه های بعد از این فایل نیز استفاده خواهیم کرد.

فایل Button.skin استیل کنترلهای Button را مشخص می کند و زمانی به کار گرفته می شود که Theme صفحه ی مورد نظر برابر با Red تنظیم شود.

```
<asp:Button runat="server" ForeColor="Red"
            Font- Name="Arial" Font-Size="28px"
            Font-Weight="Bold” />
```

فایل TextBox.skin نیز برای همین منظور مورد استفاده قرار می گیرد، اما با این تفاوت که این فایل ظاهر کنترلهای TextBox را مشخص می کند:

```
<asp:TextBox runat="server" ForeColor="Red"
             Font-Name="Arial" Font-Size="28px"
             Font-Weight="Bold" />
```

در فایل Default.aspx با استفاده از خصیصه های راهنمای Page، فایل MasterPage ای که باید به وسیله ی این صفحه مورد استفاده قرار بگیرد را مشخص کردیم. همچنین خصیصه ی Theme را نیز برابر با Red قرار دادیم تا کنترل ها ظاهر کنترل ها بر طبق آن تنظیم شوند. سپس در داخل قسمتی که به وسیله ی MasterPage برای قرار دادن کنترل ها مشخص شده بود، یک TextBox، یک خط افقی (HR) و یک دکمه قرار دادیم. با قرار دادن این کنترل ها مشاهده کردید که ظاهر آنها همانطور که در فایلهای موجود در فولدر Red مشخص شده بود تنظیم شده است.

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master"
    AutoEventWireup="true"  CodeFile="Default.aspx.cs"
    Inherits="_Default" Theme="Red" %>
<asp:Content ID="Content1"
ContentPlaceHolderID="cphPageContent" Runat="Server">
  <asp:TextBox ID="txtTest" Runat="server">
       Just some text
  </asp:TextBox>
  <hr />
  <br />
  <asp:Button ID="btnTest" Runat="server" Text="Button" />
</asp:Content>
```

در فایل Brown.skin نیز ظاهر کنترلهای Button و TextBox را مشخص کرده ایم. به این ترتیب وقتی برای Theme یک صفحه ی وب از این فایل استفاده شود، این دو کنترل به صورتی که در این قسمت تعریف شده اند نمایش داده خواهند شد.

```
<asp:Button runat="server" ForeColor="Brown"
            Font-Name="Arial" Font-Size="28px"
            Font-Weight="Bold" />
<asp:TextBox runat="server" ForeColor="Brown"
             Font-Name="Arial" Font-Size="28px"
             Font-Weight="Bold" />
```

در صفحه ی News.aspx، فایل MasterPage.master را به عنوان MasterPage مشخص کرده ایم و نیز Theme را برابر با Brown قرار داده ایم. به این ترتیب کنترل TextBox و نیز Button ای که در فرم قرار داده شده است، به رنگ قهوه ای و با فونت درشت نمایش داده می شوند (شکل ۱۷-۱۳).

```
<%@ Page Language="C#" MasterPageFile="~/MasterPage.master"
    AutoEventWireup="true" CodeFile="News.aspx.cs"
    Inherits="News" Title="Untitled Page" Theme="Brown"%>
<asp:Content ID="Content1"
  ContentPlaceHolderID="cphPageContent" Runat="Server">
  <asp:TextBox ID="txtTest" Runat="server">
       Just some text
  </asp:TextBox>
  <asp:Button ID="btnTest" Runat="server" Text="Button" />
```

```
</asp:Content>
```



شکل ۱۷-۱۳

فایل `web.sitemap` نیز به وسیله ی کنترل `SiteMap` مورد استفاده قرار می گیرد. به وسیله ی این کنترل مـی تـوان به کاربر نشان داد که در چه قسمتی از سایت قرار دارد و برای رفتن به آن قسمت از صفحه ی اصلی سایت، از چه لینک هـایی بایـد استفاده کرد. در شکل ۱۷-۱۴ مشاهده می کنید که کاربر در صفحه ی `Today's News` قرار دارد و برای دسترسی به ایـن صفحه باید در صفحه ی اصلی سایت وارد لینک `News` و سپس وارد این لینک شد.



شکل ۱۷-۱۴

استفاده از کنترل GridView برای نمایش داده ها در فرم وب:

در قسمت امتحان کنید بعد، از دو تا از بهترین کنترلهای موجود در ASP.NET 2 استفاده خواهیم کرد: کنترل SqlDataSource و کنترل GridView. در این برنامه از خاصیت ها و خصیصه های موجود در این کنترل ها و نیز کنترلهای زیر مجموعه ی آنها استفاده خواهیم کرد. در این برنامه می خواهیم بدون نوشتن کد های سمت سرور و یا سمت کلاینت، برنامه ی تحت وبی ایجاد کنیم که داده های موجود در بانک اطلاعاتی Pubs را نمایش داده و اجازه دهد که آنها را تغییر دهیم.

امتحان کنید: نمایش داده ها در فرم بدون کد نویسی

۱) با استفاده از ویژوال استودیو، وب سایت جدیدی ایجاد کرده و نام آن را DataGridView قرار دهید.
۲) به نمای Source بروید و کد مشخص شده در زیر را، در فایل Default.aspx وارد کنید. در صورت لزوم عبارتی را که برای ConnectionString در این قسمت مورد استفاده قرار گرفته است را به گونه ای تغییر دهید که بتوانید به وسیله ی آن به سرور بانک اطلاعاتی خود متصل شوید:

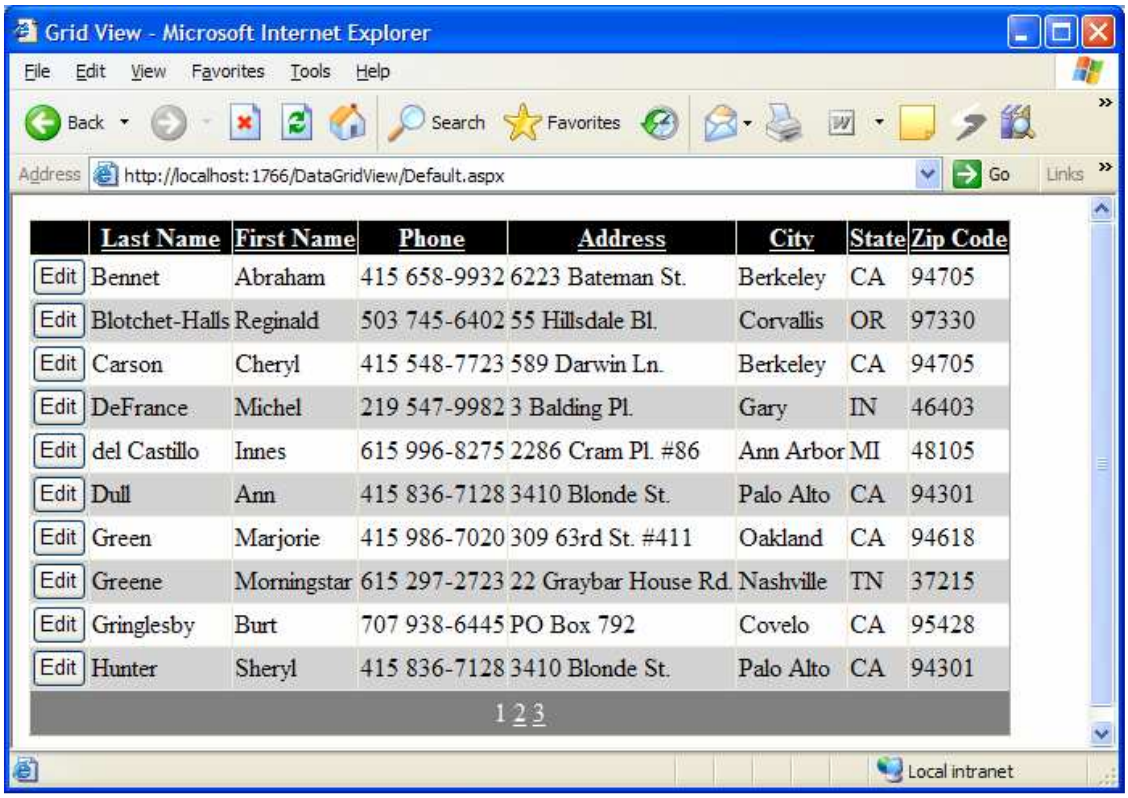
```
<%@ Page Language="C#" AutoEventWireup="true"
CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head runat="server">
    <title>Grid View</title>
  </head>
  <body>
    <form id="form1" runat="server">
      <div>
        <asp:SqlDataSource ID="sdsAuthors" Runat="server"
          ProviderName = "System.Data.SqlClient"
          ConnectionString = "Server=localhost;User ID=sa;
                              Password=;Database=pubs;"
          SelectCommand = "SELECT au_id, au_lname,
                          au_fname, phone,address, city,
                          state, zip FROM authors"
          UpdateCommand = "UPDATE authors SET au_lname =
              @au_lname,au_fname = @au_fname, phone =
              @phone, address = @address,city = @city,
              state = @state, zip = @zip WHERE au_id =
              @au_id" >
          <UpdateParameters>
            <asp:Parameter Type="String"
                Name="au_lname"></asp:Parameter>
            <asp:Parameter Type="String"
```

```
                Name="au_fname"></asp:Parameter>
            <asp:Parameter Type="String"
                Name="phone"></asp:Parameter>
            <asp:Parameter Type="String"
                Name="address"></asp:Parameter>
            <asp:Parameter Type="String"
                Name="city"></asp:Parameter>
            <asp:Parameter Type="String"
                Name="state"></asp:Parameter>
            <asp:Parameter Type="String"
                Name="zip"></asp:Parameter>
            <asp:Parameter Type="String"
                Name="au_id"></asp:Parameter>
          </UpdateParameters>
        </asp:SqlDataSource>
        <asp:GridView ID="gdvAuthors" Runat="server"
          DataSourceID="sdsAuthors" AllowPaging="True"
          AllowSorting="True" AutoGenerateColumns=False
          DataKeyNames="au_id" >
          <PagerStyle BackColor="Gray" ForeColor="White"
                      HorizontalAlign="Center" />
          <HeaderStyle BackColor="Black"
                       ForeColor="White" />
          <AlternatingRowStyle BackColor="LightGray" />
          <Columns>
            <asp:CommandField ButtonType="Button"
                              ShowEditButton="true" />
            <asp:BoundField Visible="false"
                 HeaderText="au_id" DataField="au_id"
                 SortExpression="au_id">
            </asp:BoundField>
            <asp:BoundField HeaderText="Last Name"
                            DataField="au_lname"
                            SortExpression="au_lname">
            </asp:BoundField>
            <asp:BoundField HeaderText="First Name"
                            DataField="au_fname"
                            SortExpression="au_fname">
            </asp:BoundField>
            <asp:BoundField HeaderText="Phone"
                            DataField="phone"
                            SortExpression="phone">
            </asp:BoundField>
            <asp:BoundField HeaderText="Address"
                            DataField="address"
                            SortExpression="address">
            </asp:BoundField>
```

```
            <asp:BoundField HeaderText="City"
                            DataField="city"
                            SortExpression="city">
            </asp:BoundField>
            <asp:BoundField HeaderText="State"
                            DataField="state"
                            SortExpression="state">
            </asp:BoundField>
            <asp:BoundField HeaderText="Zip Code"
                            DataField="zip"
                            SortExpression="zip">
            </asp:BoundField>
          </Columns>
        </asp:GridView>
      </div>
    </form>
  </body>
</html>
```

۳) با فشار کلیدهای Ctrl+F5 برنامه را اجرا کنید. مشاهده خواهید کرد که صفحه ی وب همانند شکل ۱۷-۱۵ در مرورگر نمایش داده می شود.

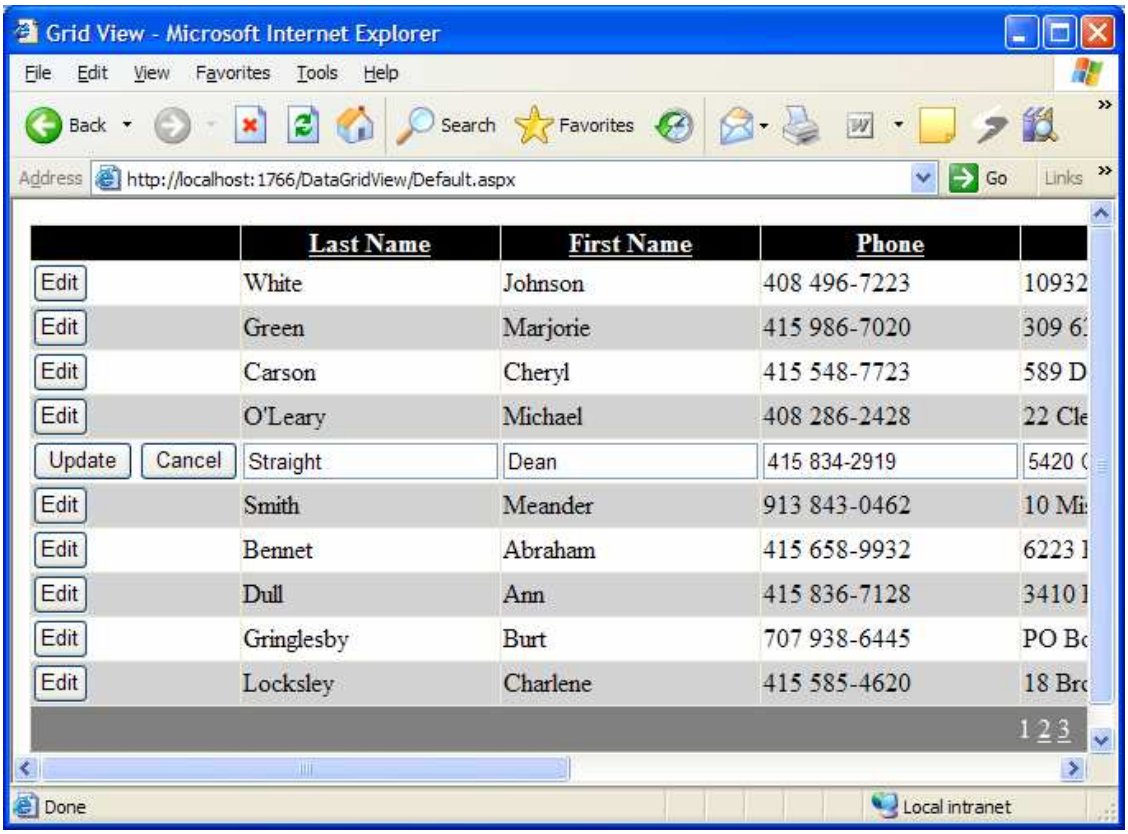


	Last Name	First Name	Phone	Address	City	State	Zip Code
Edit	Bennet	Abraham	415 658-9932	6223 Bateman St.	Berkeley	CA	94705
Edit	Blotchet-Halls	Reginald	503 745-6402	55 Hillsdale Bl.	Corvallis	OR	97330
Edit	Carson	Cheryl	415 548-7723	589 Darwin Ln.	Berkeley	CA	94705
Edit	DeFrance	Michel	219 547-9982	3 Balding Pl.	Gary	IN	46403
Edit	del Castillo	Innes	615 996-8275	2286 Cram Pl. #86	Ann Arbor	MI	48105
Edit	Dull	Ann	415 836-7128	3410 Blonde St.	Palo Alto	CA	94301
Edit	Green	Marjorie	415 986-7020	309 63rd St. #411	Oakland	CA	94618
Edit	Greene	Morningstar	615 297-2723	22 Graybar House Rd.	Nashville	TN	37215
Edit	Gringlesby	Burt	707 938-6445	PO Box 792	Covelo	CA	95428
Edit	Hunter	Sheryl	415 836-7128	3410 Blonde St.	Palo Alto	CA	94301

1 2 3

شکل ۱۷-۱۵

مشاهده می کنید که داده های مورد نظر شما در جدولی در فرم نمایش داده شده است. اما این اطلاعات تمـام داده هـای مورد نظر شما نیستند، بلکه در پایین جدول لینک هایی وجود دارد که به وسیله ی آنها می توانید صفحات دیگر داده ها را مشاهده کنید. همچنین می توانید با کلیک کردن روی نام هر ستون، داده ها را بر اساس آن سـتون بـه صـورت صـعودی مرتب کنید. برای مرتب سازی داده ها به صورت نزولی نیز کافی است که بار دیگر روی نام آن ستون کلیک کنیـد. بـرای ویرایش اطلاعات می توانید روی دکمه ی `Edit` که در سمت چپ هر ردیف قرار داده شده است کلیک کنید. مـشاهده خواهید کرد که جدول همانند شکل ۱۷-۱۶ به گونه ای تغییر خواهد کرد که بتوانید داده های آن سطر را ویرایش کنید:



شکل ۱۷-۱۶

می توانید داده های مورد نظر خود را تغییر داده و سپس روی دکمه ی `Update` کلیک کنید تا تغییرات ذخیره شـوند. برای لغو تغییراتی که ایجاد کرده اید نیز می توانید از دکمه ی `Cancel` استفاده کنید.

چگونه کار می کند؟

متوجه شدید چه قدر ساده بود؟ فقط با اضافه کردن دو کنترل، یک صفحه ی وب بسیار قوی برای دسترسی به داده های یک جدول از بانک اطلاعاتی اضافه کردیم. خوب، بهتر است بررسی کنیم که این برنامه (و در واقع این کنترل ها) چگونه کار می کنند؟

نکته: ممکن است کدی که در قسمت قبلی برای ایجاد کردن این دو کنترل نوشته بودیم مقداری مشکل و پیچیده به نظر برسد، اما همانطور که در برنامه ی قبلی نیز گفتم می توانیم به جای وارد کردن این کدها با استفاده از جعبه ابزار این کنترل ها را به فرم اضافه کرده، سپس از پنجره ی Properties برای تنظیم خاصیتهای آن استفاده کنیم. اما در اینجا برای اینکه به سرعت کنترلهای مورد نیاز را ایجاد کنیم و نیز بیشتر با نوشتن کد در قسمت Source آشنا شویم، از کد معادل آنها استفاده کردیم.

ابتدا با استفاده از تگ asp:SqlDataSource، یک کنترل SqlSataSource در فرم برنامه قرار می دهیم. جدول زیر لیستی از خصیصه های این کنترل که در این برنامه آنها را تنظیم کرده و یا تغییر داده ایم آورده شده است.

خصیصه یا تگ درونی	شرح
ID	شناسه ی کنترل را تعیین می کند.
Runat	مشخص می کند که کد های مربوط به رویدادهای این کنترل باید در سرور اجرا شوند.
ProviderName	فضای نام سرویس دهنده ی اطلاعاتی که می خواهیم برای اتصال به بانک اطلاعاتی از آن استفاده کنیم را مشخص می کند. در اینجا از فضای نام System.Data.SqlClient استفاده می کنیم. از برنامه های فصل قبل به خاطر دارید که این فضای نام برای دسترسی به بانکهای اطلاعاتی تحت SQL Server مورد استفاده قرار می گیرد.
ConnectionString	این متن نحوه ی اتصال به بانک اطلاعاتی مورد نظر را مشخص می کند. (در قسمت امتحان کنید پایان این فصل، روش بهتری را برای ذخیره ی ConnectionString مشاهده خواهیم کرد).
SelectCommand	شامل دستور SQL ای است که برای دریافت داده ها از بانک اطلاعاتی و نمایش آن در فرم مورد استفاده قرار می گیرد. همچنین در این قسمت می توانیم از نام یک پروسیجر ذخیره شده در بانک اطلاعاتی استفاده کنیم.
UpdateCommand	شامل دستور SQL ای است که برای تغییر داده های موجود در بانک اطلاعاتی مورد استفاده قرار می گیرد. در این خصیصه نیز می توان به جای استفاده از دستور SQL، نام یک پروسیجر ذخیره شده در بانک اطلاعاتی را قرار داد.
تگ UpdateParameters و Parameters	تگ UpdateParameters شامل آرایه ای از اشیایی از نوع Parameter است و برای معین کردن مقادیر لازم برای Placeholder های موجود در دستورات SQL مورد استفاده قرار می گیرد. از برنامه های ویندوزی که در قسمت قبل نوشتیم به خاطر دارید که برای مثال پارامتر @City که در دستور Update مورد استفاده قرار گرفته است، قبل از اجرای دستور در بانک اطلاعاتی باید به وسیله ی مقدار آن جایگزین شود تا بتوانیم داده های این رکورد را تغییر دهیم. مقدار این پارامتر به وسیله ی عبارتی که کاربر در کادر City وارد می کند معین می شود. بنابراین لازم است شیئ ای (تگی) از نوع Parameter ایجاد کرده و به وسیله ی آن مقدار این پارامتر را در دستور

Update مشخص کنیم.

Parameter:Type	این خصیصه از تگ Parameter، نوع داده ای که باید به جای آن پارامتر قرار بگیرد را مشخص می کند. در این برنامه نوع تمام پارامترهای دستور Update از نوع String است.
Parameter:Name	این خصیصه مشخص می کند که شیئ Parameter متناظر با آن مربوط به کدامیک از placeholderهای موجود در دستور Update است.

```
<asp:SqlDataSource ID="sdsAuthors" Runat="server"
  ProviderName = "System.Data.SqlClient"
  ConnectionString = "Server=localhost;User ID=sa;
                      Password=;Database=pubs;"
  SelectCommand = "SELECT au_id, au_lname,
        au_fname, phone,address, city,
        state, zip FROM authors"
  UpdateCommand = "UPDATE authors SET au_lname =
      @au_lname,au_fname = @au_fname, phone =
      @phone, address = @address,city = @city,
      state = @state, zip = @zip WHERE au_id =
      @au_id" >
  <UpdateParameters>
    <asp:Parameter Type="String"
         Name="au_lname"></asp:Parameter>
    <asp:Parameter Type="String"
         Name="au_fname"></asp:Parameter>
    <asp:Parameter Type="String"
         Name="phone"></asp:Parameter>
    <asp:Parameter Type="String"
         Name="address"></asp:Parameter>
    <asp:Parameter Type="String"
         Name="city"></asp:Parameter>
    <asp:Parameter Type="String"
         Name="state"></asp:Parameter>
    <asp:Parameter Type="String"
         Name="zip"></asp:Parameter>
    <asp:Parameter Type="String"
         Name="au_id"></asp:Parameter>
  </UpdateParameters>
</asp:SqlDataSource>
```

کنترل دومی که باید به فرم برنامه اضافه کنیم، کنترل GridView است. خصیصه ها و خاصیتهای مهم این کنترل در جدول زیر آمده است:

خصیصه یا تگ درونی	شرح

ID	شناسه ی کنترل را تعیین می کند.
Runat	مشخص می کند که کد های مربوط به رویدادهای این کنترل باید در سرور اجرا شوند.
DataSourceID	شناسه ی کنترل SqlDataSourceای را نگهداری می کند که داده های آن باید به وسیله ی این کنترل نمایش داده شوند.
AllowPaging	تعیین کننده ی این است که جدول داده ها دارای خاصیت صفحه بندی باشد یا نه؟ (با فعال کردن این خاصیت، اگر تعداد داده ها از عدد مشخصی بیشتر بود، مابقی در صفحات دیگر قرار می گیرند).
AllowSorting	تعیین کننده ی این است که جدول داده ها دارای خاصیت مرتب سازی باشد یا نه؟
AutoGenerateColumns	مشخص می کند که آیا ستونهای جدول به صورت اتوماتیک و بر اساس داده های موجود ایجاد شود یا برنامه نویس به صورت دستی ستونها را در جدول ایجاد می کند؟
DataKeyNames	نام ستونی که در جدول اصلی داده ها در بانک اطلاعاتی به عنوان کلید اصلی (Primary Key) مورد استفاده قرار گرفته است.
PagerStyle	با تنظیم خصیصه های این تگ، می توان استیل ناحیه ی مربوط به لینک صفحات دیگر جدول را مشخص می کند.
HeaderStyle	با تنظیم خصیصه های این تگ، می توان استیل مربوط به ناحیه ی عنوانهای ستونها را مشخص کرد.
AlternatingRowStyle	با تنظیم خصیصه های این تگ می توان استیل ردیفهای زوج را تعیین کرد. به این وسیله می توان با متمایز کردن ظاهر ردیفهای زوج و فرد، خواندن داده ها در صفحه را ساده تر کرد.
Columns	یک آرایه از ستون هایی که این جدول را تشکیل می دهند را نگه داری می کند. این آرایه می تواند شامل اشیایی (تگ هایی) از نوع BoundField، ButtonField، CheckBoxField و یا ... باشد.
CommandField	به وسیله ی این شیئ می توان ستونی ایجاد کرد که وظایف خاصی را انجام دهد، برای مثال بتوان به وسیله ی آن یک سطر از داده ها را ویرایش کرد، یک سطر از داده ها را حذف کرد یا آن را انتخاب کرد. برای مشخص کردن نوع دکمه ای که باید در این ستون قرار بگیرد می توان از خاصیت ButtonType استفاده کرد. این خاصیت می تواند یکی از مقادیر Link، Image و یا Button را دریافت کند.
BoundField	این شیئ (تگ) برای اتصال ستونهای مختلف داده به ستونها جدول مورد استفاده قرار می گیرد. برای اینکه جدول ظاهر بهتری پیدا کند، خاصیت Visible ستون مربوط به کلید اصلی را برابر با False قرار می دهیم تا در جدول نمایش داده نشود. همچنین خاصیت SortExperssion هر ستون را

نیز تنظیم می کنیم تا مشخص شود که هنگام مرتب سازی داده ها بر اساس یک ستون، از چه عبارتی باید استفاده کرد. این کار باعث می شود که نام ستون به صورت یک لینک نمایش داده شود. سپس با تنظیم خصیصه ی HeaderText، عنوانی که باید برای هر ستون نمایش داده شود را نیز تعیین می کنیم. در آخر نیز با استفاده از خصیصه ی DataField، ستون را به یکی از فیلد های داده هایی که از بانک اطلاعاتی دریافت شده است متصل می کنیم.

```
<asp:GridView ID="gdvAuthors" Runat="server"
  DataSourceID="sdsAuthors" AllowPaging="True"
  AllowSorting="True" AutoGenerateColumns=False
  DataKeyNames="au_id" >
  <PagerStyle BackColor="Gray" ForeColor="White"
              HorizontalAlign="Center" />
  <HeaderStyle BackColor="Black"
               ForeColor="White" />
  <AlternatingRowStyle BackColor="LightGray" />
  <Columns>
    <asp:CommandField ButtonType="Button"
                      ShowEditButton="true" />
    <asp:BoundField Visible="false"
          HeaderText="au_id" DataField="au_id"
          SortExpression="au_id">
    </asp:BoundField>
    <asp:BoundField HeaderText="Last Name"
                    DataField="au_lname"
                    SortExpression="au_lname">
    </asp:BoundField>
    <asp:BoundField HeaderText="First Name"
                    DataField="au_fname"
                    SortExpression="au_fname">
    </asp:BoundField>
    <asp:BoundField HeaderText="Phone"
                    DataField="phone"
                    SortExpression="phone">
    </asp:BoundField>
    <asp:BoundField HeaderText="Address"
                    DataField="address"
                    SortExpression="address">
    </asp:BoundField>
    <asp:BoundField HeaderText="City"
                    DataField="city"
                    SortExpression="city">
    </asp:BoundField>
    <asp:BoundField HeaderText="State"
```

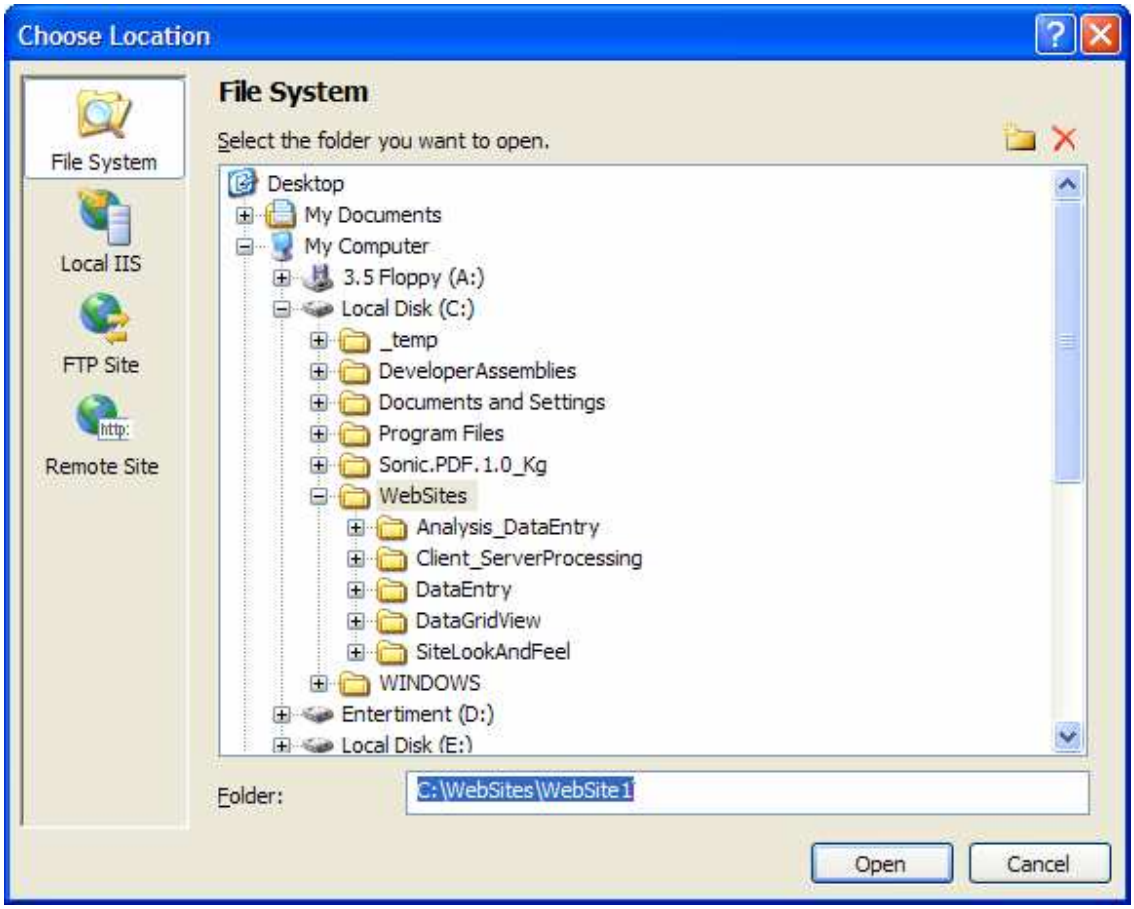
```
                    DataField="state"
                    SortExpression="state">
        </asp:BoundField>
        <asp:BoundField HeaderText="Zip Code"
                    DataField="zip"
                    SortExpression="zip">
        </asp:BoundField>
      </Columns>
    </asp:GridView>
```

نکته: ممکن است با توضیحاتی که در این قسمت در مورد تنظیم خصیصه های کنترلهای ایجاد شده گفته شد، مقداری در مورد این خصیصه ها و رابطه ی آنها با خاصیتهایی که در پنجره ی Properties قابل تنظیم است سردرگم شوید. همانطور که در برنامه های تحت ویندوز مشاهده کردیم هر کنترل در حقیقت یک کلاس است و زمانی که یک نمونه از آن را در فرم برنامه قرار می دهیم، مدر واقع یک نمونه از آن کلاس ایجاد کرده ایم. البته این کار توسط ویژوال استودیو به صورت اتوماتیک انجام می شد. یعنی زمانی که کنترلی را در فرم قرار می دادیم و سپس خاصیتهای آن را با استفاده از پنجره ی Properties تنظیم می کردیم، ویژوال استودیو در فایلی با پسوند .designer.cs شیئ مربوط به آن کنترل را ایجاد کرده و خاصیتهای آن را با توجه به تنظیمات ما، تغییر می داد.

در اینجا نیز شرایطی مشابه قبل برقرار است. به این صورت که هر کنترل در واقع یک شیئ است و دارای خاصیتهای مربوط به خود است. در اینجا هنگامی که یک کنترل را در سایت قرار می دهیم، ویژوال استودیو تگ مربوط به ایجاد آن را ایجاد می کند، همچنین به ازای خاصیت هایی که در پنجره ی Properties تنظیم می کنیم، ویژوال استودیو خصیصه هایی را تنظیم کرده و مقدار آن ها را برابر با مقادیر تعیین شده قرار می دهد. همانطور که در برنامه های ویندوزی یک کلاس می تواند فیلدی از نوع یک کلاس دیگر داشته باشد، کنترلهای موجود در این قسمت نیز می توانند فیلدی از نوع یک کلاس دیگر و یا فیلدی آرایه ای داشته باشند. در این صورت اگر بخواهیم مقادیر آنها را تنظیم کنیم، ویژوال استودیو درون تگ مربوط به کنترل، تگ دیگری ایجاد می کند که مشخص کننده ی شیئ ای از کلاس درونی است. سپس هنگامی که این صفحه از سایت بخواهد در وب سرور اجرا شود، ASP.NET وظیفه دارد که کلاسهای مربوط به این تگها را ایجاد کرده و همچنین با توجه به خصیصه های تنظیم شده، مقادیر خاصیت ها را تنظیم کند.

محل قرارگیری یک برنامه ی تحت وب در ویژوال استودیو:

هنگام ایجاد وب سایتهایی که در طول این فصل تمرین کردیم، همانند شکل ۱۷-۱۷ کادر Location را برابر با File System قرار دادیم تا سایت، در دیسک کامپیوتری که در حال استفاده از آن هستیم ایجاد شود. یکی از مزایای این کار در این است که در طی این مدت تا زمانی که طراحی سایت تمام نشده باشد، کاربران نمی توانند به آن دسترسی داشته باشند. اما همواره بهتر است قبل از اینکه سایت را در یک وب سرور روی اینترنت قرار دهیم، در یک سرور IIS واقعی تست کنیم.

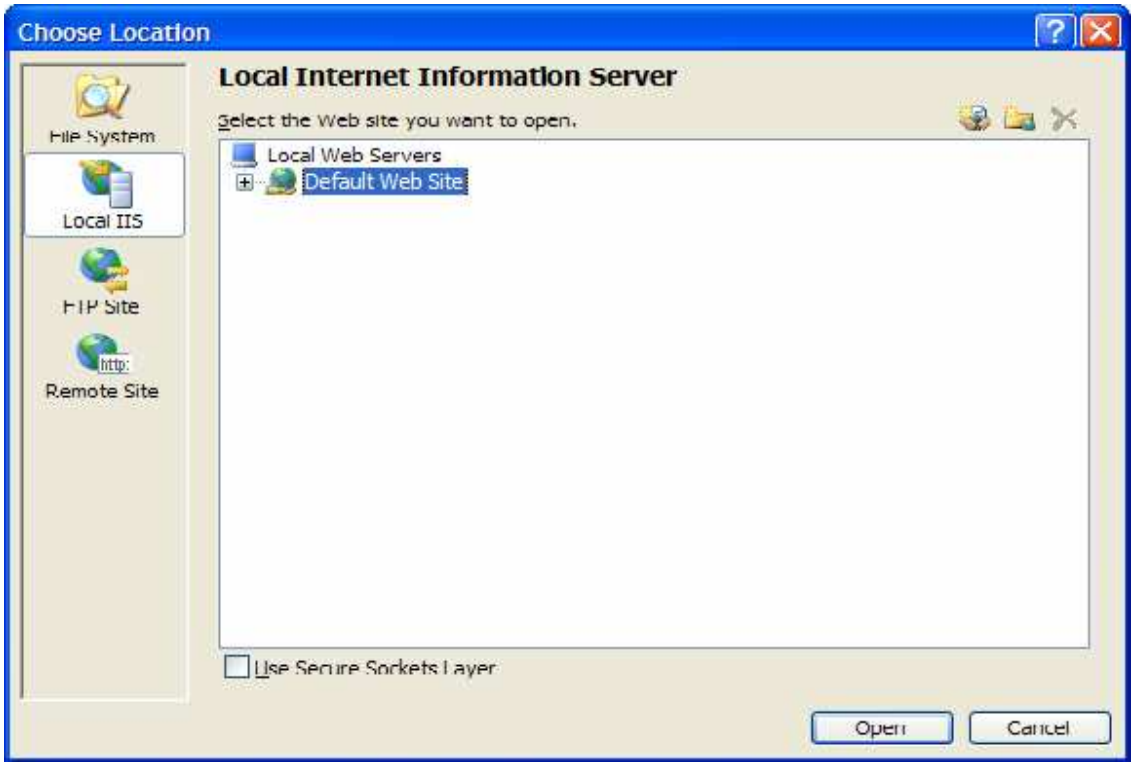


شکل ۱۷-۱۷

هنگام ایجاد یک سایت وب، سه محل دیگر نیز برای قرارگیری آن وجود دارد که می توانید از آنها استفاده کنید. برای مشاهده ی این سه محل در کادر `New Web Site` روی دکمه ی `Browse` کلیک کنید تا کادر `Choose Location` نمایش داده شود. در سمت چپ این کادر می توانید نوع مکانی که می خواهید سایت قرار گیرد را مشخص کنید. اولین انتخاب در وب سرور محلی یا `IIS` ای است که در کامپیوتر شما (کامپیوتری که به وسیله ی آن سایت را طراحی می کنید) نصب شده است. البته این گزینه زمانی می تواند مورد استفاده قرار گیرد که وب سرور `IIS` در آن کامپیوتر نصب شده باشد. در این صورت کادری مشابه شکل ۱۷-۱۸ نمایش داده می شود.

انتخاب دوم استفاده از یک سایت `FTP` برای قرار دادن فایلهای مربوط به برنامه در آن است. در این حالت، احتمالا باید از امکانات یک شرکت سرویس دهنده استفاده کنید. در این صورت تمام کاری که برای استفاده از این حالت باید انجام دهید این است که آدرس و نیز اطلاعات مربوط به ورود به آن سرور `FTP` را در پنجره ای که در شکل ۱۷-۱۹ نشان داده شده است وارد کنید.

گزینه ی آخر نیز استفاده از یک سایت اینترنتی است که می توانید به این وسیله، سایت خود را در آن قرار داده و سپس طراحی آن را آغاز کنید.



شکل ۱۷-۱۸



شکل ۱۷-۱۹

نتیجه:

در این فصل با برنامه نویسی مبتنی بر وب آشنا شدیم. در رابطه با مزایا و نیز معایب برنامه های ویندوزی و برنامه های تحت وب صحبت کردیم و سعی کردیم با درک تفاوت این دو، تشخیص دهیم در چه شرایطی باید از کدامیک از آنها استفاده کرد. احتمالا مهمترین علتی که ممکن است باعث شود به جای ایجاد برنامه های ویندوزی از برنامه های مبتنی بر وب استفاده کنید، کم بودن هزینه ی توزیع و نیز به روز رسانی این نوع برنامه ها است. سپس با قسمتهای مختلف یک برنامه ی مبتنی بر وب آشنا شدیم و سعی کردیم کارایی هر یک از آنها را در برنامه مشاهده کنیم. همچنین با کنترل هایی که در ASP.NET 2 معرفی شده اند و باعث سادگی و سرعت طراحی برنامه ها می شدند نیز آشنا شدیم. در آخر نیز با استفاده از این کنترل ها برنامه ای ایجاد کردیم که بدون نوشتن حتی یک خط کد، داده ها را از بانک اطلاعاتی دریافت کرده و در فرم نمایش دهد و همچنین بتوانیم آنها را تغییر دهیم. البته نوشتن برنامه های مبتنی بر وب بسیار گسترده تر از مباحثی است که در این فصل گفته شد و خود به چندین کتاب نیاز دارد. بنابراین برای آشنایی بیشتر با این مبحث می توانید به کتابهای ASP.NET مراجعه کنید.
در پایان این فصل باید با موارد زیر آشنا شده باشید:

- برای ایجاد یک برنامه تشخیص دهید که بهتر است از برنامه های ویندوزی استفاده کنید یا برنامه های تحت وب.
- از جعبه ابزار ASP.NET 2 استفاده کنید.
- با استفاده از ویژوال استودیو ۲۰۰۵ یک پروژه ی سایت تحت وب ایجاد کنید.
- از صحت داده هایی که در یک فرم وب وارد می شود مطمئن شوید.
- با استفاده از Theme، MasterPage و نیز کنترلهای مربوط به حرکت در بین صفحات سایت ظاهر برنامه را کنترل و مدیریت کنید.
- با استفاده از کنترل GridView داده های مورد نیاز را در فرمهای وب نمایش دهید.
- در بین مکان هایی که برای قرار دادن فایلهای برنامه ی سایت در اختیار دارید یکی را انتخاب کرده و از آن استفاده کنید.

تمرین:

برنامه ی DataGridView را باز کنید. یک راه بهتر برای ذخیره اطلاعات مربوط به اتصال به بانک اطلاعاتی این است که از فایل web.config برای نگهداری ConnectionString استفاده کنید. برای این تمرین باید متن ConnectionString را در فایل web.config ذخیره کرده و هنگامی که بخواهید با استفاده از آن خاصیت SqlDataSource را تنظیم کنید، آن را از این فایل استخراج کنید.
ابتدا با استفاده از پنجره ی Solution Explorer یک فایل web.config به برنامه اضافه کرده، سپس قسمت <appsettings/> را به صورت زیر تغییر دهید تا اطلاعاتی مربوط به ConnectionString برنامه ی شما در این فایل قرار بگیرد.

```
<appSettings>
  <add key="ConnectionString"
    value="Server=localhost;User ID=sa;
       Password=;Database=pubs;"/>
</appSettings>
```

حال خط مربوط به تنظیم خصیصه ی ConnectionString را از تعریف sdsAuthors حذف کنید.
در آخر نیز باید یک رویداد به فایل Default.aspx اضافه کنید، و در متد مربوط به آن خاصیت ConnectionString از sdsAuthors را برابر با مقدار آن در web.config قرار دهید. برای ایجاد رویداد مورد نیاز، کنترل sdsAuthors را از نمای Design انتخاب کرده و سپس با استفاده از پنجره ی Properties بر رویداد Init دو بار کلیک کنید تا متد مربوط به آن ایجاد شود.
برای دسترسی به ConnectionString در فایل web.config نیز می توانید از کد زیر استفاده کنید:

```
ConfigurationManager.AppSettings["ConnectionString"]
```

فصل هجدهم: تشخیص هویت در برنامه های تحت وب

در فصل هفدهم با نحوه های ایجاد برنامه های تحت وب آشنا شدیم. در این فصل می خواهیم در مورد نحوه ی ایجاد امنیت در این نوع برنامه ها صحبت کنیم و سعی می کنیم یک وب سایت امن ایجاد کنیم. در این فصل نیز مانند فصل قبل بدون اینکه کدی در برنامه وارد کنیم، یک سایت با ظاهری قابل قبول و دارای قابلیت تشخیص هویت افرادی که از آن استفاده می کنند ایجاد خواهیم کرد.
در این فصل:

- با دو روش کلی برای تامین امنیت یک سایت وب آشنا خواهیم شد.
- نحوه ی استفاده از ابزار مدیریت سایت وب (WAT) را مشاهده خواهیم کرد.
- با استفاده از تعیین هویت مبتنی بر فرم، یک سایت ایمن ایجاد خواهیم کرد.
- با نوشتن مقدار کمی کد، و یا حتی بدون آن یک وب سایت امن ایجاد خواهیم کرد.

تشخیص هویت در یک سایت وب:

اگر بخواهید یک برنامه ی تحت وب ایجاد کنید، یکی از مواردی که از همان ابتدای کار باید مد نظر داشته باشید امنیت آن برنامه است. در طراحی هر یک از قسمت های یک برنامه ی تحت وب، باید این نکته را در نظر داشته باشید که چه کاربرانی می توانند به این قسمت دسترسی داشته باشند و چه کاربرانی نمی توانند وارد این قسمت شوند. در بیشتر سایتهای وب، قسمتی از سایت به گونه ای طراحی می شود که همه ی افراد بتوانند از آن استفاده کنند و قسمتی دیگر نیز فقط در دسترس اعضای سایت قرار دارد. بنابراین باید در ابتدا هویت فردی که در حال استفاده از سایت است مشخص شود، تا بتوانیم در مواقع مورد نیاز تصمیم گیری کنیم که آیا کاربر می تواند از این قسمت استفاده کند یا نه؟ برای تعیین هویت کاربران در برنامه های تحت وب دو روش وجود دارد: تشخیص هویت با استفاده از ویندوز و تشخیص هویت با استفاده از فرم های وب.

تشخیص هویت با استفاده از ویندوز:

ساده ترین روش تشخیص هویت کاربران یک سایت وب، استفاده از سیستم امنیتی ویندوز است[1]. این روش به خصوص در سایتهایی که در اینترانت ها و یا شبکه های LAN اجرا می شوند بسیار پر کاربرد است و در حقیقت به وسیله ی IIS صورت می گیرد. به وسیله ی این سیستم، مکانیسم مربوط به تشخیص هویت کاربران سایت از قسمتهای مربوط به طراحی و ایجاد سایت و یا اجرای آن روی شبکه کاملاً مجزا می شود.
در این نوع تشخیص هویت قبل از اینکه IIS به درخواست یک کاربر پاسخ داده و صفحه ی مورد نظر او را نمایش دهد، ابتدا بررسی می کند که آیا هویت کاربر در سیستم مشخص شده است یا نه؟ اگر هویت کاربر تعیین نشده بود، IIS پنجره ای را به کاربر نمایش می دهد تا نام کاربری و کلمه ی عبور اکانت خود در سرور را وارد کند تا اجازه ی استفاده از سایت به او داده شود. اما

[1] این روش به Windows Authentication معروف است.

اگر هویت کاربر مشخص شده بود (برای مثال در درخواستهای قبلی کاربر، اطلاعات مربوط به تعیین هویت او توسط IIS دریافت شده بود) بدون اینکه ای پنجره ای به کاربر نمایش داده شود، صفحه ی مورد نظر او نمایش داده خواهد شد.

تشخیص هویت با استفاده از فرمهای وب:

روش قبلی معمولاً برای سایتهای عمومی که در اینترنت مورد استفاده قرار می گیرند کاربرد ندارد، زیرا به طور معمول تعداد کاربران در این حالت زیاد است و نمی توان برای هر کاربری که می خواهد از سایت استفاده کند یک اکانت در سرور ایجاد کرد. در این موارد یکی از ساده ترین روشهای تعیین هویت کاربر استفاده از فرمهای وب است[1].
روش کار این سیستم به این صورت است که هویت هر کاربری که می خواهد از سایت استفاده کند، قبل از آنکه بتواند وارد صفحات اصلی سایت شود باید مشخص شود تا اجازه ی استفاده از سایت به او داده شود. هنگامی که یک کاربر که هویت او برای سایت مشخص نیست بخواهد از برنامه استفاده کند، ابتدا به وسیله ی سایت به صفحه ی ورود هدایت می شود. برای مثال اگر کاربری که نام کاربری و کلمه ی عبور خود را برای سایت مشخص نکرده است آدرس صفحه ی Home.aspx را در مرورگر خود وارد کند تا اطلاعات آن را مشاهده کند، سایت به صورت اتوماتیک به جای اینکه صفحه ی Home.aspx را به او نشان دهد او را به صفحه ی ورود راهنمایی می کند. در این صفحه کاربر باید نام کاربری و کلمه ی عبور خود را وارد کرده (و یا در صورت نیاز یک اکانت کاربری جدید برای خود ایجاد کند) تا هویت او برای سایت مشخص شود. سپس صفحه ی Home.aspx برای او نمایش داده خواهد شد. در ASP.NET 2 کنترل های درونی زیادی وجود دارد که به وسیله ی آن می توان به سادگی سایتهای امنی با استفاده از این روش ایجاد کرد.

ابزار مدیریت سایت وب (WAT):

تنظیمات یک سایت وب معمولاً در فایلی به نام web.config قرار می گیرند. در نسخه ی قبلی ASP.NET طراحان سایت مجبور بودند تا با استفاده از XML به صورت دستی کد مربوط به تنظیمات قسمتهای مختلف سایت، از قبیل نحوه ی خطا یابی، امنیت و یا ... را در این فایل وارد کنند. اما در ASP.NET 2 یک رابط گرافیکی برای انجام تنظیمات مختلف در این فایل ایجاد شده است. این رابط گرافیکی ابزار مدیریت سایت وب یا WAT[2] نام دارد.
هنگامی که WAT را اجرا کنید، در ابتدا پنج قسمت مختلف در آن مشاهده خواهید کرد که عبارتند از: Home، Security، Profile، Application و Provider. در این فصل فقط با استفاده از قسمت Security تنظیمات مربوط به امنیت یک سایت وب را انجام خواهیم داد و از قسمتهای دیگر زیاد استفاده نخواهیم کرد.
در بخش امتحان کنید زیر، یک سایت وب شامل چندین صفحه ایجاد خواهیم کرد و سپس با استفاده از WAT روش تعیین هویت مبتنی بر فرم را برای امنیت سایت به کار خواهیم برد.

امتحان کنید: تنظیمات مربوط به تشخیص هویت مبتنی بر فرم

[1] این روش به Forms Authentication معروف است.

[2] Web Site Administration Tool

۱) با استفاده از ویژوال استودیو یک وب سایت جدید به نام TheClub ایجاد کنید. مطمئن شوید که سایت جدید شما در مکان FileSystem ایجاد می شود.

۲) با استفاده از Solution Explorer تغییرات زیر را در سایت به وجود آورید. برای اضافه کردن یک آیتم جدید به سایت، با کلیک روی نام پروژه در پنجره ی Solution Explorer گزینه ی Add New Item را انتخاب کنید. در کادری که نمایش داده می شود نوع آیتمی که می خواهید به سایت اضافه کنید را مشخص کرده (Web Form، Text File و یا ...) و سپس با استفاده از کادر Name، نام فایل را تعیین کنید. بعد از اتمام این مرحله و اضافه کردن صفحات مورد نیاز، پنجره ی Solution Explorer شما باشد مشابه شکل ۱-۱۸ باشد. توجه داشته باشید که هنگام اضافه کردن فرمهای وب به برنامه ی خود، گزینه ی Place code in separate file را از حالت انتخاب خارج کنید.

ابتدا یک فایل از نوع MastrePage به نام Main.mastert به برنامه اضافه کرده و هنگام اضافه کردن فرمهای وب به برنامه، آنها را به گونه ای تنظیم کنید که از این فایل به عنوان MasterPage خود استفاده کنند. برای این کار در کادر Add New Item گزینه ی Select master page را انتخاب کنید.

ابتدا دو فولدر زیر را به فولدر اصلی سایت اضافه کنید:

- Admin
- Members

روی نام پروژه در Solution Explorer کلیک راست کرده و گزینه ی Add ASP.NET Folder ➔ Theme را انتخاب کنید. نام فولدری که در فولدر App_Theme ایجاد می شود را نیز برابر با MainTheme قرار دهید.

فرمهای وب زیر را به فولدر اصلی سایت اضافه کنید (به خاطر داشته باشید که گزینه ی Select master page را انتخاب کنید):

- Login.aspx
- ChangePassword.aspx
- CreateNewUser.aspx

روی فرم Login.aspx کلیک کرده و گزینه ی Set as Start Page را انتخاب کنید تا این فرم به عنوان فرم اول برنامه مشخص شود.

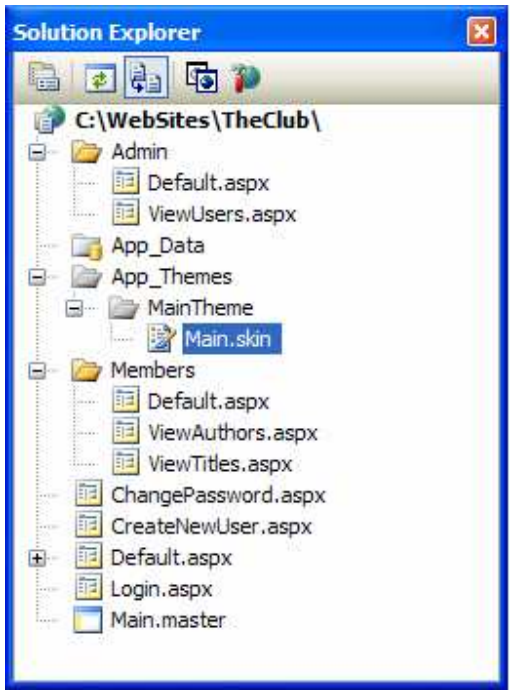
فرمهای زیر را به فولدر Admin اضافه کنید (به خاطر داشته باشید که در کادر Add New Item گزینه ی Select master page را انتخاب کنید):

- Default.aspx
- ViewUsers.aspx

فرمهای زیر را نیز به فولدر Members اضافه کنید:

- Default.aspx
- ViewAuthors.aspx
- ViewUsers.aspx

فایلی از نوع Skin File با نام Main.skin را نیز به فولدر MainTheme اضافه کنید.



شکل ۱۸-۱

۳) با انتخاب گزینه ی ASP.NET Configuration ➔ Website در نوار منوی ویژوال استودیو WAT را اجرا کنید. این منو در شکل ۱۸-۲ نشان داده شده است.

۴) به این ترتیب ابزار WAT همانند یک صفحه ی وب معمولی در مرورگر اینترنتی شما نمایش داده خواهد شد. در شکل ۱۸-۳ صفحه ی اول این ابزار را مشاهده می کنید. در این فصل از این ابزار برای تنظیمات امنیتی برنامه استفاده خواهیم کرد.

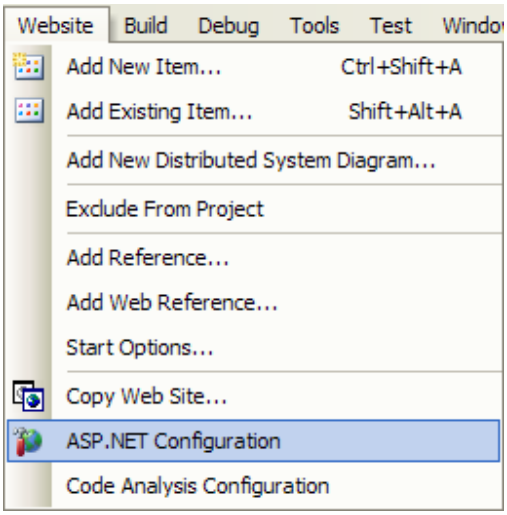
۵) حال روی لینک Security کلیک کنید تا تنظیمات مربوط به امنیت سایت را مشخص کنیم.

۶) در این قسمت می خواهیم از یک ویزارد استفاده کرده و تمام تنظیمات مربوط به امنیت سایت را با استفاده از آن مشخص کنیم. بنابراین در صفحه ی Security روی لینک Use the Security Setup Wizard کلیک کنید تا وارد ویزارد شویم.

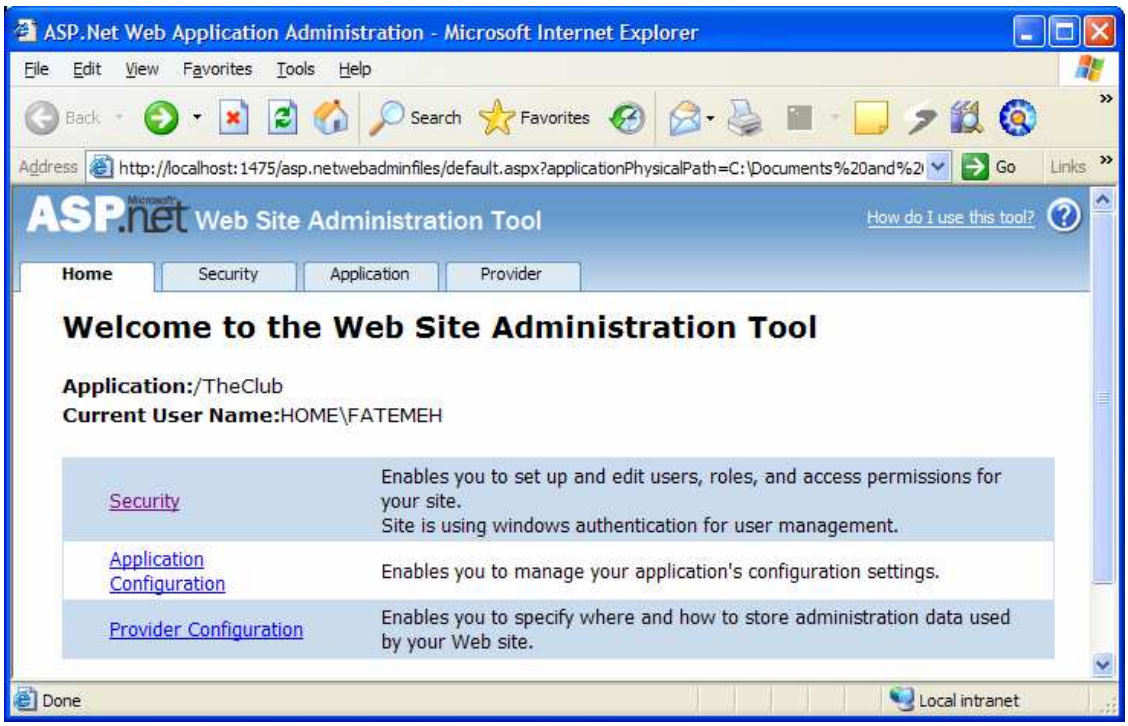
۷) ویزارد Security Setup شامل هفت مرحله است. اولین مرحله صفحه ی Welcome است که در آن درباره ی کل مراحل ویزارد توضیح مختصری داده می شود. در پایین سمت راست صفحه دکمه هایی وجود دارند که به وسیله ی آنها می توانید بین قسمتهای مختلف ویزارد حرکت کنید. روی کلید Next کلیک کنید تا به مرحله ی دوم از این ویزارد برویم.

۸) در مرحله ی دوم ویزارد باید نحوه ی دسترسی کاربران به برنامه را مشخص کنید. همانطور که در شکل ۱۸-۴ نمایش داده شده است در این قسمت دو انتخاب وجود دارد. انتخاب اول "From the Internet" است و مشخص می کند که سایت شما در اینترنت قرار خواهد گرفت و تمام کاربران اینترنت می توانند به آن دسترسی داشته باشند. در این صورت همانطور که توضیح داده شد، بهتر است از روش تشخیص هویت به وسیله ی فرمهای وب برای تامین امنیت برنامه استفاده کنید. بنابراین با انتخاب این گزینه، این روش برای تامین امنیت برنامه ی وب شما مورد

استفاده قرار می گیرد. در این روش کاربران می توانند در سایت شما اکانتی را برای خود ایجاد کرده و سپس با اسـتفاده از آن اکانت از سایت شما استفاده کنند. اطلاعات مربوط به اکانت هر کاربر نیز در یک بانک اطلاعاتی ذخیره می شود.



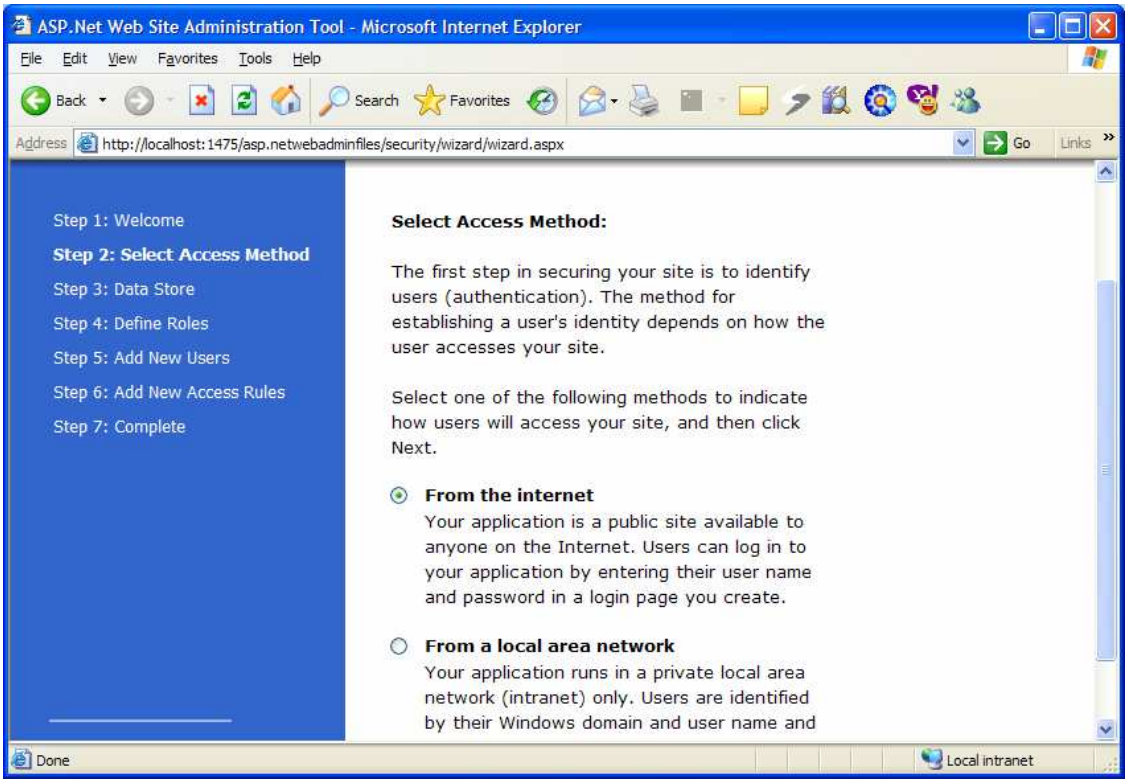
شکل ۱۸-۲



شکل ۱۸-۲

انتخاب دوم "From a Local Area Network" است و مشخص می کند که این سـایت در یـک شبکه ی کامپیوتر محلی قرار خواهد گرفت. بنابراین در صورت انتخاب این گزینه از سیستم تشخیص هویت ویندوز بـرای

تامین امنیت برنامه استفاده خواهد شد. برای سایت TheClub گزینه ی اول را انتخاب کرده و روی دکمه ی Next کلیک کنید تا به مرحله ی سوم برویم.



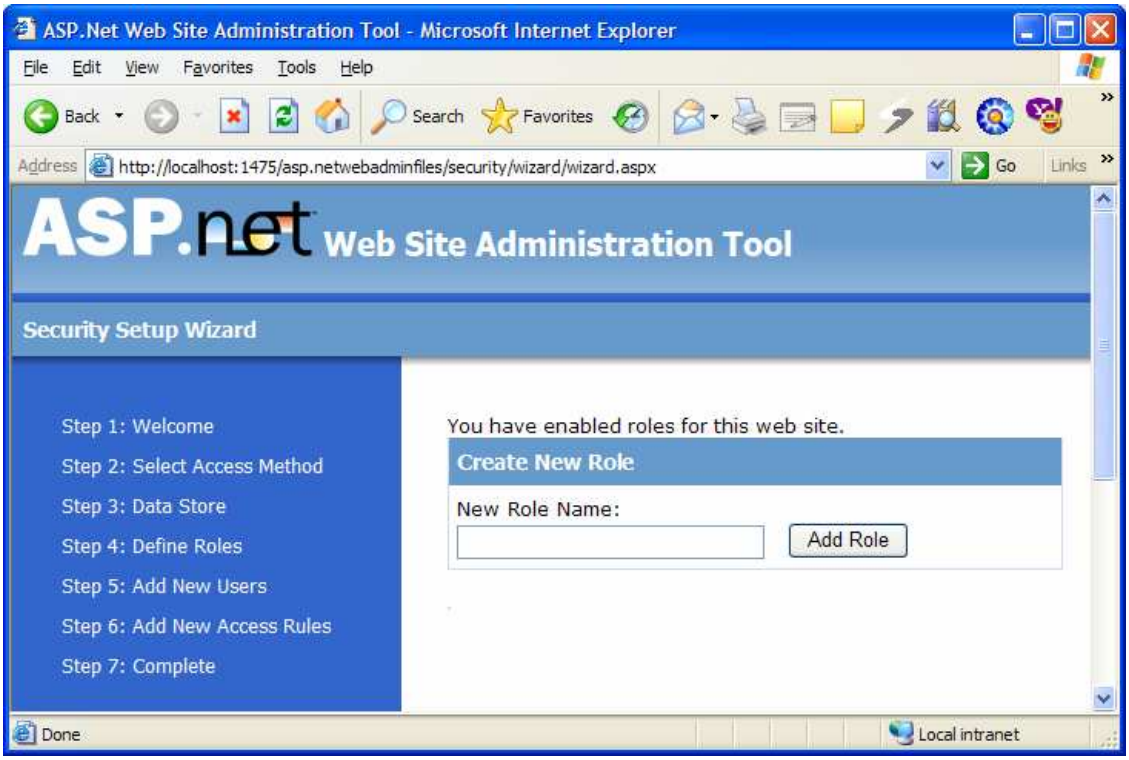
شکل ۱۸-۴

۹) در مرحله ی سوم اطلاعاتی در مورد بانک اطلاعاتی که برای ذخیره ی داده های مربوط به اکانت کاربران به کار می رود نمایش داده می شود. همانطور که در این قسمت مشاهده می کنید برای ذخیره ی داده ها در این قسمت از سرویس دهنده ی داده ای پیش فرض استفاده شده است. برای تغییر این قسمت باید از ویزارد خارج شده و در صفحه ی اصلی WAT به قسمت Provider بروید. در آن قسمت می توانید سرویس دهنده ی داده ای جدیدی را برای برنامه معرفی کنید. برای این برنامه تنظیمات پیش فرض را قبول کرده و روی دکمه ی Next کلیک کنید تا به مرحله ی بعد بروید.

۱۰) در مرحله ی چهارم می توانید امنیت مبتنی بر نقش[1] را برای این سایت فعال کنید. به این صورت می توانید نقشهای مختلفی را برای افرادی که از سایت استفاده می کنند تعریف کرده و مشخص کنید که چه گروه افرادی می توانند به هر صفحه دسترسی داشته باشند. برای مثال می توانند گروه هایی مانند گروه مدیریت، گروه کاربران عادی و ... را تعریف کرده و سپس مشخص کنید که برای مثال گروه کاربران عادی به صفحات A، B و C دسترسی دارند. گروه مدیریت به تمام صفحات دسترسی دارند و .... در این صفحه گزینه ی Enable role for this Website را انتخاب کرده و روی دکمه ی Next کلیک کنید تا یک نقش جدید تعریف کنیم. به این ترتیب صفحه ی Create New Role همانند شکل ۱۸-۵ نمایش داده خواهد شد. در کادر عبارت Admin را به عنوان نام Role وارد کنید و سپس روی دکمه ی Add Role کلیک کنید. در صفحه ی بعد می توانید گروه ایجاد شده را

[1] Role-Based Security

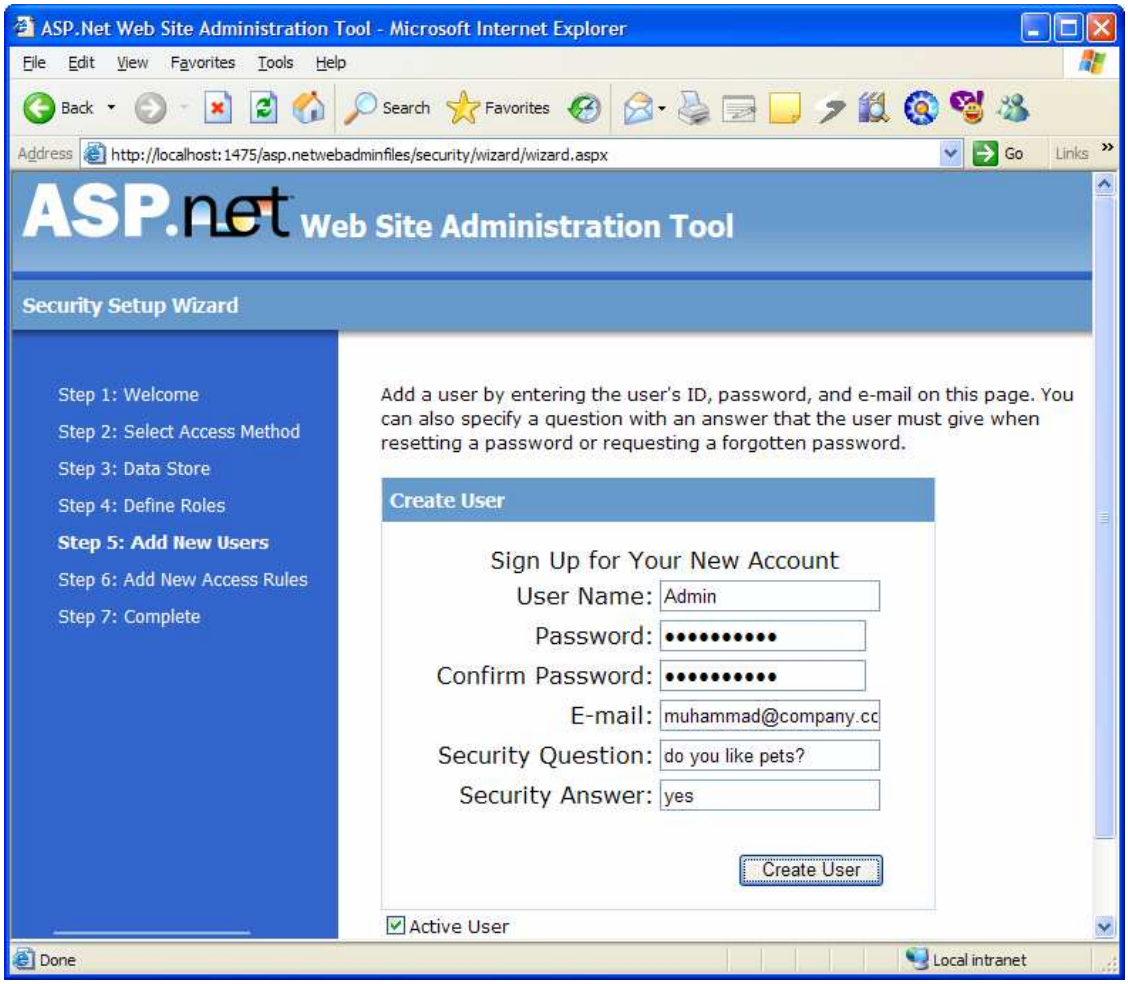
ویرایش کرده و یا گروهها های دیگری اضافه کنید. اما برای این برنامه همین گروه کافی است. پس روی دکمه ی Next کلیک کنید تا به مرحله ی پنجم بروید.



شکل ۱۸-۵

۱۱) در مرحله پنجم می توانید کاربران مورد نظر خود را ایجاد کنید. البته ضرورتی ندارد که این کار را حتماً در این قسمت انجام دهید، اما اگر می خواهید تعدادی کاربر را به برنامه اضافه کنید با استفاده از این قسمت می توانید به سادگی این کار را انجام دهید. برای این برنامه یک کاربر در گروه Admin همانند شکل ۱۸-۶ ایجاد کنید. نام کاربری او را Admin قرار دهید و بقیه ی فیلد ها را نیز به دلخواه خود تکمیل کنید، فقط مقادیر آنها را به خاطر داشته باشید. بعد از اتمام کار روی دکمه ی Create User کلیک کنید تا کاربر جدید ایجاد شود. با این کار پیغامی در صفحه نمایش داده شده و عنوان می کند که کاربر مورد نظر شما با موفقیت ایجاد شده است. با کلیک روی دکمه ی Continue می توانید کاربران دیگری ایجاد کنید. اما در این برنامه می خواهیم فقط یک کاربر داشته باشیم، بنابراین روی دکمه ی Next کلیک کنید تا به مرحله ی بعد برویم.

۱۲) آخرین مرحله قبل از اتمام این ویزارد مرحله ی ششم، یعنی Add New Access Role است. در این قسمت باید مشخص کنید که چه افرادی به چه قسمتهایی از سایت می توانند دسترسی داشته باشند. برای این برنامه سه قاعده را مشخص می کنیم. به خاطر داشته باشید که مواردی که برای سطح دسترسی کاربران مختلف در این قسمت تعریف می کنید، فقط می توانند به فولدر ها نسبت داده شوند، نه به فایلها. بنابراین برای مثال می توانید دسترسی کاربران عادی به یک فولدر از سایت که مخصوص مدیران است را محدود کنید و یا اجازه ندهید که افرادی که تعیین هویت نشده اند به صفحات وب موجود در فولدرهای مخصوص کاربران وارد شوند. بنابراین هنگامی که می خواهید در این قسمت یک قاعده ی مربوط به نحوه ی دسترسی را برای گروه خاصی مشخص کنید، ابتدا اطمینان حاصل کنید که فولدر مورد نظر خود را درست انتخاب کرده اید.

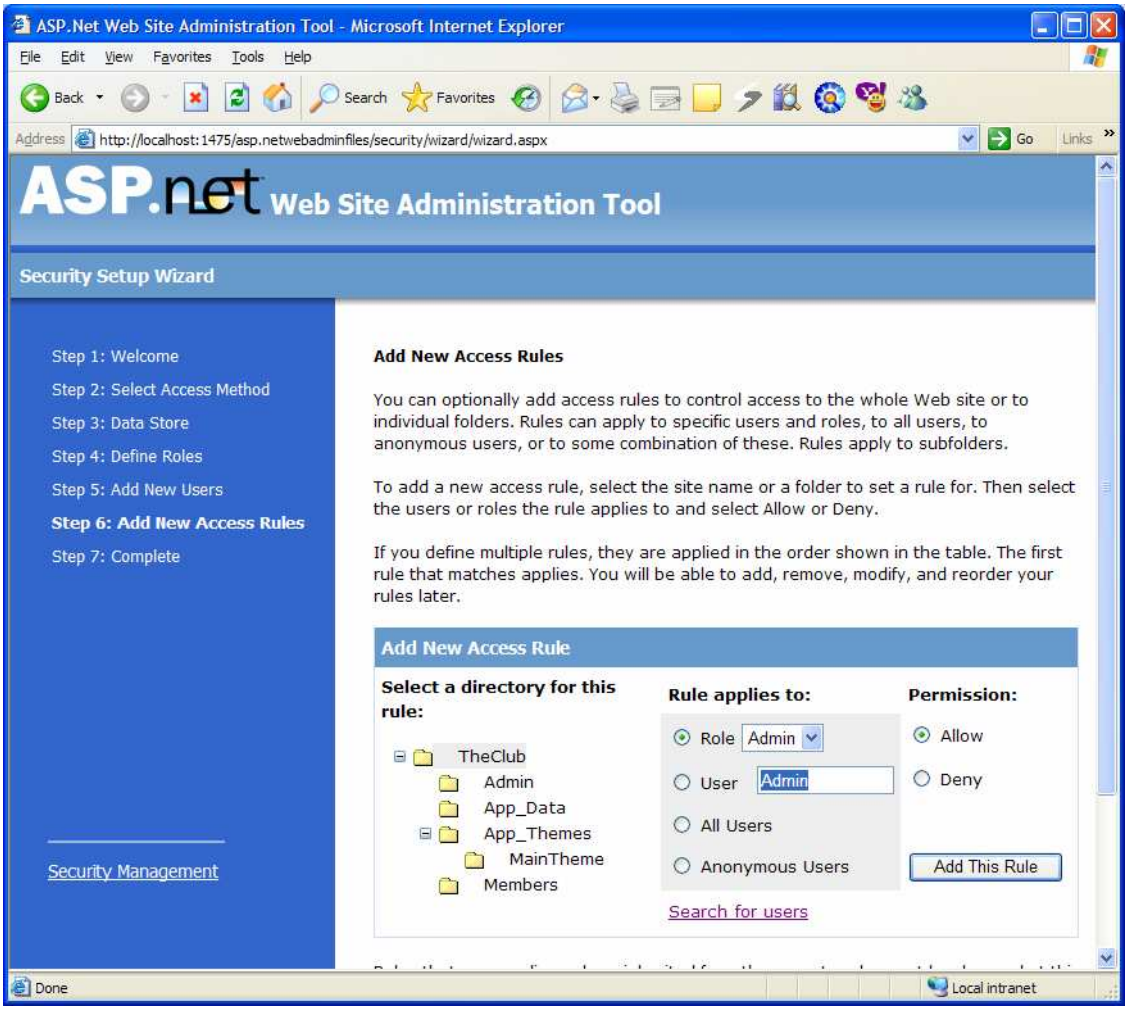


شکل ۱۸-۶

همانطور که مشاهده می کنید، در این قسمت پیش فرض به این صورت است که تمام افراد، چه افرادی که تعیین هویت شده اند و چه افرادی که تعیین هویت نشده اند، به تمام قسمتهای سایت دسترسی داشته باشند. حال می خواهیم در این قسمت یک شرط قرار دهیم. ابتدا مطمئن شوید که فولدر Admin از سمت چپ انتخاب شده است، سپس از گزینه های ردیف وسط گزینه ی Role را انتخاب کنید. در کادر مقابل این گزینه عبارت Admin را انتخاب کرده و در قسمت Permission روی Allow کلیک کنید. سپس دکمه ی Add This Role را فشار دهید تا این شرط به برنامه اضافه شود. دو قاعده ی دیگر مربوط به دسترسی به سایت را در قسمتهای بعدی اضافه می کنیم، بنابراین در این قسمت روی دکمه ی Next کلیک کنید تا به مرحله ی بعد برویم. در مرحله ی هفتم نیز اطلاعاتی در مورد تنظیماتی که در این ویزارد انجام داده ایم نمایش داده می شود. در این قسمت روی دکمه ی Finish کلیک کنید. به این ترتیب به صفحه ی اولیه ی قسمت Securityبر خواهیم گشت تا بقیه ی تنظیمات را انجام دهیم.

۱۳) در صفحه ی اصلی قسمت Security روی لینک Manage Access Rules کلیک کنید. در پنجره ی بعد روی فولدر Admin کلیک کنید تا قاعده ای که در قسمت قبلی اضافه کرده بودیم نمایش داده شود. حال در این صفحه روی لینک Add New Access Rule کلیک کنید. در این قسمت قاعده ای را به برنامه اضافه خواهیم کرد تا از دسترسی تمام افراد به فولدر Admin جلوگیری شود. به این ترتیب فقط افرادی که در گروه

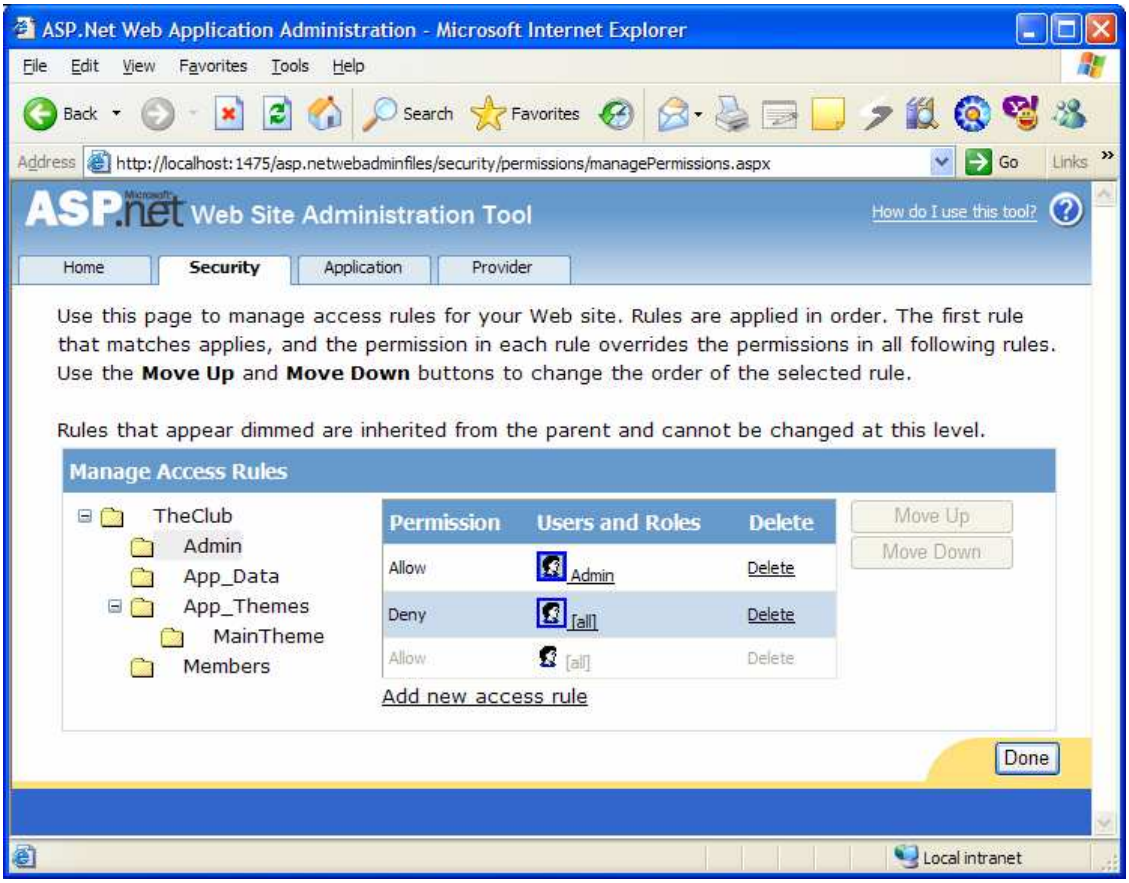
Admin باشند به صفحات این فولدر دسترسی خواهند داشت. این قواعد را به گونه ای به بالا و یا پایین حرکت دهید تا مشابه شکل ۱۸-۸ شوند.



شکل ۱۸-۷

۱۴) در سمت چپ این قسمت روی فولدر Members کلیک کنید تا انتخاب شود. سپس قاعده ای را به آن اضافه کنید تا از دسترسی افرادی که تعیین هویت نشده اند جلوگیری شود. برای این کار از ستون Rule applies to گزینه ی Anonymous Users و از ستون Permission گزینه ی Deny را انتخاب کنید. به این ترتیب قواعد تعریف شده برای فولدر Members همانند شکل ۱۸-۹ خواهد بود.

۱۵) حال می توانیم تنظیمات امنیتی که ایجاد کرده ایم را امتحان کنیم. البته مهم نیست که صفحات خالی هستند، زیرا در این قسمت فقط می خواهیم تنظیمات را تست کنیم.

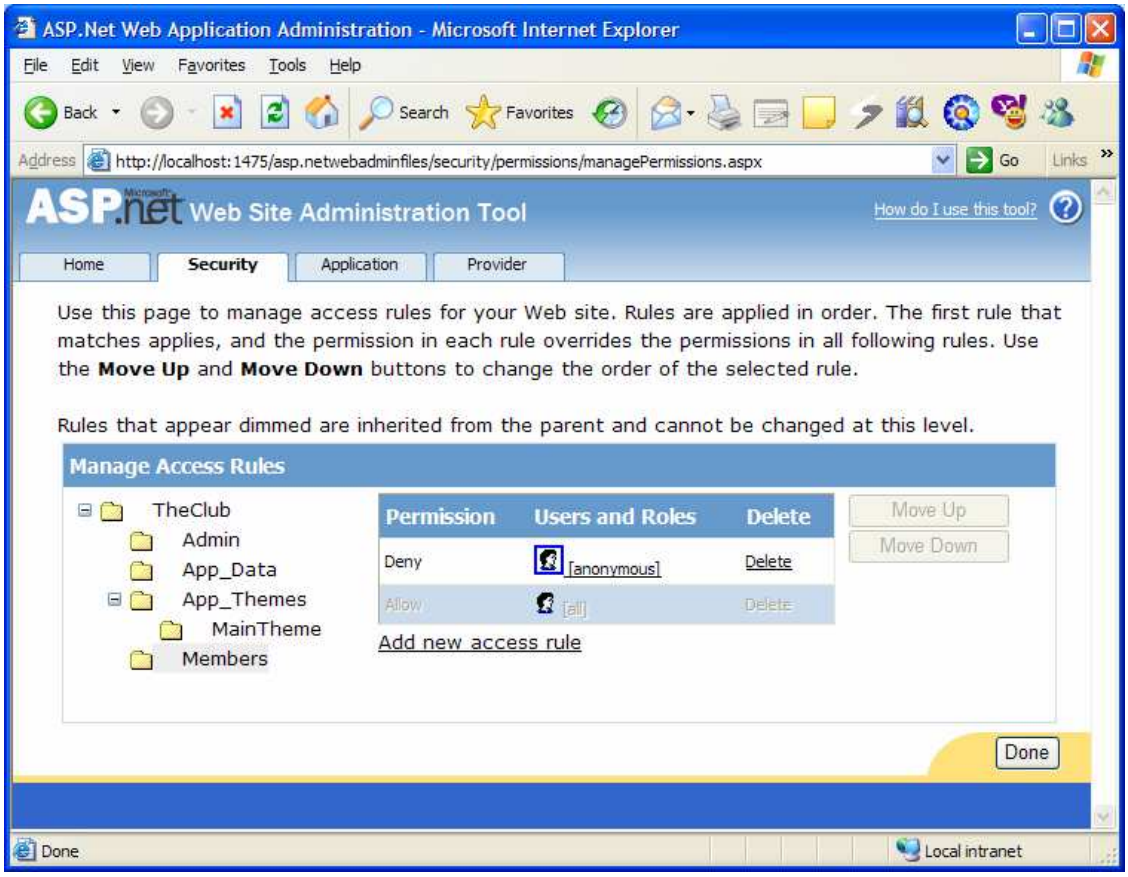


شکل ۱۸-۸

برای تست سایت کلید F5 را فشار دهید تا برنامه اجرا شود و به صفحه ی اصلی سایت برویم. به خاطر دارید که صفحه ی Login.aspx را به عنوان صفحه ی اول مشخص کردیم، پس آدرس صفحه ی اول سایت به صورت http://localhost/theClub/Login.aspx خواهد بود. با فشار کلید F5 پیغامی نمایش داده خواهد شد و از شما می پرسد که آیا می خواهید ویژگی های مربوط به خطا یابی را برای این سایت فعال کند؟ در این کادر گزینه ی Add a New Web.config File with Debugging Enabled و سپس روی دکمه ی OK کلیک کنید تا سایت اجرا شود. به آدرس نمایش داده شده در بالای مرورگر توجه کنید. مشاهده خواهید کرد که در فایل Login.aspx قرار دارید. آدرس بالا را تغییر دهید و به جای فایل Login.aspx، عبارت Members/Default.aspx را وارد کنید. مشاهده خواهید کرد که آدرسی که به آن هدایت خواهید شد، آدرس مورد نظر شما نخواهد بود. یعنی به جای اینکه مرورگر فایل Default.aspx در فولدر Members را نمایش دهد، مجدداً به فایل Login.aspx برخواهد گشت و آدرسی که در مرورگر نمایش داده می شود، چیزی مشابه آدرس زیر خواهد بود:

```
http://localhost:1463/TheClub/login.aspx?ReturnUrl=%2fTheClub%2fMembers%2fdefault.aspx
```

۱۶) اگر سعی کنید که با استفاده از نوار آدرس، به یکی از صفحات موجود در فولدر Admin بروید نیز با همین مشکل مواجه خواهید شد.

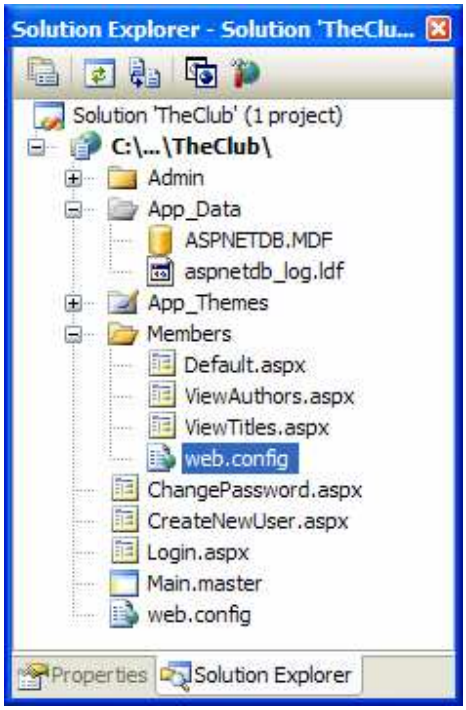


شکل ۱۸-۹

چگونه کار می کند؟

خوب، فکر کنم مهمترین قسمت این تمرین ویزاردی بود که از آن استفاده کردیم و باید آن را بررسی کنیم. اما بهتر است ابتدا به پنجره ی Solution Explorer نگاهی بیندازیم. ابتدا با فشار دادن کلید Refresh در این پنجره، فایلهای موجود را دوباره نمایش دهید. این پنجره مشابه شکل ۱۸-۱۰ خواهد بود. اگر با دقت بیشتری نگاه کنید خواهید دید که یک فایل به نام web.config و نیز یک بانک اطلاعاتی Access به برنامه اضافه شده است. این دو فایل تنظیمات امنیتی که در ویزارد مشخص کردید را نگهداری و کنترل می کنند.

تنظیمات یک سایت در فایل web.config آن ذخیره می شوند. هنگامی که با استفاده از ویزارد تنظیمات هر فولدر را مشخص می کردید، کد XML مربوط به این تنظیمات در فایل web.config موجود در هر فولدر ذخیره می شد. بنابراین اگر یکی از این فایلها را باز کنید، مشاهده می کنید که تنظیماتی که برای آن قسمت ایجاد کرده بودید، در آن ذخیره شده است. همچنین یک فایل Access نیز به برنامه اضافه می شود تا اطلاعات مربوط به کاربرانی که در برنامه تعریف می شوند و نیز گروه هایی که ایجاد می شوند در آن نگهداری شود.



شکل ۱۸-۱۰

به آدرسی که هنگام رفتن به صفحه ی `Members/Default.aspx` به آن فرستاده شدید، نگاه دقیقتری بیندازید.

```
http://localhost:1463/TheClub/login.aspx?ReturnUrl=%2fTheClub%2fMembers%2fdefault.aspx
```

همانطور که مشاهده می کنید در آدرس این فایل یک علامت سوال (?) وجود دارد. این علامت سوال در آدرس یک فایل، شروع شدن یک `QueryString` را اعلام می کند. در یک سایت وب به دلیل نحوه ی ایجاد شدن و نیز از حافظه خارج شدن صفحات، نمی توان به صورت عادی داده هایی را بین دو صفحه ی وب منتقل کرد. `QueryString`ها یک روش برای ارسال داده بین دو صفحه ی وب به شمار می رود. این روش به این صورت است که داده ای که می خواهیم از یک صفحه به صفحه ی دیگر منتقل کنیم را در آدرس آن صفحه قرار می دهیم. البته نمی توان هر داده ای را از این روش منتقل کرد، بلکه باید ابتدا آن را به رشته تبدیل کرد. البته به علت اینکه بعضی کاراکتر ها نمی توانند در این قسمت نمایش داده شوند، معادلی برای آنان در نظر گرفته می شود. برای مثال معادل کاراکتر `/`، عبارت `%2f` است. بنابراین اگر در رشته ای که در مقابل `ReturnUrl` آمده است، عبارت `%2f` را با `/` جایگزین کنید به رشته ی زیر میرسید:

```
/TheClub/Members/Default.aspx
```

همانطور که مشاهده می کنید، این رشته آدرسی است که سعی داشتیم به آن برویم. در حقیقت سرور به این وسیله به صفحه ی `Login.aspx` می گویید که اگر کاربر توانست با موفقیت وارد سیستم شود، او را به این آدرس بفرست.
خوب، تا اینجا یک سایت با تنظیمات امنیتی مورد نظر ایجاد کردیم. بهتر است در ادامه کنترلهای مربوط به ورود به سایت که به صورت درونی در `ASP.NET 2` وجود دارند را بررسی کنیم.

کنترلهای Login:

تیم برنامه نویسی ASP، بسیاری از کارهایی که معمولاً بیشتر برنامه نویسان وب مجبور بودند برای طراحی سایت خود انجام دهند را پیشاپیش انجام داده اند و در قالب چندین کنترل برای انجام کارهای مربوط به تعیین هویت، در ASP.NET 2 قرار داده اند تا کار طراحان تحت وب را تا حد ممکن ساده کنند. در طراحی یک سایت می توانید به صورت عادی از این کنترل ها استفاده کنید، و یا با تنظیم قسمتهای مختلف آنها، ظاهر و کارایی این کنترل ها را به گونه ی مطلوب تغییر دهید. جدول زیر لیستی از این کنترل ها را به همراه توضیحاتی در مورد آنها نمایش می دهد. البته در طراحی یک برنامه مجبور نیستید که از این کنترل ها استفاده کنید و می توانید با استفاده از توابع و کلاس هایی که در .NET وجود دارد، خودتان صفحاتی را برای این موارد تعیین کنید.

نام کنترل	توضیح
Login	این کنترل تمام عناصری که برای ایجاد یک صفحه ی ورود در یک سایت لازم است را داراست.
LoginView	دارای قسمتهایی برای نمایش اطلاعات در مورد کاربری که وارد سیستم شده است می باشد.
LoginStatus	لینکی را برای وارد شدن به سیستم و یا خارج شدن از آن در اختیار کاربر قرار می دهد.
LoginName	نام کاربری فردی که هم اکنون وارد سیستم شده است را نمایش می دهد.
ChangePassword	به کاربران اجازه می دهد تا کلمه ی عبور خود را تغییر دهند.
CreateUserWizard	محدوده ای را در برنامه ایجاد می کند تا کاربران در آن بتوانند اکانت جدیدی را برای خود ایجاد کنند.
PasswordRecovery	کلمه ی عبور جدید کاربر را به ایمیل او ارسال می کند. نکته: ایمیل راه امنی برای ارسال داده های مهمی مانند کلمه ی عبور به شمار نمی رود. بنابراین بهتر است هنگام استفاده از این کنترل در برنامه، اهمیت کلمه ی عبور کاربر نیز در نظر گرفته شود تا در صورت لزوم از ابزار دیگری برای این کار استفاده شود.

در بخش امتحان کنید بعد، از بیشتر این کنترل ها در برنامه استفاده خواهیم کرد تا سایت قبلی را کاملتر کنیم.

امتحان کنید: کنترلهای Login

۱) حال که امنیت مورد نظر را در سیستم ایجاد کردیم، باید کارایی و ظاهر برنامه را نیز به صورت قابل قبولی در آوریم. در این قسمت سعی می کنیم با ایجاد ظاهر سایت، هم مهارتهایی را که در فصل هفدهم به دست آوردیم مرور کنیم و هم نحوه ی استفاده از کنترلهای Login را مشاهده کنیم. برای اتمام این قسمت، تغییرات مشخص شده در زیر را در فایل Main.master ایجاد کنید. با تایپ کردن این کدها، بیشتر با خاصیت تکمیل خودکار متن به وسیله ی ویژوال

استودیو آشنا خواهید شد و می توانید به سادگی خاصیت ها و خصیصه های مورد نظر خود را تنظیم کنید. با انجام این کار متوجه خواهید شد که این روش بسیار سریعتر از اضافه کردن کنترل ها به وسیله ی جعبه ابزار است.

```
<%@ Master Language="C#" %>

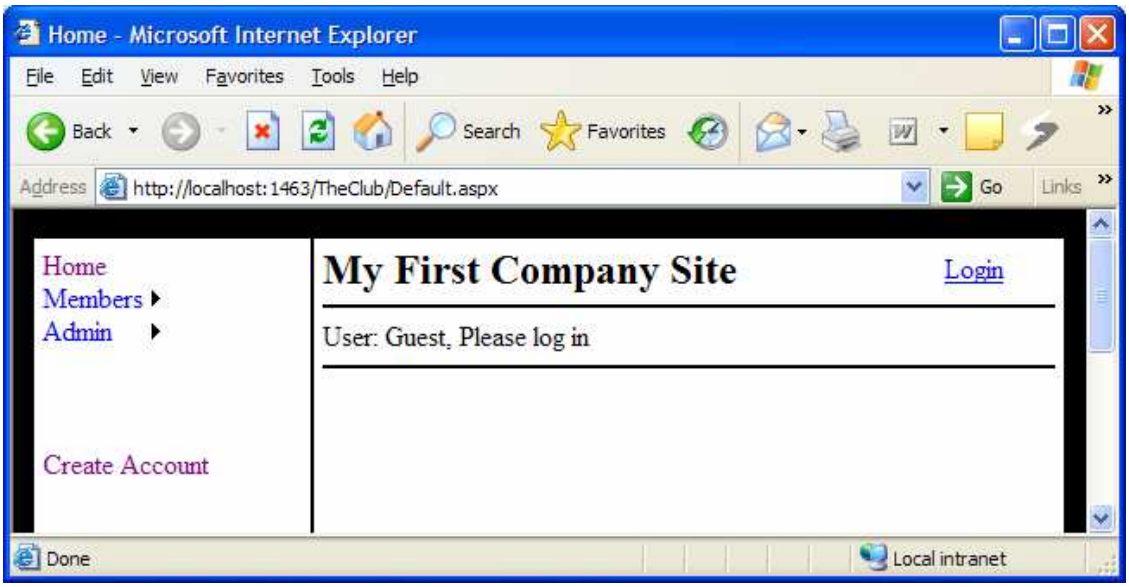
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
                      "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
  <script runat="server">
  </script>
  <html xmlns="http://www.w3.org/1999/xhtml" >
    <head runat="server">
      <title>Untitled Page</title>
    </head>
    <body bgcolor="black">
    <form id="form1" runat="server">
      <div>
        <table cellpadding="5" cellspacing="0" width="600" height="400"
               bgcolor="white" border="1" bordercolor="black">
          <tr>
            <td width="150" valign="top">
              <!-- Menu Column -->
              <asp:Menu ID="Menu1" Runat="server">
                <Items>
                  <asp:MenuItem NavigateUrl="~/Default.aspx"
                                Text="Home">
                  </asp:MenuItem>
                  <asp:MenuItem Text="Members">
                    <asp:MenuItem
                           NavigateUrl="Members/ViewAuthors.aspx"
                           Text="View Authors">
                    </asp:MenuItem>
                    <asp:MenuItem NavigateUrl="Members/ViewTitles.aspx"
                                  Text="View Titles">
                    </asp:MenuItem>
                  </asp:MenuItem>
                  <asp:MenuItem Text="Admin">
                    <asp:MenuItem NavigateUrl="Admin/ViewUsers.aspx"
                                  Text="View Users">
                    </asp:MenuItem>
                  </asp:MenuItem>
                </Items>
              </asp:Menu>
              <br />
              <br />
              <br />
              <asp:LoginView ID="LoginView1" Runat="server">
                <AnonymousTemplate>
                  <asp:Menu ID="Menu2" Runat="server">
                    <Items>
                      <asp:MenuItem NavigateUrl="~/CreateNewUser.aspx"
                                    Text="Create Account">
                      </asp:MenuItem>
                    </Items>
                  </asp:Menu>
                </AnonymousTemplate>
```

```
                <LoggedInTemplate>
                  <asp:Menu ID="Menu3" Runat="server">
                    <Items>
                      <asp:MenuItem NavigateUrl="~/ChangePassword.aspx"
                                    Text="Change Password">
                      </asp:MenuItem>
                    </Items>
                  </asp:Menu>
                </LoggedInTemplate>
              </asp:LoginView>
            </td>
            <td valign="top">
              <table cellpadding="0" cellspacing="0" width="100%"
                     border="0">
                <tr>
                  <td width="85%">
                    <asp:Label ID="Label1" Runat="server"
                        Text="My First Company Site" Font-Bold="true"
                        Font-Size="24px">
                    </asp:Label>
                  </td>
                  <td width="15%">
                    <!-- Login Status Area -->
                    <asp:LoginStatus ID="LoginStatus1"
                                     Runat="server" />
                  </td>
                </tr>
                <tr>
                  <td colspan="2">
                    <hr color="black" size="2" />
                  </td>
                </tr>
                <tr>
                  <td colspan="2">
                    User:
                    <asp:LoginView ID="LoginView2" Runat="server">
                      <AnonymousTemplate>
                        Guest, Please log in
                      </AnonymousTemplate>
                      <LoggedInTemplate>
                        <asp:LoginName ID="LoginName1"
                                       Runat="server" />
                      </LoggedInTemplate>
                    </asp:LoginView>
                  </td>
                </tr>
                <tr>
                  <td colspan="2">
                    <hr color="black" size="2" />
                  </td>
                </tr>
                <tr>
                  <td colspan="2" valign="top" height="100%">
                    <asp:contentplaceholder id="cphMain"
                                            runat="server">
                    </asp:contentplaceholder>
                  </td>
```

```
                </tr>
              </table>
            </td>
          </tr>
        </table>
      </div>
    </form>
  </body>
</html>
```

۲) فایل `Default.aspx` در فولدر اصلی برنامه را باز کرده و تمام کد های درون آن را حذف کنید. سپس کد مشخص شده در زیر را در آن وارد کنید تا این فرم همانند شکل ۱۸-۱۱ شود:

```
<%@ Page Language="C#" MasterPageFile="~/Main.master" Title="Home" %>
```



شکل ۱۸-۱۱

۳) در فایل `Login.aspx` نیز تمام کد های موجود را حذف کنید و سپس کد مشخص شده در زیر را در آن وارد کنید:

```
<%@ Page Language="C#" MasterPageFile="~/Main.master" Title="Login" %>
<asp:content ContentPlaceHolderID="cphMain" Runat="server">
  <asp:Login ID="Login1" runat="server">
  </asp:Login>
</asp:content>
```

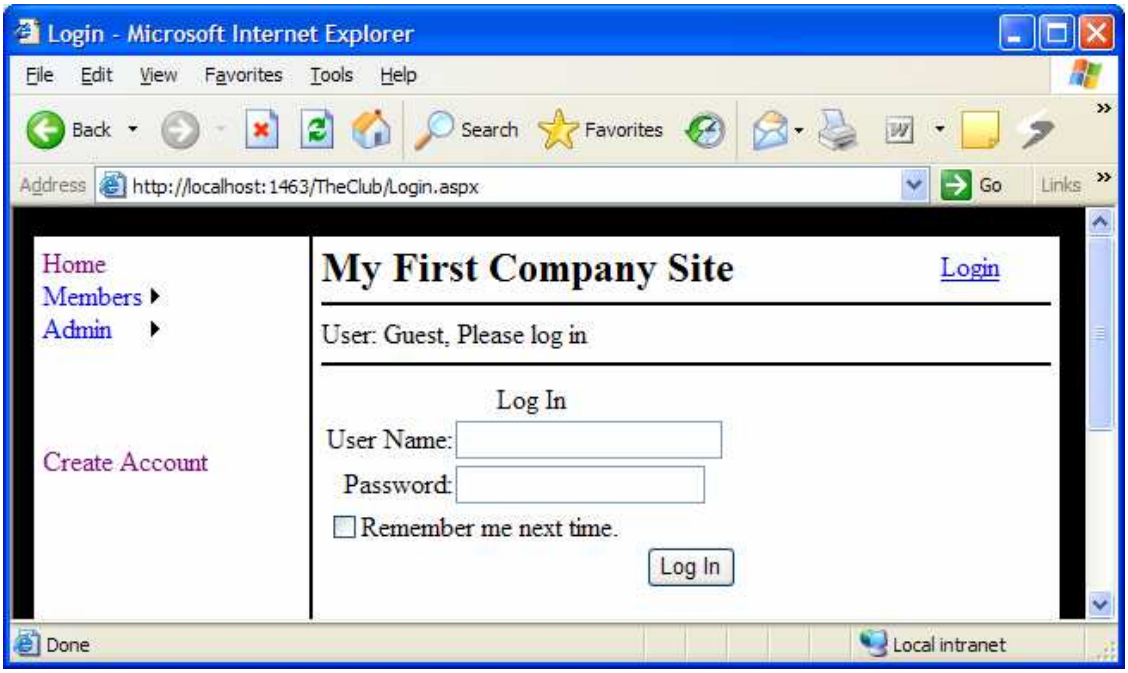
۴) کد موجود در فایل `ChangePassword.aspx` را نیز حذف کرده و آن را با کد زیر جایگزین کنید. به این ترتیب فرم `ChangePassword.aspx` مشابه شکل ۱۸-۱۳ خواهد بود:

```
<%@ Page Language="C#" MasterPageFile="~/Main.master"
```

```
        Title="Change Password" %>
<asp:Content ContentPlaceHolderID="cphMain" Runat="server">
  <asp:ChangePassword ID="ChangePassword1" Runat="server">
  </asp:ChangePassword>
</asp:Content>
```

۵) در فرم CerateNewUser.aspx نیز، کد داخل فرم را با کد زیر جایگزین کنید تا فرم، مشابه شکل ۱۸-۱۴ شود:

```
<%@ Page Language="C#" MasterPageFile="~/Main.master"
        Title="Create New Account" %>
<asp:Content ContentPlaceHolderID="cphMain" Runat="server">
  <asp:CreateUserWizard ID="CreateUserWizard1" Runat="server">
  </asp:CreateUserWizard>
</asp:Content>
```



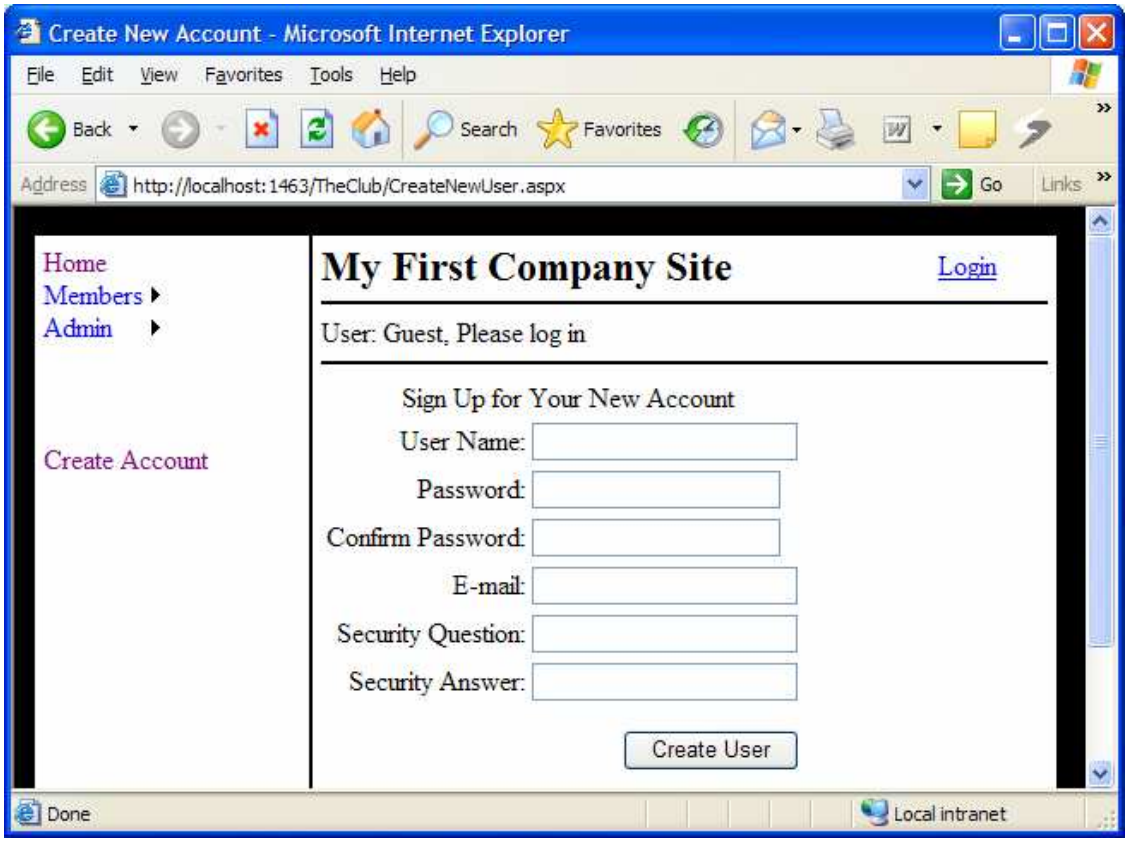
شکل ۱۸-۱۲

۶) کد موجود در فایل ViewAuthors.aspx را نیز با کد مشخص شده در زیر جایگزین کنید:

```
<%@ Page Language="C#" MasterPageFile="~/Main.master"
        Title="View Authors" %>
<asp:Content ID="Content1" ContentPlaceHolderID="cphMain"
             Runat="server">
  <asp:Label ID="Label1" Runat="server"
             Text="Add Code to View Author Info Later">
  </asp:Label>
</asp:Content>
```



شکل ۱۸-۱۳



شکل ۱۸-۱۴

۷) کد موجود در فایل ViewTitles.aspx را حذف کرده و کد زیر را به جای آن قرار دهید:

```
<%@ Page Language="C#" MasterPageFile="~/Main.master"
         Title="View Titles" %>
<asp:Content ID="Content1" ContentPlaceHolderID="cphMain"
             Runat="server">
  <asp:Label ID="Label1" Runat="server"
             Text="Add Code to View Title Info Later">
  </asp:Label>
</asp:Content>
```

۸) در فایل ViewUsers.aspx، کدی که به صورت پیش فرض وارد شده است را حذف کنید و به جای آن کد زیر را قرار دهید:

```
<%@ Page Language="C#" MasterPageFile="~/Main.master"
         Title="View Users" %>
<asp:Content ContentPlaceHolderID="cphMain" Runat="server">
  <asp:Label ID="Label1" Runat="server"
             Text="Add Code to View User Info Later">
  </asp:Label>
</asp:Content>
```

۹) حال برنامه را اجرا کنید تا عملکرد آن را تست کنیم. در این سایت می توانید یک اکانت جدید ایجاد کرده و سپس با استفاده از آن وارد سیستم شوید. تمام موارد مربوط به تعیین هویت افراد و نیز جلوگیری از دسترسی غیر مجاز،به وسیله ی همین کنترل هایی که در برنامه از آنها استفاده کردیم مدیریت می شود. البته در حال کار با این سایت ممکن است به صفحات خطایی نیز برخورد کنید که در مورد رفع آنها در این کتاب صحبتی نخواهیم کرد.

چگونه کار می کند؟

احتمالاً بعد از اتمام این سایت، با مشاهده ی عملکرد آن کاملاً شگفت زده می شوید. در تکنولوژی های قدیمی تر پیاده سازی چنین سیستمی، روزها طول می کشید.
در این سایت نیز تمام قسمتهای مربوط به طراحی ظاهر صفحات در MasterPage انجام شد، بنابراین بهتر است ابتدا کد های این قسمت را بررسی کنیم. قبل از هر چیز رنگ صفحات را به سیاه تغییر می دهیم:

```
<body bgcolor="black">
```

بعد از این کار چند جدول تودرتو در صفحه قرار می دهیم. در طراحی ظاهر سایتهای وب، جدول یکی از پر کاربرد ترین ابزار ها محسوب می شود. به وسیله ی جدولها می توانیم صفحات را به هر ترتیبی که بخواهیم به ستونها و ردیفهای مختلف تقسیم کرده و به این ترتیب سلولهایی را در صفحه به وجود آوریم. همچنین در داخل آن سلول ها نیز می توان جداول دیگری قرار داد. کار با جدولها در طراحی صفحات وب مانند یک هنر است که فقط با استفاده از تجربه می توان آن را بدست آورد. ابتدا، بهتر است نگاهی به تگ هایی بیندازیم که برای تنظیم این چند جدول از آن استفاده کرده ایم.

تگ	شرح
<table>	تگ اصلی برای ایجاد یک جدول جدید.
<tr>	این تگ در جدولی که در آن قرار گرفته است یک ردیف ایجاد می کند.
<td>	این تگ در ردیفی که در آن قرار گرفته است یک ستون ایجاد می کند.

هنگامی که ظاهر یک جدول را تنظیم می کنید، خصیصه ی colspan در تگ td مشخص می کند که ستون ایجاد شده در این ردیف می تواند تا حداکثر چند ستون از ردیفهای بالاتر و یا پایین تر را بپوشاند. برای مثال اگر در یک ردیف یک ستون قرار دهید و در ردیف دوم دو ستون، ستون دوم در ردیف دوم زیر هیچ ستونی از ردیف اول قرار نخواهد گرفت. اما اگر خصیصه ی colspan ستون ردیف اول را برابر با 2 قرار دهید، این ستون در بالای هر دو ستون موجود در ردیف دوم قرار خواهد گرفت.
در این برنامه، یک جدول با اندازه ی 600*400 ایجاد کرده و رنگ پس زمینه ی آن را نیز برابر با سفید قرار می دهیم. به این ترتیب اگر هیچ کنترلی نیز در این جدول قرار نگرفته باشند، اندازه ی این جدول ثابت می ماند. جدول اصلی در این فرم از یک ردیف و دو ستون تشکیل شده است، پس دو سلول دارد که سلول اول برای قرار دادن منو ها و سلول دوم نیز برای قرار دادن یک جدول دیگر در آن به کار می رود. در جدول ایجاد شده در سلول دوم پنج ردیف وجود دارد که هر یک از آنها یک و یا دو ستون دارند. اندازه ی این جدول برابر با 100% قرار داده شده است تا کل ناحیه ی مربوط به سلول دوم در جدول اول را پوشش دهد. جدول دوم برای قرار دادن عنوان سایت، نام و مشخصات کاربری که وارد برنامه شده است و نیز محتویات دیگر صفحه استفاده می شود.

```
<table cellpadding="5" cellspacing="0" width="600" height="400"
       bgcolor="white" border="1" bordercolor="black">
  <tr>
    <td width="150" valign="top">
    </td>
    <td valign="top">
      <table cellpadding="0" cellspacing="0" width="100%"
             border="0">
        <tr>
          <td width="85%">
            <asp:Label ID="Label1" Runat="server"
                Text="My First Company Site" Font-Bold="true"
                Font-Size="24px">
            </asp:Label>
          </td>
          <td width="15%">
            <!-- Login Status Area -->
            <asp:LoginStatus ID="LoginStatus1"
                             Runat="server" />
          </td>
        </tr>
        <tr>
          <td colspan="2">
            <hr color="black" size="2" />
          </td>
        </tr>
        <tr>
          <td colspan="2">
            User:
            <asp:LoginView ID="LoginView2" Runat="server">
              <AnonymousTemplate>
```

```
            Guest, Please log in
          </AnonymousTemplate>
          <LoggedInTemplate>
            <asp:LoginName ID="LoginName1"
                           Runat="server" />
          </LoggedInTemplate>
        </asp:LoginView>
      </td>
    </tr>
    <tr>
      <td colspan="2">
        <hr color="black" size="2" />
      </td>
    </tr>
    <tr>
      <td colspan="2" valign="top" height="100%">
        <asp:contentplaceholder id="cphMain"
                                runat="server">
        </asp:contentplaceholder>
      </td>
    </tr>
  </table>
</td>
</tr>
</table>
```

قسمت بعدی که باید در MasterPage قرار دهیم تا در همه ی صفحه ها قرار گیرد، منو ها است. مانند فصل هفدهم، در اینجا نیز از منو ها برای حرکت در بین صفحات سایت استفاده می کنیم. تنها تفاوتی که در این قسمت وجود دارد این است که در این جا چند کنترل منو ایجاد می کنیم و بر حسب اینکه کاربر تعیین هویت شده است یا نه یکی از آنها را نمایش می دهیم. کنترل LoginView این امکان را می دهد تا بر حسب نوع کاربری که در حال مشاهده ی صفحه است، نحوه ی نمایش آن را تغییر دهیم. برای مثال اگر کاربر فردی ناشناس بود، منویی با گزینه ی Create Account و اگر کاربر یکی از اعضای سایت بود، منویی با گزینه ی Change Password نمایش داده می شود.

```
<asp:Menu ID="Menu1" Runat="server">
  <Items>
    <asp:MenuItem NavigateUrl="~/Default.aspx" Text="Home">
    </asp:MenuItem>
    <asp:MenuItem Text="Members">
      <asp:MenuItem NavigateUrl="Members/ViewAuthors.aspx"
                    Text="View Authors">
      </asp:MenuItem>
      <asp:MenuItem NavigateUrl="Members/ViewTitles.aspx"
                    Text="View Titles">
      </asp:MenuItem>
    </asp:MenuItem>
    <asp:MenuItem Text="Admin">
      <asp:MenuItem NavigateUrl="Admin/ViewUsers.aspx"
                    Text="View Users">
      </asp:MenuItem>
    </asp:MenuItem>
  </Items>
</asp:Menu>
<br />
```

```
  <br />
  <br />
  <asp:LoginView ID="LoginView1" Runat="server">
    <AnonymousTemplate>
      <asp:Menu ID="Menu2" Runat="server">
        <Items>
          <asp:MenuItem NavigateUrl="~/CreateNewUser.aspx"
                        Text="Create Account">
          </asp:MenuItem>
        </Items>
      </asp:Menu>
    </AnonymousTemplate>
    <LoggedInTemplate>
      <asp:Menu ID="Menu3" Runat="server">
        <Items>
          <asp:MenuItem NavigateUrl="~/ChangePassword.aspx"
                        Text="Change Password">
          </asp:MenuItem>
        </Items>
      </asp:Menu>
    </LoggedInTemplate>
  </asp:LoginView>
```

برای عنوان فرم، از یک کنترل `Label` استفاده می کنیم:

```
  <asp:Label ID="Label1" Runat="server" Text="My First Company Site"
             Font-Bold="true" Font-Size="24px">
```

در زیر عنوان فرم، قسمتی را قرار می دهیم تا بتوانیم نام کاربری فردی که وارد سیستم شده است را نمایش دهیم. در این قسمت نیز از یک کنترل `LoginView` استفاده می کنیم تا اگر کاربری در سایت وارد شده بود نام کاربری او را نمایش دهد، و اگر فـردی ناشناس وارد سیستم شده بود `"Guest, Please Log in"` را نمایش دهد.

```
                    <!-- Login Status Area -->
<asp:LoginStatus ID="LoginStatus1" Runat="server" />
  User:
<asp:LoginView ID="LoginView2" Runat="server">
  <AnonymousTemplate>
    Guest, Please log in
  </AnonymousTemplate>
  <LoggedInTemplate>
    <asp:LoginName ID="LoginName1" Runat="server" />
  </LoggedInTemplate>
</asp:LoginView>
```

در آخر نیز یک کنترل `ContentPlaceHolder` در سایت قرار می دهیم. این کنترل مکانی را مـشخص مـی کنـد تـا صفحاتی که از این فایل به عنوان `MasterPage` خود استفاده می کنند، داده های خود را در آن قرار دهند.

```
<asp:contentplaceholder id="cphMain" runat="server">
```

در فرم Login، با استفاده از راهنمای Page عنوان صفحه را تغییر می دهیم و همچنین یک کنترل Content ایجاد کرده و آن را به کنترل ContentPlaceHolder در فایل MasterPage متصل می کنیم. به این ترتیب مکانی که به وسیله ی کنترل ContentPlaceHolder در MasterPage مشخص شده است، توسط این کنترل نیز تعیین می شود و می توانیم محتویات صفحه ی Login را در این قسمت قرار دهیم.
برای تکمیل صفحه ی Login کافی است که درون این قسمت، فقط یک کنترل Login قرار دهیم. این کنترل خود تمام کارهای لازم برای کنترل ورود کاربر را انجام می دهد:

```
<%@ Page Language="C#" MasterPageFile="~/Main.master" Title="Login" %>
<asp:content ContentPlaceHolderID="cphMain" Runat="server">
  <asp:Login ID="Login1" runat="server">
  </asp:Login>
</asp:content>
```

برای صفحه ی ChangePassword.aspx نیز با استفاده از راهنمای Page عنوان صفحه را تغییر می دهیم. سپس یک کنترل از نوع Content در صفحه ایجاد کرده و آن را به کنترل ContentPlaceHolder در فایل MasterPage متصل می کنیم.
درون این کنترل Content نیز فقط یک کنترل از نوع ChangePassword قرار می دهیم. این کنترل نیز تمام کارهای لازم برای تغییر کلمه ی عبور یک کاربر را انجام می دهد:

```
<%@ Page Language="C#" MasterPageFile="~/Main.master"
         Title="Change Password" %>
<asp:Content ContentPlaceHolderID="cphMain" Runat="server">
  <asp:ChangePassword ID="ChangePassword1" Runat="server">
  </asp:ChangePassword>
</asp:Content>
```

همین مراحل را در صفحه ی CreateNewUser.aspx نیز تکرار می کنیم تا یک کنترل Content در این صفحه نیز ایجاد شود. سپس درون این کنترل، یک کنترل از نوع CreateUserWizard قرار می دهیم. کنترل CreateUserWizard تمام کارهای لازم برای ایجاد یک اکانت جدید را انجام می دهد:

```
<%@ Page Language="C#" MasterPageFile="~/Main.master"
         Title="Create New Account" %>
<asp:Content ContentPlaceHolderID="cphMain" Runat="server">
  <asp:CreateUserWizard ID="CreateUserWizard1" Runat="server">
  </asp:CreateUserWizard>
</asp:Content>
```

در سه صفحه ی درونی برنامه که فقط اعضای سایت به آن دسترسی دارند، نیازی نیست که کنترل خاصی را قرار دهیم. فقط کافی است متنی را نمایش دهیم تا مشخص کند کاربر در حال مشاهده ی چه صفحه ای است. بنابراین، در این صفحات نیز همانند صفحات قبلی، بعد از تغییر عنوان صفحه با استفاده از خصیصه ی title در راهنمای Page یک کنترل Content در فرم ایجاد کرده و آن را به کنترل ContentPlaceHolder در فایل MasterPage متصل می کنیم. سپس یک کنترل Label در این کنترل Content قرار می دهیم و متنی را در آ نمایش می دهیم تا مشخص کند کاربر در حال مشاهده ی چه صفحه ای است.

```
<%@ Page Language="C#" MasterPageFile="~/Main.master"
         Title="View Authors" %>
<asp:Content ID="Content1" ContentPlaceHolderID="cphMain"
             Runat="server">
  <asp:Label ID="Label1" Runat="server"
             Text="Add Code to View Author Info Later">
  </asp:Label>
</asp:Content>

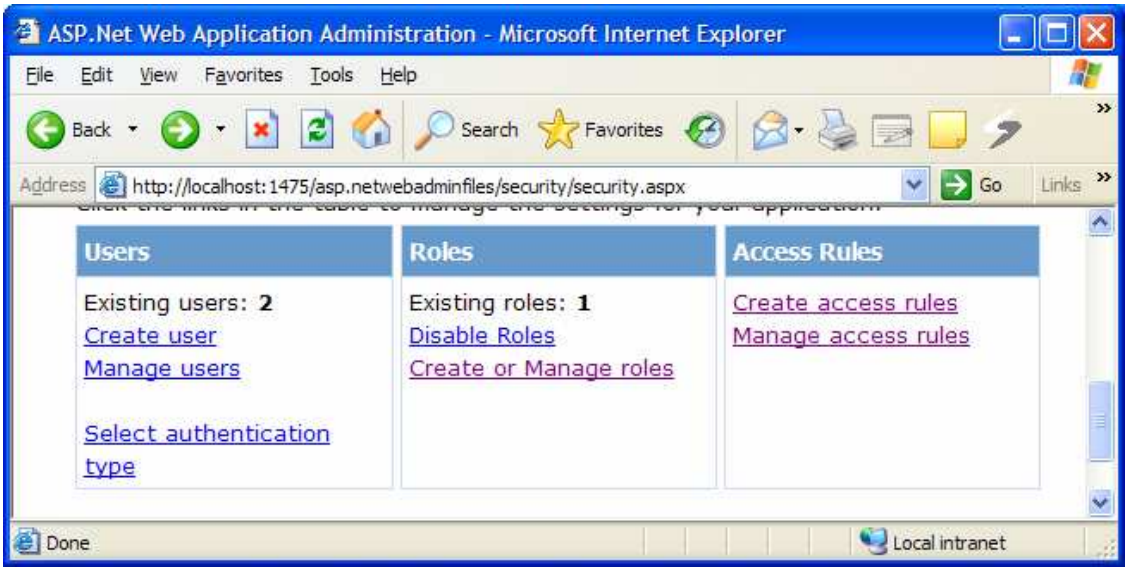
<%@ Page Language="C#" MasterPageFile="~/Main.master"
         Title="View Titles" %>
<asp:Content ID="Content1" ContentPlaceHolderID="cphMain"
             Runat="server">
  <asp:Label ID="Label1" Runat="server"
             Text="Add Code to View Title Info Later">
  </asp:Label>
</asp:Content>

<%@ Page Language="C#" MasterPageFile="~/Main.master"
         Title="View Users" %>
<asp:Content ContentPlaceHolderID="cphMain" Runat="server">
  <asp:Label ID="Label1" Runat="server"
             Text="Add Code to View User Info Later">
  </asp:Label>
</asp:Content>
```

البته احتمالاً متوجه شده اید که هنوز نمی توانید به صفحه ی ViewUsers.aspx دسترسی داشته باشید و در صورتی که سعی کنید وارد این صفحه شوید، مجدداً به فرم Login.aspx هدایت خواهید شد. دلیل این مورد نیز در این است که، در قسمت قبلی تمام صفحات موجود در فولدر Admin را به گونه ای تنظیم کردیم که فقط برای گروه Admin قابل مشاهده باشد. اکانت هایی که به صورت عادی ایجاد می شوند عضوی از گروه Admin نیستند و باید آنها را به این گروه اضافه کرد تا بتوانند وارد صفحه ی ViewUsers.aspx شوند. در قسمت امتحان کنید بعد، مشاهده خواهید کرد که چگونه می توان یکی از اکانت های ایجاد شده را به گروه Admin اضافه کرد.

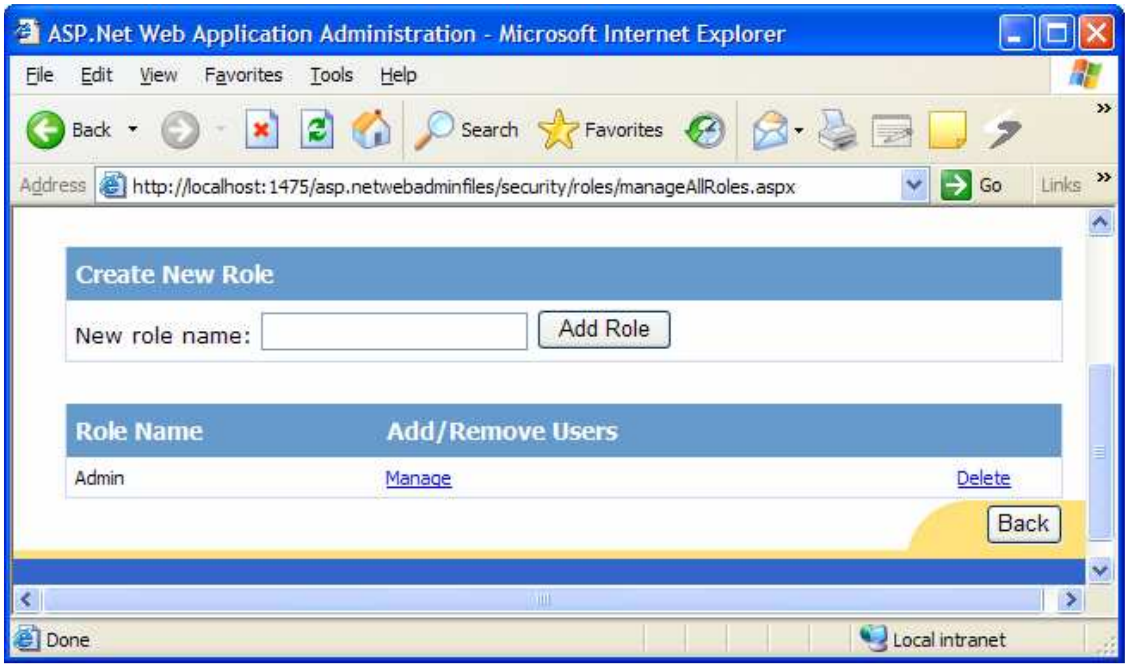
امتحان کنید: مدیریت گروه ها

۱) برای مدیریت گروه ها و اکانت هایی که در هر یک از آنها قرار دارد باید از ابزار WAT استفاده کنیم. بنابراین با انتخاب گزینه ی Website ➔ ASP.NET Configuration از نوار منوی ویژوال استودیو، صفحه ی مربوط به این ابزار را باز کنید. در صفحه ی اولیه ی این ابزار، روی قسمت Security کلیک کنید. در نیمه ی پایین این صفحه جدولی را با عنوان Roles مشاهده خواهید کرد. در این جدول روی گزینه ی Create or Manage Roles را انتخاب کنید (شکل ۱۸-۱۵)



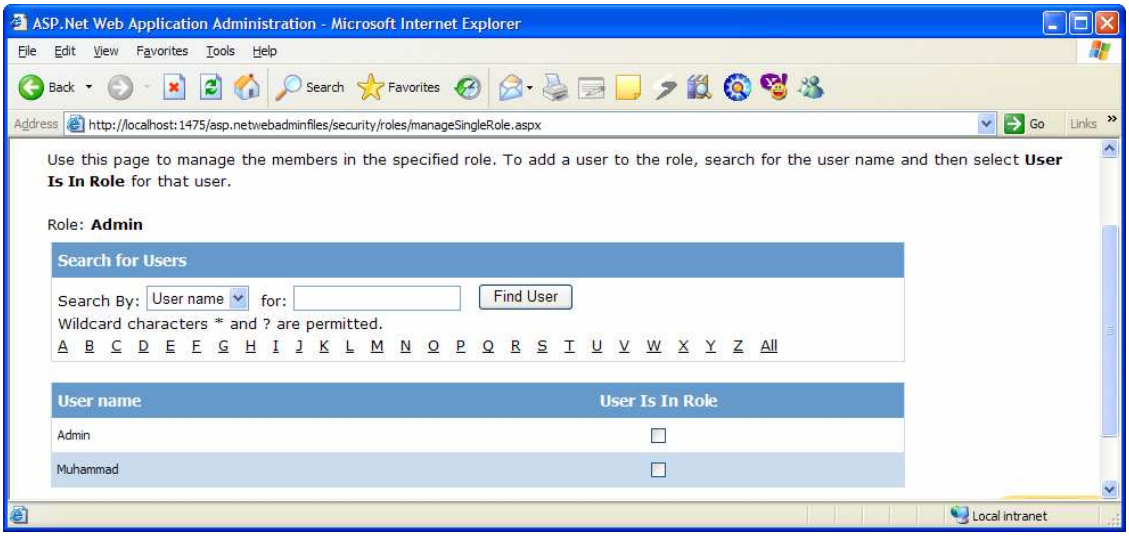
شکل ۱۸-۱۵

۲) در صفحه ی بعد، همانطور که در شکل ۱۸-۱۶ مشاهده می کنید لیستی از گروه های موجود در این برنامه نمایش داده می شود و نیز می توانید گروه های جدیدی ایجاد کنید. روی لینک `Manage` مقابل گروه `Admin` کلیک کنید.



شکل ۱۸-۱۶

۳) روی لینک `All` کلیک کنید تا نام تمام کاربران موجود در سیستم نمایش داده شود. در مقابل نام هر کاربر یک `CheckBox` وجود دارد که مشخص می کند آیا این کاربر عضوی از این گروه هست یا نه؟ با انتخاب این گزینه کاربر عضوی از این گروه خواهد بود و با خارج کردن این گزینه از حالت انتخاب، کاربر از این گروه خارج خواهد شد.



شکل ۱۸-۱۷

چگونه کار می کند؟

با تغییراتی که در مرحله ی قبل در فولدر `Admin` ایجاد کردیم، فقط افرادی که عضو گروه `Admin` بودند می توانستد به صفحات این فولدر دسترسی داشته باشند. کاربرانی نیز که به صورت عادی ایجاد می شوند عضو این گروه نخواهند بود، بنابراین باید با استفاده از این ابزار، کاربران مورد نظر را در این گروه قرار داد و یا از این گروه خارج کرد. هنگامی که در این قسمت نام کاربری خود را به عنوان عضوی از گروه `Admin` مشخص کردید، می توانید به محتویات فولدر `Admin` دسترسی داشته باشید.

نتیجه:

در این فصل سعی کردیم با ساختار وب سایتهای امن و نیز کاربردی آشنا شویم. در برنامه های این فصل نیز با استفاده از مطالبی که در فصل هفدهم آموخته بودیم و نیز تعدادی از کنترلهای درونی ASP.NET 2 توانستیم بدون نوشتن حتی یک خط کد یک سایت وب کاربردی ایجاد کنیم. این کنترل ها حتی نیاز به تنظیم خصوصیت خاصی نیز نداشتند و مقادیر پیش فرض آنها به خوبی عمل می کرد.

در ابتدای فصل با انواع مختلف تعیین هویت کاربران آشنا شدیم و بررسی کردیم که هر یک از آنها در چه شرایطی کاربرد دارند. سپس با ابزار WAT و نحوه ی استفاده از آن آشنا شدیم و مشاهده کردیم چگونه می توان با استفاده از آن امنیت قسمتهای مختلف یک سایت را، چه با استفاده از روش تعیین هویت به وسیله ی ویندوز و چه با استفاده از روش تعیین هویت به وسیله ی فرم های وب تامین کرد.

در پایان این فصل باید با موارد زیر آشنا شده باشید:

- از ابزار مدیریت وب سایت (WAT) استفاده کنید.
- قالب و استیل صفحات وب مربوط به سایت را با استفاده از `MasterPage` ها ایجاد کنید.

- با استفاده از روش تعیین هویت به وسیله ی فرمهای ویندوزی، امنیت قسمتهای مختلف یک سایت را تامین کنید.
- با تعریف گروه های مختلف در سایت، مشخص کنید که اعضای هر گروه به چه قسمتهایی از سایت دسترسی دارند.
- از کنترلهای درونی ASP.NET 2 برای Login استفاده کنید.

تمرین:

تمرین ۱:

با استفاده از فایل Main.skin، رنگ فونت تمام کنترلهای Label ای که در صفحات فولدر Members قرار دارند را برابر با قرمز قرار دهید. برای این کار می توانید خاصیت Theme در تمام صفحات را تغییر دهید و یا از فایل web.config که در فولدر Members قرار دارد استفاده کنید. برای این تمرین بهتر است که از فایل Web.config استفاده کنید. به عنوان راهنمایی می توانید بعد از باز کردن فایل، داخل آن را به صورتی که در کد XML زیر مشخص شده است تغییر دهید:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration
xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <system.web>
    <pages theme="MainTheme"/>
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</configuration>
```

به این ترتیب، ظاهر این صفحات باید مشابه شکل ۱۸-۱۸ شود:



شکل ۱۸-۱۸

فصل نوزدهم: XML و ویژوال C# ۲۰۰۵

اگر بخواهیم به زبان ساده تعریفی از XML عنوان کنیم، می توانیم بگوییم که XML یا Extensible Markup Languageبرای انتقال اطلاعات بین برنامه ها مورد استفاده قرار می گیرد. اگر چه ممکن است در آینده این اتفاق رخ دهد، اما فعلا XML به صورت غیر رسمی به عنوان استانداردی برای انتقال اطلاعات بین برنامه های تحت وب مورد استفاده قرار می گیرد. امروزه نه تنها XML برای انتقال اطلاعات بین برنامه های تحت وب مورد استفاده قرار می گیرد، بلکه برای انتقال داده ها بین پلت فرم ها و سیستم عامل های مختلف نیز از آن استفاده می شود.
در این فصل نمی خواهیم وارد جزئیات و مباحث پیچیده ی زبان XML شویم و فقط به معرفی کلی آن و بیان رابطه ی این زبان با ویژوال C# ۲۰۰۵ اکتفا می کنیم. بعد از آن نیز سعی می کنیم نحوه ی استفاده از XML در برنامه ها را بررسی کنیم.
در این فصل:

- با مفهوم XML آشنا می شویم.
- مشاهده خواهیم کرد که چگونه می توان فایلهای XML را خواند و یا نوشت.
- نحوه ی سریالایز کردن و دی سریالایز کردن فایلهای XML را بررسی خواهیم کرد.
- با چگونگی حرکت در بین یک فایل XML آشنا خواهیم شد.
- با نحوه ی تغییر داده های XML موجود و یا اضافه کردن داده ای به یک فایل XML آشنا خواهیم شد.

درک XML:

دلیل اینکه چرا به XML نیاز داریم بسیار ساده است. در محیط های تجاری، برای اینکه برنامه ها بتوانند با یکدیگر رابطه برقرار کنند لازم است که بتوانند داده ها را با یکدیگر مبادله کنند. این برقراری ارتباط بین نرم افزارها و یکپارچگی بین آنها، بیشتر در مورد برنامه های تجاری آن شرکت مورد نظر است نه نرم افزارهای معمولی که در آنجا استفاده می شود مانند آفیس و یا .... برای مثال یک شرکت ممکن است نرم افزاری داشته باشد که تعداد کالاهایی که در انبار وجود دارند را کنترل می کند. این نرم افزار یک نمونه از نرم افزارهای تجاری است که در آن شرکت استفاده می شود.
ایجاد یکپارچگی بین نرم افزارهای تجاری شرکتهای مختلف امر بسیار مشکلی است، و XML به همراه تکنولوژی دیگری به نام سرویسهای وب (که در فصل بیستم در رابطه با آنها صحبت خواهیم کرد) به این منظور طراحی شده اند که هزینه و نیز مشکلات برقراری یکپارچگی بین این نرم افزارها را رفع کنند. برای کاهش هزینه و مشکلات این کار نیز، اول باید کاری کرد که داده ها و اطلاعات بتوانند به سادگی بین این نرم افزارها منتقل شوند.
برای مثال تصور کنید که شما یک فروشنده ی جزئی قهوه هستید و می خواهید اجناس مورد نیاز خود را به یک فروشنده ی کلی سفارش دهید. یک روش قدیمی برای انجام این کار این است که از تلفن و یا فکس برای سفارش خود استفاده کنید. اما در این روش، حتماً باید انسان دخالت داشته باشد. ممکن است نرم افزاری که استفاده می کنید بتواند زمان تمام شدن یک محصول را اعلام کند و به شما پیشنهاد بدهد که آن محصول را سفارش دهید. حتی ممکن است یک فرم سفارش برای شما آماده کند و فقط شما آن را به فروشنده ی کلی محصولات ارسال کنید. در این حالت، در دفتر فروشنده ی کلی نیز باید فردی قرار داشته باشد تا بتواند درخواست شما را دریافت کرده و آن را به نرم افزار مورد استفاده ی خودشان بدهد تا کالای مورد نظر برای شما ارسال شود.
یک راه دیگر برای انجام این کار به این صورت ست که زمانی که برنامه ی تجاری شما کاهش محصولی را در انبار تشخیص داد، درخواستی را آماده کرده و آن را به صورت اتوماتیک به نرم افزار فروشنده ی کلی ارسال کند. به این ترتیب کار هم برای شما و هم برای فروشنده ساده تر و کارآمد تر خواهد شد. اما برای انجام این کار نیز لازم است که شما با شرکت فروشنده ی کلی مذاکرات و

هماهنگی های لازم را انجام دهد و نیز هزینه هایی را در این میان متحمل شوید. بنابراین این مورد فقط برای شرکتهایی مناسب است که تعامل زیادی با یکدیگر دارند.

البته این مورد نیز قبل از اینترنت صادق بود. در آن زمان برای اینکه دو شرکت بتوانند از این روش با یکدیگر تعامل داشته باشند، لازم بود که مذاکرات زیادی را انجام می دادند و بعد از مشخص کردن تمام شرایط یک اتصال اختصاصی بین این دو شرکت برقرار می کردند. بعد از اینکه این اتصال بین این دو شرکت برقرار می شد، نه تنها شرکت خریدار می توانست داده های مربوط به سفارشات خود را به شرکت فروشنده بفرستد، بلکه شرکت فروشنده نیز می توانست که گزارشات مربوط به وضعیت سفارش را به خریدار ارسال کند. امروزه با استفاده از اینترنت مشکل برقراری اتصال اختصاصی بین دو شرکت رفع شده است، زیرا اگر هر دوی آنها به اینترنت متصل باشند می توانند داده های لازم را به یکدیگر منتقل کنند.

البته این فقط نیمی از مشکل بود. تا زمانی که یک زبان عمومی برای انتقال داده ها بین نرم افزارهای مختلف نباشد، هنوز نیمه ی دیگر مشکل حل نشده باقی می ماند. این زبان عمومی XML است. به این ترتیب شرکت خریدار، یک سند XML ایجاد کرده و اطلاعات مربوط به سفارش خود را در آن قرار می دهد. سپس این شرکت می تواند سند XML خود را به وسیله ی ایمیل و یا به وسیله ی یک سرویس وب از طریق اینترنت به نرم افزاری که در شرکت فروشنده قرار دارد بدهد. سپس شرکت خریدار این فایل XML را دریافت کرده، داده های آن را استخراج می کند و کالاهای سفارش داده شده را ارسال می کند. همچنین اگر لازم باشد که شرکت فروشنده، گزارشی را به شرکت خریدار ارسال کند یک سند XML دیگر ایجاد کرده، داده های لازم را در آن قرار می دهد و مجدداً با استفاده از اینترنت آن را به نرم افزار موجود در شرکت خریدار ارسال می کند.

ساختار و الگویی که باید برای ایجاد فایلهای XML، بین خریدار و فروشنده مورد استفاده قرار گیرد به خود آنها بستگی دارد. به عبارت دیگر این شرکت خود می توانند تصمیم بگیرند که داده ها را به چه صورت در یک فایل XML قرار دهند تا نرم افزارهای هر دو شرکت آن را درک کند[1]. به همین دلیل است که زبان XML به عنوان یک زبان توسعه پذیر (eXtensible) شناخته شده است. هر دو شرکتی که بخواهند داده هایی را با استفاده از XML بین خود مبادله کنند، در مورد الگوی قرار دادن داده ها در فایل XML و اینکه می خواهند فایل XML آنها به چه صورتی باشد کاملاً مختار هستند.

البته این مورد زیاد شگفت انگیز نیست، شرکتهای بسیاری در گذشته و حتی امروز نیز برای انتقال اطلاعات بین خود از فایلهای متنی استفاده می کنند. نحوه ی قرار گیری اطلاعات در این فایلهای متنی نیز به خود این شرکتها بستگی دارد.اما ممکن است در اینجا این سوال مطرح شود که پس XML چه خاصیتی دارد که فایلهای متنی قبلی ندارند؟

خوب XML بسیار تشریحی تر است و به راحتی می توان تشخیص داد که آیا یک فایل XML بر اساس یک الگوی خاص ایجاد شده است یا نه؟ یک **الگوی XML**[2] مشخص می کند که ظاهر یک سند XML باید چگونه باشد. البته بودن یک الگو نیز، یک سند XML می تواند به اندازه ی کافی داده های درون خود را توصیف کند تا بتوان تشخیص داد که چه اطلاعاتی در آن قرار دارند. همچنین فایلهای XML، مانند فایلهایی که در گذشته استفاده می شد از فرمت متنی استفاده می کند. بنابراین می توان به سادگی برای انتقال فایلهای XML بین پلت فرم های گوناگون، از تکنولوژی های اینترنتی مانند ایمیل، وب، FTP و یا ... استفاده کرد. ابزارهایی که در گذشته برای ایجاد یکپارچگی بین نرم افزارهای تجاری مختلف استفاده می کردند، نمی توانست به سادگی داده های باینری را بین پلت فرم های گوناگون مانند لینوکس، ویندوز، مکینتاش، AS/400، OS/390 و یا بسیاری از پلت فرم های دیگر منتقل کند. بنابراین این ویژگی که اطلاعات و داده های ذخیره شده در XML در قالب متن هستند، انتقال اطلاعات بین پلت فرم ای گوناگون را بسیار ساده تر می کند.

XML شبیه به چیست؟

[1] البته شرکت فروشنده الگو و مدل فایل را مشخص کرده و شرکت خریدار نیز از آن پیروی می کند.

[2] XML Schema

اگر قبلا تجربه ی استفاده از HTML را داشته باشید، احتمالا زبان XML نیز برای تان بسیار آشنا خواهد بود. در حقیقت این دو زبان هر دو از یک زبان به نام SGML[1] به وجود آمده اند. از بسیار ی جهات XML یک زبان به شمار نمی رود، بلکه شامل یک مجموعه از قواعد است که نحوه ی ایجاد یک فایل خاص برای انتقال اطلاعات و داده ها را مشخص می کند. همچنین XML یک تکنولوژی مستقل نیست، بلکه بسیاری از تکنولوژی های دیگر نیز هستند که مشخص می کنند شما چگونه می توانید از XML استفاده کنید. بعضی از آنها در لیست زیر نام برده شده اند (البته اگر کتاب را به صورت گذرا می خوانید این نامها اهمیتی ندارند، ولی اگر می خواهید با دقت از همه چیز مطلع شوید ممکن است مفید به نظر برسند):

- URI (Uniform Resource Identifiers): www.ietf.org/rfc/rfc2396.txt
- UTF-8 (Uniform Transformation Format): www.utf-8.com
- XML (eXtensible Markup Language): www.w3.org/TR/REC-xml
- XML Schema: www.w3.org/XML/Schema
- XML Information Set: www.w3.org/TR/xml-infoset

البته درباره ی مواردی که در بالا عنوان شد، کتابهای زیادی که بتوان به سادگی آنها را درک کرد، با قالب و فرمت آنها آشنا شد و نحوه ی استفاده از آنها را مشاهده کرد وجود ندارد. اما اگر تمایل دارید که با این موارد بیشتر آشنا شوید، می توانید بعد از اتمام این فصل از منابع اینترنتی موجود استفاده کنید.
زبان XML نیز مبتنی بر تگ ها است، به این معنی که اسناد XML نیز شامل تگ هایی هستند که داده ها در آن قرار می گیرد. برای مثال اطلاعات مربوط به این کتاب را می توان به صورت زیر در قالب XML نمایش داد:

```
<Book>
  <Title>Learning Visual C# 2005</Title>
  <ISBN>XXXX-XX-XXX</ISBN>
  <Subject>Programming</Subject>
</Book>
```

در XML نیز مانند HTML برای ایجاد یک تگ از علامت های > و < استفاده می کنند. دو نوع تگ کلی نیز وجود دارند که عبارتند از تگ های شروع مانند <Title> و تگهای پایان مانند </Title>. تگ ها و محتویات آنها با نام **عنصر**[2] شناخته می شوند. بنابراین در مثال قبل، عنصر Title به صورت زیر نوشته می شود:

```
  <Title>Learning Visual C# 2005</Title>
```

عنصر ISBN نیز مشابه زیر است:

```
  <ISBN>XXXX-XX-XXX</ISBN>
```

[1] Standard Generalized Markup Language
[2] Element

و عنصر آخر نیز عنصر Subject است:

```
  <Subject>Programming</Subject>
```

دقت کنید که یک عنصر خود نیز می تواند شامل چند عنصر باشد. برای نمونه در مثال قبل عنصر Book شامل سه عنصر به صورت زیر است:

```
<Book>
  <Title>Learning Visual C# 2005</Title>
  <ISBN>XXXX-XX-XXX</ISBN>
  <Subject>Programming</Subject>
</Book>
```

نکته: معمولا عناصری که درون یک عنصر قرار می گیرد را به صورت نمای درختی نیز نمایش می دهند. برای مثال در کد XML قبلی، عناصر Title، ISBN و Subject، شاخه هایی از عنصر Book هستند که به عنوان ریشه در نظر گرفته می شود. همچنین در این زبان از اصطلاحات زیادی مانند گره، عنصر فرزند، عنصر والد و یا ... ممکن است استفاده شود که همه از این ساختار نشات می گیرند.

اگر بخواهید از این سند XML استفاده کنید، ابتدا باید ساختار داده هایی که در آن قرار گرفته اند را درک کنید. معمولا شرکتی که ایل XML را ایجاد می کند، در مورد ساختار داده های درون آن نیز به شما توضیح خواهد داد. برای نمونه در مثال قبل، شرکتی که داده های بالا را تنظیم کرده است به شما می گوید که برای مشاهده ی عنوان یک کتاب ابتدا وارد عنصر Book شده و از آنجا به عنصر Title می روید. به این ترتیب مقداری که درون تگ شروع <title> و نیز تگ پایان </title> وجود دارد، برابر با عنوان کتاب است.

همانند زبان HTML، تگهای XML نیز دارای خصیصه هستند. یک **خصیصه**[1] داده ای است که گره ای (عنصری) که در آن قرار گرفته است را توصیف می کند. هنگامی که از خصیصه ها در یک تگ استفاده می کنید، باید مقدار آنها را درون علامت "" در مقابل آنها قرار دهید. کد XML زیر، مشابه کد قبلی است. اما در این قسمت اطلاعات مربوط به ISBN کتاب را به صورت یک خصیصه از تگ Title در فایل قرار داده ایم:

```
<Book>
  <Title ISBN="XXXX-XX-XXX">Learning Visual C# 2005</Title>
  <Subject>Programming</Subject>
</Book>
```

یکی از خصوصیت هایی که باعث گسترش فراوان XML شده است، این است که داده های درون آن به سادگی می توانند توسط همه ی افراد درک شوند. برای مثال، فکر کنم به سادگی بتوانید داده هایی که در این کد XML قرار گرفته اند را درک کنید:

```
<Books>
  <Book>
    <Title>Learning Visual C# 2005</Title>
    <ISBN>XXXX-XX-XXX</ISBN>
    <Subject>Programming</Subject>
  </Book>
  <Book>
```

[1] Attribute

```
    <Title>Learning Visual Basic 2005</Title>
    <ISBN>XXXX-XX-XXX</ISBN>
    <Subject>Programming</Subject>
  </Book>
</Books>
```

XML برای افراد مبتدی:

به عنوان فردی که در برنامه نویسی و نیز ویژوال C# مبتدی هستید، احتمالا نمی توانید برنامه هایی که به صورت پیچیده درگیر موارد پیچیده ی یکپارچه سازی می شوند را درک کنید. همانطور که در قسمت قبلی نیز گفتم، یکی از دلایلی که XML بسیار فراگیر و گسترده شده است سادگی آن در برقراری یکپارچگی بین نرم افزارهای تجاری است. اما این مسئله به برنامه نویسان مبتدی چه ارتباطی دارد؟ به عبارت دیگر برنامه نویسان مبتدی چه استفاده ای می توانند از XML داشته باشند؟
جواب این سوال این است که XML نه تنها یک وسیله برای یکپارچه سازی بین نرم افزارهای تجاری به شمار می رود، بلکه یک ابزار عالی برای ذخیره سازی اطلاعات و نیز سازماندهی داده ها است. قبل از XML، برنامه ها می توانستند از دو روش برای ذخیره ی داده های مورد نیاز خود استفاده کنند: یا از یک بانک اطلاعاتی استفاده کنند و یا داده های خود را در یک فایل با فرمت خاص خود قرار دهند و سپس کدی را بنویسند که بتواند داده ها را از آن فایلها خوانده و نیز در آنها بنویسد.
در بسیار ی از موارد استفاده از یک بانک اطلاعاتی بهترین گزینه برای انجام این کار به شمار می رود، زیرا با استفاده از آن برنامه های می توانند به سرعت به داده ها دسترسی داشته باشند و نیز از امکاناتی که آن موتور بانک اطلاعاتی (مانند Access و یا Sql Server) ارائه می دهد استفاده کنند. در بعضی شرایط نیز، مانند نوشتن یک برنامه ی گرافیکی، بهتر است که داده ها در یک فایل با فرمت خاص آن برنامه ذخیره شوند. دلیل این مورد نیز این بود که این برنامه ها اغلب تمایل داشتند که ساده باشند و نمی خواستند با استفاده از یک بانک اطلاعاتی، کاربران خود را درگیر مسائل مربوط به آن بکنند.
در این گونه موارد، XML روش جدیدی را برای ذخیره ی اطلاعات برنامه ارائه می دهد که البته قالب آن نیز می تواند مخصوص آن برنامه باشد. تفاوتی که این روش با روش ذخیره کردن داده ها در یک فایل متنی مانند .doc دارد در این است که داده هایی که در XML ذخیره می شوند به صورت استاندارد هستند.

پروژه ی دفتر تلفن:

در این قسمت برای اینکه با مباحث تئوری که در قسمت قبل مرور کردیم در عمل نیز آشنا شویم، یک برنامه ی دفتر تلف ایجاد خواهیم کرد و برای ذخیره ی اطلاعات آن از یک فایل XML استفاده می کنیم. به این ترتیب می توانیم لیستی از رکورد های اطلاعاتی که در این برنامه وارد می شود را در یک فایل XML ذخیره کرده و یا اطلاعات داخل یک فایل XML را خوانده و در برنامه نمایش دهیم.

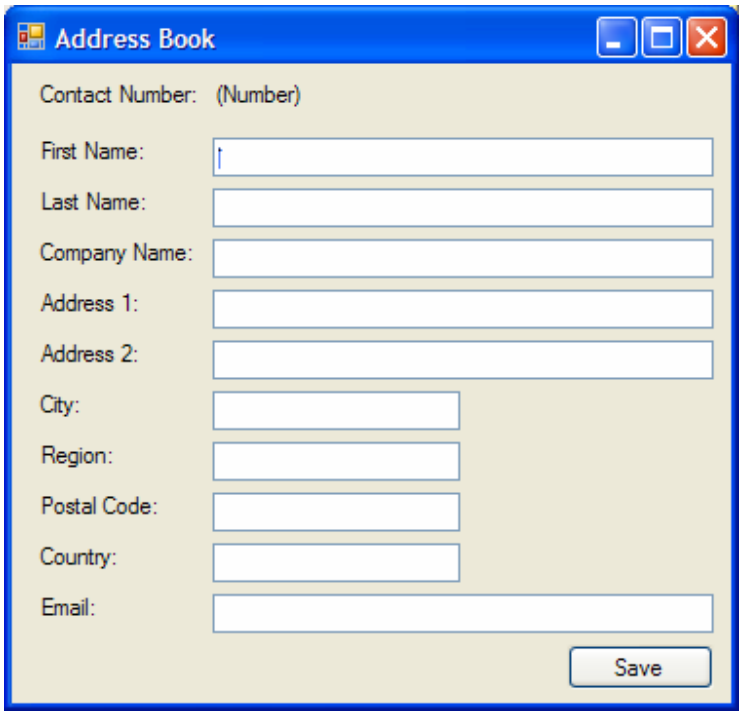
ایجاد پروژه:

مثل همیشه، اولین کاری که باید انجام دهیم این است که یک پروژه ی جدید ایجاد کنیم.

امتحان کنید: ایجاد پروژه

۱) ویژوال استودیو را باز کرده و گزینه ی `File ➔ New Project` را از نوار منو انتخاب کنید. سپس با استفاده از کادر `New Project` یک برنامه ی تحت ویندوز جدید به نام `Address Book` ایجاد کنید.

۲) به این ترتیب قسمت طراحی فرم مربوط به `Form1` نمایش داده خواهد شد. ابتدا خاصیت `Text` فرم را برابر با `Address Book` قرار دهید. حال با استفاده از جعبه ابزار، ده کنترل `TextBox`، دوازده کنترل `Label` و یک کنترل `Button` را به فرم برنامه اضافه کرده و آنها را همانند شکل ۱۹-۱ در فرم برنامه مرتب کنید. برای تنظیم مکان کنترل ها هم می توانید از خطوط مربوط به این کار در فرم استفاده کنید و هم می توانید گزینه های موجود در منوی `Format` را به کار ببرید.



شکل ۱۹-۱

۳) خاصیت `Name` کنترلهای `TextBox` موجود در فرم را به صورت زیر تغییر دهید:

- `txtFirstName`
- `txtLastName`
- `txtCompanyName`
- `txtAddress1`
- `txtAddress2`
- `txtCity`
- `txtRegion`
- `txtPostalCode`
- `txtCountry`

- txtEmail

۴) همچنین خاصیت Text کنترلهای Label و Button برنامه را نیز مانند شکل ۱۹-۱ تنظیم کنید.
۵) سپس خاصیت Name کنترل Button را برابر با btnSave قرار دهید. در آخر نیز خاصیت Name کنترل Labelای که با عبارت (Number) مشخص شده است را برابر با lblAddressNumber قرار دهید.

تمام کارهایی که برای طراحی ظاهر فرم انجام می دادیم همین بود. خوب بهتر است که در ادامه نحوه ی ذخیره ی داده ها در فایل XML را بررسی کنیم.

کلاس SerializableData:

در این برنامه از دو کلاس استفاده خواهیم کرد: کلاس Address و کتاب AddressBook. کلاس Address برای نگهداری یک رکورد از اطلاعاتی که باید در دفتر تلفن ذخیره شوند به کار می رود. کلاس AddressBook نیز شامل یک آرایه از اشیایی از کلاس Address است و تمام رکورد های موجود در دفتر تلفن را نگهداری می کند. همچنین این کلاس دارای متد هایی برای حرکت در بین داده های موجود در دفتر تلفن خواهد بود.
هر دو کلاسی که در این برنامه ایجاد خواهیم کرد از یک کلاس پایه به نام SerializableData مشتق می شوند. کلاس SerializableData شامل توابعی است که به وسیله ی آنها می توان داده های موجود در برنامه را در دیسک ذخیره کرده و یا داده هایی که در دیسک وجود دارند را در برنامه قرار داد. در XML، پروسه ی مربوط به ذخیره کردن اطلاعات یک شیئ از کلاس در دیسک به عنوان **سریالایز کردن**[1] و پروسه ی مربوط به ایجاد شیئ اولیه با توجه به اطلاعات موجود در دیسک به عنوان **دی سریالایز کردن**[2] شناخته می شود. در قسمت امتحان کنید بعد، با ایجاد کلاسهای SerializableData و نیز Address، از روش جدیدی برای ذخیره ی داده های موجود در دفتر تلفن در دیسک استفاده خواهیم کرد.

امتحان کنید: ایجاد کلاس SerializableData

۱) اولین کلاسی که باید ایجاد کنیم، کلاس SerializableData است. با استفاده از پنجره ی Solution Explorer، بر روی نام پروژه کلیک کرده و گزینه ی Add ➔ Class... را انتخاب کنید. در کادر Name عبارت SerializableData را وارد کرده و روی دکمه ی Add کلیک کنید تا کلاسی به این نام به برنامه اضافه شود.
۲) مجدداً در پنجره ی Solution Explorer روی نام پروژه کلیک راست کرده و از منوی باز شده گزینه ی Add Reference... را انتخاب کنید. در قسمت .NET از پنجره ی Add Reference گزینه ی System.XML را انتخاب کرده و روی دکمه ی OK کلیک کنید. سپس با استفاده از دستور using دو فضای نام مشخص شده را به کلاس SerializableData اضافه کنید:

---

[1] Serialization
[2] Deserialization

```
using System.IO;
using System.Xml.Serialization;

namespace Address_Book
{
      public class SerializableData
```

۳) سپس دو متد مشخص شده در زیر را به کلاس اضافه کنید:

```
// Save - serialize the object to disk...
public void Save(String filename)
{
      // make a temporary filename...
      String tempFilename;
      tempFilename = filename + ".tmp";

      // does the file exist?
      FileInfo tempFileInfo = new FileInfo(tempFilename);
      if (tempFileInfo.Exists)
            tempFileInfo.Delete();
      // open the file...
      FileStream stream = new FileStream(tempFilename,
                                         FileMode.Create);
      // save the object...
      Save(stream);
      // close the file...
      stream.Close();
      // remove the existing data file and
      // rename the temp file...
      tempFileInfo.CopyTo(filename, true);
      tempFileInfo.Delete();
}
// Save - actually perform the serialization...
public void Save(Stream stream)
{
      // create a serializer...
      XmlSerializer serializer = new XmlSerializer(this.GetType);
      // save the file...
      serializer.Serialize(stream, this);
}
```

۴) کلاس جدیدی به نام `Address` به پروژه اضافه کرده و کلاس `SerializableData` را مانند زیر به عنوان کلاس پایه ی آن مشخص کنید:

```
public class Address : SerializableData
{
}
```

۵) سپس فیلد های مشخص شده در زیر را به کلاس اضافه کنید. این فیلد ها برای نگهداری داده های یک رکورد از اطلاعات به کار می روند:

```
public class Address : SerializableData
{
	// Members...
	public String FirstName;
	public String LastName;
	public String CompanyName;
	public String Address1;
	public String Address2;
	public String City;
	public String Region;
	public String PostalCode;
	public String Country;
	public String Email;
}
```

۶) به قسمت طراحی فرم مربوط به Form1 برگشته و روی کنترل btnSave دو بار کلیک کنید تا متد مربوط به رویداد Click آن به صورت اتوماتیک ایجاد شود. سپس کد مشخص شده در زیر را به این متد اضافه کنید:

```
private void btnSave_Click(object sender, EventArgs e)
{
	// create a new address object...
	Address address = new Address();
	// copy the values from the form into the address...
	PopulateAddressFromForm(address);
	// save the address...
	String filename = DataFileName;
	address.Save(filename);
	// tell the user...
	MessageBox.Show("The address was saved to " + filename);
}
```

۷) با اضافه کردن این کد ویژوال استودیو اعلام می کند که خاصیت و یا متدی به نام DataFileName و یا PopulateAddressFromForm در کلاس Form1 وجود ندارد. برای رفع این مشکل ابتدا خاصیتی به نام DataFileName مانند زیر به کلاس Form1 اضافه می کنیم.

```
// DataFilename - where should we store our data?
public String DataFileName
{
	get
	{
		// get our working folder...
		String folder;
		folder = Environment.CurrentDirectory;
		// return the folder with the name "Addressbook.xml"...
		return folder + "\\AddressBook.xml";
	}
}
```

نکته: در مورد دلیل استفاده از دو کاراکتر \ به جای یک \ در عبارت بالا، می توانید به فصل چهارم مراجعه کنید.

۸) حال باید متد PopulateAddressFromForm را ایجاد کنیم. برای این کار کد زیر را به کلاس Form1 اضافه کنید:

```
// PopulateAddressFromForm - populates Address from the form fields...
public void PopulateAddressFromForm(Address address)
{
      // copy the values...
      address.FirstName = txtFirstName.Text;
      address.LastName = txtLastName.Text;
      address.CompanyName = txtCompanyName.Text;
      address.Address1 = txtAddress1.Text;
      address.Address2 = txtAddress2.Text;
      address.City = txtCity.Text;
      address.Region = txtRegion.Text;
      address.PostalCode = txtPostalCode.Text;
      address.Country = txtCountry.Text;
      address.Email = txtEmail.Text;
}
```

۹) برنامه را اجرا کرده و کادر های موجود در فرم را تکمیل کنید.

۱۰) روی دکمه ی Save کلیک کنید. کادر پیغامی نمایش داده شده و محل ذخیره ی فایل مربوط به داده ها را اعلام می کند.

۱۱) با استفاده از Windows Explorer به فولدری که این فایل XML در آنجا ذخیره شده است بروید. روی آن فایل دو بار کلیک کنید تا اینترنت اکسپلورر باز شده و محتویات آن را نمایش دهد. به این ترتیب داده هایی را مشابه زیر در این فایل مشاهده خواهید کرد.

```
<?xml version="1.0" ?>
<Address xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <FirstName>Muhammad</FirstName>
  <LastName>Hashemian</LastName>
  <CompanyName>MyCompany</CompanyName>
  <Address1>11 First Avenue</Address1>
  <Address2 />
  <City>No Where</City>
  <Region>North</Region>
  <PostalCode>28222</PostalCode>
  <Country>Iran</Country>
  <Email>Muhammad@email.com</Email>
</Address>
```

چگونه کار می کند؟

ابتدا بهتر است کد XML ای که به وسیله ی برنامه ایجاد شده است را بررسی کنیم. محتویات خط اول در این قسمت زیاد مهم نیستند. این خط بیان می کند که این فایل شامل کد XML نسخه ی ۱ است. همچنین خصیصه xmlns در خط دوم و سوم هم

در این برنامه اهمیت ندارند، زیرا این خصیصه نیز فقط اطلاعاتی را در مورد این فایل بیان می کند و در این مرحله می توانید به .NET اجازه دهید این اطلاعات را برای شما در فایل تنظیم کند. با حذف این موارد، کدی که باقی می ماند به صورت زیر خواهد بود:

```
<Address>
  <FirstName>Muhammad</FirstName>
  <LastName>Hashemian</LastName>
  <CompanyName>MyCompany</CompanyName>
  <Address1>11 First Avenue</Address1>
  <Address2 />
  <City>No Where</City>
  <Region>North</Region>
  <PostalCode>28222</PostalCode>
  <Country>Iran</Country>
  <Email>Muhammad@email.com</Email>
</Address>
```

مشاهده می کنید که این کد بسیار شبیه کدی است که در ابتدای این فصل بررسی کردیم، دارای تگ های شروع و تگ های پایان است که عناصر آن را تشکیل می دهند. هر عنصر در این فایل داده ای را نگهداری می کند و از نام عناصر می توان فهمید که چه داده ای در آن ذخیره شده است. برای مثال در عنصر CompanyName نام شرکتی که فرد در آن کار می کند مشخص شده است.

دقت کنید که فایل با تگ Address شروع می شود و با همین تگ نیز تمام می شود و تمام عناصر دیگری که در آن وجود دارند در این تگ قرار گرفته اند. این مورد مشخص می کند که تمام این عناصری عضوی از عنصر Address هستند. عنصر Address در این فایل اولین عنصر است، بنابراین به عنوان **عنصر ریشه**[1] شناخته می شود.

به خطی که دومین آدرس در آن قرار گرفته است توجه کنید: <Address2 /> با قرار دادن یک کاراکتر / در انتهای تگ شروع، مشخص می کنیم که این عنصر خالی است و داده ای در آن قرار ندارد. البته این خط را به این صورت نیز می توان نوشت:

```
<Address2></Address2>
```

اما همانطور که مشاهده می کنید، این نوع نوشتن مکان بیشتری را برای ذخیره سازی اشغال می کند.

خوب، تا اینجا متوجه شدیم که چه فایلی توسط برنامه تولید شده است. اما سوال اینجاست که این فایل چگونه ایجاد شده است؟ بنابراین از هنگامی که کاربر روی دکمه ی Save کلیک می کند اجرای برنامه را دنبال می کنیم.

ابتدا در این متد یک شیئ از نوع Address ایجاد کرده و سپس با فراخوانی متد PopulateAddressFromForm شیئ ایجاد شده را به این متد ارسال می کنیم. این متد نیز فقط داده های درون کادرهای روی فرم را می خواند و آنها را در فیلد های مختلف کلاس Address قرار می دهد.

```
private void btnSave_Click(object sender, EventArgs e)
{
      // create a new address object...
      Address address = new Address();
      // copy the values from the form into the address...
      PopulateAddressFromForm(address);
```

[1] Root Element

سپس با استفاده از خاصیت DataFileName نام فایلی که باید داده ها در قالب XML در آن ذخیره شوند را دریافت می کنیم. برای این کار با استفاده از خاصیت CurrentDirectory از کلاس Environment، مسیری که فایل اجرایی برنامه در آن قرار دارد را به دست آورده و نام فایل "\AddressBook.xml" را به انتهای این مسیر اضافه می کنیم تا مشخص شود داده ها باید در چه فایلی قرار گیرند. در این برنامه قرارداد می کنیم که برای ذخیره و یا بازیابی داده های برنامه، از فایل XMLای که در بالا ایجاد کردیم استفاده کنیم.

```
        // save the address...
        String filename = DataFileName;
```

سپس با فراخوانی متد Save از کلاس Address، داده ها را در فایل مشخص شده ذخیره می کنیم. این متد از کلاس SerializableData به کلاس Address به ارث رسیده است و در ادامه با نحوه ی عملکرد آن آشنا خواهیم شد. بعد از اینکه فایل را ذخیره کردیم با استفاده از یک کادر پیغام، مکان آن را به کاربر نمایش می دهیم:

```
        address.Save(filename);
        // tell the user...
        MessageBox.Show("The address was saved to " + filename);
}
```

در کلاس SerializableData دو نسخه از متد Save وجود دارند که جالب ترین قسمتهای این برنامه محسوب می شوند. یک نسخه آدرس فایل را به عنوان ورودی دریافت کرده و شیئ از نوع Stream را با استفاده از آن ایجاد می کند. سپس این شیئ را به نسخه ی دوم متد Save می فرستد تا به وسیله ی آن داده ها ذخیره شوند. نسخه ی دوم نیز با استفاده از کلاس System.Xml.Serialization.XmlSerializer که در ادامه با آن آشنا خواهیم شد و نیز شیئ Stream که به آن فرستاده شده است داده ها را در قالب XML ذخیره می کند.

در هنگام ذخیره ی داده ها در فایل باید به یک مسئله دقت کنید. هنگامی که بخواهیم یک رکورد را ذخیره کنیم، داده های آن در فایلی با همان نام فایل قبلی ذخیره می شوند. بنابراین فایل مربوط به اطلاعات رکورد قبلی از بین می رود. اما باید دقت کنیم فایل جدید را به گونه ای ایجاد شود که اگر به هر دلیلی در ایجاد آن با شکست مواجه شدیم، اطلاعات رکورد قبلی از بین نروند و کاربر بتواند به آنها دسترسی داشته باشد. روش انجام این کار نیز ساده است. ابتدا داده های مربوط به رکورد جاری را در یک فایل با نام دیگری ذخیره می کنیم. زمانی که همه چیز با موفقیت به اتمام رسید، فایل ایجاد شده را با فایل مربوط به اطلاعات رکورد قبلی جایگزین می کنیم.

برای نام فایلی که باید داده های رکورد جدید به صورت موقت در آن نگهداری شوند، به نام فایل اصلی پسوند .tmp اضافه می کنیم. بنابراین اگر نام فایلی که داده ها باید در آن ذخیره شوند برابر با C:\MyProgramms\Address Book\AddressBook.xml باشد، ابتدا داده ها در فایلی به نام C:\MyPrograms\Address Book\AddressBook.xml.tmp ذخیره می شود. خوب، ابتدا قبل از ایجاد فایل، بررسی می کنیم که فایلی با این نام وجود نداشته باشد. ولی اگر چنین فایلی ایجاد شده باشد آن را با فراخوانی متد Delete حذف می کنیم:

```
// Save - serialize the object to disk...
public void Save(String filename)
{
        // make a temporary filename...
        String tempFilename;
        tempFilename = filename + ".tmp";

        // does the file exist?
```

```
        FileInfo tempFileInfo = new FileInfo(tempFilename);
        if (tempFileInfo.Exists)
              tempFileInfo.Delete();
```

در مرحله ی بعد یک فایل با نام مشخص شده و پسوند `.tmp` ایجاد می کنیم. برای این کار باید یک شیئ از کلاس `FileStream` نمونه سازی کرده و آدرس فایل و نیز نوع دسترسی به آن را در متد سازنده ی کلاس مشخص کنیم.

```
        // open the file...
        FileStream stream = new FileStream(tempFilename,
                                           FileMode.Create);
```

سپس این شیئ را به نسخه ی دوم متد `Save` ارسال می کنیم. نحوه ی عملکرد این متد در ادامه توضیح داده شده است، اما فعلا بدانید که قسمت اصلی عمل سریالایز کردن درون این متد انجام می شود.
سپس فایل را می بندیم:

```
        // close the file...
        stream.Close();
```

در مرحله ی آخر نیز ابتدا با استفاده از متد `CopyTo` فایل را در فایل جدید با نام اصلی کپی می کنیم (پارامتر `true` مشخص می کند که هنگام کپی کردن این فایل، اگر فایلی با این نام وجود داشت آن را حذف کن)، سپس فایل موقتی را با استفاده از متد `Delete` پاک می کنیم.

```
        // remove the existing data file and
        // rename the temp file...
        tempFileInfo.CopyTo(filename, true);
        tempFileInfo.Delete();
}
```

نسخه ی دوم متد `Save` به جای یک رشته، یک شیئ از کلاس `Stream` و یا یکی از کلاسهای مشتق شده از آن را به عنوان ورودی دریافت می کند و مشابه زیر است:

```
// Save - actually perform the serialization...
public void Save(Stream stream)
{
        // create a serializer...
        XmlSerializer serializer = new XmlSerializer(this.GetType);
        // save the file...
        serializer.Serialize(stream, this);
}
```

نکته: تمام موارد مربوط به ورودی و خروجی در یک برنامه به وسیله ی کلاسهای مشتق شده از کلاس `Stream` انجام می شوند، برای مثال خواندن و نوشتن در یک فایل، خواندن و نوشتن در حافظه و .... این کلاس به همراه کلاسهای دیگری در رابطه با همین موضوع در فضای نام `System.IO` قرار دارد.

کلاس System.Xml.Srialization.XmlSerializer، کلاسی است که در این قسمت برای سریالایز کردن شیئ ایجاد شده و نوشتن آن در خروجی ای که به وسیله ی پارامتر stream مشخص شده است به کار می رود. در این قسمت، از یک خروجی استفاده کردیم که داده ها را در یک فایل قرار می دهد، اما در ادامه ی این فصل از خروجی های دیگری نیز استفاده خواهیم کرد.
کلاس XmlSerializer باید بداند که برای سریالایز کردن چه نوع شیئ مورد استفاده قرار می گیرد، بنابراین هنگام ایجاد شیئ ای از این کلاس، با استفاده از متد GetType نوع شیئ را برای آن مشخص می کنیم. متد GetType در کلاس Object تعریف شده است و شیئ ا ی را از نوع System.Type به عنوان نتیجه برمی گرداند که نوع کلاسی که هم اکنون مورد استفاده قرار گرفته است را مشخص می کند (کلاس Address). دلیل اینکه شیئ ای که از کلاس XmlSerializer ایجاد می کنیم نیاز دارد بداند چه شیئ را باید سریالایز کند در نحوه ی عملکرد این کلاس برای سریالایز کردن است. این کلاس در بین خاصیتهای یک شیئ حرکت می کند و مقدار خاصیت هایی را که به صورت خواندنی-نوشتنی باشند را (دارای هر دو قسمت get و set باشند) به وسیله ی شیئ stream که برای آن مشخص شده است در فایل XML قرار می دهد. بنابراین در اینجا، این کلاس مقدار تمام فیلد های موجود در کلاس Address را دریافت کرده و آنها را به صورت XML در فایل AddressBook.xml می نویسد.
کلاس XmlSerializer برای نام هر عنصر در فایل XML، از نام خاصیت یا فیلد مربوط به آن در کلاس استفاده می کند. برای مثال در این برنامه عنصر FirstName دارای داده های موجود در عنصر فیلد FirstName است. همچنین برای نام عنصر ریشه نیز از نام کلاس استفاده می شود. برای مثال در اینجا نام عنصر ریشه برابر با Address، یعنی نام کلاس است.
کلاس XmlSerializer یک راه مناسب برای استفاده از مزایای XML در برنامه است. زیرا با استفاده از آن دیگر مجبور نیستید خود را درگیر خواندن و یا نوشتن فایلهای XML کنید، تمام کارهای لازم را این کلاس انجام می دهد.

دریافت داده ها از یک فایل XML:

در بخش امتحان کنید بعد، قابلیتی را به برنامه اضافه خواهیم کرد تا بتواند داده ها را از یک فایل XML خوانده و آنها را در برنامه قرار دهد، به عبارت دیگر داده های موجود در دیسک را دی سریالایز خواهیم کرد.

امتحان کنید: دریافت داده ها از فایل XML

۱) ویرایشگر کد مربوط به کلاس SerializableData را باز کرده و دو متد مشخص شده در زیر را به آن اضافه کنید:

```
// Load - deserialize from disk...
public static Object Load(String filename, Type newType)
{
      // does the file exist?
      FileInfo fileInfo = new FileInfo(filename);
      if(!fileInfo.Exists)
      {
            // create a blank version of the object and return that...
            return System.Activator.CreateInstance(newType);
```

```
        }
        // open the file...
        FileStream stream = new FileStream(filename, FileMode.Open);
        // load the object from the stream...
        Object newObject = Load(stream, newType);
        // close the stream...
        stream.Close();
        // return the object...
        return newObject;
}
public static Object Load(Stream stream,Type newType)
{
        // create a serializer and load the object....
        XmlSerializer serializer = new XmlSerializer(newType);
        Object newObject = serializer.Deserialize(stream);
        // return the new object...
        return newObject;
}
```

۲) به قسمت طراحی فرم مربوط به `Form1` برگشته و با استفاده از جعبه ابزار، یک کنترل `Button` دیگر را به فرم اضافه کنید. خاصیت `Name` این کنترل را برابر با `btnLoad` و خاصیت `Text` آن را برابر با `&Load` قرار دهید.

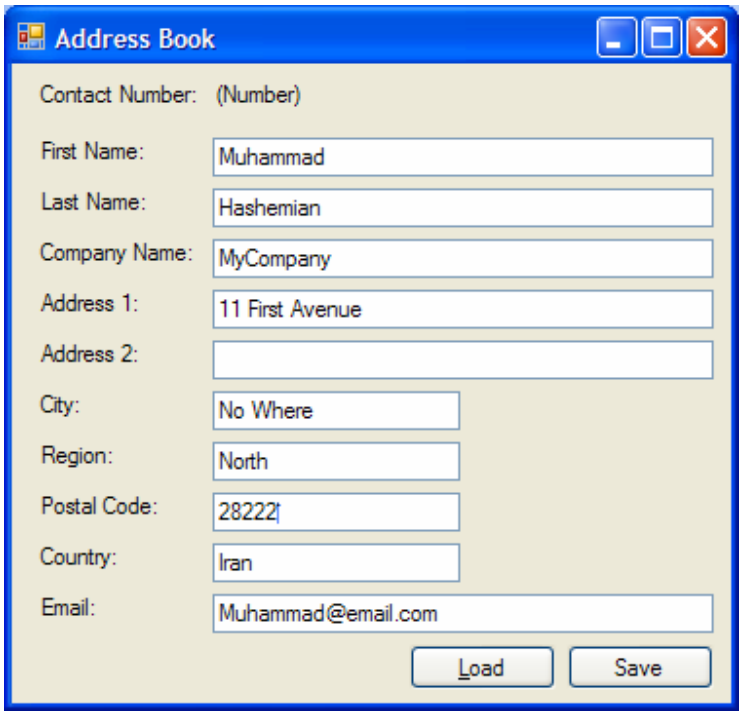
۳) روی این دکمه دو بار کلیک کرده تا متد مربوط به رویداد `Click` آن ایجاد شود. سپس کد مشخص شده در زیر را در این متد وارد کنید:

```
private void btnLoad_Click(object sender, EventArgs e)
{
        // load the address using a shared method on SerializableData...
        Address newAddress = (Address)SerializableData.Load(DataFileName,
                                                    typeof(Address) );
        // update the display...
        PopulateFormFromAddress(newAddress);
}
```

۴) همچنین لازم است که متد مشخص شده در زیر را نیز به کلاس `Form1` اضافه کنید:

```
// PopulateFormFromAddress - populates the form from an
// address object...
public void PopulateFormFromAddress(Address address)
// copy the values...
{
        txtFirstName.Text = address.FirstName;
        txtLastName.Text = address.LastName;
        txtCompanyName.Text = address.CompanyName;
        txtAddress1.Text = address.Address1;
        txtAddress2.Text = address.Address2;
        txtCity.Text = address.City;
        txtRegion.Text = address.Region;
        txtPostalCode.Text = address.PostalCode;
        txtCountry.Text = address.Country;
        txtEmail.Text = address.Email;
}
```

۵) برنامه را اجرا کرده و روی دکمه ی Load کلیک کنید یا دکمه های Alt+L را فشار دهید. اطلاعاتی که در قسمت قبل وارد کرده بودید، همانند شکل ۱۹-۲ در فرم برنامه نمایش داده خواهند شد.



شکل ۱۹-۲

چگونه کار می کند؟

دی سریالایز کردن, عمل عکس سریالایز کردن است. به وسیله ی این کار می توانید داده هایی را که در یک فایل با قالب XML ذخیره شده اند را استخراج کرده و در برنامه قرار دهید. برای دی سریالایز کردن یک شیئ نیز می توانید از کلاس XmlSerializer استفاده کنید.

برای این کار نیز مانند قبل دو نسخه ی گوناگون از متد Load ایجاد می کنیم. نسخه ی اول دو پارامتر دریافت می کند: پارامتر اول از نوع String است و آدرس فایلی را مشخص می کند که داده ها در آن قرار دارد، پارامتر دوم نیز از نوع System.Type است و نوع کلاسی را دربر دارد که باید داده ها به شیئ ای از آن نوع تبدیل شوند. برای فراخوانی این تابع پارامتر دوم به این دلیل مورد نیاز است که کلاس XmlSerializer با توجه به آن بتواند تشخیص دهد مقادیر چه خاصیت هایی باید در فایل وجود داشته باشند تا بتواند آنها را از فایل استخراج کند؟

دقت کنید که کلاس XmlSerializer هیچ اطلاعاتی در مورد فضای نام و یا فایل اسمبلی که شیئ به آن تعلق دارد را در فایل XML ذخیره نمی کند، بنابراین هنگام دی سریالایز کردن فقط بر اساس گفته ی برنامه نویس در مورد نوع شیئ داده های درون فایل را بررسی می کند و سعی می کند شیئ ای از نوع کلاس مشخص شده را ایجاد کند (کلاس XmlSerializer فقط نام کلاسی که سریالایز کرده است را به عنوان نام عنصر ریشه قرار می دهد، ولی با استفاده از آن

هم نمی تواند نوع واقعی کلاس را تشخیص دهد. زیرا برای نمونه در مثال بالا ممکن است شما چندین کلاس به نام Address در فضای نامهای گوناگون داشته باشید).
مسلماً در نسخه ی اول متد Save اولین کاری که باید انجام دهیم این است که ببینیم آیا فایل مشخص شده وجود دارد یا نه؟ اگر فایل وجود نداشت یک شیئ خالی از نوع Address ایجاد کرده و آن را به عنوان نتیجه برمی گردانیم:

```
// Load - deserialize from disk...
public static Object Load(String filename, Type newType)
{
	// does the file exist?
	FileInfo fileInfo = new FileInfo(filename);
	if(!fileInfo.Exists)
	{
		// create a blank version of the object and return that...
		return System.Activator.CreateInstance(newType);
	}
```

اما اگر فایل وجود داشت، مانند متد Save آن را باز می کنید و سپس به نسخه ی دوم متد Load می فرستید. سپس فایل را می بندید و شیئ ای که از نسخه ی دوم متد Load برگشته است را به عنوان نتیجه برمی گردانید:

```
	// open the file...
	FileStream stream = new FileStream(filename, FileMode.Open);
	// load the object from the stream...
	Object newObject = Load(stream, newType);
	// close the stream...
	stream.Close();
	// return the object...
	return newObject;
}
```

نسخه ی دوم متد Load نیز دوم پارامتر دریافت می کند: پارامتر اول یک شیئ از نوع Stream است که حاوی اتصالی به فایل XML مورد نظر است. پارامتر دوم نیز یک شیئ از کلاس System.Type است و نوع شیئ را مشخص می کند که باید به وسیله ی کلاس XmlSerializer ایجاد شود. همانطور که مشاهده می کنید این متد بسیار مشابه متد Save است که در قسمت قبل توضیح دادیم و نکته ی خاصی ندارد. فقط در این متد با استفاده از متد Deserialize از کلاس XmlSerializer شیئ مورد نظر را بر اساس داده های موجود در فایل XML ایجاد می کنیم:

```
public static Object Load(Stream stream,Type newType)
{
	// create a serializer and load the object....
	XmlSerializer serializer = new XmlSerializer(newType);
	Object newObject = serializer.Deserialize(stream);
	// return the new object...
	return newObject;
}
```

هنگامی که متد Deserialize را فراخوانی می کنیم، باید نوع شیئ ای که می خواهیم ایجاد کنیم را برای آن مشخص کنیم. به این ترتیب این متد می تواند بین تمام خاصیت های آن شیئ حرکت کرده و به دنبال آنهایی بگردد که از نوع خواندنی-نوشتنی باشند. هنگامی که چنین خاصیتی را پیدا کرد، به داخل فایل XML می رود و سعی می کند که مقدار آن خاصیت را از داخل

آن فایل استخراج کند و بنابراین شیئ را ایجاد می کند که مقادیر درون خاصیت ها و فیلد های آن دقیقاً برابر با مقادیری است که در فایل XML قرار داده بودیم.
در فرم اصلی برنامه هنگامی که متد btnLoad_Click فراخوانی شد، متد Load را فراخوانی می کنیم. برای مشخص کردن مقدار پارامتر دوم این متد، از عملگر typeof استفاده می کنیم تا نوع شیئ مورد نیاز را بدست آوریم. این عملگر یک شیئ از کلاس System.Type را برمی گرداند که مشخص کننده ی نوع کلاسی است که به آن فرستاده شده. متد Load نیز یک شیئ از کلاس Object به عنوان نتیجه برمی گرداند، پس آن را ابتدا به شیئ ای از کلاس Address تبدیل کرده و نتیجه را در newAddress قرار می دهیم. سپس با فراخوانی متد PopulateFormFromAddres، داده های شیئ جدید را در فرم قرار می دهیم.

```
private void btnLoad_Click(object sender, EventArgs e)
{
      // load the address using a shared method on SerializableData...
      Address newAddress = (Address)SerializableData.Load(DataFileName,
                                                typeof(Address) );
      // update the display...
      PopulateFormFromAddress(newAddress);
}
```

تغییر در داده ها:

برای اینکه ثابت کنیم در این مراحل هیچ چیز عجیب و خاصی رخ نمی دهد، در قسمت امتحان کنید بعد داده ها را با استفاده از یک ویرایشگر متنی مانند notepad تغییر داده و نتیجه را در برنامه مشاهده می کنیم.

امتحان کنید: ایجاد تغییر در داده ها

۱) با استفاده از Notepad فایل XMLای که در برنامه ایجاد کرده بودیم را باز کرده و مقدار موجود در عنصر FirstName را به نام دیگری تغییر دهید. سپس فایل را ذخیره کرده و از Notepad خارج شوید.
۲) برنامه را اجرا کرده و روی دکمه ی Load کلیک کنید. مشاهده خواهید کرد که نام جدید در قسمت First Name نمایش داده می شود.

چگونه کار می کند؟

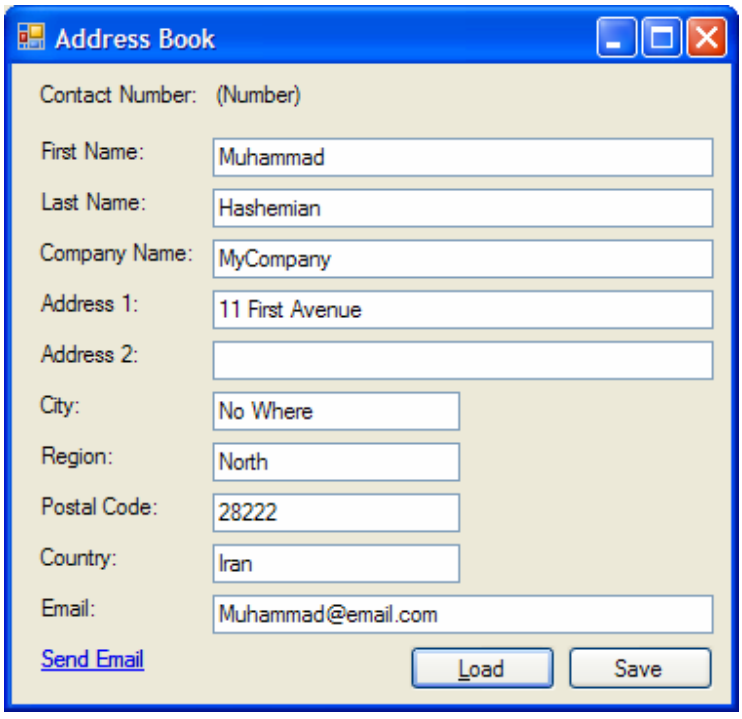
هدف از این مثال این بود که ثابت کنیم کلاس XmlSerializer فقط از اطلاعات داخل فایل AddressBook.xml برای ایجاد شیئ مورد نظر استفاده می کند. بنابراین هر تغییری که در داده های این فایل ایجاد کنید، فیلد های متناظر در شیئ موجود در برنامه نیز تغییر خواهند کرد.

فرستادن ایمیل:

در قسمت امتحان کنید بعد، مشاهده خواهیم کرد که چگونه می توان با استفاده از یک برنامه ی ارسال ایمیل مانند Outlook به آدرس ایمیل که در رکورد های این دفتر تلفن وارد می شود نامه فرستاد. برای این کار با استفاده از کلاس Process، برنامه ی مربوط به ارسال ایمیل را اجرا کرده و نامه ی را ارسال خواهیم کرد.

امتحان کنید: ارسال ایمیل در برنامه

۱) به قسمت طراحی مربوط به Form1 بروید و با استفاده از جعبه ابزار یک کنترل LinkLabel را در پایین لیبل Email قرار دهید. خاصیت Name این کنترل را برابر با lnkSendEmail خاصیت Text آن را برابر با Send Email قرار دهید (شکل ۱۹-۳).



شکل ۱۹-۳

۲) روی این کنترل دو بار کلیک کنید تا متد مربوط به رویداد LinkClicked آن به صورت اتوماتیک ایجاد شود. سپس کد مشخص شده در زیر را در این متد وارد کنید:

```
private void lnkSendEmail_LinkClicked(object sender,
                                      LinkLabelLinkClickedEventArgs e)
{
      // Start the email client...
      System.Diagnostics.Process.Start("mailto: " + txtEmail.Text);
}
```

۳) برنامه را اجرا کرده و روی دکمه ی Load کلیک کنید تا داده ها از فایل استخراج شوند و در برنامه قرار گیرند. ابتدا مطمئن شوید که آدرس ایمیل مورد نظر شما در کادر Email وارد شده است، سپس روی لینک Send Email کلیک کنید. مشاهده می کنید که برنامه ی کنترل ایمیل شما باز شده و به قسمت مربوط به ارسال ایمیل می رود، در حالی که کادر To: در این برنامه با آدرس ایمیلی که در برنامه وارد کرده بودید کامل شده است.

چگونه کار می کند؟

یکی از قابلیتهای درونی ویندوز این است که هنگامی که یک آدرس اینترنتی در آن وارد شد، بتواند نوع آن را تشخیص دهد و برنامه ی مربوط به آن را اجرا کند.
هنگامی که یک برنامه ی مربوط به کنترل ایمیل را در سیستم خود نصب می کنید، این برنامه یک پروتکل به نام mailto را در ویندوز ثبت می کند، دقیقاً مانند پروتکل HTTP که هنگام نصب یک مرورگر در ویندوز ثبت می شود تا تمام آدرس هایی که با آن شروع می شوند به مرورگر فرستاده شوند. بنابراین تمام آدرس هایی که با mailto شروع می شوند نیز به صورت اتوماتیک به وسیله ی ویندوز به برنامه ی مربوط به مدیریت ایمیل فرستاده می شوند.
بنابراین اگر بخواهید نامه ای را در ویندوز ارسال کنید، می توانید از منوی استارت گزینه ی Run را انتخاب کرده و در آن عبارت mailto به همراه آدرس ایمیل مورد نظر خود را وارد کنید. سپس روی دکمه ی OK کلیک کنید. به این ترتیب برنامه ی ارسال ایمیل باز می شود و می توانید نامه ی مورد نظر خود را پست کنید.
در برنامه نیز از روشی مشابه استفاده می کنیم، یعنی مقدار درون فیلد txtEmail را بدست آورده و آن را بعد از عبارت mailto قرار می دهیم تا آدرس مورد نظر را ایجاد کنیم. سپس آدرس ایجاد شده را به متد Start از کلاس Process ارسال می کنیم تا برنامه ی مربوط به مدیریت ایمیل را باز کند:

```
private void lnkSendEmail_LinkClicked(object sender,
                                      LinkLabelLinkClickedEventArgs e)
{
      // Start the email client...
      System.Diagnostics.Process.Start("mailto: " + txtEmail.Text);
}
```

متد Start نیز دقیقاً مشابه پنجره ی Run در ویندوز عمل می کند. برای مثال اگر به جای عبارت mailto در کد بالا از عبارت http استفاده می کردیم، آدرس وارد شده در یک صفحه ی وب در اینترنت اکسپلورر نمایش داده می شد. همچنین اگر در پارامتر این متد نام یک فایل Word و یا یک صفحه گسترده ی Excel را وارد کنید، برنامه های Word و یا Excel باز شده و فایل مورد نظر شما را نمایش می دهد. البته اگر بخواهید یک فایل را اجرا کنید نیازی نیست که از پروتکلی مانند mailto و یا http استفاده کنید، بلکه کافی است نام فایل را به متد Start ارسال کنید مانند:

```
C:\My Files\My Budgets.xls
```

ایجاد لیستی از آدرسها:

در این قسمت از امتحان کنید می خواهیم برنامه را به گونه ای تغییر دهیم تا بتواند لیستی از آدرسها را در فایل XML ذخیره کند. تا اینجا برنامه فقط می تواند یک آدرس را به درستی در خود نگهداری کند، بنابراین باید از اینجا به بعد توجه خود را روی ایجاد لیستی از آدرسها متمرکز کنیم. برای این کار کلاسی به نام AddressBook ایجاد کرده و آن را از کلاس SerializableData مشتق می کنیم. زیرا در آخر می خواهیم کلاس AddressBook را به گونه ای ایجاد کنیم که بتواند با فراخوانی یک متد داده های درون خود را در دیسک ذخیره کرده و یا داده های ذخیره شده در دیسک را در حافظه قرار دهد.

امتحان کنید: ایجاد کلاس AddressBook

۱) با استفاده از Solution Explorer یک کلاس جدید به نام AddressBook به برنامه اضافه کنید.
۲) ابتدا با استفاده از دستور using فضای نام زیر را به کلاس اضافه کنید:

```
using System.Xml.Serialization;
using System.Collections;

namespace Address_Book
{
	class AddressBook
	{
```

۳) سپس تعریف کلاس را مانند زیر به گونه ای تغییر دهید که از کلاس SerializableData مشتق شود:

```
namespace Address_Book
{
	class AddressBook : SerializableData
	{
```

۴) برای ذخیره کردن آدرسها، از کلاس ArrayList در فضای نام System.Collections استفاده می کنیم. پس در این کلاس ابتدا به متدی نیاز داریم که بتواند یک رکورد جدید را ایجاد کرده و آن را به لیست رکورد های موجود اضافه کند. برای این کار متد زیر را به کلاس AddressBook اضافه کنید:

```
class AddressBook : SerializableData
{
	// members...
	public ArrayList Items = new ArrayList();
	// AddAddress - add a new address to the book...
	public Address AddAddress()
	{
		// create one...
		Address newAddress = new Address();
		// add it to the list...
		Items.Add(newAddress);
		// return the address...
		return newAddress
	}
}
```

۵) ویرایشگر کد مربوط به کلاس Form1 را باز کرده و سپس کد زیر را به ابتدای کلاس اضافه کنید:

```
public partial class Form1 : Form
{
	// members...
	public AddressBook Addresses;
	private int _currentAddressIndex;
```

۶) سپس خاصیت زیر را به Form1 اضافه کنید:

```
// CurrentAddress - property for the current address...
Address CurrentAddress
{
	get
	{
		return AddressBook.Items[CurrentAddressIndex - 1];
	}
}
```

۷) همچنین خاصیت زیر را نیز به کلاس Form1 اضافه کنید:

```
// CurrentAddressIndex - property for the current address...
int CurrentAddressIndex
{
	get
	{
		return _currentAddressIndex;
	}
	set
	{
		// set the address...
		_currentAddressIndex = Value;
		// update the display...
		PopulateFormFromAddress(CurrentAddress);
		// set the label...
		lblAddressNumber.Text = _currentAddressIndex +
					" of " + AddressBook.Items.Count;
	}
}
```

۸) به قسمت طراحی Form1 رفته و روی قسمتی خالی از فرم دو بار کلیک کنید تا متد مربوط بـه رویـداد Click آن به صورت اتوماتیک ایجاد شود. سپس کد مشخص شده در زیر را به این متد اضافه کنید:

```
private void Form1_Load(object sender, EventArgs e)
{
	// load the address book...
	Addresses = (AddressBook)SerializableData.Load(DataFileName,
						typeof(AddressBook));
	// if the address book contains no item, add a new one...
	if (Addresses.Items.Count == 0)
		Addresses.AddAddress();
```

```
        // select the first item in the list...
        CurrentAddressIndex = 1;
}
```

۹) تا اینجا برنامه ی تواند رکورد های موجود در برنامه را لود کند. حال باید قابلیت ذخیره کردن را نیز اضافه کنیم. برای این کار در قسمت طراحی Form1، فرم را انتخاب کرده و در پنجره ی Properties روی آیکون Events کلیک کنید. در لیست رویدادهای مربوط به فرم، رویداد FormClosed را انتخاب کرده و روی آن دو بار کلیک کنید تا متد مربوط به آن ایجاد شود. سپس کد مشخص شده در زیر را به این متد اضافه کنید:

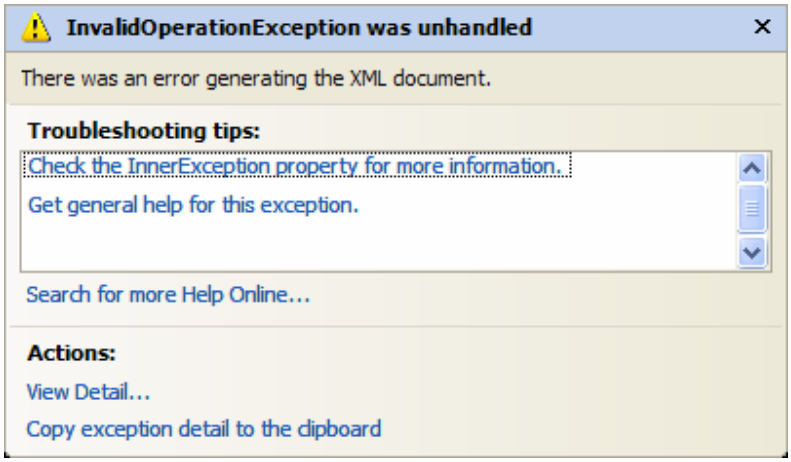
```
private void Form1_FormClosed(object sender, FormClosedEventArgs e)
{
        // save the changes...
        UpdateCurrentAddress();
        SaveChanges();
}
```

۱۰) در انتها نیز دو متد زیر را نیز در کلاس Form1 ایجاد کنید:

```
// SaveChanges - save the address book to an XML file...
public void SaveChanges()
{
        // tell the address book to save itself...
        Addresses.Save(DataFileName);
}
// UpdateCurrentAddress - make sure the book has the current
// values currently entered into the form...
private void UpdateCurrentAddress()
{
        PopulateAddressFromForm(CurrentAddress);
}
```

نکته: قبل از اینکه برنامه را اجرا کنید، باید فایل AddressBook.xml که در قسمت قبلی ایجاد شده بود را حذف کنید. اگر این کار را انجام ندهید، کلاس XmlSerializer سعی خواهد رد یک شیئ AddressBook را با استفاده از داده های فایل XML قبلی ایجاد کند که حاوی یک شیئ از نوع Address است و به همین دلیل یک استثنا رخ خواهد داد.

۱۱) برنامه را اجرا کنید، اما لازم نیست به خود زحمت دهید و داده ای را در فرم وارد کنید زیرا متد Save کار نمی کند. با بستن فرم مشاهده خواهید کرد که خطایی مشابه شکل ۱۹-۴ رخ می دهد.



شکل ۱۹-۴

چگونه کار می کند (یا چگونه کار نمی کند)؟

هنگامی که برنامه اجرا شده و متد `Form_Load` فراخوانی می شود، ابتدا این متد از کلاس `SerializableData` می خواهد تا داده های موجود در فایل `AddressBook.xml` را استخراج کرده و در برنامه قرار دهد، اما چون قبل از اجرای برنامه این فایل را پاک کرده اید، کلاس `SerializableData` نمی تواند به داده های درون آن دسترسی داشته باشد. بنابراین متد `Load` یک نمونه ی جدید از کلاسی که نوع آن را با استفاده از پارامتر دوم به آن فرستاده بودید ایجاد کرده و آن را برمی گرداند. در این حالت، یک نمونه از کلاس `AddressBook` به وسیله ی این متد برگشته می شود:

```
private void Form1_Load(object sender, EventArgs e)
{
      // load the address book...
      Addresses = (AddressBook)SerializableData.Load(DataFileName,
                                                typeof(AddressBook));
```

اما شیئ ای که در `Addresses` قرار گرفته است شامل هیچ آدرسی نمی شود و خالی است. بنابراین در یک شرط بررسی می کنیم تا اگر هیچ رکوردی در شیئ قرار نداشت، یک رکورد خالی ایجاد کرده و آن را به لیست آدرسها اضافه کنیم:

```
      // if the address book contains no item on it, add a new one...
      if (Addresses.Items.Count == 0)
            Addresses.AddAddress();
```

بنابراین بعد از اجرای این خط، یا لیست `Addresses` حاوی آدرس های موجود در فایل `XML` ای است که قبلا ایجاد شده بود، و یا این لیست فقط شامل یک رکورد خالی می شود که آن را در دستور `if` ایجاد کرده ایم. پس در هر صورت حداقل یک رکورد در این لیست وجود خواهد داشت. پس خاصیت `CurrentAddressIndex` را برابر با ۱ قرار می دهیم تا رکورد اول را در کادرهای موجود در فرم نمایش دهد:

```
      // select the first item in the list...
```

```
        CurrentAddressIndex = 1;
}
```

خاصیت CurrentAddressIndex هنگام تنظیم اندیس یک رکورد چندین کار را انجام می دهد. ابتدا مقدار فیلد _currentAddressIndex که از نوع private تعریف شده است را برابر با شماره رکورد جدید قرار می دهد:

```
// CurrentAddressIndex - property for the current address...
int CurrentAddressIndex
{
        get
        {
                return _currentAddressIndex;
        }
        set
        {
                // set the address...
                _currentAddressIndex = value;
```

سپس با استفاده از خاصیت CurrentAddress شیئ Address مربوط به رکورد کنونی را بدست آورده و آن را به متد PopulateFormFromAddress را فراخوانی می کند تا داده های آن در کادرهای موجود در فرم نمایش داده شود:

```
                // update the display...
                PopulateFormFromAddress(CurrentAddress);
```

در آخر نیز متن موجود در کنترل lblAddressNumber را به گونه ای تغییر می دهد تا شماره رکورد جدید را نمایش دهد:

```
                // set the label...
                lblAddressNumber.Text = _currentAddressIndex + " of " +
Addresses.Items.Count;
        }
}
```

در این خاصیت از خاصیت CurrentAddress استفاده کردیم تا به سادگی به شیئ Address مربوط به رکورد جاری دسترسی پیدا کنیم. وظیفه ی این خاصیت به این صورت است که با استفاده از شماره ی رکورد جاری که در _currentAddressIndex ذخیره می شود، شیئ Address مربوط به این رکورد را از داخل لیست Addresses استخراج کرده و آن را برگرداند. اما به خاطر اینکه لیست Addresses همانند ArrayList عناصر درون خود را از صفر شماره گذاری می کند اما عناصر موجود در برنامه از یک شماره گذاری می شوند، لازم است که در برنامه از شماره اندیس رکورد جاری یک واحد کم کرده تا به رکورد مورد نظر دسترسی پیدا کنیم:

```
// CurrentAddress - property for the current address...
Address CurrentAddress
{
        get
        {
                return (Address)Addresses.Items[CurrentAddressIndex - 1];
```

```
        }
}
```

تا اینجا که همه چیز خوب پیش می رود. پس چرا برنامه با خطا مواجه شده است؟ خوب، مشکل زمانی به وجود می آید که بخواهیم از برنامه خارج شویم. در این زمان متد `FormClosed` فراخوانی شده و این متد نیز، متد `Save` از کلاس `AddressBook` را فراخوانی می کند تا داده ها را ذخیره کند.
همانطور که می دانید کلاس `XmlSerializer` برای سریالایز کردن یک شیئ در بین تمام خاصیتهای آن حرکت می کند تا به یک خاصیت خواندنی–نوشتنی برسد و سپس سعی می کند تا مقدار آن خاصیت را در فایل `XML` وارد کند. اما این کلاس فقط می تواند مقادیر مربوط به نوع های داده ای ساده مانند `System.String` و یا `System.Int32` را در فایل `XML` بنویسد. بنابراین اگر در بین خاصیتهای یک کلاس به یک شیئ پیچیده برخورد کرد، وارد آن شیئ پیچیده می شود و همین مراحل را برای آن شیئ نیز تکرار می کند. یعنی در بین خاصیتهای آن حرکت می کند تا به یک خاصیت خواندنی–نوشتنی که از نوع داده ای ساده باشد برسد. سپس مقدار آن را در فایل `XML` می نویسد. اگر در شیئ دوم نیز به یک خاصیت پیچیده ی دیگر برخورد کرد همین مراحل را تکرار می کند.
اما خوب بعضی از کلاسها دارای خاصیت هایی هستند که کلاس `XmlSerializer` نمی تواند آنها را به متن تبدیل کرده و در فایل `XML` بنویسد، و در اینجاست که این کلاس با بن بست مواجه می شود. کلاس `ArrayList` یکی از این کلاسها به شمار می رود، یعنی این کلاس دارای خاصیت هایی است که نمی توانند به متن تبدیل شوند بنابراین هنگامی که کلاس `XmlSerializer` به این خاصیت ها برسد برنامه با خطا مواجه خواهد شد. کاری که باید در اینجا انجام دهیم، این است که یک خاصیت دیگر را به گونه ای ایجاد کرده که کلاس `XmlSerializer` برای دسترسی به آدرس ها از آن استفاده کند و نخواهد که از کلاس `ArrayList` استفاده کند.

در نظر نگرفتن اعضا:

با وجود اینکه کلاس `XmlSerializer` نمی تواند بعضی از نوع های داده ای را به رشته تبدیل کرده و ذخیره کند، اما این کلاس با ذخیره کردن آرایه ها هیچ مشکلی ندارد. همچنین در قسمت قبل مشاهده کردید که کلاس `XmlSerializer` با کلاس `Address` هم هیچ مشکلی ندارد و می تواند تمام فیلد ها و خاصیتهای آن را به متن تبدیل کرده و ذخیره کند. در بخش امتحان کنید بعد، خاصیتی را ایجاد خواهیم کرد که یک آرایه از نوع `Address` را برگرداند و سپس کلاس `XmlSerializer` را مجبور خواهیم کرد که به جای استفاده از `ArrayList`، داده های موجود در این خاصیت را ذخیره کند.

امتحان کنید: در نظر نگرفتن اعضا

۱) ویرایشگر کد مربوط به کلاس `AddressBook.cs` را باز کنید و به خطی بروید که فیلد `Items` تعریف شده است. سپس تعریف این فیلد را به صورت زیر تغییر دهید:

```
        // members...
        [XmlIgnore()]
        public ArrayList Items = new ArrayList();
```

۲) حال خاصیت زیر را به کلاس AddressBook اضافه کنید:

```
// Addresses - property that works with the items
// collection as an array...
public Address[] Addresses
{
      get
      {
            // create a new array...
            Address[] addressArray = new Address[Items.Count];
            Items.CopyTo(addressArray);
            return addressArray;
      }
      set
      {
            // reset the arraylist...
            Items.Clear();
            // did you get anything?
            if (value != null)
            {
                  // go through the array and populate items...
                  foreach (Address address in value)
                  {
                        Items.Add(address);
                  }
            }
      }
}
```

۳) برنامه را اجرا کرده و سپس آن را ببندید. مشاهده خواهید کرد که همه چیز به درستی عمل می کند. مجدداً برنامه را اجـرا کرده و این مرتبه داده هایی را در فیلد های برنامه وارد کنید و سپس برنامه را ببندید. بـه ایـن ترتیـب داده هـایی کـه در برنامه نوشته بودید مشابه کد زیر در فایل AddressBook.xml قرار خواهند گرفـت (خطهـای اضـافی بـرای وضوح بیشتر حذف شده اند)

```
<AddressBook>
  <Addresses>
    <Address>
      <FirstName>Muhammad</FirstName>
      <LastName>Hashemian</LastName>
      <CompanyName>MyCompany</CompanyName>
      <Address1>11 First Avenue</Address1>
      <Address2 />
      <City>No Where</City>
      <Region>North</Region>
      <PostalCode>28222</PostalCode>
      <Country>Iran</Country>
      <Email>Muhammad@email.com</Email>
    </Address>
  </Addresses>
</AddressBook>
```

چگونه کار می کند؟

داده های XML ای که درون فایل وجود دارند ثابت می کنند که برنامه این مرتبه دیگر درست کار می کند (پس احتمالا لازم نخواهد بود عنوان بخش را به چگونه کار نمی کند تغییر دهیم). اما سوال اینجاست که چرا این مرتبه برنامه درست کار می کند؟
در این مرحله برنامه ی شما دارای دو خاصیت خواهد بود، یکی خاصیت Items و دیگری خاصیت Addresses. هر دوی این خاصیت ها خواندنی–نوشتنی هستند، بنابراین کلاس XmlSerializer هر دوی آنها را بررسی می کند تا بتواند در فایل XML ذخیره کند. خاصیت Items یک شیئ از نوع ArrayList برمی گرداند و خاصیت Addresses نیز یک آرایه از اشیایی از نوع Address.
اما در این برنامه قبل از خاصیت Items کدی را درون کروشه اضافه کردیم که همانند تعریف کلاسها در برنامه است. اما می دانید که برای ایجاد یک شیئ از کلاس کد را به این صورت نمی نویسند. پس سوالی که پیش می آید این است که این کد چیست؟ با خصیصه ها در زبان HTML و حتی XML نسبتا در طول فصلهای قبل آشنا شدیم. اما باید بدانید که کد هایی که به زبان C# نوشته می شوند نیز می توانند شامل خصیصه باشند. برای مثال هر کلاس، هر متد، هر خاصیت و یا ... در زبان C# می توانند دارای خصیصه های خاص خود باشند که اطلاعات خاصی را درباره ی آن مشخص می کند. برای تعریف یک خصیصه، باید آن را در خط قبل از عضو مورد نظر در داخل کروشه بنویسیم. در این کد نیز این خط خصیصه ای را برای فیلد Items تعریف می کند. این خصیصه مشخص کننده ی این مورد است که این فیلد نباید به وسیله ی کلاس XmlSerializer مورد بررسی قرار بگیرد. بنابراین زمانی که این کلاس به این فیلد برسد و بخواهد مقدار آن را در فایل XML وارد کند، با دیدن این خصیصه از روی این فیلد عبور کرده و به سراغ خاصیت بعدی، یعنی Addresses می رود.
قسمت get از خاصیت Addresses قسمتی است که در این کد باید به آن توجه کنیم. تمام کاری که در این قسمت انجام می دهیم این است که یک آرایه از نوع Address ایجاد کرده و سپس با استفاده از متد CopyTo از کلاس ArrayList، فیلد های موجود در ArrayList را در این آرایه قرار می دهیم:

```
// Addresses - property that works with the items
// collection as an array...
public Address[] Addresses
{
	get
	{
		// create a new array...
		Address[] addressArray = new Address[Items.Count];
		Items.CopyTo(addressArray);
		return addressArray;
	}
	set
	{
		…
	}
}
```

هنگامی که کلاس XmlSerializer به این آرایه رسید، بین تک تک عناصر آن حرکت کرده و آنها را سریالایز می کند. این مورد را در فایل XMLای که تولید شده است نیز می توانید مشاهده کنید. ابتدا کلاس XmlSerializer وارد شیئ ایجاد شده از کلاس AddressBook می شود، سپس به اولین خاصیت خواندنی–نوشتنی می رسد که می تواند داده های آن را ذخیره کند، یعنی خاصیت Addresses. این خاصیت به صورت یک آرایه است، پس کلاس سعی می کند در بین عناصر آن

حرمت کند. بنابراین از عنصر اول شروع می کند و داده های آن را در عنصر Address از فایل XML قرار می دهد. اما این آرایه فقط یک عنصر دارد. پس در اینجا کار کلاس XmlSerializer تمام شده است و خروجی آن مشابه زیر خواهد بود:

```
<AddressBook>
  <Addresses>
    <Address>
      <FirstName>Muhammad</FirstName>
      <LastName>Hashemian</LastName>
      <CompanyName>MyCompany</CompanyName>
      <Address1>11 First Avenue</Address1>
      <Address2 />
      <City>No Where</City>
      <Region>North</Region>
      <PostalCode>28222</PostalCode>
      <Country>Iran</Country>
      <Email>Muhammad@email.com</Email>
    </Address>
  </Addresses>
</AddressBook>
```

استخراج رکورد ها از فایل XML:

اگر مقداری شانس داشته باشید، برنامه هم اکنون باید بتواند هنگام اجرا شدن داده های موجود در فایل XML قبلی را استخراج کرده و در فرم نمایش دهد. بنابراین با بستن برنامه و اجرای مجدد آن باید با فرمی مشابه شکل ۱۹-۵ مواجه شوید.
دلیل این مورد در این است که زمانی که در حال نوشتن کلاس AddressBook بودید، قسمت مربوط به لود کردن داده های دفتر تلفن هنگام اجرای برنامه را نیز کامل کردید. این مرتبه که برنامه را اجرا می کنید، هنگامی که متد Load از کلاس SerializableData فراخوانی شود فایل AddressBook.xml را در دیسک پیدا می کند، بنابراین دیگر یک شیئ خالی را به عنوان نتیجه برنمی گرداند بلکه سعی می کند داده های موجود در فایل را دی سریالایز کند. خوب به خاطر اینکه کلاس XmlSerializer در نوشتن آرایه ها هیچ مشکلی ندارد می توان حدس زد که این کلاس در خواندن آنها نیز مشکلی نخواهد داشت و متد Deserialize از این کلاس بتواند به درسی کار کند.
این مرتبه دیگر قسمت set از خاصیت Addresses بسیار مهم است. تنها چیزی که در کار با این خاصیت باید مد نظر داشته باشید این است که اگر یک آرایه ی خالی را به آن ارسال کنید (مقدار null را به آن بفرستید)، خاصیت با خطا مواجه خواهد شد.

```
// Addresses - property that works with the items
// collection as an array...
public Address[] Addresses
{
      get
      {
            …
      }
      set
      {
            // reset the arraylist...
            Items.Clear();
            // did you get anything?
```

```
            if (value != null)
            {
                // go through the array and populate items...
                foreach (Address address in value)
                {
                    Items.Add(address);
                }
            }
        }
}
```

در این قسمت باید هر عضو از این آرایه را با استفاده از متد `Add` به شیئ `Items` از کلاس `ArrayList` اضافه کنید.



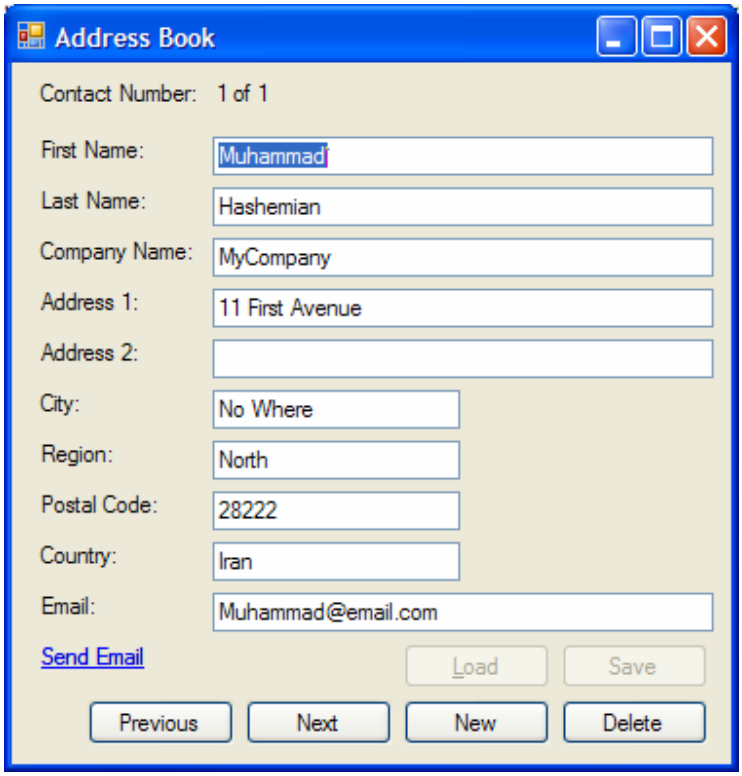
شکل ۱۹-۵

اضافه کردن رکورد های جدید:

خوب، در این قسمت مشاهده خواهیم کرد که چگونه می توان رکورد های بیشتری را به برنامه اضافه کرد. در قسمت امتحان کنید بعد، چهار کنترل `Button` جدید به فرم اضافه خواهیم کرد: دو کنترل `Button` برای حرکت در بین داده های موجود در برنامه و دو کنترل `Button` نیز برای حذف و یا اضافه کردن رکورد های جدید.

امتحان کنید: اضافه کردن رکورد های جدید

۱) قسمت طراحی فرم مربوط به `Form1` را باز کرده و دو کنترل `btnSave` و `btnLoad` را غیر فعال کنید. سپس چهار کنترل `Button` جدید همانند شکل ۱۹-۶ به برنامه اضافه کنید:



شکل ۱۹-۶

۲) خاصیت `Name` این چهار کنترل را به ترتیب برابر با `btnPrevious`، `btnNext`، `btnNew` و `btnDelete` قرار دهید. خاصیت `Text` آنها را نیز به ترتیب با مقادیر `Previous`، `Next`، `New` و `Delete` تنظیم کنید

۳) روی دکمه ی `btnNew` دو بار کلیک کنید تا متد مربوط به رویداد `Click` آن به صورت اتوماتیک ایجاد شود. سپس کد مشخص شده در زیر را به این متد اضافه کنید. همچنین متد `AddNewAddress` را نیز به کلاس `Form1` اضافه کنید:

```
private void btnNew_Click(object sender, EventArgs e)
{
      AddNewAddress();
}
public Address AddNewAddress()
{
      // save the current address...
      UpdateCurrentAddress();
      // create a new address...
      Address newAddress = Addresses.AddAddress();
      // update the display...
      CurrentAddressIndex = Addresses.Items.Count;
      // return the new address...
```

```
        return newAddress;
}
```

۴) برنامه را اجرا کنید. با کلیک کردن روی دکمه ی New یک رکورد جدید ایجاد خواهد شد و می توانید داده های خود را در آن وارد کنید. در کادرهای روی فرم یک رکورد جدید را وارد کنید و سپس برنامه را ببندید تا تغییرات ذخیره شوند.

۵) به محلی که فایل AddressBook.xml در آنجا ذخیره شده بود بروید و فایل را باز کنید تا داده های آن نمایش داده شوند. مشاهده خواهید کرد که رکورد جدید نیز در فایل قرار گرفته است.

چگونه کار می کند؟

تکمیل این قسمت از برنامه بسیار ساده بود و نکته ی خاصی نداشت. تمام کار ی که باید انجام می دادیم این بود که با فراخوانی متد AddAddress از کلاس AddressBook یک شیئ جدید از کلاس Address ایجاد کرده و سپس خاصیت CurrentAddressIndex را برابر با اندیس آن شیئ یعنی اندیس آخرین عنصر موجود در آرایه قرار دهیم. به این ترتیب متن نمایش داده شده در کنترل lblAddressNumber نیز تصحیح می شود، شماره ی رکورد را به صورت 2 of 2 نمایش داده و اجازه می دهد که داده های آن را وارد کنیم.

البته بهتر است قبل از ایجاد رکورد جدید با فراخوانی متد UpdateCurrentAddress تغییراتی را که کاربر تا این لحظه در برنامه ایجاد کرده است را ذخیره کنیم. با فراخوانی این متد، زمانی که کاربر بخواهد از برنامه خارج شود، بین رکورد ها حرکت کند و یا رکورد جدیدی را ایجاد کند می توانید مطمئن شوید که تمام تغییراتی که در داده های درون کادر ها به وجود آورده است در فایل XML ذخیره خواهند شد.

```
public Address AddNewAddress()
{
        // save the current address...
        UpdateCurrentAddress();
```

بعد از اینکه تغییرات ذخیره شد، می توانیم یک رکورد جدید ایجاد کرده و داده های آن را به کاربر نمایش دهیم (که البته هیچ داده ای در آن وجود ندارد، بنابراین تمام کادر ها در فرم خالی خواهد شد) تا تغییرات مورد نظر خود را در آن ایجاد کند:

```
        // create a new address...
        Address newAddress = Addresses.AddAddress();
        // update the display...
        CurrentAddressIndex = Addresses.Items.Count;
        // return the new address...
        return newAddress;
}
```

حرکت در بین داده ها:

حال که می توانید داده های جدید به برنامه اضافه کنید، لازم است که دکمه های Next و Previous را نیز در فرم کامل کنید تا بتوانید بین داده های موجود حرکت کرده و آنها را مشاهده کنید. در بخش امتحان کنید بعد برنامه را به گونه ای تغییر خواهیم

داد تا بتوانید بین داده های بعدی و یا قبلی موجود در ایست آدرسها در شیئ Addresses حرکت کنید. البته در اینجا نیز قبل از اینکه به رکورد بعدی و یا رکورد قبلی برویم باید داده های رکورد جاری را ذخیره کنیم تا اگر کاربر تغییراتی را در آن ایجاد کرده بود، این تغییرات از بین نروند.

امتحان کنید: حرکت در بین داده ها

۱) روی دکمه ی btnNext در فرم برنامه دو بار کلیک کنید تا متد مربوط به رویداد Click آن به صورت اتوماتیک ایجاد شود. سپس کد زیر را به این متد اضافه کنید. همچنین متد MoveNext را نیز به صورت زیر در کلاس مربوط به Form1 ایجاد کنید:

```
private void btnNext_Click(object sender, EventArgs e)
{
      MoveNext();
}

private void MoveNext()
{
      // get the next index...
      int newIndex = CurrentAddressIndex + 1;
      if( newIndex > Addresses.Items.Count)
            newIndex = 1;
      // save any changes...
      UpdateCurrentAddress();
      // move the record...
      CurrentAddressIndex = newIndex;
}
```

۲) مجدداً به قسمت طراحی فرم مربوط به Form1 برگشته و روی کنترل btnPrevious دو بار کلیک کنید. سپس کد زیر را در متد ایجاد شده وارد کنید:

```
private void btnPrevious_Click(object sender, EventArgs e)
{
      MovePrevious();
}

private void MovePrevious()
{
      // get the previous index...
      int newIndex = CurrentAddressIndex - 1;
      if( newIndex == 0)
            newIndex = Addresses.Items.Count;
      // save changes...
      UpdateCurrentAddress();
      // move the record...
      CurrentAddressIndex = newIndex;
}
```

۳) حال برنامه را اجرا کرده و روی دکمه های Previous و Next کلیک کنید. در این قسمت باید بتوانید بـین داده های موجود در فرم حرکت کنید.

چگونه کار می کند؟

تمام کاری که در این قسمت انجام دادیم این بود که با اضافه کردن و یا کم کردن مقـدار CurrentAddressIndex به جلو و یا به عقب حرکت کردیم. برای حرکت کردن به جلو کافی است که مقدار این خاصیت یک واحد افزایش داده و برای حرکت به عقب نیز کافی است که مقدار این خاصیت را یک واحد کاهش دهیم.
البته باید دقت داشته باشیم که هنگام کم کردن و یا اضافه کردن اندیس، از محدوده ها تجاوز نکنیم (یعنی سعی نکنیم کـه از اولـین رکورد نیز عقبتر برویم و یا به رکورد بعد از آخرین رکورد برویم). به همین دلیل است که بعد از تنظیم کردن مقدار اندیس، صحت آن را در یک شرط بررسی می کنیم. هنگامی که به جلو حرکت می کنیم، اگر به آخر لیست رسیده باشیم مجدداً به ابتدای آن بر خواهیم گشت و هنگامی که به عقب حرکت می کنیم اگر به اول لیست برسیم، به انتهای آن منتقل خواهیم شد.
در هر دو حالت نیز قبل از اینکه اندیس را تنظیم کنیم، با فراخوانی متد UpdateCurrentAddress تغییـرات اعمـال شده را در فایل ذخیره می کنیم:

```
private void MoveNext()
{
      // get the next index...
      int newIndex = CurrentAddressIndex + 1;
      if( newIndex > Addresses.Items.Count)
            newIndex = 1;
      // save any changes...
      UpdateCurrentAddress();
      // move the record...
      CurrentAddressIndex = newIndex;
}
```

حذف کردن داده ها از برنامه:

برای تکمیل کردن این برنامه، فقط کافی است که قابلیت حذف داده ها را نیز به آن اضافه کنیم. البته هنگام حذف یک رکـورد بایـد در نظر داشته باشیم که اگر رکوردی که باید حذف شود آخرین رکورد موجود در برنامه باشد، یک رکورد خالی به برنامه اضافه کنیم تا لیست خالی نماند. در بخش امتحان کنید بعد، کدی را به برنامه اضافه خواهیم کرد تا عمل حذف را نیز به درستی انجام دهد.

امتحان کنید: حذف یک رکورد از برنامه

۱) به قسمت طراحی فرم مربوط به Form1 بروید و روی دکمه ی btnDelete دو بار کلیک کنید. سپس کد زیـر را به متد مربوط به رویداد Click آن اضافه کنید. همچنین متد DeleteAddress را نیز به صورت زیر بـه کلاس مربوط به Form1 اضافه کنید:

```
private void btnDelete_Click(object sender, EventArgs e)
{
      // ask the user if they are ok with this?
      if( MessageBox.Show(
            "Are you sure you want to delete this address?",
            "Address Book",
            MessageBoxButtons.YesNo,MessageBoxIcon.Question) ==
                                                DialogResult.Yes )
      {
            DeleteAddress(CurrentAddressIndex);
      }
}

// DeleteAddress - delete an address from the list...
public void DeleteAddress(int index)
{
      // delete the item from the list...
      Addresses.Items.RemoveAt(index - 1);
      // was that the last address?
      if (Addresses.Items.Count == 0)
            // add a new address?
            Addresses.AddAddress();
      else
            // make sure you have something to show...
            if (index > Addresses.Items.Count)
                  index = Addresses.Items.Count;
      // display the record...
      CurrentAddressIndex = index;
}
```

۲) برنامه را اجرا کنید. حال باید بتوانید با فشار دادن دکمه ی `Delete` رکورد های مورد نظر خود را از برنامه حذف کنید. البته، اگر بخواهید آخرین رکورد موجود را حذف کنید، یک رکورد خالی به صورت اتوماتیک ایجاد شده و به لیست اضافه خواهد شد.

چگونه کار می کند؟

برنامه ای که در این فصل ایجاد کردیم، به گونه ای نوشته شده است که همواره باید اطلاعاتی را به کاربر نمایش دهد. به همین دلیل است زمانی که برای اولین بار برنامه را اجرا می کنید و هیچ فایل `AddressBook.xml` ای در دیسک وجود ندارد، یک رکورد خالی ایجاد کرده و آن را در فرم نمایش می دهیم. همچنین اگر کاربر تمام رکورد های موجود در برنامه را حذف کنید، باید به نحوی چیزی را برای نمایش دادن در فرم ایجاد کنیم.
برای اینکه رکوردی را از دیسک حذف کنیم، از متد `RemoveAt` در کلاس `ArrayList` استفاده می کنیم. این متد عنصری که اندیس آن را به عنوان پارامتر دریافت کرده است را از `ArrayList` حذف می کند.

```
// DeleteAddress - delete an address from the list...
public void DeleteAddress(int index)
{
      // delete the item from the list...
      Addresses.Items.RemoveAt(index - 1);
```

همچنین دقت کنید به علت اینکه ArrayList بر مبنای اندیس صفر کار می کند، پس برای مثال لازم است برای حذف عنصر سوم از لیست، پارامتر 2 را به متد RemoveAt ارسال کنیم.
مشکل زمانی ایجاد می شود که عنصری که حذف کرده باشیم آخرین عنصر موجود در لیست بوده باشد و چون باید همواره یک عنصر در لیست باشد تا به کاربر نمایش دهیم، یک شیئ جدید از کلاس Address ایجاد می کنیم:

```
// was that the last address?
if (Addresses.Items.Count == 0)
      // add a new address?
      Addresses.AddAddress();
```

همچنین اگر رکورد حذف شده آخرین رکورد لیست نباشد و رکورد های دیگری نیز در لیست وجود داشته باشند، باید بعد از حذف رکورد اندیس رکورد جاری را تصحیح کنیم. البته در مواردی این اندیس صحیح است و نیازی به تغییر آن وجود ندارد. برای مثال اگر پنج عضو در لیست وجود داشته باشند و عضو سوم در حال نمایش داده شدن باشد، مقدار فیلد _currentAddressIndex برابر با 3 خواهد بود. حال اگر کاربر با فشار دادن دکمه ی Delete این رکورد را حذف کند، نیازی نیست که _currentAddressIndex را تنظیم کنیم. زیرا همچنان عضوی با اندیس 3 در فرم وجود دارد تا داده های آن نمایش داده شود و اما اگر برای مثال فقط سه عضو در لیست وجود داشته باشد و مقدار _currentAddressIndex نیز برابر با 3 باشد، حذف این عنصر باعث می شود که اندیس رکورد جاری نا معتبر شود و مجبور شویم که آن را تصحیح کنیم. این حالت زمانی رخ می دهد که کاربر بخواهد آخرین عضو از لیست را حذف کند. بنابراین با استفاده از یک شرط if این مورد را نیز بررسی خواهیم کرد:

```
else
      // make sure you have something to show...
      if (index > Addresses.Items.Count)
            index = Addresses.Items.Count;
```

حال که بعد از بررسی شرایط مختلف اندیس عضوی که باید داده های آن در فرم نمایش داده شود را بدست آوردیم، آن را در خاصیت CurrentAddressIndex قرار می دهیم:

```
      // display the record...
      CurrentAddressIndex = index;
}
```

بررسی لبه ها:

در این قسمت می خواهیم یکی از تکنیکهای برنامه نویسی را هنگام تست عملکرد یک نرم افزار بررسی کنیم. هنگام نوشتن یک نرم افزار، معمولا کارهای مورد نظر ما در مرزها به درستی انجام نخواهند شد. برای مثال تصور کنید در برنامه ی خود متدی دارید که یک عدد صحیح را به عنوان ورودی دریافت می کند، اما برای اینکه این متد به درستی عمل کند این عدد باید بین اعداد ۰ تا ۹۹ باشد.
در این موارد هنگامی که الگوریتم متد خود را نوشتید و آن را برای بعضی از اعداد در بازه ی مشخص شده تست کردید، بهتر است آن را برای اعداد موجود در مرز بازه نیز تست کنید. برای مثال اعدادی مانند ۰، ۱-، ۹۹ و یا ۱۰۰ را به برنامه بدهید و مشاهده کنید

که آیا الگوریتم شما برای این اعداد نیز درست کار می کند یا نه؟ معمولا اگر متد برای یکی یا دو تا از این اعداد به درستی جواب داد، می توانید مطمئن شوید که این الگوریتم برای تمام اعداد موجود در آن بازه به درستی کار می کند.
این مشکل در متدهای MoveNext و یا MovePrevious که در برنامه ایجاد کرده بودیم نیز ممکن است به وجود آید. برای مثال تصور کنید که در برنامه ی خود ۱۰۰ رکورد اطلاعات دارید و بعد از ایجاد این دو متد، آن را برای حرکت در بین رکورد های ۱۰ تا ۲۰ بررسی می کنید و مشاهده می کنید که درست کار می کنند. به این صورت می توانید مطمئن شوید که برنامه برای حرکت بین رکورد های ۲ تا ۹۹ به درستی عمل می کند، اما اگر هنگامی که متد به رکورد ۱۰۰ رسید و کاربر مجدداً خواست به رکورد بعدی برود احتمالا با مشکل مواجه خواهید شد.

ایجاد یکپارچگی بین برنامه ی دفتر تلفن و دیگر برنامه ها:

تا اینجا برنامه هایی ایجاد کردیم که می توانستند داده های خود را در فایل XML ذخیره کرده و یا از آن استخراج کنند. همچنین در طول اجرای برنامه ها، تغییراتی که در فایل XML حاصل ایجاد می شد را نیز بررسی کردیم. بنابراین تاکنون باید درک خوبی از مفهوم XML بدست آورده باشید و بدانید که XML چگونه مورد استفاده قرار می گیرد.
در ابتدای فصل گفتیم که XML وسیله ای است که برای ایجاد یکپارچگی در بین برنامه های تجاری مورد استفاده قرار می گیرد، اما برای افرادی که در برنامه نویسی تازه کار هستند توقع نا به جایی است که انتظار داشته باشیم XML را به این مفهوم درک کرده و مورد استفاده قرار دهند. بنابراین XML برای این افراد فقط می تواند به عنوان وسیله ای برای ذخیره ی داده های برنامه مورد استفاده قرار گیرد. در ادامه ی این فصل سعی خواهیم کرد دلیل اینکه XML می تواند یک ابزار خوب برای یکپارچگی بین برنامه ها باشد را توضیح دهیم. به این منظور برنامه ای ایجاد خواهیم کرد که بتواند ساختار فایل XML مربوط به برنامه ی قبلی را درک کرده و به سادگی از داده های درون آن استفاده کند.
کاربردهای XML و نحوه ی استفاده از آن یکی از مباحث پیشرفته است، بنابراین برای یادگیری بیشتر در این مورد می توانید به کتابهایی که در این زمینه نوشته شده است رجوع کنید.

توضیح اصول یکپارچه سازی:

قبل از اینکه بتوانیم برنامه ای ایجاد کنیم تا بتواند با برنامه ی دفتر تلفن تعامل داشته باشد، بهتر است با اصول یکپارچه سازی در برنامه ها آشنا شویم. اساسا زبان XML بهترین روش برای ایجاد یکپارچگی بین برنامه ها است، زیرا کد های آن به سادگی می توانند توسط افراد و یا برنامه های دیگر خوانده شده، درک شوند و یا تغییر داده شوند. روشهای قدیمی که برای این منظور مورد استفاده قرار می گرفت عمدتا نیاز داشتند که متنی نیز همراه با آنها فرستاده شود تا ساختار داده های درون آن را توضیح دهد. بنابراین هنگامی که ساختار فایل حاوی داده ها تغییر می کرد، برنامه های قبلی دیگر نمی توانستند از فایلهای جدید استفاده کنند.
فایلهای XML به سادگی می توانند توسط افراد درک شوند. برای مثال تصور کنید که تاکنون حتی در رابطه با برنامه ای که در این فصل ایجاد کرده باشیم نیز چیزی نشنیده باشید و فایل XML زیر را به شما نشان دهند:

```
<Addresses>
  <Address>
    <FirstName>Muhammad</FirstName>
    <LastName>Hashemian</LastName>
    <CompanyName>MyCompany</CompanyName>
    <Address1>11 First Avenue</Address1>
```

```
    <Address2 />
    <City>No Where</City>
    <Region>North</Region>
    <PostalCode>28222</PostalCode>
    <Country>Iran</Country>
    <Email>Muhammad@email.com</Email>
  </Address>
</Addresses>
```

به راحتی می توانید متوجه شد که این فایل حاوی چه داده هایی است. همچنین می توانید از ابزارهای زیادی که در .NET وجود دارد برای مشاهده، تغییر و یا کار با این فایل استفاده کنید. البته با این حال نیز ممکن است در بعضی شرایط مجبور شوید که ساختار فایل را از فردی که آن را طراحی کرده است درخواست کنید، مخصوصا در مواردی که داده های مهمتری در فایل XML قرار می گیرند، اما با این وجود استفاده از فایلهای XML بسیار با معنی تر از سیستم های قدیمی است.
بعد از این که ساختار یک فایل XML را متوجه شدید می توانید داده های مورد نظر خود را به فایل اضافه کنید و یا حتی فایلهای خودتان را بر اساس آن ساختار ایجاد کنید. برای مثال در برنامه ی قبلی بعد از اینکه متوجه شدید عنصر Addresses حاوی چندین عنصر از نوع Address است که هر یک داده های مربوط به یک فرد را در دفتر تلفن نگهداری می کنند، می توانید داده های افراد جدید را خود به فایل اضافه کنید و یا برنامه ای بنویسید که بتواند بر اساس ساختار این فایل کار کند و داده ها را به وسیله ی آن برنامه کنترل کنید.
برای مثال فایل قبل را باز کنید و عنصر Address را به همراه تمام عناصر زیر مجموعه ی آن کپی کرده و یک نسخه از آن را بلافاصله بعد از اتمام تگ </Address> در همان فایل قرار دهید. سپس داده های موجود در هر عنصر را با داده های مورد نظر خود عوض کرده و فایل را ذخیره کنید. برای مثال فایل را به صورت زیر تغییر دهید:

```
<AddressBook>
  <Addresses>
    <Address>
      <FirstName>Muhammad</FirstName>
      <LastName>Hashemian</LastName>
      <CompanyName>MyCompany</CompanyName>
      <Address1>11 First Avenue</Address1>
      <Address2 />
      <City>No Where</City>
      <Region>North</Region>
      <PostalCode>28222</PostalCode>
      <Country>Iran</Country>
      <Email>Muhammad@email.com</Email>
    </Address>
    <Address>
      <FirstName>Somebody</FirstName>
      <LastName>Else</LastName>
      <CompanyName/>
      <Address1>12 First Avenue</Address1>
      <Address2 />
      <City>Big City</City>
      <Region>SE</Region>
      <PostalCode>28582</PostalCode>
      <Country>Iran</Country>
      <Email>Somebody@SomeWhere.com</Email>
    </Address>
  </Addresses>
</AddressBook>
```

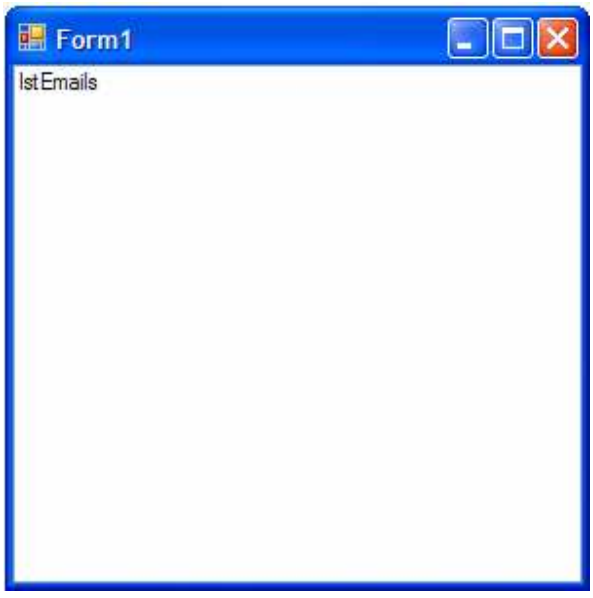
حال اگر برنامه را باز کنید، مشاهده خواهید کرد که دو رکورد اطلاعات در برنامه وجود دارند: یک رکورد که از ابتدا در برنامه ایجاد شده بود و رکورد دیگر نیز داده هایی که به صورت دستی در فالی وارد کردیم. بنابراین مشاهده کردید که با درک ساختار یک فایل XML که یک برنامه تولید می کند، می توان به سادگی تغییرات مورد نظر را در آن ایجاد کرد.

خواندن اطلاعات برنامه ی دفتر تلفن در یک برنامه ی دیگر:

برای تکمیل توضیحات این فصل، در بخش امتحان کنید بعد برنامه ای ایجاد خواهیم کرد که کاملاً از پروژه ی AddressBook جدا باشد اما بتواند از داده های درون فایل AddressBook.xml استفاده کند. این برنامه لیست افرادی که در این فایل هستند را استخراج کرده و به همراه آدرس ایمیل آنها در فرم نمایش می دهد.

امتحان کنید: خواندن داده های برنامه ی AddressBook

۱) با استفاده از ویژوال استودیو یک برنامه ی تحت ویندوز به نام Address List ایجاد کنید.
۲) با استفاده ز جعبه ابزار یک کنترل ListBox در فرم قرار دهید. خاصیت IntegralHeight این کنترل را برابر با False، خاصیت Dock آن را برابر با Fill و خاصیت Name را برابر با lstMails قرار دهید (شکل ۱۹-۷).



شکل ۱۹-۷

۳) در قسمت طراحی فرم مربوط به Form1 روی نوار عنوان فرم دو بار کلیک کنید تا متد مربوط به رویداد Load فرم به صورت اتوماتیک ایجاد شود. سپس کد مشخص شده در زیر را در این متد وارد کنید. همچنین فضای نام System.Xml را نیز به ابتدای کلاس Form1 اضافه کنید تا بتوانیم در برنامه از کلاسهای آن استفاده کنیم:

```
using System.Xml;
using System.Collections;

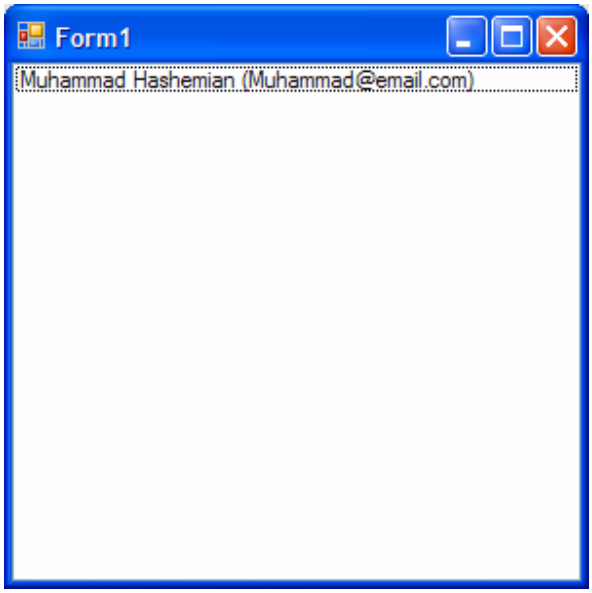
private void Form1_Load(object sender, EventArgs e)
{
      // where do we want to get the XML from...
      String filename =
            @"E:\Documents and Settings\Mohammad\My " +
            "Documents\Visual Studio 2005\Projects\" +
            "Address Book\Address Book\bin\" +
            "Debug\AddressBook.xml";
      // open the document...
      XmlTextReader reader = new XmlTextReader(filename);
      // move to the start of the document...
      reader.MoveToContent();
      // start working through the document...

      Hashtable addressData = null;
      String elementName = null;
      while(reader.Read())
      {
            // what kind of node to we have?
            switch(reader.NodeType)
            {
                  // is it the start of an element?
                  case XmlNodeType.Element:
                        // if it's an element start, is it "Address"?
                        if (reader.Name == "Address" )
                              // if so, create a new collection...
                              addressData = new Hashtable();
                        else
                        // if not, record the name of the element...
                              elementName = reader.Name;
                        break;
                  // if we have some text, try storing it in the
                  // collection...
                  case XmlNodeType.Text:
                        // do we have an address?
                        if (addressData != null)
                              addressData.Add(elementName,
                                          reader.Value);
                        break;
                  // is it the end of an element?
                  case XmlNodeType.EndElement:
                        // if it is,
                        // we should have an entire address stored...
                        if( reader.Name == "Address")
                        {
                              // try to create a new listview item...
                              String item = null;
                              try
                              {
                                    item = addressData["FirstName"] +
                                    " " + addressData["LastName"]
                                    + " (" +
```

```
                                    addressData["Email"] + ")";
                            }
                            catch
                            {
                            }
                            // add the item to the list...
                            lstEmails.Items.Add(item);
                            // reset...
                            addressData = null;
                        }
                        break;
                }
            }
}
```

نکته: قبل از اجرا این برنامه مسیری که برای فایل `AddressBook.xml` در ابتدای متد `Form1_Load` وارد شده است را با مسیر صحیح آن جایگزین کنید.

۴) با اجرای برنامه باید فرمی مشابه فرم ۱۹-۸ مشاهده کنید. همچنین اگر رکوردی دارای آدرس ایمیل نباشند نیز برنامه به درستی آن را نمایش می دهد زیرا در این صورت یک رشته ی تهی در فایل XML ذخیره می شود و هنگام خواندن داده ها نیز یک رشته ی تهی به وسیله ی `ListView` نمایش داده می شود.



شکل ۱۹-۸

چگونه کار می کند؟

قبل از اینکه درباره ی مزایای این برنامه (و یا مزایای XML) صحبت کنیم، تصور کنید که قبل از نوشتن این برنامه اصلا فایل XML ای که به وسیله ی پروژه ی `AddressBook` ایجاد می شد را ندیده بودید و با ساختار آن آشنایی نداشتید، فقط می

خواستید بر اساس اطلاعات داخل این فایل برنامه ای بنویسید که نام و آدرس ایمیل افراد را نمایش دهد. در این صورت با توجه به اینکه XML به صورت یک فایل متنی ذخیره می شود، می توانستید آن را با یک ویرایشگر متن معمولی باز کرده و آن را بخوانید. به این ترتیب متوجه می شوید که این فایل حاوی عناصری از نوع Address است که هر کدام از این عنصر ها دارای عنصرهای دیگری است که داده های مربوط به یک فرد را ذخیره می کنند. سه عنصر Email، FirstName و LastName نیز در بین این داده ها وجود دارند که در برنامه ی Address List به آنها نیز دارید. بنابراین تنها چیزی که باقی می ماند خارج کردن اطلاعات داخل عناصر از این فایل است.

هنگام معرفی .NET یکی از عجیب ترین کارهایی که به وسیله ی مایکروسافت انجام گرفت، پشتیبانی شدید چارچوب .NET از XML بود. در چارچوب .NET تعداد چشمگیری کلاس برای خواندن و یا نوشتن فایلهای XML وجود داشت. در برنامه ی قبلی فقط از کلاس XmlSerializer استفاده کردیم که یکی از ساده ترین کلاسهای .NET برای کار با XML بود. ویژگی این کلاس در این است که برای تولید فایل XML فقط به ساختار کلاس شما استناد می کند. اما اگر یک فایل XML را از یک برنامه ی دیگر دریافت کنید و بخواهید از آن استفاده کنید، نمی توانید این کلاس را به کار ببرید زیرا کلاسی در برنامه ی خود ندارید که ساختار آن مشابه ساختار فایل XML مورد نظر شما باشد. بنابراین باید از کلاس دیگری برای خواندن و نوشتن داده های XML در این گونه فایلها استفاده کنید.

در برنامه ی Address List نیز شرایط مشابه ی برقرار است. در این برنامه کلاسی به نام Address و یا AddressBook نداریم که از کلاس XmlSerializer بخواهیم داده ها را در اشیایی از این کلاسها قرار دهد. بنابراین باید با استفاده از کلاسهای دیگری، فایل XML را به صورت خط به خط بررسی کنیم. یکی از کلاس هایی که می توانیم برای این مورد استفاده کنیم، کلاس System.Xml.XmlTextReader است. این کلاس یک فایل XML را باز کرده و به ابتدای آن اشاره می کند. سپس می توانیم از آن بخواهیم که قسمت قسمت در فایل حرکت کند (هر قسمت یک گره یا node نامیده می شود). این کلاس در فایل حرکت کرده و در ابتدای هر قسمت توقف می کند، برای مثال در ابتدای تگهای شروع، در ابتدای تگهای پایان، در ابتدای داده های درون عناصر، در ابتدای خصیصه ها و ....

بنابراین زمانی که از این کلاس می خواهید که در فایل حرکت کند، ابتدا گره ی زیر را به شما نمایش می دهد:

```
<?xml version="1.0"?>
```

هنگامی که از او بخواهید به خط بعدی برود، کلاس به این خط از فایل می رسد:

```
<AddressBook xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xmlns:xsd="http://www.w3.org/2001/XMLSchema">
```

سپس، مجدداً از کلاس می خواهید که به خط بعدی برود و کلاس نیز در ابتدای این خط توقف می کند:

```
  <Addresses>
```

به همین ترتیب کلاس XmlTextReader اطلاعاتی را در رابطه با <Address>، <FirstName>Muhammad</FirstName>، <LastName> به شما می دهد تا به انتهای فایل XML برسد. این کلاس حتی فضاهای خالی موجود در فایل XML را نیز به شما اطلاع می دهد که رویهمرفته می توانید از این اطلاعات اضافی صرفنظر کنید.

الگوریتمی که باید در این قسمت به کار ببریم به این صورت است که یک شیئ از کلاس XmlTextReader ایجاد کرده و به وسیله ی آن در قسمتهای مختلف فایل حرکت می کنیم. هنگامی که بخواهیم حرکت را شروع کنیم باید متد Read از این کلاس را فراخوانی کنیم. به این ترتیب، این شیئ به اولین گره ی درون فایل اشاره می کند و با هر بار فراخوانی مجدد متد Read

نیز شیئ یک گره درون فایل به جلو حرکت خواهد کرد. بنابراین فراخوانی متد Read را درون یک حلقه ی while قرار می دهیم تا به این وسیله بین تک تک گره های موجود در فایل حرکت کرده و آنها را بررسی کنیم.

```
private void Form1_Load(object sender, EventArgs e)
{
	// where do we want to get the XML from...
	String filename =
		@"E:\Documents and Settings\Mohammad\My " +
		"Documents\Visual Studio 2005\Projects\" +
		"Address Book\Address Book\bin\" +
		"Debug\AddressBook.xml";
	// open the document...
	XmlTextReader reader = new XmlTextReader(filename);
	// move to the start of the document...
	reader.MoveToContent();
	// start working through the document...

	Hashtable addressData = null;
	String elementName = null;
	while(reader.Read())
	{
```

برای اینکه بفهمیم گره ای که هم اکنون در ابتدای آن قرار داریم چه نوع گره ای است، می توانیم از خاصیت NodeType از کلاس XmlTextReader استفاده کنیم. اگر نوع گره برابر با Element بود به این معنی است که در ابتدای تگ شروع یک عنصر قرار داریم. برای اینکه نام عنصری که در ابتدای آن قرار داریم را نیز بدست آوریم نیز می توانیم از خاصیت Name استفاده کنیم. به این ترتیب اگر نام عنصری که به آن رسیدیم برابر با Address بود، یعنی در ابتدای اطلاعات مربوط به یک فرد جدید هستیم. پس یک شیئ از نوع HashTable ایجاد می کنیم تا داده های مورد نیاز از این فرد جدید را در آن قرار دهیم. در غیر این صورت نیز نام عنصر را در یک متغییر ذخیره می کنیم تا بعدها از آن استفاده کنیم:

```
// what kind of node to we have?
switch(reader.NodeType)
{
	// is it the start of an element?
	case XmlNodeType.Element:
		// if it’s an element start, is it “Address”?
		if (reader.Name == "Address" )
			// if so, create a new collection...
			addressData = new Hashtable();
		else
		// if not, record the name of the element...
			elementName = reader.Name;
		break;
```

همچنین ممکن است عنصری که در آن قرار داریم شامل مقداری متن باشد. در این صورت بررسی می کنیم که آیا متغییر addressData مقدار دهی شده است یا نه (دارای یک شیئ از نوع HashTable است یا نه)؟ در صورتی که مقدار دهی شده بود، به این معنی است که هم اکنون درون داده های مربوط به یک فرد (در عناصر درونی عنصر address) هستیم. همچنین به خاطر داریم که در قسمت قبل، نام عنصری که درون آن هستیم را نیز در متغییر elementName ذخیره کردیم. پس اگر برای مثال عنصر elementName برابر با FirstName باشد به این معنی است که متنی که در ابتدای آن

قرار داریم، مربوط به خاصیت FirstName یکی از رکورد های این فایل است. سپس این اطلاعات را بـه HashTable اضافه می کنیم تا در انتها که لیست داده های مربوط به رکورد جاری تکمیل شد از آن استفاده کنیم.

```
// if we have some text, try storing it in the
// collection...
case XmlNodeType.Text:
      // do we have an address?
      if (addressData != null)
            addressData.Add(elementName, reader.Value);
      break;
```

به همین ترتیب که حلقه ی while اجرا شده و کلاس XmlTextReader در بین داده های درون فایـل حرکـت مـی کند، داده های مربوط به یک رکورد نیز درون HashTable قرار می گیرد تا به انتهای رکـورد جـاری، یعنـی بـه تـگ پایـانی مربوط به address برسیم.
برای اینکه بدانیم آیا به تگ </address> رسیده ایم، می توانیم بررسی کنیم که آیا نوع گره برابر بـا EndElement است یا نه؟

```
// is it the end of an element?
case XmlNodeType.EndElement:
```

اگر نوع گره ای که به آن رسیده ایم از نوع تگ پایانی و نام آن نیز برابر با Address بـود، بـه ایـن معنـی اسـت کـه بـه تـگ </address> رسیده ایم و حال دیگر تمام اطلاعات مربوط به رکوردی که در آن قرار داشـتیم در HashTable قـرار گرفته است.بنابراین رشته ای که باید در ListBox نمایش داده شود را ایجاد می کنیم:

```
// if it is, we should have an entire address stored...
if( reader.Name == "Address")
{
      // try to create a new listview item...
      String item = null;
      try
      {
            item = addressData["FirstName"] + " " +
                  addressData["LastName"] + " (" +
                  addressData["Email"] + ")";
      }
      catch
      {
      }
      // add the item to the list...
      lstEmails.Items.Add(item);
      // reset...
      addressData = null;
}
break;
```

توجه کنید که بلاک catch که بعد از بلاک try قرار گفته است، در صورت رخ دادن خطا هیچ عمـل خاصـی را انجـام نمـی دهد. در این مثال برای اینکه درگیر خطا یابی نشویم، هر گونه خطایی که رخ دهد را در نظر نخواهیم گرفت. برای مثـال اگـر عنـصر

درونی که در حال بررسی آن هستیم بعضی از تگ ها را نداشته باشد، برای مثال دارای تگ email نباشد، خطایی رخ خواهد داد که از آن صرفنظر می کنیم.
به همین ترتیب حلقه ادامه پیدا کرده و بین تمام گره هایی که در فایل وجود دارند حرکت می کند تا به انتهای فایل برسد. در این زمان تمام داده های فایل بررسی شده اند و فراخوانی متد Read نیز مقدار false را برمی گرداند که از اجرای مجدد حلقه جلوگیری می کند و به این ترتیب اجرای حلقه متوقف می شود.
امیدوارم که این مثال به اندازه ی کافی توانسته باشد قدرت XML در یکپارچه سازی برنامه ها را توضیح دهد. اگر می خواهید با تمرین بیشتر روی این مثال، تجربه ی خود را در کار با XML افزایش دهید، سعی کنید قابلیت حذف و یا اضافه کردن داده های جدید را نیز به برنامه ی Address List اضافه کنید.

نتیجه:

در این فصل با مفاهیم XML آشنا شدیم. XML زبانی است که بیشتر برای ایجاد یکپارچگی بین برنامه ها و نرم افزارهای تجاری موجود مورد استفاده قرار می گیرد. در یک سازمان می توان از XML برای تبادل و انتقال داده ها در بین برنامه های مختلف استفاده کرد. در بین چند سازمان مختلف نیز می توان یک فرمت خاص XML را تعریف کرده و سپس برنامه ها را به گونه ای تنظیم کرد تا بتوانند داده های خود را بر اساس این فرمت انتقال دهند و به این ترتیب برنامه های این چند سازمان با یکدیگر تعامل داشته باشند. همچنین به دلیل اینکه فایلهای XML به صورت متن ذخیره می شوند، می توان آنها را به سادگی از طریق اینترنت و با استفاده از تکنولوژی هایی مانند HTTP، FTP، Email و ... منتقل کرد و دیگر نیازی نیست که چند سازمانی که می خواهند به این روش با هم تبادل اطلاعات داشته باشند از برقراری اتصالات اختصاصی استفاده کنند.
XML عمدتا برای برقراری یکپارچگی بین نرم افزارهای تجاری که روی پلت فرم های گوناگون کار می کنند ایجاد شده است، اما برای افراد تازه کار XML عمدتا برای ذخیره ی داده های برنامه مورد استفاده قرار می گیرد. در این فصل نیز با نحوه ی استفاده از XML برای ذخیره ی داده ها آشنا شدیم و مشاهده کردیم که چگونه می توان داده های یک کلاس را سریالایز کرده و سپس در یک فایل در دیسک ذخیره کرد و یا داده های یک کلاس که در دیسک ذخیره شده است را دی سریالایز کرده و کلاس متناظر آن را در برنامه ایجاد کرد. به این ترتیب با استفاده از همین موارد برنامه ای به نام Address Book ایجاد کردیم که از فایل XML به عنوان منبع اصلی برای نگهداری داده های خود استفاده می کرد.
در آخر فصل نیز، برای اینکه مشاهده کنیم چگونه XML می تواند در بین برنامه ها یکپارچگی ایجاد کند، برنامه ای نوشتیم که از فایل XML تولید شده به وسیله ی Address Book استفاده می کرد و داده های آن را در فرم نمایش می داد.
در پایان این فصل باید با موارد زیر آشنا شده باشید:

- درک بهتری از XML بدست آورده باشید و بدانید که این تکنولوژی در چه مواردی استفاده می شود.
- بتوانید داده های یک کلاس را سریالایز و دی سریالایز کنید.
- بتوانید داده های XML را در برنامه مورد استفاده قرار دهید.
- بتوانید با استفاده از کلاس XmlTextReader در یک فایل XML حرکت کرده و داده های مورد نیاز خود را از آن استخراج کنید.

تمرین:

تمرین ۱:

یک فایل XML ایجاد کنید که یک لامپ را توصیف کند. برای توصیف یک لامپ می توانید از چندین روش مختلف استفاده کنید. برای مثال می توانید عناصر مربوط به ظاهر حباب لامپ، اطلاعات فنی لامپ و نیز مشخصات خود لامپ مانند قیمت و ... استفاده کنید. برای بررسی صحت فایل XML ای که ایجاد کرده اید می توانید به نمونه هایی که در اینترنت و در سایتهایی مانند http://www.w3schools.com/dom/dom_validate.asp وجود دارد رجوع کنید.

تمرین ۲:

برای این تمرین می خواهیم مطالبی را که در طول فصل آموختیم، با یادگیری نحوه ی قرار دادن توضیحات در فایلهای XML افزایش دهیم. به عنوان کسی که به تازگی برنامه نویسی را آغاز کرده است، مهمترین توانایی که باید بدست آورید نحوه ی جستجو در اینترنت و یافتن پاسخهای مورد نیازتان است. در این تمرین با استفاده از موتور جستجوی مورد نظر خود اینترنت را جستجو کرده و نحوه ی قرار دادن توضیحات در فایل XML را پیدا کنید. سپس توضیحات لازم برای فایل XML ای که در قسمت قبل ایجاد کرده بودیم را در آن بنویسید.

فصل بیستم: وب سرویس ها و .NET Remoting.

صاحبنظران صنعتی پیشبینی می کنند که وب سرویس ها رخداد عظیم بعدی خواهد بود که در اینترنت به وجود خواهد آمد. در این فصل سعی می کنیم ابتدا وب سرویس ها را معرفی کرده و سپس نحوه ی ایجاد آنها را بررسی کنیم. همچنین سعی می کنیم در طول فصل با .NET Remoting نیز آشنا شویم و نکاتی را در مورد اینکه در چه شرایطی باید از وب سرویس ها و در چه شرایطی باید از .NET Remoting استفاده کنیم.
در این فصل:

- با SOAP آشنا خواهیم شد، روشی که برای انتقال اطلاعات در وب سرویس ها مورد استفاده قرار می گیرد.
- چندین وب سرویس ایجاد خواهیم کرد.
- عملکرد وب سرویس ها را تست خواهیم کرد.
- برنامه هایی ایجاد خواهیم کرد که از وب سرویس ها استفاده کنند.
- نگاه مختصری بر .NET Remoting و موارد استفاده ی آن خواهیم داشت.

وب سرویس چیست؟

معمولا بیشتر استفاده ای که ممکن است از اینترنت داشته باشید، ارسال و دریافت ایمیل و یا گردش کردن در وب است. این دو کاربرد معمولا بیشترین دلیلی است که افراد برای آن به اینترنت متصل می شوند. اما با رشد اینترنت، نحوه ی استفاده ی افراد از اینترنت نیز در حال تغییر کردن است.
با رشد اینترنت، برنامه هایی که برای انجام کارهای خود به اتصال به اینترنت نیاز دارند نیز در حال افزایش است. بیشتر برنامه های امروزی برای اینکه بتوانند داده هایی را به سرور خود بفرستند و یا اطلاعات جدید را از سرور دریافت کنند نیاز دارند که به اینترنت متصل باشند. همین مورد موجب شده است که وب سرویس ها به یکی از بزرگترین تحولات اخیر در زمینه ی اینترنت تبدیل شوند. حتی می توانیم بگوییم به همان اندازه که وب و صفحات وب در مدت اخیر رشد داشته اند، وب سرویس ها نیز رشد خواهند کرد و فراگیر خواهند شد. اما خوب ممکن است سوال کنید که چرا وب سرویس ها تا این حد از اهمیت برخوردار اند؟
همانطور که می دانید صفحات وب یکی از بهترین راهها برای به اشتراک گذاشتن اطلاعات است. ما مشکلی که صفحات وب دارند و یا به عبارت بهتر می توان گفت محدودیتی که این صفحات دارند این است که فقط انسانها می توانند از آن استفاده کنند! صفحات وب حتماً باید به وسیله ی انسانها خوانده شود و اطلاعات درون آن نیز فقط می تواند به وسیله ی ذهن انسان درک شود. اما وب سرویس ها، از ابتدا برای خوانده شدن و تفسیر شدن به وسیله ی برنامه های کامپیوتری ایجاد شده اند و نمی توانند به وسیله ی انسانها مورد استفاده قرار بگیرند. در حقیقت می توانیم بگوییم که وب سرویس ها، سایتهای وبی هستند که فقط به وسیله ی برنامه های کامپیوتری می توانند مورد استفاده قرار بگیرند. وب سرویس ها به طور ذاتی می توانند به صورت دینامیک تغییر کنند، بنابراین لازم نیست که حاوی اطلاعات ثابت و تغییر ناپذیری باشند، بلکه می توانند با برنامه ای که از آنها استفاده می کند تعامل داشته باشند و ارتباط برقرار کنند. برای مثال من می توانم در برنامه ای که می نویسم از یک وب سرویس استفاده کنم که در مواقع مورد نیاز کمیتی را به واحد دلار از برنامه ی من دریافت کرده و مقدار معادل آن را به واحد یورو برگرداند.
خوب ممکن است سوال کنید که این مورد چه فایده ای دارد و چگونه می تواند مفید واقع شود؟ همانطور که در فصل قبل نیز در این باره صحبت کردیم، یکی از پرهزینه ترین موارد در یک سیستم تجاری، برقراری یکپارچگی بین نرم افزارهای تجاری موجود است. برای مثال تصور کنید که شرکت تجاری شما دارای نرم افزاری برای کنترل موجودی کالاها در انبار و نرم افزار دیگری برای دریافت

سفارشات مختلف از مشتریان است. این دو نرم افزار از دو شرکت برنامه نویسی مختلف در دو زمان جدا از هم خریداری شده اند و هر کدام از آنها نیز بر روی یک پلت فرم خاص کار می کنند. اما زمانی که یک سفارش به وسیله ی یک خریدار در نرم افزار دوم ثبت می شود، این نرم افزار باید به نرم افزاری که مسئول کنترل موجودی انبار است اطلاع دهد که چه مقدار و از چه کالایی فروخته شده است. سپس نرم افزار اول نیز می تواند بر اساس این اطلاعات کارهای خاصی را به صورت اتوماتیک انجام دهد. برای مثال در صورتی که موجودی آن کالا کاهش زیادی پیدا کرده بود، مجدداً از آن کالا سفارش دهد و یا به فرد خاصی اطلاع دهد که کالای فروخته شده را به آدرس خریدار ارسال کند.

هنگامی که این دو نرم افزار به این صورت با یکدیگر کار کنند و تعامل داشته باشند، می گوییم که این دو نرم افزار یکپارچه شده اند. یکپارچه سازی به ندرت به سادگی انجام می شود و مخصوصا در مورد شرکتهای بزرگ نیاز است که تیمی از مشاوران و متخصصان استخدام شوند و بعد از صرف چندین هزار دلار، نرم افزاری برای برقراری یکپارچگی بین نرم افزارها نوشته شود.

حال اگر نخواهیم زیاد وارد جزئیات این حالتها شویم، وب سرویس ها باعث می شوند که یکپارچگی بسیار بسیار ساده تر انجام شود و بنابراین هزینه ای که برای این کار صرف می شود نیز بسیار بسیار کمتر از قبل خواهد بود. به همین دلیل است که پیش بینی می کنند وب سرویس ها بزرگترین تحول اینترنتی در مدت اخیر خواهد بود. با استفاده از وب سرویس ها نه تنها شرکتهایی که می خواهند در نرم افزارهای خود یکپارچگی ایجاد کنند راههای بسیار ساده تر و ارزان تری را در اختیار خواهند داشت، بلکه نرم افزارهای موجود در شرکتهای تجاری کوچک نیز می توانند به سادگی با یکدیگر رابطه داشته باشند.

توضیح در مورد مزایا و معایب وب سرویس ها و نیز صحبت درباره ی افرادی که این تکنولوژی را گسترش می دهند از اهداف این کتاب خارج است. اما برای اطلاعات بیشتر در این زمینه می توانید به آدرس

http://msdn.microsoft.com/webservices

مراجعه کنید.

نحوه ی عملکرد وب سرویس ها:

اول از همه باید بگویم که اساس وب سرویس ها به طور کلی روی استانداردهای آزاد است و به هیچ پلت فرم و یا شرکت خاصی تعلق ندارد. یکی از جذابیت ها و دلایل موفقیت وب سرویس ها نیز در این مورد است که تفاوتی نمی کند که شما وب سرویس خود را روی چه پلت فرمی ارائه می کنید، ویندوز، مکینتاش، لینوکس، سولاریس، یونیکس و ... در هر حالت همه ی افراد می توانند به سرور شما متصل شده و از وب سرویسی که ارائه می دهید استفاده کنند. این مورد دقیقاً مشابه عملکرد سایتهای وب است. در سایتهای وب نیز تفاوتی ندارد که سروری که این سایت روی آن قرار گرفته است و یا پلت فرمی که به وسیله ی آن نوشته شده است چیست و یا چگونه کار می کند. در هر صورت شما می توانید به سادگی به آن سایت متصل شده و از اطلاعات آن استفاده کنید.

دومین نکته ای که باید بدانید این است که وب سرویسی که در .NET استفاده شده و به کار گرفته می شود تماما بر اساس مدل برنامه نویسی است که اغلب برنامه نویسان به استفاده از آن علاقه ی زیادی دارند: برنامه نویسی شیئ گرا. اگر شما نیز در اغلب برنامه های خود از اشیا استفاده می کنید (که البته تا فصل بیستم این کتاب، دیگر باید بتوانید این کار را انجام دهید) در .NET نیز به سادگی می توانید وب سرویس ها را مورد استفاده قرار دهید.

اصلی که برای ایجاد یک وب سرویس مورد استفاده قرار می گیرد به این صورت است که یک کلاس ایجاد می کنید که دارای متدهای مختلفی است. البته نحوه ی توزیع و استفاده از این کلاس مانند کلاسهای قبلی نیست و تفاوت دارد. در رابطه با کلاس هایی که تا اینجا ایجاد کرده ایم، نحوه ی استفاده از یک کلاس به صورت زیر بود:

- یک برنامه نویس، یک کلاس را ایجاد می کرد.
- این کلاس در جایی نصب می شد (در کامپیوتری که می خواست مورد استفاده قرار گیرد کپی می شد).

- قسمتی از یک نرم افزار *در همان کامپیوتر* که می خواست از این کلاس استفاده کند، یک نمونه از ایـن کـلاس را ایجـاد می کرد (یک "شیئ" ایجاد می کرد).
- آن قسمت از نرم افزار که این شیئ را ایجاد کرده بود، متد مورد نظر خود را از این کلاس فراخوانی می کرد.
- آن متد از شیئ، کارهای خاصی را انجام می داد و مقداری را به عنوان نتیجه برمی گرداند.
- آن قسمت از نرم افزار که متد را فراخوانی کرده بود، نتیجه را دریافت کرده و از آن استفاده می کرد.

اما در وب سرویس ها یک کلاس به صورت زیر مورد استفاده قرار می گیرد:

- یک برنامه نویس یک کلاس را ایجاد می کند.
- آن کلاس روی یک سرور که دارای یک وب سرور مانند IIS و یا هر وب سرور دیگری است کپی می شود.
- قسمتی از یک نرم افزار که *در یک کامپیوتر متفاوت و با فاصله از کامپیوتری که کلاس در آن قرار دارد*، (معمولا در جایی در اینترنت) از وب سرور می خواهد که یکی از متدهای موجود در کلاس را اجرا کند.
- سرور یک نمونه از کلاس (یک شیئ) را ایجاد کرده و متد درخواست شده را فراخوانی می کند.
- سرور نتیجه ی اجرای متد را به کامپیوتری که آن را فراخوانی کرده بود برمی گرداند.
- آن قسمت از نرم افزار در کامپیوتر دوردست که درخواست فراخوانی متد را ایجاد کرده بود، نتیجه را دریافت کـرده و از آن استفاده می کند.

مشاهده می کنید که روش کار در هر دو مورد مشابه است، اما در مورد دوم یک گسستگی بین کامپیوتری که کلاس به طـور واقعـی در آن قرار دارد و کامپیوتری که می خواهد از کلاس استفاده کند وجود دارد. در حقیقت با استفاده از وب سرویس ها یـک فاصـله ی پردازشی زیادی (به اندازه ی وسعت اینترنت)، بین نرم افزاری که می خواهد از کلاس استفاده کند و خود کلاس به وجـود مـی آیـد. برای حل مشکل این گسستگی و کنترل فاصله ای که در اینجا وجود دارد، از تکنولوژیها و استاندارد هـایی کـه در وب سـرویس هـا مورد استفاده قرار گرفته است (و یا حتی در اصل برای استفاده به وسیله ی وب سرویس ها ایجاد شده اند) استفاده می کنند.

:SOAP

همانطور که در ابتدای فصل گفتم "وب سرویس ها در حقیقت وب سایتهایی برای استفاده به وسیله ی نرم افزارها هستند"، بنابراین از همان تکنولوژی استفاده می کنند که باعث شده است سایتهای وب تا این حد عمومی شـوند. وب سـرویس هـا نیـز ماننـد وب از استاندارد HTTP[1] استفاده می کنند که توسط همه ی سرورهای وب به کار گرفته شده است.
هنگامی که با "ایجاد سایتهایی برای انسانها" سروکار داریم، معمولا کلاینت (مرورگر) و سرور فایلهای مختلفی را با یکدیگر مبادلـه می کنند: فایلهایی متنی حاوی کد HTML، DHTML، JavaScript و ... که ظاهر و متنهای موجود در صفحه را شـامل می شوند، فایلهای تصویر و یا صدا با فرمت های JPEG، GIF و یا ... که در قسمتهای مختلف صفحه مورد اسـتفاده قـرار مـی گیرد و غیره.
اما زمانی که بخواهید برای نرم افزارها و برنامه های کامپیوتری سایتی را ایجاد کنید، فقط با یک نوع فایـل در ارتبـاط هـستید. ایـن فایلها به نام فایلهای SOAP معروف هستند.

[1] Hyper Text Markup Language

نکته: SOAP در اصل سر نام کلمات Simple Object Access Protocol است، اما استاندارد کنونی که در W3C وجود دارد این اصطلاح را حذف کرده است.

هنگامی که یک برنامه بخواهد از یک سرویس وب تقاضای دریافت اطلاعاتی را بکند، برای مثال بخواهد موجودی یک کالا در انبار را بداند، یا بخواهد وضعیت کنونی یک سفارش را دریافت کند، و یا از کامپیوتر سرور بخواهد تا کار خاصی را مثل یک تبدیل واحد برای او انجام دهد، برنامه یک فایل درخواست با قالب SOAP ایجاد می کند. سپس این فایل با استفاده از HTTP و از طریق اینترنت به سروری که وب سرویس در آن قرار دارد فرستاده می شود. این فایل حاوی تمام اطلاعاتی است که وب سرویس نیاز دارد تا بداند چه کاری از او خواسته شده است. با توجه به اینکه وب سرویس ها مانند روش کلاس/متد معمولی کار می کنند، این فایل SOAP معمولا حاوی نام متدی است که باید اجرا شود و نیز پارامترهایی که این متد به عنوان ورودی باید دریافت کند.
در سمت سرور، این تقاضا به وسیله ی وب سرویس دریافت شده و بعد از اینکه در خواست کاربر از آن استخراج شد، متد مورد نظر فراخوانی می شود (در ادامه ی فصل چنین کلاس هایی را ایجاد کرده و از آنها در برنامه استفاده می کنیم). بعد از اینکه متد مورد نظر اجرا شده و نتیجه آماده شد، وب سرویس یک فایل SOAP ایجاد کرده و نتیجه ی درخواست برنامه را در آن قرار می دهد و به سمت کامپیوتر درخواست کننده ارسال می کند. همانند فایلی که حاوی تقاضا بود، این فایل نیز که نتیجه را نگهداری می کند با استفاده از HTTP و به وسیله ی اینترنت به برنامه ی اول فرستاده می شود.
اسناد و فایلهای SOAP با استفاده از XML ایجاد می شوند. به عبارت دیگر فایلهای SOAPای که برای این منظور مورد استفاده قرار می گیرند بسیار شبیه فایلهای XML ای هستند که در فصل قبل مشاهده کردید. البته در این سطح که در این کتاب وب سرویس ها را بررسی می کنیم، نیازی به مشاهده ی فایلهای SOAP نخواهیم داشت. با وجود این در طی این فصل هنگامی که بخواهیم از وب سرویس ها استفاده کرده و نتایجی را از آنها دریافت کنیم، نمونه هایی از SOAP را که به وسیله ی سرور برگشت داده می شود مشاهده خواهیم کرد، اما کاری با فایلهای SOAP حاوی تقاضا ها نخواهیم داشت.
با توجه به اینکه استفاده از وب سرویس ها به پلت فرم خاصی وابسته نیست و هر نرم افزاری که در هر پلت فرمی طراحی شده باشد می تواند از هر وب سرویس که به هر نحوی ایجاد شده باشد استفاده کند، برنامه نویسان معمولا برای انتخاب یک پلت فرم به این نکته دقت می کنند که در هر پلت فرم چگونه می توان فایلهای SOAP مربوط به استفاده از یک وب سرویس را ایجاد کرد و یا از فایلهای حاوی نتیجه استفاده کرد، و یا این نکته را در نظر می گیرند که در هر پلت فرم چه ابزارهایی برای طراحی وب سرویس مورد نظرشان وجود دارد. .NET در هر دو زمینه تا حد ممکن سادگی و قدرت را برای طراحی و یا استفاده از یک وب سرویس فراهم کرده است؛ با استفاده از .NET می توان بدون اینکه درگیر فایلهای SOAP شد از وب سرویس ها استفاده کرد و آنها را در برنامه به کار برد (به همین دلیل است که در این فصل زیاد وارد جزئیات SOAP نمی شویم، زیرا با وجود اینکه هیچ کاری در وب سرویس ها بدون استفاده از SOAP ممکن نیست در این فصل بدون درگیر شدن در SOAP تمام کارهای لازم را انجام خواهیم داد).
با وجود اینکه وب سرویس ها به هیچ پلت فرم خاصی وابسته نیستند، اما در این فصل روی نحوه ی ایجاد وب سرویس ها در پلت فرم .NET صحبت خواهیم کرد. البته ابتدا بهتر است به شکل ۲۰-۱ نگاهی بیندازید که معماری عملکرد یک وب سرویس را توضیح می دهد.

شکل ۲۰-۱

ایجاد یک وب سرویس:

با وجود اینکه ممکن است وب سرویس ها در ابتدا مبحثی پیچیده به نظر برسند، اما ایجاد یک وب سرویس با استفاده از ویژوال استودیو ۲۰۰۵ بسیار ساده است. در این قسمت یک وب سرویس ایجاد خواهیم کرد و با مفاهیم مربوط به این کار نیز بیشتر آشنا خواهیم شد. همچنین مشاهده خواهید کرد که برای اینکه یک متد را به گونه ای ایجاد کنید که بتواند در یک وب سرویس مورد استفاده قرار بگیرد از چه خصیصه هایی باید استفاده کرد. همچنین با روش بررسی عملکرد یک متد را در یک وب سرویس نیز آشنا خواهیم شد.

ایجاد یک وب سرویس ساده:

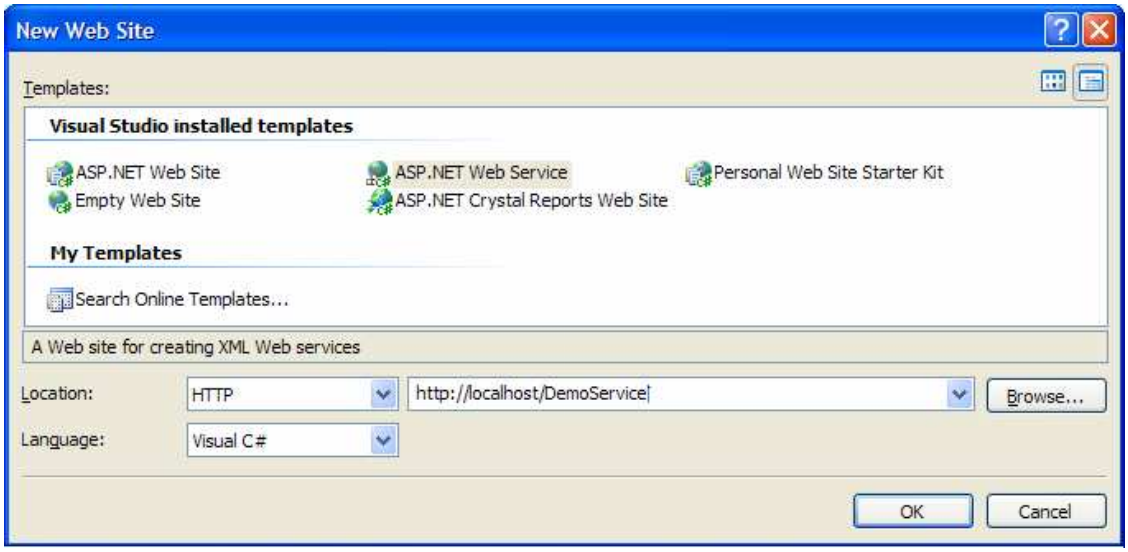
یک وب سرویس در حقیقت همانند یک کلاس است که در سرور قرار می گیرد. بعضی از متدهای این کلاس به گونه ی خاصی تعریف می شوند و .NET نیز به برنامه هایی که می خواهند از این کلاس استفاده کنند، فقط اجازه ی دسترسی به این متد ها را می دهد. در اولین بخش امتحان کنید با نحوه ی ایجاد این متد ها آشنا خواهیم شد. به این ترتیب هر فردی که بخواهد از این وب سرویس در برنامه ی خود استفاده کند می تواند به آن متصل شده و متد مورد نظر خود را فراخوانی کند، دقیقاً همانند اینکه کلاس در کامپیوتری قرار دارد که برنامه در آن اجرا شده است. همچنین متدی ایجاد خواهیم کرد تا به وسیله ی آن بتوانیم عملکرد وب سرویس را در اینترنت اکسپلورر بررسی کنیم.

امتحان کنید: ایجاد یک وب سرویس ساده

۱) ویژوال استودیو را باز کرده و گزینه ی File ➔ New Web Site را از نوار منوی ویژوال استودیو انتخاب کنید.
۲) در پنجره ی New Web Site، از کادر Language گزینه ی Visual C# و از کادر Location نیز گزینه ی HTTP را انتخاب کنید. سپس عبارتی را مانند

http://localhost/DemoService

در کادر مقابل Location وارد کرده و روی دکمه ی OK کلیک کنید (همانند شکل ۲۰-۲).



شکل ۲۰-۲

اساس ایجاد وب سرویس ها همانند ایجاد برنامه های مبتنی بر وب است که در فصل هفدهم با آنها آشنا شدیم، بنابراین ایجاد پروژه های مربوط به وب سرویس ها نیز مشابه ایجاد پروژه های تحت وب است. اگر در ایجاد پروژه های این فصل مشکلی دارید می توانید به فصل هفدهم و مطالبی که در آنجا بیان شد مراجعه کنید.

به این ترتیب ویژوال استودیو یک پروژه ی وب سرویس ایجاد کرده و یک فایل به نام `service.asmx` را نیز به آن اضافه می کند. پسوند `.asmx` مربوط به فایلهای وب سرویس است و از کلمات `Active Server Methods` گرفته شده است (حرف X نیز به منظور مشخص کردن `ASP.NET` یا همان `ASP+` به کار می رود. حرف X معادل + است که ۴۵ درجه چرخیده باشد). این صفحه یک سرویس را مشخص می کند، البته این پروژه می تواند شامل چند سرویس باشد.

۳) اگر فایل `service.cs` که حاوی کد مربوط به این وب سرویس است[1] باز نشده است، با استفاده از `Solution Explorer` آن را باز کنید. برای این کار کافی است روی فایل `Service.asmx` در `Solution Explorer` کلیک راست کرده و گزینه ی `View Code` را انتخاب کنید. هنگامی که ویژوال استودیو این فایل را ایجاد می کند، یک متد به نام `HelloWorld` نیز برای نمونه در آن قرار می دهد.کد مربوط به این متد مشابه زیر است:

```
[WebMethod]
public string HelloWorld()
{
      return "Hello World";
}
```

با انتخاب گزینه ی `Debug ➔ Start Debugging` از نوار منوی ویژوال استودیو برنامه را اجرا کنید. در این مرحله صفحه ای نمایش داده می شود و از شما می پرسد که می خواهید برنامه را بدون فعال کردن قابلیتهای مربوط به دیباگ اجرا کنید و یا می خواهید که یک فایل `config` به پروژه اضافه کرده و سپس در آن فایل قابلیتهای مربوط به دیباگ را فعال کنید؟ از این کادر گزینه ی `Add new Web.config file with`

[1] این فایلها به صورت پیش فرض به صورت `Code-Behind` هستند، یعنی کد مربوط به آنها در فایلی مجزا ذخیره می شود.

debugging enabled را انتخاب کرده و روی دکمه ی OK کلیک کنید. به دلایل امنیتی بهتر است قبل از اینکه یک برنامه ی تحت وب را در اینترنت قرار دهید، قابلیتهای مربوط به دیباگ را در آن غیر فعال کنید. با کلیک روی دکمه ی OK اینترنت اکسپلورر باز شده و صفحه ی service.asmx را نمایش خواهد داد. همانطور که مشاهده می کنید لیستی از متد هایی که در این سرویس وجود دارند در این صفحه نمایش داده می شوند. این صفحه برای تست کردن نحوه ی کارکرد وب سرویس به کار می رود.

نکته: در برنامه های واقعی، فایل web.config برای تنظیم کردن قسمتهای مختلف یک برنامه ی تحت وب مورد استفاده قرار می گیرد. در این فصل نمی خواهیم وارد جزئیات این سایت شویم، اما همین کافی است که بدانید این فایل برای ایجاد تغییرات کلی در مورد قسمتهای امنیتی، قسمتهای مربوط به دیباگ، قسمت های مربوط به کنترل کاربران و ... مورد استفاده قرار می گیرد. برای مطالعه ی بیشتر در مورد فایل web.config می توانید به سیستم راهنمای ویژوال استودیو (MSDN) مراجع کنید و یا در آدرس http://msdn2.microsoft.com به دنبال عبارت web.config بگردید.

۴) روی لینک HelloWorld کلیک کنید. این لینک شما را به صفحه ی دیگری خواهد برد که به وسیله ی آن می توانید عملکرد این متد را تست کنید. این صفحه شامل نام متد، دکمه ای برای فراخوانی متد و تست کردن آن، و نیز پروتکل هایی است که برای این متد پشتیبانی می شوند. توجه کنید که در این قسمت نام دو پروتکل عنوان شده است: دو نسخه از پروتکل SOAP و یک نسخه از پروتکل HTTP POST.

۵) با کلیک روی دکمه ی Invoke صفحه ی دیگری در یک پنجره ی جدید از اینترنت اکسپلورر باز شده و محتویات یک فایل XML را نمایش می دهد. این فایل که همانند زیر است، پاسخ وب سرویس است که به صورت SOAP به برنامه ی فرا خواننده برگشت داده می شود.

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">Hello World</string>
```

چگونه کار می کند؟

همانند برنامه های تحت وب که برای هر فایل aspx. یک کلاس وجود داشت که شامل کد مربوط به آن فایل بود، در وب سرویس ها نیز برای هر فایل asmx. یک کلاس وجود دارد که به صورت پیش فرض در یک فایل مجزا قرار می گیرد. متد HelloWorld نیز که در این قسمت به وسیله ی وب سرویس از آن استفاده کردیم نیز در این کلاس قرار دارد. اگر به تعریف این کلاس دقت کنید، مشاهده خواهید کرد که این کلاس از کلاس System.Web.Services.WebService مشتق شده است:

```
public class Service : System.Web.Services.WebService
{
```

کلاس WebService مسئول ایجاد صفحاتی است که در برنامه ی قبلی در اینترنت اکسپلورر مشاهده کرده و متد HelloWorld را از داخل آن فراخوانی کردید (برای تست کردن وب سرویس می توانید از مرورگرهای دیگر نیز استفاده کنید، اما ویژوال استودیو به صورت پیش فرض از اینترنت اکسپلورر استفاده می کند). این صفحات به نام **رابط تست**[1] شناخته می

[1] Test Interface

شوند. هنگام ایجاد یک کلاس، باید مشخص کنید که کاربر می تواند به کدامیک از متدهای آن دسترسی داشته باشد. متد هایی که می خواهد در وب سرویس مورد استفاده قرار بگیرند را باید با استفاده از خصیصه ی WebMethod() مشخص کنید. برای مثال مشاهده می کنید که این خصیصه در خط قبل از تعریف متد HelloWorld در برنامه ی قبلی نوشته شده است.

```
[WebMethod]
public string HelloWorld()
{
      return "Hello World";
}
```

قبل از اینکه صفحه ی تست وب سرویس باز شود، کادری نمایش داده خواهد شد و از شما می پرسد که آیا می خواهید یک فایل config را به برنامه اضافه کرده و سپس قابلیتهای دیباگ را در آن فعال کنید و یا می خواهید بدون فعال کردن این قابلیت ها وب سرویس را اجرا کنید؟ اگر می خواهید نحوه ی عملکرد متدهای این وب سرویس را تست کنید باید گزینه ی مربوط به اضافه کردن فایل web.config را انتخاب کنید. به این ترتیب فایلی به نام web.config به برنامه ی شما اضافه شده و قابلیت های دیباگ کردن نیز به وسیله ی آن فایل فعال می شوند. البته دقت داشته باشید که همواره قبل از اینکه سایت و یا وب سرویس خود را روی سرور قرار دهید تا افراد از آن استفاده کند، قابلیتهای دیباگ را در آن غیر فعال کنید. هنگامی که صفحه باز شده و رابط تست نمایش داده شد، لیستی از متد هایی که با استفاده از خصیصه ی WebMethod در کلاس مشخص شده بودند در صفحه نمایش داده می شوند. اگر روی نام هر یک از این متد ها کلیک کنید، صفحه ای نمایش داده خواهد شد که به وسیله ی آن می توانید متد را فراخوانی کرده و عملکرد آن را تست کنید.
هنگامی که یک متد از وب سرویس را به این صورت فراخوانی کنید، دقیقاً مشابه این است که در حال فراخوانی یک متد عادی هستید. به عبارت دیگر برای این کار هیچ چیز خاص و یا ناشناسی را مشاهده نمی کنید و تمام تجربیات قبلی شما در مورد متد ها در این قسمت نیز می تواند مورد استفاده قرار بگیرد.
همانطور که می دانید وب سرویس ها با استفاده از SOAP فراخوانی شده و یا پاسخ را ارسال می کنند. بنابراین زمانی که روی دکمه ی Invoke کلیک کنید، نتیجه ی اجرای متد در قالب یک پیغام SOAP به برنامه ای که متد را فراخوانی کرده است (اینترنت اکسپلورر) فرستاده می شود. همانطور که مشاهده می کنید پاسخی که به اینترنت اکسپلورر فرستاده شده است، دقیقاً همان عبارتی است که در متد به عنوان خروجی برگردانده بودیم که در یک تگ XML قرار گرفته است:

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">Hello World</string>
```

در این فصل ساختار XML ای که یک پیغام SOAP را تشکیل می دهد اهمیتی ندارد. با وجود این هر چه بیشتر با این فایلها کار کنیم، بهتر متوجه خواهید شد که جواب نهایی را در کدام قسمت از این فایلها می توان پیدا کرد.

اضافه کردن متدهای دیگر:

در قسمت امتحان کنید بعد می خواهیم وب سرویس را به گونه ای تغییر دهیم تا بتواند کاری را انجام دهد. به همین دلیل متدی را به برنامه اضافه خواهیم کرد تا بتواند جذر عددی که به آن فرستاده می شود را محاسبه کند.

امتحان کنید: اضافه کردن متد SquareRoot

۱) فایل service.cs را به وسیله ی ویرایشگر کد باز کرده و کد زیر را در کلاس Service بعد از متد HelloWorld وارد کنید:

```
        public Double GetSquareRoot(Double number)
        {
              return Math.Sqrt(number);
        }
```

اگر نمی توانید در قسمت کد نویسی متنی را وارد کنید، ممکن است به این علت باشد که برنامه ی قبلی هنوز در حال اجرا است. در این صورت صفحه ی مربوط به رابط تست و هر پنجره ی دیگری که پاسخ SOAP برگشته شده به وسیله ی وب سرویس را نشان می دهد را ببندید و سپس کد را وارد کنید. همچنین برای متوقف کردن برنامه می توانید گزینه ی Debug ➔ Stop Debugging را از نوار منوی ویژوال استودیو انتخاب کنید.

۲) برنامه را اجرا کنید. مشاهده خواهید کرد متدی که در این قسمت اضافه کرده بودید، در صفحه نمایش داده نمی شود. در واقع صفحه ای که در این قسمت مشاهده می کنید، هیچ تفاوتی با صفحه ای که در قسمت قبل دیده بودید ندارد. دلیل این مورد در این است که متدی که در این قسمت اضافه کردین را با خصیصه ی [WebMethod()] علامت گذاری نکردیم. دلیل اینکه این متد را ابتدا به این صورت تعریف کردیم در این بود که به شما نشان دهم می توانیم در کلاس مربوط به وب سرویس متدی داشته باشیم که حتی به صورت public باشد، اما در لیست متدهای آن وب سرویس نمایش داده نشود. مرورگر را ببندید و متد را با استفاده از خصیصه ی WebMethod به صورت زیر علامت گذاری کنید:

```
[WebMethod()]
public Double GetSquareRoot(Double number)
{
      return Math.Sqrt(number);
}
```

۳) مجدداً برنامه را اجرا کنید. این مرتبه متد را در ابتدای صفحه مشاهده خواهید کرد.

۴) قبل از اینکه متد را تست کنیم باید اینترنت اکسپلورر را به گونه ای تنظیم کنیم تا در صورت رخ دادن خطا بهجای اینکه پیغام کلی خطا را نمایش دهد، پیغام خطای کاملی که به وسیله ی HTTP برگشته می شود را بنویسد. برای این کار از نورا منوی اینترنت اکسپلورر گزینه ی Tools ➔ Internet Options را انتخاب کنید. سپس در قسمت Advanced گزینه ی "Show friendly HTTP error message" را از حالت انتخاب خارج کنید.

۵) روی نام متد GetSquareRoot در صفحه کلیک کنید. به این ترتیب صفحه ی مربوط به تست این متد نمایش داده می شود. مشاهده می کنید که کادری در این صفحه قرار گرفته است و به شما این اجازه را می دهد تا داده های مورد نیاز متد را در آن وارد کنید. اما بدون اینکه عددی را در این کادر وارد کنید، روی دکمه ی Invoke کلیک کنید.

۶) هنگامی که صفحه ی مربوط به پاسخ نمایش داده شد، مانند قسمت قبل پیغام SOAP را مشاهده نمی کنید. بلکه متنی که در این قسمت نشان داده می شود مشابه زیر خواهد بود:

```
System.ArgumentException: Cannot convert to System.Double.
Parameter name: type ---> System.FormatException: Input string was not
in a correct format.
```

این صفحه معمولا زمانی نمایش داده می شود که اجرای متد به هر دلیلی نتواند به درستی تمام شود و برنامه با خطایی مواجه شود. برای مثال در اینجا مقداری را که برای پارامتر این متد ارسال کردیم یک رشته ی تهی بود. متد نیز نتوانست رشته ی تهی را به یک مقدار از نوع `System.Double` تبدیل کند، بنابراین با خطا مواجه شد که توضیح خطای رخ داده را نیز در صفحه نمایش داد.

۷) صفحه ی مرورگر مربوط به پاسخ وب سرویس را ببندید و در صفحه ی قبلی، عدد ۲ را در کادر مربوط به پارامتر این متد وارد کنید. با فشار دادن دکمه ی `Invoke` متنی همانند زیر در پنجره ی اینترنت اکسپلورر نمایش داده می شود.

```
<?xml version="1.0" encoding="utf-8" ?>
<double xmlns="http://tempuri.org/">1.4142135623730952</double>
```

چگونه کار می کند؟

اگر به پیغامی SOAP ای که به عنوان نتیجه برگشت داده شده است نگاه کنید، متوجه خواهید شد عددی که در تگ `double` این پیغام قرار دارد، بسیار نزدیک به جذر عدد دو است :

```
<?xml version="1.0" encoding="utf-8" ?>
<double xmlns="http://tempuri.org/">1.4142135623730952</double>
```

خوب به این ترتیب متوجه شدید که متد به درستی کار می کند. تاکنون باید فهمیده باشید که ایجاد یک وب سرویس در `.NET` تا چه اندازه راحت است و این مورد نیز به دلیل پشتیبانی زیادی است که `.NET` از وب سرویس ها می کند. در کل می توانیم بگوییم که هر چیز که در قسمتهای قبل در مورد کلاسها، متد ها، پارامترهای ورودی و یا مقادیر بازگشتی یاد گرفته بودیم، می تواند به سادگی در وب سرویس ها نیز مورد استفاده قرار بگیرد. می توانید کلاس مورد نظر خود را طراحی کنید و در انتها با ایجاد تغییرات کوچکی در آن، از آن کلاس همانند یک وب سرویس استفاده کنید.

ایجاد سرور پروژه ی Picture Service:

به خاطر اینکه ایجاد یک وب سرویس ابتدایی بسیار ساده است، به سرعت به سراغ ایجاد یک برنامه ی کامل می رویم تا بتواند با استفاده از وب سرویس ها یک کار هدفمند را انجام دهد. همچنین یک برنامه ی کلاینت ویندوزی نیز ایجاد خواهیم کرد تا بتواند از این وب سرویس استفاده کند، زیرا تا کنون برای تست یک وب سرویس فقط از رابط تست و مرورگر اینترنتی استفاده می کردیم. وب سرویسی که در این قسمت طراحی خواهیم کرد به برنامه ها اجازه می دهد به لیستی از تصاویر موجود در سرور دسترسی داشته باشند.

ایجاد پروژه:

در این قسمت:

- فولدری را برای سرور مربوط به وب سرویس خود ایجاد می کنیم (این فولدر می تواند در همان کامپیوتری باشد که از آن استفاده می کنید و هم می تواند در کامپیوتری در اینترنت باشد) تا تصاویر مورد نظر در آن نگهداری شوند. همچنین عکسهای داخل این فولدر را نیز دسته بندی خواهیم کرد و در زیر فولدرهای مرتبط به هم قرار خواهیم داد.
- وب سرویسی ایجاد خواهیم کرد که بتواند فولدر را بررسی کرده و نام تمام زیر فولدر های آن را استخراج کند. همچنین این وب سرویس باید بتواند نام فایلهای داخل هر فولدر را نیز بدست آورد.
- هنگامی که وب سرویس نام یک فایل را به برنامه اضافه می کند، باید بتواند اطلاعات دیگر فایل را نیز از قبیل فرمت فایل، طول و عرض فایل و غیره را نیز بدست آورده و در برنامه قرار دهد.
- یک سایت وب ایجاد خواهیم کرد تا بتوانیم تصاویری که در این فولدر ها قرار دارند را در این سایت مشاهده کنیم.

به نظر می رسد که تمام مواردی که در بالا گفته شد را با یک سایت معمولی که با ASP.NET طراحی شده باشد نیز می توان انجام داد. با این وجود کاری که با یک وب سرویس می توان انجام داد ولی انجام آن با یک سایت ASP.NET غیر ممکن است این است که با استفاده از وب سرویس ها می توان یک برنامه ی تحت ویندوز طراحی کرد که بتواند از اطلاعات این سرور استفاده کند. با طراحی یک وب سایت شما مجبور خواهید بود که حتماً از HTML و یک مرورگر وب برای نمایش داده های خود استفاده کنید.

امتحان کنید: ایجاد پروژه

۱) با انتخاب گزینه ی ...File ➔ New Web Site از نوار منوی ویژوال استودیو، پنجره ی New Web Site را باز کرده و به وسیله ی این پنجره، یک پروژه از نوع ASP.NET Web Services به نام PictureService ایجاد کنید. می توانید فولدر مربوط به پروژه را در هر قسمتی از دیسک که مد نظر داشته باشید قرار دهید. با استفاده از وب سرور درونی که در ویژوال استودیو ۲۰۰۵ وجود دارد دیگر لازم نیست نگران تنظیمات مربوط به IIS باشید.

۲) در این پروژه نمی خواهیم از فایلهای service.asmx و Service.cs استفاده کنیم، بنابراین آنها را حذف کرده و فایلهای مورد نظر خود را اضافه خواهیم کرد. با استفاده از پنجره ی Solution Explorer روی نام این فایلها کلیک راست کرده و گزینه ی Delete را انتخاب کنید. سپس در کادر تاییدی که نمایش داده می شود روی دکمه ی OK کلیک کنید تا این فایلها حذف شوند.

۳) مجدداً با استفاده از پنجره ی Solution Explorer روی نام پروژه کلیک راست کرده و گزینه ی Add New Item را انتخاب کنید. در کادر Add New Item گزینه ی Web Service را از قسمت Templates انتخاب کرده و سپس نام Service را در کادر Name وارد کنید. با کلیک کردن روی دکمه ی Add این فایل به پروژه اضافه خواهد شد. بنابراین هم اکنون پروژه باید شامل یک فایل به نام Service.asmx باشد.

۴) حال که فایل asmx. جدید را به برنامه اضافه کردیم، باید مشخص کنیم که هنگام اجرا شدن برنامه این فایل باید توسط مرورگر اینترنتی نمایش داده شود. بنابراین با استفاده از کادر Solution Explorer روی نام Service.asmx کلیک کرده و گزینه ی Set as start page را انتخاب کنید.

۵) حال باید یک فولدر نیز به وب سایت اضافه کنیم تا بتوانیم فایلهای تصویر را در آن قرار دهیم. به این ترتیب این فایلها می توانند از طریق وب نیز در دسترس قرار گیرند. برای این کار روی نام پروژه در Solution Explorer

کلیک راست کرده و از منوی باز شده گزینه ی New Folder را انتخاب کنید تا یک فولدر جدید به پروژه اضافه شود. نام این فولدر را نیز Pictures قرار دهید.

۶) حال روی فایل Service.asmx کلیک راست کرده و گزینه ی View Code را انتخاب کنید تا کد مربوط به کلاس Service نمایش داده شود. سپس کد درون متد HelloWorld را به صورت زیر تغییر دهید:

```
[WebMethod]
public string HelloWorld()
{
      return Server.MapPath(
            Context.Request.ServerVariables["script_name"]);
}
```

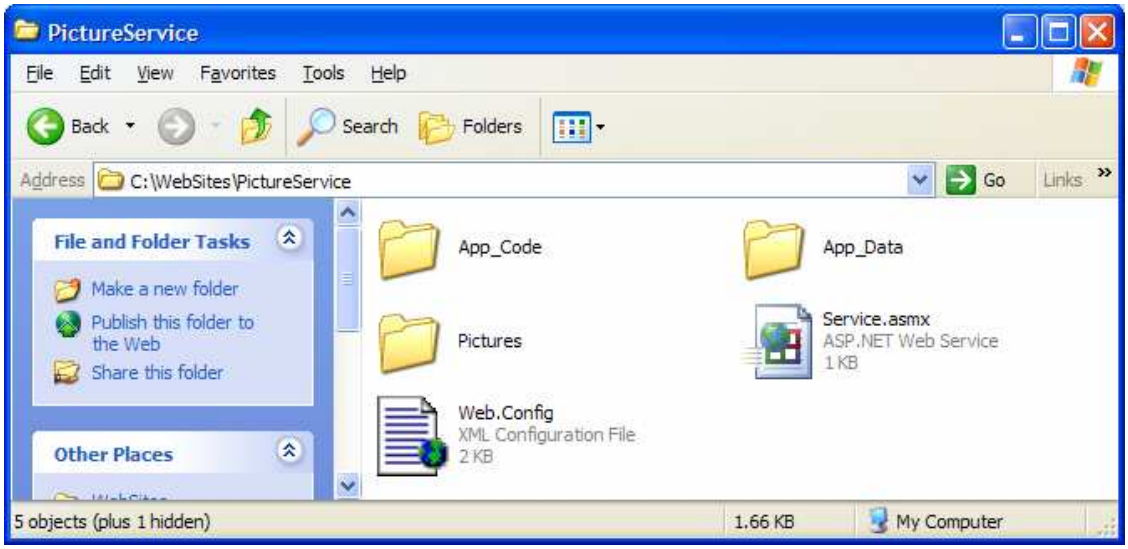
۷) برنامه را اجرا کرده و مانند قبل روی لینک مربوط به متد HelloWorld کلیک کنید تا رابط تست آن نمایش داده شود. در این صفحه روی دکمه ی Invoke کلیک کنید. به این ترتیب وب سرویس مسیری که فایل Service.asmx از آن اجرا می شود را نمایش خواهد داد.

```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">
      C:\WebSites\PictureService\Service.asmx
</string>
```

۸) مشاهده می کنید در کامپیوتری که این سرویس در آن اجرا شده است، فولدر C:\WebSites... حاوی فایل Service.asmx است. این قسمت به بعد به این فولدر، فولدر اصلی سرویس می گوییم. فولدر اصلی سرویس در برنامه ی قبل باید مشابه شکل ۲۰-۳ باشد.

۹) حال باید تعدادی عکس پیدا کرده و آنها را در فولدر Pictures که در فولدر اصلی سرویس قرار دارد کپی کنیم. برای این کار تعدادی عکس با فرمت های JPEG و یا GIF از کامپیوتر خود انتخاب کرده و در فولدر pictures قرار دهید.

۱۰) این تصاویر را به سه قسمت تقسیم کرده و هر یک را در یک فولدر مجزا درون Pictures قرار دهید. نام این سه فولدر را به دلخواه تنظیم کنید. در این مثال سه فولدر به نام Beach، Mountains و Remodel ایجاد کردیم و تصاویر را در آنها قرار دادیم.



شکل ۲۰-۳

چگونه کار می کند؟

تا اینجا برنامه شامل یک وب سرویس و نیز تعدادی عکس می شود که می توانیم داخل وب سرویس از آنها استفاده کنیم. تنها کدی که در این قسمت نوشتیم، کدی بود که درون متد `HelloWorld` از کلاس `Service` وارد کردیم:

```
[WebMethod]
public string HelloWorld()
{
      return Server.MapPath(
            Context.Request.ServerVariables["script_name"]);
}
```

متدی که در این قسمت از آن استفاده کردیم یکی از متدهای ASP.NET است (توجه داشته باشید که وب سرویس ها بر اساس تکنولوژی ASP.NET هستند) و توضیح آن فراتر از اهداف این کتاب است. در مورد این متد فقط می توانم این را بگویم که صفحات وب در ASP.NET به روشهای خاصی می توانند در مورد محیطی که در آن قرار گرفته اند اطلاعاتی را بدست آورند. این متد نیز برای بدست آوردن مسیر فیزیکی است که فایل مربوط به وب سرویس در آن قرار گرفته است.

برگرداندن آرایه ها به عنوان نتیجه ی متد:

در ابتدای این فصل مشاهده کردید که چگونه یک متد از یک وب سرویس می تواند یک مقدار را به عنوان نتیجه برگرداند، برای مثال یک عدد صحیح، یک رشته و یا ....

در شرایطی ممکن است بخواهید که لیستی از متغییر ها را به عنوان نتیجه از متد برگردانید. برای مثال زمانی که کاربر درخواست می کند که لیست فولدرهای حاوی عکس را دریافت کند، باید آرایه ای از رشته ها است به عنوان نتیجه برگشته شود که هر یک از آنها

شامل نام یکی از فولدرهای درون Pictures است. در قسمت امتحان کنید بعد این آرایه را به عنوان خروجی یکی از متدهای وب سرویس بر خواهیم گرداند.

امتحان کنید: برگرداندن یک آرایه از وب سرویس

۱) فایل Service.cs را در ویرایشگر کد باز کرده و متد HelloWorld را از کلاس Service در آن حذف کنید.

۲) با استفاده از دستور using همانند زیر فضای نام System.IO را به برنامه اضافه کنید:

```
using System.IO;
```

هنگامی که در حال ایجاد متدهای این وب سرویس هستید، لازم است که با آدرس فولدر Pictures دسترسی داشته باشید. برای این کار باید از کدی که در قسمت قبل برای بدست آوردن مسیر فیزیکی فولدر اصلی سرویس استفاده کردیم، استفاده کنیم. بنابراین خاصیت زیر را به برنامه اضافه کنید:

```
// PictureFolderPath - read-only property to return the picture
// folder...
public String PictureFolderPath
{
      get
      {
            // get the full path of this asmx page...
            String strAsmxPath, strPicturePath;
            strAsmxPath = Server.MapPath(
                  Context.Request.ServerVariables["script_name"]);
            // get the service path - everything up to and including
            // the "\"
            String strServicePath = strAsmxPath.Substring(0,
                                    strAsmxPath.LastIndexOf("\\") + 1);
            // append the word “Pictures” to the end of the path...
            strPicturePath = strServicePath + "Pictures"
            // return the path...
            return strPicturePath;
      }
}
```

۳) با بدست آوردن نام فولدر، فقط نیمی از مشکل حل شده است. برای اینکه بتوانیم کارهای مورد نظر خود را در این فولدر انجام دهیم، به شیئ ای نیاز داریم تا به ما اجازه دهد در فولدر تصاویر جستجو کرده و فولدر ها و فایلهای درون آن را بدست آوریم. برای ان کار می توانیم از کلاس System.IO.DirectoryInfo استفاده کنیم. بنابراین کد زیر را به برنامه اضافه کنید:

```
// PictureFolder - property to the DirectoryInfo containing
// the pictures...
public DirectoryInfo PictureFolder
{
      get
```

```
        {
            return new DirectoryInfo(PictureFolderPath);
        }
}
```

۴) حال می توانیم متد GetPictureFolders را ایجاد کنیم:

```
// GetPictureFolders - return an array of the picture folders...
[WebMethod(Description = "Return an array of the picture folders")]
public String[] GetPictureFolders()
{
      // get hold of the picture folder...
      DirectoryInfo pictureFolder = this.PictureFolder;
      // get the array of subfolders...
      DirectoryInfo[] objPictureSubFolder =
                              pictureFolder.GetDirectories();
      // create a string array to accommodate the names...
      String[] arrFolderNames = new String[objPictureSubFolder.Length];
      // now, loop through the folders...
      int intIndex = 0;
      foreach (DirectoryInfo pictureSubFolder in objPictureSubFolder)
      {
            // add the name...
            arrFolderNames[intIndex] = pictureSubFolder.Name;
            // next...
            intIndex += 1;
      }
      // finally, return the list of names...
      return arrFolderNames;
}
```

۵) برنامه را اجرا کرده و روی لینک مربوط به متد GetPictureFolders کلیک کنید. در صفحه ی مربوط بـه تست این متد دکمه ی Invoke را فشار دهید. به این ترتیب نام فولدر هایی را که ایجاد کرده بودید در قالـب XML همانند زیر مشاهده خواهید کرد.

```
<?xml version="1.0" encoding="utf-8" ?>
<ArrayOfString xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns="http://tempuri.org/">
  <string>Beach</string>
  <string>Mountains</string>
  <string>Remodel</string>
</ArrayOfString>
```

چگونه کار می کند؟

با مشاهده ی پاسخ SOAP متد، می توانید ببینید که واقعاً متد شما آرایه ای از رشته ها را به عنوان نتیجه برمـی گردانـد. زیـرا اولا رشته ها در تگی به نام ArrayOfStrings قرار گرفته اند و دوم نیز در اینجا ما سه تگ با نام string داریم که نام فولدر هایی که ایجاد کرده بودیم را نمایش می دهند.

```
<?xml version="1.0" encoding="utf-8" ?>
<ArrayOfString xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
               xmlns:xsd="http://www.w3.org/2001/XMLSchema"
               xmlns="http://tempuri.org/">
  <string>Beach</string>
  <string>Mountains</string>
  <string>Remodel</string>
</ArrayOfString>
```

خاصیت `PictureFolderPath` در این قسمت بسیار مهم است، زیرا در طول نوشتن متدهای این برنامه از آن بسیار استفاده خواهیم کرد. در اینجا ابتدا مسیری که فایل `service.asmx` در آن قرار دارد را با استفاده از کدی که در تمرین قبلی مشاهده کردید بدست می آوریم:

```
// PictureFolderPath - read-only property to return the picture
// folder...
public String PictureFolderPath
{
      get
      {
            // get the full path of this asmx page...
            String strAsmxPath, strPicturePath;
            strAsmxPath = Server.MapPath(
                  Context.Request.ServerVariables["script_name"]);
```

به این ترتیب متنی مشابه زیر در متغییر `strAsmxPath` قرار خواهد گرفت:

```
C:\WebSites\PictureService\Service.asmx
```

و ما می خواهیم این متن را به صورت زیر تغییر دهیم:

```
C:\WebSites\PictureService\Pictures
```

بنابراین باید قسمت `service.asmx` از این رشته را حذف کرده و آن را با نام `pictures` جایگزین کنیم. برای این کار با استفاده از متد `Substring` از کلاس `String`، قسمتی از رشته که به آن نیاز داریم را جدا کرده و در متغییر `strServicePath` قرار می دهیم. این قسمت که باید جدا شود از اولین کاراکتر شروع شده (کاراکتر با اندیس صفر) و تا آخرین کاراکتر `\` در رشته ادامه پیدا می کند. برای بدست آوردن اندیس آخرین کاراکتر `\` نیز از متد `LastIndexOf` از کلاس `String` استفاده می کنیم. همچنین اندیس آخر را با عدد یک جمع می کنیم تا مطمئن شویم که خود کاراکتر `\` آخر نیز به رشته اضافه می شود.

```
            // get the service path - everything up to and including
            // the "\"
            String strServicePath = strAsmxPath.Substring(0,
                                    strAsmxPath.LastIndexOf("\\") + 1);
```

سپس نام Pictures را به انتهای متغییر strServicePath اضافه کرده و رشته ی نهایی را بـه عنـوان نتیجـه از خاصیت برمی گردانیم:

```
            // append the word “Pictures” to the end of the path...
            strPicturePath = strServicePath + "Pictures"
            // return the path...
            return strPicturePath;
      }
}
```

کلاس System.IO.DirectoryInfo برای بدست آوردن اطلاعات بیشتر در مورد یـک فولـدر در دیـسک و یـا شبکه مورد استفاده قرار می گیرد. بنابراین یک خاصیت به نام PictureFolder ایجـاد مـی کنـیم تـا بـر اسـاس مقـدار خاصیت PictureFolderPath، شیئ ای را از کلاس DirectoryInfo ایجاد کرده و آن را برگرداند:

```
// PictureFolder - property to the DirectoryInfo containing
// the pictures...
public DirectoryInfo PictureFolder
{
      get
      {
            return new DirectoryInfo(PictureFolderPath);
      }
}
```

هنگـامی کـه توانـستیم بـه شـیئ DirectoryInfo مـورد نظـر خـود دسترسـی پیـدا کنـیم، مـی تـوانیم متـد GetPictureFolders را ایجاد کنیم. توجه کنید که در خصیصه ی WebMethod که در ایـن قـسمت اسـتفاده کرده ایم، توضیحاتی را نیز درباره ی این متد وارد کردیم. این توضیحات در صفحه ی Service در زیر لینک مربـوط بـه متـد GetPictureFolders نمایش داده شده و موجب می شود که افرادی که می خواهند از این سـرویس اسـتفاده کننـد، بهتر بتوانند کاربرد متد هایی که در آن قرار داده اید را متوجه شوند. اولـین کـاری کـه در متـد GetPictureFolders انجام می دهیم این است که با استفاده از خاصیت PictureFolder یک شیئ از نوع DirectoryInfo بدست آوریم که به فولدر C:\Websites\PictureService\ Pictures اشاره کند:

```
// GetPictureFolders - return an array of the picture folders...
[WebMethod(Description = "Return an array of the picture folders")]
public String[] GetPictureFolders()
{
      // get hold of the picture folder...
      DirectoryInfo pictureFolder = this.PictureFolder;
```

سپس برای بدست آوردن فولدر هایی که درون Pictures قرار دارند از متد GetDirectories استفاده می کنیم. ایـن متـد آرایـه ای از نـوع DirectoryInfo برمـی گـردان کـه هـر یـک از اعـضای آن بـه یکـی از فولـدرهای درون Pictures اشاره می کند:

```
      // get the array of subfolders...
      DirectoryInfo[] objPictureSubFolder =
                        pictureFolder.GetDirectories();
```

هنگامی که چنین آرایه ای را ایجاد کردیم، می توانیم از خاصیت Length آن استفاده کرده و بفهمیم کـه چنـد فولـدر در فولـدر Pictures وجود دارند. سپس یک آرایه ی رشته ای به همان طول ایجاد خواهیم کرد تا بتوانیم نـام فولـدر هـا را در آن قـرار دهیم:

```
        // create a string array to accommodate the names...
        String[] arrFolderNames = new String[objPictureSubFolder.Length];
```

هنگامی که آرایه ایجاد شد با استفاده از یک حلقه درون اشیای DirectoryInfo موجود در لیست حرکت کرده و نام هـر کدام از آنها را در آرایه قرار می دهیم:

```
        // now, loop through the folders...
        int intIndex = 0;
        foreach (DirectoryInfo pictureSubFolder in objPictureSubFolder)
        {
                // add the name...
                arrFolderNames[intIndex] = pictureSubFolder.Name;
                // next...
                intIndex += 1;
        }
```

در آخر نیز آرایه ای که شامل نام فولدر ها بود را برمی گردانیم:

```
        // finally, return the list of names...
        return arrFolderNames;
}
```

نکته: ممکن است در صحبتهایی که تاکنون کرده ایـم، متوجـه دو نـام متفـاوت شـده باشـید کـه مفهـوم یکـسانی دارنـد: فولـدر و دایرکتوری. زمانی که مایکروسافت ویندوز ۹۵ را معرفی کرد، تصمیم گرفت مفهومی که حدود یک دهه با نام دایرکتوری شناخته می شد را با نام فولدر عوض کند. اما مشاهده می کنید که در .NET همواره از نام Directory استفاده می شده است نه از نام فولدر. ممکن است دلیل این مورد ممکن است این باشد که طراحانی که در تیم .NET فعالیت می کنند عقیـده داشـته باشـند نـام Directory بهتر از نام Folder است.

برگرداندن یک ساختار به عنوان نتیجه ی یک متد در وب سرویس:

تاکنون نتیجه هایی که از متدهای یک وب سرویس برگردانده ایم همه به صورت متغیرهای ساده مانند عدد صـحیح، رشـته و یـا ... بودند. البته نحوه ی برگرداندن آرایه ای از این نوع متغییر ها را نیز در قسمت قبل مشاهده کردیم. امـا در ایـن قـسمت مـی خـواهیم ببینیم که چگونه می توان یک ساختار داده ای را به عنوان نتیجه ی یک متد از وب سرویس برگرداند.
در این قسمت می خواهیم آرایه ای از نام تصاویری که در یک فولدر وجود دارند را به عنوان نتیجه از یک متد برگردانیم. تفاوت ایـن متد با متد قبلی که نام فولدر ها را برمی گردانـد در ایـن اسـت کـه در قـسمت قبـل فقـط لازم بـود کـه نـام فولـدر هـایی کـه در Pictures وجود داشتند را برگردانیم، اما در این قسمت می خواهیم علاوه بر نام تصاویر، اطلاعات زیـر را نیـز در مـورد آنهـا برگردانیم:

- نام فایل تصویر (مانند PIC00001.JPG).
- آدرس کاملی که به فایل مربوط به این تصویر اشاره می کند (مانند C:\Websites\PictureService\Pictures\Beach\PIC00001.JPG).
- نام فولدری که حاوی تصویر است (مانند Beach).
- حجم تصویر (مانند ۲۶۷۷۵ بایت).
- تاریخی که تصویر ایجاد شده است (مانند ۲۰۰۶/۶/۲۶).
- فرمت تصویر (مانند JPG).

امتحان کنید: برگرداندن یک ساختار از متد

۱) برای اینکه بتوانیم اطلاعات لازم را برگردانیم، لازم است ساختاری را ایجاد کنیم تا بتوانیم اطلاعات مورد نظر را در آن قرار دهیم. برای ان کار با استفاده از پنجره ی Solution Explorer روی فولدر App_Code کلیک راست کرده و از منوی نمایش داده شده گزینه ی Add New Item... را انتخاب کنید. سپس با استفاده از کادر Add New Item یک کلاس به نام PictureInfo به برنامه اضافه کنید. دقت کنید که این کلاس در فولدر App_Code قرار گیرد. در این قسمت می خواهیم یک ساختار ایجاد کنیم. بنابراین تعریف کلاس PictureInfo را که به صورت اتوماتیک وارد شده است حذف کرده و آن را با کد زیر جایگزین کنید:

```
public struct PictureInfo
{
      // members...
      public String Name;
      public String Url;
      public String FolderName;
      public long FileSize;
      public DateTime FileDate;
      public String ImageFormat;
}
```

۲) برای اینکه به تصاویری که در یک فولدر وجود دارند دسترسی داشته باشیم، متدی ایجاد خواهیم کرد که نام فولدر را به عنوان ورودی دریافت کند. بنابراین کد زیر را به کلاس Service در فایل Service.cs اضافه کنید:

```
// GetPicturesInFolder - return an array of pictures from the folder...
[WebMethod(Description =
            "Return an array of pictures from the folder")]
public PictureInfo[] GetPicturesInFolder(String folderName)
{
      // get hold of the folder that we want...
      DirectoryInfo pictureSubFolder = new DirectoryInfo(
                              PictureFolderPath + "\\" + folderName);
      // we need to get the URL of the picture folder...
      String pictureFolderUrl = Context.Request.ServerVariables["URL"];
      // manipulate the URL
      // to return an absolute URL to the Pictures folder
      pictureFolderUrl = "http://" +
```

```
            Context.Request.ServerVariables["SERVER_NAME"] + ":" +
            Context.Request.ServerVariables["SERVER_PORT"] +
            pictureFolderUrl.Substring(0,
                        pictureFolderUrl.LastIndexOf("/") + 1)
                        + "Pictures";
      // get the list of files in the subfolder...
      FileInfo[] pictureFiles = pictureSubFolder.GetFiles();
      // create somewhere to put the picture infos...
      PictureInfo[] pictureList = new PictureInfo[pictureFiles.Length];
      // loop through each picture...
      int intIndex = 0;
      foreach (FileInfo pictureFile in pictureFiles)
      {
            // create a new pictureinfo object...
            PictureInfo pictureInfo = new PictureInfo();
            pictureInfo.Name = pictureFile.Name;
            pictureInfo.FolderName = folderName;
            pictureInfo.Url = pictureFolderUrl + "/" + folderName +
                              "/" + pictureFile.Name;
            pictureInfo.FileSize = pictureFile.Length;
            pictureInfo.FileDate = pictureFile.LastWriteTime;
            pictureInfo.ImageFormat =
                  pictureFile.Extension.Substring(1).ToUpper();
            // add it to the array...
            pictureList[intIndex] = pictureInfo;
            intIndex += 1;
      }
      // return the list of pictures...
      return pictureList;
}
```

۳) برنامه را اجرا کنید تا صفحه ی Service نمایش داده شود. در این صفحه روی لینک مربوط به متد GetPicturesInFolder کلیک کرده و در صفحه ی مربوط به تست این متد، نام یکی از فولدر هایی که ایجاد کرده بودید را وارد کنید، مانند Beach.

۴) هنگامی که روی دکمه ی Invoke کلیک کنید، اطلاعاتی در مورد نام فایلهای موجود در قالب XML نمایش داده می شود. فولدر Beach در این برنامه حاوی شش عکس است. بنابراین اطلاعاتی که از این متد برخواهد گشت مشابه زیر خواهد بود. البته کد زیر خلاصه شده ی اطلاعات برگشتی است:

```
<?xml version="1.0" encoding="utf-8" ?>
<ArrayOfPictureInfo
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      xmlns="http://tempuri.org/">
  <PictureInfo>
    <Name>No (1).jpg</Name>
    <Url>
      http://localhost:2459//PictureService/Pictures/Beach/No (1).jpg
    </Url>
    <FolderName>Beach</FolderName>
    <FileSize>63678</FileSize>
    <FileDate>2004-03-20T15:27:21.620875+04:30</FileDate>
    <ImageFormat>JPG</ImageFormat>
  </PictureInfo>
```

```
  <PictureInfo>
    <Name>No (2).JPG</Name>
    <Url>
      http://localhost:2459//PictureService/Pictures/Beach/No (2).JPG
    </Url>
    <FolderName>Beach</FolderName>
    <FileSize>32768</FileSize>
    <FileDate>1999-11-13T12:18:46+03:30</FileDate>
    <ImageFormat>JPG</ImageFormat>
  </PictureInfo>
</ArrayOfPictureInfo>
```

چگونه کار می کند؟

هنگامی که نام فولدر مورد نظر را بدست آوردید، می توانید یک شیئ از نوع `DirectoryInfo` ایجاد کنید که به این فولدر اشاره کند. سپس متد `GetFiles` از این شیئ را فراخوانی کنید تا لیستی از فایلهای موجود در این فولدر را در قالب یک آرایه از اشیایی از نوع `System.IO.FileInfo` برگرداند. کلاس `FileInfo` نیز همانند کلاس `DirectoryInfo` است، اما به جای اشاره کردن به یک فولدر به یک فایل اشاره می کند و اطلاعات مربوط به آن فایل را برمی گرداند. البته پارامتری که به متد ارسال می شود مسیر کامل فولدر نیست، بلکه نام فولدر است و باید با استفاده از مسیری که به وسیله ی خاصیت `PictureFolderPath` برمیگردد، مسیر کامل فولدر را ایجاد کرد:

```
// GetPicturesInFolder - return an array of pictures from the folder...
[WebMethod(Description =
           "Return an array of pictures from the folder")]
public PictureInfo[] GetPicturesInFolder(String folderName)
{
      // get hold of the folder that we want...
      DirectoryInfo pictureSubFolder = new DirectoryInfo(
                              PictureFolderPath + "\\" + folderName);
```

زمانی که کاربر تصاویر خود را در این سرور ذخیره کند، انتظار خواهد داشت تا با استفاده از این وب سرویس بتواند آدرس اینترنتی هر تصویر را نیز بدست آورد تا بتواند آن را به وسیله ی مرورگر وب مشاهده کند. بنابراین یکی از مقادیری که باید در این وب سرویس برای هر تصویر برگردانیم، آدرس URL مربوط به هر تصویر است که شامل نام سرور و عبارت `http://` نیز باشد. اگر به صورت عادی با استفاده از کد زیر از وب سرویس بخواهیم که آدرس فایل `.asmx` مربوط به خود را برگرداند، یک آدرس URL نسبی مانند `/PictureService/Service.asmx` را برخواهد گرداند:

```
      // we need to get the URL of the picture folder...
      String pictureFolderUrl = Context.Request.ServerVariables["URL"];
```

حال باید با استفاده از این آدرس URL نسبی که در `pictureFolderUrl` وجود دارد، آدرس URL کامل فولدر `Pictures` را بدست آوریم. برای این کار ابتدا عبارت "`http://`" را به همراه نام سروری که وب سرویس در آن قرار دارد به متغییر اضافه می کنیم. سپس با استفاده از متد `Substring` از کلاس `String`، فقط نام فولدر اصلی سرویس را بدست آورده و سپس نام فولدر `Pictures` را در انتهای آن قرار می دهیم. به این ترتیب رشته ای مانند زیر ایجاد خواهد شد:

http://localhost/PictureService/Pictures

```
// manipulate the URL
// to return an absolute URL to the Pictures folder
pictureFolderUrl = "http://" +
      Context.Request.ServerVariables["SERVER_NAME"] + ":" +
      Context.Request.ServerVariables["SERVER_PORT"] +
      pictureFolderUrl.Substring(0,
                  pictureFolderUrl.LastIndexOf("/") + 1)
                  + "Pictures";
```

مورد بعضی که در این متد به آن نیاز داریم، لیستی از فایلهایی است که فولدر مورد نظر وجود دارد:

```
// get the list of files in the subfolder...
FileInfo[] pictureFiles = pictureSubFolder.GetFiles();
```

سپس بازای هر فایل یک نمونه از ساختار `PictureInfo` ایجاد کرده و اطلاعات مورد نظر را در آن قرار می دهیم. البته بـه علت اینکه چندین فایل در این فولدر وجود دارند، پس چندین نمونه از ساختار `PictureInfo` نیز ایجاد خواهد شد که بایـد آنها را در یک آرایه قرار دهیم تا بتوانیم در انتها آن را از متد به عنوان نتیجه برگردانیم:

```
// create somewhere to put the picture infos...
PictureInfo[] pictureList = new PictureInfo[pictureFiles.Length];
// loop through each picture...
int intIndex = 0;
foreach (FileInfo pictureFile in pictureFiles)
{
      // create a new pictureinfo object...
      PictureInfo pictureInfo = new PictureInfo();
      pictureInfo.Name = pictureFile.Name;
      pictureInfo.FolderName = folderName;
      pictureInfo.Url = pictureFolderUrl + "/" + folderName +
                        "/" + pictureFile.Name;
      pictureInfo.FileSize = pictureFile.Length;
      pictureInfo.FileDate = pictureFile.LastWriteTime;
      pictureInfo.ImageFormat =
            pictureFile.Extension.Substring(1).ToUpper();
```

هر بار که یک ساختار ایجاد شده و تکمیل شد، با استفاده از متغییر `intIndex` آن را در مکان مربوط به خود در آرایه قرار مـی دهیم:

```
      // add it to the array...
      pictureList[intIndex] = pictureInfo;
      intIndex += 1;
}
```

در آخر نیز آرایه ی ایجاد شده را به عنوان نتیجه برمی گردانیم:

```
// return the list of pictures...
return pictureList;
```

```
}
```

با اتمام این متد وب سرویس مورد نظر ما نیز کامل می شود. حال بهتر است برنامه ای ویندوزی ایجاد کنیم و از این وب سرویس در آن برنامه استفاده کنیم.

ایجاد برنامه ی کلاینت:

تاکنون در این فصل مشاهده کردیم که چگونه می توان یک صفحه ی اینترنتی را ایجاد کرده و با استفاده از صفحه ی وبی که چارچوب .NET ایجاد می کند، عملکرد متدهای آن را تست کنیم. اما این صفحه ی وب در حقیقت یک رابط تست است و نیم توان انتظار داشت افرادی که می خواهند از این سرویس استفاده کنند نیز این صفحه را به کار ببرند.
هدف اصلی وب سرویس ها در این است که بتوانند بین برنامه ها یکپارچگی ایجاد کنند؛ بنابراین هنگامی که فردی بخواهد از یک وب سرویس استفاده کند، باید بتواند متد هایی که در آن ایجاد شده اند را در برنامه ی خود به کار ببرد.
در این قسمت یک برنامه ی تحت ویندوز ایجاد خواهیم کرد تا بتواند لیستی از فولدر های موجود در سرور که حاوی تصویر هستند را نمایش دهد. کاربر می تواند با کلیک کردن روی هر کدام از این فولدر ها، نام تمام تصاویر موجود در آن را مشاهده کند. همچنین با کلیک روی هر کدام از این تصاویر، اینترنت اکسپلورر باز می شود و تصویر انتخاب شده را نمایش می دهد.

:WSDL

برای استفاده از یک وب سرویس، باید از سندی به نام **WSDL**[1] استفاده کنید. سند WSDL یک سند از نوع XML است که شامل نام تمام متدهای می شود که در وب سرویس وجود دارند. همچنین در این سند نوشته شده است که هر متد به چه پارامترهایی و از چه نوع احتیاج دارد و یا مقداری که به وسیله ی هر متد برگشته می شود از چه نوعی است. سند WSDL مورد نیاز برای وب سرویسی که ایجاد کرده ایم به وسیله ی کلاس WebService ایجاد می شود.

ایجاد برنامه ی کلاینت:

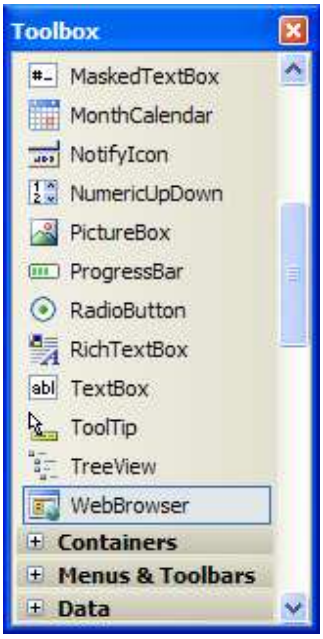
در این قسمت می خواهیم برنامه ی ویندوزی که بتواند از وب سرویس قبلی استفاده کند را ایجاد کنیم. البته در این برنامه برای نمایش تصاویر از اینترنت اکسپلورر عادی (همانند آنکه در فصل دهم، در برنامه ی FavoritesViewer مشاهده کردید) استفاده نخواهیم کرد. بلکه اینترنت اکسپلورر را به خود برنامه اضافه می کنیم! برای این کار لازم است که از کنترل Microsoft Web Browser در جعبه ابزار استفاده کنیم.

امتحان کنید: ایجاد برنامه ی کلاینت

۱) با استفاده از ویژوال استودیو ۲۰۰۵ یک پروژه ی ویندوزی جدید به نام PictureClient ایجاد کنید.

[1] Web Services Description Language

۲) در جعبه ابزار به قسمت Common Controls بروید. در آخر این قسمت کنترلی به نام WebBrowser وجود دارد که در شکل ۲۰-۴ نیز نشان داده شده است.



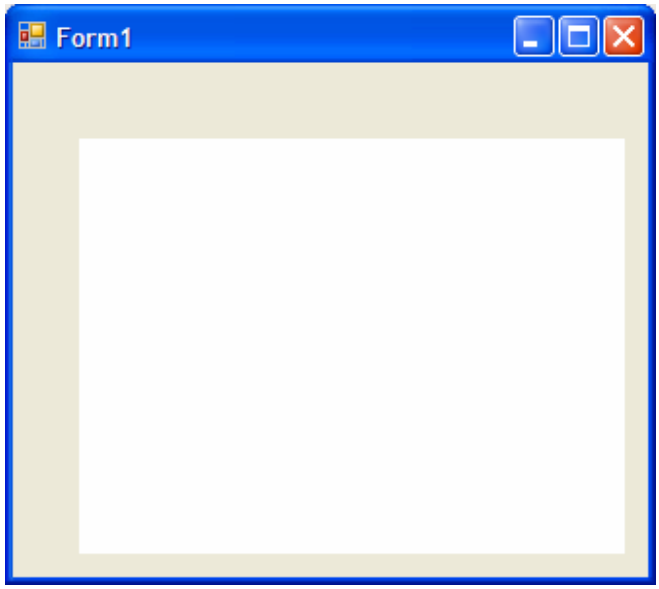
شکل ۲۰-۴

۳) بر روی این کنترل دو بار کلیک کنید تا یک نمونه از آن همانند شکل ۲۰-۵ در فرم قرار گیرد. البته لازم است که خاصیت Dock آن را برابر None قرار دهید.

۴) با استفاده از پنجره ی Properties خاصیت Name این کنترل را برابر با iePicture قرار دهید. همچنین خاصیت Anchor آن را نیز با مقدار Top, Right, Bottom, Left تنظیم کنید.

۵) در این برنامه می خواهیم با استفاده از این کنترل، تصاویری که در وب سرور وجود دارند را نمایش دهیم. اما قبل از ادامه بهتر است نحوه ی عملکرد این کنترل را مشاهده کنیم و ببینیم که چگونه می توان به وسیله ی آن از امکانات اینترنت اکسپلورر در یک فرم ویندوزی استفاده کرد. در قسمت طراحی فرم مربوط به Form1 روی نوار عنوان فرم دو بار کلیک کنید تا متد مربوط به رویداد Load آن به صورت اتوماتیک ایجاد شود. سپس کد زیر را به آن اضافه کنید:

```
private void Form1_Load(object sender, EventArgs e)
{
      // Set the browser to a default page...
      this.iePicture.Navigate("http://www.google.com");
}
```



شکل ۲۰-۵

۶) برنامه را اجرا کنید. مشاهده خواهید کرد که صفحه ی آغازین گوگل در فرم نمایش داده خواهد شد. می توانید متنی را وارد کرده و در اینترنت آن را جستجو کنید. به این ترتیب متوجه خواهید شد که این کنترل همانند اینترنت اکسپلورر به درستی کار می کند.

نکته: احتمالا متوجه شده اید که این کنترل دارای نوار ابزار نیست و بنابراین نمی توانید بسیاری از کارها را مانند رفتن به سایت قبلی در آن انجام دهید. برای انجام یک سری از کارها می توانید بر روی صفحه کلیک کرده و از منویی که باز می شود استفاده کنید.

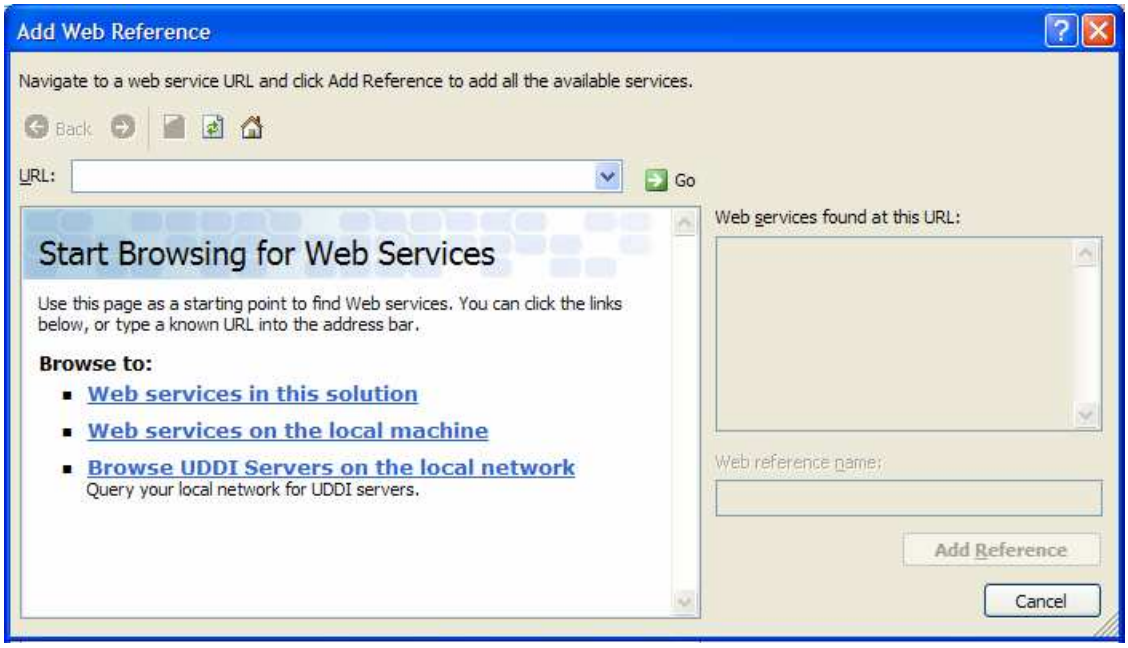
اضافه کردن یک وب سرویس به برنامه:

برای استفاده از یک وب سرویس باید آن را به صورت یک ارجاع وب به برنامه اضافه کنید. در قسمت امتحان کنید بعد، با انجام این کار از ویژوال استودیو می خواهیم که یک کلاس برای ما ایجاد کند تا بتوانیم به وسیله ی آن به متدهای درون وب سرویس در برنامه دسترسی داشته باشیم. دو کلاسی که در این قسمت ایجاد خواهند شد عبارتند از: `Service` و `PictureInfo`.

امتحان کنید: اضافه کردن یک وب سرویس به برنامه

۱) با استفاده از پنجره ی `Solution Explorer` روی نام پروژه کلیک راست کرده و گزینه ی `Add Web Reference...` را از منوی باز شده انتخاب کنید. به این ترتیب کادر `Add Web Reference` همانند شکل ۲۰-۶ نمایش داده خواهد شد. در کادر مقابل عبارت URL آدرس وب سرویسی که در قسمت قبل ایجاد کرده بودیم را وارد کنید. این آدرس مشابه زیر خواهد بود. البته باید نام `localhost` را با نام وب سرور مورد استفاده ی خود و شماره ی پورت را نیز با پورت مورد استفاده جایگزین کنید:

```
http://localhost:2459/PictureService/Service.asmx
```



شکل ۲۰-۶

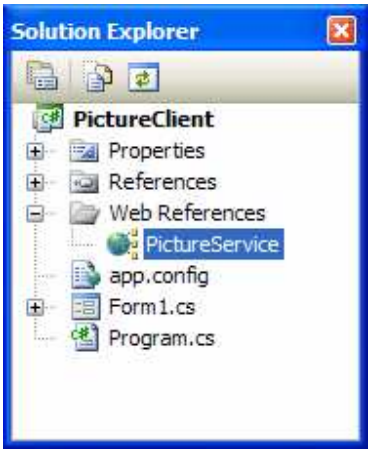
نکته: برای استفاده از وب سرویسی که در قسمت قبل ایجاد کرده بودیم، پروژه ی آن را در یک پنجره ی جدید از ویژوال استودیو باز کرده و آن را اجرا کنید. سپس آدرسی که در نوار آدرس اینترنت اکسپلورر برای صفحه ی رابط تست `Service` نشان داده می شود را در قسمت `URL` در کادر `Add Web Reference` (در ویژوال استودیو ای که حاوی پروژه ی ویندوزی است) وارد کنید تا صفحه ی `Service` در آن نیز نمایش داده شود.

۲) به این ترتیب صفحه ی آغازین وب سرویس به همراه لیست متد هایی که در آن قرار دارند نمایش داده خواهند شد. سپس در کادر `Web Reference Name` نام `PictureService` را وارد کرده و روی دکمه ی `Add Reference` کلیک کنید.

۳) به این ترتیب یک ارجاع به یک وب سرویس به پروژه اضافه خواهد شد و می توانید آن را همانند شکل ۲۰-۷ در پنجره ی `Solution Explorer` مشاهده کنید.

چگونه کار می کند؟

به این ترتیب ویژوال استودیو ۲۰۰۵ یک ارجاع را به سرور شامل وب سرویس اضافه می کند. همچنین یک کلاس به نام `PictureService.Service` نیز ایجاد می شود که حاوی متدهای موجود در وب سرویس است. با ایجاد اشیایی از نوع این کلاس می توانید متد هایی که در این وب سرویس وجود دارند را فراخوانی کنید. نامی که در کادر `Add Web Reference` برای این ارجاع وارد کردیم، در این قسمت به عنوان نام فضای نامی که کلاس در آن قرار می گیرد استفاده می شود.



شکل ۲۰-۷

نمایش لیست فولدر ها در برنامه:

تا اینجا می توانیم تمام متدهای درون وب سرویس را فراخوانی کرده و در برنامه از آنها استفاده کنیم. در بخش امتحان کنید بعد، با استفاده از یک کنترل `ComboBox`، لیستی از فولدرهای حاوی تصویر را در فرم نمایش می دهیم. برای دسترسی به لیست فولدرهای موجود می توانیم از متد `GetPicturesFolders` استفاده کنیم.

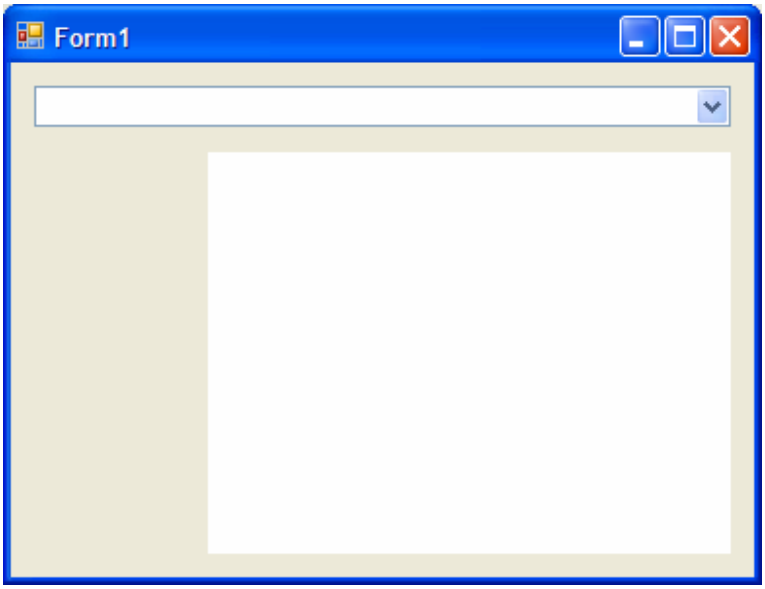
امتحان کنید: نمایش لیست فولدرهای موجود

۱) قسمت طراحی فرم مربوط به `Form1` را باز کرده و سپس با استفاده از جعبه ابزار یک کنترل `ComboBox` را همانند شکل ۲۰-۸ در این فرم قرار دهید.

۲) با استفاده از پنجره ی `Properties` خاصیتهای این کنترل را به صورت زیر تغییر دهید:

- خاصیت `Name` را برابر با `cboFolders` قرار دهید.
- خاصیت `DropDownStyle` را برابر با `DropDownList` قرار دهید.
- خاصیت `Anchor` را برابر با `Top,Left,Right` قرار دهید.

۳) بر روی نوار عنوان فرم دو بار کلیک کنید تا به قسمت مربوط به متد `Form1_Load` بروید. می خواهیم زمانی که برنامه اجرا می شود، با فراخوانی متد `GetPicturesFolders` لیستی از نام فولدر ها دریافت شده و در کادر `cboFolders` نمایش داده شود. بنابراین کد مشخص شده در زیر را در متد `Form1_Load` وارد کنید:



شکل ۲۰-۸

```
private void Form1_Load(object sender, EventArgs e)
{
      // get the pictures...
      try
      {
            // create a connection to the service...
            PictureService.Service service = new
                                    PictureService.Service();
            // get a list of the folders...
            String[] arrFolderNames;
            arrFolderNames = service.GetPictureFolders();
            // go through the list and add each name...
            foreach (String strFolderName in arrFolderNames)
            {
                  cboFolders.Items.Add(strFolderName);
            }
      }
      catch (Exception ex)
      {
            // loop through the inner exceptions...
            while (ex.InnerException != null)
                  ex = ex.InnerException;
            // report the problem...
            MessageBox.Show("An exception occurred.\n" + ex.Message);
      }
}
```

۴) برنامه را اجرا کنید. برای مرتبه ی اول ممکن است نمایش داده شدن فرم مقداری طول بکشد، زیـرا اولـین بـار فراخـوانی یکی از متدهای یک وب سرویس معمولا از فراخوانی دفعات بعد کندتر صورت می گیرد. اما بعد از اینکه متد اجرا شد مـی توانید لیستی از فولدرهای موجود در سرور را در برنامه مشاهده کنید.

چگونه کار می کند؟

احتمالاً بر خلاف چیزی که تصور می کردید، زیاد پیچیده نبود! چارچوب .NET بسیاری از پیچیدگی هایی که برای استفاده از یک وب سرویس وجود داشت را رفع کرده است.
متد مربوط به رویداد Load را با ایجاد یک بلاک try...catch شروع می کنیم تا بتوانیم خطاهای احتمالی را کنترل کنیم:

```
private void Form1_Load(object sender, EventArgs e)
{
      // get the pictures...
      try
      {
```

بهتر است حتماً هنگام استفاده از یک وب سرویس کد های مربوط به کنترل خطا را نیز وارد کنید، زیرا موارد زیادی ممکن است باعث شوند که استفاده از یک وب سرویس با مشکل مواجه شده و برنامه متوقف شود. اگر هر یک از مراحل متصل شدن به یک وب سرویس و ارسال پارامترهای مورد نیاز با مشکل مواجه شود، استثنایی در برنامه رخ می دهد که باید آن را کنترل کرد.
در ابتدای بلاک try یک شیئ از نوع PictureService.Service ایجاد می کنیم تا به وسیله ی آن بتوانیم به وب سرویس متصل شده و متدهای آن را استفاده کنیم. البته دقت کنید که در این لحظه هنوز برنامه به وب سرویس متصل نشده است، بلکه در حال آماده سازی شرایط برای اتصال به وب سرویس است:

```
            // create a connection to the service...
            PictureService.Service service = new
                                    PictureService.Service();
```

زیبایی استفاده از وب سرویس ها در .NET به این دلیل است که فراخوانی یک متد از وب سرویس هیچ تفاوتی با فراخوانی یک متد از کلاسهای موجود در برنامه ندارد. در اینجا به سادگی می توانیم متد GetPictureFolders را فراخوانی کرده و یک آرایه از نوع String را به عنوان نتیجه دریافت کنیم.

```
            // get a list of the folders...
            String[] arrFolderNames;
            arrFolderNames = service.GetPictureFolders();
```

هنگامی که آرایه را از وب سرویس بدست آوردیم می توانیم با استفاده از یک حلقه بین عناصر آن حرکت کرده و آنها را به ComboBox اضافه کنیم:

```
            // go through the list and add each name...
            foreach (String strFolderName in arrFolderNames)
            {
                  cboFolders.Items.Add(strFolderName);
            }
      }
```

اگر خطایی در سرور رخ دهد، .NET خطایی را از نوعی خاص برمی گرداند که مشخص می کند در سرور استثنایی به وجود آمده است. شیئ مربوط به خطای اصلی که باید از آن استفاده کنیم، در خاصیت InnerException قرار دارد. بنابراین ابتدا

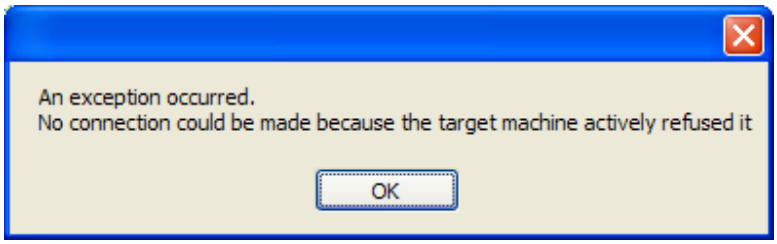
برای بدست آوردن خطای اصلی، آنقدر در بین خاصیتهای InnerException حرکت می کنیم تا مقدار ایـن خاصـیت در شیئ خطایی که بدست می آوریم برابر با null باشد. سپس پیغام این خطا را در برنامه نمایش می دهیم.

```
        catch (Exception ex)
        {
                // loop through the inner exceptions...
                while (ex.InnerException != null)
                        ex = ex.InnerException;
                // report the problem...
                MessageBox.Show("An exception occurred.\n" + ex.Message);
        }
}
```

می توانید قسمت کنترل خطای برنامه را با متوقف کردن اجرای وب سرویس تست کنید. هنگامی کـه اجـرای وب سـرویس متوقـف شود برنامه نمی تواند به متدهای آن دسترسی داشته باشد و با خطا مواجه می شود. برای متوقف کردن وب سـرویس، اگـر از IIS استفاده می کنید (هنگام ایجاد پروژه قسمت Location را برابـر بـا HTTP قـرار داده ایـد) از منـوی Start گزینـه ی Run را انتخاب کرده و در آن دستور زیر را وارد کنید:

```
net stop iisadmin
```

در این حالت توضیحاتی در یک پنجره نمایش داده می شود و در انتها از شما پرسیده می شود که آیا مطمئن هستید که می خواهیـد IIS را غیر فعال کنید؟ دکمه ی Y و سپس Enter را فشار دهید تا IIS خاموش شود. اما اگر از وب سـرور درونـی ویـژوال استودیو استفاده می کنید، بر روی آیکون آن در کنار ساعت سیستم کلیک کـرده و در پنجـره ای کـه بـاز مـی شـود روی دکمـه ی Stop کلیک کنید. حال برنامه ی ویندوزی را اجرا کنید. در این حالت پیغامی مشابه شکل ۲۰-۹ نمایش داده خواهد شد.



شکل ۲۰-۹

برای شروع مجدد IIS در پنجره ی Run دستور زیر را وارد کنید:

```
net start iisadmin
```

برای اینکه IIS را به هر دلیلی مجدداً راه اندازی کنید نیز می توانید از دستور زیر در پنجره ی Run استفاده کنید:

```
iireset
```

نمایش لیست فایلهای موجود و انتخاب آنها:

تا اینجا توانستیم که لیستی از فولدرهای موجود را از سرور دریافت کرده و آنها را نمایش دهیم. اما لازم است در صورتی که کاربر نام یک فولدر را از این لیست انتخاب کرد، مجدداً به سرور متصل شده و لیستی از فایلهای درون آن فولدر را بدست آوریم تا آنها را در برنامه نمایش دهیم. در قسمت بعد برنامه را به گونه ای تغییر خواهیم داد تا بتواند نام فولدر انتخاب شده به وسیله ی کاربر را از ComboBox بدست آورده و سپس با فراخوانی متد GetPicturesInFolder از وب سرویس، لیست تصاویر موجود در آن فولدر را بدست آورد. سپس این لیست را در فرم نمایش دهد تا کاربر بتواند تصویر مورد نظر خود را انتخاب کرده و مشاهده کند.

امتحان کنید: نمایش لیست فایلهای موجود

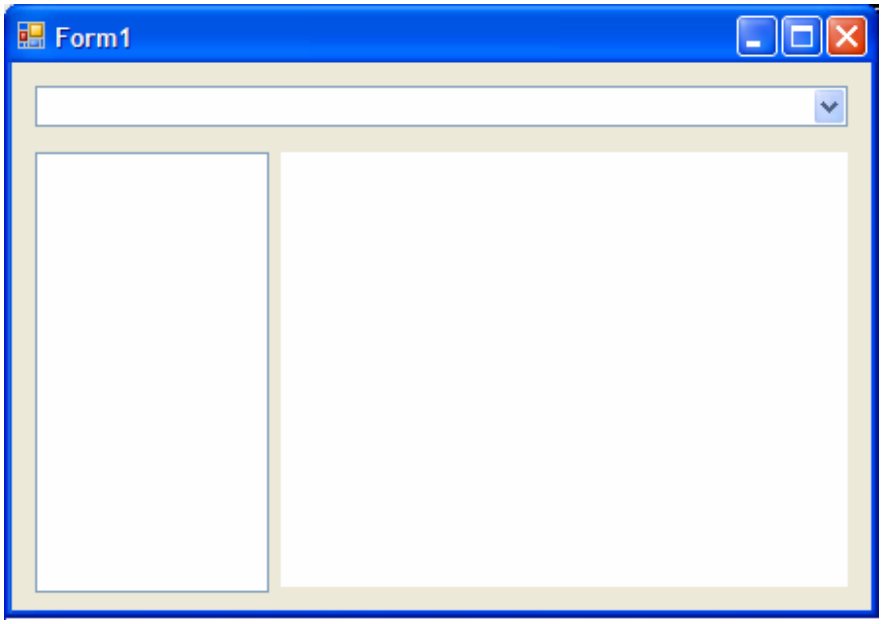
۱) برای نمایش لیست فایلهای موجود، باید ساختاری را مشابه PictureInfo ایجاد کنیم تا بتوانیم داده هایی که از طرف وب سرویس می رسند را در این ساختار قرار دهیم.برای این کار با استفاده از پنجره ی Solution Explorer کلاس جدیدی به نام PictureItem به برنامه اضافه کنید. سپس کد مشخص شده در زیر را در این کلاس وارد کنید:

```
public class PictureItem
{
      public PictureService.PictureInfo PicInfo;
      // Constructor...
      public PictureItem(PictureService.PictureInfo info)
      {
            PicInfo = info;
      }

      // ToString - provide a better representation of the object...
      public override String ToString()
      {
            return PicInfo.Name;
      }
}
```

۲) به قسمت طراحی مربوط به Form1 بروید و با استفاده از جعبه ابزار یک کنترل ListBox به فرم برنامه اضافه کنید. خاصیت Name این کنترل را برابر با lstFiles، خاصیت IntegralHeight آن را به False و خاصیت Anchor را نیز به Top, Bottom, Left تغییر دهید. به این ترتیب فرم برنامه ی شما مشابه شکل ۲۰-۱۰ خواهد شد.

۳) روی کنترل cboFolders در فرم دو بار کلیک کنید تا متد مربوط به رویداد SelectedIndexChanged این کنترل به صورت اتوماتیک ایجاد شود. سپس کد مشخص شده در زیر را در این متد وارد کنید:



شكل ٢٠-١٠

```
private void cboFolders_SelectedIndexChanged(object sender,
                                                EventArgs e)
{
      // what folder did we select?
      String folderName =
            cboFolders.Items[cboFolders.SelectedIndex].ToString();
      // clear the files list...
      lstFiles.Items.Clear();
      // connect to the service again and get the files back...
      try
      {
            // connect...
            PictureService.Service service = new
                                          PictureService.Service();
            // get the files back...
            PictureService.PictureInfo[] pictureList;
            pictureList = service.GetPicturesInFolder(folderName);
            // add the pictures to the list...
            foreach(PictureService.PictureInfo pictureInfo in
                                                      pictureList)
            {
                  // just add the name...
                  lstFiles.Items.Add(new PictureItem(pictureInfo));
            }
      }
      catch(Exception ex)
      {
            // loop through the inner exceptions...
            while (ex.InnerException != null)
                  ex = ex.InnerException;
            // report the problem...
            MessageBox.Show("An exception occurred.\n" + ex.Message);
      }
```

```
}
```

۴) بعد از اتمام این متد، مجدداً به قسمت طراحی فرم مربوط به Form1 برگشته و روی کنترل lstFiles دو بار کلیک کنید تا متد مربوط به رویداد SelectedIndexChanged این کنترل نیز به صورت اتوماتیک ایجاد شود. سپس کد مشخص شده در زیر را در این متد وارد کنید:

```
private void lstFiles_SelectedIndexChanged(object sender, EventArgs e)
{
      // get the pictureitem...
      PictureItem item =
                  (PictureItem)lstFiles.Items[lstFiles.SelectedIndex];
      if (item != null)
      {
            // tell ie to show the picture...
            iePicture.Navigate(item.PicInfo.Url);
      }
}
```

۵) برنامه را اجرا کرده و بعد از انتخاب یک فولدر از لیست، فایلی را از لیست سمت چپ انتخاب کنید تا در فرم نمایش داده شود.

نکته: برای اینکه بتوانید فایل انتخاب شده را مشاهده کنید، اینترنت اکسپلورر نباید در حالت Offline باشد. برای بررسی این مورد اینترنت اکسپلورر را باز کرده و از منوی File گزینه ی Work Offline را از حالت انتخاب خارج کنید. همچنین از نوار منوی اینترنت اکسپلورر گزینه ی ...Tools ➔ Internet Options را انتخاب کرده و در کادر Internet Options به قسمت Connections بروید. در این قسمت گزینه ی Never dial a connection را انتخاب کرده و دکمه ی OKرا فشار دهید.

چگونه کار می کند؟

کنترل ListBox دارای خاصیتی به نام Items است که می تواند لیستی از اشیا از نوع Object را در خود نگهداری کند. این کنترل برای نمایش آیتم هایی که باید در لیست قرار بگیرند، متد ToSrting اشیایی که در این لیست وجود دارند را فراخوانی می کند. در این برنامه ابتدا کلاسی ایجاد می کنیم که هر شیئ آن برای یکی از تصاویر موجود در وب سرور در نظر گرفته شود. این کلاس حاوی یک نمونه از ساختار PictureInfo می باشد که اطلاعات آن تصویر در آن ذخیره می شود. همچنین متد ToString را نیز در این کلاس به صورتی override می کنیم که نام فایل مورد استفاده را برگرداند:

```
public class PictureItem
{
      public PictureService.PictureInfo PicInfo;
      // Constructor...
      public PictureItem(PictureService.PictureInfo info)
      {
            PicInfo = info;
      }

      // ToString - provide a better representation of the object...
```

```
        public override String ToString()
        {
                return PicInfo.Name;
        }
}
```

سپس بازای هر یک فایلی که در سرور وجود داشته باشد، یک شیئ از این کلاس ایجاد کرده و آن را بـه لیـست اضـافه مـی کنـیم. هنگامی که کنترل ListBox بخواهد آیتم های درون خود را نمایش دهد، متد ToString این اشیا را فراخـوانی کـرده و این متد هم نام فایل را برمی گرداند. به این ترتیـب نـام فایلهـا در لیـست نمـایش داده مـی شـود. مـی توانیـد بـا تغییـر دادن متـد ToString، علاوه بر نام آدرس فایلها را نیز در لیست نمایش دهید. به این ترتیب لیست شامل نام و URL تصاویر موجود در وب سرور خواهد بود.

دقت کنید اشیای PictureInfo ای که در برنامه ی کلاینت وجـود دارنـد، بـا اشـیای PictureInfo ای کـه در سرور وجود دارند برابر نیستند. ویژوال استودیو ۲۰۰۵ بـر اسـاس اطلاعـاتی کـه در فایـل WSDL وجـود داشـت، کلاسـی بـه نـام PictureInfo در برنامه ایجاد کرده است تا بتوانیم از آن استفاده کنیم، دقیقاً مشابه کلاس Service که بـه صـورت اتوماتیک ایجاد کرده است. به همین دلیل است که PictureInfo در سرور یک ساختار است، اما در برنامه ی کلاینـت بـه صورت یک کلاس تعریف شده است.

هنگامی که کاربر فولدر انتخابی خود را در ComboBox تغییر داد، ابتدا نام آیتمی که انتخاب کرده است را بدست آورده و سـپس لیست موجود در ListBox را پاک می کنیم:

```
private void cboFolders_SelectedIndexChanged(object sender,
                                                EventArgs e)
{
        // what folder did we select?
        String folderName =
                cboFolders.Items[cboFolders.SelectedIndex].ToString();
        // clear the files list...
        lstFiles.Items.Clear();
```

سپس یک بلاک try...catch ایجاد کرده و بقیه ی کدها را در آن قرار می دهیم تا در صورت بروز خطا بتوانیم آن را کنتـرل کنیم:

```
        // connect to the service again and get the files back...
        try
        {
```

برای متصل شدن به سرور و استفاده از متدهای آن کافی است یک نمونه از کلاس Service ایجاد کنیم:

```
                // connect...
                PictureService.Service service = new
                                        PictureService.Service();
```

با فراخوانی متد GetPicturesInFolder و ارسال نام فولدری که کاربر انتخاب کرده است به عنوان پارامتر، لیـستی از تصاویری که در آن فولدر وجود دارد را به صورت آرایه ای از نوع PictureInfo دریافت می کنیم. اگـر فولـدر در سـرور

وجود نداشته باشد، سرویس یک استثنا ایجاد کرده و آن را پرتاب می کند. این استثنا به برنامه ی کلاینت می آید و در این قسمت به وسیله ی بلاک catch دریافت شده و پیغام مناسبی برای آن نمایش داده می شود:

```
            // get the files back...
            PictureService.PictureInfo[] pictureList;
            pictureList = service.GetPicturesInFolder(folderName);
```

هنگامی که آرایه ی شامل PictureInfo را بدست آوردیم می توانیم با استفاده از یک حلقه ی for در بین تمام عناصر آن حرکت کرده، بازای هر یک تصویر یک شیئ از نوع PictureItem ایجاد کنیم و در ListBox قرار دهیم.

```
            // add the pictures to the list...
            foreach(PictureService.PictureInfo pictureInfo in
                                                    pictureList)
            {
                  // just add the name...
                  lstFiles.Items.Add(new PictureItem(pictureInfo));
            }
      }
      catch(Exception ex)
      {
            // loop through the inner exceptions...
            while (ex.InnerException != null)
                  ex = ex.InnerException;
            // report the problem...
            MessageBox.Show("An exception occurred.\n" + ex.Message);
      }
}
```

هنگامی که آیتم انتخاب شده در ListBox را تغییر دادیم، این آیتم حاوی یک شیئ از کلاس PictureItem خواهد بود. بنابراین از خاصیت PicInfo این شیئ استفاده می کنیم تا به شیئ PictureInfoای که از طرف سرور برای این تصویر فرستاده شده بود دسترسی داشته باشیم. سپس با استفاده از خاصیت Url این شیئ، آدرس تصویر در سرور را بدست آورده و آن را با استفاده از کنترل Web Browserی که در فرم قرار داده ایم نمایش خواهیم داد. البته قبل از هر چیز بررسی می کنیم که مقدار item برابر با null نباشد، زیرا در این صورت برنامه با یک استثنا مواجه خواهد شد:

```
private void lstFiles_SelectedIndexChanged(object sender, EventArgs e)
{
      // get the pictureitem...
      PictureItem item =
                  (PictureItem)lstFiles.Items[lstFiles.SelectedIndex];
      if (item != null)
      {
            // tell ie to show the picture...
            iePicture.Navigate(item.PicInfo.Url);
      }
}
```

.NET Remoting:

.NET Remoting نیز مانند وب سرویس ها برای اتصال به یک سرویس در محلی دیگر (در همان کامپیوتر و یا کامپیوتری دیگر) مورد استفاده قرار می گیرد. همچنین .NET Remoting از پروتکل استاندارد SOAP برای برقراری ارتباط استفاده می کند که پروتکل استاندارد مورد استفاده به وسیله ی وب سرویس ها نیز به شمار می رود. از این چند جنبه، .NET Remoting و وب سرویس ها بسیار مشابه یکدیگر هستند.

تفاوت .NET Remoting با وب سرویس ها به صورت عمده به این دلیل است که در .NET Remoting می توانید از کانالهای TCP[1] نیز برای انتقال اطلاعات استفاده کنید. با استفاده از این کانالها نیز می توانید بر اینکه چه داده های منتقل می شوند و یا چگونه منتقل می شوند، کنترل کامل داشته باشید. همچنین داده ها در .NET Remoting می توانند به صورت باینری باشند و مانند وب سرویس ها لازم نیست که حتماً در قالب متن باشند. از بعضی جنبه ها .NET Remoting بسیار انعطاف پذیر تر از وب سرویس ها عمل می کند، اما هیچگاه نمی تواند جای وب سرویس ها را بگیرد. زیرا همانطور که گفتم وب سرویس ها بر پایه ی استانداردهای آزاد هستند و بنابراین برای برقراری ارتباط بین نرم افزارهای طراحی شده به وسیله ی پلت فرمهای گوناگون به کار می روند. .NET Remoting بیشتر در پلت فرم .NET مورد استفاده قرار می گیرد، اما دارای انعطاف پذیری بیشتری نسبت به وب سرویس ها است.

به دلیل اینکه این کتاب، یک کتاب مبتدی است، نمی خواهیم وارد جزئیات استفاده و کارکرد .NET Remoting شویم، اما در هر صورت باید چند نکته را در این مورد بدانید.

یکی از مهمترین مفاهیم موجود در .NET Remoting هدایت کردن اشیا از یک پردازش[2] (برنامه) به پردازش دیگر یا **مارشالینگ** است. همچنین یک شیئ می تواند از یک برنامه در یک کامپیوتر به یک برنامه در یک کامپیوتر دیگر منتقل شود. مارشالینگ در .NET Remoting از چنان اهمیتی برخوردار است که هر شیئ ای که بخواهد در بین پردازش ها جا به جا شود (برای مثال از یک برنامه در یک کامپیوتر، به برنامه ای دیگر در کامپیوتری دیگر منتقل شود) باید از کلاس MarshByRefObject مشتق شده باشد.

مفهوم مهم دیگری که باید با آن آشنا شوید، تفاوتی است که بین سرور و کلاینت در .NET Remoting وجود دارد. یک **سرور** معمولاً به برنامه ای گفته می شود که یک کلاس را در یک کانال و یک پورت خاص ثبت می کند. **کلاینت** نیز برنامه ای است که تقاضایی را برای ایجاد یک نمونه از آن شیئ، به آن کانال و آن پورت می فرستد. به این ترتیب سرور یک نمونه از کلاسی که در آن کانال و پورت ثبت شده است را ایجاد کرده و آن را به برنامه ی کلاینت ارسال می کند. به این ترتیب بعد از اینکه برنامه ی کلاینت متوجه شد که کلاس مورد نظر او در چه کانال و چه پورتی قرار دارد، تقاضایی را ایجاد کرده و آن را به آن کانال و آن پورت ارسال می کند. برنامه ی سرور نیز بر حسب اینکه تقاضا به چه کانال و چه پورتی ارسال شده است، یک شیئ از آن کلاس ایجاد کرده و آن را به برنامه ی کلاینت می فرستد. همانطور که از این توضیحات نیز مشخص است، در این روش سرور و کلاینت باید از یک کانال و یک پورت با یکدیگر ارتباط داشته باشند. در .NET Remoting برای انجام این کار از URI[3] به صورت زیر استفاده می شود:

<transport>://<machine>:<portnumber>/<name>

برای مثال اگر برای یک ارتباط بخواهیم از پروتکل TCP استفاده کنیم، URI مورد استفاده برای کلاینت و یا سرور برای برقراری ارتباط می تواند به صورت زیر باشد (maincpu نام کامپیوتر سرور است):

---

[1] Transmission Control Protocol
[2] Process
[3] Uniform Resource Identifier

```
tcp://maincpu:8000/MyPongEngine
```

اما اگر بخواهیم برای انتقال اطلاعات از HTTP استفاده کنیم، می توانیم URI زیر را به کار ببریم:

```
http://maincpu:8000/MyPongEngine
```

شماره ی پورتی که در این قسمت مورد استفاده قرار می گیرد می تواند هر چیزی باشد، فقط سرور و کلاینت هر دو باید آن را بدانند تا بتوانند از طریق آن پورت با یکدیگر ارتباط داشته باشند.
نحوه ی استفاده از این موارد را به زودی در عمل مشاهده خواهید کرد. بدون اینکه بیشتر از این وارد مباحث تئوری در مورد .NET Remoting شویم، در قسمت امتحان کنید بعد سعی خواهیم کرد که یک برنامه ی ساده که از آن استفاده کند را ایجاد کنیم. در شکل ۲۰-۱۱ معماری کلی .NET Remoting نمایش داده شده است. همانطور که در این شکل نیز مشخص است، .NET Remoting به یک سرور و یک کلاینت نیاز دارد، که در این مثال ابتدا برنامه ی سرور را ایجاد خواهیم کرد.

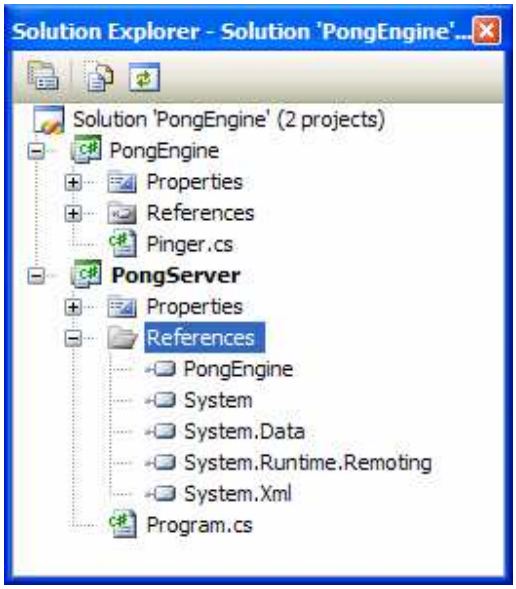
شکل ۲۰-۱۱

امتحان کنید: ایجاد سرور Pong و برنامه ی PongEngine

۱) با استفاده از ویژوال استودیو یک پروژه ی جدید از نوع Class Library به نام PongEngine ایجاد کنید.
۲) نام فایل Class1.cs که به صورت اتوماتیک ایجاد می شود را به فایل Pinger.cs تغییر دهید.
۳) برنامه را ذخیره کنید و سپس یک پروژه ی دیگر نیز از نوع Console Application به نام PongServer به این Solution اضافه کنید. برای اضافه کردن یک پروژه ی جدید به همین Solution می توانید از نوار منوی ویژوال استودیو گزینه ی File ➔ Add ➔ New Project… را انتخاب کنید.
۴) حال باید دو ارجاع به پروژه ی PongServer ایجاد کنید (دیاگرام شکل ۲۰-۱۲ را مشاهده کنید). برای این کار در پنجره ی Solution Explorer روی پروژه ی PongServer کلیک راست کرده و از منویی که باز می شود گزینه ی Add Reference را انتخاب کنید.

- یک ارجاع به پروژه ی PongEngine به برنامه اضافه کنید.
- یک ارجاع نیز به فضای نام System.Runtime.Remoting به برنامه اضافه کنید. این فضای نام در قسمت .NET از کادر وجود دارد.

۵) به این ترتیب قسمت ارجاعات برنامه ی شما باید مشابه شکل ۲۰-۱۲ باشد. برای مشاهده ی این قسمت روی نام References در Solution Explorer کلیک کنید تا ارجاعات برنامه نمایش داده شوند.



شکل ۲۰-۱۲

۶) روی نام پروژه ی PongEngine در Solution Explorer کلیک راست کرده و گزینه ی Add Reference را انتخاب کنید. سپس با استفاده از قسمت .NET از کادر Add Reference فضای نام System.Windows.Forms را به برنامه اضافه کنید.

۷) حال فایل Pinger.cs را در پروژه ی PongEngine باز کرده و کد موجود در آن را به صورت زیر تغییر دهید:

```
public class Pinger : MarshalByRefObject
{
	public String Name
	{
		get
		{
			return
			System.Windows.Forms.SystemInformation.ComputerName;
		}
	}
}
```

۸) فایل Program.cs از پروژه ی PongServer را باز کرده و تغییرات زیر را در آن ایجاد کنید:

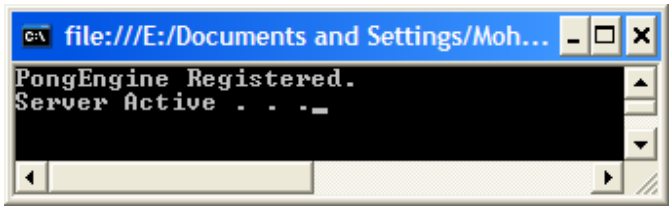
```
using System.Runtime.Remoting;
using System.Runtime.Remoting.Channels;
using System.Runtime.Remoting.Channels.Tcp;

namespace PongServer
{
	class Program
	{
		static void Main(string[] args)
		{
```

```
                TcpChannel channel = new TcpChannel(8000);
                ChannelServices.RegisterChannel(channel);
            RemotingConfiguration.RegisterWellKnownServiceType(
                                typeof(PongEngine.Pinger),
                                "MyPongEngine",
                                WellKnownObjectMode.SingleCall);
                Console.Write("PongEngine Registered." +
                                Environment.NewLine);
                Console.Write("Server Active . . .");
                Console.Read();
            }
        }
}
```

۹) حال روی پروژه ی PongServer کلیک راست کرده و از منویی که باز می شود گزینه ی Set as start project را انتخاب کنید.

۱۰) برنامه را اجرا کنید. به این ترتیب پنجره ای مشابه شکل ۲۰-۱۳ نمایش داده خواهد شد. اگر فایر وال و یا هر نرم افزار امنیتی دیگری در سیستم شما نصب شده باشد، این نوع سرور را محدود خواهد کرد. بنابراین اگر آن نرم افزار سوالی در مورد محدود کردن این برنامه از شما پرسید، به آن پاسخ منفی دهید و یا برنامه را غیر فعال کنید.



شکل ۲۰-۱۳

چگونه کار می کند؟

تا اینجا فقط یک پروژه ی کتابخانه ی کلاس و نیز یک پروژه ی تحت کنسول ایجاد کرده ایم. کدی هم که برای اضافه کردن فضای نامهای مربوط به Remoting وجود دارد زیاد نیست. اگر به پنجره ی کنسولی که به وسیله ی برنامه نمایش داده می شود (شکل ۲۰-۱۳) دقت کنید مشاهده خواهید کرد که برنامه به درستی کار می کند. اما خوب این سوال پیش می آید که تا اینجا برنامه چه کاری انجام داده است؟

ابتدا سرور یک کانال را در پورت ۸۰۰۰ ثبت می کند تا بتواند برای ارتباطات خود از این کانال استفاده کند. همانطور که گفتم برای اینکه برنامه ی سرور و برنامه ی کلاینت بتوانند با هم در ارتباط باشند باید از یک پورت استفاده کنند. بنابراین شماره پورتی که در این قسمت مشخص شده است به وسیله ی هر دو برنامه مورد استفاده قرار خواهد گرفت:

```
TcpChannel channel = new TcpChannel(8000);
ChannelServices.RegisterChannel(channel);
```

سپس باید کلاس Pinger را در این پورت ثبت کنیم. به این ترتیب .NET Remoting می داند زمانی که یک تقاضا از طرف یک برنامه ی کلاینت به این پورت رسید، چه نوع شیئ ای را باید ایجاد کرده و برای برنامه ی کلاینت ارسال کند. همچنین .NET Remoting برای ارسال این شیئ از پروتکل TCP و البته پورت 8000 استفاده خواهد کرد:

```
RemotingConfiguration.RegisterWellKnownServiceType(
                    typeof(PongEngine.Pinger),
                    "MyPongEngine",
                    WellKnownObjectMode.SingleCall);
```

شمارنده ی `WellKnownObjectMode.SingleCall` که در این متد به کار رفته است مشخص می کند که هر شیئ بعد از ایجاد شدن و فرستاده شدن به سمت برنامه ی کلاینت باید نابود شود. به این ترتیب بازای هر یک درخواستی که از طرف یک برنامه ی کلاینت به این پورت فرستاده می شود، یک شیئ جدید ایجاد شده و به سمت کلاینت فرستاده می شود. سپس شیئ ایجاد شده نابود خواهد شد[1].

حال باید یک برنامه ایجاد کنیم تا بتواند از `PongServer` استفاده کند. اما قبل از آن باید یک پروکسی ایجاد کنیم.

ایجاد پروکسی:

یک پروکسی برای برنامه ی کلاینت مورد استفاده قرار می گیرد و وظیفه دارد که رابط و ظاهر کلاس `PongEngine` را برای برنامه ی کلاینت ایجاد کند. برای استفاده از کلاس `PongEngine` می توانید یک ارجاع به فایل `PongEngine.dll` در برنامه ی کلاینت ایجاد کرده و از آن استفاده کنید. اما دلیلی برای این کار وجود ندارد، زیرا همانطور که توضیح داده شد برنامه ی کلاینت هیچگاه یک نمونه از شیئ `PongEngine` را ایجاد نمی کند و این شیئ همواره در برنامه ی سرور ایجاد می شود. بنابراین لازم نیست که کد این کلاس به برنامه ی کلاینت اضافه شود. تنها چیزی که در برنامه ی کلاینت نیاز است رابط و ظاهر کلاس است. به عبارت دیگر در برنامه ی کلاینت فقط لازم است که نام متد ها و خاصیتهای `public`، نام فیلد های عمومی و موارد دیگر که می تواند توسط برنامه ی کلاینت مورد استفاده قرار بگیرد وجود داشته باشد. نیازی نیست که کد این متد ها و خاصیت ها نیز به برنامه ی کلاینت فرستاده شود. پروکسی نیز در اینجا همین وظیفه را دارد. یک پروکسی در واقع رابط و ظاهر یک کلاس واقعی به شمار می رود. در واقع پروکسی مشابه یک کلاس واقعی است، اما هیچ کدی در آن وجود ندارد و فقط شامل تعریف متد ها و خاصیتهای و ... از کلاس است و البته کدی که خود پروکسی را ایجاد می کند. برنامه ی کلاینت به جای اینکه از کلاس واقعی در برنامه ی سرور استفاده کند، از کلاس ایجاد شده به وسیله ی پروکسی استفاده می کند، اما شیئ واقعی همواره در سرور ایجاد می شود.

در بخش امتحان کنید بعد یک پروکسی برای برنامه ایجاد خواهیم کرد.

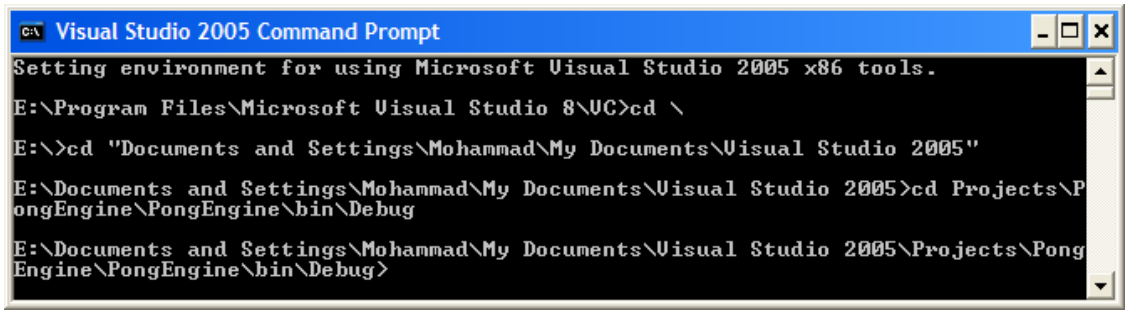
امتحان کنید: ایجاد پروکسی

۱) برای اینکه برای `PngEngine.dll` یک پروکسی ایجاد کنیم، باید از خط فرمان ویژوال استودیو استفاده کنیم. برای این کار گزینه ی زیر را از منوی `Start` در ویندوز انتخاب کنید:

`Start ➔ All Programs ➔ Microsoft Visual Studio 2005 ➔ Visual Studio Tools ➔ Visual Studio 2005 Command Prompt`

[1] این نوع رفتار معمولا به حالت `stateless` معروف است.

۲) حال مسیر خود را به مکانی تغییر دهید که فایل PongEngine.dll قرار دارد. برای مثال به شکل ۲۰-۱۴ نگاه کنید:



شکل ۲۰-۱۴

۳) سپس دستور زیر را وارد کنید:

```
soapsads -ia:PongEngine -oa:PongEngine_Proxy.dll
```

با اجرای این دستور یک فایل DLL جدید به نام PongEngine_Proxy.dll ایجاد خواهد شد.

۴) برای اینکه متوجه تفاوت بین این اسمبلی و اسمبلی قبلی بشوید می توانید از ابزار ILDasm استفاده کنید. این ابزار می تواند کد IL برنامه هایی که تحت .NET نوشته شده اند را نمایش دهد. در خط فرمان مربوط به ویژوال استودیو دستور ildasm را وارد کنید تا این برنامه اجرا شود.

۵) فایل PongEngine_Proxy.dll را با استفاده از ماوس روی این برنامه آورده و رها کنید. به این ترتیب برنامه مشابه شکل ۲۰-۱۵ خواهد شد.

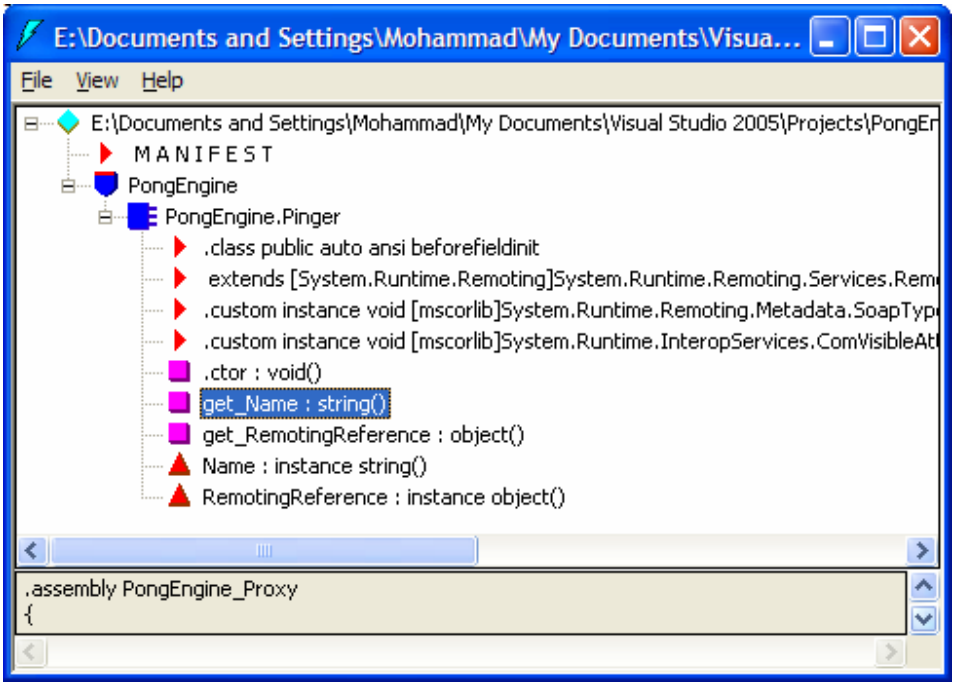
می توانید یک نسخه ی دیگر از ildasm را باز کرده و فایل PongEngine.dll را در آن قرار دهید تا کد IL آن را مشاهده کنید. به این ترتیب به سرعت متوجه تفاوتهای بین فایلهای موجود خواهید شد.

در قسمت امتحان کنید بعد، برنامه ی کلاینت را ایجاد خواهیم کرد.

امتحان کنید: ایجاد برنامه ی کلاینت

۱) در یک پنجره ی جدید از ویژوال استودیو، یک پروژه ی جدید از نوع Console Application به نام PongClient ایجاد کنید.

۲) با استفاده از پنجره ی Solution Explorer روی نام پروژه کلیک راست کرده و گزینه ی Add Reference را انتخاب کنید. سپس ارجاعی به فایل System.Runtime.Remoting را به برنامه اضافه کنید.

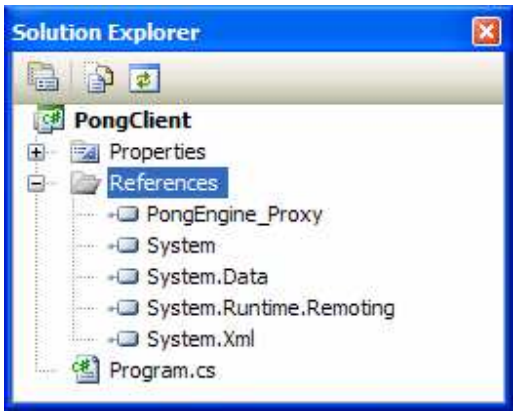


شکل ۲۰-۱۵

۳) حال باید یک ارجاع به فایل `PongEngine_Proxy.dll` که در مرحله ی قبل ایجاد کرده بودیم را به برنامه اضافه کنیم. برای این کار با استفاده از قسمت `Browse` در کادر `Add Reference`، فایل `PongEngine_Proxy.dll` را به برنامه اضافه کنید.

۴) همچنین با استفاده از قسمت `.NET` در کادر `Add Reference` نیز یک ارجاع به اسمبلی `System.Runtime.Remoting` به برنامه اضافه کنید.

۵) به این ترتیب قسمت `References` در پنجره ی `Solution Explorer` از برنامه باید مشابه شکل ۲۰-۱۶ باشد.



شکل ۲۰-۱۶

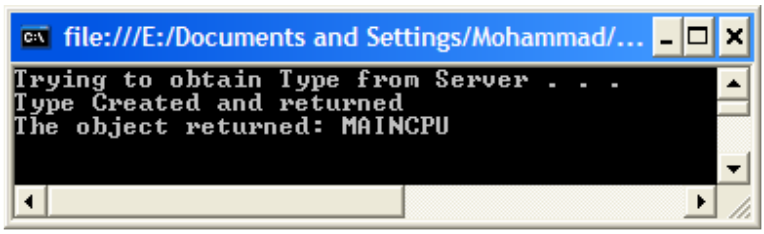
۶) حال باید یک نمونه از کلاس `PongEngine.Pinger` را ایجاد کرده و در برنامه از آن استفاده کنیم. برای این کار کد زیر را به فایل `Program.cs` اضافه کنید. البته فراموش نکنید که نام `maincpu` را با نام کامپیوتر

خـود جـایگزین کنیـد (بـرای بدسـت آوردن نـام کـامپیوتر مـی توانیـد از مقـداری کـه بـه وسـیله ی خاصـیت `ComputerName` از کلاس `System.Windows.Forms.SystemInformation` برمی گردد استفاده کنید).

```
class Program
{
      static void Main(string[] args)
      {
            PongEngine.Pinger client;
            Console.Write("Trying to obtain Type from Server . . ." +
                              Environment.NewLine);
            client = (PongEngine.Pinger)Activator.GetObject(
                        typeof(PongEngine.Pinger),
                        "tcp://Maincpu:8000/MyPongEngine");
            Console.Write("Type Created and returned" +
                              Environment.NewLine);
            Console.Write("The object returned:" + client.Name +
                              Environment.NewLine);
            Console.Read();
      }
}
```

۷) خوب، برای تـست برنامـه ابتـدا بایـد سـرور را اجـرا کنـیم. بـرای ایـن کـار بـه فولـدر `bin\debug` از پـروژه ی `PongServer` بروید و فایل `PongServer.exe` را اجرا کنید.

۸) حال برنامه ی کلاینت را به وسیله ی ویژوال استودیو اجرا کنید. به این ترتیب پنجره ای مشابه شکل ۲۰-۱۷ نمایش داده خواهد شد که نام کامپیوتر مورد استفاده ی شما نیز در انتهای آن نوشته شده است.



شکل ۲۰-۱۷

چگونه کار می کند؟

اگر فایلهای `PongClient.exe` و نیز `PongEngine_Proxy.dll` را به کامپیوتر دیگری که با استفاده از شبکه به این کامپیوتر متصل است منتقل کنید و چارچوب `.NET` را نیز در آن کامپیوتر نصب کنید، باز هم در پنجره نام کامپیوتری که برنامه ی سرور در آن اجرا شده است نمایش داده می شود. دلیل این مورد در ایـن اسـت کـه `.NET Remoting` شـیئ مورد نظر ما را ایجاد کرده و مقادیر خاصیت ها را در آن قرار می دهد. به کد زیر نگاه کنید:

```
PongEngine.Pinger client;
client = (PongEngine.Pinger)Activator.GetObject(
            typeof(PongEngine.Pinger),
```

```
"tcp://Maincpu:8000/MyPongEngine");
```

این کد ابتدا یک متغییر از نوع PongEngine.Pinger ایجاد کرده و سپس با استفاده از کلاس Activator یک نمونه از شیئ را در آن قرار می دهد. برای استفاده از کلاس Activator دو پارامتر را باید مشخص کنیم: اول نوع شیئ ای که می خواهیم ایجاد کنیم و دوم نیز URI مربوط به مکانی که آن نوع در آنجا قرار دارد.
به این ترتیب کلاس Activator به URI مشخص شده یک تقاضا ارسال می کند و یک نمونه از شیئ ای که مشخص کرده بودیم را به این وسیله دریافت می کند. البته این شیئ به صورت System.Object برگشته می شود و باید قبل از استفاده آن را به نوع مورد نظر خود تبدیل کنیم.

نتیجه:

در این فصل با وب سرویس ها و نیز .NET Remoting آشنا شدیم. وب سرویس ها به یک برنامه نویس اجازه می دهند تا به شیئ ای که در یک وب سرور قرار گرفته است دسترسی داشته باشند. وب سرویس ها بر اساس استانداردهای آزادی مانند SOAP و WSDL ایجاد شده اند و به وسیله ی تکنولوژیهایی مانند XML و یا HTTP نیز پشتیبانی می شوند.
فصل را با ایجاد یک وب سرویس که می توانست اطلاعاتی را برگرداند و همچنین کاری را انجام دهد شروع کردیم، البته آن وب سرویس فقط می توانست جذر یک عدد را محاسبه کند. سپس به عنوان یک مثال بهتر، وب سرویسی ایجاد کردیم که می توانست لیست فایلهای تصویر موجود در وب سرور را برگرداند. برای استفاده از این وب سرویس نیز یک برنامه ی کلاینت ویندوزی ایجاد کردیم که به وب سرویس متصل می شد و با فراخوانی متدهای آن لیست تصاویر موجود در سرویس را بدست می آورد. همچنین در این مثال از هماهنگی بین لایه های COM و .NET نیز برای استفاده از قابلیتهای اینترنت اکسپلورر در برنامه استفاده کردیم.
در آخر نیز به ایجاد برنامه های سرور و کلاینت با استفاده از .NET Remoting پرداختیم و بعد از ایجاد یک کلاس در یک کامپیوتر، از آن در یک برنامه در کامپیوتر دیگری استفاده کردیم. .NET Remoting بهترین روش برای ارتباط بین برنامه ها با استفاده از پروتکل هایی مانند TCP است.
در پایان این فصل باید با موارد زیر آشنا شده باشید:

- وب سرویس چیست و چگونه مورد استفاده قرار می گیرد.
- چگونه متد هایی را ایجاد کرده تا با استفاده از وب سرویس قابل دسترس باشند و سپس با استفاده از اینترنت اکسپلورر، عملکرد این متد ها را تست کنیم.
- چگونه یک برنامه ی کلاینت ایجاد کنیم که از وب سرویس ها استفاده کند.
- .NET Remoting چیست و کلاً برای چه مواردی مورد استفاده قرار می گیرد.
- تفاوت بین پروتکل هایی که برای .NET Remoting و وب سرویس ها به کار می رود چیست.

تمرین:

تمرین ۱:

یک وب سرویس ایجاد کنید تا بتواند اطلاعات مربوط به وب سرور را برگرداند. در این وب سرویس سه متد به صورت زیر قرار دهید: متد اول زمان وب سرور را برگرداند، متد دوم تاریخ وب سرور را برگرداند و متد سوم نیز نام وب سرور را برگرداند. سپس یک برنامه ی کلاینت ایجاد کرده و به وسیله ی آن وب سرویس را تست کنید.

تمرین ۲:

یک برنامه ی سرور و یک برنامه ی کلاینت ایجاد کنید که همانند وب سرویس تمرین اول عمل کند. سرور باید دو متد داشته باشد: یکی برای بدست آوردن زمان وب سرور و دیگری برای بدست آوردن تاریخ وب سرور. سپس در برنامه ی کلاینت عملکرد متدهای سرور را تست کنید. البته به دلیل اینکه به صورت محلی از سرور استفاده می کنید (سرور و کلاینت در یک کامپیوتر واقع شده اند) نیازی نیست که پروکسی ایجاد کنید.

فصل بیست و یکم: توزیع برنامه های کاربردی

یکی از مسائل مهم و نسبتاً پیچیده، مخصوصاً زمانی که در حال ایجاد برنامه های بزرگ هستید، مسئله ی توزیع برنامه بین کاربران است. یکی از عمومی ترین روشها برای این کار، ایجاد یک قسمت نصب برای برنامه و سپس توزیع آن در بین افرادی که می خواهند از برنامه استفاده کنند است. برای اینکه بتوانید قسمت نصب یک برنامه ی ویندوزی نسبتاً بزرگ را ایجاد کنید باید بتوانید اطلاعاتی را از رجیستری بخوانید و یا در آن بنویسید، با نوع های MIME کار کنید، فایلهای پیکر بندی مربوط به بانکهای اطلاعاتی را تنظیم کنید و .... البته شرکتهایی هستند که با استفاده از افراد متخصص در این زمینه ها، مسئولیت ایجاد قسمت نصب برنامه های بزرگ را بر عهده می گیرند. همچنین ویژوال استودیو ۲۰۰۵ نیز دارای قابلیت هایی برای ایجاد قسمت نصب برنامه ها است که بیشتر برای برنامه های کوچک و یا متوسط مورد استفاده قرار می گیرد.
همانطور که تا کنون در این کتاب مشاهده کرده اید، با استفاده از ویژوال استودیو ۲۰۰۵ می توان برنامه های مختلفی را ایجاد کرد مانند برنامه های تحت ویندوز، برنامه های تحت وب، وب سرویس ها و .... هر کدام از این برنامه ها، هنگام ایجاد قسمت نصب دارای پیچیدگی ها و نکات مخصوص به خود هستند که توضیح تمام آنها به چندین فصل نیاز دارد. به همین دلیل در این فصل تمام قسمتها و ابزارهای موجود برای ایجاد برنامه ی نصب در ویژوال استودیو را بررسی نمی کنیم، بلکه سعی می کنیم نگاه مختصری بر پروسه ی توزیع برنامه ها و نحوه ی انجام آن داشته باشیم.
در این فصل:

- با مفاهیم و اصطلاحات موجود در این زمینه آشنا خواهیم شد.
- با نحوه ی توزیع برنامه ها بدون نیاز به برنامه ی نصب آشنا خواهیم شد.
- با چگونگی ایجاد برنامه ی نصب برای برنامه ها آشنا خواهیم شد.
- نحوه ی تغییر ظاهر برنامه ی نصب را مشاهده خواهیم کرد.

منظور از توزیع برنامه چیست؟

منظور از **توزیع**[1] یک برنامه، عمل انتقال نسخه های کپی از یک برنامه به کامپیوتر های دیگر است به نحوی که برنامه بتواند در محیط جدید نیز کار کند. توزیع یک برنامه با چیزی که شما با نام نصب یک برنامه می شناسید متفاوت است. منظور از توزیع یک برنامه، نحوی پخش کردن آن برنامه در بین افرادی است که می خواهند از آن استفاده کنند. در توزیع یک برنامه بیشتر با این مورد درگیر هستیم که چگونه می توان یک برنامه را در اختیار کاربران قرار داد و یا به عبارت دیگر چگونه می توان آن را به قسمتهایی که می خواهیم، منتقل کنیم.
اما منظور از **نصب**[2] یک برنامه، پروسه ی بارگذاری برنامه در حافظه، پیکر بندی و سپس نصب آن است. منظور از نصب یک نرم افزار کارهایی است که برای پیکر بندی برنامه انجام می دهیم و منظور از توزیع یک نرم افزار روشی است که برای انتقال یک برنامه بین کاربران به کار می بریم.
با این توضیحات احتمالاً حدس زده اید که یکی از روشهای توزیع یک نرم افزار، استفاده از CD است. همچنین اینترنت نیز یکی دیگر از روشهای توزیع به شمار می رود. این دو روش توزیع، ممکن است مراحل نصب متفاوتی را نیز احتیاج داشته باشند. برای مثال

---

[1] Deployment
[2] Setup - Installation

اگر از CD برای توزیع برنامه ی خود استفاده می کنید، باید تمام قسمتهای اضافی برنامه را نیز در آن قرار داده و به مرحله ی نصب اضافه کنید. اما انجام این کار زمانی که می خواهید برنامه را از طریق اینترنت منتشر کنید منطقی به نظر نمی رسد، زیرا به این ترتیب حجم برنامه زیاد شده و انتقال آن از طریق اینترنت مشکل خواهد شد. همچنین اگر بخواهید برنامه ی خود را با استفاده از CD توزیع کنید می توانید در برنامه ی نصب از اسکریپت های جاوا نیز استفاده کنید، زیرا معمولاً فردی که از آن CD استفاده می کند اجازه ی اجرا کردن این اسکریپت ها را نیز دارد. اما اگر بخواهید برنامه ی خود را با استفاده از اینترنت توزیع کنید استفاده از اسکریپت های جاوا چندان صلاح نیست زیرا ممکن است اینترنت اکسپلورر اجازه ی اجرای آنها را به برنامه ندهد و بنابراین برنامه برای نصب با مشکل مواجه شود. این نوع ملاحظات، هنگامی که بخواهید روش توزیع برنامه ی خود را انتخاب کنید و یا بخواهید یک برنامه ی نصب ایجاد کنید از اهمیت زیادی برخوردار هستند.
حال که با اصطلاحات موجود در این زمینه آشنا شدیم، بهتر است نحوه ی توزیع برنامه های ایجاد شده با ویژوال استودیو را نیز بررسی کنیم. در زیر با دو روش ساده ی توزیع برنامه ها با استفاده از ویژوال استودیو آشنا خواهیم شد.

توزیع برنامه با استفاده از روش ClickOnce:

منظور از روش ClickOnce، ارسال برنامه و همچنین اسمبلی های مورد نیاز آن به کامپیوتر کاربر است به گونه ای که در مواقع مورد نیاز، برنامه بتواند به سادگی خود را به روز کند. با استفاده از این روش، سه راه برای توزیع برنامه وجود دارد: استفاده از فایلهای به اشتراک گذاشته شده در شبکه، استفاده از صفحات وب و استفاده از دیسک های مختلف مانند CD و یا DVD. البته این روش برای برنامه های کوچک و یا متوسط مناسب است و نمی توان از آن برای توزیع برنامه های بزرگ استفاده کرد.
استفاده از روش ClickOnce برای توزیع برنامه ها دارای سه مزیت عمده است. مزیت اول این است که با استفاده از این روش، یک برنامه ی ویندوزی به سادگی می تواند خود را به روز کند. برای این کار کافی است که آخرین نسخه ی برنامه را به صورت قبلی بین کاربران توزیع کنید (در شبکه قرار دهید، بر روی صفحات وب بگذارید و یا با استفاده از CD به کاربران برسانید). سپس مرتبه ی بعد که کاربر بخواهد از برنامه استفاده کند، آخرین نسخه از برنامه اجرا خواهد شد. مزیت دوم استفاده از این روش در این است که کاربران با حداقل سطح دسترسی نیز می توانند از بیشتر برنامه هایی که به این روش توزیع می شوند استفاده کنند، در صورتی که در روشهای دیگر کاربران برای استفاده از برنامه نیاز دارند که از مدیر سیستم خود اجازه دریافت کنند[1]. مزیت سوم این روش نیز در این است که استفاده از آن کمترین تاثیر را در کامپیوتر کاربر خواهد داشت. برنامه فقط در فولدر مشخص شده برای آن قرار خواهد گرفت و گزینه هایی را به منوی استارت و قسمت Add/Remove Programs اضافه خواهد کرد.
در قسمت امتحان کنید زیر یک برنامه ی ویندوزی را با استفاده از روش ClickOnce به وسیله ی وب توزیع خواهیم کرد.

امتحان کنید: توزیع یک برنامه ی ClickOnce با استفاده از وب

۱) با استفاده از ویژوال استودیو یک برنامه ی ویندوزی جدید به نام ClickOnce ایجاد کنید.
۲) با استفاده از جعبه ابزار یک کنترل Button و یک کنترل Label به فرم برنامه اضافه کنید. خاصیت Name مربوط به کنترل Button را برابر با btnVersion و خاصیت Text آن را نیز برابر با Version قرار دهید. همچنین خاصیت Name کنترل Label را نیز با مقدار lblVersion تنظیم کنید.

[1] منظور از این قسمت این است که برای ایجاد برنامه های ClickOnce لازم نیست که حتماً اکانت مورد استفاده به وسیله ی کاربر از نوع Administrator، PowerUser و یا اکانت هایی با سطح دسترسی بالا باشد. بلکه افراد با اکانت های عادی نیز می توانند این نوع برنامه ها را اجرا کنند.

۳) روی کنترل Button دو بار کلیک کنید تا متد btnVersion_Click ایجاد شود. سپس کد مشخص شده در زیر را به این متد اضافه کنید:

```
private void btnVersion_Click(object sender, EventArgs e)
{
      lblVersion.Text = "Version 1.0";
}
```

۴) برنامه را اجرا کنید تا عملکرد آن را تست کنیم. با کلیک کردن روی دکمه ی Version عبارت "Version 1.0" همانند شکل ۲۱-۱ در کنترل Label نمایش داده خواهد شد. حال برنامه را ببندید و به محیط ویژوال استودیو برگردید. سپس با استفاده از نوای منو گزینه ی Build ➔ Build ClickOnce را انتخاب کنید تا برنامه کامپایل شود.

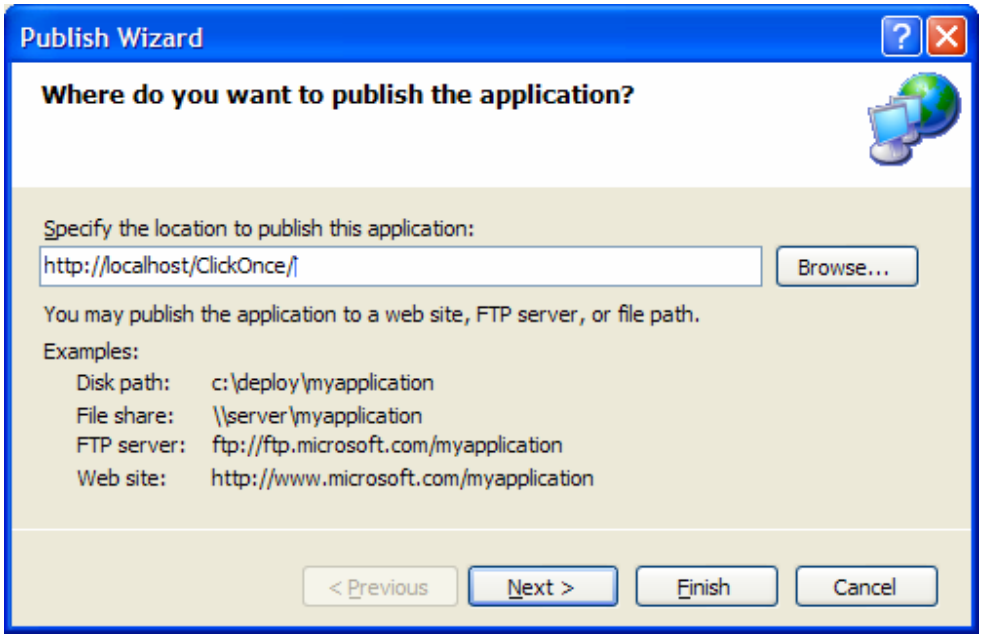


شکل ۲۱-۱

۵) حال باید اسمبلی ایجاد شده را با استفاده از وب توزیع کنیم. برای این کار از IIS محلی استفاده می کنیم. اگر IIS در کامپیوتر شما پیکر بندی نشده است می توانید از روشهای دیگر برای توزیع استفاده کنید.

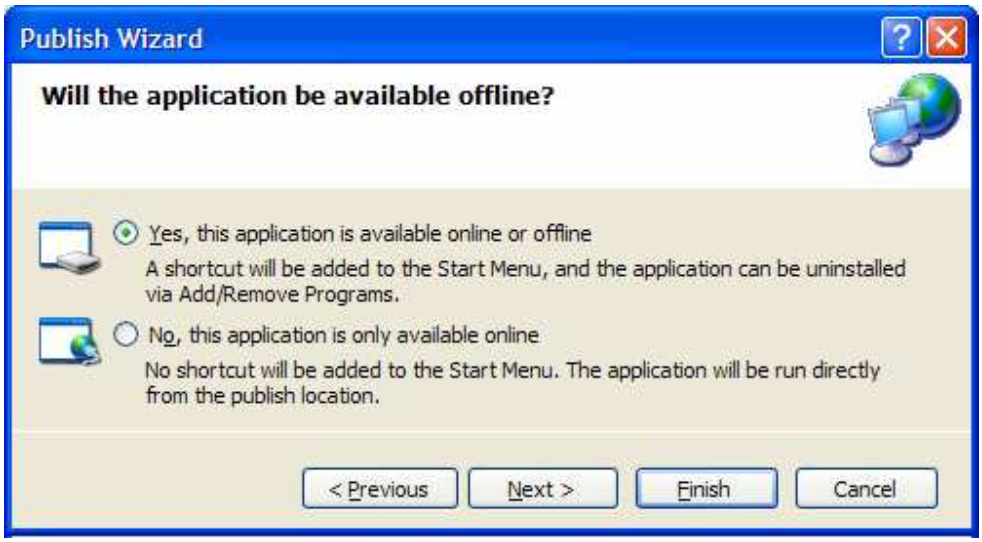
نکته: برای راه اندازی اولیه ی IIS با استفاده از Control Panel به قسمت Add Or Remove Programs بروید و از سمت چپ صفحه، گزینه ی Add/Remove Windows Components را انتخاب کنید. در کادر Windows Component Wizard، گزینه ی Internet Information Services (IIS) را از قسمت components انتخاب کرده و روی دکمه ی Next کلیک کنید تا IIS نصب شود.

۶) با استفاده از پنجره ی Solution Explorer روی نام پروژه کلیک راست کرده و از منوی باز شده گزینه ی ...Publish را انتخاب کنید. به این ترتیب کادر Publish Wizard همانند شکل ۲۱-۲ نمایش داده خواهد شد. در این قسمت می توانید مکانی را برای توزیع فایلهای برنامه انتخاب کنید. در این مثال مکان پیش فرض را قبول می کنیم.



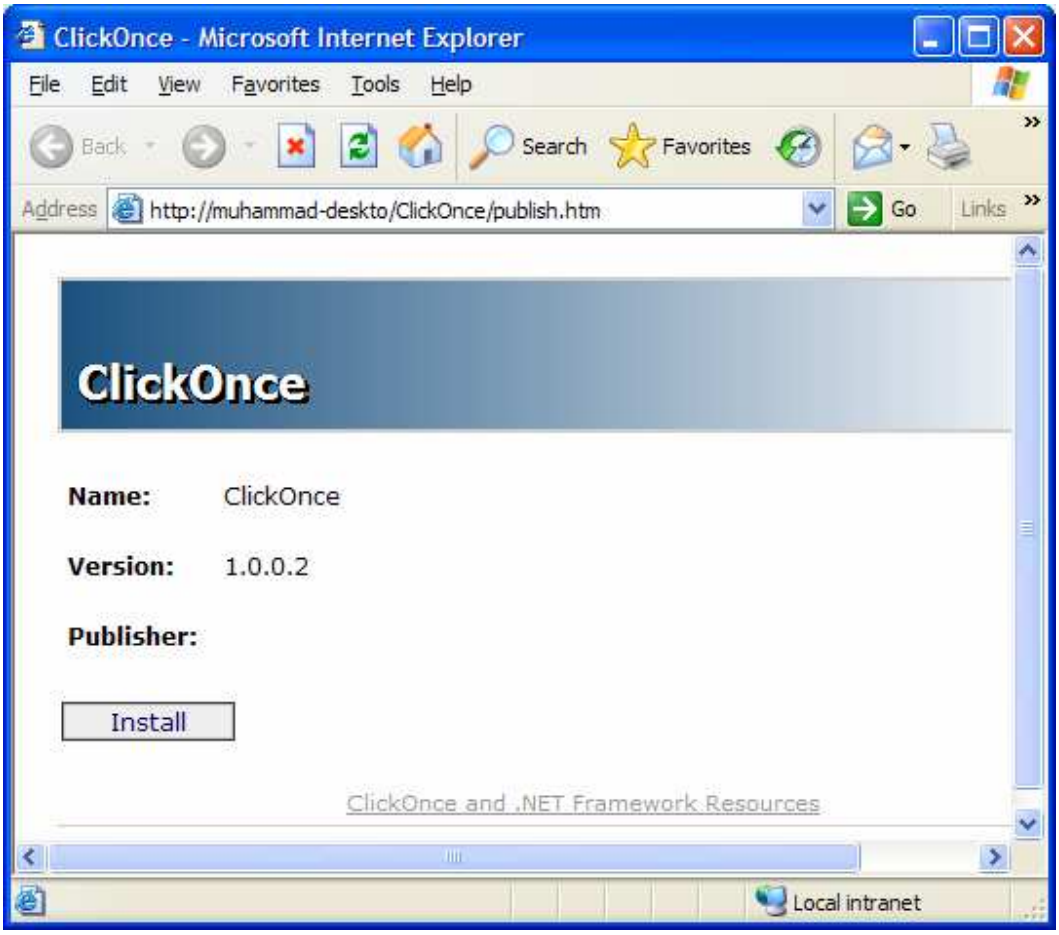
شکل ۲۱-۲

۷) روی دکمه ی `Next` کلیک کنید تا به مرحله ی دوم ویزارد بروید. در این قسمت می توانید انتخاب کنید که آیا برنامه شورت کات خود را در منوی استارت و قسمت `Add or Remove Programs` قرار دهد یا نه؟ همانند شکل ۲۱-۳ گزینه ی `Yes` را انتخاب کنید.



شکل ۲۱-۳

۸) با کلیک روی دکمه ی `Next` اطلاعاتی در مورد تنظیمات انجام شده طی این ویزارد نمایش داده خواهد شد. ریو دکمه ی `Finish` کلیک کنید تا ویزارد بسته شود. به این ترتیب صفحه ی وبی که کاربران برای نصب برنامه مشاهده خواهند کرد نمایش داده خواهد شد (شکل ۲۱-۴). روی لینک `Install` کلیک کنید تا نصب برنامه شروع شود.



شکل ۲۱-۴

۹) هنگامی که لینک `Install` را در این صفحه فشار دهید، ممکن است یک هشدار امنیتی به وسیله ی ویندوز همانند شکل ۲۱-۵ نمایش داده شود. اگر چنین پنجره ای نمایش داده شد، روی دکمه ی `Install` کلیک کنید تا نصب برنامه ادامه پیدا کند و سپس فرم برنامه نمایش داده شود. در برنامه، روی دکمه ی `Version` کلیک کنید. مشاهده خواهید کرد که عبارت `Version 1.0` در فرم نوشته می شود.

۱۰) به فولدر مربوط به `Program Files` بروید. مشاهده خواهید کرد که هیچ فایلی مربوط به این برنامه در این فولدر قرار نگرفته است، اما شورت کاتی برای اجرای این برنامه به قسمت `All Programs` از منوی استارت اضافه شده است. حال برنامه را تغییر خواهیم داد، تا نحوه ی به روز شدن آن را نیز مشاهده کنید.

۱۱) به برنامه ی `ClickOnce` در ویژوال استودیو برگردید و کد موجود در متد `btnVersion_Click` را به صورت زیر تغییر دهید تا عبارت `Version 1.1` را در فرم برنامه نمایش دهد:

```
private void btnVersion_Click(object sender, EventArgs e)
{
      lblVersion.Text = "Version 1.1";
}
```



شکل ۲۱-۵

۱۲) عملکرد برنامه را تست کرده و سپس آن را کامپایل کنید.

۱۳) در پنجره ی Solution Explorer روی نام پروژه کلیک راست کرده و از منوی باز شده گزینه ی Properties را انتخاب کنید. سپس در پنجره ای که نمایش داده می شود روی لبه ی Publish در سمت چپ کلیک کنید.

۱۴) به گزینه هایی که در این قسمت وجود دارد نگاه کنید. تمام تنظیماتی که در ویزارد انجام دادیم، در این صفحه وجود دارند و می توانیم آنها را تغییر دهیم. بعد از ایجاد تغییرات مورد نظر خود در این قسمت کافی است که در پایین صفحه روی دکمه ی Publish Now کلیک کنید.

۱۵) مجدداً صفحه ی نصب نمایش داده می شود، اما در این قسمت لازم نیست که روی لینک Install کلیک کنید. می توانید این فرم را ببندید.

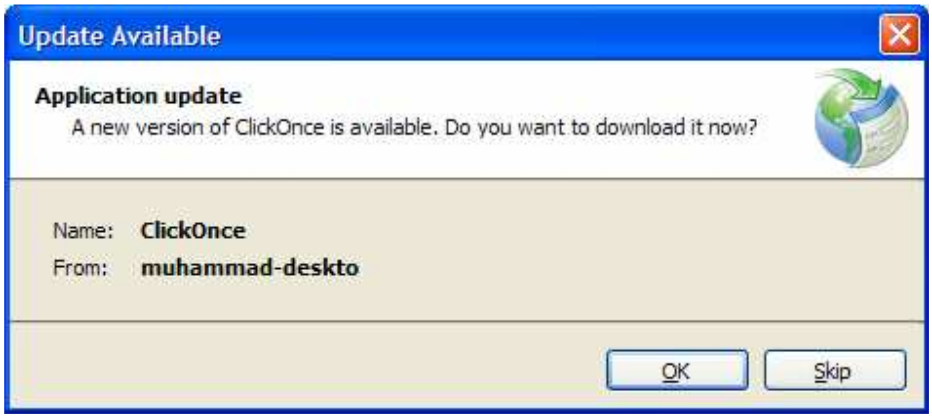
۱۶) حال با استفاده از منوی Start برنامه را اجرا کرده و روی دکمه ی Version کلیک کنید. پیغامی نمایش داده شده و اعلام می کند که برنامه تغییر کرده است. آیا می خواهید آن را به روز کنید (شکل ۲۱-۶)؟ روی دکمه ی Yes کلیک کنید. بعد از اینکه فرم برنامه باز شد، روی دکمه ی Version کلیک کنید. مشاهده خواهید کرد که عبارت Version 1.1 در صفحه نمایش داده خواهد شد.

چگونه کار می کند؟

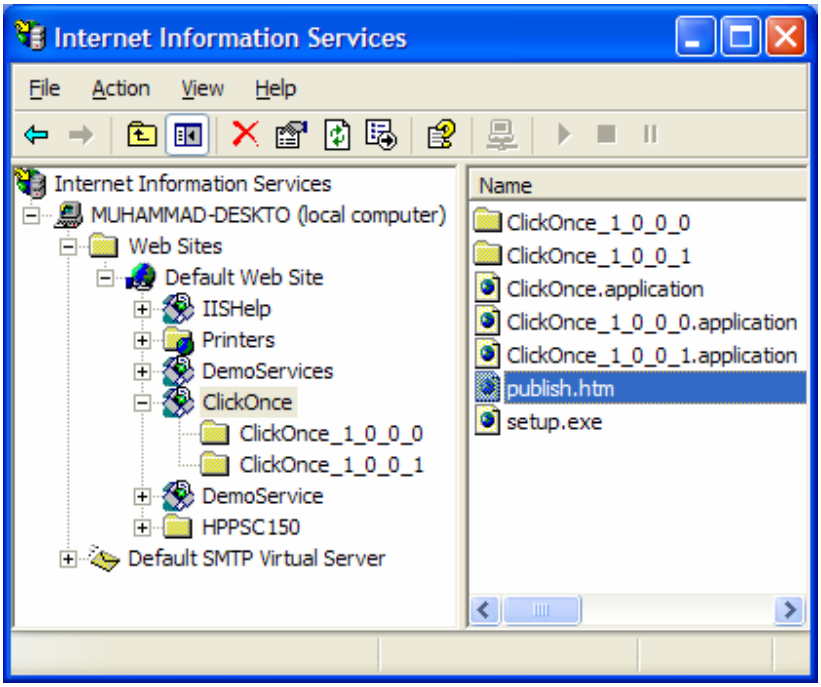
مشاهده کردید که توزیع برنامه با استفاده از این روش بسیار ساده بود. بعد از چند بار کلیک کردن، توانستید که یک برنامه ی ویندوزی را به گونه ای توزیع کنید تا بتواند در مواقع ضروری خود را به روز کند. ویژوال استودیو با انجام دادن کارهای زیادی سعی کرده است که توزیع برنامه های کاربردی با این روش را تا حد ممکن ساده نگه دارد.

ابتدا مکانی را در وب سرور مشخص کردیم تا فایلهای مربوط به نصب برنامه در آن قرار گیرند و افراد دیگر بتوانند به این فایلها دسترسی داشته باشند. برای مشاهده ی فایلهایی که در فولدر مجازی مربوط به این برنامه در وب سرور قرار گرفته اند، IIS

MMC را باز کنید[1]. به این ترتیب پنجره ای مشابه شکل ۲۱-۷ نمایش داده خواهد شد. توجه کنید که هر نسخه از برنامه دارای یک فولدر مخصوص به خود است. به صورت پیش فرض اگر چارچوب .NET در کامپیوتر کاربر نصب نشده باشد، با استفاده از سایت مایکروسافت این قسمت به صورت اتوماتیک نصب خواهد شد.



شکل ۲۱-۶



شکل ۲۱-۷

در مرحله ی بعد از ویزارد می توانید تعیین کنید که آیا برنامه بدون اتصال به اینترنت نیز می تواند قابل دسترسی باشد یا نه؟ در صورتی که بخواهید برنامه بدون اتصال به اینترنت نیز قابل دسترسی باشد، یک نسخه از آن در قسمتی از کامپیوتر شما قرار خواهد گرفت و همچنین شورت کات مربوط به اجرای آن نیز به منوی Start و قسمت Add or Remove

[1] برای مشاهده ی این قسمت حتماً باید ابتدا IIS را در کامپیوتر خود نصب کنید. سپس می توانید از قسمت Administrative Tools گزینه ی Internet Information Services را انتخاب کنید تا پنجره ی مربوط به IIS MMC نمایش داده شود.

Programs اضافه خواهد شد. در غیر این صورت کاربر هر بار که بخواهد برنامه را اجرا کند باید به وب سرور مربوطه متصل شود.
همین بود. هنگامی که روی دکمه ی Finish کلیک می کنید ویژوال استودیو تمام کارهای لازم را انجام می دهد. البته هیچ کار جادویی و یا عجیبی در این قسمت رخ نمی دهد و خودتان نیز می توانید به صورت دستی و بدون ویژوال استودیو این کار را انجام دهید.
به شکل ۲۱-۷ نگاه کنید تا توضیح دهم در این قسمت چه اتفاقی رخ داده است. برای این کار ابتدا یک دایرکتوری مجازی[1] در IIS ایجاد می کنیم. این فولدر مکانی است که برنامه به وسیله ی آن توزیع می شود. سپس بازای هر نسخه از برنامه، یک دایرکتوری جدید ایجاد خواهیم کرد و فایلهای مربوط به آن نسخه را در دایرکتوری ایجاد شده قرار می دهیم.
بقیه ی فایلهای مورد نیاز برای توزیع برنامه در دایرکتوری اصلی قرار خواهند گرفت. این فایلها شامل یک صفحه ی وب به نام Publish.htm و یک فایل اجرایی برای توزیع برنامه به نام setup.exe و ...است. به این ترتیب هر بار که بخواهیم برنامه را اجرا کنیم، ابتدا به صفحه ی publish.htm متصل خواهیم شد. سپس یک بررسی انجام می شود تا مشخص شود نسخه ای از برنامه که در حال استفاده از آن هستیم، آخرین نسخه است و یا نسخه ی جدید تر از آن نیز در سرور قرار داده شده است؟ اگر نسخه ی جدید تری در سرور قرار داشته باشد، پیغامی نمایش داده می شود و این اجازه را می دهد تا بتوانیم برنامه را به روز کنیم.

توزیع برنامه با استفاده از روش XCOPY:

در سیستم عامل DOS دستوری به نام XCOPY وجود داشت که یک فولدر و تمام محتویات درون آن را به مکان دیگری کپی می کرد. نام این روش نیز از این دستور گرفته شده است، زیرا عملکرد هر دو نسبتاً مشابه است. این روش توزیع برنامه ها اغلب برای برنامه های مبتنی بر وب به کار می رفت، اما با استفاده از .NET می توان برنامه های ویندوزی را نیز به همین روش توزیع کرد.
می دانید که یک اسمبلی در .NET می تواند بدون ثبت شدن در رجیستری و یا انجام کارهایی از این قبیل اجرا شود و فقط لازم است که فایلهای مورد نیاز آن در فولدری که اسمبلی در آن قرار دارد کپی شود. این مورد شامل یک برنامه ی ویندوزی تحت .NET نیز می شود. به عبارت دیگر فایل اجرایی یک برنامه ی تحت .NET برای اینکه بتواند در یک سیستم دیگر اجرا شود، فقط به فایلهایی نیاز دارد که در فولدر آن قرار دارد[2]. بنابراین با انتقال فولدر مربوط به برنامه به همراه تمام محتویات آن (همانند دستور XCOPY) می توان برنامه را به سیستم دیگری انتقال داد. البته دقت کنید که در این صورت برنامه نباید از اسمبلی های مشترک که در GAC نصب می شوند استفاده کند و فقط می تواند اسمبلی های موجود در .NET و یا اسمبلی هایی که در فولدر برنامه قرار می گیرند را به کار برد.

ایجاد یک برنامه ی نصب با استفاده از ویژوال استودیو ۲۰۰۵:

[1] منظور از دایرکتوری مجازی، فولدری در وب سرور است که افراد خارج از وب سرور با استفاده از نامی خاص می توانند به اطلاعات درون آن دسترسی داشته باشند. برای مثال در اینجا یک فولدر به نام ClickOnce در کامپیوتر وب سرور ایجاد کرده و سپس نام ClickOnce/ را به آن اختصاص می دهیم. به این ترتیب کاربر می تواند با مشخص کردن نام وب سرور و سپس مشخص کردن نام دایرکتوری مجازی مورد نظر خود، به اطلاعات درون آن دسترسی داشته باشد. برای مثال در اینجا برای دسترسی به اطلاعات این دایرکتوری مجازی می توانیم از آدرس http://localhost/ClickOnce استفاده کنیم.
[2] البته .NET Framework نیز باید در سیستم مقصد نصب شده باشد.

ویژوال استودیو ۲۰۰۵ برای ایجاد برنامه های نصب از نصب کننده ی ویندوز استفاده می کند. سرویس نصب کننده ی ویندوز، یک پایگاه کلی برای نصب برنامه های گوناگون در سیستم عامل ویندوز است. این سرویس قابلیتهای زیادی از قبیل قابلیت حذف برنامه های نصب شده، قابلیت نصب تراکنشی یک برنامه (حذف برنامه از کامپیوتر، در صورتی که عمل نصب آن در هر مرحله ای با شکست مواجه شود) و بسیاری دیگر را فراهم می کند. بسیاری از این قابلیت ها به صورت درونی قابل دسترس هستند و لازم نیست برای استفاده از آنها تنظیم خاصی را انجام دهید. برای استفاده از قابلیتهای دیگر نیز می توانید نصب کننده ی ویندوز را به صورت مورد نظر خود تنظیم کنید.
پشتیبانی ویژوال استودیو از نصب کننده ی ویندوز باعث شده است که به سادگی بتوان با استفاده از کادر New Project در ویژوال استودیو، برای یک برنامه ی ایجاد شده در .NET یک برنامه ی نصب ایجاد کرده و از آن استفاده کرد.
برای ایجاد یک پروژه ی نصب برای یک برنامه، ویژوال استودیو دارای پنج قالب کلی در کادر New Project است:

- **Setup Project** برای ایجاد نصب کننده ی برنامه های ویندوزی و یا برنامه های عمومی.
- **Web Setup Project** برای ایجاد نصب کننده ی برنامه های تحت وب و یا وب سرویس ها.
- **Merge Module** برای ایجاد بسته ای که فقط بتواند در یک پروژه ی نصب کننده ی دیگر ترکیب شود.
- **Cab Project** برای ایجاد یک بسته ی خاص که به عنوان نوعی از نصب کننده می تواند مورد استفاده قرار گیرد.
- **Smart Device Cab Project** برای ایجاد نصب کننده ی برنامه های مربوط به دستگاه های هوشمند، مانند PDA ها مورد استفاده قرار می گیرد.

همچنین در این کادر، قالبی به نام Setup Project Wizard قرار دارد که می تواند در ایجاد مدلهای مختلف برنامه های نصب که در بالا عنوان شد به شما کمک کند.

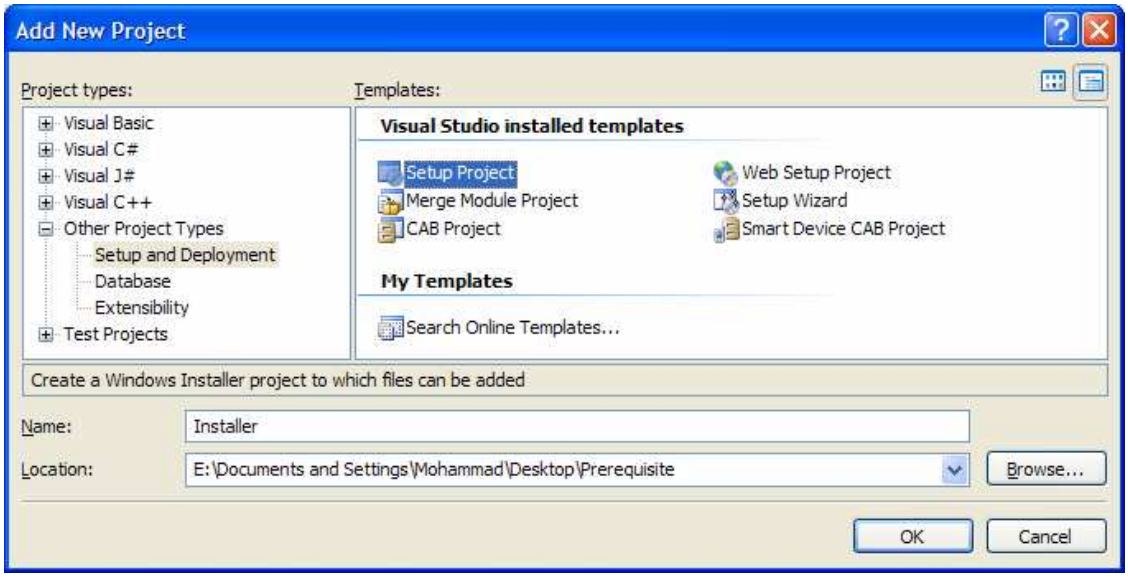
ایجاد یک برنامه ی نصب کننده:

هنگام ایجاد برنامه های نصب کننده، باید همواره کاربر نهایی را در نظر داشته باشید. تمام برنامه هایی که با استفاده از ویژوال استودیو ۲۰۰۵ ایجاد می شوند، برای اینکه در سیستم دیگری اجرا شوند به پیش نیازهایی در سیستم مقصد نیاز دارند. این پیش نیازها بر اساس نوع برنامه و اینکه در طراحی آن از چه ابزارهایی استفاده شده است تفاوت دارد. یکی از این پیش نیازها (و مهمترین آنها) .NET Framework نسخه ی 2.0 است که باید در کامپیوتر مقصد نصب شده باشد.
برای برنامه هایی که به صورت درونی و در بین کاربران موجود در مکانی خاص توزیع می کنید، معمولاً می دانید که چه قسمتهای پیش نیازی در کامپیوتر های کاربران نصب شده است و چه قسمتهایی باید همراه با برنامه ی اصلی نصب شود. اما در بسیاری از شرایط که می خواهید یک برنامه را به صورت عمومی توزیع کنید، از سیستم کاربر مقصد و اینکه چه قسمتهایی در آن نصب شده است اطلاعی ندارید. در این مواقع این مورد به شما بستگی دارد که چگونه همه ی قسمتهای مورد نیاز در برنامه را در دسترس کاربر قرار دهید.
ویژوال استودیو ۲۰۰۵ پروسه ی اضافه کردن پیش نیازهای لازم در یک برنامه به نصب کننده ی آن را تا حد ممکن ساده کرده است. بیشتر این پیش نیازها را می توان با تیک زدن نام آنها انتخاب کرد. یکی از این پیش نیازها که به صورت پیش فرض انتخاب شده است، .NET Framework نسخه ی 2.0 است. از آنجا که تا این برنامه در سیستمی نصب نشده باشد، برنامه های نوشته شده در .NET نمی توانند در آن سیستم اجرا شوند، ویژوال استودیو ۲۰۰۵ این برنامه را به صورت پیش فرض به نصب کننده اضافه می کند تا در زمان نصب، در صورت لزوم .NET Framework نیز نصب شود.

در قسمت امتحان کنید بعد، یک برنامه ی نصب کننده را با استفاده از ویژوال استودیو ۲۰۰۵ ایجاد خواهیم کرد.

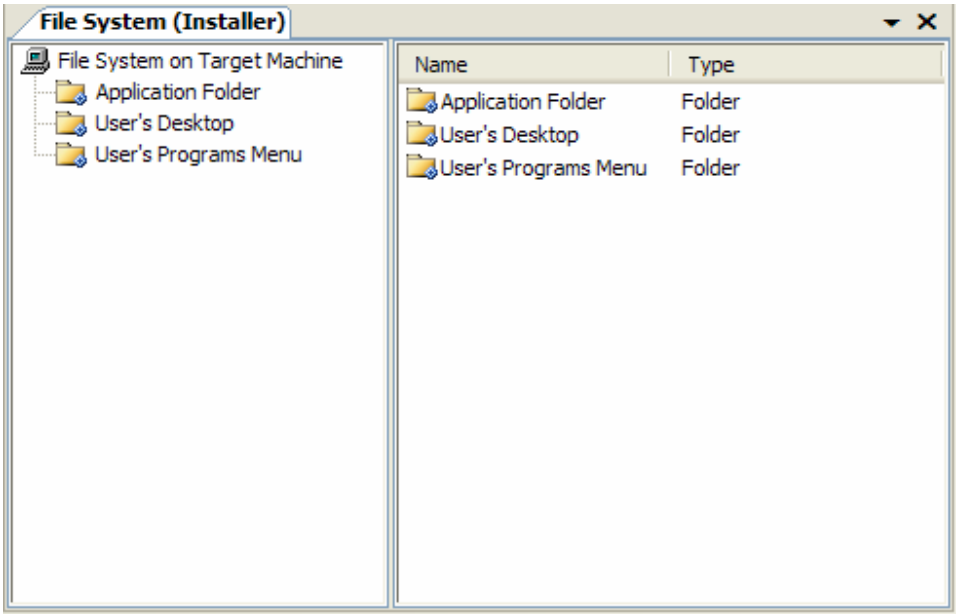
امتحان کنید: ایجاد یک برنامه ی نصب کننده

۱) با استفاده از ویژوال استودیو ۲۰۰۵ یک برنامه ی ویندوزی جدید به نام `Prerequisite` ایجاد کنید. لازم نیست در کد و یا ظاهر فرم این برنامه تغییری ایجاد کنید.
۲) برنامه را ذخیره کرده و سپس آن را کامپایل کنید.
۳) با استفاده از نوار منوی ویژوال استودیو گزینه ی `File ➔ Add ➔ New Project...` را انتخاب کرده تا کادر `Add New Project` نشان داده شود. در قسمت `Project Type` از این کادر، گزینه ی `Setup And Deployment` در `Other Project Types` را انتخاب کنید. از قسمت `Templates` نیز گزینه ی `Setup Project` را انتخاب کرده و نام پروژه را نیز برابر با `Installer` قرار دهید (شکل ۲۱-۸).



شکل ۲۱-۸

هنگامی که ویژوال استودیو این پروژه را ایجاد کرد، یک قسمت طراحی همانند شکل ۲۱-۹ نمایش داده خواهند شد. در سمت چپ این قسمت سه فولدر اصلی نشان داده شده است: `Application Folder`، `User's Desktop` و `User's Program Menu`.



شکل ۲۱-۹

۴) در پنجره ی Solution Explorer روی نام پروژه ی Installer کلیک راست کرده و از منویی که نمایش داده می شود گزینه ی Properties را انتخاب کنید.

۵) در کادر مربوط به Properties، روی دکمه ی ...Prerequisites کلیک کنید تا کادر Prerequisites همانند شکل ۲۱-۱۰ نشان داده شود. توجه کنید که .NET Framework 2.0 به صورت پیش فرض انتخاب شده است.

۶) در این کادر گزینه ی Microsoft Data Access Components 2.8 را انتخاب کرده و در هر دو کادر روی دکمه ی OK کلیک کنید. توجه کنید که به صورت پیش فرض، مؤلفه هایی که در این قسمت انتخاب می شوند از سایت شرکت تولید کننده ی آنها دانلود خواهد شد.

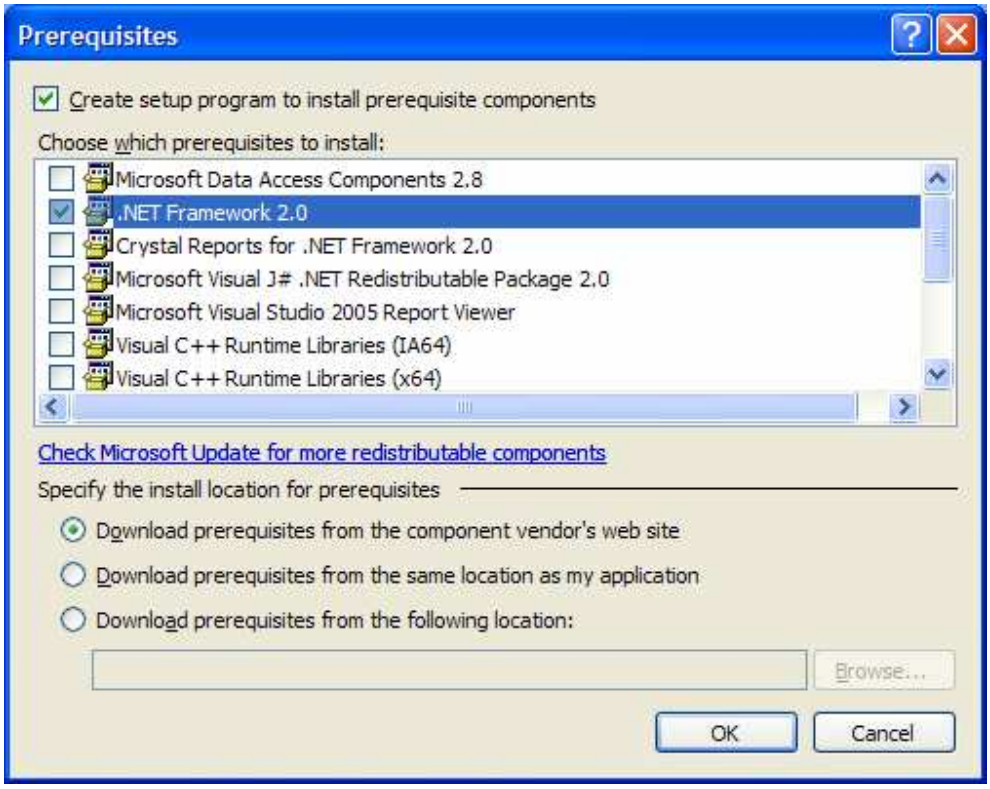
۷) در سمت چپ بخش طراحی روی گزینه ی Application Folder کلیک راست کرده و گزینه ی Add ➔ Project Output را انتخاب کنید تا فرمی همانند شکل ۲۱-۱۱ نشان داده شود.

۸) سپس در کادر Add Project Output Group گزینه ی Primary Output را انتخاب کرده و روی دکمه ی OK کلیک کنید.

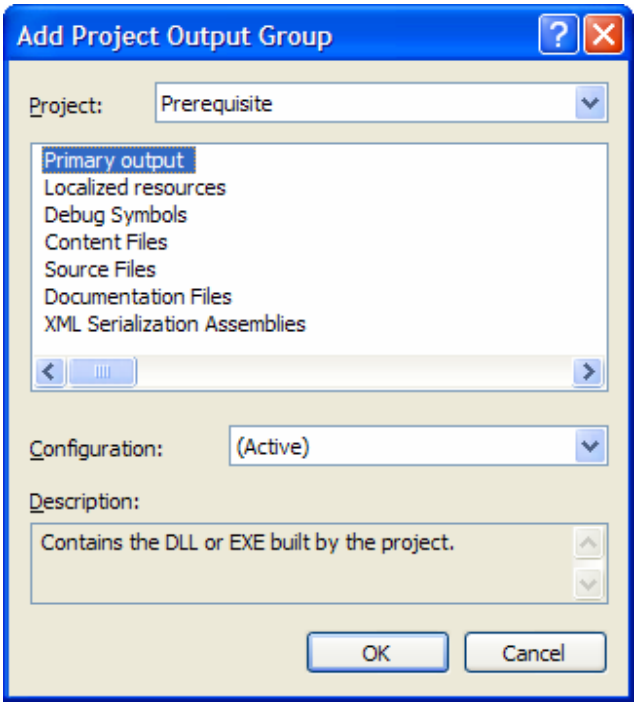
۹) حال روی Primary output from prerequisite که به قسمت سمت راست اضافه شده است، کلیک راست کرده و از منویی که باز می شود گزینه ی Create a Shortcut to Primary output from prerequisite را انتخاب کنید. نام گزینه ای که اضافه می شود را برابر با Prerequisite قرار دهید و سپس روی آن کلیک راست کرده و گزینه ی Cut را انتخاب کنید. سپس در قسمت چپ روی گزینه ی User's Program Menu کلیک راست کرده و گزینه ی Paste را انتخاب کنید تا شورت کات ایجاد شده در این قسمت قرار گیرد.

۱۰) پروژه ی Installer را ذخیره کرده و سپس آن را کامپایل کنید.

۱۱) با استفاده از پنجره ی Solution Explorer روی پروژه ی Installer کلیک راست کرده و گزینه ی Install را انتخاب کنید. به این ترتیب برنامه ی نصبی که رد این قسمت ایجاد کردیم، اجرا خواهد شد. با اتمام نصب آن، فایلهای لازم به قسمتهایی که مشخص کرده بودید اضافه خواهند شد.



شکل ۲۱-۱۰



شکل ۲۱-۱۱

چگونه کار می کند؟

هنگامی که می خواهید یک برنامه ی نصب ایجاد کنید، ویژوال استودیو یک برنامه از نوع Windows Installer ایجاد خواهد کرد. در این مثال فقط فایل اجرایی برنامه را در نصب کننده قرار دادیم، تا در سیستم مقصد کپی کند، اما می توان فایلهای دیگری را نیز از قبیل فایلهای راهنما، فایلهای متنی، فایلهای اسمبلی های مورد استفاده و ... را نیز به این قسمت اضافه کرد.
هنگامی که پروژه ی نصب را کامپایل می کنید، دو نوع فایل ایجاد می شوند:

- یک فایل با پسوند msi که حاوی برنامه ی نصب کننده است.
- یک فایل برای بارگذاری برنامه ی نصب کننده به نام setup.exe

این دو فایل در فولدر <solution directory>\Installer\Release قرار می گیرند. برای پیدا کردن آدرس مربوط به solution نیز می توانید آن را از پنجره ی Solution Explorer انتخاب کرده و سپس خاصیت Path را در کادر Properties مشاهده کنید.
با اضافه کردن دو پیش نیاز نصب در این پروژه، قبل از اینکه برنامه نصب شود بررسی می شود که آیا این دو مورد در سیستم مقصد وجود دارند یا نه؟ در صورتی که وجود نداشته باشند، فایلهای مربوط به آنها از سایت سازنده دریافت شده و نصب خواهد شد.

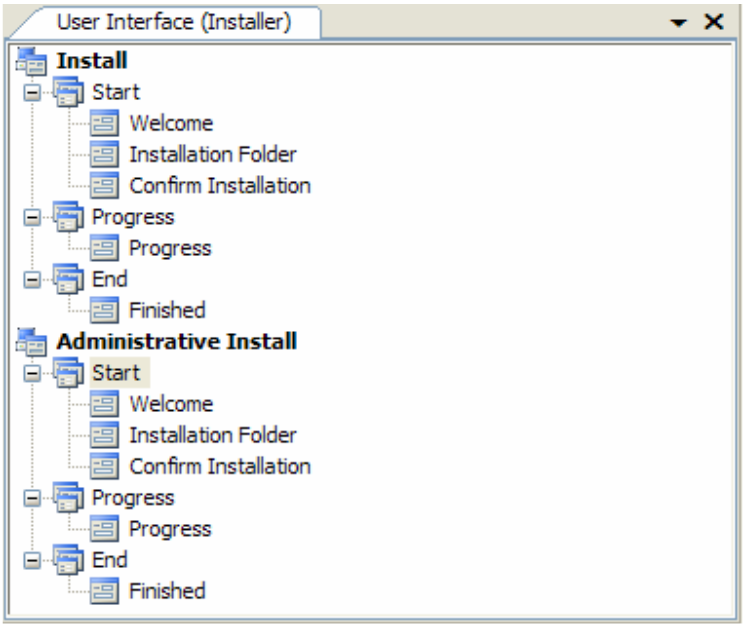
ویرایشگر رابط کاربری برنامه ی نصب:

برنامه ی نصب کننده ای که به وسیله ی ویژوال استودیو ایجاد می شود، می تواند به گونه ای پیکر بندی شود تا دقیقاً برنامه ی نصب مورد نظر شما را ایجاد کند. یکی از ساده ترین راهها برای ایجاد تغییر در برنامه ی نصب، این است که صفحات مربوط به نصب برنامه به صورت مورد نظر خود تغییر دهید. به این ترتیب برنامه ی نصبی که ایجاد می کنید حرفه ای تر به نظر خواهد رسید.
برای این کار می توانید از ابزار مربوط به ویرایش رابط کاربری برنامه ی نصب استفاده کنید.
با استفاده از این ابزار می توانید تعیین کنید که چه صفحاتی برای انجام پروسه ی نصب باد نمایش داده شوند. برای انتخاب صفحات می توانید از فرمهای پیش ساخته ی موجود در ویژوال استودیو استفاده کنید (مانند فرم مربوط به حق تالیف نرم افزار) و یا فرمهای قابل تنظیم را به کار ببرید. برای مثال می توانید فرمی را اضافه کنید تا در هنگام نصب، شماره سریال نرم افزار را از کاربر بپرسد و در صورت صحیح بودن آن، نصب را ادامه دهد.
در قسمت امتحان کنید بعد، رابط کاربری برنامه ی نصبی که در قسمت قبل ایجاد کرده بودیم را تغییر خواهیم داد.

امتحان کنید: تنظیم رابط کاربری

۱) با استفاده از ویژوال استودیو ۲۰۰۵ برنامه ی جدید از نوع Setup Project به نام UserInterface ایجاد کنید.
۲) با استفاده از نوار منوی ویژوال استودیو گزینه ی View ➔ Editor ➔ User Interface را انتخاب کنید.

۳) به این ترتیب ویرایشگر مربوط به رابط کاربری نصب همانند شکل ۲۱-۱۲ نشان داده خواهد شد.



شکل ۲۱-۱۲

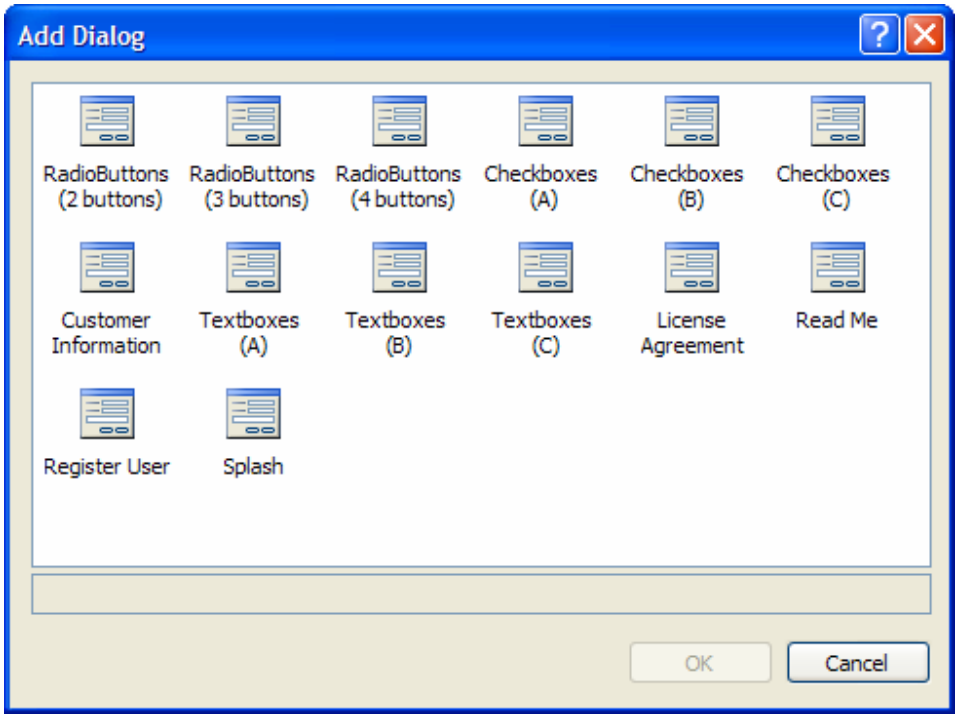
۴) در این کادر، دو قسمت عمده را مشاهده خواهید کرد: قسمت `Install` و قسمت `Administrative Install` که ظاهر هر دوی آنها را می توانید تغییر دهید. قسمت `Administrative Install` برای نوع خاصی از نصب است که در این قسمت در مورد آن صحبتی نخواهیم کرد. این قسمت زمانی به کار می رود که یک مدیر شبکه بخواهد برنامه را برای تمام کاربران در شبکه نصب کند.

۵) در قسمت `Install` روی گزینه ی `Start` کلیک راست کرده و از منویی که باز می شود گزینه ی `Add Dialog` را انتخاب کنید. به این ترتیب کادری مشابه شکل ۲۱-۱۳ نشان داده خواهد شد.

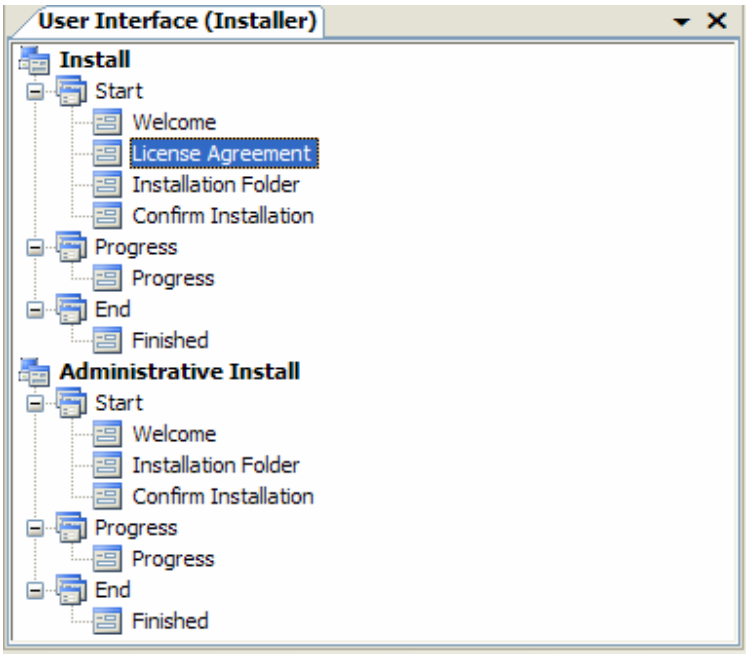
۶) از این کادر گزینه ی `License Agreement` را انتخاب کرده و روی دکمه ی `OK` کلیک کنید. به صورت پیش فرض این کادر بعد از آخرین کادر موجود در قسمت `Start` اضافه خواهد شد، اما می خواهیم که این کادر به عنوان دومین کادر در زمان نصب نمایش داده شود. بنابراین روی آن کلیک راست کرده و گزینه ی `Move Up` را انتخاب کنید تا به عنوان دومین کادر قرار بگیرد. به این ترتیب قسمت `User Interface` همانند شکل ۲۱-۱۴ خواهد بود.

۷) در این فرم می توانید حقوق مربوط به توزیع و انتشار این برنامه و کپی رایت آن را بنویسید. برای این کار می توانید از خاصیت `LicenseFile` استفاده کرده و یک فایل متنی را برای آن به عنوان حقوق کپی رایت برنامه مشخص کنید. در این مثال، این خاصیت را تنظیم نخواهیم کرد.

۸) حال باید یک کادر `Customer Information` را به قسمت `Start` اضافه کنیم به گونه ای که به عنوان کادر سوم در فرم نمایش داده شود. بعد از این کار خاصیت `SerialNumberTemplate` را برابر با `%%-###-%%%` و خاصیت `ShowSerialNumber` را برابر با `True` قرار دهید.



شکل ۲۱-۱۳

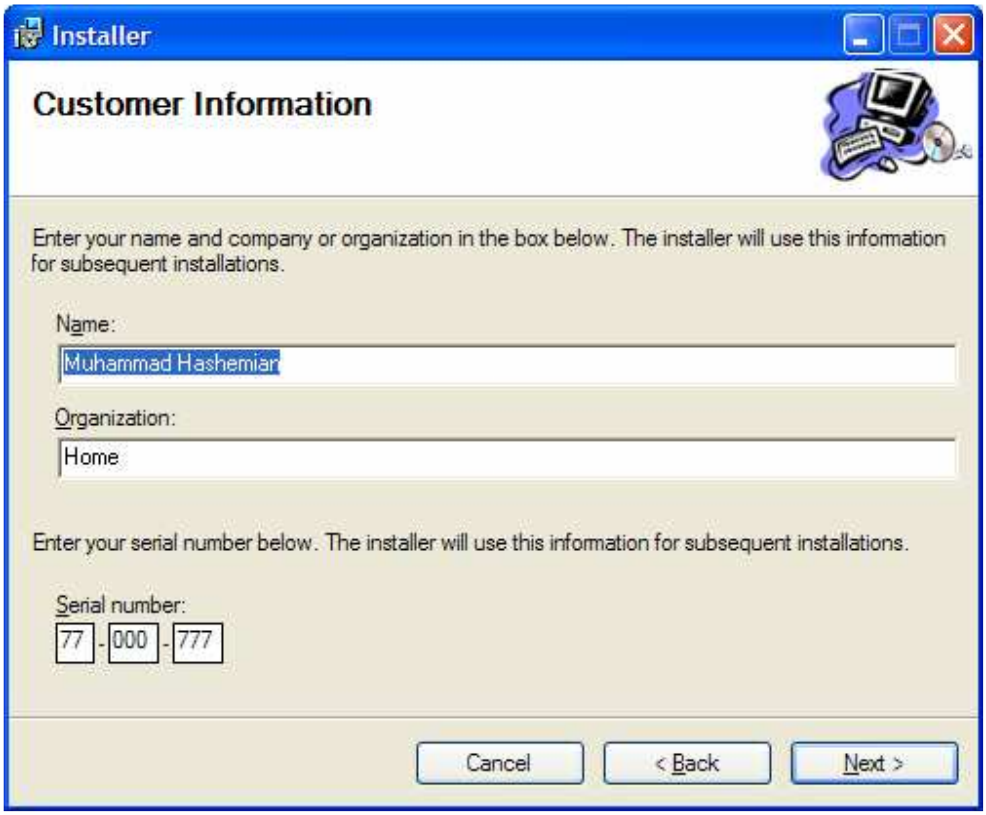


شکل ۲۱-۱۴

۹) خوب، حال برنامه را کامپایل کرده و با کلیک کردن روی پروژه ی آن در `Solution Explorer` و انتخاب گزینه ی `Install` آن را اجرا کنید. به این ترتیب کادر مربوط به قانون کپی رایت برنامه را در مرحله ی دوم نـصب مشاهده خواهید کرد. مرحله ی سوم نیز مربوط به اطلاعات فردی است که از کاربر استفاده می کند.

۱۰) در این مرحله، عبارت 77-000-777 را به عنوان شماره سریال برنامه وارد کرده و روی دکمه ی Next کلیک کنید (شکل ۲۱-۱۵).

۱۱) حال با کلیک کردن متوالی روی دکمه ی Next برنامه ی نصب را به اتمام برسانید.



شکل ۲۱-۱۵

چگونه کار می کند؟

مشاهده کردید که ایجاد تغییر در مراحل نصب نیز، مانند تمام کارهای دیگری که در این فصل انجام دادیم، به سادگی صورت گرفت. بعد از اضافه کردن فرم مربوط به حقوق تالیف برنامه، آن را به عنوان دومین فرمی که باید در هنگام نصب نشان داده شود تعیین کردیم. به این ترتیب در مرحله ی دوم حق تالیف برنامه به کاربر نشان داده می شود و تا زمانی که کاربر آنها را قبول نکرده است نمی تواند وارد مرحله ی بعد شود.

در مرحله ی سوم اطلاعات کاربر و شماره سریال برنامه را دریافت می کنیم. تا زمانی که کاربر یک شماره ی سریال معتبر را در برنامه وارد نکند، نصب برنامه ادامه پیدا نخواهد کرد. برای ایجاد شماره سریال برنامه از خاصیت SerialNumberTemplate استفاده می کنیم و آن را به %%%-###-%% تغییر می دهیم. در این عبارت کاراکتر % نشان دهنده ی عددی است که در الگوریتم شماره سریال در نظر گرفته می شود و کاراکتر # نشان دهنده ی عددی است که در الگوریتم قرار نخواهد گرفت. نحوه ی تشخیص صحت شماره سریال به این صورت است که حاصل جمع عدد هایی که با کاراکتر % مشخص شده اند بر ۷ تقسیم می شود. در صورتی شماره سریال معتبر تشخیص داده می شود که باقیمانده ی این تقسیم

برابر با ۰ باشد. در این مثال ۵ عدد در شماره سریال با کاراکتر % مشخص شده اند که همه ی آنها را برابر با ۷ قرار داده ایم. پس حاصل جمع آنها برابر با ۳۵ خواهد بود که اگر بر ۷ تقسیم شود، باقیمانده صفر می شود.

توزیع راه حلهای گوناگون:

توزیع یک برنامه معمولا کار بسیار مشکلی است که به وسیله ی ابزارهای گوناگون سعی شده است مقداری ساده تر شود. اما برای چند لحظه به برنامه های بزرگی مانند `Office` فکر کنید. در این نوع برنامه ها تعداد زیادی فایل با نوع های مختلف وجود دارند که هر یک از آنها باید در مکان مخصوصی قرار گیرد و یا در کلید خاصی در رجیستری ثبت شود. فقط زمانی برنامه می تواند به درستی کار کند که تمام این تنظیمات به دقت انجام شوند. در این نوع برنامه ها با پیچیدگی های دیگری نیز درگیر خواهیم بود، مانند پیچیدگی های مربوط به ایجاد بانکهای اطلاعاتی مورد نیاز. برای مثال اگر بانک اطلاعاتی در حال حاضر در سیستم وجود داشت چه باید کرد؟ آیا داده هایی که هم اکنون در آن بانک اطلاعاتی وجود دارند، از بین خواهند رفت؟ این نوع مسائل و مشکلات که به نام **انتقال**[1] شناخته می شوند، معمولاً نیاز دارند که یک متخصص در این زمینه ها زمان زیادی را برای حل آنها صرف کند.
یکی درگیر از مسائلی که پروسه ی نصب را پیچیده می کند، درگیر بودن با چند نوع برنامه در یک پروژه است. برای اینکه بتوانیم در این شرایط نصب کامل و موفقی داشته باشیم، به درک صحیحی از جزئیات این پروژه ها نیاز داریم. در این قسمت سعی می کنیم آیتم ها و موارد گوناگون راجع به انواع مختلف توزیع برنامه های گوناگونی که می توان با ویژوال استودیو ۲۰۰۵ ایجاد کرد را بررسی کنیم.

اسمبلی های خصوصی:

در مورد مفهوم اسمبلی در فصلهای قبلی صحبت کردیم. اسمبلی ها خود دارای دو نوع هستند: اسمبلی های خصوصی و اسمبلی های عمومی[2]. اسمبلی های خصوصی معمولاً در یک دایرکتوری به نام `bin` درون دایرکتوری اصلی برنامه قرار می گیرند. این اسمبلی ها مخصوص برنامه هستند و نمی توانند به وسیله ی دیگر برنامه ها مورد استفاده قرار گیرند. مزیت های این نوع اسمبلی ها عبارتند از:

- این نوع اسمبلی ها به داشتن نسخه ی دقیق نیازی ندارند، زیرا همان اسمبلی که هنگام کامپایل برنامه مورد به کار رفته است در فولدر `bin` نیز وجود دارد و برنامه می تواند از آن استفاده کند.
- یک اسمبلی خصوصی نمی تواند به صورت عمومی توسط دیگر برنامه ها مورد استفاده قرار گیرد، بنابراین برنامه های دیگر نمی توانند آن را تغییر دهند و یا به روز کنند.
- برنامه ای که فقط از اسمبلی های خصوصی استفاده کرده باشد را می توان به سادگی و با استفاده از روش XCOPY توزیع کرد.
- استفاده از این اسمبلی ها به هیچ پیکر بندی و یا تنظیمات خاصی نیاز ندارد. این اسمبلی ها به سادگی کار می کنند.
- اگر از یک اسمبلی در دو برنامه استفاده کنید، چون دو نسخه از آن وجود دارد و هر نسخه نیز در فولدر مربوط به برنامه ی خود قرار گرفته است، می توانید یکی از آنها را تغییر دهید بدون آنکه در دیگری تغییری به وجود آید.

---

[1] `Migration`

[2] این دو نوع اسمبلی در نحوه ی ایجاد و یا استفاده هیچ تفاوتی با یکدیگر ندارند و تنها در سطح دسترسی و محل قرار گیری آنها است که از یکدیگر متمایز می شوند.

- این نوع اسمبلی ها برای قرار دادن کلاسهای مخصوص برنامه و یا ابزارهای کوچک عمومی بسیار مناسب هستند.

معایب استفاده از اسمبلی های خصوصی نیز عبارتند از:

- اگر بخواهید از آنها در چند برنامه استفاده کنید، مجبور خواهید بود که اسمبلی مورد نظر خود را در فولدر bin هر یک از برنامه ها کپی کنید تا بتواند در آن برنامه مورد استفاده قرار گیرد.
- مجبور خواهید بود که اسمبلی را در برنامه ی نصب مربوط به تمام پروژه هایی که از آن استفاده کرده اند قرار دهید تا هنگام نصب در سیستم مقصد کپی شود.
- اسمبلی شما به صورت قوی نامگذاری نخواهد شد. بنابراین ممکن است فردی از آن سوءاستفاده کند.

نکته: منظور از سوءاستفاده از اسمبلی شما این است که، ممکن است فردی یک اسمبلی ایجاد کند که ظاهراً مشابه اسمبلی شما باشد، اما محتویات و کد داخل آن را به صورت مورد نظر خود تغییر دهد. سپس در آن کد، کارهایی را که در نظر دارد انجام دهد.

اسمبلی های عمومی:

اسمبلی های عمومی معمولاً پایدار تر از اسمبلی های خصوصی هستند. این نوع اسمبلی ها می توانند دقیقاً مانند اسمبلی های خصوصی رفتار کنند، بنابراین تمام مزایایی که اسمبلی های خصوصی دارند را نیز شامل می شوند. اما علاوه بر این، این نوع اسمبلی ها به گونه ای دیگر نیز می توانند به کار گرفته شوند که مشابه برنامه های قبل از .NET است.
استفاده از اسمبلی های عمومی همانند روشی است که در سیستم های قدیمی به کار گرفته می شد. در ویندوز 3.1 فولدری به نام Windows\System وجود داشت و هر برنامه که می خواست از DLL خاصی استفاده کند، مجبور بود که بعد از ثبت این DLL، آن را در این فولدر کپی کرده و آن را به کار ببرد. روش کار اسمبلی های عمومی نیز به همین صورت است، ولی به جای استفاده از فولدر System از فولدر دیگری به نام Global Assembly Cache یا GAC استفاده می شود. البته بسیاری از مشکلاتی که در سیستم قبلی وجود داشت در این روش رفع شده است که درباره ی آنها در این قسمت صحبتی نخواهیم کرد.
برای اینکه یک اسمبلی عمومی را در سیستم مقصد نصب کنید، باید آن را در فولدر GAC کپی کنید. این فولدر در پروژه ی نصب که با استفاده از ویژوال استودیو ایجاد می کنید، به صورت پیش فرض دیده نمی شود. برای مشاهده ی آن باید در قسمت File System on Target Machine در بخش File System کلیک راست کرده و سپس از منوی Add Special Folder گزینه ی Global Assembly Cache Folder را انتخاب کنید.
لیست زیر شامل بعضی از مزایای اسمبلی های عمومی است:

- این اسمبلی به وسیله ی شما امضا می شود، بنابراین نمی توان از آن سوءاستفاده کرد.
- نسخه ی این اسمبلی به صورت دقیق مشخص شده است.
- این اسمبلی در یک فولدر مشخص (GAC) کپی می شود و می تواند به وسیله ی همه ی برنامه ها مورد استفاده قرار گیرد. بنابراین لازم نیست که در فولدر تمام برنامه هایی که می خواهند از آن استفاده کنند کپی شود.
- در هر لحظه، چندین نسخه از این اسمبلی می توانند به طور همزمان مورد استفاده قرار گیرند.

توزیع برنامه های ویندوزی:

در برنامه ی دومی که در این فصل تمرین کردیم، با یک نمونه از توزیع برنامه های ویندوزی آشنا شدیم. البته برنامه ای که در آن قسمت به کار بردیم بسیار ساده بود و فقط از یک فایل اجرایی تشکیل می شد، بنابراین این برنامه فقط به .NET Framework نیاز داشت و همانطور که مشاهده کردید به صورت پیش فرض در قسمت پیش نیازها انتخاب شده بود. اما در برنامه های واقعی معمولاً از کنترلهای ویندوزی، فایلهای اسمبلی حاوی کلاسهای مورد نیاز، فایلهای راهنمای برنامه و یا بسیاری از فایلهای دیگر در برنامه استفاده می کنیم. پس باشد هنگام ایجاد نصب کننده، آنها را نیز اضافه کنیم. برای اضافه کردن یک اسمبلی خصوصی به برنامه کافی است که آن را در فولدر اصلی برنامه کپی کنیم. برای اضافه کردن یک اسمبلی عمومی نیز کافی است که آن را در GAC قرار دهیم.

توزیع برنامه های مبتنی بر وب:

اگر یک برنامه ی تحت وب فقط از اسمبلی های خصوصی استفاده کرده باشد، توزیع آن بسیار ساده خواهد بود. برای این کار می توانیم با استفاده از ویژوال استودیو ۲۰۰۵ یک پروژه از نوع Web Setup ایجاد کنیم تا به وسیله ی آن برنامه ی تحت وب خود را نصب کنیم. پروژه های Web Setup برای نصب یک برنامه ی تحت وب، ابتدا یک دایرکتوری مجازی در وب سرور ایجاد می کنند و سپس فایلهای مربوط به پروژه را در دایرکتوری فیزیکی معادل آن در هارد دیسک وب سرور کپی می کنند.

نکته: برای توزیع یک برنامه ی مبتنی بر وب لازم نیست که فایلهای CS که حاوی کد C# هستند را نیز در وب سرور قرار دهید، زیرا کد موجود در این فایلها در یک اسمبلی در فولدر bin کامپایل می شود. در این نوع برنامه ها کافی است که فایلهای aspx، ascx، js، css، HTML و یا دیگر فایلهای اسکریپتی را در وب سرور قرار دهید.

توزیع وب سرویس ها:

توزیع وب سرویس ها بسیار مشابه توزیع برنامه های وب است. در توزیع وب سرویس ها نیز باید یک دایرکتوری مجازی در وب سرور ایجاد کرده و فایلهای مورد نیاز را در آن کپی کنید. البته این فایلها با فایلهای برنامه های تحت وب تفاوت دارند. برای توزیع وب سرویس ها باید فایلهای asmx و discovery را به همراه اسمبلی مربوط به وب سرویس، در سرور قرار دهید.

ابزارهای مفید:

در این قسمت بعضی از ابزارهای مفیدی که به همراه .NET عرضه می شوند و یا به صورت درونی در ویندوز وجود دارند و می توانید در برنامه های خود از آنها استفاده کنید را معرفی خواهیم کرد. هنگام ایجاد برنامه ی نصب، باید آن را در چندین سیستم مختلف تست کرد تا مشخص شود این برنامه در شرایط گوناگون چگونه کار می کند. در این مواقع اگر اوضاع به آن صورت که انتظار دارید پیش نرود، باید به صورت دستی همه ی موارد را تنظیم کنید تا مشخص کنید که کدام قسمت از کار با مشکل مواجه شده است. برای مثال فرض کنید در هنگام تست یک برنامه ی تحت وب، پروسه ی ASPNET_WP.DLL با مشکل مواجه می شود و از کار می افتد. در این شرایط باید بتوانید وب سرور را مجدداً راه اندازی کرده و سپس برنامه را دوباره تست کنید. در شرایطی

مشابه ممکن است یک اسمبلی را در GAC ثبت کنید، اما برنامه نتواند به آن دسترسی داشته باشد. در این مواقع باید بتوانید به صورت دستی اسمبلی را نصب کنید تا متوجه شوید در کدام مرحله از نصب اسمبلی، مشکل به وجود می آید. لیست زیر به صورت مختصر بعضی از این ابزارها را توضیح می دهد. البته دقت کنید که تمام این ابزارها به صورت دستورات خط فرمان هستند و برای استفاده از آن باید از خط فرمان ویژوال استودیو یا Visual Studio 2005 Command Prompt استفاده کنید:

- **ASPNET_RegIIS.exe:** معمولاً اگر ASP.NET را ابتدا نصب کرده باشید و سپس IIS را در سیستم خود نصب کنید، IIS نمی تواند به درستی از ASP.NET استفاده کند. برای رفع این مشکل می توانید با استفاده از این ابزار مجدداً ASP.NET را نصب کنید تا بتواند خود را با وب سرور (IIS) تطبیق دهد.
- **IISReset:** این دستور باعث می شود که وب سرور راه اندازی شود. با استفاده از آن می توانید بدون اینکه به قسمت IIS در MMC بروید، آن را مجدداً راه اندازی کنید.
- **ILDasm:** همانطور که در قسمتهای قبلی نیز در این رابطه صحبت شد، یک فایل ایجاد شده با .NET بعد از کامپایل شدن به کد IL تبدیل می شود. برای مشاده ی کد ILای که در یک فایل قرار دارد می توانید از این ابزار استفاده کنید. همچنین این ابزار اطلاعات دیگر موجود در فایلها از قبیل Metadata، Manifest و ... را نیز نشان می دهد.
- **GACUtil:** این ابزار برای نصب کردن و یا حذف کردن یک اسمبلی از Global Assembly Cache به کار می رود. برای نصب یک اسمبلی با استفاده از این دستور در GAC می توانید از سوییچ /I و برای حذف آن می توانید از سوییچ /u استفاده کنید.
- **RegAsm:** این دستور می تواند یک اسمبلی ایجاد شده به وسیله ی .NET را به صورت یک شیئ COM[1] در سیستم ثبت کند و معمولاً زمانی به کار می رود که بخواهیم بین اشیای COM و .NET هماهنگی ایجاد کنیم. این دستور نیز دارای دو سوییچ /I و /u است که برای نصب کردن و یا حذف کردن یک کلاس به کار می رود.

نتیجه:

خوب، امیدوارم از بررسی روشهای مختلف توزیع یک نرم افزار لذت برده باشید. در قسمت اول فصل، ابتدا با اصطلاحاتی که در این زمینه وجود دارند آشنا شدیم، سپس دیدیم که چگونه می توان با استفاده از روش ClickOnce به سادگی یک برنامه را توزیع کرد. بعد از آن با روش XCOPY آشنا شدیم و همچنین نحوه ی ایجاد یک برنامه ی نصب کننده را با استفاده از ویژوال استودیو ۲۰۰۵ مشاهده کردیم. در بخش بعدی فصل با اسمبلی های خصوصی و عمومی نیز مزایا و معایب هر کدام آشنا شدیم. هدف از این فصل این نبود که تمام این موارد را به شما آموزش دهم، بلکه در این فصل سعی کرده با معرفی اجمالی آنها به شما نشان دهم که مطالب زیادی در این زمینه وجود دارد که می توان در مورد آنها مطالعه کرد.
در پایان این فیل باید با موارد زیر آشنا شده باشید:

- چگونگی توزیع یک برنامه با استفاده از روش ClickOnce.
- ایجاد یک برنامه ی نصب کننده با استفاده از ویژوال استودیو ۲۰۰۵.
- استفاده از روشهای توزیع ساده ی برنامه ها در شرایط ممکن مانند XCOPY.

---

[1] Component Object Model

- ویرایش رابط کاربری برنامه ی نصب کننده.

تمرین:

تمرین ۱:

یک برنامه ی نصب کننده برای برنامه ی `Notepad` ایجاد کنید. برای این کار ابتدا باید فایل اجرایی برنامه، یعنی `notepad.exe` را در فولدر ویندوز خود پیدا کرده و سپس آن را به برنامه ی نصب کننده اضافه کنید. برنامه ی نصب کننده را به گونه ای تنظیم کنید تا یک شورت کات از `notepad` را در منوی `Start` قرار دهد و فایل اجرایی اصلی را نیز در فولدر `Program Files` کپی کند. می توانید نام شرکت سازنده و نام نویسنده ی برنامه را نیز به دلخواه خود تغییر دهید.

تمرین ۲:

در برنامه ی نصب کننده ای که در تمرین اول ایجاد کردید، یک کادر `Splash Screen` نیز اضافه کنید تا در ابتدای نصب نمایش داده شود. سپس یک عکس با اندازه ی در حدود ۴۸۰*۳۲۰ را از کامپیوتر خود پیدا کرده و آن را در این صفحه نمایش دهید. دقت کنید قبل از اینکه بتوانید عکس را در فرم نمایش دهید، باید آن را به پروژه ی نصب کننده اضافه کنید.

فصل بیست و دوم: ایجاد برنامه های موبایل

امروزه استفاده از دستگاه های موبایل، به ویژه دستیار های دیجیتال شخصی یا PAD[1]ها به شدت در بین مردم در حال افزایش است. بازاری که در گذشته به وسیله ی سیستم عامل Palm تسخیر شده بود، امروزه در دست سیستم عامل ویندوز CE قرار گرفته است. بر اساس گزارشی که در ۱۲ نوامبر ۲۰۰۴ به وسیله ی Associated Press به چاپ رسید، در بین دستگاه های PDA ای که در نیمه ی دوم سال ۲۰۰۴ به فروش رسیده اند، سیستم عامل ویندوز CE در رتبه ی اول قرار گرفته است. از بین ۲/۸ میلیون دستگاه PAD که در این مدت در سراسر جهان به فروش رسیده است، ویندوز CE با ۴۸/۱ درصد، به عنوان پرفروش ترین سیستم عامل معرفی شده است. به همین اندازه که فروش دستگاه های PDA و نیز سیستم عامل ویندوز CE در حال افزایش است، درخواست برنامه هایی که بتوانند در این دستگاه ها و با این سیستم عامل کار کنند نیز روز به روز در حال رشد است.

به همین علت در ویژوال استودیو ۲۰۰۵ سعی شده است که طراحی این نوع برنامه ها برای سیستم عامل های ویندوز CE، Pocket PC 2003 و SmartPhone 2003 تا حد ممکن ساده شود. در طی این فصل نحوه ی ایجاد برنامه هایی که بتوانند در دستگاه های PDA و با سیستم عامل ویندوز Pocket PC اجرا شوند را بررسی خواهیم کرد.

درک محیط:

همانطور که احتمالاً تا کنون مشاهده کرده اید، PDA دستگاهی است همانند یک کامپیوتر که بسیار کوچکتر است و امکانات کمتری نیز دارد. یک دستگاه PDA معمولی در سال ۲۰۰۲ دارای ۶۴ مگا بایت رم داخلی و یک پردازنده ی ۲۰۶ مگا هرتزی است. وزن این دستگاه در حدود نیم کیلوگرم است و دارای یک صفحه نمایش ۳,۵ اینچی است که می تواند صفحه ای را با دقت ۲۴۰*۳۲۰ پیکسل و ۶۵ هزار رنگ نشان دهد. حال این دستگاه را با یک کامپیوتر عادی مقایسه کنید. یک کامپیوتر عادی دارای یک پردازنده ی ۳ گیگاهرتزی، ۱۲۰ گیگابایت هارد دیسک، ۵۱۲ مگا بایت رم و یک صفحه نمایش ۱۹ اینچی است که می تواند تصویری را با دقت ۱۶۰۰*۱۲۰۰ و بیش از ۴ بیلیون رنگ نمایش دهد. تفاوت عمده ی دیگری که بین این نوع دستگاه ها و کامپیوتر های عادی وجود دارد در این است که صفحه نمایش دستگاه های PDA کشیده است اما صفحه نمایش در کامپیوتر های عادی پهن و وسیع است. همچنین باید در نظر داشته باشید که در دستگاه های PDA زمانی که باتری تمام می شود، همه ی اطلاعات برنامه که در رم قرار دارند از بین می روند. بنابراین باید با در نظر گرفتن تمام این نکات و موارد به طراحی برنامه برای این نوع وسایل بپردازید.

حال که با محیط کار آشنا شدید، سعی می کنم امکاناتی که ویژوال استودیو در اختیار شما قرار می دهد تا ساده تر بتوانید برای این دستگاه ها برنامه بنویسید را خلاصه کنم. برای شروع .NET Compact Framework یا CF در اختیار شما قرار می گیرد. چارچوب CF یک زیر مجموعه از چارچوب .NET است که در فصلهای قبلی با آن آشنا شدید. بنابراین هم اکنون با بیشتر بخشهای آن آشنا هستید (زیرا دقیقاً مشابه چارچوب .NET است) و به سادگی و با کمترین مقدار یادگیری می توانید از آن برای طراحی برنامه ها استفاده کنید. دقیقاً مشابه چارچوب .NET، CF نیز از CLR برای اجرای برنامه ها استفاده می کند و کد هایی که به وسیله ی آن تولید می شوند به زبان MSIL هستند تا بتواند به صورت مستقل از پلت فرم عمل کند. مهمترین نکته ای که ممکن است هنگام کار با CF توجه شما را جلب کند در این است که بسیاری از کنترل هایی که در قسمتهای قبل از آنها استفاده می کردید هم اکنون وجود ندارند و یا بسیاری از متد ها و یا خاصیت هایی که در برنامه های قبلی با آنها کار می کردید، در CF

[1] Personal Digital Assistant

حذف شده اند. توزیع برنامه هایی که با CF نوشته می شوند نیز متفاوت است. برای توزیع این برنامه ها نمی توانید همانند برنامه های قبلی از CD و یا فلاپی دیسک استفاده کنید، بلکه باید ابزاری به نام ActiveSync را به کار ببرید تا بتوانید به سادگی برنامه ی خود را در بین دستگاه های PDA توزیع کنید. در این قسمت سعی می کنیم مواردی که لازم است برای طراحی برنامه های موبایل بدانیم را با جزئیات دقیق بررسی کنیم.

.Common Language Runtime

یکی از اهداف CLR که به عنوان **موتور اجرایی**[1] شناخته می شود، این است که اجازه می دهد چند برنامه به صورت همزمان و موازی با یکدیگر اجرا شوند. معمولا CLR موارد زیر را در اجرای هر برنامه کنترل و مدیریت می کند:

- مدیریت حافظه با استفاده از سیستمی به نام GC[2]
- رفع وابستگی برنامه ها به پلت فرم با استفاده از کد های MSIL.
- سیستم نوع داده ای عمومی.
- سیستم کنترل خطاها و استثنا ها.
- کامپایل برنامه ها در مواقع مورد نیاز با استفاده از JIT.

:ActiveSync

برای اینکه بتوانید PDA خود را به PCتان متصل کنید، می توانید از برنامه ی ActiveSync استفاده کنید. با استفاده از این برنامه، PDA شما به سرعت به PC متصل می شود و می تواند علاوه بر استفاده از PC، از منابع شبکه ای آن و یا حتی از اتصال اینترنت آن نیز برای متصل شدن به اینترنت استفاده کند. با استفاده از این برنامه می توانید داده های موجود در برنامه های خود را بین PDA و PC هماهنگ کنید[3]، به یک وب سرویس دسترسی داشته باشید، و یا حتی از داده های موجود در یک بانک اطلاعاتی SQL Server استفاده کنید.

از آنجایی که معمولا اغلب شما از دستگاه Pocket PC استفاده نمی کنید، به همین دلیل در این قسمت فقط به اختصار نوه ی توزیع برنامه ها در این نوع دستگاه ها را توضیح می دهم. همانطور که در فصل قبل هنگام ایجاد برنامه ی نصب کننده با استفاده از ویژوال استودیو مشاهده کردیم، یکی از انواع این برنامه ها که در کادر New Project وجود داشت، Smart Device Cab Project بود. با استفاده از این نوع پروژه می توانید از برنامه ی خود یک خروجی با پسوند .cab دریافت کرده و سپس آن را با استفاده از ActiveSync به دستگاه PDA خود انتقال دهید. سپس می توانید با استفاده از Pocket PC روی این فایل دو بار کلیک کنید تا نصب شود و بتوانید از آن استفاده کنید.

برای دریافت نرم افزار ActiveSync می توانید به قسمت Download از سایت مایکروسافت مراجعه کنید. فایل اجرایی این نرم افزار در حدود ۴ مگا بایت حج دارد و می توانید از آدرس زیر آن را دریافت کنید:

http://www.microsoft.com/downlaods/

---

[1] Execution Engine
[2] Garbage Collection
[3] Synchronization

نوع های داده ای در CF:

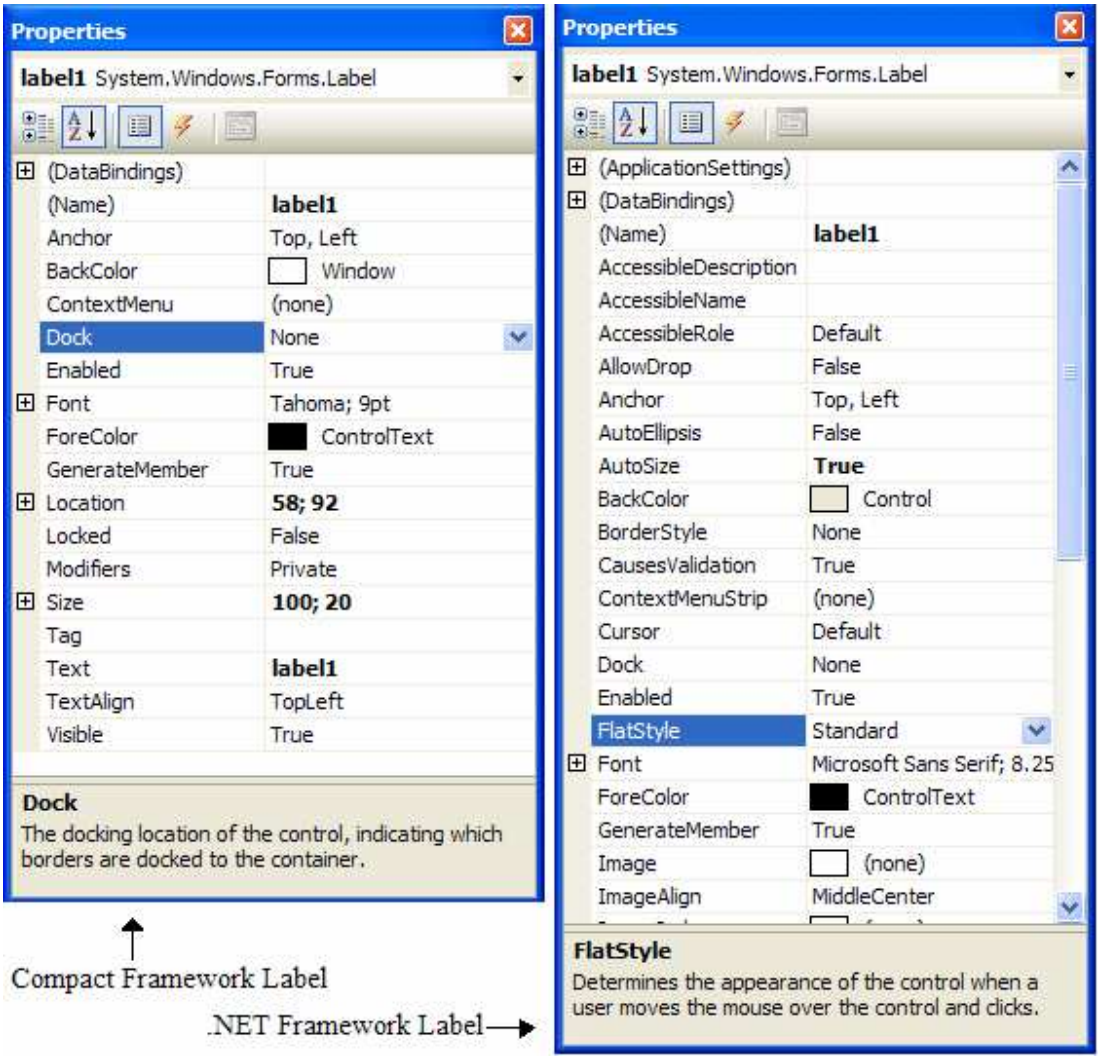
معمولاً زمانی که برنامه نویسی برای دستگاه های هوشمند را با استفاده از ویژوال استودیو ۲۰۰۵ شروع می کنید، نـوع هـای داده ای که برای آنها وجود دارند برای شما آشنا خواهد بود. جدول زیر لیستی از نوع های داده ای که در CF وجود دارند را به همراه فـضایی که اشغال می کنند و همچنین بازه هایی که می توانند داشته باشند را نشان می دهد. البته در هنگام نوشـتن برنامـه بـرای ایـن نـوع دستگاه ها باید مد نظر داشته باشید که منابع آنها بسیار محدود است و تا حد ممکن باید از آنها به صورت بهینه استفاده کرد.

نوع داده در C#	نوع داده ی عمومی در CLR	اندازه	بازه ی مقادیر
bool	Boolean	بـستگی دارد	True و یا False (درست یا غلط)
byte	Byte	۱ بایت	۰ تا ۲۵۵ بدون علامت
char	Char	۲ بایت	کاراکتر های یونیکد
DateTime	DateTime	۸ بایت	از سـاعت ۰۰:۰۰:۰ روز ۱ ژانویـه سـال ۰۰۰۱ تـا سـاعت ۱۱:۵۹:۵۹ روز ۳۱ دسامبر سال ۹۹۹۹
decimal	Decimal	۱۶ بایت	از ۰ تـا ۷۹۲۲۸۱۶۲۵۱۴۲۶۴۳۳۷۵۹۳۵۴۳۹۵۰۳۳۵ (مثبـت و منفـــی) بـــدون رقـــم اعـــشار؛ از ۰ تـــا ۷/۹۲۲۸۱۶۲۵۱۴۲۶۴۳۳۷۵۹۳۵۴۳۹۵۰۳۳۵ با بیست و هشت رقم اعشار. کوچکترین عـددی کـه مـی توانـد نمـایش دهـد ۰/۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۰۱ (+ و –) است.
double	Double	۸ بایت	از ۱/۷۹۷۶۹۳۱۳۴۸۶۲۳۱۵۷۰*E۳۰۸– تا ۴/۹۴۰۶۵۶۴۵۸۴۱۲۴۶۵۴۴*E–۳۲۴ برای اعداد منفی و ۴/۹۴۰۶۵۶۴۵۸۴۱۲۴۶۵۴۴*E-۳۲۴ تا ۱/۷۹۷۶۹۳۱۳۴۸۶۲۳۱۵۷۰*E۳۰۸ برای اعداد مثبت.
int	Int32	۴ بایت	۲۱۴۷۴۸۳۶۴۸– تا ۲۱۴۷۴۸۳۶۴۸
long	Int64	۸ بایت	۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۵۸۰۸– تا ۹۲۲۳۳۷۲۰۳۶۸۵۴۷۷۵۸۰۷
Object	Object	۴ بایت	هر نوع داده ای که بتوانـد در شـیئ ای از نـوع Object ذخیره شود.
sbyte	Sbyte	۱ بایت	۱۲۸– تا ۱۲۸
short	Int16	۲ بایت	۳۲۷۶۸– تا ۳۲۷۶۷
string	String	بـستگی دارد	از ۰ تا تقریباً ۲ بیلیون کاراکتر یونیکد

uint	UInt32	۴ بایت	۰ تا ۴۲۹۴۹۶۷۲۹۵ بدون علامت
ulong	UInt64	۸ بایت	۰ تا ۱۸۴۴۶۷۴۴۰۷۳۷۰۹۵۵۱۶۱۵ بدون علامت
ushort	UInt16	۲ بایت	۰ تا ۶۵۵۳۵ بدون علامت

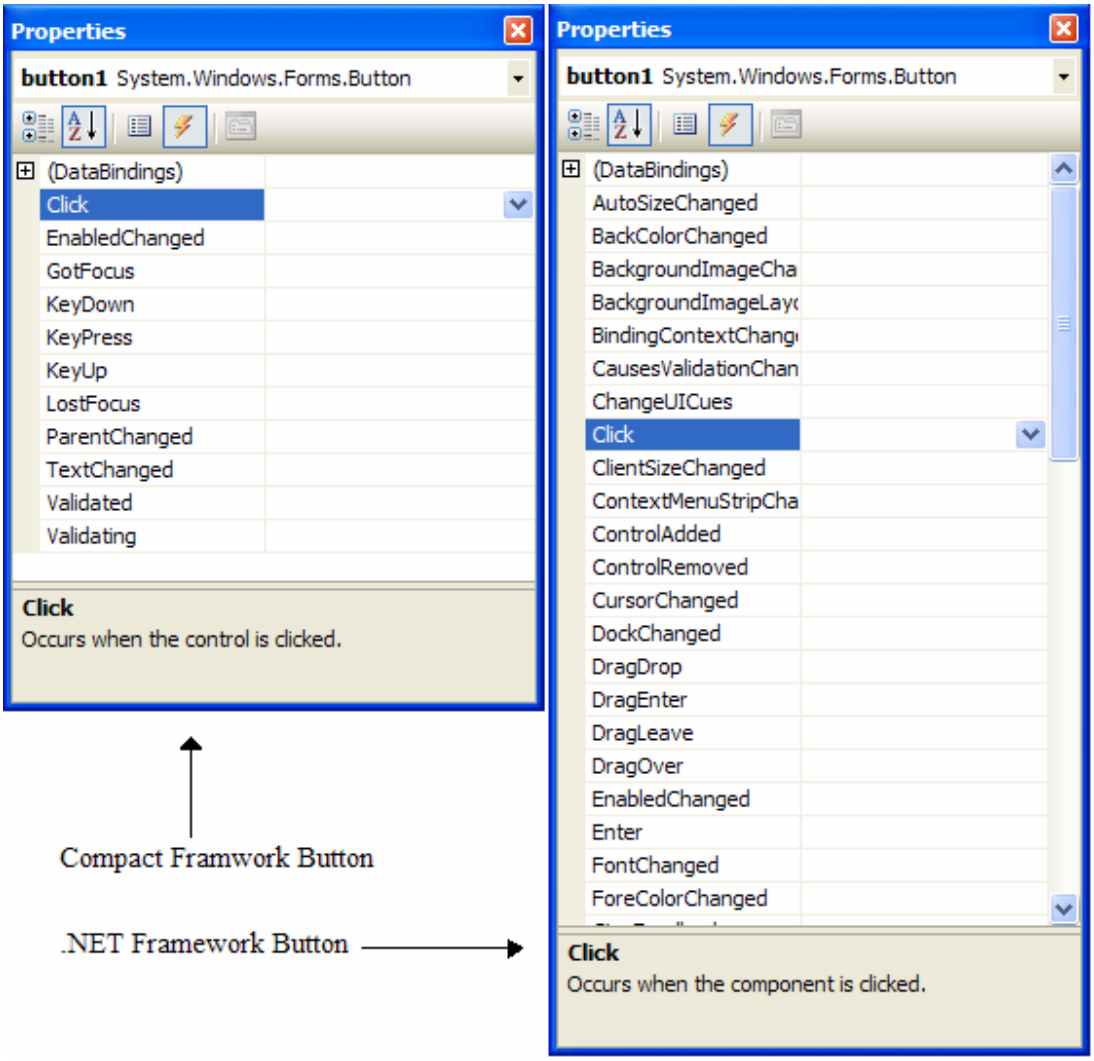
کلاسهای موجود در Compact Framework:

کلاس هایی که در CF وجود دارند، زیر مجموعه ای از کلاس هایی هستند که در چارچوب NET. با آنها آشنا شدیم به همراه تعدادی کلاس دیگر که معمولاً در برنامه های مربوط به کامپیوتر های عادی از آنها استفاده نمی شود.
چارچوب CF در حدود ۱۲ درصد کلاسهای موجود در چارچوب NET. را شامل می شود. احتمالاً با همین جمله متوجه شده اید که تا چه اندازه از کارایی کلاس هایی که در NET. وجود دارد در CF حذف شده است. به شکل ۲۲-۱ نگاه کنید. این شکل خاصیتهای مربوط به کنترل Label در یک برنامه ی معمولی را به همراه خاصیت های مربوط به کنترل Label در یک برنامه ی دستگاه های هوشمند نمایش می دهد. مشاهده می کنید که بسیاری از خاصیت های موجود حذف شده اند و قابل استفاده نیستند.



شکل ۲۲-۱

دو قسمت دیگر که در کلاسهای CF به شدت کاهش پیدا کرده اند نسخه های سربار گذاری شده از متد ها و همچنین رویداد های موجود برای کنترل ها است. شکل ۲۲-۲ لیست رویداد های موجود برای کنترل Button در هر دو نوع برنامه را نشان می دهد. در یک برنامه ی عادی که در سمت راست نشان داده شده است، تعداد رویداد ها به حدی است که نمی توان همه ی آنها را در یک صفحه مشاهده کرد. اما در یک برنامه ی CF که در سمت چپ نشان داده شده است تعداد رویدادها بسیار کمتر است.



شکل ۲۲-۲

بر اساس اندازه ی کلاسهای موجود در CF می توان حدس زد که چیزی در حدود ۷۵ درصد نسخه های سربار گذاری شده از متدهای مختلف نیز حذف شده اند. به عنوان مثال کلاس `System.IO.FileStream` در چارچوب `.NET` دارای ۱۴ متد سازنده است، اما این کلاس در CF فقط ۵ متد سازنده دارد.

در لیست زیر نام بعضی از کنترل ها که به صورت پیش فرض در CF وجود دارند و می توان برای طراحی برنامه های دستگاه های هوشمند از آنها استفاده کرد را مشاهده می کنید:

نام	نام	نام	نام
`Button`	`CheckBox`	`ComboBox`	`ContextMenu`
`DataConnector`	`DataGrid`	`DateTimePicker`	`DocumentList`
`Spiltter`	`DomainUpDown`	`HScrollBar`	`ImageList`

ListBox	LinkLabel	Label	InputPanel
Notification	MonthCalendar	MainMenu	ListView
PicturePox	Panel	OpenFileDialog	NumericUpDown
SaveFileDialog	RichInk	RadioButton	ProgressBar
Timer	TextBox	TabControl	StatusBar
VScrollBar	TreeView	TrackBar	ToolBar
			WebBrowser

ایجاد یک بازی برای Pocket PC:

به عنوان اولین برنامه برای دستگاه های هوشمند، در این قسمت یک بازی ساده طراحی خواهیم کرد و سعی خواهیم کرد در طی این برنامه با محیط این نوع دستگاه ها و صفحه نمایش آنها بیشتر آشنا شویم.

امتحان کنید: ایجاد یک بازی

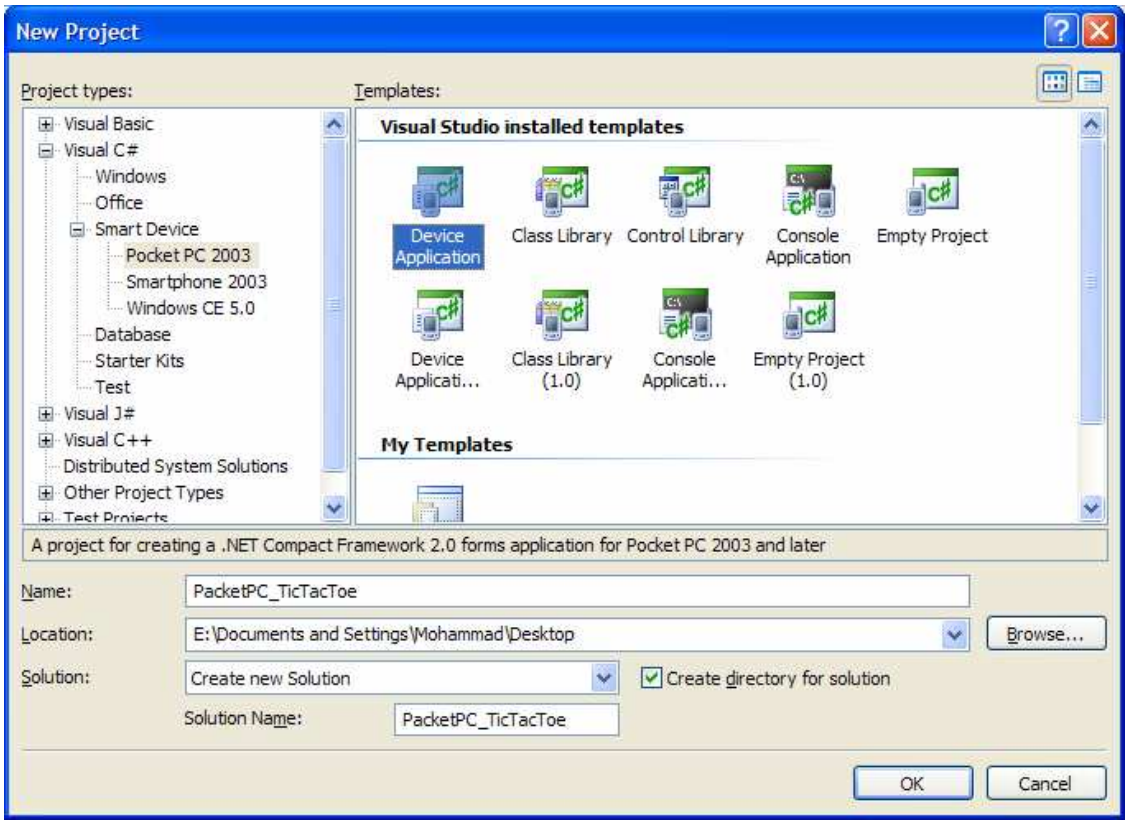
۱) در محیط ویژوال استودیو با استفاده از نوار ابزار گزینه ی File ➔ New Project... را انتخاب کنید. به این ترتیب کادر New Project نمایش داده خواهد شد.

۲) از قسمت سمت چپ گزینه ی Visual C# ➔ Smart Device ➔ Pocket PC 2003 Device Application را انتخاب کنید. سپس در قسمت سمت راست روی نام کلیک کنید. نام پروژه را برابر با PocketPC_TicTacToe وارد کرده و روی دکمه ی OK کلیک کنید. کادر New Project باید مشابه شکل ۲۲-۳ باشد.

۳) با کلیک کردن روی دکمه ی OK پروژه ایجاد شده و صفحه ی همانند یک Pocket PC همانطور که در شکل ۲۲-۶ نشان داده شده است نمایش داده می شود. این قسمت محیط طراحی برنامه است. به این ترتیب زمانی که در حال ایجاد برنامه هستید، می توانید ظاهر و رابط گرافیکی برنامه ی خود را در محیطی کاملاً مشابه مشاهده کنید. بنابراین با استفاده از این محیط می توانید در هر لحظه مشاهده کنید که کاربر چه محیطی را مشاهده خواهد کرد.

۴) برای ایجاد برنامه ابتدا باید ظاهر آن را طراحی کنیم. برای این کار ۱۰ کنترل Button را همانند شکل ۲۲-۵ به فرم برنامه اضافه کنید. سه سطر که در هر یک از آنها سه کنترل Button قرار دارد، قسمت اصلی بازی را تشکیل می دهند. خاصیت Size این ۹ کنترل Button را برابر با 40;40 قرار دهید تا فرم برنامه ایجاد شود. برای نام گذاری کنترل ها از بالا سمت چپ شروع کرده و به سمت راست حرکت کنید تا ردیف تمام شود. سپس به ردیف بعدی بروید و به همین ترتیب ردیف سوم را نیز طی کنید و نام کنترل ها را به ترتیب برابر با btn00، btn01، btn02، btn10، btn11، btn12، btn20، btn21، btn22 قرار دهید. نام ها همانطور که مشاهده می کنید از سه قسمت تشکیل شده است: قسمت btn که مشخص می کند کنترل مورد نظر یک

Button است، قسمت شماره ردیف (۰، ۱ یا ۲) و قسمت شماره ستون (۰، ۱ یا ۲). بنابراین کنترل btn02 در ردیف اول (ردیف صفرم) و ستون سوم (دومین ستون) قرار دارد. به این ترتیب زمانی که از این نامها در کد برنامه استفاده کنید، می دانید که در حال کار با کدام کنترل Button در فرم برنامه هستید.



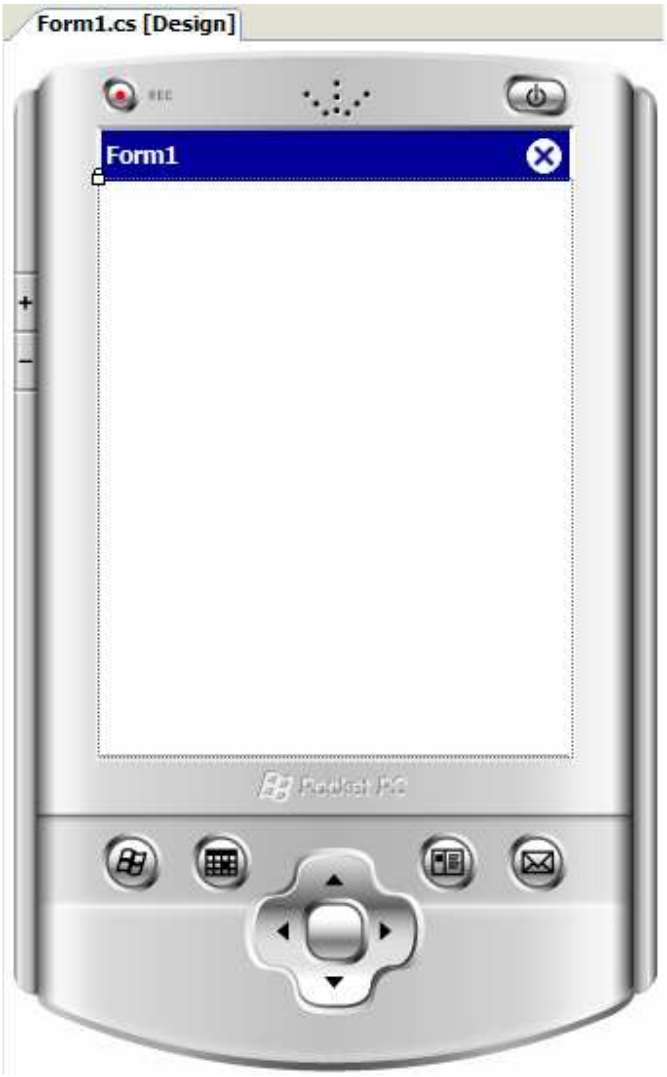
شکل ۲۲-۳

سپس خاصیت Font تمام این کنترل ها را برابر با Tahoma, 24PT, Bold قرار دهید. در آخر نیز خاصیت Text تمام آنها را برابر با X و خاصیت Anchor را نیز برابر با None قرار دهید. دکمه ی آخر در برنامه نیز مربوط به New Game است. خاصیت Name این کنترل را برابر با btnNewGame و خاصیت Text آن را برابر با &New Game قرار دهید. در پایین ۹ دکمه ی مربوط به بازی نیز یک کنترل Label قرار داده و خاصیت Name آن را با مقدار lblMessage تنظیم کنید. اندازه ی کنترل Label را نیز به گونه ای تنظیم کنید تا بتواند دو خط متن را نمایش دهد. حال برای تکمیل ظاهر برنامه، خاصیت Text فرم را نیز برابر با Tic Tac Toe قرار دهید.

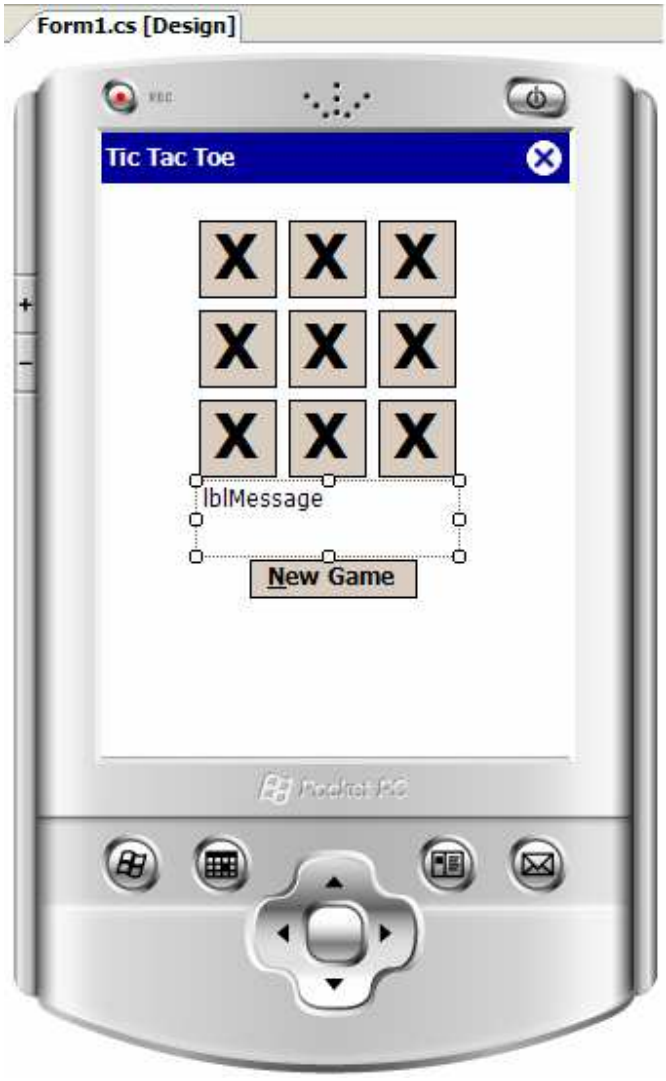
۵) بر روی کنترل btn00 دو بار کلیک کنید تا متد مربوط به رویداد Click این کنترل ایجاد شود. سپس کد مشخص شده در زیر را در این متد وارد کنید:

```
private void btn00_Click(object sender, EventArgs e)
{
      CorrectEnabledState(false);
      Application.DoEvents(); // Allows the screen to refresh
      ((Button)sender).Text = "X";
```

```
        if(IsGameOver())
                MessageBox.Show("Game Over");
        else
        {
                lblMessage.Text = "Computer selecting ...";
                Application.DoEvents(); // Allows the screen to refresh
                ComputerPlay();
                if(IsGameOver())
                        MessageBox.Show("Game Over");
                else
                {
                        lblMessage.Text = "Select your next position ...";
                        CorrectEnabledState();
                }
        }
}
```



شکل ۲۲-۴



شکل ۲۲-۵

۶) کنترل `btn01` را از فرم انتخاب کرده و در پنجره ی `Properties` روی آیکون `Events` کلیک کرده تا لیست رویدادهای مربوط به این کنترل نمایش داده شود. سپس از این لیست رویداد `Click` را انتخاب کرده و روی مثلث آبی رنگ مقابل آن کلیک کنید تا لیست متد هایی که می توانند برای این رویداد انتخاب شوند نمایش داده شود. مشاهده خواهید کرد که این لیست فقط شامل یک متد به نام `btn00_Click` است که در قسمت قبل برای رویداد `Click` کنترل `btn00` ایجاد کرده بودیم. این متد را انتخاب کنید. سپس همین مراحل را برای هفت کنترل `Button` باقی مانده (کنترلهای `btn02` تا `btn22`) تکرار کنید.

۷) در قسمت طراحی فرم برنامه دکمه ی `F7` را فشار داده تا وارد قسمت کد مربوط به کلاس `Form1` شوید. سپس کد مشخص شده در زیر را در این کلاس وارد کنید:

```
// Get the game ready to start again
void ResetGame()
{
      // Loop through the board controls and set them to ""
```

```
        foreach(Control ctrl in this.Controls)
        {
            if(ctrl.GetType() == typeof(Button) &&
                ctrl.Name != "btnNewGame")
            {
                ctrl.Text = String.Empty;
            }
        }

        lblMessage.Text = String.Empty;
        //Enable the board buttons
        CorrectEnabledState(true);
}

private void CorrectEnabledState(Boolean ButtonEnabledState)
{
        foreach(Control ctrl in this.Controls)
        {
            if(ctrl.GetType() == typeof(Button) &&
                ctrl.Name != "btnNewGame")
            {
                ctrl.Enabled = ButtonEnabledState;
            }
        }
}

private void CorrectEnabledState()
{
        foreach(Control ctrl in this.Controls)
        {
            if(ctrl.GetType() == typeof(Button) &&
                ctrl.Name != "btnNewGame")
            {
                if(ctrl.Text == String.Empty)
                    ctrl.Enabled = true;
                else
                    ctrl.Enabled = false;
            }
        }
}

void ComputerPlay()
{
        Random RandomGenerator = new Random();
        int intRandom;
        int intCount = 0;

        intRandom = RandomGenerator.Next(20, 100);
        while (intCount < intRandom)
        {
            foreach (Control ctrl in this.Controls)
            {
                if (ctrl.GetType() == typeof(Button) &&
                    ctrl.Name != "btnNewGame")
                {
                    if (ctrl.Text == String.Empty)
                    {
```

```
                              intCount += 1;
                              if (intCount == intRandom)
                              {
                                    ctrl.Text = "O";
                                    ctrl.Enabled = false;
                                    break;
                              }
                        }
                  }
            }
      }
}

void Winner(String strWinner)
{
      String strMessage;
      if(strWinner == "X")
            strMessage = "You win!!";
      else if(strWinner == "O")
            strMessage = "Computer wins!!";
      else
            strMessage = strWinner;

      lblMessage.Text = strMessage;
}

Boolean IsGameOver()
{
      if(btn00.Text == btn01.Text && btn01.Text == btn02.Text &&
            btn02.Text != String.Empty )
      {
            // Winner on top Row
            Winner(btn00.Text);
            return true;
      }
      if(btn10.Text == btn11.Text && btn11.Text == btn12.Text &&
            btn12.Text != String.Empty )
      {
            // Winner on middle Row
            Winner(btn10.Text);
            return true;
      }
      if(btn20.Text == btn21.Text && btn21.Text == btn22.Text &&
            btn22.Text != String.Empty)
      {
            // Winner on bottom Row
            Winner(btn20.Text);
            return true;
      }
      if(btn00.Text == btn10.Text && btn10.Text == btn20.Text &&
            btn20.Text != String.Empty)
      {
            // Winner on first column
            Winner(btn00.Text);
            return true;
      }
      if(btn01.Text == btn11.Text && btn11.Text == btn21.Text &&
```

```
            btn21.Text != String.Empty)
      {
            // Winner on second column
            Winner(btn01.Text);
            return true;
      }
      if(btn02.Text == btn12.Text && btn12.Text == btn22.Text &&
            btn22.Text != String.Empty)
      {
            // Winner on third column
            Winner(btn02.Text);
            return true;
      }
      if(btn00.Text == btn11.Text && btn11.Text == btn22.Text &&
            btn22.Text != String.Empty)
      {
            // Winner on diagonal top left to bottom right
            Winner(btn00.Text);
            return true;
      }
      if(btn20.Text == btn11.Text && btn11.Text == btn02.Text &&
            btn02.Text != String.Empty)
      {
            // Winner on diagonal bottom left to top right
            Winner(btn20.Text);
            return true;
      }
      // Test for a tie, all square full
      int intOpenings = 0;
      foreach(Control ctrl in this.Controls)
      {
            if(ctrl.GetType() == typeof(Button) &&
                  ctrl.Name != "btnNewGame")
                  if(ctrl.Text == String.Empty)
                        intOpenings = intOpenings + 1;
      }
      if(intOpenings == 0)
      {
            Winner("It's a tie.");
            return true;
      }
      return false;
}
```

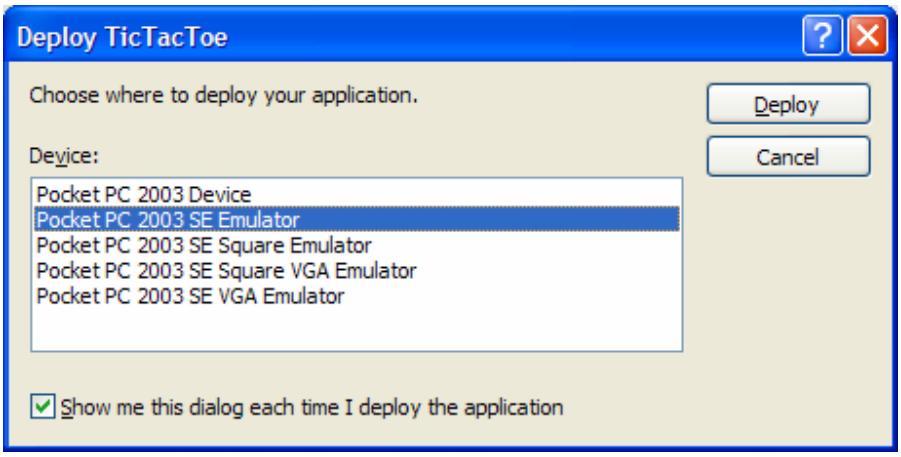
۸) به قسمت طراحی فرم برنامه برگشته و روی کنترل `btnNewGame` دو بار کلیک کنید تا متد مربوط به رویداد `Click` این کنترل ایجاد شود. سپس کد مشخص شده در زیر را به آن اضافه کنید:

```
private void btnNewGame_Click(object sender, EventArgs e)
{
      ResetGame();
}
```

۹) مجدداً به قسمت طراحی فرم برگشته و روی فرم برنامه دو بار کلیک کنید تا متد مربوط به رویداد `Load` آن ایجاد شود. سپس کد مشخص شده در زیر را به این متد اضافه کنید:

```
private void Form1_Load(object sender, EventArgs e)
{
      CorrectEnabledState(false);
      lblMessage.Text = "Click new game to begin.";
}
```

۱۰) کلید F5 را فشار دهید تا برنامه اجرا شود و بتوانیم عملکرد آن را تست کنیم. به این ترتیب ابتدا کادر Deploy همانند شکل ۲۲-۶ نشان داده می شود و از شما می خواهد که چگونگی توزیع این برنامه را مشخص کنید. از این لیست Pocket PC 2003 SE Emulator را مشخص کرده و دکمه ی Deploy را فشار دهید. باید مقداری صبر کنید تا اجرای برنامه آغاز شود.



شکل ۲۲-۶

۱۱) با اجرای برنامه یک شبیه ساز Pocket PC نشان داده شده و برنامه ی شما نیز در آن اجرا می شود. برای آغاز برنامه باید مقداری صبر کنید.

۱۲) با اجرا شدن برنامه می توانید روی دکمه ی New Game کلیک کرده و در مقابل کامپیوتر بازی کنید. البته شکست دادن این برنامه بسیار ساده است، زیرا اعداد خود را به صورت تصادفی انتخاب می کند.

چگونه کار می کند؟

با تکمیل این برنامه مقداری با برنامه نویسی برای دستگاه های هوشمند آشنا شدید و مشاهده کردید که این نوع برنامه ها نیز تفاوت چندانی با برنامه هایی که در فصول قبلی می نوشتیم ندارد. اولین موردی که در این نوع برنامه ها ممکن است توجه شما را جلب کرده باشد، اندازه ی صفحه نمایش است. صفحه نمایش این دستگاه ها بسیار محدود است، بنابراین طراحی برنامه ها در آن کمی مشکل تر از برنامه های معمولی است.

در این برنامه نیز همانند برنامه های عادی دیگر، از طراحی ظاهر و رابط برنامه شروع کرده و کنترلهای مورد نیاز را در فرم قرار دادیم. با تمام خاصیت هایی که در این قسمت از آنها استفاده کردیم در برنامه های ویندوزی قبلی آشنا شده اید، بنابراین نباید در طراحی رابط گرافیکی این برنامه مشکلی داشته باشید.

در این برنامه اغلب کارها به وسیله ی کد انجام می شود و همانطور که مشاهده می کنید کدی هم که برای این کارها نوشته ایم، مشابه کدی است که در برنامه های قبلی در این کتاب استفاده می کردیم. این کد از چندین تابع و چندین زیر برنامه تشکیل شده است که با فراخوانی یکدیگر در مواقع لزوم، کارهای لازم را در برنامه انجام نمی دهند. بنابراین در فهم کد برنامه نیز نباید مشکلی داشته باشید. در این برنامه نیز مانند برنامه های قبلی از ابتدا شروع کرده و متد ها را یکی یکی بررسی می کنیم تا با نحوه ی عملکرد برنامه بهتر آشنا شویم.

قبل از شروع هر بازی و یا بعد از اینکه هر بازی به اتمام رسید، به روشی نیاز دارید که بتوانید کنترلهای موجود در فرم را برای مرتبه ی بعدی آماده کنید. برای این کار متدی به نام `ResetGame` ایجاد می کنیم. وظیفه ی این متد این است که در بین کنترلهای موجود در فرم حرکت کرده و اگر نوع کنترل برابر با `Button` بوده و نیز جزئی از دکمه های مربوط به بازی بود، خاصیت `Text` آن را برابر با رشته ی تهی (`String.Empty`) قرار دهد. بعد از اینکه متن روی تمام کنترل های `Button` را تنظیم کرد، خاصیت `Text` کنترل `Label` را نیز پاک کرده و سپس تمام کنترلهای `Button` موجود در فرم را فعال کند.

```
// Get the game ready to start again
void ResetGame()
{
	// Loop through the board controls and set them to ""
	foreach(Control ctrl in this.Controls)
	{
		if(ctrl.GetType() == typeof(Button) &&
			ctrl.Name != "btnNewGame")
		{
			ctrl.Text = String.Empty;
		}
	}

	lblMessage.Text = String.Empty;
	//Enable the board buttons
	CorrectEnabledState(true);
}
```

سپس دو نسخه ی مختلف از متد `CorrectEnabledState` ایجاد می کنیم. هنگامی که برای فراخوانی این متد از یک پارامتر `Boolean` استفاده کنید، این متد خاصیت `Enabled` تمام ۹ کنترل `Button` مربوط به بازی را برابر با پارامتر مشخص شده به وسیله ی شما قرار می دهد. متد دیگر پارامتری دریافت نمی کند، بلکه در بین دکمه های مربوط به بازی حرکت می کند. اگر خاصیت `Text` این دکمه ها برابر با `String.Empty` بود مقدار `Enabled` آن را برابر با `true` قرار داده، در غیر این صورت این مقدار را برابر با `False` قرار می دهد.

```
private void CorrectEnabledState(Boolean ButtonEnabledState)
{
	foreach(Control ctrl in this.Controls)
	{
		if(ctrl.GetType() == typeof(Button) &&
			ctrl.Name != "btnNewGame")
		{
			ctrl.Enabled = ButtonEnabledState;
		}
	}
}
```

```
private void CorrectEnabledState()
{
	foreach(Control ctrl in this.Controls)
	{
		if(ctrl.GetType() == typeof(Button) &&
			ctrl.Name != "btnNewGame")
		{
			if(ctrl.Text == String.Empty)
				ctrl.Enabled = true;
			else
				ctrl.Enabled = false;
		}
	}
}
```

یک متد دیگر نیز به نام `ComputerPlay` در برنامه ایجاد می کنیم. این متد وظیفه دارد تا به عنوان کامپیوتر در مقابل شما بازی کند. در ابتدای این متد متغیرهای لازم را تعریف می کنیم، اما عملکرد اصلی متد از خط قبل از حلقه ی `while` شروع می شود. در این خط با استفاده از متد `Next` در کلاس `Random` یک عدد تصادفی بین ۲۰ تا ۱۰۰ تولید می کنیم. سپس برنامه با استفاده از یک حلقه در بین مربع های خالی صفحه حرکت می کند تا زمانی که به اندازه ی عدد تصادفی تولید شده از روی مربع های خالی عبور کند. بعد از این کار اولین مربع خالی را با علامت `O` پر می کند.

```
void ComputerPlay()
{
	Random RandomGenerator = new Random();
	int intRandom;
	int intCount = 0;

	intRandom = RandomGenerator.Next(20, 100);
	while (intCount < intRandom)
	{
		foreach (Control ctrl in this.Controls)
		{
			if (ctrl.GetType() == typeof(Button) &&
				ctrl.Name != "btnNewGame")
			{
				if (ctrl.Text == String.Empty)
				{
					intCount += 1;
					if (intCount == intRandom)
					{
						ctrl.Text = "O";
						ctrl.Enabled = false;
						break;
					}
				}
			}
		}
	}
}
```

با کلیک روی هر یک از مربع های موجود در صفحه، متد `btn00_Click` فراخوانی می شود، زیرا هنگام طراحی فرم رویداد `Click` دیگر کنترل ها را نیز برابر با این متد قرار دادیم. بنابراین با کلیک کردن روی هر کدام از این مربع ها در فرم همین متد

فراخوانی می شود. این متد ابتدا تمام مربع های درون بازی را غیر فعال می کرده و سپس متد DoEvents از کلاس Application را فراخوانی می کند تا برنامه بتواند تمام رویدادهای فراخوانی شده را به اتمام برساند. با انجام این کار از کلیک کردن کاربر روی دو دکمه، قبل از اینکه کامپیوتر حرکتی را انجام دهد جلوگیری خواهیم کرد. اگر این خطوط را حذف کنید، هنگام بازی می توانید به سرعت روی سه دکمه کلیک کنید بدون اینکه کامپیوتر بتواند انتخابی را انجام دهد و به این ترتیب به سادگی بازی را ببرید. سپس با استفاده از پارامتر sender کنترلی که باعث شده است این متد فراخوانی شود را بدست آورده و آن را به شیئ ای از نوع Button تبدیل می کنیم، زیرا هم اکنون این شیئ از نوع Object است. سپس خاصیت Text این کنترل را با مقدار X تنظیم می کنیم. بعد از اینکه انتخاب شما صورت گرفت، صفحه برای اینکه مشخص شود آیا برنده ای وجود دارد یا نه بررسی خواهد شد. اگر برنده ای وجود نداشته باشد، کامپیوتر انتخابی را انجام خواهد داد و مجدداً صفحه برای مشخص شدن برنده بررسی می شود. در صورت که باز هم برنده ای نداشته باشیم، به شما اجازه داده می شود تا روی دکمه ای دیگر کلیک کنید:

```
private void btn00_Click(object sender, EventArgs e)
{
	CorrectEnabledState(false);
	Application.DoEvents(); // Allows the screen to refresh
	((Button)sender).Text = "X";
	if(IsGameOver())
		MessageBox.Show("Game Over");
	else
	{
		lblMessage.Text = "Computer selecting ...";
		Application.DoEvents(); // Allows the screen to refresh
		ComputerPlay();
		if(IsGameOver())
			MessageBox.Show("Game Over");
		else
		{
			lblMessage.Text = "Select your next position ...";
			CorrectEnabledState();
		}
	}
}
```

هنگامی که یکی از طرفین به عنوان برنده مشخص شد، متد Winner فراخوانی شده و متن مناسبی در صفحه نمایش داده می شود:

```
void Winner(String strWinner)
{
	String strMessage;
	if(strWinner == "X")
		strMessage = "You win!!";
	else if(strWinner == "O")
		strMessage = "Computer wins!!";
	else
		strMessage = strWinner;

	lblMessage.Text = strMessage;
}
```

بعد از اینکه هر حرکت صورت گرفت، متد `IsGameOver` فراخوانی می شود تا مشخص شود که آیا برنده ای به وجـود آمـده است یا نه. در این متد تمام حالتهایی که می تواند باعث برنده شدن هر یک از بازی کنندگان شود بررسی می شـود. اگـر سـه مربـع متوالی به دنبال هم برای یک طرف بازی باشند، متد `Winner` فراخوانی شده و مقدار `true` به عنوان نتیجه ی تـابع برگـشته می شود. اگر هیچ برنده ای پیدا نشود، متد مربع های موجود در بازی را بررسی می کند تا مشخص شود آیا بازی تمام شده اسـت یـا نه. اگر تمام مربع ها پر شده بود، بازی مساوی اعلام می شود.

```
Boolean IsGameOver()
{
	if(btn00.Text == btn01.Text && btn01.Text == btn02.Text &&
		btn02.Text != String.Empty )
	{
		// Winner on top Row
		Winner(btn00.Text);
		return true;
	}
	if(btn10.Text == btn11.Text && btn11.Text == btn12.Text &&
		btn12.Text != String.Empty )
	{
		// Winner on middle Row
		Winner(btn10.Text);
		return true;
	}
	if(btn20.Text == btn21.Text && btn21.Text == btn22.Text &&
		btn22.Text != String.Empty)
	{
		// Winner on bottom Row
		Winner(btn20.Text);
		return true;
	}
	if(btn00.Text == btn10.Text && btn10.Text == btn20.Text &&
		btn20.Text != String.Empty)
	{
		// Winner on first column
		Winner(btn00.Text);
		return true;
	}
	if(btn01.Text == btn11.Text && btn11.Text == btn21.Text &&
		btn21.Text != String.Empty)
	{
		// Winner on second column
		Winner(btn01.Text);
		return true;
	}
	if(btn02.Text == btn12.Text && btn12.Text == btn22.Text &&
		btn22.Text != String.Empty)
	{
		// Winner on third column
		Winner(btn02.Text);
		return true;
	}
	if(btn00.Text == btn11.Text && btn11.Text == btn22.Text &&
		btn22.Text != String.Empty)
	{
		// Winner on diagonal top left to bottom right
```

```
            Winner(btn00.Text);
            return true;
      }
      if(btn20.Text == btn11.Text && btn11.Text == btn02.Text &&
            btn02.Text != String.Empty)
      {
            // Winner on diagonal bottom left to top right
            Winner(btn20.Text);
            return true;
      }
      // Test for a tie, all square full
      int intOpenings = 0;
      foreach(Control ctrl in this.Controls)
      {
            if(ctrl.GetType() == typeof(Button) &&
                  ctrl.Name != "btnNewGame")
                  if(ctrl.Text == String.Empty)
                        intOpenings = intOpenings + 1;
      }
      if(intOpenings == 0)
      {
            Winner("It's a tie.");
            return true;
      }
      return false;
}
```

بقیه ی کدی که در این برنامه نوشته شده است مربوط به رویداد Load فـرم و رویـداد Click کنتـرل btnNewGame است. هنگامی که فرم بار گذاری می شود، با فراخوانی متد CorrectEnabledState تمام دکمه های بـازی را غیـر فعال می کنیم. به این ترتیب کاربر برای شروع بازی مجبور است روی دکمه ی New Game کلیک کند. ایـن دکمـه نیـز متـد ResetGame را فراخوانی می کند تا بازی آغاز شود.

```
private void btnNewGame_Click(object sender, EventArgs e)
{
      ResetGame();
}

private void Form1_Load(object sender, EventArgs e)
{
      CorrectEnabledState(false);
      lblMessage.Text = "Click new game to begin.";
}
```

نتیجه:

ویژوال استودیو ۲۰۰۵ به همراه Compact Framework باعث شده است که برنامه نویسی برای دستگاه های هوشـمند تا حد زیادی مشابه برنامه نویسی تحت ویندوز باشد. یکی از دلایلی هم که باعث شده اسـت سیـستم عامـل ویـنـدوز بـیش از دیگـر سیستم عاملها برای دستگاه های PDA استفاده شود در این است که همانطور که مشاهده کردید طراحی برنامه ها برای این سیستم عامل بسیار ساده است.

به علاوه گسترش روز افزون استفاده از این نوع دستگاه ها به همراه سادگی طراحی برنامه برای آنها باعث شده است کـه شـرکتهای نرم افزاری روز به روز بیشتر تمایل داشته باشند که از افرادی مسلط به برنامه نویسی موبایل اسـتفاده کننـد. بنـابراین بهتـر اسـت بـا یادگیری برنامه نویسی موبایل، از توانایی های خود را در برنامه نویسی روز به روز بیشتر استفاده کنید.
در این فصل با مبانی برنامه نویسی موبایل آشنا شدیم. مشاهده کردید که در این نوع برنامه نویسی از چه چارچوبی استفاده می شـود و تفاوت و شباهت این چارچوب با چارچوب .NET در چیست؟ مشاهده کردید که چه کلاسها و متد های در CF حذف شده اند تا این چارچوب توانسته است با حجمی در حدود ۲۰ درصد چارچوب .NET مورد استفاده قرار گیرد. در انتهای فـصل نیـز بـه عنـوان تمرین یک بازی برای موبایل ایجاد کردیم.
در پایان این فیل باید با موارد زیر آشنا شده باشید:

- تفاوت های بین نسخه ی کامل چـارچوب .NET و نیـز چـارچوب .NET Compact Framework را درک کنید.
- با استفاده از ActiveSync یک دستگاه PDA را به کامپیوتر شخصی خود متصل کنید.
- برنامه هایی برای دستگاه های موبایل و هوشمند طراحی کنید.
- با استفاده از شبیه ساز درونی ویژوال استودیو، عملکرد برنامه های خود را تست کنید.

تمرین:

در بازی که در این فصل ایجاد کردیم، کامپیوتر یک خانه ی تصادفی را انتخاب کرده و آن را پر مـی کنـد. بـرای ایـن تمـرین مـی خـواهیم برنامـه را بـه گونـه ای تغییـر دهـیم تـا کـامپیوتر مقـداری هوشـمندانه تـر عمـل کنـد. متـدی بـه نـام ComputerPlayToWin را به برنامه اضافه کرده و زمانی که نوبت حرکت به کامپیوتر می رسد، آن را فراخـوانی کنیـد. این متد را به گونه ای بنویسید تا کامپیوتر بتواند با استفاده از آن در بازی پیروز شود.

ضمیمه ی ۱: ادامه ی مسیر

حال که توانسته اید مطالب این کتاب را با موفقیت پشت سر بگذارید، باید درک خوبی از نحوه ی برنامه نویسی با استفاده از ویـژوال C# بدست آورده باشید. در تمام مثال ها، برنامه ها و تمرین هایی که در طی این کتاب بررسی کردیم سعی داشتیم که بتوانیم پایـه ای قوی در مورد نحوه ی برنامه نویسی در شما ایجاد کنیم، اما تمام این موارد که مشاهده کردید فقط آغاز مسیر بودنـد. در حقیقـت این کتاب یکی از چندین کتابی است که باید مطالعه کنید تا بتوانید به یک برنامه نویس مسلط در ویژوال C# تبدیل شوید. با وجـود اینکه تاکنون راه زیادی را طی کرده اید، اما هنوز مسیر بسیار طولانی را در پیش دارید، و احتمالاً باید در ایـن مـسیر پاسـخ سـوالات زیادی را درک کنید.

اما مهمترین سوالی که در این قسمت با آن مواجه هستید این است که بعد از این کتاب چگونه باید مسیر خود را ادامه دهید، و یـا بـه بیان ساده تر "بعد از اتمام این کتاب چه کار باید کرد؟".

در این ضمیمه سعی می کنم مسیر هایی که می توانید بعد از اتمام این کتاب ادامه دهید را بیان کنم. همانطور که ممکن است حدس زده باشید بعد از این کتاب مسیر های گوناگونی وجود دارند که می توانید در آنها به مطالعه بپردازید، و این که چـه مـسیری را ادامـه دهید به شما بستگی دارد و اینکه بخواهید در آینده با تخصص در چه زمینه ای به کار مشغول شـوید. ممکـن اسـت بعـضی از افـراد بخواهند در زمینه ی برنامه نویسی ویژوال C# تخصص بیشتری کسب کنند و اطلاعات مختصری هم در زمینه های دیگـر بدسـت آورند، بعضی دیگر ممکن است بخواهند در زمینه ی برنامه نویسی تحت شبکه و ایجاد برنامه های مبتنی بر وب فعالیت کنند، بعضی از افراد بخواهند بر روی برنامه نویسی گرافیکی تمرکز کنند و ....

خوب، اولین نکته و مهمترین نکته ی که در این قسمت باید توجه داشت در این است که بعد از اتمام این کتاب، زیاد با برنامه نویسی فاصله نگیرید. اگر مدت زمان طولانی بگذرد و از ویژوال C# استفاده نکنید، به زودی تمام مطالبی که در این کتـاب آموختـه ایـد را فراموش خواهید کرد. تنها راه جلوگیری از این فراموشی تمرین در مورد مطالبی است که خوانده اید. برای این کار چندین روش وجود دارد:

- مثالها و تمرین هایی که در طی فصلهای این کتاب بررسی شدند را ادامه دهید. سعی کنید ویژگی های بیشتری را به ایـن برنامه ها اضافه کنید و آنها را تا حد ممکن گسترش دهید. حتی در صورت امکان سعی کنید برنامه های فـصول مختلـف این کتاب را با یکدیگر ترکیب کرده و یک برنامه ی کاملتر ایجاد کنید.
- ممکن است ایده ی جدیدی در مورد نحوه ی نوشتن یکی از برنامه های کتاب داشته باشید. خوب، بروید و آن برنامه را به صورتی که در نظر دارید بنویسید.
- سعی کنید اصطلاحات و مفاهیمی که در طی کتاب با آنها آشنا شدیم را به دقت درک کنید. برای این کار حتـی اگـر لازم است، یک بار دیگر کتاب را از ابتدا بخوانید. این کار باعث می شود تا قسمتهایی از برنامه هایی که در ابتدای کتاب ایجـاد می کردیم و در آن زمان درک درستی از آنها نداشتید را بهتر متوجه شوید.
- تا حد ممکن به خواندن مقالات مختلف در این زمینه بپردازید، حتی اگر در ابتدا مطالب زیادی از آنها متوجه نمی شوید. اما بعد از مدتی مفهوم اغلب آنها را درک خواهید کرد.
- سعی کنید در مورد مطالبی که خوانده اید با افراد دیگر صحبت کنید. اگر دوستانی دارید که در این زمینه فعالیت می کنند، با آنها صحبت کرده و سوالات خود را از آنها بپرسید. اگر هم فردی را در این زمینه نمی شناسـید، سـعی کنیـد از صـحبت کردن با افرادی که در اینترنت هستند و در این زمینه فعالیت می کننـد اسـتفاده کنیـد. فـوروم هـا و سـایتهای فارسـی و انگلیسی بسیاری در این مورد وجود دارند که می توانید از آنها استفاده کنید.
- از سایتهایی که در ادامه ی این بخش نام برده شده اند استفاده کنید.

در ادامه ی این ضمیمه به معرفی منابع آنلاین و آفلاین موجود در این زمینه خواهیم پرداخت تا راحت تر بتوانید تصمیم بگیرید که بعد از اتمام این کتاب چگونه برنامه نویسی را ادامه دهید.

منابع آنلاین:

اساسا چندین هزار سایت در اینترنت وجود دارند که در این زمینه مطالبی را ارایه می دهند و می توانید از آنها برای رفع مشکلات خود استفاده کنید. هر زمان که به مشکلی در این زمینه برخورد کردید، مطمئن باشید افراد بسیاری در اینترنت وجود دارند که به شما برای حل مشکلتان کمک خواهند کرد. این افراد ناشناخته ممکن است افراد مبتدی مانند شما باشند که با مشکلی همانند مشکل شما برخورد کرده باشند، و یا افراد حرفه ای و متخصصی باشند که از سطح اطلاعات بالایی برخوردار باشند. این مسئله چیز عجیبی نیست. کافی است درک کنید که هر فردی در هر مرحله ای می تواند یک مبتدی برای مراحل بالاتر به شمار رود و همچنین هر فردی می تواند برای مراحل قبل از خود به عنوان یک متخصص جلوه کند. حتی خود شما نیز می توانید در این نوع سایتها به کمک افرادی که تخصص کمتری نسبت به شما دارند بپردازید و مشکلات (هر چند ساده ی) آنها را بر طرف کنید. در لیست زیر سعی می کنیم مهمترین منابع و سایتهای اینترنتی موجود را معرفی کنیم. اگر در این سایتها هم نتوانستید مشکلات خود را برطرف کنید، کافی است با استفاده از یک موتور جستجو به دنبال پاسخ خود بگردید. قطعا به سرعت جواب خود را خواهید یافت.

منابع مایکروسافت:

احتمالاً یکی از مهمترین سایتهایی که به آن مراجعه خواهید کرد، وب سایت شرکت مایکروسافت خواهد بود. این سایت شامل منابع و اطلاعات زیادی در مورد .NET است. همچنین در این سایت گروه های خبری زیادی در مورد برنامه نویسی با استفاده از ویژوال C# ۲۰۰۵ وجود دارد. برای دسترسی به این گروه های خبری می توانید از آدرس زیر استفاده کنید:

```
http://communities2.microsoft.com/communities/newsgroups/en-us/default.aspx
```

همچنین در این زمینه چندین سایت دیگر که زیر مجموعه ی سایت مایکروسافت هستند نیز وجود دارند که مطمئنا اطلاعات مفیدی برای شما خواهند داشت. لیست زیر تعدادی از آنها را معرفی می کند:

- **سایت ویژوال استودیو ۲۰۰۵:**

```
http://lab.msdn.microsoft.com/vs2005/
```

- **مستندات مربوط به ویژوال استودیو ۲۰۰۵:**

```
http://lab.msdn.microsoft.com/library/
```

- **کتابخانه ی MSDN:**

```
http://msdn.microsoft.com/library/
```

- **سایت MSDN:**

```
http://msdn.microsoft.com
```

- **سایت منابع قابل دانلود برای .NET:**

```
http://msdn.microsoft.com/netframework/downloads/
```

منابع دیگر:

همانطور که در قسمت قبل نیز گفتم، چندین هزار سایت در مورد ویژوال .NET C# و ویژوال C# ۲۰۰۵ در اینترنت وجود دارد. این سایتها شامل اطلاعاتی در مورد سمینارهای موجود در این زمینه در سرتاسر دنیا، خبرهای جدید در مورد .NET و ویژوال C#، مقالات مختلف در این زمینه و ... می شوند. اگر در این مورد در اینترنت به جستجو بپردازید، تعداد زیادی از سایتهایی را پیدا خواهید کرد که در این بازه فعالیت می کنند. در این قسمت با سه سایت انگلیسی مهم و همچنین یک سایت فارسی آشنا خواهیم شد.

اولین و مهمترین سایتی که در این باره وجود دارد، سایت www.gotdotnet.com است که یکی از زیر مجموعه های سایت مایکروسافت به شمار می رود. این سایت دارای قسمتهای مختلفی برای ارسال مقالات در مورد .NET و زمینه های مرتبط به آن است. همچنین این سایت دارای قسمت فوروم و گفتگو در مورد مسائل مربوط به برنامه نویسی در این پلت فرم است. اگر در زمینه ی برنامه نویسی مشکلی داشتید، همواره افرادی در این سایت وجود دارند که به شما کمک کنند تا مشکل خود را برطرف کنید. حتی ممکن است مشکل شما قبل از این برای فرد دیگری نیز به وجود آمده شده و به وسیله ی افرادی رفع شده باشد. بنابراین می توانید با جستجو در مطالب این سایت، اطلاعات مورد نیاز خود را بدست آورید.

سایت دیگری که در زمینه ی C# وجود دارد، وب سایت www.csharp-corner.com است. این سایت دارای مقالات و نمونه برنامه های ارزنده ای در مورد مسائل و جنبه های مختلف برنامه نویسی به زبان C# است که می تواند تاثیر زیادی در یادگیری شما داشته باشد.

وب سایت مهم و مفید دیگری که در این زمینه وجود دارد، سایت www.codeproject.com است. در این سایت دارای بالغ بر یازده هزار مقاله، نمونه برنامه، اخبار و ... است که می تواند به شما در یادگیری مطالب جدید در برنامه نویسی C# کمک کند.

وب سایت فارسی برنامه نویس (www.barnamenevis.org) ، یکی از سایتهای تخصصی در زمینه ی برنامه نویسی است. این سایت با تالارهای گفتگو برای زمینه های مختلف برنامه نویسی و داشتن حدود نوزده هزار کاربر، یکی از بزرگترین سایتهای تخصصی برنامه نویسی به زبان فارسی به شمار می رود. اگر در استفاده از سایتهای قبلی به علت زبان آنها با مشکل مواجه شدید، این سایت و افرادی که در آن وجود دارند یکی از بهترین گزینه ها برای حل مشکل شما محسوب می شوند.

البته سایتهایی که در این قسمت معرفی شد فقط بخشی از سایتهایی بودند که در این زمینه وجود دارند. بعضی از شما ممکن است ترجیح دهید فقط از این چند سایت استفاده کنید و برخی دیگر نیز ممکن است بخواهید در اینترنت به دنبال سایتهای دیگری گشته و آنها را به کار ببرید. انتخاب این مورد که از چه سایتی استفاده کنید فقط به شما بستگی دارد، اما مهم این است که بدانید اینترنت می تواند نقش به سزایی در سطح یادگیری مطالب و همچنین تنوع آنها داشته باشد.

ضمیمه ۲: برنامه نویسی تجاری و چند لایه در .NET

در معرفی .NET می توان گفت که .NET شامل یک مجموعه جدید از تکنولوژی ها و نیز یک نمونه جدید برای طراحی و توسعه نرم افزار می شود. .NET فقط یک محیط طراحی و توسعه جدید نیست، بلکه یک دنباله ی کامل از سرور ها و سرویس ها است که برای حل مشکلات تجاری امروزه به صورت موازی با یکدیگر کار می کنند. در طی فصول این کتاب، با برنامه نویسی تحت .NET و نیز تعدادی از تکنولوژی هایی که در آن به کار رفته است آشنا شدیم و آنها را به طور عملی مورد استفاده قرار دادیم. اما در این ضمیمه سعی می کنم با نگرشی متفاوت .NET را بررسی کرده و بعد از معرفی آن، دلیل ایجاد و ابداع .NET را بررسی کنم.

به راحتی می توان دریافت که پوشش دادن تمام تکنولوژی هایی که برای ارائه یک مدل و راه حل در .NET مورد نیاز است حتی در یک کتاب نیز قابل گنجایش نیست. .NET علاوه بر در بر داشتن نسخه های جدید از تکنولوژی های قبلی، چندین تکنولوژی جدید را نیز شامل می شود. این مجموعه تکنولوژی ها عبارتند از SQL Server، Windows XP، .NET Enterprise Services و همچنین تکنولوژی های استاندارد مثل SOAP و XML.

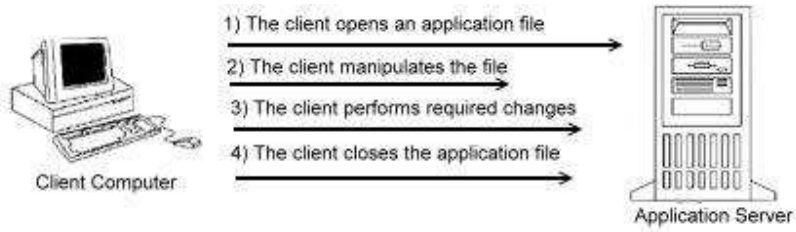
چرا .NET؟

همانند هر تکنولوژی دیگری، .NET نیز باید قبل از اینکه به مورد استفاده قرار گیرد به صورت کامل بررسی شود. بنابراین در این قسمت سعی می کنیم با مزایای استفاده از .NET آشنا شویم. در ابتدا نگاه کوتاهی بر مشکلاتی خواهیم داشت که می توان با .NET به ارائه راه حل برای آنها پرداخت.

مشکلات تجاری رفع شده توسط .NET

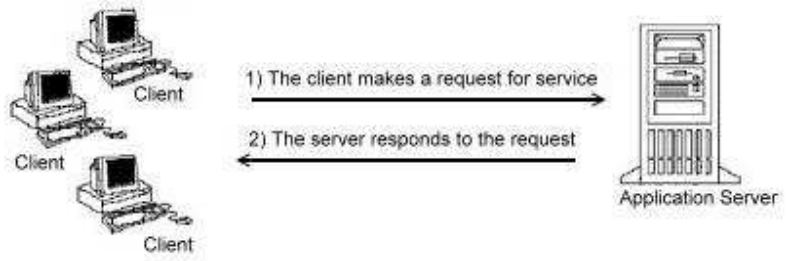
از بسیاری جهات دنیای کامپیوترهای شخصی با حرکت از حالت محاسبه به صورت مجزا و نیز ایستگاه های کاری غیر مرتبط به سوی شبکه های کامپیوتری و نیز رابطه های سرویس گیرنده/ سرویس دهنده دچار تغییراتی شده است. شبکه های سرویس دهنده فایل و یا چاپگر راهی را برای به اشتراک گذاری اطلاعات با مدیریت مرکزی فراهم می کند. تولد سیستم های سرویس گیرنده/سرویس دهنده به کاهش حجم کار از روی سیستم های سرویس گیرنده و انتقال آن به سرور ها، و نیز افزایش کارایی و قابلیت اعتماد در برنامه ها کمک قابل توجهی کرد. در این نوع برنامه ها مدیران سیستم نمی توانستند به کامپیوتر های سرویس گیرنده برای مدیریت و اداره فایلهای موجود اعتماد کنند، زیرا ممکن بود این کامپیوتر ها به هر دلیلی از کار بایستند و باعث خرابی داده های موجود شوند. بنابراین با استفاده از این سرویس ها این اطمینان ایجاد می شد که سرویس گیرنده محدود به دریافت و مشاهده ی فایلها و استفاده از آنها است و سرور ها کارهای اصلی را انجام می دهند. این مورد باعث افزایش قابلیت اعتماد به برنامه ها می شد، زیرا احتمال رخ دادن خرابی در آنها به شدت کاهش پیدا می کرد.

اولین شبکه های کامپیوتری شامل سرورهای فایل و چاپگر به همراه یک سیستم متمرکز برای اشتراک اطلاعات و نیز مدیریت شبکه بود (همانند شکل زیر). اما این مدل فقط شامل اشتراک فایل و چاپگر از طریق شبکه بود. برنامه ی اصلی در حقیقت به صورت کامل در قسمت سرویس گیرنده عمل کرد و فقط از مزایای مشخصی از سرویس های موجود در مدل سرویس گیرنده/سرویس دهنده استفاده می کرد.



مدل قدیمی سرویس گیرنده/سرویس دهنده – در این مدل فقط داده ها از سرور دریافت می شد، اما برنامه ی اصلی به وسیله سرویس گیرنده کنترل می شد

مشکلات این نوع سیستم های متمرکز شامل مواردی از قبیل فقدان کارایی و یا خرابی و از دست رفتن اطلاعات می شوند. فقدان کارایی به این دلیل بود که برنامه های سمت کاربر بایستی تمامی محاسبات مورد نیاز را انجام می دادند. به علت ریسک ثابت و همیشه موجود نا پایداری سرویس گیرنده[1] , پتانسیل خرابی اطلاعات نیز بالا بود. اگر موقع دستکاری فایل بر روی سرور ارتباط سرویس گیرنده از شبکه قطع می شد برنامه یا اطلاعات فایل به راحتی تخریب می شدند.
بنابراین مدل قدیمی را تصحیح کردند و مدلی مانند شکل زیر را ارائه دادند. مدل تصحیح شده ی سرویس گیرنده/سرویس دهنده هرگز به کاربر اجازه نمی داد که به صورت واقعی به اطلاعات یا برنامه ها دسترسی داشته باشد. سرویس گیرنده هیچ تماسی با فایل یا برنامه ی مورد استفاده نداشت و این موجب کاهش قابل توجه ریسک از دست رفتن اطلاعات می شد.

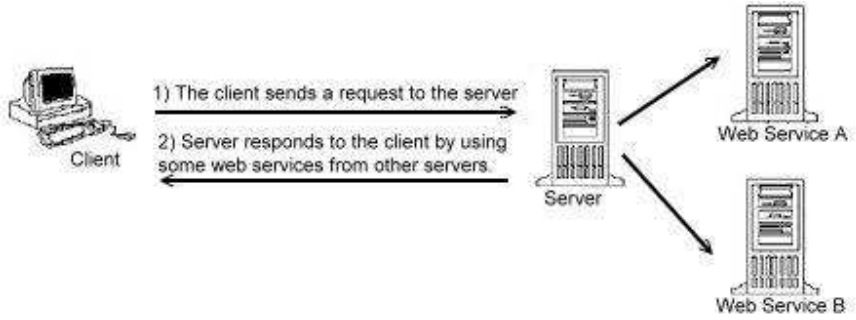


این شکل مدل تصحیح شده ی سرویس گیرنده/سرویس دهنده را نمایش می دهد

امروزه یک سرور تمامی سرویسهای اینترنتی مربوط به برنامه از احراز هویت کاربر تا دسترسی به داده ها را انجام می دهد. این امر مشکلات قابل توجهی را در مقیاس های کوچک ایجاد نمی کند، اما وقتی فراوانی درخواست داده ها از توانایی سرور بانک اطلاعاتی تجاوز کرد و سرور برنامه تقاضا هایی بیشتر از آنچه توانایی پاسخ گویی به آنها دارد را دریافت کرد، مشکلات عمده ای ایجاد می شود. پس این مدل نیز دارای ضعف هایی است که در مدلهای بعدی رفع خواهند شد.
و اما در مورد قابلیت اطمینان[2]. اگر هر بخشی از برنامه که در بیشتر حالتها بر روی سرور قرار می گیرد دچار نقص شود، برنامه از کار افتاده و برنامه هایی که در حال کار با برنامه مذکور هستند آسیب می بینند. پس این مدل از نظر پایداری نیز دارای مشکلاتی است.
مجموعه تکنولوژی هایی که در .NET وجود دارند شامل تمام عوامل مورد نیاز برای پیاده سازی راه حلهای تجاری، از ابزارهای طراحی گرفته تا سرویسهای تحت وب می شود. این مجموعه که باعث افزایش پایداری، قابلیت اعتماد ، انعطاف پذیری و قدرت مدیریت میشود، تمام مشکلات مذکور در مدل قبلی را رفع می کند. برای نمونه .NET سرور ها را از این محدودیت که فقط با کاربران در ارتباط باشند رها می کند و به آنها اجازه می دهد که علاوه بر ارتباط با کاربران، با دیگر سرور ها نیز در پشت پرده ارتباط داشته باشد. تعامل سرور ها از طریق وب سرویس های ارائه شده، نمونه خوبی از این زمینه می باشند.

[1] ممکن بود به هر دلیلی سرویس گیرنده ای که کاربر از آن استفاده می کند از شبکه قطع شده و یا متوقف شود.

[2] Reliability



سروری که در کنار کار با سرویس گیرنده با دیگر وب سرویس ها نیز تعامل دارد و از آنها استفاده می کند

کارایی و مقیاس پذیری:

اگر یک سیستم قدرت مقیاس پذیری[1] نداشته باشد، هیچ راهی برای تهیه برنامه هایی که به تعداد زیادی کاربر به طور همزمان پاسخ دهد نخواهد بود. در حقیقت، در سیستم های سازمان مقیاس هر چه کارایی سیستم مهم باشد، پایداری و مقیاس پذیری آن از اهمیت بیشتری برخوردار خواهد بود.

در تعریف کارایی[2] سیستم میتوان گفت که این عبارت به تعداد سیکل هایی که یک پردازنده نیاز دارد تا اجرای یک متد را به پایان برساند اطلاق می شود. اما منظور از مقیاس پذیری تعداد کاربرانی است که می توانند به صورت همزمان یک وظیفه خاص را در سرور اجرا کنند. برای مثال تصور کنید که یک متد از برنامه، اطلاعات خواسته شده را با سرعتی شگفت انگیز باز میگرداند. ارزش این متد به علت به کارگیری صد درصد پردازنده است. بنابراین در حالی که کارایی چنین سیستمی خوب است اما مقیاس پذیری آن ضعیف است، زیرا پردازنده تنها یک یا دو کاربر همزمان که درخواست اجرای متد مذکور را دارند را می تواند پشتیبانی کند. مقیاس پذیری یک سیستم کاملاً به معماری درونی برنامه وابسته است.

مزایای .NET:

برای توسعه دهندگان پاسخ به سوال "چرا .NET؟" در دو گروه مزایای موجود در IDE[3] مربوط به .NET و نیز مفهوم .NET قرار می گیرد.

یک سری از تکنولوژی های موجود در .NET شامل مجموعه ای از تکنولوژیهایی است که در برقراری ارتباط بین پلت فرم های گوناگون مورد استفاده قرار می گیرد مانند پروتکلهای استانداردی مثل HTTP، و نیز تکنولوژیهایی که به پلت فرم خاصی وابسته نیستند مثل XML. این تکنولوژیها باعث می شود که سیستم های ایجاد شده با .NET بتوانند با سیستم های قبلی مانند COM[4] و CORBA[5] از طریق وب تعامل داشته باشند. به علاوه به علت اینکه توجه به پلت فرمهای مقصد نیز هم اکنون رفع شده است، بنابراین برنامه نویسان می توانند از آن لحاظ نگرانی نداشته باشند و روی نیازهای تجاری برنامه تمرکز کنند.

---

[1] Scalability
[2] Performance
[3] Integrated Development Environment
[4] Component Object Model
[5] Common Object Request Broker Architecture

البته باید توجه داشته باشید که .NET یک هدف در حال تغییر است. اگر چه بسیاری از مواردی کـه امـروزه مـا از ایـن تکنولـوژی میدانیم تغییر نخواهد کرد، اما همانطور که روز به روز تکنولوژیهای جدید تری در حال توسعه هستند، به یقین این موارد به .NET اضافه شده و یا مواردی از آن حذف خواهند شد.

قبل از پیدایش اینترنت بیشتر برنامه ها مبتنی بر چند فرم ویندوزی و بدون ارتباط با جهان خارج بود. اما بـا پیـدایش اینترنـت شـاهد گسترش روزافزون نرم افزارهای تحت وب و نیز تحت شبکه هستیم که موجب تغییرات بسیاری در دنیای امروزه شده اند.

در گذشته بیشتر سایتهای وب شامل صفحات ایستا بودند که با ارائه ی چندین صفحه اطلاعات که به یکـدیگر لینـک شـده بودنـد، نیازهای کاربران را برطرف می کردند. اما با گسترش روز افزون اینترنت کاربران نیاز به صفحات پویایی که اطلاعات خاص خودشـان را نمایش دهد را روز به روز بیشتر احساس می کردند.

پذیرش استانداردهای همگانی:

یکی ار مهمترین جنبه های .NET پذیرش استانداردهای صنعتی همگانی توسط مایکروسافت است که XML یکـی از مهمتـرین آنها است. تصور برخی از افراد از این مورد این است که XML پیشرفته ترین تکنولوژی عصر حاضر است، که قطعا این امر برداشـتی نادرست است. اما XML یکی از بهترین راههای موجود برای یکپارچه سازی سیستمهای نا متجانس محسوب میشود.

بزودی تمامی سرورهای مایکروسافت به یک سرور .NET تبدیل خواهند شد که XML و پلت فرم .NET را کاملاً پشتیبانی می کنند. بنابراین به وسیله پیاده سازی پروتکلهای استاندارد توسط این سرور ها، برنامه های ارائه شده مبتنی بر سرورهای مایکروسـافت قدرت تعامل با پلت فرمهای دیگر را نیز خواهند داشت. در جدول زیر لیستی از سرور های مایکروسافت و همچنین توضیح کاربرد آنها ذکر شده است.

سرور	شرح
Microsoft Application Center Server 2000	این سرور برنامه هـای مبتنـی بـر وب را توزیـع کرده و همچنین سرور ها را به صورت گروهـی مدیریت می کند.
Microsoft BizTalk Server 2000	پردازش های تجاری را پیاده سازی مـی کنـد و اطلاعات را بـه وسـیله یـک رابـط اسـتاندارد و پذیرفته شده ارائه می دهد.
Microsoft Commerce Server 2000	بـرای ایجـاد برنامـه هـای تجـارت الکترونیـک استفاده می شود.
Microsoft Exchange Server 2000	قابلیت انجام مبادلات از طریق اینترنت را فراهم می کند.
Microsoft Host Integration 2000	تعامل با کامپیوترهای بزرگ را برقرار می کند.
Internet Security And Acceleration 2000	به عنوان یک دیوار آتش عمل می کند.
Microsoft SQL Server 2000	سرویسهای تجزیـه و تحلیـل و نیـز نگـه داری بانک اطلاعاتی را ارائه می دهد.

سرویس های وب:

یکی از عواملی که ممکن است کاملاً جدید به نظر برسد وب سرویس ها هستند. وب سرویس ها در حقیقت اصولی هستند که زیرساخت بیشتر استراتژی های .NET را تشکیل می دهند و به جرات می توان گفت که هدف اصلی از ایجاد .NET به شمار می روند. وب سرویس ها سرویس هایی هستند که به وسیله یک برنامه تحت اینترنت ارائه شده و توسط دیگر وب سرویس ها یا برنامه های سرویس گیرنده استفاده می شوند. وب سرویس ها برپایه ابزارهای استاندارد مثل XML و HTTP تولید می شوند و مستقل از پلت فرم و محیط تولید آنها می باشند. بنابراین برای استفاده از آنها نیازی به .NET نیست.
با ترکیب HTTP و XML و تولید SOAP, .NET یک راه حل قابل اعتماد را برای توسعه برنامه های تحت وب ارائه می دهد. استاندارد SOAP که در حقیقت انتقال داده های XML به وسیله HTTP است، پایه و اساس سرویس های وب محسوب می شود. نه تنها SOAP می تواند از امکانات COM بهرمند شود بلکه قدرت تعامل و استفاده از مزایای استانداردهای دیگری مثل CORBA را نیز دارد.

ویژگی های محیط توسعه Visual Stadio.NET:

با وجود اینکه ظاهر VS.NET نسبت به قبل تغییرات قابل ملاحظه ای کرده است، پیشرفت اصلی این برنامه به علت تکنولوژی هایی است که محیط توسعه ی .NET بر پایه آنها استوار است. در حقیقت این تکنولوژی ها است که باعث تولید سریع برنامه همراه با پایداری و قابلیت اعتماد محیط های توسعه کلاسیک مانند C++ می شود.

Common Language Runtime:

یکی از مهمترین پیشرفت های ویژوال استودیو اضافه شدن یک محیط مدیریت زمان اجرای مستقل از زبان برنامه نویسی است که Common Language Runtime یا CLR نامیده می شود. CLR قدرت طراحی به هر زبانی را در یک محیط مدیریت شده می دهد که کمتر دچار نشست حافظه می شود و یا با فراهم آوردن metadata برای کامپوننت ها به آنها اجازه خطا یابی و یا امور دیگر را می دهد.
ویژگی های کنترل نسخه ی برنامه ها و یا امنیت آنها در CLR، توزیع برنامه را به صورت یک سرویس بسیار ساده تر می کند. همچنین CLR باعث سادگی و نیز تسریع تولید برنامه های تحت وب نسبت به نسخه های قبلی می شود. در باره ی این قسمت از .NET Framework در ضمیمه ی بعد صحبت شده است.

زبان های برنامه نویسی .NET:

با ظهور .NET مایکروسافت زبان C# را که ترکیبی از قدرت C++ و راحتی ویژوال بیسیک بود معرفی کرد اما حقیقت در مورد دو زبان اصلی .NET بدین صورت است که C# و ویژوال بیسیک از لحاظ امکانات بسیار مشابه اند.

تمامی زبان های NET. یک زیر مجموعه از امکانات CLR را ارائه می دهند، اما سطح پشتیبانی آنها از CLR متفاوت است[1]. برای طراحی بیشتر برنامه ها انتخاب زبان تفاوت چندانی ندارد. اما با وجود قابلیت برنامه نویسی با بیش از ۲۰ زبان در NET. می توان با در نظر گرفتن اینکه بعضی از آنها برای محاسبات علمی بسیار مناسب اند و بعضی دیگر برای پردازش ورودی/خروجی ایده آل هستند زبان مناسبی را برای نوشتن یک برنامه انتخاب کرد.
نکته: با وجود اینکه C# و VB کاملا شی گرا هستند اما ممکن است در شرایطی VC++ کارایی بیشتری را ارائه دهد.

Intermediate Language:

این زبان که در NET. یک زبان سطح میانی محسوب می شود، مستقل از ساختار درونی پردازنده است. تمام کد های NET. در ابتدا به این زبان کامپایل می شوند و به علت مستقل بودن این زبان از پردازنده، کامپوننت تولیدی می تواند مستقل از پلت فرم عمل کند. از آنجا که IL نیز دستورالعمل های خود را دارد، کد های IL قبل از اجرا در هر پلت فرمی باید با استفاده از کامپایلر های مخصوص آن پلت فرم به صورت دستورهای پردازنده ی محلی در آید که این امر توسط JIT صورت می گیرد.

تکامل برنامه نویسی لایه ای:

یکی از مزایای برنامه نویسی مبتنی بر کامپوننت، توانایی برنامه نویس برای تقسیم عملکرد برنامه به چند کامپوننت قابل مدیریت و عمومی است. این امر باعث استفاده مجدد از کد و انعطاف پذیری برنامه می شود. همچنین یکی از مزایای اصلی این مدل قدرت تقسیم برنامه به چند لایه موازی است به گونه ای که برنامه را بر اساس سرویس های آن به چند بخش منطقی تقسیم می کند.

تعریف:

یک **لایه برنامه**[2] به یک قسمت از برنامه که به صورت موازی با دیگر قسمت ها در حال تعامل است گفته می شود. هر لایه ی در حال کار در برنامه، وظیفه خاصی را انجام می دهد. به عبارت دیگر پیاده سازی برنامه های چند لایه عبارت است از تقسیم فیزیکی برنامه به چند قسمت و توزیع هر قسمت بر روی یک سرور.
مدل برنامه نویسی لایه ای در حقیقت برای حل بسیاری از مشکلات برنامه های امروزی ایجاد شده است. از این دسته مشکلات می توان از مدیریت متمرکز، محاسبات توزیع شده، کارایی برنامه ها و مقیاس پذیری آنها نام برد.

مدیریت متمرکز:

در یک محیط که به صورت متمرکز مدیریت می شود، تغییرات موردنظر در یک قسمت مرکزی صورت می گیرند و این تغییرات به سرتاسر سیستم توزیع می شوند. یک سیستم با مدیریت مرکزی از تعداد نقاط محدودیت قابل مدیریت است.

---

[1] برای اطلاعات بیشتر در این مورد به بخش "خصوصیات عمومی زبانهای برنامه نویسی" در ضمیمه ی بعد مراجعه کنید.
[2] Application Tier

برای نمونه MS Application Center 2000، یک ابزار برای توزیع و مدیریت برنامه ها، سیستمی است که از مدیریت متمرکز استفاده می کند. Application Center به علت مدیریت گروهی سرور ها باعث افزایش مقیاس پذیری و قابلیت اعتماد برنامه ها می شود. قابلیت مهمتر این سیستم در این است که اگر یک برنامه در چند سرور اجرا شود و تمام آن سرور ها به وسیله ی این سیستم به صورت گروهی مدیریت شود، می توان از این سیستم برای مدیریت مرکزی برنامه نیز استفاده کرد.
سیستم های مدیریت متمرکز، عموماً مدیریت یک گروه از سرور ها را به سادگی کنترل یک سرور مستقل انجام می دهند و زمانی که یکی از کامپوننت های برنامه تغییر کند یا به روز شود، این تغییرات بین تمام سرور هایی که برنامه مذکور را پشتیبانی می کنند توزیع می شود.

محاسبات توزیع شده:

در یک محیط محاسبات توزیع شده، پردازش ها بین چند سیستم و حتی در صورت نیاز بین چند منطقه تقسیم می شود. هدف اصلی این امر افزایش مقیاس پذیری و کارایی شبکه، افزایش قدرت و تحمل نقص است.

کارایی:

همانطور که ذکر شد منظور از کارایی تعداد سیکل های مصرفی برای انجام یک وظیفه خاص است. با وجود اینکه یک وظیفه ممکن است به سرعت انجام شود که این امر حاکی از کارایی قابل قبول برنامه است، اما این کارایی باعث مقیاس پذیری خوب برنامه نمی شود. کاربران همیشه کارایی برنامه را در زمان پاسخ گویی آن می دانند. زمانی که یک برنامه به خوبی مقیاس پذیر نباشد کاربر تصور می کنند که برنامه کارایی خوبی ندارد. بنابراین برای یک برنامه نویس درک تفاوت بین کارایی و مقیاس پذیری از اهمیت بسزایی برخوردار است.

مقیاس پذیری:

همانطور که ذکر شد منظور از مقیاس پذیری تعداد کاربرانی است که می توانند یک وظیفه خاص را به صورت همزمان از سیستم درخواست کنند. نکته اصلی در طراحی برنامه های مقیاس پذیر، طراحی کامپوننت ها به صورتی است که با کمترین مقدار استفاده از منابع، یک وظیفه خاص را اجرا کند. نتیجه ی مطلوب برنامه ای خواهد بود که به تعداد موردنظر از کاربران، به صورت همزمان و در یک زمان قابل قبول پاسخ دهد.

قواعد و دستورات تجاری:

منظور از قواعد تجاری در حقیقت محدودیت هایی است که تجارت موردنظر بر روی یک برنامه اعمال می کند. کاربرد این قواعد تجاری بر روی جامعیت و یکپارچگی اطلاعات یک برنامه تاثیر می گذارد و عدم اجرای کافی این قواعد موجب تاثیرات منفی برنامه

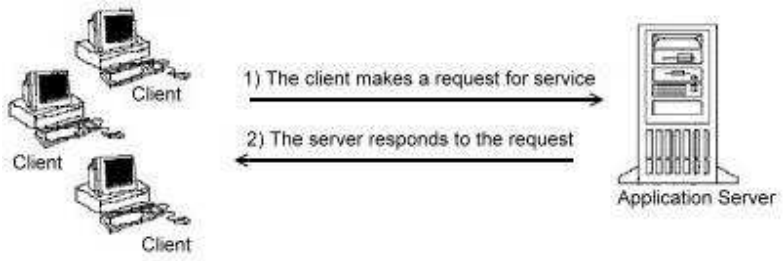
می شود. با وجود اینکه ممکن است یک راه دقیق برای پیاده سازی این قواعد وجود نداشته باشد، اما همـواره راههـای قابـل قبـولی برای اجرای این قواعد به حد کافی وجود دارد.

راحتی کاربر:

در دنیایی که درک بصری یک واقعیت است، صرف زمان و هزینه بر روی برنامه هایی که معماری و قدرت خوبی دارند اما استفاده از آنها مشکل است بی ثمر است. بنابراین حتی اگر یک برنامه کارایی بسیار بالایی داشته باشد و بتواند بـه طـور همزمـان بـه ۱۰۰۰۰۰ کاربر پاسخ دهد، اما کاربران از استفاده از آن برنامه نفرت داشته باشند، نمی توان آن را یک برنامه ی قابـل قبـول دانـست. بـا وجـود اینکه این مطالب در این قسمت بررسی نمی شود، اما محیط های طراحی .NET ایجاد برنامه هایی با ظاهر بـسیار زیبـا را تـا حـد ممکن برای برنامه نویسان ساده کرده اند.

برنامه های دو لایه:

مدل برنامه نویسی دو لایه عموماً به عنوان مدل سرویس گیرنده/ سرویس دهنده یاد می شود و این دو واژه با یکدیگر معادل اند. در مدل دو لایه، برنامه ی سرویس گیرنده تقاضای اطلاعات یا خدمات را مانند پست الکترونیکی، به یک سرور یا سرویس می دهد.
در محیط های سرویس گیرنده/ سرویس دهنده معمولا پردازش برنامه ها بین سرویس گیرنده و سرویس دهنـده تقـسیم مـی شـود. برنامه سرویس گیرنده رابط کاربر را نمایش می دهد و اطلاعات لازم را از کاربر دریافت می کند. برنامه سرور هم معمولاً بـر اسـاس اطلاعات دریافت شده از سرویس گیرنده خدمات مناسبی را ارائه می دهد.
شکل زیر یک نمونه از این برنامه ها را نشان می دهد. با این که مدل نمایش داده شده در شکل کـاملاً درسـت اسـت امـا در حالـت واقعی تعداد کاربرانی که از یک سرور استفاده می کنند بیشتر از چیزی است که در شکل مشاهده می کنید.



مدل دو لایه (سرویس گیرنده/سرویس دهنده)

مدیریت کد در برنامه های دو لایه:

دو جنبه ی مختلف در مدیریت کد برای محیط های سرویس گیرنده/ سرویس دهنده وجود دارد کـه عبـارت انـد از مـدیریت کـد در قسمت سرویس گیرنده و مدیریت کد در قسمت سرویس دهنده.
مدیریت کد در قسمت سرور ها عموماً واضح و روشن است. تا زمانی که سرور مورد نظر عضو یک گروه از سـرور هـا نیـست، توزیـع تغییرات اعمال شده بر روی کد فقط در یک قسمت انجام می شود. اما اگر سرور عضو یک گروه از سرور ها باشد، تغییرات بـه وجـود

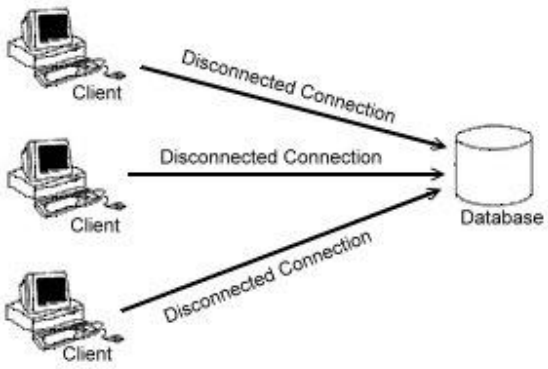
آمده در کد باید به صورت دستی یا اتوماتیک به تمام سرور ها اعمال شود که معمولا این امر به وسیله نرم افزارهای مدیریتی کلاستری سرور ها مانند MS Application Center 2000 انجام می شود.
از سوی دیگر مدیریت کد در بخش سرویس گیرنده عموماً مشکل تر از مدیریت کد در سرور ها است. تمام تغییراتی که در برنامه ایجاد می شوند باید به تمام کامپیوترهای سرویس گیرنده اعمال شود. چنین توزیع هایی که بایستی به صورت همزمان صورت گیرند معمولاً به سطح بالایی از هماهنگی و بر اساس کامپیوتر هایی که تغییرات را دریافت می کند به یک بسته ی قابل توزیع مورد اطمینان نیاز دارند.
توزیع کد در بخش سرویس گیرنده با مشکلات دیگری نیز از قبیل هماهنگی با سرور روبرو است. برای مثال تغییراتی که در برنامه ی سرور اعمال می شود نیاز به بروز رسانی برنامه های سرویس گیرنده دارد و تا زمانی که تمام برنامه های سرویس گیرنده بروز نشود برنامه سرور بدون استفاده می ماند.

کارایی:

استفاده از مدل سرویس گیرنده/سرویس دهنده بر روی کارایی شبکه نیز تاثیر می گذارد. در این مدل تمام تقاضاهای اطلاعات و خدمات بایستی از طریق شبکه بین سرویس دهنده و سرویس گیرنده جا به جا شود و این امر باعث محدودیت در شبکه می شود. این مشکل زمانی که کاربران زیادی در حال استفاده از سیستم باشند بیشتر مشهود خواهد بود.
هنگامی که کارایی شبکه نزول کند، تقاضاهای دریافتی برای خدمات صف می شوند و وقتی تقاضاهای در حال انتظار افزایش یابد سیستم از کار می افتد. در این حال تنها راه حل قطع کردن تمام کاربران از شبکه، بازگرداندن تمام تراکنش ها به حالت قبلی و در بعضی مواقع راه اندازی مجدد سرور است.
برحسب پروتکل مورد استفاده در شبکه، توزیع برنامه های سرویس گیرنده/سرویس دهنده در سطح جهانی غیر ممکن است. زیرا ممکن است برنامه ی سرویس گیرنده نیاز داشته باشد در همان شبکه ای قرار گیرد که برنامه سرور قرار گرفته است.

دسترسی داده ها:

هنگام ارزیابی کارایی، دسترسی به داده ها نیز باید مدنظر قرار گیرند. نه تنها بیشتر برنامه های سرویس گیرنده نمی دانند که چگونه به سرور بانک اطلاعاتی وصل شوند، بلکه بیشتر آنها احتیاج به یک ارتباط اختصاصی با بانک اطلاعاتی دارند. این مورد که بازای تمام کاربران در حال ارتباط با سرور بانک اطلاعاتی باید یک اتصال فعال برقرار باشد، چه از نظر هزینه پرداختی برای ارتباط با سرور چه از نظر مدیریت مربوط به اتصالات پرهزینه است. همچنین این اتصالات قابلیت به اشتراک گذاشته شدن بین برنامه ها را ندارند که این امر تاثیر منفی بر مقیاس پذیری برنامه دارد.



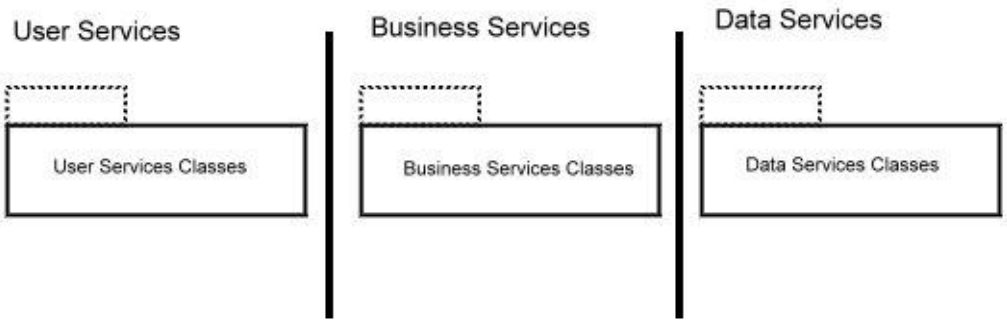
هر سه سرویس گیرنده به اتصال اختصاصی به بانک اطلاعاتی نیاز دارند.

قواعد تجاری:

در مدل دو لایه یا سرویس گیرنده/سرویس دهنده فقط دو قسمت برای ایجاد و اجرای قواعد تجاری وجود دارد: قسمت سرویس گیرنده و قسمت سرور که خدمات بانک اطلاعاتی را ارائه می دهد. البته اجرای این قواعد در سمت سرور منطقی تر است زیرا این امر امکان اینکه یک سرویس گیرنده تغییرات جدید را دریافت نکند را رفع می کند.

برنامه های سه لایه:

مدل برنامه نویسی سه لایه با تقسیم برنامه های دو لایه به سه قسمت: سرویس های کاربران، سرویس های تجاری و سرویس های اطلاعاتی باعث بهبود برنامه های سرویس گیرنده/سرویس دهنده می شود. این تقسیم موجب افزایش مقیاس پذیری و قابلیت اعتماد برنامه می شود.



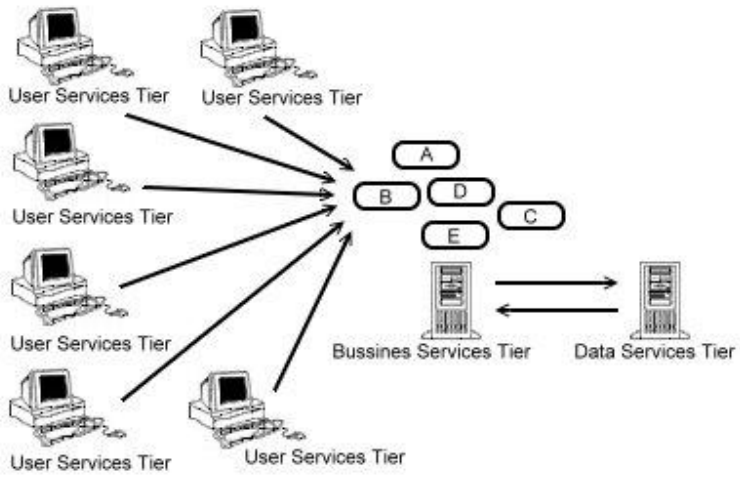
مدل منطقی معماری سه لایه

سرویس های کاربران:

لایه سرویس های کاربران که همچنین لایه ارائه دهنده[1] نیز نامیده می شود از فایل های اجرایی ویندوز یا صفحات وب مثل HTMLهای دینامیک و یا صفحات ASP تشکیل شده اند. لایه خدمات کاربران در حقیقت لایه ای است که اطلاعات سرور را به کاربر نمایش می دهد و اطلاعات مورد نیاز سرور را از کاربر دریافت می کند. در مدل سه لایه، لایه ارائه دهنده به دانستن ساختار بانک اطلاعاتی و یا هر سرویس دیگری که به وسیله لایه سرویس های اطلاعاتی ارائه می شود احتیاج ندارد.

سرویس های تجاری:

لایه دوم، لایه سرویس های تجاری است. در برنامه های سه لایه، در حقیقت این لایه مسئول چگونگی دسترسی به اطلاعات است. این مسئولیت شامل دریافت تقاضای اطلاعات موردنیاز از طرف لایه سرویس های کاربر، فرستادن آن به لایه ی داده ای و بازگرداندن نتایج درخواست ها به کاربر است. این لایه می تواند، و در بیشتر شرایط باید، قواعد تجاری اعمال شده در برنامه را نگهداری و اجرا کند.
لایه سرویس های تجاری قادر به انجام تمام مواردی است که لایه ارائه دهنده درخواست می کند. با وجود اینکه یکی از مهمترین اهداف این لایه، جداسازی لایه ارائه دهنده از لایه سرویس های اطلاعاتی است اما این لایه کارهای بسیار بیشتری را انجام می دهد. تمام توابع، اعم از محاسباتی و یا وظایف ویژه برنامه، بوسیله این لایه ارائه می شوند. تمام کاربران به واسطه لایه ارائه دهنده به امکانات این لایه دسترسی دارند. این لایه به صورت فیزیکی در یکی از سرورهای موجود در شبکه قرار می گیرد. تمام قواعد تجاری جدید و یا تغییرات اعمال شده در آنها فقط باید در این لایه توزیع شوند، که این امر موجب حذف نیاز به انتشار این قواعد بین کامپیوترهای سرویس گیرنده می شود.



کامپوننت های لایه سرویسهای تجاری به وسیله تمام سرویسهای کاربر استفاده میشوند

سرویس های اطلاعاتی:

[1] Presentation Layer

لایه سرویس های اطلاعاتی امکان دسترسی به اطلاعات را برای لایه سرویس های تجاری فراهم می کند که این لایه نیز اطلاعات را در اختیار کاربران سرویس گیرنده در لایه سرویس های کاربر قرار می دهد.ADO.NET و سیستم مدیریت بانک اطلاعاتی (DBMS) هر دو در این لایه نگهداری می شوند. ADO.NET راه حل مایکروسافت برای دسترسی اطلاعات در شبکه و MS SQL Server نیز راه حل مایکروسافت برای سیستم های مدیریت بانک اطلاعاتی است. هر دو این موارد دسترسی به اطلاعات را فراهم می کند. ADO.NET روشی را برای دریافت اطلاعات ارائه می دهد و SQL Server موتور بانک اطلاعاتی است که برای نگهداری خود اطلاعات به کار می رود.

مدیریت کد:

مدیریت کد در برنامه های سه لایه بسیار راحت تر از برنامه های دو لایه است و دچار مشکلات کمتری می شود. به دلیل جداسازی منطقی و فیزیکی برنامه دیگر نیازی به یک تیم توسعه برای کل برنامه نیست. طراحان لایه ارائه دهنده می توانند بدون دسترسی به بانک اطلاعاتی به طراحی این لایه بپردازند. برنامه نویسان لایه سرویس های تجاری می تواند از درگیری با لایه ارائه دهنده آزاد شوند و برنامه نویسان بانک اطلاعاتی می توانند بر روی رابطه اطلاعات و یا پیاده سازی قواعد تجاری تمرکز کنند. بواسطه این امر که این لایه ها به طور منطقی مجزا از یکدیگر هستند، هر کدام از آنها می توانند به طور مجزا کامپایل شده و یا مجدداً پیکر بندی شوند بدون اینکه بر لایه های دیگر تاثیر بگذارند.

مقیاس پذیری:

در برنامه های سه لایه مقیاس پذیری بسیار بهبود می یابد، زیرا اتصالات بانک اطلاعاتی می تواند بدون آنکه قطع شود از کاربری که به آن نیاز ندارد گرفته شده و برای دیگر کاربران نگه داشته شود. این امر تعداد کاربرانی که می توانند از طریق لایه میانی (لایه سرویس های تجاری) با بانک اطلاعاتی در تماس باشند را افزایش می دهد. همچنین در این نوع معماری پردازش از سمت سرویس گیرنده به لایه سرویس های تجاری انتقال می یابد. کارایی شبکه به علت ارتباط لایه سرویس های تجاری با لایه بانک اطلاعاتی به جای ارتباط لایه سرویس گیرنده با بانک اطلاعاتی نیز افزایش می یابد. در کل مقیاس پذیری در این سیستم ها به واسطه ی دسته بندی لایه سرویس های تجاری و سرورهای بانک اطلاعاتی افزایش می یابد.
کامپوننت های تجاری می توانند به وسیله ابزارهای ارائه شده توسط COM+ و یاEnterprise Services در حافظه بارگذاری شوند و تا موقع نیاز در آنجا باقی بمانند. این امر موجب افزایش تعداد کاربرانی می شود که لایه سرویس های تجاری می تواند به آنها پاسخ دهد زیرا زمان مورد نیاز برای بارگذاری کامپوننت های لازم حذف می شود.

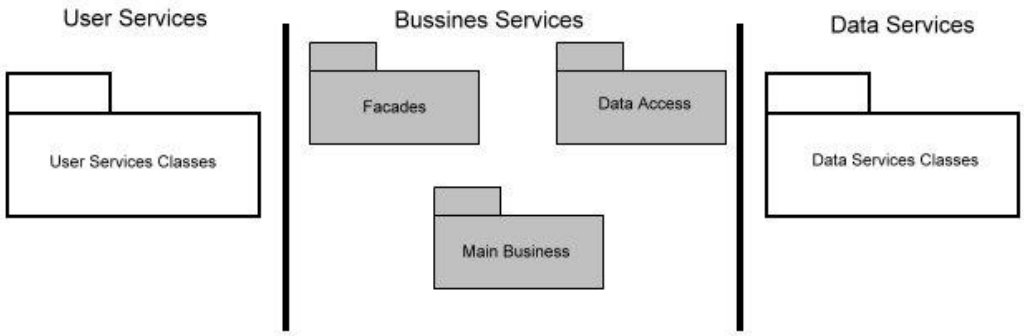
قواعد تجاری:

در برنامه های سه لایه قواعد تجاری نباید در لایه ارائه دهنده(لایه کاربر) اعمال شوند. زیرا در این صورت کاربر، به راحتی می تواند از این قواعد عبور کند. علاوه بر این با قرار دادن یک قاعده تجاری در سمت سرویس گیرنده این عمل باید برای تمامی کامپیوترهای این لایه تکرار شود.
این قواعد می توانند در هر یک از لایه های سرویس های تجاری و یا سرویس های اطلاعاتی قرار داده شوند. زمانی که این قواعد در لایه سرویس های تجاری قرار می گیرند، بایستی اطمینان حاصل شود که کاربران فقط از طریق کامپوننت های این لایه به لایه

سوم دسترسی دارند و نمی توانند از این لایه عبور کرده و به لایه بانک اطلاعاتی دسترسی پیدا کنند. زیرا در غیر این صورت این قواعد به درستی اجرا نمی شوند، اما زمانی که این قواعد در لایه بانک اطلاعاتی اجرا شدند دیگر کاربران توانایی عبور از آنها را نخواهند داشت.
زبانهای برنامه نویسی از قبیل VC++ ، VB.NET و یا C# برای پیاده سازی این قواعد بهینه شده اند. با وجود این، می توان با استفاده از امکاناتی که در برنامه هایی از قبیل SQL Server وجود دارد این قواعد را در لایه بانک اطلاعاتی پیاده سازی کرد.

برنامه نویسی چند لایه:

تقسیم بندی یک برنامه به چند لایه، به صورت استراتژیک، می تواند باعث افزایش مقیاس پذیری، کارایی، انعطاف پذیری و قدرت مدیریت شود، این مورد که به هر لایه یک وظیفه خاص داده شود باعث می شود که طراحان آن لایه، بر روی توسعه و پیکر بندی وظیفه مشخص شده عمل کنند.
هر برنامه ای که شامل بیش از سه لایه شود، جزء برنامه های چند لایه قرار می گیرد. اما در این بخش منظور از برنامه های چند لایه برنامه های پنج لایه است. این برنامه ها مشابه برنامه های سه لایه هستند ولی در آنها لایه سرویس های تجاری به سه لایه یا سه کلاس دیگر تقسیم می شود که عبارت اند از: کلاس خارجی، کلاس اصلی تجاری و کلاس دسترسی اطلاعات.

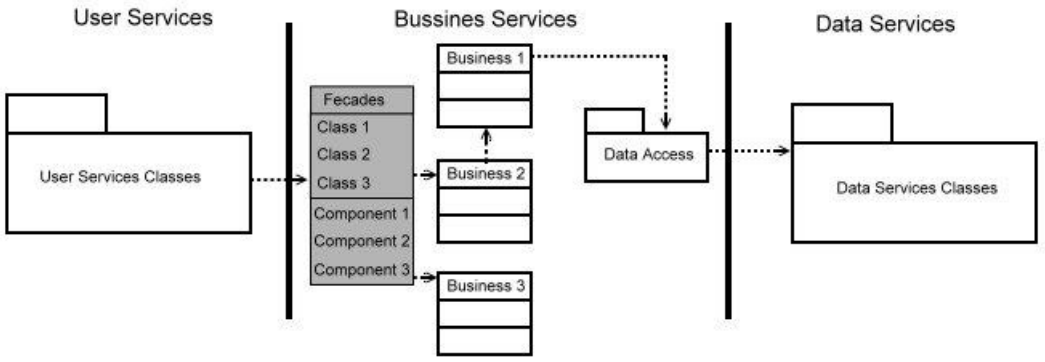


کلاسهای لایه تجاری به چند کلاس جدید تقسیم میشوند

کلاس خارجی:

کلاس خارجی[1] در حقیقت به عنوان یک بافر بین لایه سرویس های کاربر و امکانات ارائه شده توسط لایه سرویس های تجاری برنامه عمل می کند. یکی از مزایای استفاده از این کلاس ها این است که می توان چندین کلاس برای این قسمت تعریف کرد و با استفاده از آنها اطلاعات ایستا را به اطلاعاتی که به سمت سرویس گیرنده فرستاده می شوند اضافه کرد و بدین وسیله به طراحان و برنامه نویسان لایه سرویس های تجاری در تسریع طراحی و توسعه کمک کرد.

[1] Facades Layer

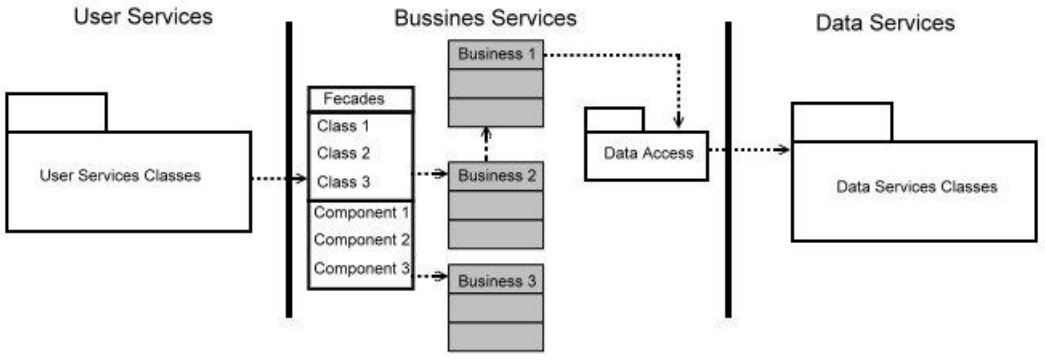


لایه خارجی به عنوان یک بافر برای کامپوننت ها عمل میکند

یکی دیگر از مزایای استفاده از این کلاسهای خارجی، توانایی از بین بردن پیچیدگی موجود در توابع لایه سرویس های تجاری است. کامپوننت های موجود در لایه سرویس های تجاری اغلب به گونه ای طراحی می شوند که بتوانند به طور عمومی به وسیله چند برنامه مورد استفاده قرار گیرند. همچنین هر کاربر و یا سرویسی از سوی صفحات وب، نیاز به نمونه سازی و یا بارگذاری چندین کامپوننت برای اجرای یک وظیفه خاص دارد. به وسیله کلاس های خارجی، کاربر لایه سرویس گیرنده فقط نیاز دارد یک نمونه از کلاس خارجی را شبیه سازی کرده و از پیچیدگی کار در لایه میانی جدا می ماند.

کلاس اصلی تجاری:

کلاس اصلی تجاری یا لایه مرحله تجاری[1]، در حقیقت فراهم کننده وظایف اصلی تجاری برنامه است که عبارت اند از: اجرای قواعد تجاری، تضمین کردن کارایی منطق تجاری و فراهم کردن دسترسی به کامپوننت های اطلاعاتی. به عبارت دیگر این کلاس، هوشمندی واقعی برنامه را ارائه می دهد. کلاسهای خارجی نیز با استفاده از کامپوننت های این لایه وظیفه موردنظر خود را انجام می دهند.

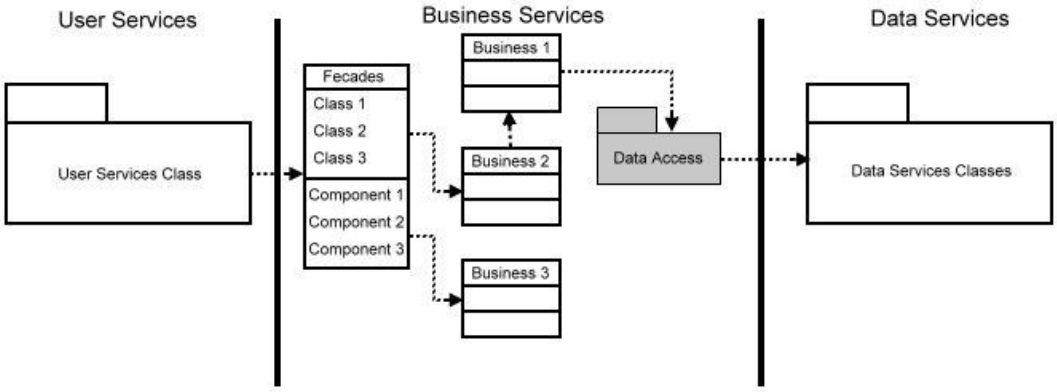


رابطه بین کلاس اصلی تجاری با کلاسهای خارجی و کلاسهای دسترسی اطلاعات

[1] Business Level Layer

کلاسهای دسترسی اطلاعات:

کامپوننت های دسترسی به اطلاعات یا لایه دسترسی به اطلاعات[1] امکان استفاده از اطلاعات را برای لایه اصلی تجاری فراهم می کند. در مدل پنج لایه فقط این لایه به ساختار بانک اطلاعاتی دسترسی دارد بنابراین تغییرات در بانک اطلاعاتی فقط نیاز به ایجاد تغییرات در این لایه دارد. قسمت های دیگر این برنامه نیازی به اطلاع از زیر ساخت بانک اطلاعاتی ندارند.
بدون توجه به نوع برنامه استفاده شده در لایه بانک اطلاعاتی، ADO.NET روش پیشنهادی مایکروسافت برای این لایه محسوب می شود. ADO.NET توانایی برقراری ارتباط با انواع منابع اطلاعاتی را از قبیل SQL Server، Oracle، Sybase، Access، Word، Excel و... را دارد. در طراحی لایه ای معمولا منبع اطلاعاتی، یک سیستم مدیریت بانک اطلاعاتی (DBMS) است که دسترسی به اطلاعات را فراهم می کند. هنگام استفاده از ADO.NET طراحان می توانند از کامپوننت های این کلاس برای ایجاد پرس و جو های مختلف استفاده کنند. علاوه بر این استفاده از روش های داخلی سیستم های بانک اطلاعاتی نیز امکان پذیر است. مثلا در SQL Server می توان از پروسیجر های آماده[2] استفاده کرد که حدود ۴۰ در صد سریعتر از روش های دیگر است.



کامپوننت های دسترسی اطلاعات برای دریافت و ارسال اطلاعات از لایه سرویسهای اطلاعاتی به لایه سرویسهای تجاری آماده هستند

سرویس های وب:

یک سرویس وب، یک تابع یا مجموعه ای از توابع است که از طریق اینترنت قابل دسترسی است و از ترکیب XML برای ارائه داده ها و HTTP برای انتقال داده ها بهره می برد. وب سرویس ها نیز مانند کامپوننت های COM و یا .NET توابع خود را برای کاربران عرضه می کنند. استفاده کنندگان این توابع عموماً برنامه های سرویس گیرنده هستند. در نتیجه وب سرویس ها در طراحی لایه ای، به راحتی به عنوان یک لایه الحاقی قابل استفاده اند.
وب سرویس ها این امکان را به یک توسعه گر می دهند که بدون طراحی کامل یک برنامه[3]، توابع و وظایف یک برنامه را برای سرویس گیرنده ارائه دهد. یک طراح برنامه های سرویس گیرنده می تواند در ساخت برنامه خود از امکانات یک یا چند سرویس مبتنی بر وب بهره مند شود.

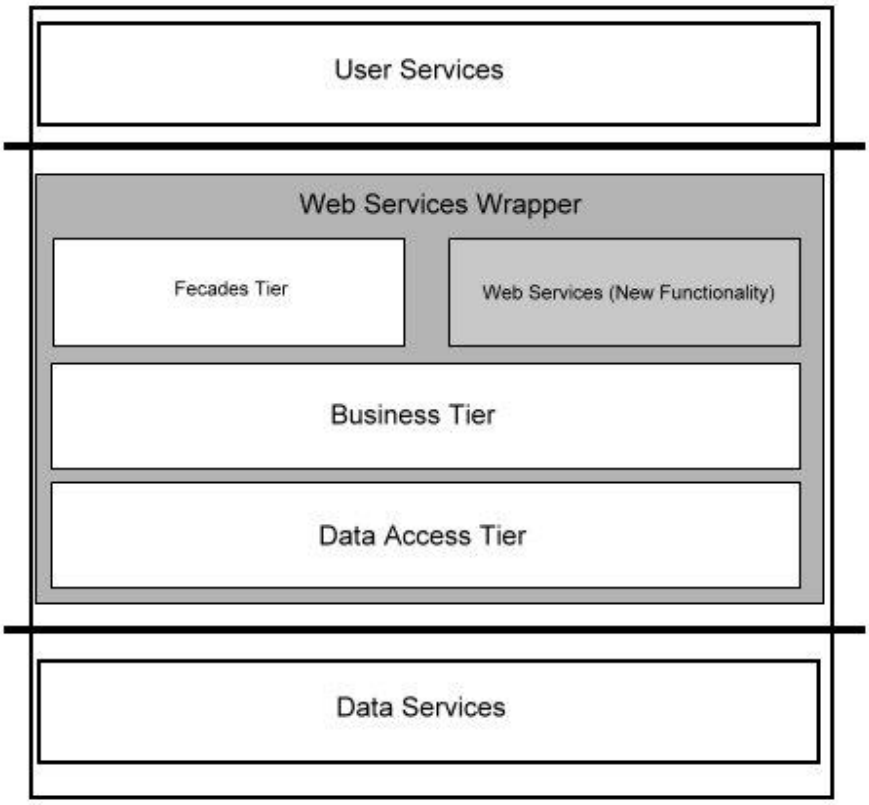
---

[1] Data Access Layer
[2] Stored Procedures

[3] زیرا این برنامه ها نیازی به طراحی رابط کاربری ندارند.

مدل لایه وب سرویس ها:

همانند لایه خارجی(کلاسهای خارجی که پیشتر توضیح داده شد)، وب سرویس ها باعث سادگی استفاده از توابع و نیز کاهش پیچیدگی های لایه میانی برای برنامه های در حال استفاده می شود. به عبارت دیگر برای استفاده از یک وب سرویس در یک برنامه ی چند لایه که دارای لایه های دسترسی اطلاعات، لایه اصلی تجاری و لایه خارجی است، تنها کافی است که توابع مدنظر از لایه خارجی گرفته شده و توسط وب سرویس عرضه شوند و مابقی توابع توسط لایه خارجی قابل دسترسی باشند. شکل زیر یک مدل منطقی از یک وب سرویس را ارائه می دهد که در آن توابعی که توسط لایه خارجی فراهم نشده اند، به وسیله این وب سرویس قابل دسترسی هستند.



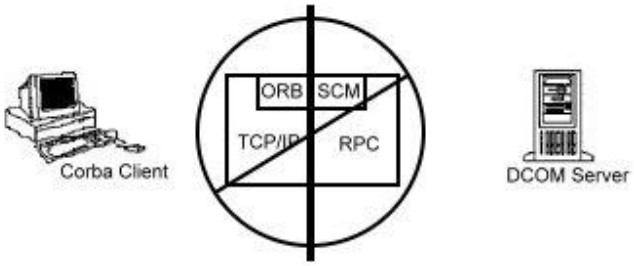
یک وب سرویس میتواند علاوه بر توابعی که در لایه خارجی ارایه میشوند، توابع مکمل را نیز به لایه سرویس گیرنده ارایه کند

چرا وب سرویس ها؟

یکی از اصلی ترین سوالاتی که در این بخش با آن روبرو هستیم این است که با وجود اینکه می توان از تکنولوژی های قبلی و راه حل های پیشین مایکروسافت همانند DCOM برای دسترسی از راه دور استفاده کرد چه نیازی به وب سرویس ها است؟

دلیل اصلی این امر مستقل بودن وب سرویس ها از پلت فرم است. با وجود اینکه DCOM بسیاری از مشکلات توزیع برنامه ها و نیز مقیاس پذیری آنها را حل می کند، اما وابسته به پلت فرم است. با استفاده از وب سرویس ها و تکنولوژی های استاندارد صنعتی از قبیل XML،HTTP و SOAP می توان علاوه بر بهره مند شدن از تمام امکانات سیستم های قبلی، از قابلیت سازگاری بین پلت فرم های گوناگون نیز استفاده کرد.

سیستم های قدیمی مانند DCOM،RPC و MSMQ و ... همگی سعی بر این دارند که پلی را برای ارتباطات اینترنتی ایجاد کنند و با وجود اینکه این موارد در داخل پلت فرم های مایکروسافت موفق بوده اند اما در ارتباط با دیگر پلت فرم ها (همانطور که در شکل نشان داده شده است) با مشکلات زیادی روبرو بودند. با استفاده از وب سرویس ها سعی شده است که عمده مشکلات این ناهماهنگی رفع شود.



ناسازگاری قابلیت استفاده از راه دور بدون پروتکل استاندارد

ضمیمه ی ۳: معماری پلت فرم .NET Framework

در این ضمیمه سعی شده است که معماری درونی پلت فرم .NET به صورتی گذرا مورد بررسی و تحلیل قرار گیرد. هدف از این بخش نگاهی سطحی به معماری داخلی .NET Framework و نیز معرفی تکنولوژی هایی است که این پلت فرم دربر دارد. همچنین در این بخش پروسه تبدیل یک سورس کد به یک برنامه قابل توزیع و روش اجرای آن در سیستم عامل مورد بررسی قرار می گیرد.

کامپایل سورس کد به ماژول های مدیریت شده:

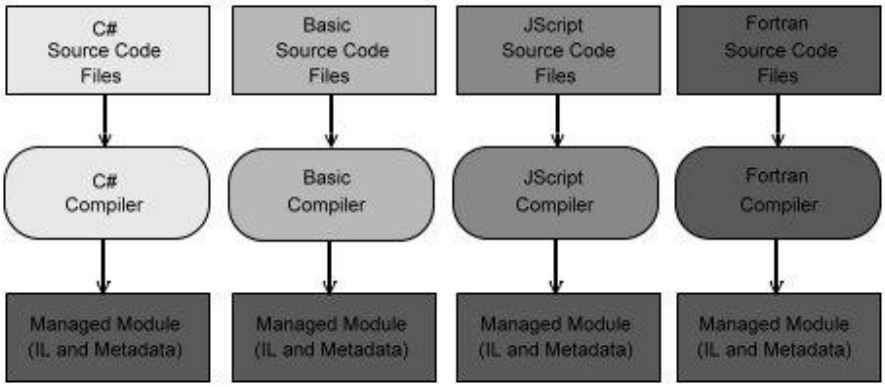
در طراحی یک برنامه، اولین مرحله تعیین نوع برنامه ای است که قصد ایجاد آن را داریم، برای مثال برنامه ی تحت وب، برنامه ی تحت ویندوز، وب سرویس و یا انواع برنامه های دیگری که در .NET می توان ایجاد کرد. فرض میکنیم که این قسمت مهم از برنامه به پایان رسیده است و مدل کلی برنامه و ویژگی های آن با جزئیات کامل مشخص شده اند.
در مرحله ی بعد بایستی زبانی که برنامه با آن نوشته خواهد شد انتخاب شود. این مرحله از اهمیت بالایی برخوردار است، زیرا زبانهای مختلف امکانات متفاوتی را ارائه می دهند. برای نمونه در C (یا زبانهای وابسته به آن مانند C++ و ...) که یک زبان نسبتا سطح پایین است طراح برنامه قدرت کنترل کامل بر روی سیستم را دارد، میتواند به روش دلخواه حافظه را مدیریت کند و یا به راحتی ترد های[1] جدید در برنامه ایجاد کند. از سوی دیگر زبانهایی مثل ویژوال بیسیک، به طراح برنامه قدرت ایجاد سریع رابطهای قوی گرافیکی کاربر را میدهد. به علاوه این زبانها به راحتی میتوانند با اشیای COM و یا بانکهای اطلاعاتی رابطه برقرار کنند.
البته در .NET با تغییراتی که نسبت به سیستم های قبلی به وجود آمده است، تا حد ممکن زبانهای مختلف به یکدیگر شبیه شده اند و از قابلیتهای نسبتاً یکسانی برخوردار هستند. دلیل این مورد نیز در این است که برنامه های نوشته شده در این زبانها، هنگام اجرا از یک محیط مخصوص به نام CLR استفاده می کنند. CLR یا Common Language Runtime همانطور که از اسم آن نیز مشخص است یک محیط زمان اجرا برای برنامه هایی است که تحت .NET نوشته می شوند. بنابراین ویژگیها و قابلیتهای موجود در CLR برای تمام زبانهایی که از آن استفاده می کنند قابل استفاده است. برای مثال اگر CLR توانایی ایجاد ترد ها را داشته باشد، تمام زبانهای برنامه نویسی که از آن استفاده می کنند نیز قابلیت ایجاد ترد ها را دارند. اگر این محیط از Exceptionها برای پیگیری استثنا های برنامه استفاده کند، تمام زبانها نیز همین روش را دنبال خواهند کرد.
در حقیقت در زمان اجرا این مورد که در طراحی برنامه از چه زبانی استفاده شده است تفاوتی ندارد. بنابراین هنگام انتخاب زبان برنامه نویسی مهمترین موضوعی که باید مدنظر قرار گیرد این است که با استفاده از چه زبانی می توان برنامه را ساده تر و سریعتر پیاده سازی کرد.
سوالی که ممکن است در این قسمت ایجاد شود این است که اگر استفاده از زبانهای مختلف تفاوتی ندارد، پس دلیل وجود چند زبان برای برنامه نویسی چیست؟ در پاسخ به این سوال میتوان گفت که زبانهای مختلف گرامر و سینتکس های متفاوتی دارند. اهمیت این مورد را نباید ناچیز در نظر گرفت. برای مثال طراحی یک برنامه برای استفاده در امور اقتصادی، با استفاده از گرامر موجود در زبانهایی مثل APL نسبت به Perl موجب چندین روز صرفه جویی در زمان پیاده سازی برنامه می شود.
مایکروسافت چندین کامپایلر برای استفاده از CLR برای زبانهای مختلف ارائه داده است که در مجموعه ای به نام Visual Studio قرار دارند. این زبانها عبارتند از:

[1] Thread

- Visual C++ with Managed Extensions
- C#
- Visual Basic
- Jscript
- J#
- IL که یک اسمبلر سطح میانی به شمار می رود.

علاوه بر مایکروسافت شرکتهای دیگر نیز برای زبانهای خود کامپایلر هایی را عرضه کرده اند که CLR را بـه عنـوان محـیط زمـان اجرای نهایی مورد استفاده قرار داده اند. تاکنون کامپایلر هایی بـرای زبانهـای زیـر ارایـه شـده انـد:Alice، APL، Cobol، Component Pascal، Eiffel، Fortran، Haskell، Mercury، ML، Mondrian، Oberon، Perl، Python، RPG، Scheme و Smalltalk.

شکل زیر پروسه ی کامپایل فایلهای سورس کد را نمایش میدهد. همانطور که در شکل مشخص است، بدون توجه به کامپایلری که مورد استفاده قرار می گیرد خروجی تمام آنها یک ماژول مدیریت شده[1] است. ماژول های مدیریت شده در حقیقت فایلهـای اجرایـی[2] استاندارد ویندوز هستند که برای اجرا شدن به CLR نیاز دارند.



کامپایل سورس کدها به ماژول های مدیریت شده

یک ماژول مدیریت شده از قسمتهای مختلفی تشکیل شده است که در جدول زیر توضیح داده شده است:

بخش	توضیح
PE Header	در ایـــن هـــدر مـــواردی از قبیـــل نـــوع فایـــل (GUI, CUI, DLL) و یا تاریخ های مربـوط بـه ایجـاد فایل، آخرین تغییرات و ... قرار دارند. برای فایلهایی کـه فقـط شامل کد IL هستند این قسمت در نظر گرفته نمی شود امـا

[1] Managed Module – منظور کد هایی هستند که در زمان اجرا شدن به وسیله ی CLR مدیریت می شوند. در مقابل ماژول های مـدیریت شـده، ماژول های مدیریت نشده وجود دارند که به صورت عادی و بدون نظارت یک محیط زمان اجرا مانند CLR، به وسیله ی سیستم عامل اجـرا مـی شـوند. نمونـه برنامه های مدیریت شده، برنامه هایی است که به صورت عادی به وسیله ی .NET ایجاد می شوند. نمونه برنامه های مـدیریت نـشده نیـز، فایلهـای اجرایـی معمولی هستند که خارج از محیط .NET ایجاد شده اند.

[2] Portable Executable Files (PE Files)

CLR header	برای فایلهایی که شامل کد زبان ماشین هستند این قسمت محتوی اطلاعاتی راجع به کد نیز هست. این قسمت شامل اطلاعاتی است که فایل مورد نظر را به یک ماژول مدیریت شده تبدیل میکند (این قسمت از فایل به وسیله CLR و یا برنامه های وابسته تفسیر میشود). این قسمت شامل نسخه CLR مورد استفاده برای فایل، مکان و اندازه metadata در فایل، منابع، تعدادی فلگ خاص و آدرس ورودی مربوط به فایل آغازین برنامه در metadata است.
Metadata	هر ماژول مدیریت شده شامل جدول هایی است که به نام metadata شناخته می شوند. دو نوع جدول کلی در این قسمت وجود دارند. نوع اول شامل جداولی است که اطلاعات مربوط به کلاس ها و توابع موجود در برنامه را نگهداری می کند. نوع دوم شامل جداولی است که محتوی کلاس ها و توابع خارجی است که به وسیله این فایل مورد استفاده قرار گرفته است.
کد های Intermediate Language (IL)	کدی است که کامپایلر موقع کامپایل سورس کد تولید می کند. موقع اجرای برنامه، CLR این کد را به وسیله ی JIT به کد زبان ماشین تبدیل می کند.

بیشتر کامپایلر های قبلی، کد مربوط به یک معماری خاص از پردازنده را تولید میکردند. برای مثال کد هایی مربوط به x86، IA64، Alpha و یا PowerPC. اما کامپایلر های سازگار با CLR هنگام کامپایل سورس برنامه کد های IL تولید می کنند. در حقیقت این کد های IL هستند که از آنها به عنوان ماژول های مدیریت شده یاد می شود، زیرا CLR مسئول مدیریت این کدها در زمان اجرا است.

علاوه بر تولید کد IL، کامپایلر های سازگار با CLR وظیفه دارند که اطلاعات کامل metadata را نیز در فایل قرار دهند. به بیان ساده، metadata جدولی درونی در فایل است که توضیحاتی راجع به کلاس ها و توابع استفاده شده در برنامه را شامل می شود. به علاوه، metadata شامل جدولی است که محتوی کلاس های خارجی است که برنامه استفاده می کند. Metadata در حقیقت نسخه جدید تکنولوژی هایی مثل Type Library و یا فایلهای IDL[1] است. اما نکته مهم این است که metadata نسبت به این تکنولوژی ها بسیار کاملتر است و بر خلاف موارد ذکر شده، metadataها در فایلی قرار می گیرند که محتوی کد IL است. به دلیل اینکه کامپایلر این اطلاعات را هنگام اجرای کد تولید می کند، امکان ناهماهنگی بین این اطلاعات و کد اصلی نیز از بین خواهد رفت.

بعضی از موارد استفاده metadataعبارت اند از:

۱) Metadata نیاز به فایل های هدر و یا کتابخانه ها را هنگام کامپایل حذف می کند، زیرا تمام اطلاعات مورد نیاز در مورد توابع و کلاسهای استفاده شده در فایل حاوی کد IL در خود فایل قرار دارند. کامپایلر ها می توانند اطلاعات metadata را به صورت مستقیم از داخل فایلهای مدیریت شده استخراج کرده و از آنها استفاده کنند.

[1] Interface Definition Language

۲) محیط های طراحی از قبیل ویژوال استودیو، از این اطلاعات برای کمک به برنامه نویس هنگام نوشتن کد استفاده می کنند. در حقیقت ویژگی هوشیاری در ویژوال استودیو (IntelliSense) که برای کامل کردن کدها و نمایش اطلاعات لازم در مورد توابع مورد استفاده در کد به کار می رود، با استفاده از تحلیل اطلاعات موجود در metadata انجام می شود.

۳) ویژگی بررسی و تایید امنیت کد، با استفاده از اطلاعات موجود در metadata صورت می گیرد. به کمک این ویژگی CLR موجب می شود که فقط کد هایی که دارای دستورات تبدیل متغییر به صورت امن (Safe) هستند صورت گیرند (Verification).

۴) Metadata به یک شیئ اجازه می دهد که سریالایز شده و در حافظه قرار گیرد[1]، به وسیله شبکه به یک وسیله دیگر منتقل شود و در آنجا مجدداً به حالت اولیه تبدیل شود.

۵) Metadata به GC[2] این امکان را می دهد که طول عمر اشیا را کنترل کند، در صورت لزوم آنها را حذف کرده و حافظه تخصیص داده شده به آنها را آزاد کند. برای تمام انواع اشیا، GC می تواند نوع شیئ را تشخیص دهد و سپس با استفاده از metadata ترتیب اشاره اشیا به یکدیگر را تشخیص دهد.

کامپایلر های زبان های C#، Visual Basic، Jscript، J# و نیز اسمبلر IL همواره ماژول های مدیریت شده ای تولید می کنند که برای اجرا به CLR احتیاج دارند. همانطور که برای اجرای برنامه های VB 6 و یا برنامه های MFC در کامپیوتر مقصد، به کتابخانه های Visual Basic و یا کتابخانه های MFC[3] نیاز است، اجرای فایل های مدیریت شده که توسط این کامپایلر ها تولید می شوند نیز به CLR احتیاج دارند.
کامپایلر C++ مایکروسافت بر خلاف دیگر زبان ها، کد های مدیریت نشده (کد های عادی EXE و یا DLL که هم اکنون وجود دارند) تولید می کند. این فایلها برای اجرا به CLR نیاز ندارند. برای تولید کد مدیریت شده توسط این کامپایلر باید هنگام کامپایل از یکی از سوییچ های خط فرمان استفاده کرد. بین تمام کامپایلر هایی که ذکر شد فقط کامپایلر C++ مایکروسافت این امکان را به برنامه نویس می دهد که در یک ماژول از هر دو نوع کد مدیریت شده و مدیریت نشده استفاده کند. این موضوع هنگام طراحی کد این امکان را به برنامه نویس می دهد که کلاسهای جدید خود را به صورت مدیریت شده ایجاد کرده، ولی همچنان از کلاسهای مدیریت نشده قبلی نیز استفاده کند.

ترکیب ماژول های مدیریت شده در اسمبلی ها:

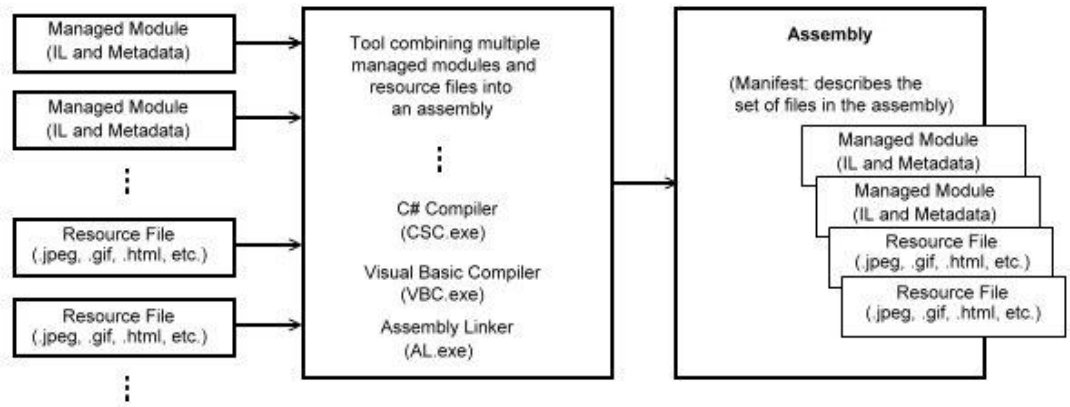
در واقعیت CLR با ماژول های مدیریت شده کار نمی کند، بلکه با اسمبلی ها کار می کند. اسمبلی، مفهومی انتزاعی است که درک آن ممکن است ابتدا مشکل به نظر رسد. در تعریف اول، اسمبلی یک ترکیب منطقی از یک یا چند ماژول مدیریت شده است. در تعریف دوم میتوان گفت که اسمبلی کوچکترین واحدی است که قابلیت استفاده مجدد[4]، تعیین سطوح امنیتی و یا تعیین نسخه[5] را دارد. بسته به انتخاب برنامه نویس می توان به وسیله ابزارهای مورد استفاده برای کامپایل برنامه، فایلهای اسمبلی یا ماژول های مدیریت شده تولید کرد.
شکل زیر به درک مفهوم اسمبلی کمک می کند. همانطور که در این شکل مشاهده می کنید، بعضی ماژول های مدیریت شده و منابع برنامه به وسیله یک ابزار تحت پردازش قرار گرفته اند. این ابزار یک فایل اجرایی تولید می کند که گروه بندی منطقی

---

[1] Serialization
[2] Garbage Collector
[3] Microsoft Foundation Class
[4] Reusability
[5] Versioning

چند فایل را بیان می کند. فایل تولید شده شامل یک بلاک داده ای است که manifest نامیده می شود. Manifest که در واقع به تعدادی از جدول های موجود در metadata اطلاق می شود، شامل اطلاعاتی در رابطه با فایل هایی است که اسمبلی را تشکیل می دهد. به علاوه اطلاعاتی راجع به توابع و کلاس هایی از این اسمبلی که می تواند به وسیله ی دیگر اسمبلی ها مورد استفاده قرار گیرد (مانند کلاسهای public)، و نیز منابع و یا فایلهای اطلاعاتی که با این اسمبلی در ارتباط هستند نیز در manifest قرار داده می شود.

به صورت پیش فرض کامپایلر ها هنگام کامپایل، ماژول های مدیریت شده را به فایلهای اسمبلی تبدیل می کنند. به طور مثال، کامپایلر C# هنگام کامپایل، با اضافه کردن manifest به ماژول مدیریت شده و انجام کارهایی از این قبیل یک اسمبلی تولید می کند. بنابراین هنگام استفاده از این کامپایلر ها، برای تولید یک اسمبلی نیازی به استفاده از سوییچ خاصی نیست. اما در شرایطی که می خواهیم با استفاده از چند ماژول مدیریت شده که هر یک در فایل مخصوص به خود قرار دارند، یک اسمبلی ایجاد کنیم باید ابزارهای دیگری مثل لینکر ها[1] را به کار ببریم.



ترکیب ماژول های مدیریت شده در اسمبلی ها

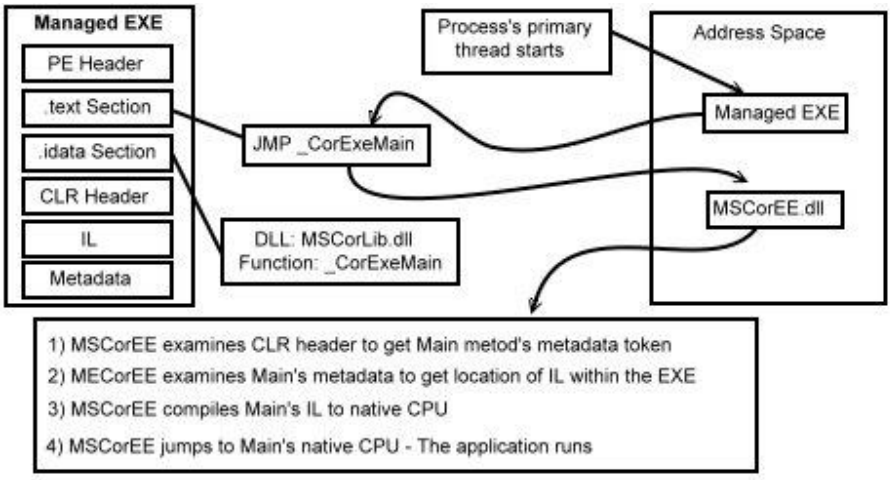
این امر که یک اسمبلی از چند فایل تشکیل شده و هر کلاس در یک فایل قرار داده شود و یا اینکه تمام کلاسهای موجود در اسمبلی، همه در یک فایل قرار گیرند کاملا به طراح برنامه و برنامه نویس بستگی دارد. به طور مثال تصور کنید که می خواهید برنامه ی خود را از طریق وب توزیع کنید. به این ترتیب می توانیم کلاس هایی از اسمبلی مربوط به برنامه که کمتر مورد استفاده قرار می گیرد را در یک فایل و کلاسهای مهم و پر کاربرد را در فایل دیگر قرار دهیم. بدین وسیله فایل حاوی قسمتهای غیر ضروری فقط زمانی از اینترنت دریافت می شود که به آنها نیاز است. این عمل موجب افزایش سرعت بارگذاری برنامه از وب می شود و نیز فضای کمتری را در دیسک اشغال می کند.

یکی دیگر از ویژگی های یک اسمبلی این است که اطلاعات کافی درباره منابع و اسمبلی های دیگری که از آنها استفاده می کند (برای مثال شماره نسخه آنها) را نیز در خود نگهداری می کند. این امر موجب میشود که CLR برای اجرای اسمبلی به موارد اضافی نیاز نداشته باشد. به عبارت دیگر اجرای این اسمبلی ها به ایجاد تغییرات در رجیستری و یا Active Directory ندارد. به همین دلیل توزیع این اسمبلی ها بسیار ساده تر از توزیع کامپونت های مدیریت نشده است. خاصیت استفاده از روش XCOPY برای توزیع برنامه های نوشته شده به وسیله ی .NET نیز به همین دلیل است که اسمبلی های مورد استفاده در یک برنامه می توانند به صورت کامل خود و همچنین فایلهای مورد نیازشان را توصیف کنند.

[1] Assembly Linker (al.exe)

بارگذاری Common Language Runtime:

هر اسمبلی که ایجاد می شود می تواند یا یک فایل اجرایی باشد و یا یک فایل DLL که شامل چند کلاس برای استفاده توسط فایلهای اجرایی است. در هر صورت CLR مسئول اجرای کد های داخل این اسمبلی است. این امر بدین معنی است که برای اجرای این فایلها باید ابتدا .NET Framework در کامپیوتر مقصد نصب شود. البته در نسخه های جدید ویندوز به صورت درونی .NET Framework وجود دارد و نیازی به نصب آن نیست.
هنگام ایجاد یک اسمبلی اجرایی (EXE)، کامپایلر مقداری اطلاعات خاص را به بخش .text در قسمت هدر این فایل اضافه می کند. این اطلاعات زمان اجرای برنامه موجب اجرای CLR می شوند. سپس CLR با تشخیص نقطه ورودی برنامه، به برنامه اجازه می دهد که آغاز به کار کند.
به طور مشابه اگر یک فایل اجرایی مدیریت نشده، با استفاده از LoadLibrary سعی در بارگذاری و استفاده از یک اسمبلی مدیریت شده را داشته باشد، قبل از اجرای فایل برنامه، CLR اجرا شده و کنترل فایل مدیریت شده را در دست می گیرد.



بارگذاری و مقدار دهی اولیه به CLR

هنگامی که کامپایلر یک فایل اجرایی را ایجاد میکند، عبارت زیر را به بخش .text در هدر مربوط به فایل اضافه میکند:

```
JMP _CorExeMain
```

این دستور، یک دستور به زبان اسمبلی است که زیر برنامه ای به نام _CorExeMain را فراخوانی می کند. به دلیل اینکه زیر برنامه ی _CorExeMain در فایل MSCorEE.dll[1] قرار دارد، پس اسم MSCorEE.dll نیز به عنوان یکی از کتابخانه های کلاسی که توسط فایل مورد استفاده قرار گرفته است در بخش .idata اضافه می شود. زمانی که یک فایل اجرایی مدیریت شده فراخوانی می شود، ویندوز با آن به صورت یک فایل مدیریت نشده و معمولی برخورد می کند. برنامه ی مخصوص بارگذاری در ویندوز، با بررسی قسمت .idata شروع به بارگذاری فایل MSCorEE.dll در حافظه می کند و آن را در حافظه قرار می دهد. سپس برنامه با بدست آوردن آدرس تابع _CorExeMain این متد را اجرا می کند.

[1] Microsoft Component Object Runtime Execution Engine

به این ترتیب پروسه اصلی که مربوط به اجرای برنامه است به اجرای تابع _CorExeMain می پردازد. اجرای این تابع منجر به اجرا شدن CLR و بارگذاری آن در حافظه خواهد شد. با اجرا شدن CLR، کنترل برنامه در اختیار آن قرار می گیرد. CLR نیز ابتدا با بررسی بخش هدر CLR مربوط به فایل، نقطه آغازین برنامه را مشخص می کند. سپس CLR کد IL مربوط به تابع فراخوانی شده را به وسیله ی JIT به کد زبان ماشین تبدیل کرده و آن را اجرا میکند (این عمل در پروسه اصلی برنامه صورت می گیرد). در این مرحله اجرای کد مدیریت شده آغاز می شود.
همین شرایط برای ایجاد DLLهای مدیریت شده نیز صادق است. اگر این DLL به وسیله ی یک فایل اجرایی که با .NET ایجاد شده است مورد استفاده قرار بگیرد، پس حتماً تاکنون CLR اجرا شده است و می تواند کنترل این DLL را نیز در دست بگیرد. اما تصور می کنیم که این DLL می خواهد به وسیله ی یک فایل اجرایی عادی، با استفاده از تابع LoadLibrary استفاده شود. در این شرایط قبل از اینکه DLL اجرا شود، باید CLR در حافظه قرار گیرد تا بتواند کد های موجود در DLL را کنترل کند. بنابراین در این شرایط نیز اتفاقاتی همانند اجرا شدن یک فایل EXE که با .NET ایجاد شده است رخ می دهد. یعنی هنگام ساختن DLLهای مدیریت شده، کامپایلر دستور زیر را به بخش .text هدر مربوط به اسمبلی اضافه می کند:

```
JMP _CorDllMain
```

زیر برنامه ی _CorDllMain نیز در فایل MSCorEE.dll قرار داده شده است و این امر موجب می شود که نام این فایل به بخش .idata اضافه شود. هنگامی که ویندوز فایل مربوط به DLL را بارگذاری می کند، ابتدا فایل MSCorEE.dll را در حافظه قرار می دهد (البته اگر تاکنون توسط برنامه ی دیگری بارگذاری نشده باشد) و سپس آدرس تابع مذکور را در حافظه بدست می آورد. پروسه ای که تابع LoadLibrary را در ابتدا فراخوانی کرده بود تا کد های درون DLL را اجرا کند، تابع _CorDllMain را از فایل MSCorEE.dll اجرا می کند. اجرای این تابع نیز موجب راه اندازی CLR شده و بدین ترتیب کد مدیریت شده میتواند به صورت عادی اجرا شود..
مواردی که برای اجرای فایلهای محتوی کد مدیریت شده ذکر شد فقط در ویندوز های 98، SE 98، ME، 4 NT و 2000 لازم است زیرا این سیستم عاملها قبل از عرضه CLR به وجود آمده اند. ویندوز XP و ویندوز سرور ۲۰۰۳ به صورت درونی اجرای فایلهای مدیریت شده را پشتیبانی می کنند و نیازی به موارد ذکر شده نیست. نکته دیگر این است که این توابع مخصوص دستگاههای x86 هستند و در مدل ها و ساختارهای دیگر پردازنده درست عمل نمی کنند.

اجرای کد های مدیریت شده:

همانطور که ذکر شد فایلهای اسمبلی محتوی metadata و کد های IL هستند. IL یک زبان سطح میانی است که توسط مایکروسافت ایجاد شده است. این زبان نسبت به زبان ماشین بسیار سطح بالاتر است. تشخیص اشیای ایجاد شده از کلاسها و دستوراتی برای ایجاد و مقدار دهی اولیه آنها، توانایی فراخوانی توابع داخل اشیا، قابلیت خطا یابی در داخل برنامه ها و نگهداری عناصر در یک آرایه به صورت مستقیم از توانایی های این زبان محسوب می شود. در حقیقت این زبان یک نسخه شیئ گرا از زبان ماشین به شمار می رود.
معمولا برنامه نویسان برای طراحی برنامه از زبانهای سطح بالا مانند C# و یا Visual Basic استفاده می کنند. سپس کامپایلر کد این زبانها را به IL تبدیل می کند.
بعضی از برنامه نویسان عقیده دارند که زبان IL نمی تواند به طور کامل از الگوریتم هایی که آنها استفاده کرده اند محافظت کند. به عبارت دیگر، به نظر آنها می توان یک فایل حاوی کد مدیریت شده تولید کرده و سپس با استفاده از برنامه هایی که می توانند کد IL را نمایش دهند (مانند ildasm[1]) و مشاهده کد IL یک برنامه، به الگوریتم اصلی آن دست پیدا کنند.

[1] IL Disassembler

این امر کاملاً درست است و برنامه هایی که به کد مدیریت شده تبدیل می شوند از امنیت کد پایین تری نسبت به برنامه های محتوی کد زبان ماشین دارند. البته برنامه های تحت وب و یا وب سرویس ها و به طور کلی برنامه هایی که در یک سرور مرکزی اجرا می شوند کمتر از این لحاظ در خطر هستند، زیرا فقط تعداد محدودی از افراد به برنامه های موجود در سرور دسترسی دارند و بنابراین کدها کاملاً ایمن خواهند ماند.

اما برای برنامه هایی که بایستی به صورت عمومی توزیع شوند، میتوان از نرم افزارهای شرکتهای دیگر که کد های IL را گنگ و نامفهوم میکنند استفاده کرد. این برنامه ها تمامی نامهای به کار رفته در کد (مانند نام توابع، متغییر ها و ...) را تغییر می دهند و آنها را با نامهای بدون معنی جایگزین می کنند. به این ترتیب هدف و وظیفه هر تابع از روی نام آن قابل تشخیص نخواهد بود و فرد نمی تواند با مشاهده ی کد IL یک برنامه به سادگی از الگوریتم آن مطلع شود. شایان ذکر است که قدرت این برنامه ها در نامفهوم کردن کد IL تولید شده محدود است، زیرا اگر کد بیش از حد تغییر کند CLR توانایی پردازش آن را نخواهد داشت.

اگر اهمیت کد به حدی بالا است که نتوان به برنامه های نامفهوم کننده اطمینان کرد، میتوان کد های مهم برنامه را در یک کلاس مجزا قرار داد و آن را به صورت کد زبان ماشین کامپایل کرد. سپس از ویژگی CLR برای ارتباط کد های مدیریت شده و کد های مدیریت نشده برای دسترسی به مابقی کد استفاده کرد.

باید در نظر داشت که فقط زبان IL تمام امکانات موجود در CLR را ارائه می دهد. تمام زبانهای دیگر قسمتی از امکانات ارائه شده توسط CLR را شامل می شوند. بنابراین هنگام انتخاب زبان برنامه نویسی بایستی این نکته در نظر گرفته شود و اگر نیاز به امکاناتی از CLR به وجود آمد که در زبان مورد استفاده وجود نداشت می توان آن قسمت از برنامه را به صورت کد IL نوشت و یا از زبانهای دیگر که آن ویژگی را ارائه می دهند استفاده کرد.

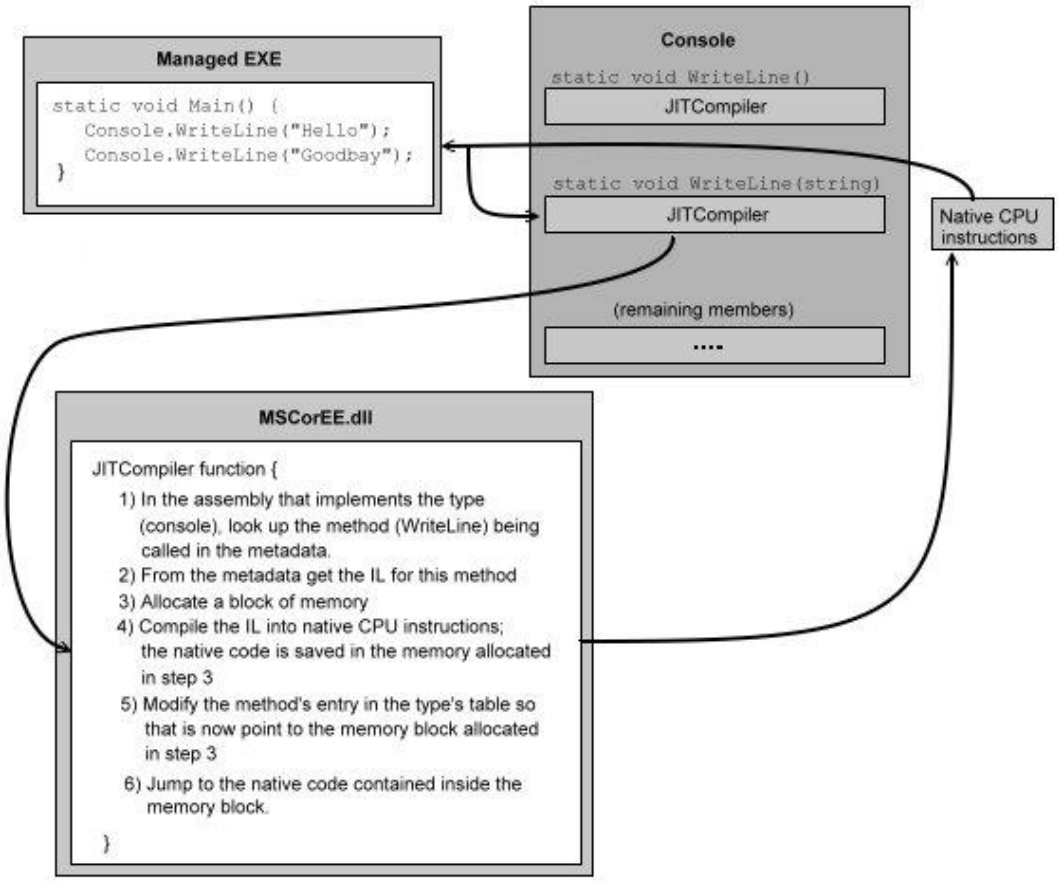
نکته مهم دیگر این است که زبان IL به هیچ پردازنده ای وابسته نیست. این امر بدین معنا است که کد های مدیریت شده در هر پلت فرمی که CLR برای آن ارائه شده باشد اجرا می شود. البته نسخه اولیه ی CLR برای ویندوز های ۳۲بیتی عرضه شده است، اما در هر صورت این موضوع برنامه نویسان را از توجه به پردازنده ای که توسط کاربر مورد استفاده قرار گرفته است آزاد می کند.

برای اجرای کد های مدیریت شده که به زبان IL در فایل ذخیره شده اند ابتدا باید آنها را به کد های زبان ماشین تبدیل کرد. این امر وظیفه ی بخشی از CLR به نام JIT[1] است. شکل زیر مراحلی که هنگام اجرای یک تابع برای اولین بار رخ می دهد را نشان می دهد.

درست قبل از اینکه تابع Main اجرا شود، CLR نوع تمام اشیایی که توسط این تابع استفاده شده اند را تشخیص می دهد. این امر باعث میشود که CLR یک جدول داده ای داخلی برای مدیریت دسترسی به نوع های ارجاعی تشکیل دهد. برای مثال در شکل بالا تابع Main از کلاس ارجاعی Console استفاده می کند که این مورد توسط CLR در جدول داده ای قرار می گیرد. ساختار مذکور بازای هر تابع که در کلاس استفاده شده، یک ردیف اطلاعات را در جدول ایجاد می کند. هر ردیف شامل آدرس مکانی از حافظه است که پیاده سازی تابع در آن قرار دارد. هنگام مقدار دهی اولیه ی این ساختار، تمام ردیفهای آن به یکی از توابع درونی CLR به نام JITCompiler اشاره می کنند.

زمانی که تابع Main برای اولین مرتبه تابع WriteLine را فراخوانی می کند، CLR به جدول داده ای که ایجاد شده بود نگاه می کند تا تشخیص دهد تابعی که باید احضار کند در کدام قسمت از حافظه قرار دارد. اما همانطور که گفتم هنگام ایجاد جدول، مقدار اولیه ی هر تابع به جای اینکه به آدرس خود تابع اشاره کند به آدرس تابعی به نام JITCompiler اشاره می کند. بنابراین در این قسمت به جای تابع WriteLine، تابع JITCompiler به وسیله ی CLR احضار می شود. این تابع وظیفه تبدیل کد IL به کد زبان ماشین را بر عهده دارد. به دلیل این که کد های IL فقط زمانی که به آنها نیاز است به کد زبان ماشین تبدیل می شوند، از این قسمت از CLR عموماً به عنوان JITter و یا JIT Compiler یاد می شود.

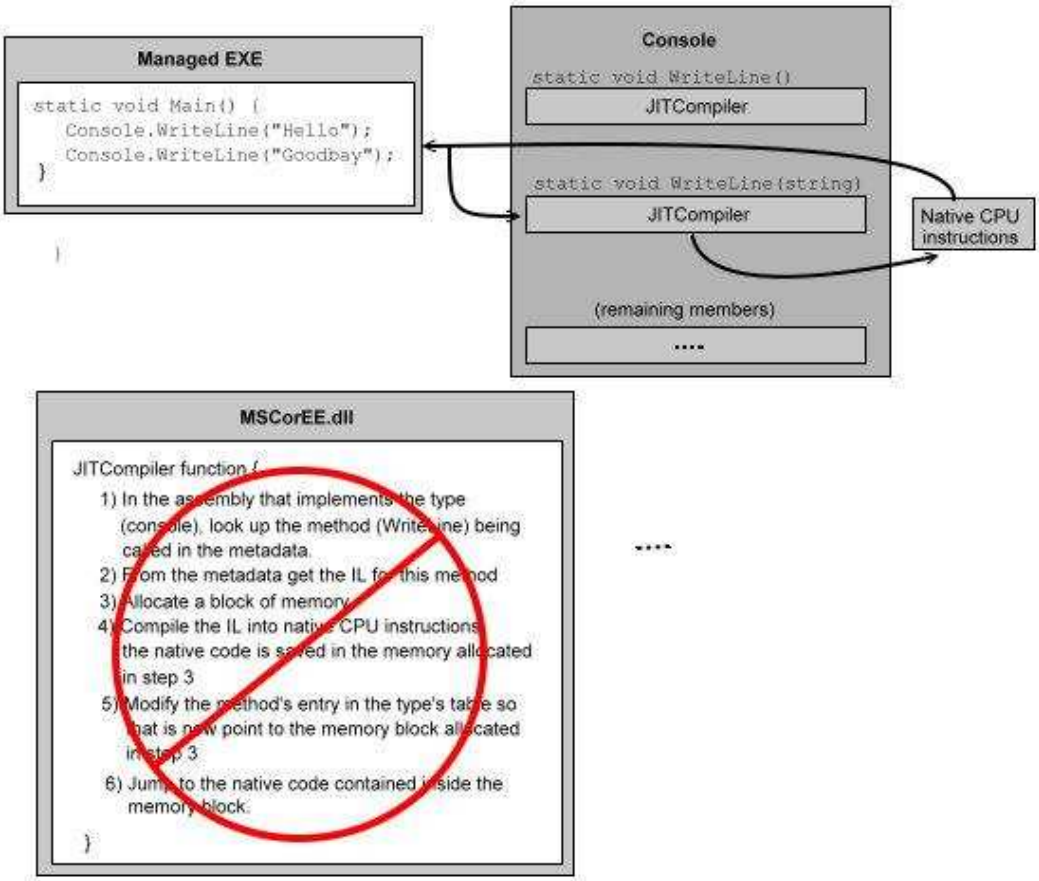
[1] Just-In-Time Compiler



فراخوانی تابع برای اولین بار

هنگام احضار تابع `JITCompiler`، این تابع می داند که چه تابعی در اصل فراخوانی شده است و این تابع در چه کلاسی قرار دارد. پس `JITCompiler` کد `IL` مربوط به تابع فراخوانی شده را بدست آورده و آن را به کد زبان ماشین تبدیل می کند. کد تولید شده در این مرحله در یک بلاک از حافظه که به صورت دینامیک تعیین شده است قرار می گیرد. در مرحله بعد، `JITCompiler` با تغییر ردیف مرتبط با تابع مذکور در جدول داده ای داخلی که توسط `CLR` ایجاد شده بود، آدرس موجود در ردیف را با آدرس کد زبان ماشین تولید شده جایگزین می کند. در نهایت `JITCompiler` به آدرس موجود در حافظه که محتوی کد تولید شده است رفته و کد را اجرا می کند. در پایان این تابع اجرای برنامه به تابع `Main` باز گشته و اجرای برنامه ادامه پیدا می کند.

با بازگشت اجرای برنامه به متد `Main`، همانطور که در شکل مشخص است تابع `WriteLine` برای دومین مرتبه فراخوانی می شود. در این مرتبه، کد `IL` مربوط به تابع احضار شده قبلا توسط `JITCompiler` به کد زبان ماشین تبدیل شده و در حافظه قرار گرفته است. بنابراین اجرای برنامه در این قسمت به بلاک حافظه ای می رود که کد مربوط به تابع قرار دارد و `JITCompiler` اجرا نمی شود.



فراخوانی تابع برای مرتبه دوم

این مورد در مرتبه اول فراخوانی یک تابع باعث ایجاد یک مکث کوتاه در برنامه می شود اما در مرتبه های بعدی احضار تابع، تابع با سرعتی برابر کد های زبان ماشین اجرا می شوند. در حقیقت این کامپایل فقط یک بار در طول برنامه رخ می دهد.

همانطور که ذکر شد کد زبان ماشین تابع فراخوانی شده در یک بلاک حافظه دینامیک قرار میگیرد. بنابراین هنگام خاتمه برنامه کد مورد نظر از حافظه پاک می شود. در مراتب بعدی که برنامه اجرا می شود هنگام فراخوانی تابع مورد نظر، `JITCompiler` مجدداً احضار شده و تابع را به زبان ماشین تبدیل می کند.

طراحانی با پیش زمینه از محیطهایی مانند C++ ممکن است عقیده داشته باشند که این عمل تاثیر منفی بر کارایی برنامه خواهد داشت. با وجود اینکه کد های مدیریت نشده برای یک ساختار پردازنده خاص تولید می شوند، اما هنگام اجرا این کدها به راحتی اجرا می شوند. اما اجرای کد های مدیریت شده شامل دو مرحله می شود. در مرحله اول سورس برنامه بایستی به کد `IL` تبدیل شود و سپس در مرحله بعد در هر بار اجرای برنامه، کد بایستی به زبان ماشین تبدیل شده و در حافظه ذخیره شود. این عمل موجب صرف بیشتر حافظه و زمان پردازنده می شود.

در نگاه اول ممکن است نظر این افراد درست جلوه کند اما حقیقت این است که این اعمال تاثیر چندانی بر کارایی برنامه ندارد و بررسی های انجام شده در این زمینه نشان میدهد که تاثیری که انجام امور ذکر شده برای اجرای کد های مدیریت شده بر سرعت برنامه میگذارد اصلا مشهود و قابل توجه نیست.

حتی در بعضی از شرایط اجرای کد های مدیریت شده سریعتر از کد های مدیریت نشده انجام می شود. این امر بـه ایـن دلیـل است که کامپایلر JIT هنگام کامپایل برنامه ها نسبت به کامپایلر های مدیریت نشده اطلاعـات بیـشتری از محـیط اجـرا در دست دارد. به طور مثال در شرایط زیر کامپایلر JIT سریعتر از دیگر کامپایلر ها عمل می کند:

۱) یک کامپایلر JIT از نوع ساختار پردازنده که برنامه بر روی آن اجرا می شود اطلاع دارد و می تواند از دستورات ویژه ای که آن نوع پردازنده ارائه می دهد حداکثر استفاده را داشته باشد. برای مثال این کامپایلر میتوانـد برنامـه را بـرای پردازنـده های Pentium4 بهینه کند. اما معمولا برنامه های مدیریت نشده به صورتی کامپایل می شوند کـه بـا تمـام انـواع پردازنده ها سازگار باشند. این امر از بهره وری از امکانات ویژه ی پردازنده جلوگیری می کند.
۲) یک کامپایلر JIT می تواند شرایطی را که همواره نادرست هستند تشخیص دهد و آنها را کامپایل نکند. برای مثال یک دستور مشابه زیر در یک تابع هرگز کامپایل نمی شود و این امر موجب می شود که کد مورد نظر برای محیطی که باید در آن اجرا شود بهینه شود

```
if (numberOfCPUs > 1)
{
      .
      .
      .
}
```

اینها تنها تعداد محدودی از دلایلی بودند که نشان دهنده اجرای بهتر کد های مدیریت شده نسبت به کد های مـدیریت نـشده هستند. همانطور که ذکر شد سرعت این کدها برای برنامه ها کاملاً مناسب است. اما با وجود این اگـر همچنـان ایـن احـساس وجود دارد که این کدها سرعت مطلوب را ارائه نمی دهند می توان از ابزارهایی که برای تبدیل کد IL به کد زبـان ماشـین در .NET Framework SDK وجود دارد استفاده کرد. برای مثال ابزاری به نام ngen.exe وجود دارد کـه کـد زبان ماشین مربوط به یک فایل محتوی کد IL را در دیسک ذخیره می کند. هنگامی که CLR مشاهده کنـد کـه کـد زبـان ماشین فایلی که قصد اجرای آن را دارد در دیسک موجود است از آن کد به جای کد IL استفاده می کند.

مجموعه کتابخانه کلاس .NET Framework:

علاوه بر CLR، در مجموعه .NET Framework بخش دیگری نیز وجود دارد که شامل یـک کتابخانـه ی کـلاس می شود. این بخش که FCL[1] نام دارد شامل چندین هزار کلاس است که هر کدام وظیفه خاصـی را انجـام مـی دهنـد. ایـن مجموعه با هم، یعنی CLR و FCL، به طراحان اجازه می دهند که چندین مدل برنامه را طراحی کنند که عبارت اند از:

۱) **وب سرویسهای مبتنی بر XML:** وب سرویس ها توابعی هستند که به راحتی از طریق شبکه وب قابـل دسترسـی و فراخوانی هستند. این قسمت در حقیقت اصلی ترین فلسفه ی ظهور و ابداع .NET محسوب می شود.
۲) **برنامه های تحت وب:** این برنامه ها، برنامه ها و یا وب سایت هایی مبتنی بر صفحات HTML هستند. عموماً ایـن برنامه ها با استفاده از درخواست اطلاعات از سرورهای بانک اطلاعـاتی و چنـدین وب سـرویس، اطلاعـات مـورد نیـاز را دریافت کرده و بعد از تحلیل و انجام پردازش روی آنها، صفحات HTML مبنـی بـر درخواسـت کـاربر را تـشکیل داده و

[1] Framework Class Library

اطلاعات خواسته شده را از طریق مرورگر کامپیوترهای سرویس گیرنده نمایش می دهند. این قسمت در حقیقت لایه میانی یا لایه منطق تجاری در برنامه های چند لایه محسوب می شود.

۳) **برنامه های تحت ویندوز:** برنامه هایی با رابط گرافیکی برای سیستم عامل ویندوز. همانند برنامه های تحت وب، این برنامه ها نیز قدرت برقراری ارتباط با سرورهای بانک اطلاعاتی را داشته و میتوانند از وب سرویسهای مبتنی بر XML استفاده کند. بنابراین در مواقعی که نیاز به برنامه های تحت وب نباشد میتوان از این نوع برنامه ها با امکانات شبکه و نیز قدرت بیشتر در طراحی رابط گرافیکی استفاده کرد.

۴) **برنامه های تحت کنسول در ویندوز:** در مواقعی که نیازی به رابط گرافیکی کاربر نیست و یا یک رابط ساده کافی است می توان از این نوع برنامه ها استفاده کرد. کامپایلر ها و بعضی از برنامه های کاربردی و نیز ابزارها از این دسته هستند.

۵) **سرویسهای ویندوز:** یکی دیگر از انواع برنامه هایی که قابلیت طراحی آن با .NET وجود دارد، سرویسهای ویندوزی است که به وسیله مرکز کنترل سرویس ویندوز (SCM)[1] و نیز .NET Framework قابل کنترل هستند.

۶) **کامپوننت ها و کتابخانه های کلاس:** به وسیله .NET Framework می توان به طراحی کامپوننت هایی پرداخت که به راحتی قابل توزیع و نیز قابل استفاده در محیطهای دیگر باشد.

به دلیل اینکه FCL شامل چندین هزار کلاس می شود، تمام کلاسهای مرتبط به یکدیگر در یک فضای نام گردآوری شده اند. اصلی ترین فضای نام، فضای نام System است که محتوی کلاس Object و تعدادی کلاس پایه ای دیگر است. کلاس Object یک کلاس پایه است که تمام کلاسهای FCL از این کلاس مشتق می شوند. لازم به ذکر است که تمام قواعد شیئ گرایی در کلاسهای FCL رعایت شده اند و بدین ترتیب این امکان را به برنامه نویس داده اند که در هر مرحله بتواند امکانات و یا توابع برنامه را مطابق نیازهای خود تغییر دهد و کلاسهای جدیدی طراحی کند که علاوه بر امکانات کلاسهای قبلی، نیازهای طراح و برنامه نویس را نیز به طور کامل برطرف کند.
جدول زیر بعضی از فضای نام های پر کاربرد در FCL را معرفی کرده و به طور مختصر راجع به آنها توضیح میدهد.

فضای نام	شرح
System	محتوی تمام کلاسهای پایه ای است که به وسیله تمام برنامه ها استفاده می شوند.
System.Collections	مجموعه ای است از کلاسها که برای نگهداری آرایه هایی از اشیا که شامل آرایه های عمومی مثل صف، پشته، لیست پیوندی و ... می شود به کار می رود.
System.Diagnostics	محتوی کلاس هایی برای مستند سازی و نیز خطا یابی در برنامه است.
System.Drawing	شامل کلاس هایی برای نگهداری اشیای مربوط به گرافیک دو بعدی است. این مجموعه بیشتر برای برنامه های تحت ویندوز و نیز نمایش عکس در برنامه های تحت وب به کار می رود.
System.EnterpriseServices	این فضای نام شامل کلاس هایی برای مدیریت تراکنش ها،

[1] Windows Service Control Manager

	فعال سازی JIT، امنیت و ... است که موجب کارایی بیشتر کد های مدیریت شده در سرور می شوند.
System.Globalization	این قسمت شامل کلاس هایی برای پشتیبانی زبانهای دیگر از قبیل حروف آن زبانها، نحوه ی نمایش تاریخ و قالب بندی در آن و ... می شود.
System.IO	کلاس هایی برای انجام عملیات ورودی و خروجی از قبیل کار با فایلها و دایرکتوری ها را دربر دارد.
System.Management	شامل کلاس هایی برای مدیریت دیگر کامپیوترها به وسیله دستگاه های مدیریتی ویندوز یا WMI است.
System.Net	شامل کلاس هایی برای برقراری ارتباطات شبکه ای است.
System.Reflection	شامل کلاس هایی برای بررسی و استفاده از اطلاعات داخل metadata است.
System.Resources	کلاس هایی برای مدیریت منابع خارجی استفاده شده در برنامه را شامل می شود.
System.Runtime.InteropServices	کلاس های این فضای نام به کد های مدیریت شده این امکان را می دهد که از امکانات قبلی ویندوز از قبیل کامپوننت های COM و یا توابع موجود در فایلهای Win32 استفاده کنند.
System.Runtime.Remoting	کلاس هایی را شامل می شود که به دیگر کلاسها اجازه میدهند از را دور مورد استفاده قرار گیرند.
System.Runtime.Serialization	کلاس هایی را در بر دارد که به اشیا این امکان را می دهند به رشته ای تبدیل شوند که معرف آنها باشد و از رشته ای که معرف آنها است به یک شیئ تبدیل شوند.
System.Security	کلاس هایی برای محافظت از اطلاعات و منابع برنامه.
System.Text	کلاس هایی برای کار با متن در قالبهای مختلف از قبیل ASCII و یا Unicode.
System.Threading	شامل کلاس هایی است که برای اجرای همزمان پردازش ها و نیز هماهنگی در دسترسی به اطلاعات مورد استفاده قرار می گیرد.
System.Xml	کلاس هایی برای بررسی و پردازش داده ها در قالب XML را دربر دارد.

سیستم نوع داده ای عمومی:

احتمالاً تا کنون متوجه این نکته شده اید که یکی از مهمترین قسمتهای CLR درباره ی کار با کلاسها است. این کلاسها کارایی لازم را به برنامه و یا کامپوننت در حال طراحی ارائه می دهند. کلاسها موجب می شوند که کد های طراحی شده در یک زبان در زبانهای دیگر قابل استفاده باشد. به علت اینکه هسته اصلی CLR را کلاسها تشکیل می دهند، مایکروسافت یک

سری خصوصیات عمومی در مورد کلاسها در .NET را تحت عنوان CTS[1] عنوان کرد که شامل چگونگی تعریف کلاسها و چگونگی رفتار آنها می شود.
طبق خصوصیات CTS هر کلاس میتواند شامل چند عضو باشد و یا شامل هیچ عضوی نباشد. اعضای قابل تعریف در کلاس عبارت اند از:

۱) **فیلد ها:** یک متغیر داده ای است که معرف وضعیت کنونی شیئ است. فیلد ها به وسیله نام و نوع داده ای آنها مشخص میشوند.
۲) **متد ها:** تابعی است که پردازش خاصی را بر روی شیئ انجام می دهد. این پردازش معمولا شامل تغییر وضعیت آن می شود. متد ها معمولا شامل یک نام، لیست مقادیر ورودی، لیست مقادیر خروجی و نیز عبارت هایی برای تعیین سطح دسترسی به آن هستند.
۳) **مشخصات[2]:** برای کاربر کلاس این عضو همانند یک فیلد است اما برای طراح کلاس این عضو همانند یک (یا دو) متد است. این نوع عضو های کلاس به طراح و برنامه نویس این امکان را می دهند که قبل از قرار دادن ورودی کاربر در متغیر مربوطه، به بررسی صحت آن بپردازد و یا پردازش های مورد نیاز دیگر را قبل از قرار دادن مقدار انجام دهد. علاوه بر این، این نوع عضوها به طراحان اجازه ایجاد فیلد های فقط خواندنی یا فقط نوشتنی را می دهند.
۴) **رویدادها:** رویدادها مکانیسمی برای اطلاع دیگر اشیا از یک رخداد خاص در شیئ محسوب می شوند. برای مثال یک دکمه می تواند هنگامی که فشرده شد اشیای دیگر را از این اتفاق مطلع کند.

CTS علاوه بر این موارد امکاناتی را برای تعیین سطح دسترسی به اعضا ارائه می دهد. برای مثال اشیایی که به عنوان public در نظر گرفته می شوند توسط کلاسهای دیگر، چه در داخل و چه در خارج اسمبلی قابل دسترسی هستند. از سوی دیگر تعیین یک عضو داده ای به عنوان Assembly (در C#، internal نامیده می شود) باعث میشود آن عضو فقط در کلاسهای موجود در داخل اسمبلی قابل دسترس باشد.
لیست زیر سطح دسترسی های مختلف را برای اعضای داده ای کلاس تعریف میکند.

۱) **Private:** متد مورد نظر فقط به وسیله دیگر متدهای همان کلاس قابل دسترسی است.
۲) **Family:** متد مورد نظر فقط به وسیله کلاس هایی که از آن کلاس مشتق شده اند، بدون توجه به اینکه در همان اسمبلی قرار دارد یا نه، قابل استفاده است.
۳) **Family and Assembly:** متد مورد نظر فقط در توابعی از همان اسمبلی که علاوه بر آن از کلاس مورد نظر نیز مشتق شده اند قابل استفاده است. بسیاری از زبانهای برنامه نویسی از قبیل C# و Visual Basic این نوع سطح دسترسی را ارائه نمی دهند.
۴) **Assembly:** متد مورد نظر فقط به وسیله کلاسهای داخل همان اسمبلی قابل استفاده است.
۵) **Family or Assembly:** متد مورد نظر به وسیله تمام کلاس هایی که از این کلاس مشتق می شوند، در همه اسمبلی ها، قابل استفاده است. علاوه بر این، این متد در همه کلاسهای آن اسمبلی نیز قابل دسترسی است.
۶) **Public:** متد مورد نظر د همه کلاسها صرف نظر از اینکه در چه اسمبلی قرار دارد، قابل دسترسی است.

---

[1] Common Type System
[2] Properties

علاوه بر این، CTS قواعدی را برای وراثت، طول عمر اشیا، توابع مجازی و غیره نیز بیان می کند.

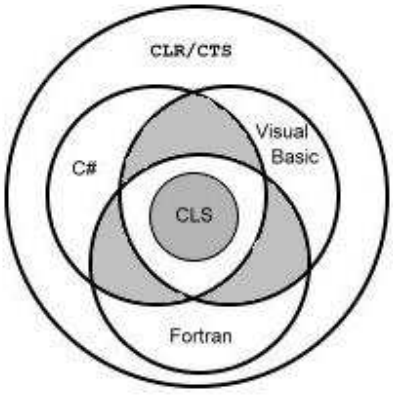
خصوصیات عمومی زبانهای برنامه نویسی:

یکی از امکاناتی که همراه با COM عرضه شد، استفاده از کد های نوشته شده در یک زبان به وسیله زبانی دیگر بود. این امکان به وسیله ی CLR نیز به نحوی بسیار کامل تر ارائه شده است. از سوی دیگر، CLR با یکی کردن تمام زبانها این امکان را می دهد که اشیای تولید شده در یک زبان در زبانهای دیگر قابل استفاده باشد. این قابلیت با توجه به نوع داده ای عمومی مشترک بین همه ی زبانها، استفاده از metadataها برای شرح اطلاعات کلاس و محیط اجرای عمومی به وجود آمده است.

با توجه به اینکه زبانهای برنامه نویسی امکانات کاملا متفاوتی را عرضه می کنند، این قابلیت بسیار شگفت انگیز به نظر می رسد. برای مثال بعضی از زبانهای برنامه نویسی حساس به بزرگی و کوچکی حروف هستند، بعضی سربار گذاری عملگر ها را پشتیبانی می کنند و یا بعضی این امکان را می دهند تا متدهای با تعداد پارامترهای نامحدود ایجاد کنیم.

اگر در ایجاد یک برنامه از امکانات خاصی یک زبان استفاده کنیم، استفاده از آن برنامه در برنامه های دیگر بسیار مشکل خواهد بود. بنابراین برای طراحی کلاس هایی که به راحتی در تمام زبانهای دیگر قابل استفاده باشند بایستی از امکاناتی در برنامه استفاده کرد که تضمین بشود در همه زبانهای برنامه نویسی دیگر وجود دارند. برای مشخص تر کردن این امکانات که در بین همه ی زبانهای برنامه نویسی مشترک است، مایکروسافت یک مجموعه از خصوصیات عمومی را معین کرده که CLS[1] نام دارد.

مجموعه ی CLR و CTS، امکاناتی بسیار بیشتر از آنچه CLS تعیین کرده است را ارائه می دهند. بنابراین اگر کلاس تحت طراحی نباید در زبانهای دیگر استفاده شود می توان از تمام امکانات CTS و CLR استفاده کرد. شکل زیر سعی مفهوم ذکر شده را بیان می کند.



تمام زبانها زیر مجموعه ای از تواناییهای CLR و تمام امکانات CLS را ارایه می دهند

همانطور که در شکل مشخص است هیچ زبانی تمامی امکانات CLR را ارائه نمی کند. بنابراین برای استفاده از تمام این امکانات بایستی از زبان IL کمک گرفت. به جز زبان IL، دیگر زبانها از قبیل C#، Visual Basic، Fortran و غیره فقط قسمتی از امکانات CLR و CTS را ارایه می دهند.

[1] Common Language Specifications

اگر کلاسی به منظور استفاده در دیگر زبانها طراحی می شود، بایستی این نکته را مد نظر قرار داد که در آن کلاس نباید از امکاناتی که خارج از محدوده CLS هستند استفاده کرد. در غیر این صورت آن قسمت از برنامه در زبانهای دیگر که ویژگی مورد استفاده را پشتیبانی نمی کنند قابل استفاده نخواهد بود.
برای مثال در کد زیر یک برنامه با استفاده از خصیصه ی CLSCompliant، به صورت سازگار با CLS تعریف شده است. اما همانطور که مشخص است، تعدادی از ویژگی های به کار رفته در آن قواعد CLS را نقض میکنند. این موارد باعث می شوند که کامپایلر هنگام کامپایل کد با خطا روبرو شود.

```
// Tell the compiler for check the CLS compliance
[CLSCompliant(true)]
// Errors appear because the class is public
public class App
{
	// Error: Return type of 'App.Abc' is not CLS-Compliant
	public UInt32 Abc()
	{
		return 0;
	}

	// Error: Identifier 'App.abc' differing only in case is not
	// CLS-Compliant
	public void abc()
	{ }

	// No error: method is private
	private UInt32 ABC()
	{
		return 0;
	}
}
```

در این کلاس، عبارت [CLSCompliant(true)] باعث می شود که کامپایلر کد هایی از برنامه که به صورت public هستند را برای هماهنگ بودن با CLS بررسی کند. هنگام کامپایل این کد، کامپایلر در دو قسمت برنامه با خطا مواجه می شود. خطای اول به علت نوع داده بازگشتی تابع Abc است که در بعضی از زبانها مانند Visual Basic پشتیبانی نمی شود. خطای دوم نیز مربوط به هم نام بودن توابع Abc و abc است. این توابع فقط در حساس بودن به حروف تفاوت دارند که تمام زبانها به اندازه حروف حساس نیستند.
البته تمام این خطاها به این علت ایجاد می شود که سطح دسترسی کلاس از نوع عمومی است. اگر سطح دسترسی کلاس و یا هر کدام از توابع از public به private تغییر کند خطای مربوط به آن نیز از بین خواهد رفت.

ضمیمه ۴: مدیریت حافظه در .NET

در بخش قبلی، CLR که به عنوان هسته اصلی .NET Framework در نظر گرفته میشود، بررسی شد. به عنوان دو ویژگی مهم CLR، میتوان از توانایی برقراری ارتباط آن بین زبانهای مختلف برنامه نویسی و نیز مدیریت حافظه قوی آن یاد کرد. در توضیح ویژگی اول، همانطور که در قسمت قبلی ذکر شد، CLR با استفاده از یک سری خصوصیات عمومی که به وسیله تمام زبانها پشتیبانی میشود(CLS)، امکان استفاده از کلاس نوشته شده در یک زبان را، در زبانهای دیگر فراهم میکند.
ویژگی مهم دیگر CLR که موجب افزایش کارایی برنامه در آن میشود، مدیریت حافظه آن است. این بخش در CLR بر عهده Garbage Collector است که به علت اهمیت این موضوع در این قسمت کاملا توضیح داده میشود.

درک مبانی کار Garbage Collector.

هر برنامه به نحوی از منابع مشخصی استفاده می کند. این منابع می توانند فایلها، بافر های حافظه، فضا های صفحه نمایش، ارتباطات شبکه ای، منابع بانک اطلاعاتی و مانند اینها باشند. در حقیقت در یک محیط شیئ گرا هر نوع داده تعریف شده در برنامه بعضی از منابع موجود برای برنامه را ارائه می دهد. برای استفاده از هر نوع از این داده ها لازم است که برای ارائه آن نوع مقداری حافظه تخصیص داده شود. موارد زیر برای دسترسی به یک منبع مورد نیاز است:

۱) تخصیص حافظه برای نوع داده ای که منبع مورد نظر را ارائه می دهد. این تخصیص حافظه با استفاده از دستور newobj در زبان IL[1] صورت می گیرد که این دستور از ترجمه دستور new در زبان هایی مثل C# و Visual Basic و دیگر زبان های برنامه نویسی ایجاد می شود.
۲) مقدار دهی اولیه حافظه برای تنظیم حالت آغازین[2] منابع و قابل استفاده کردن آن. توابع Constructor در این نوع داده ها مسئول این تنظیمات برای ایجاد این حالت آغازین هستند.
۳) استفاده از منابع با دسترسی به اعضای موجود در نوع داده.
۴) از بین بردن حالت کلی منابع برای پاک کردن آن.
۵) آزاد سازی حافظه. Garbage Collector مسئول مطلق این مرحله به شمار می رود.

این نمونه به ظاهر ساده یکی از ریشه های اصلی خطاهای ایجاد شده در برنامه نویسی به شمار می رود. مواقع زیادی پیش می آید که برنامه نویس آزادسازی یک حافظه را وقتی دیگر مورد نیاز نیست فراموش می کند. مواقع زیادی پیش می آید که برنامه نویس از یک حافظه که قبلا آزاد شده استفاده کند.
این دو باگ برنامه ها از اکثر آنها بدتر اند زیرا معمولا برنامه نویس نمیتواند ترتیب یا زمان به وجود آمدن این خطاها را پیش بینی کند. برای دیگر باگ ها شما میتوانید با مشاهده رفتار اشتباه یک برنامه آن را به سادگی تصحیح کنید. اما این دو باگ موجب نشت منابع[3] (مصرف بی جای حافظه) و از بین رفتن پایداری اشیا میشوند که کارایی برنامه را در زمانهای مختلف تغییر میدهد. برای کمک به یک برنامه نویس برای تشخیص این نوع خطاها ابزارهای ویژه ای مانند Windows Task Manager و System Monitor ActiveX Control و NuMega Bounds Checker و ... طراحی شده اند.

---

[1] Intermediate Language
[2] Initial State
[3] Resource leaks

یک مدیریت منبع مناسب بسیار مشکل و خسته کننده است. این مورد تمرکز برنامه نویس را بر روی مطلب اصلی از بین میبرد. به همین دلیل نیاز به یک مکانیسم که مدیریت حافظه را به بهترین نحو انجام دهد در این زمینه به وضوح احساس میشد. در پلت فرم .NET این امر توسط Garbage Collector انجام میشود.

Garbage Collection کاملا برنامه نویس را از کنترل استفاده از حافظه و بررسی زمان آزادسازی آن راحت میکند. اگرچه Garbage Collector درمورد منابع ارائه شده توسط نوع داده در حافظه هیچ چیز نمیداند، یعنی Garbage Collector نمیداند چه طور میتواند مرحله ۴ از موارد بالا را انجام دهد: از بین بردن حالت کلی منابع برای پاک کردن آن. برنامه نویس باید کد های مربوط به این قسمت را انجام دهد چون او میداند باید چه گونه حافظه را به درستی و کاملا آزاد کند. البته Garbage Collector میتواند در این زمینه نیز قسمتهایی از کار را برای برنامه نویس انجام دهد. البته، بیشتر نوع داده ها، مانندInt32 ، Point ، Rectangle ، String، ArrayList و SerializationInfo از منابعی استفاده می کنند که احتیاجی به نوع ویژه ای از آزادسازی حافظه ندارند. برای مثال منابع یک شیئ از نوع Point به راحتی و با نابود کردن فیلد های X و Y در حافظه شیئ آزاد میشود.

از طرف دیگر، یک نوع داده که منابع مدیریت نشده ای را ارائه میدهد، مانند یک فایل، یک ارتباط بانک اطلاعاتی، یک سوکت، یک Bitmap، یک آیکون و مانند اینها همیشه به اجرای مقداری کد ویژه برای آزاد کردن حافظه گرفته شده نیاز دارند.

CLR[1] نیاز دارد که حافظه تمام منابع از یک heap مخصوص که managed heap نامیده میشود تخصیص داده شود. این heap شبیه heap زمان اجرای C است و فقط از یک لحاظ متفاوت است و آن این است که در این heap شما هیچ وقت حافظه تخصیص داده شده را آزاد نمیکنید. در حقیقت اشیا موجود در این heap وقتی دیگر نیازی به آنها نباشد آزاد میشوند. این مورد این سوال را ایجاد میکند که چگونه managed heap متوجه میشود که دیگر نیازی به یک شیئ خاص نیست؟

چندین الگوریتم از Garbage Collector در حال حاضر در مرحله آزمایش هستند و هر کدام از این الگوریتم ها برای یک محیط خاص و نیز برای کسب بهترین راندمان بهینه سازی شده اند. در این مقاله روی الگوریتم Garbage Collector استفاده شده در Microsoft .NET Framework CLR متمرکز شده است.

زمانی که یک پروسه مقدار دهی اولیه[2] میشود، CLR یک قسمت پیوسته از آدرس حافظه را برای آن اختصاص میدهد این آدرس فضای حافظه managed heap نامیده میشود. این heap همچنین یک اشاره گر مخصوص هم دارد که ما از این به بعد آن را NextObjPtr می نامیم. این اشاره گر مکان قرار گیری شیئ بعدی را در heap مشخص میکند. در ابتدا این اشاره گر به آدرس ابتدای فضای گرفته شده برای managed heap اشاره میکند.

دستور newobj در زبان IL باعث ایجاد یک شیئ جدید میشود. بیشتر زبانها از جمله C# و Visual Basic برای درج این دستور در کد IL عملگر new را در برنامه ارائه میدهند. این دستور IL باعث میشود که CLR مراحل زیر را انجام دهد:

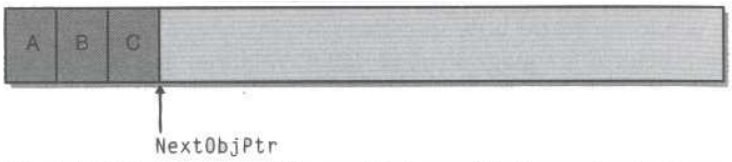
۱) محاسبه تعداد بایت های مورد نیاز برای این نوع داده
۲) اضافه کردن بایت های مورد نیاز برای overhead شیئ. هر شیئ دو فیلد overhead دارد: یک اشاره گر به جدول تابع و یک SyncBlockIndexه. در سیستمهای ۳۲بیتی، هر کدام از این فیلد ها ۳۲ بیت هستند، که ۸ بایت را به هر شیئ اضافه می کند. در سیستم های ۶۴ بیتی، هر کدام از این فیلد ها ۶۴ بیت است که ۱۶ بایت را برای هر شیئ اضافه می کند.
۳) سپس CLR چک میکند که حافظه مورد نیاز برای شیئ جدید در managed heap موجود باشد. اگر فضای کافی موجود باشد این شیئ در آدرسی که NextObjPtr به آن اشاره میکند ایجاد میشود. تابع constructor شیئ مذکور فراخوانی میشود (اشاره گر NextObjPtr به عنوان پارامترthis به constructor فرستاده میشود) و دستور newobj آدرس شیئ ایجاد شده را برمیگرداند. درست قبل از اینکه

---

[1] Common Language Runtime
[2] Initialize

آدرس برگردانده شود، NextObjPtr به بعد از شیئ ایجاد شده پیشروی میکند و مثل قبل آدرسی که باید شیئ بعدی در آن قرار گیرد را در خود نگه میدارد.

شکل زیر یک managed heap را که سه شیئ A و B و C را درخود نگه میدارد را نشان میدهد. اگر یک شیئ جدید ایجاد شود این شیئ دقیقا در جایی که NextObjPtr به آن اشاره میکند قرار میگیرد (درست بعد از شیئ C).



Newly initialized managed heap with three objects constructed in it

در عوض اجازه دهید تخصیص حافظه را در heap زمان اجرای C بررسی کنیم. در یک heap زمان اجرای C تخصیص حافظه برای یک شی به حرکت در میان ساختارهای داده از یک لیست پیوندی نیاز دارد. زمانی که یک بلاک حافظه با اندازه لازم پیدا شد این بلاک حافظه تقسیم میشود و شیئ مذکور در آن ایجاد میشود و اشاره گرهای موجود در لیست پیوندی برای نگه داری در آن شیئ تغییر داده میشوند. برای managed heap تخصیص حافظه برای یک شیئ به معنای اضافه کردن یک مقدار به اشاره گر است. در حقیقت تخصیص حافظه به یک شیئ در managed heap تقریبا به سرعت ایجاد یک متغییر در stack است! به علاوه در بیشتر heapها مانند heap زمان اجرای C حافظه در جایی اختصاص داده میشود که فضای خالی کافی یافت شود. بنابراین اگر چند شیئ بلافاصله بعد از هم در برنامه ایجاد شوند، ممکن است این اشیا چندین مگا بایت آدرس حافظه با هم فاصله داشته باشند ولی در managed heap ایجاد چند شیئ بلافاصله بعد از هم باعث قرار گرفتن ترتیبی این اشیا در حافظه میشود.

در بیشتر برنامه ها وقتی برای یک شیئ حافظه در نظر گرفته میشود که یا بخواهد با یک شیئ دیگر ارتباط قوی داشته باشد یا بخواهد چندین بار در یک قطعه کد استفاده شود. برای مثال معمولا وقتی یک حافظه برای شیئ BinaryWriter ایجاد شد بلافاصله بعد از آن یک حافظه برای FileStream گرفته شود. سپس برنامه از BinaryWriter استفاده میکند که در حقیقت به صورت درونی از شیئ FileStream هم استفاده میکند. در یک محیط کنترل شده به وسیله Garbage Collector برای اشیای جدید به صورت متوالی فضا در نظر گرفته میشود که این عمل موجب افزایش راندمان به دلیل موقعیت ارجاع ها میشود. به ویژه این مورد به این معنی است که مجموعه کارهای پروسه شما کمتر شده و این نیز متشابه قرار گرفتن اشیای مورد استفاده توسط برنامه در CPU Cache است.

تا کنون اینگونه به نظر میرسید که managed heap بسیار برتر از heap زمان اجرای C است و این نیز به دلیل سادگی پیاده سازی و سرعت آن است. اما نکته دیگری که اینجا باید در نظر گرفته شود این است که managed heap این توانایی ها را به این دلیل به دست می آورد که یک فرض بزرگ انجام میدهد و آن فرض این است که فضای آدرس و حافظه بینهایت هستند. به وضوح این فرض کمی خنده دار به نظر میرسد و مسلما managed heap باید یک مکانیسم ویژه ای را به کار برد تا بتواند این فرض را انجام دهد. این مکانیسم Garbage Collector نامیده میشود ، که در ادامه طرز کار آن شرح داده میشود.

زمانی که یک برنامه عملگر new را فراخوانی میکند ممکن است فضای خالی کافی برای شیئ مورد نظر وجود نداشته باشد. heap این موضوع را با اضافه کردن حجم مورد نیاز به آدرس موجود در NextObjPtr متوجه میشود. اگر نتیجه از فضای در نظر گرفته شده برای برنامه تجاوز کرد heap پر شده است و Garbage Collector باید آغاز به کار کند.

مهم: مطالبی که ذکر شد در حقیقیت صورت ساده شده مسئله بود. در واقعیت یک Garbage Collection زمانی رخ می دهد که نسل صفر کامل شود. بعضی Garbage Collector ها از نسل ها استفاده می کنند که یک مکانیسم به شمار می رود و هدف اصلی آن افزایش کارایی است. ایده اصلی به این صورت است که اشیای تازه ایجاد شده نسل صفر به شمار می روند و اشیایی قدیمی تر در طول عمر برنامه در نسل های بالاتر قرار می گیرند. جداسازی اشیا و دسته بندی آنها به نسل های مختلف می تواند به Garbage Collector اجازه دهد اشیایی موجود در نسل خاصی را به جای تمام اشیا مورد بررسی قرار دهد. در بخش های بعدی نسل ها با جزئیات تمام شرح داده می شوند. اما تا ان مرحله فرض می شود که Garbage collector وقتی رخ می دهد که heap پر شود.

الگوریتم Garbage Collection:

Garbage Collection بررسی می کند که آیا در heap شیئ وجود دارد که دیگر توسط برنامه استفاده نشود. اگر چنین اشیای در برنامه موجود باشند حافظه گرفته شده توسط این اشیا آزاد میشود (اگر هیچ حافظه ای برای اشیای جدید در heap موجود نباشد خطای OutOfMemoryException توسط عملگر new رخ میدهد). اما چگونه Garbage Collector تشخیص میدهد که آیا برنامه یک متغیر را نیاز دارد یا خیر؟ همانطور که ممکن است تصور کنید این سوال پاسخ ساده ای ندارد.

هر برنامه دارای یک مجموعه از rootها است. یک root اشاره گری است به یک نوع داده ارجاعی. این اشاره گر یا به یک نوع داده ارجاعی در managed heap اشاره میکند یا با مقدار null مقدار دهی شده است.برای مثال تمام متغییر های استاتیک و یا عمومی[1] یک root به شمار میروند . به علاوه هر متغیر محلی که از نوع ارجاع باشد و یا پارامترهای توابع در stack نیز یک root به شمار میروند. در نهایت، درون یک تابع، یک ثبات CPU که به یک شیئ از نوع ارجاع اشاره کند نیز یک root به شمار میرود.

زمانی که کامپایلر JIT یک کدIL را کامپایل میکند علاوه بر تولید کد های Native یک جدول داخلی نیز تشکیل میدهد. منطقا هر ردیف از این جدول یک محدوده از بایت های آفست را در دستورات محلی CPU برای تابع نشان میدهند و برای هر کدام از این محدوده ها یک مجموعه از آدرس های حافظه یا ثبات های CPU را که محتوی rootها هستند مشخص میکند. برای مثال جدول ممکن است مانند جدول زیر باشد:

Sample of a JIT compiler-produced table showing mapping of native code offsets to a method's roots

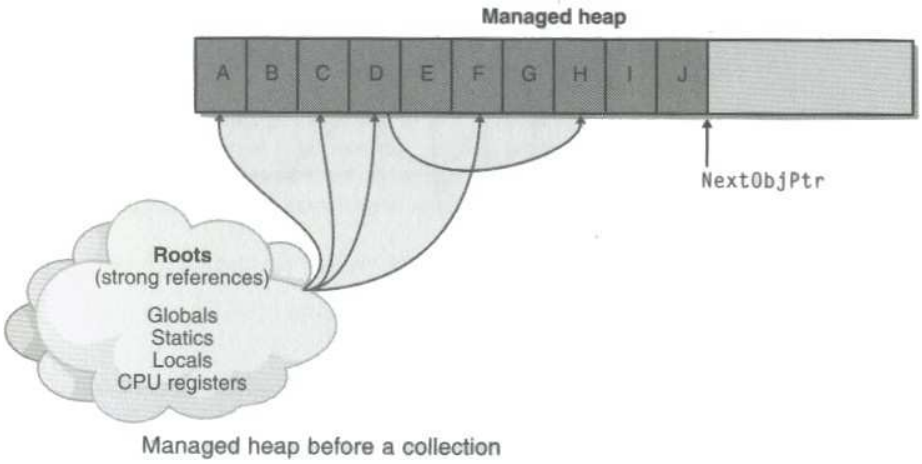
Starting Byte Offset	Ending Byte Offset	Roots
0x00000000	0x00000020	this, arg1, arg2, ECX, EDX
0x00000021	0x00000122	this, arg2, fs, EBX
0x00000123	0x00000145	Fs

[1] Global Variables

اگر یک Garbage Collector زمانی که کدی بین آفست 0x00000021 و 0x00000122 در حال اجرا است آغاز شود، Garbage Collector میداند که پارامترهای this، arg2 و متغییر های محلی fs و ثبات EBX همه root هستند و به اشیایی درون heap اشاره میکنند که نباید زباله تلقی شوند. به علاوه Garbage Collector میتواند بین stack حرکت کند و rootها را برای تمام توابع فراخوانی شده با امتحان کردن جدول داخلی هر کدام از این توابع مشخص کند. Garbage Collector وسیله دیگری را برای بدست آوردن مجموعه rootهای نگه داری شده توسط متغیرهای ارجاعی استاتیک و عمومی به کار میبرد.

نکته: در جدول بالا توجه کنید که آرگومان arg1 تابع بعد از دستورات CPU در آفست 0x00000020 دیگر به چیزی اشاره نمیکند و این امر بدین معنی است که شیئی که arg1 به آن اشاره میکند هر زمان بعد از اجرای این دستورات میتواند توسط Garbage Collector جمع آوری شود (البته فرض بر اینکه هیچ شیئ دیگری در برنامه به شیئ مورد ارجاع توسط arg1 اشاره نمیکند). به عبارت دیگر به محض اینکه یک شیئ غیر قابل دسترسی باشد برای جمع آوری شدن توسط Garbage Collector داوطلب میشود و به همین علت باقی ماندن اشیا تا پایان یک متد توسط Garbage Collector تضمین نمیشود.
با وجود این زمانی که یک برنامه زمانی که در حالت debug اجرا شده باشد و یا ویژگی System.Diagnostics.DebuggableAttribute به اسمبلی برنامه اضافه شده باشد و یا اینکه پارامتر isJITOptimizeDisabled با مقدار true در constructor برنامه تنظیم شده باشد، کامپایلر JIT طول عمر تمام متغیر ها را، چه از نوع ارجاعی و چه از نوع مقدار، تا پایان محدوده شان افزایش میدهد که معمولا همان پایان تابع است(کامپایلر C# مایکروسافت یک سوییچ خط فرمان به نام /debug را ارائه میدهد که باعث اضافه شدن DebuggableAttribute به اسمبلی اضافه میشود و نیز پارامتر isJITOptimizeDisabled را نیز true میکند). این افزایش طول عمر از جمع آوری شدن متغیر ها توسط Garbage Collector در محدوده اجرایی آنها در طول برنامه جلوگیری میکند و این عمل فقط در زمان debug یک برنامه مفید واقع میشود.

زمانی که Garbage Collector شروع به کار میکند، فرض میکند که تمام اشیای موجود در heap زباله هستند. به عبارت دیگر فرض میکند که هیچ کدام از rootهای برنامه به هیچ شیئ در heap اشاره نمیکند. سپس Garbage Collector شروع به حرکت در میان rootهای برنامه میکند و یک گراف از تمام rootهای قابل دسترسی تشکیل میدهد. برای مثال Garbage Collector ممکن است یک متغیر عمومی را که به یک شیئ در heap اشاره میکند موقعیت یابی کند. شکل زیر یک heap را با چندین شیئ تخصیص داده شده نشان میدهد. همانطور که در شکل مشخص است rootهای برنامه فقط به اشیای A و C و D و F به طور مستقیم اشاره میکنند. بنابراین تمام این اشیا از اعضای گراف محسوب میشوند. زمان اضافه کردن شیئ D، Garbage Collector متوجه میشود که این شیئ به شیئ H اشاره میکند، بنابراین شیئ H نیز به گراف برنامه اضافه میشود و به همین ترتیب Garbage Collector تمام اشیای قابل دسترسی در heap را مشخص میکند.

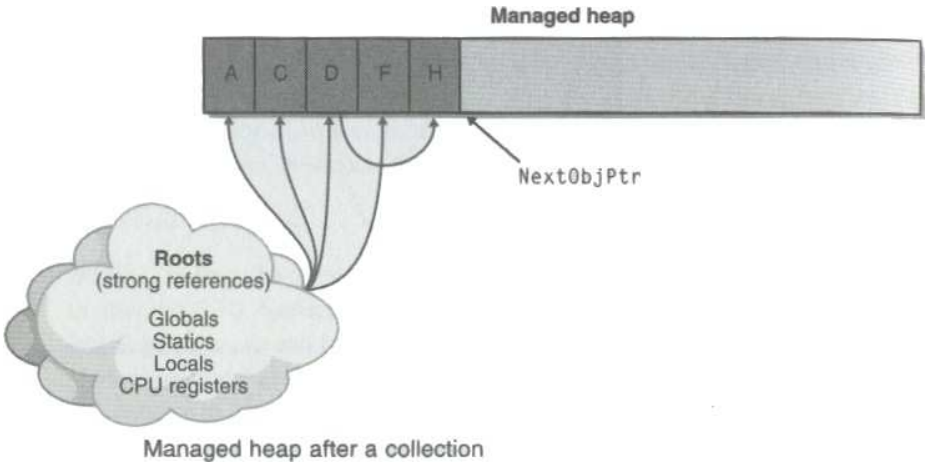


Managed heap before a collection

زمانی که این بخش از گراف کامل شد Garbage Collector، rootهای بعدی را چک میکند و مجددا اشیا را بررسی میکند. در طول اینکه Garbage Collector از یک شیئ به یک شیئ دیگر منتقل میشود، اگر سعی کند که یک شیئ تکراری را به گراف اضافه کند، Garbage Collector حرکت در آن مسیر را متوقف میکند. این نوع رفتار دو هدف را دنبال میکند: اول اینکه چون Garbage Collector از هیچ شیئ دو بار عبور نمیکند راندمان برنامه را به شکل قابل توجهی افزایش میدهد. دوم اینکه هر قدر هم در برنامه لیست های پیوندی دایره ای از اشیا موجود باشند ، Garbage Collector در حلقه های بینهایت نمی ماند.

زمانی که تمام rootها بررسی شدند، گراف Garbage Collector محتوی تمام اشیایی است که به نحوی از طریق rootهای برنامه قابل دسترسی می باشند و هر شیئ که در گراف نباشد به این معنی است که توسط برنامه قابل دسترسی نیست و یک زباله محسوب میشود. بعد از این Garbage Collector به صورت خطی heap را طی میکند و دنبال بلاک های پیوسته از زباله های مشخص شده توسط گراف میگردد (که هم اکنون فضای خالی محسوب میشوند). اگر بلاک های کوچکی پیدا شوند Garbage Collector این بلاک ها را به همان حال قبلی رها میکند.

اگر یک بلاک پیوسته وسیع توسط Garbage Collector یافت شد، در این حال، Garbage Collector اشیای غیر زباله را به سمت پایین حافظه heap شیفت میدهد (و این کار با تابع استاندارد memcopy انجام میشود) و به این طریق heap را فشرده میکند. طبیعتاً حرکت دادن اشیا به سمت پایین در heap باعث نا معتبر شدن تمام اشاره گرهای موجود برای آن اشیا میشود. به علاوه، اگر شیئ محتوی اشاره گری به شیئ دیگری بود Garbage Collector مسئول تصحیح این اشاره گرها میشود. بعد از این که این عمل فشرده سازی روی heap انجام شد NextObjPtr به آخرین شیئ غیر زباله اشاره میکند. شکل زیر یک managed heap را بعد از اتمام کار Garbage Collector نشان میدهد.



Managed heap after a collection

همانطور که در شکل می بینید،Garbage Collector یک افزایش بازدهی قابل توجهی را ایجاد میکند. اما به یاد داشته باشید زمانی Garbage Collector شروع به کار میکند که نسل صفر کامل شود و تا آن زمان managed heap به صورت قابل توجهی سریعتر از heap زمان اجرای C است. در نهایت Garbage Collector مربوط به CLR روش بهینه سازی را ارائه می دهد که راندمان کاری Garbage Collector را مقدار زیادی افزایش می دهد.

کد زیر نشان میدهد که چگونه به اشیا حافظه تخصیص داده میشود و آنها مدیریت میشوند:

```
Class App
{
	static void Main()
	{
		// ArrayList object created in heap, a is now a root
		ArrayList a = new ArrayList();

		// Create 10000 objects in the heap
		for(Int32 x=0;x<10000;x++
		{
			a.Add(new Object()); // Object created in heap
		}

		//Right now, a is a root (on the thread's stack). So a is
		// reachable and the 10000 objects it refers to
		// are reachable.
		Console.WriteLine(a.Length);

		//After a.Length returns, a isn't referred to in the code
		//andis no longer a root. If another thread were to start
		//a garbage collection before the result of a.Length were
		//passed to WriteLine, the 10001 objects would have their
		// memory reclaimed.

		Console.WriteLine("End Of Method");
	}
}
```

نکته: اگر فکر میکنید که Garbage Collector یک تکنولوژی با ارزش محسوب میشود، ممکن است تعجب کنید که چرا در ANSI C++ قرار نمی گیرد. دلیل این مورد این است که Garbage Collector احتیاج دارد که

rootهای موجود در برنامه را تعیین هویت کند و نیز باید بتواند تمام اشاره گرهای اشیا را پیدا کند. مشکل با C++ مدیریت نشده این است که این برنامه تغییر نوع یک اشاره گر را از یک نوع به یک نوع دیگر مجاز میداند و هیچ راهی برای فهمیدن این که این اشاره گر به چه چیز اشاره میکند وجود ندارد. در CLR، managed heap همیشه میداند که نوع واقعی یک شیئ چیست و از اطلاعات metadata برای مشخص کردن اینکه کدام عضوها از یک شی به اشیای دیگر اشاره می کنند استفاده می کند.

ارجاع های ضعیف:

زمانی که یک root به یک شیئ اشاره میکند شیئ نمیتواند توسط Garbage Collector جمع آوری شود چون ممکن است برنامه به آن دسترسی پیدا کند. زمانی که یک root به یک شیئ اشاره میکند، اصطلاحا گفته میشود که یک ارجاع قوی به آن شیئ موجود است. با این وجود Garbage Collector همچنین از ارجاعات ضعیف هم پشتیبانی میکند. ارجاعات ضعیف به Garbage Collector اجازه می دهند که شیئ را جمع آوری کند و همچنین به برنامه اجازه دهد که به آن شیئ دسترسی داشته باشد.
اگر فقط ارجاعات ضعیف به یک شیئ موجود باشند و Garbage Collector آغاز به کار کند آن شیئ از heap پاک میشود و سپس زمانی که برنامه سعی کند به آن دسترسی پیدا کند این عمل با شکست مواجه میشود. به عبارت دیگر برای دسترسی به یک ارجاع ضعیف برنامه باید یک ارجاع قوی به آن داشته باشد. اگر برنامه قبل از اینکه Garbage Collector آن را جمع آوری کند یک ارجاع قوی به آن شیئ تشکیل دهد، بنابراین Garbage Collector نمی تواند این شیئ را حذف کند چون یک ارجاع قوی به آن شیئ موجود است.
اجازه دهید معنی واقعی موضوع را در کد بررسی کنیم:

```
Void SomeMethod()
{
      // Create a string reference to a new Object.
      Object o = new Object();

      // Create a strong reference to a short WeakReference object.
      // The WeakReference object tracks the Object's lifetime.
      WeakReference wr = new WeakReference(o);

      o = null; // Remove the strong reference to the object.

      O = wr.Target;
      if(o==null)
      {
            // A garbage collection occurred and Object's memory was
            // reclaimed.
      }
      else
      {
      //A garbage collection didn't occur and I can successfully
            // access the Object using o.
      }
}
```

ممکن است این سوال ایجاد شود که استفاده از ارجاعات ضعیف چه مزیتی را ایجاد میکند؟ خوب بعضی از ساختار داده ها به سادگی ایجاد میشوند اما مقدار زیادی حافظه را اشغال می کنند. برای مثال ممکن است شما برنامه ای داشته باشید که به لیستی از تمام

دایرکتوری ها و فایلهای موجود در کامپیوتر نیاز داشته باشد. شما به سادگی می توانید یک درخت از تمام این اطلاعات تشکیل دهید و به محض اینکه برنامه شما اجرا شد به جای مراجعه به هارد به این درخت در حافظه مراجعه کند این امر سرعت برنامه شما را افزایش میدهد.
اما مشکل این است که این ساختار داده حجم بسیار زیادی را از حافظه اشغال می کند. اگر کاربر بخش دیگری از برنامه را آغاز کند، این درخت ممکن است دیگر مورد نیاز نباشد اما مقدار زیادی از حافظه را اشغال کرده است. شما می توانید ارجاع اصلی به این درخت در حافظه را رها کنید. اما اگر کاربر به قسمت اول برنامه شما برگشت، شما مجددا نیاز به بازسازی این درخت خواهید داشت. ارجاعات ضعیف این امکان ار به شما میدهد که به بهترین نحو این موقعیت را کنترل کنید.
زمانی که کاربر از قسمت اول برنامه شما خارج شود، شما می توانید یک ارجاع ضعیف به `root` مربوط به این درخت در حافظه ایجاد کنید و تمام ارجاعات قوی به آن را از بین ببرید. اگر حافظه برای بخشهای دیگر برنامه کم شد، `Garbage Collector` حافظه گرفته شده توسط درخت مذکور را می گیرد و زمانی که کاربر مجددا به قسمت اول برنامه برگشت و برنامه سعی کرد از ارجاع ضعیف یک ارجاع قوی بدست آورد، این تلاش شکست می خورد و برنامه مجددا درخت را تشکیل میدهد. در غیر این صورت برنامه از همان درخت قبلی موجود در حافظه استفاده می کند.
نوع داده `System.WeakReference` دو `Constructor` عمومی را ارائه میدهد:

```
public WeakReference(Object target);
public WeakReference(Object target, Boolean trackResurrection);
```

پارامتر `target` شیئ را که `WeakReference` باید نگه داری کند مشخص می کند. پارامتر `trackResurrection` مشخص می کند که آیا `WeakReference` باید شیئ را حتی بعد از `finalize` شدن هم نگه داری کند یا خیر که معمولا مقدار این پارامتر `false` است.
برای راحتی یک ارجاع ضعیف را که بعد از احیای شیئ هم آن را نگه داری میکند ارجاع ضعیف بلند و ارجاع ضعیفی را که بعد از احیای شیئ آن را نگه داری نمی کند را ارجاع ضعیف کوتاه می نامیم. اگر نوع داده شیئ تابع `finalize` را ارئه ندهد ارجاع ضعیف کوتاه و بلند مانند هم عمل می کنند. اما به شدت توصیه می شود که از به کار بردن ارجاعات ضعیف خودداری کنید، زیرا این نوع ارجاعات به شما اجازه میدهند که یک شیئ را حتی بعد از `finalize` شدن هم احیا کنید که موجب به وجود آمدن یک موقعیت غیر قابل پیش بینی برای شیئ می شود.
یک بار که شما یک ارجاع ضعیف را برای یک شیئ ایجاد کردید عموما باید تمام ارجاعات قوی به آن را برابر `null` قرار دهید، در غیر این صورت `Garbage Collector` توانایی جمع آوری این شیئ را در مواقع ضروری نخواهد داشت.
برای استفاده مجدد شیئ شما باید ارجاع ضعیف را به یک ارجاع قوی تبدیل کنید. برای این کار شما می توانید از `WeakReference.Target` استفاده کنید و این شیئ را به یکی از `root` های برنامه اختصاص دهید. اگر این عبارت مقدار `null` را برگرداند، یعنی شیئ مذکور توسط `Garbage Collector` جمع آوری شده است، در غیر این صورت `root` حاوی آدرس یک ارجاع قوی به این مقدار محسوب می شود و برنامه از این طریق می تواند شیئ مذکور را کنترل کند.

نسلها[1]:

همانطور که پیشتر ذکر شد نسلها مکانیسمی درون CLR Garbage Collector به شمار میرود که هدف اصلی آن بهبود کارایی برنامه است. یک Garbage Collector که با مکانیسم نسلها کار میکند (همچنین به عنوان Garbage Collector زودگذر[2] هم نامیده میشود) فرضهای زیر را برای کار خود در نظر میگیرد:

۱) هر چه یک شیئ جدیدتر ایجاد شده باشد طول عمر کوتاهتری هم خواهد داشت.
۲) هر چه یک شیئ قدیمیتر باشد طول عمر بلندتری هم خواهد داشت.
۳) جمع آوری قسمتی از heap سریعتر از جمع آوری کل آن است.

مطالعات زیادی معتبر بودن این فرضیات را برای مجموعه بزرگی از برنامه های موجود تایید کرده اند و این فرضیات بر طرز پیاده سازی Garbage Collector تاثیر داشته اند. در این قسمت طرز کار این مکانیسم شرح داده شده است.
زمانی که یک managed heap برای بار اول ایجاد میشود دارای هیچ شیئ نیست. اشیایی که به heap اضافه شوند در نسل صفر قرار میگیرند. اشیای موجود در نسل صفر اشیای تازه ایجاد شده ای هستند که تا کنون توسط Garbage Collector بررسی نشده اند. تصویر زیر یک برنامه را که تازه آغاز به کار کرده است نشان میدهد که دارای پنج شیئ است(از A تا E). بعد از مدتی اشیای C و E غیر قابل دسترسی میشوند.



A newly initialized heap containing some objects; all in generation 0. No collections have occurred yet.

زمانی که CLR آغاز به کار میکند یک مقدار نهایی را برای نسل صفر در نظر میگیرد که به طور پیش فرض ۲۵۶ کیلو بایت است(البته این مقدار مورد تغییر قرار میگیرد). بنابراین اگر شیئ بخواهد ایجاد شود و در نسل صفر فضای کافی وجود نداشته باشد Garbage Collector آغز به کار میکند. اجازه دهید تصور کنیم که اشیای A تا E ۲۵۶ کیلو بایت فضا اشغال کرده اند زمانی که شیئ F بخواهد تشکیل شود Garbage Collector باید آغاز به کار کند. Garbage Collector تشخیص میدهد که اشیایC و E زباله محسوب میشوند و بنابراین شیئ D باید فشرده شود بنابراین این شیئ به کنار شیئ B میرود. اشیایی که بعد از این مرحله باقی میمانند (اشیای A و B و D) وارد نسل یک میشوند. اشیای موجود در نسل یک به این معنی هستند که یک بار توسط Garbage Collector بررسی شده اند. Heap برنامه مفروض بعد از اولین مرحله به صورت تصویر زیر در می آیند.

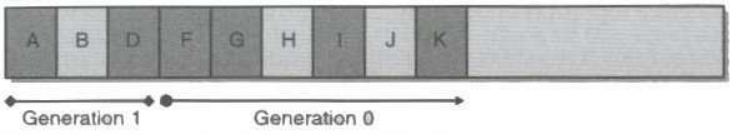
---

[1] Generations
[2] Ephemeral Garbage Collector



After one collection: generation 0 survivors are promoted to generation 1; generation 0 is empty.

بعد از یک بار اجرای Garbage Collector هیچ شیئ در نسل صفر باقی نمی ماند. مثل همیشه اشیایی که بعد از این ایجاد می شوند به نسل صفر اضافه میشوند. شکل زیر اجرای برنامه و به وجود آمدن اشیای F تا K را نشان میدهد. به علاوه در طول اجرای برنامه اشیایB و H و J غیر قابل استفاده شده اند و حافظه گرفته شده توسط آنها باید آزاد شود.



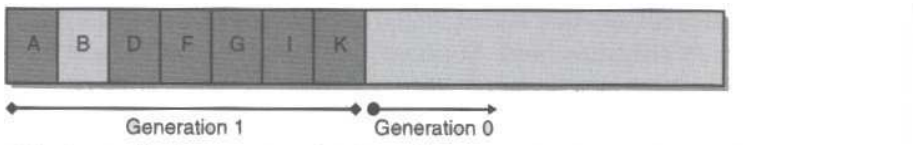
New objects are allocated in generation 0;

حال فرض کنیم با تخصیص حافظه برای شیL در نسل صفر مقدار داده های موجود در این نسل از ۲۵۶کیلوبایت فراتر رود. چون نسل صفر به سر حد خود رسیده است Garbage Collector باید آغاز به کار کند. زمانی که عمل Garbage Collection آغاز میشود Garbage Collector باید تصمیم بگیرد که کدام نسل باید مورد بررسی قرار گیرد. پیشتر ذکر شد که زمانی که CLR آغاز به کار میکند برای نسل صفر ۲۵۶ کیلو بایت فضا اختصاص میدهد. همچنین CLR یک سر حد نیز برای نسل یک در نظر میگیرد. فرض میکنیم این مقدار فضا شامل ۲ مگا بایت باشد.

زمانی که عمل Garbage Collection انجام میشود، Garbage Collector همچنین بررسی میکند که چه مقدار فضا توسط نسل یک اشغال شده است. در این حالت نسل یک فضایی کمتر از ۲ مگا بایت را اشغال کرده است بنابراین Garbage Collector فقط نسل صفر را مورد بررسی قرار میدهد. یک بار دیگر فرضیات Garbage Collector را که در ابتدا ذکر شد مرور کنید. اولین فرض این بود که اشیای تازه ایجاد شده دارای عمر کوتاه تری هستند. بنابراین این گونه به نظر میرسد که نسل صفر دارای بیشترین مقدار زباله باشد و جمع آوری حافظه از این نسل موجب آزاد سازی مقدار زیادی حافظه میشود. بنابراین Garbage Collector نسل یک را رها میکند و فقط به جمع آوری نسل صفر میپردازد که این عمل موجب افزایش سرعت کارکرد پروسه Garbage Collector میشود.

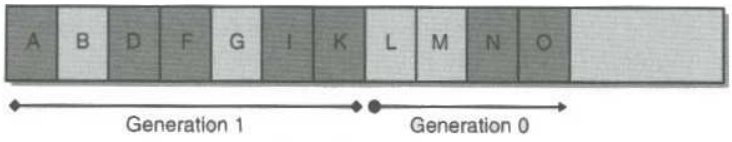
به وضوح، رها سازی اشیای نسل یک وجب افزایش سرعت و کارایی Garbage Collector میشود. با وجود این کارایی Garbage Collector بیشتر افزایش پیدا میکند چون تمام اشیای موجود در Managed Heap را بررسی نمیکند. اگر یک root یا یک شیئ از این نسل به شیئ از نسل قدیمی تر اشاره کند، Garbage Collector میتواند ارجاعات داخلی اشیای قدیمی تر را در نظر نگیرد، و بدین وسیله زمان مورد نیاز را برای تشکیل گرافی از اشیای قابل دسترس کاهش میدهد. البته این امر ممکن است که یک شیئ قدیمی به یک شیئ جدید اشاره کند. برای اطمینان از این که این شیئ قدیمی نیز مورد بررسی قرار میگیرد Garbage Collector از یک مکانیسم داخلی JIT استفاده میکند به این نحو که زمانی که یک فیلد ارجاع یک شیئ تغییر کرد یک بیت را تنظیم میکند. این پشتیبانی توسط JIT باعث میشود که Garbage Collector بتواند تشخیص دهد کدام از اشیای قدیمی از زمان آخرین عمل جمع آوری تا کنون تغییر کرده اند. فقط اشیای قدیمی که دارای فیلد های تغییر کرده هستند احتیاج به بررسی شدن برای اینکه آیا به شیئی از نسل صفر اشاره میکنند احتیاج دارند.

نکته: تستهای کارایی مایکروسافت نشان میدهند که عمل Garbage Collection در نسل صفر کمتر از یک میلی ثانیه در یک کامپیوتر پنتیوم با سرعت ۲۰۰ مگا هرتز زمان میبرد.
یک Garbage Collector که از نسلها استفاده میکند همچنین تصور میکند که اشیایی که مدت زیادی است که در حافظه مانده اند به زودی نیز از حافظه خارج نمیشوند. بنابراین این احتمال میرود که اشیای موجود در نسل یک همچنان در طول برنامه قابل دسترس خواهند بود. بنابراین اگر Garbage Collector اشیای موجود در نسل یک را بررسی کند احتمالا مقدار زیادی متغییر غیر قابل دسترسی در برنامه نخواهد یافت و احتمالا حافظه زیادی را آزاد نخواهد کرد. بنابراین آزاد سازی نسل یک چیزی جز اتلاف وقت نخواهد بود. اگر هر زباله ای در نسل یک به وجود بیاید در همان نسل باقی خواهد ماند. بعد از اجرای عملیات ذکر شده شکل heap به صورت زیر در می آیند.



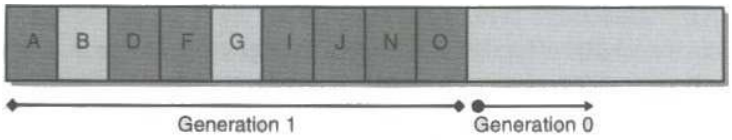
After two collections: generation 0 survivors are promoted to generation 1 (growing the size of generation 1); generation 0 is empty.

همانطور که میبینید تمام اشیای که از نسل صفر باقی مانده اند وارد نسل یک شده اند. چون Garbage Collector نسل یک را بررسی نمیکند شیئ B حافظه ای را که گرفته است آزاد نمیکند با وجود اینکه از آخرین عمل Garbage Collector تا کنون این متغییر در برنامه قابل استفاده نبوده است. مجددا بعد از جمع آوری نسل صفر دارای هیچ شیئ نخواهد بود و بنابراین مکانی برای قرارگیری اشیای جدید محسوب میشود. در ادامه برنامه به کار خود ادامه میدهد و اشیای L تا O را ایجاد میکند. و در حال اجرا برنامه استفاده از اشیای G و L و M را پایان میدهد و آنها را غیر قابل دسترس میکند. بنابراین heap به صورت زیر تبدیل میشود.



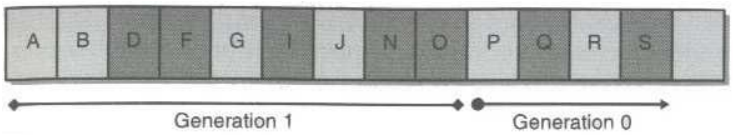
New objects are allocated in generation 0; generation 1 has more garbage.

اجازه دهید فکر کنیم که تخصیص حافظه برای شیئ P باعث تجاوز نسل صفر از سر حد خود شود و این عمل موجب اجرای مجدد Garbage Collector شود. چون تمام اشیای موجود در نسل یک کمتر از ۲ مگا بایت است Garbage Collector مجددا تصمیم میگیرد که فقط نسل صفر را بررسی کند و از اشیای غیر قابل دسترسی در نسل یک چشم پوشی کند(اشیای B و G). بعد از عمل جمع آوری heap به صورت زیر در می آید.



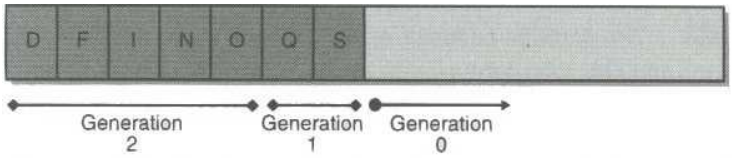
After three collections: generation 0 survivors are promoted to generation 1 (growing the size of generation 1 again); generation 0 is empty.

در تصویر بالا، مشاهده میکنید که نسل یک به مرور در حال رشد و افزایش حجم است. اجازه دهید تصور کنیم که اشیای موجـود در نسل یک تا سر حد ۲ مگا بایت فضای قابل استفاده در نسل یک را اشغال کرده اند. در ایـن مرحلـه، برنامـه مراحـل اجـرای خـود را همچنان ادامه میدهد و در این مرحله اشیای P تا S تولید میشوند که این اشیا نسل صفر را نیز تا سر حد خود پر میکنند. Heap در این مرحله مشابه شکل زیر میشود.



New objects are allocated in generation 0; generation 1 has more garbage.

زمانی که برنامه سعی در تخصیص حافظه برای شیئ T دارد نسل صفر کاملا پر است و Garbage Collector بایـد آغاز به کار کند. این مرتبه، اشیای موجود در نسل یک هر ۲ مگا بایت فضای خود را تا سر حد فضای نـسل یـک اشـغال کـرده انـد. علاوه بر اشیای موجود در نسل صفر، تصور میشود که بعضی از اشیای موجود در نسل یک هم به صورت غیر قابل اسـتفاده در آمـده اند. بنابراین این مرتبه، Garbage Collector تصمیم میگیرد که تمام اشیای موجود در نسل یک و نسل صفر را مورد بررسی قرار دهد. بعد از اینکه هر دو نسل به طـور کامـل توسـط Garbage Collector مـورد بررسـی قـرار گرفتنـد، heap به صورت شکل زیر در می آید.



After four collections: generation 1 survivors are promoted to generation 2, generation 0 survivors are promoted to generation 1, and generation 0 is empty.

مانند قبل، اشیایی که در این مرحله از جمع آوری از نسل صفر باقی ماندند وارد نسل یک میشوند و نیز اشیایی که در ایـن مرحلـه از نسل یک باقی ماندند وارد نسل دو میشوند. مثل همیشه، بلافاصله نسل صفر از اشیا خالی میـشود و اشـیای جدیـد میتواننـد در ایـن قسمت قرار گیرند. اشیای موجود در نسل دو اشیای هـستند کـه حـداقل دو بـار توسـط Garbage Collector مـورد بررسی قرار گرفته اند. ممکن است بعد از یک یا دو بار جمع آوری نسل صفر، با انجام این عمل در نـسل یـک مقـداری حافظـه آزاد شود، اما این عمل تا زمانی که نسل یک به سر حد خود نرسد انجام نمیشود که این کار ممکن است نیاز به چندین بـار اجـرای جمـع آوری در نسل صفر باشد.

Managed heap فقط سه نسل را پشتیبانی میکند: نسل صفر، نسل یک و نسل دو. بنابراین چیزی به نام نـسل سـه وجـود ندارد. زمانی که CLR آغاز به کار میکند، سر حد هایی را برای هر سه نسل در نظر میگیرد. همانطور که پیـشتر ذکـر شـد، سـر حـد برای نسل صفر حدود ۲۵۶کیلوبایت است، سر حد برای نسل یک حدوداً ۲ مگا بایت است و سر حد برای نسل دو حدود ۱۰ مگا بایت است. بنابراین سر حد نسلها به گونه ای انتخاب شده است که موجب افزایش بازدهی و راندمان برنامه شود. هرچه سر حد یـک نـسل بیشتر باشد عمل Garbage Collection کمتر روی آن نسل صورت میگیرد. و دوباره، بهبود کارایی به وجود می آید که به دلیل فرضیات اولیه است: اشیای جدید دارای طول عمر کوتاهتری هستند، اشیای قدیمی طول عمر بیشتری دارند.

Garbage Collector موجود در CLR یک جمع آوری کننده با تنظیم کننده خودکار است. این بدین معنا اسـت کـه Garbage Collector از رفتار برنامه شما می آموزد که چه زمانی باید عمل جمع آوری را انجام دهد. برای مثال اگـر

برنامه شما اشیای زیادی را ایجاد کند و از آنها برای مدت زمان کوتاهی استفاده کند، این امر ممکن است کـه آزاد سـازی حافظـه در نسل صفر مقدار زیادی حافظه را آزاد کند.حتی ممکن است تمام حافظه گرفته شده در نسل صفر آزاد شود.
اگر Garbage Collector مشاهده کند که بعد از انجام جمع آوری نسل یک تعـداد محـدودی از اشـیا بـاقی ماندنـد، ممکن است که تصمیم بگیرد که سر حد نسل صفر را از ۲۵۶ کیلو بایت به ۱۲۸ کیلو بایت کاهش دهد. این کاهش در فضای معـین به این معنی است که عمل جمع آوری باید در فواصل زمانی کوتاه تری رخ دهد اما فضای کمتری را بررسی کند. بنـابراین کارهـای پروسه شما به صورت قابل توجهی افزایش نمی یابد. اگر تمام اشیای موجود در نسل صفر زباله محـسوب شـوند دیگـر احتیـاجی بـه فـشرده سـازی حافظـه توسـط Garbage Collector نیـست. ایـن عمـل میتوانـد بـه سـادگی بـا آوردن اشـاره گـر NextObjPtr به ابتدای حافظه مورد نظر برای نسل صفر انجام شود. این عمل به سرعت حافظه را آزاد میکند!
نکته: Garbage Collector به بهترین نحو با برنامه های ASP.NET و سرویسهای وب مبتنـی بـر XML کـار میکند. برای برنامه های تحت ASP.NET، یک تقاضا از طرف کلاینت میرسد، یک جعبه از اشیای جدید تشکیل میشود، اشـیا کارهای تعیین شده توسط کلاینت را انجام میدهند، و نتیجه به سمت کلاینت بر میگردد. در این مرحله تمـام اشـیای موجـود بـرای انجام تقاضای کلاینت زباله تلقی میشوند. به بیان دیگر، هر تقاضای برنامه های تحت ASP.NET باعث ایجاد حجـم زیـادی از زباله میشوند. چون این اشیا اغلب بلافاصله بعد از ایجاد دیگر قابل دسترسی نیستند هر عمل جمـع آوری موجـب آزاد سـازی مقـدار زیـادی از حافظـه میـشود. ایـن کـار مجموعـه کارهـای پروسـه را بـسیار کـاهش میدهـد بنـابراین رانـدمان Garbage Collector محسوس خواهد بود.
به بیان دیگر، اگر Garbage Collector نسل صفر را مورد بررسی قرار دهد و مشاهده کند که مقـدار زیـادی از اشـیا وارد نسل یک شدند، مقدار زیادی از حافظه توسط Garbage Collection آزاد نمیـشود، بنـابراین Garbage Collector سر حد نسل صفر را تا ۵۱۲ کیلو بایت افزایش میدهد. در این مرحله کمتر انجام میشود اما با هر بار انجـام ایـن عمل مقدار زیادی حافظه آزاد میشود.
در طول این قسمت چگونگی تغییر دینامیک سر حد نسل صفر شرح داده شد. اما علاوه بر سر حد نسل صفر سر حد نـسلهای یـک و دو نیز بر اساس همین الگوریتم تغییر میکننـد. بـه ایـن معنـی کـه زمـانی کـه ایـن نـسلها مـورد عمـل جمـع آوری قـرار میگیرنـد Garbage Collector بررسی میکند که چه مقدار فضا آزاد شده است و چه مقدار از اشیا به نـسل بعـد رفتـه انـد. بـر اساس نتایج این بررسیها Garbage Collector ممکن است ظرفیت این نسلها را کاهش یا افزایش دهـد کـه باعـث افزایش سرعت اجرای برنامه میشود.

دیگر نتایج کارایی Garbage Collector.

پیشتر در این مقاله الگوریتم کار Garbage Collector شرح داده شد. با این وجود در طول این توضیحات یک فرض بزرگ صورت گرفته بود: اینکه فقط یک ترد[1] در حال اجـرا اسـت. امـا در مـدل واقعـی چنـدین تـرد بـه managed heap دسترسی دارند و یا حداقل اشیای قرار گرفته در managed heap رو تغییر میدهند. زمانی که یک ترد موجب اجـرای عمـل جمع آوری توسط Garbage Collector میشود، دیگر ترد ها حق دسترسی بـه اشـیای موجـود در managed heap را ندارند(این مورد شامل ارجاع های اشیای موجود در stack هـم میـشود) زیـرا Garbage Collector ممکن است مکان این اشیا را تغییر دهد.

[1] Thread

بنابراین وقتی Garbage Collector بخواهد عمل جمع آوری را آغاز کند، تمام ترد هایی که در حال اجرای کد های مدیریت شده[1] هستند به حال تعلیق در می آیند.CLR دارای چندین مکانیسم نسبتاً متفاوت است که میتواند ترد ها را به حالت تعلیق در آورد بنابراین عمل جمع آوری میتواند به درستی اجرا شود. دلیل اینکه CLR از چندین مکانیسم استفاده میکند به حالت اجرا نگاه داشتن ترد ها تا حداکثر زمان ممکن و کاهش سربار کردن آنها در حافظه تا حداقل زمان ممکن است. تشریح این مکانیسم ها از اهداف این مقاله خارج است اما تا این حد لازم است ذکر شود که مایکروسافت فعالیتهای زیادی را برای کاهش فشار پردازشی ناشی از Garbage Collector انجام داده است. و نیز این مکانیسم ها به سرعت در حال تغییر هستند تا به بهترین کارایی خود برسند.

زمانی که CLR میخواهد Garbage Collector را اجرا کند، ابتدا تمام ترد ها در پروسه جاری را که در حال اجرای کد های مدیریت شده هستند به حال تعلیق در می آورد. سپس CLR برای تعیین موقعیت هر ترد تمام اشاره گرهای دستورات در حال اجرا توسط ترد ها را بررسی میکند. سپس برای تعیین اینکه چه کدی توسط ترد در حال اجرا بوده آدرس اشاره گر دستور با جدول ایجاد شده توسط کامپایلرJIT مقایسه میشود.

اگر دستور درحال اجرا توسط ترد در یک آفست مشخص شده به وسیله جدول مذکور باشد گفته میشود که ترد به یک نقطه امن دسترسی دارد. یک نقطه امن نقطه ای است که در آنجا میتوان بدون هیچ مشکلی ترد را به حال تعلیق در آورد تا Garbage Collector کار خود را آغاز کند.اگر اشاره گر دستور در حال اجرای ترد در روی یک آفست مشخص شده توسط جدول درونی تابع قرار نداشت، بنابراین ترد در یک نقطه امن قرار ندارد و CLR نمیتواند Garbage Collector را اجرا کند. در این حالتCLR ترد را هایجک[2] میکند: به این معنی که CLR استک مربوط به ترد را به گونه ای تغییر میدهد که آدرس بازگشت به یک تابع خاص پیاده سازی شده درون CLR اشاره کند. سپس ترد به ادامه کار خود بازمیگردد. زمانی که متد در حال اجرا توسط ترد ادامه پیدا کند، این تابع ویژه اجرا خواهد شد و ترد به حالت تعلق درخواهد آمد.

با وجود این ممکن است در بعضی مواقع ترد از متد خود باز نگردد. بنابراین زمانی که ترد به اجرای خود ادامه میدهد، CLR ۲۵۰میلی ثانیه صبر میکند. سپس دوباره بررسی میکند که آیا ترد به یک نقطه امن طبق جدول JIT رسیده است یا نه. اگر ترد به یک نقطه امن رسیده بود CLR ترد را به حالت تعلیق درمی آورد و Garbage Collector را اجرا میکند در غیر این صورت مجددا سعی میکند با تغییر Stack مربوط به ترد اجرای آن را به تابع دیگری انتقال دهد در صورت شکست مجددا CLR برای چند میلی ثانیه دیگر نیز صبر میکند. زمانی که تمام ترد ها به یک نقطه امن رسیدند یا اینکه با موفقیت هایجک شدند، Garbage Collector میتواند کار خود را آغاز کند. زمانی که عمل جمع آوری انجام شد تمام ترد ها به وضعیت قبلی خود بر میگردند و اجرای برنامه ادامه پیدا میکند. ترد های هایجک شده هم به متدهای اولیه خود بازمیگردند.

نکته: این الگوریتم یک پیچ خوردگی کوچک دارد. اگر CLR یک ترد را به حالت تعویق درآورد و دریابد که ترد در حال اجرای یک کد مدیریت نشده[3] بود آدرس بازگشت ترد هایجک میشود و به ترد اجازه داده میشود که به اجرای خود ادامه دهد. با این وجود در این حالت به Garbage Collector اجازه داده میشود که اجرا شود در حالی که ترد مذکور در حال اجرا است. این مورد هیچ اشکالی را به وجود نمی آورد زیرا کد های مدیریت نشده به اشیای موجود در managed heap دسترسی ندارند تا زمانی که آن اشیا پین[4] شوند. یک شیئ پین شده شی است که Garbage Collector حق حرکت دادن آن را در managed heap ندارد. اگر تردی که در حال حاضر در حال اجرای یک کد مدیریت نشده بود، شروع به اجرای یک کد مدیریت شده کند، ترد هایجک میشود و به حالت تعلیق درمی آید تا زمانی که Garbage Collection به درستی به اتمام برسد.

---

[1] Managed Code
[2] Hijack
[3] Unmanaged Code
[4] Pin

علاوه بر مکانیسمهای ذکر شده(نسلها، نقاط امن، و هایجک کردن)، Garbage Collector از بعضی از مکانیسمهای اضافی دیگری نیز استفاده میکند که باعث افزایش بازدهی آن میشود.

اشیای بزرگ:

فقط یک نکته قابل ذکر دیگر که باعث افزایش سرعت و بازدهی بهتر میشود باقی مانده است. هر شیئ که ۸۵۰۰۰ بایت یا بیشتر فضای حافظه را اشغال کند یک شیئ بزرگ در نظر گرفته میشود. اشیای بزرگ در یک heap ویژه اشیای بزرگ قرار میگیرند. اشیای درون این heap مانند اشیای کوچک (که راجع به آنها صحبت شد) finalize و آزاد میشوند. با این وجود این اشیا هیچ وقت تحت فشرده سازی قرار نمی گیرند زیرا شیفت دادن ۸۵۰۰۰ بایت بلاک حافظه درون heap مقدار زیادی از زمان CPU را هدر میدهد.
اشیای بزرگ همواره به عنوان نسل دو در نظر گرفته میشوند، بنابراین این اشیا باید فقط برای منابعی که مدت زمان زیادی در حافظه می مانند ایجاد شوند. تخصیص اشیایی که دارای طول عمر کوتاه هستند در قسمت اشیای بزرگ باعث میشود که عمل جمع آوری نسل دو سریعتر انجام شود و این مورد نیز به بازدهی و کارایی برنامه صدمه وارد میکند.

منابع:

1) Wrox Press – Beginning Visual C# 2005
2) Worx Press – Beginnign Visual Basic .NET 2005
3) O'Reilly Press – C# Essentials Second Edition
4) Microsoft Press – Microsoft Visual C# .NET Step By Step
5) O'Reilly Press – Learning C#
6) William Pollock – The Book Of Visual Studio .NET
7) Micorosft Press – Applied Microsoft .NET Framework Programming