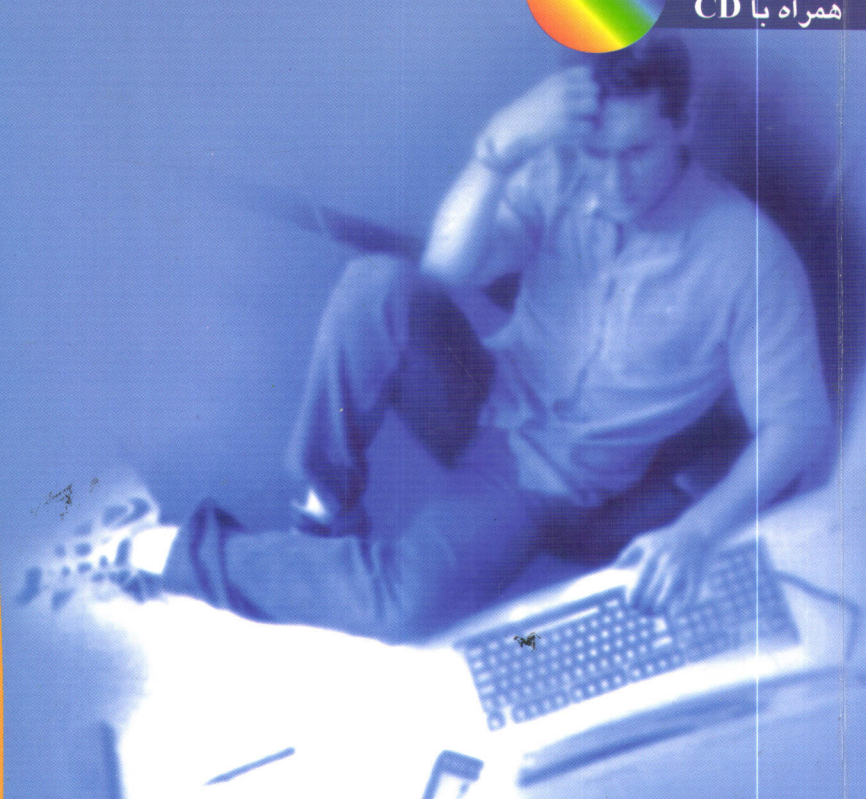
چاپ دوم

همراه با CD

ویژوال بیسیک .NET



آموزش گام به گام

مایکل هالورسن

علیرضا زارع پور

میکروسافت .net

همراه با CDROM

آموزش گام به گام

MICROSOFT
VISUAL BASIC .NET
STEP BY STEP

میکروسافت ویژوال بیسیک.NET

مایکل هالورسن

علیرضا زارع پور

هالورسون، مایکل، ۱۹۶۳ Halvorson, Michel
آموزش گام به گام ویژوال بیسیک .NET [ویژوال بیسیک دات نت]
مایکل هالورسون؛
[ترجمه] علیرضا زارع پور.
تهران: نص، ۱۳۸۳.
۴۶۴ ص،
ISBN: 964-5801-68-0 : CD با ۴۲۰۰۰ ریال
فهرستنویسی بر اساس اطلاعات فیپا.
عنوان اصلی: Microsoft Visual Basic .NET Step by step.
۱. ویژوال بیسیک میکروسافت - ۲. بیسیک (زبان برنامه نویسی کامپیوتر). ۳. چارچوب دات نت میکروسافت. الف. زارع پور، علیرضا، ۱۳۴۰- مترجم. ب. عنوان
QA ۷۶/۷۳/ب ۹ ۱۶۵هـ ۹ ب ۰۰۵/۲۷۶۸
محل نگهداری:
کتابخانه ملی ایران ۸۱-۳۶۶۲۸م

موسسه علمی فرهنگی

آموزش گام به گام ویژوال بیسیک .NET

مایکل هالورسون
علیرضا زارع پور
چاپ دوم: پاییز ۸۳
چاپ و صحافی: سازمان چاپ و انتشارات وزارت فرهنگ و ارشاد اسلامی
طراحی، آماده سازی: موسسه علمی فرهنگی «نص» قیمت با CD : ۴۲۰۰ تومان

تهران: میدان انقلاب، خیابان اردیبهشت، بن بست مبین، شماره ۲۳۷
تلفکس: ۶۹۵۳۸۸۳ - تلفن: ۴-۶۹۵۳۸۸۳ / ص.پ. ۱۳۱۴۵/۸۶۳

شابک: ۰-۶۸-۵۸۰۱-۹۶۴ ISBN:964-5801-68-0

فهرست مطالب

فصل ۳

فصل ۴

فصل ۵

فصل ۶

فصل ۷

فصل ۸

فصل ۹

فصل ۱۰

فصل ۱۱

فصل ۱۲

فصل ۱۳

فصل ۱۴

فصل ۱۵

فصل ۱۶

فصل ۱۷

فصل ۱۸

VISUAL BASIC .NET

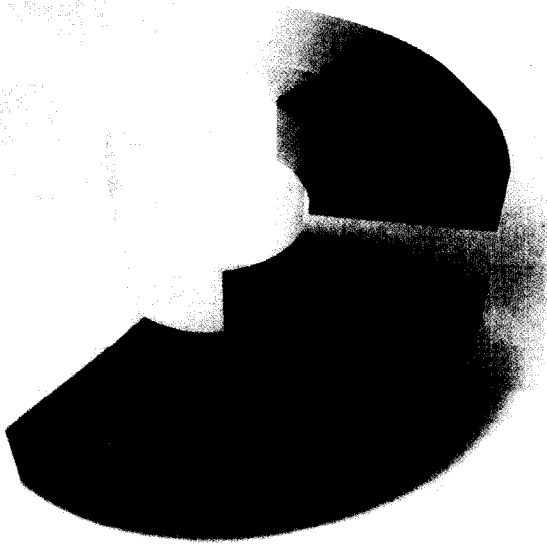
آموزش گام به گام

۱

بخش

آشنایی با

ویژوال بیسیک.NET



فصل

۱

اجرای یک برنامهٔ ویژوال بیسیک .NET

در این فصل یاد می گیرید چگونه :

- ✔ برنامهٔ ویژوال استودیو .NET را اجرا کنید.
- ✔ از محیط برنامه‌نویسی ویژوال استودیو استفاده کنید.
- ✔ یک برنامهٔ ویژوال بیسیک را باز کرده، و اجرا کنید.
- ✔ مقدار یک خاصیت را تغییر دهید.
- ✔ پنجره‌های ابزار را باز کرده، جابجا کنید، و ببندید.
- ✔ از سیستم راهنمای ویژوال استودیو استفاده کنید.

شاید اغراق نباشد اگر **ویژوال بیسیک .NET** (آنرا ویژوال بیسیک دات نت بخوانید)، یا بطور ساده VB.NET، را بزرگترین پیشرفت بعد از اختراع زبان ویژوال بیسیک بدانیم (برنامه‌ای که متجاوز از سه میلیون برنامه‌نویس حرفه‌ای در سرتاسر دنیا با آن برنامه می‌نویسند). در این فصل با مهارتهای اولیهٔ ویژوال استودیو .NET - یعنی طرز باز و اجرا کردن برنامه‌ها - آشنا خواهید شد. حتی اگر قبلاً با ویژوال بیسیک کار کرده‌اید، این فصل را نخوانده رد نکنید، چون ویژوال استودیو .NET تفاوتهای زیادی با سلف خود (ویژوال استودیو ۶) کرده است. ویژگیهای ویژوال استودیو .NET در زمینهٔ برنامه‌نویسی اینترنت و پایگاه داده بنحو جالب توجهی بهبود یافته است، و این مزایای زیادی برای برنامه‌نویسان حرفه‌ای دارد. حتی اگر از این مزایا استفاده نکنید، باز هم ویژوال استودیو .NET را جالب خواهید یافت: محیط برنامه‌نویسی آن برای تمام کامپایلرهای موجود (ویژوال بیسیک .NET، ویژوال سی++.NET و ویژوال سی #.NET - که سی‌شارپ خوانده می‌شود) یکسان است، و می‌توانید از مهارتهایی که بدست می‌آورید، در تمام آنها (حتی آنهایی که بعداً به بازار خواهند آمد، مانند ویژوال جی++.NET) براحتی استفاده کنید.

در این فصل یاد می‌گیرید چگونه ویژوال بیسیک .NET را اجرا کرده، یک برنامهٔ ساده را باز و اجرا کنید، و سپس از آن خارج شوید. در ضمن با تعدادی از پنجره‌های ویژوال استودیو .NET (و نحوهٔ کار با آنها) نیز آشنا خواهید شد.

تازه‌های ویژوال بیسیک .NET

اگر از ویژوال بیسیک ۶ به ویژوال بیسیک .NET آمده‌اید، میزان و وسعت تغییرات انجام شده شما را به تعجب خواهد انداخت، به همین دلیل در هر فصل بخشی را به تازه‌های آن اختصاص داده‌ام. قبل از هر چیز با تغییراتی که در محیط ویژوال استودیو .NET بعمل آمده، شروع می‌کنم:

- با اینکه ویژوال بیسیک .NET تفاوتهای خود را با زبانهای دیگر از قبیل ویژوال سی++.NET و ویژوال سی#.NET حفظ کرده، اما از نظر یکپارچگی محیط برنامه‌نویسی (محیط ویژوال استودیو .NET) بسیار به آنها نزدیک شده است.
- وقتی ویژوال استودیو .NET را اجرا می‌کنید، صفحهٔ شروع جدیدی خواهید دید. از طریق این صفحه می‌توانید به پروژه‌های قبلی خود، سایتهایی که با ویژوال استودیو ایجاد کرده‌اید، و یا اطلاعات پروفایل دسترسی داشته باشید.
- ابزارهای برنامه‌نویسی ویژوال استودیو .NET چند تغییر عمده کرده‌اند. پنجرهٔ پروژه به کاوشگر راه‌حل (Solution Explorer) تغییر نام داده، و پنجرهٔ کمک حساس به محتوای جدیدی بنام کمک دینامیک (Dynamic Help) اضافه شده است. جعبه ابزار نیز کمی تغییر کرده، و کنترلهای آن بر حسب عملکرد به چند دسته تقسیم شده‌اند.
- اکثر پنجره‌های ابزار بصورت خودکار - وقتی کاری با آنها ندارید - خود را مخفی می‌کنند، تا جلوی دست و پای شما را نگیرند.
- روش ذخیره کردن پروژه‌ها نیز عوض شده است: نامگذاری پروژه قبل از ایجاد آن صورت می‌گیرد، نه بعد از آن. هر پروژه تعداد زیادی فایل و پوشه به خود اختصاص می‌دهد - حتی بیشتر از ویژوال بیسیک ۶. در ویژوال بیسیک ۶ به برنامه‌هایی که از تلفیق چند پروژه بوجود می‌آمدند، گروه پروژه (Project Group) گفته می‌شد؛ در ویژوال استودیو .NET این قبیل برنامه‌ها با نام راه‌حل (Solution) شناخته می‌شوند.

محیط برنامه‌نویسی ویژوال استودیو .NET

ویژوال استودیو .NET یک محیط برنامه‌نویسی پُرقدرت و قابل‌انعطاف است، که به کمک آن می‌توانید بسرعت برای ویندوز برنامه بنویسید. بسیاری از ویژگیهای که در ویژوال استودیو .NET وجود دارد، به یکسان در ویژوال بیسیک .NET ، ویژوال سی++.NET و ویژوال سی#.NET قابل استفاده است.

اجرای ویژوال استودیو .NET

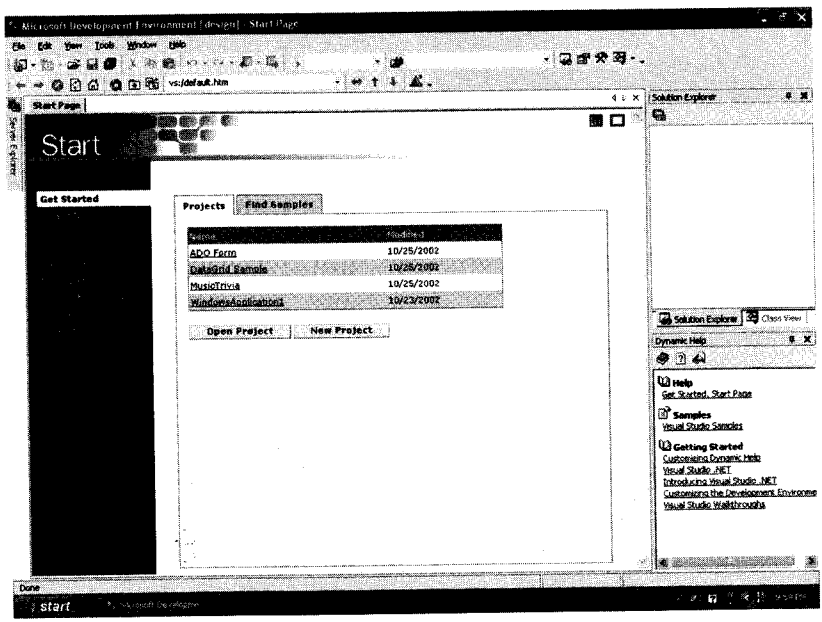
۱ منوی را Start|Programs|Microsoft Visual Studio .NET باز کنید.

توجه

ویژوال استودیو .NET دارای چهار ویرایش مختلف است: استاندارد، حرفه‌ای، توسعه‌دهنده، و

آرشیتکت. با هر یک از این ویرایشها می‌توانید کتاب حاضر را دنبال کنید - اما توجه داشته باشید که این کتاب برای یادگیری ویرایشهای قبلی ویژوال بیسیک مناسب نیست. اگر همچنان مایلید ویرایشهای قبلی ویژوال بیسیک (مانند ویژوال بیسیک ۶) را یاد بگیرید، می‌توانید به کتاب آموزش ویژوال بیسیک ۶ در ۲۱ روز - از همین انتشارات - مراجعه کنید.

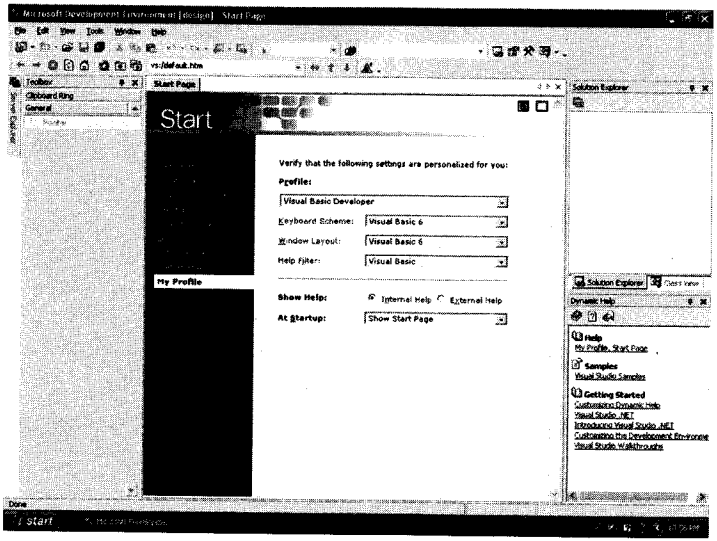
۲ روی آیکون برنامهٔ Microsoft Visual Studio .NET کلیک کنید؛ با این کار برنامهٔ ویژوال استودیو .NET اجرا می‌شود. اگر ویژوال استودیو را تازه نصب کرده‌اید، در شروع کار صفحه‌ای بنام صفحهٔ شروع (Start Page) خواهید دید (اگر این صفحه را نمی‌بینید، می‌توانید با فرمان Help|Show Start Page آنرا باز کنید).



۳ در سمت چپ صفحهٔ شروع، روی لینک My Profile کلیک کنید؛ با این کار صفحهٔ تنظیمات My Profile باز می‌شود. صفحهٔ My Profile اجازه می‌دهد تا ویژوال استودیو را با توجه به نیازهای خود پیکربندی کنید. یکی از تنظیمات این صفحه انتخاب نوع کامپایلریست که می‌خواهید با آن برنامه بنویسید (و البته واضحست که باید ویژوال بیسیک را انتخاب کنید!). بنابراین:

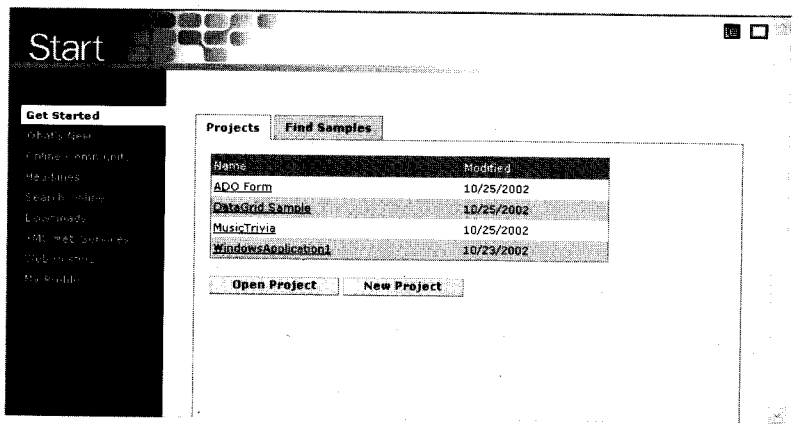
٤ از لیست Profile آیتم Visual Basic Developer را انتخاب کنید، تا ویژوال استودیو همه چیز را برای شروع برنامه‌نویسی با ویژوال بیسیک آماده کند.

۵ از لیست At Startup آیتم Show Start Page را انتخاب کنید، تا ویژوال استودیو همیشه در آغاز صفحهٔ شروع را نمایش دهد.



۶ سایر گزینه‌ها را بررسی کنید – و همچنین توجه کنید که چگونه ویژوال استودیو با هر انتخاب گزینه‌های متفاوتی در اختیار شما می‌گذارد. البته هر زمان که خواستید، می‌توانید به صفحهٔ My Profile برگردید، و تنظیمات آنرا تغییر دهید. بعد از نوشتن چند برنامهٔ ویژوال بیسیک با مفهوم این گزینه‌ها بیشتر آشنا خواهید شد.

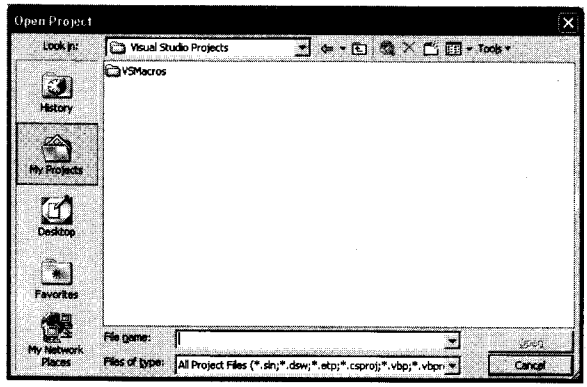
۷ بعد از آن که کارتان در صفحهٔ My Profile تمام شد، لینک Get Started را (در سمت چپ صفحهٔ شروع) کلیک کنید، تا مجدداً به این صفحه برگردید. صفحهٔ Get Started نقطهٔ شروع پروژه‌های جدید (و یا کار روی پروژه‌های موجود) است. در این صفحه می‌توانید یکی از پروژه‌هایی را که قبلاً ایجاد کرده‌اید، باز کنید (البته در اول کار طبعاً این لیست خالیست)؛ و یا پروژهٔ جدیدی را از ابتدا شروع کنید. برای هر یک از وظایف فوق یک دکمه در این صفحه خواهید دید: دکمهٔ Open Project برای باز کردن پروژه‌های موجود، و دکمهٔ New Project برای ایجاد پروژهٔ جدید.



متداولترین کاری که معمولاً برنامه‌نویسان در شروع کار با ویژوال استودیو انجام می‌دهند، باز کردن یک پروژهٔ موجود است؛ پس اجازه دهید ما هم چنین کنیم. در قسمت بعد پروژه‌ای بنام MusicTrivia را -که آنرا روی دیسک ضمیمهٔ کتاب خواهید یافت - باز می‌کنیم.

باز کردن یک پروژهٔ ویژوال بیسیک

۱ در قسمت Get Started صفحهٔ شروع، روی دکمهٔ Open Project کلیک کنید؛ با این کار پنجرهٔ Open Project (باز کردن پروژه) باز خواهد شد. حتی اگر قبلاً هرگز با ویژوال بیسیک کار نکرده باشید، پنجرهٔ Open Project برایتان ناآشنا نخواهد بود، چون این پنجره بسیار شبیه پنجره‌هائیست که در سایر برنامه‌های ویندوز (مانند Word یا Excel) دیده‌اید.



نکته

در سمت چپ پنجرهٔ Open Project تعدادی آیکون میانبُر می‌بینید؛ یکی از مفیدترین این آیکونها، آیکون My Projects است، که به محل ذخیره شدن پروژه‌های ویژوال استودیو اشاره می‌کند. ویژوال استودیو، بصورت پیش‌فرض، پروژه‌ها را در پوشه‌ای بنام Visual Studio Projects (که خود در پوشهٔ My Documents قرار دارد) ذخیره می‌کند.

۲ به محل قرار گرفتن پروژه‌های دیسک ضمیمه بروید؛ اگر برنامهٔ Setup را از روی دیسک ضمیمه اجرا کرده باشید، این محل پوشهٔ c:\vbnetsbs است.

۳ پوشهٔ chap01\musictrivia را باز کرده، و روی فایل پروژهٔ MusicTrivia دو-کلیک کنید (این فایل MusicTrivia.vbproj نام دارد). با این کار ویژوال بیسیک فرم و کُدهای برنامهٔ MusicTrivia را بار می‌کند، که برای دیدن فایلهای پروژه می‌توانید از کاوشگر راه‌حل (Solution Explorer - گوشهٔ راست-بالای فضای کاری ویژوال بیسیک) استفاده کنید.

نکته

اگر در پنجرهٔ Open Project پسوند نام فایلها را نمی‌بینید، به احتمال زیاد گزینهٔ Hide file extensions ویندوز را فعال کرده‌اید. برای غیرفعال کردن این گزینه، از منوی Tools کاوشگر ویندوز آیتم Folder Options را انتخاب کرده، و در برگهٔ View پنجرهٔ Fold Options ، علامت چک را از کنار گزینهٔ Hide file extensions for known file types بردارید.

پروژه‌ها و راه‌حل‌ها

به برنامه‌هایی که در ویژوال استودیو نوشته می‌شوند، پروژه یا راه‌حل گفته می‌شود، چون هر یک از آنها معمولاً از چندین فایل تشکیل می‌شوند. در ویژوال بیسیک .NET ، فایل پروژه با پسوند vbproj. و فایل راه‌حل با پسوند sln. مشخص می‌شود. هر پروژه معمولاً یک برنامهٔ مستقل است، در حالیکه یک راه‌حل می‌تواند از چندین پروژه تشکیل شده باشد. آنهایی که قبلاً با ویژوال بیسیک ۶ کار کرده‌اند، راه‌حل را با نام گروه پروژه (project group - با پسوند vbg.) می‌شناسند. اگر راه‌حلی فقط یک پروژه داشته باشد (و اکثر برنامه‌های این کتاب چنین هستند)، باز کردن پروژه معادل باز کردن راه‌حل نیز هست. اما اگر در یک راه‌حل چندین پروژه وجود داشته باشد، برای باز کردن همزمان تمام پروژه‌ها بایستی فایل راه‌حل را باز کنید.

ابزارهای ویژوال استودیو .NET

اگر قبلاً با ویژوال بیسیک برنامه نوشته باشید، با تعدادی از ابزارهای این محیط جدید آشنا هستید (ولی نه با همهٔ آنها). اینها در واقع ابزارهایی هستند که برای نوشتن یک برنامه (و مدیریت منابع سیستم) باید از آنها استفاده کنید. پس، اجازه دهید کمی بیشتر دربارهٔ آنها توضیح دهم.

در بالاترین قسمت پنجرهٔ ویژوال استودیو منو (Menu) قرار دارد؛ اکثر قریب به اتفاق فرمانهای ویژوال بیسیک را در این منو خواهید یافت. منو یکی از اصلی‌ترین اجزای برنامه‌های ویندوز است، و همانطور که می‌دانید با ماوس و کی‌بورد می‌توان از آن استفاده کرد. در زیر منو میله ابزار استاندارد (Standard Toolbar) ویژوال بیسیک را می‌بینید؛ میله ابزار مجموعه‌ایست از پُرکاربردترین فرمانهای برنامه، که برای دسترسی سریعتر در این قسمت قرار داده شده‌اند. اگر قبلاً با برنامه‌های Word و Excel کار کرده باشید، میله ابزار و کارکرد آنرا بخوبی می‌شناسید. برای کار کردن با اجزای میله ابزار باید از ماوس استفاده کنید.

در پائین‌ترین بخش از صفحهٔ مانیتور میلهٔ تکالیف ویندوز (Windows Taskbar) را می‌بینید؛ به کمک این میلهٔ تکالیف می‌توانید بین برنامه‌های مختلف سوئیچ کرده، و یا برنامهٔ دیگری را اجرا کنید.

برخی از ابزارها و پنجره‌های محیط ویژوال استودیو را در تصویر ذیل می‌بینید. اگر این تصویر با آنچه که در کامپیوتر خود می‌بینید، دقیقاً یکی نیست، نگران نشوید. در طی این فصل بتدریج با اجزای مختلف محیط برنامه‌نویسی ویژوال استودیو آشنا خواهید شد.

مهمترین اجزای محیط ویژوال استودیو عبارتند از: کاوشگر راه‌حل (Solution Explorer –که قبلاً کاوشگر پروژه نامیده می‌شد)، پنجرهٔ خواص (Properties)، کمک دینامیک (Dynamic Help)، طراح فرمهای ویندوز (Windows Forms Designer)، جعبه ابزار (Toolbox)، و پنجرهٔ خروجی (Output). البته شاید پنجره‌های دیگری، مانند کاوشگر میزبان (Server Explorer)، نمایش منابع (Resource View) یا نمایش کلاس (Class View)، را نیز ببینید، اما اینها معمولاً در گوشه و کنار ویژوال استودیو مخفی شده‌اند، و تا کاری با آنها نداشته باشید، خود را نشان نمی‌دهند. اگر هر یک از این پنجره‌ها (یا پنجره‌های دیگری که لازم دارید) را نمی‌بینید، منوی View را باز کرده، و پنجرهٔ موردنظر را انتخاب کنید. برای جلوگیری از شلوغی بیش از حد منوی View ، میکروسافت تعدادی از پنجره‌های کمتر مورد استفاده را در زیرمنویی بنام View|Other Windows قرار داده، که می‌توانید آنجا را هم نگاه کنید.

اندازه و محل هر یک از این پنجره‌ها را می‌توانید بدلخواه خود تنظیم کنید، و حتی می‌توانید آنها را بصورت دائم به سایر قسمتهای ویژوال استودیو بچسبانید (dock). سعی کنید فقط پنجره‌های مهمی را که برای کار خود بیشتر به آنها نیاز دارید، روی صفحه نگه دارید، و بقیه را مخفی کنید تا محیط کار را شلوغ نکنند. در این میان وضوح مانیتور اهمیت زیادی دارد، و می‌تواند بیشترین فضای کاری را در اختیارتان بگذارد؛ شاید بهترین وضوح برای کار با ویژوال استودیو ۱۰۲۴×۷۶۸ باشد.

با توجه به ابزارهای متعددی که در اختیار دارید، محیط کاری ویژوال استودیو بسرعت به سمت آشفتگی میل می‌کند. برای اجتناب از این وضعیت، میکروسافت راهکارهای متعددی در نظر گرفته است، که از میان آنها می‌توان به پنجره‌های چسبان (docking windows) و پنجره‌هایی که بطور خودکار مخفی می‌شوند (autohide) اشاره کرد. توصیه می‌کنم که در ابتدای کار فقط پنجره‌های مهم (مانند کاوشگر راه‌حل، پنجرهٔ

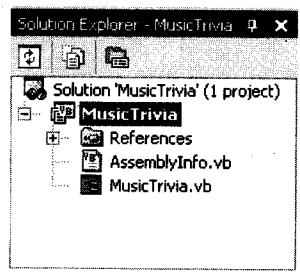
خواص، طراح فرمهای ویندوز، جعبه ابزار، کمک دینامیک و پنجرهٔ خروجی) را نگه دارید، و سایر پنجره‌ها را مخفی کنید. در قسمت بعد خواهید دید که چگونه می‌توان این پنجره‌ها را ظاهر یا مخفی کرد.

طراح فرمهای ویندوز

اگر تمرین قسمت قبل را دنبال کرده باشید، اکنون پروژهٔ MusicTrivia در ویژوال استودیو بار شده است، ولی احتمال دارد هنوز فرم برنامه را روی صفحه نبینید (هر پروژه می‌تواند چندین فرم – Form – داشته باشد؛ پروژهٔ MusicTrivia فقط یک فرم دارد). برای نمایش فرمهای یک پروژه باید از کاوشگر راه‌حل کمک بگیرید.

نمایش طراح فرمهای ویندوز

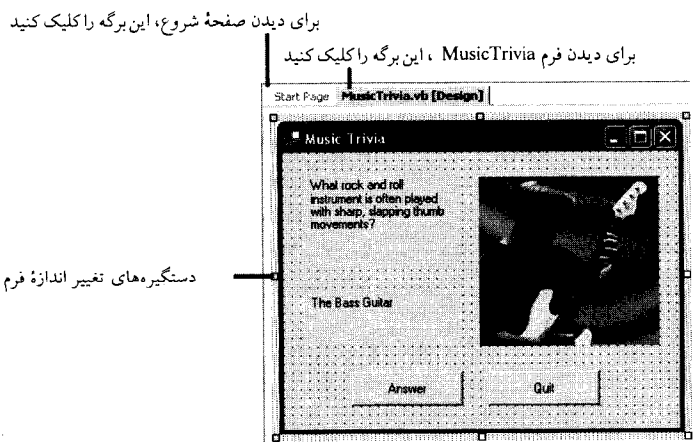
۱ کاوشگر راه‌حل در سمت راست (و متمایل به بالای) محیط کاری ویژوال استودیو قرار دارد. اگر پنجرهٔ کاوشگر راه‌حل را نمی‌بینید، از منوی View آیتم Solution Explorer را انتخاب کنید. کاوشگر راه‌حل، بعد از بار شدن برنامهٔ MusicTrivia ، مانند ذیل خواهد بود:



فرم

۲ در پنجرهٔ کاوشگر راه‌حل، روی فرم MusicTrivia.vb کلیک کنید. تمام فرمهای ویژوال بیسیک با یک آیکون کوچک مشخص می‌شوند، بگونه‌ایکه براحتی می‌توان آنها را تشخیص داد. وقتی فرمی را در کاوشگر راه‌حل انتخاب می‌کنید (روی آن کلیک می‌کنید)، مشخصات آن در پنجرهٔ خواص نشان داده می‌شود (البته اگر این پنجره باز باشد).

۳ در کاوشگر راه‌حل روی دکمهٔ View Designer کلیک کنید. با اینکار فرم MusicTrivia در طراح فرمهای ویندوز دیده خواهد شد:



توجه کنید که در طراح فرمهای ویندوز همچنان می‌توانید صفحهٔ شروع ویژوال استودیو را نیز ببینید. به کمک این ویژگی می‌توانید براحتی تنظیمات ویژوال استودیو را تغییر دهید، و یا پروژه‌های دیگری را باز کنید. برای بازگشت به فرم اصلی پروژه، کافیست روی برگهٔ MusicTrivia.vb [Design] کلیک کنید. قدم بعدی اجرای برنامه است.

نکته

اگر برگه‌های Start Page و MusicTrivia.vb [Design] را نمی‌بینید، محیط کاری ویژوال استودیو در حالت Tabbed Documents قرار ندارد. برای رفتن به این حالت، از منوی Tools آیتم Options را انتخاب کنید. در سمت چپ پنجره‌ای که باز می‌شود، پوشهٔ Environment را انتخاب کرده، و سپس روی برگهٔ General کلیک کنید. در قسمت Settings نیز دکمهٔ رادیویی Tabbed Documents را انتخاب کرده، و سپس OK را کلیک کنید. توجه داشته باشید که این حالت تا اجرای بعدی ویژوال استودیو خود را نشان نخواهد داد.

اجرای یک برنامهٔ ویژوال بیسیک

پروژهٔ MusicTrivia یک برنامهٔ ساده است، که صرفاً برای آشنا شدن شما با ابزارهای برنامه‌نویسی ویژوال استودیو نوشته شده است. روی فرم این برنامه پنج شیء (دو برچسب، یک تصویر، و دو دکمه) وجود دارد، و تمام کُد آن از سه خط تجاوز نمی‌کند! این کُد سؤال ساده‌ای از کاربر می‌پرسد، و سپس جواب آنرا می‌دهد. در حالت طراحی برنامه سؤال و جواب (هر دو) را می‌بینید، اما در هنگام اجرای برنامه جواب سؤال مخفی است، و فقط در موقع مناسب نمایش داده خواهد شد. در فصل‌های آینده با اشیاء ویژوال بیسیک و کُدنویسی آشنا خواهید شد؛ در حال حاضر فقط کافیست برنامه اجرا کنید.

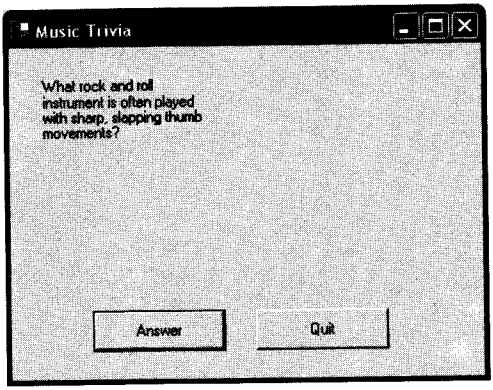
اجرای برنامهٔ MusicTrivia

۱ برای اجرای برنامهٔ MusicTrivia ، کافیست روی دکمهٔ Start در میله ابزار استاندارد کلیک کنید.

نکته

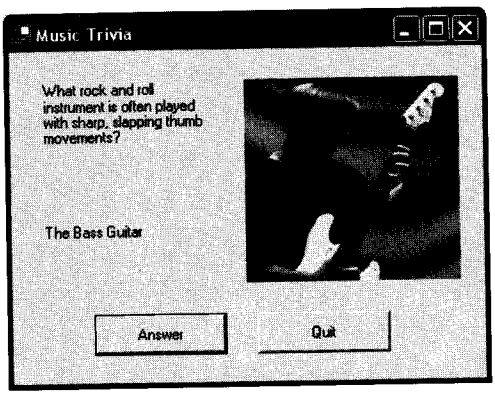
برای اجرای یک برنامه، می‌توانید کلید F5 را بزنید، و یا فرمان Start را از منوی Debug اجرا کنید. (در ویژوال بیسیک ۶ فرمان Start در منوی Run قرار داشت.)

ویژوال استودیو برنامه را به چیزی بنام اسمبلی (assembly – مجموعه‌ای از کُدها و داده‌های برنامه، و مشخصات آن) کامپایل کرده و سپس آنرا اجرا می‌کند. دقت کنید که آیکون برنامه در میلهٔ تکالیف ویندوز نیز ظاهر می‌شود. در حین کامپایل شدن برنامه، مراحل کار در پنجرهٔ خروجی نمایش داده می‌شود؛ خطاهای برنامه نیز در همین پنجره نشان داده خواهد شد، که می‌توانید بعداً نسبت به رفع آنها اقدام کنید. وقتی برنامهٔ MusicTrivia اجرا شود، پنجره‌ای شبیه ذیل خواهید دید:



سؤال مهمی که برنامه از شما می‌کند، چنین است: آن کدام ساز موسیقی است که با ضربات تند انگشتِ شست نواخته می‌شود؟

۲ برای گرفتن جواب سؤال، دکمهٔ Answer را کلیک کنید. با این عمل، برنامه پاسخ را - که گیتار باس است - بهمراه تصویری از یک نوازندهٔ گیتار نمایش می‌دهد. همانطور که ملاحظه می‌کنید، برنامهٔ ما بخوبی کار می‌کند.



۳ دکمهٔ Quit را کلیک کنید، تا برنامه بسته شده و ویژوال استودیو مجدداً فعال شود. توجه کنید که ظاهر فرم در این حالت با وقتی که در حال اجرا بود، تفاوت دارد - مخصوصاً به نقاط کوچکی که روی فرم تشکیل یک شبکه می‌دهند، دقت کنید. این شبکه به شما کمک می‌کند تا اشیاء روی فرم را با دقت بیشتری مرتب کنید (و در ضمن با یک نگاه می‌توانید بفهمید که برنامه در حال اجراست، یا در حال طراحی).

پنجرهٔ خواص

برای تغییر دادن مشخصات (یا خواص) فرم، و یا یکی از اشیاء روی آن، می‌توانید از پنجرهٔ خواص کمک بگیرید. خاصیت (property) عبارتست از یکی از کیفیات هر یک از اشیاء برنامه. برای مثال، سؤال برنامهٔ

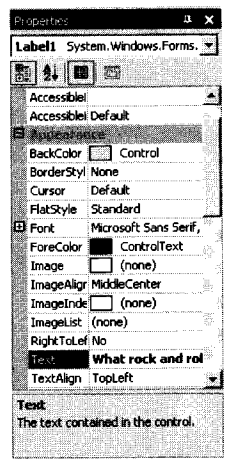
MusicTrivia را می‌توان با فونت، اندازه و یا رنگ دیگری نمایش داد. خواص یک شیء را می‌توان از طریق پنجرهٔ خواص، و یا در هنگام اجرای برنامه (از طریق کُد آن) تغییر داد.
در بالای پنجرهٔ خواص لیستی وجود دارد، که می‌توانید به کمک آن هر یک از اشیاء فرم فعال را انتخاب کرده، و خواص آنرا تنظیم کنید. (خواص اشیاء را می‌توان بصورت الفبایی و یا بصورت موضوعی در پنجرهٔ خواص لیست کرد.) در این قسمت می‌خواهیم خواص یکی از برچسب‌های برنامهٔ MusicTrivia را تغییر دهیم.

تغییر دادن یک خاصیت

۱ در فرم برنامهٔ MusicTrivia ، روی شیء Label1 (که سؤال برنامه را نشان می‌دهد) کلیک کنید. برای کار کردن روی یک شیء ابتدا باید آنرا انتخاب کنید. وقتی یک شیء را انتخاب می‌کنید (روی آن کلیک می‌کنید)، دستگیره‌های تغییر اندازه در اطراف آن ظاهر شده، و خواص آن در پنجرهٔ خواص نمایش داده می‌شود.

۲ در میله ابزار استاندارد ویژوال بیسیک، روی دکمهٔ پنجرهٔ خواص کلیک کنید. پنجرهٔ خواص معمولاً سمت راست فضای کاری ویژوال بیسیک، و زیر کاوشگر راه‌حل واقع شده است. (اگر آنرا نمی‌بینید، می‌توانید از طریق منوی View این پنجره را باز کنید.) در شکل زیر پنجرهٔ خواص شیء Label1 را می‌بینید:

مجموعه خواص فونت



در این پنجره تمام خواص شیء Label1 را (که مجموعاً ۴۵ خاصیت است) خواهید دید. در ستون سمت چپ نام خاصیت، و در ستون سمت راست هم مقدار فعلی هر خاصیت را می‌بینید. از آنجائیکه تعداد خواص هر شیء بسیار زیاد است، و برخی از این خواص بندرت به دستکاری نیاز دارند، ویژوال استودیو آنها را بصورت موضوعی دسته‌بندی می‌کند. آیتمهایی که در کنار آنها علامت + دیده می‌شود، در واقع مجموعه‌ای از چند خاصیت مرتبط به هم هستند، که می‌توانید با کلیک کردن این علامت خواص ذیل آنرا ببینید (و بالعکس، علامت - نشان می‌دهد که در حال دیدن خواص ذیل یک مجموعه هستید).

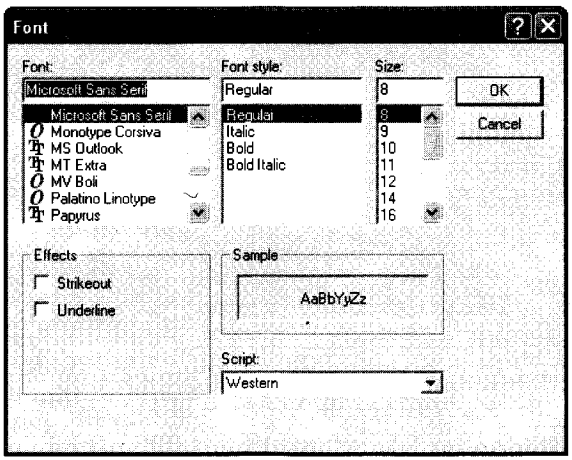
نکته

در بالای پنجرهٔ خواص دو دکمهٔ دیگر نیز می‌بینید، که می‌توانید به کمک آنها نحوهٔ نمایش خواص را بدلخواه خود تنظیم کنید. دکمهٔ «ترتیب الفبایی» (Alphabetic) خواص را بترتیب حروف الفبا مرتب می‌کند (و برای وقتی مناسب است که نام خاصیت موردنظر را بدانید)؛ دکمهٔ «ترتیب موضوعی» (Categorized) نیز خواص را بصورت موضوعی مرتب می‌کند (و برای مواقعی مناسب است که نوع خاصیت موردنظر را بشناسید). اگر برنامه‌نویسی با ویژوال بیسیک را تازه شروع کرده‌اید و هنوز با خواص اشیاء آشنایی کامل ندارید، توصیه می‌کنم از ترتیب موضوعی خواص استفاده کنید.

۳ در پنجرهٔ خواص آنقدر پائین بروید، تا به خاصیت Font (فونت) برسید.

٤ روی خاصیت Font (در ستون سمت چپ) کلیک کنید. با این کار نام فونت برچسب Label1 را (که در حال حاضر Microsoft Sans Serf است) در ستون سمت راست خواهید دید. در انتهای این ستون یک دکمهٔ کوچک می‌بینید، که علامت ... روی آن نقش بسته است (به این علامت ellipsis می‌گویند). علامت ... به ما می‌گوید که با کلیک شدن این دکمه، یک دیالوگ باز می‌شود، که از طریق آن می‌توانیم مقدار خاصیت را تنظیم کنیم.

۵ روی دکمهٔ ... خاصیت Font در پنجرهٔ خواص کلیک کنید - با این کار، ویژوال استودیو دیالوگ تنظیم فونت ویندوز را (که در بسیاری از برنامه‌های دیگر دیده‌اید) باز می‌کند.



۶ اندازهٔ فونت را (در لیست Size) از ۸ به ۱۰، و سبک آنرا (در لیست Font Style) از Regular به Italic تغییر داده، و دکمهٔ OK را کلیک کنید. همانطور که می‌بینید، ویژوال بیسیک بلافاصله تغییرات خواسته شده را روی برچسب Label1 اعمال می‌کند (اگر علامت + کنار خاصیت Font را کلیک کنید، خواص مختلف فونت - از قبیل Font Size و Font Style - را خواهید دید).

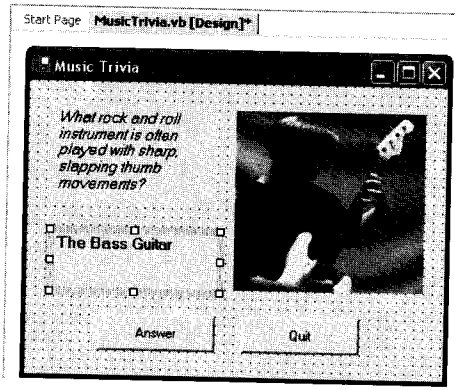
۷ سپس در طراح فرمهای ویندوز، روی برچسب Label2 (که پاسخ سؤال را نشان می‌دهد) کلیک کنید، تا انتخاب شود.

۸ مجدداً خاصیت Font را در پنجرهٔ خواص انتخاب کنید. همانطور که مشاهده می‌کنید، خواص فونت شیء Label2 کاملاً مستقل است، و هیچ ارتباطی با Label1 ندارد.

۹ روی دکمهٔ ... خاصیت Font کلیک کرده، و پس از ست کردن Font style به Bold ، دکمهٔ OK را کلیک کنید.

۱۰ در پنجرهٔ خواص، خاصیت ForeColor (رنگ پیش‌زمینه یا قلم) را پیدا کرده، و در ستون سمت چپ آن کلیک کنید.

۱۱ روی پیکان کوچک ستون سمت راست خاصیت ForeColor کلیک کرده، و در برگهٔ Custom رنگ ارغوانی تیره را انتخاب کنید. اکنون باید متن برچسب Label2 را (در طراح فرمهای ویندوز) با فونت ضخیم و رنگ ارغوانی تیره ببینید.



احسنت! اکنون یکی از مهارتهای مهم برنامه‌نویسی با ویژوال بیسیک - یعنی، ست کردن خواص اشیاء برنامه - را فراگرفته‌اید.

این خاصیت دیگر چیست؟!

در ویژوال بیسیک، هر عنصری که روی فرم برنامه قرار داده می‌شود، دارای خواص مخصوص به خود است، که می‌توان از طریق پنجرهٔ خواص، و یا کُد برنامه، آنها را ست کرد. خواص یک شیء در اطلاعاتی که به کاربر منتقل می‌کند، اهمیت اساسی دارند. شاید در برخورد اول درک مفهوم خاصیت کمی دشوار باشد - با یک مثال از زندگی روزمره این مفهوم را بهتر درک خواهید کرد.

یک دوچرخه را در نظر بگیرید: این دوچرخه یک شیء است، که برای رفتن از نقطه‌ای به نقطهٔ دیگر از آن استفاده می‌کنید. این دوچرخه (مانند هر شیء فیزیکی دیگر) یکسری خصوصیات ذاتی دارد: کارخانهٔ سازنده، مدل، رنگ، دنده، ترمز، و چرخها. در ویژوال بیسیک به این خصوصیات **خواص** شیء دوچرخه گفته می‌شود. بسیاری از خواص یک دوچرخه در زمان ساخت آن شکل می‌گیرند، ولی خواصی نیز هستند که فقط در زمان استفاده از دوچرخه می‌توان آنها را تصور کرد (خواصی مانند نام مالک، نوع لاستیکها، سرعت حرکت، میزان فرسودگی، و یا وسائل تزئینی آن). همانطور که در کار با ویژوال بیسیک جلو می‌رویم، با این دو نوع خاصیت بیشتر آشنا خواهید شد.

مدیریت ابزارهای برنامه‌نویسی ویژوال استودیو

با ابزارهای برنامه‌نویسی متعددی که در ویژوال استودیو وجود دارد، هر لحظه بیم آن می‌رود که در میان شلوغی پنجره‌ها و پانل‌های مختلف و جورواجور غرق شوید! اما ویژوال استودیو در اینجا نیز شما را تنها رها نکرده است، و برای بهتر شدن اوضاع کارهایی می‌توان انجام داد.

برای جابجا کردن یک پنجرهٔ ابزار، کافیست میلهٔ عنوان آنرا با ماوس گرفته، و به محل جدید بکشید (و البته رها کنید!). اگر این پنجره را روی لبه پنجرهٔ دیگر رها کنید، به آن خواهد چسبید (dock). مزیت پنجره‌های بهم چسبیده اینست که همیشه دیده می‌شوند، و چیزی روی آنها نمی‌پوشاند. اگر می‌خواهید یک پنجرهٔ چسبیده را بزرگتر کنید، کافیست یکی از لبه‌های آنرا بکشید. برای بستن یک پنجره می‌توانید روی دکمهٔ Close آن (علامت x گوشهٔ راست-بالا) کلیک کنید - و برای باز کردن پنجره هم منوی View همیشه در اختیار شماست.

اگر چیزی که می‌خواهید بین بستن و چسباندن پنجره است، می‌توانید از خصوصیت پنجره‌های Autohide استفاده کنید - این پنجره‌ها وقتی ماوس را روی آنها می‌برید، ظاهر می‌شوند، و وقتی با آنها کار ندارید، خود را مخفی می‌کنند. برای تبدیل یک پنجره به پنجرهٔ Autohide ، کافیست روی آیکون Auto Hide (که به شکل یک سنجاق فشاری کوچک و در سمت راست میلهٔ عنوان واقع است) کلیک کنید؛ با این کار پنجرهٔ مزبور ناپدید شده، و به سمت راست فضای کاری ویژوال استودیو (جایی که سایر پنجره‌های Autohide قرار دارند) منتقل می‌شود.

برای باز کردن یک پنجرهٔ Autohide ، کافیست روی لبهٔ آن (در محل قرار گرفتن پنجره‌های Autohide) کلیک کنید، و یا ماوس را روی عنوان آن ببرید. بعد از آن که کارتان با پنجره تمام شد، ماوس را از روی آن دور کنید، تا بطور خودکار ناپدید شود. برای بازگرداندن یک پنجرهٔ Autohide به وضعیت باز بودن دائم، نیز می‌توانید مجدداً روی آیکون سنجاق فشاری (که اکنون بصورت افقی در آمده) کلیک کنید.

مدیریت پنجره‌های ابزار کاریست که برای تسلط بر آن به تمرین و تکرار نیازمند دارید - هدف تمرینهای ذیل نیز همین است.

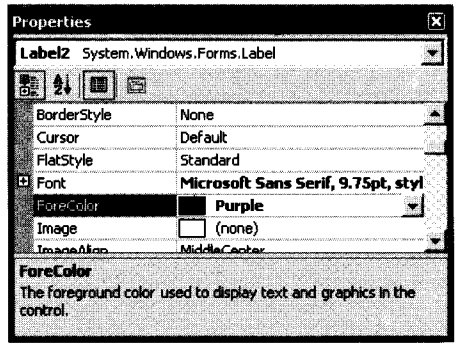
جابجاکردن و تغییر اندازهٔ پنجره‌های ابزار در ویژوال استودیو

جابجا کردن و تغییر دادن اندازهٔ پنجره‌های ابزار از جمله کارهائیست که زیاد با آن سروکار خواهید داشت. در تمرین زیر یاد می‌گیرید چگونه پنجرهٔ خواص را جابجا کنید، و یا آنرا به اندازهٔ دلخواه درآورید.

جابجاکردن و تغییر دادن اندازهٔ پنجرهٔ خواص

۱ (اگر پنجرهٔ خواص باز نیست) با کلیک کردن روی دکمهٔ پنجرهٔ خواص (Properties Window) در میله ابزار استاندارد ویژوال بیسیک، آنرا باز کنید.

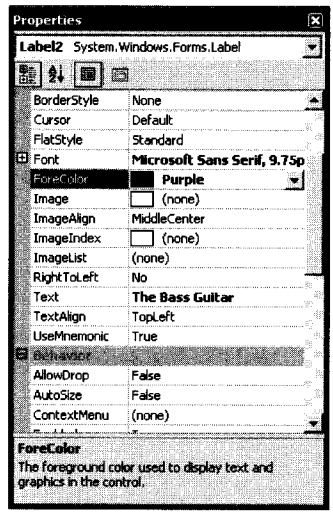
۲ روی میلهٔ عنوان پنجرهٔ خواص دو-کلیک کنید، تا بصورت شناور (غیر-چسبیده) درآید؛ تصویر زیر را ببینید:



۳ میلهٔ عنوان پنجرهٔ خواص را با ماوس گرفته، و به محل دیگری بکشید (اما اجازه ندهید به پنجره‌های دیگر بچسبد).

٤ ماوس را روی گوشهٔ راست-پائین پنجرهٔ خواص ببرید، تا کرسر ماوس به شکل اشاره گر تغییر اندازه (resizing pointer -که یک پیکان دو-سر مورب است) در آید. تغییر دادن اندازهٔ پنجرهٔ خواص هیچ فرقی با سایر پنجره‌های ویندوز ندارد.

۵ برای بزرگ کردن پنجرهٔ خواص، اشاره گر تغییر اندازه را به سمت راست و پائین بکشید:



کار کردن در یک پنجرهٔ بزرگ سریعتر و راحتتر است؛ پس در بزرگ کردن پنجره‌های ابزار (تا آن حد که احساس راحتی کنید) تردید نکنید.

چسباندن یک پنجرهٔ ابزار در ویژوال استودیو

برای چسباندن یک پنجرهٔ شناور (floating window) به پنجره‌های دیگر، کافیست روی میلهٔ عنوان آن دو-کلیک کنید. (همانطور که در تمرین قبل دیدید، دو-کلیک کردن روی میلهٔ عنوان پنجره‌های چسبیده آنها

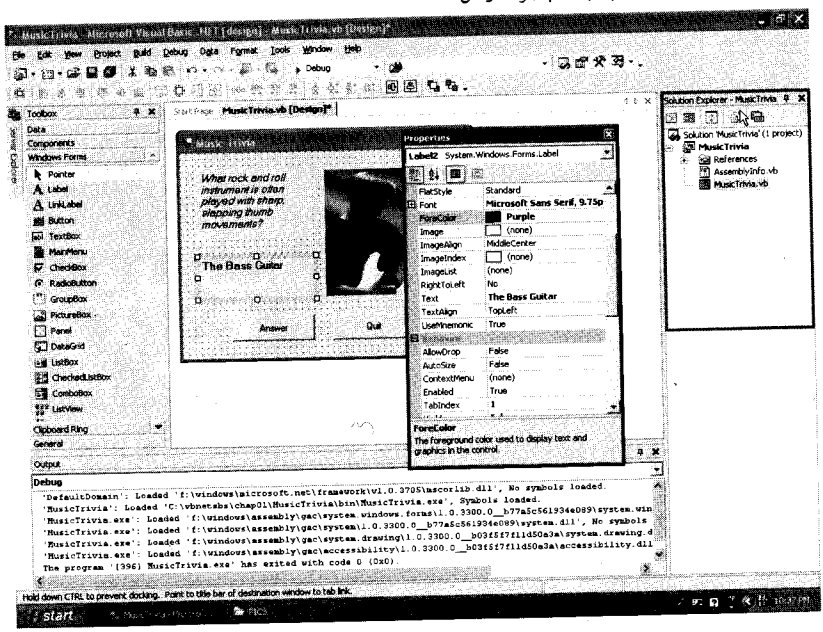
را به حالت شناور درمی‌آورد.) پنجره‌های چسبیده همیشه دیده خواهند شد، و جای کمتری هم می‌گیرند. در تمرین زیر یاد می‌گیرید که چگونه پنجرهٔ خواص را به پنجره‌های دیگر بچسبانید.

چسباندن پنجرهٔ خواص

۱ دقت کنید پنجره‌ای که می‌خواهید این تمرین را روی آن انجام دهید، در حالت شناور باشد (اگر تمرین قبل را دنبال کرده باشید، پنجرهٔ خواص اکنون در این حالت است).

۲ میلهٔ عنوان پنجرهٔ خواص را با ماوس گرفته و آنقدر به بالا (پائین، چپ یا راست - هر طرف که مایلید!) بکشید تا لبهٔ آن با لبهٔ یکی از پنجره‌های دیگر مماس و جفت شود. (از آنجائیکه پنجره‌ها و سمتهای متعددی برای چسباندن وجود دارد، بهترین محل چسباندن یک پنجره را فقط با سعی و خطا می‌توان بدست آورد.)

چسباندن پنجرهٔ خواص



۳ کلید ماوس را رها کنید، تا پنجرهٔ خواص به پنجرهٔ مزبور بچسبد و در جای خود ثابت شود.

نکته

اگر نمی‌خواهید پنجره‌های ابزار در حین جابجا شدن به هم بچسبند، هنگام کشیدن آنها کلید Ctrl را نگه دارید. و یا اگر میل دارید دو پنجره را در هم ادغام کرده و به یک پنجره تبدیل کنید، اولی را کشیده و روی میلهٔ عنوان دومی رها کنید - با اینکار یک پنجره خواهید داشت با دو برگه (برای هر یک از پنجره‌های قبلی) در پائین آن. برای کار با هر یک از این پنجره‌ها، کافیست روی برگهٔ مربوط به آن کلیک کنید. این یکی از روشهای مؤثر صرفه‌جویی در جاست. (برای مثال، پنجره‌های کاوشگر راه‌حل و نمایش کلاس اغلب به همین شکل و با هم هستند.)

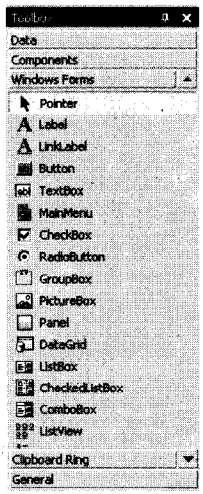
٤ چند جای مختلف را برای چسباندن پنجرهٔ خواص امتحان کنید. شاید برای بار اول این وضعیت کمی گیج‌کننده بنظر برسد، اما بعد از مدتی کاملاً به آن عادت خواهید کرد. توجه داشته باشید که مهمترین هدف رسیدن به بهترین کارایی - سهولت کار با ابزارها و بیشترین فضا برای طراحی فرمها و کُدنویسی - است.

مخفی کردن یک ابزار در ویژوال استودیو

ویژوال استودیو .NET دارای مکانیزمی برای ظاهر و مخفی کردن سریع پنجره‌های ابزار است، که به آن Autohide گفته می‌شود. برای فعال کردن این حالت دکمهٔ Autohide را، که به شکل یک سنجاق فشاری کوچک است و سمت راست میلهٔ عنوان پنجره (کنار دکمهٔ Close) قرار دارد، کلیک کنید. با این کار، پنجرهٔ مزبور خود را در لبهٔ محیط ویژوال استودیو مخفی می‌کند. برای برگشت به حالت اولیه، کافیست یک بار دیگر این دکمه را (که اکنون ۹۰° چرخیده) کلیک کنید. توجه داشته باشید که، ویژگی Autohide فقط در پنجره‌های چسبیده وجود دارد، و برای پنجره‌های شناور نمی‌توانید از آن استفاده کنید.

استفاده از ویژگی AutoHide

۱ جعبه ابزار را پیدا کنید (این پنجره معمولاً در سمت چپ طراح فرمهای ویندوز قرار دارد). بسیاری از کنترلهای استاندارد ویژوال بیسیک را در جعبه ابزار خواهید یافت. کنترلهایی مانند **دکمه**، **برچسب** و **جعبه تصویر**، که در برنامهٔ MusicTrivia از آنها استفاده کردیم، در همین جعبه ابزار قرار دارند. این کنترل‌ها بصورت موضوعی دسته‌بندی شده‌اند، و هر دسته در یک برگهٔ جداگانه قرار داده شده است. اگر جعبه ابزار ویژوال بیسیک را نمی‌بینید، با انتخاب فرمان View|Toolbox آنرا باز کنید:



۲ دکمهٔ Auto Hide را در میلهٔ عنوان جعبه ابزار پیدا کنید. این دکمه در حال حاضر بصورت عمودیست، و نشان می‌دهد که ویژگی Autohide فعال نیست.

۳ دکمهٔ Auto Hide را کلیک کنید، ولی کرسر ماوس را روی جعبه ابزار نگه دارید. توجه کنید که دکمهٔ Auto Hide باندازهٔ ۹۰° به سمت چپ چرخیده، و بصورت افقی در می‌آید. این حالت نشان می‌دهد که ویژگی Autohide فعال شده، و جعبه ابزار دیگر بصورت دائم باز نخواهد ماند.

٤ کرسر ماوس را از روی پنجرهٔ جعبه ابزار بیرون ببرید - با این کار، جعبه ابزار بلافاصله به سمت چپ می‌لغزد، و خود را در زیر برگه‌ای بنام Toolbox پنهان می‌کند. باحتمال زیاد (که البته به پیکربندی ویژوال استودیو بستگی دارد) در بالای برگهٔ Toolbox برگهٔ دیگری بنام Server Explorer (کاوشگر میزبان) می‌بینید، که آن نیز یک پنجرهٔ ابزار است، با این تفاوت که ویژگی Autohide آن از قبل فعال شده است. مهمترین مزیت پنجره‌های Autohide آزاد شدن فضای کاری ویژوال استودیو برای کارهای ضروری‌تر می‌باشد.

۵ کرسر ماوس را روی برگهٔ Toolbox ببرید (و یا روی آن کلیک کنید): پنجرهٔ Toolbox بلافاصله ظاهر می‌شود، و می‌توانید مانند قبل از آن استفاده کنید. (برای آشنایی بیشتر با جعبه ابزار ویژوال بیسیک و طرز کار با آن باید تا فصل آینده صبر کنید.)

۶ کرسر ماوس را از پنجرهٔ جعبه ابزار دور کنید، تا مجدداً مخفی شود.

۷ یک بار دیگر ماوس را روی برگهٔ Toolbox ببرید، و این بار دکمهٔ Autohide را کلیک کنید، تا این پنجره به حالت قبلی (باز بودن دائمی) برگردد.

کمی با پنجره‌های ابزار کار کنید، و تمرینات این قسمت (و قسمتهای قبل) را روی آنها انجام دهید، تا به بهترین ترکیب ممکنه برسید. و فراموش نکنید که همیشه می‌توانید این تنظیمات را تغییر دهید.

کمک!

ویژوال استودیو .NET دارای یک سیستم کمک فعال (Online Help) است، که می‌توانید مطالب مختلف و متنوعی دربارهٔ محیط و امکانات ویژوال استودیو، زبان برنامه‌نویسی ویژوال بیسیک و چارچوب NET. در آن بیابید. اجازه دهید قبل از ادامهٔ کار (و شروع برنامه نویسی)، کمی دربارهٔ این سیستم کمک صحبت کنیم.

نکته

سیستم کمک فعال ویژوال استودیو .NET را می‌توانید از روی CD های آن نصب کنید. اگر مقدار زیادی جای خالی (و بلااستفاده!) روی هارددیسک خود دارید، می‌توانید مستندات ویژوال استودیو .NET را هم از روی این CD ها نصب کنید.

برای کسب اطلاعات از سیستم کمک فعال راههای مختلفی وجود دارد:

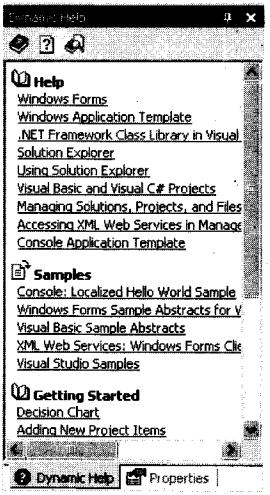
برای کسب اطلاعات دربارهٔ ...	انجام دهید
کاری که در حال انجام آن هستید	روی برگهٔ Dynamic Help در محیط کاری ویژوال استودیو کلیک کنید، و یا فرمان Help|Dynamic Help را اجرا کنید.
یک موضوع خاص	در منوی Help ویژوال استودیو فرمان Contents را اجرا کنید.
دستورات ویژوال بیسیک	در پنجرهٔ ادیتور کُد (Code Editor) روی دستور موردنظر کلیک کرده، و سپس کلید F1 را بزنید.
طرز کار دیالوگ‌ها	در دیالوگ موردنظر، دکمهٔ Help را کلیک کنید.
یک کلمهٔ کلیدی خاص را	فرمان Search را از منوی Help اجرا کرده، و کلمهٔ موردنظر وارد کنید.
پنجره یا برنامه‌ای که مستقل از ویژوال استودیو .NET است	از منوی Start ویندوز، آیتم Programs|Microsoft Visual Studio.NET|Microsoft Visual Studio.NET Documentation را انتخاب کرده، و با استفاده از برگه‌های Contents ، Index و Search اطلاعات موردنظر را پیدا کنید.
سایتهای وب و گروه‌های خبری Community	در صفحهٔ شروع ویژوال استودیو .NET دکمهٔ Online Community را کلیک کرده، و سپس سایت وب یا گروه خبری موردنظر را انتخاب کنید.
چگونگی تماس با پشتیبانی میکروسافت	فرمان Technical Support را از منوی Help اجرا کنید.

در این قسمت با طرز استفاده از کمک دینامیک ویژوال استودیو .NET ، و ویژگیهای آن، آشنا خواهید شد. این ابزار جدید با توجه به کاری که در حال انجام آن هستید، مطالب خود را در اختیار شما می‌گذارد. برای مثال، اگر مشغول کار در ویژوال بیسیک هستید، اطلاعات خود را روی این زبان متمرکز می‌کند، و دربارهٔ زبانهای دیگر ویژوال استودیو (مانند ویژوال سی++ یا ویژوال سی#) صحبتی نخواهد کرد.

اگر بلافاصله بعد از تمریناتی که در قسمتهای قبل انجام دادید، کمک دینامیک ویژوال استودیو را باز کنید، به احتمال زیاد با اطلاعاتی دربارهٔ جعبه ابزار روبرو خواهید شد، چون کمک دینامیک قدم بقدم فعالیتهای شما را زیر نظر می‌گیرد، و خود را با آن هماهنگ می‌کند.

استفاده از کمک دینامیک

۱ روی برگهٔ Dynamic Help در محیط ویژوال استودیو کلیک کنید (یا فرمان Help\|Dynamic Help را اجرا کنید)، تا پنجرهٔ کمک دینامیک باز شود:



پنجرهٔ کمک دینامیک نیز یکی از ابزارهای ویژوال استودیو است، و مانند سایر پنجره‌ها می‌توانید آنرا جابجا کرده، به دیگر پنجره‌ها بچسبانید، و یا حتی بصورت Autohide درآورید.

۲ روی یکی از سرفصل‌های پنجرهٔ کمک دینامیک کلیک کنید. با این کار فرم MusicTrivia ناپدید شده، و مطالب سیستم کمک جای آنرا می‌گیرد.

نکته

اگر مستندات ویژوال استودیو را نصب نکرده باشید، نمی‌توانید از کمک دینامیک استفاده کنید. برای نصب این مستندات، برنامهٔ نصب ویژوال استودیو .NET را مجدداً اجرا کرده، و آیتم Documentation را انتخاب کنید.

۳ چند سرفصل دیگر پنجرهٔ کمک دینامیک را بررسی کنید، و ببینید تا چه حد دربارهٔ نیّات شما هوشمند بوده است.

نکته

برای کنترل رفتار پنجرهٔ کمک دینامیک، فرمان Tools|Options را اجرا کرده، و در پوشهٔ Environment آیتم Dynamic Help را انتخاب کنید. در این قسمت می‌توانید نحوهٔ نمایش اطلاعات (و تعداد لینک‌ها) را مشخص کنید.

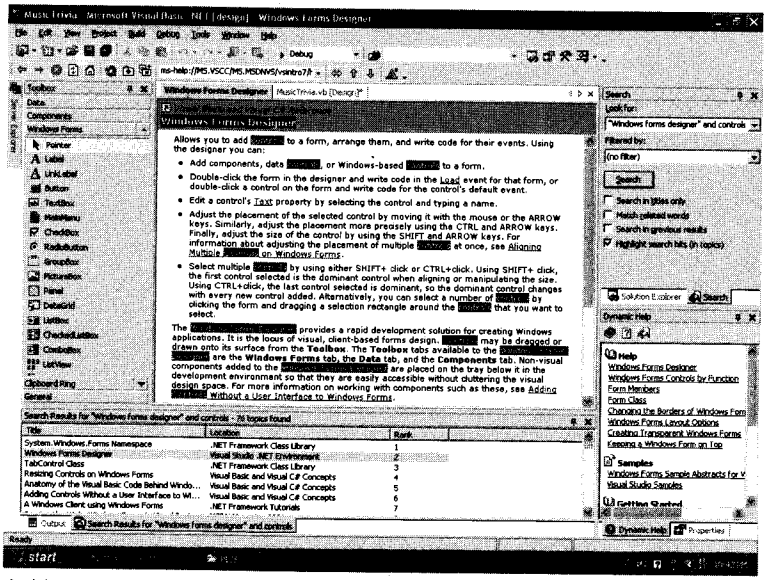
جستجوی کلمات و عبارات در سیستم کمک

۱ از منوی Help آیتم Search را انتخاب کنید، تا پنجرهٔ جستجو در محیط کاری ویژوال استودیو باز شود.

۲ در لیست Look For عبارت **"windows forms designer" and controls** را وارد کنید (" را فراموش نکنید، چون یکی از عبارتها دارای فاصله است).

۳ گزینهٔ Highlight Search Hits (In Topics) را فعال کرده، و سپس دکمهٔ Search را کلیک کنید، تا ویژوال استودیو سرفصل‌های منطبق با عبارت خواسته شده را نمایش دهد.

٤ در پنجرهٔ نتایج جستجو (Search Results) روی یکی از سرفصل‌ها دو-کلیک کنید، تا مطلب مربوطه در پنجرهٔ اصلی نمایش داده شود. دقت کنید که چگونه کلمات مورد جستجو از سایر قسمتهای متن متمایز شده‌اند.



نکته

پنجره‌های Contents و Index نیز جزیی یکپارچه از محیط ویژوال استودیو هستند، که می‌توانید با فرمانهای Help|Contents و Help|Index از آنها استفاده کنید.

یک گام فراتر: خروج از ویژوال استودیو .NET

در پایان هر فصل از این کتاب سرفصلی با عنوان «**یک گام فراتر**» خواهید دید، که مطلبی را در ارتباط با موضوع همان فصل ارائه می‌کند. و بعد از این سرفصل پایانی، تمام مطالب و مفاهیم مهم فصل را در یک «**مرجع سریع**» بصورت فشرده آورده‌ام.

بعد از اینکه کارتان در ویژوال استودیو تمام شد، بایستی پروژه‌های باز شده را ذخیره کرده، و از این محیط خارج شوید. چطور؟

از ویژوال استودیو خارج شوید

۱ با کلیک کردن دکمهٔ Save All در میله ابزار استاندارد، تغییرات برنامه را ذخیره کنید. برخلاف ویژوال بیسیک ۶، در ویژوال بیسیک .NET نام پروژه را باید در همان ابتدای کار وارد کنید، نه زمانی که برای ذخیره کردن آن آماده می‌شوید. (همانطور که بعداً خواهید دید، هر پروژهٔ ویژوال بیسیک .NET تعداد زیادی فایل و پوشه دارد، که همگی آنها در ذیل پوشهٔ اصلی ایجاد می‌شوند.)

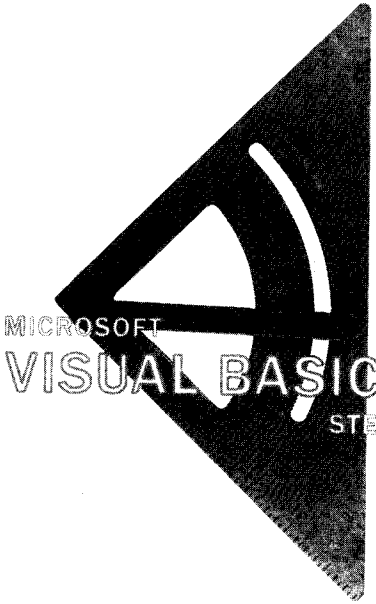
۲ در منوی File روی فرمان Exit کلیک کنید؛ با این کار ویژوال استودیو .NET بسته می‌شود، و فرصت دارید قبل از شروع فصل ۲ نفسی تازه کنید!

مرجع سریع فصل ۱

برای ...	انجام دهید
شروع ویژوال استودیو .NET	در منوی Start|Programs|Microsoft Visual Studio.NET آیتم Microsoft Visual Studio.NET را انتخاب کنید.
باز کردن یک پروژهٔ موجود	از منوی File ویژوال استودیو .NET آیتم Open را انتخاب کرده، و سپس پروژهٔ موردنظر را باز کنید؛ و یا در صفحهٔ شروع ویژوال استودیو، دکمهٔ Open Project را کلیک کنید.
اجرای یک برنامه	در میله ابزار استاندارد دکمهٔ Start را کلیک کنید؛ و یا کلید F5 را بزنید.
ست کردن خواص یک شیء	شیء موردنظر را در فرم برنامه انتخاب کرده، و سپس با کلیک کردن دکمهٔ Properties Window در میله ابزار استاندارد، پنجرهٔ خواص را (اگر باز نیست) باز کنید.
تغییر اندازهٔ یک پنجرهٔ ابزار	روی میلهٔ عنوان پنجرهٔ موردنظر دو-کلیک کنید، تا به حالت شناور درآید. سپس با کشیدن گوشه‌های پنجره، اندازهٔ آنرا تغییر دهید.
جابجا کردن یک پنجرهٔ ابزار	روی میلهٔ عنوان پنجرهٔ موردنظر دو-کلیک کنید، تا به حالت شناور درآید. سپس با گرفتن میلهٔ عنوان پنجره، آنرا به محل دلخواه بکشید (و رها کنید).
چسباندن یک پنجرهٔ ابزار	روی میلهٔ عنوان پنجرهٔ موردنظر دو-کلیک کنید؛ و یا آنقدر آنرا جابجا کنید، تا لبهٔ آن با لبهٔ پنجرهٔ دیگر مماس شود.
فعال کردن ویژگی Autohide	روی دکمهٔ Autohide - که سمت راست میلهٔ عنوان پنجره قرار یک پنجرهٔ ابزار دارد - کلیک کنید.
غیرفعال کردن ویژگی Autohide	بعد از ظاهر کردن پنجرهٔ موردنظر، مجدداً روی دکمهٔ یک پنجرهٔ ابزار Autohide آن کلیک کنید.
خارج شدن از ویژوال استودیو .NET	در منوی File ویژوال استودیو .NET ، آیتم Exit را کلیک کنید.

فصل

۲



اولین برنامه را بنویسید

در این فصل یاد می گیرید چگونه :

- ✔ برای یک برنامهٔ جدید واسط کاربر ایجاد کنید.
- ✔ خواص اشیاء فرم برنامه را ست کنید.
- ✔ برای برنامه کُد بنویسید.
- ✔ برنامه را ذخیره و اجرا کنید.
- ✔ فایل اجرایی برنامه (.exe) را بسازید.

همانطور که در فصل قبل دیدید محیط ویژوال استودیو .NET هر آنچه را که برای نوشتن برنامه‌های ویندوز لازمست، در خود دارد. در این فصل خواهید دید که چگونه می‌توان یک برنامه ساده (ولی جذاب) را با ابزارهای ویژوال بیسیک .NET طراحی و ایجاد کرد. هر برنامهٔ ویندوز از دو بخش مهم تشکیل می‌شود: واسط کاربر (user interface) و کُد (code) . برای ایجاد واسط کاربر از جعبه ابزار و پنجرهٔ خواص ویژوال بیسیک .NET استفاده می‌کنیم، و برای کُدنویسی از پنجرهٔ کُد (Code Window) آن. در پایان فصل هم خواهید دید که چگونه می‌توان یک برنامه را ذخیره، کامپایل (compile) و اجرا کرد.

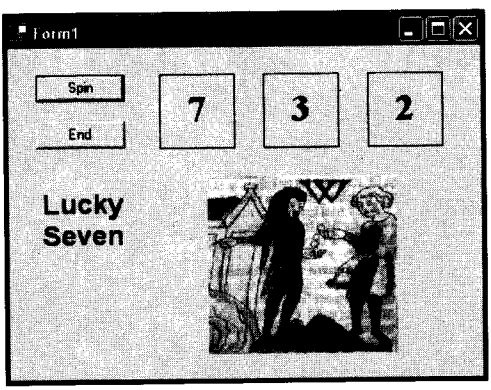
تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- منوها و میله ابزارهای ویژوال بیسیک .NET نسبت به ویژوال بیسیک ۶ تغییرات زیادی کرده‌اند. برای مثال، در ویژوال بیسیک ۶ منوهای Format ، Run و Add-Ins را دیده بودید، که در ویژوال بیسیک .NET اثری از آنها دیده نمی‌شود. اکثر این فرمانها بصورت جدیدی سازماندهی شده‌اند - مثلاً، بسیاری از فرمانهای منوی Run را در منوی Debug خواهید یافت.
- نام کنترل CommandButton در ویژوال بیسیک .NET به Button تغییر یافته، و بسیاری از خواص آن نیز عوض شده‌اند. برای مثال، خاصیت Caption این کنترل به Text تغییر نام داده است.
- برخی از خواص و متدهای کنترل Label جدید هستند، و یا نام آنها عوض شده است. برای مثال، خاصیت Caption این کنترل به Text تغییر نام داده، و خاصیت TextAlign (که جدید است) گزینه‌های بیشتری نسبت به خاصیت قدیمی Alignment دارد.
- کنترل Image دیگر در ویژوال بیسیک .NET وجود ندارد، و برای نمایش هر نوع تصویری باید از PictureBox استفاده کنید.
- تعداد دستوراتی که کامپایلر ویژوال بیسیک .NET بطور خودکار به هر فرم اضافه می‌کند، بسیار بیشتر از ویژوال بیسیک ۶ است. مهمترین این کُدها، که به بالای هر فرم اضافه می‌شود، با عنوان "Windows Forms Designer generated code" مشخص می‌شود - این کُدها اساسی‌ترین خصوصیات فرم را مشخص می‌کنند، و مطلقاً نباید آنها را دستکاری کنید. هر کُدی را که می‌خواهید بنویسید، زیر این قسمت وارد کنید.
- ویژوال استودیو .NET برای هر پروژه دو نوع فایل اجرایی می‌تواند ایجاد کند: debug build و release build . فایل اجرایی نوع debug build حاوی اطلاعات دیباگ برنامه است، و هنگام تست برنامه بکار می‌رود. فایل اجرایی نوع release build کوچکتر (و بهینه‌تر) است، و معمولاً پس از تکمیل برنامه ایجاد می‌شود.

اولین برنامهٔ ویژوال بیسیک: Locky Seven

برنامه‌ای که در این فصل خواهیم نوشت، یک بازی شانسی است بنام Locky Seven . این برنامه بسیار ساده است، و در چند دقیقه می‌توان آنرا نوشت. (نسخهٔ کامل و اجرایی برنامهٔ Locky Seven را در CD ضمیمهٔ کتاب خواهید یافت.) وقتی این برنامه کامل شود، هنگام اجرای آن پنجره‌ای مانند زیر خواهید دید:



مراحل برنامه‌نویسی

واسط کاربر برنامهٔ Locky Seven دارای دو دکمه (یکی برای انجام بازی و دیگری برای خروج از برنامه)، سه برچسب (برای نمایش اعداد شانسی)، یک تصویر (برای نمایش برد یا باخت)، و یک برچسب دیگر (برای نمایش عنوان برنامه) است. پس از تکمیل واسط کاربر، باید برای دکمه‌های Spin (ایجاد اعداد شانسی) و End (خروج از برنامه) کُد بنویسیم. برای نوشتن یک برنامه (مانند Locky Seven) بایستی سه مرحله را طی کنیم: ایجاد واسط کاربر، ست کردن خواص اشیاء برنامه، و کُدنویسی. در جدول زیر مراحل نوشتن برنامهٔ Locky Seven را ملاحظه می‌کنید:

مراحل برنامه‌نویسی	برای ...
۱- واسط کاربر را ایجاد کنید	۷ شیء
۲- خواص اشیاء را ست کنید	۱۲ خاصیت
۳- کُد برنامه را بنویسید	۲ شیء

ایجاد واسط کاربر

برای ایجاد برنامهٔ Locky Seven ، ابتدا باید واسط کاربر آنرا بسازیم. برای اینکار، یک پروژهٔ جدید باز کرده، و با استفاده از کنترلهای جعبه ابزار فرم برنامه را طراحی می‌کنیم.

ایجاد یک پروژهٔ جدید

۱ ویژوال استودیو .NET را باز کنید.

۲ در صفحهٔ شروع ویژوال استودیو، ابتدا روی لینک Get Started کلیک کرده، و سپس دکمهٔ New Project را کلیک کنید.

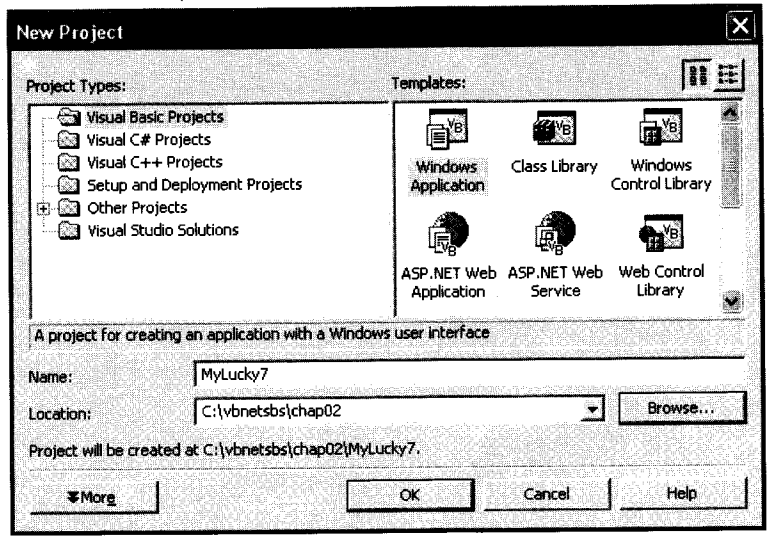
نکته

راه دیگر ایجاد یک پروژهٔ جدید، باز کردن منوی File|New و انتخاب آیتم Project است.

در پنجرهٔ New Project ویژوال استودیو انواع مختلفی از برنامه‌ها و سرویسهای ویندوز و وب می‌بینید، که می‌توانید از میان آنها انتخاب کنید. زبان برنامه‌نویسی، نام پروژه و محل ذخیره کردن آنرا هم باید در همین پنجره انتخاب کنید.

۳ در قسمت Project Types پوشهٔ Visual Basic Projects را انتخاب کرده، و در قسمت Templates آیکون Windows Application را کلیک کنید. با این انتخابها ویژوال استودیو متوجه می‌شود که می‌خواهید با ویژوال بیسیک .NET یک برنامهٔ ویندوز بنویسید.

٤ در فیلد Name نام پروژه را MyLocky7 بگذارید، و در فیلد Location محل ذخیره شدن آن (مثلاً، c:\vbnetsbs\chap02) را مشخص کنید (برای این کار از دکمهٔ Browse نیز می‌توانید استفاده کنید).



نکته

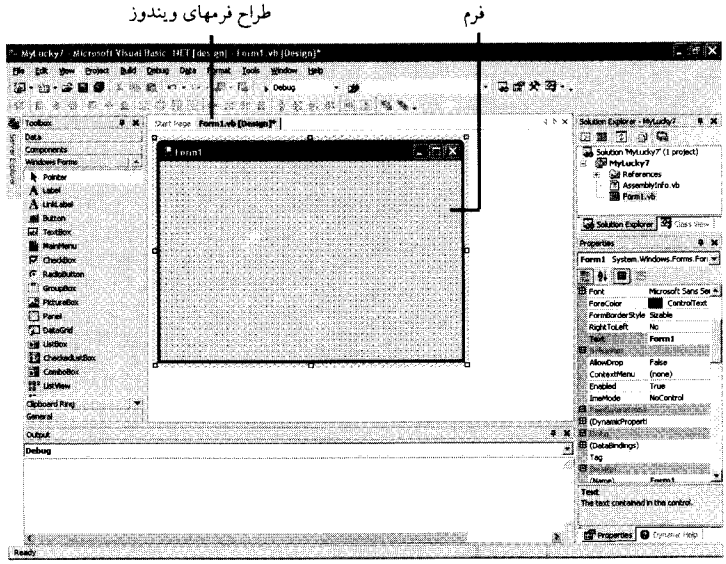
آیتمهایی که در قسمتهای Project Types و Templates می‌بینید، به ویرایش ویژوال استودیوی شما بستگی دارد. آنچه در تصویر بالا می‌بینید مربوط به ویژوال استودیو .NET ویرایش حرفه‌ای (Professional Edition) است -که الزاماً با ویرایشی که شما از استفاده می‌کنید، یکی نیست، و می‌تواند بیشتر یا کمتر باشد.

۵ OK را کلیک کنید، تا ویژوال استودیو پروژهٔ جدید را با مشخصاتی که خواسته‌اید، ایجاد کند. چیزی که بعد از این خواهید دید، یک فرم ویندوز (Windows form) خالیست (به فرم ویندوز اغلب به سادگی فرم گفته می‌شود). این فرمیست که باید واسط کاربر را روی آن بنا کنید.

مرحلهٔ بعدی قرار دادن دکمه‌ها روی این فرم است.

ایجاد واسط کاربر

۱ ماوس را روی گوشهٔ راست-پائین فرم ببرید، تا کرسر به پیکان دو-سر مورب تبدیل شود؛ این گوشه را گرفته و بکشید تا فرم برای آنچه می‌خواهیم روی آن قرار دهیم، جای کافی داشته باشد. بسته به وضوح مانیتورتان (یا فضایی که در اختیار دارید) شاید نتوانید تمام این فرم را روی صفحه ببینید، اما جای نگرانی نیست؛ هر گاه بخواهید می‌توانید به تمام نقاط فرم برنامه دسترسی داشته باشید. فرم را به اندازه‌ای درآورید، که تقریباً معادل تصویر زیر باشد:

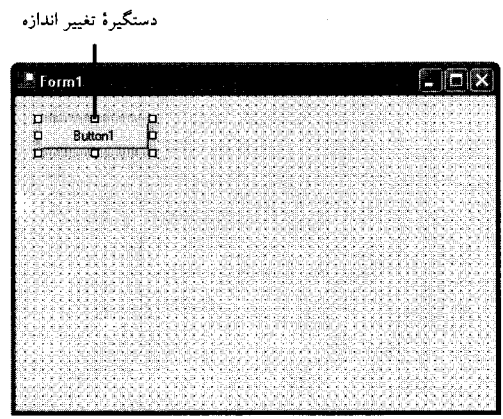


(برای دیدن تمامی فرم می‌توانید تعدادی از پنجره‌های ابزار را ببندید، یا آنها را کوچکتر کنید.)

۲ برای اضافه کردن اولین دکمه، روی کنترل Button در جعبه ابزار ویژوال بیسیک کلیک کنید.

۳ ماوس را روی فرم ببرید: کرسر به علامت + و آیکون دکمه تبدیل می‌شود. به کمک این + می‌توانید دکمهٔ موردنظر را روی فرم رسم کنید. برای این کار، بایستی کلید سمت چپ ماوس را گرفته و یک مستطیل (با ابعاد دلخواه) روی فرم بکشید.

٤ ماوس را به نزدیکی گوشهٔ چپ-بالای فرم برده، و یک دکمه آنجا بکشید. شکل زیر را ببینید:



در اطراف دکمه‌ای که ایجاد کرده‌اید، هشت دستگیرهٔ تغییر اندازه (resizing handle) - به شکل مربعهای سفید کوچک - می‌بینید. نام پیش‌فرض اولین دکمه Button1 است - این نام را بخاطر بسپارید، چون موقع نوشتن کُد برنامه به آن احتیاج خواهیم داشت.

مکان و اندازهٔ دکمه‌ها را براحتی می‌توانید در محیط طراحی ویژوال بیسیک تغییر دهید (اما جابجا کردن کنترل‌های برنامه در هنگام اجرا کمی مشکل است، و نیاز به ست کردن یک خاصیت ویژه دارد). در قسمت آینده روش جابجا کردن و تغییر دادن اندازهٔ دکمهٔ Button1 را خواهید دید.

جابجا کردن و تغییر دادن اندازهٔ یک دکمه

۱ ماوس را روی دکمهٔ Button1 ببرید، تا کرسر به پیکان چهار-سر تبدیل شود؛ کلید چپ ماوس را گرفته، کمی به سمت راست و پائین بکشید، و سپس آنرا رها کنید. از آنجائیکه ویژگی تراز شدن کنترل‌ها با نقاط شبکه‌ای فرم (Align to Grid) بصورت پیش‌فرض فعال است، دکمه به محاذات این نقاط قرار خواهد گرفت. (نقاط شبکه‌ای به منظور طراحی دقیقتر فرمها تعبیه شده‌اند. برای تغییر دادن فاصلهٔ این نقاط، بعد از اجرای فرمان Tools|Options، پوشهٔ Windows Forms Designer را انتخاب کرده و مقدار موردنظر را به خاصیت GridSize بدهید.)

۲ ماوس را روی گوشهٔ راست-پائین دکمهٔ Button1 ببرید، تا کرسر به پیکان دو-سر مورب تبدیل شود؛ با گرفتن و کشیدن این نقطه می‌توانید اندازهٔ دکمه را تغییر دهید.

۳ دستگیرهٔ تغییر اندازهٔ گوشهٔ راست-پائین دکمهٔ Button1 را کمی به سمت پائین و راست بکشید، تا بزرگتر شود.

٤ به همان روشی که در مراحل قبل دیدید، دکمهٔ Button1 را کوچک کرده، و به جای قبلیش برگردانید.

حال باید دکمهٔ دوم را به فرم اضافه کنیم.

اضافه کردن دکمهٔ دوم

۱ روی کنترل Button در جعبه ابزار کلیک کنید.

۲ دکمهٔ دوم را زیر اولی رسم کنید (و سعی کنید هم اندازهٔ آن باشد).

نکته

سریعترین راه برای اضافه کردن دکمه به فرم، دو-کلیک کردن روی کنترل Button در جعبه ابزار است. با این کار یک دکمه با اندازهٔ پیش‌فرض روی فرم قرار خواهد گرفت.

۳ مکان و اندازهٔ این دکمه را هم مانند قبل ست کنید. اگر اشتباه کردید، می‌توانید دکمه را حذف کرده، و کار را از اول شروع کنید. (برای حذف یک کنترل، آنرا انتخاب کنید، و کلید Del را بزنید.)

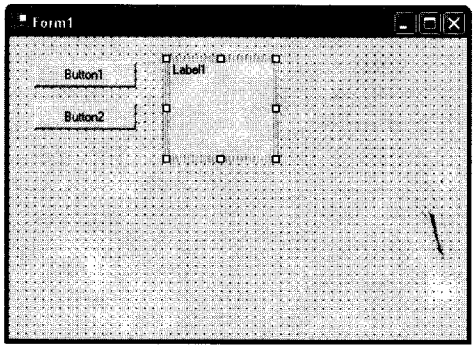
در مرحلهٔ بعد برچسب‌های نمایش اعداد شانسی را به فرم اضافه می‌کنیم. برچسب (label) کنترلیست که عمدتاً از آن برای نمایش متن و عدد استفاده می‌شود. وقتی کاربر دکمهٔ Spin را کلیک کند، سه عدد بطور اتفاقی در این برچسب‌ها نمایش داده خواهد شد -که برای برنده شدن حداقل یکی از آنها باید عدد شانس، یعنی ۷، باشد.

اضافه کردن برچسب‌های اعداد

۱ روی کنترل Label در جعبه ابزار کلیک کنید.

۲ ماوس را روی فرم ببرید: کرسر به علامت + و آیکون برچسب (حرف A بزرگ) تبدیل می‌شود.

۳ یک مستطیل کوچک، مانند آنچه در شکل زیر می‌بینید، رسم کنید:



نام پیش‌فرض این برچسب Label1 خواهد بود. برنامهٔ ما، علاوه برای این برچسب، به دو برچسب دیگر نیاز دارد.

٤ دوباره روی کنترل Label در جعبه ابزار کلیک کنید، و یک برچسب دیگر (با همان ابعاد) در سمت راست برچسب قبلی رسم کنید. این برچسب Label2 نام خواهد گرفت.

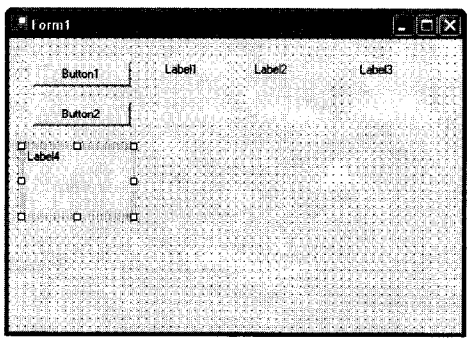
۵ با همان روش فوق، برچسب سوم را هم در سمت راست Label2 رسم کنید. نام این برچسب Label3 خواهد بود.

اجازه دهید همین جا برچسب نام برنامه را هم به فرم اضافه کنیم. این چهارمین (و آخرین) برچسب برنامه است.

۶ روی کنترل Label در جعبه ابزار کلیک کنید.

۷ یک برچسب بزرگ، درست زیر دکمه‌ها، رسم کنید.

تا اینجای کار، فرم برنامه باید چیزی شبیه شکل زیر شده باشد:

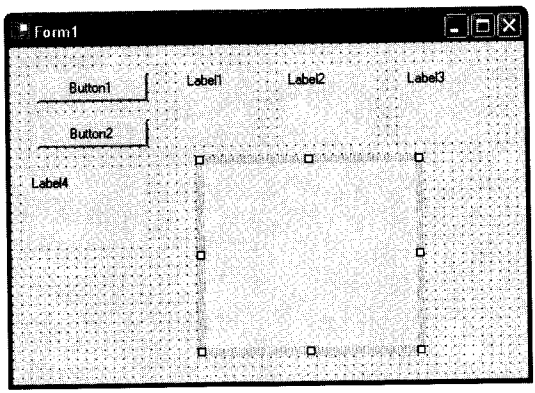


در مرحلهٔ بعد تصویر اعلام برنده شدن کاربر را به فرم اضافه می‌کنیم. برای این کار از یک کنترل جعبه تصویر استفاده خواهیم کرد. جعبه تصویر (picture box) کنترلیست برای نمایش هر نوع تصویر، عکس، آیکون و مانند اینها.

اضافه کردن تصویر

۱ روی کنترل PictureBox در جعبه ابزار کلیک کنید.

۲ ماوس را روی فرم ببرید، و یک مستطیل بزرگ زیر برچسب‌های اعداد شانسی رسم کنید؛ شکل زیر را ببینید:



نام پیش‌فرض این جعبه تصویر PictureBox1 خواهد بود - آنرا نیز بخاطر بسپارید، چون در کُد برنامه به آن مراجعه خواهیم کرد.

بعد از اضافه کردن کنترل‌ها به فرم، باید چند خاصیت را نیز ست کنیم.

ست کردن خواص کنترل‌ها

همانطور که در فصل قبل دیدید، برای ست کردن خواص یک شیء ابتدا باید آنرا انتخاب کرده، و سپس از طریق پنجرهٔ خواص، خاصیتهای موردنظر را تغییر دهید. اجازه دهید از دکمه‌ها شروع کنیم.

ست کردن خواص دکمه‌ها

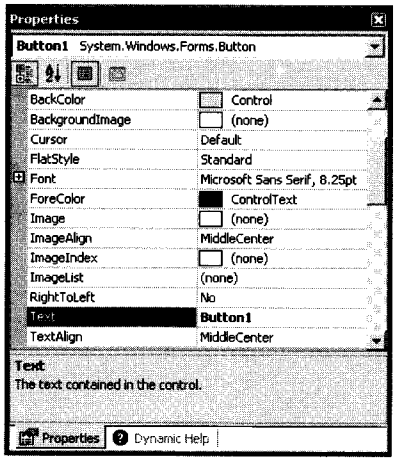
۱ روی دکمهٔ اول (Button1) کلیک و آنرا انتخاب کنید.

۲ روی میلهٔ عنوان پنجرهٔ خواص دو-کلیک کنید.

نکته

اگر پنجرهٔ خواص را نمی‌بینید، با فرمان View|Properties Window (یا زدن کلید F4) آنرا باز کنید.

۳ پنجرهٔ خواص را آنقدر بزرگ کنید که بتوانید نام هر خاصیت و مقدار آنرا بطور واضح ببینید (وقتی به کار با این پنجره عادت کنید، دیگر نیازی به بزرگ کردن بیش از حد آن نیست):



پنجرهٔ خواص نام و مقدار تمام خواص دکمهٔ Button1 را نشان می‌دهد. بدلیل تعداد زیاد خواص هر کنترل، ویژوال استودیو آنها را بصورت موضوعی دسته‌بندی می‌کند. برای دیدن خواص ذیل هر موضوع، کافیست روی علامت +کنار نام آن کلیک کنید.

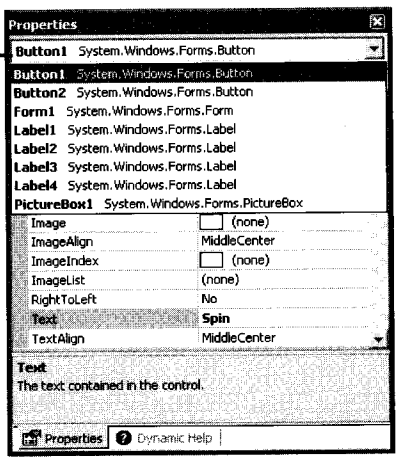
۴ آنقدر در پنجرهٔ خواص پائین بروید، تا به خاصیت Text برسید (این خاصیت در ذیل موضوع Appearance قرار دارد).

۵ روی نام خاصیت Text در ستون سمت چپ دو-کلیک کنید، تا مقدار آن در ستون سمت راست انتخاب شود (مقدار این خاصیت در حال حاضر "Button1" است).

۶ کلمهٔ **Spin** را وارد کرده، و کلید Enter را بزنید. تأثیر این عمل را بلافاصله مشاهده خواهید کرد: عنوان دکمهٔ اول روی فرم به "Spin" تغییر می‌کند. بعد از آن نوبت عوض کردن عنوان دکمهٔ دوم است.

۷ در بالای پنجرهٔ خواص، لیست Object را باز کنید؛ در این لیست تمام اشیاء موجود در فرم برنامه را می‌بینید:

لیست Object به شما اجازه می‌دهد اشیاء مختلف را به سرعت انتخاب، و خواص آنها را ست کنید



۸ در این لیست، روی آیتم **Button2** System.Windows.Forms.Button کلیک کنید. با این کار، خواص دکمهٔ دوم در پنجرهٔ خواص نشان داده خواهد شد (اگر دقت کنید، روی فرم برنامه هم این دکمه بصورت انتخاب شده درمی‌آید).

۹ روی خاصیت Text این کنترل دو-کلیک کرده، و پس از وارد کردن **End** (بجای Button2)، Enter را بزنید - با این کار، عنوان دکمهٔ دوم به "End" تغییر خواهد کرد.

نکته

استفاده از لیست Object بهترین راه برای سوئیچ کردن بین کنترلهای مختلف یک فرم است.

پس از دکمه‌ها، نوبت به ست کردن خواص برچسب‌های اعداد می‌رسد. این برچسب‌ها بسیار شبیه یکدیگرند، و خواص آنها را می‌توان بصورت گروهی ست کرد (برچسب معرفی برنامه کمی متفاوت است، و خواص آنرا جداگانه ست خواهیم کرد).

ست کردن خواص برچسب اعداد

۱ روی برچسب اول (Label1) کلیک کرده، و سپس با نگه داشتن کلید Shift روی برچسب‌های دوم و سوم نیز کلیک کنید، تا هر سه با هم انتخاب شوند. با این کار می‌توان خواص موردنظر (TextAlign ، BorderStyle ، و Font) را همزمان برای هر سه برچسب ست کرد. (این خاصیت‌ها همگی در دستهٔ Appearance قرار دارند.)

نکته

وقتی چند شیء را با هم انتخاب کنید، فقط آن خواصی را در پنجرهٔ خواص خواهید دید که بتوان آنها را همزمان برای چند شیء ست کرد.

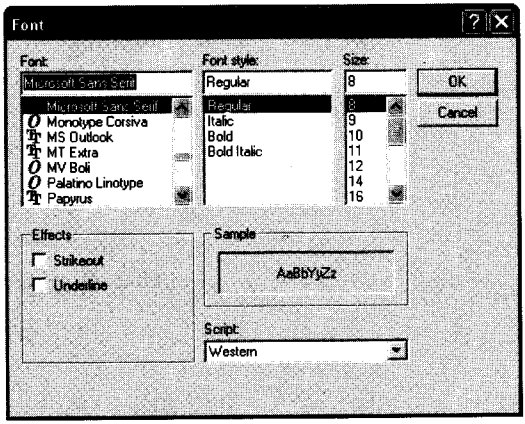
۲ خاصیت TextAlign - تراز متن - را در پنجرهٔ خواص انتخاب کرده، و لیست ستون سمت راست آنرا باز کنید. به کمک آیتمهای موجود در این لیست (که به شکل آیکونهای گرافیکی هستند، و آنها را در بسیاری از برنامه‌های ویندوز دیده‌اید) می‌توانید تراز متن برچسب‌ها را انتخاب کنید.

۳ گزینهٔ تراز وسط (MiddleCenter) را انتخاب کنید.

٤ پس از آن، خاصیت BorderStyle - نوع حاشیه - را انتخاب کرده، و لیست ستون سمت راست آنرا باز کنید. در این لیست سه گزینه می‌بینید: None (بدون حاشیه)، FixedSingle (حاشیهٔ ثابت تک خطی)، و Fixed3D (حاشیهٔ ثابت سه‌بُعدی).

۵ گزینهٔ FixedSingle را انتخاب کنید.

۶ سپس، خاصیت Font - فونت - را انتخاب کرده، و دکمهٔ ... را در ستون سمت راست آن کلیک کنید. با این کار، دیالوگ فونت ظاهر می‌شود:



۷ فونت Times New Roman را انتخاب کرده، و بعد از ست کردن نوع فونت به Bold و اندازهٔ آن به 24 ، OK را کلیک کنید.

پس از آن باید متنی را که در حال حاضر در برچسب‌ها وجود دارد، پاک کنیم. برای این کار لازمست که هر یک از برچسب‌ها بصورت مستقل انتخاب شود.

۸ روی یک قسمت خالی فرم کلیک کنید تا برچسب‌ها از حالت انتخاب شده خارج شوند، و سپس روی برچسب اول کلیک و آنرا انتخاب کنید.

۹ روی خاصیت Text دو-کلیک کرده، و پس از زدن کلید Del ، Enter را بزنید، تا متن این برچسب پاک شود.

۱۰ متن برچسب‌های دوم و سوم را هم به روش فوق پاک کنید.

این از برچسب‌های اعداد - حالا نوبت برچسب چهارم (برچسب نام و معرفی برنامه) است.

ست کردن خواص برچسب معرفی برنامه

۱ روی برچسب چهارم (Label4) کلیک کنید.

۲ خاصیت Text این کنترل را به **Lucky Seven** تغییر دهید.

۳ روی دکمهٔ ... در ستون سمت راست خاصیت Font کلیک کنید.

٤ بعد از ست کردن فونت این برچسب به Arial ، نوع Bold و اندازهٔ 18 ، OK را کلیک کنید.

۵ خاصیت ForeColor - رنگ متن - را انتخاب کرده، و لیست ستون سمت راست آنرا باز کنید. در این لیست سه برگه وجود دارد: Custom (مجموعه‌ای از تمام رنگهای موجود در سیستم)، Web (مجموعه‌ای از رنگهای مناسب برای نمایش در صفحات وب) و System (مجموعه رنگهایی که ویندوز در قسمتهای مختلف از آنها استفاده می‌کند - اینها در واقع همان رنگهای بخش Appearance دیالوگ Display Properties هستند).

۶ از برگهٔ Custom رنگ ارغوانی تیره را انتخاب کنید.

پس از آن نوبت به ست کردن خواص آخرین شیء برنامه می‌رسد.

خواص جعبه تصویر

جعبه تصویری که روی فرم قرار داده‌ایم، مسئول نمایش تصویریست که برنده شدن کاربر را در مسابقه اعلام می‌کند؛ این تصویر فایلیست با فرمت JPEG . برای این شیء سه خاصیت را باید ست کنیم: خاصیت SizeMode (که رفتار جعبه تصویر را کنترل می‌کند)، Image (تصویری که جعبه تصویر باید نمایش دهد)، و Visible (عدم نمایش تصویر تا لحظهٔ موعود).

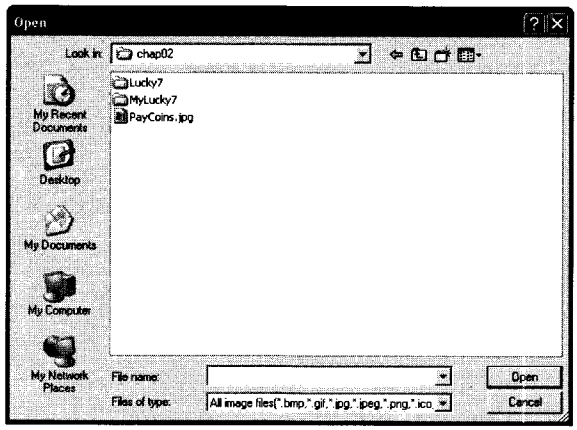
ست کردن خواص جعبه تصویر

۱ روی جعبه تصویر (PictureBox1) کلیک و آنرا انتخاب کنید.

۲ روی خاصیت SizeMode در پنجرهٔ خواص کلیک کرده، و از لیست ستون سمت راست آن گزینهٔ StretchImage را انتخاب کنید. ست کردن خاصیت SizeMode به StretchImage (قبل از باز کردن تصویر موردنظر) باعث می‌شود تا تصویر دقیقاً خود را به اندازهٔ جعبه تصویر درآورد.

۳ خاصیت Image را انتخاب کرده، و روی دکمهٔ ... در ستون سمت راست آن کلیک کنید. با این کار دیالوگ بازکردن فایل (Open) باز می‌شود.

۴ در این دیالوگ، به پوشهٔ c:\vbnetsbs\chap02 بروید.



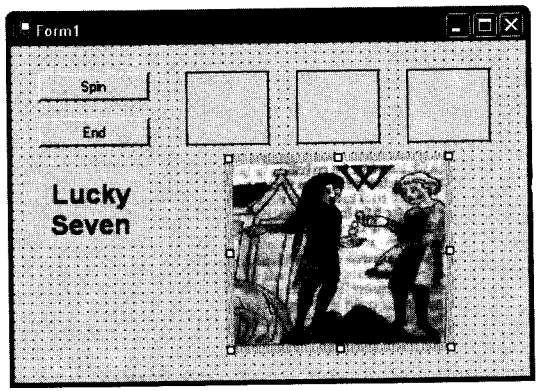
۵ فایل PayCoins.jpg را انتخاب کرده، و دکمهٔ Open را کلیک کنید - این فایل بلافاصله در جعبه تصویر بار خواهد شد.

از آنجائیکه تصویر فوق فقط در حالت برنده شدن کاربر باید نمایش داده شود، بایستی در شروع برنامه آنرا ناپدید کنیم. برای این منظور از خاصیت Visible کنترل جعبه تصویر استفاده خواهیم کرد (نمایش تصویر بر عهدهٔ کُد برنامه است). پس:

۶ خاصیت Visible را (که در دستهٔ Behavior پنجرهٔ خواص قرار دارد) انتخاب کرده، و لیست ستون سمت راست آنرا باز کنید.

۷ گزینهٔ False را انتخاب کنید، تا تصویر PayCoins.jpg در شروع برنامه دیده نشود.

۸ ست شدن خاصیت Visible جعبه تصویر به False اثری روی نمایش تصویر آن در مرحلهٔ طراحی برنامه ندارد، و فرم برنامه تا اینجا باید چیزی شبیه شکل زیر باشد:



۹ روی میلهٔ عنوان پنجرهٔ خواص دو-کلیک کنید، تا دوباره به حالت چسبیده برگردد.

نوشتن کُد برنامه

اکنون آماده‌ایم تا کُد برنامهٔ Locky Seven را بنویسیم. اکثر اشیاء برنامه خودشان می‌دانند چکار باید بکنند، و وظیفهٔ خود را بطور خودکار انجام می‌دهند؛ آنها این ویژگی‌ها را به ارث برده‌اند، که این کمک بزرگی به برنامه‌نویسان است. اما چاشنی اصلی برنامه یعنی تولید اعداد شانسی، نمایش آنها و تشخیص برد یا باخت، چیزی نیست که ویژوال بیسیک بتواند بخودی خود از عهدهٔ آن برآید -این ما هستیم که باید با جزئیات دقیق و کامل به او بگوئیم چه کاری می‌خواهیم برایمان انجام دهد. اینها دستوراتی هستند که دکمه‌های Spin و End باید اجرا کنند، چون عملکرد اصلی برنامه بر عهدهٔ آنهاست. برای برنامه‌نویسی در ویژوال بیسیک از ادیتور کُد (Code Editor) استفاده می‌کنیم.

روش دیگری برای نمایش خواص اشیاء

در این فصل خواص اشیاء برنامهٔ Locky Seven را قدم بقدم ست کردیم؛ این روش بسیار وقت‌گیر و گیج‌کننده است. روش ساده‌تری نیز برای نمایش خواص اشیاء یک برنامه وجود دارد، و آن استفاده از جدول است. در جدول زیر خواص اشیاء برنامهٔ Locky Seven را بصورت خلاصه می‌بینید:

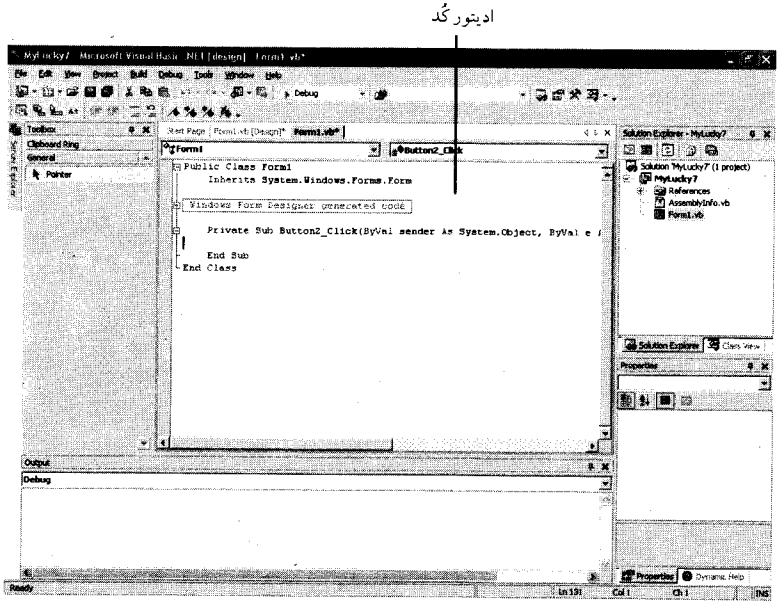
شیء	خاصیت	مقدار
Button1	Text	"Spin"
Button2	Text	"End"
Label1, Label2	BorderStyle	FixedSingle
Label3	Font	Times New Roman, Bold, 24-point

شیء	خاصیت	مقدار
	Text	(خالی)
	TextAlign	MiddleCenter
Label4	Text	"Locky Seven"
	Font	Arial, Bold, 18-point
	ForeColor	(ارغوانی)
PictureBox1	Image	"c:\vbnetsbs\chap02\paycoins.jpg"
	SizeMode	StretchImage
	Visible	False

در این قسمت خواهید دید که چگونه می‌توان کُد برنامهٔ Locky Seven را در ادیتور کُد وارد کرد.

کار با ادیتور کُد

۱ برای نوشتن کُد برنامه، روی یکی از دکمه‌ها (مثلاً، دکمهٔ End) دو-کلیک کنید - بعد از چند لحظه پنجرهٔ ادیتور کُد باز می‌شود (شکل زیر را ببینید). در ادیتور کُد تمام کُدهای مربوط به یک فرم را خواهید دید.



به چند دستور که یک کار مشخص را انجام می‌دهند، روال (procedure) گفته می‌شود. در ویژوال بیسیک دو نوع روال وجود دارد، که یکی از آنها سابروتین (subroutine) است؛ این نوع روال با کلمهٔ کلیدی Sub در اولین خط شروع شده، و با کلمهٔ کلیدی End Sub پایان می‌یابد. روال‌ها معمولاً به یک رویداد (event) خاص وابسته‌اند، و فقط زمانی اجرا می‌شوند که آن رویداد (مثلاً، کلیک شدن یک دکمه) اتفاق افتاده باشد. از اینرو به روال‌ها مجری رویداد (event handler) یا روال رویداد (event procedure) نیز گفته می‌شود.

وقتی روی دکمهٔ End (Button2) دو-کلیک کنید، ویژوال بیسیک بطور خودکار اولین و آخرین خط روال رویداد دکمهٔ End را برایتان ایجاد خواهد کرد (دستورات زیر را ببینید). در ادیتور کُد دستورات دیگری نیز می‌بینید، که آنها را هم ویژوال بیسیک برایتان نوشته است - همانطور که قبلاً هم گفتم اینها برای اجرای صحیح برنامه بسیار مهم هستند، و نباید آنها را دستکاری کنید (بهترین کار اینست که آنها را نادیده بگیرید!).

```
Private Sub Button2_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button2.Click

End Sub
```

روال فوق با کلیک شدن دکمهٔ Button2 اجرا می‌شود - به این رویداد رویداد تحریک‌کنندهٔ روال (trigger event) گفته می‌شود. همانطور که خواهید دید، رویدادهای زیادی وجود دارند که می‌توانند باعث تحریک یک روال شوند.

در محلی که کرسر چشمک می‌زند، کلمهٔ End را نوشته، و کلید (⊡) Down را بزنید. با این کار، کلمهٔ End به رنگ آبی درمی‌آید، و این نشان می‌دهد که ویژوال بیسیک آنرا بعنوان یک دستور معتبر (یا کلمهٔ کلیدی - keyword) شناخته است. دستور End باعث توقف اجرای برنامه (و پایان یافتن آن) می‌شود.

ویژوال بیسیک دارای صدها کلمهٔ کلیدی مانند این است، که نوشتن صحیح آنها اهمیت زیادی دارد، چون فقط از این طریق است که کامپایلر (compiler) می‌تواند تفاوت آنها را با سایر کلماتی که در برنامه می‌نویسید، درک کند.

نکته

به املای صحیح دستورات ویژوال بیسیک، و طرز بکارگیری آنها، دستور زبان (syntax) می‌گویند.

اگر توجه کنید، دستور End دارای مقدار کمی تورفتگی (indent) نسبت به نقطهٔ شروع دو دستور دیگر (Sub و End Sub) نیز هست - این کار را هم کامپایلر ویژوال بیسیک انجام می‌دهد، و هدف از آن افزایش خوانایی برنامه است.

نکته

به مجموعه قواعدی که باعث افزایش خوانایی برنامه می‌شوند، سبک برنامه‌نویسی (programming style) گفته می‌شود.

کُد دکمهٔ End بسیار ساده بود، ولی دکمهٔ Spin چنین نیست. دستوراتی که در روال رویداد این دکمه خواهید دید، اکثراً برایتان ناآشنا خواهند بود؛ ولی نگران نباشید، چون در طول کتاب تمامی آنها را یاد خواهید گرفت.

فقط کافیست دستورات را به همان شکل که می‌بینید، در ادیتور کُد وارد کنید. (ویژوال بیسیک در مورد املای کلمات کلیدی و طرز نوشتن دستورات بسیار سختگیر و وسواسی است!)

نوشتن کُد دکمهٔ Spin

۱ در کاوشگر راه‌حل دکمهٔ نمایش طراح فرم (View Designer) را کلیک کنید، تا فرم برنامه دوباره ظاهر شود (وقتی ادیتور کُد باز است، نمی‌توانید فرم را ببینید). اگر برنامه چندین فرم دارد، ابتدا فرم موردنظر را در کاوشگر راه‌حل انتخاب کرده و سپس این دکمه را کلیک کنید.

نکته

برای دیدن فرم برنامه می‌توانید روی برگهٔ "Form1.vb [Design]" در بالای ادیتور کُد نیز کلیک کنید. اگر این برگه را نمی‌بینید، باید گزینهٔ Tabbed Documentd را در دیالوگ Tools|Options فعال کنید (به فصل قبل مراجعه کنید).

۲ روی دکمهٔ Spin دو-کلیک کنید - ادیتور کُد مجدداً باز شده، و روال رویداد کلیک دکمهٔ Button1 در اختیار شما قرار خواهد گرفت.

همانطور که می‌بینید، با اینکه متن این دکمه (خاصیت Text) را به Spin تغییر دادیم، ولی نام آن همچنان Button1 است - نام (Name) و متن (Text) دو خاصیت مستقل از هم هستند، و می‌توانند یکی نباشند.

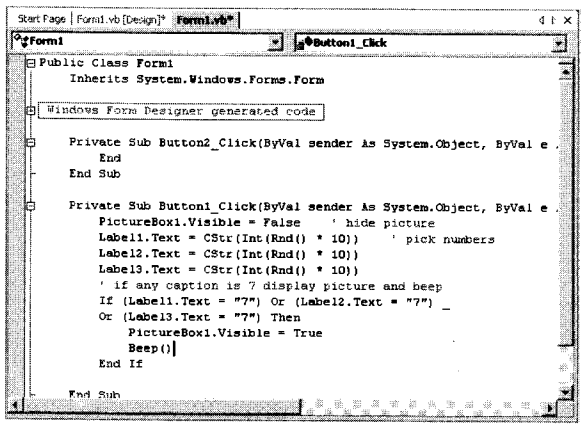
۳ دستورات زیر را بین ...Private Sub Button1_Click و End Sub دکمهٔ Button1 وارد کنید - دقت کنید که بعد از وارد کردن هر خط باید Enter را بزنید. دستورات را دقیقاً به همان شکلی که می‌بینید، وارد کنید. اگر دستوری را اشتباه نوشتید (و ویژوال بیسیک زیر آن یک خط موج‌دار قرمز کشید)، آنرا حذف کرده و از اول وارد کنید.

نکته

همانطور که دستورات را می‌نویسید، ویژوال بیسیک آنها را با رنگهای مختلف فرمت خواهد کرد، تا بتوانید بخشهای مختلف هر دستور را بهتر تشخیص دهید. وقتی می‌خواهید بعد از نام یک شیء خاصیتی را بنویسید، ویژوال بیسیک لیستی از خواص آن شیء را به شما نشان می‌دهد، که می‌توانید روی خاصیت موردنظر دو-کلیک کرده، و دیگر زحمت نوشتن را نکشید. اگر دستوری را اشتباه نوشته باشید، ویژوال بیسیک با یک پیام خطا به شما اخطار خواهد کرد - دستور را مجدداً بررسی کرده، و بعد از رفع اشتباه به کار ادامه دهید. گاهی ویژوال بیسیک خود دستوری را اضافه می‌کند؛ برای مثال، اگر شروع به نوشتن یک دستور If کنید، ویژوال بیسیک بطور خودکار End If را به انتهای آن اضافه می‌کند. (کسانی که قبلاً با ویژوال بیسیک کار کرده‌اند، در اوایل کار با VB.NET این ویژگی را کمی ناراحت‌کننده خواهند یافت!)

```
PictureBox1.Visible = False    ' hide picture
Label1.Text = CStr(Int(Rnd() * 10))    ' pick numbers
Label2.Text = CStr(Int(Rnd() * 10))
Label3.Text = CStr(Int(Rnd() * 10))
' if any caption is 7 display picture and beep
If (Label1.Text = "7") Or (Label2.Text = "7") Or (Label3.Text = "7") Then
    PictureBox1.Visible = True
    Beep()
End If
```


بعد از نوشتن کُد فوق، ادیتور کُد به شکل زیر در خواهد آمد:



٤ از منوی File فرمان Save All را اجرا کنید، تا برنامه ذخیره شود. فرمان Save All هر چیزی که در پروژه وجود دارد (پروژه، فرمها و کُدهای آن) را ذخیره می‌کند. برای ذخیره کردن آیتمی که مشغول کار روی آن هستید، باید فرمان File|Save را اجرا کنید. فرمان File|Save As نیز آیتم موردنظر را با نام جدید ذخیره می‌کند.

تحلیل روال Button1_Click

روال Button1_Click زمانی اجرا می‌شود که کاربر روی دکمهٔ Spin کلیک کند. این روال در نگاه اول کمی پیچیده بنظر می‌رسد، چون با دستوراتی که در آن بکار رفته، آشنایی ندارید. اما اگر قدری در آن دقیق شوید، شاید چیزهای آشنایی پیدا کنید.

روال Button1_Click سه کار انجام می‌دهد: تصویر را پنهان می‌کند، سه عدد شانسی تولید می‌کند، و اگر یکی از این اعداد "۷" بود، تصویر را نمایش می‌دهد. اجازه دهید به هر یک از این کارها جداگانه نگاه کنیم.

مخفی کردن تصویر در خط اول انجام می‌شود:

```
PictureBox1.Visible = False    ' hide picture
```

این خط خود از دو بخش تشکیل شده است: یک دستور (statement) ، و یک توضیح (comment) . دستور فوق (یعنی، PictureBox1.Visible = False) خاصیت Visible شیء PictureBox1 (جعبه تصویر) را به

False ست می‌کند (این خاصیت فقط دو مقدار False و True می‌تواند بگیرد). اگر بخاطر داشته باشید، قبلاً خاصیت Visible جعبه تصویر را در پنجرهٔ خواص به False ست کرده بودیم؛ پس چرا این کار را دوباره انجام داده‌ایم؟ مهمترین دلیل آنست که ست کردن این خاصیت در زمان طراحی برنامه فقط در شروع کار مؤثر است، و اگر این خاصیت به هر دلیل تغییر کند (مثلاً، در حین بازی کاربر یک بار برنده شده باشد) بایستی آنرا دوباره به حالت نامرئی برگردانیم. در فصل آینده باز هم در این زمینه (ست کردن خواص در هنگام اجرای برنامه) صحبت خواهم کرد.

دومین بخش از خط اول (که روی مانیتور آنرا به رنگ سبز می‌بینید)، یک توضیح است. توضیح (همانطور که از نامش برمی‌آید) کُد برنامه را توضیح می‌دهد(!)، و با علامت ' شروع می‌شود. در واقع، ویژوال بیسیک هر آنچه را که بعد از علامت ' در یک خط ببیند، نادیده خواهد گرفت. توضیحات برای اجرای برنامه مطلقاً الزامی نیستند، و فقط آنرا مستند می‌کنند. اما شاید هیچ کاری مهمتر از مستند کردن کُد برنامه وجود نداشته باشد! دستورات و کُدها بعد از مدتی کوتاه، حتی برای کسی که آنها را نوشته، نامفهوم می‌شوند؛ وجود توضیحات خوب برای کسی که یک برنامه را می‌خواند، نعمتی واقعاً بزرگ است.

سه خط بعدی اعداد شانسی را تولید می‌کنند. اما یک عدد شانسی (تصادفی) چیست؟ عدد شانسی به عددی گفته می‌شود که مقدار آن از قبل معلوم نیست (حتی برای خود ویژوال بیسیک). تابع Rnd یک عدد شانسی بین 0 و 1 (بزرگتر از 0 و کوچکتر از 1) تولید می‌کند؛ این عدد سپس در 10 ضرب می‌شود (تا عددی بین 0 و 10 بدست آید)؛ تابع Int نیز قسمت صحیح عدد مزبور را جدا می‌کند. تا اینجا عددی داریم که بین 0 و 9 است – و این دقیقاً همان چیزیست که لازم داریم. اما اعداد را نمی‌توان مستقیماً در برچسب نوشت، چون برچسب فقط برای نمایش متن است. برای نوشتن عدد بدست آمده در برچسب‌ها، بایستی ابتدا آنرا به متن تبدیل کنیم – و این کاریست که تابع CStr انجام می‌دهد.

```
Label1.Text = CStr(Int(Rnd() * 10))    ' pick numbers
```

دقت کنید که چگونه توابع Rnd ، Int و CStr در یک دستور جمع شده‌اند، تا کاری را انجام دهند. اعداد مزبور با فونت Times New Roman, Bold, 24-point در برچسب‌ها نمایش داده خواهند شد، چون قبلاً خواص مربوط به فونت آنها را چنین ست کرده‌ایم. در زیر مراحل این کار را بصورت تفکیک شده ملاحظه می‌کنید:

دستور
`Label1.Text = CStr(Int(Rnd() * 10))`

کُد	نتیجه
Rnd()	0.7055475
Rnd() * 10	7.055475
Int(Rnd() * 10)	7
CStr(Int(Rnd() * 10))	"7"
Label1.Text = CStr(Int(Rnd() * 10))	7

آخرین دسته از دستورات روال Button1_Click چک می‌کنند که آیا در میان اعداد شانسی تولید شده عدد **هفت** وجود دارد، یا خیر. اگر حداقل یکی از این اعداد هفت باشد، برنامه تصویر PayCoins.jpg را نشان داده، و با یک بوق برنده شدن شما را اعلام می‌کند:

```
' if any caption is 7 display picture and beep
If (Label1.Text = "7") Or (Label2.Text = "7") Or (Label3.Text = "7") Then
    PictureBox1.Visible = True
    Beep()
End If
```


هر بار که کاربر دکمهٔ Spin (Button1) را کلیک کند، روال Button1_Click فراخوانی شده و تمام دستورات فوق اجرا می‌شوند.

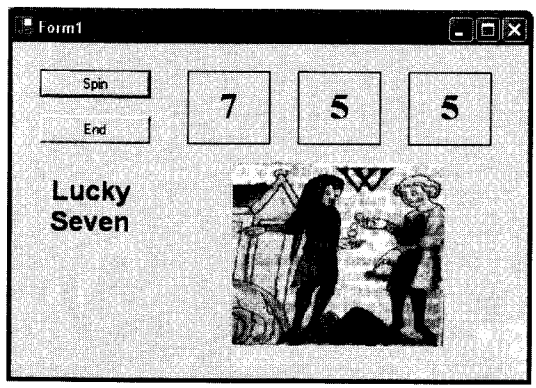
اجرای برنامه‌های ویژوال بیسیک .NET

بعد از نوشتن برنامه، آماده‌ایم که آنرا اجرا کنیم. برای اجرای یک برنامه در محیط ویژوال استودیو، کافیست فرمان Start را از منوی Debug اجرا کرده، دکمهٔ Start را در میله ابزار استاندارد کلیک کنید، و یا کلید F5 را بزنید. با هر یک از این روشها، برنامهٔ Locky Seven را اجرا کنید - اگر ویژوال بیسیک خطایی نشان داد، برگردید، آنرا برطرف کنید، و سپس دوباره برنامه را اجرا کنید.

اجرای برنامهٔ Lucky Seven

۱ در میله ابزار استاندارد، دکمهٔ Start (با آیکون ▶) را کلیک کنید - برنامهٔ Locky Seven کامپایل و اجرا می‌شود، و شما می‌توانید فرم برنامه را به همان شکلی که طراحی کرده‌اید، ببینید.

۲ روی دکمهٔ Spin کلیک کنید. برنامه سه عدد شانسی تولید کرده، و در برچسب‌ها نمایش می‌دهد:



همانطور که می‌بینید، یکی از این اعداد هفت است، بنابراین برنامه تصویر PayCoins.jpg را نشان داده، و بوق آنرا هم حتماً شنیده‌اید. (صدایی که دستور beep تولید می‌کند، به تنظیمات Sounds And Multimedia در پانل کنترل ویندوز بستگی دارد، و می‌توانید آنرا بدلخواه خود تغییر دهید.)

۳ چند بار دیگر دکمهٔ Spin را کلیک کرده، و نتیجه را ببینید. همانطور که متوجه شده‌اید، احتمال برد در

این برنامه نسبتاً زیاد است (در واقع، احتمال آن ۲/۸ به ۱۰ است). شاید میل داشته باشید برنامه را کمی سخت‌تر کنید، مثلاً تعداد هفت‌های لازم برای برنده شدن را (به دو یا سه) افزایش دهید.

۴ بعد از آن که باندازهٔ کافی بازی کردید، با کلیک کردن دکمهٔ End برنامه را ببندید.

نکته

اگر برنامه را دوباره اجرا کنید، متوجه می‌شوید که اعداد شانسی درست به همان ترتیب قبلی تولید می‌شوند. نگران نباشید، هیچ اشکالی وجود ندارد - تابع Rnd ویژوال بیسیک طوری طراحی شده که همیشه در ابتدا یک سری اعداد تصادفی مشابه تولید کند، تا بتوان برنامه را بدرستی و در شرایط مشابه تست کرد. برای تولید اعداد تصادفی واقعی باید از تابع دیگری بنام Randomize نیز کمک بگیرید (تمرین پایان فصل را ببینید).

ایجاد فایل اجرایی برنامه

آخرین کاری که در این فصل انجام خواهیم داد، ایجاد یک برنامهٔ مستقل ویندوز (برنامه‌ای که برای اجرا شدن متکی به محیط ویژوال بیسیک نباشد) است. برای این کار باید فایل اجرایی (executable file) برنامه را بسازیم؛ این فایل پسوند exe. دارد، و می‌تواند روی هر کامپیوتری که ویندوز (و تعدادی فایل پشتیبانی دیگر) را دارد، اجرا شود. (با نصب ویژوال بیسیک .NET این فایلها بطور خودکار روی سیستم نصب خواهند شد.) برای توزیع یک برنامه (و اجرای آن روی کامپیوترهایی که ویژوال بیسیک ندارند) باید بطریق خاصی عمل کنید - روش کار را در فصل ۱۴ خواهید دید.

چیزی که در اینجا باید بدانید اینست که، ویژوال استودیو دو نوع فایل اجرایی می‌تواند تولید کند: **تولید آزمایشی** (debug build) ، و **تولید نهایی** (release build) . فایل اجرایی پیش‌فرض ویژوال استودیو، نوع آزمایشی است، که در فرآیند تولید و تست برنامه بدفعات از آن استفاده خواهید کرد. فایل اجرایی آزمایشی بسرعت کامپایل و ساخته می‌شود، ولی بدلیل وجود اطلاعات دیباگ کمی کُندتر است. بعد از آن که از تکمیل یک برنامه مطمئن شدید، باید فایل اجرایی نهایی آنرا بسازید. این نوع فایل کوچکتر و سریعتر است، و اطلاعات دیباگ را هم ندارد.

نکته

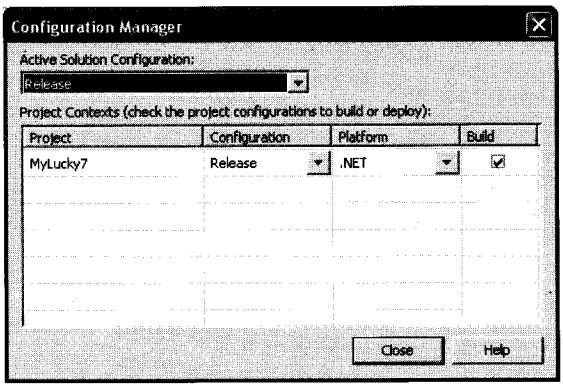
علیرغم وجود برخی شباهتها، روش کامپایل و توزیع برنامه‌ها در ویژوال بیسیک .NET با ویژوال بیسیک ۶ متفاوت است. برای مثال، فرمان Make Project.exe در منوی File ویژوال بیسیک ۶ دیگر وجود ندارد، و بجای آن باید از فرمان Build Solution در منوی Build استفاده کنید. فایلهای اسمبلی (Assembly) و مانیفست (Manifest) نیز از ویژگیهای جدید ویژوال بیسیک .NET محسوب می‌شوند. (فصل ۱۴ بطور کامل به مبحث توزیع برنامه‌های ویژوال بیسیک .NET اختصاص داده شده، که می‌توانید به آن مراجعه کنید.)

در این قسمت فایل اجرایی نهایی برنامهٔ Locky Seven را ایجاد خواهیم کرد.

ایجاد فایل اجرایی MyLucky7.exe

۱ از منوی Build فرمان Configuration Manager را انتخاب کنید، تا پنجرهٔ مدیر پیکربندی باز شود. در این پنجره می‌توانید تعیین کنید که چه نوع فایل اجرایی باید ایجاد شود.

۲ در لیست Active Solution Configuration آیتم Release را انتخاب کرده، و سپس با کلیک کردن دکمهٔ Close پنجرهٔ مدیر پیکربندی را ببندید.



از این پس، پروژه بصورت تولید نهایی کامپایل خواهد شد.

نکته

نوع کامپایل شدن پروژه را از لیست Solution Configuration در میله ابزار استاندارد نیز می‌توانید انتخاب کنید.

۳ از منوی Build فرمان Build Solution را انتخاب کنید. این فرمان یک پوشه بنام bin در محل ذخیره شدن پروژه ایجاد کرده، و خروجی کامپایلر (فایل MyLocky7.exe) را در آن قرار می‌دهد. البته ویژوال استودیو بطور خودکار و به فواصل منظم پروژه را کامپایل و در این محل ذخیره می‌کند؛ اما توصیه می‌کنم هر از چند گاه خودتان بصورت دستی نیز این کار را انجام دهید (بویژه وقتی می‌خواهید فایل اجرایی نهایی را بسازید).

پس از کامپایل کردن برنامه، بهتر است آنرا اجرا کنیم.

٤ منوی Start ویندوز را باز کرده، و آیتم Run را انتخاب کنید، تا پنجرهٔ Run باز شود.

۵ دکمهٔ Browse را کلیک کرده، و به پوشهٔ c:\vbnetsbs\chap02\mylocky7\bin بروید.

۶ فایل MyLocky7.exe را انتخاب کرده، و بعد از کلیک کردن Open، OK را کلیک کنید.

۷ با این کار برنامهٔ Locky Seven در محیط ویندوز (خارج از ویژوال استودیو) اجرا می‌شود.

۸ بازی را چند بار اجرا کرده، و سپس با کلیک کردن دکمهٔ End آنرا ببندید.

نکته

روش دیگر برای اجرای یک برنامهٔ اجرایی، دو-کلیک کردن روی فایل مربوطه در کاوشگر ویندوز است. اگر میل دارید برنامهٔ Locky Seven را از روی میز کار ویندوز اجرا کنید، به روش زیر عمل کنید: روی یک نقطهٔ خالی میز کار ویندوز راست-کلیک کرده، و در منویی که ظاهر می‌شود، ابتدا New و سپس Shortcut را کلیک کنید. در پنجرهٔ Create Shortcut ، به کمک دکمهٔ Browse فایل اجرایی MyLocky7.exe را انتخاب کنید. بترتیب، دکمه‌های Open ، Next و Finish را کلیک کنید، تا آیکون میانبر فایل MyLocky7 روی میز کار ویندوز قرار گیرد. با دو-کلیک روی این آیکون، برنامهٔ Locky Seven اجرا خواهد شد.

۹ با انتخاب فرمان File|Exit پروژهٔ Locky Seven را بسته، و از ویژوال استودیو خارج شوید.

یک گام فراتر: اصلاح برنامه

یک پروژهٔ برنامه‌نویسی را هر گاه که بخواهید می‌توانید دستکاری کنید. برای این منظور کافیست آنرا در ویژوال استودیو باز کرده، و تغییرات لازم را در آن بدهید. در این قسمت دستور Randomize را به برنامهٔ Locky Seven اضافه خواهیم کرد.

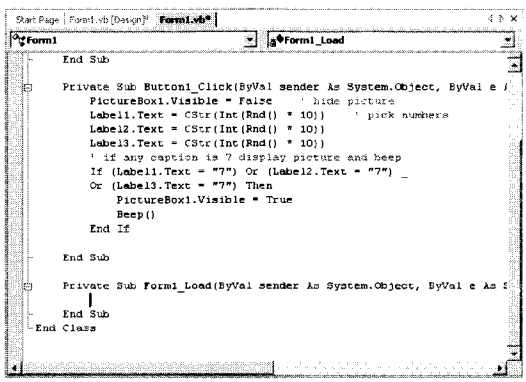
پروژهٔ Locky Seven را باز کنید

۱ با کلیک کردن روی آیکون Start|Programs|Microsoft Visual Studio .NET|Microsoft Visual Studio .NET ، ویژوال استودیو را باز کنید. در صفحهٔ شروع، لیستی از آخرین پروژه‌هایی که در ویژوال استودیو باز شده‌اند، خواهید دید. از آنجائیکه پروژهٔ Locky Seven آخرین پروژه‌ای بود که روی آن کار می‌کردیم، آنرا در بالای لیست خواهید دید.

۲ روی لینک MyLocky7 کلیک کنید، تا پروژهٔ Locky Seven باز شود.

در اینجا می‌خواهیم دستور Randomize را به روال Form_Load اضافه کنیم. روال Form_Load روال خاصیست که با هر بار اجرای برنامه فراخوانی می‌شود.

۳ روی یک نقطهٔ خالی فرم Form1.vb دو-کلیک کنید، تا روال Form_Load در ادیتور کُد باز شود:



۴ دستور Randomize را وارد کرده، و کلید (⊡) Down را بزنید - از این پس، هر بار که برنامه اجرا شود، تابع Randomize نیز اجرا خواهد شد. تابع Randomize با استفاده از ساعت سیستم یک نقطهٔ شروع مناسب برای تولید اعداد تصادفی بوجود می‌آورد، که تابع Rnd می‌تواند به کمک آن اعداد تصادفی واقعی تولید کند.

۵ ویرایش جدید برنامهٔ Locky Seven را اجرا کنید؛ همانطور که می‌بینید، اعداد شانسی دیگر از نظم خاصی پیروی نمی‌کنند، و واقعاً تصادفی هستند. (در صورت نیاز، فایل اجرایی برنامه را مجدداً ایجاد کنید.)

۶ با فرمان File|Close Solution فایلهای پروژه را ببندید.

مرجع سریع فصل ۲

برای ...	انجام دهید
ایجاد واسط کاربر	اشیاء مورد نیاز را از جعبه ابزار ویژوال بیسیک برّداشته، روی فرم قرار داده، و خواص آنها را ست کنید.
جابجا کردن یک شیء	ماوس را روی شیء موردنظر ببرید، تا کرسر به شکل پیکان چهار-سر درآید. شیء را گرفته، به محل جدید بکشید، و رها کنید.
تغییر دادن اندازهٔ یک شیء	شیء موردنظر را انتخاب کرده، و سپس ماوس را روی آن ببرید؛ یکی از دستگیره‌های تغییر اندازه را گرفته و بکشید.
حذف یک شیء	شیء موردنظر را انتخاب کرده، و کلید Del را بزنید.
باز کردن ادیتور کُد	روی یکی از اشیاء (یا فرم) برنامه دو-کلیک کنید؛ و یا در پنجرهٔ کاوشگر راه‌حل، روی دکمهٔ View Code کلیک کنید.
نوشتن کُد برنامه	دستورات موردنظر را در ادیتور کُد وارد کنید.
ذخیره کردن برنامه	فرمان File|Save All را انتخاب کنید؛ و یا دکمهٔ Save All را در میله ابزار استاندارد کلیک کنید.
ذخیره کردن یک فرم	پس از انتخاب فرم موردنظر، فرمان File|Save را اجرا کنید؛ و یا دکمهٔ Save را در میله ابزار استاندارد کلیک کنید.
تغییر دادن نوع فایل اجرایی تولید شده	بعد از اجرای فرمان Build|Configuration Manager ، نوع فایل اجرایی موردنظر (Debug یا Release) را از لیست Active Solution Configuration انتخاب کنید؛ و یا نوع فایل اجرایی را از لیست Solution Configuration در میله ابزار استاندارد انتخاب کنید.
ایجاد یک فایل exe.	فرمان Build را از منوی Build انتخاب کنید.
باز کردن مجدد یک پروژه	فرمان File|Open|Project را انتخاب کنید؛ یا در قسمت File|Recent Projects روی پروژهٔ موردنظر کلیک کنید؛ و یا در صفحهٔ شروع ویژوال استودیو، روی نام پروژهٔ موردنظر کلیک کنید.

فصل

۳

کار با کنترل‌های ویژوال بیسیک .NET

در این فصل یاد می گیرید چگونه:

- با استفاده از کنترل دکمه یک برنامه "Hello World" بنویسید.
- با کنترل تاریخ‌وقت‌نگر زمان روز تولد خود را نمایش دهید.
- با استفاده از کنترل‌های جعبه چک، دکمهٔ رادیویی، جعبه لیست و جعبه ترکیبی ورودی کاربر را پردازش کنید.
- با کنترل برچسب‌لینک یک صفحهٔ وب را نمایش دهید.
- کنترل‌های اکتیوایکس را نصب کنید.

در فصلهای قبل یاد گرفتید که کنترل‌های ویژوال بیسیک ابزارهای گرافیکی هستند، که برنامه به کمک آنها ساخته می‌شود. این کنترل‌ها در جعبه ابزار (Toolbox) قرار دارند، و با چند کلیک ساده می‌توانید از آنها در برنامهٔ خود استفاده کنید. کنترل‌های فرم ویندوز (Windows Forms Controls) را، که برای کار در برنامه‌های ویندوز طراحی شده‌اند، در برگهٔ Windows Forms جعبه ابزار خواهید یافت (در فصل قبل از چند تا از آنها استفاده کردیم). این کتاب عمدتاً دربارهٔ کار با کنترل‌های ویندوز است.

در این فصل با چند تا از کنترل‌های مهم ویژوال بیسیک .NET آشنا خواهید شد، و یاد می‌گیرید چگونه به کمک آنها با کاربر ارتباط فعال برقرار کنید. همچنین خواهید دید که چگونه می‌توان با استفاده از کنترل‌های اکتیوایکس (ActiveX) توانائیهای ذاتی ویژوال بیسیک را توسعه داد.

تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- کنترل جدیدی بنام تاریخ-وقت-گزین (DateTimePicker) اضافه شده است، که با آن می‌توان تاریخ و وقت را به کاربر نشان داد. کنترل جدید دیگری نیز وجود دارد بنام برچسب-لینک (LinkLabel)، که نمایش صفحات وب را بسیار آسان کرده است.
- کنترل جدیدی بنام دکمهٔ رادیویی (RadioButton) جای کنترل قدیمی OptionButton را گرفته است.
- بجای کنترل قدیمی Frame می‌توانید از کنترل جدیدی بنام جعبهٔ گروهی (GroupBox) استفاده کنید.
- خاصیت ListIndex در کنترل جعبه لیست (و جعبه ترکیبی) جای خود را به خاصیت جدیدی بنام SelectedIndex داده است.
- کنترل Image دیگر در ویژوال بیسیک .NET وجود ندارد، و برای نمایش هر نوع تصویری باید از کنترل جعبه تصویر (PictureBox) استفاده کنید.
- برای اضافه کردن تصاویر به کنترل جعبه تصویر، بجای تابع LoadPicture، باید از متد System.Drawing.Image.FromFile استفاده کنید.
- برای اجرای سایر برنامه‌ها از دل یک برنامهٔ ویژوال بیسیک .NET، باید از متد System.Diagnostics.Process.Strat استفاده کنید.
- روش اضافه کردن کنترل‌های اکتیوایکس به جعبه ابزار تغییر کرده است، و ویژوال استودیو بگونه‌ای آنها را در یک "لفاف" می‌پوشاند، که ویژوال بیسیک .NET بتواند از این اشیاء قدیمی استفاده کند.

استفاده از کنترل‌ها: برنامهٔ "Hello World"

یکی از برنامه‌هایی که در کتابهای برنامه‌نویسی به سنت تبدیل شده، برنامهٔ "Hello World" است. این یک برنامهٔ کوچک است، که پایه‌ای‌ترین مفاهیم برنامه‌نویسی را آموزش می‌دهد. در دوران کهن DOS برنامهٔ "Hello World" یک برنامهٔ دو (یا سه) خطی بود که بسرعت نوشته و اجرا می‌شد؛ اما در روزهای اولیهٔ ویندوز برای نوشتن همین برنامهٔ ساده، برنامه‌نویس مجبور بود دهها (و گاهی صدها) خط کُد بنویسد. با این حال، نوشتن برنامهٔ "Hello World" با ویژوال بیسیک .NET کار چندان دشواری نیست. این برنامه را می‌توان با قرار دادن دو شیء روی یک فرم، ست کردن دو خاصیت، و نوشتن یک خط کُد ایجاد کرد. اجازه دهید دست به کار شویم.

نوشتن برنامهٔ Hello World

۱ ویژوال استودیو .NET را باز کنید.

۲ از منوی File|New آیتم Project را انتخاب کنید، تا پنجرهٔ پروژهٔ جدید باز شود.

نکته

روش زیر یک روش استاندارد برای ایجاد پروژه‌های جدید است، که در آینده نیز می‌توانید از آن استفاده کنید.

۳ در قسمت Project Types پوشهٔ Visual Basic Projects را انتخاب کرده، و در قسمت Templates روی آیکون Windows Application کلیک کنید. با این انتخابها ویژوال استودیو متوجه می‌شود که می‌خواهید یک برنامهٔ ویندوز با ویژوال بیسیک .NET بنویسید.

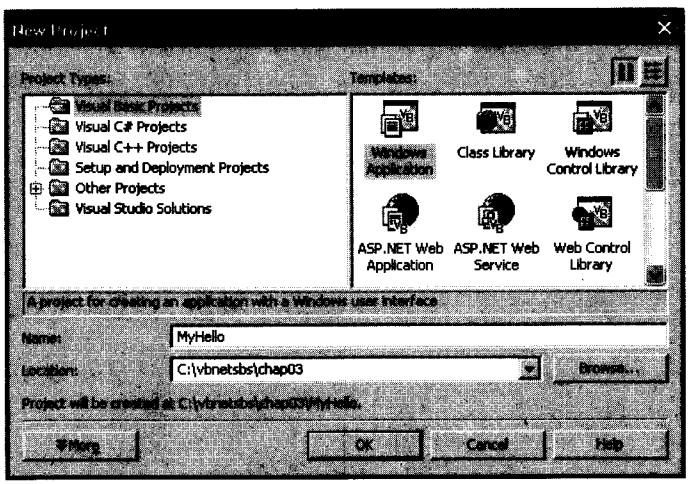
٤ در فیلد Name نام پروژه را (MyHello) وارد کرده، و در قسمت Location محل ذخیره شدن آن را مشخص کنید (برای این کار از دکمهٔ Browse نیز می‌توانید استفاده کنید). توصیه می‌کنم تمرینات این کتاب را در پوشه‌ای مستقل (مثلاً، c:\vbnetsbs) ذخیره کنید - بدین ترتیب می‌توانید هر وقت خواستید همهٔ آنها را یکباره پاک کنید!

نکته

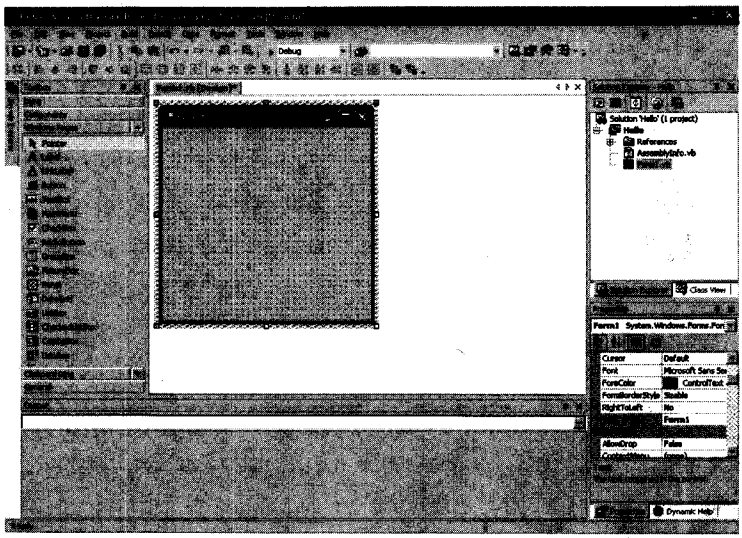
در تمرینات این کتاب از شما خواسته می‌شود تا نام پروژه‌ها را با کلمهٔ My شروع کنید؛ این کار برای آن است که بتوانید فایلهایی را که خودتان ساخته‌اید، و آنهایی که از روی CD کپی کرده‌اید، از یکدیگر تشخیص دهید.

۵ در پنجرهٔ انتخاب محل پروژه، پوشهٔ c:\vbnetsbs\chap03 را انتخاب کنید.

۶ دکمهٔ Open را کلیک کنید؛ با این کار ویژوال استودیو پوشه‌ها و فایلهای موردنیاز پروژه را در پوشه‌ای بنام c:\vbnetsbs\chap03\MyHello خواهد ساخت. تا اینجا پنجرهٔ پروژهٔ جدید باید چیزی شبیه این باشد:

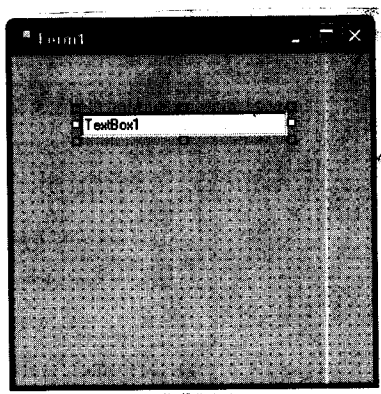


۷ OK را کلیک کنید؛ بلافاصله بعد از این کار، یک فرم خالی در طراح فرمهای ویندوز ظاهر می‌شود (شکل زیر را ببینید). برای برنامهٔ MyHello به یک دکمه و یک جعبه متن نیاز داریم، که باید آنها را از جعبه ابزار ویژوال بیسیک برداریم.



۸ در برگهٔ Windows Forms (فرمهای ویندوز) جعبه ابزار، روی کنترل TextBox (جعبه متن) کلیک کنید.

۹ یک جعبه متن مانند این روی فرم رسم کنید:

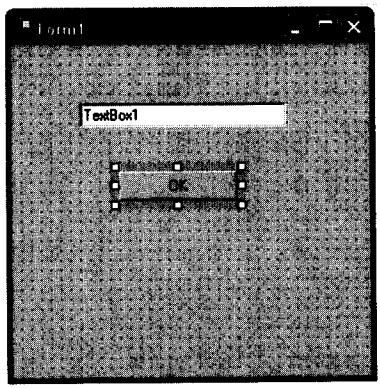


جعبه متن ابزاریست برای نمایش متن در برنامه‌های ویژوال بیسیک. طرز کار این کنترل به نحوهٔ ست شدن خواص، و کُدی که برای آن نوشته‌اید، وابسته است. در این برنامهٔ ساده، وقتی کاربر یک دکمه را کلیک کند، پیام "Hello World" در جعبه متن نمایش داده می‌شود.

بعد از جعبه متن، نوبت اضافه کردن دکمه است.

۱۰ در جعبه ابزار، روی کنترل Button (دکمه) کلیک کنید.

۱۱ یک دکمه مانند این روی فرم رسم کنید:



دکمه ابزاریست برای تعامل بین کاربر و برنامه. وقتی کاربر یک دکمه را کلیک می‌کند، انتظار دارد کاری انجام شود، و یا اتفاقی بیفتد. در زبانهای برنامه‌نویسی ویندوز (که ویژوال بیسیک هم از این دسته است) اصطلاحاً گفته می‌شود که، کاربر با استفاده از دکمه یک رویداد (event) ایجاد کرده، و برنامه باید این رویداد را پردازش کند.

دکمه‌ها کارهای زیادی در برنامه‌های ویندوز انجام می‌دهند: اجرای دستورات کاربر (مانند دکمه‌های OK یا Continue)، لغو دستورات قبلی (مانند دکمه‌های Cancel یا Abort)، و خروج از برنامه (مانند دکمه‌های Exit یا Quit). رفتار دکمه‌ها را هم می‌توان به کمک خواص آنها (و یا از طریق کُد برنامه) کنترل کرد.

۱۲ خواص زیر را برای کنترلهای جعبه متن و دکمه ست کنید:

شیء	خاصیت	مقدار
TextBox1	Text	(خالی)
Button1	Text	"OK"

(دقت کنید که " های اطراف "OK" را نباید در پنجرهٔ خواص وارد کنید.)

۱۳ روی دکمهٔ OK دو-کلیک کنید، تا ادیتور کُد باز شود - دستور زیر را بین دستورات Private Sub Button1_Click و End Sub وارد کنید:

```
TextBox1.Text = "Hello, world!"
```

(متن کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap03\hello بیابید.)

نکته

به محض اینکه نقطهٔ بعد از TextBox1 را وارد کنید، ویژوال استودیو لیست کامل خواص (و متدهای) این شیء را نشانتان خواهد داد، که می‌توانید خاصیت موردنظر را از آن انتخاب کنید (قابل توجه افراد کم‌حافظه!).

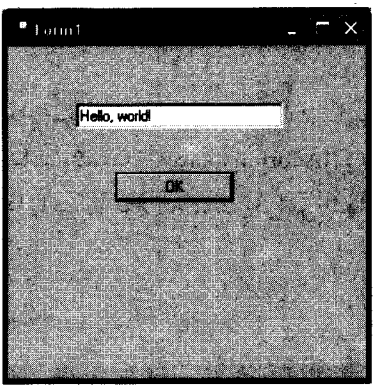
دستوری که نوشتیم، متن TextBox1 را (بعد از کلیک شدن دکمهٔ Button1) به "Hello, world!" تغییر می‌دهد. در واقع، علامت = هر آنچه را که سمت راست آن (و بین دو گیومه) قرار دارد، در خاصیت Text کنترل TextBox1 کپی می‌کند. این کار (تغییر دادن خاصیت یک کنترل در حین اجرای برنامه) یکی از مهمترین مفاهیم برنامه‌نویسی با ویژوال بیسیک است.

بعد از نوشتن برنامهٔ Hello World ، آماده‌ایم تا آنرا اجرا کنیم.

اجرای برنامهٔ Hello World

۱ در میله ابزار استاندارد ویژوال بیسیک، دکمهٔ Start را کلیک کنید، تا ویژوال استودیو برنامه را کامپایل و اجرا کند.

۲ OK را کلیک کنید - بله! پیام "Hello, world!" بلافاصله در جعبه متن ظاهر می‌شود:



(اگر بعد از کلیک کردن OK پیام ظاهر نشد، برگردید و همه چیز را به دقت کنترل کنید. به احتمال زیاد جایی اشتباه کرده‌اید، و یا چیزی را از قلم انداخته‌اید!)

۳ با کلیک کردن دکمهٔ Close (گوشهٔ راست-بالای پنجره)، برنامهٔ MyHello را ببندید.

نکته

برای بستن برنامه‌ای که از محیط ویژوال استودیو اجرا شده، می‌توانید از دکمهٔ Stop Debugging در میله‌ابزار دیباگ ویژوال استودیو نیز استفاده کنید.

٤ با کلیک کردن دکمهٔ Save All در میله ابزار استاندارد، پروژه را ذخیره کنید.

تبریک! شما هم به جرگهٔ برنامه‌نویسانی که با نوشتن Hello World شروع کرده‌اند، ملحق شدید!

کنترل DateTimePicker

برخی از کنترلهای ویژوال بیسیک اطلاعات را نمایش می‌دهند، و برخی دیگر اطلاعات را از کاربر گرفته و پردازش می‌کنند. در تمرین این قسمت با کنترل تاریخ-وقت-گزین (DateTimePicker) کار می‌کنیم؛ این کنترل با یک تقویم گرافیکی به کاربر اجازه می‌دهد تا تاریخ و وقت موردنظرش را انتخاب کند. با آنکه کاری که در این قسمت با کنترل تاریخ-وقت-گزین می‌کنیم، بسیار ساده و ابتدایی است، ولی بخوبی نشان می‌دهد که کنترلهای ویژوال بیسیک تا چه حد می‌توانند کارها را ساده (و خودکار) کنند.

برنامهٔ Birthday

برنامهٔ Birthday با استفاده از یک کنترل تاریخ-وقت-گزین و یک دکمه تاریخ تولد کاربر را گرفته، و آنرا بصورت یک پیام به وی نشان می‌دهد.

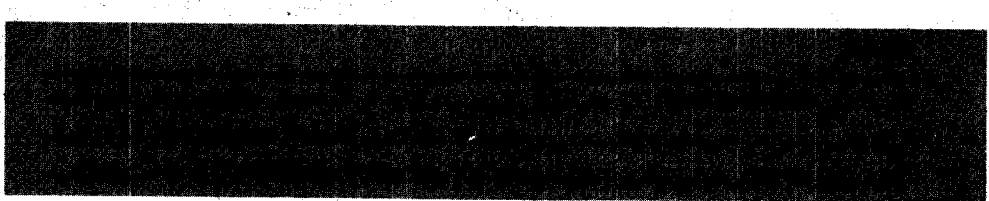
ایجاد برنامهٔ Birthday

۱ با فرمان File|Close Solution پروژهٔ MyHello را ببندید.

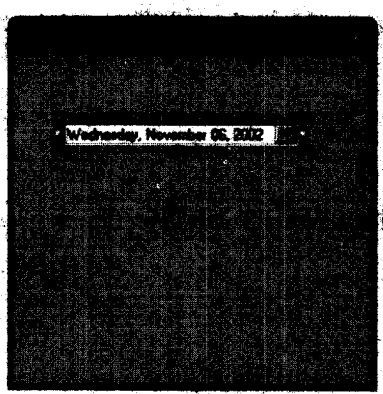
۲ از منوی File|New آیتم Project را انتخاب کنید، تا پنجرهٔ پروژهٔ جدید باز شود.

۳ یک پروژهٔ Visual Basic Windows Application بنام MyBirthday در پوشهٔ c:\vbnetsbs\chap03 ایجاد کنید.

٤ در جعبه ابزار ویژوال بیسیک، روی کنترل DateTimePicker کلیک کنید.



۵ یک شیء تاریخ-وقت-گزین وسط فرم برنامه رسم کنید:



کنترل تاریخ-وقت-گزین بصورت پیش‌فرض تاریخ و وقت فعلی سیستم را نشان می‌دهد، ولی می‌توانید با استفاده از خاصیت Value آنرا تغییر دهید. اندازهٔ این کنترل را هم بدلخواه ست کنید.

۶ در جعبه ابزار Button را انتخاب کرده، و یک دکمه درست زیر کنترل تاریخ-وقت-گزین رسم کنید. این دکمه تاریخ تولد کاربر را نمایش خواهد داد.

۷ خاصیت Text دکمه را (در پنجرهٔ خواص) به **Show My Birthday** تغییر دهید.

برای آنکه این دکمه تاریخ تولد انتخاب شده در کنترل DateTimePicker را نمایش دهد، باید چند خط کُد در روال رویداد کلیک آن بنویسیم.

۸ روی دکمهٔ Show My Birthday دو-کلیک کنید، تا ادیتور کُد باز شود - سپس، دستورات زیر را بین Private Sub Button1_Click و End Sub وارد کنید:

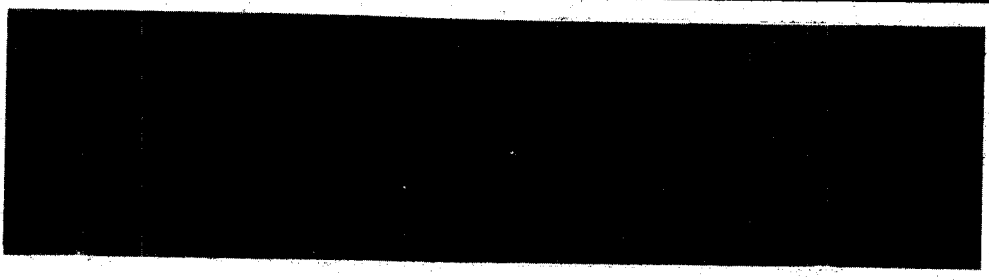
```
MsgBox("Your birth date was " & DateTimePicker1.Text)
MsgBox("Day of the year: " & _
        DateTimePicker1.Value.DayOfYear.ToString())
MsgBox("Today is " & DateTimePicker1.Value.Now.ToString())
```

این دستورات سه جعبه پیام (message box) را پشت سر هم نشان می‌دهند. هر یک از این پیامها اطلاعات خود را بنحوی از کنترل تاریخ-وقت-گزین می‌گیرند. خط اول مقدار خاصیت Text کنترل DateTimePicker1 را (که کاربر بعنوان تاریخ تولدش انتخاب کرده) نشان می‌دهد. تابع MsgBox این مقدار را با عبارت "Your birth date was" ترکیب می‌کند؛ دقت کنید که چگونه برای اینکار از عملگر ترکیب رشته (&) استفاده کرده‌ایم.

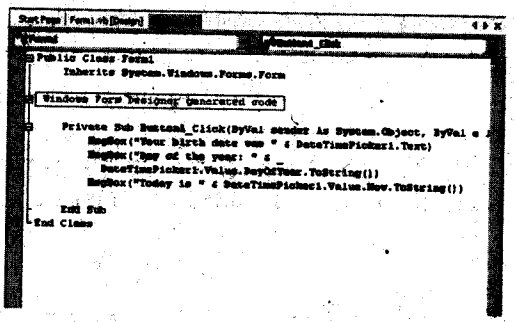
خط‌های دوم و سوم در واقع یک دستور هستند، که بعلت طول زیاد با استفاده از کاراکتر ادامهٔ خط (_) به هم چسبانده شده‌اند. دستور DateTimePicker1.Value.DayOfYear یکی از خاصیتهای داخلی کنترل DateTimePicker است، که نشان می‌دهد کاربر ما در چندمین روز سال بدنیا آمده است (روز اول سال، اول ژانویه است). از آنجائیکه این خاصیت یک عدد است، بکمک متد ToString آنرا به متن تبدیل کرده‌ایم، تا تابع MsgBox بتواند آنرا نمایش دهد.

هر شیء، علاوه بر خواصی که قبلاً دیدید، دارای تعدادی متد (method) نیز هست که عملی را انجام می‌دهند، یا سرویسی را در اختیار کاربر می‌گذارند. در واقع، متد یک عمل است، در حالیکه خاصیت یک مقدار است. متد با روال رویداد نیز فرق دارد، چون روال رویداد در پاسخ به وقوع یک رویداد اجرا می‌شود، در حالیکه اجرای متد دارای چنین شرطی نیست.

خط چهارم برنامه تاریخ فعلی سیستم را با استفاده از خاصیت DateTimePicker1.Value.Now استخراج کرده، و بعد از تبدیل آن به متن (با متد ToString) نمایش می‌دهد.



تا اینجا ادیتور کُد باید چیزی شبیه شکل زیر باشد:



(متن کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap03\birthday پیدا کنید.)

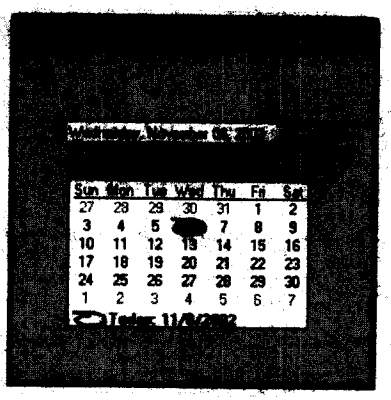
۹ باکلیک کردن دکمهٔ Save All ، پروژه را ذخیره کنید.

بعد از نوشتن برنامهٔ Birthday ، آماده‌ایم آنرا اجرا کنیم.

اجرای برنامهٔ Birthday

۱ دکمهٔ Start را در میله ابزار استاندارد کلیک کنید، تا ویژوال استودیو برنامه را کامپایل و اجرا کند.

۲ در کنترل DateTimePicker روی آیکون ▼ کلیک کنید، تا تقویم گرافیکی آن باز شود (شکل زیر را ببینید).



۳ برای رفتن به ماه قبل در تقویم، روی دکمهٔ ◄ (بالا، سمت چپ) کلیک کنید - با این کار تقویم ماه قبل را خواهید دید؛ اما قسمت Today عوض نمی‌شود، و همچنان تاریخ روز جاری را نشان می‌دهد.

با اینکه می‌توانید ماه به ماه عقب بروید، تا به ماه تولد خود برسید، اما این کار حوصلهٔ زیادی می‌خواهد - چیزی که این روزها کمیاب است! اما، راه آسانتری هم وجود دارد.

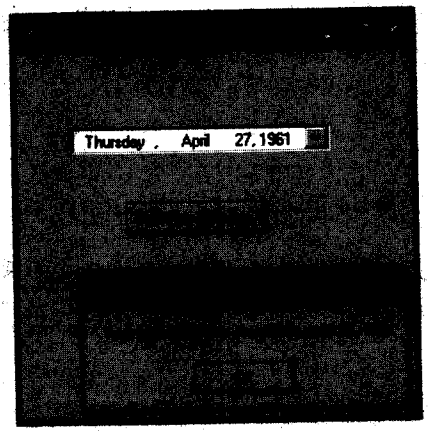
٤ در قسمت متن کنترل تاریخ-وقت-گزین، عدد چهار رقمی سال را انتخاب کنید (با این کار تقویم بسته می‌شود).

۵ سال تولد خود را وارد کرده، و دوباره دکمهٔ ▼ را کلیک کنید، تا تقویم باز شود.

۶ با دکمه‌های ◄ (ماه قبل) یا ► (ماه بعد) به ماه تولد خود بروید، و روی روزی که بدنیا آمده‌اید، کلیک کنید. اگر تا حالا نمی‌دانستید کدام روز هفته بدنیا آمده‌اید، اکنون دیگر می‌دانید.

با انتخاب روز تولد، تقویم بسته می‌شود، و تاریخ دقیق تولد خود را در جعبه متن کنترل DateTimePicker می‌بینید. حالا وقت آنست که دکمهٔ Show My Birthday را کلیک کنید، و ببینید چه اتفاقی می‌افتد.

۷ روی دکمهٔ Show My Birthday کلیک کنید. اولین جعبه پیامی که ویژوال بیسیک نمایش می‌دهد، سال، ماه و روز تولد شما را نشان می‌دهد:



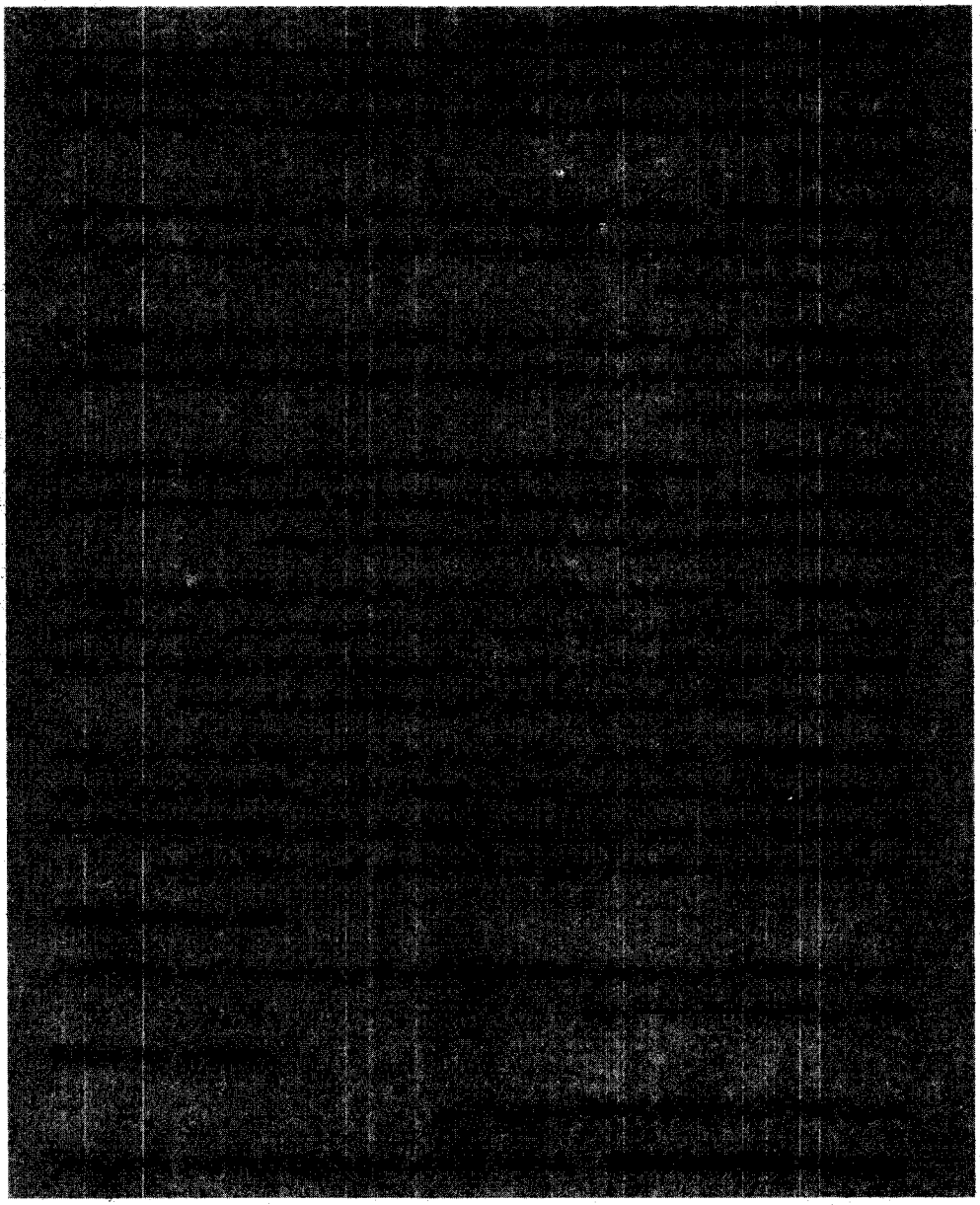
۸ OK را کلیک کنید، تا پیام دوم ظاهر شود. این پیام نشان می‌دهد که در چندمین روز از سال بدنیا آمده‌اید.

۹ باز هم OK را کلیک کنید، تا پیام آخر - تاریخ و وقت فعلی - ظاهر شود. بله، برنامه بخوبی کار می‌کند!

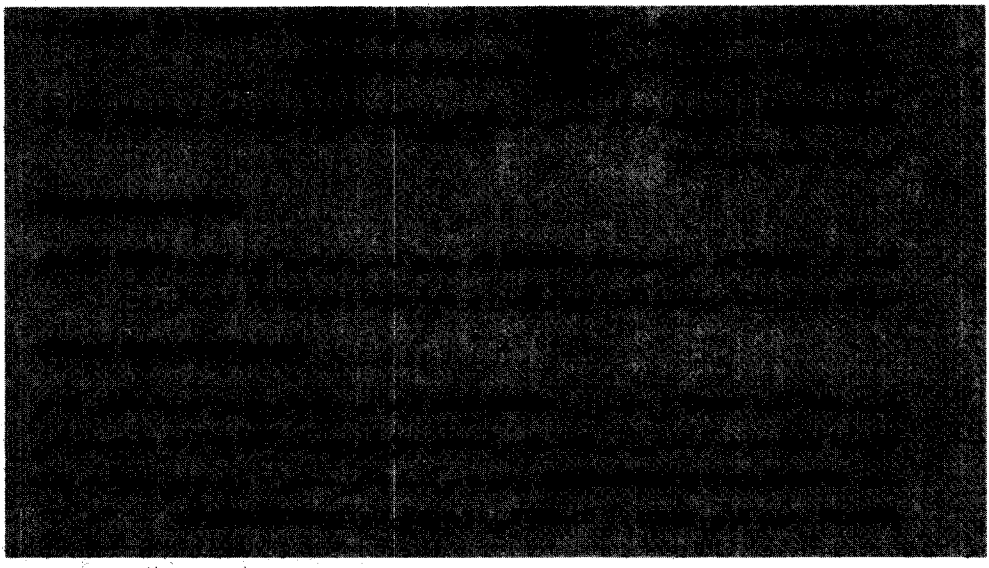
همانطور که می‌بینید، کنترل تاریخ-وقت-گزین در نمایش تاریخ و وقت (حال، گذشته و حتی آینده) بسیار قابل است.

۱۰ باکلیک کردن OK ، پیام سوم را ببندید؛ و بعد از آنکه کمی دیگر با برنامه کار کردید، آنرا هم ببندید.

کار باکنترل DateTimePicker فعلاً کافیست!



ادامه در صفحه بعد



کنترل‌هایی برای گرفتن اطلاعات از کاربر

ویژوال بیسیک مکانیسمهای متعددی برای دریافت اطلاعات از کاربر در اختیار برنامه‌نویس گذاشته است. جعبه متن می‌تواند متن ورودی کاربر را بگیرد، منو مقاصد کاربر را بصورت کلیک‌های ماوس دریافت می‌کند، و دیالوگ نیز با وسایل متنوع این اطلاعات را از وی می‌گیرد. در این قسمت با چهار تا از کنترلهای ورودی ویژوال بیسیک، و کارکرد آنها در موقعیتهای مختلف، آشنا می‌شوید: دکمهٔ رادیویی (RadioButton)، جعبه چک (CheckBox)، جعبه لیست (ListBox)، و جعبه ترکیبی (ComboBox). برنامه‌ای که در این قسمت می‌نویسیم، یک فرم سفارش کالاست، و در آن از انواع کنترلهای ورودی استفاده خواهیم کرد. (با یکی دیگر از کنترلهای ورودی، یعنی منو، در فصل آینده آشنا خواهید شد.)

برنامه را با طرز استفاده از جعبه چک شروع می‌کنیم.

طرز کار با کنترل جعبه چک

۱ فرمان Close Solution را از منوی File انتخاب کنید، تا پروژهٔ Birthday بسته شود.

۲ از منوی File|New آیتم Project را انتخاب کنید، تا پنجرهٔ پروژهٔ جدید باز شود.

۳ یک پروژهٔ Visual Basic Windows Application بنام MyCheckBox در پوشهٔ c:\vbnetsbs\chap03 ایجاد کنید.

٤ در جعبه ابزار ویژوال بیسیک، روی کنترل CheckBox کلیک کنید.

۵ دو جعبه چک (زیر یکدیگر) روی فرم برنامه رسم کنید.

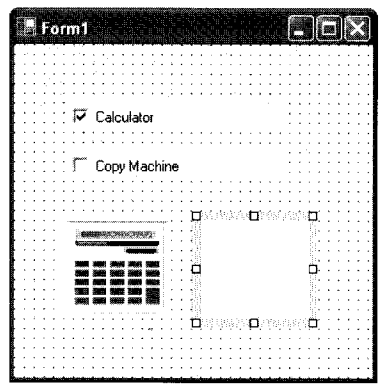
۶ کنترل PictureBox (جعبه تصویر) را انتخاب کرده، و دو جعبه تصویر (زیر جعبه چک‌ها) رسم کنید.

۷ خواص این چهار شیء را مطابق جدول زیر ست کنید:

شیء	خاصیت	مقدار
CheckBox1	Checked	True
	Text	"Calculator"
CheckBox2	Text	"Copy machine"
PictureBox1	Image	"x:\vbnetsbs\chap03\calcultr.bmp"
	SizeMode	StretchImage
PictureBox2	SizeMode	StretchImage

در این مثال، از جعبه چک‌ها برای ظاهر یا مخفی کردن تصویرها استفاده می‌کنیم. خاصیت Text جعبه چک آن چیزیست که کاربر در کنار هر جعبه چک می‌بیند؛ خاصیت Checked نیز مقدار پیش‌فرض هر جعبه چک را (در شروع برنامه) ست می‌کند -اگر این خاصیت مقدار True داشته باشد، جعبه چک علامت چک (✓) خواهد داشت، و اگر False باشد (مقدار پیش‌فرض)، خیر.

تا اینجا، فرم برنامه باید چیزی شبیه شکل زیر باشد:



۸ روی جعبه چک اول دو-کلیک کنید، تا روال رویداد CheckBox1_CheckedChanged در ادیتور کُد باز شود - دستورات زیر را بین Private Sub CheckBox1_CheckedChanged و End Sub وارد کنید:

```
If CheckBox1.CheckState = 1 Then
    PictureBox1.Image = System.Drawing.Image.FromFile _
        ("c:\vbnetsbs\chap03\calcultr.bmp")
    PictureBox1.Visible = True
Else
    PictureBox1.Visible = False
End If
```

روال CheckBox1_CheckedChanged فقط وقتی اجرا می‌شود که جعبه چک CheckBox1 کلیک شود. در این روال برای تعیین وضعیت CheckBox1 از یک بلوک If...Then استفاده کرده‌ایم (در

فصل ۶ با این دستور بیشتر آشنا خواهید شد). اگر این جعبه چک دارای علامت چک باشد (CheckBox1.CheckState = 1) ، تصویر ماشین حساب (جعبه تصویر PictureBox1) نمایش داده می‌شود (خاصیت Visible). اگر خاصیت CheckState مقدار 1 داشته باشد، یعنی علامت چک وجود دارد، و اگر 0 باشد، یعنی این علامت وجود ندارد.

۹ با کلیک کردن View Designer در کاوشگر راه‌حل، به فرم برنامه برگردید، و این بار روی جعبه چک دوم دو-کلیک کنید - دستورات زیر را بین Private Sub CheckBox2_CheckedChanged و End Sub وارد کنید:

```
If CheckBox2.CheckState = 1 Then
    PictureBox2.Image = System.Drawing.Image.FromFile _
        ("c:\vbnetsbs\chap03\copymach.bmp")
    PictureBox2.Visible = True
Else
    PictureBox2.Visible = False
End If
```

این روال بسیار شبیه CheckBox1_CheckedChanged است، با این تفاوت که با PictureBox2 سروکار دارد.

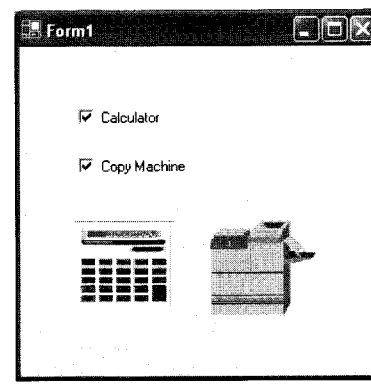
(متن کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap03\checkbox پیدا کنید.)

۱۰ با کلیک کردن دکمهٔ Save All ، پروژه را ذخیره کنید.

اجرای برنامهٔ CheckBox

۱ دکمهٔ Start را در میله ابزار استاندارد کلیک کنید، تا ویژوال استودیو برنامه را کامپایل و اجرا کند. از آنجائیکه در شروع برنامه فقط کنترل CheckBox1 دارای علامت چک است، تصویر ماشین حساب روی فرم دیده می‌شود.

۲ روی جعبه چک Copy machine کلیک کنید؛ اکنون ویژوال بیسیک هر دو تصویر را به نمایش می‌گذارد.



۳ کمی با این دو کنترل (و حالتهای مختلف آنها) کار کنید، تا مطمئن شوید برنامه در تمام شرایط درست کار می‌کند.

٤ با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

برنامهٔ نمایشی Input Controls

حال که با یکی از کنترلهای ورودی و طرز کار آن آشنا شدید، به سراغ برنامهٔ سفارش کالا که بنحو وسیعتری از این کنترلها استفاده می‌کند، می‌رویم. برنامهٔ Input Controls یک برنامهٔ ساده است، که براحتی می‌توانید آنرا به یک برنامهٔ واقعی و عملیاتی تبدیل کنید.

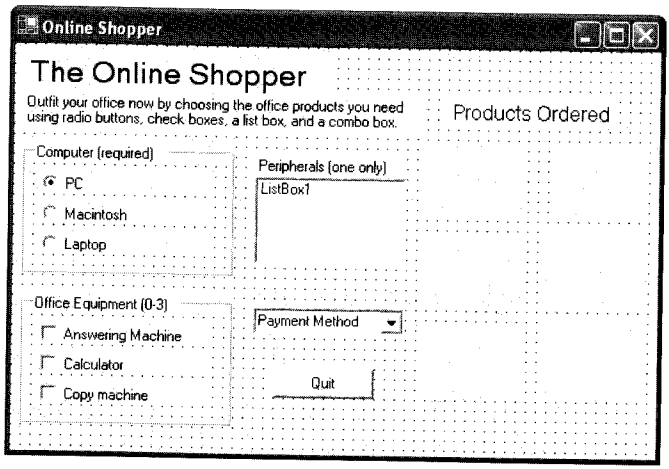
طرز کار باکنترل جعبه چک

۱ از منوی File|Open آیتم Project را انتخاب کنید، تا پنجرهٔ بازکردن پروژه ظاهر شود.

۲ پوشهٔ c:\vbnetsbs\chap03\input controls را باز کرده، و روی فایل پروژهٔ Input Controls (فایل Input Controls.vbproj) دو-کلیک کنید، تا این پروژه در محیط برنامه‌نویسی ویژوال بیسیک باز شود.

۳ اگر فرم پروژه را نمی‌بینید، روی فرم Form1.vb در کاوشگر راه‌حل کلیک کنید.

٤ برای بهتر دیدن فرم پروژه (شکل زیر)، پنجره‌های اضافی را ببندید.



در این فرم تمام کنترلهای دکمه، برچسب، جعبه تصویر، دکمهٔ رادیویی، جعبه چک، جعبه لیست و جعبه ترکیبی را می‌بینید؛ این کنترلها با هم کار می‌کنند تا یک فرم سفارش کالا بسازند. کار اصلی برنامهٔ Input Controls نمایش شش تصویر (که همگی در پوشهٔ c:\vbnetsbs\chap03\input controls قرار دارند) می‌باشد.

نکته

اگر فایلهای CD ضمیمه را در محل دیگری نصب کرده‌اید، باید تعدادی از دستورات برنامه را تغییر دهید، تا اجرای آن با خطا مواجه نشود. این دستورات که همگی با c:\vbnetsbs\chap03\input controls شروع می‌شوند، به محل فایلهای موردنیاز برنامه اشاره می‌کنند. برای تغییر این عبارات می‌توانید از فرمان Find And Replace در منوی Edit استفاده کنید.

۵ دکمهٔ Start را کلیک کنید، تا برنامه کامپایل و اجرا شود.

۶ در کادر Computer ، روی دکمهٔ رادیویی Laptop کلیک کنید؛ با این کار تصویر یک کامپیوتر کیفی در قسمت Products Ordered ظاهر می‌شود. در این کادر سه دکمهٔ رادیویی می‌بینید، که می‌توانید هر یک از آنها را انتخاب کنید؛ توجه داشته باشید که، از هر دسته دکمهٔ رادیویی در هر لحظه یکی - و فقط یکی - می‌تواند انتخاب شود. (در ویژوال بیسیک ۶ به دکمهٔ رادیویی دکمهٔ گزینه - OptionButton - گفته می‌شد.)

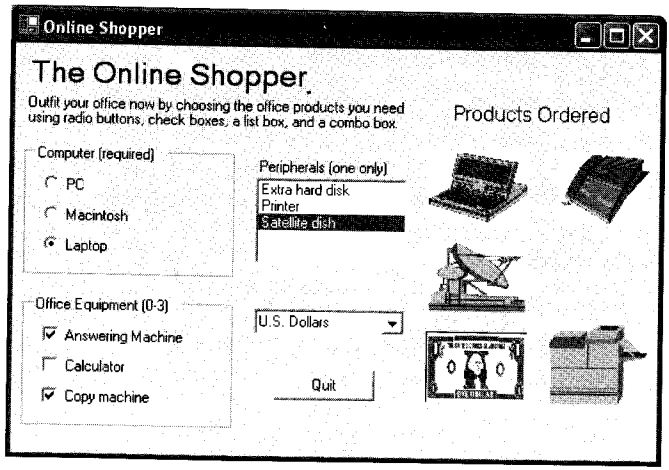
وقتی چند دکمهٔ رادیویی در یک جعبهٔ گروهی (که در ویژوال بیسیک ۶ فریم - Frame - نامیده می‌شد) قرار داده می‌شوند، تشکیل یک گروه منطقی را می‌دهند، و در هر لحظه فقط یکی از آنها را می‌توان انتخاب کرد. کنترل جعبهٔ گروهی (GroupBox) نیز در برگهٔ Windows Forms جعبه ابزار قرار دارد، و خاصیت Text آن عبارتیست که در بالای این کادر دیده می‌شود. وقتی یک جعبه گروهی را جابجا می‌کنید، تمام کنترلهایی که داخل آن هستند، نیز جابجا می‌شوند.

۷ در کادر Office Equipment ، روی جعبه چک‌های Answering Machine ، Calculator و Copy Machine کلیک کنید؛ از هر دسته جعبه چک می‌توانید هر تعداد را که می‌خواهید، انتخاب کنید. دقت کنید که با فعال یا غیرفعال کردن آیتمها، تصاویر قسمت Products Ordered نیز ظاهر یا مخفی می‌شوند.

۸ در لیست Peripherals روی آیتم Sattelite Dish کلیک کنید، تا تصویر یک آنتن ماهواره در قسمت Products Ordered ظاهر شود. جعبه لیست (مانند یک گروه دکمهٔ رادیویی) به کاربر اجازه می‌دهد تا آیتم موردنظر را از یک لیست انتخاب کند (اما برخلاف دکمه‌های رادیویی نیازی نیست که حتماً یکی را انتخاب کند). یکی از ویژگیهای مهم جعبه لیست اینست که، آیتمهای جعبه لیست را می‌توان در زمان اجرای برنامه به آن اضافه کرد. اگر میل دارید که کنار آیتم انتخاب شده در لیست یک علامت چک نیز دیده شود، می‌توانید بجای جعبه لیست معمولی از جعبه لیست چک‌دار (CheckedListBox) استفاده کنید.

۹ از لیست Payment (زیر لیست Peripherals) آیتم U.S. Dollars را انتخاب کنید - این لیست یک جعبه ترکیبی است. جعبه ترکیبی، در واقع، ترکیبیست از یک جعبه متن و یک جعبه لیست بازشو (drop-down). این کنترل بسیار شبیه جعبه لیست است، با این تفاوت که جای کمتری اشغال می‌کند، و آیتم انتخاب شده را هم می‌توان دستکاری کرد.

با انتخابهایی که تا اینجا کرده‌اید، فرم برنامه باید چیزی شبیه شکل زیر شده باشد:



۱۰ کمی دیگر با برنامه کار کنید، و در پایان با کلیک کردن دکمهٔ Quit آنرا ببندید.

نگاهی به کُد برنامهٔ Input Controls

با اینکه هنوز چیز زیادی دربارهٔ کُد و کُدنویسی نمی‌دانید، با این حال یک نگاه سریع به کُد برنامهٔ Input Controls می‌تواند مفید باشد. در این برنامه دستورات زیادی می‌بینید، که بعداً دربارهٔ آنها توضیح خواهم داد، ولی اجازه دهید در اینجا کمی بیشتر دربارهٔ دو خاصیت CheckState و SelectedIndex (که اولی مربوط به جعبه چک، و دومی مربوط به جعبه لیست است) صحبت کنم.

بررسی کُد جعبه چک و جعبه لیست

۱ در کادر Office Equipment روی جعبه چک Answering Machine دو-کلیک کنید، تا روال رویداد CheckBox1_CheckedChanged در ادیتور کُد باز شود. کُد این روال چنین است:

```
'If the CheckState property for a check box is 1, it has a mark in it
If CheckBox1.CheckState = 1 Then
    PictureBox2.Image = System.Drawing.Image.FromFile _
      ("c:\vbnetsbs\chap03\input controls\answmach.bmp")
    PictureBox2.Visible = True
Else
    'If there is no mark, hide the image
    PictureBox2.Visible = False
End If
```

اولین خط این روال یک توضیح است. توضیحات برنامه در ادیتور کُد به رنگ سبز دیده می‌شوند، و علیرغم آن که کاری انجام نمی‌دهند، بسیار مهم هستند. این خط توضیح می‌دهد که یک جعبه چک در چه حالتی دارای علامت چک است. بقیهٔ دستورات بسیار شبیه کُدی هستند که برای برنامهٔ CheckBox نوشتیم. اگر کمی در ادیتور کُد پائین بروید، می‌بینید که مشابه همین کُد برای

جعبه چک‌های CheckBox2 و CheckBox3 نیز وجود دارد.

۲ با کلیک کردن روی برگهٔ "Form1.vb [Design]" در بالای ادیتور کُد، به فرم برنامه برگردید. روی جعبه لیست Peripherals دو-کلیک کنید، تا روال رویداد ListBox1_SelectedIndexChanged در ادیتور کُد باز شود. کُد این روال چنین است:

```
'The item you picked (0-2) is held in the SelectedIndex property
Select Case ListBox1.SelectedIndex
   Case 0
      PictureBox3.Image = System.Drawing.Image.FromFile _
        ("c:\vbnetsbs\chap03\input controls\harddisk.bmp")
   Case 1
      PictureBox3.Image = System.Drawing.Image.FromFile _
        ("c:\vbnetsbs\chap03\input controls\printer.bmp")
   Case 2
      PictureBox3.Image = System.Drawing.Image.FromFile _
        ("c:\vbnetsbs\chap03\input controls\satedish.bmp")
End Select
```

این کُد وقتی اجرا می‌شود که کاربر روی یکی از آیتمهای لیست Peripherals کلیک کند. در این کُد خط دوم بسیار مهم است: وقتی کاربر روی یکی از آیتمهای لیست کلیک می‌کند، خاصیت SelectedIndex جعبه لیست مقدار متناظر با محل آن آیتم در لیست را برمی‌گرداند. (شماره گذاری آیتمهای لیست از 0 شروع می‌شود، یعنی اولین آیتم 0 است، دومی 1 و الی‌آخر.)

در این روال برای تعیین تصویر آیتمی که کاربر روی آن کلیک کرده، از یک دستور Select Case استفاده کرده‌ایم. (در فصل ۶ با دستور Select Case بیشتر آشنا خواهید شد.) این دستور مقدار خاصیت SelectedIndex را خوانده، و بر حسب آن تصویر مربوطه را بار می‌کند.

۳ با کلیک کردن روی برگهٔ "Form1.vb [Design]" در بالای ادیتور کُد، به فرم برنامه برگردید. روی خود فرم (نه هیچیک از اشیاء روی آن) دو-کلیک کنید، تا روال رویداد Form1_Load در ادیتور کُد باز شود. روال Form1_Load هر بار که برنامه شروع (بار) می‌شود، اجرا خواهد شد. اغلب برنامه‌نویسان از این روال برای اجرای دستوراتی که در شروع برنامه بایستی اجرا شوند، استفاده می‌کنند. (یک برنامه می‌تواند بیش از یک فرم داشته باشد، ولی ویژوال بیسیک فقط روال Form1_Load - فرم اول برنامه - را در شروع اجرا خواهد کرد.) دستوراتی که معمولاً در روال Form1_Load اجرا می‌شوند، آنهایی هستند که نمی‌توان از طریق پنجرهٔ خواص یا جعبه ابزار انجام داد. در زیر روال Form1_Load برنامهٔ Input Controls را ملاحظه می‌کنید:

```
'These program statements run when the form loads
PictureBox1.Image = System.Drawing.Image.FromFile _
   ("c:\vbnetsbs\chap03\input controls\pcomputr.bmp")
'Add items to a list box like this:
ListBox1.Items.Add("Extra hard disk")
ListBox1.Items.Add("Printer")
```

```
ListBox1.Items.Add("Satellite dish")
'Combo boxes are also filled with the Add method:
ComboBox1.Items.Add("U.S. Dollars")
ComboBox1.Items.Add("Check")
ComboBox1.Items.Add("English Pounds")
```

در این روال سه خط توضیح می‌بینید: خطوط ۱، ۴ و ۸ خط دوم (و سوم) تصویر کامپیوتر کیفی را در جعبه تصویر بار می‌کنند. دستورات خطوط پنجم تا هفتم آیتمهای جعبه لیست Peripherals را در آن بار می‌کنند؛ خطوط نهم تا یازدهم همین کار را برای جعبه ترکیبی Payment انجام می‌دهند. همانطور که می‌بینید، برای این کار از متد Add کنترلهای مزبور استفاده کرده‌ایم.

طرز کار باکنترل LinkLabel

دسترسی وب به یکی از ویژگیهای استاندارد در تمام برنامه‌های ویندوز تبدیل شده، و این کار در ویژوال استودیو .NET حتی از قبل نیز ساده‌تر شده است. با تکنولوژی جدید **فرمهای وب** (Web Forms) در ویژوال استودیو می‌توانید براحتی برنامه‌های وب-آگاه (Web-aware) بنویسید، و یا اگر فقط می‌خواهید در کاوشگر وب یک صفحهٔ وب باز کنید، این کار را بسادگی در برنامهٔ خود انجام دهید.

در این قسمت خواهید دید که چگونه می‌توان لینکهای وب را روی فرمهای ویژوال بیسیک قرار داد. کنترل جدید برچسب-لینک (LinkLabel) ، وقتی در کنار متد Process.Start بکار گرفته شود، می‌تواند یک صفحهٔ وب را در هر کاوشگری (از قبیل Internet Explorer یا Netscape Navigator) باز کند.

نکته

فصلهای ۲۱ و ۲۲ این کتاب به نوشتن برنامه‌های وب-آگاه با ویژوال بیسیک .NET اختصاص داده شده است.

ایجاد برنامهٔ WebLink

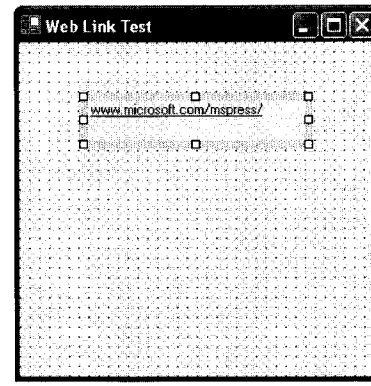
۱ با اجرای فرمان File|Close Solution ، پروژهٔ Input Controls را ببندید.

۲ از منوی File|New آیتم Project را انتخاب کنید، تا پنجرهٔ پروژهٔ جدید باز شود.

۳ در پوشهٔ c:\vbnetsbs\chap03 یک پروژهٔ Visual Basic Windows Application جدید بنام MyWebLink ایجاد کنید.

٤ از جعبه ابزار ویژوال بیسیک، کنترل LinkLabel را انتخاب کرده، و یک برچسب-لینک روی فرم برنامه رسم کنید. کنترل برچسب-لینک بسیار شبیه برچسب‌های معمولیست، با این تفاوت که متن آن بصورت لینک (با رنگ آبی و زیرخط) دیده خواهد شد.

۵ خاصیت Text این کنترل را به **www.microsoft.com/mspress/** - سایت وب شرکت انتشارات میکروسافت - ست کنید (شکل زیر را ببینید).



۶ روی فرم برنامه کلیک کرده، و آنرا انتخاب کنید؛ بدین ترتیب می‌توانید خواص آنرا در پنجرهٔ خواص ببینید.

۷ خاصیت Text فرم را به **Web Link Test** ست کنید (این خاصیت در ویرایشهای قبلی ویژوال بیسیک Caption نام داشت). عبارت مزبور در **میلهٔ عنوان** فرم (مستطیل آبی رنگ بالای فرم) نشان داده خواهد شد.

۸ روی کنترل برچسب-لینک دو-کلیک کنید، تا روال رویداد LinkLabel1_LinkClicked در ادیتور کُد باز شود - دستورات زیر را در این روال وارد کنید:

```
' Change the color of the link by setting LinkVisited to True.
LinkLabel1.LinkVisited = True
' Use the Process.Start method to open the default broswer
' using the Microsoft Press URL:
System.Diagnostics.Process.Start _
  ("http://www.microsoft.com/mspress/")
```

در این قطعه کُد هم چند خط توضیح می‌بینید، که دستورات دیگر را تشریح می‌کنند. در واقع، این روال فقط دو دستور اجرایی دارد. دستور اول با استفاده از خاصیت LinkVisited رنگ لینک را تغییر می‌دهد، تا رفتار کاوشگر را شبیه‌سازی کرده باشد (وقتی در یک کاوشگر وب روی لینکی کلیک می‌کنید، رنگ آن تغییر می‌کند تا نشان دهد قبلاً این لینک را بازدید کرده‌اید). این دستور بهیچوجه الزامی نیست، اما وجود آن ظاهری حرفه‌ای به برنامه می‌دهد.

دستور دوم با استفاده از متد Start کلاس Process (در کتابخانهٔ System.Diagnostics) کاوشگر وب پیش‌فرض سیستم را راه‌اندازی کرده، و صفحهٔ وب مشخص شده را در آن بار می‌کند. یکی از ویژگیهای جالب متد Process.Start اینست که می‌تواند هر برنامه‌ای را اجرا کند. برای مثال، اگر بخواهید صفحهٔ وب موردنظر را در یک کاوشگر خاص (مانند Internet Explorer) باز کنید، می‌توانید از دستور ذیل استفاده کنید:

```
System.Diagnostics.Process.Start ("IExplore.exe", _
  "http://www.microsoft.com/mspress/")
```

آرگومان اول متد Process.Start نام برنامه‌ایست که باید اجرا شود؛ وقتی مسیر کامل برنامه را وارد نمی‌کنید، ویژوال بیسیک در مسیرهای جاری سیستم بدنبال آن خواهد گشت. در دستور زیر نیز، با استفاده از متد Start ، فایل c:\myletter.doc در برنامهٔ Microsoft Word باز شده است:

```
System.Diagnostics.Process.Start ("Winword.exe", _
  "c:\myletter.doc")
```

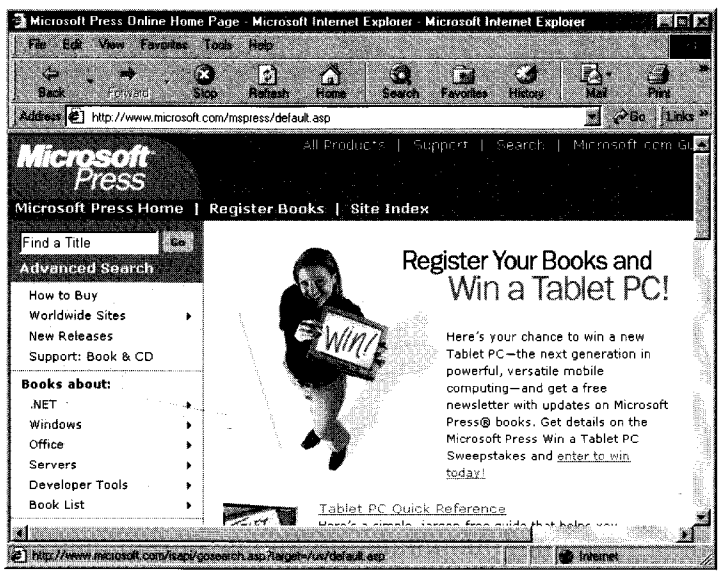
همانطور که می‌بینید، متد Process.Start متدی بسیار مفید است، و در این کتاب به کرات از آن استفاده خواهیم کرد.

۹ با کلیک کردن دکمهٔ Save All ، پروژه را ذخیره کنید. (متن کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap03\weblink پیدا کنید.)

اجرای برنامهٔ WebLink

۱ دکمهٔ Start را در میله ابزار استاندارد کلیک کنید، تا ویژوال استودیو برنامه را کامپایل و اجرا کند.

۲ روی لینک موجود در کنترل برچسب-لینک کلیک کنید، تا کاوشگر وب پیش‌فرض سیستم اجرا شده و صفحهٔ www.microsoft.com/mspress/ باز شود؛ شکل زیر را ببینید:



توجه کنید که یکسان بودن متنی که در برچسب-لینک دیده می‌شود، با آدرس URL صفحهٔ وب موردنظر در این مثال، یک تصادف محض است. در واقع متن کنترل برچسب-لینک می‌تواند هر چیزی (حتی یک تصویر) باشد.

۳ به فرم برنامه برگردید. دقت کنید که رنگ لینک تغییر کرده، و این نشان می‌دهد که لینک www.microsoft.com/mspress/ قبلاً (حداقل یک بار) بازدید شده است.

٤ با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

یک گام فراتر: نصب کنترل‌های اکتیوایکس

ویژوال استودیو .NET یک محصول جدید است، و اگر با ویرایشهای قبلی ویژوال بیسیک کار کرده باشید، حتماً متوجه شده‌اید که تعداد کنترلهای موجود در جعبه ابزار آن کمتر از قبلی‌هاست. برای توسعهٔ جعبه ابزار ویژوال استودیو .NET می‌توانید از کنترلهای قدیمی \اکتیوایکس (ActiveX) استفاده کنید. این کنترلها را در برنامه‌های بسیاری (از قبیل ویژوال بیسیک ۶، ویژوال سی++ ۶ و میکروسافت آفیس) دیده‌اید. برای استفاده از این کنترلها، ویژوال استودیو .NET باید آنها را در لفاف خاصی بپوشاند (و البته در اغلب موارد قادر است این کار را بدون دخالت شما انجام دهد).

نکته

کنترلهای اکتیوایکس معمولاً دارای پسوند ocx. یا dll. هستند، و در دایرکتوری Windows\ یا Windows\System\ نصب می‌شوند. هر برنامه‌ای می‌تواند از کنترلهای اکتیوایکس موجود در یک سیستم استفاده کند، مشروط باینکه بداند چگونه. ویژوال استودیو .NET قادر است کنترلهای اکتیوایکسی را که در رجیستری سیستم ثبت می‌شوند، شناسایی کرده و از آنها استفاده کند.

کنترل Microsoft Chart

به احتمال زیاد همین حالا هم تعداد زیادی کنترل اکتیوایکس در سیستم خود دارید، بویژه اگر ویرایش قبلی ویژوال بیسیک یا میکروسافت آفیس را داشته باشید. اجازه دهید یکی از این کنترلهای اکتیوایکس را در ویژوال استودیو .NET نصب کنیم، حتی اگر فعلاً طرز استفاده از آن را ندانیم (در این قسمت فقط می‌خواهم طرز نصب کنترلهای اکتیوایکس را نشان دهم، بعداً به طرز استفاده از آنها هم خواهم پرداخت). کنترلی که در این قسمت نصب می‌کنیم Microsoft Chart (چارت) نام دارد، که اجازه می‌دهد روی فرم برنامه نمودارهای گرافیکی رسم کنید. این کنترل در تمام ویرایشهای میکروسافت آفیس موجود است.

نصب کنترل Chart

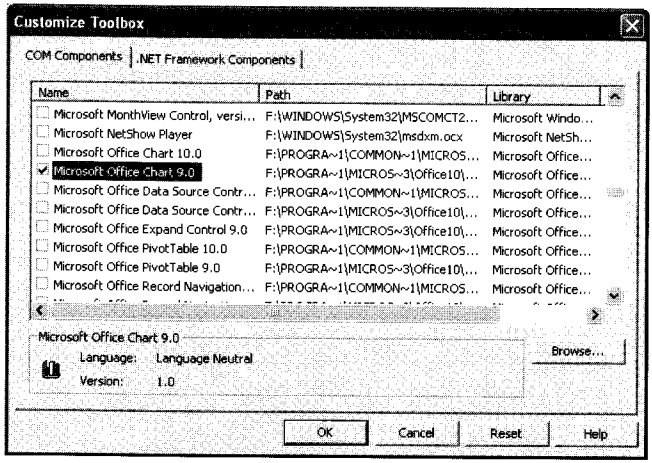
۱ اگر در ادیتور کُد هستید، آنرا ببندید، و به فرم برنامهٔ WebLink برگردید.

۲ در جعبه ابزار، روی برگه‌ای که می‌خواهید کنترل اکتیوایکس به آن اضافه شود (مثلاً، برگهٔ Windows Forms) کلیک کنید. وقتی یک کنترل اکتیوایکس به جعبه ابزار اضافه شود، درست مثل سایر کنترلهای ذاتی ویژوال بیسیک .NET عمل خواهد کرد.

۳ روی برگهٔ موردنظر راست-کلیک کرده، و از منویی که ظاهر می‌شود، آیتم Custmize Toolbox را انتخاب کنید، تا پنجرهٔ Customize Toolbox باز شود.

٤ در این پنجره، روی برگهٔ COM Components کلیک کنید. با این کار لیستی از کنترلهای اکتیوایکس موجود در سیستم را (بر حسب حروف الفبا) خواهید دید.

٥ کنترل Microsoft Office Chart 9.0 (و یا هر کنترل دیگری که مایلید) را انتخاب کنید - با این کار یک علامت چک در کنار آن ظاهر خواهد شد (شکل زیر را ببینید).



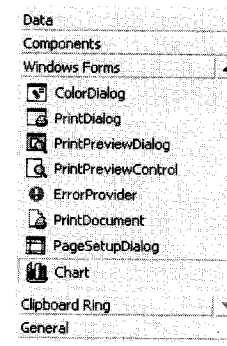
٦ برای اضافه کردن کنترلهای دیگر، آنها را هم به همان ترتیب فوق انتخاب کنید.

توجه

تمام کنترلهای پنجرهٔ Customize Toolbox بگونه‌ای نیستند، که بتوان آنها را به جعبه ابزار ویژوال استودیو .NET اضافه کرد.

٧ OK را کلیک کنید، تا کنترل (یا کنترلهای) انتخاب شده به انتهای جعبه ابزار اضافه شوند. توجه کنید که این کنترلها فقط به جعبه ابزار پروژهٔ فعال اضافه خواهند شد، و در سایر پروژه‌ها آنها را نخواهید دید.

٨ برای دیدن کنترل اضافه شده، به انتهای لیست جعبه ابزار بروید.



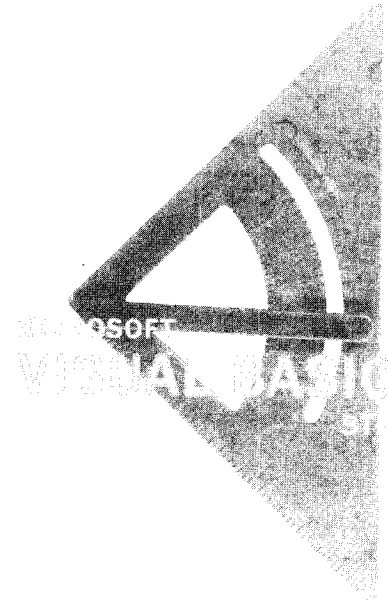
کنترلهای اکتیوایکس چنان شبیه کنترلهای ذاتی ویژوال استودیو .NET هستند، که گاهی تشخیص آنها از یکدیگر مشکل است؛ در واقع، اگر از قبل ندانید که این کنترلها اکتیوایکس هستند، نمی‌توانید بین آنها فرق بگذارید.

مرجع سریع فصل ۳

برای ...	انجام دهید
ایجاد جعبه متن	روی کنترل TextBox کلیک کرده، و یک جعبه متن رسم کنید.
ایجاد دکمه	روی کنترل Button کلیک کرده، و یک دکمه رسم کنید.
تغییر دادن یک خاصیت در زمان اجرای برنامه	مقدار خاصیت را در کُد برنامه تغییر دهید. مثلاً، Label1.Text = "Hello!"
ایجاد دکمهٔ رادیویی	روی کنترل RadioButton کلیک کرده، و یک دکمهٔ رادیویی رسم کنید. برای مرتبط کردن چند دکمهٔ رادیویی، بایستی آنها را روی یک جعبهٔ گروهی (GroupBox) قرار دهید.
ایجاد جعبه چک	روی کنترل CheckBox کلیک کرده، و یک جعبه چک رسم کنید.
ایجاد جعبه لیست	روی کنترل ListBox کلیک کرده، و یک جعبه لیست رسم کنید.
ایجاد جعبه لیست بازشو	روی کنترل ComboBox کلیک کرده، و یک جعبه ترکیبی رسم کنید.
اضافه کردن آیتم به جعبه لیست	در روال Form1_Load ، از متد Add استفاده کنید. برای مثال، ListBox1.Items.Add("Printer")
نمایش یک صفحهٔ وب	یک کنترل برچسب-لینک (WebLink) روی فرم برنامه قرار دهید، و با متد Process.Start صفحهٔ وب موردنظر را باز کنید.
نصب کنترل اکتیوایکس	روی جعبه ابزار راست-کلیک کرده، و فرمان Customize ToolBox را انتخاب کنید. در برگهٔ COM Components ، کنترل اکتیوایکس موردنظر را انتخاب کرده، و OK را کلیک کنید.

فصل

۴



کار با منوها و دیالوگ‌ها

در این فصل یاد می گیرید چگونه :

- ✔ با استفاده از کنترل MainMenu به برنامهٔ خود منو اضافه کنید.
- ✔ برای منوی برنامه کُد بنویسید.
- ✔ از دیالوگ‌های «بازکردن فایل» و «انتخاب رنگ» استفاده کنید.

در فصل قبل دیدید که چگونه می‌توان با استفاده از کنترل‌های ویژوال بیسیک ورودی را از کاربر گرفت. در این فصل با کنترلهای منو (menu) و دیالوگ (dialog box) ظاهر حرفه‌ای‌تری به برنامه‌های خود خواهیم داد. منو عبارتست از مجموعهٔ فرمانهای قابل انجام در برنامه، که معمولاً جای آن در بالاترین قسمت از فرم برنامه است. اکثر فرمانهای منو بلافاصله بعد از کلیک شدن اجرا می‌شوند (فرمانهای Copy ، Paste و Exit از این دسته‌اند)، اما فرمانهایی نیز هستند که قبل از اجرا شدن به اطلاعات تکمیلی نیاز دارند (مانند فرمان Open یا Save)؛ در جلوی این فرمانها معمولاً یک علامت ... دیده می‌شود، و وقتی آنها را اجرا می‌کنید، ویژوال بیسیک یک دیالوگ نمایش داده و اطلاعات موردنیاز را از شما می‌گیرد. در این فصل با طرز استفاده از کنترل MainMenu ، و دو تا از دیالوگ‌های ویژوال بیسیک، آشنا خواهید شد.

تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- ادیتور منو (که اکثر برنامه‌نویسان دل خوشی از آن نداشتند) جای خود را به کنترل جدیدی بنام MainMenu داده است؛ بعد از آنکه منو را ایجاد کردید، می‌توانید به کمک پنجرهٔ خواص و طراح منو (Menu Designer) ، مشخصات آنرا تغییر دهید.
- کنترل قدیمی CommonDialog دیگر در ویژوال بیسیک .NET وجود ندارد؛ بجای آن می‌توانید از کنترلهای مستقل OpenFileDialog (دیالوگ باز کردن فایل)، SaveFileDialog (دیالوگ ذخیره کردن فایل)، FontDialog (دیالوگ انتخاب فونت)، ColorDialog (دیالوگ انتخاب رنگ)،PrintDialog (دیالوگ چاپ)،PrintPreviewDialog (دیالوگ پیش‌نمایش چاپ)، و PageSetupDialog (دیالوگ تنظیم صفحه) -که در برگهٔ Windows Forms جعبه ابزار قرار دارند - استفاده کنید.
- فرمها دارای یک متد جدید بنام ShowDialog ، و یک خاصیت جدید بنام DialogResult هستند، که نمایش و کار با دیالوگ‌های استاندارد را ساده‌تر کرده است.

اضافه کردن منو: کنترل MainMenu

برای اضافه کردن منو به یک برنامه باید از کنترل MainMenu استفاده کنید؛ خواص این منو را هم می‌توان از طریق پنجرهٔ خواص ست کرد. کنترل MainMenu تمام امکانات ادیتور منو (از قبیل، اضافه یا حذف کردن آیتمهای منو، جابجا کردن آیتمها، تعریف کلیدهای دسترسی سریع و میانبُر، و اضافه کردن علامت چک به آیتمهای منو) را در اختیار شما قرار می‌دهد. برای فعال کردن آیتمهای یک منو، کماکان بایستی برای رویداد کلیک آنها کُد بنویسیم. در این قسمت یک منوی ساده می‌سازیم، و برای آن کُد می‌نویسیم.

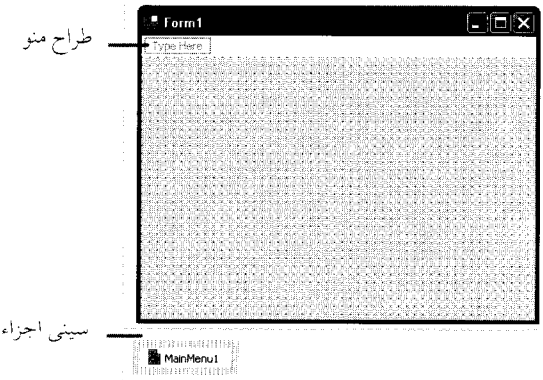
ایجاد یک منوی ساده

۱ ویژوال استودیو .NET را باز کنید.

۲ از منوی File|New آیتم Project را انتخاب کنید، تا پنجرهٔ پروژهٔ جدید باز شود.

۳ یک پروژهٔ Visual Basic Windows Application با نام MyMenu ، در پوشهٔ c:\vbnetsbs\chap04 ، ایجاد کنید.

٤ کنترل MainMenu را در جعبه ابزار انتخاب کرده، و یک منو روی فرم رسم کنید - نگران اندازه و محل آن نباشید، ویژوال استودیو خود این کار را بر عهده خواهد گرفت (شکل زیر را ببینید):



شیء منو روی فرم ظاهر نمی‌شود، بلکه آنرا در زیر فرم (که **سینی اجزاء** – Components tray – نامیده می‌شود) خواهید دید. تا قبل از ویژوال بیسیک .NET تمام اشیاء (حتی آنهایی که مانند تایمر عملاً چیزی برای دیدن نداشتند) روی فرم دیده می‌شدند. اما از این به بعد دیگر اشیاء غیر-بصری در سینی اجزاء قرار می‌گیرند، که می‌توانید از همانجا آنها را کنترل کنید.

علاوه بر شیء منو، که ویژوال بیسیک .NET در سینی اجزاء قرار می‌دهد، خود منو را هم در بالای فرم می‌بینید. این منو با علامت "Type Here" مشخص شده، و شما را تشویق می‌کند که روی آن کلیک کرده، و همانجا عنوان منو را وارد کنید. بعد از وارد کردن عنوان منوی اصلی، می‌توانید با کلید ⊡ عناوین زیر-منو را هم ایجاد کنید. این چیزی که در بالای فرم می‌بینید، طراح منو (Menu Designer) نام دارد، و ابزاریست بسیار مناسب برای ایجاد یک منوی کامل و حرفه‌ای.

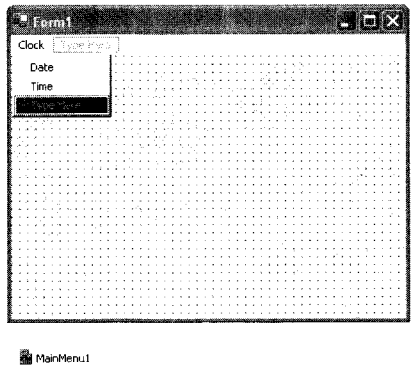
۵ روی "Type Here" در طراح منو کلیک کرده، و بعد از وارد کردن **Clock** ، Enter را بزنید. با این کار دو "Type Here" جدید (یکی زیر، و دیگری سمت راست) منوی Clock ظاهر می‌شود: اولی به شما اجازه می‌دهد تا آیتمهای ذیل منوی Clock را ایجاد کنید، و دومی برای ایجاد یک منوی کاملاً جدید است.

نکته

اگر منو را نمی‌بینید، روی کنترل MainMenu1 در سینی اجزاء کلیک کنید، تا طراح منو ظاهر شود.

۶ در "Type Here" زیر منوی Clock کلمهٔ **Date** را وارد کرده، و Enter را بزنید. با این کار فرمان جدیدی به منوی Clock اضافه شده، و یک "Type Here" جدید زیر آن ظاهر می‌شود.

۷ در "Type Here" زیر فرمان Date کلمهٔ **Time** را وارد کرده، و Enter را بزنید. اکنون منوی Clock دو فرمان به نامهای Date و Time دارد (که فعلاً برایمان کافیست)؛ شکل زیر را ببینید:



۸ برای بستن طراح منو، روی فرم کلیک کنید. با این کار منوی Clock بکلی ناپدید می‌شود. منویی که با این همه زحمت ساختیم، کجا رفت!؟ نگران نباشید، این منو جایی نرفته، و فقط موقتاً مخفی شده است؛ برای دیدن مجدد آن کافیست، روی کنترل MainMenu1 کلیک کنید.

۹ روی کنترل MainMenu1 در سینی اجزاء کلیک کنید؛ با این کار منوی Clock مجدداً در بالای فرم برنامه ظاهر می‌شود.

اضافه کردن کلید دسترسی سریع به فرمانهای منو

در بسیاری از برنامه‌ها دیده‌اید که برای اجرای فرمانی در یک منو الزاماً نباید از ماوس استفاده کنید، و با زدن یک کلید می‌توانید منویی را باز کرده، و یا فرمانی را اجرا کنید. مثلاً، برای باز کردن منوی File معمولاً می‌توان از کلیدهای ترکیبی Alt و F (که در کتابها با Alt+F نشان داده می‌شود) استفاده کرد. به کلیدی که باعث باز شدن یک منو می‌شود (و با Alt همراه است)، یا فرمانی را در یک منوی باز شده اجرا می‌کند، **کلید دسترسی سریع** (access key) می‌گویند. تشخیص کلیدهای دسترسی سریع ساده است، چون کلید مربوطه با یک زیرخط مشخص می‌شود.

تعریف کلید دسترسی سریع برای یک منو یا فرمان ساده است: کافیست در طراح منو هنگام نوشتن عنوان منو (یا فرمان) قبل از حرف موردنظر یک کاراکتر & (که آمپرساند خوانده می‌شود) قرار دهید. حرفی که بعد از & می‌آید، در هنگام اجرای برنامه به کلید دسترسی سریع آن منو یا فرمان تبدیل خواهد شد.

نکته

در ویندوز ۲۰۰۰ تا وقتی برای اولین بار کلید Alt را نزنید، زیرخط کلیدهای دسترسی سریع دیده نخواهند شد. برای تغییر دادن این حالت (که حالت پیش‌فرض ویندوز ۲۰۰۰ است) به برگهٔ Effects در اپلت Display پانل کنترل ویندوز بروید.

اجازه دهید در این قسمت به منوی Clock کلیدهای دسترسی سریع اضافه کنیم.

قواعد مرسوم طراحی منو

در اینجا قصد دارم چند تا از قواعد مرسوم در طراحی منو برای برنامه‌های ویندوز را بیان کنم. یکی از این قواعد اینست که، «حرف اول عنوان هر منو باید بزرگ باشد». منوی اول و دوم برنامه معمولاً File و Edit ، و آخرین آنها نیز منوی Help است؛ منوهای View ، Format و Window نیز در اکثر برنامه‌ها وجود دارند. صرفنظر از هر منو یا فرمانی که دارید، مهم اینست که آنها واضح و قابل‌فهم بوده، و تفاوت فاحشی با منوی سایر برنامه‌های ویندوز نداشته باشند. برای ایجاد منوهای یکدست می‌توانید از خطوط راهنمای زیر استفاده کنید:

- برای منوها از عناوین گویا و کوتاه (یک، یا حداکثر دو کلمه‌ای) استفاده کنید.
- برای تمام منوها و فرمانها کلید دسترسی سریع (ترجیحاً حرف اول آنها) ایجاد کنید.
- کلید دسترسی سریع فرمانهایی که در یک سطح قرار دارند، بایستی منحصربفرد باشد.
- اگر فرمانی دارای دو حالت فعال/غیرفعال (On/Off) است، برای آن علامت چک در نظر بگیرید؛ برای این منظور می‌توانید در پنجرهٔ خواص خاصیت Checked آنرا به True ست کنید.
- برای فرمانهایی که نیاز به اطلاعات اضافی دارند، علامت ... در نظر بگیرید؛ این علامت به کاربر می‌گوید که انتخاب این فرمان باعث باز شدن یک دیالوگ خواهد شد.

اضافه کردن کلید دسترسی سریع

۱ کنترل MainMenu1 را انتخاب کرده، و روی عنوان منوی Clock کلیک کنید، تا کرسر ادیت متن در آن ظاهر شود.

۲ با استفاده از کلید ⊡ به ابتدای کلمهٔ Clock (قبل از حرف C) بروید.

۳ در این نقطه یک کاراکتر & وارد کنید، تا حرف C به کلید دسترسی سریع این منو تبدیل شود.

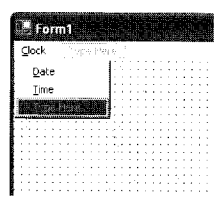
٤ فرمان Date را انتخاب کرده، و روی کلمهٔ Date کلیک کنید، تا کرسر ادیت متن در آن ظاهر شود.

۵ یک کاراکتر & قبل از حرف D وارد کنید، تا حرف D به کلید دسترسی سریع این فرمان تبدیل شود.

۶ فرمان Time را انتخاب کرده، و روی کلمهٔ Time کلیک کنید، تا کرسر ادیت متن در آن ظاهر شود.

۷ یک کاراکتر & قبل از حرف T وارد کنید، تا این حرف به کلید دسترسی سریع فرمان Time تبدیل شود.

۸ Enter را بزنید. تا اینجا منوی Clock باید شبیه شکل زیر شده باشد.



بعد از آنکه یک منو را طراحی کردید، شاید لازم شود تا ترتیب فرمانهای آنرا عوض کنید (کاری که اغلب پیش می‌آید). در قسمت آینده خواهید دید که چگونه می‌توان ترتیب فرمانهای منوی Clock را عوض کرد.

تغییر دادن ترتیب فرمانهای منو

۱ روی کنترل MainMenu1 کلیک کنید، تا منوی Clock انتخاب شود.

۲ عوض کردن جای یک فرمان بسیار آسان است، و کافیست فرمان موردنظر را با ماوس گرفته و به محل جدید بکشید؛ پس: فرمان Time را گرفته، و آنرا به بالای آیتم Date کشیده و رها کنید. به همین سادگی!

این از ظاهر منوی Clock ؛ اما این منو هنوز هیچ کاری انجام نمی‌دهد، و باید برای فرمانهای آن روال رویداد بنویسیم.

نکته

برای حذف یک آیتم اضافی در منو، ابتدا آنرا انتخاب کرده، و سپس کلید Del را بزنید.

پردازش فرمانهای منو

منو و فرمانهای آن نیز مانند سایر کنترلهای ویژوال بیسیک هستند، و برای اینکه کاری انجام دهند باید برای آنها روال رویداد بنویسیم. گاهی یک فرمان به تنهایی می‌داند چکار باید بکند (مانند فرمان Exit که برنامه را پایان می‌دهد)، اما گاهی نیز برای انجام کار خود به اطلاعات اضافی نیاز دارد (مانند فرمان Open که بایستی ابتدا فایل موردنظر را انتخاب کنید، تا برنامه بتواند آنرا باز کند). فرمانهای منوی Clock بسیار ساده‌اند، و فقط تاریخ و وقت را نشان می‌دهند - و این کار را در یک برچسب انجام می‌دهند.

اضافه کردن یک برچسب به فرم

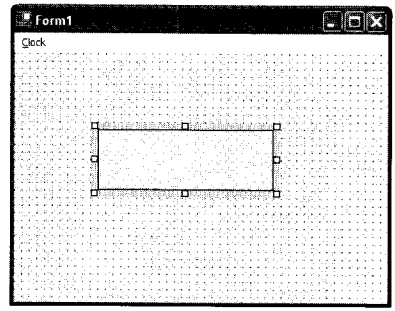
۱ کنترل برچسب (Label) را در جعبه ابزار انتخاب کنید.

۲ یک برچسب روی فرم رسم کنید؛ نام این برچسب Label1 خواهد بود.

۳ خواص برچسب Label1 را با توجه به جدول زیر ست ست کنید:

شیء	خاصیت	مقدار
Label	BorderStyle	FixedSingle
	Font	Microsoft Sans Serif, Bold, 14-point
	Text	(خالی)
	TextAlign	MiddleCenter

فرم برنامه را در شکل زیر می‌بینید:



در قسمت آینده برای روال رویداد کلیک فرمانهای Time و Date کُد خواهیم نوشت.

نکته

اگر برخی از دستورات کُدهای زیر برایتان ناآشناست، و نمی‌دانید چه کاری انجام می‌دهند، نگران نباشید - در فصلهای ۵ تا ۷ با منطق برنامه‌نویسی ویژوال بیسیک آشنا خواهید شد.

نوشتن روال رویداد فرمانهای منو

۱ روی منوی Clock کلیک کنید، تا فرمانهای ذیل آن دیده شوند.

۲ روی فرمان Time دو-کلیک کنید، تا روال رویداد آن (MenuItem3_Click) در ادیتور کُد باز شود. از آنجائیکه فرمان Time سومین آیتمی بود که ایجاد کردیم، نام آن MenuItem3 است، و ویژوال بیسیک با وجود عوض کردن جای آن، این موضوع را فراموش نکرده است. استفاده از نامهایی که ویژوال بیسیک به کنترلها می‌دهد، چندان اشکالی ندارد، اما خیلی بهتر خواهد بود اگر از نامهای مناسبتری برای اشیاء برنامه استفاده کنیم (در قسمتهای آینده خواهید دید که چگونه می‌توان با استفاده از خاصیت Name نام فرمانهای منوی Clock را عوض کرد).

۳ دستور زیر را در روال MenuItem3_Click بنویسید:

```
Label1.Text = TimeString
```

این دستور وقت فعلی را (که از ساعت سیستم می‌خواند) در برچسب Label1 نشان می‌دهد. خاصیت TimeString در هر لحظه وقت فعلی را در خود دارد، و در هر نقطه از برنامه می‌توانید از آن استفاده کنید. (در واقع، این دستور جایگزین دستور Time$ در ویرایشهای قبلی ویژوال بیسیک شده است.)

نکته

خاصیت TimeString وقت را از ساعت سیستم می‌گیرد، و برای فرمت کردن آن از تنظیمات Date/Time در اپلت Regional Settings پانل کنترل ویندوز استفاده می‌کند.

٤ کلید ▶ را بزنید. (اگر در نوشتن این دستور مرتکب خطایی شده باشید، ویژوال بیسیک بلافاصله به شما تذکر خواهد داد.)

٥ با کلیک کردن دکمهٔ View Designer در کاوشگر راه‌حل، به فرم برنامه برگردید، و این بار روی فرمان Date دو-کلیک کنید، تا روال رویداد کلیک این آیتم (MenuItem2_Click) در ادیتور کُد باز شود.

٦ دستور زیر را در روال MenuItem2_Click بنویسید:

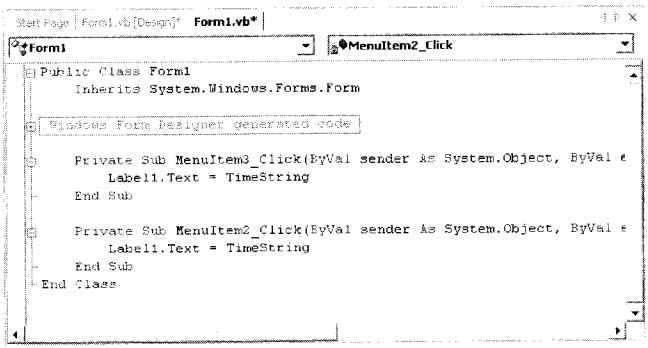
```
Label1.Text = DateString
```

این دستور تاریخ فعلی را (که از ساعت سیستم می‌خواند) در برچسب Label1 نمایش می‌دهد. خاصیت DateString در هر لحظه تاریخ فعلی را در خود دارد، و در هر نقطه از برنامه می‌توانید از آن استفاده کنید. (این دستور نیز جایگزین دستور $Date در ویرایشهای قبلی ویژوال بیسیک شده است.)

نکته

خاصیت DateString تاریخ را از ساعت سیستم می‌گیرد، و برای فرمت کردن آن از تنظیمات Date/Time در اپلت Regional Settings پانل کنترل ویندوز استفاده می‌کند.

٧ کلید ▶ را بزنید. (شکل زیر را ببینید:)



فرمانهای منوی Clock با همین دو دستور ساده فعال شدند.

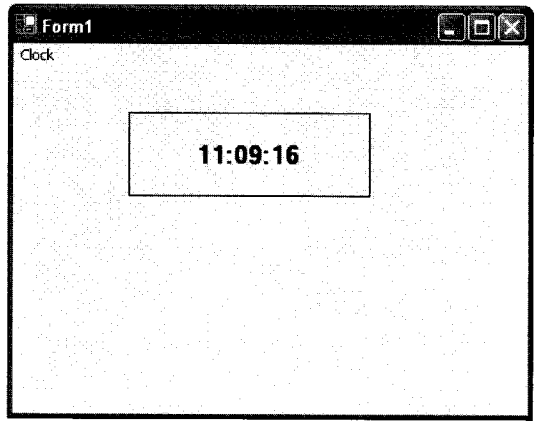
٨ با کلیک کردن دکمهٔ Save All در میله ابزار استاندارد، پروژه را ذخیره کنید.

اجرای برنامهٔ MyMenu

١ دکمهٔ Start را در میله ابزار استاندارد کلیک کنید، تا برنامهٔ MyMenu کامپایل و اجرا شود.

۲ روی منوی Clock در بالای فرم برنامه کلیک کنید، تا باز شود.

۳ روی فرمان Time کلیک کنید، تا وقت فعلی در برچسب Label1 نمایش داده شود:



در قسمت بعد، برای دیدن تاریخ از کلیدهای دسترسی سریع استفاده خواهیم کرد.

٤ کلید Alt را گرفته و سپس رها کنید، تا منوی Clock فعال شود.

۵ کلید C را بزنید، تا منوی Clock باز شود.

۶ کلید D را بزنید، تا تاریخ فعلی در برچسب Label1 نشان داده شود.

۷ با کلیک کردن روی دکمهٔ Close در میلهٔ عنوان برنامه، آنرا ببندید.

احسنت! منوی برنامهٔ ما به همین راحتی عملیاتی شده، و کار می‌کند.

خواص و توابع ساعت سیستم

ساعت سیستم دارای خواص و توابع متعددیست، که به کمک آنها می‌توانید اطلاعات مربوط به وقت و تاریخ را در اختیار داشته باشید. در جدول زیر مهمترین این خاصیت‌ها و توابع را می‌بینید (برای کسب اطلاعات بیشتر، به سیستم کمک ویژوال بیسیک مراجعه کنید):

خاصیت یا تابع	توضیح
TimeString	این خاصیت وقت فعلی را از ساعت سیستم برمی‌گرداند.
DateString	این خاصیت تاریخ فعلی را از ساعت سیستم برمی‌گرداند.
Now	این خاصیت وقت و تاریخ فعلی را از ساعت سیستم برمی‌گرداند.
Hour (*time*)	این تابع قسمت ساعت زمان داده شده را (بصورت 0 تا 23) برمی‌گرداند.
Minute (*time*)	این تابع قسمت دقیقهٔ زمان داده شده را (بصورت 0 تا 59) برمی‌گرداند.
Second (*time*)	این تابع قسمت ثانیهٔ زمان داده شده را (بصورت 0 تا 59) برمی‌گرداند.

Day (*date*)	این تابع قسمت روز تاریخ داده شده را (بصورت 1 تا 31) برمی‌گرداند.
Month (*date*)	این تابع قسمت ماه تاریخ داده شده را (بصورت 1 تا 12) برمی‌گرداند.
Year (*date*)	این تابع قسمت سال تاریخ داده شده را برمی‌گرداند.
Weekday (*date*)	این تابع روز هفته را برای زمان داده شده (1 برای یکشنبه، 2 برای دوشنبه، و الی‌آخر) برمی‌گرداند.

استفاده از کنترل دیالوگ

در برگهٔ Windows Forms جعبه ابزار ویژوال بیسیک هفت کنترل دیالوگ استاندارد وجود دارد. این کنترلها بدون کمترین زحمتی اکثر عملکردهای مهم ویندوز (از قبیل باز و ذخیره کردن فایل، چاپ و غیره) را انجام می‌دهند. البته گاهی لازم می‌شود چند خط کُد برای این دیالوگها بنویسید، ولی شکل و ظاهر آنها با سایر قسمتهای ویندوز کاملاً هماهنگ است.

در جدول زیر این هفت کنترل را می‌بینید. (آنها، با چند استثنا، بسیار شبیه کنترلهای CommonDialog در ویژوال بیسیک ۶ هستند؛ برای مثال، کنترل PrintPreviewDialog یک دیالوگ جدید است، که در ویرایشهای قبلی ویژوال بیسیک وجود نداشت.)

نام کنترل	کاربرد
OpenFileDialog	درایو، پوشه، و نام یک فایل موجود را برمی‌گرداند.
SaveFileDialog	درایو، پوشه، و نام یک فایل جدید را می‌گیرد.
FontDialog	به کاربر امکان انتخاب فونت را می‌دهد.
ColorDialog	به کاربر امکان انتخاب رنگ را از یک پالت می‌دهد.
PrintDialog	به کاربر امکان انتخاب تنظیمات چاپ را می‌دهد.
PrintPreviewDialog	به کاربر امکان می‌دهد سند را قبل از چاپ ببیند (مانند برنامهٔ Word).
PageSetupDialog	به کاربر امکان انتخاب تنظیمات صفحهٔ چاپی (مانند حاشیه‌ها، اندازهٔ کاغذ و جهت آن) را می‌دهد.

در این قسمت، منوی جدیدی به برنامهٔ MyMenu اضافه کرده، و با دیالوگهای OpenFileDialog و ColorDialog تمرین خواهیم کرد. (البته می‌توانید یک پروژهٔ جدید نیز ایجاد کرده، و تمرین زیر را در آن انجام دهید – همان کاری که من در CD ضمیمه کرده‌ام.)

اضافه کردن دیالوگهای OpenFileDialog و ColorDialog

۱ با فرمان File|Open|Project ، پروژهٔ MyMenu را (که در پوشهٔ c:\vbnetsbs\chap04\mymenu قرار دارد) باز کنید. برای باز کردن فرم Form1.vb (در کاوشگر راه‌حل) روی آن دو-کلیک کنید.

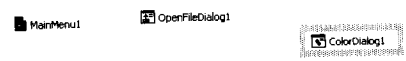
در این تمرین دو دیالوگ به برنامهٔ MyMenu اضافه می‌کنیم: یک دیالوگ OpenFileDialog برای باز کردن فایلهای بیت‌مَپ، و یک دیالوگ ColorDialog برای تغییر دادن رنگ برچسب ساعت. کنترلهای دیالوگ هم جزء کنترلهای نامریی ویژوال بیسیک هستند، و در سینی اجزاء (کنار کنترل منو) قرار خواهند گرفت.

۲ کنترل OpenFileDialog را در برگهٔ Windows Forms جعبه ابزار انتخاب کرده، و در سینی اجزاء کلیک کنید؛ با این کار یک «دیالوگ باز کردن فایل» به این سینی اضافه خواهد شد.

نکته

اگر کنترل OpenFileDialog را در جعبه ابزار نمی‌بینید، با کلیک کردن روی دکمهٔ ▼ (کنار عنوان برگهٔ Clipboard Ring) به قسمتهای پائین‌تر برگهٔ Windows Forms بروید.

۳ کنترل ColorDialog را در برگهٔ Windows Forms جعبه ابزار انتخاب کرده، و در سینی اجزاء کلیک کنید؛ با این کار یک «دیالوگ انتخاب رنگ» به این سینی اضافه خواهد شد؛ شکل زیر را ببینید:



خواص مختلف دیالوگها را هم می‌توان، مانند سایر اشیاء، از طریق پنجرهٔ خواص کنترل کرد.

همانطور که گفتم، از دیالوگ OpenFileDialog برای باز کردن فایلهای بیت‌مپ استفاده خواهیم کرد – بنابراین، به یک کنترل جعبه تصویر (PictureBox) برای نمایش این تصاویر نیاز داریم.

اضافه کردن جعبه تصویر

۱ کنترل جعبه تصویر (PictureBox) را در جعبه ابزار انتخاب کنید.

۲ با این ابزار یک مستطیل زیر برچسب Label1 رسم کنید.

۳ خاصیت SizeMode این شیء را به StretchImage ست کنید.

پس از آن باید یک منوی File (با سه فرمان بنامهای Open ، Close و Exit) به برنامه اضافه کنیم.

اضافه کردن منوی File

۱ منوی Clock را انتخاب کرده، و سپس روی Type Here سمت راست آن کلیک کنید.

۲ در اینجا، کلمهٔ **&File** را وارد کنید (حرف F کلید دسترسی سریع این منو خواهد بود).

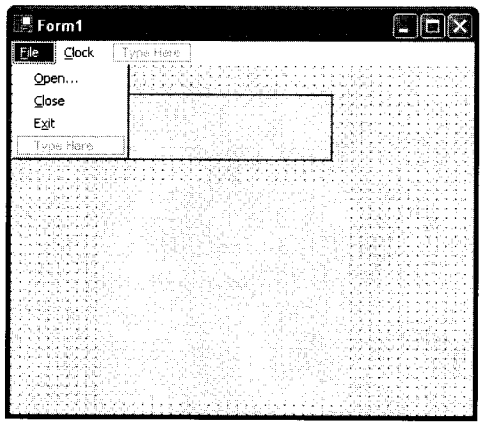
۳ کلید ↵ را زده، و در Type Here زیر آن، کلمهٔ **&Open...** را وارد کنید (حرف O به کلید دسترسی سریع این فرمان تبدیل خواهد شد). اضافه کردن ... برای آنست که کاربر متوجه باشد، انتخاب این فرمان با باز شدن یک دیالوگ همراه است.

٤ یک باز دیگر کلید ⊡ را زده، و در Type Here زیر فرمان Open ، کلمهٔ **&Close** را وارد کنید (حرف C کلید دسترسی سریع این فرمان است). این فرمان فایل بیت‌مپ باز شده را می‌بندد.

٥ باز هم کلید ⊡ را زده، و در Type Here زیر Close ، کلمهٔ **E&xit** را وارد کنید (حرف x کلید دسترسی سریع این فرمان است). فرمان Exit به برنامه پایان می‌دهد. (در اغلب برنامه‌های ویندوز – مانند Word – حرف x کلید دسترسی سریع فرمان بستن برنامه است، و ما هم به این برنامه‌ها تأسی کرده‌ایم.)

٦ همانطور که دیده‌اید، منوی File معمولاً اولین منو در برنامه‌های ویندوز است.

برای جابجا کردن کل یک منو، کافیست آنرا با ماوس گرفته و به محل جدید بکشید. پس، منوی File را گرفته و روی منوی Clock بکشید، تا جای آنرا بعنوان منوی اول بگیرد؛ شکل زیر را ببینید:



تغییر دادن نام آیتمهای منوی File

هر شیء در برنامه‌های ویژوال بیسیک (از جمله آیتمهای منو) دارای خاصیتی بنام Name است، که می‌توان آنرا بدلخواه ست کرد. خاصیت Name هیچ تأثیری روی شکل و ظاهر شیء ندارد، و فقط در کُد برنامه بکار می‌آید. در واقع، نامگذاری اشیاء برنامه با نامهای واقعی (و با مسمی) یکی از مهمترین تکنیکهای برنامه‌نویسی حرفه‌ای محسوب می‌شود (و ما هم از این پس آنرا رعایت خواهیم کرد). در تمرین زیر، نام فرمانهای منوی File را عوض می‌کنیم.

تغییر دادن نام اشیاء

١ روی منوی File کلیک کرده، و بعد از باز کردن پنجرهٔ خواص، خاصیت Name آنرا به **mnuFile** تغییر دهید. (خاصیت Name در دستهٔ Design و داخل یک جفت پرانتز قرار دارد، تا همیشه در بالاترین نقطهٔ لیست دیده شود).

روش نامگذاری منوی File (که در فوق دیدید) یکی از قراردادهای مرسوم بین برنامه‌نویسان است. در این روش سه حرف اول نام شیء مشخص‌کنندهٔ نوع آن است – در مثال فوق، حروف mnu

مشخص می‌کنند که این شیء یک منو است. حروف (و کلمات) بعدی نیز نام اصلی شیء (و گویای ماهیت آن) هستند.

۲ آیتم ...Open را انتخاب کرده، و خاصیت Name آنرا به **mnuOpenItem** تغییر دهید.

۳ روی آیتم Close کلیک کرده، و خاصیت Name آنرا به **mnuCloseItem** تغییر دهید.

٤ آیتم Exit را انتخاب کرده، و خاصیت Name آنرا به **mnuExitItem** تغییر دهید.

۵ با کلیک کردن دکمهٔ Save All در میله ابزار استاندارد، پروژه را ذخیره کنید.

همانطور که گفتم، نامگذاری اشیاء بهیچوجه الزامی نیست، و می‌توانید از همان نامهایی که ویژوال بیسیک به کنترلها داده، استفاده کنید. اما اکیداً توصیه می‌کنم که، «**اشیاء برنامهٔ خود را با نامهای مناسب و با معنی - که نشاندهندهٔ عملکرد و هدف آنها باشد - نامگذاری کنید.**»

غیرفعال کردن فرمانهای منو

در یک برنامهٔ حرفه‌ای ویندوز، تمام فرمانهای منو همزمان با هم فعال نیستند. مثلاً، فرمان Paste منوی Edit فقط زمانی فعال می‌شود که قبل از آن با فرمان Copy چیزی در **تخته برش ویندوز** (clipboard) کپی شده باشد. برای غیرفعال کردن یک آیتم منو، باید خاصیت Enabled آنرا به False ست کرد. فرمانهای غیرفعال بصورت خاکستری کم رنگ درمی‌آیند، و به کلیک شدن هم پاسخ نمی‌دهند.

در تمرین زیر، آیتم Close را در شروع برنامه غیرفعال می‌کنیم (چون تا فایلی باز نشده باشد، بستن آن مفهومی ندارد). این آیتم فقط زمانی باید فعال شود که فایلی (با فرمان Open) باز شده باشد (بعداً با روش این کار هم آشنا خواهید شد).

غیرفعال کردن فرمان Close

۱ روی فرمان Close در منوی File کلیک کرده، و آنرا انتخاب کنید.

۲ پنجرهٔ خواص را باز کرده، و خاصیت Enabled شیء mnuCloseItem را به False ست کنید.

در قسمت بعد برای ست کردن رنگ برچسب ساعت از دیالوگ ColorDialog استفاده خواهیم کرد. این دیالوگ رنگ انتخاب شده را از طریق خاصیت Color برمی‌گرداند، که می‌توان به کمک آن رنگ برچسب Label1 را ست کرد. اما قبل از آن باید یک فرمان جدید (بنام Text Color) به منوی Clock اضافه کنیم.

اضافه کردن فرمان Text Color به منوی Clock

۱ منوی Clock را انتخاب کرده، و سپس روی Text Here انتهای آن کلیک کنید.

۲ کلمهٔ **Text Co&lor...** را در اینجا وارد کنید (کلید دسترسی سریع این فرمان حرف l است، چون قبلاً از حرف T برای فرمان Time استفاده کرده‌ایم - **کلیدهای دسترسی سریع در یک منو نباید تکراری باشند**). علامت ... بعد از کلمهٔ Text Color نیز حاکی از اینست که انتخاب این فرمان باعث باز شدن

یک دیالوگ خواهد شد.

۳ خاصیت Name فرمان Text Color را به mnuTextColorItem تغییر دهید.

کُدنویسی برای دیالوگ‌ها

برای باز کردن یک دیالوگ از دستور ShowDialog در روال کلیک فرمان موردنظر استفاده می‌کنیم. البته گاهی قبل از باز کردن یک دیالوگ لازمست یک یا دو خاصیت آنرا ست کنیم. و بعد از بسته شدن دیالوگ، انتخاب کاربر را از طریق مقداری که دیالوگ برمی‌گرداند، پردازش می‌کنیم.

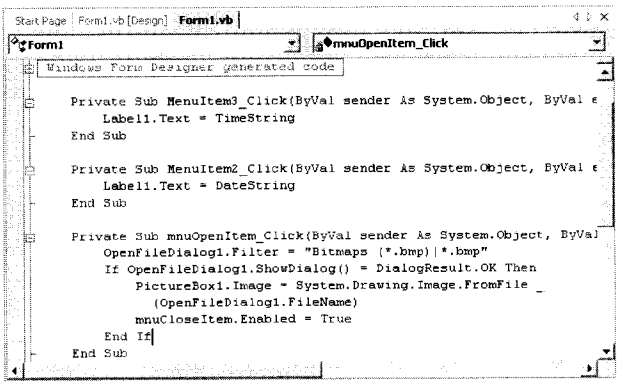
در تمرین زیر، ابتدا خاصیت Filter کنترل OpenFileDialog1 را در روال mnuOpenItem_Click ست کرده، و سپس این دیالوگ را (با متد ShowDialog) باز می‌کنیم. بعد از آن که کاربر تصویر (فایل بیت‌مپ) موردنظر را انتخاب کرد و دیالوگ را بست، این تصویر را در کنترل جعبه تصویر باز می‌کنیم. و بالاخره، فرمان Close را فعال می‌کنیم، تا کاربر در صورت تمایل بتواند تصویر را ببندد.

نوشتن روال رویداد فرمان Open

۱ روی فرمان Open در منوی File پروژهٔ MyMenu دو-کلیک کنید، تا روال mnuOpenItem_Click در ادیتور کُد باز شود.

۲ دستورات زیر را بین Private Sub و End Sub این روال وارد کنید (شکل زیر را ببینید):

```
OpenFileDialog1.Filter = "Bitmaps (*.bmp)|*.bmp"
If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
    PictureBox1.Image = System.Drawing.Image.FromFile _
      (OpenFileDialog1.FileName)
    mnuCloseItem.Enabled = True
End If
```



در هر یک از سه دستور اول این کُد از یک خاصیت یا متد کنترل دیالوگ استفاده کرده‌ایم. در خط اول، به کمک خاصیت Filter فایلهایی را که این دیالوگ باید نمایش دهد، فیلتر می‌کنیم. این فیلتر فقط یک نوع فایل

(فایلهای بیت‌مپ) را نشان می‌دهد، چون از عبارت *.bmp استفاده کرده‌ایم. ست کردن این فیلتر بسیار مهم است، چون کاربر نباید فایلهای غیر-تصویری را باز کند (بار کردن فایلهای غیر-تصویری در جعبه تصویر همان، و خطای برنامه همان!). انواع فایلهایی که کنترل جعبه تصویر می‌تواند باز کند، عبارتند از: .bmp ، .wmf ، .emf ، .ico ، .jpg ، .png و .gif .

برای اضافه کردن انواع دیگر فایل به یک فیلتر، از کاراکتر | استفاده می‌کنیم:

```
OpenFileDialog1.Filter = "Bitmaps (*.bmp)|*.bmp|Metafiles (*.wmf)|*.wmf"
```

فیلتر فوق هر دو نوع فایل بیت‌مپ و متافایل را نشان خواهد داد.

دستور دوم دیالوگ OpenFileDialog1 را با متد ShowDialog باز می‌کند. متد ShowDialog شبیه متد Show در ویژوال بیسیک ۶ است، با این تفاوت که می‌تواند هر نوع فرمی را نمایش دهد. متد ShowDialog مقداری برمی‌گرداند، بنام DialogResult ، که نشان می‌دهد کاربر کدام دکمهٔ دیالوگ را کلیک کرده است. برای تست مقدار برگشتی دیالوگ (و فهمیدن اینکه آیا کاربر دکمهٔ Open را کلیک کرده یا خیر)، از یک دستور If...Then استفاده کرده‌ایم. این دستور مقدار برگشتی دیالوگ را با DialogResult.OK (که یک مقدار ثابت است) مقایسه می‌کند. اگر چنین باشد (یعنی کاربر Open را کلیک کرده باشد)، به معنای آنست که خاصیت FileName دیالوگ نام فایل موردنظر کاربر را در خود دارد. (در فصل ۶ با دستور If...Then بیشتر آشنا خواهید شد.)

وقتی کاربر فایلی را انتخاب و دکمهٔ Open را کلیک کند، نام و مسیر کامل آن فایل در خاصیت OpenFileDialog1.FileName قرار می‌گیرد - و دستور سوم (که در دو خط نوشته شده) این فایل را با متد System.Drawing.Image.FromFile در جعبه تصویر بار می‌کند.

دستور چهارم نیز فرمان Close را با ست کردن خاصیت Enabled آن به True فعال می‌کند، تا کاربر قادر به بستن فایل باز شده باشد - کاری که در تمرین زیر انجام می‌دهیم.

نوشتن روال رویداد فرمان Close

۱ به فرم برنامه برگردید، و روی فرمان Close منوی File دو-کلیک کنید، تا روال mnuCloseItem_Click در ادیتور کُد باز شود.

۲ دستورات زیر را بین Private Sub و End Sub وارد کنید:

```
PictureBox1.Image = Nothing
mnuCloseItem.Enabled = False
```

دستور اول با نسبت دادن کلمهٔ کلیدی Nothing به خاصیت Image جعبه تصویر، آن را پاک می‌کند. دستور دوم نیز فرمان Close منوی File را دوباره غیرفعال می‌کند (چون دیگر تصویری در جعبه تصویر وجود ندارد، که بخواهیم آنرا ببندیم).

منوی File فرمان دیگری دارد بنام Exit ، که با آن می‌توان برنامه را بست - در قسمت بعد کُد این فرمان را می‌نویسیم.

نوشتن روال رویداد فرمان Exit

۱ به فرم برنامه برگردید، و روی فرمان Exit در منوی File دو-کلیک کنید، تا روال mnuExitItem_Click در ادیتور کُد باز شود.

۲ دستور زیر را بین Private Sub و End Sub این روال بنویسید:

```
End
```

متد End (که قبلاً هم آنرا دیده‌اید) خیلی ساده برنامه را پایان می‌دهد.

نوشتن روال رویداد فرمان Text Color

۱ به فرم برنامه برگردید، و روی فرمان Text Color در منوی Clock دو-کلیک کنید، تا روال mnuTextColorItem_Click در ادیتور کُد باز شود.

۲ دستورات زیر را بین Private Sub و End Sub وارد کنید:

```
ColorDialog1.ShowDialog()
Label1.ForeColor = ColorDialog1.Color
```

نکته

از کنترل ColorDialog برای ست کردن رنگ هر چیزی که رنگ داشته باشد (از جمله رنگ پس‌زمینه، رنگ پیش‌زمینه، و رنگ آشکال)، می‌توان استفاده کرد.

اولین دستور دیالوگ انتخاب رنگ را با متد ShowDialog باز می‌کند. دستور دوم نیز رنگ انتخاب شده (که در خاصیت ColorDialog1.Color برگشت داده می‌شود) را به خاصیت Label1.ForeColor نسبت می‌دهد، و بدین ترتیب رنگ فونت این برچسب را تغییر می‌دهد.

۳ با کلیک کردن دکمهٔ Save All در میله ابزار استاندارد، پروژه را ذخیره کنید.

خواص دیالوگ ColorDialog

یکی از چیزهایی که می‌توان در دیالوگ ColorDialog کنترل کرد، گزینه‌های موجود در آن است. برای ست کردن این خواص می‌توان از پنجرهٔ خواص استفاده کرد، و یا آنها را قبل از اجرای متد ShowDialog در کُد برنامه ست کرد. چند تا از مهمترین این خواص را (که فقط مقدار True یا False می‌گیرند) در جدول زیر می‌بینید:

خاصیت	توضیح
AllowFullOpen	با True کردن این خاصیت، دکمهٔ Define Custom Colors در دیالوگ انتخاب رنگ فعال خواهد شد.
AnyColor	با True کردن این خاصیت، کاربر می‌تواند هر کدام از رنگهای دیالوگ ColorDialog را انتخاب کند.

خاصیت	توضیح
FullOpen	با True کردن این خاصیت، هنگام باز شدن دیالوگ انتخاب رنگ قسمت Custom Colors دیده خواهد شد.
ShowHelp	با True کردن این خاصیت، دکمهٔ Help در دیالوگ انتخاب رنگ فعال خواهد شد.
SolidColorOnly	با True کردن این خاصیت، دیالوگ انتخاب رنگ فقط رنگهای ساده (رنگهای غیر ترکیبی) را نشان خواهد داد.

و حالا نوبت اجرای برنامه است.

اجرای برنامهٔ MyMenu

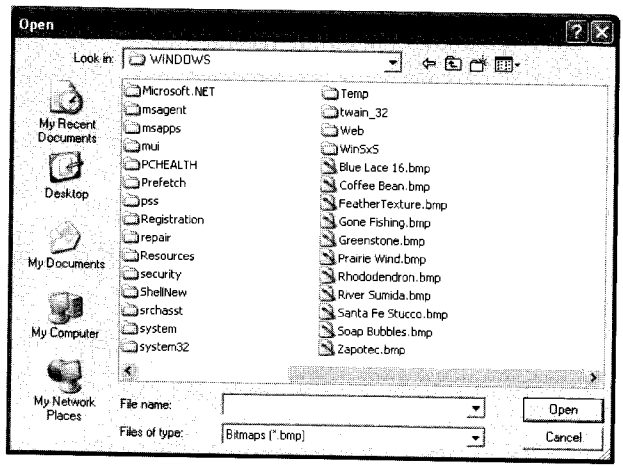
۱ دکمهٔ Start را در میله ابزار استاندارد کلیک کنید، تا برنامهٔ MyMenu کامپایل و اجرا شود. اکنون برنامهٔ ما دو منو بنامهای File و Clock دارد.

۲ فرمان Open را از منوی File انتخاب کنید؛ دیالوگ Open باز می‌شود – خیلی جالبست، نه؟ توجه کنید که در قسمت File of type عبارت Bitmaps (*.bmp) دیده می‌شود، چون خاصیت Filter این دیالوگ را بصورت زیر ست کردیم:

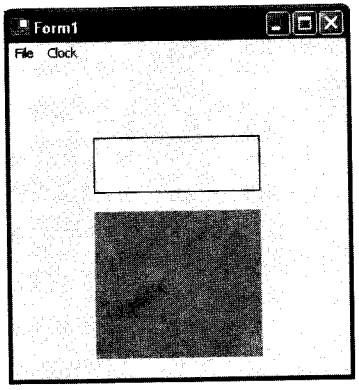
```
OpenFileDialog1.Filter = "Bitmaps (*.bmp)|*.bmp"
```

بخش اول این فیلتر (قبل از |) عبارتیست که در قسمت File of type دیالوگ Open دیده خواهد شد، و بخش دوم آن (بعد از |) مشخص می‌کند که این دیالوگ چه نوع فایلهایی را باید نشان دهد.

۳ اگر به پوشهٔ c:\windows (یا c:\winnt) بروید، تعداد زیادی فایل بیت‌مپ خواهید دید:



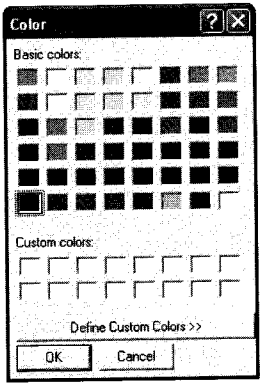
٤ یکی از این فایلها را انتخاب کرده، و دکمهٔ Open را کلیک کنید، تا در جعبه تصویر به نمایش در آید:



اکنون اجازه دهید سراغ منوی Clock برویم.

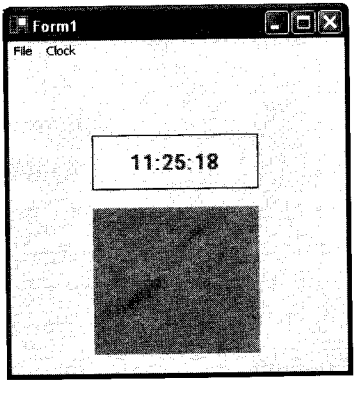
۵ فرمان Time را از منوی Clock انتخاب کنید، تا وقت فعلی در برچسب نمایش داده شود.

۶ از همان منوی Clock ، فرمان Text Color را انتخاب کنید، تا دیالوگ انتخاب رنگ باز شود:



به کمک این دیالوگ می‌توانید رنگ فونت ساعت را تغییر دهید (رنگ فعلی آن سیاه است).

۷ روی رنگ آبی کلیک کرده، و سپس دکمهٔ OK را کلیک کنید - دیالوگ Color بسته شده، و برچسب ساعت به رنگ آبی در می‌آید.



۸ فرمان Date را از منوی Clock انتخاب کنید؛ همانطور که می‌بینید، تاریخ هم با رنگ آبی نمایش داده می‌شود. در واقع، تا زمانی که مجدداً رنگ این برچسب را عوض نکرده‌اید، به همان رنگ باقی خواهد ماند.

۹ منوی File را باز کنید؛ می‌بینید که فرمان Close فعال است، چون بعد از باز کردن تصویر در روال mnuOpenItem_Click آنرا فعال کرده بودیم.

۱۰ کلید C (کلید دسترسی سریع فرمان Close) را بزنید – بله، تصویر بیت‌مپ (به لطف کلمهٔ کلیدی Nothing) ناپدید می‌شود!

۱۱ دوباره منوی File را باز کنید؛ همانطور که می‌بینید، فرمان Close غیرفعال شده است، چون دیگر تصویری در جعبه تصویر وجود ندارد.

۱۲ فرمان Exit را انتخاب کنید – برنامه بسته شده، و دوباره به محیط برنامه‌نویسی ویژوال بیسیک برمی‌گردید.

در این فصل کارهای زیادی انجام دادید، و طرز کار با منو و دیالوگ را یاد گرفتید. اما هنوز برای آنکه یک برنامهٔ واقعی بنویسید، راه زیادی در پیش دارید – اما، قدم بقدم جلو خواهیم رفت.

اضافه کردن دیالوگ‌های غیراستاندارد به برنامه

اگر دیالوگی بخواهید که شبیه هیچیک از دیالوگهای استاندارد ویژوال بیسیک نباشد، چکار باید کرد؟ مشکلی نیست - فقط کافیست آستین همت بالا بزنید! برای ایجاد یک دیالوگ غیراستاندارد باید یک فرم معمولی به برنامه اضافه کرده (بعداً خواهید دید، چگونه)، و سپس کنترلهای ورودی/خروجی مورد نیاز را روی آن قرار دهیم. در آخر هم باید ببینیم کاربر کدام دکمه را کلیک کرده، و عکس‌العمل مناسب را نشان دهیم (فصل ۱۵ را ببینید). در فصل آینده با دو دیالوگ سادهٔ دیگر بنامهای InputBox (برای گرفتن ورودی) و MsgBox (برای نمایش پیامهای ساده) آشنا خواهید شد. این دو دیالوگ، در واقع، شکاف بین دیالوگهای استاندارد ویژوال بیسیک و آنهایی که خودمان مجبوریم بسازیم، را پُر می‌کنند.

یک گام فراتر: اضافه کردن میانبُر به فرمانهای منو

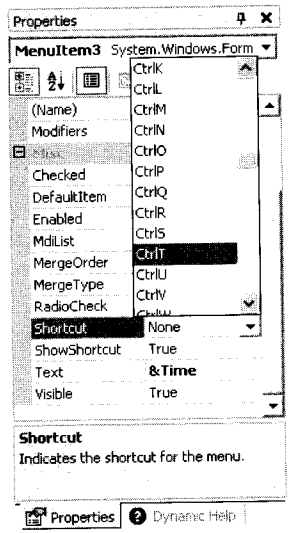
کنترل MainMenu به شما اجازه می‌دهد تا برای فرمانهای منو میانبُر (shortcut) تعریف کنید. میانبر یک کلید ترکیبی است که می‌توان به کمک آن، بدون باز کردن منو، یک فرمان را اجرا کرد. برای مثال، می‌دانید که در برنامهٔ Word بجای فرمان Copy از منوی Edit می‌توان کلید ترکیبی Ctrl+C را بکار برد. برای ست کردن میانبر باید از خاصیت Shortcut کنترل MainMenu استفاده کنیم. در تمرین این قسمت دو میانبر برای فرمانهای منوی Clock تعریف خواهیم کرد.

اضافه کردن میانبُر به منوی

۱ اگر برنامهٔ MyMenu هنوز در حال اجراست، آنرا ببندید، و به محیط برنامه‌نویسی ویژوال بیسیک برگردید.

۲ در منوی Clock فرمان Time را انتخاب کنید. همانطور که گفتم، برای ست کردن میانبر باید از خاصیت Shortcut کنترل MainMenu استفاده کنیم. در لیست خاصیت Shortcut تعداد زیادی میانبر از پیش تعریف شده وجود دارد؛ برای فرمان Time میانبر Ctrl+T را بکار خواهیم برد.

۳ در پنجرهٔ خواص، خاصیت Shortcut را انتخاب کرده، و لیست مقابل آنرا باز کنید. در این لیست آیتم CtrlT را انتخاب کنید - شکل زیر را ببینید:



نکته

هنگام اجرای برنامه، ویژوال بیسیک کلید میانبر هر فرمان را در مقابل آن نشان می‌دهد، تا کاربر بتواند آنرا تشخیص دهد و در دفعات بعد از آن استفاده کنید. اگر به هر دلیل (مثلاً کم آمدن منابع سیستم) نمی‌خواهید این میانبرها دیده شوند، می‌توانید خاصیت ShowShortcut کنترل MainMenu را به False ست کنید. توجه داشته باشید که در این حالت میانبرها همچنان فعال هستند، فقط کاربر دیگر آنها را نخواهد دید.

۴ فرمان Date را انتخاب کرده، و خاصیت Shortcut آنرا به CtrlD ست کنید.

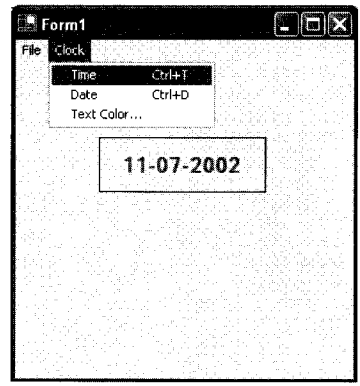
اکنون برای تست میانبرها، برنامه را اجرا می‌کنیم.

۵ با کلیک کردن دکمهٔ Start برنامه را اجرا کنید.

۶ کلید ترکیبی Ctrl+T را بزنید، تا فرمان Time اجرا شده، و برنامه وقت فعلی را نمایش دهد.

۷ کلید ترکیبی Ctrl+D را بزنید، تا فرمان Date اجرا شده، و برنامه تاریخ فعلی را نشان دهد.

۸ منوی Clock را باز کنید - همانطور که می‌بینید، ویژوال بیسیک میانبر فرمانهای Time و Date را جلوی آنها نقش کرده است:



۹ با انتخاب فرمان Exit از منوی File ، برنامه را ببندید.

مرجع سریع فصل ٤

برای ...	انجام دهید
ایجاد یک آیتم منو	کنترل MainMenu را انتخاب کرده، و یک منو روی فرم برنامه رسم کنید. سپس، در قسمت Type Here کلیک کرده، و نام منو را در آنجا وارد کنید.
اضافه کردن کلید دسترسی سریع به یک فرمان	روی آیتم مورد نظر دو بار پشت سر هم (ولی با فاصلهٔ زمانی کافی) کلیک کنید، تا کرسر ادیت در آن ظاهر شود. قبل از حرفی که می‌خواهید تبدیل به کلید دسترسی سریع این فرمان شود، یک کاراکتر & قرار دهید.
اضافه کردن کلید میانبر به یک فرمان	فرمان مورد نظر را انتخاب کرده، و میانبر دلخواه را از لیست خاصیت Shortcut (در پنجرهٔ خواص) برای آن ست کنید.
عوض کردن جای آیتمهای منو	آیتم مورد نظر را گرفته، و محل دلخواه بکشید.
اضافه کردن دیالوگهای استاندارد به برنامه	یکی از هفت دیالوگ استاندارد ویژوال بیسیک را روی فرم برنامه قرار داده، و خواص آنرا ست کنید.
نمایش دیالوگ «باز کردن فایل»	یک دیالوگ OpenFileDialog روی فرم برنامه قرار داده، و در موقع مقتضی با متد ShowDialog آنرا باز کنید.
نمایش دیالوگ «انتخاب رنگ»	یک دیالوگ ColorDialog روی فرم برنامه قرار داده، و در موقع مقتضی با متد ShowDialog آنرا باز کنید.
غیرفعال کردن یک منو	خاصیت Enabled آیتم مورد نظر را (در پنجرهٔ خواص) به False ست کنید.
فعال کردن یک منودر کُد برنامه	از دستور *mnuMenuName*.Enabled = True استفاده کنید.
پاک کردن تصویر یک جعبه تصویر	از دستور *PictureBoxName*.Image = Nothing استفاده کنید.

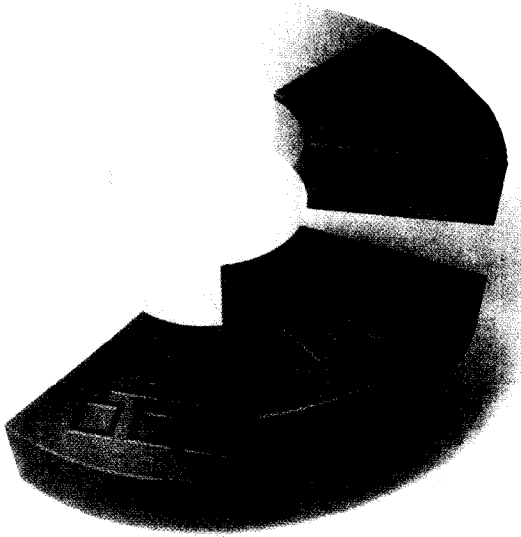
VISUAL BASIC .NET

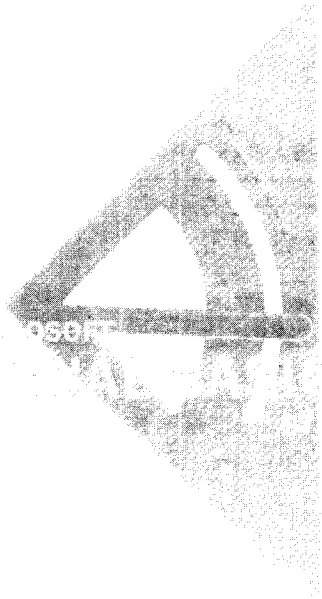
آموزش گام به گام

۲

بخش

مبانی برنامه‌نویسی





فصل

۵

متغیرها و عملگرها در ویژوال بیسیک .NET

در این فصل یاد می گیرید چگونه :

- ✔ برای ذخیره کردن داده‌های برنامه از متغیر (variable) استفاده کنید.
- ✔ با استفاده از تابع InputBox ورودی را از کاربر بگیرید.
- ✔ با تابع MsgBox پیامهای خود را به کاربر نشان دهید.
- ✔ با انواع مختلف داده کار کنید.
- ✔ با ترکیب توابع و عملگرهای ریاضی ویژوال بیسیک، محاسبات مورد نیاز را انجام دهید.
- ✔ با متدهای ریاضی کلاس System.Math کار کنید.

در بخش اول این کتاب دیدید که چگونه می‌توان یک برنامهٔ ساده را در محیط ویژوال بیسیک .NET نوشت و اجرا کرد. در این بخش (که از پنج فصل تشکیل شده) با اصول برنامه‌نویسی ویژوال بیسیک آشنا می‌شوید، و در پایان آماده خواهید بود تا سرفصل‌های پیشرفته‌تر را شروع کنید.

در این فصل یاد می‌گیرید که چگونه داده‌های موقتی برنامه را در متغیرها ذخیره کرده، و با استفاده از توابع و عملگرهای ریاضی محاسبات مورد نیاز را انجام دهید. با دو تابع مفید بنامهای InputBox و MsgBox نیز آشنا شده، و به کمک آنها اطلاعات را با کاربر رد و بدل می‌کنید. و بالاخره، خواهید دید که چگونه می‌توان از **کلاس‌های چارچوب .NET** .NET Framework (classes) برای انجام کارهای واقعی و سودمند در برنامه استفاده کرد.

تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- برای استفاده از متغیرها در ویژوال بیسیک .NET بایستی آنها را قبل تعریف کرده باشید. اگر می‌خواهید بدون تعریف متغیرها، آنها را بکار ببرید، باید قبلاً دستور Option Explicit Off را اجرا کنید - کاری که بهیچوجه توصیه نمی‌شود.
- ویژوال بیسیک .NET دیگر از نوع دادهٔ واریانت (Variant) پشتیبانی نمی‌کند؛ نوع دادهٔ هر متغیری باید بصورت صریح (با دستورات Dim و As) تعریف شود.
- در ویژوال بیسیک .NET چند نوع دادهٔ جدید اضافه شده، و محدودهٔ عمل چند تای دیگر تغییر کرده است. برای مثال، نوع دادهٔ Integer ۳۲ بیتی است، و متغیرهای نوع Short و Long بترتیب ۱۶ و ۶۴ بیتی خواهند بود. بجای نوع دادهٔ قدیمی Currency هم، در ویژوال بیسیک .NET باید از نوع دادهٔ Decimal استفاده کنید.
- در ویژوال بیسیک .NET دستور جدیدی اضافه شده، بنام Option Strict . اگر این دستور On باشد، فقط متغیرهای همنوع را می‌توان با هم ترکیب، مقایسه و یا اضافه کرد. (گاهی از متغیرهای غیر-همنوع هم می‌توان استفاده کرد، مشروط بر اینکه داده‌ای از بین نرود.) این بدان معناست که تبدیل نوع (Type Cast) در ویژوال بیسیک .NET از اهمیت بیشتری نسبت به ویژوال بیسیک ۶ برخوردار است (و این یعنی استفادهٔ گسترده‌تر از توابعی مانند CInt ، CLng و CType). البته می‌توانید با Off کردن این ویژگی مانند قبل عمل کنید (بویژه وقتی می‌خواهید یک برنامهٔ قدیمی را در ویژوال بیسیک .NET کامپایل کنید)؛ اما سعی کنید از این حالت کمتر استفاده کنید، چون احتمال دارد ویژوال بیسیک .NET در آینده از آن پشتیبانی نکند.
- در ویژوال بیسیک .NET برخی از محاسبات ریاضی را می‌توان بصورت فشرده‌تر انجام داد. برای مثال، فرمول X = X + 2 را می‌توان بصورت X += 2 نوشت.
- در ویژوال بیسیک .NET دیگر توابع ریاضی داخلی از قبیل Cos و Abs وجود ندارد؛ بجای آنها باید از متدهای کلاس System.Math برای انجام محاسبات ریاضی استفاده کنید. در اغلب موارد نام این متدها مشابه نام توابع ویژوال بیسیک ۶ است (البته استثناهایی هم وجود دارد: مثلاً، Sqr به Sqrt تغییر کرده است).
- با اینکه در ویژوال بیسیک .NET هنوز از شیء MessageBox پشتیبانی می‌شود، اما بهتر است بجای آن از تابع MsgBox استفاده کنید. برای نمایش پیام با شیء MessageBox می‌توانید متد MessageBox.Show را بکار ببرید.

آناتومی یک دستور ویژوال بیسیک

در فصل ۲ دیدید که به هر خط کُد در یک برنامهٔ ویژوال بیسیک، **دستور** (program statement) گفته می‌شود. یک دستور عبارتست از ترکیبی از کلمات کلیدی، خواص، متدها، توابع، و عملگرهای ویژوال بیسیک، که حاصل آن برای کامپایلر ویژوال بیسیک شناخته شده باشد. یک دستور می‌تواند از فقط یک کلمهٔ کلیدی تشکیل شده باشد، مانند

End

که به برنامه پایان می‌دهد؛ یا ترکیبی باشد از چند خاصیت و کلمهٔ کلیدی، مانند دستور زیر که وقت فعلی را در برچسب Label1 نمایش می‌دهد:

```
Label1.Text = TimeString
```

به قواعد حاکم بر دستورات برنامه‌نویسی، **قواعد دستوری** (syntax) گفته می‌شود. برنامه‌نویسی اصولاً چیزی نیست جز تسلط بر قواعد دستوری یک زبان برنامه‌نویسی، و استفاده از این قواعد برای پردازش داده‌ها. قواعد دستوری ویژوال بیسیک با بسیاری از زبانهای برنامه‌نویسی دیگر مشترک است. خوشبختانه، ویژوال بیسیک مقدار زیادی از بار برنامه‌نویسی را بر دوش گرفته، و به همین دلیل می‌توانید در زمانی نسبتاً کوتاه برنامه‌های موردنظر خود را بنویسید – و حتی در برنامه‌های دیگر از کُدهایی که قبلاً نوشته‌اید، استفاده کنید.

در فصلهای آینده با بسیاری از کلمات کلیدی و دستورات ویژوال بیسیک ، و اشیاء، خواص و متدهای موجود در ویژوال استودیو و چارچوب NET. آشنا شده، و به کمک آنها برنامه‌های خوبی خواهیم نوشت. اما شاید هیچیک از آنها مهمتر و کلیدی‌تر از متغیرها و انواع داده (data type) نباشند؛ پس اجازه دهید با اینها شروع کنیم.

استفاده از متغیر برای ذخیره کردن اطلاعات

متغیر محلیست برای ذخیره کردن موقتی داده‌ها در برنامه. تقریباً هر چیزی را می‌توان در متغیرها ذخیره کرد، و می‌توان به هر تعداد که نیاز است، در برنامه از آنها استفاده کرد. در واقع، متغیرها به هر قطعه از داده‌ها نامی با معنا می‌دهند.

برای استفاده از متغیرها در ویژوال بیسیک .NET به مقدماتی نیاز داریم. قبل از هر چیز باید جایی در حافظه به آنها اختصاص دهیم (مانند رزرو کردن صندلی در سینما). به این عمل **تعریف متغیر** (variable declaring) گفته می‌شود.

تعریف متغیر: دستور Dim

در ویژوال بیسیک .NET قبل از استفاده از یک متغیر باید آنرا تعریف کنید –این وضعیت با ویرایشهای قبلی ویژوال بیسیک، که در مواقعی می‌توانستید بدون تعریف یک متغیر از آن استفاده کنید، متفاوت است. روش قبلی، با اینکه باعث راحتی کار برنامه‌نویسان می‌شد، اما خطراتی در پی داشت. مثلاً، اگر در نوشتن نام یک متغیر اشتباه می‌کردید، ویژوال بیسیک آنرا متغیر جدیدی فرض می‌کرد، و بدون هیچگونه هشداری به کار خود ادامه می‌داد، و این می‌توانست برنامه‌نویس را مدتها سر در گم کند.

برای تعریف یک متغیر باید از دستور Dim استفاده کنیم (Dim مخفف dimension است)؛ این دستور فضای لازم را در حافظه به متغیر اختصاص می‌دهد، و به ویژوال بیسیک می‌گوید که چه نوع داده‌ای را می‌تواند در آن ذخیره کند. تعریف یک متغیر را در هر نقطه‌ای از کُد برنامه می‌توان انجام داد (مشروط به

اینکه قبل از اولین استفاده از آن باشد)، ولی اغلب برنامه‌نویسان ترجیح می‌دهند آنها را در یک محل (و در شروع کُد) تعریف کنند. دستور زیر متغیری بنام LastName از نوع رشته (یا متن) تعریف می‌کند:

```
Dim LastName As String
```

در اینجا کلمهٔ کلیدی As مشخص می‌کند که این متغیر از چه نوعی باید باشد (در همین فصل با انواع داده هم آشنا خواهید شد). نوع دادهٔ String کاربرد بسیار زیادی در برنامه‌های ویژوال بیسیک دارد، چون می‌تواند (تقریباً) هر نوع داده‌ای را در خود جای دهد - البته کاربرد اصلی آن ذخیره کردن داده‌های متنی است.

اما چرا باید متغیرها را تعریف کنیم؟ علت آنست که کامپایلر ویژوال بیسیک قبل از اینکه بتواند به یک متغیر حافظه اختصاص دهد، بایستی نام متغیر و نوع داده‌ای که می‌خواهد در آن ذخیره شود، را بداند. شاید مدیریت حافظه برای کامپیوترهای امروزی که دهها مگابایت حافظهٔ RAM دارند، چندان ضروری بنظر نرسد، اما از طرف دیگر برنامه‌ها هم بسیار حریص و پُرخور شده‌اند، و بهر حال نباید این موضوع را نادیده گرفت. همانطور که خواهید دید، مقدار حافظه‌ای که هر نوع متغیر نیاز دارد، متفاوت است.

نکته

در ویرایشهای بسیار قدیمی بیسیک، اساساً چیزی بنام انواع متغیر وجود نداشت، و تمام متغیرها در یک نوع خاص (بنام Variant - که تمام انواع داده را می‌توانست در خود جای دهد، و بسیار حافظه‌خور هم بود) ذخیره می‌شدند. ویژوال بیسیک .NET علیرغم اینکه برنامه‌نویسان مبتدی با آن بسیار راحت هستند، دیگر از نوع دادهٔ Variant پشتیبانی نمی‌کند، چون کارایی برنامه را بشدت پائین می‌آورد، و باعث بروز رفتارهای غیرعادی در آن می‌شود.

بعد از اینکه متغیر را تعریف کردید، می‌توانید به آن مقدار دهید:

```
LastName = "Jefferson"
```

توجه کنید که متغیر LastName یک متغیر رشته‌ای است، و داده‌ای که در آن ذخیره می‌کنید باید از این نوع باشد. البته هر چیزی (حتی اعداد) را می‌توان در متغیرهای رشته‌ای ذخیره کرد، ولی بهر حال آنها متن هستند. به مثال زیر نگاه کنید:

```
Address = "1313 Mockingbird Lane"
```

روی اعدادی که بصورت رشته ذخیره شده باشند، نمی‌توان محاسبات ریاضی انجام داد، مگر اینکه دوباره (با توابع تبدیل نوع) آنها را به عدد تبدیل کنید.

وقتی به یک متغیر مقدار دادید، می‌توانید از آن بجای دادهٔ درون آن استفاده کنید:

```
Label1.Text = LastName
```

بعد از اجرای این دستور، کنترل Label1 کلمهٔ *Jefferson* را نشان خواهد داد.

نکته

اگر هنوز به روش قدیمی ویژوال بیسیک علاقه دارید، و نمی‌خواهید متغیرها را صریحاً با دستور Dim تعریف کنید، می‌توانید در ابتدای برنامه (قبل اجرای هر دستور دیگر) دستور زیر را اجرا کنید:

```
Option Explicit Off
```

استفاده از متغیرها در برنامه

مقدار ذخیره شده در یک متغیر را می‌توان در طول برنامه (و هر گاه لازم است) تغییر داد (و اساساً نام متغیر هم از همین جا می‌آید). در تمرین زیر خواهید دید که چگونه می‌توان در طول برنامه مقدار یک متغیر را بدفعات تغییر داد.

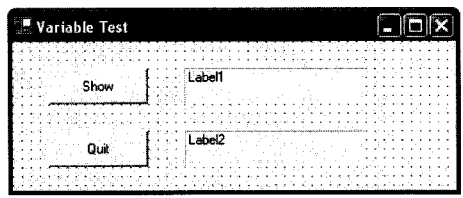
تغییر دادن مقدار یک متغیر

۱ ویژوال استودیو .NET را باز کنید.

۲ از منوی File|Open آیتم Project را انتخاب کنید، تا پنجرهٔ «بازکردن پروژه» ظاهر شود.

۳ در پوشهٔ c:\vbnetsbs\chap05\variable ، پروژهٔ Variable Test را باز کنید.

٤ اگر فرم پروژه را نمی‌بینید، فرم Form1.vb را در کاوشگر راه‌حل انتخاب کرده، و روی دکمهٔ View Designer کلیک کنید. شکل زیر را ببینید:



روی این فرم دو دکمه و دو برچسب (که برای نمایش مقدار متغیرها از آنها استفاده می‌کنیم) می‌بینید. در این پروژه هیچ کُدی وجود ندارد -کُدهای آنرا در این تمرین با هم می‌نویسیم.

نکته

برچسب‌ها حالت سه‌بُعدی دارند، چون خاصیت BorderStyle آنها را به ست Fixed3D کرده‌ام.

۵ روی دکمهٔ Show دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد باز شود.

۶ دستورات زیر را در این روال وارد کنید:

```
Dim LastName As String

LastName = "Luther"
```

```
Label1.Text = LastName

LastName = "Bodenstein von Karlstadt"
Label2.Text = LastName
```

این دستورات به سه گروه تقسیم شده‌اند. در خط اول با دستور Dim یک متغیر از نوع String (رشته‌ای) بنام LastName تعریف کرده‌ایم. در خط دوم و سوم عبارت "Luther" به متغیر LastName داده شده، و سپس در برچسب Label1 نشان داده می‌شود. این کار –انتقال اطلاعات به یک خاصیت – یکی از متداولترین کاربردهای متغیرهاست. خط چهارم عبارت "Bodenstein von Karlstadt" را به متغیر LastName می‌دهد –بعبارت دیگر، مقدار آنرا تغییر می‌دهد.

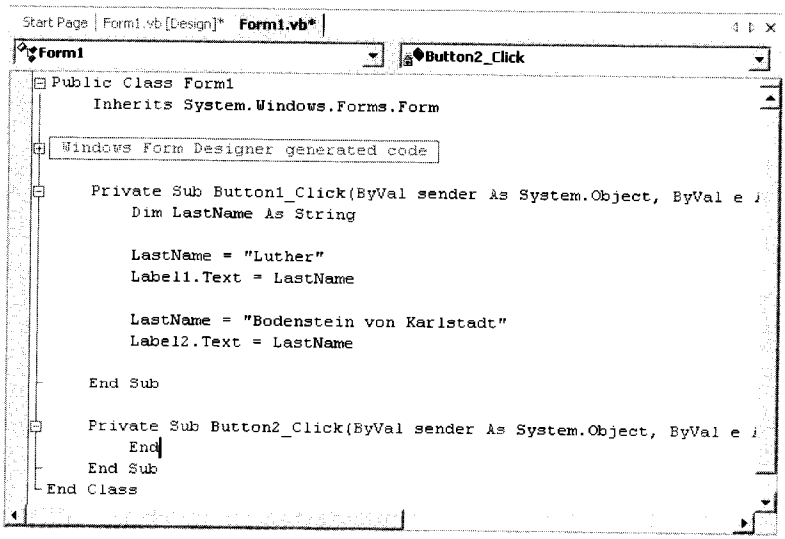
یکی از نکات بسیار مهمی که باید دربارهٔ متغیرهایی که در یک روال تعریف می‌شوند، بخاطر بسپارید، اینست که «**این متغیرها فقط در همان روال دیده می‌شوند، و نمی‌توان در جاهای دیگر از آنها استفاده کرد**». بزودی خواهید دید چگونه می‌توان متغیرهایی تعریف کرد که در تمام برنامه و فرمهای آن بتوان از آنها استفاده کرد.

۷ روی برگهٔ Form1.vb [Design] کلیک کنید، و به فرم برنامه برگردید.

۸ روی دکمهٔ Quit دو –کلیک کنید، تا روال رویداد Button2_Click در ادیتور کُد باز شود.

۹ دستور زیر را در این روال وارد کنید (شکل زیر را ببینید):

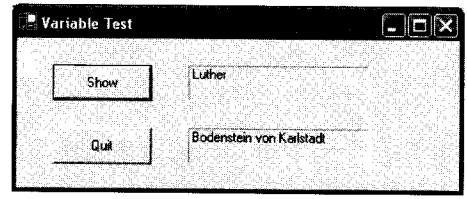
```
End
```



۱۰ با کلیک کردن دکمهٔ Save All ، پروژه را ذخیره کنید.

۱۱ دکمهٔ Start را کلیک کنید، تا پروژه کامپایل و اجرا شود.

۱۲ روی دکمهٔ Show کلیک کنید. برنامه پس از تعریف متغیر LastNAme ، دو مقدار به آن داده و آنها را در برچسب‌ها نشان می‌دهد:



قراردادهای نامگذاری متغیرها

نامگذاری متغیرها، آنچنان که در نگاه اول بنظر می‌رسد، کار چندان ساده‌ای هم نیست. نام یک متغیر باید در عین وضوح و بیان هدف آن، تا حد امکان کوتاه باشد. در زیر قواعد نامگذاری متغیرها را می‌بینید:

- نام یک متغیر باید با یک حرف الفبا شروع شود (_ [زیرخط] هم جزء حروف الفبا محسوب می‌شود). این یکی از الزامات ویژوال بیسیک است. نام متغیرها فقط می‌تواند از حروف الفبا، اعداد و زیرخط [_] تشکیل شود.
- نام متغیرها در ویژوال بیسیک .NET هیچ محدودیتی از نظر طول ندارد، اما سعی کنید طول آنها از ۳۳ حرف بیشتر نشود.
- سعی کنید از نامهای چند کلمه‌ای و با معنا برای متغیرهای خود استفاده کنید. مثلاً، نام متغیر SalesTaxRate بسیار واضحتر از Tax یا Rate است.
- در ترکیب نام متغیر از حروف بزرگ و کوچک استفاده کنید. معمولاً رسم بر اینست که حرف اول هر کلمه از نام متغیر بصورت بزرگ، و سایر حروف بصورت کوچک نوشته می‌شود (مثلاً، DateOfBirth). در روش دیگری، که به شیوهٔ «کوهان شتری» معروف است، حرف اول نام متغیر همیشه کوچک نوشته می‌شود (مثلاً، dateOfBirth یا employeeName) - با این کار نام متغیرها از نام توابع و روال‌ها متمایز می‌شود.
- از نامهای رزرو شدهٔ ویژوال بیسیک .NET (کلمات کلیدی، نام اشیاء، خواص و متدها) بعنوان نام متغیر استفاده نکنید، چون در اینصورت با خطا مواجه خواهید شد.
- (اختیاری) نام متغیر را با یک عبارت دو یا سه حرفی، که نشاندهندهٔ نوع متغیر است، آغاز کنید. برای مثال، نام متغیر strName نشان می‌دهد که این متغیری از نوع String (رشته‌ای) است، یا intCounter متغیری از نوع Integer (عدد صحیح) است. این روش، که توسط چارلز سیمونی ابداع شده، به «شیوهٔ نامگذاری مجارستانی» معروف است.

۱۳ دکمهٔ Quit را کلیک کنید، تا برنامه بسته شود.

استفاده از متغیر برای ذخیره کردن اطلاعات ورودی

یکی از موارد کاربرد متغیرها، ذخیره کردن اطلاعاتیست که کاربر وارد می‌کند. قبلاً دیدید که این کار را با

کنترلهای جعبه متن و جعبه لیست هم می‌توان انجام داد، ولی گاهی استفاده از متغیرها برای این منظور ساده‌تر و مؤثرتر است. یکی از روشهای متداول گرفتن ورودی از کاربر، استفاده از تابع InputBox (و ذخیره کردن مقدار برگشتی آن در متغیر موردنظر) است. این روش را در تمرین زیر خواهید دید.

گرفتن ورودی با InputBox

۱ از منوی File|Open آیتم Project را انتخاب کنید، تا پنجرهٔ «بازکردن پروژه» ظاهر شود.

۲ در پوشهٔ c:\vbnetsbs\chap05\input box ، پروژهٔ Input Box را باز کنید. روی فرم این پروژه دو دکمه و یک برچسب می‌بینید، اما کُد آنرا خودتان باید وارد کنید.

۳ اگر فرم پروژه را نمی‌بینید، فرم Form1.vb را در کاوشگر راه‌حل انتخاب کرده، و روی دکمهٔ View Designer کلیک کنید.

٤ روی دکمهٔ Input Box دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد باز شود.

۵ دستورات زیر را در این روال وارد کنید:

```
Dim Prompt, FullName As String
Prompt = "Please enter your name."

FullName = InputBox(Prompt)
Label1.Text = FullName
```

در خط اول همزمان و با یک دستور Dim ، دو متغیر رشته‌ای تعریف شده‌اند: Prompt و FullName. مادامیکه چند متغیر از یک نوع باشند، می‌توانید آنها را با یک دستور Dim تعریف کنید. دستور فوق در ویژوال بیسیک ۶ به نتیجهٔ متفاوتی می‌انجامد: متغیر Prompt از نوع Variant و FullName از نوع String تعریف می‌شود، چون نوع دادهٔ متغیر Prompt صریحاً مشخص نشده است. خوشبختانه، این تناقض منطقی در ویژوال بیسیک .NET رفع شده است.

در خط دوم به متغیر Prompt مقدار داده‌ایم؛ از این عبارت بعنوان آرگومان ورودی تابع InputBox استفاده خواهیم کرد. (آرگومان - argument - مقدار یا عبارتیست که به یک تابع یا روال فرستاده می‌شود.) خط سوم تابع InputBox را فراخوانی کرده، و مقدار برگشتی آنرا در متغیر FullName می‌نویسد. (تابع InputBox غیر از Prompt آرگومانهای دیگری نیز می‌گیرد، که می‌توانید در سیستم کمک ویژوال بیسیک آنها را ببینید.)

و بالاخره، دستور چهارم مقدار برگشتی InputBox را در برچسب Label1 نمایش می‌دهد.

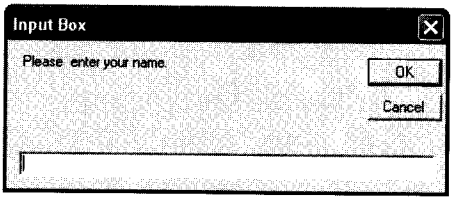
نکته

در ویرایشهای قدیمی‌تر بیسیک، در انتهای تابع InputBox یک کاراکتر $ وجود داشت، که نشان می‌داد مقدار برگشتی این تابع از نوع String است. متغیرهای رشته‌ای را هم می‌توانستید با استفاده از کاراکتر $ تعریف کنید. هر نوع داده یک علامت خاص داشت: برای مثال، کاراکتر % علامت نوع Integer ، کاراکتر ! علامت نوع Single ، و کاراکتر # علامت نوع Double بود. در ویژوال بیسیک .NET دیگر این کاراکترهای تعریف نوع داده وجود ندارند.

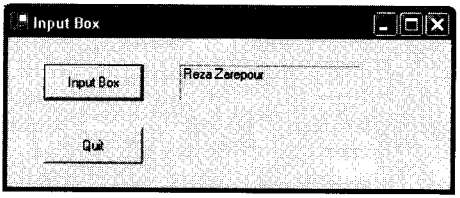
۶ پروژه را ذخیره کنید. (دیگر باید بدانید کدام دکمه را کلیک کنید؛ نه؟)

۷ دکمهٔ Start را کلیک کنید، تا پروژه کامپایل و اجرا شود.

۸ روی دکمهٔ Input Box کلیک کنید. با اجرای روال Button1_Click ، یک دیالوگ InputBox روی صفحه ظاهر می‌شود:



۹ نام کامل خود را وارد کرده، و دکمهٔ OK را کلیک کنید. تابع InputBox نام شما را برگردانده، و در متغیر FullName قرار می‌دهد - و این همان چیزیست که در برچسب خواهید دید.



تابع InputBox یکی از بهترین ابزارهای ویژوال بیسیک برای گرفتن اطلاعات از کاربر است، که می‌توانید همه جا از آن استفاده کنید.

۱۰ دکمهٔ Quit را کلیک کنید، تا برنامه بسته شود.

تابع چیست؟

در تمرین فوق، چند بار از اصطلاح «تابع» استفاده کردم. تابع (function) قطعه کُدیست که کاری خاص و با معنی (مثلاً، محاسبهٔ یک فرمول) انجام داده، و نتیجهٔ حاصله را به برنامه برمی‌گرداند. هر تابع می‌تواند تعدادی آرگومان ورودی نیز بگیرد (که البته الزامی هم نیست) - تابع InputBox جزء توابعیست که می‌تواند چند آرگومان ورودی بگیرد. وقتی تابعی آرگومان می‌گیرد، آنها را در داخل یک جفت پرانتز قرار می‌دهیم (و اگر چند تا باشند، آنها را با کاما - , - از هم جدا می‌کنیم). به مثال زیر نگاه کنید:

```
FullName = InputBox(Prompt, Title)
```

استفاده از متغیر برای نمایش خروجی

مقدار متغیرها را می‌توان با نسبت دادن آنها به یک خاصیت (مثلاً، خاصیت Text برچسب)، و یا با استفاده از یک دیالوگ نمایش داد. یکی از دیالوگهای مناسب برای این منظور، تابع MsgBox است. مانند InputBox ،

تابع MsgBox هم می‌تواند آرگومانهای ورودی مختلفی بگیرد، و حتی یک مقدار برگشتی هم دارد، که می‌توان آنرا در یک متغیر ذخیره کرد. شکل کلی تابع MsgBox چنین است:

```
ButtonClicked = MsgBox(Prompt, Buttons, Title)
```

که در آن، *Prompt* پیامیست که در جعبه پیام نشان داده می‌شود، *Buttons* عددیست که مشخص می‌کند چه دکمه‌ها و آیکونی باید در جعبه پیام نمایش داده می‌شود، و *Title* عبارتیست که در میلهٔ عنوان جعبه پیام دیده خواهد شد - *ButtonClicked* نیز متغیریست که مقدار برگشتی تابع MsgBox در آن ذخیره می‌شود.

اگر فقط می‌خواهید یک پیام ساده نشان دهید، می‌توانید متغیر *ButtonClicked* ، و آرگومانهای *Buttons* و *Title* را حذف کنید. (در تمرین زیر فقط از آرگومانهای *Prompt* و *Title* استفاده خواهیم کرد؛ برای آگاهی از طرز کار آرگومانهای دیگر، به سیستم کمک ویژوال بیسیک - تحت عنوان *MsgBox* - مراجعه کنید.)

نکته

در ویژوال بیسیک .NET کلاس دیگری بنام MessageBox نیز وجود دارد، که می‌توانید برای نشان دادن پیام از آن استفاده کنید. کلاس MessageBox عضوی از کتابخانهٔ System.Windows.Forms است، و با متد MessageBox.Show به نمایش در می‌آید. آرگومانهای MessageBox بسیار شبیه تابع MsgBox است.

نمایش پیام با MsgBox

۱ روی دکمهٔ Input Box دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد باز شود.

۲ دستور زیر را، که آخرین دستور روال Button1_Click است، انتخاب کنید:

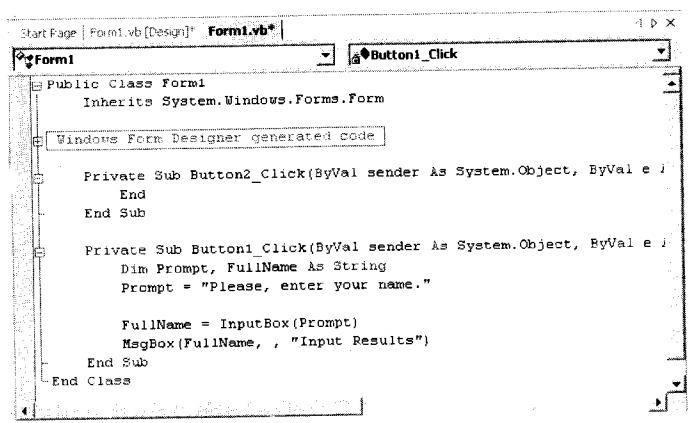
```
Label1.Text = FullName
```

۳ با زدن کلید Del ، این دستور را حذف کنید.

٤ بجای آن، دستور زیر را وارد کنید:

```
MsgBox(FullName, , "Input Results")
```

این دستور جدید، با فراخوانی تابع MsgBox ، محتویات متغیر FullName را در یک دیالوگ (با عنوان Input Results) نمایش می‌دهد. شکل زیر را ببینید:

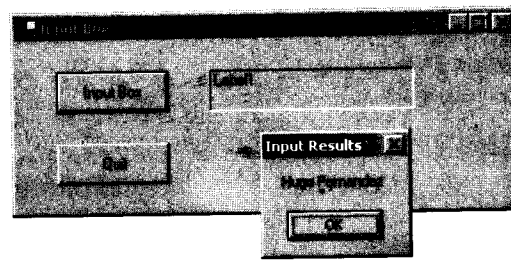


(در اینجا، وجود آرگومان *Buttons* و متغیر *ButtonClicked* ضروری نیست، بنابراین آنها را حذف کرده‌ایم.)

۵ دکمهٔ Start را کلیک کنید، تا پروژه کامپایل و اجرا شود.

۶ روی دکمهٔ Input Box کلیک کرده، و پس از واردکردن نام خود در دیالوگ InputBox ، OK را کلیک کنید.

ویژوال بیسیک نام شما را در متغیر FullName ذخیره کرده، و سپس آنرا در یک جعبه پیام نمایش می‌دهد؛ شکل زیر را ببینید:



۷ OK را کلیک کنید، تا جعبه پیام بسته شود؛ سپس با کلیک کردن Quit ، برنامه را نیز ببندید.

نکته

اگر دقت کرده باشید، در تمرین بالا مرحلهٔ «ذخیره کردن پروژه» را انجام ندادیم. یکی از ویژگیهای امنیتی ویژوال استودیو .NET اینست که قبل از کامپایل کردن هر پروژه، بطور خودکار آنرا ذخیره می‌کند، تا جلوی از بین رفتن احتمالی پروژه گرفته شود.

کار با انواع داده

نوع دادهٔ رشته (String) برای کار با متن خوب بود، ولی با اعداد، تاریخ‌ها، و انواع دیگر اطلاعات چکار کنیم؟ برای مدیریت بهینهٔ حافظه در تمام انواع داده، ویژوال بیسیک .NET چندین نوع دادهٔ جدید (بویژه برای کار در محیطهای ۶۴ بیتی) عرضه کرده است.

در جدول ذیل مهمترین انواع دادهٔ ویژوال بیسیک .NET را ملاحظه می‌کنید. انتخاب بهترین نوع داده برای اطلاعاتی که با آن کار می‌کنید (نه خیلی بزرگ، نه خیلی کوچک)، کمک مؤثری به بهینه شدن برنامه‌هایتان خواهد کرد.

نکته

اندازهٔ متغیرها (انواع داده) بر حسب بیت (bit) بیان می‌شود. یک کاراکتر استاندارد آسکی (ASCII) هشت بیت (یک بایت - byte) حافظه اشغال می‌کند، و یک کاراکتر یونی‌کُد (Unicode) ، ۱۶ بیت (دو بایت).

نوع داده	اندازه	محدوده	نمونه
Short	۱۶ بیت	از 32,768- تا 32,767	Dim Birds As Short Birds = 12500
Integer	۳۲ بیت	از 2,147,483,648- تا 2,147,483,647	Dim Insects As Integer Insects = 37500000
Long	۶۴ بیت	از 9,223,372,036,854,775,808- تا 9,223,372,036,854,775,807	Dim WorldPop As Long WorldPop = 4800000004
Single	۳۲ بیت (اعشاری)	از 3.4028235E38- تا 3.4028235E38	Dim Price As Single Price = 899.99
Double	۶۴ بیت (اعشاری)	از 1.79769313486231E308- تا 1.79769313486231E308	Dim Pi As Double Pi = 3.1415926535
Decimal	۱۲۸ بیت	اعدادی تا $\pm 79,228 \times 10^{24}$	Dim Debt As Decimal Debt = 7600300.50
Byte	۸ بیت	از 0 تا 255 (اعداد غیرمنفی)	Dim RetKey As Byte RetKey = 13
Char	۱۶ بیت	هر کاراکتر یونی‌کُد در محدودهٔ 0-65,535	Dim UnicodeChar As Char UnicodeChar = "à"
String	۱۶ بیت بازای هر کاراکتر	0 تا تقریباً 2 میلیارد کاراکتر یونی‌کُد ۱۶ بیتی	Dim Dog As String Dog = "pointer"
Boolean	۱۶ بیت	True (هر مقدار غیرصفر) یا False (هر مقدار معادل صفر)	Dim Flag As Boolean Flag = True
Date	۶۴ بیت	از اول ژانویه 0001 تا ۳۱ دسامبر 9999	Dim Birthday As Date Birthday = #3/1/1963#
Object	۳۲ بیت	هر مقداری که بتوان در نوع دادهٔ Object ذخیره کرد	Dim MyApp As Object MyApp = CreateObject _ ("Word.Application")

استفاده از انواع داده در برنامه

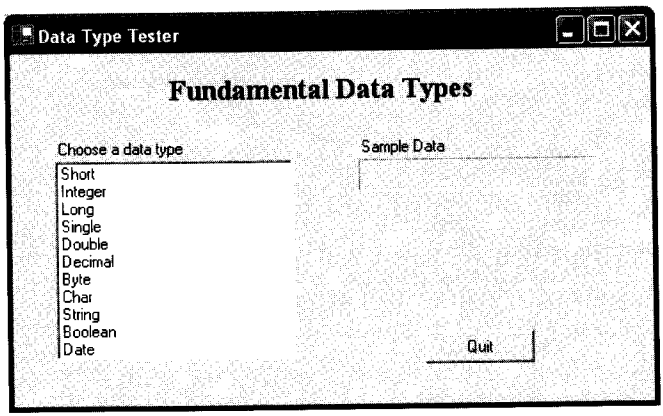
۱ از منوی File|Open آیتم Project را انتخاب کنید، تا پنجرهٔ «بازکردن پروژه» ظاهر شود.

۲ در پوشهٔ c:\vbnetsbs\chap05\data types ، پروژهٔ Data Types را باز کنید.

۳ اگر فرم پروژه را نمی‌بینید، فرم Form1.vb را در کاوشگر راه‌حل انتخاب کرده، و روی دکمهٔ View Designer کلیک کنید.

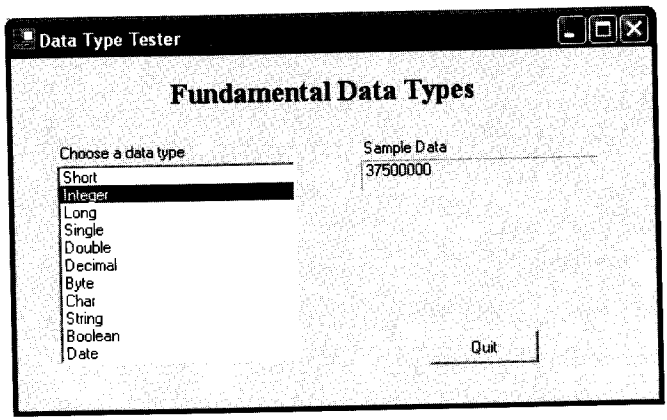
پروژهٔ Data Types یک برنامهٔ کامل است، که انواع دادهٔ ویژوال بیسیک را معرفی می‌کند. برای اجرای این برنامه:

٤ دکمهٔ Start را کلیک کنید، تا پروژه کامپایل و اجرا شود. شکل زیر را ببینید:



یکی از انواع دادهٔ موجود در لیست Choose a data type را انتخاب کنید، تا نمونه‌ای از آنرا در قسمت Sample Data ببینید.

۵ نوع دادهٔ Integer را انتخاب کنید؛ عدد 37,500,000 در برچسب Sample Data نشان داده خواهد شد (البته بدون کاماهای جداکنندهٔ هزار -این کاماها را باید با تابع Format اضافه کنید):



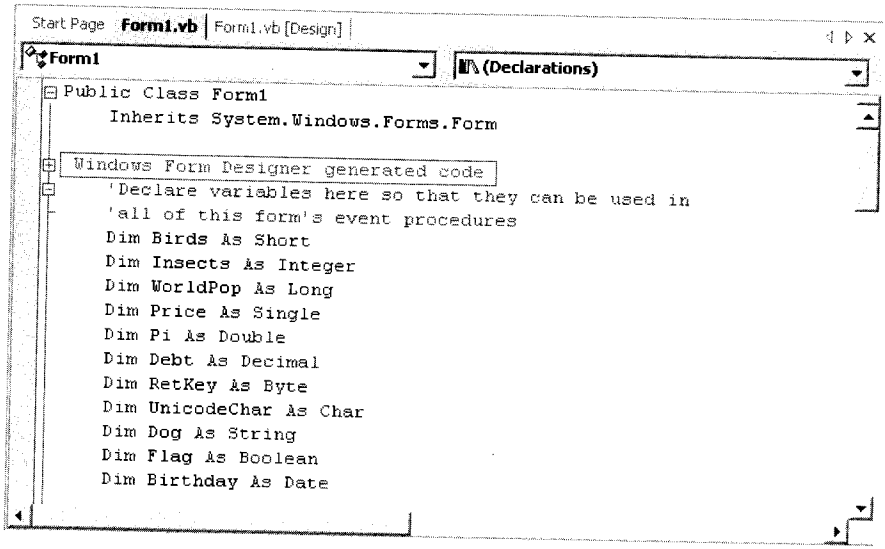
۶ نوع دادهٔ Date را انتخاب کنید؛ تاریخ 3/1/1963 را در برچسب Sample Data خواهد دید.

۷ انواع دادهٔ دیگر را انتخاب کنید، و نمونه اطلاعات هر یک را ببینید.

۸ با کلیک کردن دکمهٔ Quit ، از برنامه خارج شوید.

متغیرهای برنامه در بالای فرم تعریف، و در روال رویداد ListBox1_SelectedIndexChanged مورد استفاده قرار می‌گیرند.

۹ روی فرم برنامه (نه هیچکدام از اشیاء روی آن) دو-کلیک کنید، و پنجرهٔ ادیتور کُد را بزرگتر کنید تا بتوانید مقدار بیشتری از کُد برنامه را ببینید:



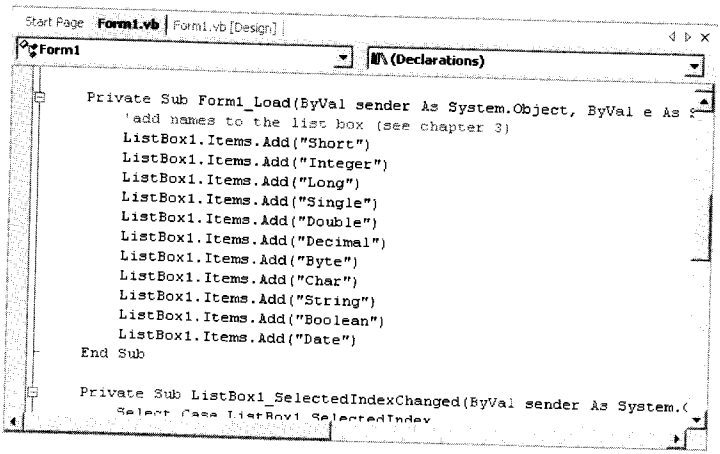
در بالای این کُد خطی می‌بینید با عنوان "Windows Form Designer generated code" ، که اگر روی علامت + کنار آن کلیک کنید، کُدهایی که ویژوال بیسیک به برنامه اضافه کرده، را خواهید دید. این دستورات خواص اشیاء فرم را ست می‌کنند، و باعث می‌شوند تا فرم برنامه بدرستی دیده شود. در ویرایشهای قبلی ویژوال بیسیک این دستورات دیده نمی‌شدند، اما در ویژوال بیسیک .NET حتی می‌توانید این کُدها را دستکاری کنید (البته توصیه می‌کنم قبل از خواندن این کتاب دست به چنین کار خطرناکی نزنید).

در زیر این خط ۱۱ دستور می‌بینید، که در هر یک از آنها یک متغیر از انواع دادهٔ ویژوال بیسیک (بجز نوع دادهٔ Object) تعریف شده است. با تعریف این متغیرها در بالای فرم، آنها در تمام روالهای برنامه دیده خواهند شد. همانطور که قبلاً گفتم، متغیرها فقط در همان روالی که تعریف شده‌اند، قابل دسترسی هستند. اما متغیرهایی که در بالای فرم (خارج از هر روالی) تعریف شده باشند، در تمام روالهای آن فرم قابل دسترسی خواهند بود.

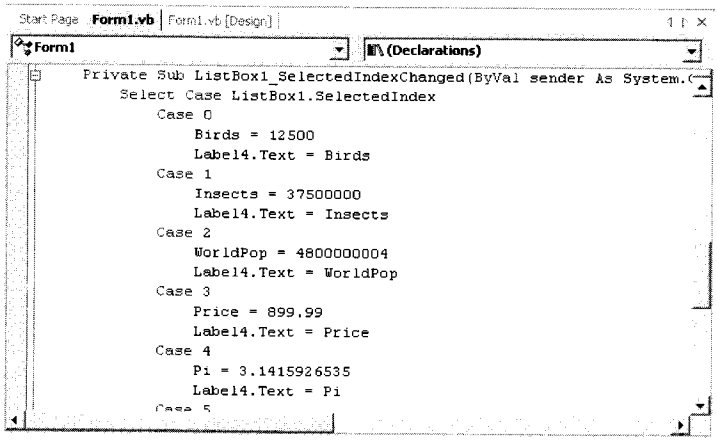
نکته

متغیرهای برنامهٔ Data Types با همان نامهایی تعریف شده‌اند، که در جدول صفحهٔ قبل دیدید. بدین‌ترتیب می‌توانید عملکرد هر یک از این انواع داده را در عمل ببینید.

۱۰ در ادیتور کُد، به روال Form1_Load بروید. در این روال آیتمهای جعبه لیست را به آن اضافه کرده‌ایم:



۱۱ باز هم در ادیتور کُد پائین بروید، تا به روال ListBox1_SelectedIndexChanged برسید. این روال انتخاب کاربر را پردازش می‌کند:



قلب این روال یک ساختار تصمیم‌گیری Select Case است. در فصل آینده با دستورات تصمیم‌گیری بیشتر آشنا خواهید شد؛ اما فعلاً همین قدر توجه کنید که در بخشهای مختلف این دستور Select Case مقداری به هر متغیر داده شده، و سپس در برچسب Label4 نمایش داده می‌شود.

نکته

اگر در پروژه‌ای بیش از یک فرم داشته باشید، و می‌خواهید متغیرهایی تعریف کنید که در تمام فرمهای برنامه دیده شوند، باید یک فایل خاص بنام ماژول کُد (code module) به پروژه اضافه کرده، و متغیرها را (با استفاده از دستور Public) در آن تعریف کنید. به این نوع متغیرها، متغیرهای عمومی (global) گفته می‌شود (برای آشنایی بیشتر با متغیرهای عمومی به فصل ۱۰ مراجعه کنید).

۱۲ دستورات روال ListBox1_SelectedIndexChanged را بدقت بررسی کنید. مقدار نسبت شده به برخی از متغیرها را عوض کرده، و برنامه را از نو اجرا کنید.

بویژه سعی کنید مقداری خارج از محدودهٔ هر یک از انواع داده به آنها بدهید. در این حالت ویژوال بیسیک زیر این مقدار نادرست یک خط زیگزاگ می‌کشد، و تا آنرا درست نکنید، برنامه را اجرا نخواهد کرد. (اگر فکر می‌کنید، مقداری را درست وارد کرده‌اید، ولی ویژوال بیسیک از آن ایراد می‌گیرد، ماوس را روی آن ببرید و چند لحظه نگه دارید - ویژوال بیسیک توضیح می‌دهد که اشکال کار کجاست.)

۱۳ اگر تغییری در برنامه داده‌اید که می‌خواهید ذخیره شود، دکمهٔ Save All را کلیک کنید.

انواع دادهٔ تعریف شده توسط کاربر

ویژوال بیسیک به ما اجازه می‌دهد تا انواع دادهٔ جدیدی ایجاد کنیم. این ویژگی بخصوص در مواقعی که داده‌های ما نه یکی از انواع دادهٔ موجود، بلکه ترکیبی از آنهاست، بسیار مفیدست. برای ایجاد یک نوع دادهٔ جدید (که به آن نوع دادهٔ تعریف شده توسط کاربر - User Defined Type -گفته می‌شود) از دستور Structure استفاده می‌کنیم. از دستور Structure نمی‌توان در روالهای رویداد استفاده کرد - این دستور باید در بالای فرم (خارج از تمام روالها) و یا در ماژول کُد قرار داشته باشد.

در مثال زیر، یک نوع دادهٔ تعریف شده توسط کاربر - یا مختصراً، نوع دادهٔ کاربر - بنام Employee می‌بینید، که در آن نام، تاریخ تولد و تاریخ استخدام یک کارمند را می‌توان ذخیره کرد:

```
Structure Employee
    Dim Name As String
    Dim DateOfBirth As Date
    Dim HireDate As Date
End Structure
```

پس از ایجاد این نوع دادهٔ جدید، می‌توانیم از آن مانند سایر انواع دادهٔ دیگر استفاده کنیم. در دستورات زیر، ابتدا یک متغیر بنام ProductManager از نوع دادهٔ Employee تعریف کرده، و سپس Name آنرا به "Erick Cody" ست کرده‌ایم:

```
Dim ProductManager As Employee
ProductManager.Name = "Erick Cody"
```

شبیه ست کردن خاصیت یک شیء است، اینطور نیست؟ در ویژوال بیسیک برای مقدار دادن (یا خواندن مقدار) آیتمهای یک نوع دادهٔ کاربر از همان روش ست کردن (یا خواندن) خواص اشیاء استفاده می‌شود.

ثابت: متغیری که تغییر نمی‌کند!

اگر متغیری دارید که مقدار آن هرگز عوض نمی‌شود (مانند عدد π در برنامه‌های ریاضی)، می‌توانید بجای آن از یک ثابت (constant) استفاده کنید. یک ثابت، درست مانند متغیر، دارای نامی است که در برنامه با آن

شناخته می‌شود، ولی مقدار آن هیچوقت تغییر نمی‌کند. ثابت‌ها خوانایی برنامه را افزایش داده، اشتباهات برنامه‌نویسی را کاهش می‌دهند، و انجام تغییرات در برنامه را ساده‌تر می‌کنند. برای تعریف یک ثابت از کلمهٔ کلیدی Const استفاده می‌کنیم:

```
Const Pi As Double = 3.14159265
```

این دستور ثابتی بنام Pi ایجاد می‌کند، که می‌توان از آن بجای عدد π در برنامه استفاده کرد. اگر می‌خواهید یک ثابت در تمام روالهای فرم دیده شود، آنرا در بالای فرم (خارج از تمام روالها) تعریف کنید؛ و اگر می‌خواهید در تمام پروژه (و نه فقط یک فرم) دیده شود، باید آنرا با دستور Public (و در ماژول کُد) تعریف کنید:

```
Public Const Pi As Double = 3.14159265
```

در تمرین زیر طرز استفاده از یک ثابت در برنامه را خواهید دید.

استفاده از ثابت در برنامه

۱ از منوی File|Open آیتم Project را انتخاب کنید، تا پنجرهٔ «بازکردن پروژه» ظاهر شود.

۲ در پوشهٔ c:\vbnetsbs\chap05\constant tester ، پروژهٔ Constant Tester را باز کنید.

۳ اگر فرم پروژه را نمی‌بینید، فرم Form1.vb را در کاوشگر راه‌حل انتخاب کرده، و روی دکمهٔ View Designer کلیک کنید.

پروژهٔ Constant Tester فقط واسط کاربر برنامه را دارد -کُد آنرا با هم خواهیم نوشت.

٤ روی دکمهٔ Show Constant دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد باز شود.

۵ دستورات زیر را در روال Button1_Click وارد کنید:

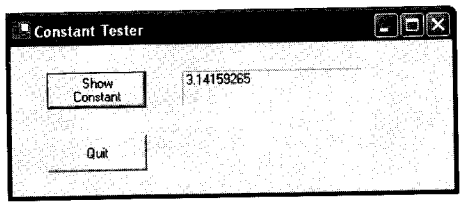
```
Const Pi As Double = 3.14159265
Label1.Text = Pi
```

نکته

محل تعریف ثابت‌ها (یا متغیرها) به میدان دید (scope) مورد نیاز آنها بستگی دارد. سعی کنید میدان دید آنها را بیش از آنچه نیاز دارید، بزرگ نکنید. برای مثال، اگر ثابتی فقط در یک روال مورد نیاز است، آنرا در همان روال تعریف کنید. تعریف یک ثابت در بالای فرم (خارج از تمام روالها) باعث می‌شود آن ثابت در تمام روالهای فرم قابل دسترسی باشد.

۶ دکمهٔ Start را کلیک کنید، تا پروژه کامپایل و اجرا شود.

۷ روی دکمهٔ Show Constant کلیک کنید، تا ثابت Pi در برچسب Label1 نشان داده شود:



۸ با کلیک کردن دکمهٔ Quit ، از برنامه خارج شوید.

ثابتها، بویژه وقتی پای محاسبات و فرمولهای ریاضی (مانند $Area = \pi r^2$) در میان باشد، بسیار مفیدند. در قسمت آینده خواهید دید که چگونه می‌توان با ترکیب متغیرها و عملگرهای ویژوال بیسیک، فرمولهای ریاضی را محاسبه کرد.

عملگرهای ویژوال بیسیک

فرمول دستوریست مرکب از اعداد، متغیرها، عملگرها و کلمات کلیدی که مقدار جدیدی تولید می‌کند. ویژوال بیسیک دارای ابزارهای متعددی برای ایجاد فرمول است، که در این قسمت با عملگرهای ریاضی آن آشنا خواهید شد. عملگرهای ویژوال بیسیک را در جدول زیر ملاحظه می‌کنید:

عملگر	توضیح
+	جمع
-	تفریق
*	ضرب
/	تقسیم
\	تقسیم صحیح (فقط قسمت صحیح خارج قسمت را برمی‌گرداند)
Mod	باقیمانده
^	توان
&	ترکیب (بهم چسباندن) رشته‌ها

چهار عمل اصلی: عملگرهای + ، - ، * و /

عملگرهای چهار عمل اصلی ریاضی (+ ، - ، * و /) بسیار سر راست هستند، و می‌توان در تمام فرمولهایی که در آنها عدد (و متغیرهای عددی) وجود دارد، از این عملگرها استفاده کرد. در تمرین زیر طرز استفاده از این عملگرها را در یک برنامه می‌بینید.

استفاده از عملگرهای چهار عمل اصلی در برنامه

۱ از منوی File|Open آیتم Project را انتخاب کنید.

۲ در پوشهٔ c:\vbnetsbs\chap05\basic math ، پروژهٔ Basic Math را باز کنید.

۳ اگر فرم پروژه را نمی‌بینید، فرم Form1.vb را در کاوشگر راه‌حل انتخاب کرده، و روی دکمهٔ View Designer کلیک کنید.

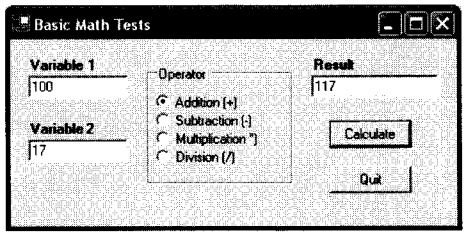
این پروژه طرز کار عملگرهای جمع، تفریق، ضرب و تقسیم (روی اعدادی که وارد می‌کنید) را نشان می‌دهد.

٤ دکمهٔ Start را کلیک کنید، تا پروژه کامپایل و اجرا شود. در برنامهٔ Basic Math دو جعبه متن (Variable 1 و Variable 2) می‌بینید، که اعداد موردنظر را در آنها وارد می‌کنید؛ نوع محاسبه نیز توسط دکمه‌های رادیویی Operator انتخاب می‌شود. وقتی دکمهٔ Calculate را کلیک کنید، نتیجهٔ محاسبه در جعبه متن Result نمایش داده خواهد شد. دکمهٔ Quit هم که (طبق معمول) برای خارج شدن از برنامه است.

۵ عدد **100** را در جعبه متن Variable 1 وارد کرده، و کلید Tab را بزنید، تا کرسر به جعبه متن دوم برود.

۶ در جعبه متن Variable 2 عدد **17** را وارد کنید؛ حالا آماده‌اید تا هر یک از چهار عمل اصلی را روی این دو عدد انجام دهید.

۷ پس از انتخاب دکمهٔ رادیویی Addition (جمع)، روی دکمهٔ Calculate کلیک کنید؛ عملگر جمع دو عدد را با هم جمع کرده، و حاصل جمع 117 را در جعبه متن Result نشان می‌دهد:



۸ سایر اعمال ریاضی را نیز روی این دو عدد انجام دهید (پس از انتخاب عملگر، حتماً دکمهٔ Calculate را کلیک کنید). توجه کنید که در وارد کردن اعداد Variable 1 و Variable 2 محدودیتی ندارید، چون آنها را از نوع Double تعریف کرده‌ام (با اعداد اعشاری و خیلی بزرگ هم می‌توانید امتحان کنید).

اجازه دهید در ادامه یک تست جالب انجام دهیم.

۹ در جعبه متن Variable 2 عدد **0** را وارد کرده، و پس از انتخاب دکمهٔ رادیویی Division (تقسیم)، روی دکمهٔ Calculate کلیک کنید.

می‌دانید که تقسیم بر صفر یکی از ممنوعیت‌های بزرگ در ریاضیات است، چون حاصل آن بینهایت خواهد شد. خبر خوب اینست که، ویژوال بیسیک .NET ، بر خلاف اسلافش، حتی از عهدهٔ این محاسبه برمی‌آید، و خارج قسمت را بصورت *infinity* (بینهایت) نشان می‌دهد!

۱۰ بعد از انجام تست‌های کافی، با کلیک کردن دکمهٔ Quit از برنامه خارج شوید.

اما اجازه دهید ببینیم این برنامه چگونه نتایج را محاسبه می‌کند. در پروژهٔ Basic Math دو متغیر از نوع Double در خارج از تمام روالها تعریف کرده‌ایم، تا همهٔ روالهای برنامه به آن دسترسی داشته باشند. سپس، با این متغیرها و عملگرهای چهار عمل اصلی، محاسبات لازم را انجام می‌دهیم.

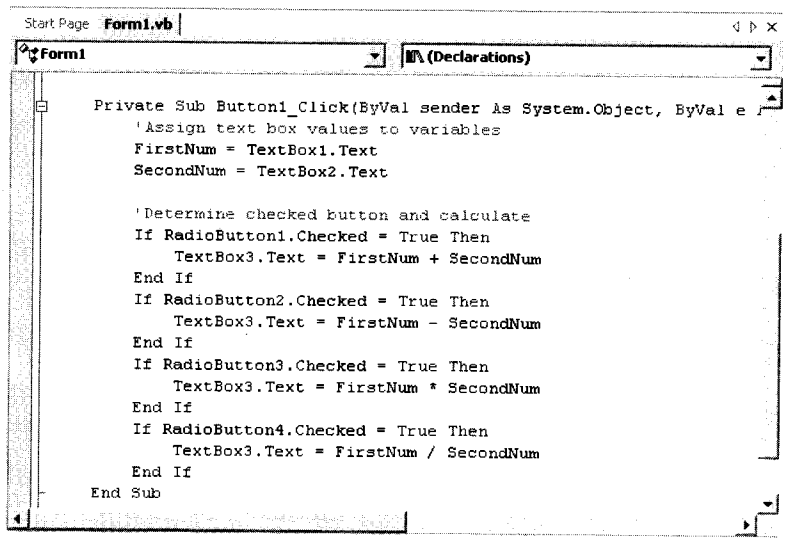
بررسی کُد برنامهٔ Basic Math

۱ روی دکمهٔ Calculate دو-کلیک کنید، تا روال Button1_Click در ادیتور کُد باز شود. در بالای فرم با دستور زیر یک متغیر از نوع Double تعریف شده است:

```
'Declare FirstNum and SecondNum variables
Dim FirstNum, SecondNum As Double
```

علت استفاده از نوع دادهٔ Double اینست که این نوع متغیر می‌تواند تقریباً تمام انواع اعداد (صحیح یا اعشاری، خیلی بزرگ یا خیلی کوچک) را در خود جای دهد. اعدادی که کاربر در جعبه متن‌های Variable 1 و Variable 2 وارد می‌کند، بترتیب در متغیرهای FirstNum و SecondNum ذخیره خواهند شد.

۲ در ادیتور کُد به روال Button1_Click بروید:



انتقال اعداد از جعبه‌های متن به متغیرها، در دو دستور اول این روال انجام می‌شود:

```
'Assign text box values to variables
FirstNum = TextBox1.Text
SecondNum = TextBox2.Text
```

...ک می‌کند که کدام دکمهٔ رادیویی انتخاب شده، و بر اساس آن نتیجه را محاسبه ... نشان می‌دهد. تست دکمهٔ رادیویی اول (دکمهٔ جمع) چنین انجام

```
'Determine checked button and calculate
If RadioButton1.Checked = True Then
    TextBox3.Text = FirstNum + SecondNum
End If
```

از فصل ۳ بیاد دارید که، از هر گروه دکمهٔ رادیویی در هر لحظه فقط یکی می‌تواند انتخاب شده (Checked = True) باشد (اگر Checked = False ، یعنی این دکمهٔ رادیویی انتخاب نشده است). بعد از این تست ساده، محاسبه انجام می‌شود.

این از برنامهٔ Basic Math.

عملگرهای پیشرفته: \، Mod، ^ و &

ویژوال بیسیک علاوه بر چهار عمل اصلی، چهار عمل دیگر نیز دارد، که عبارتند از: تقسیم صحیح (\)، باقیماندهٔ تقسیم (Mod)، توان (^) و ترکیب رشته‌ها (&). این عملگرها برای اعمال خاص ریاضی و پردازش متن مناسب هستند. در برنامهٔ زیر (که شکل اصلاح شدهٔ برنامهٔ Basic Math است) طرز کار این عملگرها را خواهید دید.

عملگرهای میانبُر

آنهایی که با زبانهای برنامه‌نویسی C و C++ آشنایند، عملگرهای میانبُر (shortcut operator) را می‌شناسند - و حالا برنامه‌نویسان ویژوال بیسیک هم می‌توانند به داشتن این امکان افتخار کنند! عملگر میانبُر عملگریست که با یک علامت = ترکیب شده، و محاسبات ریاضی را ساده‌تر می‌کند (چون دیگر لازم نیست نام متغیر را دو بار بنویسیم). برای مثال، اگر بخواهید 6 واحد به متغیری بنام X اضافه کنید، باید بنویسید X = X + 6 - در حالیکه با استفاده از عملگر میانبُر جمع می‌توان همین دستور را بصورت X += 6 نوشت. در جدول زیر عملگرهای میانبُر ویژوال بیسیک را ملاحظه می‌کنید:

جمع (+)	X = X + 6	X += 6
تفریق (-)	X = X - 6	X -= 6
ضرب (*)	X = X * 6	X *= 6
تقسیم (/)	X = X / 6	X /= 6
تقسیم صحیح (\)	X = X \ 6	X \= 6
توان (^)	X = X ^ 6	X ^= 6
ترکیب رشته‌ها (&)	X = X & "ABC"	X &= "ABC"

استفاده از عملگرهای پیشرفته

۱ از منوی File|Open آیتم Project را انتخاب کنید.

۲ در پوشهٔ c:\vbnetsbs\chap05\advanced math ، پروژهٔ Advanced Math را باز کنید.

۳ اگر فرم پروژه را نمی‌بینید، فرم Form1.vb را در کاوشگر راه‌حل انتخاب کرده، و روی دکمهٔ View Designer کلیک کنید.

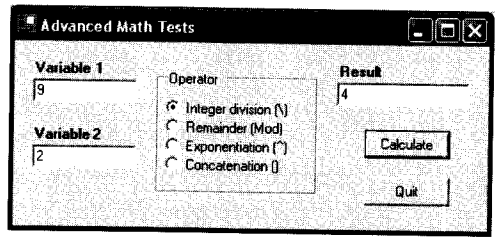
این پروژه بسیار شبیه Basic Math است، و فقط در نوع عملگرها با آن تفاوت دارد.

٤ دکمهٔ Start را کلیک کنید، تا پروژه کامپایل و اجرا شود. این برنامه هم (مانند Basic Math) دو جعبه متن برای وارد کردن اعداد (Variable 1 و Variable 2)، یک گروه دکمهٔ رادیویی برای عملگرها (Operator)، یک جعبه متن برای نمایش نتیجهٔ محاسبه (Result)، و دو دکمه برای انجام محاسبه (Calculate) و خروج از برنامه (Quit) دارد.

۵ عدد **9** را در جعبه متن Variable 1 وارد کرده، و با زدن کلید Tab به جعبه متن دوم بروید.

۶ در جعبه متن Variable 2 عدد **2** را وارد کنید؛ حالا آماده‌اید تا هر یک از چهار عملگر پیشرفته را روی این دو عدد اجرا کنید.

۷ پس از انتخاب دکمهٔ رادیویی Integer Division (تقسیم صحیح)، دکمهٔ Calculate را کلیک کنید؛ با این کار جزء صحیح خارج قسمت تقسیم 9 بر 2 را در جعبه متن Result خواهید دید:



می‌دانید که خارج قسمت تقسیم 9 بر 2 برابر 4.5 است، ولی عملگر \ فقط جزء صحیح این خارج قسمت (یعنی عدد صحیح 4) را برمی‌گرداند. شاید فکر کنید این نوع تقسیم به چه دردی می‌خورد؟ گاهی اوقات بعضی کمیت‌ها را نمی‌توان بصورت غیر صحیح تقسیم کرد - مثلاً، ۹ نفر آدم را چگونه می‌توان به ۲ گروه تقسیم کرد؟ آیا می‌توان یکی از آنها را نصف کرد!؟

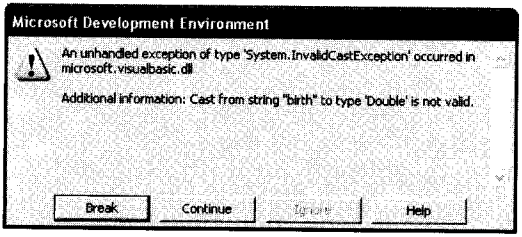
۸ دکمهٔ رادیویی Remainder (باقیماندهٔ تقسیم) را انتخاب کرده، و دکمهٔ Calculate را کلیک کنید؛ حاصل این عمل عدد 1 است. در واقع این عدد باقیماندهٔ تقسیم صحیح 9 بر 2 است. برای مثال، اگر بخواهیم ۹ نفر را به گروههای ۲ نفری تقسیم کنیم، یک نفر زیاد خواهد آمد.

۹ دکمهٔ رادیویی Exponentiation (توان) را انتخاب، و Calculate را کلیک کنید؛ همانطور که می‌دانید، $9^2 = 81$ ، و این عددیست که در جعبه متن Result خواهید دید. در فرمولهای ویژوال بیسیک، 9^2 بصورت 9 ^ 2 نوشته می‌شود.

۱۰ دکمهٔ رادیویی Concatenation (ترکیب رشته‌ها) را انتخاب کرده، و Calculate را کلیک کنید؛ عددی که در جعبه متن Result می‌بینید، 92 است - در واقع، این عملگر 9 و 2 را بصورت دو

کاراکتر ساده کنار هم قرار داده و بهم می‌چسباند. عملگر & می‌تواند روی اعداد عمل کند، ولی کارکرد اصلی آن ترکیب رشته‌های متن است. تست زیر را انجام دهید:

۱۱ عبارت **birth** را در جعبه متن Variable 1 و **day** را در جعبه متن Variable 2 وارد کرده، و Calculate را کلیک کنید؛ شاید انتظار دارید برنامه کلمهٔ birthday را در جعبه متن نشان دهد – ولی تنها چیزی که خواهید دید، یک پیام خطا است:



به این قبیل پیامهای خطا که فقط در حین اجرای برنامه خود را نشان می‌دهند، **خطای زمان اجرا** (runtime error) می‌گویند. ویژوال بیسیک در این پیام به شما می‌گوید که **منظور شما را نمی‌فهمد و قادر به انجام کاری که از وی خواسته‌اید، نیست.** اما قضیه چیست؟ کمی به پیام خطا دقت کنید: Cast from string "birth" to type 'Double' is not valid – معنای سادهٔ این پیام اینست که «ویژوال بیسیک نتوانسته کلمهٔ "birth" را به نوع دادهٔ Double تبدیل کند.»

اگر بیاد داشته باشید، متغیرهای FirstNum و SecondNum را از نوع Double تعریف کردیم، و اکنون می‌خواهیم در آنها متن ذخیره کنیم، در حالیکه متغیرهای Double فقط می‌توانند عدد ذخیره کنند. به همین سادگی!

۱۲ در پنجرهٔ Microsoft Development Environment ، دکمهٔ Continue را کلیک کنید؛ اگر پنجرهٔ خطای دیگری ظاهر شد، در آنجا هم OK را کلیک کنید، تا برنامه بسته شده و به محیط ویژوال استودیو برگردید.

اکنون اجازه دهید نگاهی به کُد این برنامه انداخته، و علت خطا را پیدا کنیم.

۱۳ ادیتور کُد را باز کرده، و آنقدر پائین بروید تا به دستور زیر برسید:

```
'Declare FirstNum and SecondNum variables
Dim FirstNum, SecondNum As Double
```

همانطور که می‌دانید، FirstNum و SecondNum دو متغیری هستند که مقدار کنترلهای TextBox1 و TextBox2 را در خود ذخیره خواهند کرد.

۱۴ نوع Double را به String تغییر دهید، تا بتوانیم تست بهم چسباندن عبارتهای متن را نیز انجام دهیم.

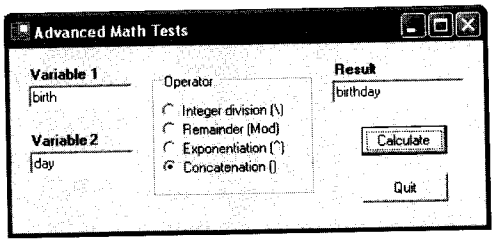
۱۵ در ادیتور کُد آنقدر پائین بروید تا به دستورات زیر (که محاسبات را انجام می‌دهند) برسید:

```
'Assign text box values to variables
FirstNum = TextBox1.Text
SecondNum = TextBox2.Text

'Determine checked button and calculate
If RadioButton1.Checked = True Then
    TextBox3.Text = FirstNum \ SecondNum
End If
If RadioButton2.Checked = True Then
    TextBox3.Text = FirstNum Mod SecondNum
End If
If RadioButton3.Checked = True Then
    TextBox3.Text = FirstNum ^ SecondNum
End If
If RadioButton4.Checked = True Then
    TextBox3.Text = FirstNum & SecondNum
End If
```

(همانطور که می‌بینید، طرز کار این برنامه بسیار شبیه برنامهٔ Basic Math است.) بعد از تغییر فوق، برنامه را اجرا کنید تا طرز کار عملگر & را هم ببینید.

۱۶ دکمهٔ Start را کلیک کنید.

۱۷ کلمهٔ **birth** را در جعبه متن Variable 1 و **day** را در جعبه متن Variable 2 وارد کرده، و بعد از انتخاب عملگر Concatenation (در قسمت Operator)، دکمهٔ Calculate را کلیک کنید؛ این بار برنامه بدون هیچ خطایی کار خود را انجام خواهد داد:



۱۸ برای خارج شدن از برنامه، دکمهٔ Quit را کلیک کنید.

این هم از برنامهٔ Advanced Math.

نکته

اجتناب از خطاهای زمان اجرا بسیار مشکل است، و حتی برنامه‌های بسیار معروف (مانند Word و Excel) هم گاهی دچار آنها می‌شوند. در فصل ۹ خواهید دید که چگونه می‌توان برنامه را برای مقابله با این قبیل خطاها مجهز کرد.

متدهای ریاضی در چارچوب NET.

کار با اعداد فقط جمع و تفریق نیست، و گاهی پیش می‌آید که بخواهید محاسبات پیچیده‌تری در برنامه‌های خود انجام دهید. در چارچوب NET. متدهای متنوعی برای انجام محاسبات پیشرفته وجود دارد، که تعدادی از آنها را در جدول ذیل ملاحظه می‌کنید.

متد	کاربرد
Abs( )	قدر مطلق n را برمی‌گرداند.
Atan( )	آرک‌تانژانت n را (بر حسب رادیان) برمی‌گرداند.
Cos( )	کسینوس کمان n را برمی‌گرداند. (n بر حسب رادیان است.)
Exp( )	مقدار e^n را برمی‌گرداند (e یک ثابت ریاضی است برابر با 2.718281828).
Sign( )	علامت n را برمی‌گرداند: 1- اگر n منفی باشد، 0 اگر n صفر باشد، 1+ اگر n مثبت باشد.
Sin( )	سینوس کمان n را برمی‌گرداند. (n بر حسب رادیان است.)
Sqrt( )	جذر (ریشهٔ دوم) n را برمی‌گرداند.
Tan( )	تانژانت کمان n را برمی‌گرداند. (n بر حسب رادیان است.)

چارچوب NET. که خود یکی از واسطهای زیربنایی سیستم ویندوز محسوب می‌شود، اکنون جزیی از ویژوال استودیو .NET (و عاملی مشترک بین ، ویژوال بیسیک .NET ، ویژوال سی++.NET ، ، ویژوال سی#.NET و سایر اجزاء ویژوال استودیو) است. چارچوب NET. از کلاسهای متعددی تشکیل می‌شود، که کلاس System.Math یکی از آنهاست، و با دستور Imports می‌توانید از متدهای آن در برنامه‌های خود استفاده کنید:

```
Imports System.Math
```

فراموش نکنید که، باید این دستور را در بالاترین نقطه از کُد فرم قرار دهید.

محاسبهٔ جذر اعداد با استفاده از کلاس System.Math

۱ از منوی File|New آیتم Project را انتخاب کنید، تا پنجرهٔ «پروژهٔ جدید» باز شود.

۲ یک پروژهٔ جدید از نوع Visual Basic Windows Application و با نام **My Framework Math** ، در پوشهٔ c:\vbnetsbs\chap05 ایجاد کنید.

۳ ابزار دکمه (Button) را از جعبه ابزار ویژوال بیسیک انتخاب کرده، و با آن یک دکمه در بالای فرم بکشید.

٤ ابزار جعبه متن (TextBox) را از جعبه ابزار انتخاب کرده، و با آن یک جعبه متن زیر دکمه رسم کنید.

۵ خاصیت Text کنترل TextBox1 را پاک کرده، و خاصیت Text کنترل Button1 را به **Square Root** ست کنید.

۶ روی دکمهٔ Square Root دو-کلیک کنید، تا ادیتور کُد باز شود.

۷ در بالاترین نقطهٔ ادیتور کُد (بالاتر از دستور Public Class Form1) دستور زیر را وارد کنید:

```
Import System.Math
```

با این دستور اشیاء، خواص و متدهای کلاس System.Math به برنامهٔ شما اضافه خواهند شد. این دستور باید اولین دستور برنامه (حتی قبل از دستورات تعریف فرم) باشد.

۸ آنقدر در ادیتور کُد پائین بروید، تا به روال Button1_Click برسید؛ دستورات زیر را بین Private Sub Button1_Click و End Sub وارد کنید:

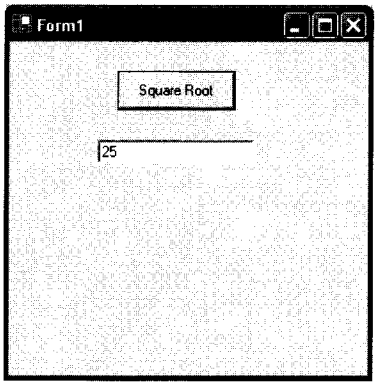
```
Dim Result As Double
Result = Sqrt(625)
TextBox1.Text = Result
```

در دستور اول یک متغیر بنام Result تعریف شده، جذر عدد 625 محاسبه و در این متغیر ذخیره شده (دستور دوم)، و سپس در جعبه متن TextBox1 نمایش داده می‌شود (دستور سوم).

۹ با کلیک کردن دکمهٔ Save All در میله ابزار استاندارد، پروژه را ذخیره کنید.

۱۰ دکمهٔ Start را کلیک کنید، تا برنامه کامپایل و اجرا شود.

۱۱ روی دکمهٔ **Square Root** کلیک کنید. ویژوال بیسیک جذر عدد ۶۲۵ را محاسبه کرده، و آنرا (۲۵) در جعبه متن نشان می‌دهد. جالب است، نه!



۱۲ دکمهٔ Close (آیکون X در گوشهٔ راست-بالای پنجره) را کلیک کنید، و از برنامه خارج شوید.

با اضافه کردن کلاسهای چارچوب .NET. می‌توانید قابلیتهای برنامهٔ خود را به میزان زیادی بالا ببرید. در طول این کتاب باز هم از این کلاسها استفاده خواهیم کرد.

تقدم عملگرها

در قسمتهای قبل با تعدادی از عملگرهای ویژوال بیسیک کار کردیم. ویژوال بیسیک به شما اجازه می‌دهد تا عملگرها را با هم ترکیب کرده و فرمولهای پیچیده‌تری بسازید (البته مشروط بر اینکه بین این عملگرها حداقل یک عدد یا متغیر وجود داشته باشد). دستور زیر یک فرمول صحیح در ویژوال بیسیک است:

```
Total = 10 + 15 * 2 / 4 ^ 2
```

این فرمول چندین عدد را با عملگرهای مختلف ترکیب کرده، و نتیجه را در متغیری بنام Total قرار می‌دهد. اما سؤال اینجاست که، ویژوال بیسیک این فرمول را چگونه محاسبه می‌کند؟ بعبارت دیگر، کدام عمل

ریاضی ابتدا انجام می‌شود؟ اگر کمی دقت کنید، ترتیب عمل عملگرها تأثیر بسزایی در نتیجهٔ محاسبه می‌گذارد.

از آنجائیکه ترتیب محاسبهٔ عملگرها در یک فرمول ریاضی از اهمیت زیادی برخوردار است، ویژوال بیسیک این مسئله را با قراردادی موسوم به **تقدم عملگرها** (operator precedence) حل کرده است. این قرارداد تعیین می‌کند که در یک فرمول عملگرها با چه ترتیبی بایستی محاسبه شوند. در جدول زیر ترتیب و اولویت عملگرها را در یک فرمول ملاحظه می‌کنید (عملگرهای دارای تقدم یکسان از چپ به راست محاسبه می‌شوند):

عملگر	تقدم محاسبه
( )	عبارتهایی که در داخل پرانتز قرار دارند، همواره در ابتدا محاسبه می‌شوند.
^	به توان رساندن در اولویت دوم قرار دارد.
-	منفی کردن در اولویت سوم است.
* /	ضرب و تقسیم اولویت‌های چهارم هستند.
\	تقسیم صحیح در اولویت پنجم قرار دارد.
Mod	باقیماندهٔ تقسیم اولویت ششم است.
+ -	جمع و تفریق در آخرین اولویت قرار دارد.

با توجه به جدول فوق، فرمول Total = 10 + 15 * 2 / 4 ^ 2 به صورت زیر محاسبه خواهد شد (قسمتی که در هر مرحله محاسبه می‌شود، با حروف ضخیم نشان داده شده است):

Total = 10 + 15 * 2 / **4 ^ 2**
Total = 10 + **15 * 2** / 16
Total = 10 + **30 / 16**
Total = **10 + 1.875**
Total = **11.875**

یک گام فراتر: استفاده از پرانتز در فرمول‌ها

برای کنترل نحوهٔ محاسبهٔ یک فرمول می‌توان از پرانتز استفاده کرد. همانطور که در جدول صفحهٔ قبل دیدید، ویژوال بیسیک همواره در ابتدا عبارات داخل پرانتز را محاسبه می‌کند، و بعد سراغ قسمتهای دیگر فرمول می‌رود. فرمول زیر را در نظر بگیرید:

```
Number = (8 - 5 * 3) ^ 2
```

در این فرمول، با اینکه عملگر ^ تقدم بالاتری نسبت به عملگرهای * و - دارد، اما ویژوال بیسیک ابتدا عبارت داخل پرانتز را محاسبه کرده (-7)، و بعد آنرا به توان 2 می‌رساند. در داخل پرانتز نیز، ابتدا عبارت 3 * 5 محاسبه شده، و سپس 15 - 8 (که حاصل آن -7 است). اگر می‌خواهید عمل تفریق قبل از ضرب انجام شود، می‌توانید فرمول فوق را بصورت زیر بنویسید:

```
Number = ((8 - 5) * 3) ^ 2
```

در چنین فرمولهایی، ویژوال بیسیک محاسبه را از داخلی‌ترین پرانتز شروع کرده، و به سمت بیرون حرکت می‌کند. همانطور که می‌توانید ببینید، حاصل فرمول اول 49 و دومی 81 است - و این تأثیر پرانتزها را بخوبی

نشان می‌دهد. پرانتزها علاوه بر تغییر دادن ترتیب محاسبه، کمک زیادی به خواناتر شدن فرمول‌ها نیز می‌کنند.

مرجع سریع فصل ۵

برای ...	انجام دهید
تعریف یک متغیر	با استفاده از دستور Dim ، مشخص کردن نام متغیر، و کلمهٔ کلیدی As (برای تعیین نوع دادهٔ متغیر) می‌توانید یک متغیر تعریف کنید: Dim Country As String
تغییر دادن مقدار یک متغیر	با استفاده از عملگر انتساب، مقدار جدید را به متغیر بدهید: Country = "Japan"
گرفتن ورودی از کاربر	از تابع InputBox استفاده کرده، و مقدار برگشتی آنرا به متغیر موردنظر بدهید: UserName = InputBox("What is your name?")
نمایش پیام به کاربر	از تابع MsgBox استفاده کرده،و پیام موردنظر را به‌کاربر نشان دهید: MsgBox(Forecast, , "Spain Weather Report")
ایجاد یک ثابت	با استفاده از کلمهٔ کلیدی Const ، مشخص کردن نام ثابت، کلمهٔ کلیدی As (برای تعیین نوع دادهٔ ثابت)، و دادن مقدار به آن (با عملگر =) می‌توانید یک ثابت تعریف کنید: Const JackBennysAge As Short = 39
ایجاد یک فرمول	اعداد، متغیرها و عملگرهای ویژوال بیسیک را با هم ترکیب کنید: Result = 1 ^ 2 * 3 \ 4 'this equals 0
بهم چسباندن رشته‌های متن	برای بهم چسباندن رشته‌های متن، از عملگر & استفاده کنید: Msg = "Hello" & "," & "world!"
اضافه کردن یکی از کلاسهای کتابخانه‌ای چارچوب .NET.	کلاس مورد نظر را با دستور Import *classname* (در بالاترین نقطهٔ ادیتور کُد) به برنامه اضافه کنید.
فراخوانی یکی از متدهای کلاس کتابخانه‌ای	از متد موردنظر در فرمولهای برنامه استفاده کنید: Hypotenuse = Sqrt(x ^ 2 + y ^ 2)
کنترل ترتیب محاسبه در فرمول	در فرمول خود از پرانتز استفاده کنید. برای مثال: Result = 1 + 2 ^ 3 \ 4 'this equals 3 Result = (1 + 2) ^ (3 \ 4) 'this equals 1

فصل

۶

ساختارهای تصمیم‌گیری

در این فصل یاد می گیرید چگونه :

- ✔ عبارتهای شرطی بنویسید.
- ✔ برای تصمیم‌گیری در شرایط مختلف از دستور If...Then استفاده کنید.
- ✔ دستورات If...Then را اتصال کوتاه کنید.
- ✔ برای انتخاب بین گزینه‌های مختلف از دستور Select Case استفاده کنید.
- ✔ رویدادهای ماوس را آشکارسازی و مدیریت کنید.

در چند فصل گذشته با اشیاء، منوها و دیالوگ‌های ویژوال بیسیک .NET آشنا شدید، و دیدید که چگونه می‌توان ورودی را از کاربر گرفت، آنرا پردازش کرد، و سپس نتیجهٔ حاصله را به وی نشان داد. در این فصل خواهید دید که چگونه می‌توان بر اساس شرایط مختلف تصمیم‌گیری کرد، و مسیر برنامه را تغییر داد. اجرای دستورات متفاوت در شرایط متفاوت، یکی از کلیدی‌ترین تکنیکهای برنامه‌نویسی است، که در این فصل با نحوهٔ انجام آن آشنا می‌شوید.

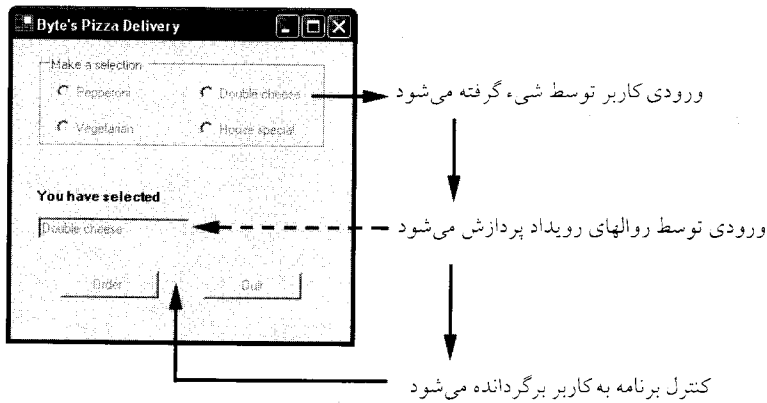
تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- **در ویژوال بیسیک .NET دو عملگر جدید بنامهای AndAlso و OrElse معرفی شده است. گاهی در یک ساختار تصمیم‌گیری مانند If...Then ، لزومی ندارد که کلیهٔ حالتها ارزیابی شوند. در این قبیل موارد می‌توان برخی از حالتها را با استفاده از عملگرهای AndAlso و OrElse دور زد، و یا باصطلاح آنها را «اتصال کوتاه» کرد.**

برنامه‌نویسی رویداد-گرا

برنامه‌هایی که تا بحال نوشتیم، همگی بگونه‌ای بودند که کاربر می‌توانست بدلخواه خود با آنها کار کند. این برنامه‌ها صبورانه منتظر می‌ماندند تا کاربر عملی انجام دهد، و بعد آنها عکس‌العمل نشان می‌دادند. در میان برنامه‌نویسان این روش به برنامه‌نویسی رویداد-گرا (event-driven programming) معروف است. در برنامه‌نویسی رویداد-گرا، اشیاء برنامه آنقدر هوشمندند که خود به کاربر پاسخ مناسب را می‌دهند، و تنها وظیفهٔ برنامه‌نویس پردازش داده‌های ورودیست. در شکل زیر می‌توانید ببینید که یک برنامهٔ رویداد-گرا در ویژوال بیسیک چگونه کار می‌کند.

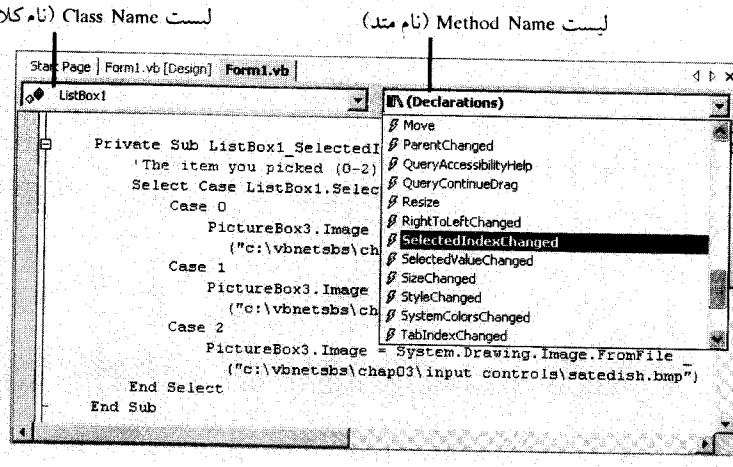


ورودی یک برنامه می‌تواند از خود کامپیوتر هم بیاید. برای مثال، این ورودی می‌تواند رسیدن یک پیام پُست‌الکترونیک (email) و یا سپری شدن زمانی معین باشد؛ این رویدادها از کامپیوتر می‌آیند، نه از طرف کاربر. اینکه منشاء رویداد چیست، اهمیتی برای ویژوال بیسیک ندارد، و او فقط کار خودش (اجرای روال رویداد) را انجام می‌دهد. تا اینجا با رویدادهایی از قبیل Click (کلیک شدن شیء)، CheckedChanged (تغییر حالت دکمهٔ رادیویی یا جعبه چک)، و SelectedIndexChanged (انتخاب آیتم جدید در لیست) آشنا شده‌اید، اما اشیاء ویژوال بیسیک به رویدادهای متنوع دیگری نیز می‌توانند پاسخ دهند.

هر شیء از چه رویدادهایی پشتیبانی می‌کند؟

رویدادهایی که هر شیء ویژوال بیسیک به آنها پاسخ می‌دهد، از قبل تعریف شده‌اند. برای دیدن این رویدادها، شیء موردنظر را از لیست Class Name در ادیتور کُد انتخاب کرده، و سپس لیست Method Name را باز کنید: آیتمهایی که در کنار آنها آیکون «رعد و برق» دیده می‌شود، رویدادهای آن شیء هستند. برای هر کدام از این رویدادها می‌توان روال رویداد (event procedure) نوشت، و به محض رُخ دادن یک رویداد، ویژوال بیسیک روال مربوط به آن را اجرا خواهد کرد. برای مثال، شیء جعبه لیست (کنترل ListBox) متجاوز از ۶۰ رویداد دارد، که برخی از آنها عبارتند از: Click ، KeyUp ، KeyPress ، KeyDown ، GotFocus ، DragOver ، DragDrop ، DoubleClick ، TextChanged ، MouseWheel ، MouseUp ، MouseMove ، MouseDown ، LostFocus ، و Validate .

دانستن اینکه هر شیء از چنین رویدادهای متنوعی پشتیبانی می‌کند، جالب است، اما بسیار بندرت پیش می‌آید که لازم شود برای بیش از چهار رویداد یک شیء برنامه‌نویسی کنید. در شکل زیر تعدادی از رویدادهای شیء جعبه لیست (کنترل ListBox) را در ادیتور کُد می‌بینید:



رویداد-گرا بودن برنامه‌های ویژوال بیسیک بدان معناست که، قسمت اعظم کُد یک برنامهٔ ویژوال بیسیک مربوط به روال‌های رویداد آن است. در فصلهای قبل دیدید که چگونه می‌توان محاسبات و پردازشهای موردنظر را در این روالها انجام داد. در این فصل با ساختارهای تصمیم‌گیری، و نحوهٔ کنترل اجرای برنامه در شرایط مختلف، آشنا می‌شوید. به کمک ساختارهای تصمیم‌گیری و حلقه‌ها (که موضوع فصل آینده هستند) می‌توانید برنامه‌هایی بنویسید که (تقریباً) در هر شرایطی کار کنند.

عبارات شرطی

یکی از مهمترین ابزارهای برنامه‌نویسی رویداد-گرا **عبارت شرطی** (conditional expression) است. یک عبارت شرطی عبارتست از: **قسمتی از یک دستور که پرسشی با پاسخ آری-یا-خیر دربارهٔ یک متغیر، خاصیت یا هر چیز دیگری می‌پرسد.** در زیر یک عبارت شرطی را می‌بینید:

```
Price < 100
```

جواب این پرسش آری (True) است، اگر متغیر Price کوچکتر از 100 باشد؛ و پاسخ خیر (False) است، اگر Price بزرگتر یا مساوی 100 باشد. در هر عبارت شرطی بایستی یک عملگر مقایسه (comparison operator) وجود داشته باشد؛ در جدول زیر عملگرهای مقایسهٔ ویژوال بیسیک را ملاحظه می‌کنید:

عملگر مقایسه	مفهوم
=	تساوی
<>	عدم تساوی
>	بزرگتر از
<	کوچکتر از
>=	بزرگتر یا مساوی
<=	کوچکتر یا مساوی

نکته

به عباراتی که نتیجهٔ ارزیابی آنها True یا False است، عبارات منطقی (یا بولی) نیز گفته می‌شود. متغیرهایی از این نوع را نیز متغیرهای بولی می‌نامند؛ متغیرهای بولی را می‌توانید با دستور Dim *VariableName* As Boolean تعریف کنید.

در جدول زیر نیز چند عبارت شرطی و نتیجهٔ ارزیابی آنها را می‌بینید:

عبارت شرطی	نتیجه
10 <> 20	True (10 مساوی 20 نیست)
Score < 20	True اگر Score کوچکتر از 20 باشد؛ در غیراینصورت، False
Score = Label1.Text باشد؛	True اگر Score مساوی مقدار خاصیت Text شیء Label1 درغیراینصورت، False
Text1.Text = "Bill" درغیراینصورت،	True اگر خاصیت Text شیء Text1 معادل کلمهٔ "Bill" باشد؛ False

ساختار تصمیم‌گیری If...Then

ساختار تصمیم‌گیری (decision structure) به قسمتی از کُد برنامه گفته می‌شود که بتواند در شرایط مختلف به گونه‌های متفاوت اجرا شود. اگر می‌خواهید اجرای قسمتی از کُد برنامه را به شرط خاصی منوط کنید، می‌توانید از ساختار تصمیم‌گیری If...Then استفاده کنید. ساده‌ترین شکل یک ساختار If...Then چنین است:

If *condition* Then *statement*

که در آن *condition* یک عبارت شرطی، و *statement* یک دستور معتبر ویژوال بیسیک است. مثلاً، در دستور زیر

```
If Score >= 20 Then Label1.Text = "You win!"
```

Score >= 20 عبارت شرطی است - اگر مقدار متغیر Score بزرگتر یا مساوی 20 باشد، خاصیت Text شیء Label1 به "You win!" ست خواهد شد؛ اما اگر چنین نباشد، ویژوال بیسیک این دستور را نادیده گرفته و به سراغ دستور بعدی خواهد رفت. بیاد داشته باشید که، مقدار یک عبارت شرطی همواره یا True (آری) است، یا False (خیر)؛ شک و تردید معنا ندارد!

تست چند شرط در یک ساختار تصمیم‌گیری If...Then

در یک ساختار If...Then می‌توان بیش از یک شرط را تست کرد. چنین ساختاری علاوه بر If...Then دارای کلمات کلیدی دیگری از قبیل ElseIf ، Else و End If نیز خواهد بود.

```
If condition1 Then
    statements executed if condition1 is True
ElseIf condition2 Then
    statements executed if condition2 is True
[Additional ElseIf clauses can be placed here]
Else
    statements executed if none of the conditions are True
End If
```

در این ساختار، ابتدا شرط *condition1* ارزیابی می‌شود. اگر این شرط True باشد، دستور (یا دستورات) زیر آن (بترتیب) اجرا خواهند شد. اما اگر این شرط True نباشد، شرط دوم *(condition2)* ارزیابی می‌شود؛ و اگر این شرط True باشد، دستور (یا دستورات) زیر آن اجرا خواهند شد. (تعداد بلوکهای ElseIf در یک ساختار If...Then هیچ محدودیتی ندارد.) اگر هیچیک از عبارتهای شرطی True نباشند، دستورات بلوک Else اجرا می‌شوند. و بالاخره، کل ساختار تصمیم‌گیری با کلمهٔ کلیدی End If خاتمه می‌یابد.

در مثال زیر، مقدار مالیات بر درآمد تصاعدی (بر اساس نرخهای مصوب ادارهٔ مالیات بر درآمد ایالات متحده در سال ۲۰۰۱) با یک ساختار مرکب If...Then محاسبه شده است:

```
Dim AdjustedIncome, TaxDue As Double
AdjustedIncome = 32000

If AdjustedIncome <= 27050 Then              '15% tax bracket
    TaxDue = AdjustedIncome * 0.15
ElseIf AdjustedIncome <= 65550 Then          '28% tax bracket
    TaxDue = 4057.5 + ((AdjustedIncome - 27050) * 0.28)
ElseIf AdjustedIncome <= 136750 Then         '31% tax bracket
    TaxDue = 14837.5 + ((AdjustedIncome - 65550) * 0.31)
ElseIf AdjustedIncome <= 297350 Then         '36% tax bracket
    TaxDue = 36909.5 + ((AdjustedIncome - 136750) * 0.36)
Else                                         '39.6% tax bracket
```

```
    TaxDue = 94725.5 + ((AdjustedIncome - 297350) * 0.396)
End If
```

در این کُد یک متغیر از نوع Double بنام AdjustedIncome ارزیابی شده، و بعد از تعیین گروه درآمدی، مالیات آن محاسبه می‌شود.

بسیار مهم

ترتیب عبارتهای شرطی در ساختار If...Then و Elself بسیار مهم است. مثال بالا را در نظر بگیرید: اگر ترتیب عبارات شرطی را معکوس و آنها را از بزرگ به کوچک مرتب کنیم، چه اتفاقی می‌افتد؟ در این حالت، تمام گروههای مالیاتی 15 ، 28 ، و 31 درصدی همگی در گروه 36 درصد قرار می‌گیرند، چون همهٔ آنها درآمدی کمتر از 297,350 دارند. علت آنست که ویژوال بیسیک با رسیدن به اولین شرط True دیگر سایر شرطها را ارزیابی نمی‌کند (حتی اگر در میان آنها هم شرط True وجود داشته باشد). از آنجائیکه در تمام شرطهای کُد فوق یک متغیر (AdjustedIncome) ارزیابی می‌شود، بایستی آنها را بصورت صعودی مرتب کنید تا گروه درآمدی بدرستی تعیین شود. نتیجهٔ اخلاقی: وقتی پای چند متغیر در میان است، باید بیشتر دقت کنید!

در تمرین این قسمت، به کمک یک ساختار If...Then برنامه‌ای می‌نویسیم که هویت کاربر را ارزیابی و تعیین می‌کند (این برنامه شبیه برنامه‌های تعیین هویت کاربر در محیط‌های شبکه است).

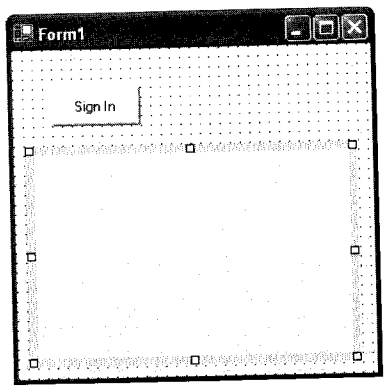
تعیین هویت کاربر با استفاده از ساختار If...Then

۱ ویژوال استودیو .NET را باز کرده، و یک پروژهٔ Visual Basic Windows Application بنام **My User Validation** در پوشهٔ c:\vbnetsbs\chap06 ایجاد کنید.

۲ یک شیء دکمه (کنترل Button) در بالای فرم برنامه (کمی متمایل به سمت چپ) رسم کنید.

۳ خاصیت Text این دکمه را به **Sign In** ست کنید.

٤ کنترل PictureBox را از جعبه ابزار انتخاب کرده، و یک کادر بزرگ روی فرم برنامه (درست زیر دکمه) رسم کنید:



۵ روی دکمهٔ Sign In دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد باز شود.

۶ دستورات زیر را در این روال رویداد بنویسید:

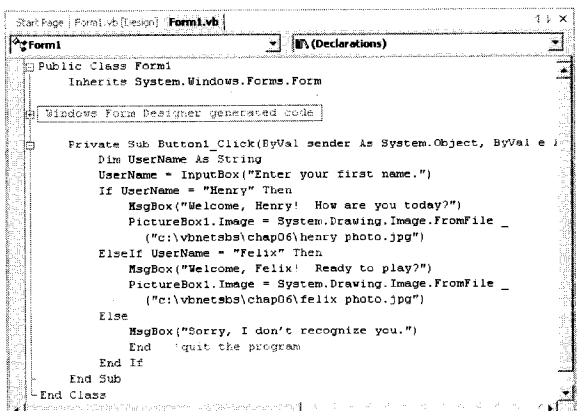
```
Dim UserName As String
UserName = InputBox("Enter your first name.")
If UserName = "Henry" Then
    MsgBox("Welcome, Henry!  How are you today?")
    PictureBox1.Image = System.Drawing.Image.FromFile _
      ("c:\vbnetsbs\chap06\henry photo.jpg")
ElseIf UserName = "Felix" Then
    MsgBox("Welcome, Felix!  Ready to play?")
    PictureBox1.Image = System.Drawing.Image.FromFile _
      ("c:\vbnetsbs\chap06\felix photo.jpg")
Else
    MsgBox("Sorry, I don't recognize you.")
    End   'quit the program
End If
```


(این یک قرارداد مرسوم بین برنامه‌نویسان است که، دستورات ذیل هر قسمت از If...Then یا ElseIf و Else همراه با تورفتگی نوشته شوند، تا تشخیص آنها از سایر قسمتهای برنامه بآسانی میسر باشد.)

نکته

در ادیتور کُد ویژوال استودیو، هر خط برنامه می‌تواند بیش از 65,000 کاراکتر طول داشته باشد، اما واضحست که کار کردن با چنین دستوری ابداً ساده نیست - در واقع، بهتر است طول هر خط از 80 کاراکتر تجاوز نکند. اگر دستوری خیلی بزرگ باشد، می‌توان آنرا به کمک کاراکتر ادامهٔ خط (_) - در آخر هر خط - به قطعات کوچکتر شکست. تنها جایی که نمی‌توان از این کاراکتر استفاده کرد، داخل رشته‌هاست (یعنی بین دو علامت نقل، " ").

در شکل زیر ادیتور کُد را بعد از نوشتن این دستورات، می‌بینید (کُد کامل این برنامه را هم می‌توانید در پوشهٔ c:\vbnetsbs\chap06\user validation بیابید):



۷ با کلیک کردن دکمهٔ Save All ، پروژه را ذخیره کنید.

۸ دکمهٔ Start را کلیک کنید، تا پروژه کامپایل و اجرا شود. در ابتدا فقط یک فرم خالی می‌بینید، که دکمه‌ای بنام Sign In در گوشهٔ چپ-بالای آن دیده می‌شود.

۹ دکمهٔ Sign In را کلیک کنید. با اجرای روال Button1_Click (و دومین دستور آن یعنی تابع InputBox)، پنجره‌ای ظاهر می‌شود تا بتوانید نام خود را در آن وارد کنید.

۱۰ نام **Henry** را وارد کرده، و OK را کلیک کنید.

نکته

دستور تست شرط (If **UserName = "Henry"** Then) به نوع حروف (کوچک یا بزرگ) حساس است؛ از نظر این دستور "Henry" و "henry" دو نام متفاوت محسوب می‌شوند.

دستور If UserName = "Henry" Then نام کاربر را (که در متغیر UserName ذخیره شده) با "Henry" مقایسه می‌کند، و اگر این شرط True باشد، پیام خوشامدگویی به "Henry" را نشان می‌دهد (تابع MsgBox).

۱۱ OK را کلیک کنید، تا پنجرهٔ پیام بسته شده، و تصویر «هِنری» در جعبه تصویر به نمایش درآید:

همانطور که در فصل ۳ یاد گرفتید، برای بار کردن فایل تصویر در جعبه تصویر از متد System.Drawing.Image.FromFile استفاده کرده‌ایم.

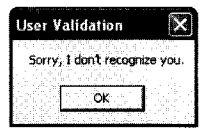
۱۲ دوباره دکمهٔ Sign In را کلیک کرده، و پس از وارد کردن نام **Felix** ، OK را کلیک کنید.

این بار شرط اول False می‌شود، ولی شرط دوم (دستور ElseIf) مقدار True برمی‌گرداند: برنامه پیام خوشامد به Felix را نشان می‌دهد (دستور MsgBox دوم).

۱۳ OK را کلیک کنید، تا پنجرهٔ پیام بسته شده، و این بار تصویر «فلیکس» در جعبه تصویر به نمایش درآید:

۱۴ یک بار دیگر Sign In را کلیک کرده، ولی این بار نام **Sally** را وارد، و سپس OK را کلیک کنید.

این بار هیچیک از دو شرط ...If و ElseIf ارزش True ندارند، بنابراین قسمت Else اجرا شده، و برنامه پیام زیر را نمایش می‌دهد:



۱۵ OK را کلیک کنید، تا پنجرهٔ پیام و برنامه (هر دو با هم) بسته شوند. بدین ترتیب افراد غیرمجاز امکان ورود به برنامه را نخواهند داشت.

استفاده از عملگرهای منطقی در عبارات شرطی

شرط یک عبارت شرطی می‌تواند ترکیبی از چند حالت مختلف نیز باشد. برای ایجاد یک شرط مرکب باید از عملگرهای منطقی (logical operator) استفاده کنیم. عملگرهای منطقی ویژوال بیسیک را در جدول زیر ملاحظه می‌کنید:

عملگر منطقی	مفهوم
And	فقط وقتی هر دو عبارت شرطی ارزش True داشته باشند، ارزش ترکیب آنها True خواهد بود.
Or	اگر هر کدام از دو عبارت شرطی ارزش True داشته باشند، ارزش ترکیب آنها True خواهد بود.
Not	ارزش یک عبارت را معکوس می‌کند: بعبارت دیگر، True را به False ، و False را به True تبدیل می‌کند.
Xor	فقط وقتی ارزش این عبارت True می‌شود که ارزش دو جزء آن متضاد باشد (یکی True و دیگری False). اگر دو جزء هم ارزش باشند، ارزش کل عبارت False خواهد شد. (Xor به معنای «یا انحصاری» - eXclusive Or - است.)

نکته

در یک عبارت که شامل تمام انواع عملگرها (محاسباتی، مقایسه‌ای و منطقی) باشد، ابتدا عملگرهای محاسباتی ارزیابی می‌شوند، سپس عملگرهای مقایسه‌ای، و در آخر عملگرهای منطقی.

در جدول زیر چند مثال از کاربرد عملگرهای منطقی را ملاحظه می‌کنید. (در این مثالها، فرض را بر این گذاشته‌ایم که متغیر رشته‌ای Vehicle مقدار "Bike" دارد، و متغیر صحیح Price مقدار 200 .

عبارت منطقی	نتیجه
Vehicle = "Bike" And Price < 300	True (هر دو شرط ارزش True دارند)
Vehicle = "Car" Or Price < 500	True (یکی از شرط‌ها ارزش True دارد)
Not Price < 100	True (ارزش اولیهٔ شرط False است)
Vehicle = "Bike" Xor Price < 300	False (هر دو شرط ارزش True دارند، و یکسان هستند)

در تمرین این قسمت، برنامهٔ My User Validation را بگونه‌ای تغییر می‌دهیم که هویت کاربر (علاوه بر نام) توسط یک کلمهٔ رمز تعیین شود. برای این منظور از عملگر منطقی And استفاده خواهیم کرد.

حفاظت برنامه باکلمهٔ رمز: عملگر And

۱ روال Button1_Click را در ادیتور کُد باز کنید.

۲ دستور Dim را بصورت زیر تغییر دهید، و یک متغیر جدید بنام Pass تعریف کنید:

```
Dim UserName, Pass As String
```

۳ دستور زیر را بین دستور InputBox و If...Then (بین خط دوم و سوم) وارد کنید:

```
Pass = InputBox("Enter your password.")
```

٤ دستور If...Then را بصورت زیر تغییر دهید:

```
If UserName = "Henry" And Pass = "flower" Then
```

این دستور با استفاده از عملگر منطقی And ، نام کاربر و کلمهٔ رمز وی را همزمان چک می‌کند.

۵ دستور ElseIf را نیز بصورت زیر تغییر دهید:

```
ElseIf UserName = "Felix" And Pass = "sand" Then
```

در این دستور نیز با عملگر منطقی And ، نام کاربر و کلمهٔ رمز (بصورت یک ترکیب واحد) چک شده است.

۶ دکمهٔ Start را کلیک کنید، تا پروژه کامپایل و اجرا شود.

۷ دکمهٔ Sign In را کلیک کرده، و پس از وارد کردن نام **Henry** ، OK را کلیک کنید. با این کار، پنجرهٔ وارد کردن کلمهٔ رمز ظاهر می‌شود.

۸ کلمهٔ رمز **flower** را وارد کرده، و OK را کلیک کنید. از آنجائیکه نام و کلمهٔ رمز هر دو صحیح هستند، برنامه تصویر «هِنری» را نمایش خواهد داد.

۹ OK را کلیک کنید، تا پنجرهٔ پیام بسته شود.

۱۰ مجدداً دکمهٔ Sign In را کلیک کرده، و پس از وارد کردن نام **Felix** ، OK را کلیک کنید. این بار هم برنامه تقاضای کلمهٔ رمز می‌کند.

۱۱ کلمهٔ رمز **sand** را وارد کرده، و OK را کلیک کنید. باز هم نام و کلمهٔ رمز هر دو صحیح هستند، و برنامه به «فلیکس» اجازهٔ ورود می‌دهد.

۱۲ یک بار دیگر سعی کنید با نامهای Henry یا Felix و کلمهٔ رمز اشتباه وارد برنامه شوید؛ بله، برنامهٔ ما باندازهٔ کافی با هوش است، و بدون اینکه کاری انجام دهد، بسته می‌شود.

نکته

اگر می‌خواهید یک برنامهٔ واقعی گرفتن کلمهٔ رمز بنویسید، بجای تابع InputBox از یک جعبه متن (کنترل TextBox) برای این منظور استفاده کنید. کنترل جعبه متن خاصیتی دارد بنام PasswordChar ، که بجای حروف واقعی کلمهٔ رمز فقط یک کاراکتر خاص (مثلاً، *) نشان می‌دهد، و باعث می‌شود تا کلمهٔ رمز از چشم افراد نامحرم مخفی بماند. خاصیت دیگری که در این زمینه می‌تواند مفید باشد، خاصیت MaxLength است؛ این خاصیت تعداد حروفی که کاربر می‌تواند وارد کند، را محدود می‌کند.

اتصال کوتاه ساختار تصمیم‌گیری با AndAlso و OrElse

ویژوال بیسیک .NET دو عملگر منطقی جدید دارد، بنامهای AndAlso و OrElse ، که می‌توانید از آنها در ترکیبهای شرطی استفاده کنید. این عملگرهای جدید بترتیب مشابه And و Or هستند، اما خصوصیات دیگری نیز دارند که برای برنامه‌نویسان باتجربه بسیار جالب است.

یک دستور If را که در آن از AndAlso استفاده شده، در نظر بگیرید. برای آنکه دستورات ذیل این If اجرا شوند، هر دو شرط آن باید مقدار True داشته باشند. اما اگر اولین شرط False باشد، ویژوال بیسیک بدون آنکه شرط دوم را تست کند، به اولین دستور بعد از بلوک If (و یا بخش Else -اگر وجود داشته باشد) می‌رود. این ارزیابی ناقص (که به آن اتصال کوتاه هم گفته می‌شود) منطقی بنظر می‌رسد، چون وقتی یکی از اجزاء شرط False باشد، نتیجهٔ عملگر And دیگر هرگز نمی‌تواند True شود -و بررسی شرط دیگر کاری جز اتلاف وقت نیست.

عملگر OrElse نیز رفتار مشابهی دارد؛ یک دستور If را که در آن از OrElse استفاده شده، در نظر بگیرید. برای آنکه این بلوک If اجرا شود، کافیست حداقل یکی از شرطهای آن True شود -اگر اولین شرط True

باشد، ویژوال بیسیک دیگر شرط دوم را تست نمی‌کند، و بلافاصله بلوک If را اجرا خواهد کرد (چون نتیجهٔ این عبارت هرگز False نخواهد بود).

به مثال زیر نگاه کنید:

```
Dim Number As Integer = 0
If Number = 1 AndAlso MsgBox("Second condition test") Then
    MsgBox("Inside If")
Else
    MsgBox("Inside Else")
End If
```

استفاده از تابع MsgBox بعنوان یک شرط در عبارات شرطی چندان معمول نیست، اما کاملاً مجاز و درست است - این کار به ما فرصت می‌دهد تا عملکرد اتصال کوتاه را بهتر ببینیم. توجه کنید که پیام "Second condition test" فقط وقتی ظاهر می‌شود که شرط اول True باشد (یعنی Number = 0)، در غیراینصورت عملگر AndAlso دستور If را اتصال کوتاه کرده، و این تابع (شرط دوم) را اجرا نخواهد کرد (و این همان اتفاقی است که در کُد بالا رُخ می‌دهد، چون شرط Number = 1 مقدار False برمی‌گرداند). اما اگر در خط اول مقدار متغیر Number را از 0 به 1 تغییر دهید، ویژوال بیسیک مجبور است شرط دوم را هم تست کند، بنابراین تابع MsgBox را اجرا خواهد کرد.

مثال دیگری از طرز کار عملگر AndAlso را در قطعه کُد زیر می‌بینید. در این برنامه از یک شرط پیچیده‌تر برای تعیین «سن سگی» یک انسان استفاده کرده‌ایم:

```
Dim HumanAge As Integer
HumanAge = 7
'One year for a dog is seven years for a human
If HumanAge <> 0 AndAlso 7 / HumanAge <= 1 Then
    MsgBox("You are at least one dog year old")
Else
    MsgBox("You are less than one dog year old")
End If
```

این روتین بسادگی تست می‌کند که آیا سن فرد از ۷ سال کمتر است یا خیر. (برای تعیین «سن سگی» یک انسان، کافیست سن وی را بر ۷ تقسیم کنیم. برای مثال، یک انسان ۲۸ ساله، ۴ سال سگی سن دارد.) شرط اول دستور If چک می‌کند، که سن فرد بدرستی وارد شده است (سن یک انسان هرگز 0 یا منفی نیست!). شرط دوم تست می‌کند، که آیا این فرد حداقل ۷ سال سن دارد یا خیر. اگر هر دو شرط True باشند، برنامه پیام "You are at least one dog year old" را نشان خواهد داد. و اگر سن فرد از ۷ سال کمتر باشد، پیام "You are less than one dog year old" نمایش داده خواهد شد.

اما اگر سن فرد را (در خط دوم) به 0 ست کنیم، چه اتفاقی می‌افتد؟ در این حالت شرط اول False شده، و ویژوال بیسیک بدون اینکه سراغ شرط دوم برود، قسمت Else را اجرا می‌کند، و خطای تقسیم بر صفر در شرط دوم دیگر اتفاق نخواهد افتاد - این هم یکی دیگر از مزایای اتصال کوتاه. ولی اگر بخواهید همین برنامه

را در ویژوال بیسیک ۶ اجرا کنید، به محض اینکه کامپایلر شروع به ارزیابی شرط دوم کند، با خطای تقسیم بر صفر مواجه شده، و متوقف خواهد شد.

عملگرهای AndAlso و OrElse (علاوه بر جلوگیری از خطاهای زمان اجرا) کارایی و سرعت برنامه را هم بهبود می‌بخشند، چون شرطهایی که ارزیابی آنها بی‌تأثیر است، دیگر بیهوده تست نخواهند شد.

ساختار تصمیم‌گیری Select Case

یکی دیگر از ساختارهای تصمیم‌گیری ویژوال بیسیک .NET دستور Select Case است (این دستور را قبلاً در فصلهای ۳ و ۵ دیده‌اید). ساختار Select Case بسیار شبیه If...Then...Else است، با این تفاوت که برای تست‌هایی که بر اساس یک کلید صورت می‌گیرند، بهینه شده است (و از If...Then بسیار خوانا‌تر است).

شکل کلی یک دستور Select Case چنین است:

```
Select Case variable
    Case value1
        'program statements executed if variable = value1
    Case value2
        'program statements executed if variable = value2
    Case value3
        'program statements executed if variable = value3
    ⋮
    Case Else
        'program statements executed if no match is found
End Select
```

همانطور که می‌بینید، این ساختار با Select Case شروع شده، و با End Select پایان می‌یابد؛ و *variable* نیز می‌تواند هر چیزی که یک مقدار برمی‌گرداند، باشد (*value1* ، *value2* و *value3* نیز مقدارهایی هستند که انتظار دارید این متغیر داشته باشد). اگر متغیر *variable* با یکی از این مقدارها منطبق شود، ویژوال بیسیک دستورات Case مربوط به آن را اجرا کرده، و سپس به اولین دستور بعد از End Select می‌رود. تعداد Case ها در یک دستور Select Case هیچ محدودیتی ندارد، و هر Case نیز می‌تواند چندین مقدار را چک کند (در این حالت، مقدارها را باید با کاما - , - از هم جدا کنید).

در مثال زیر، با یک دستور Select Case پیامی مناسب برای هر گروه سنی چاپ می‌کنیم:

```
Dim Age As Integer
Age = 18

Select Case Age
    Case 16
        Label1.Text = "You can drive now!"
    Case 18
        Label1.Text = "You can vote now!"
    Case 21
```

```
        Label1.Text = "You are in the drive for success!"
    Case 65
        Label1.Text = "Time to retire and have fun!"
End Select
```

همانطور که می‌توانید ببینید، پیامی که چاپ خواهد شد، "You can vote now!" است، چون متغیر Age را در ابتدای برنامه به 18 ست کرده‌ایم.

ساختار Select Case دارای قسمتی بنام Case Else نیز هست، برای مواردی که در هیچیک از دستورات Case نمی‌گنجد. به مثال زیر نگاه کنید:

```
Dim Age As Integer
Age = 25

Select Case Age
    Case 16
        Label1.Text = "You can drive now!"
    Case 18
        Label1.Text = "You can vote now!"
    Case 21
        Label1.Text = "You are in the drive for success!"
    Case 65
        Label1.Text = "Time to retire and have fun!"
    Case Else
        Label1.Text = "You're a great age! Enjoy it!"
End Select
```

از آنجائیکه این بار متغیر Age دارای مقدار 25 است، دستور Case Else اجرا شده و پیام "You're a great age! Enjoy it!" نمایش داده خواهد شد.

استفاده از عملگرهای مقایسه در ساختار Select Case

برای تست یک محدوده از مقادیر در دستور Select Case می‌توان از عملگرهای مقایسه استفاده کرد؛ و این عملگرها را باید با کلمات کلیدی Is و To ترکیب کرد. کلمهٔ کلیدی Is به کامپایلر می‌گوید که متغیر تست Case را با لیستی که بعد از Is می‌آید، مقایسه کند؛ کلمهٔ کلیدی To نیز برای تعیین یک محدوده از مقادیر بکار می‌رود. در مثال زیر همان ساختار Select Case صفحهٔ قبل را می‌بینید که این بار بجای یک سن خاص، یک محدودهٔ سنی را تست می‌کند:

```
Select Case Age
    Case Is < 13
        Label1.Text = "Enjoy your youth!"
    Case 13 To 19
        Label1.Text = "Enjoy your teens!"
    Case 21
        Label1.Text = "You are in the drive for success!"
    Case Is > 100
        Label1.Text = "Looking good!"
```

```
    Case Else
        Label1.Text = "That's a nice age to be."
End Select
```

در این مثال، اگر سن فرد کمتر از ۱۳ سال باشد، پیام "Enjoy your youth!" نمایش داده می‌شود، اگر سن وی بین ۱۳ و ۱۹ باشد، پیام "Enjoy your teens!" ، و الی آخر.

ساختار Select Case از If...Then کارایی بهتری دارد، بویژه در مواردی که تعداد شاخه‌های تصمیم‌گیری زیاد باشد. اما اگر تعداد شاخه‌های تصمیم‌گیری کم باشد، بهتر است از همان ساختار If...Then استفاده کنید.

در تمرین این قسمت، برای پردازش ورودی کاربر (که آنرا از یک لیست انتخاب می‌کند) از ساختار Select Case استفاده کرده‌ایم.

استفاده از Select Case برای پردازش یک جعبه لیست

۱ از منوی File|New آیتم Project را انتخاب کنید، تا پنجرهٔ «پروژهٔ جدید» ظاهر شود.

۲ یک پروژهٔ Visual Basic Windows Application بنام **My Case Greeting** در پوشهٔ c:\vbnetsbs\chap06 ایجاد کنید.

۳ کنترل Label را انتخاب کرده، و یک برچسب بزرگ در بالای فرم بکشید؛ این برچسب عنوان برنامه را نمایش خواهد داد.

٤ یک برچسب کوچک نیز زیر برچسب قبلی رسم کنید.

۵ کنترل ListBox را انتخاب کرده، و یک جعبه لیست زیر برچسب‌ها بکشید.

۶ مجدداً کنترل Label را انتخاب کرده، و دو برچسب دیگر زیر این جعبه لیست رسم کنید؛ این برچسب‌ها خروجی برنامه را نمایش خواهند داد.

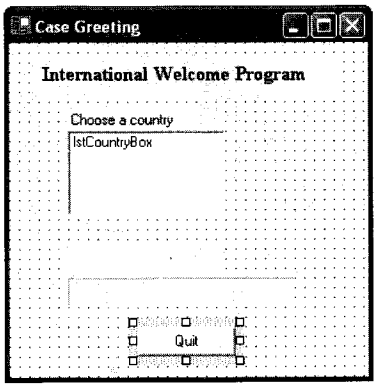
۷ کنترل Button را انتخاب کرده، و یک دکمهٔ کوچک در پائین فرم بکشید.

۸ به پنجرهٔ خواص بروید، و خواص کنترلهایی که ایجاد کردید، را با توجه به جدول زیر ست کنید. (چون تعداد کنترلهای مشابه زیاد است، برای متمایز کردن آنها از خاصیت Name هم استفاده کرده‌ایم.)

شیء	خاصیت	مقدار
Form1	Text	"Case Greeting"
Label1	Font	Times New Roman, Bold, 12-point
	Name	lblTitle
	Text	"International Welcome Program"
Label2	Name	lblTextBoxLabel
	Text	"Choose a country"
Label3	Font	10-point
	Name	lblCountry

شیء	خاصیت	مقدار
	Text	(خالی)
Label4	BorderStyle	Fixed3D
	ForeColor	Red
	Name	lblGreeting
	Text	(خالی)
ListBox1	Name	lstCountryBox
Button1	Name	btnQuit
	Text	"Quit"

در پایان باید فرمی شبیه شکل زیر داشته باشید:



و حالا باید کُد برنامه را بنویسیم.

۹ روی فرم برنامه دو-کلیک کنید، تا روال رویداد Form1_Load در ادیتور کُد باز شود.

۱۰ دستورات زیر را در این روال بنویسید:

```
lstCountryBox.Items.Add("England")
lstCountryBox.Items.Add("Germany")
lstCountryBox.Items.Add("Mexico")
lstCountryBox.Items.Add("Italy")
```

در این کُد با استفاده از متد Add آیتمهای جعبه لیست را به آن اضافه کرده‌ایم.

۱۱ با کلیک کردن روی برگهٔ Form1.vb [Design] (در بالای ادیتور کُد) به فرم برنامه برگردید، و روی جعبه لیست lstCountryBox دو-کلیک کنید، تا روال رویداد lstCountryBox_SelectedIndexChanged باز شود.

۱۲ دستورات زیر را در این روال وارد کنید:

```
lblCountry.Text = lstCountryBox.Text
```

```
Select Case lstCountryBox.SelectedIndex
   Case 0
      lblGreeting.Text = "Hello, programmer"
   Case 1
      lblGreeting.Text = "Hallo, programmierer"
   Case 2
      lblGreeting.Text = "Hola, programador"
   Case 3
      lblGreeting.Text = "Ciao, programmatore"
End Select
```


اولین دستور این کُد آیتمی را که کاربر از جعبه لیست انتخاب کرده، در برچسب سوم (برچسب lblCountry) کپی می‌کند (و برای این کار از خاصیت Text جعبه لیست استفاده می‌کند). بقیهٔ این کُد یک ساختار Select Case است، که از خاصیت SelectedIndex جعبه لیست بعنوان متغیر تست استفاده می‌کند. عددی که در این خاصیت وجود دارد، نشاندهندهٔ آیتمی است که انتخاب شده –این عدد از 0 برای اولین آیتم شروع شده و تا تعداد کل آیتمها منهای یک ادامه می‌یابد. با استفاده از این خاصیت بسرعت می‌توان فهمید که کاربر کدام آیتم را در جعبه لیست انتخاب کرده است.

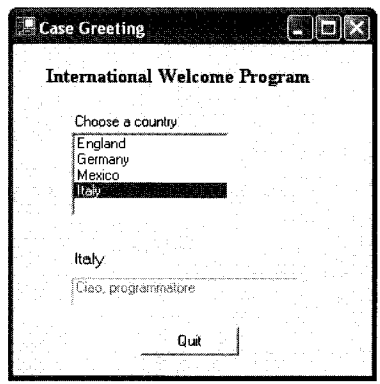
۱۳ دوباره به فرم برنامه برگردید، و این بار روی دکمهٔ Quit دو–کلیک کنید، تا روال رویداد btnQuit_Click در ادیتور کُد باز شود.

۱۴ دستور **End** را در این روال بنویسید.

۱۵ برای ذخیره کردن برنامه، دکمهٔ Save All را کلیک کنید.

۱۶ دکمهٔ Start را کلیک کنید، تا برنامه کامپایل و اجرا شود.

۱۷ یکی از کشورهای لیست Choose a country را انتخاب کنید، تا برنامه پیام خوشامدی به زبان آن کشور نمایش دهد (شکل زیر را ببینید):



۱۸ برای بستن برنامه، دکمهٔ Quit را کلیک کنید.

ساختارهای If...Then و Select Case جزء مهمترین دستورات ویژوال بیسیک محسوب می‌شوند، و در فصلهای آینده باز هم از آنها برای تصمیم‌گیری در برنامه استفاده خواهیم کرد.

یک گام فراتر: آشکارسازی رویدادهای ماوس

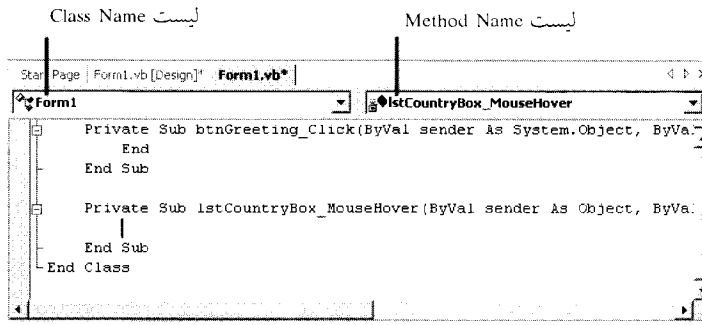
در ابتدای این فصل دربارهٔ تعدادی از رویدادهای ویژوال بیسیک .NET صحبت کردم، و در ادامه دیدید که چگونه می‌توان این رویدادها را ساختارهایی از قبیل If...Then و Select Case مدیریت کرد. در این قسمت یاد می‌گیرید چگونه حرکات و رویدادهای ماوس را کنترل کنید. یکی از رویدادهایی که ماوس از آن پشتیبانی می‌کند، MouseHover نام دارد - این رویداد زمانی اتفاق می‌افتد که ماوس روی یک شیء قرار گیرد. در تمرین این قسمت تغییری در برنامهٔ My Case Greeting می‌دهیم تا وقتی کاربر ماوس را روی جعبه لیست lstCountryBox می‌برد و لحظه‌ای تأمل می‌کند (احتمالاً به این علت که نمی‌داند چکار باید بکند)، پیامی با مضمون "Please click the country name" نمایش داده شود.

برنامه‌ای برای کنترل رویدادهای ماوس

۱ ادیتور کُد را باز کنید.

۲ لیست Class Name را (در بالای ادیتور کُد) باز کرده، و شیء lstCountryBox را انتخاب کنید.

۳ لیست Method Name را باز کرده، و رویداد MouseHover را انتخاب کنید. با این کار، ویژوال بیسیک شما را به روال رویداد lstCountryBox_MouseHover می‌برد (شکل زیر را ببینید):



هر شیء ویژوال بیسیک یک رویداد پیش‌فرض دارد، که وقتی (در فرم برنامه) روی آن شیء دو-کلیک می‌کنید، باز می‌شود. برای باز کردن سایر رویدادها، باید آنها را از لیست Method Name انتخاب کنید.

٤ دستورات زیر را در روال lstCountryBox_MouseHover بنویسید:

```
If lstCountryBox.SelectedIndex < 0 Or lstCountryBox.SelectedIndex > 4 Then
    lblGreeting.Text = "Please click the country name"
End If
```

در اینجا شرط دستور If از دو قسمت که توسط عملگر Or با هم ترکیب شده‌اند، تشکیل شده است.

در این شرط فرض بر آنست که اگر مقدار خاصیت SelectedIndex شیء lstCountryBox بزرگتر از 0 و کوچکتر از 4 باشد، کاربر دیگر نیازی به راهنمایی ندارد (چون قبلاً کشور موردنظر را انتخاب کرده است!). اما اگر مقدار این خاصیت در محدودهٔ 4-0 نباشد، کاربر هنوز چیزی انتخاب نکرده، و لازمست پیام "Please click the country name" به وی نشان داده شود.

(بعد از ذخیره کردن پروژه) دکمهٔ Start را کلیک کنید، تا برنامه کامپایل و اجرا شود.

۵ ماوس را روی جعبه لیست Chosse a country ببرید، و بدون آنکه کلیک کنید، چند لحظه صبر

۶ کنید. بله! پیام راهنمای "Please click the country name" در برچسب lblGreeting ظاهر می‌شود:

یکی از کشورها را انتخاب کنید، تا پیام راهنما ناپدید شود.

۷ برای بستن برنامه، دکمهٔ Quit را کلیک کنید.

۸ این هم از کنترل رویدادهای ماوس. در فصل ۱۶ با این تکنیک بیشتر آشنا شده، و در یک برنامهٔ انیمیشن از آن استفاده خواهیم کرد.

مرجع سریع فصل ۶

برای ...	انجام دهید
نوشتن یک عبارت شرطی	یکی از از عملگرهای شرطی (=، <>، <، >، =<، =>) را بین دو مقدار قرار دهید.

ایجاد ساختار تصمیم‌گیری If...Then	از ساختاری به شکل زیر استفاده کنید:
	If *condition1* Then
	statements executed if condition1 is True
	ElseIf *condition2* Then
	statements executed if condition2 is True
	Else
	statements executed if none are True
	End If

ایجاد ساختار تصمیم‌گیری Select Case	از ساختاری به شکل زیر استفاده کنید: Select Case *variable* Case *value1* *statements executed if value1 matches* Case *value2* *statements executed if value2 matches* Case Else *statements executed if none match* End Select
انجام دو مقایسه در یک عبارت شرطی	از عملگرهای منطقی (And، Or، Not، Xor) بین دو قسمت شرط استفاده کنید.
اتصال کوتاه دستور If...Then	از عملگرهای AndAlso و OrElse بین اجزاء ترکیب شرطی استفاده کنید.
نوشتن یک روال رویداد	در ادیتور کُد، شیء را از لیست Class Name ، و رویداد موردنظر را لیست Method Name انتخاب کنید؛ سپس، کُد لازم را در این روال رویداد بنویسید.

فصل

۷

حلقه‌ها و تایمرها

در این فصل یاد می گیرید چگونه :

- ✔ برای اجرای چند دستور بدفعات معین، از حلقهٔ For...Next استفاده کنید.
- ✔ رشته‌های متن را به هم بچسبانید، و در یک جعبه متن چند-خطی نمایش دهید.
- ✔ برای اجرای چند دستور تا برقراری یک شرط خاص، از حلقهٔ Do...Loop استفاده کنید.
- ✔ برای اجرای یک قطعه کُد در زمانهای معین، از شیء تایمر (Timer) استفاده کنید.
- ✔ یک ساعت دیجیتالی بسازید.

در فصل ۶ دیدید که چگونه می‌توان با ساختارهای تصمیم‌گیری If...Then و Select Case برنامه را به مسیرهای مختلف هدایت کرد. در این فصل با تکرار سروکار داریم، و خواهید دید که چگونه به کمک حلقه‌های For...Next و Do...Loop ، می‌توان یک قطعه کُد را بدفعات معین (و یا تا رسیدن به یک حالت خاص) تکرار کرد. تکنیک دیگری که در این فصل فرا خواهید گرفت، نمایش متن در جعبه متن (شیء TextBox) بصورت چند-خطی (multi-line) است؛ برای این کار از عملگر & کمک خواهیم گرفت. و در پایان، خواهید دید که چگونه می‌توان با استفاده از شیء تایمر یک قطعه کُد را در فواصل معین اجرا کرد.

تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- در ویژوال بیسیک ۶ می‌توانستید با متد Print مستقیماً متنی را روی فرم برنامه بنویسید. اما در ویژوال بیسیک .NET متد Print فقط روی فایل یا دیسک می‌تواند بنویسد.
- در ویژوال بیسیک ۶ حلقهٔ خاصی وجود داشت بنام While...Wend - در ویژوال بیسیک .NET ، این حلقه به While...End While تبدیل شده، تا شبیه سایر ساختارهای مشابه باشد.
- شیء تایمر (Timer) در ویژوال بیسیک .NET چند تفاوت کوچک با شیء متناظر خود در ویژوال بیسیک ۶ دارد؛ برای مثال، روال رویداد Timer1_Timer به Timer1_Tick تغییر نام داده است. علاوه بر این، برای غیرفعال کردن تایمر حتماً باید از خاصیت Enable استفاده کنید، و بر خلاف ویژوال بیسیک ۶ ست کردن فاصله زمانی تایمر به 0 آنرا غیرفعال نمی‌کند.

حلقهٔ For...Next

برای تکرار یک یا چند دستور بدفعات معین می‌توان از حلقهٔ For...Next بهره گرفت. حلقه‌ها برای کارهای تکراری بسیار مفید هستند، و بار زیادی از دوش برنامه‌نویس برمی‌دارند. در واقع، حلقهٔ For...Next چیزی نیست جز شکل متراکم و فشردهٔ تعداد زیادی دستورات تکراری و مشابه. شکل کلی یک حلقهٔ For...Next چنین است:

```
For variable = strat To end
    statements to be repeated
Next [variable]
```

در یک حلقهٔ For...Next چهار عنصر ضروری وجود دارد: کلمات کلیدی For ، To ، Next ، و عملگر = . این متغیر *variable* و دو مقدار *start* و *end* هستند، که تعیین می‌کنند این حلقه چند بار باید تکرار شود. (توجه داشته باشید که متغیر *variable* قبلاً باید تعریف شده باشد.) تمام دستوراتی که بین For و Next وجود دارند، به تعداد مشخص شده در دستور For تکرار می‌شوند. برای مثال، حلقهٔ زیر باعث می‌شود تا بلندگوی کامپیوتر چهار بار پشت سر هم بوق بزند (اگر چه ممکنست این بوقها را بزحمت بشنوید):

```
Dim i As Integer
For i = 1 To 4
    Beep()
Next i
```

این حلقه کار چهار دستور ()Beep متوالی را انجام می‌دهد، و در واقع کامپایلر آنرا بصورت زیر تفسیر می‌کند:

```
Beep()
Beep()
Beep()
Beep()
```

در حلقهٔ فوق، متغیر حلقه i است، که قبل از شروع حلقه آنرا تعریف کرده‌ایم. هر بار که حلقه تکرار می‌شود، مقدار این متغیر (که به شمارندهٔ حلقه نیز معروفست) یکی اضافه می‌شود. مقدار *start* در این حلقه، 1 و مقدار *end* در آن 4 است. همانطور که بعداً خواهید دید، از شمارندهٔ حلقه در داخل آن هم می‌توان استفاده کرد، و به نتایج جالبی دست یافت.

نمایش شمارندهٔ حلقه در یک جعبه متن

متغیر (یا شمارندهٔ) حلقه نیز متغیریست شبیه سایر متغیرهای دیگر، و می‌توان از آن مانند سایر متغیرها استفاده کرد. یکی از کارهای عملی که با این متغیر می‌توان انجام داد، نمایش مقدار آن در یک جعبه متن (کنترل TextBox) است. تا اینجا همواره از جعبه متن برای نمایش یک خط متن استفاده کردیم؛ ولی در تمرین این قسمت خواهید دید که چگونه می‌توان در یک کنترل TextBox چند خط متن نمایش داد. راز این کار، در True کردن خاصیت Multiline جعبه متن، و ست کردن خاصیت ScrollBars آن به Vertical است. با این تنظیمات ساده، جعبه متن می‌تواند تعداد زیادی خط را در خود جای دهد، و حتی می‌توانید در آن بالا و پائین بروید (مانند برنامهٔ Word).

نمایش اطلاعات با استفاده از حلقهٔ For...Next

۱ ویژوال استودیو .NET را باز کرده، و یک پروژهٔ Visual Basic Windows Application بنام **My For Loop** در پوشهٔ c:\vbnetsbs\chap07 ایجاد کنید.

۲ روی کنترل Button در جعبه ابزار دو-کلیک کنید - ویژوال استودیو یک دکمه با اندازهٔ استاندارد در وسط فرم قرار می‌دهد، که بعداً می‌توانید آنرا به هر جایی که مایل هستید، منتقل کنید (و یا اندازهٔ آنرا تغییر دهید). این روش دیگری برای قرار دادن کنترل روی فرمهاست، که از روش قبلی نیز سریعتر است.

۳ دکمهٔ Button1 را به بالای فرم منتقل کنید.

٤ پنجرهٔ خواص را باز کرده، و خاصیت Text دکمه را به **Loop** ست کنید.

۵ روی کنترل TextBox در جعبه ابزار دو-کلیک کنید - ویژوال استودیو یک جعبه متن نسبتاً کوچک در وسط فرم قرار می‌دهد.

۶ خاصیت Multiline جعبه متن را به True ، و خاصیت دکمه ScrollBars آنرا به Vertical ست کنید.

۷ خاصیت Text جعبه متن را پاک کنید.

۸ جعبه متن را به زیر دکمه منتقل کنید، و آنرا به اندازه‌ای در آورید که تقریباً تمام سطح فرم را بپوشاند.

۹ روی دکمهٔ Loop دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد باز شود.

۱۰ دستورات زیر را در این روال رویداد بنویسید:

```
Dim i As Integer
Dim Wrap As String
Wrap = Chr(13) & Chr(10)
```

```
For i = 1 To 10
    TextBox1.Text = TextBox1.Text & "Line " & i & Wrap
Next i
```

در کُد فوق ابتدا دو متغیر یکی از نوع عدد صحیح (بنام i) و دیگری از نوع رشته (بنام Wrap) تعریف کرده‌ایم. سپس، به متغیر دوم مقداری داده‌ایم که معادل زدن کلید Enter روی صفحه کلید است (در اصطلاح برنامه‌نویسان به این ترکیب سرِ خط - carriage return -گفته می‌شود). البته می‌توانستیم از این ترکیب مستقیماً هم استفاده کنیم، ولی قرار دادن آنها در یک متغیر کُد برنامه را خواناتر می‌کند.

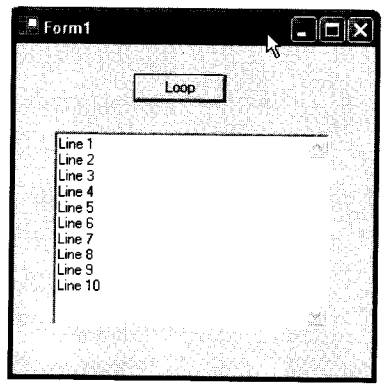
پس از آن در یک حلقهٔ For...Next ، ده بار جملهٔ Line X (که در آن X شمارهٔ خط است) را در جعبه متن TextBox1 نوشته‌ایم. توجه کنید که چگونه برای بهم چسباندن رشته‌های متن از عملگر & استفاده کرده‌ایم؛ و برای اینکه متن قبلی جعبه متن از بین نرود، مقدار جدید را به خاصیت TextBox1.Text چسبانده‌ایم. وقتی متغیر Wrap را به آخر رشته می‌چسبانیم، درست مثل اینست که در پایان جمله یک Enter زده باشیم.

وقتی شروع به نوشتن یک حلقهٔ For می‌کنید، ویژوال بیسیک بلافاصله Next را به انتهای آن اضافه می‌کند. تنها کاری که ما کرده‌ایم، اضافه کردن متغیر i به این Next است -البته این کار اختیاریست، ولی باعث وضوح بیشتر کُد خواهد شد، و نشان می‌دهد که این Next مربوط به کدام For است.

۱۱ با کلیک کردن دکمهٔ Save All ، پروژه را ذخیره کنید. (متن کامل این پروژه را می‌توانید در پوشهٔ x:\vbnetsbs\chap07\for loop روی CD ضمیمه بیابید.)

۱۲ دکمهٔ Start را کلیک کنید، تا پروژه کامپایل و اجرا شود.

۱۳ دکمهٔ Loop را کلیک کنید. با این کار حلقهٔ For...Next ده بار اجرا خواهد شد:



۱۴ یک بار دیگر دکمهٔ Loop را کلیک کنید - ده خط دیگر به جعبه متن اضافه می‌شود، و از آنجائیکه تمام این خطوط در جعبه متن جا نمی‌شوند، یک میلهٔ لغزشی عمودی (vertical scrollbar) در کنار آن ظاهر می‌شود، که با آن می‌توانید در جعبه متن بالا و پائین بروید.

نکته

آیا نگران پُر شدن جعبه متن هستید؟ جای نگرانی نیست؛ یک کنترل TextBox می‌تواند تا 32 KB متن را در خود جای دهد. اگر متنی که دارید از این مقدار بیشتر است، می‌توانید از کنترل RichTextBox (که امکانات فرمتی بیشتری هم دارد) استفاده کنید.

۱۵ دکمهٔ Close را کلیک کنید، تا برنامه بسته شود.

نکته

حلقهٔ For...Next طول برنامه و تعداد دستورات آن را بشدت کاهش می‌دهد. برای مثال، در برنامهٔ My For Loop یک حلقهٔ سه خطی جایگزین ده دستور جداگانه شده است.

حلقه‌های For...Next پیچیده

متغیر شمارندهٔ حلقه‌های For...Next می‌تواند ابزار بسیار مفیدی در دست برنامه‌نویس باشد، و با بکار بردن کمی تخیل می‌توانید از آن استفاده‌های جالبی کنید. یک حلقهٔ For...Next مقید نیست که حتماً اعداد را بصورت 1 ، 2 ، 3 ، 4 (و الی آخر) بشمارد، و می‌توان با تغییر مقدار *start* آنرا از نقطهٔ دیگری شروع کرد. یا اینکه می‌توان با استفاده از کلمهٔ کلیدی Step گام حلقه را تغییر داد. به مثال زیر نگاه کنید:

```
Dim i As Integer
Dim Wrap As String
Wrap = Chr(13) & Chr(10)

For i = 5 To 25 Step 5
    TextBox1.Text = TextBox1.Text & "Line " & i & Wrap
Next i
```

خروجی این کُد به شکل زیر خواهد بود:

```
Line 5
Line 10
Line 15
Line 20
Line 25
```

حتی لازم نیست که متغیر حلقه عدد صحیح باشد، و از اعداد اعشاری هم می‌توان بعنوان نقطهٔ شروع و یا گام حلقه استفاده کرد؛ برای مثال، خروجی حلقهٔ زیر

```
Dim i As Single
Dim Wrap As String
Wrap = Chr(13) & Chr(10)

For i = 1 To 2.5 Step 0.5
    TextBox1.Text = TextBox1.Text & "Line " & i & Wrap
Next i
```

به این شکل خواهد بود:

```
Line 1
Line 1.5
Line 2
Line 2.5
```

از متغیر حلقه حتی برای انجام محاسبات (و یا باز کردن فایلها) هم می‌توان استفاده کرد. در تمرین این قسمت خواهید دید که چگونه می‌توان با استفاده از یک حلقهٔ For...Next تعدادی از آیکون ویژوال بیسیک را (که در نام فایل آنها عدد وجود دارد) باز کرد و نمایش داد. (تعداد بیشتری از این آیکونها را می‌توانید در پوشهٔ c:\program files\microsoft visual studio .net\common7\graphics\icons\misc بیابید.)

باز کردن فایل با استفاده از حلقهٔ For...Next

۱ از منوی File|New آیتم Project را انتخاب کنید، تا پنجرهٔ «پروژهٔ جدید» ظاهر شود.

۲ یک پروژهٔ Visual Basic Windows Application بنام **My For Loop Icons** در پوشهٔ c:\vbnetsbs\chap07 ایجاد کنید.

۳ کنترل PictureBox را انتخاب کرده، و یک جعبه تصویر با اندازهٔ متوسط در بالای فرم (و وسط آن) بکشید.

٤ کنترل BoxButton را (از جعبه ابزار) انتخاب کرده، و یک دکمه با پهنای زیاد زیر جعبه تصویر رسم کنید.

۵ خواص این دو کنترل را مطابق جدول زیر ست کنید:

شیء	خاصیت	مقدار
PictureBox1	BorderStyle	Fixed3D
	SizeMode	StretchImage
Button1	Text	"Display four faces"

۶ روی دکمهٔ Display four faces دو-کلیک کنید، تا روال رویداد Button_Click در ادیتور کُد باز شود.

۷ کُد زیر را در این روال بنویسید:

```
Dim i As Integer
For i = 1 To 4
    PictureBox1.Image = System.Drawing.Image.FromFile _
      ("c:\vbnetsbs\chap07\face0" & i & ".ico")
    MsgBox("Click here for next face.")
Next
```

نکته

متد FromFile آنقدر طولانیست که در یک خط جا نمی‌شود، و بهمین دلیل با استفاده از کاراکتر ادامهٔ خط (کاراکتر _) آنرا به دو خط شکسته‌ایم. از این کاراکتر می‌توان در هر دستوری (بجز داخل رشته‌ها) استفاده، و آنرا به چند بخش کوچکتر تقسیم کرد.

در حلقهٔ فوق فایلهای آیکون در پوشهٔ c:\vbnetsbs\chap07 با متد FromFile باز می‌شوند. توجه کنید که چگونه نام فایل موردنظر از بهم چسباندن سه جزء مختلف (مسیر فایل، شمارهٔ فایل، و پسوند ico.) ایجاد شده است:

```
PictureBox1.Image = System.Drawing.Image.FromFile _
    ("c:\vbnetsbs\chap07\face0" & i & ".ico")
```

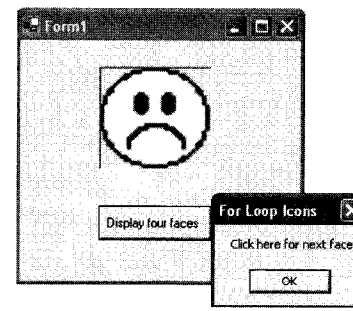
بدین‌ترتیب حلقهٔ فوق چهار فایل به نامهای face01.ico ، face02.ico ، face03.ico و face04.ico را در جعبه تصویر بار خواهد کرد. (شناخت فایلهای موردنظر و تشخیص یک الگوی تکراری در نام آنها، اولین قدم در نوشتن یک حلقهٔ مؤثر مانند مثال بالاست.)

نکته

در این کُد، تابع MsgBox() نقش یک ترمز را بازی می‌کند، که باعث می‌شود تا کاربر بتواند توالی تصاویر را بهتر ببیند. در برنامه‌های واقعی نیازی به استفاده از چنین ترمزهایی نیست.

۸ با کلیک کردن دکمهٔ Save All ، پروژه را ذخیره کنید.

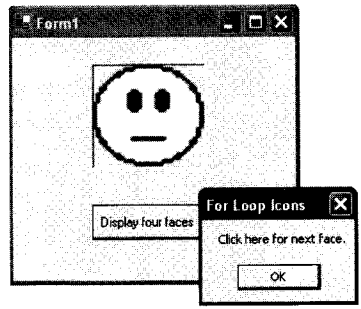
۹ با کلیک کردن دکمهٔ Start ، برنامه را اجرا کرده، و سپس دکمهٔ Display four faces را کلیک کنید. با اجرای اولین تکرار حلقهٔ For...Next ، پیام "Click here for next face." را خواهید دید:



نکته

احتمال زیادی وجود دارد که در اجرای این برنامه با خطا مواجه شوید؛ و مهمترین علت آن هم اشتباه در نوشتن مسیر فایلهاست. در صورت بروز این اتفاق، یک بار دیگر دستور بارکردن تصاویر را کنترل کنید، تا مطمئن شوید به مسیر درست اشاره می‌کند.

۱۰ برای نمایش تصویر بعدی، OK را کلیک کنید:



۱۱ برای دیدن تمام تصاویر باید سه بار دیگر OK را کلیک کنید. (در صورت تمایل می‌توانید باز هم این کار را تکرار کنید.)

۱۲ بعد از آن که باندازهٔ کافی برنامه را تست کردید، با کلیک کردن دکمهٔ Close آنرا ببندید.

یک روش بهتر برای بازکردن فایل‌ها

حلقه‌ای که در برنامهٔ قبل دیدید، چندان جالب و راحت نبود، و در واقع وجود دستور ()MsgBox به نوعی آنرا آزاردهنده کرده بود. آیا راهی برای خلاص شدن از شرّ این ()MsgBox وجود ندارد؟

اولین قدم برای حذف ()MsgBox مزاحم، تعریف یک متغیر عمومی است که در تمام فرم برنامه قابل دسترسی باشد. (در فصل ۵ دیدید که با انتقال دستور تعریف یک متغیر به بالای فرم، آن متغیر از تمام نقاط فرم قابل دسترسی می‌شود.) در تمرین این قسمت، تابع ()MsgBox (و حلقهٔ For...Next) را از روال Button1_Click حذف، و بجای آنها از یک متغیر عمومی بنام Counter برای کنترل نمایش آیکونها استفاده خواهیم کرد.

استفاده از متغیر عمومی Counter

۱ روال رویداد Button1_Click را در پروژهٔ My For Loop Icons باز کنید.

۲ بیرون این روال (و درست زیر قسمت Windows Forms Designer generated code) یک متغیر عمومی بنام Counter تعریف کنید:

```
Dim Counter As Integer = 1
```

مقدار دادن به یک متغیر در هنگام تعریف آن، یکی از ویژگیهای جدید ویژوال استودیو .NET است که باعث سهولت بسیاری هم می‌شود (در ویرایشهای قبلی ویژوال بیسیک این کار مجاز نبود).

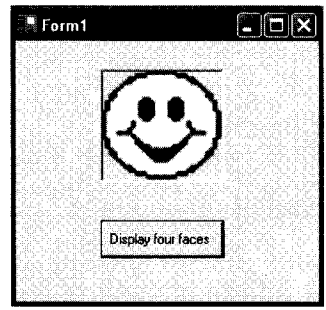
۳ کُد روال Button1_Click را مانند زیر تغییر دهید (توجه کنید که هر چیزی غیر از این دستورات را حذف کنید):

```
PictureBox1.Image = System.Drawing.Image.FromFile _
    ("c:\vbnetsbs\chap07\face0" & Counter & ".ico")
Counter += 1
If Counter = 5 Then Counter = 1
```

همانطور که می‌بینید، روش کار را بکلی عوض کرده‌ایم و دیگر خبری از حلقهٔ For...Next و تابع MsgBox نیست؛ متغیر i هم جای خود را به Counter داده است. در خط دوم، دستور Counter += 1 یکی به متغیر Counter اضافه می‌کند (قبلاً دیدید که، این دستور شکل فشردهٔ Counter = Counter + 1 است). در دستور سوم، اگر مقدار این متغیر به 5 رسیده باشد، آنرا به 1 ریست می‌کند، و کار از نو ادامه می‌یابد.

٤ با کلیک کردن دکمهٔ Start ، برنامه را کامپایل و اجرا کنید.

٥ چند بار روی دکمهٔ Display four faces کلیک کنید - دقت کنید که چگونه حالت آیکون، از اخمو به خندان، تغییر می‌کند:



٦ بعد از آن که باندازهٔ کافی برنامه را تست کردید، با کلیک کردن دکمهٔ Close آنرا ببندید.

همانطور که می‌بینید، دیگر خبری از توقفهای آزاردهنده نیست، و برنامه خیلی بهتر عمل می‌کند. البته مشکل برنامهٔ قبلی ربطی به توانائیهای حلقهٔ For...Next ندارد، و این خود ما بودیم که آنطور می‌خواستیم.

دستور Exit For

در اکثر مواقع حلقه‌های For...Next بخوبی و خوشی کامل می‌شوند، و به کار خود پایان می‌دهند. اما گاهی پیش می‌آید که بخواهیم قبل از پایان یک حلقهٔ For...Next (و در شرایطی خاص) از آن خارج شویم. برای این منظور دستور ویژه‌ای وجود دارد بنام Exit For ، که به اجرای حلقهٔ For...Next خاتمه می‌دهد، و برنامه را از اولین دستور بعد از حلقه پی می‌گیرد.

در مثال زیر، یک حلقهٔ For...Next می‌بینید که در حالت عادی ۱۰ بار تکرار می‌شود، و هر بار نامی را کاربر گرفته و در یک جعبه متن می‌نویسد. ولی اگر کاربر در حین کار تمایلی به ادامهٔ آن نداشت، می‌تواند با وارد کردن کلمهٔ "Done" به اجرای برنامه پایان دهد:

```
Dim i As Integer
Dim InpName As String
For i = 1 To 10
    InpName = InputBox("Input your name, or type Done to quit.")
    If InpName = "Done" Then Exit For
    TextBox1.Text = InpName
Next
```

در این کُد، اگر کاربر در تابع InputBox کلمهٔ "Done" را وارد کند، شرط دستور If مقدار True پیدا می‌کند، و با اجرای دستور Exit For حلقهٔ For...Next (و برنامه) خاتمه خواهد یافت.

حلقهٔ Do...Loop

بر خلاف For...Next ، حلقهٔ Do...Loop تا زمانیکه شرط خاصی محقق باشد (True باشد)، ادامه خواهد یافت. حلقهٔ Do برای مواردی مناسب است، که از قبل دقیقاً ندانید به چند تکرار نیاز دارید. برای مثال، شاید بخواهید عملیاتی آنقدر بدون وقفه تکرار شود، تا کاربر کلمهٔ "Done" را وارد کند.

حلقهٔ Do (بسته به محل و چگونگی تست شرط حلقه) انواع مختلفی دارد –اما متداولترین شکل آن چنین است:

```
Do While condition
    block of statements to be executed
Loop
```

برای مثال، در حلقهٔ زیر تا زمانی که کاربر کلمهٔ "Done" را وارد نکند، عملیات دریافت ورودی (و نمایش آن در یک جعبه متن) ادامه خواهد یافت:

```
Dim InpName As String
Do While InpName <> "Done"
    InpName = InputBox("Input your name, or type Done to quit.")
    If InpName <> "Done" Then TextBox1.Text = InpName
Loop
```

شرط این حلقه (یعنی InpName <> "Done") به کامپایلر ویژوال بیسیک می‌گوید که «تا زمانی که کاربر کلمهٔ "Done" را وارد نکرده، به کار خود ادامه بده.» همین جا یکی از نکات جالب در مورد حلقه‌های Do را می‌بینید: اگر شرط حلقهٔ Do While در اولین اجرای آن True نباشد، حلقهٔ Do هرگز اجرا نخواهد شد. در مثال فوق، اگر متغیر InpName قبل از شروع حلقه مقدار "Done" داشته باشد، ویژوال بیسیک این حلقه را اجرا نخواهد کرد.

اگر می‌خواهید یک حلقهٔ Do حداقل یک بار اجرا شود، شرط حلقه را به انتهای آن منتقل کنید:

```
Dim InpName As String
Do
    InpName = InputBox("Input your name, or type Done to quit.")
    If InpName <> "Done" Then TextBox1.Text = InpName
Loop While InpName <> "Done"
```

این حلقهٔ Do اساساً شبیه حلقهٔ قبلیست، ولی از آنجائیکه شرط حلقه در انتها تست می‌شود، تابع InputBox حداقل یک بار اجرا خواهد شد. بدین ترتیب اگر کلمهٔ "Done" از قبل در متغیر InpName مانده باشد، امکان این را پیدا می‌کنید، که آنرا عوض کنید.

نکته

تست شرط حلقهٔ Do نسبت به نوع حروف (بزرگی یا کوچکی) حساس است. در مثال بالا، اگر بجای "Done" کلمات "done" یا "DONE" را وارد کنید، برنامه متوقف نخواهد شد. اگر می‌خواهید تست کلمات نسبت به نوع حروف حساس نباشد، می‌توانید از تابع StrComp استفاده کنید. (در فصل ۱۲ دربارهٔ تابع StrComp بیشتر صحبت خواهم کرد.)

امان از حلقه‌های بی‌انتها!

بدلیل خصلت ذاتی حلقه‌های Do ، بسیار مهم است که آنها را طوری طراحی کنید که شرط حلقه حتماً در یک نقطه False شود. حلقه‌ای که شرط آن هرگز False نشود، تا ابد به تکرار خود ادامه می‌دهد، و گاه حتی دیگر به کاربر هم اعتنایی نخواهد کرد. حلقهٔ زیر را در نظر بگیرید:

```
Dim Number As Double
Do
    Number = InputBox("Input a number to square. Type -1 to quit.")
    Number = Number * Number
    TextBox1.Text = Number
Loop While Number >= 0
```

این برنامه مربع اعدادی را که کاربر وارد می‌کند، محاسبه کرده و نمایش می‌دهد. اما اگر آن را اجرا کنید، خواهید دید که هرگز نمی‌توانید با روشی که خود برنامه اعلام می‌کند (وارد کردن عدد 1-) از آن خارج شوید. علت اینست که وقتی نوبت به تست متغیر Number در انتهای حلقه می‌رسد، این عدد هرگز نمی‌تواند 1- باشد، چون قبلاً در خودش ضرب شده، و اکنون دیگر 1 است، نه 1-. به این قبیل حلقه‌ها (حلقه‌هایی که شرط خروج از آنها هرگز محقق نمی‌شود) حلقه‌های بی‌انتها (endless loop) گفته می‌شود. اجتناب از حلقه‌های بی‌انتها یکی از نکات مهم در برنامه‌نویسی است - و خوشبختانه این کار چندان دشوار نیست، چون آنها بسرعت خود را نشان می‌دهند. (در مثال بالا، کافیست از متغیر دیگری برای ذخیره کردن مجذور عدد ورودی استفاده کنیم، تا مقدار متغیر Number در حین محاسبه عوض نشود.)

بسیار مهم

مطمئن شوید که شرط خروج از حلقه بنحوی محقق می‌شود.

در تمرین این قسمت با استفاده از حلقهٔ Do یک برنامهٔ تبدیل درجه حرارت فارنهایت به سلسیوس خواهیم نوشت. این برنامه ورودی خود را از یک تابع InputBox() گرفته، و بعد از تبدیل فارنهایت به سلسیوس، نتیجه را با تابع MsgBox() نمایش می‌دهد.

تبدیل درجه حرارت با استفاده از حلقهٔ Do...Loop

۱ از منوی File|New آیتم Project را انتخاب کنید، تا پنجرهٔ «پروژهٔ جدید» ظاهر شود.

۲ یک پروژهٔ Visual Basic Windows Application بنام **My Celsius Conversion** در پوشهٔ c:\vbnetsbs\chap07 ایجاد کنید.

از آنجائیکه تمام کُد این برنامه را در روال Form1_Load خواهیم نوشت، بمحض آنکه برنامه را اجرا کنید، ویژوال بیسیک تابع InputBox() را اجرا کرده، و یک درجه حرارت فارنهایت از شما طلب می‌کند؛ و بعد از تبدیل آن به درجهٔ سلسیوس (سانتیگراد)، خروجی را با تابع MsgBox() نمایش خواهد داد.

۳ روی فرم برنامه دو-کلیک کنید، تا روال رویداد Form1_Load در ادیتور کُد باز شود.

٤ دستورات زیر را در این روال بنویسید:

```
Dim FTemp, Celsius As Single
Dim strFTemp As String
Dim Prompt As String = "Enter a Fahrenheit temperature."
Do
    strFTemp = InputBox(Prompt, "Fahrenheit to Celsius")
    If strFTemp <> "" Then
        FTemp = CSng(strFTemp)
        Celsius = Int((FTemp + 40) * 5 / 9 - 40)
        MsgBox(Celsius, , "Temperature in Celsius")
    End If
Loop While strFTemp <> ""
End
```

نکته

دستور End را (در پایان روال Form1_Load) فراموش نکنید!

در ابتدای این کُد دو متغیر از نوع اعشاری (برای درجه حرارتهای فارنهایت و سلسیوس) و دو متغیر رشته‌ای (یکی برای ذخیره کردن درجه حرارت فارنهایت ورودی، و دیگری برای ذخیره کردن پیام تابع InputBox) تعریف کرده‌ایم. بلافاصله بعد از آن، حلقهٔ Do شروع می‌شود. در این حلقه ابتدا تابع InputBox یک درجه حرارت فارنهایت از کاربر گرفته، و آنرا در متغیر strFTemp ذخیره می‌کند. تابع InputBox همیشه مقداری از نوع String (رشته) برمی‌گرداند، حتی اگر کاربر در آن عدد وارد کرده باشد (و در واقع، اعداد را هم به رشتهٔ متنی تبدیل می‌کند). اما از آنجائیکه باید از این عدد در محاسبات خود استفاده کنیم، آنرا با تابع CSng دوباره به عدد تبدیل کرده‌ایم (تابع CSng یکی از چندین تابع تبدیل انواع داده در ویژوال بیسیک است). عدد بدست آمده در متغیری بنام FTemp (که از نوع اعشاری است) ذخیره می‌شود.

حلقهٔ Do در دو وضعیت به کار خود خاتمه می‌دهد: کلیک کردن دکمهٔ Cancel ، و کلیک کردن OK بدون وارد کردن عدد (این وضعیت دوم در شرط While تست می‌شود). قلب این روتین (تبدیل فارنهایت به سلسیوس) دستور زیر است:

```
Celsius = Int((FTemp + 40) * 5 / 9 - 40)
```

تنها نکتهٔ این دستور در تابع Int نهفته است: این تابع قسمت اعشاری عدد حاصله را دور می‌اندازد، و فقط قسمت صحیح آن (قبل از ممیز) را برمی‌گرداند. بدین‌ترتیب، دیگر اعداد عجیب و غریبی مانند 21.11111 (که معادل 70 °F است) نخواهیم دید.

۵ با کلیک کردن دکمهٔ Save All ، پروژه را ذخیره کنید. (متن کامل این پروژه را می‌توانید در پوشهٔ

کنترل تایمر

برای اجرای دستورات در فواصل زمانی معین، می‌توانید از کنترل تایمر (Timer) استفاده کنید. کنترل تایمر یک کرونومتر دیجیتالی است، که ساعت سیستم را در اختیار شما می‌گذارد. از تایمر (که یک کنترل نامریی است) می‌توان برای شمارش معکوس، ایجاد تأخیر در برنامه، و یا تکرار دستورات در فواصل زمانی مشخص استفاده کرد.

کنترل تایمر فقط یک رویداد بنام Tick دارد، که با هر تیک ساعت فراخوانی می‌شود. فاصلهٔ زمانی تیک‌های تایمر توسط خاصیت Interval (که بر حسب میلی‌ثانیه است) تعیین می‌شود. البته قبل از اینکه یک تایمر بتواند کار خود را انجام دهد، خاصیت Enabled آن باید به True ست شود. بمحض اینکه تایمر فعال شد (Timer1_Enabled = True) ، روال رویداد Timer1_Tick را در فواصل زمانی معین اجرا می‌کند، تا زمانیکه دوباره خاصیت Enabled به False ست شود (و یا برنامه بسته شود).

ایجاد یک ساعت دیجیتالی با استفاده از کنترل تایمر

یکی از کاربردهای عملی کنترل تایمر ایجاد ساعت دیجیتالی است. در تمرین این قسمت، برنامهٔ یک ساعت دیجیتالی ساده را می‌نویسیم، که گذشت زمان را با دقت ثانیه نمایش می‌دهد. برای اینکه ساعت ما گذشت ثانیه‌ها را نشان دهد، خاصیت Interval را به 1000 (میلی‌ثانیه، که معادل 1 ثانیه است) ست می‌کنیم؛ بدین‌ترتیب، روال Timer1_Tick در هر ثانیه یک بار اجرا خواهد شد. البته، به علت خصلت ذاتی سیستم عامل ویندوز (که قادر است چند برنامه را بصورت همزمان اجرا کند)، ممکنست رأس هر ثانیه برنامهٔ ما فرصت اجرا پیدا نکند، ولی این عقب‌افتادگی در دفعات بعدی جبران خواهد شد.

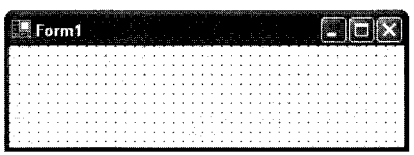
برنامهٔ ساعت دیجیتالی

۱ از منوی File|New آیتم Project را انتخاب کنید، تا پنجرهٔ «پروژهٔ جدید» ظاهر شود.

۲ یک پروژهٔ Visual Basic Windows Application بنام **My Digital Clock** در پوشهٔ c:\vbnetsbs\chap07 ایجاد کنید.

۳ ساعت ما به فضای چندانی نیاز ندارد، پس فرم را تا حد یک مستطیل با ابعاد تقریبی ۲×۵ سانتیمتر، کوچک کنید.

٤ در برگهٔ Windows Forms جعبه ابزار ویژوال بیسیک، روی کنترل Timer دو-کلیک کنید، تا یک شیء تایمر به سینی اجزاء (که در زیر فرم قرار دارد) اضافه شود (از آنجائیکه تایمر یک کنترل نامریی است، مستقیماً روی فرم برنامه قرار نمی‌گیرد):



۵ کنترل Label را از جعبه ابزار انتخاب کرده، و یک برچسب بزرگ روی فرم برنامه رسم کنید (بگونه‌ایکه تقریباً تمام آن را بپوشاند)؛ از این برچسب برای نمایش ساعت استفاده خواهیم کرد.

۶ پنجرهٔ خواص را باز کرده، و خواص اشیاء برنامه را با توجه به جدول زیر ست کنید.

شیء	خاصیت	مقدار
Form1	Text	"Digital Clock"
Label1	Font	Times New Roman, Bold, 24-point
	Text	(خالی)
	TextAlign	MiddleCenter
Timer1	Enabled	True
	Interval	1000

نکته

اگر می‌خواهید ساعت قشنگ‌تری داشته باشید، می‌توانید خاصیت BackgroundImage فرم را به یک تصویر زیبا و مناسب ست کنید.

۷ در سینی اجزاء، روی شیء Timer1 دو-کلیک کنید، تا روال رویداد Timer1_Tick در ادیتور کُد باز شود. (اگر قبلاً با ویژوال بیسیک ۶ کار کرده باشید، این روال را با نام Timer1_Timer می‌شناسید.)

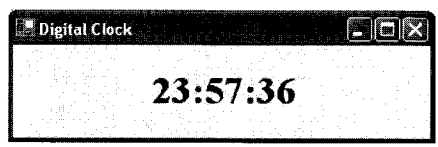
۸ دستور زیر را در این روال وارد کنید:

```
Label1.Text = TimeString
```

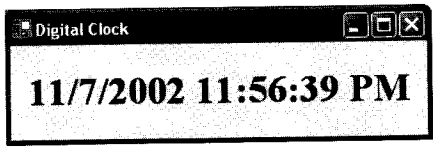
این دستور، وقت فعلی را از ساعت سیستم گرفته، و در برچسب Label1 می‌نویسد. (اگر می‌خواهید علاوه بر وقت، تاریخ را هم نمایش دهید، می‌توانید از خاصیت System.DateTime.Now استفاده کنید.) می‌بینید که به همین سادگی (و فقط با یک دستور) یک ساعت دیجیتال نوشتیم، چون قسمت اصلی کار را کنترل تایمر انجام می‌دهد.

۹ با کلیک کردن دکمهٔ Save All ، پروژه را ذخیره کنید. (متن کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap07\digital clock بیابید.)

۱۰ با کلیک کردن دکمهٔ Start ، برنامه را کامپایل و اجرا کنید. شکل زیر اجرای برنامهٔ ساعت دیجیتالی را نشان می‌دهد:



اگر بجای خاصیت TimeString از System.DateTime.Now استفاده کنیم، ساعت ما به شکل زیر در می‌آید:



۱۱ برای مدتی ساعت دیجیتالی را تحت نظر بگیرید؛ همانطور که می‌بینید، ویژوال بیسیک ثانیه به ثانیه ساعت را به روز در می‌آورد.

۱۲ دکمهٔ Close را کلیک کنید، و از برنامه خارج شوید.

برنامهٔ ساعت دیجیتالی آنقدر ساده و کارآمد است، که می‌توانید پس از کمی دستکاری (مثلاً، رنگی کردن فونت یا زمینهٔ آن) آنرا کامپایل کرده، و از آن در کامپیوتر خود استفاده کنید.

یک گام فراتر: استفاده از تایمر برای ایجاد محدودیت زمانی

یکی دیگر از کاربردهای جالب شیء تایمر تعیین محدودهٔ زمانی برای انجام یک عمل است. این کار در واقع نوعی شمارش معکوس است. برای این منظور، بعد از فراهم آوردن مقدمات عمل موردنظر و ست کردن مقدار تأخیر موردنظر، خاصیت Enabled کنترل Timer را True می‌کنیم، تا گذشت زمان را اندازه گیری کند.

در تمرین زیر، از همین تکنیک برای محدود کردن زمانی که کاربر می‌تواند کلمهٔ رمز برنامه را وارد کند، استفاده خواهیم کرد: اگر کاربر نتواند کلمهٔ رمز را (که کلمهٔ "secret" است) در مدت ۱۵ ثانیه وارد کند، برنامه بسته خواهد شد. (معمولاً برنامه‌ای مانند این، بخشی از یک برنامهٔ بزرگتر است.) از این تکنیک برای نمایش آرم برنامه برای مدت زمانی مشخص در ابتدای برنامه، و یا ذخیره کردن متناوب فایلهای برنامه، نیز استفاده می‌شود.

ایجاد محدودیت زمانی برای وارد کردن کلمهٔ رمز

۱ از منوی File|New آیتم Project را انتخاب کنید، تا پنجرهٔ «پروژهٔ جدید» ظاهر شود.

۲ یک پروژهٔ Visual Basic Windows Application بنام **My Timed Password** در پوشهٔ c:\vbnetsbs\chap07 ایجاد کنید.

۳ این برنامه هم به فضای چندانی نیاز ندارد، پس فرم را تا حد پنجرهٔ تابع InputBox() (مستطیلی با ابعاد تقریبی ۶×۳ سانتیمتر) کوچک کنید.

٤ از برگهٔ Windows Forms جعبه ابزار ویژوال بیسیک، کنترل TextBox را انتخاب کرده، و یک جعبه متن (با طول نسبتاً زیاد) در وسط فرم برنامه رسم کنید.

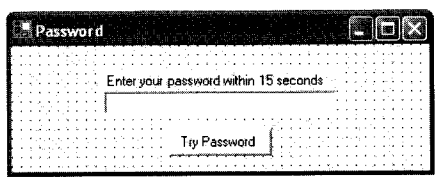
۵ کنترل Label را از جعبه ابزار انتخاب کرده، و یک برچسب (آن هم با طول نسبتاً زیاد) بالای جعبه متن رسم کنید.

۶ کنترل Button را از جعبه ابزار انتخاب کرده، و یک دکمه زیر جعبه متن رسم کنید.

۷ در برگهٔ Windows Forms جعبه ابزار ویژوال بیسیک، روی کنترل Timer دو-کلیک کنید، تا یک شیء تایمر به سینی اجزاء (که در زیر فرم قرار دارد) اضافه شود. ۸ خواص اشیاء برنامه را با توجه به جدول زیر ست کنید.

شیء	خاصیت	مقدار
Button1	Text	"Try Password"
Form1	Text	"Password"
Label1	Text	"Enter your password within 15 seconds"
TextBox1	PasswordChar	"*"
	Text	(خالی)
Timer1	Enabled	True
	Interval	15000

ست کردن خاصیت PasswordChar جعبه متن به "*" باعث می‌شود تا بجای هر کاراکتری که کاربر وارد می‌کند، کاراکتر * دیده شود. خاصیت Interval تایمر را به 15000 میلی‌ثانیه (معادل 15 ثانیه)، و خاصیت Enabled آنرا به True ست کرده‌ایم، تا با شروع برنامه بلافاصله شمارش را آغاز کند (البته می‌توانید این خاصیت را False کرده، و فقط وقتی لازم است شمارش شروع شود، آن را True کنید). شکل نهایی فرم برنامه را در زیر می‌بینید:



۹ در سینی اجزاء، روی شیء Timer1 دو-کلیک کنید، تا روال رویداد Timer1_Tick در ادیتور کُد باز شود. دستورات زیر را در این روال وارد کنید:

```
MsgBox("Sorry, your time is up.")
End
```

این دستورات ۱۵ ثانیه پس از شروع برنامه اجرا شده، و بعد از اعلام سپری شدن زمان لازم برای وارد کردن کلمهٔ رمز، به برنامه پایان می‌دهند.

۱۰ به فرم برنامه برگردید، و بعد از دو-کلیک کردن روی دکمهٔ Button1 ، دستورات زیر را در روال رویداد Button1_Click بنویسید:

```
If TextBox1.Text = "secret" Then
    Timer1.Enabled = False
    MsgBox("Welcome to the system!")
    End
Else
    MsgBox("Sorry, friend, I don't know you.")
End If
```

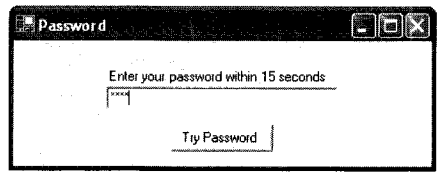
این برنامه رمز وارد شده در جعبه متن را با کلمهٔ "secret" مقایسه می‌کند. اگر رمز صحیح باشد، تایمر غیرفعال شده، و پیام خوشامد نمایش داده می‌شود (برنامهٔ ما در اینجا پایان می‌یابد، ولی در برنامه‌های واقعی این تازه اول کار است). اما اگر رمز صحیح نباشد، این موضوع به کاربر اطلاع داده شده، و فرصت دیگری به وی داده می‌شود – البته باز هم در همان محدودهٔ ۱۵ ثانیه‌ای.

۱۱ با کلیک کردن دکمهٔ Save All ، پروژه را ذخیره کنید. (متن کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap07\timed password بیابید.)

تست برنامهٔ Timed Password

۱ با کلیک کردن دکمهٔ Start ، برنامه را کامپایل و اجرا کنید. با اجرای برنامه، تایمر ۱۵ ثانیه‌ای شروع به شمارش می‌کند.

۲ کلمهٔ **open** را در جعبه متن وارد کنید (البته خود شما هم نمی‌توانید این کلمه را ببینید، چون این جعبه متن بجای هر چیزی فقط * نشان می‌دهد):



۳ دکمهٔ Try Password را کلیک کنید؛ پیام زیر ظاهر شده، و می‌گوید که رمز اشتباه است:

٤ OK را کلیک کرده، و کمی صبر کنید تا پیام «پایان مهلت» ظاهر شود:

۵ OK را کلیک کنید، تا برنامه بسته شود.

۶ یک بار دیگر برنامه را اجرا کرده، و این بار بعد از وارد کردن کلمهٔ **secret** ، دکمهٔ Try Password را کلیک کنید.

۷ پیام زیر ظاهر می‌شود:

۵ OK را کلیک کنید، تا برنامه بسته شود.

این هم یکی دیگر از کاربردهای جالب شیء تایمر.

مرجع سریع فصل ۷

برای ...	انجام دهید

اجرای یک یا چند دستور بدفعات معین	این دستورات را در یک حلقهٔ For...Next قرار دهید:
	`For i = 1 To 10`
	`  MsgBox("Press OK already!")`
	`Next i`

اجرای یک یا چند دستور با گامهای معین	در حلقهٔ For...Next از کلمهٔ کلیدی Step استفاده کنید:
	`For i = 2 To 8 Step 2`
	`  TextBox1.Text = TextBox1.Text & i`
	`Next i`

اجتناب از حلقهٔ Do بی‌انتها	کاری کنید که شرط حلقه در حالت خاصی False شود.

خروج زود هنگام از یک حلقهٔ For...Next	از دستور Exit For استفاده کنید:
	`For i = 1 To 10`
	`  InpName = InputBox("Name?")`
	`  If InpName = "Done" Then Exit For`
	`  TextBox1.Text = InpName`
	`Next i`

اجرای یک یا چند دستور
تا وقتی که یک وضعیت خاص برقرار است
	این دستورات را در یک حلقهٔ Do...Loop قرار دهید:
	`Do While Query <> "Yes"`
	`  Query = InputBox("Are you enemy?")`
	`  If Query = "Yes" Then MsgBox("Hi!")`
	`Loop`

اجرای یک یا چند دستور
تا برقراری یک وضعیت خاص
	در حلقهٔ Do...Loop از کلمهٔ کلیدی Until استفاده کنید:
	`Do`
	`  GiveIn = InputBox("Say 'Uncle'")`
	`Loop Until GiveIn = Uncle`

اجرای یک یا چند دستور برای مدتی معین	از کنترل تایمر (Timer) استفاده کنید.

فصل

۸

دیباگ کردن برنامه‌های ویژوال بیسیک .NET

در این فصل یاد می گیرید چگونه :

- ✔ انواع خطاها را در برنامه تشخیص دهید.
- ✔ از ابزارهای دیباگ ویژوال استودیو .NET برای ست کردن نقاط وقفه، و تصحیح خطاهای برنامه استفاده کنید.
- ✔ برای بررسی مقدار متغیرهای برنامهٔ در حال اجرا، از پنجرهٔ Watch کمک بگیرید.
- ✔ از پنجرهٔ Command برای تغییر دادن مقدار متغیرهای برنامه، و اجرای فرمانهای ویژوال استودیو استفاده کنید.

در برنامه‌نویسی هیچ کاری ساده‌تر از اشتباه کردن نیست، و احتمالاً شما هم تا همین جا چند بار مرتکب اشتباه شده‌اید. بر خلاف محاورهٔ انسانها، که اشتباهات جزئی در طرز ادای کلمات و جمله‌بندی‌ها خللی در ارتباط بوجود نمی‌آورد، رابطهٔ یک برنامهٔ نوشته شده با کامپایلری مانند ویژوال بیسیک متکی به یک سری اصول دقیق و غیر قابل تخطی است. عدم رعایت (سهوی یا عمدی) این اصول و قواعد موجب می‌شود تا کامپایلر نتواند کار خود را بدرستی انجام دهد، و از کار باز بماند (به این قبیل خطاهای برنامه اصطلاحاً باگ – bug – گفته می‌شود). در این فصل با انواع خطاها آشنا شده، و یاد می‌گیرید چگونه با استفاده از ابزارهای دیباگ (debugging tools) ویژوال استودیو .NET آنها را کشف و تصحیح کنید. مهارتهایی که در این فصل کسب می‌کنید، در تمام طول کتاب (و در آینده) همراه و یاور شما خواهند بود.

تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- ویژوال بیسیک .NET علاوه بر ابزارهای دیباگ ویژوال بیسیک ۶ (فرمانهایی از قبیل Start، Break، End، Next، Step Into، و Step Over)، تعدادی ابزار دیباگ جدید دارد، که از جملهٔ آنها می‌توان به ابزاری برای عیب‌یابی برنامه‌های چندزبانه اشاره کرد.
- در محیط برنامه‌نویسی ویژوال استودیو .NET چندین پنجرهٔ دیباگ جدید اضافه شده، که برخی از آنها عبارتند از: Autos، Command، Call Stack، Threads، Memory، Disassembly، و Registers. (البته اینها ابزارهایی هستند که شاید در برنامه‌های ساده نیاز چندانی به آنها پیدا نشود، ولی برای دیباگ کردن برنامه‌های بزرگ و پیچیده بسیار مفیدند.)

یافتن خطاهای برنامه و تصحیح آنها

اغلب خطاها و اشتباهاتی که تا بدین جا در برنامه‌های خود با آن روبرو شده‌اید، از نوع اشتباهات تایپی و گرامری بوده، که با نگاهی دقیق‌تر به کُد برنامه بآسانی می‌توان آنها را پیدا و برطرف کرد. اما اگر به موردی برخورد کنید که حتی با بررسی مفصل برنامه قادر به یافتن منشأ آن نباشید، چطور؟ محیط برنامه‌نویسی ویژوال استودیو .NET دارای ابزارهای متعددی برای ردیابی و تصحیح خطاهای برنامه است. البته این ابزارها باعث نمی‌شوند که شما دیگر در برنامه‌نویسی اشتباه نکنید، اما کمک خوبی برای یافتن و از بین بردن این اشتباهات بحساب می‌آیند.

انواع خطاها

در یک برنامهٔ ویژوال بیسیک (و اصولاً هر برنامهٔ دیگری) سه نوع خطا می‌تواند اتفاق بیفتد: خطاهای دستوری (syntax errors)، خطاهای زمان اجرا (runtime errors)، و خطاهای منطق برنامه (logic errors).

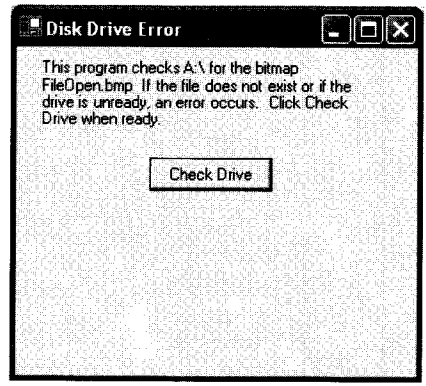
- خطای دستوری (یا کامپایلری) از نقض قواعد برنامه‌نویسی ویژوال بیسیک (مانند، املای غلط یک دستور یا خاصیت) ناشی می‌شود. ویژوال بیسیک ابزار بسیار قدرتمندی بنام IntelliSense دارد، که جلوی بسیاری از این قبیل خطاها را می‌گیرد – و اساساً تا برنامه‌ای فاقد این گونه خطاها نباشد، کامپایلر آنرا اجرا نخواهد کرد.
- خطای زمان اجرا خطائیست که در حین اجرای برنامه روی می‌دهد، و باعث توقف اجرای آن می‌شود. کامپایلر قادر به تشخیص این قبیل خطاها نیست، و آنها فقط هنگام اجرای برنامه خود را نشان می‌دهند. مثلاً، اگر نام یک فایل را اشتباه وارد کرده باشید، کامپایلر فقط هنگام اجرای متدی مانند System.Drawing.Image.FromFile متوجه آن خواهد شد؛ و یا اگر بخواهید روی دیسکتی که یادتان رفته در درایو فلاپی قرار دهید، چیزی بنویسید، مسلماً با خطا مواجه خواهید شد – و اصولاً کامپایلر چگونه می‌تواند از قبل بداند که دیسکت را در درایو نگذاشته‌اید؟
- خطای منطق برنامه هم یک خطای کاملاً انسانی است – اگر در جایی که باید ضرب کنید، تقسیم کرده

باشید، برنامه خروجی اشتباه تولید می‌کند، بدون آنکه گناهی متوجه آن باشد. خطاهای منطق برنامه یکی از دردسرسازترین انواع خطاها هستند، و تشخیص آنها نیز بسیار مشکل است.

تشخیص خطاهای نوع اول و دوم عمدتاً بر عهدهٔ برنامه‌نویس است؛ اغلب ابزارهای دیباگ ویژوال بیسیک برای ردیابی خطاهای نوع سوم طراحی شده‌اند.

تشخیص خطاهای دستوری با کمک IntelliSense ویژوال بیسیک بسیار ساده است، و فقط کافیست کمی دقت کنید تا بتوانید علت خطا را پیدا کرده و آنرا برطرف کنید. در ادیتور کُد، کامپایلر ویژوال بیسیک زیر دستوراتی که اشتباه هستند، یک خط زیگزاگ آبی رنگ می‌کشد - و کافیست ماوس را روی این دستور برده و چند لحظه صبر کنید (کلیک نکنید!)، تا کامپایلر علت خطا را در یک کادر زرد رنگ توضیح دهد. به شکل زیر نگاه کنید:

کامپایلر ویژوال بیسیک یک خطای دستوری تشخیص داده



تشخیص منشأ خطاهای زمان اجرا کمی دشوارتر است، اما به کمک روتین‌های ساخت‌یافتهٔ مقابله با خطا جلوگیری از این نوع خطاها نیز براحتی امکانپذیر است. در فصل آینده دربارهٔ روتین‌های مقابله با خطا بیشتر صحبت خواهم کرد.

تشخیص خطاهای منطق برنامه

ردیابی، تشخیص و برطرف کردن خطاهای منطق برنامه از دو نوع دیگر بسیار دشوارتر است. منشأ این نوع خطا در ذهن برنامه‌نویس است، نه در ابزارهای ویژوال بیسیک. بلوک If...Then زیر را در نظر بگیرید:

```
If Age > 13 And Age < 20 Then
    TextBox1.Text = "You're a teenager"
Else
    TextBox1.Text = "You're not a teenager"
End If
```

همانطور که می‌دانید، به شخصی که بین ۱۳ تا ۱۹ سال سن داشته باشد، "teenager" می‌گویند - اما در برنامهٔ فوق فردی با سن ۱۳ سال، به اشتباه پیام "You're not a teenager" را دریافت می‌کند. این چنین اشتباهی از

ذهن برنامه‌نویس ناشی شده، و ویژوال بیسیک هیچ راهی برای تشخیص آن ندارد. در مثال فوق، اگر عبارت Age > 13 را به Age >= 13 تبدیل کنیم، برنامه بدرستی کار خواهد کرد:

```
If Age >= 13 And Age < 20 Then
```

این قبیل کُدها که در اغلب شرایط (ولی نه همیشه) درست عمل می‌کنند، مشکلترین انواع خطا از نظر تشخیص هستند - و متأسفانه متداولترین نوع خطاها نیز همانها هستند.

دیباگ کردن: حالت وقفه

یکی از راههای تشخیص خطاهای منطق برنامه، اجرای برنامه بصورت خط به خط (و بررسی نحوهٔ تغییر متغیرها و خواص اشیاء) است. برای این منظور، باید بعد از اجرای برنامه، وارد حالت وقفه (break mode) شویم. حالت وقفه به ما اجازه می‌دهد تا اجرای برنامه (و طرز عمل کامپایلر) را بدقت تحت نظر بگیریم. این کار درست مثل اینست که پشت سر خلبان هواپیما یا ناخدای کشتی بایستیم، و کار او را زیر نظر بگیریم (و حتی این اجازه را داشته باشیم که به جای آنها هواپیما یا کشتی را هدایت کنیم!).

وقتی برنامه‌ای را دیباگ می‌کنیم، باید ابزارهای دیباگ ویژوال بیسیک را در اختیار داشته باشیم. این ابزارها در میله ابزار دیباگ (Debug toolbar) قرار دارند - برای دیدن آنها باید این میله ابزار را از منوی View|Toolbars انتخاب کنید.

در تمرین زیر با استفاده از حالت وقفه و ست کردن یک نقطهٔ وقفه (breakpoint) ، بلوک If...Then را که در بالا به آن اشاره کردیم، عیب‌یابی خواهیم کرد. در این تمرین از دکمهٔ Step Into میله ابزار دیباگ برای اجرای برنامه بصورت خط به خط، و پنجرهٔ Autos برای نمایش مقدار متغیرها و خواص اشیاء برنامه، استفاده خواهیم کرد. راهبردی که در این تمرین خواهید دید، یکی از بهترین روش‌ها برای عیب‌یابی برنامه‌ها محسوب می‌شود - و توصیه می‌کنم آنرا با دقت دنبال کنید.

دیباگ کردن برنامهٔ Debug Test

۱ ویژوال استودیو .NET را باز کنید.

۲ از منوی File|Open آیتم Project را انتخاب کنید، تا پنجرهٔ «باز کردن پروژه» ظاهر شود.

۳ پروژهٔ Debug Test را (که در پوشهٔ x:\vbnetsbs\chap08\debug test قرار دارد) باز کنید.

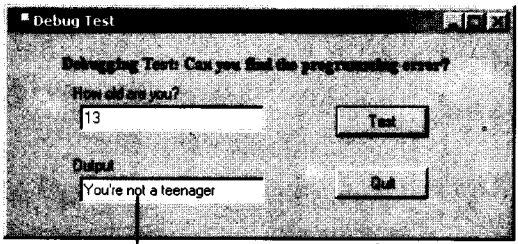
٤ در اینجا وقتی کاربر سن خود را در جعبه متن ?How old are you وارد کرده، و دکمهٔ Test را کلیک کند، برنامه به وی اعلام می‌کند که آیا Teenager هست یا خیر. متأسفانه بعد از اجرای آن متوجه شده‌ایم که برنامه افراد ۱۳ ساله را بدرستی تشخیص نمی‌دهد، و با اینکه آنها هم Teenager هستند، ولی به غلط اعلام می‌کند که "You're not a teenager"! برای یافتن علت این خطا، از نقاط وقفه کمک می‌گیریم.

۵ منوی View|Toolbars را باز کرده و با انتخاب آیتم Debug ، میله ابزار دیباگ را باز کنید.

۶ در میله ابزار دیباگ، دکمهٔ Start را کلیک کنید، تا برنامهٔ Debug Test کامپایل و اجرا کند.

۷ بعد از وارد کردن عدد **14** در جعبه متن ?How old are you ، دکمهٔ Test را کلیک کنید؛ پاسخ برنامه "You're a teenager" است - خوب، تا اینجا که درست کار می‌کند!

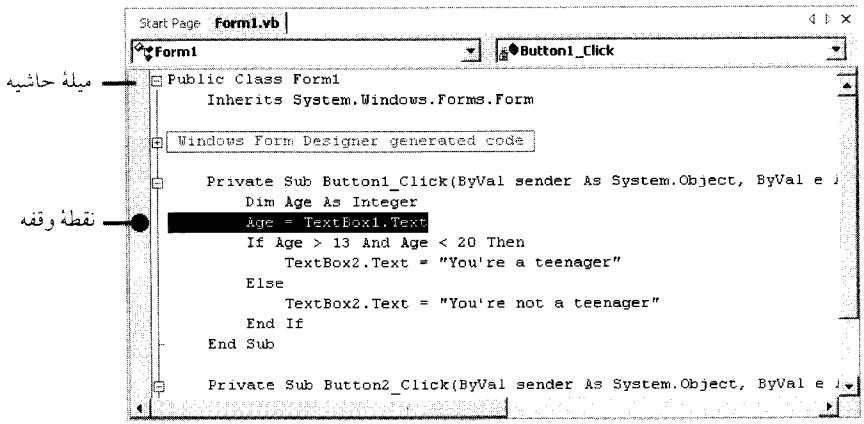
۸ در جعبه متن ?How old are you عدد **13** را وارد کرده، و دکمهٔ Test را کلیک کنید؛ برنامه پاسخ می‌دهد: "You're not a teenager" !(و این همان اشتباهیست که باید علت آنرا بفهمیم.)



این یک باگ است!

۹ بعد از بستن برنامه (با دکمهٔ Quit)، ادیتور کُد را باز کنید.

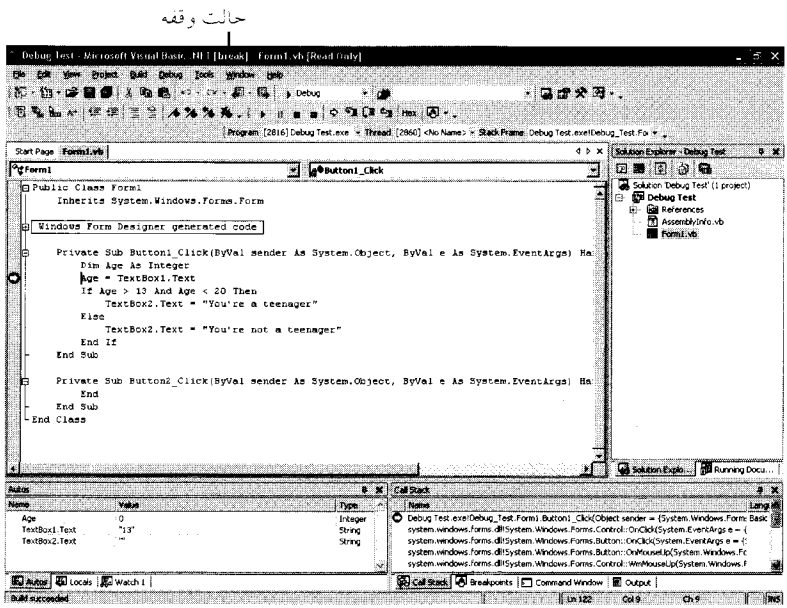
۱۰ ماوس را روی حاشیهٔ خاکستری سمت چپ ادیتور کُد (که به آن میلهٔ حاشیه - Margin Indicator Bar - گفته می‌شود)، درست مقابل دستور Age = TextBox1.Text (در روال رویداد Button1_Click) برده و کلیک کنید، تا یک دایرهٔ قرمز (که همان نقطهٔ وقفه است) در آنجا ظاهر شود:



نقطهٔ وقفه جائیست که اجرای برنامه موقتاً متوقف شده، و به محیط ویژوال استودیو برمی‌گردیم، تا بتوانیم از ابزارهای آن استفاده کنیم.

۱۱ مجدداً دکمهٔ Start را (در میله ابزار دیباگ) کلیک کنید؛ برنامه مانند قبل، و بدون هیچ تفاوت ظاهری اجرا می‌شود.

۱۲ عدد **13** را در جعبه متن ?How old are you وارد کرده، و دکمهٔ Test را کلیک کنید. با این کار ویژوال بیسیک اجرای برنامه را موقتاً متوقف کرده، به ادیتور کُد برمی‌گردد، و دستور Age = TextBox1.Text (همان دستوری که نقطهٔ وقفه در آنجا ست شده) را به رنگ زرد و با یک علامت پیکان در میلهٔ حاشیه نشان می‌دهد (شکل زیر را ببینید):



اگر دقت کنید، در میلهٔ عنوان ویژوال استودیو کلمهٔ "[break]" بچشم می‌خورد؛ این کلمه نشان می‌دهد که ویژوال بیسیک اکنون در حالت وقفه بسر می‌برد.

نکته

برای ورود به حالت وقفه می‌توانید در نقطه‌ای که می‌خواهید برنامه دچار وقفه شود، از دستور Stop استفاده کنید. این یک روش قدیمیست، اما ویژوال بیسیک .NET همچنان از آن پشتیبانی می‌کند.

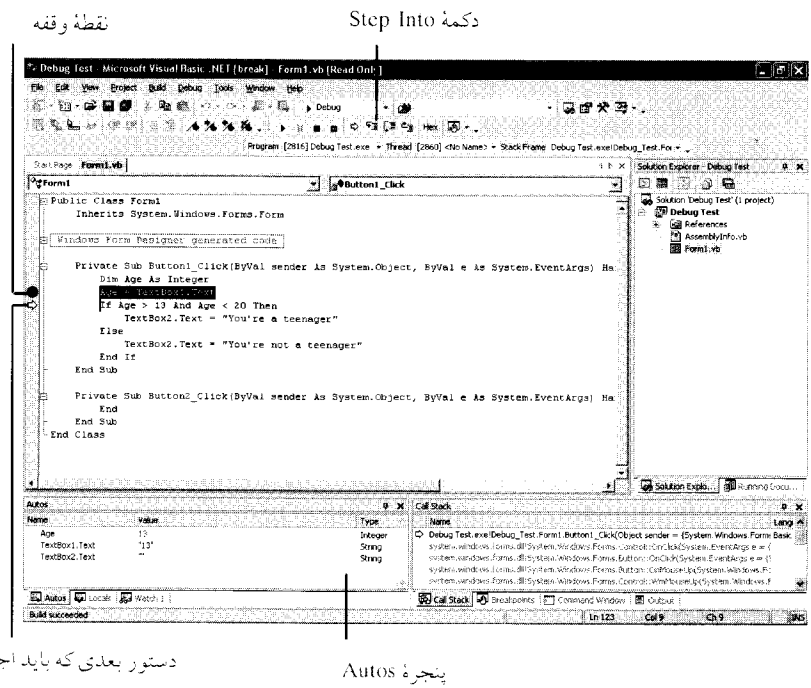
۱۳ ماوس را روی متغیر Age ببرید، و چند لحظه صبر کنید (کلیک نکنید!) - ویژوال استودیو در یک کادر زردرنگ اعلام می‌کند: "Age = 0" . وقتی در حالت وقفه هستید، می‌توانید مقدار فعلی تمام متغیرها و خواص اشیاء برنامه را با بردن کرسر ماوس روی آنها بخوانید. متغیر Age در حال حاضر مقدار 0 دارد، چون دستور Age = TextBox1.Text هنوز اجرا نشده است (در واقع، این دستوریست که بلافاصله پس از خروج از حالت وقفه اجرا خواهد شد).

نکته

در حالت وقفه فقط می‌توانید مقدار متغیرها و خواص اشیاء را بخوانید، و نمی‌توانید کُد برنامه را دستکاری کنید - برنامه هنوز در حال اجراست، و ویژوال استودیو اجازهٔ تغییر دادن کُد آنرا نمی‌دهد.

۱۴ در میله ابزار دیباگ، روی دکمهٔ Step Into کلیک کنید، تا دستور بعدی (دستوری که با رنگ زرد مشخص شده) اجرا شود. اگر یک بار دیگر ماوس را روی متغیر Age ببرید، خواهید دید که مقدار آن 13 شده است.

۱۵ منوی Debug|Windows را باز کرده، و آیتم Autos را انتخاب کنید، تا پنجرهٔ Autos باز شود. پنجرهٔ Autos یکی از پنجره‌های دیباگ ویژوال استودیو است، که بطور خودکار مقدار تمام متغیرها و خواص اشیاء برنامه را نشان می‌دهد. همانطور که در شکل زیر می‌بینید، مقدار متغیر Age در این لحظه 13 است، خاصیت TextBox1.Text مقدار "13" دارد، و خاصیت TextBox2.Text هم "" (خالی) است.



۱۶ دو بار دیگر دکمهٔ Step Into را کلیک کنید.

دستور If هم اجرا شده، و چون شرط آن False است، کنترل برنامه به قسمت Else می‌رود. باگ برنامهٔ ما همین جاست: ۱۳ ساله‌ها هم Teenager هستند، و برنامه نباید به قسمت Else برود.

۱۷ کرسر ماوس را روی عبارت **Age > 13** ببرید، و چند لحظه صبر کنید؛ کادر زردرنگ ظاهر شده، و نشان می‌دهد که: "Age > 13 = False" .

۱۸ کرسر ماوس را روی عبارت **Age < 20** ببرید، و چند لحظه صبر کنید؛ کادر زردرنگ باز هم ظاهر شده، و نشان می‌دهد که: "Age < 20 = True" .

چه نتیجه‌ای از این دو تست می‌توان گرفت؟ بله، قسمت دوم شرط درست عمل کرده، ولی قسمت اول آن باعث خراب شدن کار شده است. قسمت اول هم باید مقدار True برگرداند، چون ۱۳ ساله‌ها

هم Teenager اند. وقتی یکی از دو قسمت عبارت ترکیبی And مقدار False بگیرد، کل عبارت را False می‌کند، و برنامه به قسمت Else می‌رود. متوجه منشأ مشکل شدید؟ بله، عملگر < باید =< باشد. پس، اجازه دهید آنرا تصحیح کنیم.

۱۹ در میله ابزار دیباگ، روی دکمهٔ Stop کلیک کنید، تا دیباگ متوقف شود.

۲۰ در ادیتور کُد، دستور If Age > 13 And Age < 20 Then را بصورت زیر تصحیح کنید:

```
If Age >= 13 And Age < 20 Then
```

۲۱ برنامه را دوباره اجرا کرده، و این بار اعداد ۱۲، ۱۳، ۱۹ و ۲۰ (که مقادیر مرزی برنامه محسوب می‌شوند) را تست کنید.

برای درک بهتر عملکرد برنامه، باز هم می‌توانید وارد حالت وقفه شده، و آنرا بصورت خط به خط اجرا کرده، و مقدار متغیرها و عبارتها را در هر مرحله تست کنید.

۲۲ وقتی مطمئن شدید که برنامه تحت هر شرایطی درست عمل می‌کند، با کلیک کردن دکمهٔ Stop (در میله ابزار دیباگ) آنرا ببندید.

احسنت! پیدا کردن باگ منطق یک برنامه و تصحیح آن، کار کوچکی نیست!

ردگیری مقدار متغیرها با استفاده از پنجرهٔ Watch

پنجرهٔ Autos ابزاری مفید برای ردگیری مقدار متغیرها و خواص اشیاء برنامه است، ولی مقدار این متغیرها فقط مربوط به دستور فعلی (دستوری که کامپایلر روی آن متوقف شده) و دستور قبلی (آخرین دستوری که کامپایلر اجرا کرده) است. وقتی کامپایلر به دستوری می‌رسد که هیچ متغیری در آن بکار نرفته، تمام آیتمهای پنجرهٔ Autos ناپدید می‌شوند.

برای ردگیری مقدار یک متغیر در تمام طول حالت وقفه، باید از یکی دیگر از [illegible] استودیو بنام Watch کمک بگیریم. در ویژوال بیسیک .NET می‌توانید [illegible] حالیکه ویرایشهای قبلی ویژوال بیسیک به یک پنجرهٔ Watch محدود بودند. [illegible] مقدار عبارتهایی نظیر Age >= 13 را نیز ارزیابی کرد.

بازکردن یک پنجرهٔ Watch

۱ پروژهٔ Debug Test را (که در پوشهٔ c:\vbnetsbs\chap08\debug test قرار دارد) باز کرده، و بعد از قرار دادن یک نقطهٔ وقفه در دستور Age = TextBox1.Text ، آنرا اجرا کنید.

۲ عدد **20** را در جعبه متن ?How old are you وارد کرده، و دکمهٔ Test را کلیک کنید، تا برنامه وارد حالت وقفه شود.

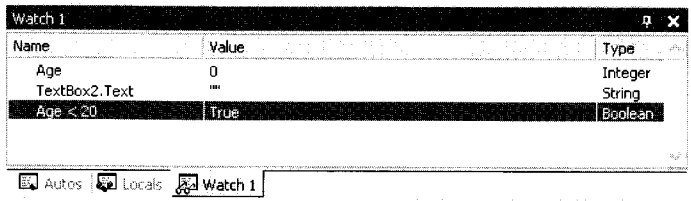
برای اضافه کردن یک متغیر یا عبارت به پنجرهٔ Watch ، کافیست آنرا انتخاب کنید، و سپس روی آن

راست-کلیک کرده و فرمان Add Watch را انتخاب کنید.

۳ متغیر **Age** را انتخاب کنید؛ سپس روی آن راست-کلیک کرده، و فرمان Add Watch را انتخاب کنید. با این کار پنجره‌ای بنام Watch 1 باز می‌شود، که متغیر Age را در آن می‌بینید (مقدار فعلی این متغیر 0 است). در ستون Value هم نوع این متغیر (عددصحیح - Integer) دیده می‌شود.

٤ خاصیت **TextBox2.Text** را انتخاب کرده، و آنرا روی یکی از سطرهای خالی پنجرهٔ Watch 1 بکشید و رها کنید.

۵ عبارت **Age < 20** را انتخاب کرده، و مانند بالا آنرا به پنجرهٔ Watch 1 اضافه کنید. عبارت Age < 20 یک عبارت شرطی است، و در پنجرهٔ Watch نوع آن بعنوان Boolean درج می‌شود (بیاد دارید که نوع Boolean فقط می‌تواند دو مقدار True یا False بگیرد). در شکل زیر پنجرهٔ Watch 1 را تا این مرحله می‌بینید:



حال وقت آنست که برنامه را بصورت خط به خط اجرا، و مقدار متغیرها را چک کنیم.

۶ در میله ابزار دیباگ، روی دکمهٔ Step Into کلیک کنید.

نکته

بجای کلیک کردن دکمهٔ Step Into ، می‌توانید کلید F8 را بزنید.

از آنجائیکه متغیر Age به 20 ست شده، شرط Age < 20 به False ارزیابی می‌شود. رنگ قرمز این متغیر در پنجرهٔ Watch نشان می‌دهد که بتازگی تغییر کرده است.

۷ سه بار پشت سر هم دکمهٔ Step Into را کلیک کنید. با این کار شرط If به False ارزیابی شده، و کنترل برنامه به قسمت Else می‌رود، که نتیجهٔ آن نمایش پیام "You're not a teenager" است. پس، برنامه تا اینجا درست کار می‌کند.

۸ روی سطر Age < 20 در پنجرهٔ Watch کلیک کنید، و کلید Delete را بزنید، تا این سطر حذف شود. (همانطور که می‌بینید، اضافه و کم کردن متغیرها و عبارتها به پنجرهٔ Watch بسیار ساده است.)

از آنجائیکه هنوز با متغیر Age کار داریم، از حالت وقفه خارج نشوید.

استفاده از پنجرهٔ Command

تا اینجا با ابزارهایی آشنا شدید که اجازه می‌دادند مقدار متغیرها و خواص اشیاء برنامه را تست کنید. اما ویژوال استودیو دارای ابزاریست که می‌توانید به کمک آنها مقدار متغیرها را تغییر داده، و تأثیر این کار را روی نحوهٔ اجرای برنامه ببینید. این ابزار پنجرهٔ Command نام دارد، و می‌تواند در دو حالت کار کند: حالت Immediate و حالت Command . وقتی در حالت Immediate هستید، می‌توانید با کُد برنامه تعامل داشته باشید، ولی در حالت Command می‌توانید فرمانهای ویژوال استودیو (از قبیل Save All و Print) را نیز اجرا کنید، و نتیجهٔ آنها ببینید.

در تمرین زیر نحوهٔ استفاده از پنجرهٔ Command را خواهید دید. در این تمرین فرض بر اینست که برنامهٔ Debug Test هنوز در حال اجرا در حالت وقفه است.

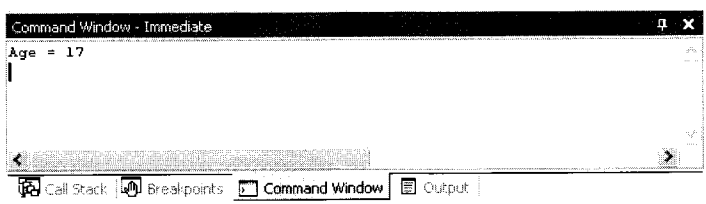
بازکردن پنجرهٔ Command در حالت Immediate

۱ از منوی Debug|Windows آیتم Immediate را انتخاب کنید، تا پنجرهٔ Command در حالت Immediate باز شود. تشخیص اینکه پنجرهٔ Command در چه حالتی قرار دارد، ساده است: عنوان این پنجره نشان می‌دهد که در کدام حالت است (برای مثال، عنوان "Command Window - Immediate" نشان می‌دهد که این پنجره در حالت Immediate قرار دارد). این اطلاع برای اجتناب از اجرای دستورات اشتباه لازم است.

نکته

سوئیچ کردن بین حالتهای پنجرهٔ Command بسادگی امکانپذیر است. اگر در حالت Command هستید، و می‌خواهید به حالت Immediate بروید، کافیست فرمان Immed را در این پنجره وارد کنید. و اگر در حالت Immediate قرار دارید، و می‌خواهید به حالت Command بروید، باید فرمان >cmd را وارد کنید (علامت > را فراموش نکنید).

۲ در پنجرهٔ Command دستور **Age = 17** را وارد کرده، و Enter را بزنید. با این کار مقدار متغیر Age بلافاصله تغییر می‌کند، که می‌توانید آنرا در پنجرهٔ Watch ببینید:



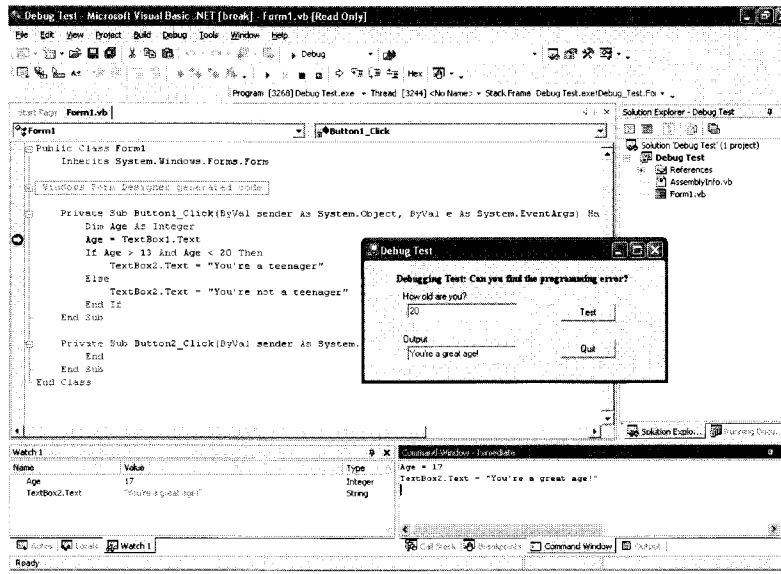
بار بعد که دستور If اجرا شود، شرط آن True شده، و پیام "You're a teenager" را نمایش خواهد داد.

۳ دستور زیر را در پنجرهٔ Command وارد کرده، و Enter را بزنید:

```
TextBox2.Text = "You're a great age!"
```

همانطور که می‌بینید، جعبه متن TextBox2 بلافاصله به "You're a great age!" تغییر می‌کند! در پنجرهٔ Command می‌توان هر متغیر یا خاصیتی را عوض کرد.

٤ دو بار دکمهٔ Step Into را کلیک کنید. توجه کنید که جعبه متن TextBox2 جملهٔ "You're a great age!" را نشان می‌دهد، ولی در جعبه متن TextBox1 همچنان عدد 20 دیده می‌شود (نه 17). علت اینست که ما فقط متغیر Age را تغییر دادیم، نه خاصیت TextBox1 را (شکل زیر را ببینید):



حالت Immediate پنجرهٔ Command (در کنار پنجرهٔ Watch) دو ابزار بسیار قوی برای ردیابی مشکلترین خطاهای منطق برنامه بحساب می‌آیند.

سوئیچ کردن به حالت Command در پنجرهٔ Command

از پنجرهٔ Command برای اجرای فرمانهای ویژوال استودیو نیز می‌توان استفاده کرد. یکی از فرمانهایی که تا بحال بدفعات آنرا بکار برده‌اید، فرمان File.SaveAll است (این فرمان معادل انتخاب فرمان Save All از منوی File است). اما برای اینکه بتوانید فرمانی را در این پنجره اجرا کنید، ابتدا باید به حالت Command سوئیچ کنید -کاری که در تمرین زیر انجام خواهیم داد.

اجرای فرمان File.SaveAll

۱ در پنجرهٔ Command دستور **cmd>** را وارد کرده، و Enter را بزنید، تا وارد حالت Command شوید. با این کار عنوان پنجرهٔ Command به "Command Window" تغییر کرده، و یک علامت > در ابتدای خط فرمان ظاهر می‌شود.

۲ فرمان **File.SaveAll** را وارد کرده، و Enter را بزنید –ویژوال استودیو کلیهٔ فایلهای پروژه را ذخیره کرده، و دوباره به پنجرهٔ Command برمی‌گردد.

نکته

بمحض اینکه کلمهٔ .File را در پنجرهٔ Command وارد کنید، ویژوال استودیو تمام فرمانهای مجاز File را نمایش خواهد داد، که می‌توانید فرمان موردنظر را از میان آنها انتخاب کنید. این یکی از ویژگیهای بسیار جالب پنجرهٔ Command است.

۳ با کلیک کردن دکمهٔ Close پنجرهٔ Command را ببندید.

یک گام فراتر: حذف نقاط وقفه

یک نقطهٔ وقفه تا زمانی که آنرا حذف نکنید، در برنامه باقی خواهد ماند –اتفاقی که اصلاً خوشایند نیست. در تمرین زیر نقطهٔ وقفه‌ای راکه در برنامهٔ Debug Test قرار دادیم، حذف خواهیم کرد.

حذف نقطهٔ وقفه از برنامهٔ Debug Test

۱ در ادیتور کُد، روی نقطهٔ قرمزی که در کنار دستور Age = TextBox1.Text قرار دارد، کلیک کنید: نقطهٔ وقفه به همین راحتی حذف می‌شود! اگر در یک برنامه چندین نقطهٔ وقفه دارید، می‌توانید با فرمان Debug|Clear All Breakpoints همهٔ آنها به یکباره پاک کنید. از آنجائیکه نقاط وقفه با پروژه ذخیره می‌شوند، پاک کردن آنها (بعد از آن که وظیفه‌اشان را انجام دادند) بسیار مهم است.

۲ در میله ابزار دیباگ، روی دکمهٔ Stop Debugging کلیک کنید، تا برنامهٔ Debug Test بسته شود.

۳ برای بستن میله ابزار دیباگ نیز می‌توانید، فرمان View|Toolbars|Debug را انتخاب کنید.

این فصل یکی از فصلهای بسیار مهم کتاب حاضر است، چون دربارهٔ یکی از مهمترین و ضروری‌ترین تکنیکهای برنامه‌نویسی صحبت کردیم. یک تکه کاغذ لای این قسمت بگذارید: گذرتان باز هم اینجا می‌افتد!

مرجع سریع فصل ۸

برای ...	انجام دهید
نمایش میله ابزار دیباگ	فرمان View|Toolbars|Debug را اجرا کنید.
برای ست کردن یک نقطهٔ وقفه	در ادیتور کُد، کنار دستور موردنظر و در میلهٔ حاشیه، کلیک کنید. یا، بعد از دستور موردنظر، دستور Stop را قرار دهید.
اجرای یک خط دستور در ادیتور کُد	در میله ابزار دیباگ، دکمهٔ Step Into را کلیک کنید.
بررسی مقدار یک متغیر، خاصیت یا عبارت	کرسر ماوس را روی آن ببرید، و چند لحظه صبر کنید (کلیک نکنید!).
نمایش مقدار متغیرها در پنجرهٔ Autos	در حالت وقفه، با فرمان Debug|Windows|Autos پنجرهٔ Autos را باز کنید.
اضافه کردن یک متغیر، خاصیت، یا عبارت به پنجرهٔ Watch	در حالت وقفه، مقدار موردنظر را انتخاب کرده، روی آن راست-کلیک کرده، و سپس فرمان Add Watch را انتخاب کنید.
باز کردن یک پنجرهٔ Watch	در حالت وقفه، فرمان Debug|Windows|Watch را اجرا کنید.
باز کردن پنجرهٔ Command در حالت Immediate	فرمان Debug|Windows|Immediate را اجرا کنید.
سوئیچ کردن بین حالتهای Immediate و Command درپنجرهٔ Command	برای رفتن به حالت Immediate فرمان Immed ، و برای سوئیچ به حالت Command فرمان cmd> را در پنجرهٔ Command اجرا کنید.
حذف یک نقطهٔ وقفه	روی نقطهٔ وقفهٔ موردنظر در میلهٔ حاشیه کلیک کنید (برای حذف تمام نقاط وقفه نیز می‌توانید فرمان Debug|Clear All Breakpoints را اجرا کنید).
توقف عملیات دیباگ کنید.	در میله ابزار دیباگ، روی دکمهٔ Stop Debugging کلیک

فصل

۹

مقابله با خطاهای برنامه با استفاده از روتین‌های ساخت‌یافتهٔ مقابله با خطا

در این فصل یاد می گیرید چگونه :

- ✔ با استفاده از دستور جدید Try...Catch با خطاهای زمان اجرا مقابله کنید.
- ✔ با دستور Catch When خطاها را در شرایط خاص تست کنید.
- ✔ برای تشخیص ماهیت خطاهای رُخ داده،از خواص Err.Number و Err.Description استفاده کنید.
- ✔ ساختارهای Try...Catch تودرتو ایجاد کنید.
- ✔ برنامه‌های خود را با استراتژی دفاعی بنویسید.
- ✔ برای خروج زودرس از روتین مقابله با خطا، از دستور Exit Try استفاده کنید.

در فصل قبل دیدید که چگونه می‌توان با استفاده از ابزارهای دیباگ ویژوال استودیو .NET خطاها و اشکالات منطق برنامه را کشف و برطرف کرد. در این فصل طرز مقابله با خطاهای زمان اجرا (که به آنها استثنا – exception – هم گفته می‌شود) را یاد خواهید گرفت. علت اطلاق لفظ استثنا به این قبیل خطاها آنست که اجتناب از آنها گاهی غیرممکن و از ارادهٔ برنامه‌نویس خارج است. برای مثال، هیچ برنامه‌ای نمی‌تواند با درایوی که اصلاً دیسکتی در آن نیست، و یا چاپگری که کاغذ ندارد، کار کند. بهترین تدبیر برای مقابله با این قبیل خطاها اینست که سعی کنیم کنترل برنامه را از دست ندهیم، و ضمن اخطار به کاربر (و امید به اینکه علت خطا را رفع کرده باشد) کار را از نو شروع کنیم. در ویژوال بیسیک .NET دستور جدیدی برای ایجاد روتین‌های ساخت‌یافتهٔ مقابله با خطا معرفی شده، که Try...Catch نام دارد. در این فصل ضمن آشنایی با طرز استفاده از دستور Try...Catch ، با خاصیت‌های Err.Number و Err.Description برای تشخیص نوع خطای رُخ داده آشنا می‌شوید، و یاد می‌گیرید که چگونه بلوک‌های Try...Catch تودرتو (nested) بسازید، و یا با دستور Exit Try بصورت زودرس از بلوک مقابله با خطا خارج شوید. ویژوال بیسیک .NET این تکنیکها را از زبانهای برنامه‌نویسی C++ و Java به ارث برده است (در ویرایشهای قبلی ویژوال بیسیک دستور کلیدی روتینهای مقابله با خطا، دستور ...On Error Goto بود).

تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- **دستور Try...Catch مکانیزم جدیدی برای نوشتن روتینهای مقابله با خطا در ویژوال بیسیک .NET است. با اینکه همچنان می‌توان از دستورات On Error Goto ، Resume، و Resume Next استفاده کرد، ولی ساختار Try...Catch کارایی بسیار بهتری دارد، و از مشکلات و پیچیدگیهای دستورات Goto نیز عاریست.**
- **با دستور جدید Catch When حتی می‌توان روتینهای مقابله با خطا را شرطی کرده، و با بیش از یک نوع استثنا مقابله کرد.**
- **دستور Exit Try روش جدیدی برای خروج زودرس از روتین مقابله با خطاست.**
- **ویژوال بیسیک .NET همچنان از متدهای Err.Number و Err.Description پشتیبانی می‌کند. علاوه بر آن، با خاصیت Err.GetException می‌توان اطلاعاتی دربارهٔ علت اصلی خطا (یا استثنا) بدست آورد.**

پردازش خطاها با دستور Try...Catch

خطای زمان اجرا (runtime error) یا استثنا (exception) خطائیست که خود ویژوال بیسیک نتواند آنرا رفع کند. علت این نوع از خطاها عدم توانایی (یا هوشمندی) ویژوال بیسیک نیست، علت آنست که او اساساً نمی‌داند چه باید بکند.

در مقابله با این نوع از خطاها (که برنامه را بکلی از کار می‌اندازند) چندان هم دست و پا بسته نیستیم، و می‌توانیم با نوشتن روتینهای مقابله با خطا (error handler) خود را از این موقعیتها خلاص کنیم. یک روتین مقابله با خطا به برنامه می‌گوید که در صورت عدم امکان اجرای یک دستور چکار باید بکند. روتینهای مقابله با خطا را هم می‌توان به دو صورت محلی (در داخل روال‌های رویداد) و یا عمومی (برای کل برنامه) نوشت. مهمترین عناصر یک روتین مقابله با خطا، دستور Try...Catch و شیء Err هستند. شیء Err دارای دو خاصیت بنامهای Numeber و Description است، که اولی شمارهٔ شناسایی خطا، و دومی شرحی کوتاه دربارهٔ آن برمی‌گرداند.

محل بکارگیری روتینهای مقابله با خطا

در هر موقعیتی که امکان بروز یک حادثهٔ مترقبه یا غیرمترقبه وجود داشته باشد (که بتواند برنامه را از کار بیندازد)، باید از روتینهای مقابله با خطا استفاده کرد. این روتینها معمولاً برای مقابله با شرایطی نوشته می‌شوند که خارج از حیطهٔ ارادهٔ برنامه و برنامه‌نویس است، مثلاً دیسکتی که داخل درایو نیست، فایلی که وجود خارجی ندارد، کابل شبکه‌ای که قطع است، و یا چاپگری که آمادهٔ چاپ نیست. در جدول زیر عمده‌ترین مسائل بالقوه‌ای که می‌توانند سبب بروز این نوع خطا شوند، را ملاحظه می‌کنید.

مشکل	توضیح
مشکلات شبکه/اینترنت قطع	سرورهای شبکه از کار افتاده‌اند؛ مودمها و ارتباطات اینترنت شده‌اند.
مشکلات دیسک/درایو	دیسک فرمت نشده؛ دیسک داخل درایو نیست؛ دیسک خراب است؛ دیسک پُر شده است.
مشکلات فایل	نام و مسیر فایل اشتباه است؛ فایل وجود ندارد.
مشکلات چاپگر	چاپگر آمادهٔ کار (online) نیست؛ چاپگر کاغذ ندارد؛ حافظهٔ چاپگر برای عملیات خواسته شده کافی نیست.
مشکلات نصب نرم‌افزار	یکی از فایلهای برنامه نصب نشده، و یا با سیستم عامل موجود سازگار نیست.
مشکلات دسترسی	کاربر مجوزهای لازم برای عملیات خواسته شده را در اختیار ندارد.
خطاهای سرریز	حجم عملیات خواسته شده از فضای حافظهٔ موجود فراتر رفته است.
خطاهای کمبود حافظه برنامه	سیستم عامل (ویندوز) حافظه‌ای برای تخصیص دادن به ندارد.
مشکلات تخته بُرش (clipboard)	سیستم عامل (ویندوز) مشکلاتی در تبادل اطلاعات با تخته بُرش دارد.
خطاهای منطق برنامه	خطاهایی در منطق برنامه که کامپایلر قادر به کشف آنها نبوده است (مانند نام اشتباه یک فایل).

کشف موقعیت خطا: دستور Try...Catch

همانطور که گفتم، دستور اصلی یک روتین مقابله با خطا دستور Try...Catch است. دو بخش این مجموعه (یعنی Try و Catch) کُدی راکه احتمال بروز خطا در آن می‌رود، محاصره می‌کنند. بعبارت دیگر، این بلوک کُد با Try شروع شده و با Catch پایان می‌یابد. (یک بلوک Try...Catch می‌تواند قسمتهای دیگری نیز داشته باشد، که در این فصل با آنها هم آشنا خواهید شد.) شکل کلی یک بلوک Try...Catch چنین است:

```
Try
    Statements that might produce a runtime error
Catch
    Statements to run if a runtime error occurs
Finally
    Optional statements to run whether an error occurs or not
End Try
```

که در آن Try ، Catch و End Try کلمات کلیدی هستند، ولی Finally (و دستورات بعد از آن) اختیاری‌اند. به دستوراتی که در یک بلوک Try...Catch قرار می‌گیرند، کُد حفاظت شده (protected code) نیز می‌گویند، چون این کُد دیگر نمی‌تواند باعث از کار افتادن برنامه شود (در هر حال ویژوال بیسیک می‌داند چکار کند).

خطاهای دیسک و درایو

در تمرین این قسمت یکی از خطاهای زمان اجرای متداول - خطای نام فایل یا درایو دیسک - را شبیه‌سازی می‌کنیم. برای اجرای این تمرین (و سایر تمرینهای این فصل) به یک دیسکت که فایلی بنام fileopen.bmp روی آن کپی شده، نیاز داریم.

تمرین خطای دیسک و درایو

۱ یک دیسکت خالی در درایو A: قرار داده، و فایل c:\vbnetsbs\chap09\disk drive error\fileopen.bmp را روی آن کپی کنید.

۲ ویژوال استودیو را اجرا کرده، و پروژهٔ Disk Drive Error را (که در پوشهٔ c:\vbnetsbs\chap09\disk drive error قرار دارد) باز کنید.

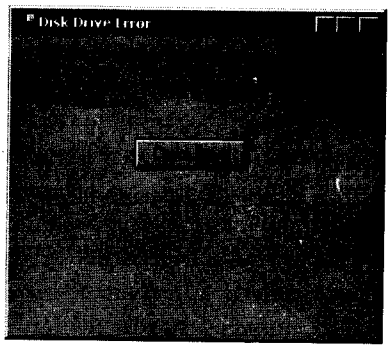
۳ روی فرم پروژهٔ Disk Drive Error یک دکمه بنام Check Drive قرار دارد، که وقتی روی آن کلیک کنید، برنامه فایل a:\fileopen.bmp را باز کرده، و در یک جعبه تصویر (زیر دکمهٔ Check Drive) نمایش می‌دهد.

٤ روی دکمهٔ Check Drive دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد باز شود؛ دستور زیر را در این روال خواهید دید:

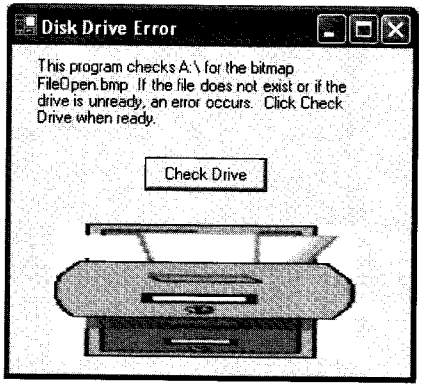
```
PictureBox1.Image = System.Drawing.Bitmap.FromFile("a:\fileopen.bmp")
```

همانطور که از فصلهای قبل بیاد دارید، متد FromFile یک فایل (در اینجا a:\fileopen.bmp) را از روی دیسک بار می‌کند. اما اگر این متد نتواند فایل را پیدا کند (مثلاً، دیسکت داخل درایو نباشد، یا مسیر آن اشتباه باشد)، ویژوال بیسیک یک خطای "File Not Found" تولید خواهد کرد - و این همان خطائیست که باید آنرا کشف کنیم (و یا اصطلاحاً بدام اندازیم).

۵ با کلیک کردن دکمهٔ Start ، برنامه را اجرا کنید (شکل زیر را ببینید):



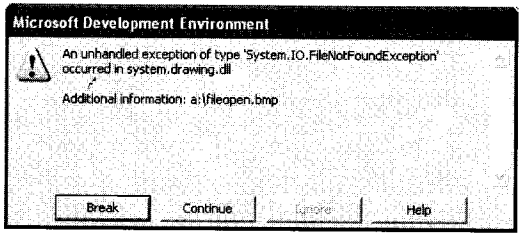
۶ دیسکت حاوی فایل fileopen.bmp را داخل درایو قرار داده، و سپس دکمهٔ Check Drive را کلیک کنید - روال Button1_Click اجرا شده، و پس از خواندن فایل مزبور از روی دیسکت، آنرا در جعبه تصویر نمایش می‌دهد:



حال اجازه دهید ببینیم وقتی دیسکت داخل درایو نیست، برنامه چگونه عمل می‌کند.

۷ دیسکت را از درایو خارج کنید.

۸ مجدداً دکمهٔ Check Drive را کلیک کنید - این بار ویژوال بیسیک یک خطای کنترل نشدهٔ زمان اجرا (unhandled exception) تولید کرده، و برنامه از کار می‌افتد:



۹ دکمهٔ Continue را کلیک کنید، تا برنامه بسته شده و به محیط ویژوال استودیو برگردید.

نوشتن روتین مقابله با خطای دیسک

پیام خطایی که در تمرین قبل دیدید ناشی از ناتوانی ویژوال بیسیک در مقابله با خطای دیسک نیست - مشکل اینجاست که هنوز به ویژوال بیسیک نگفته‌ایم وقتی خطایی اتفاق افتاد، چه عکس‌العملی باید نشان دهد. و این کاریست که با یک بلوک Try...Catch انجام خواهیم داد.

بدام انداختن خطا با دستور Try...Catch

۱ باز کردن فایل در روال رویداد Button1_Click صورت می‌گیرد، و روتین مقابله با خطا را باید در این روال بنویسیم - پس آنرا باز کنید.

۲ متد FromFile را بصورت زیر در یک بلوک Try...Catch قرار دهید:

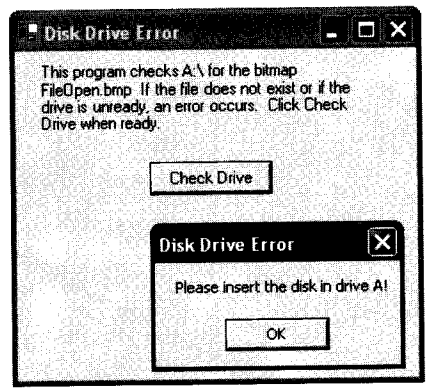
```
Try
    PictureBox1.Image = System.Drawing.Bitmap.FromFile("a:\fileopen.bmp")
Catch
```

```
    MsgBox("Please insert the disk in drive A!")
End Try
```

این ساده‌ترین شکل استفاده از دستور Try...Catch است. اگر دستور (یا دستوراتی) که در قسمت Try نوشته شده، باعث بروز خطای زمان اجرا شود، دستور (یا دستورات) قسمت Catch اجرا خواهد شد. در مثال فوق، فقط یک پیام در قسمت Catch نوشته‌ایم، که به کاربر گوشزد می‌کند که "دیسکت داخل درایو نیست؛ برای ادامهٔ برنامه دیسکت را داخل درایو A: قرار دهید." (این بلوک با End Try خاتمه یافته، و دارای قسمت Finally نیز نیست.)

۳ برنامه را اجرا کنید.

٤ مطمئن شوید دیسکت داخل درایو نیست، و سپس دکمهٔ Check Drive را کلیک کنید. این بار برنامه بجای اینکه از کار بیفتد، پیام زیر را نمایش می‌دهد:



۵ OK را کلیک کرده، و دوباره روی دکمهٔ Check Drive کلیک کنید. همانطور که می‌بینید، تا وقتی دیسکت را داخل درایو A: قرار ندهید، هر بار برنامه همان پیام را نشان می‌دهد.

۶ OK را کلیک کنید، دیسکت را داخل درایو قرار دهید، و یک بار دیگر دکمهٔ Check Drive را کلیک کنید. برنامه فایل fileopen.bmp را از روی دیسکت خوانده، و نمایش می‌دهد! بله، روتین مقابله با خطای ما بخوبی کار می‌کند!

روتینی که نوشتیم، کاربر را از شرّ پیامهای خطای عجیب و غریب (و اغلب نامفهوم) ویژوال بیسیک راحت کرده، و با نشان دادن علت خطا، او را در برطرف کردن آن راهنمایی می‌کند.

۷ با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

استفاده از دستور Finally برای مرتب کردن بیشتر کارها

در توضیح دستور Try...Catch دیدید که این دستور یک قسمت Finally نیز می‌تواند داشته باشد؛ این قسمت حاوی دستور (یا دستوراتی) است که در هر حال (چه خطایی روی دهد، یا ندهد) باید اجرا شود. این

کُد اغلب برای مرتب کردن نهایی کارها بکار می‌رود (و نام آن هم حکایت از همین موضوع دارد). در تمرین زیر طرز استفاده از بلوک Finally را خواهید دید.

نمایش پیام پایان کار با دستور Finally

۱ روال رویداد Button1_Click را باز کرده، و آنرا بصورت زیر اصلاح کنید:

```
Try
    PictureBox1.Image = System.Drawing.Bitmap.FromFile("a:\fileopen.bmp")
Catch
    MsgBox("Please insert the disk in drive A!")
Finally
    MsgBox("Error handler complete")
End Try
```

برای اینکه بهتر متوجه عملکرد قسمت Finally شوید، از تابع ()MsgBox برای اعلام پایان عملیات استفاده کرده‌ایم. (البته این فقط برای نمایش است، و در برنامه‌های واقعی معمولاً از بلوک Finally برای به روز در آوردن متغیرها، نمایش داده‌ها، فعال/غیرفعال کردن کنترل‌های برنامه، و کارهایی از این قبیل استفاده می‌شود.)

۲ برنامه را اجرا کنید.

۳ دیسکت را از درایو خارج کرده، و سپس دکمهٔ Check Drive را کلیک کنید؛ برنامه پیام می‌دهد که "دیسکت را داخل درایو :A قرار دهید." (این در واقع قسمت Catch برنامه بود.)

٤ OK را کلیک کنید، تا برنامه بلوک Finally را اجرا کرده، و پیام زیر را نشان دهد:

۵ OK را کلیک کنید، دیسکت را داخل درایو قرار دهید، و یک بار دیگر دکمهٔ Check Drive را کلیک کنید. برنامه (همانطور که انتظار داریم) فایل fileopen.bmp را از روی دیسکت خوانده، و نمایش می‌دهد - اما این بار کار دیگری نیز می‌کند، و آن نمایش مجدد پیام قسمت Finally است (همانطور که می‌بینید، بلوک Finally در هر دو حالت اجرا شد).

۶ OK را کلیک کنید، و سپس برنامه را ببندید.

روتینهای Try...Catch پیچیده‌تر

بلوکهای Try...Catch می‌توانند بسیار پیچیده‌تر باشند، و با خطاها در حالتهای مختلف مقابله کنند. برای این منظور می‌توان

- در هر یک از قسمتهای بلوک Try...Catch تعداد زیادی دستور ویژوال بیسیک نوشت.

- از دستور Catch When برای تست خطا در شرایط خاص استفاده کرد.
- بلوکهای Try...Catch را بصورت تودرتو (nested) نوشت.

علاوه بر این، می‌توان از شیء Err و خواص آن برای تشخیص دقیقتر نوع خطای رُخ داده بهره گرفت - کاری که در قسمت آینده انجام خواهیم داد.

شیء Err

شیء Err اطلاعات مربوط به آخرین خطای برنامه را در خود نگه می‌دارد. مهمترین خواص این شیء Err.Number و Err.Description هستند. خاصیت Err.Number شمارهٔ آخرین خطای رُخ داده را برمی‌گرداند، و خاصیت Err.Description توضیحاتی دربارهٔ این خطا را در خود نگه می‌دارد. با استفاده از این دو خاصیت می‌توان خطاها را شناخت، و نسبت به آنها عکس‌العمل مناسب را نشان داد.

برای پاک کردن اطلاعات قبلی شیء Err می‌توانید متد Err.Clear را بکار ببرید، ولی این کار در روتینهای Try...Catch لزومی ندارد، چون کُد Catch فقط وقتی اجرا می‌شود که در اجرای قسمت Try خطایی روی داده باشد.

در جدول زیر لیستی از متداولترین خطاهایی که در یک برنامهٔ ویژوال بیسیک می‌تواند روی دهد، را ملاحظه می‌کنید. (در سیستم کمک ویژوال بیسیک نیز می‌توانید اطلاعات کاملتری دربارهٔ این خطاها - و خطاهای خاص سیستمهای دیگر، از قبیل سیستمهای پایگاه داده - بدست آورید.)

شمارهٔ خطا	پیام خطای پیش‌فرض
5	فرخوانی روال یا آرگومانهای آن معتبر نیست
6	سرریز
7	حافظه موجود نیست
9	خارج از محدودهٔ زیرنویس (اندیس)
11	تقسیم بر صفر
13	عدم انطباق نوع داده
48	خطا در بارگیری DLL
51	خطای داخلی
52	شماره یا نام فایل صحیح نیست
53	فایل پیدا نشد
55	فایل قبلاً باز شده
57	خطای ابزار I/O
58	فایل از قبل وجود دارد
61	دیسک پُر است
62	عبور ورودی از مرز انتهای فایل
67	تعدا فایلها از حد مجاز بیشتر است
68	ابزار موجود نیست
70	عدم وجود مجوز دسترسی

شمارهٔ خطا	پیام خطای پیش‌فرض
71	دیسک آماده نیست
74	تغییر نام فایل در درایو دیگر ممکن نیست
75	خطای دسترسی مسیر/فایل
76	مسیر پیدا نشد
91	متغیر Object یا متغیر بلوک With ست نشده
321	فرمت فایل معتبر نیست
322	ایجاد فایل موقتی موردنیاز ممکن نیست
380	مقدار خاصیت معتبر نیست
381	اندیس آرایهٔ خاصیت معتبر نیست
422	خاصیت پیدا نشد
423	خاصیت یا متد پیدا نشد
424	شیء موردنیاز است
429	ایجاد شیء اکتیوایکس ممکن نیست
430	کلاس از اتوماسیون (یا واسط خواسته شده) پشتیبانی نمی‌کند
438	شیء از این خاصیت یا متد پشتیبانی نمی‌کند
440	خطای اتوماسیون
460	فرمت تخته‌بُرش (clipboard) معتبر نیست
461	متد یا دادهٔ عضو پیدا نشد
462	کامپیوتر میزبان راه دور در دسترس نیست یا وجود ندارد
463	کلاس روی کامپیوتر محلی رجیستر (ثبت) نشده
481	تصویر معتبر نیست
482	خطای چاپگر

در تمرین زیر به کمک خواص Err.Number و Err.Description در یک بلوک Try...Catch (و همچنین استفاده از دستور Catch When) چندین خطا را کنترل خواهیم کرد.

تست چند خطا در یک بلوک Try...Catch

۱ روال رویداد Button1_Click را باز کرده، و آنرا بصورت زیر اصلاح کنید:

```
Try
    PictureBox1.Image = System.Drawing.Bitmap.FromFile("a:\fileopen.bmp")
Catch When Err.Number = 53    'if File Not Found error
    MsgBox("Check pathname and disk drive")
Catch When Err.Number = 7     'if Out Of Memory error
    MsgBox("Is this really a bitmap?", , Err.Description)
Catch
    MsgBox("Problem loading file", , Err.Description)
End Try
```

در این کُد، دو بار از دستور Catch When برای تست نوع خاصی از خطا استفاده کرده‌ایم. اگر

خاصیت Err.Number مقدار 53 داشته باشد، یعنی برنامه نتوانسته فایل fileopen.bmp را پیدا کند، و پیام "Check pathname and disk drive" را نشان خواهد داد. اما اگر این خاصیت مقدار 7 داشته باشد، یعنی سیستم حافظهٔ کافی برای نمایش فایل در اختیار ندارد، و پیام این قسمت هم همین اخطار را به کاربر می‌دهد -این خطا معمولاً زمانی روی می‌دهد که بخواهید یک فایل غیرتصویری را در جعبه تصویر بار کنید.

سومین Catch هم برای بقیهٔ مواردیست که در دو حالت قبلی نگنجیده باشند. این قسمت فقط توضیح خطا (خاصیت Err.Description) را به کاربر ارائه می‌کند، و تصمیم‌گیری را بر عهدهٔ خود وی می‌گذارد.

۲ برنامه را اجرا کنید.

۳ دیسکت را از درایو :A خارج کنید.

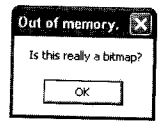
٤ دکمهٔ Check Drive را کلیک کنید؛ برنامه پیام "Check pathname and disk drive" را می‌دهد، و این به معنای اجرای اولین Catch When است.

۵ OK را کلیک کنید، و سپس برنامه را ببندید.

۶ دیسکت را داخل درایو قرار داده، و یک فایل غیرتصویری (مثلاً، یک سند Word یا یک کاربرگ Excel) را در آن کپی کنید.

۷ بجای فایل fileopen.bmp در متد FromFile (در روال رویداد Button1_Click) نام این فایل را بنویسید. این کار به ما اجازه می‌دهد تا عمداً یک خطای Out Of Memory تولید کرده، و Catch When دوم را تست کنیم.

۸ برنامه را اجرا کرده، و دکمهٔ Check Drive را کلیک کنید. روتین مقابله با خطا پیام زیر را نمایش می‌دهد:



۹ OK را کلیک کنید، و سپس برنامه را ببندید.

۱۰ نام فایل متد FromFile را مجدداً به fileopen.bmp تغییر دهید، و پروژه را ذخیره کنید - در تمرین بعد باز هم با این برنامه کار داریم.

دستور Catch When در کنار خاصیتهای Err.Number و Err.Description ابزاری بسیار قدرتمند برای مقابله با انواع خطاهای زمان اجرا بحساب می‌آید، و با آن می‌توان چندین خطا را در یک روتین کنترل کرد.

تولید عمدی خطا

گاهی برای تست یک برنامه در شرایط مختلف لازم است بطور عمدی در آن خطا ایجاد کرده، و عکس‌العمل آنرا بررسی کنیم. شیء Err برای این منظور متد ویژه‌ای بنام Raise دارد. برای مثال، بلوک Try...Catch زیر ابتدا خطای Disk Full را تولید کرده، و سپس در دستور Catch When به آن عکس‌العمل نشان می‌دهد:

```
Try
    Err.Raise(61)    'raise Disk Full error
Catch When Err.Number = 61
    MsgBox("Error: Disk is full")
End Try
```

محدود کردن تکرارها

بدام انداختن خطاها و اجازه دادن به کاربر برای تصحیح اشتباهاتش و تکرار عملیات، یکی از کلیدی‌ترین ویژگیهای هر برنامه‌ایست، اما بالاخره هر چیزی حدی دارد! (آیا باید تا ابد صبر کنیم، تا شاید کاربر به صرافت بیفتد و دیسکت را توی درایو بگذارد!) در تمرین زیر برای محدود کردن تعداد دفعاتی که کاربر می‌تواند عملیات را تکرار کند، از یک متغیر بنام Retries استفاده کرده‌ایم. در این مثال تعداد دفعات تکرار را به ۲ بار محدود خواهیم کرد.

استفاده از یک متغیر برای ردگیری خطاهای زمان اجرا

۱ در ادیتور کُد و در بالاترین نقطهٔ آن (درست زیر عنوان "Windows Forms Designer generated code") متغیر زیر را تعریف کنید:

```
Dim Retries As Short = 0
```

متغیر Retries از نوع عدد صحیح کوچک (Short) انتخاب شده، و هر بار در شروع برنامه مقدار آن 0 می‌شود.

۲ در روال Button1_Click ، بلوک Try...Catch را مانند زیر تغییر دهید:

```
Try
    PictureBox1.Image = System.Drawing.Bitmap.FromFile("a:\fileopen.bmp")
Catch
    Retries += 1
    If Retries <= 2 Then
        MsgBox("Please insert the disk in drive A!")
    Else
        MsgBox("File Load feature disabled")
        Button1.Enabled = False
    End If
End Try
```

بلوک Try باز هم فایل fileopen.bmp را باز می‌کند، ولی این بار Catch حداکثر دو بار اجرا می‌شود،

و اگر کاربر نتواند در این فرصت عکس‌العمل مناسب نشان دهد، دکمهٔ Check Drive را غیرفعال می‌کند.

۳ برنامه را اجرا کنید. (کُد کامل این برنامه را می‌توانید در پوشهٔ x:\vbnetsbs\chap09\disk drive handler بیابید.)

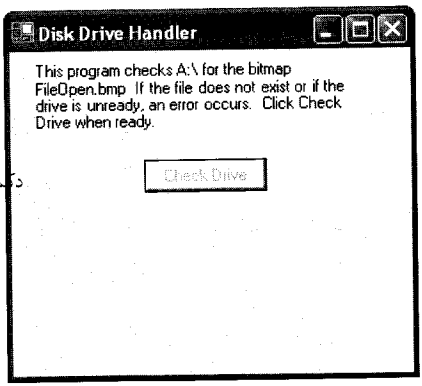
٤ دیسکت را از درایو :A خارج کنید.

۵ دکمهٔ Check Drive را کلیک کنید؛ برنامه پیام "Please insert the disk in drive A!" را نشان می‌دهد (و قبل از آن متغیر Retries را یکی اضافه می‌کند):

۶ OK را کلیک کرده، و مجدداً روی دکمهٔ Check Drive کلیک کنید. برنامه باز هم همان پیام "Please insert the disk in drive A!" را می‌دهد، و این بار متغیر Retries (با یکی اضافه شدن) به عدد 2 می‌رسد.

۷ OK را کلیک کرده، و برای بار سوم Check Drive را کلیک کنید. وقتی Retries به 3 برسد، دیگر نوبت قسمت Else و نمایش پیام "File Load feature disabled" است:

۸ باز هم OK را کلیک کنید - بله، دکمهٔ Check Drive غیرفعال شده، و کاربر دیگر راهی برای خطا کردن ندارد! (البته از آنجائیکه برنامهٔ ما کار دیگری ندارد که انجام دهد، با این وضعیت عملاً از کار می‌افتد.)



دکمهٔ Check Drive غیرفعال شده

۹ برنامه را ببندید.

بلوک‌های Try...Catch تودرتو

بلوکهای Try...Catch را می‌توان داخل یکدیگر و بصورت **تودرتو** (nested) نیز بکار برد. در مثال زیر همان تکنیک محدود کردن دفعات تکرار را این بار با استفاده بلوکهای Try...Catch تودرتو پیاده‌سازی کرده‌ایم:

```
Try
    PictureBox1.Image = System.Drawing.Bitmap.FromFile("a:\fileopen.bmp")
Catch
    MsgBox("Please insert the disk in drive A!")
    Try
        PictureBox1.Image = System.Drawing.Bitmap.FromFile("a:\fileopen.bmp")
    Catch
        MsgBox("File Load feature disabled")
        Button1.Enabled = False
    End Try
End Try
```

وقتی کاربر بار اول دچار خطا می‌شود، دستور Catch اول اجرا شده، و بعد از نمایش پیام "Please insert the disk in drive A!" ، مجدداً (در دستور Try دوم) اقدام به بار کردن فایل fileopen.bmp می‌کند. اگر در این بار هم شکست بخورد، دستور Catch دوم اجرا شده، و ضمن نمایش پیام "File Load feature disabled" ، دکمهٔ Check Drive را غیرفعال می‌کند.

البته تکنیک قبلی (استفاده از یک متغیر عمومی) برای ردگیری تعداد دفعات تکرار روتین Try...Catch نسبت به این روش ارجحیت دارد، چون کنترل خطاهای چندگانه در آن راحتتر است.

تکنیکهای برنامه‌نویسی دفاعی

روتینهای مقابله با خطا تنها روش حفاظت برنامه در مقابل خطاهای زمان اجرا نیست. در کُد زیر به منظور اطمینان از وجود یک فایل، قبل از اقدام برای باز کردن آن، از متد File.Exists (که جزء کتابخانهٔ System.IO است) استفاده کرده‌ایم:

```
If File.Exists ("a:\fileopen.bmp") Then
    PictureBox1.Image = System.Drawing.Bitmap.FromFile("a:\fileopen.bmp")
Else
    MsgBox("Cannot find fileopen.bmp on drive A.")
End If
```

در واقع، بلوک If...Then یک سیستم مقابله با خطا نیست، چون اگر خطایی رُخ دهد، نمی‌تواند جلوی از کار افتادن برنامه را بگیرد - این فقط یک اقدام احتیاطی است، که به آن برنامه‌نویسی **دفاعی** گفته می‌شود. در این روش، برنامه‌نویس قبل از اقدام به یک کار، ابتدا از صحت آن اطمینان حاصل کرده، و سپس آنرا انجام می‌دهد. مزیت دیگر این روش، سرعت بیشتر آن نسبت به روتینهای مقابله با خطاست.

توجه

برای آن که این برنامه بتواند بدرستی اجرا شود، دستور زیر را باید به بالای فرم اضافه کنید:

```
Imports System.IO
```

جای استفاده از تکنیکهای برنامه‌نویسی دفاعی کجاست؟ بعنوان یک قاعدهٔ کلی، وقتی احتمال وقوع یک خطای ساده در قسمتی از کُد برنامه بیش از ۲۵٪ باشد، قبل از اجرای آن کُد اقدامات احتیاطی واجب است. ولی اگر احتمال بروز یک خطا کمتر از این مقدار باشد (و یا می‌خواهید در صورت وقوع خطای زمان اجرا، امکانات متنوعتری در اختیار کاربر قرار دهید) بهتر است از روتینهای Try...Catch استفاده کنید.

البته برای رسیدن به بهترین نتیجه، می‌توانید بصورت همزمان از تکنیکهای برنامه‌نویسی دفاعی و روتینهای مقابله با خطا استفاده کنید.

یک گام فراتر: دستور Exit Try

این فصل را نمی‌توانیم به پایان ببریم، بدون اینکه سخن از یک گزینهٔ مهم دیگر در روتینهای Try...Catch زده باشیم: دستور Exit Try . دستور Exit Try نیز (مانند همتایانش: Exit For و Exit Sub) یک راه میانبُر برای خروج زودرس از بلوک Try...Catch است (البته اگر بلوک Finally وجود داشته باشد، دستور Exit Try قبل از خروج آنرا اجرا خواهد کرد).

در مثال زیر، قبل از بار کردن فایل تصویر، ابتدا خاصیت Enabled کنترل PictureBox1 چک می‌شود: اگر این کنترل غیرفعال باشد، ادامهٔ برنامه لزومی ندارد (چون جعبه تصویر هنوز آمادهٔ بار کردن تصویر نیست).

```
Try
    If PictureBox1.Enabled = False Then Exit Try
    PictureBox1.Image = System.Drawing.Bitmap.FromFile("a:\fileopen.bmp")
Catch
    Retries += 1
    If Retries <= 2 Then
        MsgBox("Please insert the disk in drive A!")
    Else
        MsgBox("File Load feature disabled")
        Button1.Enabled = False
    End If
End Try
```

اکنون دیگر روشهای مقابله با خطا را در برنامه‌های ویژوال بیسیک .NET یاد گرفته‌اید، و آماده‌ایم تا به سرفصلهای پیشرفته‌تر برنامه‌نویسی بپردازیم.

مرجع سریع فصل ۹

برای ...	انجام دهید

تشخیص و پردازش خطاها

کُد برنامه را در یک بلوک Try...Catch قرار دهید:

```
Try
    Statements to be executed normally
Catch
    Statements to be executed if error occurs
Finally
    Statements to be executed anyway
End Try
```

تشخیص خطاهای خاص

از دستور Catch When و خاصیت Err.Number استفاده کنید:

```
Try
    Statements to be executed normally
Catch When Err.Number = xxx
    Statements to be executed if error xxx occurs
Catch When Err.Number = yyy
    Statements to be executed if error yyy occurs
Catch
    Statements to be executed if other errors occurs
End Try
```

ایجاد خطاهای عمدی در برنامه

از دستور Err.Raise استفاده کنید:

```
Try
    Err.Raise(xxx)
Catch When Err.Number = xxx
    Statements to be executed if error xxx occurs
End Try
```

خروج زودرس از بلوک مقابله با خطا

از دستور Exit Try استفاده کنید:

```
Try
    If condition Then Exit Try
    Statements to be executed normally
Catch
    Statements to be executed if error occurs
End Try
```

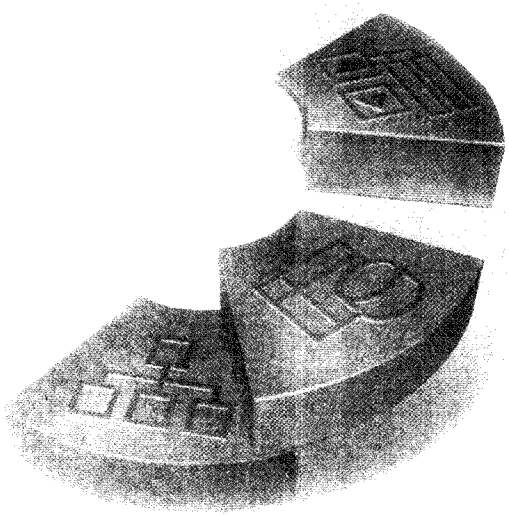
VISUAL BASIC .NET

آموزش گام به گام

۳

بخش

کار با انواع داده



فصل

۱۰

ماژول و روال

در این فصل یاد می گیرید چگونه :

- ✔ ماژول‌های استاندارد ایجاد کنید.
- ✔ متغیرهای عمومی (با میدان دید وسیع) تعریف کرده، و از آنها استفاده کنید.
- ✔ تابع (function) و سابروتین (subroutine) ـ که در مجموع روال نامیده می‌شوند ـ بنویسید.
- ✔ روال‌هایی را که نوشته‌اید، فراخوانی کنید.

بعد از به پایان رساندن فصلهای ۱ تا ۹، اکنون می‌توانید خود را یک برنامه‌نویس متوسط ویژوال بیسیک بخوانید. از این بخش قدم به وادی تکنیکهای پیشرفتهٔ برنامه‌نویسی می‌گذاریم، و در همین قدم اول با طرز ایجاد ماژولهای استاندارد آشنا خواهید شد. ماژول استاندارد (standard module) بخشی مستقل از برنامه است، که متغیرها و روالهای Sub (سابروتین) و Function (تابع) خاص خود را دارد.

در این فصل همچنین با نحوهٔ تعریف و استفاده از متغیرهای عمومی (public) نیز آشنا می‌شوید.

تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- ویژوال بیسیک .NET همچنان از ماژولهای استاندارد پشتیبانی می‌کند، اما برای ایجاد این ماژولها باید آنها را در یک بلوک Module...End Module قرار دهید. (نحوهٔ تعریف متغیرهای عمومی در ویژوال بیسیک .NET تفاوتی با ویژوال بیسیک ۶ ندارد.)
- نحوهٔ ایجاد روالهای Sub و Function در ویژوال بیسیک .NET تفاوتی با ویرایشهای قبلی ویژوال بیسیک نکرده است، اما تعریف و فراخوانی آنها فرق کوچکی با ویژوال بیسیک ۶ کرده است.
- اگر برای کنترل تعریف متغیرها از گزینهٔ پیش‌فرض Option Explicit استفاده کرده باشید، بهتر است نوع مقدار برگشتی روالهای Function (و همچنین نوع دادهٔ لیست آرگومانهای آن) را با کلمهٔ کلیدی As مشخص کنید. اگر چنین نکنید، ویژوال بیسیک .NET از نوع دادهٔ Object استفاده خواهد کرد، که کارایی برنامه را بشدت پائین می‌آورد.
- مکانیزم پیش‌فرض ارسال آرگومانها در ویژوال بیسیک .NET به روش ByVal (ارسال مقدار) تغییر کرده است؛ در ویرایشهای قبلی ویژوال بیسیک بصورت پیش‌فرض از مکانیزم ByRef (ارسال آدرس) استفاده می‌شد. البته هر گاه خواسته باشید، می‌توانید با استفاده از کلمات کلیدی ByVal و ByRef این مکانیزم را عوض کنید.
- هنگام فراخوانی یک روال در ویژوال بیسیک .NET نوشتن پرانتزها الزامیست (حتی اگر روال موردنظر هیچ آرگومانی نداشته باشد).
- برای برگشت دادن مقدار برگشتی یک تابع در ویژوال بیسیک .NET ، می‌توانید از کلمهٔ کلیدی Return نیز استفاده کنید (البته روش قدیمی نسبت دادن این مقدار به نام تابع نیز همچنان پشتیبانی می‌شود).

ماژول‌های استاندارد

هر قدر یک برنامه بزرگتر و پیچیده‌تر می‌شود، احتمال اینکه بخواهید از چند متغیر در فرمها و روال‌های رویداد متعددی استفاده کنید، بیشتر خواهد شد. بصورت پیش‌فرض، میدان دید (scope) یک متغیر (محدوده‌ای که این متغیر دیده می‌شود، و می‌توان آنرا دستکاری کرد) به همان روالی که در آن تعریف شده، محدود است؛ به همین دلیل به آنها متغیر محلی (local variable) می‌گویند. همانطور که در فصلهای قبل دیدید، اگر متغیری را در بالای فرم (خارج از تمام روالهای رویداد) تعریف کنید، میدان دید آن به تمامی فرم گسترش می‌یابد. اما اگر برنامهٔ شما چندین فرم داشته باشد، متغیرهای هر فرم (حتی اگر عمومی باشند) به همان فرم محدود خواهند ماند. همین وضعیت در مورد روالهای رویداد نیز صادق است؛ برای مثال، روال رویداد کلیک دکمه‌ای بنام Button1 (که روی Form1 قرار دارد) را نمی‌توان در فرمهای دیگر برنامه (مثلاً، Form2) فراخوانی کرد.

توجه

در فصل ۱۵ خواهید دید که چگونه می‌توان فرمهای دیگری به برنامه اضافه کرد.

برای به اشتراک گذاشتن یک متغیر بین تمام فرمها و روالهای رویداد برنامه، باید آنرا در یک ماژول استاندارد تعریف کنید. ماژول استاندارد (یا ماژول کُد) فایلیست با پسوند vb. ،که متغیرها و روالهای داخل آن از تمام برنامه قابل دسترسی‌اند. (در ویرایشهای قبلی ویژوال بیسیک، ماژولهای استاندارد پسوند bas. دارند.) ماژول استاندارد (بر خلاف فرم) دارای عنصر گرافیکی نیست، و فقط کُد دارد؛ ولی آنها نیز مانند فرمها در کاوشگر راه‌حل بصورت آیتمهای مستقل دیده می‌شوند، و می‌توان با فرمان Save *Module1* As مستقلاً آنها را ذخیره کرد.

ایجاد ماژول‌های استاندارد

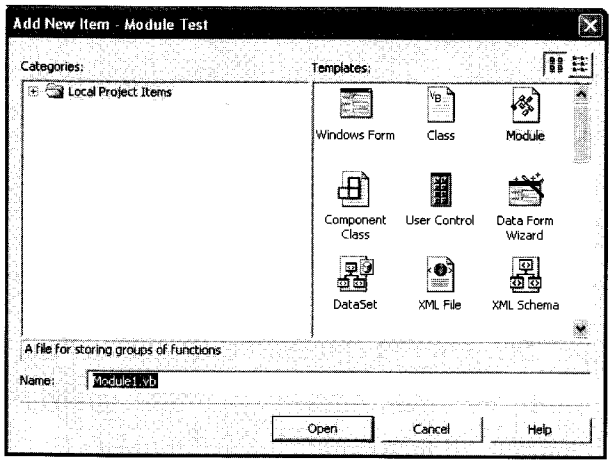
برای اضافه کردن یک ماژول استاندارد به برنامه می‌توانید از فرمان Project|Add New Item (و انتخاب آیتم Module در پنجره‌ای که باز می‌شود) استفاده کنید. نام پیش‌فرض اولین ماژولی که به برنامه اضافه می‌شود، Module1.vb است. برای تغییر دادن نام این ماژول می‌توانید آنرا با فرمان Save Module1 As (با نام دیگری) ذخیره کنید؛ و یا در کاوشگر راه‌حل روی آن راست-کلیک کرده، و نام آنرا عوض کنید. در تمرین زیر یک ماژول استاندارد به پروژه اضافه خواهیم کرد.

یک ماژول استاندارد ایجاد و ذخیره کنید

۱ ویژوال استودیو .NET را باز کرده، و یک پروژهٔ Visual Basic Windows Application بنام **My Module Test** در پوشهٔ c:\vbnetsbs\chap10 ایجاد کنید.

۲ فرمان Add New Item را از منوی Project انتخاب کنید، تا پنجرهٔ Add New Item باز شود.

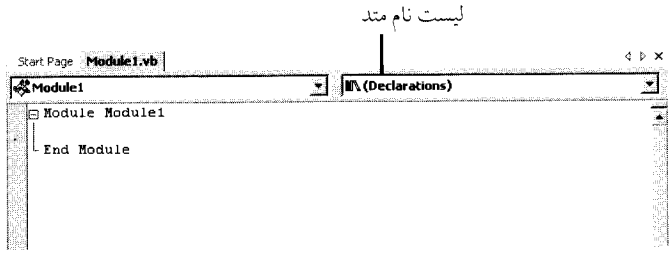
۳ آیتم Module را انتخاب کنید؛ نام این ماژول جدید Module1.vb است (شکل زیر را ببینید):



نکته

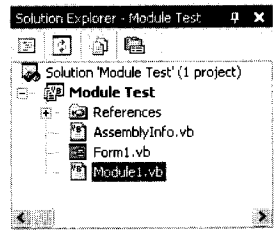
در این پنجره آیتمهای متعددی می‌بینید، ولی سه تا از آنها اهمیت بیشتری دارند: Windows Form (فرم ویندوز)، Class (کلاس)، Module (ماژول). ماژول جائیست که کُدها و متغیرهای عمومی برنامه تعریف می‌شوند. با فرمهای ویندوز در فصلهای قبل آشنا شدید (و در فصل ۱۵ خواهید دید که چگونه می‌توان فرمهای دیگری به برنامه اضافه کرد). با مفهوم کلاس و طرز ایجاد آنها نیز در فصل ۱۷ آشنا خواهید شد.

٤ دکمهٔ Open را کلیک کنید، تا ماژول Module1 به برنامه اضافه شده و در ادیتور کُد باز شود.



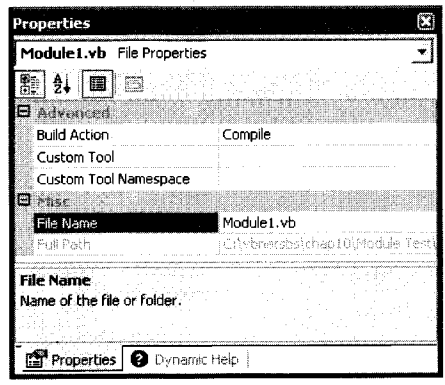
متغیرها و روالهایی که در این قسمت تعریف و نوشته شوند، در کل برنامه دیده خواهند شد.

۵ روی میلهٔ عنوان کاوشگر راه‌حل (Solution Explorer) دو-کلیک کنید، تا به حالت شناور در آید:



همانطور که می‌بینید، کاوشگر راه‌حل ماژول جدید (که نام پیش‌فرض آن Module1.vb است) را در کنار سایر اجزاء برنامه نشان می‌دهد. ویژوال استودیو به شما اجازه می‌دهد تا نام این فایل را براحتی تغییر دهید. برای این کار،

۶ روی میلهٔ عنوان پنجرهٔ خواص (Properties window) دو-کلیک کنید، تا به حالت شناور در آید:



از آنجائیکه یک ماژول فقط حاوی کُد است، خواص بسیار محدودی دارد که مهمترین آنها خاصیت File Name است. (با سایر خواص ماژول فعلاً کاری نداریم.)

۷ خاصیت File Name ماژول Module1.vb را به Math Functions.vb (یا هر نام دیگری که میل دارید، چون این فقط یک مثال است) تغییر دهید.

۸ پنجرهٔ خواص و کاوشگر راه‌حل را به حالت چسبیده برگردانید.

نکته

برای حذف یک ماژول از پروژه، ابتدا در کاوشگر راه‌حل آن را انتخاب کرده، و سپس فرمان Project|Exclude From Project را اجرا کنید. توجه کنید که این فرمان فایل ماژول را از روی دیسک پاک نمی‌کند، و فقط ارتباط آنرا با پروژه قطع می‌کند. برای اضافه کردن مجدد این ماژول به پروژه نیز می‌توانید از فرمان File|Add Existing Item استفاده کنید.

متغیرهای عمومی

تعریف متغیرهای عمومی در ماژول استاندارد ساده است: کافیست آنها را با استفاده از کلمهٔ کلیدی Public تعریف کنید:

```
Public RunningTotal As Integer
```

این متغیر در تمام روالهای برنامه قابل دسترسی است و می‌توان آنرا دستکاری کرد.

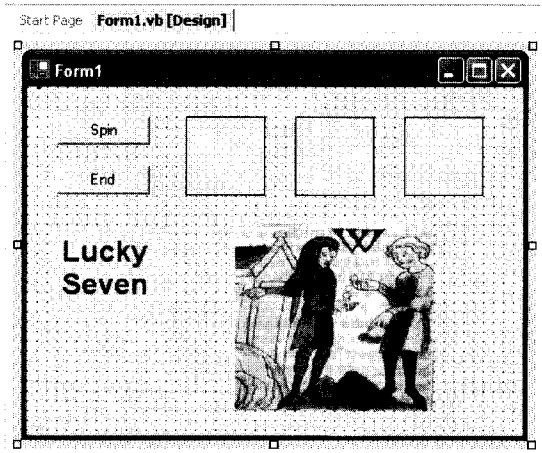
در تمرین زیر دوباره به برنامهٔ Locky Seven (که در فصل ۲ نوشتیم) برمی‌گردیم، و برای تعیین تعداد دفعاتی که کاربر برنده شده است از یک متغیر عمومی استفاده می‌کنیم.

یک اصلاح در برنامهٔ Locky Seven

۱ پروژهٔ قبلی را با فرمان Close Solution از منوی File ببندید.

۲ به پوشهٔ c:\vbnetsbs\chap10\track wins بروید، و پروژهٔ **Track Wins** را باز کنید.

۳ از طریق کاوشگر راه‌حل، فرم برنامه (Form1.vb) را باز کنید:



این پروژه دقیقاً همان Locky7 فصل ۲ است، و فقط یک کپی از آن گرفته و نام آنرا عوض کرده‌ایم.

٤ برنامه را اجرا کنید.

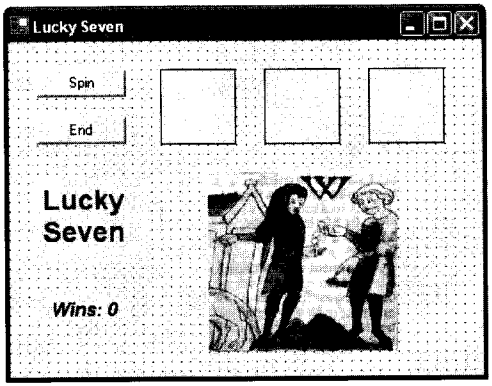
٥ شش یا هفت بار روی دکمهٔ Spin کلیک کنید؛ بخاطر دارید که در این برنامه اگر هر یک از برچسب‌ها عدد 7 را نشان دهد، برنده محسوب می‌شوید. با کلیک کردن دکمهٔ End برنامه را ببندید.

برای اینکه برنامه بتواند تعداد بردها را نشان دهد، از یک ماژول استاندارد استفاده می‌کنیم (به یک برچسب هم برای نمایش تعداد بردها نیاز داریم).

اضافه کردن یک ماژول استاندارد

۱ به فرم پروژهٔ Track Wins برگردید.

۲ کنترل برچسب (Label) را از جعبه ابزار ویژوال بیسیک انتخاب کرده، و یک برچسب جدید زیر برچسب عنوان برنامه رسم کنید (شکل زیر راببینید).



۳ خواص این برچسب را مانند جدول زیر ست کنید.

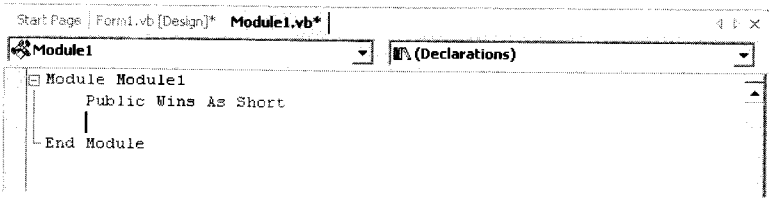
شیء	خاصیت	مقدار
Form1	Text	"Locky Seven"
Label5	Font	Arial, Bold Italic, 12-point
	ForeColor	Green
	Name	lblWins
	Text	"Wins: 0"
	TextAlign	MiddleCenter

سپس ماژول استاندارد را به برنامه اضافه می‌کنیم.

٤ از منوی Project فرمان Add New Item را انتخاب کرده، و پس از انتخاب آیتم Module دکمهٔ Open را کلیک کنید؛ ماژول Module1.vb به برنامه اضافه می‌شود.

۵ دستور زیر را بین Module و End Module وارد کنید:

```
Public Wins As Short
```

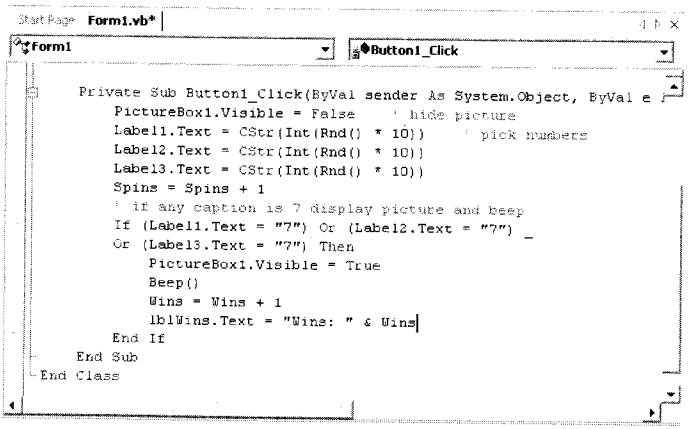


۶ به فرم برنامه برگردید، و روی دکمهٔ Spin دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد باز شود.

۷ دستورات زیر را درست بعد از دستور ()Beep بنویسید:

```
Wins = Wins +1
lblWins.Text = "Wins: " & Wins
```

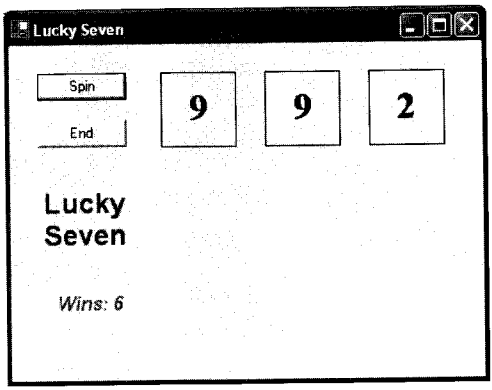
این دو دستور هر بار که شرط برنده شدن تحقق یابد، مقدار متغیر عمومی Wins را افزایش داده و سپس آنرا در برچسب lblWins می‌نویسند.



۸ با کلیک کردن دکمهٔ Save All در میله ابزار استاندارد، پروژه را ذخیره کنید. فرمان Save All تمام اجزای پروژه از جمله ماژول Module1.vb را ذخیره می‌کند. (کُد کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap10\final track wins پیدا کنید.)

۹ برنامه را اجرا کنید.

۱۰ دکمهٔ Spin را ۱۰ بار کلیک کنید. همانطور که می‌بینید، هر بار که برنده می‌شوید، برچسب lblWins یکی اضافه می‌شود.



۱۱ با کلیک کردن دکمهٔ End برنامه را ببندید.

اگر متغیر Wins را داخل روال Button1_Click تعریف می‌کردیم، هر بار که این روال به انتها می‌رسید و از نو شروع می‌شد، مقدار آن دوباره صفر شده، و نمی‌توانست تعداد واقعی بردها را در خود ذخیره کند. اما با تعریف یک متغیر عمومی این کار امکانپذیر خواهد بود. متغیرهای عمومی در واقع مثل یک کیلومترشمار در اتومبیل عمل می‌کنند.

متغیر عمومی یا متغیر فرم؟

در تمرین فوق برای ثبت تعداد بردها از یک متغیر عمومی استفاده کردیم. البته این کار را با استفاده از یک متغیر فرم (متغیری که در بالای فرم تعریف می‌شود، و از تمام روالهای آن فرم قابل دسترسی است) نیز می‌توانستیم انجام دهیم. مزیت متغیرهای عمومی اینست که اگر برنامه‌ای چندین فرم داشته باشد، آنها در تمام فرمهای پروژه (و بعبارت دیگر در فضای نام - namespace - آن) دیده می‌شوند. فضای نام یک پروژه در بدو ایجاد آن بوجود می‌آید. برای دیدن فضای نام پروژه (یا عوض کردن آن)، پنجرهٔ Project|Properties را باز کرده، و به قسمت Root Namespace نگاه کنید.

ایجاد روال‌های جدید

روال (procedure) مجموعه‌ایست از چند دستور برای انجام یک کار خاص. در ویژوال بیسیک .NET دو نوع روال وجود دارد: روال تابع (function) ، روال سابروتین (subroutine) . روال تابع (یا بطور مختصر، تابع) روالی است که می‌تواند یک یا چند آرگومان ورودی بگیرد، و همیشه مقداری را از طریق نام خود برمی‌گرداند. روال سابروتین (یا بطور مختصر، سابروتین) روالی است که می‌تواند یک یا چند آرگومان ورودی بگیرد، و مقداری را برگرداند، ولی برخلاف تابع هرگز این مقدار را از طریق نام خود برنمی‌گرداند، بلکه این کار را از طریق آرگومانهای خود انجام می‌دهد.

روالهای تابع و سابروتین را می‌توان در فرم برنامه هم نوشت، ولی برای بیشترین استفاده معمولاً آنها را در ماژول استاندارد می‌نویسند، تا از تمام نقاط پروژه قابل فراخوانی باشند. این رهیافت بویژه برای روالهای با کاربرد عمومی صحیح است، چون آنها باید بگونه‌ای نوشته شوند که در شرایط مختلف بتوانند کار خود را انجام دهند. روالهای با کاربرد عمومی زحمت کُدنویسی را کم کرده، و احتمال بروز خطا را کاهش می‌دهند.

مزایای روال‌های با کاربرد عمومی

مهمترین مزایای روالهای با کاربرد عمومی عبارتند از:

- دستوراتی را که بصورت مکرر استفاده می‌شوند، تحت یک نام گرد می‌آورند.
- مانع تکرار در کُدنویسی می‌شوند.
- خواندن برنامه را ساده‌تر می‌کنند. برنامه‌ای که کُد آن به روالهای مختلف تقسیم شده است، بسیار راحتتر فهمیده می‌شود.
- توسعهٔ برنامه را تسهیل می‌کنند. برنامه‌های مرکب از روالهای کوچک را براحتی می‌توان طراحی کرد، نوشت، و دیباگ کرد.
- از روالهای با کاربرد عمومی می‌توان در برنامه‌های مختلف استفاده کرد. روالهایی که برای یک برنامه نوشته شده‌اند، براحتی قابل استفاده در برنامه‌های دیگر هستند.
- آنها قابلیتهای زبان ویژوال بیسیک را توسعه می‌دهند. کارهایی که ویژوال بیسیک بصورت مستقل قادر به انجام آنها نیست، با این روالها براحتی امکانپذیر می‌شود.

نوشتن روال‌های تابع

یک تابع مجموعه‌ایست از چند دستور که بین Function و End Function قرار گرفته‌اند، و کار خاصی (مثلاً، محاسبات ریاضی) انجام می‌دهند. برای اجرا یا فراخوانی تابع (function call) کافیست نام آن را (بهمراه آرگومانهای ورودی مورد نیاز) در یک دستور بنویسیم. آرگومان (argument) داده‌ایست که تابع برای انجام کار خود به آن نیاز دارد. آرگومانهای یک تابع حتماً باید در داخل پرانتز قرار داشته باشند. طرز استفاده از توابع نوشته شده در برنامه هیچ تفاوتی با توابع داخلی ویژوال بیسیک (از قبیل Rnd ، Int یا FromFile) ندارد.

نکته

توابعی که در ماژول استاندارد نوشته می‌شوند، بصورت پیش‌فرض عمومی هستند، یعنی در تمام برنامه می‌توان آنها را فراخوانی کرد.

ساختار تابع

ساختار کلی یک تابع چنین است:

```
Function FunctionName([arguments]) As Type
    function statements
    [Return value]
End Function
```

که در آن

- *FunctionName* نام تابعیست که در حال نوشتن آن هستید.
- *Type* نوع دادهٔ مقدار برگشتی تابع است (و نوشتن آن در ویژوال بیسیک .NET اکیداً توصیه می‌شود).

- *arguments* لیستی اختیاری از آرگومانهای ورودی تابع است (و اگر بیش از یکی باشند، باید آنها را با کاما - , - از هم جدا کرد). نوع دادهٔ تک تک آرگومانها نیز باید مشخص باشد. وقتی کلمهٔ کلیدی ByVal جلوی یک آرگومان باشد (و این حالت پیش‌فرض در ویژوال بیسیک .NET است)، یعنی یک کپی از آن به تابع فرستاده می‌شود؛ تغییراتی که تابع در این آرگومان می‌دهد، مستقیماً به روال فراخوانی‌کنندهٔ تابع فرستاده نخواهد شد.
- *function statements* دستوراتیست که عملکرد اصلی تابع را انجام می‌دهند.
- *Return* (که جزء دستورات جدید ویژوال بیسیک .NET است) مشخص می‌کند که در کدام نقطه از تابع (و چه مقداری) باید برگشت داده شود. به محض رسیدن به دستور Return ، ویژوال بیسیک .NET اجرای تابع را خاتمه داده و به روال فراخوانی‌کننده برمی‌گردد؛ هر دستوری که بعد از Return قرار داشته باشد، اجرا نخواهد شد. (البته همچنان می‌توانید از روش ویژوال بیسیک ۶ - نسبت دادن مقدار برگشتی به نام تابع - نیز استفاده کنید.)

آنچه که در [ ] می‌بینید، جزء آیتمهای اختیاری است و می‌توان آنها را حذف کرد؛ اما سایر قسمتها حتماً باید حضور داشته باشند.

تابع همیشه مقداری به روال فراخوانی‌کننده (calling procedure) برمی گرداند، که جای *FunctionName* خواهد نشست. به همین دلیل، آخرین دستور یک تابع معمولاً مقدار محاسبه شده در تابع را به *FunctionName* نسبت می‌دهد. در مثال زیر، تابع TotalTax بعد از محاسبهٔ مقدار مالیات شهری و استانی، آنرا در نام تابع قرار می‌دهد (این تابع فقط یک آرگومان می‌گیرد):

```
Function TotalTax(ByVal Cost As Single) As Single
    Dim StateTax, CityTax As Single

    StateTax = Cost * 0.05        'State tax is 5%
    CityTax = Cost * 0.015        'City tax is 15%
    TotalTax = StateTax + CityTax
End Function
```

اگر بخواهید از روش جدید ویژوال بیسیک .NET (یعنی دستور Return) استفاده کنید، باید تابع فوق را بصورت زیر بنویسید:

```
Function TotalTax(ByVal Cost As Single) As Single
    Dim StateTax, CityTax As Single

    StateTax = Cost * 0.05        'State tax is 5%
    CityTax = Cost * 0.015        'City tax is 15%
    Return StateTax + CityTax
End Function
```

تا جایی که امکان داشته باشد، ما در این کتاب روش دوم را بکار خواهیم برد (ولی شما می‌توانید از هر روشی که بیشتر می‌پسندید، استفاده کنید).

فراخوانی یک تابع

برای فراخوانی تابع TotalTax می‌توان مانند دستور زیر عمل کرد:

```
lblTaxes.Text = TotalTax(500)
```

این دستور مقدار مالیات یک آیتم ۵۰۰ دلاری را محاسبه کرده، و در برچسب lblTaxes می‌نویسد. از این تابع بعنوان جزیی از یک دستور محاسباتی نیز می‌توان استفاده کرد:

```
Dim TotalCost, SalesPrice As Single
SalesPrice = 500
TotalCost = SalesPrice + TotalTax(SalesPrice)
```

در دستور آخر، تابع TotalTax مالیات متغیر SalesPrice را محاسبه کرده، و سپس با خود این مقدار جمع می‌کند، تا قیمت تمام شده (TotalCost) را بدست آورد.

تابع: ابزار محاسبه

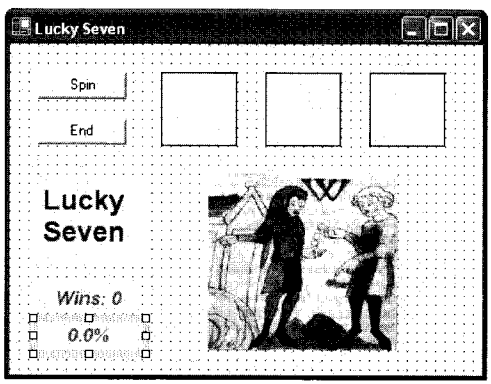
همانطور که گفتم، مهمترین کاربرد توابع انجام محاسبات است. در تمرین زیر، با استفاده از یک تابع درصد برد را در برنامهٔ Locky Seven محاسبه کرده، و نمایش می‌دهیم. این تابع که نام آنرا HitRate گذاشته‌ایم، در روال رویداد Button1_Click فراخوانی شده، و خروجی آن در یک برچسب جدید نشان داده می‌شود.

نوشتن تابع HitRate

۱ فرم برنامهٔ Locky Seven را باز کنید.

۲ یک برچسب جدید (زیر برچسب lblWins) ایجاد کرده، و خواص آنرا با توجه به جدول زیر ست کنید (شکل زیر را ببینید):

شیء	خاصیت	مقدار
Label5	Font	Arial, Bold Italic, 12-point
	ForeColor	Red
	Name	lblRate
	Text	"0.0%"
	TextAlign	MiddleCenter



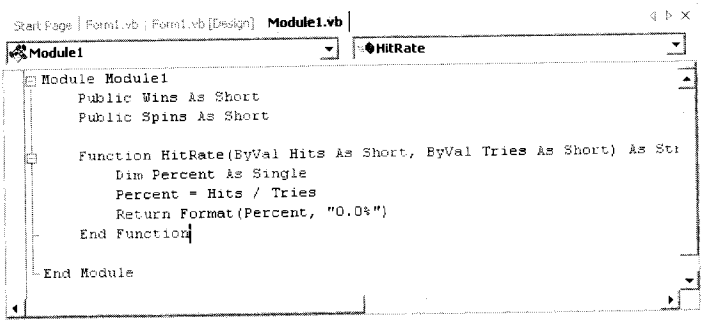
۳ ماژول Module1.vb را در کاوشگر راه‌حل انتخاب کرده، و سپس دکمهٔ View Code را کلیک کنید تا این ماژول در ادیتور کُد باز شود.

٤ متغیر Spins را در این ماژول تعریف کنید (از این متغیر برای ذخیره کردن تعداد دفعات انجام بازی استفاده خواهیم کرد):

```
Public Spins As Short
```

۵ (با زدن Enter) یک خط خالی در ماژول Module1.vb باز کرده، و پس بدنبال آن تابع HitRate را بصورت زیر ایجاد کنید (شکل زیر را ببینید):

```
Function HitRate(ByVal Hits As Short, ByVal Tries As Short) As String
    Dim Percent As Single

    Percent = Hits / Tries
    Return Format(Percent, "0.0%")
End Function
```



تابع HitRate برای محاسبهٔ درصد برد، تعداد دفعات برد (Hits) را بر تعداد دفعات بازی (Tries) تقسیم کرده، و نتیجهٔ حاصله را با تابع Format بصورت درصد فرمت می‌کند. تابع HitRate یک خروجی از نوع String برمی‌گرداند، بنابراین می‌توان آنرا مستقیماً در برچسب lblRate نوشت. این تابع دو آرگومان از نوع Short (عدد صحیح کوچک) می‌گیرد (که در داخل تابع آنها را با بنامهای Hits و Tries می‌شناسد).

۶ به فرم برنامه برگشته، و روی دکمهٔ Spin دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد باز شود.

۷ دستور زیر را بعد از دستور (Label3.Text = CStr(Int(Rnd() * 10))) (خط چهارم) وارد کنید:

```
Spins = Spins +1
```

این دستور با هر بار کلیک شدن دکمهٔ Spin مقدار متغیر عمومی Spins را یکی زیاد می‌کند.

۸ در آخرین خط روال Button1_Click (بین End If و End Sub) نیز دستور زیر را بنویسید:

```
lblRate.Text = HitRate(Wins, Spins)
```

توجه کنید که چگونه با نوشتن نام تابع HitRate ، ویژوال استودیو نام و نوع دادهٔ آرگومانهای آنرا نشان می‌دهد، تا احتمال اشتباه به حداقل برسد (ویژگی جالبی است، نه؟).

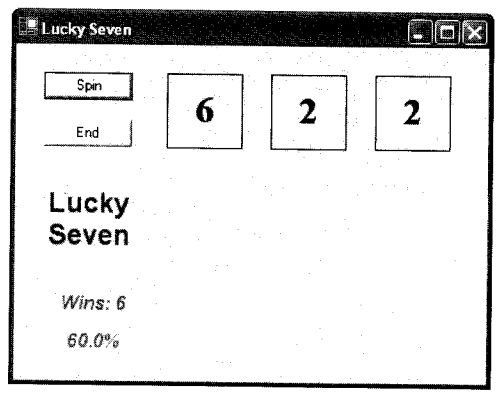
۹ با کلیک کردن دکمهٔ Save All ، پروژه را ذخیره کنید. (کُد کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap10\final track wins بیابید.)

حال نوبت اجرای برنامه است.

اجرای برنامهٔ Locky Seven

۱ برنامه را اجرا کلیک کنید.

۲ دکمهٔ Spin را ۱۰ بار کلیک کنید. شاید در ابتدای بازی درصد برد شما بسیار بالا باشد، اما با ادامهٔ کار این درصد بسرعت افت می‌کند، و در صورت ادامهٔ بازی به حدود ۲۸٪ می‌رسد.



۳ بعد از آن که باندازهٔ کافی برنامه را تست کردید، با کلیک کردن دکمهٔ End آنرا ببندید.

نکته

اگر می‌خواهید برنامه واقعاً اعداد شانسی تولید کند، دستور Randomize را در روال رویداد Form_Load قرار دهید (برای توضیح بیشتر، به فصل ۲ مراجعه کنید).

نوشتن روال‌های سابروتین

روال سابروتین بسیار شبیه تابع است، و تنها تفاوت آنها اینست که سابروتین مقدار برگشتی ندارد (بنابراین در تعریف آن لازم نیست نوع دادهٔ برگشتی مشخص شود). یک سابروتین هم می‌تواند تعدادی آرگومان (به شکل لیست آرگومانها) بگیرد. این موضوع که سابروتینها دارای مقدار برگشتی نیستند به معنای آن نیست که قادر به برقراری ارتباط دو طرفه با روال فراخوانی‌کنندهٔ خود نیستند؛ آنها می‌توانند این کار را از طریق نوع خاصی از آرگومانها انجام دهند.

ساختار سابروتین

ساختار کلی یک سابروتین چنین است:

```
Sub ProcedureName([arguments])
    procedure statements
End Sub
```

که در آن

- *ProcedureName* نام سابروتینی است که در حال نوشتن آن هستید.
- *arguments* لیستی اختیاری از آرگومانهای ورودی سابروتین است (و اگر بیش از یکی باشند، باید آنها را با کاما - , - از هم جدا کرد). نوع دادهٔ تک تک آرگومانها نیز باید مشخص باشد. وقتی کلمهٔ کلیدی ByVal جلوی یک آرگومان باشد (و این حالت پیش‌فرض در ویژوال بیسیک .NET است)، یعنی یک کپی از آن به تابع فرستاده می‌شود؛ تغییراتی که تابع در این آرگومان می‌دهد، مستقیماً به روال فراخوانی‌کنندهٔ تابع فرستاده نخواهد شد.
- *procedure statements* دستوراتیست که عملکرد اصلی سابروتین را انجام می‌دهند.

هنگام فراخوانی یک سابروتین تعداد، ترتیب و نوع دادهٔ آرگومانها بایستی دقیقاً منطبق با لیست آرگومانها باشد (و داخل یک جفت پرانتز قرار گیرند). اگر سابروتین مقدار یکی از این آرگومانها را تغییر دهد، روال فراخوانی‌کننده قادر به تشخیص این موضوع نخواهد بود، مگر اینکه آن آرگومان بصورت ByRef (ارسال آدرس) تعریف شده باشد. سابروتینهایی که در یک ماژول استاندارد تعریف می‌شوند، از تمام برنامه قابل فراخوانی هستند.

بسیار مهم

در ویژوال بیسیک .NET (برخلاف ویرایشهای قبلی ویژوال بیسیک) پرانتزهای روال سابروتین حتماً باید نوشته شود - حتی اگر سابروتین هیچ آرگومانی نگیرد. در ویرایشهای قبلی ویژوال بیسیک، نوشتن پرانتزها فقط زمانی که آرگومانها بصورت ByVal ارسال می‌شوند، لازم است.

در مثال زیر، سابروتینی بنام BirthdayGreeting می‌بینید که نام فرد را (بصورت آرگومان ورودی) گرفته، و پیام تبریک تولد وی را نشان می‌دهد.

```
Sub BirthdayGreeting(ByVal Person As String)
    Dim Msg As String

    If Person <> "" Then
        Msg = "Happy birthday " & Person & "!"
    Else
        Msg = "Name not specified."
    End If
    MsgBox(Msg, , "Best Wishes")
End Sub
```

همانطور که می‌بینید، این سابروتین بسته به مقدار آرگومان یک پیام نمایش می‌دهد، و هیچ مقداری برنمی‌گرداند.

فراخوانی یک سابروتین

برای فراخوانی سابروتین BirthdayGreeting می‌توان مانند دستور زیر عمل کرد:

```
BirthdayGreeting("Robert")
```


وقتی این روال را اجرا کنید، پیامی مانند زیر خواهید دید:

شاید این روش برای نمایش یک پیام ساده کمی پیچیده بنظر برسد (و در واقع، همینطور هم هست!). اما وقتی قرار باشد، همین پیام را برای دهها و صدها نفر بنویسید، آنوقت به مزیت نوشتن سابروتین بیشتر پی خواهید برد. در حلقهٔ Do...Until زیر از سابروتین BirthdayGreeting استفاده کرده‌ایم، و در آن براحتی متوجه مزیت نوشتن سابروتین می‌شوید:

```
Dim NewName As String
Do
    NewName = InputBox("Enter a name for greeting.", "Birthday List")
    BirthdayGreeting(NewName)
Loop Until NewName = ""
```

در این حلقه کاربر می‌تواند دهها و صدها نام وارد کرده، و پیام تبریک هر کدام را مشاهده کند. در تمرین زیر نیز برای پردازش ورودی از یک سابروتین استفاده خواهیم کرد.

سابروتین: ابزار پردازش ورودی

همانطور که گفتم، مهمترین کاربرد سابروتین پردازش ورودی‌هاست. در تمرین زیر، با استفاده از یک سابروتین بنام AddName اطلاعاتی را از کاربر گرفته، و (پس از فرمت کردن) در دو جعبه متن نمایش می‌دهیم. (میزان صرفه‌جویی در کُدنویسی با این سابروتین بسیار قابل ملاحظه خواهد بود.)

نوشتن سابروتین AddName

۱ با اجرای فرمان File|Close Solution ، برنامهٔ Locky Seven را ببندید.

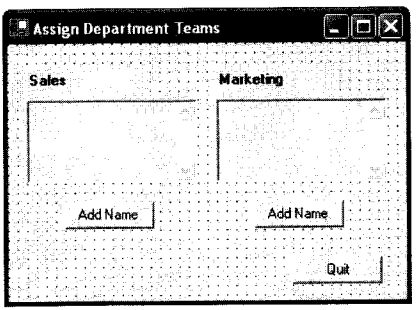
۲ یک پروژهٔ جدید Visual Basic Windows Application بنام **My Text Box** در پوشهٔ c:\vbnetsbs\chap10 ایجاد کنید.

۳ روی فرم این برنامه به دو جعبه متن، دو برچسب، و سه دکمه نیاز داریم، که مشخصات آنها را در جدول زیر ملاحظه می‌کنید:

شیء	خاصیت	مقدار
Form	Text	"Assign Department Teams"
Button1	Name	btnSales
	Text	"Add Name"
Button2	Name	btnMkt
	Text	"Add Name"
Button3	Name	btnQuit
	Text	"Quit"
Label1	Font	Bold
	Name	lblSales
	Text	"Sales"
Label2	Font	Bold
	Name	lblMarketing
	Text	"Marketing"
TextBox1	Multiline	True
	Name	txtSales
	ReadOnly	True
	Scrollbars	Vertical
	TabStop	False
	Text	(خالی)
TextBox2	Multiline	True
	Name	txtMkt
	ReadOnly	True
	Scrollbars	Vertical
	TabStop	False
	Text	(خالی)

خاصیت Multiline جعبه متن‌ها به True ست شده، چون می‌خواهیم چندین اسم را در آنها بنویسیم؛ خواص ReadOnly و TabStop آنها را نیز False کرده‌ایم، تا کاربر نتواند مستقیماً آنها را دستکاری کند.

٤ برای تنظیم دقیق فرم می‌توانید از شکل زیر کمک بگیرید:



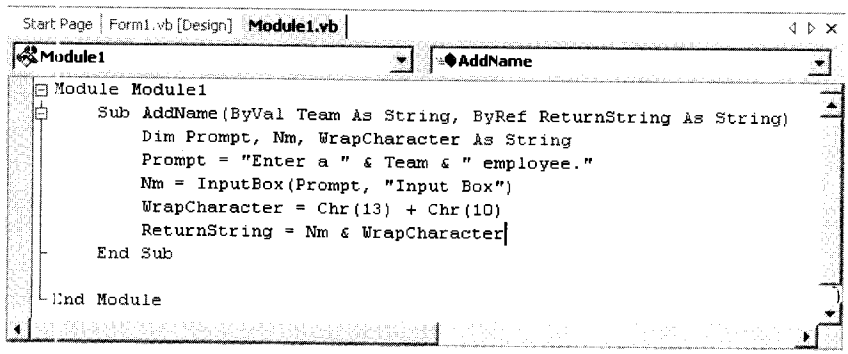
برای نوشتن سابروتین AddName ، ابتدا باید یک ماژول استاندارد به برنامه اضافه کنیم.

۵ فرمان Project|Add New Item را اجرا کرده، و بعد از انتخاب آیتم Module دکمهٔ Open را کلیک کنید، تا ماژول Module1.vb ایجاد شده و در ادیتور کُد باز شود.

۶ سابروتین AddName را بصورت زیر (بین Module Module1 و End Module) ایجاد کنید (شکل زیر را ببینید):

```
Sub AddName(ByVal Team As String, ByRef ReturnString As String)
    Dim Prompt, Nm, WrapCharacter As String
    Prompt = "Enter a " & Team & " employee."
    Nm = InputBox(Prompt, "Input Box")
    WrapCharacter = Chr(13) + Chr(10)
    ReturnString = Nm & WrapCharacter
End Sub
```


سابروتین AddName دو آرگومان ورودی می‌گیرد: آرگومان اول بنام Team که از نوع String است و نام اداره را در خود دارد؛ آرگومان دوم بنام ReturnString که آن هم یک متغیر خالی از نوع String است، و ورودی کاربر را پس از اضافه کردن دو کاراکتر Chr(10) و Chr(13) -که در مجموع معادل Enter هستند - برمی‌گرداند. توجه کنید که چگونه برای برگرداندن خروجی سابروتین از کلمهٔ کلیدی ByRef استفاده کرده‌ایم. وقتی این رشته‌ها به جعبه متن اضافه می‌شوند، در خطوط جداگانه و پشت سرهم قرار خواهند گرفت. (در این سابروتین برای گرفتن نام کارمند از تابع InputBox استفاده کرده‌ایم.)



۷ به فرم برنامه برگردید، و با دو-کلیک روی دکمهٔ Add Name (زیر جعبه متن Sales) روال رویداد btnSales_Click را باز کرده و دستورات زیر را در آن بنویسید:

```
Dim SalesPosition As String
AddName("Sales", SalesPosition)
txtSales.Text = txtSales.Text & SalesPosition
```

این روال سابروتین AddName را با آرگومانهای "Sales" و متغیر SalesPosition (که آدرس آن به سابروتین فرستاده می‌شود) فراخوانی می‌کند. وقتی سابروتین AddName خروجی خود را (که

شامل یک نام جدید بعلاوهٔ یک Enter در انتهای آن است) در متغیر SalesPosition نوشت، روال btnSales_Click آنرا به جعبه متن txtSales اضافه می‌کند.

۸ در ادیتور کُد، لیست Class Name را باز کرده و شیء btnMkt را انتخاب کنید؛ از لیست Method Name هم رویداد Click را انتخاب کنید. با اینکار روال رویداد btnMkt_Click باز خواهد شد.

۹ دستورات زیر را در روال رویداد btnMkt_Click بنویسید:

```
Dim MktPosition As String
AddName("Marketing", MktPosition)
txtMkt.Text = txtMkt.Text & MktPosition
```

این روال همان کار btnSales_Click را (این بار با آرگومانهای "Marketing" و متغیر MktPosition) انجام می‌دهد.

۱۰ در ادیتور کُد، لیست Class Name را باز کرده و شیء btnQuit را انتخاب کنید؛ از لیست Method Name هم رویداد Click را انتخاب کنید. با اینکار روال رویداد btnQuit_Click باز خواهد شد.

۱۱ دستور **End** را در روال رویداد btnQuit_Click بنویسید.

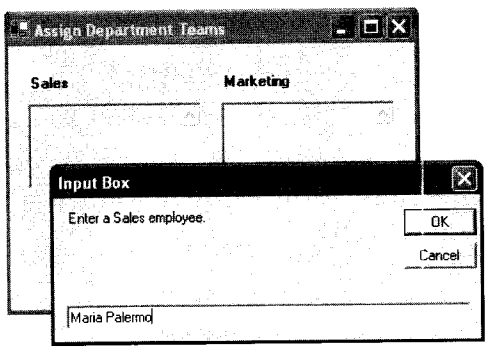
۱۲ با کلیک کردن دکمهٔ Save All ، پروژه را ذخیره کنید. (کُد کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap10\text box پیدا کنید.)

حالا باید برنامه را اجرا کنیم.

اجرای برنامهٔ My Text Box

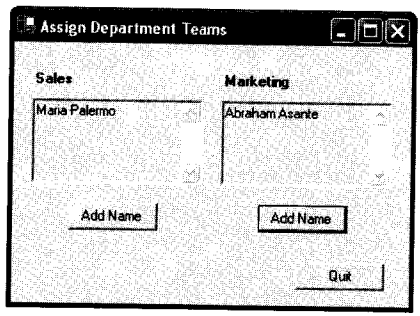
۱ با کلیک کردن دکمهٔ Start ، برنامه را اجرا کلیک کنید.

۲ دکمهٔ Add Name (زیر جعبه متن Sales) را کلیک کرده، و نام **Maria Palermo** (یا هر نام دیگری که میل دارید) را در پنجرهٔ InputBox وارد کنید (شکل زیر را ببینید).



۳ OK را کلیک کنید، تا این نام به جعبه متن Sales اضافه شود.

۴ دکمهٔ Add Name (زیر جعبه متن Marketing) را کلیک کرده، و پس از وارد کردن نام **Abraham Asante** (یا هر نام دیگری که می‌خواهید) در پنجرهٔ InputBox ، OK را کلیک کنید تا این نام هم به جعبه‌متن Marketing اضافه شود:



۵ چند نام دیگر به جعبه‌متن‌های Sales و Marketing اضافه کنید.

۶ در آخر هم با کلیک کردن دکمهٔ Quit ، برنامه را ببندید.

یک گام فراتر: ارسال آرگومان بصورت ByVal و ByRef

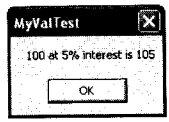
در بحث آرگومانهای تابع و سابروتین، دیدید که یک آرگومان را می‌توان با مقدار (ByVal) یا با آدرس (ByRef) به روال فرستاد. در روش ByVal (ارسال مقدار -که روش پیش‌فرض ویژوال بیسیک .NET است) یک کپی از مقدار متغیر به روال فرستاده می‌شود. در این روش تغییراتی که تابع یا سابروتین روی این متغیر می‌دهد، در روال فراخوانی‌کننده (و نسخهٔ اصلی متغیر) دیده نخواهد شد. اما در روش ByRef (ارسال آدرس) بجای یک کپی از متغیر، آدرس آن به تابع یا سابروتین فرستاده می‌شود؛ هر تغییری که در این متغیر داده شود، مستقیماً به روال فراخوانی‌کننده منعکس خواهد شد. روش ByRef مزایای زیادی دارد، مشروط بر اینکه با دقت و احتیاط از آن استفاده شود. عدم استفادهٔ صحیح از این روش می‌تواند منجر به نتایج عجیب و غریبی شود. مثال زیر را در نظر بگیرید:

```
Sub CostPlusInterest(ByRef Cost As Single, ByRef Total As Single)
    Cost = Cost * 1.05          'add 5% to cost
    Total = Int(Cost)           'then make integer and return
End Sub
.
.
.
Dim Price, TotalPrice As Single
Price = 100
TotalPrice = 0
CostPlusInterest(Price, TotalPrice)
MsgBox(Price & " at 5% interest is " & TotalPrice)
```

در این مثال، برنامه‌نویس دو متغیر از نوع Single (بنامهای Price و TotalPrice) را با روش ByRef به

سابروتین CostPlusInterest می‌فرستد. برنامه‌نویس ما می‌خواهد بعد از فراخوانی این روال، مقدار Price و TotalPrice را چاپ کند، غافل از اینکه متغیر Price بصورت ByRef ارسال شده، و در داخل روال CostPlusInterest تغییر کرده است (دستور Cost = Cost * 1.05)، و دیگر آن مقدار اصلی (در اینجا، 100) را ندارد. خروجی این برنامه شبیه شکل زیر خواهد بود:

در حالیکه وی (احتمالاً) انتظار چیزی شبیه این را داشته:



کدام روش: ByVal یا ByRef ؟

مشکل فوق را چگونه می‌توان برطرف کرد؟ ساده‌ترین راه اینست که آرگومان Cost را بصورت ByVal به سابروتین CostPlusInterest بفرستیم:

```
Sub CostPlusInterest(ByVal Cost As Single, ByRef Total As Single)
```

با این روش متغیر Cost از تغییرات ناخواسته در سابروتین CostPlusInterest مصون خواهد بود (ضمن آن که متغیر Total هنوز ByRef است، و سابروتین ما می‌تواند مقدار محاسبه شده را به روال فراخوانی‌کننده برگرداند).

در اینجا چند قاعدهٔ کلی برای استفاده از آرگومانهای ByVal و ByRef می‌بینید:

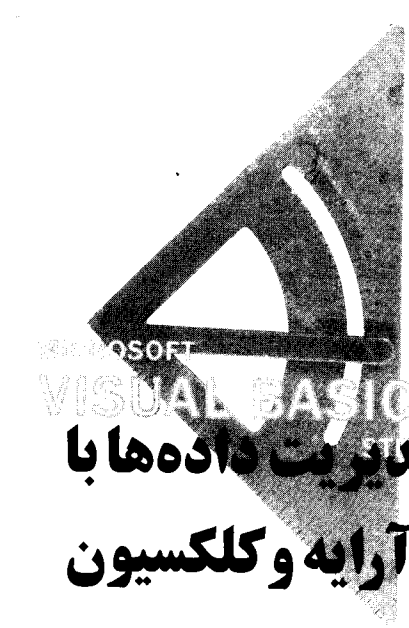
- اگر نمی‌خواهید تابع یا سابروتین متغیرهای ارسال شده را تغییر دهد، از آرگومانهای ByVal استفاده کنید.
- اگر می‌خواهید به تابع یا سابروتین اجازه دهید تا متغیرهای ارسال شده را تغییر دهد، از آرگومانهای ByRef استفاده کنید.
- هر جا شک داشتید، از آرگومانهای ByVal استفاده کنید.

مرجع سریع فصل ۱۰

برای ...	انجام دهید
ایجاد یک ماژول جدید	پس از اجرای فرمان Project|Add New Item ، آیتم Module را انتخاب کنید.
ذخیره کردن ماژول با نام جدید	پس از انتخاب ماژول موردنظر در کاوشگر راه‌حل، فرمان File|Save Module1.vb As را اجرا کرده، و نام جدید را وارد کنید.
حذف یک ماژول از برنامه	پس از انتخاب ماژول موردنظر در کاوشگر راه‌حل، فرمان File|Exclude From Project را اجرا کنید.
اضافه کردن یک ماژول به برنامه	فرمان Project|Add Existing Item را اجرا کنید.
ایجاد یک متغیر عمومی	در ماژول استاندارد (بین Module و End Module)، متغیر موردنظر را با استفاده از کلمهٔ کلیدی Public تعریف کنید: `Public TotalSales As Integer`
ایجاد یک تابع عمومی	تابع موردنظر را در ماژول استاندارد (بین Module و End Module) قرار دهید (این تابع بصورت پیش‌فرض عمومی خواهد بود).
فراخوانی یک تابع	نام تابع را (با آرگومانهای لازم) در دستور موردنظر نوشته، و مقدار برگشتی آنرا به یک متغیر نسبت دهید: `lblRate.Text = HitRate(Wins, Spins)`
ایجاد یک سابروتین عمومی	سابروتین موردنظر را در ماژول استاندارد (بین Module و End Module) قرار دهید (این سابروتین بصورت پیش‌فرض عمومی خواهد بود).
فراخوانی یک سابروتین	نام سابروتین را (با آرگومانهای لازم) در دستور موردنظر بنویسید: `CostPlusInterest(Price, TotalPrice)`
ارسال آرگومان با مقدار	در تعریف روال، از کلمهٔ کلیدی ByVal استفاده کنید: `Sub GreetPerson(ByVal Name As String)`
ارسال آرگومان با آدرس	در تعریف روال، از کلمهٔ کلیدی ByRef استفاده کنید: `Sub GreetPerson(ByRef Name As String)`

فصل

۱۱



مدیریت داده‌ها با آرایه و کلکسیون

در این فصل یاد می گیرید چگونه :

- ✔ برای سازماندهی اطلاعات خود از آرایه‌های طول‌ثابت و دینامیک استفاده کنید.
- ✔ هنگام تغییر بُعد یک آرایه، اطلاعات موجود در آن را حفظ کنید.
- ✔ با کلکسیون Controls در فرم کار کنید.
- ✔ برای کار با اشیاء یک کلکسیون، از حلقهٔ For Each...Next استفاده کنید.
- ✔ برای مدیریت اطلاعات خود، کلکسیون ایجاد کنید.

جمع‌آوری، ذخیره‌سازی و پردازش داده‌ها یکی از مهمترین وظایفی است که هر زبان برنامه‌نویسی (از جمله ویژوال بیسیک .NET) باید بتواند بخوبی از عهدهٔ آن برآید - و این کار وقتی حجم داده‌ها افزایش می‌یابد، از اهمیت بیشتری برخوردار می‌شود. در این فصل خواهید دید که چگونه می‌توان با استفاده از آرایه (array) اطلاعات را سازماندهی کرد. آرایه ابزاریست برای مدیریت اطلاعات، و یکی از مفاهیم پایه در برنامه‌نویسی پایگاه داده محسوب می‌شود. یکی دیگر از ابزارهای ویژوال بیسیک .NET برای مدیریت مؤثرتر اشیاء برنامه، کلکسیون (collection) نام دارد. کلکسیون مجوعه‌ایست از چند شیء که تحت یک نام قابل دسترسی هستند. آرایه و کلکسیون دو ابزاری هستند که در کنار هم می‌توانند برای مدیریت اطلاعات (در حجمهای بالا) بنحو مؤثری به ما کمک کنند.

تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- آرایه‌ها در ویژوال بیسیک .NET از صفر شروع می‌شوند. در ویرایشهای قبلی ویژوال بیسیک این امکان وجود داشت که با دستور Option Base اندیس شروع آرایه را خود تعیین کنید: ویژوال بیسیک .NET دیگر از Option Base پشتیبانی نمی‌کند.
- در ویرایشهای قبلی ویژوال بیسیک هنگام تعریف یک آرایه، می‌توانستید با کلمهٔ کلیدی To کران پائین و بالای آرایه را مشخص کنید؛ در ویژوال بیسیک .NET کران پائین یک آرایه همیشه 0 است (بعبارت دیگر، تابع LBound همیشه مقدار 0 برمی‌گرداند). بدین ترتیب، مقداری که تابع UBound برمی‌گرداند، همواره عبارتست از تعداد کل عناصر آرایه منهای یک.
- در ویژوال بیسیک .NET می‌توانیم در همان دستوری که آرایه را تعریف می‌کنیم، به آن مقدار هم بدهیم. بعبارت دیگر، دستور زیر کاملاً صحیح است:

```
Dim myList() As Integer = {5, 10, 15, 20}
```

- ویژوال بیسیک .NET همچنان از دستور ReDim پشتیبانی می‌کند، اما با آن نمی‌توان تعداد ابعاد یک آرایهٔ موجود را تغییر داد. همچنین نمی‌توان از ReDim برای تعریف آرایه‌های جدید استفاده کرد.
- نوع دادهٔ Collection دیگر در ویژوال بیسیک .NET وجود ندارد؛ برای ایجاد کلکسیون بجای آن باید از کتابخانهٔ System.Collections استفاده کنید. انواع کلکسیونهای موجود در این کتابخانه عبارتند از: Stack ، Queue ، Dictionary ، و Hashtable .
- ویژوال بیسیک .NET دیگر از آرایهٔ کنترل (مجموعه‌ای از چند کنترل که یک نام مشترک دارند) پشتیبانی نمی‌کند. اگر به چیزی شبیه آن نیاز دارید، می‌توانید یک آرایه از نوع Object تعریف کرده، و کنترلها را در آن قرار دهید.

آرایه‌ای از متغیرها

یکی از مفاهیمی که از اولین زبانهای برنامه‌نویسی (مانند بیسیک، پاسکال و سی) شکل گرفت و پا به پای تکامل آنها پیش آمد، آرایه است. آرایه وسیله‌ایست برای ذخیره کردن حجم زیادی از اطلاعات تحت یک نام واحد (کاری که انجام آن با متغیرهای ساده تقریباً غیرممکن است).

برای مثال، فرض کنید می‌خواهیم اطلاعات مربوط به نمرات امتحانی یک کلاس ۳۰ نفره را در برنامهٔ خود پردازش کنیم. اگر فقط دو امتحان داشته باشیم، برای ذخیره کردن نمرات دانش‌آموزان به ۶۰ متغیر ساده نیاز داریم (که اسم آنها احتمالاً Student1Test1 ، Student1Test2 ، Student2Test1 ، Student2Test2 – و مانند آن – خواهد بود). کار با چنین متغیرهایی بسیار دشوار، وقت‌گیر و بسیار مستعد بروز خطا خواهد بود. خوشبختانه، ویژوال بیسیک راه ساده‌تری برای این کار دارد: آرایه. یک آرایهٔ ۳۰×۲ همان کار ۶۰ متغیر ساده را می‌کند، ولی بسیار ساده‌تر و راحتتر. اجازه دهید ببینیم، چطور.

ایجاد یک آرایه

تعریف (یا ایجاد) آرایه بسیار شبیه تعریف متغیرهاست. مانند متغیرها، میدان دید یک آرایه به جایی که آنرا تعریف می‌کنید، بستگی دارد. اگر یک آرایه در داخل روال تعریف شده باشد، فقط در همان روال دیده می‌شود، اگر در بالای فرم تعریف شود، در تمام روالهای آن فرم دیده می‌شود، و اگر در یک ماژول استاندارد آنرا تعریف کنید، در تمام برنامه دیده خواهد شد. معمولاً در تعریف یک آرایه اطلاعات زیر باید وجود داشته باشد.

اطلاعات موجود در دستور تعریف آرایه	مفهوم
نام آرایه	هر آرایه، مانند متغیرها، باید یک نام داشته باشد. قواعد نامگذاری آرایه‌ها همان قواعد نامگذاری متغیرهاست (فصل ۵ را ببینید).
نوع داده	نوع داده‌ای که در هر آرایه ذخیره می‌شود، باید مشخص باشد. معمولاً تمام عناصر یک آرایه از یک نوع هستند. اگر نمی‌دانید داده‌های شما از چه نوعی هستند، و یا می‌خواهید انواع مختلفی از داده‌ها را در یک آرایه ذخیره کنید، آنرااز نوع Object تعریف کنید.
تعداد بُعدهای آرایه	یک آرایه می‌تواند دارای چندین بُعد باشد. آرایه‌ها معمولاً یک بُعدی (لیست)، دو بُعدی (جدول) و یا سه بُعدی هستند.
تعداد عناصر آرایه	هنگام تعریف یک آرایه، معمولاً تعداد عناصر آن هم مشخص می‌شود.

نکته

به آرایه‌ای که تعداد عناصر آن از قبل مشخص باشد، آرایهٔ طول-ثابت (fixed-size array) می‌گویند. اگر تعداد عناصر یک آرایه از قبل معلوم نباشد (و این تعداد بتواند در طول اجرای برنامه تغییر کند)، به آن آرایهٔ دینامیک (dynamic array) گفته می‌شود.

تعریف یک آرایهٔ طول-ثابت

دستور کلی تعریف یک آرایهٔ طول-ثابت عمومی چنین است:

```
Dim ArrayName(Dim1Index, Dim2Index, ...) As DataType
```

که در آن

- Dim کلمهٔ کلیدی تعریف آرایه است (اگر این تعریف در یک ماژول استاندارد است، می‌توانید بجای Dim از Public استفاده کنید).
- *ArrayName* نام آرایه است.
- *Dim1Index* کران بالای بُعد اول آرایه (تعداد عناصر این بُعد منهای یک) است.
- *Dim2Index* کران بالای بُعد دوم آرایه (تعداد عناصر این بُعد منهای یک) است.
- *DataType* نوع دادهٔ عناصر آرایه است.

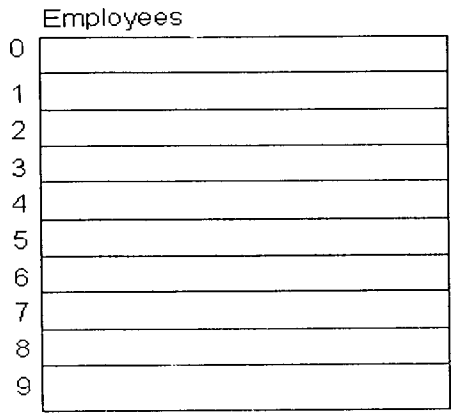
برای مثال، دستور زیر آرایه‌ای یک بُعدی بنام Employees تعریف می‌کند، که می‌توان در آن ۱۰ متغیر از نوع String (از شمارهٔ 0 تا 9) ذخیره کرد:

```
Dim Employees(9) As String
```

اگر همین دستور در یک ماژول استاندارد باشد، باید آنرا چنین نوشت:

```
Public Employees(9) As String
```

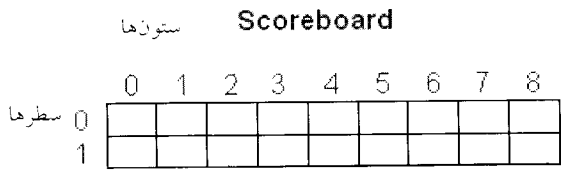
وقتی یک آرایه را تعریف می‌کنید، ویژوال بیسیک حافظهٔ مورد نیاز آنرا در حافظه کنار می‌گذارد (تخصیص می‌دهد). آرایهٔ فوق ۱۰ عنصر دارد، که از 0 تا 9 تا شماره گذاری می‌شوند. اندیس اولین عنصر آرایه همیشه 0 است. (ویژوال بیسیک .NET دیگر از دستور Option Base که امکان می‌داد تا آرایه بجای 0 از 1 شروع شود، پشتیبانی نمی‌کند). شکل زیر آرایهٔ Employees را نشان می‌دهد:



در دستور زیر یک آرایهٔ دوبُعدی بنام Scoreboard (با ۲ سطر و ۹ ستون) تعریف شده است:

```
Dim Scoreboard(1, 8) As Short
```

وقتی این آرایه را تعریف می‌کنید، ویژوال بیسیک فضای لازم را در حافظه به آن تخصیص می‌دهد (شکل زیر را ببینید). یک آرایهٔ دوبُعدی در واقع یک جدول است، که برای کار با آن به دو اندیس (اندیس سطر و اندیس ستون) نیاز داریم.

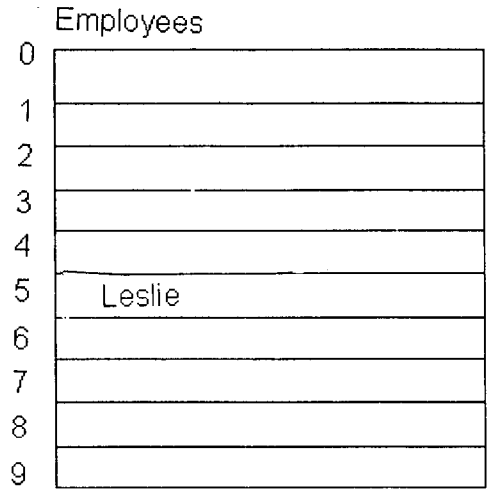


کار با عناصر آرایه

بعد از آن که با دستور Dim یا Public آرایه را تعریف و ایجاد کردید، می‌توانید کار با آنرا شروع کنید. برای دسترسی به هر یک از عناصر آرایه، از نام آرایه و اندیس (index) آن عنصر استفاده می‌کنیم. اندیس یک عدد صحیح (یا هر عبارتی که نتیجهٔ آن یک عدد صحیح باشد) است. در دستور زیر می‌بینید که چگونه اسم "Leslie" را در عنصر ششم آرایهٔ Employees (عنصری که اندیس آن 5 است) نوشته‌ایم:

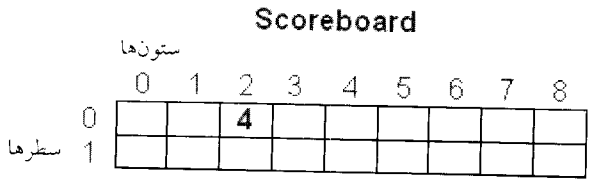
```
Employees(5) = "Leslie"
```

شکل زیر حاصل این دستور را نشان می‌دهد:



تکنیک استفاده از اندیس برای خواندن و نوشتن آرایه‌ها (هر دو) کاربرد دارد. دستور زیر عدد 4 را در سطر اول-ستون سوم جدول (آرایهٔ دوبُعدی) Scoreboard می‌نویسد (به شکل زیر نگاه کنید):

```
Scoreboard(0, 2) = 4
```



ایجاد آرایه‌های طول-ثابت

در تمرین این قسمت برای ذخیره کردن حداکثر دمای روزهای هفته از یک آرایهٔ یک بُعدی بنام Temperatures استفاده خواهیم کرد. در این تمرین خواهید دید که چگونه می‌توان اعداد را در آرایه ذخیره، و سپس آنها را پردازش کرد. برای پُر کردن آرایه از یک حلقهٔ For...Next و تابع InputBox استفاده کرده‌ایم. پس از نمایش این اعداد روی فرم برنامه، برنامه متوسط دمای هفته را نیز محاسبه کرده و نمایش می‌دهد.

توابع LBound و UBound

یکی از توابعی که در کار با آرایه‌ها بسیار کاربرد دارد، و در تمام ویرایشهای ویژوال بیسیک وجود داشته، تابع UBound است. این تابع بالاترین اندیس آرایه را برمی‌گرداند، و شما را از مراجعه به تعریف آرایه بی نیاز می‌کند. تابع دیگری که همین کار را در انتهای دیگر آرایه انجام می‌دهد، تابع LBound است؛ این تابع کوچکترین اندیس آرایه را برمی‌گرداند. همانطور که گفتم، کران پائین آرایه‌ها در ویژوال بیسیک .NET (مقداری که تابع LBound برمی‌گرداند) همیشه 0 است، ولی برای حفظ سازگاری با ویرایشهای قبلی، ویژوال بیسیک .NET همچنان از این تابع پشتیبانی می‌کند. دستورات زیر، بترتیب، کران بالا و پائین آرایهٔ *ArrayName* را برمی‌گردانند:

```
UBound(ArrayName)
LBound(ArrayName)
```

کار با یک آرایهٔ طول-ثابت

۱ ویژوال استودیو .NET را باز کرده، و یک پروژهٔ Visual Basic Windows Application بنام **My Fixed Array** در پوشهٔ c:\vbnetsbs\chap11 ایجاد کنید.

۲ یک جعبه متن روی فرم قرار دهید.

۳ خاصیت Multiline جعبه متن را True کنید.

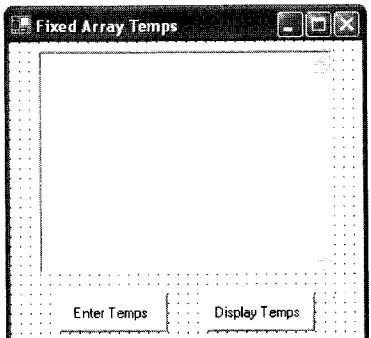
٤ اندازهٔ آنرا آنقدر بزرگ کنید، که تقریبا تمام فرم را بپوشاند.

۵ دو دکمه کنار هم و زیر جعبه متن رسم کنید.

۶ خواص فرم و کنترلهای روی آنرا با توجه به جدول زیر ست کنید:

شیء	خاصیت	مقدار
Form1	Text	"Fixed Array Temps"
TextBox1	Multiline	True
	ScrollBars	Vertical
	Text	(خالی)
Button1	Text	"Enter Temps"
Button2	Text	"Display Temps"

شکل زیر فرم برنامه را تا اینجا نشان می‌دهد:



۷ با کلیک کردن دکمهٔ View Code در کاوشگر راه‌حل، ادیتور کُد را باز کنید.

۸ در بالای این پنجره و درست زیر جملهٔ "Windows Form Designer generated code" ، دستور زیر را وارد کنید:

```
Dim Temperatures(6) As Single
```

این دستور یک آرایهٔ هفت عنصری (از 0 تا 6) از اعداد اعشاری بنام Temperatures تعریف می‌کند. این آرایه در تمام فرم برنامه دیده می‌شود، چون در بالای فرم (و خارج از تمام روالها) تعریف شده است.

۹ به فرم برنامه برگردید، و با دو-کلیک روی دکمهٔ Enter Temps روال Button1_Click را باز کنید.

۱۰ دستورات زیر را در این روال بنویسید:

```
Dim Prompt, Title As String
Dim i As Short
Prompt = "Enter the day's high temperature."
For i = 0 To UBound(Temperatures)
    Title = "Day " & (i + 1)
    Temperatures(i) = InputBox(Prompt, Title)
Next
```

در این کُد حلقهٔ For...Next بکمک متغیر i (که از آن بعنوان اندیس آرایه استفاده کرده‌ایم) و تابع InputBox ، دمای روزهای هفته را گرفته و در آرایهٔ Temperatures ذخیره می‌کند.

با اینکه از قبل می‌دانیم آرایهٔ Temperatures فقط هفت عنصر (تا اندیس 6) دارد، اما برای تعیین حد بالای حلقهٔ For...Next از تابع UBound() استفاده کرده‌ایم. این یک روش کلی است، و اگر بعدها تعداد عناصر آرایه تغییر کرد، حلقهٔ For...Next بدون مشکل کار خود را انجام خواهد داد.

۱۱ به فرم برنامه برگردید، و با دو-کلیک روی دکمهٔ Display Temps روال Button2_Click را باز کنید.

۱۲ دستورات زیر را در این روال بنویسید:

```
Dim Result As String
Dim i As Short
Dim Total As Single = 0
Result = "High temperatures for the week:" & vbCrLf & vbCrLf
For i = 0 To UBound(Temperatures)
    Result = Result & "Day " & (i + 1) & vbTab & Temperatures(i) & vbCrLf
    Total = Total + Temperatures(i)
Next
Result = Result & vbCrLf & "Average temperature:  " & Format(Total / 7, "0.0")
TextBox1.Text = Result
```

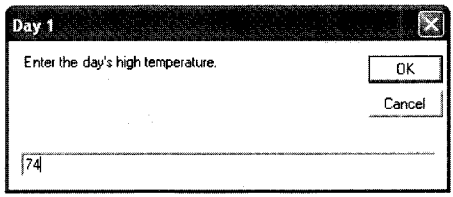
در این روال هم برای خواندن آرایه از یک حلقهٔ For...Next استفاده کرده‌ایم. قطعات عبارت نهایی

(متغیر Result) را با عملگر & و ثابت vbCrLf (که معادل زدن Enter است) به هم چسبانده‌ایم. جدا کردن روز هفته و دمای آن روز هم با اضافه کردن ثابت vbTab (که معادل زدن کلید Tab است) انجام شده است. در پایان هم، بعد از محاسبهٔ متوسط دمای هفته و اضافه کردن آن به متغیر Result ، این عبارت را در جعبه متن TextBox1 نوشته‌ایم.

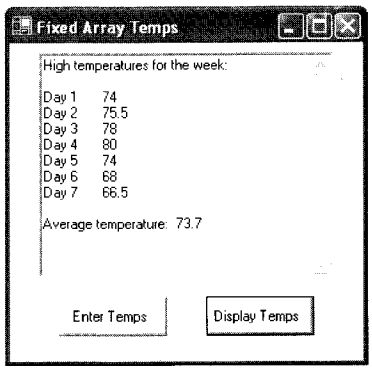
۱۳ برنامه را ذخیره کنید (متن کامل این برنامه را می‌توانید در پوشهٔ c:\vbnetsbs\chap11\fixed array پیدا کنید).

۱٤ برنامه را اجرا کنید.

۱۵ روی دکمهٔ Enter Temps کلیک کرده، و هفت دمای مختلف (برای هفت روز هفته) در پنجرهٔ InputBox وارد کنید؛ شکل زیر را ببینید:



۱۶ بعد از وارد کردن دماها، روی دکمهٔ Display Temps کلیک کنید. در شکل زیر اجرای روال Button2_Click را می‌بینید:



۱۷ با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

ایجاد آرایه‌های دینامیک

همانطور که دیدید آرایه ابزاری مناسب برای کار با اعداد و عبارت‌هاست. اما گاهی پیش می‌آید که از قبل نمی‌دانیم دقیقاً با چند عدد یا عبارت سروکار داریم. برای این موارد چه آرایه‌ای باید تعریف کنیم؟ مثلاً، اگر بخواهیم در برنامه قبل تصمیم‌گیری دربارهٔ تعداد دماهای ورودی را بر عهدهٔ کاربر بگذاریم، آرایهٔ

Temperatures را چگونه باید تعریف کنیم؟

ویژوال بیسیک برای حل این مشکل ابزاری انعطاف‌پذیر بنام آرایهٔ **دینامیک** (dynamic array) دارد. تعیین تعداد عناصر یک آرایهٔ دینامیک در هنگام اجرای برنامه و توسط برنامه‌نویس صورت می‌گیرد. ایجاد یک آرایهٔ دینامیک فرآیندی چند مرحله‌ایست، چون با اینکه تعداد عناصر این آرایه معلوم نیست، اما اختصاص فضای حافظه به آن همچنان باید صورت گیرد. مراحل ایجاد یک آرایهٔ دینامیک را در زیر می‌بینید:

- آرایه را، بدون تعیین تعداد عناصر آن، تعریف کنید:

```
Dim Temperatures() As Single
```

- تعداد عناصر آرایه را مشخص کنید (مثلاً، آنرا از کاربر بگیرید):

```
Dim Days As Short
Days = InputBox("How many days?", "Create Array")
```

- با استفاده از دستور ReDim تعداد عناصر آرایه را تعیین کنید (البته، باید عدد مرحلهٔ قبل را منهای 1 کنید، چون اندیس آرایه‌ها از 0 شروع می‌شود):

```
ReDim Temperatures(Days - 1)
```

- در حلقه‌های For...Next از تابع UBound استفاده کنید (چون تعداد عناصر آرایه را از قبل معلوم نیست):

```
For i = 0 To UBound(Temperatures)
    Temperatures(i) = InputBox(Prompt, Title)
Next
```

در تمرین زیر، برنامهٔ Fixed Array را با استفاده از آرایهٔ دینامیک بازنویسی خواهیم کرد.

کار با یک آرایهٔ دینامیک

۱ ادیتور کُد برنامهٔ My Fixed Array را باز کنید.

۲ به دستور تعریف آرایهٔ Temperatures (در بالای فرم) بروید.

۳ عدد 6 را از داخل پرانتز حذف کنید، تا این دستور بصورت زیر در آید:

```
Dim Temperatures() As Single
```

٤ دستور زیر را در همان قسمت اضافه کنید:

```
Dim Days As Integer
```

(از این متغیر برای نگهداری تعداد عناصر آرایه استفاده خواهیم کرد.)

۵ به روال Button1_Click بروید، و آنرا بصورت زیر تغییر دهید (تغییرات را با حروف ضخیم ملاحظه می‌کنید):

```
Dim Prompt, Title As String
Dim i As Short
Prompt = "Enter the day's high temperature."
Days = InputBox("How many days?", "Create Array")
If Days > 0 Then ReDim Temperatures(Days - 1)
For i = 0 To UBound(Temperatures)
    Title = "Day " & (i + 1)
    Temperatures(i) = InputBox(Prompt, Title)
Next
```

در خط چهارم و پنجم، ابتدا تعداد دماها (روزها) از کاربر پرسیده شده، و سپس (اگر عددی که کاربر وارد کرده، بزرگتر از 0 باشد) تعداد عناصر آرایه با دستور ReDim ست می‌شود. همانطور که می‌بینید، از آنجائیکه در حلقهٔ For...Next از تابع UBound استفاده کرده‌ایم، نیازی به تغییر این قسمت وجود ندارد.

۶ به روال Button2_Click بروید، و آنرا بصورت زیر تغییر دهید (در اینجا هم تغییرات با حروف ضخیم مشخص شده‌اند):

```
Dim Result As String
Dim i As Short
Dim Total As Single = 0
Result = "High temperatures:" & vbCrLf & vbCrLf
For i = 0 To UBound(Temperatures)
    Result = Result & "Day " & (i + 1) & vbTab & Temperatures(i) & vbCrLf
    Total = Total + Temperatures(i)
Next
Result = Result & vbCrLf & "Average temperature:  " & Format(Total / Days, "0.0")
TextBox1.Text = Result
```

تنها تغییر عمده در این روال، جایگزین کردن عدد ثابت 7 با متغیر Days است.

۷ برنامه را ذخیره کنید (متن کامل این برنامه را می‌توانید در پوشهٔ c:\vbnetsbs\chap11\dynamic array پیدا کنید).

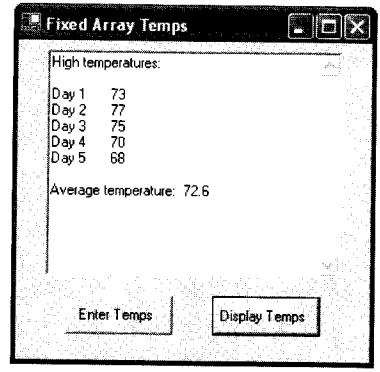
۸ برنامه را اجرا کنید.

۹ روی دکمهٔ Enter Temps کلیک کنید.

۱۰ (در اولین پنجرهٔ InputBox) عدد 5 را وارد کرده، و OK را کلیک کنید.

۱۱ برنامه پنج دمای مختلف از شما طلب می‌کند: آنها را وارد کنید.

۱۲ بعد از وارد کردن دماها، روی دکمهٔ Display Temps کلیک کنید. شکل زیر اجرای روال Button2_Click و خروجی برنامه را نشان می‌دهد:



۱۳ با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

حفظ محتویات آرایه در دستور ReDim

در تمرین قبل با استفاده از دستور ReDim تعداد عناصر یک آرایه را در هنگام اجرای برنامه تعیین کردید. اما این روش یک نقص کوچک هم دارد: اگر آرایه‌ای که دستور ReDim را روی آن اجرا می‌کنید، از قبل دارای اطلاعات باشد، این اطلاعات برای همیشه (و بصورت غیر قابل برگشت) از بین خواهند رفت. با اجرای دستور ReDim تمام عناصر آرایهٔ جدید مقدار پیش‌فرض خود را (که یا 0 است یا null) می‌گیرند. شاید این ویژگی درمواردی خوب هم باشد،اما گاهی پیش می‌آید که نتیجهٔ آن فاجعه‌باراست وباید فکری برای آن کرد. خوشبختانه ویژوال بیسیک .NET برای این مورد هم تعبیری اندیشیده است: کلمهٔ کلیدی Preserve ؛ این کلمه وقتی با دستور ReDim ترکیب شود، باعث می‌شود تا محتویات آرایه در حین تغییر بُعد حفظ شود. شکل این دستور چنین است:

```
ReDim Preserve ArrayName(Dim1Elements, Dim2Elements, ...)
```

در این دستور تعداد ابعاد آرایه و نوع دادهٔ آن نباید تغییر کند. البته باید بدانید که اگر یک آرایه بیش از یک بُعد داشته باشد، فقط آخرین بُعد آن را می‌توان با ReDim تغییر داد، بدون اینکه اطلاعات آن از بین برود.

با یک مثال نحوهٔ تغییر دادن تعداد عناصر و ابعاد یک آرایه را (بدون از دست دادن اطلاعات آن) بهتر درک خواهید کرد. در دستور زیر یک آرایهٔ دینامیک معمولی بنام Philosophers تعریف کرده‌ایم:

```
Dim Philosophers() As String
```

سپس تعداد عناصر آنرا تغییر داده، و در آخرین خانهٔ آن اطلاعات وارد می‌کنیم:

```
ReDim Philosophers(200)
Philosophers(200) = "Steve Harrison"
```

حال اگر بخواهیم بدون از دست دادن اطلاعات موجود، تعداد عناصر این آرایه را تا ۳۰۱ (0 تا 300) افزایش دهیم، باید از دستور زیر استفاده کنیم:

```
ReDim Preserve Philosophers(300)
```

در آرایه‌های چندبُعدی نیز روش کار یکسان است. یک آرایهٔ سه‌بُعدی از اعداد اعشاری بنام myCube را در نظر بگیرید:

```
Dim myCube(,,) As Single
```

با دستورات زیر، تعداد عناصر آرایه را مشخص کرده، و در آن اطلاعات ذخیره می‌کنیم:

```
ReDim myCube(25, 25, 25)
myCube(10, 1, 1) = 150.46
```

پس از آن می‌توان با دستور ReDim Preserve تعداد عناصر آخرین بُعد این آرایه را (بدون نگرانی از دست دادن اطلاعات) تغییر داد:

```
ReDim Preserve muCube(25, 25, 50)
```

افزایش یا کاهش آخرین بُعد آرایه با دستور ReDim Preserve کاملاً امکانپذیر است، اما همین که بخواهید با این دستور ابعاد دیگر را تغییر دهید، با یک خطا مواجه خواهید شد.

فعلاً صحبت دربارهٔ آرایه‌ها کافیست؛ اجازه دهید به مبحث بعدی، یعنی کلکسیون، بپردازیم.

کلکسیونی از اشیاء

در قسمت قبل دربارهٔ آرایه‌ها، و طرز اختصاص حافظه به آنها، یاد گرفتید. کلکسیون (collection) نوع پیشرفته‌تری از آرایه است، که می‌تواند بجای چند متغیر اشیاء را در خود ذخیره کند. دیدید که ویژوال بیسیک تمام اشیاء روی فرم را در یک فایل ذخیره می‌کند؛ اما آیا می‌دانید که از دید ویژوال بیسیک این اشیاء یک گروه نیز می‌سازند؟ در قاموس ویژوال بیسیک، به مجموعهٔ اشیاء روی فرم، کلکسیون کنترل‌ها (Controls collection) گفته می‌شود. این کلکسیون به محض ایجاد فرم برنامه بوجود می‌آید، و تمام کنترلهای برنامه در آن قرار داده می‌شوند. در واقع، هر برنامه تعداد زیادی کلکسیون دارد، که می‌توانید با آنها کار کنید.

هر کلکسیون با یک نام مشخص می‌شود؛ برای مثال، نام کلکسیون کنترلهای فرم، کلکسیون Controls است. اضافه کردن کنترل جدید به این کلکسیون از طریق کُد برنامه نیز امکانپذیر است. (در هر پروژه به تعداد فرمها کلکسیون Controls وجود دارد.) اگر می‌خواهید بدانید در یک برنامه (و یا حتی کل سیستم) چه کلکسیونهایی وجود دارد، می‌توانید از کاوشگر شیء (Object Browser -که جزء ابزارهای محیط ویژوال استودیو است) کمک بگیرید.

کار با اشیاء کلکسیون

کار با اشیاء یک کلکسیون دقیقاً شبیه کار با عناصر آرایه است، یعنی از طریق اندیس شیء در کلکسیون. با این حال در ویژوال بیسیک، ترتیب قرار گرفتن اشیاء در کلکسیون عکس ترتیب ایجاد آنهاست. برای مثال،

دستور زیر خاصیت Text آخرین شیء کلکسیون Controls را به "Business" ست می‌کند:

```
Controls(0).Text = "Business"
```

(شیء ماقبل آخر اندیس 1 دارد، شیء قبل از آن اندیس 2 و الی‌آخر.) با این منطق، هرگز نباید از یک اندیس ثابت برای یک شیء خاص استفاده کنید، چون هر آن ممکنست شیء جدیدی به کلکسیون کنترلها اضافه شود، و تمام اندیس‌ها را یکی بالا ببرد. حلقهٔ For...Next زیر نام آخرین چهار کنترل فرم را در پنجرهٔ MsgBox نمایش می‌دهد:

```
Dim i As Integer
For i = 0 To 3
    MsgBox(Controls(i).Name)
Next i
```

همانطور که حدس زده‌اید، حلقهٔ For...Next وسیلهٔ مناسبی برای کار با اشیاء کلکسیون نیست. خبر خوب اینست که، ویژوال بیسیک ابزار کارآمدتری برای کار با کلکسیونها دارد.

حلقهٔ For Each...Next

با اینکه با اشیاء یک کلکسیون بصورت تکی هم می‌توان کار کرد، ولی اغلب باید آنها را بعنوان یک گروه در نظر بگیریم (در واقع، فلسفهٔ وجودی کلکسیون نیز همین است). ویژوال بیسیک حلقهٔ خاصی بنام For Each...Next برای کار با کلکسیونها دارد، که با کمک آن دیگر نیازی نیست بدانیم کلکسیون چند عضو دارد، و یا ترتیب آنها چگونه است. شکل کلی حلقهٔ For Each...Next چنین است:

```
Dim CtrlVar As Control
...
For Each CtrlVar In Controls
    process object
Next CtrlVar
```

که در آن، *CtrlVar* متغیریست از نوع Control که به شیء فعلی در کلکسیون Controls اشاره می‌کند (دقت کنید که این دومی یک "s" اضافه دارد). در بدنهٔ حلقه می‌توانید با تمام خواص و متدهای کنترل موردنظر کار کنید، مثلاً خاصیت Text ، Enabled یا Visible آنرا عوض کنید.

کار با اشیاء کلکسیون Controls

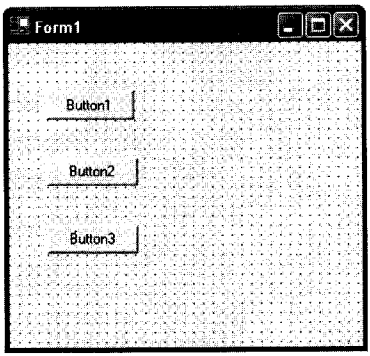
در تمرین زیر، با استفاده از کلکسیون Controls و حلقهٔ For Each...Next ، خاصیت Text سه دکمه را در هنگام اجرای برنامه تغییر داده، و آنها را روی فرم جابجا می‌کنیم.

عوض کردن خاصیت Text با استفاده از حلقهٔ For Each...Next

۱ پروژهٔ قبلی را با فرمان File|Close Solution ببندید.

۲ یک پروژهٔ جدید Visual Basic Windows Application بنام **My Controls Collection** در پوشهٔ c:\vbnetsbs\chap11 ایجاد کنید.

۳ سه دکمه روی فرم برنامه رسم کنید (شکل زیر را ببینید):



٤ خاصیت Name دکمهٔ سوم (Button3) را به btnMoveObjects ست کنید.

٥ روی دکمهٔ اول (Button1) دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد باز شود.

٦ دستورات زیر را در این روال بنویسید:

```
For Each ctrl In Controls
    ctrl.Text = "Click Me!"
Next
```

این حلقه خاصیت Text هر سه دکمه (تمام کنترلهای فرم) را به Click Me! ست می‌کند. اما در این کُد از یک متغیر بنام ctrl استفاده کرده‌ایم، که باید آنرا تعریف کنیم.

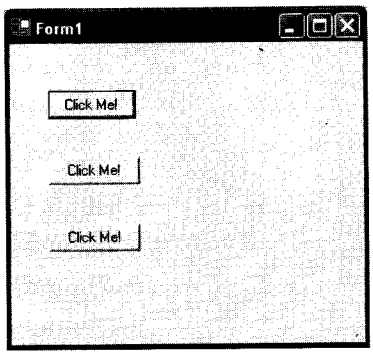
۷ به بالای ادیتور کُد رفته، و درست زیر جملهٔ "Windows Form Designer generated code" ، دستورات زیر را وارد کنید:

```
'Declare a variable of type Control to represent form controls
Dim ctrl As Control
```

(متغیر ctrl یک متغیر عمومی از نوع Control است، که در تمام روالهای فرم می‌توان از آن استفاده کرد.)

۸ برنامه را اجرا کنید.

۹ روی دکمهٔ اول (Button1) کلیک کنید، تا روال رویداد Button1_Click اجرا شود، و متن تمام دکمه‌ها عوض شود؛ شکل زیر را ببینید:



۱۰ با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

در مرحلهٔ بعد می‌خواهیم این دکمه‌ها را روی فرم حرکت دهیم، و برای این کار از تغییر دادن خاصیت Left آنها استفاده خواهیم کرد.

حرکت دادن کنترل‌ها با استفاده از حلقهٔ For Each...Next

۱ به فرم پروژه برگردید، و بعد از دو-کلیک روی دکمهٔ دوم (Button2) ، دستورات زیر را در روال رویداد Button2_Click بنویسید:

```
For Each ctrl In Controls
    ctrl.Left = ctrl.Left + 25
Next
```

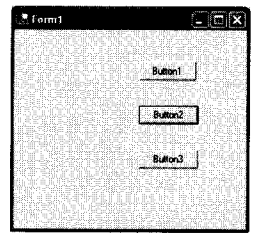
هر بار که روی دکمهٔ دوم کلیک کنیم (و این روال اجرا شود)، تمام دکمه‌ها باندازهٔ ۲۵ پیکسل (pixel -کوچکترین نقطهٔ نورانی روی مانیتور) به سمت راست جابجا خواهند شد. (برای حرکت دادن دکمه‌ها به سمت چپ، باید ۲۵ پیکسل را از خاصیت Left کم کنیم.)

نکته

واحد اندازه‌گیری پیش‌فرض در ویرایشهای قبلی ویژوال بیسیک Twip (معادل $\frac{۱}{۱۴۴۰}$ اینچ) نام داشت، و برای عوض کردن آن می‌توانستید از خاصیت ScaleMode فرم استفاده کنید. اما ویژوال بیسیک .NET دیگر از خاصیت ScaleMode پشتیبانی نمی‌کند، و واحد اندازه‌گیری پیش‌فرض آن Pixel (پیکسل) است.

۲ برنامه را اجرا کنید.

۳ چند بار روی دکمهٔ دوم (Button2) کلیک کنید؛ بله! هر سه دکمه به سمت راست فرم حرکت می‌کنند (شکل زیر را ببینید:)



۴ با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

گاهی پیش می‌آید که بخواهید با چند کنترل (و نه همهٔ آنها) کار کنید. در تمرین زیر خواهید دید که چگونه می‌توان دکمه‌های اول و دوم را حرکت داد، ولی دکمهٔ سوم را سر جای خود ثابت نگه داشت.

استفاده از خاصیت Name در حلقه‌های For Each...Next

برای پردازش انتخابی اشیاء یک کلکسیون می‌توانید از خاصیت Name آنها کمک بگیرید. تا اینجا همیشه در هنگام طراحی برنامه با این خاصیت سروکار داشتید، اما اکنون می‌خواهیم در کُد برنامه از آن استفاده کنیم. برای اینکار باید خاصیت Name تک تک اشیاء را با یک دستور If...Then در حلقهٔ For Each...Next چک کنیم.

نکته

اگر تعداد اشیایی که مایلید استثنا شوند، بیش از یکی باشد، می‌توانید در حلقهٔ For Each...Next از If...Then...Else یا Select Case کمک بگیرید.

در تمرین زیر با استفاده از این تکنیک دکمه‌های اول و دوم را حرکت داده، ولی دکمهٔ سوم (btnMoveObjects) را ثابت نگه خواهیم داشت.

نکته

یکی دیگر از خواصی که می‌توان برای این منظور بجای Name از آن کمک گرفت، خاصیت Tag است. خاصیت Tag (که اکثر اشیاء از آن پشتیبانی می‌کنند) بسیار شبیه Name است، و می‌توان در آن اطلاعاتی را دربارهٔ شیء نوشت، و بعدها از آن استفاده کرد.

استفاده از خاصیت Name برای پردازش انتخابی اشیاء کلکسیون

۱ به فرم پروژه برگردید، و روی دکمهٔ سوم (btnMoveObjects) دو-کلیک کنید، تا روال رویداد btnMoveObjects_Click در ادیتور کُد باز شود.

۲ دستورات زیر را در این روال بنویسید:

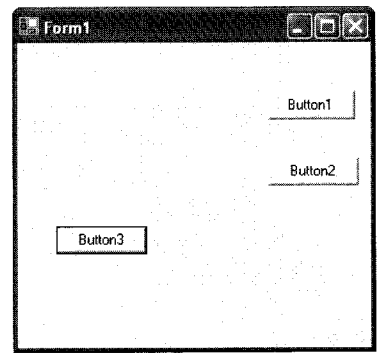
```
For Each ctrl In Controls
    If ctrl.Name <> "btnMoveObjects" Then
        ctrl.Left = ctrl.Left + 25
    End If
Next
```

بلوک If در این روال باعث می‌شود تا دستور حرکت (ctrl.Left = ctrl.Left + 25) برای تمام کنترلها بجز دکمهٔ btnMoveObjects اجرا شود، و همین باعث می‌شود تا این دکمه در جای خود ثابت بماند و حرکت نکند.

۳ با کلیک کردن دکمهٔ Save All ، برنامه را ذخیره کنید. (متن کامل این برنامه را می‌توانید در پوشهٔ c:\vbnetsbs\chap11\controls collection پیدا کنید).

٤ برنامه را اجرا کنید.

٥ پنج یا شش بار روی دکمهٔ سوم کلیک کنید؛ همانطور که می‌بینید، دکمه‌های اول و دوم به سمت راست فرم حرکت می‌کنند، ولی خود دکمهٔ سوم ثابت و بی‌حرکت می‌ماند (شکل زیر را ببینید:)



پردازش انتخابی اشیاء یک کلکسیون تکنیک بسیار سودمندیست، که می‌تواند کاربردهای متعددی داشته باشد.

٦ با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

خودتان کلکسیون بسازید

کلکسیون یکی از ابزارهای بسیار قدرتمند ویژوال بیسیک برای مدیریت داده‌هاست، که به اشیاء و کنترلها محدود نیست، و می‌توانید بعنوان آرایه (البته آرایه‌ای با امکانات بیشتر) هم از آن استفاده کنید.

تعریف و ایجاد کلکسیون بسیار شبیه متغیرها و آرایه‌هاست، و حتی در زمینهٔ میدان دید نیز از همان قواعد پیروی می‌کند. شکل کلی تعریف یک کلکسیون چنین است:

```
Dim CollectionName As New Collection()
```

که در آن *CollectionName* نام کلکسیون جدید است (اگر کلکسیون را در ماژول استاندارد تعریف می‌کنید، بجای Dim از Public استفاده کنید). برای اضافه کردن عنصر جدید به یک کلکسیون متدی وجود دارد بنام Add ؛ خواندن کلکسیونها هم معمولاً با حلقهٔ For Each...Next انجام می‌شود.

در تمرین زیر برای نگهداری آدرسهای وب (URL) بازدید شده، از یک کلکسیون استفاده خواهیم کرد. (ارتباط با اینترنت را هم از طریق اجرای کاوشگر اینترنت اکسپلورر با متد کتابخانه‌ای System.Diagnostics.Process.Start برقرار خواهیم کرد.)

نگهداری آدرسهای اینترنت در یک کلکسیون

١ با اجرای فرمان File|Close Solution ، پروژهٔ Controls Collection را ببندید.

۲ یک پروژهٔ جدید Visual Basic Windows Application بنام **My URL Collection** در پوشهٔ c:\vbnetsbs\chap11 ایجاد کنید.

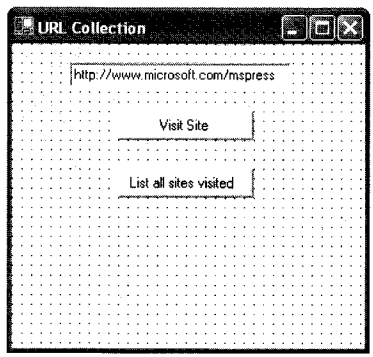
۳ یک جعبه متن پهن در بالای فرم (و وسط آن) رسم کنید.

٤ دو دکمه در زیر این جعبه متن رسم کنید.

۵ خواص فرم برنامه و کنترلهای روی آنرا با توجه به جدول زیر ست کنید:

شیء	خاصیت	مقدار
Form1	Text	"URL Collection"
Button1	Text	"Visit Site"
Button2	Text	"List all sites visited"
TextBox1	Text	"http://www.microsoft.com/mspress"

۶ در زیر شکل نهایی فرم برنامه را می‌بینید:



۷ با کلیک کردن دکمهٔ View Code در کاوشگر راه‌حل، ادیتور کُد را باز کنید.

۸ در بالای ادیتور کُد و درست زیر جملهٔ "Windows Form Designer generated code" ، دستور زیر را وارد کنید:

```
Dim URLsVisited As New Collection()
```

این دستور یک کلکسیون بنام URLsVisited ایجاد می‌کند، که در تمام فرم دیده خواهد شد.

۹ به فرم برنامه برگردید، روی دکمهٔ Visit Site دو-کلیک کنید تا روال Button1_Click باز شود، و سپس دستورات زیر را در این روال بنویسید:

```
URLsVisited.Add(TextBox1.Text)
System.Diagnostics.Process.Start(TextBox1.Text)
```

این کُد تمام آدرسهایی که کاربر در جعبه متن وارد کرده و بازدید می‌کند، را به کلکسیون

URLsVisited اضافه می‌کند (به طرز استفاده از متد Add توجه کنید). پس از ثبت آدرس، متد کتابخانه‌ای System.Diagnostics.Process.Start صفحهٔ وب خواسته شده را در کاوشگر پیش‌فرض سیستم باز می‌کند.

توجه

این برنامه فقط URL هایی را ثبت می‌کند، که در جعبه متن TextBox1 نوشته شده باشند. اگر بعد از باز شدن کاوشگر وب به آدرسهای دیگری بروید، آنها را در کلکسیون URLsVisited نخواهید دید. (در فصل ۲۱ با استفاده از مدل شیء Internet Explorer برنامه‌های بهتری برای اینترنت خواهیم نوشت.)

۱۰ به فرم برنامه برگردید، و روی دکمهٔ List all sites visited دو-کلیک کنید تا روال Button2_Click باز شود.

۱۱ دستورات زیر را در این روال بنویسید:

```
Dim URLName, AllURLs As String
For Each URLName In URLsVisited
    AllURLs = AllURLs & URLName & vbCrLf
Next URLName
MsgBox(AllURLs, MsgBoxStyle.Information, "Web sites visited")
```

این کُد تمام URL های بازدید شده را در یک حلقهٔ For Each...Next به هم می‌چسباند، و سپس آنها را با تابع MsgBox نمایش می‌دهد. دقت کنید که چگونه پیام نهایی با چسباندن تک تک URL ها ایجاد می‌شود.

نکتهٔ جدیدی که در تابع MsgBox می‌بینید، آرگومان MsgBoxStyle.Information است؛ این یکی از ثابتهای ویژوال بیسیک است، که مشخص می‌کند جعبه پیام چگونه باید نمایش داده شود.

توجه

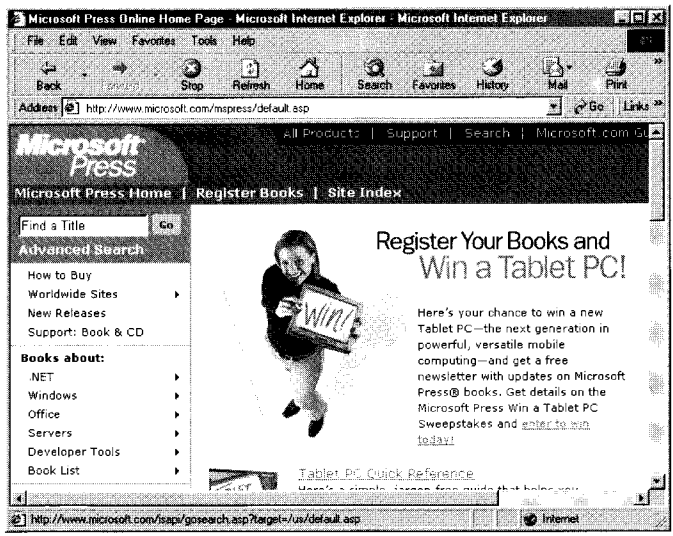
برای اجرای برنامهٔ URL Collection کامپیوتر شما باید مجهز به یکی از کاوشگرهای Internet Explorer یا Netscape Navigator باشد، و به اینترنت نیز دسترسی داشته باشد.

۱۲ برنامه را ذخیره کنید. (متن کامل این برنامه را می‌توانید در پوشهٔ c:\vbnetsbs\chap11\url collection پیدا کنید).

اجرای برنامهٔ URL Collection

۱ با کلیک کردن دکمهٔ Start ، برنامه را اجرا کنید.

۲ دکمهٔ Visit Site را کلیک کنید؛ ویژوال بیسیک کاوشگر وب پیش‌فرض سیستم را اجرا کرده، و صفحهٔ وب http://www.microsoft.com/mspress را در آن باز می‌کند (شکل زیر را ببینید).

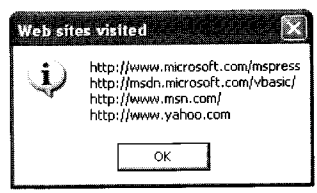


۳ به برنامهٔ URL Collection برگردید.

٤ دکمهٔ List all sites visited را کلیک کنید؛ پیام زیر ظاهر می‌شود:

۵ با کلیک کردن OK این پنجره را ببندید، و بعد از وارد کردن یک آدرس جدید در جعبه متن، دکمهٔ Visit Site را کلیک کنید. (اگر می‌خواهید اطلاعات بیشتری دربارهٔ ویژوال بیسیک .NET بدست آورید، به آدرس http://msdn.microsoft.com/vbasic/ بروید.)

۶ به چند سایت دیگر سر بزنید، و دوباره دکمهٔ List all sites visited را کلیک کنید؛ آدرس تمام سایتهایی را که تاکنون بازدید کرده‌اید، در یک جعبه پیام (مانند زیر) خواهید دید.



اگر تعداد این سایتها زیاد باشد، بهتر است بجای جعبه پیام (MsgBox) از یک جعبه متن (Multiline=True) استفاده کنید (آیا می‌دانید چگونه؟).

۷ بعد از اینکه کمی با برنامه کار کردید، با کلیک کردن دکمهٔ Close ، آن را ببندید.

این هم از کلکسیون، و طرز کار با آن!

یک گام فراتر: کلکسیونهای VBA (Visual Basic for Application)

اگر تاکنون برای برنامه‌های مجموعهٔ نرم‌افزاری میکروسافت آفیس، شامل Word ، Excel ، Access ، PowerPoint ، و یا دهها برنامهٔ دیگر که از VBA پشتیبانی می‌کنند، ماکرو (macro) نوشته باشید، می‌دانید که کلکسیونها چه نقش مهمی در آنجا بازی می‌کنند. مثلاً، در Word تمام سندهای باز در کلکسیون Documents ، و تمام پاراگرافهای سند فعلی در کلکسیون Paragraphs گرد آورده شده‌اند. پردازش این اشیاء با یک حلقهٔ For Each...Next (همانطور که در تمرین این قسمت خواهید دید) چندان دشوار نیست.

تمرین زیر یک ماکروی Word 2002 است، که در کلکسیون Documents بدنبال فایلی بنام MyLetter.doc می‌گردد، و اگر چنین سندی را یافت، آنرا با متد Save ذخیره می‌کند. اما اگر در کلکسیون Documents چنین فایلی وجود نداشت، سعی می‌کند آنرا از پوشهٔ C:\My Documents باز کند.

```
Dim aDoc As Object
Dim docFound As Boolean
Dim docLocation As String
docFound = False
docLocation = "c:\my documents\myletter.doc"
For Each aDoc In Documents
    If InStr(1, aDoc.Name, "myletter.doc", 1) Then
        docFound = True
        aDoc.Save
        Exit For
    End If
Next aDoc
If docFound = False Then
    Documents.Open FileName:=docLocation
End If
```


نکته

این ماکرو برای Microsoft Word 2002 نوشته شده است، نه ویژوال بیسیک. برای اجرای آن، Word را باز کرده و فرمان Tools|Macro|Macros را اجرا کنید، تا یک ماکروی جدید ایجاد شود؛ کُد فوق را در این ماکرو نوشته و آنرا اجرا کنید.

مرجع سریع فصل ۱۱

برای ...	انجام دهید
ایجاد یک آرایه	از کلمهٔ کلیدی Dim بصورت زیر استفاده کنید: `Dim MyArray(10) As Integer`
ایجاد یک آرایهٔ عمومی	در ماژول استاندارد، از کلمهٔ کلیدی Public بصورت زیر استفاده کنید: `Public MyArray(10) As Integer`
مقدار دادن به یک آرایه	از نام آرایه و اندیس عنصر موردنظر استفاده کنید: `MyArray(5) = 123`
فرمت کردن یک رشته	از ثابتهای vbCrLf و vbTab استفاده کنید.
ایجاد یک آرایهٔ دینامیک	ابتدا آرایه را بدون بُعد تعریف کرده، و سپس هنگام اجرای برنامه، بادستور ReDim ابعاد آرایه و تعداد عناصر آنرا تعیین کنید: `ReDim MyArray(20)`

پردازش عناصر یک آرایه	از حلقهٔ For...Next استفاده کنید:
	`For i = 0 To UBound(MyArray)`
	`  Total = Total + MyArray(i)`
	`Next`

تغییر دادن بُعد یک آرایه بدون از دست دادن اطلاعات	در دستور ReDim از کلمهٔ کلیدی Preserve استفاده کنید: `ReDim Preserve MyCube(20, 20, 100)`
ایجاد یک کلکسیون	از کلمات کلیدی Dim و New Collection() بصورت زیر استفاده کنید: `Dim MyCollection As New Collection()`

پردازش اشیاء یک کلکسیون	از حلقهٔ For Each...Next استفاده کنید:
	`For Each obj In MyCollection`
	`  obj.Name = "Hello!"`
	`Next`

حرکت دادن کنترلها	خاصیت Left کنترل موردنظر را تغییر دهید: `ctrl.Left = ctrl.Left + 25`
اضافه کردن عضو جدید به یک کلکسیون	از متد Add استفاده کنید: `MyCollection.Add(TextBox1.Text)`

فصل

۱۲

فایل‌های متنی و پردازش متن

در این فصل یاد می گیرید چگونه :

- ✔ یک فایل متنی را بازکرده، و در یک جعبه متن نمایش دهید.
- ✔ جملات و عبارات را در فایلهای متنی ذخیره کنید.
- ✔ با استفاده از تکنیکها و توابع رشته، فایلهای متنی را مرتب و رمزنگاری کنید.

یکی از مهمترین کاربردهای کامپیوتر در دنیای امروز پردازش سندهای الکترونیکی است. متداولترین نوع سند الکترونیکی نیز فایل‌های متنی (text files) است، که تشکیل شده‌اند از حروف، اعداد و کاراکترهای فرمت نشده. در این فصل یاد می‌گیرید که چگونه فایلهای متنی را باز کرده، محتویات آنها را بخوانید، و یا اینکه چگونه یک فایل متنی جدید ایجاد کنید. پردازش متن (ترکیب، مرتب‌سازی، رمزنگاری، و استخراج کلمات و جملات) از دیگر تکنیکهایی است که در این فصل با آن آشنا خواهید شد.

تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- توابع پردازش فایلهای متنی در ویژوال بیسیک ۶ عبارت بودند از: Open ، # Line Input ، # Print ، EOF ، و Close . مجموعه توابع پردازش فایل‌های متنی در ویژوال بیسیک .NET کاملاً جدید هستند. این توابع که زیرمجموعهٔ شیء FileSystem (در فضای نام Microsoft.VisualBasic) هستند، عبارتند از: FileOpen ، LineInput ، PrintLine ، و FileClose .
- علاوه بر توابع فوق، از توابع فضای نام System.IO نیز می‌توان برای مدیریت فایلها (جستجو در پوشه‌ها و درایوها، کپی و حذف فایلها، و پردازش استریمهای متنی) استفاده کرد. در واقع، توابع System.IO مکمل توابع Microsoft.VisualBasic.FileSystem محسوب می‌شوند.
- ویژوال بیسیک .NET همچنان از توابع سابق پردازش رشته پشتیبانی می‌کند، اما جایگزین‌های جدیدی نیز (در کلاسی بنام String) برای آنها عرضه کرده است. برای مثال، متد SubString همان کار تابع Mid را انجام می‌دهد، و بجای تابع UCase می‌توانید از متد ToUpper استفاده کنید. توصیه می‌کنم برای پردازش رشته‌های متنی از متدهای چارچوب NET. استفاده کنید.

نمایش فایل متنی در کنترل جعبه متن

ساده‌ترین راه نمایش فایلهای متنی استفاده از کنترل جعبه متن (TextBox) است. اگر محتویات فایل از جعبه متن بیشتر باشد، می‌توان با ست کردن خاصیت ScrollBars تمام متن را به نمایش درآورد. برای بار کردن یک فایل متنی در جعبه متن به چهار تابع نیاز داریم، که آنها را در جدول زیر ملاحظه می‌کنید (همانطور که گفتم، برخی از این توابع خاص ویژوال بیسیک .NET هستند).

تابع	توضیح
FileOpen	فایل متنی را برای خواندن یا نوشتن (ورودی یا خروجی) باز می‌کند.
LineInput	یک خط از فایل متنی را می‌خواند.
EOF	رسیدن به آخر فایل متنی را چک می‌کند.
FileClose	فایل متنی را می‌بندد.

باز کردن یک فایل متنی برای ورودی

فایل متنی تشکیل می‌شود از چند خط جمله، عدد و کاراکترهای دیگر. تفاوت فایل متنی با سند (document) این است که حاوی هیچ نوع کُد خاصی برای فرمت متن نیست؛ فایل‌های اجرایی (executable) نیز دارای کُدهای اجرایی سیستم عامل هستند، که در فایلهای متنی وجود ندارد. معروفترین فایلهای متنی آنهایی هستند که پسوند txt. ، ini. ، log. یا inf. دارند. از آنجائیکه در فایلهای متنی فقط کاراکترهای معمولی و قابل نمایش وجود دارد، می‌توان آنها را در کنترل جعبه متن بار کرد.

اگر می‌خواهید به کاربر اجازه دهید تا فایل را خود انتخاب کند، می‌توانید از کنترل OpenFileDialog استفاده کنید. البته این کنترل فایل را باز نمی‌کند، بلکه نام و مسیر آن را (در خاصیت FileName) برمی‌گرداند؛ باز کردن فایل را خود شما باید انجام دهید.

تابع FileOpen

بعد از گرفتن نام و مسیر فایل موردنظر، آنرا با تابع FileOpen باز می‌کنیم. شکل کلی این تابع چنین است:

```
FileOpen(filenumber, pathname, mode)
```

که در آن،

- *filenumber* (عدد فایل) یک عدد صحیح بین 1 تا 255 است.
- *pathname* نام و مسیر فایل موردنظر است.
- *mode* (حالت فایل) آرگومانیست که تعیین می‌کند فایل برای چه منظوری باز شده است. (در این فصل از حالتهای OpenMode.Input و OpenMode.Output استفاده خواهیم کرد.)

(البته این تابع آرگومانهای دیگری نیز دارد، که می‌توانید آنها را در سیستم کمک ویژوال بیسیک پیدا کنید.)

وقتی فایلی باز می‌شود، یک عدد فایل به آن اختصاص داده می‌شود. از آن به بعد برای ارجاع به این فایل باید از عدد آن استفاده کنیم. (ویژوال بیسیک نیز برای کنترل فایلهای باز در برنامه از عدد آنها استفاده می‌کند.) در دستور زیر برای باز کردن فایل از مقدار برگشتی کنترل OpenFileDialog کمک گرفته‌ایم:

```
FileOpen(1, OpenFileDialog1.FileName, OpenMode.Input)
```

که در آن عدد فایل 1 است، فایل برای خواندن باز شده (OpenMode.Input) ، و نام آن از یک دیالوگ بنام OpenFileDialog1 گرفته شده است.

نکته

فایلهایی که بصورت فوق باز شوند، فایل ترتیبی (sequential file) نامیده می‌شوند، چون محتویات آنها را بایستی بصورت ترتیبی (متوالی) بخوانید. (فایلهای پایگاه داده بطریق دیگری خوانده می‌شوند، که در فصل ۱۹ در این باره صحبت خواهم کرد.)

در تمرین زیر طرز باز کردن یک فایل متنی را به کمک کنترل OpenFileDialog خواهید دید. با طرز استفاده از توابع LineInput ، EOF ، و FileClose نیز آشنا خواهید شد.

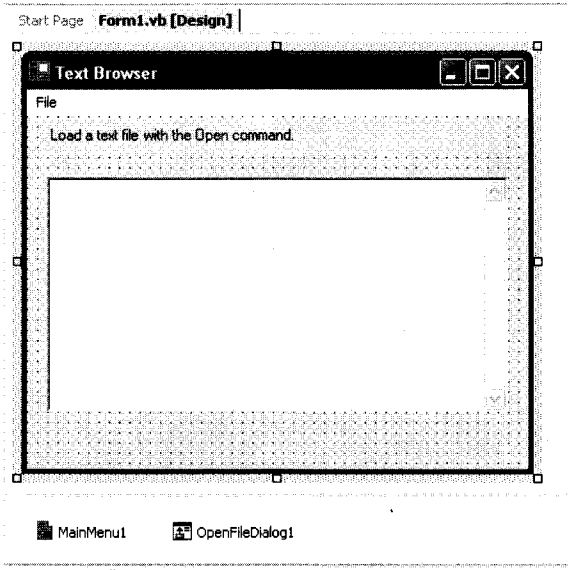
نکته

برای آشنایی با طرز اضافه کردن دیالوگهای استاندارد ویندوز به برنامه، به فصل ۳ مراجعه کنید.

اجرای برنامهٔ Text Browser

۱ در محیط ویژوال استودیو .NET ، پروژهٔ Text Browser را (که در پوشهٔ c:\vbnetsbs\chap12\text browser قرار دارد) باز کنید.

۲ در این برنامه یک جعبه متن بزرگ می‌بینید، که تقریباً تمام فرم را پوشانده است. فرمانهای Open (برای باز کردن فایل)، Close (برای بستن فایل) و Exit (برای خروج از برنامه) در منوی File قرار دارند. یک برچسب هم در بالای فرم می‌بینید، که طرز باز کردن فایل را خاطر نشان می‌کند. شکل فرم و مشخصات کنترلهای آنرا در شکل و جدول زیر ملاحظه می‌کنید:

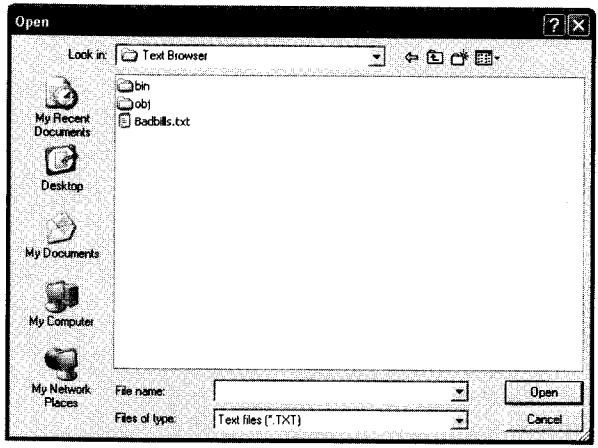


شیء	خاصیت	مقدار
Form1	Text	"Text Browser"
txtNote	Enabled	False
	Multiline	True
	Name	txtNote
	ScrollBars	Both
	Text	(خالی)
mnuOpenItem	Name	mnuOpenItem
mnuCloseItem	Enabled	False
	Name	mnuOpenItem
mnuExitItem	Name	mnuExitItem
lblNote	Name	lblNote
	Text	"Load a text file with the Open command"

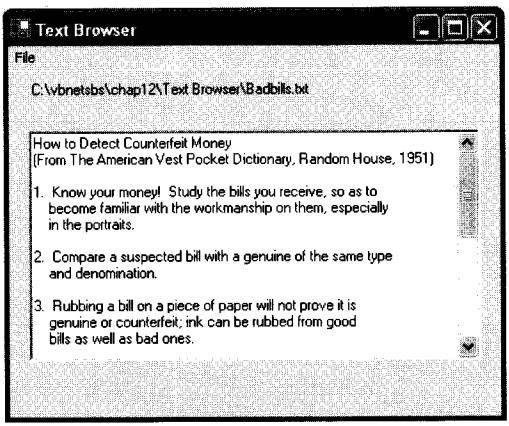
۳ برنامه را اجرا کنید.

٤ منوی File را باز کرده، و روی فرمان Open کلیک کنید، تا دیالوگ "بازکردن فایل" ظاهر شود.

۵ به پوشهٔ c:\vbnetsbs\chap12\text browser بروید:



۶ روی فایل Badbills.txt دو-کلیک کنید، تا محتویات آن در جعبه متن txtNote بار شود:



۷ به کمک **میله‌های لغزشی** (scroll bars) در این فایل بالا و پائین بروید. آیتم شماره ۵ را بخاطر بسپارید.

۸ با اجرای فرمان Close از منوی File ، فایل Badbills.txt را ببندید. سپس، فرمان File|Exit را اجرا کنید، تا برنامه هم بسته شود.

اجازه دهید نگاهی به کُد این برنامه بیندازیم.

بررسی کُد برنامهٔ Text Browser

۱ در محیط ویژوال استودیو .NET ، منوی File پروژهٔ Text Browser را باز کرده، و روی آیتم Open دو-کلیک کنید، تا روال رویداد mnuOpenItem_Click در ادیتور کُد باز شود.

۲ در زیر کُد روال mnuOpenItem_Click را می‌بینید:

```
Dim AllText, LineOfText As String
OpenFileDialog1.Filter = "Text files (*.TXT)|*.TXT"
OpenFileDialog1.ShowDialog()    'display Open dialog box
If OpenFileDialog1.FileName <> "" Then
    Try    'open file and trap any errors using handler
        FileOpen(1, OpenFileDialog1.FileName, OpenMode.Input)
        Do Until EOF(1)    'read lines from file
            LineOfText = LineInput(1)
            'add each line to the AllText variable
            AllText = AllText & LineOfText & vbCrLf
        Loop
        lblNote.Text = OpenFileDialog1.FileName    'update label
        txtNote.Text = AllText        'display file
        txtNote.Select(1, 0)          'remove text selection
        txtNote.Enabled = True        'allow text cursor
        mnuCloseItem.Enabled = True       'enable Close command
        mnuOpenItem.Enabled = False       'disable Open command
    Catch
        MsgBox("Error opening file.")
    Finally
        FileClose(1)    'close file
    End Try
End If
```

این روال اعمال زیر را انجام می‌دهد:

- متغیرهای لازم را تعریف کرده، و سپس خاصیت Filter کنترل OpenFileDialog1 را ست می‌کند.
- با متد ShowDialog دیالوگ را باز کرده، و نام فایل موردنظر را از کاربر می‌گیرد.
- خطاهای احتمالی را با یک بلوک Try...Catch بدام می‌اندازد.
- فایل را با متد FileOpen باز می‌کند.
- با متد LineInput فایل را خط به خط خوانده، و در متغیر AllText می‌نویسد.
- خواندن فایل تا رسیدن به علامت EOF ادامه می‌یابد (متغیر AllText می‌تواند فایلهای بسیار بزرگ را هم در خود جای دهد). اما اگر در حین کار (باز کردن یا خواندن فایل) خطایی رُخ دهد، قسمت Catch یک پیام خطا نشان می‌دهد.
- متغیر AllText را در جعبه متن txtNote می‌نویسد، و آنرا فعال می‌کند.
- تغییرات لازم را در آیتمهای منوی File می‌دهد، و سپس با متد FileClose فایل را می‌بندد.

(برای کسب اطلاعات بیشتر دربارهٔ هر یک از متدهای FileOpen ، LineInput ، EOF و FileClose آنرا انتخاب کرده و کلید F1 را بزنید.)

۳ به روال mnuCloseItem_Click بروید؛ کُد این روال را در زیر می‌بینید:

```
txtNote.Text = ""                    'clear text box
```

```
lblNote.Text = "Load a text file with the Open command."
mnuCloseItem.Enabled = False        'disable Close command
mnuOpenItem.Enabled = True          'enable Open command
```

این کُد همه چیز را به حالت شروع برنامه در می‌آورد، و تغییراتی را که پس از اجرای روال رویداد mnuOpenItem_Click رُخ داده‌اند، خنثی می‌کند.

این یک برنامهٔ ساده بود، که فایلهای متنی را باز می‌کرد و می‌خواند. در قسمت بعد خواهید دید که چگونه می‌توان فایلهای متنی را ایجاد، و روی دیسک ذخیره کرد.

باز کردن فایلهای متنی با استفاده از کلاس StreamReader

کتابخانهٔ دیگری که به کمک آن می‌توانید فایلهای متنی را باز کنید و بخوانید، کلاس StreamReader نام دارد (در این کتاب از هر دو روش استفاده خواهیم کرد).

برای استفاده از کلاس StreamReader ، ابتدا دستور زیر را در بالای فرم بنویسید:

```
Imports System.IO
```

سپس از کُد زیر برای باز کردن فایل، و نمایش آن در یک جعبه متن استفاده کنید:

```
Dim StreamToDisplay As StreamReader
StreamToDisplay = New StreamReader("c:\vbnetsbs\chap14\readme.txt")
TextBox1.Text = StreamToDisplay.ReadToEnd
StreamToDisplay.Close()
TextBox1.Select(0, 0)
```

همانطور که می‌بینید، طرز استفاده از کلاس StreamReader بسیار ساده‌تر از متدهای ...File است. بعد از آن که فایل را بصورت یک استریم باز کردید، می‌توانید با متد ReadToEnd محتویات آنرا تا به آخر بخوانید و در جعبه متن بنویسید. در پایان نیز باید استریم را (با متد Close) ببندید.

در فصل ۱۵ باز هم از کلاس StreamReader برای خواندن فایلها استفاده خواهیم کرد.

ایجاد فایل متنی جدید

یکی از کارهایی که در اکثر برنامه‌ها به آن نیاز خواهیم داشت، ایجاد فایل جدید و نوشتن اطلاعات در آن است. اغلب توابعی که در قسمت قبل دیدید، در اینجا هم به کار خواهند آمد. مراحل ایجاد یک فایل جدید و نوشتن اطلاعات در آن را در زیر ملاحظه می‌کنید:

۱ اطلاعات را از کاربر بگیرید، یا آنها در برنامهٔ خود تولید کنید (و یا هر دو).
۲ این اطلاعات را در یک (یا چند) متغیر ذخیره کنید.
۳ با استفاده از یک دیالوگ SaveFileDialog (و اجرای متد ShowDialog)، نام و مسیر فایل موردنظر را از کاربر بگیرید.
٤ فایل مزبور را برای نوشتن (حالت OpenMode.Output) باز کنید.
۵ متغیرهای مرحلهٔ ۲ را با استفاده از تابع PrintLine در این فایل بنویسید.
۶ فایل را با تابع FileClose ببندید.

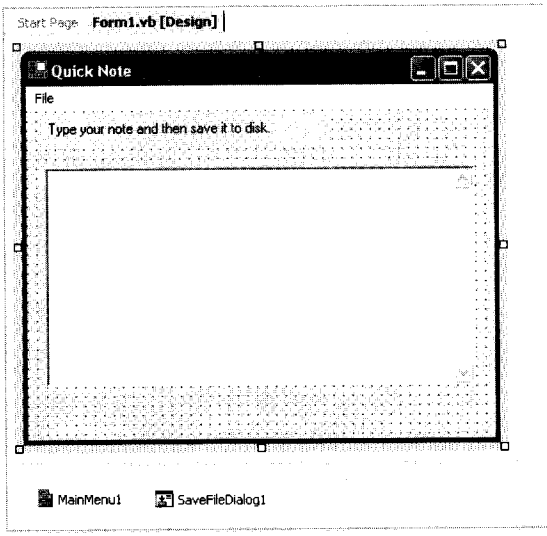
در زیر یک برنامهٔ ساده بنام Quick Note می‌بینید، که با توابع فوق فایلی را باز کرده و متن نوشته شده در یک جعبه متن را در آن ذخیره می‌کند. این برنامه ویرایش ساده وابتدایی برنامهٔ Windows NotePad است.

اجرای برنامهٔ Quick Note

۱ پروژهٔ قبلی را با فرمان File|Close Solution ببندید.

۲ به پوشهٔ c:\vbnetsbs\chap12\quick note بروید، و پروژهٔ Quick Note را باز کنید.

۳ ظاهر این پروژه بسیار شبیه برنامهٔ Text Browser است، با این تفاوت که بجای دیالوگ OpenFileDialog از SaveFileDialog استفاده می‌کند. منوی File آن هم شامل دو آیتم جدید بنامهای Save As و Insert Date است:



در جدول زیر خواص فرم برنامه و کنترلهای آن را ملاحظه می‌کنید:

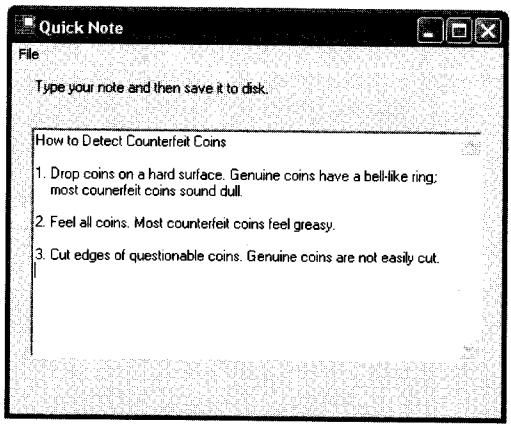
شیء	خاصیت	مقدار
Form1	Text	"Quick Note"
txtNote	Multiline	True
	Name	txtNote
	ScrollBars	Both
	Text	(خالی)
mnuInsertDateItem	Name	mnuInsertDateItem
mnuSaveAsItem	Name	mnuSaveAsItem
mnuExitItem	Name	mnuExitItem
lblNote	Name	lblNote
	Text	"Type your note and then save it to disk"

٤ برنامه را اجرا کنید.

۵ متن زیر (یا هر متن دیگری که مایلید) را در جعبه متن بنویسید (شکل زیر را ببینید):

How to Detect Conterfeit Coins

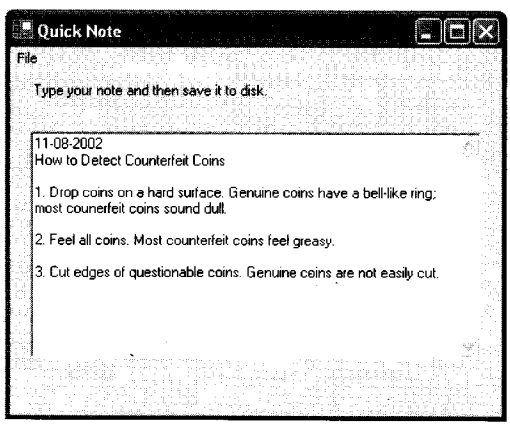
1. **Drop coins on a hard surface. Genuine coins have a bell-like ring; most counerfeit coins sound dull.**
2. **Feel all coins. Most counterfeit coins feel greasy.**
3. **Cut edges of questionable coins. Genuine coins are not easily cut.**



نکته

برای کپی کردن متن در تخته‌بُرش ویندوز، ابتدا متن موردنظر را در جعبه متن انتخاب کرده و سپس کلید Ctrl+C یا Ctrl+Ins را بزنید. برای برگرداندن متن از تخته‌بُرش به جعبه متن، کلید Ctrl+V یا Shift+Ins را بزنید.

۶ از منوی File فرمان Insert Date را انتخاب کنید؛ با این کار، تاریخ فعلی در بالای جعبه متن نوشته می‌شود:



۷ از منوی File فرمان Save As را انتخاب کنید.

۸ در دیالوگ Save As به پوشهٔ c:\vbnetsbs\chap12\quick note بروید. در فیلد File Name نام **Badcoins.txt** را وارد کرده، و دکمهٔ Save را کلیک کنید؛ با این کار، متن موجود در جعبه متن در فایلی بنام Badcoins.txt ذخیره خواهد شد.

۹ از منوی File فرمان Exit را اجرا کنید، تا برنامه بسته شود.

اجازه دهید نگاهی به کُد این برنامه بیندازیم.

بررسی کُد برنامهٔ Quick Note

۱ در محیط ویژوال استودیو .NET، منوی File پروژهٔ Quick Note را باز کرده، و روی آیتم Insert Date دو-کلیک کنید، تا روال رویداد mnuInsertDateItem_Click در ادیتور کُد باز شود. کُد این روال را در زیر می‌بینید:

```
txtNote.Text = DateString & vbCrLf & txtNote.Text
txtNote.Select(1, 0)        'Remove selection
```

این کُد با خاصیت DateString تاریخ فعلی را به ابتدای متن موجود در txtNote.Text اضافه می‌کند (به طرز ترکیب رشته‌ها و استفاده از ثابت vbCrLf دقت کنید).

۲ در همان ادیتور کُد، به روال mnuSaveAsItem_Click بروید؛ کُد این روال را در زیر ملاحظه می‌کنید:

```
SaveFileDialog1.Filter = "Text files (*.TXT)|*.TXT"
SaveFileDialog1.ShowDialog()    'display Open dialog box
If SaveFileDialog1.FileName <> "" Then
    FileOpen(1, SaveFileDialog1.FileName, OpenMode.Output)
    PrintLine(1, txtNote.Text)    'copy text to disk
    FileClose(1)    'close file
End If
```

این روال اعمال زیر را انجام می‌دهد:

- خاصیت Filter کنترل SaveFileDialog1 را ست می‌کند.
- با متد ShowDialog دیالوگ را باز کرده، و نام فایل موردنظر را از کاربر می‌گیرد؛ و در صورتیکه کاربر نام فایل را بدرستی وارد کرده باشد:
- فایل را با متد FileOpen باز می‌کند.
- با متد PrintLine محتویات جعبه متن txtNote.Text را (به یکباره) در فایل می‌نویسد. (آرگومان اول متد PrintLine شمارهٔ فایل است.)
- و در پایان، با متد FileClose فایل را می‌بندد.

(برای کسب اطلاعات بیشتر دربارهٔ هر یک از متدهای FileOpen ، PrintLine ، و FileClose آنرا

انتخاب کرده و کلید F1 را بزنید.)

۳ پس از بررسی بیشتر کُد برنامهٔ Quick Note ، با فرمان File|Close Solution آنرا ببندید.

پردازش رشته‌های متنی

در قسمت قبل دیدید که چگونه فایلها را باز کنید، بخوانید، و بنویسید. اما ویژوال بیسیک برای پردازش متن نیز دستورات متعدد و قدرتمندی دارد. در این قسمت خواهید دید که چگونه اطلاعات موردنظر را از متن استخراج کنید، لیستی از متن را در یک آرایه کپی کنید، و یا لیستی از کلمات را مرتب کنید.

یکی از مهمترین پردازشهایی که روی لیستها انجام می‌شود، مرتب‌سازی (sort) است. اصول مرتب‌سازی بسیار ساده است: آیتمهای لیست را یکی یکی با هم مقایسه، و آنها را بصورت صعودی یا نزولی مرتب کنید. مقایسه کردن رشته‌ها را می‌توان توسط عملگرهای مقایسه‌ای انجام داد (فصل ۵ را ببینید). مهمترین و پیچیده‌ترین قسمت کار چگونگی این مقایسه کردن است (سالیان متمادیست که دانشمندان به بحث در این زمینه مشغول هستند). در اینجا قصد ندارم وارد بحث نظری الگوریتمهای مرتب‌سازی شوم، ولی همین قدر بگویم که نقطهٔ مشترک این بحث‌ها سرعت پردازش است (عاملی که فقط در حجمهای بالا خود را نشان می‌دهد). بجای آن توجه خود را روی اصول مقایسهٔ رشته‌ها در عملیات مرتب‌سازی متمرکز خواهیم کرد. مهارتهایی که در این قسمت فرا می‌گیرید، در حوزه‌های متعددی (مانند برنامه‌نویسی پایگاه داده) کاربرد دارند.

متدهای پردازش متن

مهمترین کاری که تا اینجا روی رشته‌های متنی انجام دادیم، چسباندن آنها با عملگر & بود. در مثال زیر سه رشتهٔ متنی به کمک این عملگر بهم چسبانده شده، و عبارت "Bring on the circus!" را ساخته‌اند:

```
Dim Slogan As String
Slogan = "Bring" & " on the " & "circus!"
```

برای چسباندن رشته‌ها متد دیگری بنام Concat نیز وجود دارد، که عضو کلاس String است. در زیر همان کُد بالا را با متد String.Concat نوشته‌ایم:

```
Dim Slogan As String
Slogan = String.Concat("Bring", " on the ", "circus!")
```

علاوه بر متدهای جدید ویژوال بیسیک .NET در زمینهٔ پردازش رشته‌ها، از توابع و عملگرهای قدیمی ویژوال بیسیک (مانند، Mid ، UCase ، LCase و غیره) نیز می‌توانید برای این منظور استفاده کنید. در این کتاب اغلب از متدهای چارچوب .NET برای پردازش رشته‌ها استفاده خواهیم کرد، ولی گاهی توابع قدیمی را نیز بکار خواهیم برد. هیچیک از این دو روش مزیت خاصی نسبت به دیگری ندارد، بنابراین می‌توانید بدلخواه از هر کدام (و یا از ترکیب) آنها استفاده کنید.

در جدول زیر تعدادی از متدهای پردازش متن در چارچوب .NET. (و معادل قدیمی آنها) را ملاحظه می‌کنید (برای هر متد مثالی نیز آورده‌ام):

Dim Name, NewName As String Name = "Reza" NewName = Name.ToUpper 'NewName = "REZA"	تمام حروف رشته را بزرگ می‌کند	UCase	ToUpper
Dim Name, NewName As String Name = "Reza" NewName = Name.ToLower 'NewName = "reza"	تمام حروف رشته را کوچک می‌کند	LCase	ToLower
Dim River As String Dim Size As Short River = "Mississippi" Size = River.Length 'Size = 11	تعداد حروف رشته را برمی‌گرداند	Len	Length
Dim Cols, Middle As String Cols = "First Second Third" Middle = Cols.SubString(6, 6) 'Middle = "Second"	تعداد مشخصی از حروف رشته را از محل تعیین شده برمی‌گرداند (توجه: اندیس رشته از 0 شروع می‌شود)	Mid	SubString
Dim Name As String Dim Start As Short Name = "Hamed" Start = Name.IndexOf("med") 'Start = 2	نقطهٔ شروع یک رشته را در رشتهٔ بزرگتر برمی‌گرداند	InStr	IndexOf
Dim Spacy, Trimmed As String Spacy = " God " Trimmed = Spacy.Trim 'Trimmed = "God"	فاصله‌ها را از ابتدا و انتهای رشته حذف می‌کند	Trim	Trim
Dim RawStr, CleanStr As String RawStr = "Hello333 there!" CleanStr = RawStr.Remove(5, 3) 'CleanStr = "Hello there!"	حروف را از وسط رشتهٔ بزرگتر حذف می‌کند		Remove
Dim OldStr, NewStr As String OldStr = "Hi Alireza" NewStr = OldStr.Insert(3, "there ") 'NewStr = "Hi there Alireza"	حروف را در وسط رشتهٔ بزرگتر می‌نویسد		Insert
Dim str1 As String = "Football" Dim str2 As String = "FOOTBALL" Dim Match As Short Match = StrComp(str1, _ str2, CompareMethod.Text) 'Match = 0 [strings match]	دو رشته را (بدون توجه به نوع حروف) مقایسه می‌کند		StrComp

مرتب‌کردن متن

ویژوال بیسیک حروف و کاراکترها را بر اساس مقدار عددی آنها در **مجموعه کاراکترهای آسکی** (ASCII character set) -که به ANSI نیز معروف است - مقایسه می‌کند. (ASCII مخفف **کُد استاندارد آمریکایی برای تبادل اطلاعات** - American Standard Code for Information Interchange -است.) هر حرف یا علامتی که در کامپیوتر کاربرد دارد، دارای یک کُد در جدول آسکی می‌باشد. کُدهای آسکی به دو دسته تقسیم می‌شوند: کُدهای قابل نمایش (از 32 تا 127) و کُدهای کنترلی (از 0 تا 31). برای مثال، حرف "a" در جدول آسکی معادل 97، و حرف "A" معادل 65 است. از همین جا می‌توان فهمید که چرا ویژوال بیسیک هنگام مرتب کردن رشته‌ها بین "a" و "A" تفاوت قایل می‌شود.

در دهۀ ۱۹۸۰، شرکت IBM جدول آسکی را گسترش داد، و کُدهای 128 تا 255 را به آن اضافه کرد. این کاراکترها شامل حروف یونانی، حروف خاص زبانهای اروپایی، علائم گرافیکی و برخی نمادهای ریاضی بود. به این جدول جدید **مجموعه کاراکترهای توسعه‌یافتۀ** *IBM* گفته می‌شود. (کُدهای آسکی را می‌توانید در سیستم کمک ویژوال بیسیک مشاهده کنید.)

با اینکه جدول آسکی همچنان نقش مهمی در سیستمهای کامپیوتری بازی می‌کنند، اما تنها مجموعه کاراکتر موجود نیست. با گسترش روزافزون نقش کامپیوتر و جهانی شدن نرم‌افزارها، استاندارد جدیدی بنام **یونی‌کُد** (Unicode) وضع شده که تا 65,536 کاراکتر را می‌تواند نمایش دهد، و تقریباً تمام زبانهای زندۀ دنیا را پوشش می‌دهد (تا زمان نوشته شده این کتاب، نزدیک به 45,000 کاراکتر در این سیستم تعریف شده است). تعریف کاراکترهای این استاندارد (و تعیین کُدهای آن) بر عهدۀ یک هیأت بین‌المللی گذاشته شده است. ویندوز NT، ویندوز 2000، ویندوز XP، و ویژوال بیسیک .NET از استانداردهای آسکی و یونی‌کُد پشتیبانی می‌کنند. اگر می‌خواهید نرم‌افزارهای چندزبانه و بین‌المللی بنویسید، باید با استاندارد یونی‌کُد و برنامه‌نویسی آن آشنا شوید.

کار با کُدهای آسکی (ASCII Codes)

در ویژوال بیسیک تابعی وجود دارد بنام Asc، که کُد آسکی حروف را برمی‌گرداند. بعد از اجرای دستورات زیر، متغیر AscCode مقدار 122 خواهد داشت:

```
Dim AscCode As Short
AscCode = Asc("z")
```

تابع Chr درست عکس Asc عمل می‌کند، یعنی کاراکتر متناظر با عدد داده شده را برمی‌گرداند. برای مثال، خروجی کُد زیر کاراکتر "z" خواهد بود:

```
Dim Letter As Char
Letter = Chr(122)
```

این توابع را براحتی می‌توان با هم ترکیب کرد:

```
AscCode = Asc("z")
Letter = Chr(AscCode)
```

همانطور که قبلاً هم گفتم، برای مقایسهٔ رشته‌های متنی می‌توانید از عملگرهای مقایسه استفاده کنید (جدول زیر را ببینید).

عملگر مقایسه	مفهوم
=	تساوی
<>	عدم تساوی
>	بزرگتر از
<	کوچکتر از
>=	بزرگتر یا مساوی
<=	کوچکتر یا مساوی

یک کاراکتر بزرگتر ازکاراکتر دیگر است، اگر کُد اَسکی آن بالاتر باشد. مثلاً، کُد اَسکی "B" بزرگتر از "A" است، بنابراین عبارت زیر به True ارزیابی خواهد شد:

```
"A" < "B"
```

در حالیکه عبارت زیر مقدار False برمی‌گرداند:

```
"A" > "B"
```

هنگام مقایسهٔ رشته‌های چندحرفی، ویژوال بیسیک ابتدا اولین کاراکتر دو رشته را مقایسه می‌کند؛ اگر این دو یکسان بودند، سراغ کاراکتر دوم می‌رود، و این کار را تا زمانی ادامه می‌دهد که به یک تفاوت برسد. برای مثال، دو رشتهٔ Mike و Michael تا حرف دوم یکسان هستند، و تفاوت آنها از کاراکتر سوم ("k" و "c") شروع می‌شود؛ و از آنجائیکه کُد اَسکی "k" بزرگتر از "c" است، عبارت زیر True خواهد بود:

```
"Mike" > "Michael"
```

دو رشته زمانی مساوی تلقی می‌شوند، که تمام کاراکترهای آنها یکسان باشد. اگر تمام کاراکترهای دو رشته یکی باشند، ولی طول یکی بیشتر از دیگری باشد، رشتهٔ بلندتر بزرگتر خواهد بود؛ بعبارت دیگر، مقایسهٔ زیر مقدار True برمی‌گرداند:

```
"AAAAA" > "AAA"
```

مرتب‌کردن رشته‌ها در یک جعبه متن

در تمرین زیر برنامه‌ای می‌بینید بنام Sort Text ، که تغییریافتهٔ Quick Note است (با همان منوهای Open و Close). علاوه بر آن، فرمانی بنام Sort Text نیز دارد، که متن یک جعبه متن را مرتب می‌کند. از آنجائیکه متن موجود در یک جعبه متن تماماً یک رشته است، قبل از شروع مرتب‌سازی باید به رشته‌های کوچکتر شکسته شود. پس از آن، این رشته‌ها توسط سابروتین ShellSort (بر اساس الگوریتمی به همین نام، که در سال ۱۹۵۹ توسط دونالد شِل ابداع شده) مرتب می‌شوند. رشته‌های کوچک در یک آرایهٔ دینامیک عمومی

(که در ماژول استاندارد تعریف شده) ذخیره می‌شوند؛ سابروتین ShellSort هم در همین ماژول قرار دارد. یکی از قسمتهای جالب این برنامه روتینی است که تعداد خطوط جعبه متن را تعیین می‌کند (ویژوال بیسیک هیچ تابعی ندارد که این کار را بطور خودکار انجام دهد). در این برنامه از متدهای SubString و Chr برای تفکیک کلمات هر خط استفاده شده است. بویژه دقت کنید که چگونه از متد SubString در کنار خاصیت Text جعبه متن استفاده کرده‌ام (از آنجائیکه این متد یکی از اعضای کلاس String است، با هر چیزی که متن باشد می‌توان آنرا بکار برد).

```
Dim ln, curline, letter As String
Dim i, charsInFile, lineCount As Short
'determine number of lines in text box object (txtNote)
lineCount = 0       'this variable holds total number of lines
charsInFile = txtNote.Text.Length       'get total characters
For i = 0 To charsInFile - 1        'move one char at a time
    letter = txtNote.Text.Substring(i, 1)        'get letter
    If letter = Chr(13) Then        'if carriage return found
        lineCount += 1        'go to next line (add to count)
        i += 1        'skip linefeed char (always follows cr)
    End If
Next i
```

این کُد تعداد خطوط جعبه متن را در متغیر lineCount برمی‌گرداند. بعداً از این متغیر برای تعیین تعداد عناصر آرایهٔ رشته‌ها (آرایه‌ای که به سابروتین ShellSort فرستاده می‌شود) استفاده خواهیم کرد؛ این سابروتین آرایهٔ مزبور را بصورت مرتب شده برمی‌گرداند.

اجرای برنامهٔ Sort Text

۱ پروژهٔ Sort Text را (که در پوشهٔ c:\vbnetsbs\chap12\sort text قرار دارد) باز کنید.

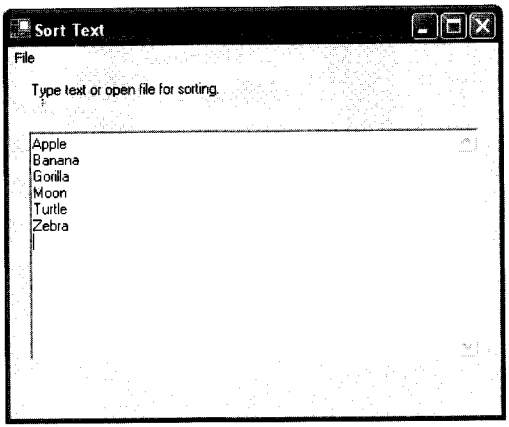
۲ با کلیک کردن دکمهٔ Start ، برنامه را اجرا کنید.

۳ کلمات زیر (یا هر چیز دیگری که مایلید) را در جعبه متن وارد کنید:

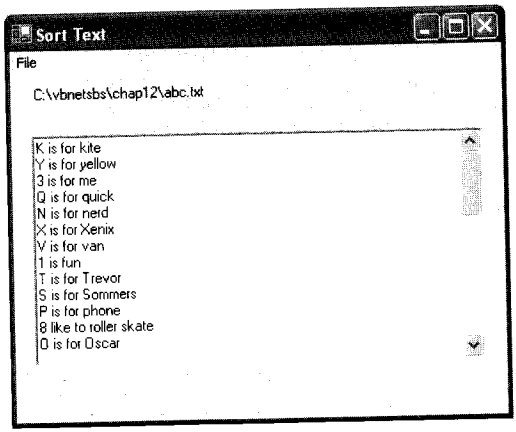
Zebra
Gorilla
Moon
Banana
Apple
Turtle

دقت کنید که بعد از آخرین کلمه هم Enter را بزنید، تا برنامه بتواند تعداد کلمات (خطوط) را بدرستی محاسبه کند.

۴ فرمان Sort Text را از منوی File انتخاب کنید. برنامه کلمات را بترتیب حروف الفبا مرتب کرده، و دوباره در جعبه متن نمایش می‌دهد:

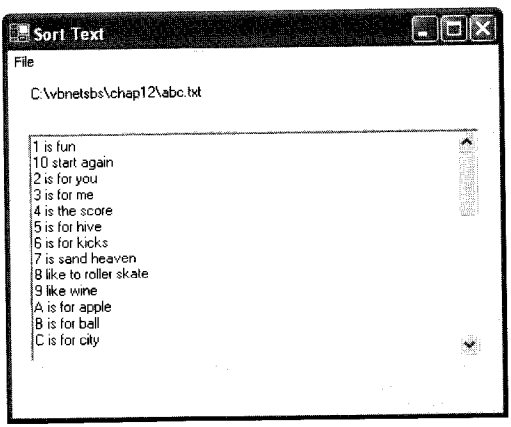


۵ به کمک فرمان Open (از منوی File)، فایل abc.txt را در پوشهٔ c:\vbnetsbs\chap12 باز کنید:



این فایل ۳۶ خط دارد، که هر خط با یک حرف (از A تا Z) یا عدد (از 0 تا 10) شروع می‌شود.

۶ فرمان Sort Text را از منوی File انتخاب کنید، تا برنامه فایل abc.txt را بترتیب حروف الفبا مرتب کند (شکل زیر را ببینید):



۷ به شکل مرتب شدهٔ فایل abc.txt دقت کنید؛ ظاهراً همه چیز درست است. اما نه! یک مشکل کوچک وجود دارد: چرا خطی که با عدد 10 شروع شده، بعد از 1 آمده، در حالیکه جای آن بعد از 9 است؟ در حقیقت، ویژوال بیسیک حروف "1" و "0" را بعنوان دو حرف مستقل در نظر می‌گیرد، نه بعنوان عدد 10 ؛ به همین دلیل (با توجه به جدول آسکی) "10" بعد "1" (و قبل از "2") می‌آید، نه بعد از "9" . اگر می‌خواهید با این برنامه اعداد را مرتب کنید، باید ابتدا رشته‌ها را به عدد تبدیل کنید، و سپس مقایسه را انجام دهید.

بررسی کُد برنامهٔ Sort Text

۱ با فرمان File|Exit ، برنامهٔ Sort Text را ببندید.

۲ ادیتور کُد را باز کرده، و به روال mnuSortTextItem_Click بروید. قبلاً دربارهٔ قسمت اول این روتین (تعیین تعداد خطوط جعبه متن) صحبت کردیم؛ در ادامهٔ این روتین، ابتدا تعداد عناصر آرایهٔ strArray (که کلمات هر خط در آن ذخیره خواهد شد) ست می‌شود، و سپس این آرایه به روتین مرتب‌سازی (سابروتین ShellSort) فرستاده می‌شود؛ پس از مرتب شدن، آرایهٔ strArray به کمک یک حلقهٔ For...Next در جعبه متن txtNote نوشته می‌شود. کُد کامل این روال را در زیر ملاحظه می‌کنید:

```
Dim ln, curline, letter As String
Dim i, charsInFile, lineCount As Short

'determine number of lines in text box object (txtNote)
lineCount = 0       'this variable holds total number of lines
charsInFile = txtNote.Text.Length       'get total characters
For i = 0 To charsInFile - 1        'move one char at a time
    letter = txtNote.Text.Substring(i, 1)        'get letter
    If letter = Chr(13) Then        'if carriage return found
        lineCount += 1        'go to next line (add to count)
        i += 1        'skip linefeed char (always follows cr)
    End If
Next i

'build an array to hold the text in the text box
ReDim strArray(lineCount)       'create array of proper size
curline = 1
ln = ""       'use ln to build lines one character at a time
For i = 0 To charsInFile - 1        'loop through text again
    letter = txtNote.Text.Substring(i, 1)        'get letter
    If letter = Chr(13) Then        'if carriage return found
        curline = curline + 1        'increment line count
        i += 1        'skip linefeed char
        ln = ""        'clear line and go to next
    Else
        ln = ln & letter        'add letter to line
        strArray(curline) = ln        'and put in array
    End If
Next i

'sort array
```

```
ShellSort(strArray, lineCount)

'then display sorted array in text box
txtNote.Text = ""
curline = 1
For i = 1 To lineCount
    txtNote.Text = txtNote.Text & strArray(curline) & vbCrLf
    curline += 1
Next i
txtNote.Select(1, 0)          'remove text selection
```

آرایهٔ دینامیک strArray در ماژول استاندارد (Module1.vb) تعریف شده است. دستور ReDim strArray(lineCount) آرایه‌ای می‌سازد که تعداد عناصر آن دقیقاً معادل تعداد خطوط جعبه متن است. پس از این دستور، هر خط جعبه متن (رشته‌ای که به یک کاراکتر باکُد اَسکی 13 ختم می‌شود) در آرایهٔ strArray نوشته می‌شود. پس از کامل شدن این آرایه، آنرا به سابروتین ShellSort می‌فرستیم، تا بترتیب حروف الفبا مرتب شود.

۳ در ادیتور کُد، ماژول استاندارد (Module1.vb) را باز کنید. در این ماژول، علاوه بر تعریف متغیر strArray ، سابروتین ShellSort نیز قرار دارد. این سابروتین به کمک عملگر => عناصر آرایهٔ strArray را مقایسه، و در صورت نیاز جابجا می‌کند. کُد این روتین را در زیر می‌بینید:

```
Sub ShellSort(ByRef sort() As String, ByVal numOfElements As Short)
    Dim temp As String
    Dim i, j, span As Short
    'The ShellSort procedure sorts the elements of sort()
    'array in descending order and returns it to the calling
    'procedure.

    span = numOfElements \ 2
    Do While span > 0
        For i = span To numOfElements - 1
            j = i - span + 1
            For j = (i - span + 1) To 1 Step -span
                If sort(j) <= sort(j + span) Then Exit For
                'swap array elements that are out of order
                temp = sort(j)
                sort(j) = sort(j + span)
                sort(j + span) = temp
            Next j
        Next i
        span = span \ 2
    Loop
buS dnE
```

این روتین لیست کلمات را بصورت پیوسته نصف کرده و لیستهای کوچکتری ایجاد می‌کند، و در هر مرحله بالاترین و پائین‌ترین عنصر هر یک از این لیست‌ها را مقایسه کرده، و در صورت نیاز جابجا می‌کند. (اگر می‌خواهید ترتیب مرتب‌سازی را عوض کنید، کافیست بجای عملگر => از =< استفاده کنید.)

این هم از مرتب کردن رشته‌ها. در قسمت آینده برنامهٔ Quick Note را طوری تغییر می‌دهیم که بتواند رشته‌های متنی را به رمز درآورد.

حفاظت متن و رمزنگاری

حال که با کُدهای اَسکی آشنا شدید، می‌خواهیم ببینیم چگونه می‌توان این کُدها را بهم ریخت، و متن واقعی را از چشم نامحرم مخفی کرد. این عمل که به آن رمزنگاری (encryption) گفته می‌شود، خصلت هر کاراکتر را عوض می‌کند، بگونه‌ای که برای غریبه‌ها نامفهوم شود. البته متنی که به رمز درمی‌آید، باید بتواند از رمز هم خارج شود، در غیر اینصورت حفاظت معنا نخواهد یافت (و در واقع کاری نیست جز خراب کردن فایل). رمزنگاری باید بگونه‌ای صورت گیرد که شکستن آن برای دیگران باندازهٔ کافی سخت باشد، بعبارت دیگر رمز نباید بسادگی قابل تشخیص باشد.

در این قسمت برنامه‌ای می‌بینید بنام Encrypt Text که متنی را به رمز در می‌آورد، و از رمز خارج می‌کند.

به رمز در آوردن متن با تغییر کُدهای اَسکی

۱ پروژهٔ Encrypt Text را (که در پوشهٔ c:\vbnetsbs\chap12\encrypt text قرار دارد) باز کنید.

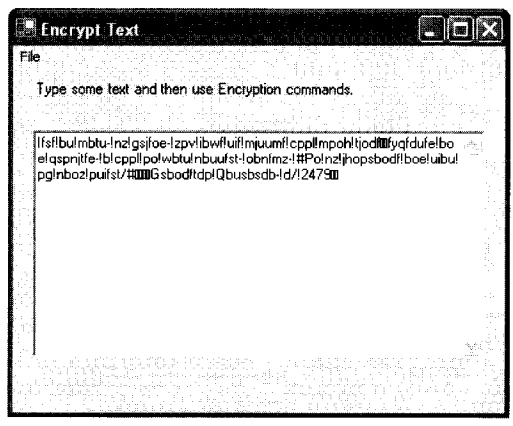
۲ با کلیک کردن دکمهٔ Start ، برنامه را اجرا کنید.

۳ متن زیر (یا هر متن دیگری که مایلید) را در جعبه متن وارد کنید:

Here at last, my friend, you have the little book long since expected and promised, a book on vast matters, namely, "On my ignorance and that of many others."

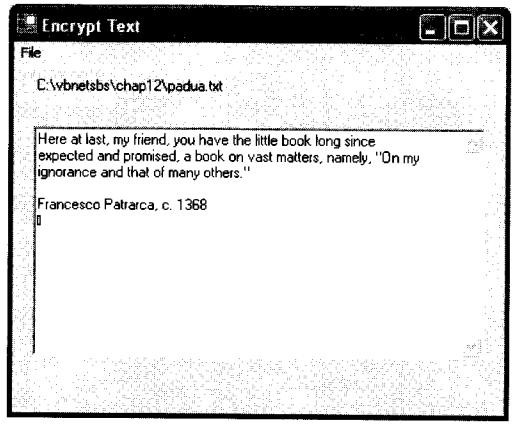
Francesco Patrarca, c. 1368

٤ فرمان Save Encrypted File As را از منوی File انتخاب کرده، و فایل را با نام **padua.txt** در پوشهٔ c:\vbnetsbs\chap12 ذخیره کنید. برنامه متن را به شکل زیر به هم ریخته، و در فایل مشخص شده ذخیره می‌کند.



اگر فایل padua.txt را در هر برنامهٔ دیگری (مانند Word یا NotePad) هم باز کنید، جز یکسری کاراکترهای نامفهوم چیز دیگری نخواهید دید.

۵ برای خواندن فایل رمز شده، از منوی File فرمان Open Encrypted File را اجرا کرده، و فایل padua.txt را (از پوشهٔ c:\vbnetsbs\chap12) باز کنید. برنامه فایل مزبور را از حالت رمز خارج کرده و دوباره بصورت اولیه نمایش می‌دهد:



(کاراکترهای اضافه‌ای که در انتهای فایل می‌بینید، معادل Enter هائیست که در آخر فایل زده‌اید.)

۶ با انتخاب فرمان Exit (از منوی File)، از برنامه خارج شوید.

بررسی کُد برنامهٔ Encrypt Text

۱ ادیتور کُد را باز کرده، و به روال mnuSaveAsItem_Click بروید؛ این روالیست که متن را به حالت رمز درمی‌آورد. الگوریتم رمزنگاری در این روال بسیار ساده است: در یک حلقهٔ For...Next کاراکترهای متن خوانده شده، و به کُد اَسکی هر کاراکتر عدد 1 اضافه می‌شود. کُد کامل این روال را در زیر ملاحظه می‌کنید (به دستوراتی که با حروف ضخیم مشخص شده، دقت کنید):

```
Dim Encrypt As String = ""
Dim letter As Char
Dim i, charsInFile As Short

SaveFileDialog1.Filter = "Text files (*.txt)|*.txt"
SaveFileDialog1.ShowDialog()
If SaveFileDialog1.FileName <> "" Then
    'save text with encryption scheme (ASCII code + 1)
    charsInFile = txtNote.Text.Length
    For i = 0 To charsInFile - 1
        letter = txtNote.Text.Substring(i, 1)
        'determine ASCII code and add one to it
        Encrypt = Encrypt & Chr(Asc(letter) + 1)
    Next
```

```
    FileOpen(1, SaveFileDialog1.FileName, OpenMode.Output)
    PrintLine(1, Encrypt)          'copy text to disk
    FileClose(1)
    txtNote.Text = Encrypt
    txtNote.Select(1, 0)           'remove text selection
    mnuCloseItem.Enabled = True
End If
```

قلب روتین رمزنگاری دستور Encrypt = Encrypt & Chr(Asc(letter) + 1) است. این دستور ابتدا کُد اَسکی کاراکتر (متغیر letter) را بدست آورده، یکی بـه آن اضـافه مـی‌کند، و دوبـاره آنـرا بصورت یک کاراکتر به رشتهٔ رمز (متغیر Encrypt) می‌چسباند.

۲ به روال mnuOpenItem_Click بروید، تا روش از رمز خارج کردن فایل رمز شده را ببینید. طرز کار این روال بسیار شبیه mnuSaveAsItem_Click است، با این فرق که در اینجا از کُد اَسکی کاراکترها یکی کم می‌شود. کُد کامل این روال را در زیر می‌بینید:

```
Dim AllText, LineOfText As String
Dim i, charsInFile As Short
Dim letter As Char
Dim Decrypt As String = ""

OpenFileDialog1.Filter = "Text files (*.TXT)|*.TXT"
OpenFileDialog1.ShowDialog()          'display Open dialog box
If OpenFileDialog1.FileName <> "" Then
    Try     'open file and trap any errors using handler
        FileOpen(1, OpenFileDialog1.FileName, OpenMode.Input)
        Do Until EOF(1)          'read lines from file
            LineOfText = LineInput(1)
            'add each line to the AllText variable
            AllText = AllText & LineOfText & vbCrLf
        Loop

        'now, decrypt string by subtracting one from ASCII code
        charsInFile = AllText.Length       'get length of string
        For i = 0 To charsInFile - 1       'loop once for each char
            letter = AllText.Substring(i, 1)          'get character
            Decrypt = Decrypt & Chr(Asc(letter) - 1)          'subtract 1
        Next i      'and build new string
        txtNote.Text = Decrypt          'then display converted string
        lblNote.Text = OpenFileDialog1.FileName
        txtNote.Select(1, 0)      'remove text selection
        txtNote.Enabled = True          'allow text cursor
        mnuCloseItem.Enabled = True          'enable Close command
        mnuOpenItem.Enabled = False          'disable Open command
    Catch
        MsgBox("Error opening file. It might be too big.")
    Finally
        FileClose(1)          'close file
    End Try
End If
```

این روتین ساده ممکنست همهٔ آن چیزی باشد که برای مخفی کردن اطلاعات خود نیاز دارید؛ ولی این روش بسیار ابتدایی است، و به آسانی کشف می‌شود. فردی که در رمزنگاری تجربه دارد، با کمی دقت می‌تواند بفهمد که این یک الگوریتم جابجایی سادهٔ حروف (ASCII code shift) است. در این الگوریتم کُد اَسکی هر کاراکتر باندازهٔ مشخصی به جلو یا عقب منتقل (جابجا) می‌شود. پیدا کردن میزان جابجایی برای یک خبرهٔ رمزنگاری به سادگی آب خوردن است، و بعد از آن دیگر رمز لو رفته است!. پس چارهٔ کار چیست؟

یک گام فراتر: رمزنگاری با عملگر Xor

رمزنگاری با الگوریتم جابجایی برای متون ساده کاملاً مناسب است، ولی باید دقت کنید که مقدار جابجایی زیاد بزرگ نباشد، چون در اینصورت برای خارج کردن فایل از حالت رمز به مشکل برخورد می‌کنید. برای مثال، اگر مقدار جابجایی را 500 بگیرید، و بخواهید حرف "A" را (که کُد اَسکی آن 65 است) رمز کنید، عدد 565 بدست می‌آید که در تابع Chr باعث خطا خواهد شد.

راه حل این مشکل تبدیل حروف به عدد و به رمز در آوردن این اعداد است، چون دیگر با مشکل اعداد بزرگ مواجه نخواهید بود. در این روش، می‌توان با انجام محاسبات پیچیده (مانند ضرب، لگاریتم و غیره) روی اعداد رمزهای بسیار پیچیده تولید کرد.

یکی از ابزارهایی که کاربرد بسیار وسیع در رمزنگاری دارد، عملگر Xor (eXclusive Or) است. ویژگی مهمی که این عملگر را برای رمزنگاری مناسب کرده، اینست که اگر یک عدد را دو بار متوالی با عددی دیگر Xor کنید، دوباره به همان عدد اولیه خواهید رسید. به یک مثال توجه کنید. دستور زیر عدد 50 را با کُد اَسکی حرف "A" (65) Xor می‌کند؛ حاصل این عمل عدد 115 است.

```
MsgBox(Asc("A") Xor 50)
```

حال اگر این عدد را دوباره با 50 Xor کنیم،

```
MsgBox(115 Xor 50)
```

به عدد 65 (کُد اَسکی حرف "A") می‌رسیم! رفتار جالب عملگر Xor مبنای ایجاد تعداد زیادی الگوریتمهای رمزنگاری پیچیده و معروف بوده است.

در زیر اجرای برنامهٔ Xor Encryption را می‌بینید، که در آن از عملگر Xor برای بـه رمـز در آوردن مـتن موردنظر استفاده کرده‌ایم.

به رمز در آوردن متن با عملگر Xor

۱ پروژهٔ Xor Encryption را (که در پوشهٔ c:\vbnetsbs\chap12\xor encryption قرار دارد) باز کنید.

۲ برنامه را اجرا کنید.

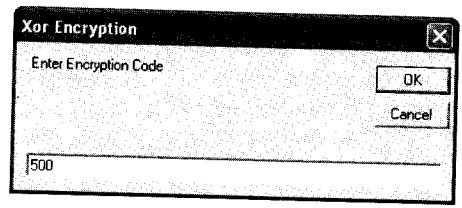
۳ متن زیر (یا هر متن دیگری که مایلید) را در جعبه متن وارد کنید:

Rothair's Edict (Lombard Italy, c. 643)

296. On Stealing Grapes. He who takes more than three grapes from another man's vine shall pay six soldi as compensation. He who takes less than three shall bear no guilt.

٤ فرمان Save Encrypted File As را از منوی File انتخاب کرده، و فایل را با نام **oldlaws.txt** در پوشهٔ c:\vbnetsbs\chap12 ذخیره کنید. برنامه عددی از شما می‌گیرد که متن را بر اساس آن رمز خواهد کرد. (این عدد را جایی یادداشت کنید، چون بدون آن امکان بازگرداندن فایل وجود ندارد.)

۵ عدد **500** را وارد کرده، و Enter را بزنید.



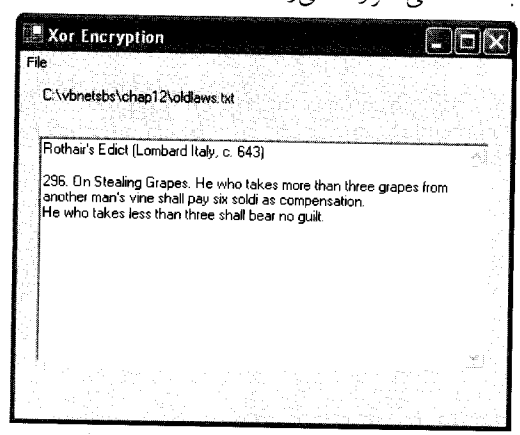
ویژوال بیسیک حروف متن را با عددی که وارد کرده‌اید، Xor کرده و اعداد حاصله را روی دیسک ذخیره می‌کند. بعد از این کار هیچ تغییری در ظاهر برنامه مشاهده نخواهید کرد، اما مطمئن باشید که فایل رمز شده روی دیسک ذخیره شده است.

۶ با فرمان Close (از منوی File)، فایل را ببندید. حال باید بتوانیم فایل رمز شده را دوباره باز کنیم.

۷ فرمان File|Open Encrypted File را اجرا کنید.

۸ (در پوشهٔ c:\vbnetsbs\chap12) روی فایل oldlaws.txt دو-کلیک کنید.

۹ در دیالوگی که ظاهر می‌شود، عدد **500** (و یا هر عدد دیگری که در مرحلهٔ ۵ انتخاب کرده بودید) را وارد کرده و دکمهٔ OK را کلیک کنید. برنامه مجدداً این عدد را با اعداد موجود در فایل Xor کرده، و به کُد اَسکی حروف می‌رسد.



۱۰ با انتخاب فرمان File|Exit ، برنامه را ببندید.

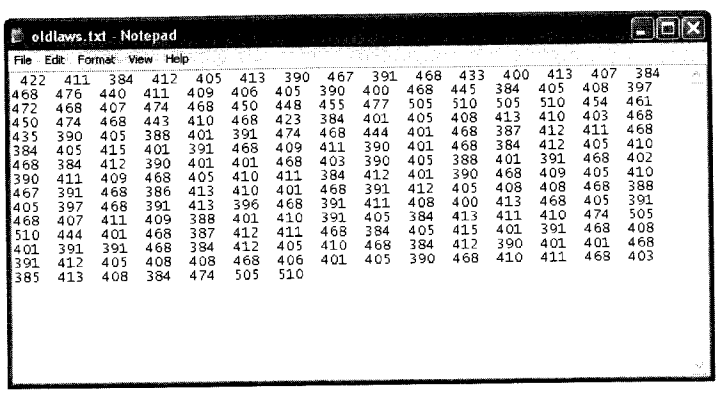
بررسی کُد برنامهٔ Xor Encryption

از عملگر Xor در روالهای mnuSaveAsItem_Click و mnuOpenItem_Click به یکسان استفاده کرده‌ایم. کُد روال mnuSaveAsItem_Click را در زیر می‌بینید (دستورات مهم با فونت ضخیم مشخص شده‌اند):

```
Dim letter As Char
Dim strCode As String
Dim i, charsInFile, Code As Short

SaveFileDialog1.Filter = "Text files (*.txt)|*.txt"
SaveFileDialog1.ShowDialog()
If SaveFileDialog1.FileName <> "" Then
    strCode = InputBox("Enter Encryption Code")
    If strCode = "" Then Exit Sub          'if cancel clicked
    'save text with encryption scheme
    Code = CShort(strCode)
    charsInFile = txtNote.Text.Length
    FileOpen(1, SaveFileDialog1.FileName, OpenMode.Output)
    For i = 0 To charsInFile - 1
        letter = txtNote.Text.Substring(i, 1)
        'convert to number w/ Asc, then use Xor to encrypt
        Print(1, Asc(letter) Xor Code)          'and save in file
    Next
    FileClose(1)
    mnuCloseItem.Enabled = True
End If
```

در دستور Print ابتدا با تابع Asc کُد اَسکی هر کاراکتر را بدست آورده، و سپس این عدد را با متغیر Code (که آنرا از کاربر گرفته، و به عددی از نوع Short تبدیل کرده‌ایم) Xor می‌کنیم؛ عددی که بدست می‌آید در فایل خروجی نوشته خواهد شد. (در اینجا بجای تابع PrintLine از Print استفاده کرده‌ایم، چون این تابع هر بار فقط یک کاراکتر را در فایل می‌نویسد.)

فایل خروجی این روتین دیگر هیچ کاراکتری در خود ندارد، بلکه تماماً از عدد تشکیل می‌شود. در زیر فایل oldlaws.txt را در برنامهٔ NotePad می‌بینید:



کُد روال mnuOpenItem_Click را نیز در زیر ملاحظه می‌کنید (باز هم دستورات مهم با فونت ضخیم مشخص شده‌اند):

```
Dim ch As Char
Dim strCode As String
Dim Code, Number As Short
Dim Decrypt As String = ""

OpenFileDialog1.Filter = "Text files (*.TXT)|*.TXT"
OpenFileDialog1.ShowDialog()          'display Open dialog box
If OpenFileDialog1.FileName <> "" Then
   Try 'open file and trap any errors using handler
      strCode = InputBox("Enter Encryption Code")
      If strCode = "" Then Exit Sub          'if cancel clicked
      Code = CShort(strCode)
      FileOpen(1, OpenFileDialog1.FileName, OpenMode.Input)
      Do Until EOF(1)          'read lines from file
         Input(1, Number)          'read encrypted numbers
         ch = Chr(Number Xor Code)          'convert with Xor
         Decrypt = Decrypt & ch          'and build string
      Loop
      txtNote.Text = Decrypt          'then display converted string
      lblNote.Text = OpenFileDialog1.FileName
      txtNote.Select(1, 0)          'remove text selection
      txtNote.Enabled = True          'allow text cursor
      mnuCloseItem.Enabled = True          'enable Close command
      mnuOpenItem.Enabled = False          'disable Open command
   Catch
      MsgBox("Error opening file.")
   Finally
      FileClose(1)          'close file
   End Try
End If
```

این روتین بعد از گرفتن کُد رمز و باز کردن فایل رمز شده، اعداد درون فایل را یکی یکی (با تابع Input -که برای اولین بار می‌بینید) خوانده، و در متغیر Number ذخیره می‌کند. سپس این عدد با کُد رمز (متغیر Code) Xor می‌شود، تا کُد اَسکی کاراکتر متناظر با آن بدست آید، و با تابع Chr به حرف تبدیل شود:

```
Input(1, Number)          'read encrypted numbers
ch = Chr(Number Xor Code)          'convert with Xor
Decrypt = Decrypt & ch          'and build string
```

روتینهای رمزنگاری از نظر آموزشی بسیار مفید هستند، چون در آنها بصورت وسیعی از تکنیکهای پردازش متن استفاده می‌شود، و این تکنیکها جزء مهارتهای کلیدی و بسیار مهم ویژوال بیسیک محسوب می‌شوند. فقط مواظب باشید کلید رمز را گم نکنید!

مرجع سریع فصل ۱۲

برای ...	انجام دهید
باز کردن یک فایل متنی	از تابع FileOpen استفاده کنید: `FileOpen(filenum, filename, OpenMode.Input)`
خواندن یک خط از فایل متنی	از تابع LineInput استفاده کنید: `LineOfText = LineInput(filenum)`

چک کردن آخر فایل	از تابع EOF استفاده کنید:
	`Do Until EOF(filenum)`
	`  LineOfText = LineInput(filenum)`
	`  ...`
	`Loop`

بستن یک فایل باز	از تابع FileClose استفاده کنید: `FileClose(filenum)`

نمایش یک فایل متنی	با تابع LineInput فایل را خط به خط خوانده و در یک متغیر رشته‌ای ذخیره می‌کنیم:
	`Do Until EOF(filenum)`
	`  LineOfText = LineInput(filenum)`
	`  AllText = AllText & LineOfText & vbCrLf`
	`Loop`

نمایش دیالوگ Open	از متد ShowDialog کنترل OpenFileDialog استفاده کنید: `OpenFileDialog1.ShowDialog()`
ایجاد یک فایل متنی جدید	از تابع FileOpen استفاده کنید: `FileOpen(filenum, filename, OpenMode.Output)`
نمایش دیالوگ Save As	از متد ShowDialog کنترل SaveFileDialog استفاده کنید: `SaveFileDialog1.ShowDialog()`
ذخیره کردن متن در فایل	از تابع Print یا PrintLine استفاده کنید: `PrintLine(filenum, TextToBeSaved)`
تبدیل کاراکتر به کُد آسکی	از تابع Asc استفاده کنید: `Code = Asc("A")  'Code equals 65`
تبدیل کُد آسکی به کاراکتر	از تابع Chr استفاده کنید: `Letter = Chr(65)  'Letter equals "A"`

استخراج رشته‌های کوچک از یک رشتهٔ بزرگتر	از تابع SubString یا Mid استفاده کنید:
	`Cols = "First Second Third"`
	`Middle = Cols.SubString(6, 6)  'Middle = "Second"`

برای ...	انجام دهید

رمز کردن متن	کاراکترهای متن را به کُد آسکی تبدیل کرده، و سپس با یک عدد ثابت (رمز) Xor کنید:
	`For i = 0 To LengthOfFile - 1`
	`  letter = TextToBeCoded.SubString(i, 1)`
	`  Print(filenum, Asc(letter) Xor Code)`
	`Next`

از رمز در آوردن متن	اعداد را یکی یکی از فایل رمز شده خوانده، و بعد از Xor کردن با یک عدد ثابت (رمز) و استخراج کُد آسکی، آنرا به کاراکتر تبدیل کنید:
	`Do Until EOF(filenum)`
	`  Input(filenum, NumRead)`
	`  ch = Chr(NumRead Xor Code)`
	`  DecryptedText = DecryptedText & ch`
	`Loop`

فصل

۱۳

اتوماسیون برنامه‌های آفیس و مدیریت پروسس‌ها

در این فصل یاد می گیرید چگونه :

- ✔ با ابزار Object Browser اشیاء برنامه را بررسی کنید.
- ✔ برای محاسبهٔ اقساط وام از Microsoft Excel کمک بگیرید.
- ✔ در برنامه‌های ویژوال بیسیک .NET باکاربرگ‌های Excel کار کنید.
- ✔ با استفاده از کلاس Process برنامه‌های ویندوز را اجرا کنید و ببندید.

در این فصل یاد می‌گیرید چگونه برنامه‌های آفیس XP را با ویژوال بیسیک .NET کنترل کنید. با کاوشگر شیء (Object Browser - یکی از ابزارهای ویژوال استودیو برای بررسی اشیایی که یک برنامه در اختیار دارد) نیز آشنا خواهید شد. اتوماسیون (Automation - استفاده از اشیاء برنامه‌های کاربردی ویندوز در یک برنامهٔ دیگر) مفهوم محوری این فصل است.

تکنیک دیگری که در این فصل یاد می‌گیرید، کنترل پروسس (Process)های سیستم است؛ بویژه برنامه‌ای می‌نویسیم، و با متدهای Start و CloseMainWindow برنامه‌های دیگر را اجرا کرده و یا می‌بندیم (در فصل‌های ۳ و ۱۱ بطور اجمالی با متد Start آشنا شدید).

تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- در ویژوال بیسیک ۶ برای اضافه کردن اشیاء برنامه‌های دیگر بایستی از کنترلی بنام OLE Container استفاده می‌کردیم. در ویژوال بیسیک .NET روش بکارگیری اشیاء اتوماسیون فرق کرده است.
- در ویژوال بیسیک ۶ کنترلهای اکتیوایکس بر اساس تکنولوژی COM (Component Object Model) ساخته می‌شدند. کنترلهای ویژوال بیسیک .NET دیگر با این تکنولوژی ساخته نمی‌شوند، اما همچنان می‌توانید از آنها در برنامه‌های ویژوال بیسیک .NET استفاده کنید (کمی بعد، خواهید دید چگونه).
- از طریق اتوماسیون می‌توان از برنامه‌ها و اشیاء آفیس XP (که همچنان به تکنولوژی COM متکی هستند) در برنامه‌های ویژوال بیسیک .NET استفاده کرد. اما ویژوال بیسیک .NET دیگر از پیوند دیر-رس (late binding) پشتیبانی نمی‌کند. بعبارت دیگر، پیوند این اشیاء باید در هنگام کامپایل برنامه برقرار شود (پیوند زود-رس: early binding)، نه در هنگام اجرای برنامه (پیوند دیر-رس).
- در ویرایشهای قبلی ویژوال بیسیک، برای اجرای یک برنامهٔ ویندوز باید از تابع Shell استفاده می‌کردید. کنترل برنامه‌های دیگر ویندوز (اجرا و بستن) در ویژوال بیسیک .NET بسیار ساده‌تر شده، و کلاس خاصی بنام Process برای این کار طراحی شده است.

اتوماسیون: برنامه‌نویسی با اشیاء برنامه‌های دیگر

اتوماسیون تکنولوژیی است که بر اساس استاندارد COM ایجاد شده، و وظیفهٔ آن برقراری ارتباط بین برنامه‌های کاربردی مختلف است. برنامه‌های ویندوز برای پشتیبانی از اتوماسیون، کلکسیون اشیاء خود را در معرض استفادهٔ دیگران قرار می‌دهند. برنامه‌ای که اشیاء خود را در اختیار برنامه‌های دیگر می‌گذارد، برنامهٔ میزبان (server) ، و برنامه‌ای که از این اشیاء استفاده و آنها را کنترل می‌کند، برنامهٔ مشتری (client) نامیده می‌شود. با اینکه ویژوال استودیو .NET بر اساس استاندارد COM بنا نشده، اما همچنان از این مدل پشتیبانی می‌کند، و می‌توانید از اشیاء COM در برنامه‌های ویژوال بیسیک .NET استفاده کنید.

برنامه‌هایی که در حال حاضر می‌توانید از آنها بعنوان میزبان یا مشتری اتوماسیون استفاده کنید، عبارتند از:

- ویژوال استودیو .NET ، ویژوال بیسیک ۶
- Word 97 ، Word 2000 ، Word 2002
- Excel 5.0 ، Excel 95 ، Excel 97 ، Excel 2000 ، Excel 2002
- PowerPoint 97 ، PowerPoint 2000 ، PowerPoint 2002
- Project 95 ، Project 97 ، Project 2000
- Outlook 97/98 ، Outlook 2000 ، Outlook 2002

نکته

شرکت میکروسافت امتیاز استفاده از زبان برنامه‌نویسی VBA (Visual Basic for Application) را به شرکتهای نرم‌افزاری دیگر نیز می‌فروشد، بنابراین برنامه‌های زیادی را در بازار خواهید یافت که از اتوماسیون و تکنیکهای برنامه‌نویسی ویژوال بیسیک پشتیبانی می‌کنند.

اتوماسیون در ویژوال بیسیک

با ویژوال بیسیک .NET می‌توانید برنامه‌های میزبان و مشتری اتوماسیون بسازید. نوشتن برنامه‌های میزبان اتوماسیون از حیطهٔ این کتاب خارج است، اما ایجاد برنامه‌های مشتری اتوماسیون کاریست نسبتاً ساده.

نکته

تمام برنامه‌های آفیس XP (Excel 2002 ، Word 2002 ، Access 2002 ، PowerPoint 2002 و Outlook 2002) همگی می‌توانند میزبان اتوماسیون باشند؛ ولی امکانات و اشیایی که ارائه می‌کنند، خاص هر یک از آنهاست، که برای استفاده از آنها بایستی به سیستم کمک هر یک از این برنامه‌ها مراجعه کنید. برای دیدن اشیاء و متدهایی که هر برنامه در معرض استفادهٔ دیگران گذاشته، می‌توانید از کاوشگر شیء (Object Browser) نیز کمک بگیرید.

در قسمتهای آینده برنامه هایی می‌نویسیم، و در آنها با Microsoft Excel 2002 کار می‌کنیم. در کار با این برنامه‌ها متوجه خواهید شد که اشیاء، خواص و متدهای هر برنامه (معمولاً) ارتباط نزدیکی با منوهای آن دارد. مهارتهایی که در این تمرین‌ها کسب می‌کنید، در برنامه‌های Word ، PowerPoint و Outlook (و هر برنامه‌ای که از مدل COM پشتیبانی کند) نیز به یکسان کاربرد دارد.

کاوشگر شیء - Object Browser

ابزار کاوشگر شیء دو کاربرد مهم دارد:

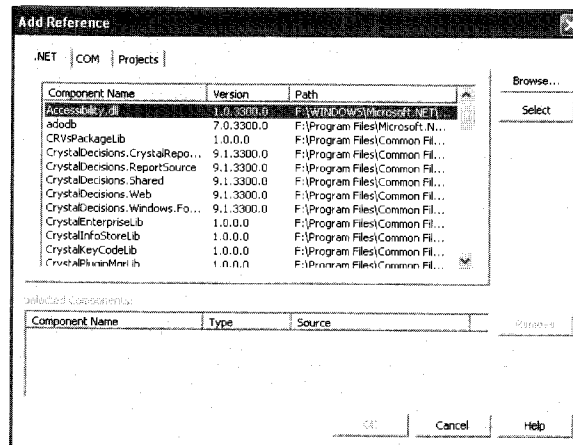
- نمایش اشیاء، خواص و متدهایی که یک برنامه از آنها استفاده می‌کند.
- نمایش اشیاء، خواص و متدهایی که برنامه‌های دیگر سیستم در معرض استفاده قرار داده‌اند.

در تمرین زیر اشیاء اتوماسیون Excel 2002 را با کاوشگر شیء مورد بررسی قرار می‌دهیم.

بررسی اشیاء Excel 2002 باکاوشگر شیء

۱ ویژوال استودیو .NET را باز کرده، و یک پروژهٔ جدید Visual Basic Windows Application بنام **My Excel Automation** در پوشهٔ c:\vbnetsbs\chap13 ایجاد کنید.

۲ فرمان Add Reference را از منوی Project اجرا کنید، تا پنجرهٔ Add Reference باز شود:



در پنجرهٔ Add Reference سه برگه می‌بینید، که هر یک از آنها نشاندهندهٔ طیفی از اشیاء موجود در سیستم است: .NET ، COM ، و Project . در برگهٔ .NET اشیایی را می‌بینید که منطبق با استاندارد .NET هستند (مانند اشیاء ویژوال استودیو .NET ، اشیاء برنامهٔ Crystal Reports و غیره). در برگهٔ COM اشیاء منطبق با مدل COM را می‌بینید (مانند اشیاء برنامه‌های Microsoft Office)؛ برگهٔ Project نشاندهندهٔ اشیائیست که برنامهٔ شما در اختیار دیگران می‌گذارد. اضافه کردن رفرنس (reference) به برنامه آنرا چندان بزرگتر نمی‌کند، اما باعث طولانی‌تر شدن زمان بار شدن و کامپایل برنامه خواهد شد (خود ویژوال بیسیک تا زمانی که از آن نخواهید، هیچ رفرنسی به برنامه اضافه نمی‌کند).

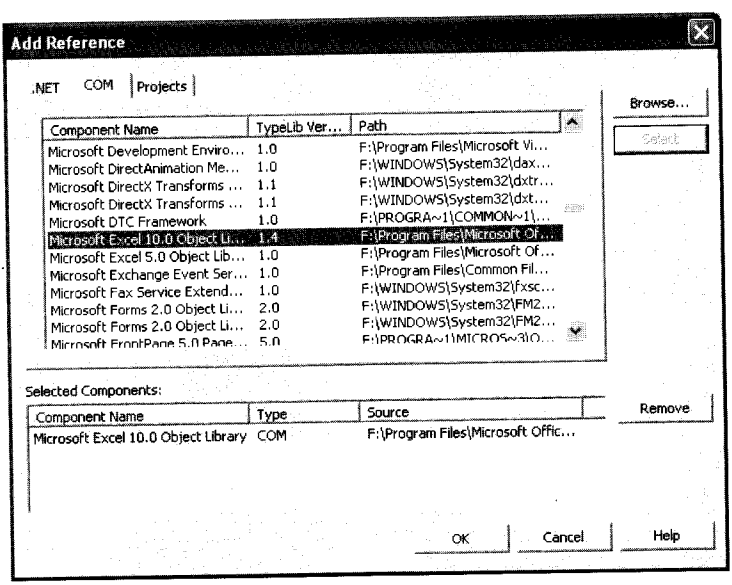
۳ روی برگهٔ COM کلیک کنید، تا لیستی از اشیاء منطبق با مدل COM را که در سیستم‌تان موجود است، ببینید (این لیست در هر کامپیوتر فرق می‌کند).

٤ آنقدر در لیست اشیاء برگهٔ COM پائین بروید، تا به رفرنس Microsoft Excel 10.0 Object Library برسید (لیست اشیاء در پنجرهٔ Add Reference بترتیب حروف الفبا مرتب شده است). لیست اشیاء برگهٔ COM (مخصوصاً قسمت کتابخانه‌های Microsoft) معمولاً بسیار مفصل است.

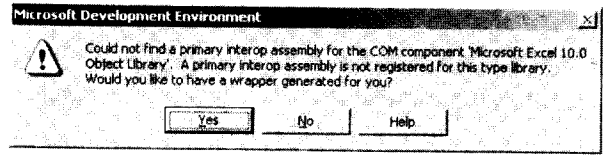
نکته

کتابخانهٔ Microsoft Excel 10.0 Object Library مربوط به آفیس XP است؛ اگر آفیس XP ندارید، می‌توانید با کاوشگر شیء اشیاء برنامه‌های دیگر را بررسی کنید.

۵ بعد از انتخاب Microsoft Excel 10.0 Object Library ، روی دکمهٔ Select کلیک کنید:



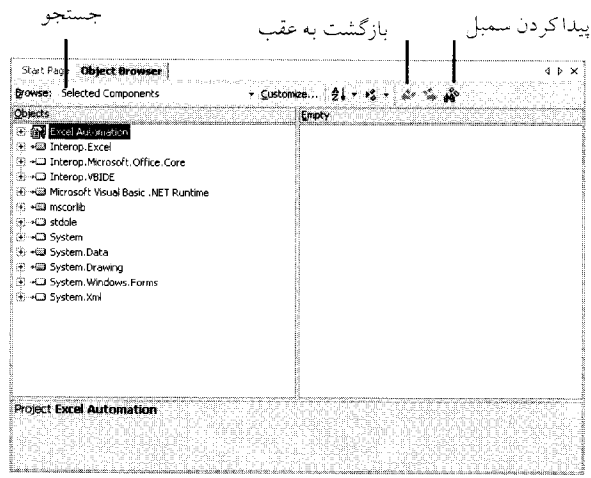
۶ OK را کلیک کنید، تا پنجرهٔ Add Reference بسته شده، و رفرنس Microsoft Excel 10.0 Object Library به برنامه اضافه شود. احتمال دارد پیامی مانند زیر ببینید:



از آنجائیکه ویژوال استودیو .NET دیگر ذاتاً از اشیاء COM پشتیبانی نمی‌کند، باید آنها را در "لفافی" بپوشاند تا با سیستم اشیاء ویژوال استودیو سازگار باشند (پیام فوق هم به همین موضوع اشاره می‌کند).

۷ اگر این پیام را دیدید، Yes را کلیک کنید؛ با این کار اشیاء کتابخانهٔ Microsoft Excel 10.0 Object Library را در کاوشگر راه‌حل خواهید دید. اکنون آماده‌ایم تا از کاوشگر شیء استفاده کنیم.

۸ از منوی View|Other Windows آیتم Object Browser را انتخاب کنید (یا کلید F2 را بزنید)، تا پنجرهٔ کاوشگر شیء باز شود:



کاوشگر شیء اشیاء را بصورت ساختار سلسله مراتبی درختی نمایش می‌دهد: با انتخاب هر شاخه در قاب سمت چپ (قاب Object)، اشیاء ذیل آن شاخه در قاب سمت راست (قاب Members) نشان داده خواهند شد. وقتی یک شیء را در قاب سمت راست انتخاب کنید، در قاب پائین پنجرهٔ کاوشگر شیء دستور بکارگیری آنرا خواهید دید.

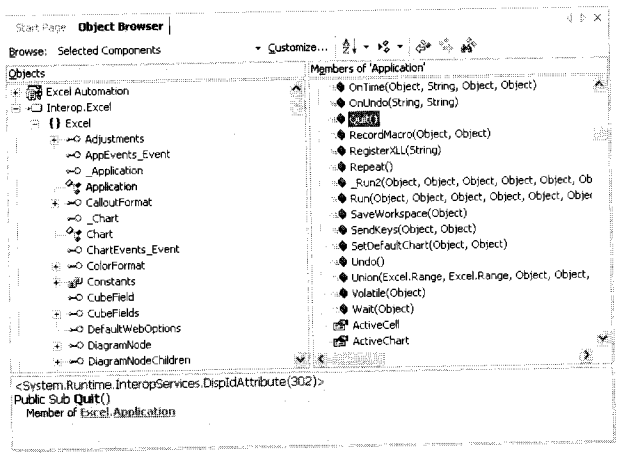
کاوشگر شیء ابزارهایی نیز برای جستجوی سریع و مؤثر در اشیاء نمایش داده شده، دارد. به کمک این ابزارها می‌توانید اشیاء، خواص و متدهای موردنظر را بسادگی پیدا کنید.

۹ ابتدا علامت + کنار کتابخانهٔ Interop.Excel و سپس علامت + کنار کتابخانهٔ Excel را کلیک کنید، تا لیستی از اشیاء اتوماسیون Excel در قاب Object (قاب سمت چپ) ظاهر شود.

۱۰ از لیست Excel ، شیء Application را انتخاب کنید، تا اشیاء، خواص و متدهای آن را در قاب Members ببینید. اینها اشیاء، خواص و متدهایی هستند که Excel برای پردازش اطلاعات کاربرگ‌های خود از آنها استفاده می‌کند.

۱۱ از لیست Members ، متد Quit را انتخاب کنید (شاید لازم باشد کمی در این لیست پائین بروید)؛ شکل زیر را ببینید.

با انتخاب متد Quit ، طرز استفاده از آن را در قاب پائین کاوشگر شیء ملاحظه خواهید کرد. این متد برنامهٔ Excel را می‌بندد. اگر متد انتخاب شده دارای آرگومان باشد (که متد Quit چنین نیست)، تعداد و نوع آنها را هم خواهید دید (شیء مادر متد یا خاصیت انتخاب شده نیز در همین قسمت نشان داده می‌شود).



۱۲ اگر مایلید، به بررسی کتابخانه‌های دیگر ادامه دهید.

۱۳ در پایان، کاوشگر شیء را ببندید.

اکنون نوبت آنست که از فرمانهای اتوماسیون Excel در برنامهٔ خود استفاده کنیم.

اتوماسیون Excel در برنامه‌های ویژوال بیسیک

برای اجرای فرمانهای Excel در برنامه‌های ویژوال بیسیک مراحل ذیل را بایستی طی کنید. (از آنجائیکه این مراحل برای سایر برنامه‌ها نیز قابل بکارگیری است، آنها را بعنوان رهیافت کلی اتوماسیون تلقی کنید.)

مرحلهٔ اول از طریق پنجرهٔ Add Reference ، رفرنس کتابخانهٔ موردنظر را به پروژه اضافه کنید.

مرحلهٔ دوم در برنامهٔ ویژوال بیسیک و در روالی که می‌خواهید عمل اتوماسیون انجام گیرد، با دستور Dim شیء اتوماسیون را تعریف کنید. سپس، با دو تابع CreateObject و CType شیء موردنظر را ایجاد کرده، و نوع آنرا تغییر دهید:

```
Dim xlApp As Excel.Application
xlApp = CType(CreateObject("Excel.Application"), Excel.Application)
```

در ویژوال بیسیک ۶ می‌توانستید یک شیء از نوع Object تعریف کرده، و سپس هنگام اجرای برنامه نوع اتوماسیون آنرا تعیین کنید. این روش، که به آن پیوند دیر-رس گفته می‌شود، در ویژوال بیسیک .NET توصیه نمی‌شود. در ویژوال بیسیک .NET ارتباط شیء اتوماسیون و برنامهٔ وابسته باید هنگام کامپایل برنامه صورت گیرد (روشی که به آن پیوند زود-رس می‌گویند). CType تابعیست که نوع برنامهٔ اتوماسیون را در هنگام کامپایل شدن پروژه برمی‌گرداند.

مرحلهٔ سوم خواص و متدهای برنامهٔ اتوماسیون را بکار بگیرید (برای دیدن نحوهٔ استفاده از این متدها و خواص، به کاوشگر شیء مراجعه کنید). در کُد زیر از متد Pmt (که یکی از متدهای Excel برای محاسبهٔ اقساط وام است) استفاده کرده‌ایم:

```
Dim LoanPayment As Single
LoanPayment = xlApp.WorksheetFunction.Pmt _
        (txtInterest.Text / 12, txtMonths.Text, txtPrincipal.Text)
MsgBox("The monthly payment is " & _
        Format(Abs(LoanPayment), "$#.##"), , "Mortgage")
```

مرحلهٔ چهارم پس از پایان کار، با متد Quit برنامهٔ اتوماسیون را ببندید (مگر اینکه همچنان با آن کار داشته باشید):

```
xlApp.Quit()
```

در تمرین زیر با استفاده از متد Pmt اقساط یک وام مسکن را محاسبه خواهیم کرد. این یک برنامهٔ ویژوال بیسیک است، که از امکانات محاسباتی Excel بهره خواهد برد.

نکته

برای اجرای این برنامه باید Excel 2002 داشته باشید. می‌توان یک برنامهٔ مشتری اتوماسیون را طوری نوشت که با ویرایشهای مختلف میزبان اتوماسیون کار کند، اما این کار بهیچوجه ساده نیست. به همین دلیل استفاده از اتوماسیون در برنامه‌هایی که باید بصورت گسترده توزیع شوند، توصیه نمی‌شود.

برنامه‌ای برای محاسبهٔ اقساط وام

۱ به فرم برنامهٔ My Excel Automation بروید.

۲ در بالای فرم یک برچسب (برای نوشتن عنوان برنامه) رسم کنید.

۳ زیر برچسب عنوان و کمی متمایل به راست، سه جعبه متن بکشید. از این سه جعبه متن برای وارد کردن اطلاعات لازم استفاده خواهیم کرد.

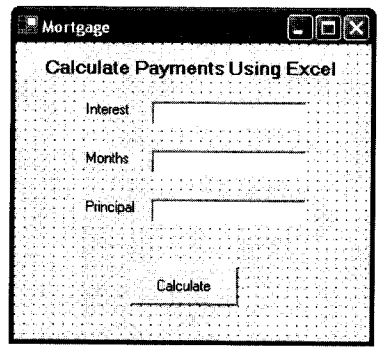
٤ کنار هر جعبه متن (و سمت چپ آن) یک برچسب رسم کنید. این برچسب‌ها عنوان هر جعبه متن را نمایش خواهند داد.

۵ در پائین فرم هم یک دکمه قرار دهید.

۶ خواص فرم برنامه و کنترلهای آنرا با توجه به جدول زیر ست کنید:

شیء	خاصیت	مقدار
Label1	Font	Microsoft Sans Serif, Bold, 11 point
	Text	"Calculate Payments Using Excel"
	TextAlign	MiddleCenter
TextBox1	Name	txtInterset
	Text	(خالی)
TextBox2	Name	txtMonths
	Text	(خالی)
TextBox3	Name	txtPrincipal
	Text	(خالی)
Label2	Text	"Interset"
Label3	Text	"Months"
Label4	Text	"Principal"
Button1	Name	btnCalculate
	Text	"Calculate"
Form1	Text	"Mortgage"

شکل نهایی فرم را در زیر ملاحظه می‌کنید:



۷ روی دکمهٔ Calculate دو-کلیک کنید، تا روال رویداد btnCalculate_Click در ادیتور کُد باز شود.

۸ دستورات زیر را در این روال بنویسید:

```
Dim xlApp As Excel.Application
Dim LoanPayment As Single
xlApp = CType(CreateObject("Excel.Application"), Excel.Application)
LoanPayment = xlApp.WorksheetFunction.Pmt _
        (txtInterest.Text / 12, txtMonths.Text, txtPrincipal.Text)
MsgBox("The monthly payment is " & _
        Format(Abs(LoanPayment), "$#.##"), , "Mortgage")
xlApp.Quit()
```

در این کُد ابتدا یک شیء اتوماسیون Excel.Application بنام xlApp تعریف کرده‌ایم (البته این کار فقط بعد از اضافه کردن رفرنس کتابخانهٔ Microsoft Excel 10.0 به برنامه امکانپذیر شده است).

نکته

اگر بعد از نوشتن کُد فوق، زیر Excel.Application یک خط زیگزاگ آبی رنگ ظاهر شد، به احتمال زیاد اضافه کردن رفرنس Microsoft Excel 10.0 Object Library را جا انداخته‌اید. برگردید، و از طریق پنجرهٔ Add Reference این رفرنس را به پروژه اضافه کنید.

سپس یک وهله از شیء Excel.Application ایجاد شده، و به متغیر xlApp نسبت داده می‌شود؛ بعد از این کار، می‌توانیم از توابع Excel در برنامه استفاده کنیم (توجه کنید که چگونه مقدار آرگومانهای متد Pmt را از جعبه متن‌ها گرفته‌ایم). از آنجائیکه مقدار برگشتی متد Pmt منفی است، آنرا با تابع Abs (قدر مطلق) به یک عدد مثبت تبدیل کرده، و بعد از فرمت کردن آن با دو رقم اعشار، آنرا با تابع MsgBox نمایش داده‌ایم. اگر هر یک از آرگومانهای متد Pmt وجود نداشته باشد (یا 0 باشد)، برنامه با خطا متوقف خواهد شد. (آیا می‌توانید به کمک یک ساختار تصمیم‌گیری یا روتین مقابله با خطا، مانع بروز این وضعیت شوید؟) در پایان هم، با متد Quit برنامهٔ میزبان اتوماسیون را بسته‌ایم. (البته کاربر هرگز متوجه باز شدن برنامهٔ Excel نخواهد شد، چون برنامه‌های اتوماسیون معمولاً بصورت مخفی و در پس‌زمینه اجرا می‌شوند.)

۹ همانطور که دیدید، برای تبدیل مقدار برگشتی متد Pmt از تابع Abs استفاده کردیم. تابع Abs عضو کلاس System.Math است، بنابراین باید آنرا به فرم اضافه کنیم. برای اینکار، به بالای فرم رفته، و دستور زیر را آنجا بنویسید:

```
Imports System.Math
```

(بعد از نوشتن این دستور، خط زیگزاگ آبی از زیر Abs(LoanPayment) نیز محو خواهد شد.)

اکنون آماده‌ایم برنامه را اجرا کرده، و اتوماسیون Excel را در عمل ببینیم.

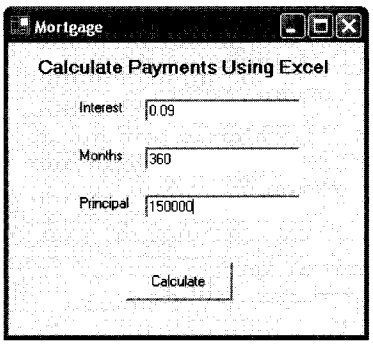
اجرای برنامهٔ Excel Automation

۱ برنامهٔ My Excel Automation را اجرا کنید.

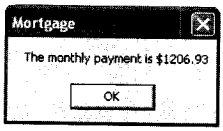
۲ در جعبه متن Interest عدد **0.09** را وارد کنید.

۳ در جعبه متن Months عدد **360** را وارد کنید.

٤ در جعبه متن Principal عدد **150000** را وارد کنید. (شکل زیر را ببینید:)



۵ دکمهٔ Calculate را کلیک کنید، تا برنامه اقساط ماهیانهٔ وام 150,000 دلاری با بهرهٔ 9% و دورهٔ بازپرداخت 360 ماه (30 سال) را محاسبه کند. همانطور که ملاحظه می‌کنید اقساط ماهیانهٔ این وام 1206.93 دلار است (البته بدون احتساب مالیات، بیمه، و مخارج دیگر!).



۶ OK را کلیک کنید. چند محاسبهٔ دیگر با برنامه انجام داده، و با طرز کار آن بیشتر آشنا شوید.

۷ در پایان با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

همانطور که گفتم، در این برنامه کاربر هرگز اجرای Excel را نخواهد دید؛ اما این تنها روش کار با Excel نیست. در تمرین زیر با شکل دیگری از اتوماسیون Excel آشنا می‌شوید.

کار با کاربرگ‌های Excel

۱ با فرمان File|Close Solution پروژهٔ My Excel Automation را ببندید، و یک پروژهٔ جدید Visual Basic Windows Application بنام **My Excel Sheet Tasks** در پوشهٔ c:\vbnetsbs\chap13 ایجاد کنید. در این برنامه بعد از ایجاد یک کاربرگ Excel ، چند عدد و متن در سلولهای آن وارد کرده، آنها را فرمت کرده و پس از انجام یک محاسبهٔ ساده، کاربرگ را روی دیسک ذخیره می‌کنیم. (این اعمال جزء مهمترین مهارتهای Excel محسوب می‌شوند.)

۲ با فرمان Project|Add Reference پنجرهٔ Add Reference را باز کنید.

۳ به برگهٔ COM بروید، و بعد از انتخاب کتابخانهٔ Microsoft Excel 10.0 Object Library ، دکمهٔ Select را کلیک کنید؛ با کلیک کردن OK پنجرهٔ Add Reference را ببندید (اگر پیام primary interoperability assembly ظاهر شد، Yes را کلیک کنید).

٤ در وسط فرم یک دکمهٔ بزرگ رسم کنید. خاصیت Text این دکمه را به **Create Worksheet** ست کنید.

۵ خاصیت Text فرم برنامه را نیز به **Excel Worksheet Builder** ست کنید.

۶ روی دکمهٔ Create Worksheet دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد ظاهر شود.

۷ دستورات زیر را در این روال بنویسید:

```
' Declare Excel object variables and create types
Dim xlApp As Excel.Application
Dim xlBook As Excel.Workbook
```

```
Dim xlSheet As Excel.Worksheet
xlApp = CType(CreateObject("Excel.Application"), Excel.Application)
xlBook = CType(xlApp.Workbooks.Add, Excel.Workbook)
xlSheet = CType(xlBook.Worksheets(1), Excel.Worksheet)

' Insert data
xlSheet.Cells(1, 2) = 5000
xlSheet.Cells(2, 2) = 75
xlSheet.Cells(3, 1) = "Total"
' Insert a Sum formula in cell B3
xlSheet.Range("B3").Formula = "=Sum(R1C2:R2C2)"
' Format cell B3 with bold
xlSheet.Range("B3").Font.Bold = True
' Display the sheet
xlSheet.Application.Visible = True
' Save the sheet to c:\vbnetsbs\chap13 folder
xlSheet.SaveAs("C:\vbnetsbs\chap13\myexcelsheet.xls")
' Leave Excel running and sheet open
```

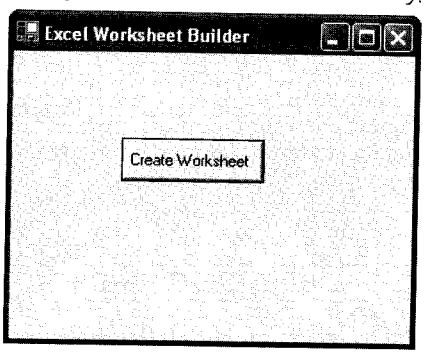
در سه خط اول این کُد متغیرهای لازم را تعریف کرده‌ایم: xlApp (رفرنس برنامهٔ Excel)، xlBook (رفرنس کارکتاب Excel)، و xlSheet (رفرنس کاربرگ Excel). این سه متغیر برای کار با کاربرگهای Excel لازم هستند. در سه دستور بعدی این سه شیء ایجاد شده‌اند (به ارتباط سلسله مراتبی آنها دقت کنید).

دستورات بعدی مقادیر را در سلولهای کاربرگ Excel وارد می‌کنند (به تفاوت عدد و متن توجه کنید). سپس یک فرمول در سلول B3 نوشته‌ایم؛ این فرمول سلولهای B1 (R1C2) و B2 (R2C2) را با هم جمع می‌کند (تابع Sum). پس از آن این سلول با فونت ضخیم فرمت می‌شود. در آخر هم نوبت ذخیره کردن کاربرگ در پوشهٔ c:\vbnetsbs\chap13 است. از آنجائیکه در آخر کار از متد Quit استفاده نکرده‌ایم، Excel بسته نخواهد شد و می‌توانیم همچنان به کار با آن ادامه دهیم.

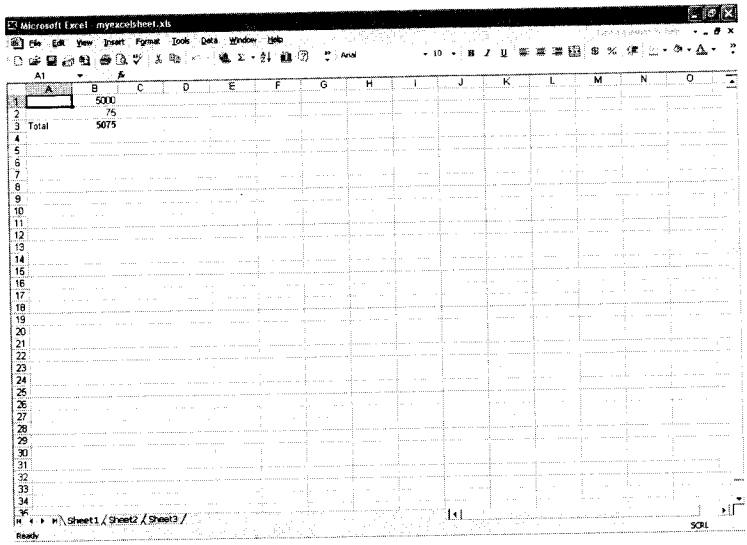
۸ پروژه را ذخیره کنید. (کُد کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap13\excel sheet tasks پیدا کنید.)

اجرای برنامهٔ Excel Sheet Tasks

۱ برنامهٔ My Excel Sheet Tasks را اجرا کنید، تا فرم سادهٔ برنامه ظاهر شود:



۲ دکمهٔ Create Worksheet را کلیک کنید، تا ویژوال بیسیک Excel را اجرا کرده و کارهای خواسته شده را انجام دهد. البته کاربرگ Excel فقط بعد از اجرای دستور xlSheet.Application.Visible = True ظاهر خواهد شد:



توجه کنید که Excel چگونه اتوماسیون خواسته شده را انجام داده است (نام فایل myexcelsheet.xls در میلهٔ عنوان Excel نیز حاکی از ذخیره شدن کاربرگ است).

۳ کمی با Excel کار کرده، و سپس آنرا ببندید. (توجه کنید که برنامهٔ My Excel Sheet Tasks بدون توجه به وضعیت Excel به کار خود ادامه خواهد داد.)

٤ در پایان با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

یک گام فراتر: باز کردن و بستن برنامه‌های ویندوز

در فصل ۳ و ۱۱ برای اجرای کاوشگر وب و نمایش صفحهٔ وب از متد Process.Start استفاده کردیم. با متد Process.Start می‌توان هر برنامه‌ای را که در رجیستری ویندوز ثبت شده باشد، اجرا کرد. مادامیکه این برنامه برای ویندوز شناخته شده باشد، برای اجرای آن حتی لازم نیست مسیر دقیق برنامه مشخص شود (متد Process.Start برای انجام کار خود از اتوماسیون استفاده نمی‌کند). در زیر با این متد برنامهٔ NotePad را اجرا کرده‌ایم:

```
System.Diagnostics.Process.Start("notepad.exe")
```

عیب روش فوق اینست که بعد از اجرای برنامهٔ موردنظر، کنترل آن از دست برنامه‌ای که آنرا اجرا کرده، خارج می‌شود. برای رفع این مشکل بهتر است از کنترلی بنام Process برای اجرای برنامه‌ها استفاده کنیم. در تمرین زیر خواهید دید که چگونه می‌توان با این کنترل یک برنامه را اجرا کرد، و سپس آنرا بست.

کنترل اجرای پروسس Notepad

۱ با فرمان File|Close Solution پروژهٔ My Excel Sheet Tasks را ببندید، و یک پروژهٔ جدید Visual Basic Windows Application بنام **My Start App** در پوشهٔ c:\vbnetsbs\chap13 ایجاد کنید.

۲ دو دکمه روی فرم برنامه قرار دهید.

۳ خاصیت Text دکمهٔ اول و دوم را بترتیب به **Start Notepad** و **Stop Notepad** ست کنید. خاصیت Text فرم را هم به **Process Start Examples** تغییر دهید.

٤ روی برگهٔ Components جعبه ابزار ویژوال استودیو کلیک کنید (شکل زیر را ببینید):

تا اینجا فقط از کنترلهای برگهٔ Windows Forms استفاده کرده‌ایم، اما در برگهٔ Components هم کنترلهای جالب و مفیدی وجود دارد. در این برگه کنترلهایی را می‌بینید که بیشتر به درد کنترل محیط سیستم عامل (و نظارت بر اتفاقاتی که آنجا می‌افتد) می‌خورند.

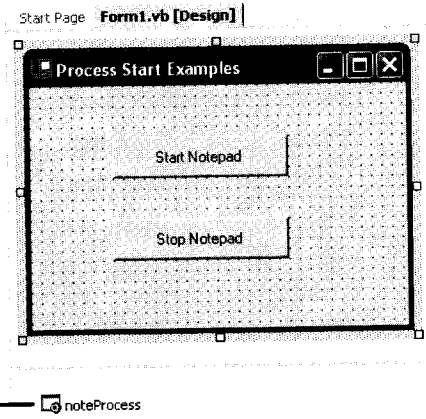
۵ روی کنترل Process دو-کلیک کنید. از آنجائیکه این کنترل نمایش ظاهری ندارد، به سینی اجزاء اضافه خواهد شد.

۶ با استفاده از پنجرهٔ خواص، نام (خاصیت Name) این کنترل را از Process1 به **noteProcess** تغییر دهید.

۷ در همان پنجرهٔ خواص شیء noteProcess ، روی علامت + کنار گروه StartInfo کلیک کرده، و سپس خاصیت FileName را به **notepad.exe** (یا هر برنامهٔ دیگری که می‌خواهید، مانند winword.exe یا excel.exe) ست کنید. از این پس می‌توانید با کمک شیء noteProcess برنامهٔ Notepad (یا هر برنامهٔ دیگری که در خاصیت FileName این شیء وارد کرده‌اید) را کنترل کنید (شکل زیر را ببینید).

نکته

اگر برنامه‌ای که می‌خواهید اجرا کنید، به آرگومان خط فرمان نیاز دارد، آنها را در خاصیت Arguments گروه StartInfo وارد کنید.



۸ روی دکمهٔ Start Notepad دو-کلیک کنید، و سپس دستور زیر را در روال رویداد Button1_Click وارد کنید:

```
noteProcess.Start()
```

توجه کنید که هیچ نامی از برنامه نبرده‌ایم، چون قبلاً آنرا از طریق خاصیت FileName مشخص کرده‌ایم.

۹ به فرم برنامه برگردید، و روی دکمهٔ Stop Notepad دو-کلیک کنید.

۱۰ دستور زیر را در روال رویداد Button2_Click بنویسید:

```
noteProcess.CloseMainWindow()
```

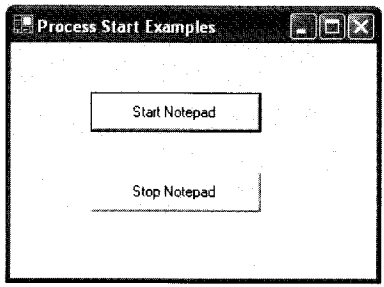
متد ()CloseMainWindow دقیقاً معادل کلیک کردن دکمهٔ Close در میله عنوان برنامهٔ Notepad است (یعنی اگر اطلاعات ذخیره‌نشده‌ای داشته باشید، برنامه به شما فرصت ذخیره کردن آنرا خواهد داد). متد دیگری نیز برای بستن برنامه‌ها وجود دارد، که Kill نام دارد؛ البته این متد بدون هیچ هشداری برنامه را می‌بندد.

۱۱ به بالای پنجرهٔ کُد فرم بروید، و دستورات زیر را در آنجا وارد کنید:

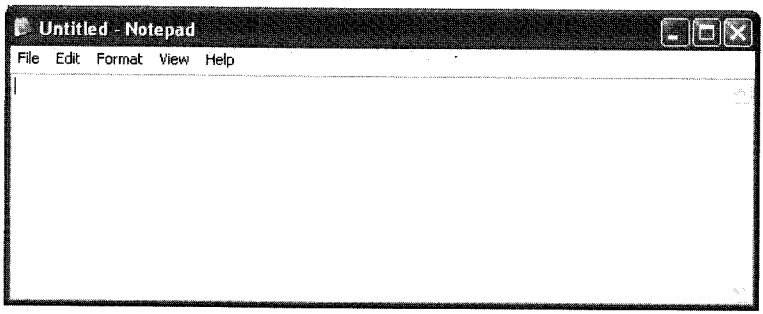
```
Imports System.Threading
Imports System.Diagnostics
```

اضافه کردن این دستورات برای برنامهٔ حاضر اختیاریست، چون برای اجرای متدهای ()Start و ()CloseMainWindow نیازی به کلاسهای System.Threading و System.Diagnostics ندارند. اما اگر بخواهید کارهای دیگری با پروسس noteProcess انجام دهید، به این کلاس‌ها نیاز خواهید داشت.

۱۳ برنامه را ذخیره، و سپس اجرا کنید، تا فرم زیر ظاهر شود (کُد کامل این برنامه را می‌توانید در پوشهٔ c:\vbnetsbs\chap13\start app پیدا کنید):



۱۳ دکمهٔ Start Notepad را کلیک کنید، تا برنامهٔ Notepad (در یک پنجرهٔ جداگانه) اجرا شود:



۱٤ به برنامهٔ My Start App برگردید، و روی دکمهٔ Stop Notepad کلیک کنید؛ این بار ویژوال بیسیک برنامهٔ Notepad را (با متد CloseMainWindow) خواهد بست.

(برای کسب اطلاعات بیشتر دربارهٔ کارهایی که با شیء Process می‌توان انجام داد، به سیستم کمک ویژوال بیسیک مراجعه کنید.)

۱۵ در پایان با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

مرجع سریع فصل ۱۳

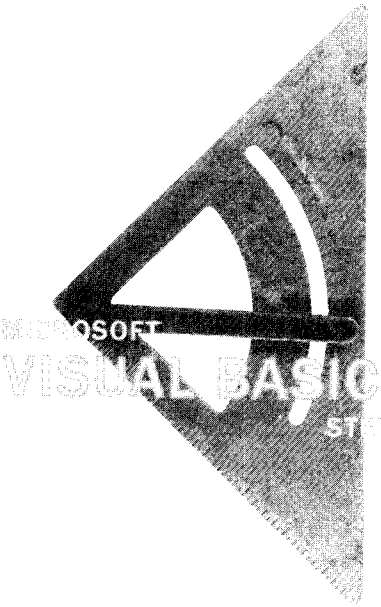
برای ...	انجام دهید
اضافه کردن رفرنس یک شیء یا برنامهٔ میزبان اتوماسیون	پنجرهٔ Project|Add Reference را باز کرده، و در برگهٔ COM کتابخانه یا شیء موردنظر را انتخاب کرده، و سپس Select را کلیک کنید.
دیدن اشیایی که در دسترس برنامه قرار دارند	با انتخاب فرمان View|Other Windows|Object Browser کاوشگر شیء را باز کرده، و اشیاء موردنظر را در سلسله مراتب درختی قاب Objects انتخاب کنید.

ایجاد یک شیء اتوماسیون	از دستورات Dim ، CType و CreateObject به شکل زیر استفاده کنید:
	`xlApp = CType(CreateObject _`
	`("Excel.Application"), Excel.Application)`

کار با شیء اتوماسیون	از خواص و متدهای شیء اتوماسیون استفاده کنید: `xlApp.Quit()`
اجرا کردن و بستن برنامه‌های ویندوز با ویژوال بیسیک	یک کنترل Process به برنامه اضافه کرده، و خاصیت FileName آنرا به نام برنامهٔ موردنظر ست کنید. سپس با متدهای Start() و CloseMainWindow() برنامهٔ مزبور را (بترتیب) اجرا کنید یا ببندید: *Objectname*.Start() *Objectname*.CloseMainWindow()

فصل

۱۴



توزیع برنامه‌های ویژوال بیسیک .NET

در این فصل یاد می گیرید چگونه :

- ✔ یک پروژهٔ توزیع به برنامهٔ خود اضافه کنید.
- ✔ به کمک «جادوگر نصب»، برای پروژهٔ خود یک برنامهٔ نصب بسازید.
- ✔ رفتار و ویژگیهای برنامهٔ نصب را بدلخواه تنظیم کنید.
- ✔ نصب و حذف برنامهٔ خود را تست کنید.

وقتی یک برنامه کامل شد، باحتمال زیاد گام بعدی توزیع آن (بین دوستان، روی اینترنت و یا به قصد فروش) است. توزیع (deployment) یک برنامه شامل نصب و راه‌اندازی برنامه روی کامپیوترهای دیگر است. در این فصل خواهید دید که چگونه می‌توان با اضافه کردن یک «پروژهٔ توزیع» (deployment project) به برنامه، و با کمک «جادوگر نصب» (Setup Wizard) فایلهای لازم برای نصب اتوماتیک برنامه را ایجاد کرد.

ایجاد پروژه‌های توزیع فرآیندیست نسبتاً پیچیده، و فراهم آوردن مقدمات آن در هر یک از ویرایشهای ویژوال بیسیک متفاوت است. برای مثال، ویرایش استاندارد ویژوال بیسیک .NET شامل جادوگر نصب نیست، و نمی‌توانید با آن برنامه‌های نصب خودکار ایجاد کنید. اما ویرایشهای حرفه‌ای و پیشرفتهٔ ویژوال بیسیک .NET ، علاوه بر داشتن الگوهای نصب متعدد، می‌توانند برای توزیع برنامه روی وب نیز فایلهای کابینت (cab files.) مخصوص بسازند. (با دنبال کردن تمرینهای این فصل، شاید به نقطه‌ای برسید که کامپایلر شما از عملیات گفته شده پشتیبانی نکند.)

تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- در ویژوال بیسیک ۶ برای توزیع یک برنامه از «جادوگر بسته‌بندی و توزیع» (Package and Deployment Wizard) استفاده می‌کردیم. اما در ویژوال بیسیک .NET برای توزیع یک برنامه باید یک پروژهٔ ویژه بنام «پروژهٔ توزیع» (deployment project) به مجموعهٔ راه‌حل (solution) اضافه کرده و سپس آنرا پیکربندی کنیم.
- برنامه‌های ویژوال بیسیک ۶ عمدتاً به تکنولوژی COM متکی بودند، و معلوم شده است که این تکنولوژی در نصب، رجیستر شدن، و حذف اشیاء مشکلات زیادی دارد. در چارچوب .NET هر برنامه یک واحد مستقل و خودکفا (از نظر کلاسهای موردنیاز) است، و بدین ترتیب بسیاری از مشکلات COM و DLL دیگر وجود ندارد.
- نصب برنامه‌های ویژوال بیسیک .NET دیگر هیچ ارتباطی با رجیستری ویندوز ندارد، و فقط با یک کپی ساده نیز انجام می‌شود (اینگونه برنامه‌ها به «نصب XCOPY» معروفند). اما تجربه نشان داده است که، نصب با استفاده از برنامه‌های نصب و Windows Installer مزایایی دارد که بهتر است آنها را از دست ندهیم.

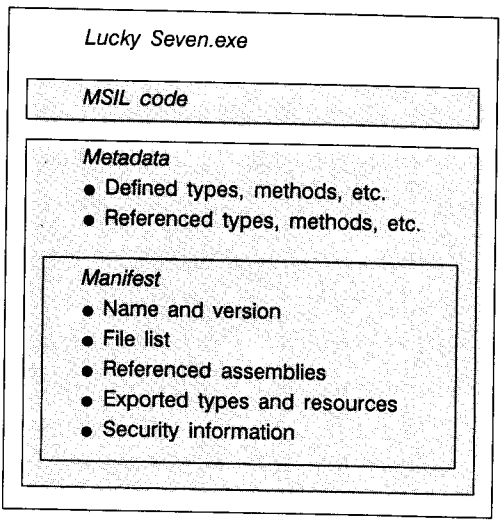
آماده کردن مقدمات توزیع برنامه

در روزهای خوش گذشته برای نصب یک برنامه فقط کافی بود فایل exe. آنرا از روی دیسکت کپی می‌کردید، و دیگر کار تمام بود! با پیچیده‌تر شدن برنامه‌ها، تعداد فایلها نیز بشدت بالا رفت، و به دهها (و گاه صدها) فایل رسید، که وجود برنامه‌های نصب (setup programs) را اجتناب‌ناپذیر می‌کرد. با اینکه سیستم عامل ویندوز بسیاری از وظایف (از قبیل مدیریت حافظه، مدیریت چاپ، و واسط کاربر) را از دوش برنامه‌نویس برداشت، اما برنامه‌های ویندوز برای نصب و رجیستر کردن مؤلفه‌های خود (بویژه فایلهای DLL) محتاج برنامه‌های نصب ویژه شدند.

مشکلاتی که این راهبرد ایجاد می‌کرد، بسیار زیاد بود. کمتر کسی را سراغ دارید که تجربهٔ تلخی از نصب برنامه‌های ویندوز نداشته باشد: برنامه‌ای که با موفقیت نصب شده، اما کار نمی‌کند؛ برنامه‌ای که به محض نصب شدن برنامه‌های دیگر را (که تا قبل از آن بدرستی کار می‌کردند) به هم می‌ریزد؛ و از همه بدتر برنامه‌هایی که دیگر حذف نمی‌شوند (و یا اگر حذف شوند، آنچنان رجیستری ویندوز را به هم می‌ریزند که درست کردن آن تقریباً محال است)! چند ماه بعد از نصب ویندوز سری به دایرکتوری Windows\ یا Windows\System\ (و یا رجیستری) بزنید، تا با خیل بیشماری از فایلهای DLL یا OCX عجیب و غریب روبرو شوید؛ فایلهایی که نه بدرد می‌خورند، و نه کسی جرأت پاک کردن آنها را دارد! این وضعیت -که بعضی از برنامه‌نویسان و کاربران عصبی ویندوز به آن «جهنم DLL» می‌گویند - عمدتاً ناشی از اشکالات بالقوهٔ مدل COM است.

یکی از اهداف ویژوال استودیو .NET در کنار گذاشتن مدل COM و فراخوانی‌های Windows API (و روی آوردن به کلاسهای .NET.)، فائق آمدن بر این مشکل بوده است. هر برنامهٔ کامپایل شدهٔ ویژوال استودیو .NET مجموعه‌ایست از تمام فایلهای لازم برای اجرای برنامه (مجموعه‌ای که به آن اسمبلی - assembly - می‌گویند).

هر اسمبلی چهار جزء دارد: کُد MSIL (زبان بینابینی میکروسافت - Microsoft Intermediate Language)، مِتادیتا (metadata) ، یک مانیفست (manifest) ، و فایلهای پشتیبانی و منابع برنامه. کُد MSIL همان برنامهٔ شماست که، بصورت کُد قابل فهم برای CLR (Common Language Runtime - کامپایلری که می‌تواند دستورات تمام زبانهای ویژوال استودیو را اجرا کند) در آورده شده است. مِتادیتا عبارتست از اطلاعاتی دربارهٔ انواع داده، متدها و رفرنس‌های تعریف شده در برنامه. مانیفست در واقع اطلاعاتیست دربارهٔ خود اسمبلی (نام و ویرایش برنامه، لیست فایلهای اسمبلی، اطلاعات امنیتی و غیره). در زیر اسمبلی برنامهٔ Locky Seven را (که در این فصل با آن کار خواهیم کرد) ملاحظه می‌کنید:



اسمبلی‌ها چنان جامع و گویا هستند که برنامه‌های ویژوال استودیو .NET برای اجرا شدن نیازی به ثبت در سیستم عامل ندارند. این بدان معناست که برای اجرای یک برنامهٔ کامپایل شدهٔ ویژوال بیسیک .NET در یک کامپیوتر جدید فقط کافیست اسمبلی آنرا روی این کامپیوتر کپی کنید (البته چارچوب .NET باید قبلاً روی این کامپیوتر نصب شده باشد). این فرآیند که به «نصب XCOPY» معروفست، یادآور نصب سادهٔ برنامه‌ها در دوران DOS است. اما بسیاری از کاربران جدید کامپیوتر آن دوران را بخاطر ندارند، و تنها چیزی که می‌شناسند برنامه‌های نصب معروف ویندوز است. بنابراین، فراهم آوردن یک برنامهٔ نصب برای برنامه‌های کامپایل شدهٔ ویژوال بیسیک .NET امری منطقی می‌نماید. از سوی دیگر، این کار باعث می‌شود که بتوان برنامه را از طریق اپلت Add/Remove Programs ویندوز از روی سیستم حذف (uninstall) کرد.

برای ایجاد برنامهٔ نصب برای برنامه‌ای که می‌نویسید، ابتدا باید یک پروژهٔ توزیع (deployment project) به راه‌حل (solution) خود اضافه کنید. این پروژهٔ توزیع (که می‌توان آنرا بنحو دلخواه پیکربندی کرد) می‌تواند برنامه را برای نصب روی محیط‌های مختلف (مانند CD و میزبان وب) بسته‌بندی و آماده کند. پروژهٔ توزیع را می‌توان در هر زمان (شروع راه‌حل، وسط و یا آخر آن) به پروژه اضافه کرد.

روشهای توزیع یک برنامه

راههای مختلف و متنوعی برای توزیع یک برنامه وجود دارد، که می‌توانید بسته به نیاز از آنها استفاده کنید. این راهها عبارتند از:

- برنامه را روی کامپیوتر خود نصب، و آنرا در رجیستری ویندوز ثبت کنید.
- یک برنامهٔ نصب از روی شبکه و یا اینترنت برای برنامهٔ خود بسازید.
- برای برنامهٔ خود یک برنامهٔ نصب از روی CD ایجاد کنید (در این روش به دستگاه CD-Writer نیاز دارید).
- یک برنامهٔ نصب با استفاده فایلهای کابینت (cab files.) برای برنامهٔ خود بسازید؛ این تکنیک برای نصب برنامه از روی میزبان وب مناسب است.

هر کامپیوتری که بخواهد برنامه‌های ویژوال بیسیک .NET را اجرا کند، باید قبلاً چارچوب .NET در آن نصب شده باشد. فایل نصب چارچوب .NET. (که Dotnetfx.exe نام دارد) روی Visual Studio .NET Windows Component Update CD (و سایت وب میکروسافت) موجود است. فایل Dotnetfx.exe نسبتاً بزرگ است (بیش از 20 MB)، و وقتی نصب شود، حجم آن به 30 MB خواهد رسید؛ ولی میکروسافت قصد دارد چارچوب .NET را در تمام سیستم عامل‌ها و برنامه‌های آتی خود بگنجاند. بدین‌ترتیب تنها کاری که برای نصب یک برنامهٔ .NET باید انجام داد، کپی کردن فایل اسمبلی آن در سیستم مقصد است. با این حال، برای اینکه یک برنامهٔ نصب کامل باشد، باید فایل نصب چارچوب .NET را هم در خود داشته باشد.

توجه

چارچوب .NET از نصب فلاپی‌دیسک پشتیبانی نمی‌کند (حداقل در زمان نوشته شدن این کتاب)، چون فایلهای ایجاد شده چنان بزرگند که نمی‌توان آنها را روی دیسکت‌های 1.44 MB جا داد.

ایجاد پروژهٔ توزیع

خوب اجازه دهید کار را شروع کنیم، و یک پروژهٔ توزیع و برنامهٔ نصب برای یکی از برنامه‌هایی که در این کتاب نوشته‌ایم (برنامهٔ Lucky Seven)، بسازیم. برنامهٔ نصبی که ایجاد می‌کنیم، این برنامه را در پوشه‌ای بنام c:\program files\microsoft press\lucky seven نصب کرده، و میانبُر آنرا در منوی Start|Programs قرار خواهد داد. علاوه بر آن، این برنامه در رجیستری ویندوز هم ثبت خواهد شد، بنابراین می‌توانید از طریق اپلت Add/Remove Programs آنرا حذف کنید. (این برنامهٔ نصب را می‌توان روی یک CD نیز کپی کرده، و آنرا بصورت «نصب CD» در آورد.)

بسیار مهم

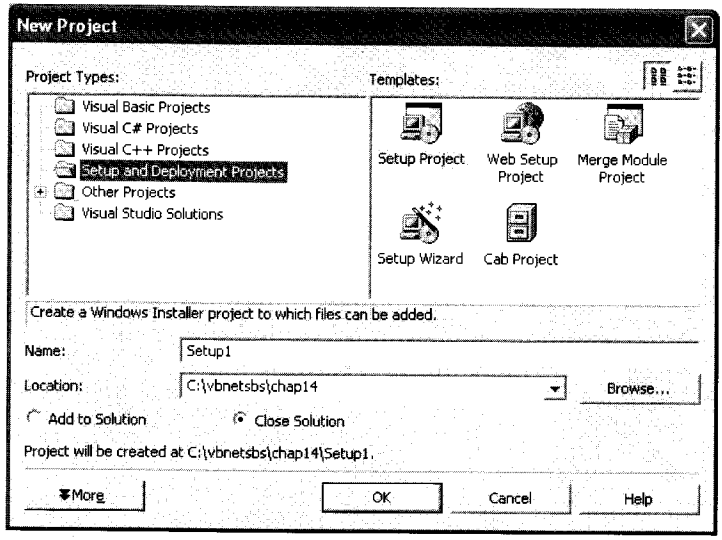
در تمرین زیر از آیتم Setup Wizard در پوشهٔ Setup and Deployment Projects دیالوگ New Project استفاده خواهیم کرد. اگر این آیتم را در پوشهٔ Setup and Deployment Projects نمی‌بینید، نمی‌توانید این تمرین را دنبال کنید. با این حال می‌توانید برنامه‌های نصب را بصورت دستی و از طریق الگوی Setup Project ایجاد کنید. (برای دیدن روش کار به قسمت بعدی این فصل، «ایجاد یک پروژهٔ توزیع با استفاده از الگوی پروژهٔ نصب»، مراجعه کنید.)

ایجاد یک پروژهٔ توزیع با استفاده از جادوگر نصب

۱ در محیط ویژوال استودیو .NET ، و از پوشهٔ c:\vbnetsbs\chap14\lucky seven ، پروژهٔ Lucky Seven را باز کنید. این برنامه در واقع همان پروژهٔ Track Wins (فصل ۱۰) است.

۲ فرمان File|New|Project را اجرا کنید، تا پنجرهٔ «پروژهٔ جدید» باز شود. در اینجا می‌خواهیم یک پروژهٔ توزیع به راه‌حل Lucky Seven اضافه کنیم.

۳ روی پوشهٔ Setup and Deployment Projects کلیک کنید. در این پوشه چهار الگو (template) و یک جادوگر (wizard) می‌بینید:



هر یک از این الگوها یک پروژهٔ توزیع با تنظیمات خاص هستند. الگوی Cab Project بگونه‌ای تنظیم شده که یک یا چند فایل کابینت (با اندازهٔ قابل تنظیم) می‌سازد؛ از این الگو برای نصب از روی اینترنت (بویژه برای کاوشگرهای قدیمی) استفاده کنید. الگوی Merge Module Project بگونه‌ای تنظیم شده که می‌توان در آن واحد چندین برنامهٔ ویژوال بیسیک را در یک برنامهٔ نصب جمع کرد؛ خروجی این الگو یک فایل msm. است، که می‌توان آنرا با فایلهای مشابه ادغام کرد. الگوی Setup Project یک برنامهٔ نصب می‌سازد، که از Windows Installer برای نصب استفاده می‌کند. خروجی الگوی Web Setup Project یک برنامهٔ نصب است، که از Windows Installer و یک میزبان وب برای نصب از روی اینترنت استفاده می‌کند.

اما شاید مفیدترین آیتم این پوشه Setup Wizard (جادوگر نصب) است، که بر اساس پاسخهایی که به سؤالات آن می‌دهید، یک پروژهٔ توزیع می‌سازد. جادوگر نصب قادر است هر یک از الگوهای Cab Project ، Merge Module Project ، Windows Installer یا Web-Windows Installer را نیز ایجاد کند.

نکته

با کلیک کردن دکمهٔ More می‌توانید نام و محل ذخیره شدن پروژهٔ توزیع را تغییر دهید. با اینکه این کار الزامی نیست، اما جدا کردن فایلهای برنامه از فایلهای توزیع ایدهٔ خوبیست.

٤ روی آیکون Setup Wizard کلیک کنید.

٥ نام پروژهٔ توزیع (فیلد Name) را **Lucky** گذاشته، و محل آن (فیلد Location) را به **c:\vbnetsbs\chap14** ست کنید.

٦ گزینهٔ Add to Solution را انتخاب کرده، و OK را کلیک کنید. این گزینه بسیار مهم است، چون در غیراینصورت ویژوال استودیو قبل از باز کردن پروژهٔ توزیع، راه‌حل Lucky Seven را می‌بندد - و این یعنی از بین رفتن تمام مزایای ادغام برنامهٔ اصلی با برنامهٔ نصب.

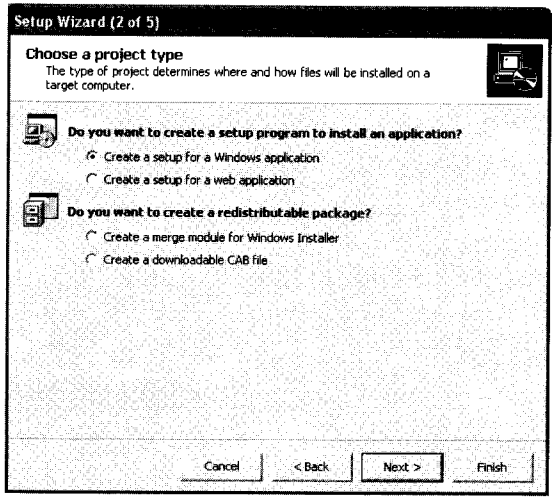
با کلیک کردن دکمهٔ OK جادوگر نصب کار خود را شروع می‌کند.

اجرای جادوگر نصب

١ جادوگر نصب پنج مرحله دارد، که مرحلهٔ اول آن (... Welcome to) را در شکل زیر می‌بینید.

این صفحه توضیح می‌دهد که وظیفهٔ جادوگر نصب، ایجاد و پیکربندی یک پروژهٔ توزیع برای برنامه است. البته جادوگر نصب قادر نیست تمام جزئیات یک پروژهٔ توزیع را پیکربندی کند، اما آنرا بگونه‌ای می‌سازد که در اغلب موارد قابل استفاده باشد.

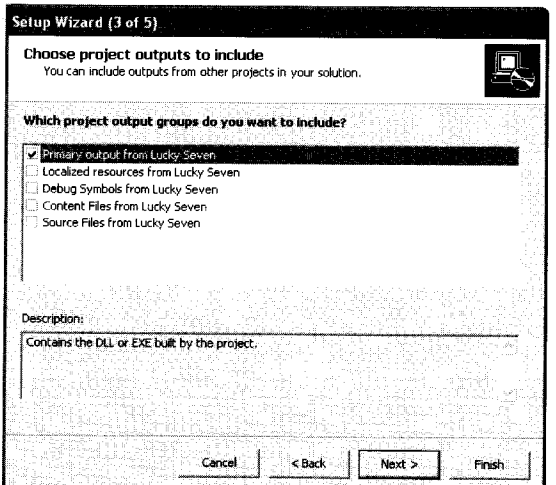
٢ دکمهٔ Next را کلیک کنید، تا مرحلهٔ دوم (دیالوگ Choose a project type) باز شود:



در پنجرهٔ «انتخاب نوع پروژه» می‌توانید یکی از چهار الگوی پوشهٔ Setup and Deployment Projects را انتخاب کنید. برای این تمرین گزینهٔ پیش‌فرض (آیتم Create a setup for a Windows application) را انتخاب می‌کنیم.

۳ دکمهٔ Next را کلیک کنید، تا مرحلهٔ سوم (دیالوگ Choose project outputs to include) باز شود. در این پنجره می‌توانید فایلهایی راکه برنامهٔ نصب Lucky Seven روی کامپیوتر مقصد کپی می‌کند، انتخاب کنید. گزینهٔ Primary output الزامیست -این گزینه باعث می‌شود تا خروجی کامپایل شدهٔ برنامه (فایل exe. یا dll.) به پروژهٔ توزیع اضافه شود. سایر گزینه‌ها بویژه برای توزیع بین‌المللی برنامه مفید هستند: Localized resources -برای برنامه‌های چندزبانه؛ Debug Symbols -برای برنامه‌هایی که نیاز به دیباگ کردن در مقصد دارند؛ Content Files و Source Files -برای برنامه‌هایی که نیاز به توسعهٔ بیشتر در مقصد دارند.

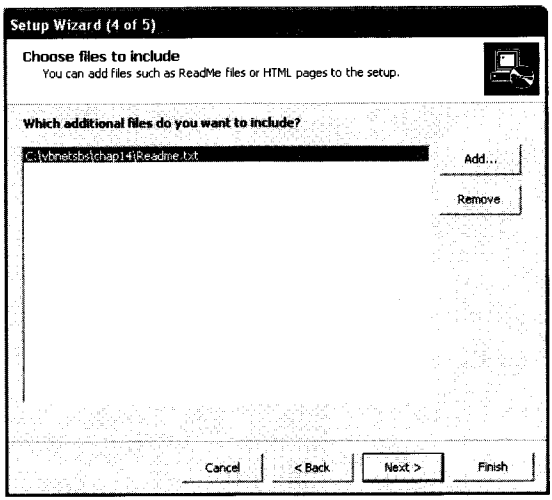
۴ گزینهٔ Primary output را انتخاب کنید:



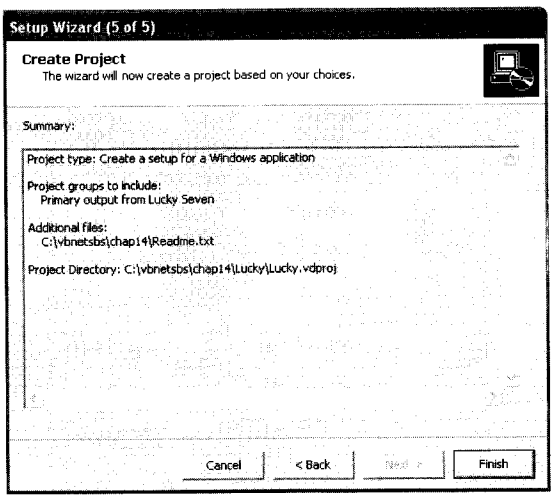
۵ دکمهٔ Next را کلیک کنید، تا مرحلهٔ چهارم (دیالوگ Choose files to include) باز شود. در این پنجره می‌توانید فایلهای اضافی را که لازم می‌دانید (فایلهای Readme.txt ، Howto.txt ، نکات عیب‌یابی، اطلاعات خرید و بازاریابی)، به برنامهٔ نصب ملحق کنید.

۶ اجازه دهید یک فایل Readme.txt به برنامهٔ نصب Lucky Seven اضافه کنیم (این فایل در پوشهٔ c:\vbnetsbs\chap14 قرار دارد). برای این منظور، روی دکمهٔ ...Add کلیک کنید.

۷ به پوشهٔ c:\vbnetsbs\chap14 بروید، و بعد از انتخاب فایل Readme.txt ، دکمهٔ Open را کلیک کنید؛ شکل زیر را ببینید:



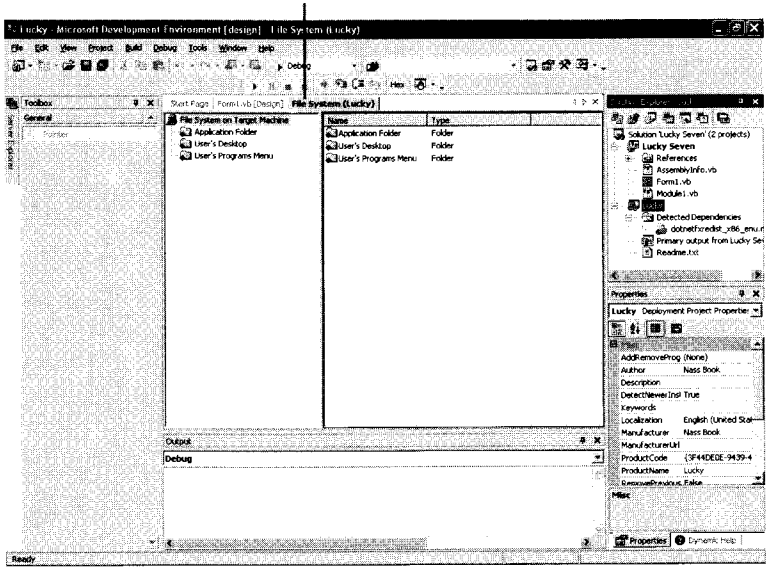
۸ دکمهٔ Next را کلیک کنید، تا به آخرین مرحلهٔ جادوگر نصب (دیالوگ Create project) برسید. در این پنجره خلاصه‌ای از گزینه‌های انتخاب شده در مراحل قبل را خواهید دید:



اگر احساس می‌کنید جایی اشتباه کرده‌اید، با کلیک کردن دکمهٔ Back به پنجرهٔ موردنظر برگشته، و بعد از تصحیح گزینه‌ها، کار را به همان ترتیب قبل ادامه دهید.

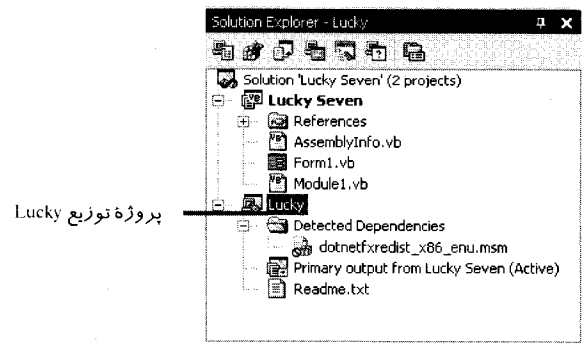
۹ دکمهٔ Finish را کلیک کنید، تا پروژهٔ توزیع برنامهٔ Lucky Seven ساخته شود. با این کار، پروژهٔ توزیع Lucky به راه‌حل اضافه شده (کاوشگر راه‌حل را ببینید)، و ادیتور سیستم فایل (File System Editor) باز می‌شود:

ادیتور سیستم فایل



از ادیتور سیستم فایل می‌توان برای اضافه کردن فایلها و آیتمهای موردنیاز به پروژه، تعیین محل آنها در هنگام نصب، و حتی ایجاد میانبُر برنامه استفاده کرد. ادیتور سیستم فایل پوشه‌هایی را که هنگام نصب برنامه در کامپیوتر میزبان ایجاد خواهند شد، نشان می‌دهد.

در ذیل پروژهٔ توزیع Lucky (در کاوشگر راه‌حل) سه آیتم می‌بینید: پوشه‌ای بنام Detected Dependencies ، فایل اجرایی برنامه (Primary output from Lucky Seven)، و یک فایل Readme.tx .



۱۰ اگر مراحل تمرین فوق را تا اینجا دنبال کرده‌اید، لازم نیست قسمت آینده را انجام دهید، و می‌توانید مستقیماً به قسمت «سفارشی کردن پروژهٔ توزیع» بروید.

نکته

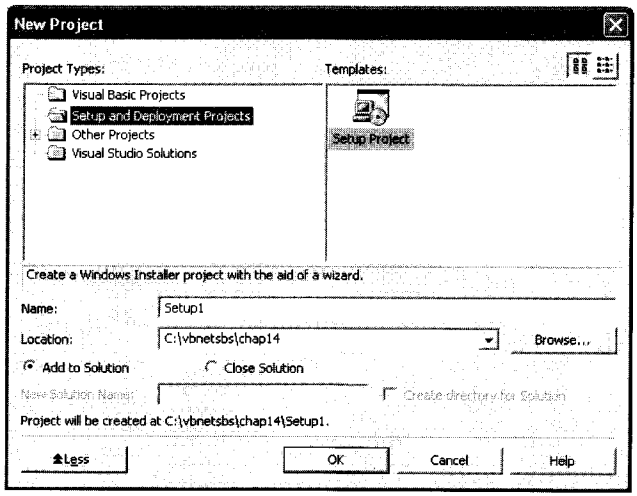
با اینکه (در ذیل پوشهٔ Detected Dependencies) چارچوب .NET. بعنوان یکی از ملزومات پروژه ذکر شده، نمی‌توان آنرا به فایلهای برنامهٔ نصب منضم کرد. اگر سعی کنید با False کردن خاصیت Exclude فایلهای چارچوب .NET. را به برنامهٔ نصب اضافه کنید، هنگام کامپایل کردن برنامه با خطا روبرو خواهید شد (البته در ویرایش بتای ویژوال استودیو .NET. این کار امکانپذیر بود). دلیل این موضوع آنست که، منضم کردن فایلهای چارچوب .NET. امکان ارتقاء (و برطرف کردن باگهای) آنرا از بین می‌برد (چون ممکنست گزینه‌های نصب ویژوال استودیو و چارچوب .NET. در آینده تغییر کند).

ایجاد یک پروژهٔ توزیع با استفاده از الگوی Setup Project

۱ در محیط ویژوال استودیو .NET ، و از پوشهٔ c:\vbnetsbs\chap14\lucky seven ، پروژهٔ Lucky Seven را باز کنید.

۲ فرمان File|New|Project را اجرا کنید، تا پنجرهٔ «پروژهٔ جدید» باز شود. در اینجا می‌خواهیم یک پروژهٔ توزیع به راه‌حل Lucky Seven اضافه کنیم.

۳ روی پوشهٔ Setup and Deployment Projects کلیک کنید. در این پوشه یک الگو بنام Setup Project می‌بینید:



٤ روی آیکون Setup Project کلیک کنید.

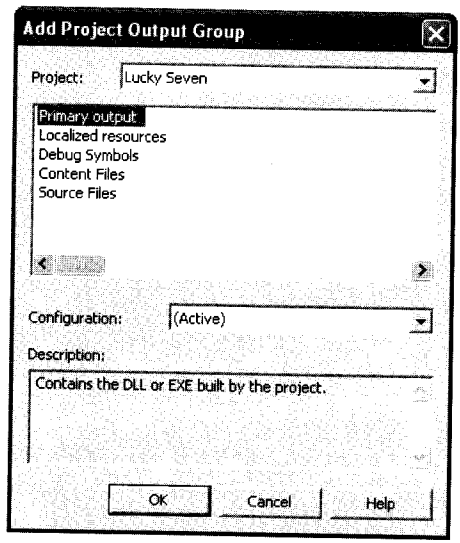
۵ نام پروژه (فیلد Name) را **Lucky** گذاشته، و محل آن (فیلد Location) را به **c:\vbnetsbs\chap14** ست کنید.

۶ گزینهٔ Add to Solution را انتخاب کرده، و OK را کلیک کنید. با این کار، پروژهٔ توزیع Lucky به راه‌حل اضافه شده (کاوشگر راه‌حل را ببینید)، و ادیتور سیستم فایل (File System Editor) باز می‌شود. از ادیتور سیستم فایل می‌توان برای اضافه کردن فایلها و آیتمهای موردنیاز به پروژه، تعیین محل آنها در هنگام نصب، و حتی ایجاد میانبُر برنامه استفاده کرد. ادیتور سیستم فایل پوشه‌هایی را که هنگام نصب برنامه در کامپیوتر میزبان ایجاد خواهند شد، نشان می‌دهد.

در اینجا باید فایل اجرایی برنامه (فایل Lucky Seven.exe ، که Primary Output نامیده می‌شود) را به پروژهٔ توزیع اضافه کنیم.

۷ پروژهٔ توزیع Lucky را در کاوشگر راه‌حل انتخاب کنید.

۸ از منوی Project|Add فرمان Project Output را انتخاب کنید، تا پنجرهٔ Add Project Output Group باز شود:



در این پنجره فایلهایی را که می‌خواهید به پروژه اضافه کنید، انتخاب می‌کنید. گزینهٔ Primary output معمولاً اجباریست - این گزینه اجازه می‌دهد تا فایل اجرایی برنامه (فایل exe. یا dll.) را به پروژهٔ توزیع اضافه کنید.

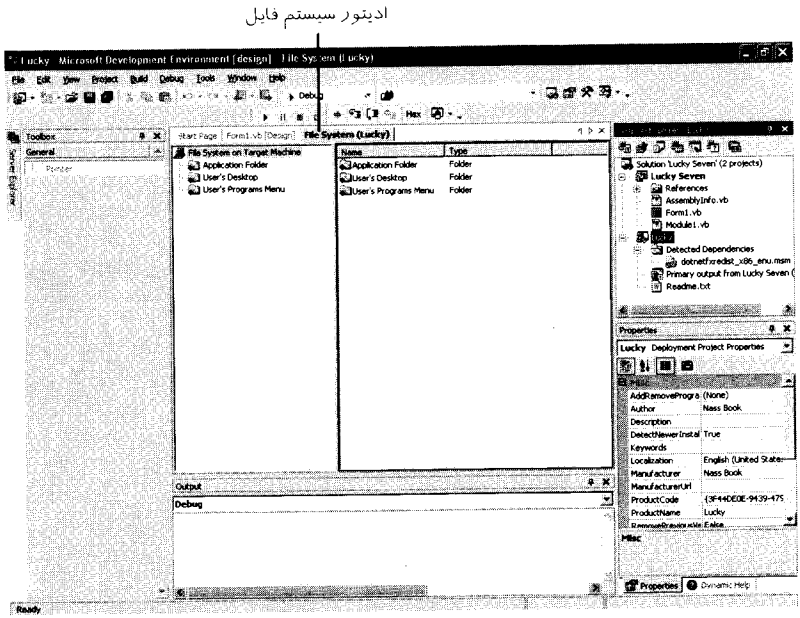
۹ گزینهٔ Primary output را انتخاب کرده، و OK را کلیک کنید. (دقت کنید که با این کار، علاوه بر فایل اجرایی برنامه، فایل چارچوب NET. نیز به پوشهٔ Detected Dependencies اضافه می‌شود.)

۱۰ پروژهٔ توزیع Lucky را در کاوشگر راه‌حل انتخاب کرده، و سپس از منوی Project|Add فرمان File را انتخاب کنید، تا پنجرهٔ Add Files باز شود. این دیالوگ به شما اجازه می‌دهد تا فایلهای موردنظر (فایلهای Readme.txt ، Howto.txt ، نکات عیب‌یابی، اطلاعات خرید و بازاریابی) را به پروژهٔ

توزیع اضافه کنید.

اجازه دهید یک فایل Readme.txt به برنامهٔ نصب Lucky Seven اضافه کنیم (این فایل در پوشهٔ c:\vbnetsbs\chap14 قرار دارد).

۱۱ به پوشهٔ c:\vbnetsbs\chap14 بروید، و بعد از انتخاب فایل Readme.txt ، دکمهٔ Open را کلیک کنید؛ شکل زیر را ببینید:



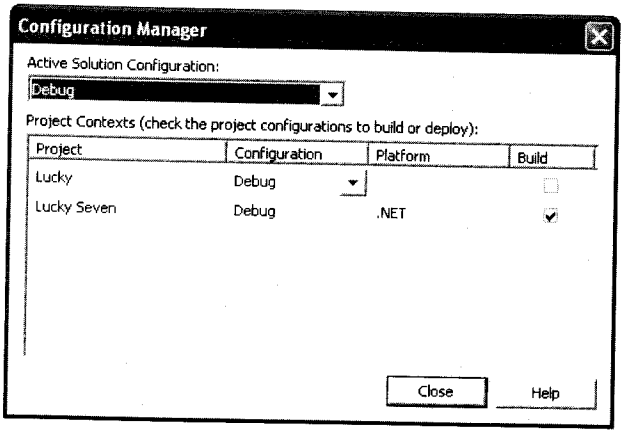
پس از آن نوبت به تنظیم پروژهٔ توزیع و سفارشی کردن آن می‌رسد.

سفارشی کردن پروژهٔ توزیع

پروژهٔ توزیع ما اکنون آمادهٔ کار است – دفعهٔ بعد که راه‌حل Lucky Seven را کامپایل کنید، یک برنامهٔ نصب (که فایلیست با پسوند msi.) در پوشهٔ c:\vbnetsbs\chap14\lucky ساخته خواهد شد. اما هنوز چند چیز کوچک وجود دارد که با تنظیم آنها می‌توان برنامهٔ نصب بهتری ایجاد کرد. در این قسمت خواهید دید که چگونه می‌توان با استفاده از مدیر پیکربندی (Configuration Manager) تنظیمات پروژهٔ توزیع (از قبیل نحوهٔ ایجاد میانبُر برنامه در کامپیوتر مقصد، نمایش اطلاعات شرکت سازندهٔ نرم‌افزار و ویرایش برنامه در برنامهٔ نصب) را تغییر داد.

مدیر پیکربندی

۱ از منوی Build فرمان Configuration Manager را انتخاب کنید، تا پنجرهٔ مدیر پیکربندی باز شود:



در این پنجره تنظیمات فعلی ساخت پروژه را مشاهده می‌کنید. همانطور که می‌بینید، در حال حاضر هر دو پروژهٔ Lucky Seven و Lucky بصورت Debug (دیباگ) ساخته می‌شوند؛ این بدان معناست که کامپایلر هنگام کامپایل کردن راه‌حل، فایلها و اطلاعات لازم برای تست و دیباگ آن را نیز تولید خواهد کرد. معمولاً در پایان فرآیند تولید برنامه (هنگامی که دیگر می‌خواهید آنرا منتشر کنید)، بایستی شکل ساخت تمام پروژه‌ها را به Release (انتشار) تغییر دهید.

لیست Active Solution Configuration را باز کرده، و آیتم Release را انتخاب کنید.

گزینهٔ Configuration پروژهٔ Lucky Seven را به Release تغییر دهید.

گزینهٔ Configuration پروژهٔ Lucky را نیز به Release تغییر دهید.

اگر در هر زمان خواستید فرآیند توسعهٔ پروژه را از سر بگیرید، می‌توانید براحتی نوع ساخت آنرا به Debug تغییر دهید.

دقت کنید که گزینهٔ Build برای دو پروژهٔ علامت خورده باشد.

نکته

هر پروژه‌ای که علامت Build نداشته باشد، در هنگام اجرا یا کامپایل شدن برنامه ساخته نخواهد شد. سعی کنید گزینهٔ Build پروژهٔ توزیع (جز در مراحل پایانی کار) علامت نداشته باشد، چون علاوه بر آنکه این کار ضرورتی ندارد، معمولاً بسیار وقت‌گیر هم هست.

با کلیک کردن دکمهٔ Close ، مدیر پیکربندی را ببندید.

در قدم بعد، به کمک ادیتور سیستم فایل، یک میانبُر برای برنامهٔ Lucky Seven می‌سازیم تا کاربر بتواند آنرا براحتی اجرا کند.

ایجاد میانبر برنامه

۱ در قاب سمت چپ ادیتور سیستم فایل، پوشهٔ Application Folder را انتخاب کنید.

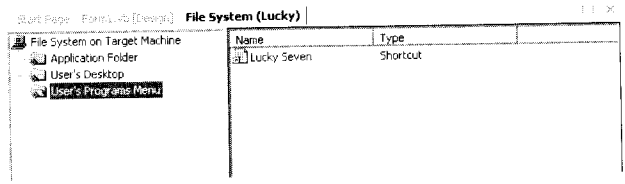
نکته

اگر ادیتور سیستم فایل را نمی‌بینید، ابتدا پروژهٔ توزیع Lucky را در کاوشگر راه‌حل انتخاب کرده، و سپس فرمان File System را از منوی View|Editor اجرا کنید.

۲ در قاب سمت راست ادیتور سیستم فایل، روی Primary output from Lucky Seven راست-کلیک کرده، و آیتم Create shortcut to Primary output from Lucky Seven را انتخاب کنید. با این کار، یک آیکون میانبُر ایجاد می‌شود، که می‌توانید نام آنرا عوض کنید.

۳ نام این میانبُر را به **Lucky Seven** تغییر داده، و Enter را بزنید.

۴ میانبُر Lucky Seven را (که در قاب سمت راست ادیتور سیستم فایل قرار دارد) با ماوس گرفته و روی پوشهٔ User's Programs Menu (در قاب سمت چپ) بکشید. در شکل زیر نتیجهٔ کار را ملاحظه می‌کنید:



وقتی این برنامهٔ نصب اجرا شود، یک میانبُر در منوی Start|Programs کامپیوتر مقصد ایجاد می‌کند، که کاربر بسادگی می‌تواند با آن برنامه را اجرا کند.

در قدم بعد، اطلاعات شرکت سازندهٔ نرم‌افزار و شمارهٔ ویرایش برنامه را به برنامهٔ نصب اضافه می‌کنیم.

تعیین شرکت سازنده، شماره ویرایش و ... برنامه

۱ پروژهٔ توزیع Lucky را در کاوشگر راه‌حل انتخاب کنید.

۲ پنجرهٔ خواص را باز کرده و آنقدر آنرا بزرگ کنید، تا چند خاصیت (و مقدار آنها) با هم دیده شود (اینها خواص خود پروژه هستند).

خاصیت‌های Author و Manufacturer نام برنامه‌نویس و شرکت نویسندهٔ نرم‌افزار را نشان می‌دهند (این اطلاعات در قسمت Contact و Supprt Info هر برنامه ثبت می‌شوند).

خاصیت‌های Title و Version عنوان برنامهٔ نصب و ویرایش برنامه را (که در هنگام اجرای برنامهٔ نصب نشان داده می‌شود) مشخص می‌کنند. خواص Product Code و Package Code نیز کُدهای الفبایی-عددی منحصربفردی هستند، که توسط ویژوال استودیو تولید شده، و هویت برنامه را مشخص می‌کنند.

۳ خاصیت‌های Author و Manufacturer را به **Microsoft Press** تغییر دهید.

٤ خاصیت Version را به **1.5.0** تغییر دهید. وقتی بعد از تغییر دادن این خاصیت، Enter را بزنید، ویژوال استودیو سؤال می‌کند که آیا میل دارید Product Code و Package Code های جدیدی تولید کند.

۵ Yes را کلیک کنید، تا کُدهای جدید تولید شود.

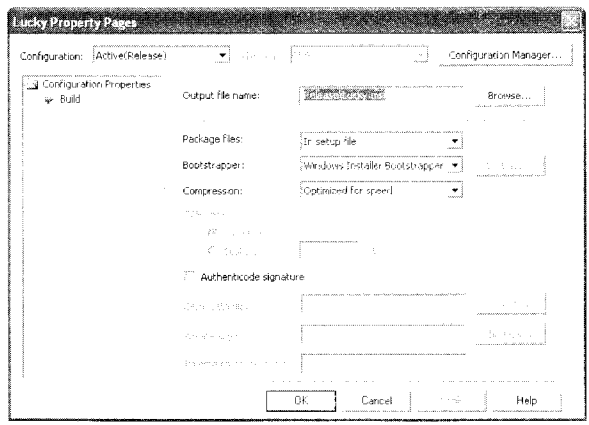
۶ خواص دیگر را کمی بیشتر بررسی کرده، و سپس پنجرهٔ خواص را به حالت قبل برگردانید.

برای بررسی تنظیمات دیگر پروژهٔ توزیع، باید از صفحات خواص پروژهٔ Lucky استفاده کنیم.

صفحات خواص پروژهٔ توزیع

۱ پروژهٔ توزیع Lucky را در کاوشگر راه‌حل انتخاب کنید.

۲ فرمان Properties را از منوی Project انتخاب کنید، تا صفحات خواص (Property Pages) این پروژه باز شود:



در این پنجره می‌توانید برخی از خواص پروژهٔ توزیع (که با جادوگر نصب ست کرده بودید، و خواصی که جادوگر نصب در اختیار شما نگذاشته بود) را تغییر دهید.

خاصیت Output file name نام فایل نصب برنامه را (که پسوند msi. دارد) مشخص می‌کند. این یک فایل بزرگ است، که همراه با چند فایل کوچکتر (از قبیل Setup.ini و Setup.exe) برنامهٔ نصب را تشکیل می‌دهند.

۳ در لیست Package files در حال حاضر گزینهٔ In setup file را می‌بینید (این گزینه که در جادوگر نصب انتخاب کرده بودیم، یک فایل بزرگ msi. می‌سازد). در لیست Package files دو گزینهٔ دیگر نیز وجود دارد: As loose uncompressed files (برای ایجاد فایلهای نصب غیرفشرده)، و In cabinet file(s) (برای ایجاد چند فایل cab. کوچکتر).

٤ گزینهٔ In cabinet file(s) را انتخاب کنید؛ با اینکار فیلد Cab size فعال می‌شود. اگر Custom را انتخاب کنید، می‌توانید اندازهٔ فایلهای cab. را مشخص کنید.

٥ بار دیگر از لیست Package files گزینهٔ In setup file را انتخاب کنید.

٦ با کمک لیست Bootstrapper می‌توانید فایلهای اضافی را که باید به برنامهٔ نصب منضم شوند، انتخاب کنید. این گزینه کمک می‌کند تا Windows Installer 2.0 روی کامپیوتر مقصد نصب شود (این برنامه همراه ویژوال استودیو .NET و Windows XP می‌آید، و نیازی به نصب آن نیست).

٧ از این لیست گزینهٔ Windows Installer Bootstrapper را انتخاب کنید.

٨ در لیست Compression دو گزینه وجود دارد: Optimized for size (انتخاب متداول) برای فشرده کردن هر چه بیشتر فایلهای نصب؛ و Optimized for speed (برای مواردی که نگرانی جا ندارند، مثلاً در CD-ROM) بمنظور بهینه کردن فایلهای نصب برای رسیدن به حداکثر سرعت. از این لیست گزینهٔ Optimized for size را انتخاب کنید (چون فایلهای نصب روی کامپیوتر شما باقی خواهند ماند).

٩ گزینهٔ آخر در پنجرهٔ صفحات خواص، Authenticode signature است؛ این گزینه کمک می‌کند تا امضای دیجیتالی خود را پای برنامه بگذارید. بحث دربارهٔ این گزینه از حوصلهٔ کتاب حاضر خارج است (و به همین دلیل آنرا فعال نخواهیم کرد)، ولی اگر می‌خواهید برنامه‌های خود را بصورت تجاری منتشر کنید، باید Authenticode signature را در برنامه‌های نصب خود داشته باشید، تا از طرف سیستم عامل بعنوان یک منبع معتبر شناخته شوید.

١٠ با کلیک کردن OK تغییرات داده شده در صفحات خواص را ذخیره کرده، و آنرا ببندید. اکنون آمادهٔ کامپایل کردن پروژه هستیم.

ساخت پروژهٔ توزیع و تست برنامهٔ نصب

بعد از تنظیم گزینه‌های پروژهٔ توزیع، آماده‌ایم تا راه‌حل را کامپایل کنیم. مراحل کار عبارتند از:

١ **ساخت راه‌حل با فرمان Build Solution** فرمان Build Solution -که در منوی Build قرار دارد -کل راه‌حل (برنامهٔ اصلی و پروژهٔ توزیع) را کامپایل می‌کند.

٢ **اجرای برنامهٔ نصب** برنامهٔ نصب را اجرا کرده، و برنامهٔ نصب شده را تست کنید.

٣ **بررسی فایلهای نصب شده** پس از نصب برنامه، بررسی کنید که آیا تمام فایلهای لازم در محلهای صحیح نصب شده‌اند.

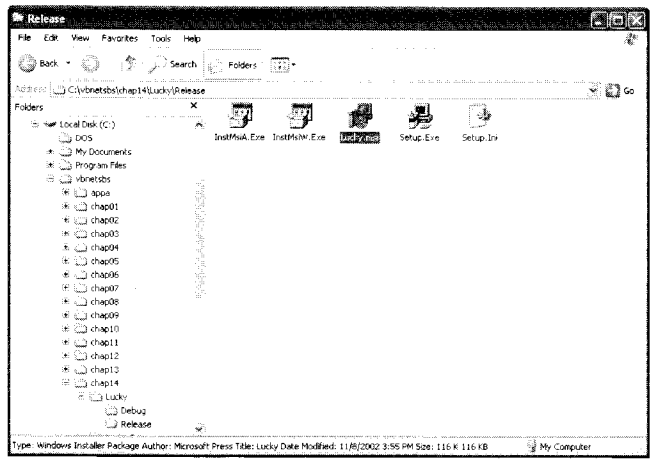
ساخت پروژه

١ فرمان Build|Build Solution را اجرا کنید، تا ساخت راه‌حل Lucky Seven شروع شود. در حین کامپایل شدن راه‌حل، در میلهٔ وضعیت ویژوال استودیو درصد تکمیل کار را مشاهده خواهید دید:

در پایان، پیام "Build Succeeded" موفقیت عملیات را اعلام می‌کند.

۲ از منوی Start|Programs|Accessories ، برنامهٔ Windows Explorer (کاوشگر ویندوز) را اجرا کنید؛ می‌خواهیم ببینیم که آیا فایلهای خروجی کامپایل برنامه بدرستی ایجاد شده‌اند یا خیر.

۳ به پوشهٔ c:\vbnetsbs\chap14\lucky\release بروید، و فایل Lucky.msi را انتخاب کنید:



با انتخابهایی که در مدیر پیکربندی و صفحات خواص کردیم، برنامهٔ نصب Lucky بصورت Release ساخته می‌شود، و فایلهای Windows Installer نیز به آن منضم خواهند شد (این فایلها عبارتند از: Instmsia.exe و Instmsiw.exe). دقت کنید که تمام اطلاعات فایل Lucky.msi (نوع فایل، نام مؤلف برنامه، و اندازهٔ فایل) در میلهٔ وضعیت کاوشگر ویندوز نشان داده می‌شود.

نکته

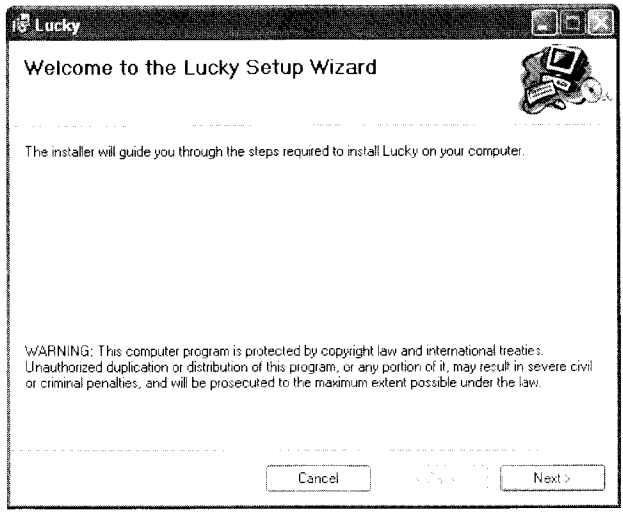
برای ایجاد یک «CD نصب» کافیست تمام فایلهای پوشهٔ Release را روی یک CD-R کپی کنید (البته برای این کار به یک CD-Writer نیاز دارید). اگر احتمال می‌دهید که چارچوب NET. روی کامپیوتر مقصد وجود نداشته باشد، فایل نصب چارچوب NET. (که Dotnetfx.exe نام دارد) را نیز روی این CD کپی کنید. (چارچوب NET. باید بصورت جداگانه نصب شود.)

اجرای برنامهٔ نصب

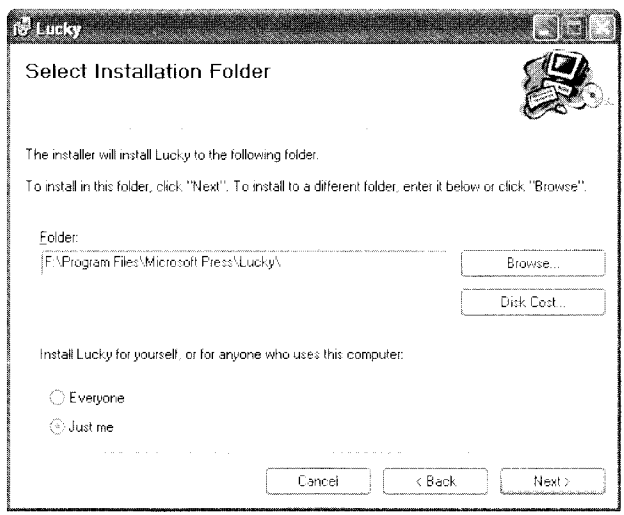
۱ روی فایل Setup.exe در پوشهٔ c:\vbnetsbs\chap14\lucky\release دو-کلیک کنید، تا برنامهٔ نصب Lucky Seven اجرا شود.

اگر کاربر روی کامپیوترش Windows Installer را داشته باشد، برنامهٔ نصب شروع می‌شود (در غیر اینصورت، این فرصت به وی داده می‌شود تا Windows Installer را نصب کند، که این کار ممکنست نیاز به راه‌اندازی مجدد کامپیوتر داشته باشد). بعد از چند لحظه پنجرهٔ Welcome to the

Lucky Setup Wizard باز می‌شود:



۲ Next را کلیک کنید؛ با این کار پنجرهٔ تعیین محل نصب برنامهٔ Lucky Seven باز می‌شود:



توجه کنید که برنامه بصورت پیش‌فرض در پوشهٔ c:\program files\microsoft press\lucky نصب می‌شود (پوشهٔ Microsoft Press در واقع همان خاصیت Author و Manufacturer است). گزینهٔ Everyone و Just me نیز با تنظیمات امنیتی سیستم عامل سروکار دارند.

۳ گزینهٔ Everyone را انتخاب کرده، و Next را کلیک کنید. در پاسخ به پرسش تأئید انتخابهای انجام شده، یک بار دیگر Next را کلیک کنید (و یا اگر نظرتان عوض شده، با Back برگردید، و تغییرات لازم را انجام دهید).

٤ Next را کلیک کنید، تا نصب برنامه (ایجاد پوشه‌های لازم، کپی کردن فایلها، و ثبت برنامه در رجیستری ویندوز) شروع شود.

۵ در پایان نصب، دکمهٔ Close را کلیک کنید.

تمام! این هم از یک برنامهٔ نصب کاملاً حرفه‌ای!

نکته

کامپیوتری که این برنامه را روی آن نصب می‌کنید، باید دارای یک حداقل ملزومات باشد. برنامه‌هایی که با ویژوال استودیو .NET ایجاد می‌شوند، برای اجرا به ویندوز ۹۸ با اینترنت اکسپلورر ۵/۰۱ (یا بالاتر)، ویندوز NT 4.0 با Service Pack 6a (یا بالاتر) و اینترنت اکسپلورر ۵/۰۱ (یا بالاتر) نیاز دارند. ویژوال استودیو .NET از ویندوز ۹۵ پشتیبانی نمی‌کند؛ و وجود چارچوب .NET نیز که الزامیست.

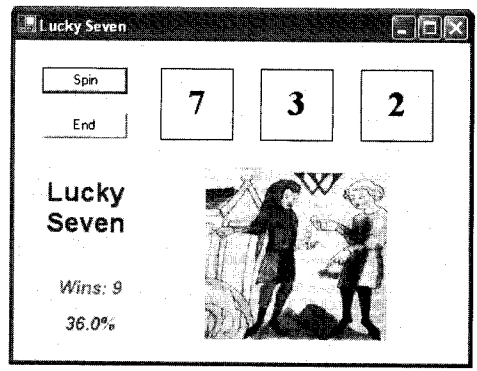
اجرای برنامهٔ Lucky Seven

۱ منوی Programs|Start را باز کرده، و روی آیکون Lucky Seven کلیک کنید (این همان محلیست که توسط ادیتور سیستم فایل آیکون میانبُر را در آنجا قرار دادیم). بله، برنامه اجرا می‌شود!

نکته

برای اجرای این برنامه می‌توانید به پوشهٔ c:\program files\microsoft press\lucky نیز بروید، و روی فایل Lucky Seven.exe دو-کلیک کنید.

۲ کمی با برنامه کار کنید (در شکل زیر برنامهٔ Lucky Seven را پس از ۲۵ بار کلیک کردن دکمهٔ Spin می‌بینید:)



۳ در پایان، با کلیک کردن دکمهٔ End برنامه را ببندید.

برنامهٔ ما هم بدرستی نصب شده، و هم بخوبی کار می‌کند!

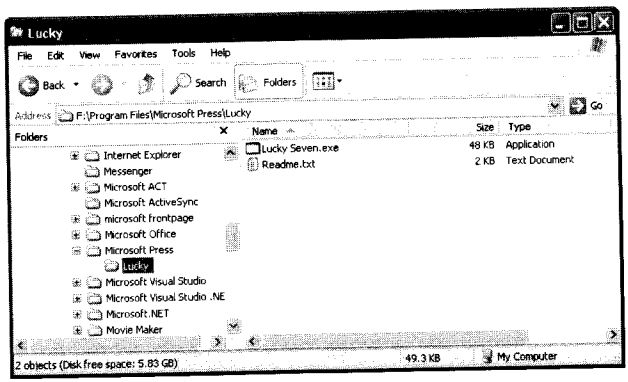
یک گام فراتر: بررسی فایلهای نصب و حذف برنامه

کار دیگری که می‌خواهیم انجام دهیم، بررسی فایلهای کپی شده در پوشهٔ c:\program files\microsoft press\lucky ، و سپس حذف (uninstall) برنامهٔ Lucky Seven است. تست حذف برنامه یک قدم مهم است، چون باید ببینیم که آیا برنامهٔ ما این عملیات را نیز بدرستی پشت سر می‌گذارد. (یک برنامهٔ خوب برنامه‌ایست که براحتی بتوان آنرا حذف کرد!)

بررسی فایلهای نصب شده

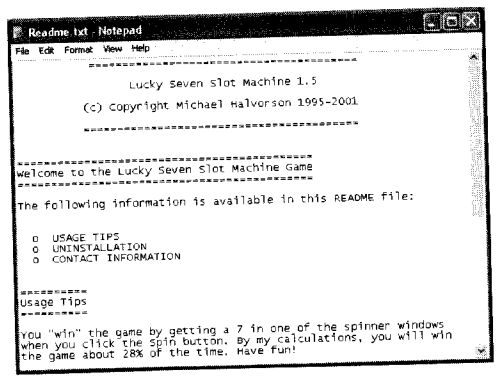
۱ کاوشگر ویندوز را باز کرده، و به پوشهٔ c:\program files\microsoft press\lucky بروید. در این پوشه دو فایل بنامهای Lucky Seven.exe و Readme.txt می‌بینید، که حاصل پیکربندی پروژهٔ توزیع ماست.

۲ از منوی View کاوشگر ویندوز آیتم Details را انتخاب کنید. با این کار کاوشگر ویندوز اطلاعات بیشتری از هر فایل (از قبیل اندازه، و نوع آن) را نشان خواهد داد؛ به شکل زیر نگاه کنید:



همانطور که می‌بینید اندازهٔ فایل Lucky Seven.exe فقط 48 KB است، در حالیکه فایلهای نصب چارچوب NET. چیزی نزدیک به 30 MB فضا (در پوشهٔ Windows\Microsoft.NET) اشغال می‌کنند.

۳ روی فایل Readme.txt دو-کلیک کنید، تا این فایل در برنامهٔ NotePad باز شود:



اگر برنامهٔ شما احتیاج به تنظیمات خاصی دارد (و یا می‌خواهید کاربران امکان تماس با شما را داشته باشند)، حتماً این اطلاعات را بصورت یک فایل Readme.txt در برنامهٔ نصب قرار دهید.

٤ محتویات فایل Readme.txt را بررسی کرده، و سپس آنرا ببندید.

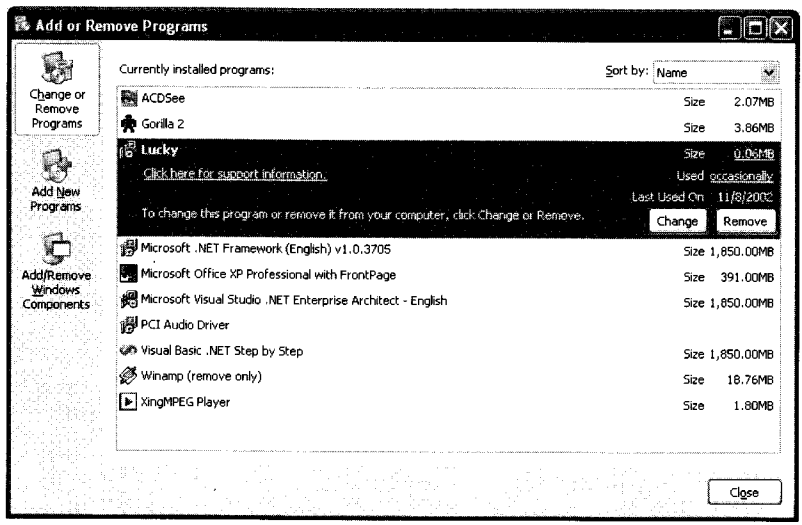
و بالاخره، در تمرین زیر نحوهٔ حذف برنامهٔ Lucky Seven را مشاهده خواهید کرد.

حذف برنامهٔ Lucky Seven

۱ با اجرای فرمان Start|Settings|Control Panel ، پانل کنترل (مرکز فرماندهی و تنظیم ویندوز) را باز کنید.

۲ روی آیکون Add/Remove Programs دو-کلیک کنید. این برنامه به شما اجازه می‌دهد تا برنامه‌ها را نصب و یا (آنهایی که با Windows Installer نصب شده‌اند، را) حذف کنید.

۳ در لیست برنامه‌های نصب شده، برنامهٔ Lucky را پیدا و انتخاب کنید:



٤ در این قسمت لینکی می‌بینید بنام support information ؛ روی آن کلیک کنید، تا پنجرهٔ اطلاعات پشتیبانی برنامهٔ Lucky باز شود:

۵ روی دکمهٔ Close کلیک کنید، تا پنجرهٔ اطلاعات پشتیبانی بسته شود.

۶ برای حذف برنامه، دکمهٔ Remove را کلیک کنید.

۷ برای تأئید حذف برنامه، دکمهٔ Yes را کلیک کنید. پس از آن Windows Installer تمام اجزای برنامهٔ Lucky (فایلها، پوشه‌ها، میانبُر، و کلیدهای ثبت شده در رجیستری) را حذف می‌کند.

۸ با کلیک کردن دکمهٔ Close ، پنجرهٔ Add/Remove Programs را هم ببندید.

مرجع سریع فصل ۱۴

برای ...	انجام دهید
ایجاد یک برنامهٔ نصب	پس از باز کردن پنجرهٔ «پروژهٔ جدید»، با انتخاب آیتم Setup Wizard از پوشهٔ Setup and Deployment Project یک پروژهٔ توزیع به راه‌حل خود اضافه کنید.
ایجاد یک برنامهٔ نصب خودکار	از پوشهٔ Setup and Deployment Project آیتم Setup Wizard را انتخاب و مراحل آنرا دنبال کنید.
تنظیم نحوهٔ کامپایل شدن برنامه	از مدیر پیکربندی (Build|Configuration Manager) استفاده کنید.
تنظیم گزینه‌های پروژهٔ توزیع	خواص پروژهٔ توزیع را از طریق پنجرهٔ خواص آن ست کنید.
ایجاد میانبُر برای برنامه	در ادیتور سیستم فایل، روی Primary Output راست-کلیک کرده و فرمان Create Shortcut to Primary Output را انتخاب کنید. سپس نام این میانبُر را عوض کرده، و آنرا به محل دلخواه منتقل کنید.
کامپایل کردن یک برنامه	فرمان Build|Build Solution را اجرا کنید.
اجرای برنامهٔ نصب	روی فایل Setup.exe یا msi. دو-کلیک کنید.
حذف یک برنامه	از برنامهٔ Add/Remove Programs (در پانل کنترل ویندوز استفاده کنید.

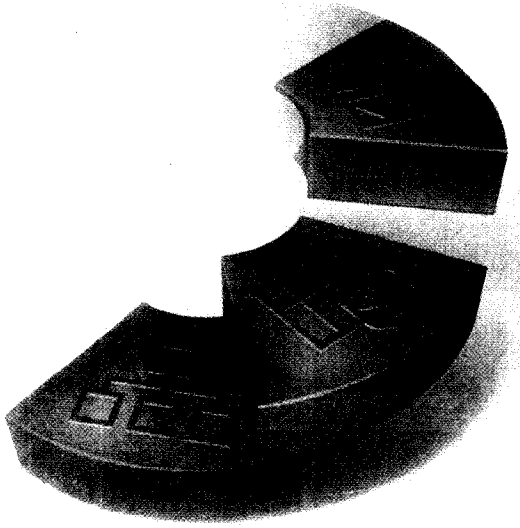
VISUAL BASIC .NET

آموزش گام به گام

۴

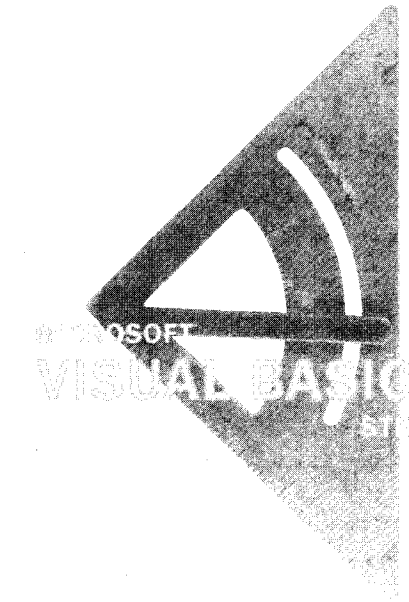
بخش

طراحی واسط کاربر پیشرفته



فصل

۱۵



مدیریت فرمهای ویندوز

در این فصل یاد می گیرید چگونه :

- ✔ یک فرم جدید به برنامهٔ خود اضافه کنید.
- ✔ محل فرم روی میزکار ویندوز را تغییر دهید.
- ✔ کنترل‌ها را در زمان اجرای برنامه به فرم اضافه کنید.
- ✔ اشیاء روی فرم را نسبت به یکدیگر تراز کنید.
- ✔ شیء شروع برنامه را مشخص کنید.

در بخش‌های قبل یاد گرفتید چگونه یک برنامهٔ نسبتاً کامل ویژوال بیسیک .NET بنویسید. در این بخش روی واسط کاربر تمرکز خواهیم کرد، و یاد می‌گیرید چگونه پروژه‌های چند فرمه بنویسید، جلوه‌های انیمیشن ایجاد کنید، از وراثت بصری بهره بگیرید، و مطالب خود را چاپ کنید.

در این فصل با برنامه‌های چند فرمه سروکار داریم. خواهید دید که چگونه با خاصیت DesktopBounds می‌توان اندازه و مکان فرم روی میز کار ویندوز را تغییر داد، چگونه کنترلها را در زمان اجرای برنامه به آن اضافه و آنها را تراز کرد، و چگونه نحوهٔ شروع برنامه را کنترل کرد.

تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- در ویژوال بیسیک .NET بدون داشتن یک متغیر وهله (Instance variable) برای فرم دوم نمی‌توان خواص آنرا ست کرد.
- در ویژوال بیسیک ۶ برای تعیین مکان فرم روی میزکار ویندوز از پنجرهٔ Form Layout استفاده می‌کردیم. در ویژوال بیسیک .NET پنجرهٔ Form Layout دیگر وجود ندارد، و برای این منظور باید از خاصیتی بنام DesktopBounds کمک بگیریم.
- در ویژوال بیسیک .NET نیز مانند در ویژوال بیسیک ۶ امکان اضافه کردن کنترل به فرم در زمان اجرای برنامه وجود دارد، فقط مکانیزم این کار فرق کرده است.
- کنترلهای ویژوال بیسیک .NET خاصیت جدیدی بنام Anchor دارند که مشخص می‌کند هنگام تغییر اندازهٔ فرم فاصلهٔ کدام لبهٔ کنترل باید نسبت به لبه‌های فرم ثابت بماند. خاصیت جدید دیگری نیز بنام Dock وجود دارد، که اشیاء را وادار می‌کند تا همواره به یک فاصله از لبه‌های فرم بمانند. باترکیب این دوخاصیت می‌توانید فرمهایی بسازید که کنترلها پابه‌پای فرم کوچک و بزرگ شوند.
- در ویژوال بیسیک ۶ برای ایجاد برنامه‌های MDI (واسط چند سندی) بایستی با فرمان Project|Add MDI Form فرم مادر MDI را به پروژه اضافه می‌کردید. در ویژوال بیسیک .NET فرم مادر MDI مانند سایر فرمهاست، که فقط خاصیت IsMdiContainer آن True شده است. فرمهای دختر MDI نیز فرمهای معمولی هستند، که خاصیت MdiParent آنها را به نام فرم مادر ست کرده‌ایم.

اضافه‌کردن فرمهای جدید به برنامه

برنامه‌هایی که تا اینجا نوشتیم، فقط یک فرم داشتند و همهٔ کارها را در این فرم انجام می‌دادیم. در بسیاری از برنامه‌ها همین یک فرم کافیست، اما اگر بخواهید تبادل اطلاعات گسترده‌تری با کاربر داشته باشید، ویژوال بیسیک به شما اجازه می‌دهد تا فرمهای دیگری به برنامهٔ خود اضافه کنید. هر فرم جدید شیء مستقلیست که خواص خود را از کلاس System.Windows.Forms.Form به ارث می‌برد. همانطور که دیدید، فرم اول برنامه Form1.vb نام دارد؛ فرمهای بعدی به ترتیب Form2.vb ، Form3.vb و الی‌آخر نامگذاری می‌شوند (که البته این نامگذاری قابل تغییر است). در جدول زیر چند مورد از مواردی که فرمهای اضافی می‌توانند مفید باشند، را ملاحظه می‌کنید:

فرم یا فرمهای	توضیح
فرم معرفی	فرمی که در شروع برنامه ظاهر می‌شود، و آن را معرفی می‌کند.
فرم طرز کار برنامه	فرمی که دستورات و اطلاعات لازم برای کار با برنامه را ارائه می‌کند.
دیالوگ‌ها	فرمهای خاصی که برای تبادل اطلاعات با کاربر بکار می‌روند.
محتویات سندها	فرمهایی که برای نمایش محتویات اسناد برنامه بکار می‌روند.

طرز استفاده از فرمها

کار با فرمها در ویژوال بیسیک بسیار ساده است: می‌توانید همهٔ آنها را با هم نشان دهید، و یا آنها را به نوبت و یکی یکی نمایش داده و سپس مخفی کنید. دیالوگ‌ها (که در ویژوال بیسیک ۶ به آنها **فرمهای مودال** – modal forms – گفته می‌شود) آنهایی هستند که تا از روی صفحه کنار نروند، امکان کار با فرمهای دیگر وجود نخواهد داشت. برای نمایش یک فرم بصورت دیالوگ، باید از متد ShowDialog برای نمایش آن استفاده کنید. ولی اگر می‌خواهید فرم را بصورتی نمایش دهید که همزمان برای کاربر امکان سوئیچ کردن به فرمهای دیگر وجود داشته باشد، باید از متد Show برای نمایش آن استفاده کنید (به این نوع از فرمها، **فرمهای غیرمودال** – modeless forms – گفته می‌شود).

در اکثر برنامه‌های ویندوز از فرمهای غیرمودال استفاده می‌شود، بنابراین فرمهای پیش‌فرض ویژوال استودیو نیز از همین نوع هستند. اگر می‌خواهید ببینید فرم برنامهٔ شما از چه نوعیست، می‌توانید کُد نوشته شده در قسمت Windows Form Designer generated code را بررسی کنید (اکنون دیگر آنقدر با کُدنویسی آشنا هستید، که بتوانید تا حد زیادی کُدهای این قسمت را درک کنید).

برنامه‌های چندفرمه

در تمرین زیر فرم دیگری برای نمایش اطلاعات **کمک** (Help) به برنامهٔ Lucky Seven اضافه می‌کنیم. این فرم، که یک فرم مودال (یا دیالوگ) خواهد بود، محتویات فایل Readme.txt را نمایش خواهد داد (فصل ۱۴ را ببینید).

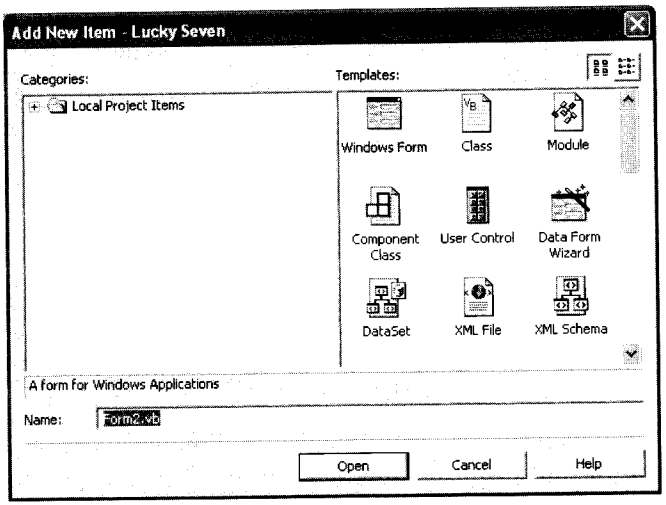
اضافه کردن فرم دوم

۱ در محیط ویژوال استودیو .NET ، و از پوشهٔ c:\vbnetsbs\chap15\lucky seven ، پروژهٔ Lucky Seven را باز کنید. (این برنامه دقیقاً همان برنامهٔ فصل قبل است، البته بدون پروژهٔ توزیع.)

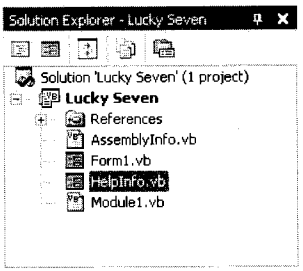
۲ به فرم اصلی برنامه (Form1.vb) بروید.

۳ برای اضافه کردن فرم جدید به برنامه، فرمان Add Windows Form را از منوی Project اجرا کنید، تا پنجرهٔ «**اضافه کردن آیتم جدید**» باز شود (شکل زیر را ببینید).

دیالوگ «اضافه کردن آیتم جدید» اجازه می‌دهد تا فرم، ماژول، کلاس، و یا هر کدام از دیگر اجزای موجود در ویژوال بیسیک را به برنامهٔ خود اضافه کنید. با این که فرمانی که ما اجرا کردیم، فرمان Add Windows Form بود، اما در این دیالوگ آیتمهای دیگری را نیز خواهید دید (البته آیتم پیش‌فرض همان Windows Form است). اگر وسط کار نظرتان عوض شد، لازم نیست جای دیگری بروید، پنجرهٔ «اضافه کردن آیتم جدید» همه کاره است!



٤ نام فرم جدید را (در فیلد Name) **HelpInfo.vb** گذاشته، و دکمهٔ Open را کلیک کنید. با این کار، فرم جدیدی بنام HelpInfo.vb به کاوشگر راه‌حل پروژهٔ Lucky Seven اضافه خواهد شد:



نکته

کاوشگر راه‌حل به شما اجازه می‌دهد تا نام فایلها را عوض کرده، و یا آنها را حذف کنید. برای تغییر دادن نام یک فایل، روی آن راست-کلیک کرده و فرمان Rename را انتخاب کنید. برای خارج کردن یک فایل از پروژه، روی آن راست-کلیک کرده و فرمان Exclude From Project را انتخاب کنید (با این کار فایل مزبور فقط از پروژه حذف می‌شود، ولی همچنان در کامپیوتر باقی خواهد ماند). برای حذف کامل و همیشگی یک فایل از روی هارد دیسک، روی آن راست-کلیک کرده و فرمان Delete را انتخاب کنید.

یک برچسب با پهنای زیاد (تقریباً معادل پهنای فرم) بالای فرم HelpInfo.vb رسم کنید.

یک جعبه متن روی فرم رسم کنید.

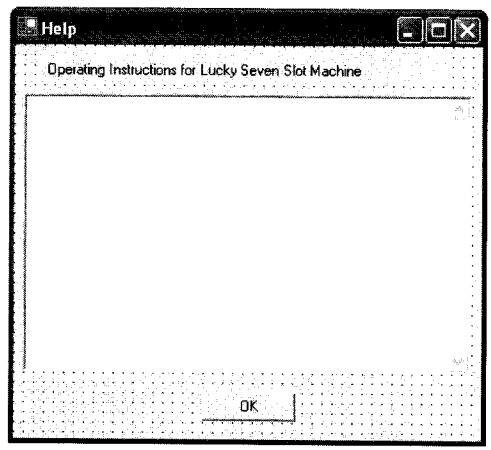
خاصیت Multiline جعبه متن را به True ست کنید.

جعبه متن را به اندازه‌ای درآورید، که تقریباً تمام فرم HelpInfo.vb را بپوشاند.

۹ یک دکمه هم در پائین فرم قرار دهید.

۱۰ خواص فرم HelpInfo.vb و کنترلهای روی آنرا با توجه به جدول زیر ست کنید (شکل زیر را ببینید):

Form1	Text	"Help"
Button1	Text	"OK"
Label1	Text	"Operating Instructions for Lucky Seven Slot Machine"
TextBox1	Multiline	True
	Scrollbars	Vertical
	Text	(خالی)



۱۱ روی دکمهٔ OK دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد باز شود.

۱۲ دستور زیر را در این روال بنویسید:

```
Me.DialogResult = DialogResult.OK
```

فرم HelpInfo.vb در این پروژه بصورت یک دیالوگ عمل می‌کند، چون فرم Form1 با متد ShowDialog آنرا باز خواهد کرد. بعد از اینکه کاربر کارش با فرم HelpInfo.vb تمام شد، برای بستن آن روی دکمهٔ OK کلیک خواهد کرد، بهمین دلیل این فرم مقدار DialogResult.OK را به روال فراخوانی‌کننده برمی‌گرداند. یک دیالوگ می‌تواند بطرق دیگری نیز بسته شود، و در هر موقعیت باید مقدار مناسب را برگرداند؛ برخی از این مقادیر عبارتند از: DialogResult.Cancel ، DialogResult.No ، DialogResult.Yes ، و DialogResult.Abort . توجه کنید که بمحض دادن مقدار به خاصیت DialogResult ، فرم بسته خواهد شد.

۱۳ در ادیتور کُد، به بالای فرم رفته و دستور زیر را در آنجا وارد کنید:

```
Imports System.IO
```

از آنجائیکه می‌خواهیم در فرم HelpInfo.vb از کلاس StreamReader استفاده کنیم، کلاس System.IO را به آن اضافه کرده‌ایم.

۱۴ به فرم HelpInfo.vb برگردید، و روی زمینهٔ فرم دو-کلیک کنید، تا روال رویداد HelpInfo_Load در ادیتور کُد باز شود. روال HelpInfo_Load روالیست که بمحض بار شدن فرم در حافظه (و ظاهر شدن روی صفحه) اجرا می‌شود.

۱۵ دستورات زیر را در این روال بنویسید:

```
Dim StreamToDisplay As StreamReader
StreamToDisplay = New StreamReader("c:\vbnetsbs\chap14\readme.txt")
TextBox1.Text = StreamToDisplay.ReadToEnd
StreamToDisplay.Close()
TextBox1.Select(0, 0)
```

با کلاس StreamReader در فصل ۱۲ آشنا شدید؛ اینجا هم بجای نوشتن مستقیم محتویات فایل کمک، با این کلاس فایل Readme.txt را در جعبه متن بار کرده‌ایم.

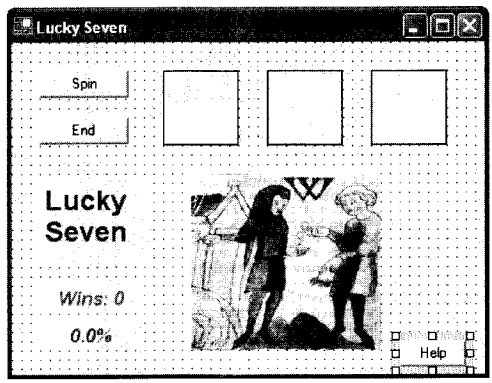
این از فرم HelpInfo.vb . اما فرم اصلی برنامه (Form1.vb) نیز به چند دستکاری کوچک نیاز دارد.

نمایش فرم دوم

۱ در کاوشگر راه‌حل، فرم Form1.vb را انتخاب کرده و روی دکمهٔ View Designer کلیک کنید، تا این فرم در محیط طراحی ویژوال استودیو ظاهر شود.

۲ یک دکمهٔ کوچک در گوشهٔ راست-پائین فرم رسم کنید.

۳ خاصیت Text این دکمه را به **Help** ست کنید (شکل زیر را ببینید):



۴ روی دکمهٔ Help دو-کلیک کنید، تا روال رویداد Button3_Click در ادیتور کُد باز شود.

دستورات زیر را در این روال بنویسید:

```
Dim frmHelpDialog As New HelpInfo()
frmHelpDialog.ShowDialog()
```

این دو دستور تمام کاریست که برای نمایش فرم دوم (بصورت دیالوگ) باید انجام دهید. بر خلاف ویژوال بیسیک ۶که برای نمایش فرم جدید فقط کافی بود از نام آن استفاده کنید (تکنیکی که به ایجاد وهله بصورت ضمنی - Implicit Instantiation - معروف است)، در ویژوال بیسیک .NET برای کار با فرم دوم باید ابتدا یک متغیر از آن ایجاد کنید. وقتی فرم HelpInfo.vb را به پروژه اضافه می‌کنید، یک کلاس جدید بنام HelpInfo نیز در برنامه ایجاد خواهد شد. بدین ترتیب، دستور Dim frmHelpDialog As New HelpInfo() یک متغیر وهله (بنام frmHelpDialog) از کلاس HelpInfo تعریف می‌کند.

دستور دوم، فرم HelpInfo را با متد ShowDialog (بصورت یک دیالوگ) باز می‌کند. اما اگر از متد Show برای باز کردن فرم HelpInfo استفاده کنید، این فرم دیگر مودال نخواهد بود، و کاربر می‌تواند بین فرمهای برنامه سوئیچ کند. یک فرم غیرمودال دیگر با ست کردن خاصیت DialogResult بسته نخواهد شد، بلکه برای بستن آن باید دستور Me.Close را بکار ببرید. (تفاوت فرمهای مودال و غیرمودال را بخاطر داشته باشید، و از هر کدام در جای خود استفاده کنید.)

اکنون وقت آنست که ببینیم یک برنامهٔ چندفرمه چگونه کار می‌کند. (کُد کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap15\multiple forms پیدا کنید.)

استفاده از خاصیت DialogResult

در برنامهٔ Multiple Forms از مقدار برگشتی دیالوگ HelpInfo (خاصیت DialogResult) استفاده‌ای نکردیم، ولی با آن کارهای زیادی می‌توان کرد. همانطور که قبلاً هم گفتم، یک دیالوگ می‌تواند (بسته به دکمه‌ای که کاربر با آن دیالوگ را بسته) مقدارهای دیگری نیز برگرداند - بعبارت دیگر، هر یک از دکمه‌های دیالوگ می‌تواند مقدار خاصی را برگرداند. مثلاً، اگر دیالوگ ما یک دکمهٔ Cancel نیز داشته باشد، روال رویداد Click این دکمه می‌تواند چنین باشد:

```
Me.DialogResult = DialogResult.Cancel        'user clicked Cancel button
```

و برای تشخیص مقدار برگشتی دیالوگ در روال فراخوانی‌کننده، می‌توانیم از ساختارهای تصمیم‌گیری استفاده کرده و عکس‌العمل مناسب را نشان دهیم:

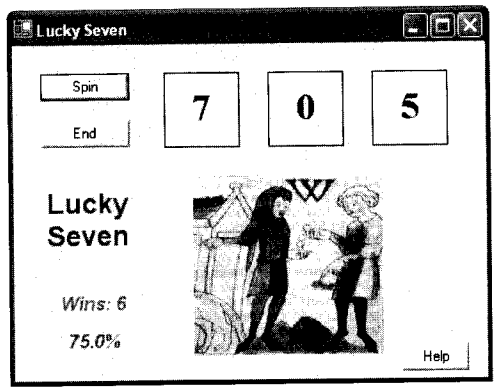
```
Dim frmHelpDialog As New HelpInfo()
frmHelpDialog.ShowDialog()

If frmHelpDialog.DialogResult = DialogResult.OK Then
    MsgBox("The user clicked OK")
ElseIf frmHelpDialog.DialogResult = DialogResult.Cancel Then
    MsgBox("The user clicked Cancel")
Else
    MsgBox("Another button was clicked")
End If
```

استفادهٔ خلاقانه از فرمها و دیالوگها یکی از عناصر کلیدی در تولید برنامه‌های حرفه‌ای است.

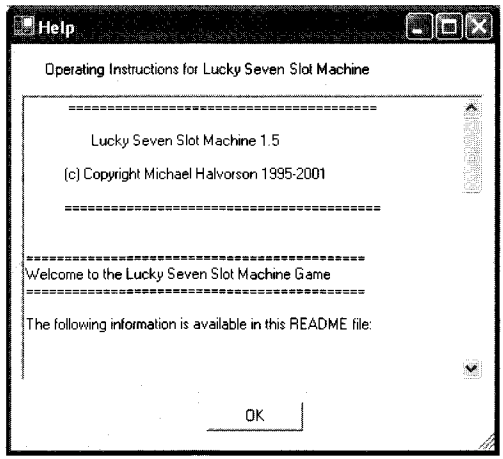
اجرای برنامه

۱ با کلیک کردن دکمهٔ Start ، برنامهٔ Multiple Forms را اجرا کنید.

۲ کمی با برنامه بازی کنید (شکل زیر را ببینید):



۳ دکمهٔ Help را کلیک کنید؛ با این کار ویژوال بیسیک فرم دوم (HelpInfo.vb) را باز کرده، و محتویات فایل Readme.txt را در جعبه متن نمایش می‌دهد:



٤ کمی در جعبه متن بالا و پائین رفته، و آنرا مطالعه کنید.

۵ برای بستن فرم HelpInfo.vb ، دکمهٔ OK را کلیک کنید؛ با این کار دوباره به فرم اصلی برنامه برخواهید گشت.

۶ کمی دیگر با برنامه بازی کرده، و دوباره دکمهٔ Help را کلیک کنید؛ بله، فرم Help بخوبی کار می‌کند! توجه کنید که وقتی فرم HelpInfo.vb باز است، امکان رفتن (سوئیچ کردن) به فرم اول برنامه وجود ندارد، چون این فرم را بصورت مودال باز کرده‌ایم.

۷ با کلیک کردن دکمهٔ OK فرم HelpInfo.vb را بسته، و با دکمهٔ End از هم برنامه خارج شوید.

تعیین مکان فرمها روی میزکار ویندوز

تا اینجا یاد گرفتید که چگونه یک فرم به برنامه اضافه کرده، آنرا باز کنید و سپس ببندید. اما این فرم کجای صفحهٔ مانیتور (و در واقع میزکار ویندوز) ظاهر می‌شود؟ همانطور که قبلاً هم گفتم، محل قرار گرفتن فرم روی مانیتور وقتیکه در حال طراحی است، از محل ظاهر شدن آن هنگام اجرای برنامه متفاوت است. در این قسمت خواهید دید که چگونه می‌توان محل ظاهر شدن فرم برنامه را در زمان اجرای آن کنترل کرد.

در ویژوال بیسیک ۶ برای تعیین محل فرم روی میزکار ویندوز از پنجرهٔ Form Layout استفاده می‌کردیم: با کشیدن آیکون کوچکی که نمایندهٔ فرم برنامه بود، می‌توانستیم محل آنرا براحتی تعیین کنیم. در ویژوال بیسیک .NET پنجرهٔ Form Layout دیگر وجود ندارد، اما همچنان می‌توانید به کمک خاصیت DesktopBounds محل فرم را روی میزکار ویندوز مشخص کنید. با این خاصیت می‌توانید مختصات محل قرار گرفتن فرم را بصورت دقیق ست کنید. خاصیت DesktopBounds مختصات یک مستطیل (و در واقع دو رأس آن) را بعنوان آرگومان می‌گیرد. این اعداد بر حسب پیکسل هستند، و از لبه‌های چپ و بالای صفحهٔ مانیتور (و بعبارت بهتر، از گوشهٔ چپ-بالای آن) اندازه گیری می‌شوند. (در فصل آینده با سیستم مختصات ویژوال بیسیک بیشتر آشنا خواهید شد.)

برای تعیین محل فرم روی صفحه مکانیزم دیگری نیز وجود دارد، و آن استفاده از خاصیت StartPosition است (که البته امکانات آن از خاصیت DesktopBounds کمتر است). خاصیت StartPosition پنج مقدار می‌تواند بگیرد: Manual (تنظیم دستی)، CenterScreen (وسط صفحه)، WindowsDefaultLocation (محل پیش‌فرض ویندوز)، WindowsDefaultBounds (مستطیل پیش‌فرض ویندوز)، یا CenterParent (وسط پنجرهٔ مادر). مقدار پیش‌فرض این خاصیت WindowsDefaultLocation است، که به ویندوز اجازه می‌دهد تا محل فرم را خود تعیین کند (او هم اغلب آنرا به گوشهٔ چپ-بالای صفحه می‌چسباند!).

اگر خاصیت StartPosition را به Manual ست کنید، می‌توانید محل فرم را بصورت دستی تنظیم کنید؛ این کار از طریق خاصیت Location صورت می‌گیرد. خاصیت Location دو عدد می‌گیرد، که اولی (x) فاصلهٔ فرم از لبهٔ چپ صفحه، و دومی (y) فاصلهٔ آن از لبهٔ بالای چپ صفحه است (در فصل آینده با خاصیت Location بیشتر آشنا می‌شوید).

اگر خاصیت StartPosition را به CenterScreen ست کنید، فرم همیشه وسط صفحه ظاهر خواهد شد (و شاید این مناسبترین حالت باشد). اگر این خاصیت را به WindowsDefaultBounds ست کنید، فرم به اندازهٔ پنجره‌های استاندارد ویندوز در آورده شده، و در محل استاندارد پنجره‌های ویندوز نیز ظاهر خواهد شد. و بالاخره، اگر خاصیت StartPosition را به CenterParent ست کنید، فرم در وسط فرم مادر ظاهر می‌شود (این مقدار بویژه برای برنامه‌های MDI مفید است).

در تمرینهای زیر از خاصیت‌های StartPosition و DesktopBounds برای تعیین مکان فرم برنامه روی میزکار ویندوز استفاده خواهیم کرد.

استفاده از خاصیت StartPosition برای تغییر مکان فرم

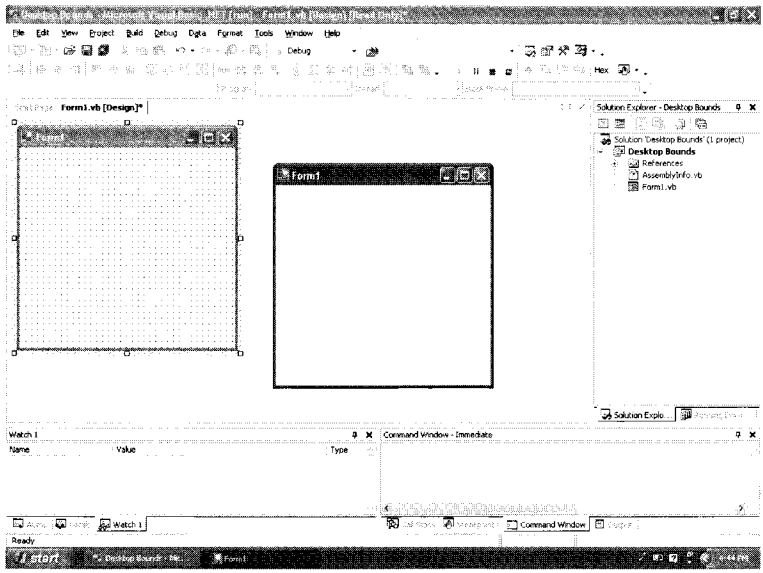
۱ با فرمان File|Close Solution پروژهٔ قبلی را بسته، و یک پروژهٔ جدید Visual Basic Windows Application بنام **My Desktop Bounds** در پوشهٔ c:\vbnetsbs\chap15 ایجاد کنید.

۲ فرم پروژه را باز کنید.

۳ روی فرم کلیک کنید، تا خواص آن در پنجرهٔ خواص ظاهر شود.

٤ خاصیت StartPosition را به CenterScreen ست کنید؛ با این کار فرم برنامه همیشه در وسط میزکار ویندوز ظاهر خواهد شد.

۵ برنامه را اجرا کنید؛ می‌بینید که ویژوال بیسیک فرم را در وسط صفحهٔ مانیتور قرار داده است:

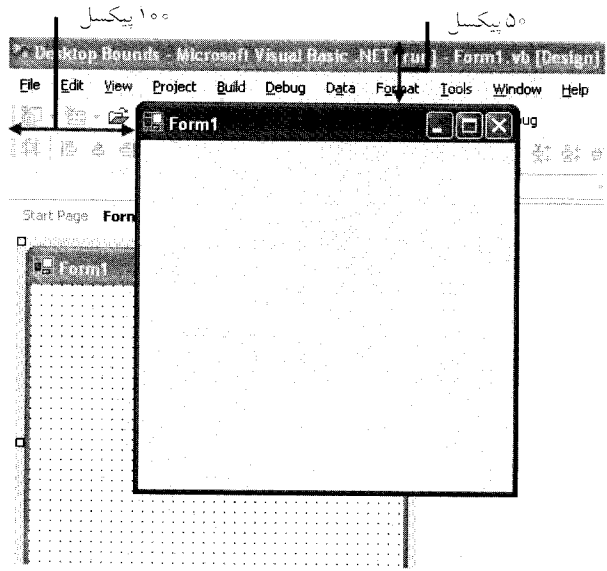


۶ برنامه را ببندید، و به محیط طراحی ویژوال استودیو برگردید.

۷ خاصیت StartPosition را به Manual ست کنید؛ با این کار می‌توانید محل فرم را بصورت دستی تنظیم کنید.

۸ خاصیت Location را به **100, 50** ست کنید. خاصیت Location مختصات محل گوشهٔ چپ-بالای فرم را مشخص می‌کند.

۹ برنامه را اجرا کنید؛ همانطور که می‌بینید، ویژوال بیسیک فرم را بگونه‌ای روی میزکار ویندوز قرار می‌دهد که ۱۰۰ پیکسل از لبهٔ چپ صفحه، و ۵۰ پیکسل از لبهٔ بالای صفحه فاصله داشته باشد:



۱۰ برنامه را ببندید، و به محیط طراحی ویژوال استودیو برگردید.

در تمرین بعدی، ابتدا (بدون استفاده از فرمان Project|Add Windows Form) یک فرم جدید ساخته، و سپس با خاصیت DesktopBounds محل آنرا تعیین می‌کنیم.

ست‌کردن خاصیت DesktopBounds

۱ یک دکمه روی فرم قرار داده، و خاصیت Text آنرا به **Create Form** ست کنید.

۲ روی دکمهٔ Create Form دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد ظاهر شود.

۳ کُد زیر را در این روال بنویسید:

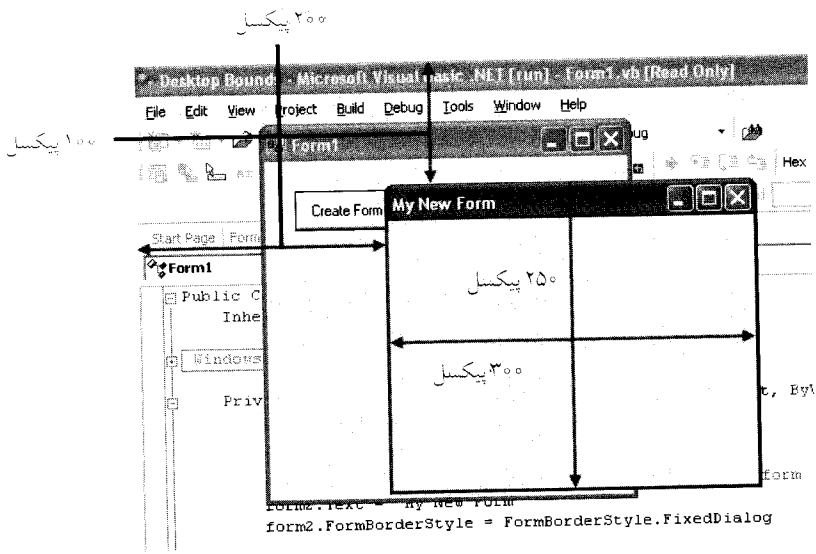
```
'Create a second form named form2
Dim form2 As New Form()

'Define the Text property and border style of the form
form2.Text = "My New Form"
form2.FormBorderStyle = FormBorderStyle.FixedDialog

'Specify that the position of the form will be set manually
form2.StartPosition = FormStartPosition.Manual

'Declare a Rectangle structure to hold the form dimensions
'Upper left corner of form (200, 100)
'Width and height of form (300, 250)
Dim Form2Rect As New Rectangle(200, 100, 300, 250)

'Set the bounds of the form using the Rectangle object
form2.DesktopBounds = Form2Rect

'Display the form as a modal dialog box
form2.ShowDialog()
```

همانطور که می‌بینید، برای ایجاد فرم جدید ابتدا یک وهله از کلاس Form (کلاسی که جزء فضای نام System.Windows.Forms است) تعریف می‌شود. پس از آن می‌توان هر یک از خواص این فرم جدید (مانند FormBorderStyle ، Text ، StartPosition ، و DesktopBounds) را ست کرد. در اینجا خاصیت StartPosition را به FormStartPosition.Manual ست کرده‌ایم، تا بتوانیم محل فرم را بدلخواه تعیین کنیم.

خاصیت DesktopBounds برای تعیین اندازه و مکان فرم به آرگومانی از نوع Rectangle (مستطیل) نیاز دارد (ساختار Rectangle جزء ساختارهای تعریف شده در ویژوال بیسیک است). دو آرگومان اول در ساختار Rectangle مختصات گوشهٔ چپ-بالای مستطیل فرضی، و دو آرگومان بعدی پهنا و ارتفاع آن هستند. در پایان هم، فرم form2 با استفاده از متد ShowDialog (بصورت یک دیالوگ) روی صفحه ظاهر می‌شود.

٤ برنامه را اجرا کنید، تا فرم اصلی برنامه روی صفحه ظاهر شود.

٥ روی دکمهٔ Create Form کلیک کنید؛ همانطور که می‌بینید، ویژوال بیسیک فرم form2 را در مستطیلی با مختصات (200, 100) و (500, 350) ، و با ابعاد مشخص شده (300 و 250) روی میزکار ویندوز قرار می‌دهد:



از آنجائیکه خاصیت FormBorderStyle فرم دوم به FixedDialog ست شده، امکان تغییر دادن اندازهٔ آن وجود ندارد.

٦ ابتدا فرم دوم و سپس فرم اول برنامه را ببندید، و به محیط طراحی ویژوال استودیو برگردید.

حداقل و حداکثر کردن پنجره‌ها

علاوه بر تعیین مکان یک فرم روی میزکار ویندوز، می‌توان آنرا حداقل (minimize) یا حداکثر (maximize)

کرد، و یا به حالت اولیه برگرداند (restore) . برای آنکه کاربر بتواند یک فرم را حداقل یا حداکثر کند، باید آیکونهای Minimize و Maximize را (در میلهٔ عنوان فرم) در اختیار داشته باشد. برای این منظور کافیست از طریق پنجرهٔ خواص، خاصیت‌های MinimizeBox و MaximizeBox را به True ست کنید.

حال اگر در برنامه بخواهیم حالت پنجرهٔ فرم را عوض کنیم، باید خاصیت WindowState آنرا به Minimized ، Maximized ، یا Normal ست کنیم. دستور زیر

```
WindowState = FormWindowState.Minimized
```

پنجرهٔ فرم را به حالت حداقل (یک دکمه در میلهٔ تکالیف ویندوز) در می‌آورد.

اگر می‌خواهید اندازهٔ حالتهای حداقل و حداکثر فرم را در کنترل خود داشته باشید، می‌توانید خواص MinimumSize و MaximumSize را باندازهٔ دلخواه ست کنید؛ و برای این منظور باید از ساختاری بنام Size (که شبیه Rectangle است) استفاده کنید:

```
Dim FormSize As New Size(400, 300)
MaximumSize = FormSize
```

اضافه کردن کنترل‌ها به فرم در زمان اجرای برنامه

تا اینجا هر وقت می‌خواستیم کنترلی را به فرم برنامه اضافه کنیم، از جعبه ابزار ویژوال بیسیک استفاده می‌کردیم. با این حال مواردی پیش می‌آید که بخواهیم کنترلها را در زمان اجرای برنامه به آن اضافه کنیم؛ ایجاد فرمهای متنوعی که وابسته به انتخابهای کاربر باشند، یکی از این موارد است.

ایجاد اشیاء جدید در حین اجرای برنامه بسیار ساده است، چون کلاسهای کلیهٔ کنترلهای موجود در جعبه ابزار ویژوال بیسیک در اختیار تمام برنامه‌ها قرار دارد. برای ایجاد یک شیء جدید باید از دستور

```
Dim ObjectName As New BaseClassName
```

(که تاکنون بارها آنرا دیده‌اید) استفاده کنیم. برای مثال، دستور زیر یک دکمهٔ جدید بنام button1 می‌سازد:

```
Dim button1 As New Button()
```

پس از آن باید خواص این دکمه را ست کنیم؛ و قبل از هر چیز به دو خاصیت Text و Location مقدار بدهیم، چون ویژوال بیسیک هیچ مقدار پیش‌فرضی به آنها نمی‌دهد:

```
button1.Text = "Click Me"
button1.Location = New Point(20, 25)
```

و بالاخره، باید این شیء جدید را به کلکسیون کنترلهای فرم اضافه کنیم، تا فعال و روی فرم ظاهر شود:

```
form2.Collection.Add(button1)
```

با این روش می‌توان تمام انواع کنترلهای موجود در جعبه ابزار ویژوال بیسیک را به هر فرمی اضافه کرد. در تمرین زیر یک برچسب (برای نمایش تاریخ فعلی) و یک دکمه (برای بستن دیالوگ) به فرم form2 اضافه خواهیم کرد.

ایجاد کنترلهای برچسب و دکمه

۱ با فرمان File|Close Solution پروژهٔ قبلی را بسته، و یک پروژهٔ جدید Visual Basic Windows Application بنام **My Add Controls** در پوشهٔ c:\vbnetsbs\chap15 ایجاد کنید.

۲ فرم پروژه (Form1.vb) را باز کنید.

۳ یک دکمه روی فرم قرار داده، و خاصیت Text آنرا به **Display Date** ست کنید.

٤ روی دکمهٔ Display Date دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد ظاهر شود.

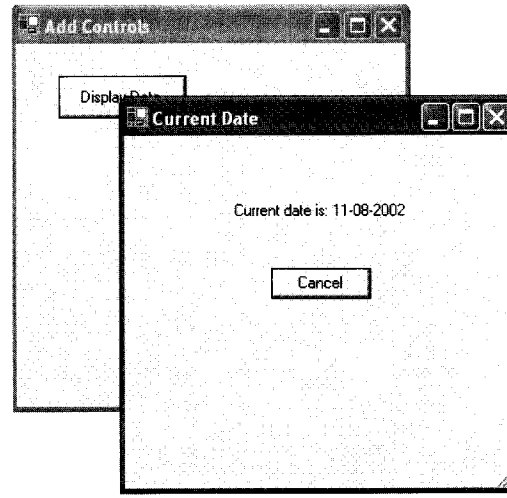
۵ کُد زیر را در این روال بنویسید:

```
'Declare new form and control objects
Dim form2 As New Form()
Dim lblDate As New Label()
Dim btnCancel As New Button()

'Set label properties
lblDate.Text = "Current date is: " & DateString
lblDate.Size = New Size(150, 50)
lblDate.Location = New Point(80, 50)

'Set button properties
btnCancel.Text = "Cancel"
btnCancel.Location = New Point(110, 100)

'Set form properties
form2.Text = "Current Date"
form2.CancelButton = btnCancel
form2.StartPosition = FormStartPosition.CenterScreen

'Add new objects to Controls collection
form2.Controls.Add(lblDate)
form2.Controls.Add(btnCancel)

'Display form as a dialog box
form2.ShowDialog()
```

این کُد به همان روش تمرین قبل، فرم جدیدی بنام form2 بوجود آورده، سپس دو کنترل (برچسب و دکمه) ایجاد کرده، و با ست کردن خواص Size و Location آنها را باندازهٔ دلخواه درآوده و در محل مناسب قرار می‌دهد. برای آنکه کاربر بتواند با کلیک کردن دکمهٔ Cancel (و یا زدن کلید Esc) فرم را ببندد، خاصیت CancelButton آنرا به دکمهٔ btnCancel ست کرده‌ایم.

۶ برنامه را اجرا کنید، تا فرم اصلی برنامه روی صفحه ظاهر شود.

۷ روی دکمهٔ Display Date کلیک کنید؛ همانطور که می‌بینید، ویژوال بیسیک فرم جدیدی ایجاد کرده، و یک دکمه (با عنوان Cancel) و یک برچسب (که تاریخ فعلی سیستم را نشان می‌دهد) روی آن قرار می‌دهد.



۸ با کلیک کردن دکمهٔ Cancel ، فرم جدید را ببندید.

۹ یک بار دیگر روی دکمهٔ Display Date کلیک کنید، تا فرم Current Date دوباره باز شود.

۱۰ برای بستن فرم Current Date ، این بار کلید Esc را بزنید. از آنجائیکه خاصیت CancelButton فرم form2 به دکمهٔ btnCancel ست شده، زدن کلید Esc معادل کلیک کردن دکمهٔ Cancel است، و باعث بسته شدن فرم خواهد شد.

۱۱ برنامه را ببندید، و به محیط طراحی ویژوال استودیو برگردید.

سازماندهی کنترلهای فرم

وقتی کنترلها را با استفاده از کُد روی فرم قرار می‌دهید، مرتب و چشم‌نواز کردن آنها کمی مشکل است. در این وضعیت دیگر ابزاری مانند طراح فرمهای ویندوز را در اختیار ندارید، و فقط باید با استفاده از خواص Size و Location کنترلها را باندازهٔ مناسب درآورده، و در جای مناسب قرار دهید (و در این حالت هم یا باید قوهٔ تخیلی قوی داشته باشید، و یا بارها برنامه را اجرا کنید تا به نتیجهٔ دلخواه برسید).

خوشبختانه در ویژوال بیسیک .NET خواص متعددی وجود دارد، که به کمک آنها می‌توانید اشیاء را روی فرم سازماندهی و مرتب کنید. یکی از این خواص Anchor است، که باعث می‌شود تا یک کنترل همواره در فاصلهٔ معینی از لبه‌های فرم قرار گیرد؛ خاصیت دیگر Dock است، که باعث می‌شود تا یک کنترل نسبت به یکی از لبه‌های فرم در وضعیت ثابتی داشته باشد. در تمرین زیر طرز استفاده از خواص Anchor و Dock را برای سازماندهی کنترلها از طریق کُد برنامه خواهید دید.

استفاده از خواص Anchor و Dock

۱ با فرمان File|Close Solution پروژهٔ قبلی را بسته، و یک پروژهٔ جدید Visual Basic Windows Application بنام **My Anchor and Dock** در پوشهٔ c:\vbnetsbs\chap15 ایجاد کنید.

۲ فرم پروژه (Form1.vb) را باز کنید.

۳ یک کنترل جعبه تصویر (PictureBox) در بالای فرم (و با فاصلهٔ یکسان از دو طرف آن) قرار دهید.

٤ یک جعبه متن در وسط فرم (زیر جعبه تصویر) قرار دهید.

۵ خاصیت Multiline جعبه متن را به True ست کنید.

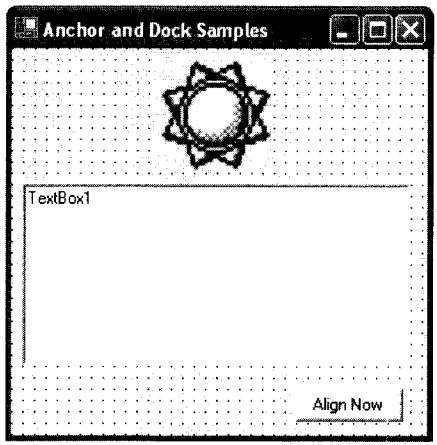
۶ جعبه متن را آنقدر بزرگ کنید، که تقریباً نیمهٔ پائین فرم را بپوشاند.

۷ یک دکمه زیر جعبه متن (و متمایل به سمت راست) قرار دهید.

۸ خواص فرم و کنترلهای روی آن را با توجه به جدول زیر ست کنید:

شیء	خاصیت	مقدار
Form1	Text	"Anchor and Dock Samples"
Button1	Text	"Align Now"
PictureBox1	Image	"c:\vbnetsbs\chap16\sun.ico"
	SizeMode	StretchImage

تا اینجا، فرم برنامه باید چیزی شبیه شکل زیر شده باشد:



روی دکمهٔ Align Now دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد ظاهر شود.

کُد زیر را در این روال بنویسید:

```
PictureBox1.Dock = DockStyle.Top
TextBox1.Anchor = AnchorStyles.Bottom Or AnchorStyles.Left _
    Or AnchorStyles.Right Or AnchorStyles.Top
Button1.Anchor = AnchorStyles.Bottom Or AnchorStyles.Right
```

اجرای دستور PictureBox1.Dock = DockStyle.Top باعث می‌شود تا جعبه تصویر برای همیشه به لبهٔ بالای فرم (درست زیر میلهٔ عنوان) بچسبد. اما این کنترل رفتار عجیب دیگری نیز از خود نشان می‌دهد که در مرحلهٔ بعد (وقتی برنامه را اجرا کردید) متوجه آن خواهید شد: لبه‌های چپ و راست جعبه تصویر نیز به لبه‌های فرم می‌چسبند، بگونه‌ایکه وقتی فرم را بزرگ یا کوچک می‌کنید، جعبه تصویر نیز همپای فرم کوچک و بزرگ می‌شود.

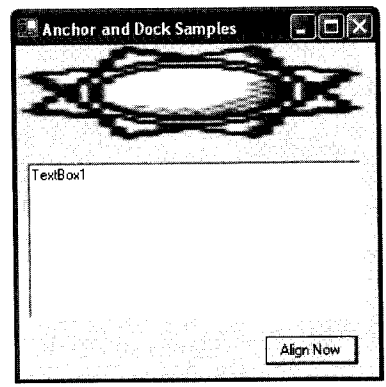
دستور بعدی خواص Anchor جعبه متن را ست می‌کند: هر خاصیت AnchorStyles.*SideName* یکی از لبه‌های جعبه متن را در فاصله‌ای ثابت نسبت به همان لبهٔ فرم نگه می‌دارد. توجه کنید که چگونه برای ترکیب خواص چهارگانهٔ Anchor از Or استفاده کرده‌ایم. دستور آخر نیز باعث می‌شود تا فاصلهٔ لبه‌های راست و پائین دکمهٔ Align Now نسبت به لبه‌های متناظر فرم ثابت بماند.(کُد کامل این برنامه را می‌توانید در پوشهٔ c:\vbnetsbs\chap15\anchor and dock ببینید.)

۱۱ برنامه را اجرا کنید، تا فرم اصلی برنامه روی صفحه ظاهر شود.

۱۲ ماوس را روی گوشهٔ راست-پائین فرم ببرید، تا شکل کرسر به پیکان دو-سر مورب تبدیل شود؛ این نقطه را گرفته و فرم برنامه را بزرگ کنید. توجه کنید که اندازه و مکان هیچیک از اشیاء روی فرم تغییری نمی‌کند.

۱۳ فرم را به اندازهٔ قبلی برگردانید.

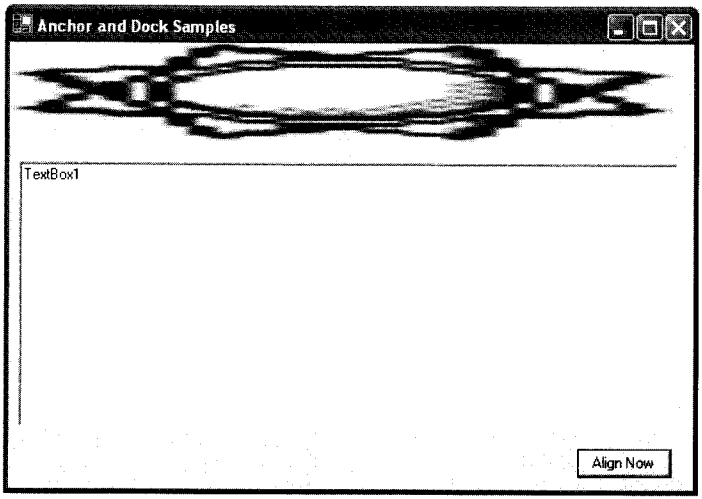
۱۴ روی دکمهٔ Align Now کلیک کنید. با این کار، لبهٔ بالایی جعبه تصویر به بالای فرم می‌چسبد؛ و همانطور که گفتم، لبه‌های چپ و راست آن هم به لبه‌های فرم خواهند چسبید (شکل زیر را ببینید):



همانطور که می‌بینید، تصویر sun.ico بعلت اجبار در چسبیدن به لبه‌های فرم، دچار اعوجاج شده است.

۱۵ فرم را بزرگ کنید. همانطور که می‌بینید، همراه با بزرگ شدن فرم، جعبه تصویر و جعبه متن نیز بزرگ می‌شوند. از آنجائیکه خاصیت Anchor جعبه متن باعث شده تا فاصلهٔ لبه‌های آن نسبت به

لبه‌های فرم ثابت بماند، بزرگ شدن فرم باعث کشیده شدن جعبه متن خواهد شد. اما با بزرگ شدن فرم، دکمهٔ Align Now فقط به سمت راست و پائین جابجا می‌شود، چون فقط این دو لبهٔ به لبه‌های فرم قفل شده‌اند.



۱۶ کمی دیگر با برنامه کار کنید، و سپس آن را ببندید.

یک گام فراتر: تعیین شیء شروع برنامه

اگر برنامه‌ای بیش از یک فرم داشته باشد، در شروع برنامه کدامیک از آنها ابتدا بار و نمایش داده می‌شود؟ با اینکه ویژوال بیسیک معمولاً اولین فرم (Form1.vb) را در ابتدای کار نمایش می‌دهد، اما با استفاده از دیالوگ خواص پروژه (Project|Project Properties) می‌توان این وضعیت را تغییر داد. حتی می‌توان ویژوال بیسیک را وادار کرد که اجرای برنامه را از روالی بنام Sub Main شروع کند. در تمرین زیر خواهید دید که چگونه می‌توان فرم شروع برنامه (Startup form) را مشخص کرد.

شروع کردن برنامه با فرم Form2

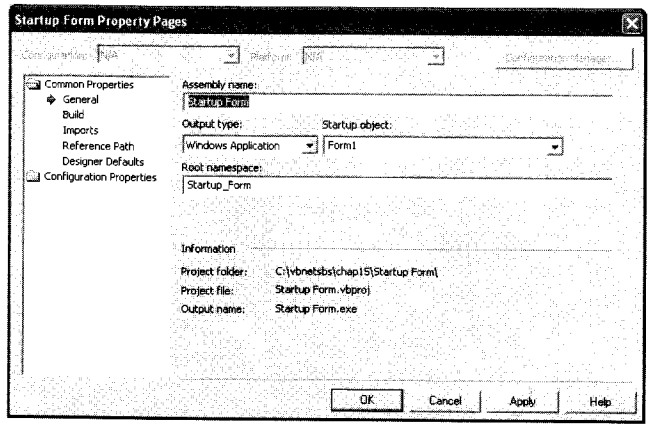
۱ با فرمان File|Close Solution پروژهٔ قبلی را بسته، و یک پروژهٔ جدید Visual Basic Windows Application بنام **My Startup Form** در پوشهٔ c:\vbnetsbs\chap15 ایجاد کنید.

۲ فرم اول پروژه (Form1.vb) را باز کنید.

۳ با فرمان Project|Add Windows Form یک فرم جدید به پروژه اضافه کنید.

۴ در کاوشگر راه‌حل، روی فرم دوم (Form2.vb) کلیک و آنرا باز کنید.

۵ بعد از انتخاب آیکون پروژهٔ My Startup Form در کاوشگر راه‌حل، با فرمان Project|Project Properties دیالوگ خواص پروژه را باز کنید:



دیالوگ خواص پروژه به شما اجازه می‌دهد تا فرم شروع برنامه را از لیست Startup Object انتخاب کنید.

۶ لیست Startup Object را باز کرده، و روی Form1 کلیک کنید. با این کار ویژوال بیسیک در شروع برنامه بجای Form1 فرم دوم (Form2) را باز خواهد کرد.

۷ با کلیک کردن OK ، دیالوگ خواص پروژه را ببندید.

۸ برنامه را اجرا کنید؛ همانطور که می‌بینید، ابتدا فرم Form2 روی صفحه ظاهر می‌شود.

۹ با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

در تمرین بعدی یک روال بنام Sub Main را بعنوان شیء شروع برنامه ست خواهیم کرد.

شروع برنامه با روال Sub Main

۱ با انتخاب فرمان Project|Add New Item و سپس انتخاب الگوی Module ، یک ماژول به پروژهٔ My Startup Form اضافه کنید (نام این ماژول Module1.vb خواهد بود).

۲ در کاوشگر راه‌حل، روی ماژول Module1.vb کلیک کرده و سپس دکمهٔ View Code را کلیک کنید، تا این ماژول در ادیتور کُد باز شود.

۳ دستورات زیر را بین Module و End Module بنویسید:

```
Public MyForm1 As New Form1()
Public MyForm2 As New Form2()
Public Sub Main()
    MsgBox("This is Sub Main")
    'place additional program code here
    MyForm1.ShowDialog()
End Sub
```

روال Sub Main باید در یک ماژول استاندارد و بصورت Public تعریف شده باشد، تا بتوان آنرا بعنوان شیء شروع برنامه ست کرد؛ این روال هیچ آرگومانی هم نباید بگیرد. روال فوق (بعد از ایجاد متغیرهای وهله برای فرمهای Form1 و Form2) و نمایش یک پیام، فرم اول را با متد ShowDialog نمایش می‌دهد.

٤ بعد از انتخاب آیکون پروژهٔ My Startup Form در کاوشگر راه‌حل، با فرمان Project|Project Properties دیالوگ خواص پروژه را باز کنید.

٥ لیست Startup Object را باز کرده، و روی Sub Main کلیک کنید. با این کار ویژوال بیسیک در شروع برنامه روال Sub Main را اجرا خواهد کرد.

٦ با کلیک کردن OK ، دیالوگ خواص پروژه را ببندید.

٧ برنامه را اجرا کنید؛ روال Sub Main اجرا شده و در اولین گام پیام زیر ظاهر می‌شود:

٨ OK را کلیک کنید تا این پیام بسته شده، و فرم Form1 ظاهر شود.

٩ با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

نکته

اگر می‌خواهید برنامه‌ای بنویسید که هیچ فرمی نداشته باشد، بهتر است از برنامهٔ کنسول (Console Application) استفاده کنید. این نوع از برنامه‌ها ورودی خود را از خط فرمان (command line) گرفته، و خروجی خود را نیز به همانجا می‌فرستند (درست شبیه برنامه‌های DOS). (برای کسب اطلاعات بیشتر دربارهٔ برنامه‌های کنسول، به سیستم کمک ویژوال بیسیک مراجعه کنید.)

مرجع سریع فصل ۱۵

برای ...	انجام دهید
اضافه کردن یک فرم جدید به برنامه	فرمان Project|Add Windows Form را اجرا کرده، و Open را کلیک کنید.
ایجاد یک فرم جدید از طریق کُد	از دستور Dim و کلمهٔ کلیدی New استفاده کنید: `Dim form2 As New Form()`
نمایش فرم روی صفحه	از متد ShowDialog یا Show استفاده کنید: `form2.ShowDialog()`
تعیین مکان فرم روی میزکار ویندوز	خاصیت StartPosition را به CenterScreen یا CenterParent ست کنید.

تعیین مکان فرم روی صفحه از طریق کُد	خاصیت StartPosition را به Manual ست کرده، و سپس با استفاده از خاصیت DesktopBounds ابعاد و محل دقیق فرم را ست کنید:
	`form2.StartPosition = FormStartPosition.Manual`
	`Dim Form2Rect As New Rectangle(200, 100, 300, 250)`
	`form2.DesktopBounds = Form2Rect`

حداقل یا حداکثر کردن فرم از طریق کُد	ابتدا خواص MinimizeBox و MaximizeBox را True کرده، و سپس کُد برنامه خاصیت WindowState را به WindowState.Minimized یا WindowState.Maximized ست کنید.

اضافه کردن کنترل به فرم در زمان اجرای برنامه	کنترل موردنظر را ایجاد کرده، و سپس آنرا به کلکسیون کنترلهای فرم اضافه کنید:
	`Dim button1 As New Button()`
	`button1.Text = "Click Me"`
	`button1.Location = New Point(20, 25)`
	`form2.Controls.Add(button1)`

ثابت نگه داشتن فاصلهٔ یک کنترل از لبه‌های فرم	خاصیت Anchor را به مقدار مناسب ست کنید (برای ترکیب چند حالت، از عملگر Or استفاده کنید):
	`button1.Anchor = AnchorStyles.Bottom Or _`
	`AnchorStyles.Right`

چسباندن یک کنترل به لبه‌های فرم	خاصیت Dock را به مقدار مناسب ست کنید: `PictureBox1.Dock = DockStyle.Top`
تعیین شیء شروع برنامه	بعد از انتخاب پروژه در کاوشگر راه‌حل، و اجرای فرمان Project|Project Properties ، شیء موردنظر را از لیست Startup Object انتخاب کنید.

فصل

۱۶

گرافیک و انیمیشن

در این فصل یاد می گیرید چگونه :

- ✔ برای اضافه کردن گرافیک به فرمهای برنامه، از فضای نام System.Drawing استفاده کنید.
- ✔ جلوه‌های گرافیکی متحرک و انیمیشن ایجاد کنید.
- ✔ اشیاء را در زمان اجرای برنامه منقبض و منبسط کنید.
- ✔ میزان شفافیت فرم را تغییر دهید.

برای بسیاری از برنامه‌نویسان قسمتهای هنری و گرافیکی برنامه جالبترین و مهیج‌ترین بخش آنست، و خوشبختانه در ویژوال بیسیک .NET این کار از جمله ساده‌ترین کارها نیز هست. در این فصل یاد خواهید گرفت چگونه به برنامه‌های خود رنگ و لعاب بدهید، و آنها را چشم‌نواز کنید.

انیمیشن یکی دیگر از جنبه‌های جالب برنامه‌نویسی است، که در این فصل با آن نیز آشنا می‌شوید، و یاد می‌گیرید چگونه از یک تایمر برای ایجاد انیمیشن‌های ساده استفاده کنید. مهارتهایی که در این فصل بدست می‌آورید، از جمله عناصر اساسی برای ایجاد برنامه‌های جذاب و حرفه‌ای محسوب می‌شوند.

تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- در ویژوال بیسیک ۶ برای ایجاد شکلهای ساده‌ای مانند خط، دایره، و مستطیل می‌توانستید کنترلهای Line و Shape را بکار ببرید؛ در جعبه ابزار ویژوال بیسیک .NET دیگر خبری از این کنترلهای گرافیکی نیست، و بجای آن باید مستقیماً از سرویسهای گرافیکی GDI+ (از طریق فضای نام System.Drawing) استفاده کنید.
- در ویژوال بیسیک .NET متدهای DrawEllipse و DrawLine بترتیب جای دستورات Circle و Line را گرفته‌اند، و بجای دستور PSet نیز باید از ساختار Point (که جزء کلاس System.Drawing.Graphics است) استفاده کنید.
- واحد اندازه‌گیری پیش‌فرض در ویژوال بیسیک .NET (بجای twips) پیکسل (pixel) است.
- اکثر کنترلهای ویژوال بیسیک ۶ متدی داشتند بنام Move ، که برای حرکت دادن کنترلها روی فرم می‌توانستید از آن استفاده کنید. کنترلهای ویژوال بیسیک .NET دیگر متد Move ندارند، اما برای حرکت دادن کنترلها می‌توانید از خواص Left ، Top ، Location ، و یا متد SetBounds کمک بگیرید.
- ویژوال بیسیک .NET همچنان از تکنیک کشیدن‌رها کردن (Drag-and-Drop) پشتیبانی می‌کند، و کنترلها رویدادی بنام DragDrop دارند، اما خواص DragIcon و DragMode دیگر وجود ندارند.
- فرمتهای گرافیکی که ویژوال استودیو .NET پشتیبانی می‌کند، بسیار متنوعتر از ویژوال استودیو ۶ است. در فضای نام System.Drawing.Imaging توابع متنوعی برای کار با فرمتهای BMP ، EMF ، EXIF ، GIF ، Icon ، JPEG ، MemoryBMP ، PNG ، TIFF ، و WMF وجود دارد.

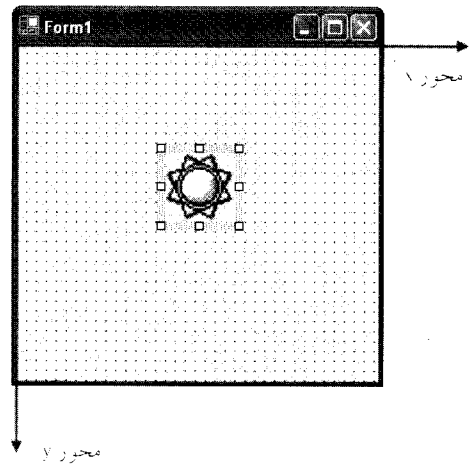
اضافه کردن گرافیک با استفاده از فضای نام System.Drawing

در بخشهای قبل با اضافه کردن تصاویر بیت‌مَپ و آیکون از طریق کنترل جعبه تصویر آشنا شدید، و دیدید که این کار چقدر ساده است. در این فصل خواهید دید که چگونه می‌توان با استفاده از توابع GDI+ (که جزء فضای نام System.Drawing است) جلوه‌های گرافیکی (از قبیل رنگ، شکل، و تصویر) به فرمهای خود اضافه کنید.

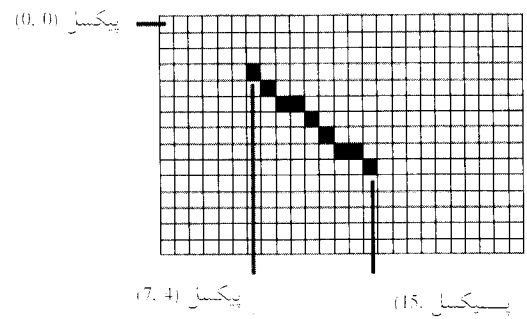
سیستم مختصات فرم

اولین قدم در خلق کارهای گرافیکی، آشنا شدن با سیستم مختصات (coordinate system) فرمهاست. در ویژوال بیسیک، هر فرم دارای یک سیستم (یا دستگاه) مختصات است، که مبدأ آن در گوشهٔ چپ-بالای فرم قرار دارد. واحد اندازه‌گیری این سیستم نیز پیکسل است (پیکسل عبارتست از کوچکترین واحد روی صفحه، که می‌توان آن را مستقلاً ست کرد).

برای کار با یک پیکسل روی فرم، باید مختصات افقی و عمودی (x و y) آن را مشخص کنیم. مختصات مبدأ همیشه (0, 0) است. در شکل زیر دستگاه مختصات یک فرم ویژوال بیسیک، و مختصات یک کنترل روی آن، را مشاهده می‌کنید:



ویژوال بیسیک برای تعیین مکان یک پیکسل روی فرم (و ترسیم اشکال گرافیکی مانند خط، دایره و مستطیل) با نرم‌افزار درایور کارت ویدئویی کامپیوتر کار می‌کند. توجه کنید که این عملیات از حوزهٔ اختیار شما (بعنوان برنامه‌نویس) خارج است، و کلاً بر عهدهٔ توابع کتابخانهٔ گرافیکی +GDI قرار دارد. در شکل زیر نمای نزدیک خط راستی را که ویژوال بیسیک بین دو نقطهٔ (7, 4) و (15, 10) کشیده، ملاحظه می‌کنید (این خط وقتی از فاصلهٔ مناسب دیده شود، کاملاً راست بنظر خواهد رسید).



کلاس System.Drawing.Graphics

فضای نام System.Drawing کلاسهای متعددی دارد، که برای کارهای گرافیکی می‌توان از آنها استفاده کرد. در این قسمت با امکانات یکی از این کلاسها بنام System.Drawing.Graphics آشنا خواهید شد. (برای آشنایی با کلاسهای دیگر به سیستم کمک ویژوال بیسیک .NET مراجعه کنید.)

شکلهای سادهٔ هندسی نقش مهمی در عملیات گرافیکی (خواه ساده باشد، یا پیچیده) دارند. در جدول زیر تعدادی از مهمترین شکل‌های هندسی (و متدهای گرافیکی متناظر با آن در کلاس System.Drawing.Graphics) را ملاحظه می‌کنید:

شکل هندسی	متد گرافیکی	توضیح
خط	DrawLine	یک خط راست بین دو نقطه رسم می‌کند
چهارضلعی	DrawRectangle	یک مستطیل یا مربع بین چهار نقطه رسم می‌کند
قوس	DrawArc	یک قوس (قسمتی از بیضی) بین دو نقطه رسم می‌کند
دایره/بیضی	DrawEllipse	یک دایره یا بیضی (محصور در یک چهارضلعی) رسم می‌کند
چندضلعی	DrawPolygon	یک چندضلعی نامنظم (بین چند نقطه) رسم می‌کند
منحنی	DrawCurve	یک منحنی (که از چند نقطه عبور می‌کند) رسم می‌کند
منحنی بزییر	DrawBezier	یک منحنی که از چهار نقطه عبور می‌کند (و نقاط ۲ و ۳ نقاط کنترل هستند) رسم می‌کند

متدهای جدول فوق همگی شکلهایی توخالی رسم می‌کنند. برای رسم شکلهای توپُر می‌توانید از متدهای متناظر، که با پیشوند "Fill" شروع می‌شوند (مانند FillRectangle ، FillEllipse و FillPolygon)، استفاده کنید.

قبل از اینکه بتوانید با متدهای کلاس System.Drawing.Graphics شکلی را رسم کنید، ابتدا باید یک شیء Graphics ایجاد کرده، و یک قلم (Pen) یا قلم‌مو (Brush) - برای تعیین مشخصات ظاهری شکل، از قبیل رنگ پُر کردن، ضخامت قلم و غیره - نیز داشته باشید. اگر می‌خواهید یک شکل توخالی رسم کنید، باید یک شیء قلم را بعنوان آرگومان به متد موردنظر ارسال کنید، و اگر شکل توپُر می‌خواهید، یک شیء قلم‌مو. در مثال زیر، ابتدا یک شیء Graphics (بنام GraphicsFun) و یک قلم قرمز (بنام PenColor) ایجاد کرده، و سپس خط راستی بین دو نقطهٔ (20, 30) و (100, 80) رسم کرده‌ایم:

```
Dim GraphicsFun As System.Drawing.Graphics
Dim PenColor As New System.Drawing.Pen(System.Drawing.Color.Red)
GraphicsFun = Me.CreateGraphics
GraphicsFun.DrawLine(PenColor, 20, 30, 100, 80)
```

دستورات قبل از متد DrawLine نیز از اهمیت یکسانی برخوردار هستند، چون بدون آنها نمی‌توان از این متد استفاده کرد. (برای استفاده از متدهای گرافیکی کلاس System.Drawing.Graphics نیازی نیست دستور Imports System.Drawing.Graphics را صریحاً قید کنید؛ این کلاس بطور خودکار به تمام فرمها اضافه می‌شود.)

رویداد Paint فرم

اگر کُد فوق را اجرا کرده باشید، متوجه شده‌اید که این خط فقط تا زمانی دیده می‌شود (دوام می‌آورد) که چیزی روی آن را نپوشاند؛ اگر یک پنجره یا دیالوگ را روی این خط برده و سپس کنار ببرید، خط محو خواهد شد (حداقل کردن و دوباره باز کردن فرم هم همان بلا را سر آن می‌آورد)! برای غلبه بر این مشکل، باید کُد فوق را در رویداد Paint فرم قرار دهید، تا با هر تغییری مجدداً اجرا شود و خط را دوباره رسم کند.

در تمرین زیر با استفاده از همین تکنیک سه شکل هندسی روی فرم برنامه رسم خواهیم کرد.

رسم خط، مستطیل، و بیضی

۱ در محیط ویژوال استودیو .NET ، یک پروژهٔ جدید Visual Basic Windows Application بنام **My Draw Shapes** در پوشهٔ c:\vbnetsbs\chap16 ایجاد کنید.

۲ فرم برنامه را بزرگتر کنید، چون برای رسم شکلها به فضای بیشتری نیاز داریم.

۳ خاصیت Text فرم Form1.vb را به **Draw Shapes** ست کنید.

٤ با کلیک کردن دکمهٔ View Code در کاوشگر راه‌حل، ادیتور کُد را باز کنید.

۵ از لیست Class Name آیتم Base Class Events را انتخاب کنید. تمام رویدادهای یک فرم در لیست Base Class Events قرار دارند.

۶ از لیست Method Name رویداد Paint را انتخاب کنید.

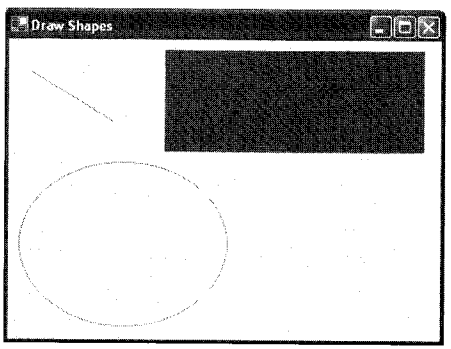
۷ روال رویداد Form1_Paint در ادیتور کُد باز می‌شود (هر گاه فرم برنامه به هر دلیلی تغییر کند، کُد این روال اجرا خواهد شد).

۸ دستورات زیر را در این روال بنویسید:

```
'Prepare GraphicsFun variable for graphics calls
Dim GraphicsFun As System.Drawing.Graphics
GraphicsFun = Me.CreateGraphics

'Use a red pen color to draw a line and an ellipse
Dim PenColor As New System.Drawing.Pen(System.Drawing.Color.Red)
GraphicsFun.DrawLine(PenColor, 20, 30, 100, 80)
GraphicsFun.DrawEllipse(PenColor, 10, 120, 200, 160)

'Use a green brush color to create a filled rectangle
Dim BrushColor As New SolidBrush(Color.Green)
GraphicsFun.FillRectangle(BrushColor, 150, 10, 250, 100)
```

این کُد ساده سه شکل هندسی روی فرم رسم می‌کند: یک خط راست قرمز، یک بیضی قرمز، و یک مستطیل توپُر سبز. همانطور که قبلاً دیدید، ابتدا یک شیء Graphics بنام GraphicsFun تعریف کرده، و با متد CreateGraphics آنرا ایجاد کرده‌ایم. برای رسم شکلهای توخالی (خط و بیضی) به یک قلم، و برای رسم شکل توپُر (مستطیل) نیز به یک قلم‌مو نیاز داریم؛ متغیرهای PenColor و BrushColor بترتیب قلم و قلم‌موی ما هستند.

۹ برنامه را اجرا کنید (کُد کامل این برنامه را می‌توانید در پوشهٔ c:\vbnetsbs\chap16\draw shapes بیابید). در تصویر زیر نتیجهٔ اجرای برنامه، و رسم شکلها در روال Form1_Paint را ملاحظه می‌کنید:



۱۰ فرم را حداقل (Minimize) کنید، و دوباره به حالت اول برگردانید. با این کار رویداد Paint تحریک شده، و با اجرای مجدد روال Form1_Paint شکلها از نو روی فرم رسم می‌شوند.

۱۱ برنامه را ببندید.

اضافه کردن انیمیشن به برنامه

نمایش تصویر و رسم شکلهای هندسی تکنیکهای جالبی هستند، ولی برای یک برنامه‌نویس هیچ چیز جذابیت انیمیشن را ندارد. انیمیشن عبارتست از ایجاد توهم حرکت با استفاده از نمایش متوالی و سریع تصاویر ثابت. در این تکنیک هر تصویر تفاوتی جزیی (از نظر شکل یا موقعیت) با تصویر قبلی دارد، و توالی سریع آنها منجر به ایجاد تصور حرکت در بیننده می‌شود.

در این قسمت با جابجا کردن یک تصویر روی فرم در فواصل زمانی کوتاه (به کمک یک تایمر)، یک انیمیشن ساده خلق خواهیم کرد.

حرکت دادن اشیاء روی فرم

در ویژوال بیسیک ۶ برای حرکت دادن کنترلها روی فرم متد خاصی وجود داشت بنام Move . ویژوال بیسیک .NET دیگر از متد Move پشتیبانی نمی‌کند، اما بجای آن می‌توانید از خواص و متدهای زیر استفاده کنید:

متد	توضیح
Left	این خاصیت باعث حرکت افقی شیء (به چپ یا راست) می‌شود
Top	این خاصیت باعث حرکت عمودی شیء (به بالا یا پائین) می‌شود
Location	این خاصیت باعث منتقل شدن شیء به یک نقطهٔ خاص می‌شود
SetBounds	با این خاصیت می‌توان چارچوب و موقعیت (اندازه و مکان) شیء را همزمان ست کرد

با خواص Left و Top می‌توانید یک شیء را بصورت افقی یا عمودی جابجا کنید. شکل استفاده از این خواص چنین است:

```
object.Left = horizontal
object.Top = vertical
```

که در آن *object* نام شیء موردنظر، و *horizontal* و *vertical* بترتیب فاصلهٔ افقی و عمودی شیء از لبه‌های فرم هستند. خواص Left و Top را می‌توان بصورت مطلق یا نسبی ست کرد. با دستورات زیر، جعبه تصویر PictureBox1 در نقطه‌ای به فاصلهٔ ۳۰۰ پیکسل از لبهٔ چپ فرم، و ۱۵۰ پیکسل از لبهٔ بالای آن (زیر میلهٔ عنوان) قرار داده شده است (در اینجا از مختصات مطلق استفاده کرده‌ایم):

```
PictureBox1.Left = 300
PictureBox1.Top = 150
```

برای جابجا کردن همین شیء نسبت به نقطه‌ای که اکنون در آن قرار دارد، باید از روش مختصات نسبی استفاده کنیم:

```
PictureBox1.Left = PictureBox1.Left + 50
PictureBox1.Top = PictureBox1.Top + 30
```

این دستورات جعبه تصویر PictureBox1 را ۵۰ پیکسل به راست و ۳۰ پیکسل به پائین (نسبت به محل فعلی آن) می‌برند. (برای منتقل کردن شیء به سمت چپ و بالا، باید مقدار موردنظر را از خواص Left و Top کم کنید.)

خاصیت Location

دیدید که گاهی برای جابجا کردن یک شیء لازم است دو خاصیت Left و Top را همزمان ست کنیم؛ اما طراحان ویژوال بیسیک .NET راه ساده‌تری برای این منظور تعبیه کرده‌اند، و آن استفاده از خاصیت Location است. شکل کلی این خاصیت چنین است:

```
object.Location = New Point(horizontal, vertical)
```

که در آن *object* نام شیء موردنظر، و *horizontal* و *vertical* بترتیب فاصلهٔ افقی و عمودی شیء از لبه‌های فرم هستند (ساختار Point مختصات گوشهٔ چپ-بالای شیء را در خود نگه می‌دارد). در هنگام کار با خاصیت Location هم می‌توانید از مختصات مطلق و نسبی استفاده کنید. دستور زیر، جعبه تصویر PictureBox1 را به نقطه‌ای با مختصات مطلق (300, 200) - نسبت به لبه‌های فرم - می‌برد:

```
PictureBox1.Location = New Point(300, 200)
```

و دستور زیر همین شیء را ۵۰ پیکسل به چپ و ۴۰ پیکسل به بالا (نسبت به مکان فعلی آن) منتقل می‌کند:

```
PictureBox1.Location = New Point(PictureBox1.Location.X - 50, _
                                 PictureBox1.Location.Y - 40)
```


ایجاد انیمیشن با استفاده از تایمر

برای به حرکت در آوردن یک شیء در برنامه، می‌توانیم خاصیت Location آنرا در فواصل زمانی منظم و متوالی (که توسط یک تایمر اندازه‌گیری می‌شود) تغییر دهیم. در فصل ۷ با کنترل تایمر (Timer) و طرز کار آن آشنا شدید، و یک ساعت دیجیتال با آن نوشتید. فاصلهٔ زمانی تایمری که برای انیمیشن بکار می‌رود، باید خیلی کمتر - در حدود ۱۰۰ یا ۲۰۰ میلی‌ثانیه - باشد (البته مقدار دقیق آن به نوع انیمیشنی که می‌خواهید بسازید، بستگی دارد).

در تمرین زیر یک آیکون بنام Sun را با استفاده از تکنیک فوق به حرکت درمی‌آوریم. در این تمرین، با استفاده از خاصیت Top جعبه تصویری که آیکون در آن بار شده، و خاصیت Size.Height فرم، رسیدن آیکون به لبه‌های فرم را تشخیص داده و جهت حرکت آنرا عوض می‌کنیم.

انیمیشن آیکون Sun

۱ پروژهٔ قبلی را با فرمان File|Close Solution بسته، و یک پروژهٔ جدید Visual Basic Windows Application بنام **My Moving Icon** در پوشهٔ c:\vbnetsbs\chap16 ایجاد کنید.

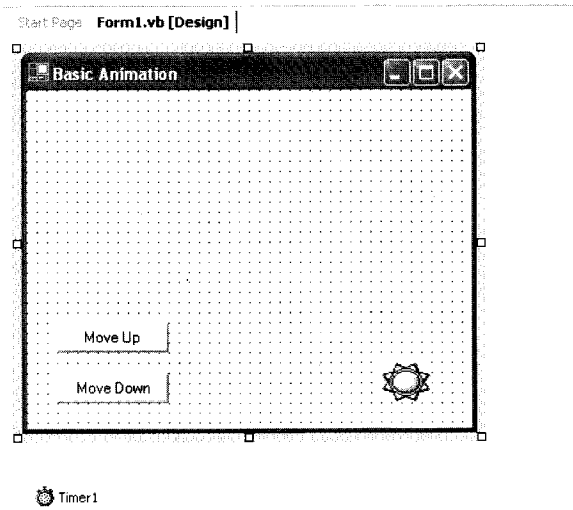
۲ دو دکمه در گوشهٔ چپ-پائین فرم رسم کنید.

۳ یک جعبه تصویر کوچک در گوشهٔ راست-پائین فرم رسم کنید.

٤ در برگهٔ Windows Forms جعبه ابزار ویژوال بیسیک، روی کنترل Timer دو-کلیک کنید، تا یک تایمر به سینی اجزاء برنامه اضافه شود (توجه کنید که تایمر جزء کنترلهای نامریی است، و روی فرم دیده نخواهد شد).

۵ خواص فرم برنامه و کنترلهای روی آنرا با توجه به جدول زیر ست کنید (شکل زیر را ببینید):

شیء	خاصیت	مقدار
Form1	Text	"Basic Animation"
Button1	Text	"Move Up"
Button2	Text	"Move Down"
PictureBox1	Image	"c:\vbnetsbs\chap16\sun.ico"
	SizeMode	StretchImage
Timer1	Enabled	False
	Interval	75



۶ روی دکمهٔ Move Up دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد باز شود.

۷ دستورات زیر را در این روال بنویسید:

```
GoingUp = True
Timer1.Enabled = True
```

این کُد فقط متغیر GoingUp را به True ست کرده، و تایمر Timer1 را فعال می‌کند (حرکت واقعی آیکون در روال رویداد Timer1_Tick انجام خواهد شد).

۸ به بالای پنجرهٔ کُد فرم رفته، و متغیر زیر را در آنجا تعریف کنید:

```
Dim GoingUp As Boolean      'GoingUp stores current direction
```

متغیر GoingUp یک متغیر منطقی است که فقط دو مقدار True یا False می‌گیرد، و جهت حرکت آیکون - رو به بالا، یا رو به پائین - را مشخص می‌کند.

۹ به فرم برنامه برگردید و روی دکمهٔ Move Down دو-کلیک کنید، تا روال رویداد Button2_Click در ادیتور کُد باز شود. دستورات زیر را در این روال بنویسید:

```
GoingUp = False
Timer1.Enabled = True
```

این روتین بسیار شبیه Button1_Click است، و فقط جهت حرکت آیکون را عوض می‌کند.

۱۰ دوباره به فرم برنامه برگردید و روی تایمر Timer1 دو-کلیک کنید، تا روال رویداد Timer1_Tick در ادیتور کُد باز شود. دستورات زیر را در این روال بنویسید:

```
If GoingUp = True Then
    'move picture box toward the top
    If PictureBox1.Top > 10 Then
        PictureBox1.Location = New Point(PictureBox1.Location.X - 10, _
            PictureBox1.Location.Y - 10)
    End If
Else
    'move picture box toward the bottom
    If PictureBox1.Top < (Me.Size.Height - 75) Then
        PictureBox1.Location = New Point(PictureBox1.Location.X + 10, _
            PictureBox1.Location.Y + 10)
    End If
End If
```

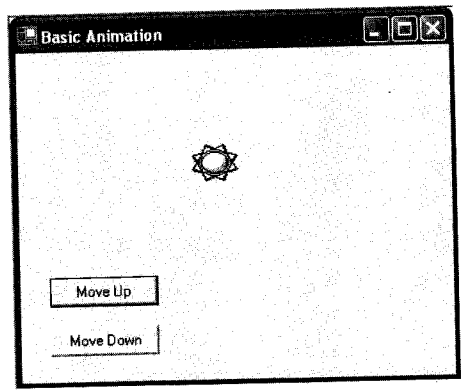
مادامیکه تایمر فعال است، کُد ساختار If...Then هر ۷۵ میلی‌ثانیه اجرا می‌شود. در شرط اول دستور If مقدار متغیر GoingUp چک شده، و بر اساس آن جهت حرکت مشخص می‌شود. اگر این متغیر True باشد، جعبه تصویر ۱۰ پیکسل به چپ و ۱۰ پیکسل به بالا رانده می‌شود. اما اگر متغیر GoingUp مقدار False داشته باشد، بخش Else اجرا شده و جهت حرکت تصویر به سمت پائین و راست خواهد بود (هر بار ۱۰ پیکسل). اگر می‌خواهید تصویر سریعتر حرکت کند، مقدار خاصیت Interval تایمر را کم کنید؛ و اگر می‌خواهید حرکت سریعتر شود، این خاصیت را بیشتر کنید.

برای تشخیص لبه‌های فرم از خاصیت Me.Size.Height استفاده کرده‌ایم (و کم کردن ۷۵ پیکسل از آن برای اینست که تصویر همیشه دیده شود). در اینجا کلمهٔ کلیدی Me به فرم Form1 اشاره می‌کند (چون این کُد در یکی از رویدادهای فرم اجرا می‌شود).

اجرای برنامهٔ Moving Icon

۱ برنامه را اجرا کنید (متن کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap16\moving icon بیابید).

۲ دکمهٔ Move Up را کلیک کنید. با این کار تصویر آیکون بصورت قطری به سمت بالای فرم حرکت می‌کند (شکل زیر را ببینید). بعد از چند ثانیه آیکون به لبهٔ بالایی فرم (زیر میلهٔ عنوان) می‌رسد، و متوقف می‌شود.



۳ دکمهٔ Move Down را کلیک کنید. با این کار آیکون به سمت گوشهٔ راست-پائین فرم به حرکت درمی‌آید.

٤ چند بار دیگر دکمه‌های Move Up و Move Down را کلیک کنید، تا آیکون به حرکت در آید. توجه کنید که برای تغییر دادن جهت حرکت آیکون نیازی نیست منتظر پایان مسیر آن شوید، و می‌توانید بین راه هم جهت حرکت آنرا (با کلیک کردن دکمهٔ مخالف) عوض کنید. این ویژگی بخصوص در هنگام نوشتن بازیهای ویدئویی بسیار مفید است، و می‌توانید به کمک آن حرکت اشیاء و شخصیتهای بازی را براحتی کنترل کنید.

۵ با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

ایجاد انیمیشن با منقبض و منبسط کردن تصاویر

اغلب اشیاء ویژوال بیسیک، علاوه بر خواص Top و Left ، دارای دو خاصیت دیگر بنامهای Height و Width نیز هستند. با استفادهٔ هوشمندانه از این دو خاصیت برای منقبض و منبسط کردن اشیاء، نیز می‌توان انیمیشن‌های جالبی خلق کرد.

منبسط کردن جعبه تصویر

۱ پروژهٔ قبلی را با فرمان File|Close Solution ببندید.

۲ یک پروژهٔ جدید Visual Basic Windows Application بنام **My Zoom In** در پوشهٔ c:\vbnetsbs\chap16 ایجاد کنید.

۳ یک جعبه تصویر کوچک در نزدیکی گوشهٔ چپ-بالای فرم رسم کنید.

٤ خواص فرم برنامه و جعبه تصویر را با توجه به جدول زیر ست کنید:

شیء	خاصیت	مقدار
Form1	Text	"Approaching Earth"
PictureBox1	Image	"c:\vbnetsbs\chap16\earth.ico"
	SizeMode	StretchImage

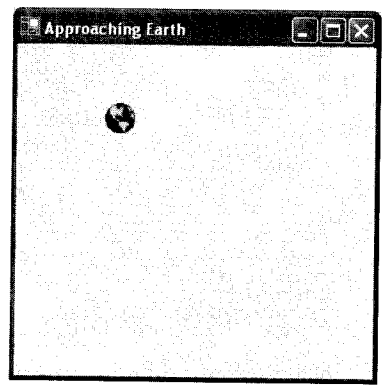
۵ روی جعبه تصویر PictureBox1 دو-کلیک کنید، تا روال رویداد PictureBox1_Click در ادیتور کُد باز شود.

۶ دستورات زیر را در این روال بنویسید:

```
PictureBox1.Height = PictureBox1.Height + 15
PictureBox1.Width = PictureBox1.Width + 15
```

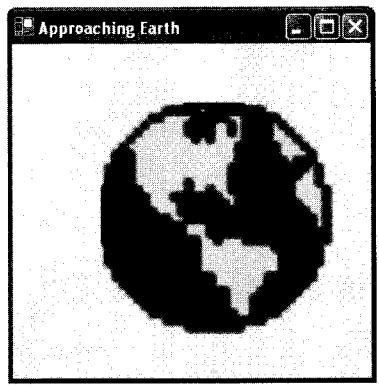
این دو دستور باعث می‌شوند تا با هر کلیک روی جعبه تصویر PictureBox1 پهنا و ارتفاع آن باندازهٔ ۱۵ پیکسل بزرگ شود (که با این کار خود آیکون هم بزرگ خواهد شد). این انیمیشن (با بکار انداختن کمی تخیل) تصور نزدیک شدن به زمین از دید یک سفینهٔ فضایی را بوجود می‌آورد.

۷ برنامه را اجرا کنید (متن کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap16\zoom in بیابید). تصویر اولیهٔ فرم (و آیکون کرهٔ زمین) را در شکل زیر می‌بینید:



۸ چند بار روی تصویر کرهٔ زمین کلیک کنید.

در زیر آیکون Earth.ico را بعد از ۱۰ یا ۱۱ بار کلیک شدن، ملاحظه می‌کنید:



۹ بعد از اینکه باندازهٔ کافی به زمین نزدیک شدید و در مدار قرار گرفتید، برنامه را ببندید.

یک گام فراتر: تغییر دادن شفافیت فرم برنامه

فکر می‌کنید چیزهای جالب تمام شده؟ نه! کتابخانهٔ GDI+ کارهایی می‌تواند بکند که در ویرایشهای قبلی ویژوال بیسیک حتی خوابش را هم نمی‌توانستید ببینید. برای مثال، می‌توان فرمها را بگونه‌ای شفاف کرد، که پشت آنها قابل دیدن باشد! این ویژگی به چه دردی می‌خورد؟ فقط تصور کنید می‌خواهید برنامه‌ای بنویسید که کاربر بتواند عکسها و تصاویر خود را با آن ببیند و آنها را دستکاری کند (چیزی شبیه فتوشاپ). اگر برای انتخاب (و دستکاری) تصاویر یک فرم اصلی داشته باشید، و تصویرها را در فرم دیگری باز کنید، فرم اصلی می‌تواند شفاف باشد تا تصویر از پشت آن دیده شود.

در تمرین زیر خواهید دید که چگونه می‌توان با استفاده از خاصیت Opacity میزان شفافیت فرمها را کنترل کرد.

ست کردن خاصیت Opacity

۱ با فرمان File|Close Solution پروژهٔ قبلی را ببندید.

۲ یک پروژهٔ جدید Visual Basic Windows Application بنام **My Transparent Form** در پوشهٔ c:\vbnetsbs\chap16 ایجاد کنید.

۳ فرم پروژه را باز کرده، و دو دکمه روی آن رسم کنید.

٤ خواص فرم برنامه و دکمه‌ها را با توجه به جدول زیر ست کنید:

شیء	خاصیت	مقدار
Form1	Text	"Transparent Form"
Button1	Text	"Set Opacity"
Button2	Text	"Restore"

۵ روی دکمهٔ Set Opacity دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد باز شود.

۶ دستور زیر را در این روال بنویسید:

```
Me.Opacity = 0.75
```

خاصیت Opacity مقداری بین 0 (بیشترین شفافیت) تا 1 (کمترین شفافیت - مقدار پیش‌فرض) می‌گیرد. دستور فوق شفافیت فرم Form1 را به میزان ۷۵٪ ست می‌کند.

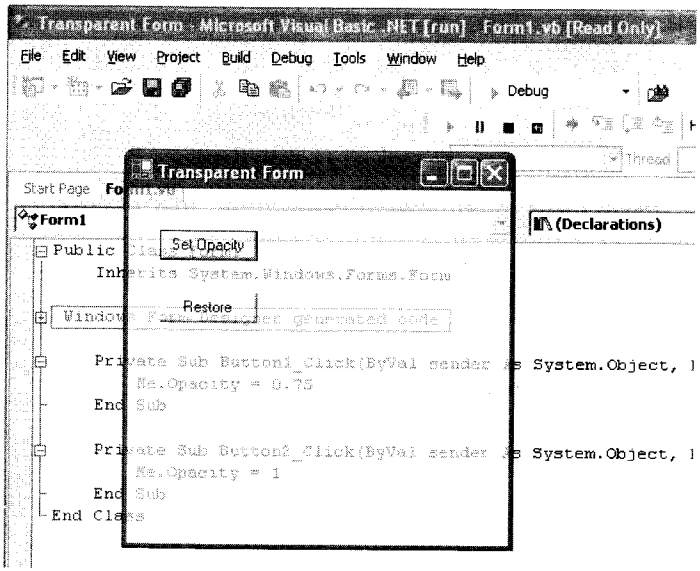
۷ به فرم برنامه برگردید، و این بار روی دکمهٔ Restore دو-کلیک کنید، تا روال رویداد Button2_Click در ادیتور کُد باز شود. دستور زیر را در این روال بنویسید:

```
Me.Opacity = 1
```

این دستور شفافیت فرم را به حالت ۱۰۰٪ (مقدار پیش‌فرض) برمی‌گرداند.

۸ برنامه را اجرا کنید.

۹ روی دکمهٔ Set Opacity کلیک کنید. توجه کنید که چگونه فرم حالت شفاف به خود می‌گیرد، و می‌توانید پشت آنرا ببینید:



۱۰ دکمهٔ Restore را کلیک کنید، تا فرم به حالت قبل برگردد.

۱۱ کمی با برنامه کار کنید و سپس، با کلیک کردن دکمهٔ Close ، آنرا ببندید.

مرجع سریع فصل ۱۶

برای ...	انجام دهید

ترسیم خط و شکلهای هندسی روی فرم	از متدهای کلاس System.Drawing.Graphics استفاده کنید. در مثال زیر نحوهٔ رسم یک بیضی را ملاحظه می‌کنید:
	`Dim GraphicsFun As System.Drawing.Graphics`
	`GraphicsFun = Me.CreateGraphics`
	`Dim PenColor As New System.Drawing.Pen _`
	`(System.Drawing.Color.Red)`
	`GraphicsFun.DrawEllipse(PenColor, 10, 120, 200, 160)`

ترسیم خطوط و شکلهایی که پاک نمی‌شوند، و بادوام هستند	متدهای گرافیکی را در روال رویداد Paint فرم قرار دهید.
جابجا کردن یک شیء روی فرم	با استفاده از خاصیت Location شیء را به نقطهٔ دلخواه منتقل کنید: *object*.Location = New Point(300, 200)
ایجاد انیمیشن حرکتی	در روال رویداد Tick یک تایمر، خواص Top ، Left یا Location شیء موردنظر را تغییر دهید. برای کنترل سرعت انیمیشن از خاصیت Interval تایمر استفاده کنید.
ایجاد انیمیشن انقباضی-انبساطی	در روال رویداد Tick یک تایمر، خواص Height یا Width شیء موردنظر را تغییر دهید. (برای کنترل سرعت انیمیشن از خاصیت Interval تایمر استفاده کنید.)
تغییر شفافیت فرم	خاصیت Opacity فرم را به مقدار دلخواه ست کنید.

فصل

۱۷

وراثت فرم و ایجاد کلاس‌های پایه

در این فصل یاد می گیرید چگونه :

- ✔ با استفاده از Inheritance Picker فرمهای موجود در ویژوال بیسیک را به پروژهٔ خود اضافه کنید.
- ✔ یک کلاس پایه (base class) با متدها و خواص موردنظر ایجاد کنید.
- ✔ با استفاده از دستور Inherits، از یک کلاس پایه کلاسهای جدیدی مشتق کنید.

امروزه برنامه‌نویسی شیءگرا (Object Oriented Programming - OOP) یکی از متداولترین اصطلاحات بین برنامه‌نویسان است. از ویرایش ۴ به بعد میکروسافت برخی از امکانات برنامه‌نویسی شیءگرا را به ویژوال بیسیک اضافه کرد، ولی این زبان همچنان از نظر افراد خبره یک زبان شیءگرای واقعی محسوب نمی‌شد، چون از وراثت (Inheritance – مکانیزمی که به یک کلاس اجازه می‌دهد تا رفتارها و ویژگیهای کلاسهای موجود را به ارث ببرد) پشتیبانی نمی‌کرد – و اکنون بعد از مدتها انتظار، ویژوال بیسیک .NET از وراثت پشتیبانی می‌کند. این بدان معناست که شما می‌توانید فرمهایی ایجاد کنید که ویژگیها و عملکردهای خود را در اختیار فرمهای دیگر می‌گذارند. علاوه بر آن، در ویژوال بیسیک .NET می‌توان کلاسهای پایه نیز ایجاد کرد. در این فصل با هر دو شکل وراثت (از فرمهای موجود، و از کلاسهای پایه) آشنا خواهید شد. برای وراثت از فرمهای موجود از Inheritance Picker (یکی از ابزارهای جدید ویژوال استودیو .NET) استفاده خواهیم کرد؛ نوع دوم وراثت را نیز با استفاده از دستور Inherits انجام خواهیم داد. ویژوال بیسیک محبوبترین زبان برنامه‌نویسی دنیاست، و اینک با این ویژگیها و امکانات جدید، ویژوال بیسیک .NET تبدیل به یکی از قویترین زبانهای برنامه‌نویسی نیز شده است.

تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- با ابزار جدید Inheritance Picker می‌توانید از ویژگیها و رفتارهای فرمهای موجود در فرمهای جدید بهره ببرید (و در واقع، این ویژگیها را به ارث ببرید).
- برای ایجاد کلاسهای جدید در ویژوال بیسیک .NET ، باید از ساختار Public Class...End Class استفاده کنید.
- در ویژوال بیسیک ۶ هر کلاس بایستی در یک فایل جداگانه ذخیره می‌شد؛ در ویژوال بیسیک .NET می‌توان چندین کلاس را در یک فایل ذخیره کرد.
- روش اضافه کردن خواص جدید به یک کلاس در ویژوال بیسیک .NET فرق کرده است؛ ویژوال بیسیک .NET دیگر از ساختارهای Property Get ، Property Let ، و Property Set پشتیبانی نمی‌کند.
- در ویژوال بیسیک .NET برای مشتق کردن یک کلاس از کلاسهای موجود از دستور Inherits استفاده می‌شود.

وراثت فرم با استفاده از Inheritance Picker

در زبانهای شیءگرا، وراثت (Inheritance) عبارتست از دریافت خواص، متدها و سایر ویژگیهای یک کلاس موجود توسط کلاس جدید. همانطور که در فصل ۱۵ گفتم، فرمهای ویژوال بیسیک همگی از کلاس System.Windows.Forms.Form مشتق می‌شوند (و بعبارت دیگر، از آن ارث می‌برند). در واقع، هر بار که با فرمان Project|Add Windows Form فرم جدیدی به برنامه اضافه می‌کنید، دستور زیر به بالای کُد فرم اضافه می‌شود:

```
Inherits System.Windows.Forms.Form
```

که حاکی از مشتق شدن آن فرم از کلاس System.Windows.Forms.Form است. (پس تا اینجا، بدون اینکه خودتان بفهمید، بارها از وراثت استفاده کرده‌اید.)

با اینکه وراثت را می‌توان بسادگی از طریق کُد برنامه انجام داد، اما طراحان ویژوال استودیو .NET برای راحتی کار شما ابزار جدیدی بنام Inheritance Picker ایجاد کرده‌اند، که مشتق کردن فرمها را بسیار ساده کرده است. در تمرین زیر طرز استفاده از این ابزار را خواهید دید.

۱ در محیط ویژوال استودیو .NET ، یک پروژهٔ جدید Visual Basic Windows Application بنام **My Form Inheritance** در پوشهٔ c:\vbnetsbs\chap17 ایجاد کنید.

۲ در پائین فرم برنامه دو دکمه (کنار هم) رسم کنید.

۳ خاصیت Text دکمه‌های Button1 و Button2 را بترتیب به **OK** و **Cancel** ست کنید.

٤ روی دکمهٔ OK دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد باز شود.

۵ دستور زیر را در این روال بنویسید:

```
MsgBox("You clicked OK")
```

۶ به فرم برنامه برگردید، و روی دکمهٔ Cancel دو-کلیک کنید، تا روال رویداد Button2_Click در ادیتور کُد باز شود. دستور زیر را در این روال بنویسید:

```
MsgBox("You clicked Cancel")
```

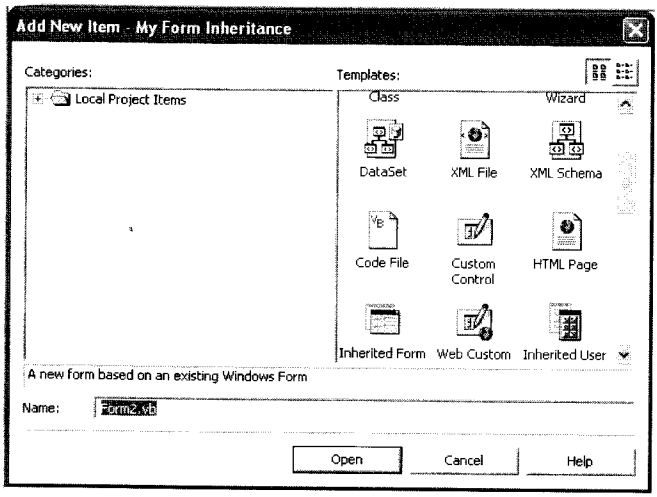
۷ به فرم برنامه برگردید، و خاصیت Text آنرا به **Dialog Box** تغییر دهید.

اکنون یک فرم ساده دارید، که می‌توان (با کمی تغییر) از آن بعنوان یک دیالوگ در برنامه‌ها استفاده کرد (فقط کافیست در هر برنامه کنترلهای مناسب را به آن اضافه کنید).

اولین قدم برای به ارث بردن از این فرم، کامپایل کردن آن است؛ چون برای وراثت از یک فرم، باید بصورت EXE. یا DLL. در آمده باشد. هر بار که این فرم پایه را تغییر داده و مجدداً کامپایل کنید، این تغییرات به فرمهایی که از آن ارث برده‌اند، نیز منتقل خواهد شد.

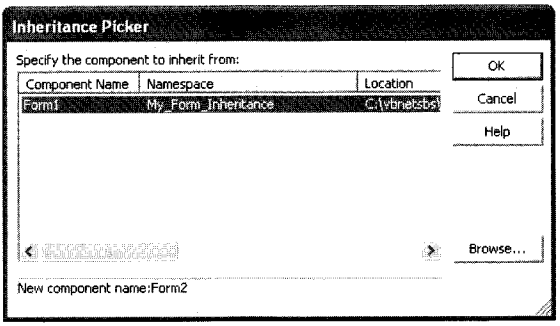
۸ فرمان Build Solution را از منوی Build انتخاب کنید، تا ویژوال بیسیک فرم را کامپایل کرده، و فایل EXE. آنرا بسازد.

۹ فرمان Add Inherited Form را از منوی Project انتخاب کنید، تا پنجرهٔ زیر باز شود:



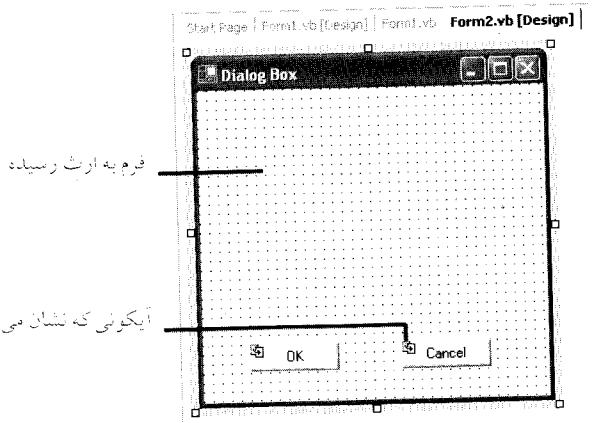
همانطور که می‌بینید، در اینجا تمام انواع فرمهای موجود در ویژوال استودیو لیست شده‌اند، اما چون در قسمت Templates آیتم Inherited Form بصورت پیش‌فرض انتخاب شده، لازم نیست چیزی را تغییر دهید. (در فیلد Name نیز می‌توانید نام این فرم جدید را تغییر دهید.)

۱۰ با قبول انتخابهای پیش‌فرض، دکمهٔ Open را کلیک کنید، تا دیالوگ Inheritance Picker باز شود:



در این دیالوگ تمام فرمهایی که می‌توان در پروژهٔ فعلی از آنها ارث برد، دیده می‌شود. با دکمهٔ Browse می‌توانید فرمهای دیگری را که روی دیسک وجود دارند، برای وراثت انتخاب کنید. (اگر فرمی که می‌خواهید از آن ارث ببرید، در پروژهٔ فعلی نیست، باید بصورت یک فایل DLL. کامپایل شده باشد.)

۱۱ در دیالوگ Inheritance Picker ، فرم Form1 را انتخاب کرده، و OK را کلیک کنید. با این کار، ویژوال استودیو فرم Form2 را به پروژه اضافه می‌کند. توجه کنید که این فرم کاملاً شبیه فرم Form1 است، با این تفاوت که در بالای دکمه‌های آن یک آیکون کوچک دیده می‌شود؛ این آیکون نشان می‌دهد که این اشیاء از جای دیگری به ارث رسیده‌اند:



ظاهر فرم به ارث رسیده بسیار شبیه فرم پایه است، و آیکونهای وراثت هم کوچک هستند و ممکنست براحتی دیده نشوند، بنابراین لازمست برای تمایز بین آنها دقت کافی بخرج دهید.

حال وقت آنست که عناصر جدیدی به این فرم به ارث رسیده اضافه کنیم.

تکمیل فرم به ارث رسیده

۱ یک دکمهٔ جدید روی فرم Form2 (فرم به ارث رسیده) رسم کنید.

۲ خاصیت Text این دکمه را به **Click Me!** ست کنید.

۳ روی دکمهٔ Click Me! دو-کلیک کنید.

٤ دستور زیر را در روال رویداد Button3_Click بنویسید:

```
MsgBox("This is the inherited form!")
```

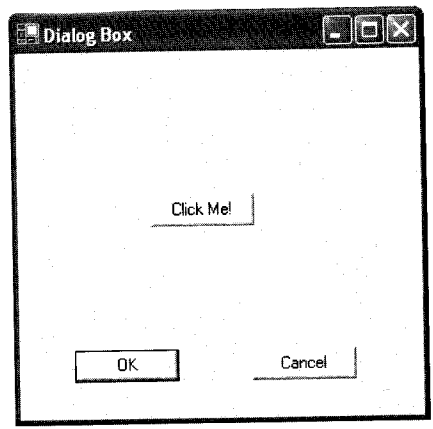
۵ به فرم Form2 برگردید، و روی دکمهٔ OK یا Cancel دو-کلیک کنید. همانطور که می‌بینید، دستکاری کُد اشیاء به ارث رسیده به سادگی امکانپذیر نیست (و از حوصلهٔ این کتاب خارج است)، اما می‌توانید اشیاء جدیدی به این فرم اضافه کنید.

۶ هر یک از خواص این فرم را می‌توانید مستقلاً ست کنید، مثلاً می‌توانید اندازهٔ آنرا بزرگتر کنید. اما توجه کنید که اگر برای ست کردن خواص فرم Form2 از پنجرهٔ خواص استفاده کنید، خواص فرم پایه را در آن خواهید دید. اجازه دهید فرم Form2 را بعنوان فرم شروع برنامه انتخاب کنیم.

۷ در کاوشگر راه‌حل، روی آیکون پروژهٔ My Form Inheritance کلیک کرده، و سپس فرمان Project|Properties را انتخاب کنید، تا دیالوگ خواص پروژه باز شود.

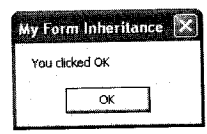
۸ لیست Startup Object را باز کنید؛ فرم Form2 را انتخاب کرده، و سپس OK را کلیک کنید.

۹ برنامه را اجرا کنید:



۱۰ دکمهٔ OK را کلیک کنید.

همانطور که می‌بینید، این فرم روال رویدادی را که از فرم Form1 به ارث برده، اجرا می‌کند:



۱۱ این پنجره را بسته، و سپس روی دکمهٔ Click Me! کلیک کنید، تا پیام نوشته شده برای آنرا ببینید. بله، فرم Form2 علاوه بر کُدهای خود، کُد فرمی را که از آن ارث برده، نیز اجرا می‌کند!

۱۲ این پنجره را هم بسته، و سپس با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

کلاس‌های پایه ایجاد کنید

همانطور که دیدید، ابزار Inheritance Picker فرآیند وراثت را کنترل کرده، و کلاس جدید را به پروژه اضافه می‌کند. برای این کار، Inheritance Picker ارتباطی بین کلاس پایه و کلاس جدید برقرار می‌کند. برای مثال، کُد وراثت Form2 (که از Form1 مشتق شده) را در زیر می‌بینید:

```
Public Class Form2
    Inherits My_Form_Inheritance.Form1
.
.
    Private Sub Button3_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles Button3.Click
            MsgBox("This is the inherited form!")
    End Sub
End Class
```

علاوه بر دستور Inherits در بالای کُد فرم Form2 ، روال رویداد Button3_Click نیز عضو کلاس جدید است. اما بیاد دارید که Form1 خود از کلاس System.Windows.Forms.Form مشتق شده و از آن ارث می‌برد. این نکته نشان می‌دهد که یک کلاس جدید می‌تواند از کلاسهایی ارث ببرد که خود از کلاسهای بالاتر مشتق شده‌اند.

با توجه به نقش اساسی و محوری کلاس‌ها در برنامه‌های ویژوال بیسیک .NET ، شاید از خود بپرسید یک کلاس پایه چگونه ایجاد می‌شود، و چگونه می‌توان کلاسهای جدید را از آن مشتق کرد. بدلیل اهمیت این موضوع، بقیهٔ این فصل را به نحوهٔ ایجاد کلاسهای پایه، و انشقاق کلاسهای جدید از آنها اختصاص داده‌ام. تکنیکهایی که در این بخش یاد می‌گیرید، جزء مهمترین مهارتهای برنامه‌نویسی با چارچوب .NET محسوب می‌شوند.

هشدار

وراثت و ایجاد کلاسهای جدید جزء موضوعاتیست که همچنان بر سر اصطلاحات آن اختلاف وجود دارد. تعدادی از برجسته‌ترین دانشمندان این رشته سالهاست به بحث در این زمینه ادامه می‌دهند، و این موضوع آن چنان وسیع است که پرداختن به آن از حوصلهٔ کتاب حاضر خارج است. شما هم لازم نیست زیاد وارد این قبیل جزئیات شوید؛ فقط مطالب کتاب را با دقت دنبال کنید، تا ببینید که ایجاد کلاسهای جدید با ویژوال بیسیک .NET چقدر ساده است.

کلاس‌های تعریف شده توسط کاربر (user-defined classes) به شما اجازه می‌دهند تا در یک برنامه اشیاء اختصاصی ایجاد کنید - اشیایی که مانند کنترلهای جعبه ابزار ویژوال بیسیک دارای خواص، متدها و رویدادهای خاص خود هستند. ایجاد کلاسهای جدید در ویژوال بیسیک .NET به کاری بسیار ساده و مفرّح تبدیل شده است.

در تمرین زیر برنامه‌ای می‌نویسیم که نام، نام خانوادگی و تاریخ تولد یک کارمند جدید را دریافت می‌کند. این اطلاعات در خواص یک کلاس جدید بنام Person ذخیره شده، و با متدی بنام Age سن این کارمند محاسبه می‌شود. در این تمرین با نحوهٔ ایجاد کلاسهای جدید، و بکارگیری آنها آشنا خواهید شد.

ایجاد پروژهٔ Person Class

۱ پروژهٔ قبلی را با فرمان File|Close Solution بسته، و یک پروژهٔ جدید Visual Basic Windows Application بنام **My Person Class** در پوشهٔ c:\vbnetsbs\chap17 ایجاد کنید.

۲ یک برچسب پهن (تقریباً به پهنای فرم) در بالای فرم برنامه رسم کنید.

۳ دو جعبه متن پهن (تقریباً به همان پهنای برچسب) زیر برچسب رسم کنید.

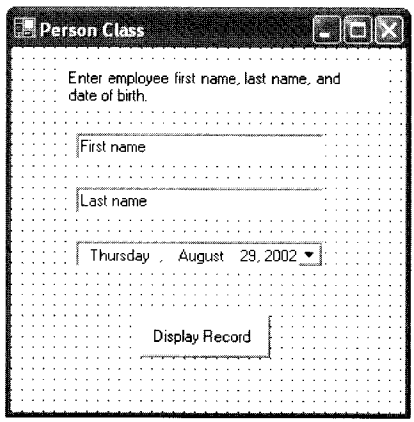
٤ یک کنترل تاریخ-وقت-گزین (DateTimePicker) زیر جعبه متن‌ها قرار دهید. (برای آشنایی بیشتر با کنترل تاریخ-وقت-گزین به فصل ۳ مراجعه کنید.)

۵ یک دکمه زیر کنترل تاریخ-وقت-گزین رسم کنید.

۶ خواص فرم برنامه و کنترلهای روی آنرا با توجه به جدول زیر ست کنید:

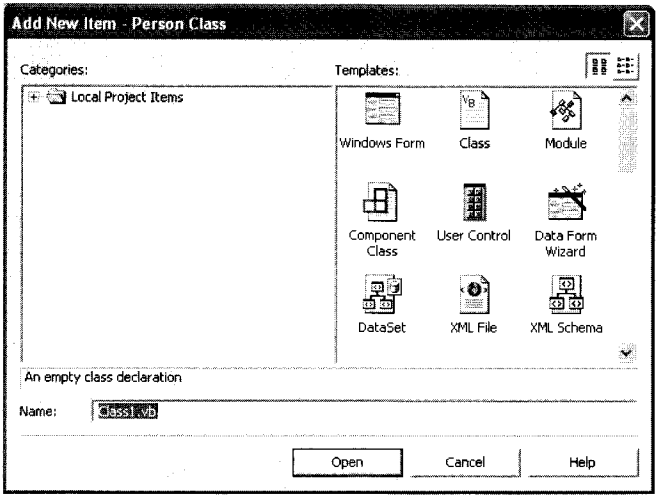
شیء	خاصیت	مقدار
Form1	Text	"Person Class"
Button1	Text	"Display Record"
Label1	Text	"Enter employee first name, last name, and date of birth"
TextBox1	Text	"First name"
TextBox2	Text	"Last name"

۷ فرم برنامه را در شکل زیر ملاحظه می‌کنید:



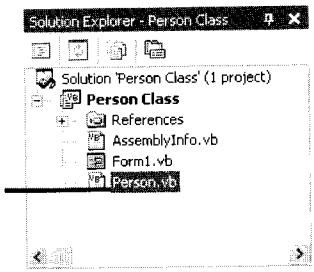
(از آنجائیکه این فرم به پایگاه داده متصل نیست، فقط یک رکورد را می‌توان در آن ذخیره کرد.) پس از آن نوبت به اضافه کردن کلاس به پروژه است.

۸ فرمان Project|Add Class را اجرا کنید، تا دیالوگ «اضافه کردن آیتم جدید» ظاهر شود:



در این دیالوگ می‌توانید نام دلخواه را به کلاس جدید بدهید. بیاد داشته باشید که در یک ماژول کلاس بیش از یک کلاس می‌توانید داشته باشید، بنابراین سعی کنید نام آن را کلی انتخاب کنید.

۹ در فیلد Name نام **Person.vb** را وارد کرده، و دکمهٔ Open را کلیک کنید. با این کار، ویژوال بیسیک یک ماژول کلاس خالی در ادیتور کُد باز کرده، و نام آنرا به لیست فایلهای پروژه هم (در کاوشگر راه‌حل) اضافه می‌کند:



تعریف کلاس جدید در ماژول کلاس شامل سه مرحله است: تعریف متغیرهای کلاس، ایجاد خواص، ایجاد متدها.

تعریف متغیرهای کلاس

۱ زیر دستور Public Class Person ، متغیرهای زیر را تعریف کنید:

```
Private Name1 As String
Private Name2 As String
```

در اینجا دو متغیر بنامهای Name1 و Name2 تعریف کرده‌ایم، که اطلاعات را در خود ذخیره می‌کنند - و از آنجائیکه این اطلاعات مختص کلاس Person هستند، آنها را بصورت Private تعریف کرده‌ایم (بعبارت دیگر، این متغیرها از خارج کلاس قابل دسترسی نیستند).

ایجاد خواص کلاس

۱ زیر دستورات تعریف متغیرهای کلاس، دستور زیر را وارد کنید:

```
Public Property FirstName() As String
```

این دستور یک خاصیت از نوع String در کلاس Person ایجاد می‌کند، که نام آن FirstName است. به محض زدن Enter ، ویژوال استودیو دو بلوک کُد در زیر آن اضافه می‌کند: یک بلوک Get (برای خواندن مقدار فعلی خاصیت FirstName)، و یک بلوک Set (برای ست کردن مقدار خاصیت FirstName)؛ و البته یک End Property در آخر آن.

توجه

در ویژوال بیسیک ۶ برای خواندن خواص از بلوک Property Get ، و برای ست کردن آنها از بلوکهای Property Set و Property Let استفاده می‌کردیم. ویژوال بیسیک .NET دیگر از این دستورات پشتیبانی نمی‌کند.

۲ بلوکهای Get و Set را مانند زیر پُر کنید (دستوراتی که شما باید بنویسید، با فونت ضخیم مشخص شده‌اند):

```
Public Property FirstName() As String
    Get
        Return Name1
    End Get
    Set(ByVal Value As String)
        Name1 = Value
    End Set
End Property
```

کلمهٔ کلیدی Return مشخص می‌کند که وقتی کاربر خاصیت FirstName را می‌خواند، چه مقداری باید برگردانده شود. در بلوک Set هم مقدار Value (مقداری که کاربر به خاصیت FirstName می‌دهد) در متغیر Name1 نوشته می‌شود. با اینکه کُد فوق کمی پیچیده بنظر می‌رسد، اما در واقع این ساده‌ترین روش خواندن و ست کردن خواص یک کلاس است؛ کلاسهای دیگر ممکنست قبل از ذخیره کردن مقدار ورودی آن را چک کنند، و یا حتی روی آن پردازشهایی انجام دهند.

۳ زیر End Property خاصیت FirstName ، خاصیت دوم یعنی LastName را تعریف می‌کنیم:

```
Public Property LastName() As String
```

```
    Get
        Return Name2
    End Get
    Set(ByVal Value As String)
        Name2 = Value
    End Set
End Property
```

این خاصیت بسیار شبیه اولیست، با این تفاوت که از متغیر دوم (Name2) استفاده می‌کند.

خواص کلاس Person را ایجاد کردیم؛ اکنون نوبت به متد آن (یعنی Age) می‌رسد -این متد سن کارمند را بر اساس تاریخ تولد وی محاسبه می‌کند.

ایجاد متد کلاس

۱ زیر End Property خاصیت LastName ، تابع زیر را وارد کنید:

```
Public Function Age(ByVal Birthday As Date) As Integer
    Return Int(Now.Subtract(Birthday).Days / 365.25)
End Function
```

متد در واقع یک تابع یا سابروتین است، که عمل خاصی انجام می‌دهد. متد Age یک آرگومان از نوع Date (تاریخ تولد کارمند جدید) می‌گیرد، آنرا از تاریخ فعلی (Now) کم کرده، و سپس بر عدد 365.25 (تعداد تقریبی روزهای سال) تقسیم می‌کند تا سن کارمند بدست آید. از آنجائیکه فقط به قسمت صحیح سن کارمند نیاز داریم، عدد بدست آمده را با متد Int به عدد صحیح تبدیل می‌کنیم. توجه کنید که چگونه برای برگرداندن این عدد از تابع Age ، از کلمهٔ کلیدی Return استفاده کرده‌ایم.

کار ایجاد کلاس Person به پایان رسید! اکنون اجازه دهید از این کلاس در رویدادهای Form1 استفاده کنیم.

نکته

یکی از مهمترین کارهایی که در هر کلاس باید انجام دهید، چک کردن مقدار ورودی خواص و متدهای آنست (کاری که ما در اینجا انجام ندادیم). این کار باعث می‌شود تا احتمال وقوع خطا در برنامه به مقدار زیادی کاسته شود.

ایجاد یک شیء از کلاس جدید

۱ فرم Form1.vb را در کاوشگر راه‌حل انتخاب کرده، و دکمهٔ View Designer را کلیک کنید.

۲ روی دکمهٔ Display Record دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد باز شود.

۳ دستورات زیر را در این روال بنویسید:

```
Dim Employee As New Person()
```

```
Dim DOB As Date

Employee.FirstName = TextBox1.Text
Employee.LastName = TextBox2.Text
DOB = DateTimePicker1.Value.Date

MsgBox(Employee.FirstName & " " & Employee.LastName _
    & " is " & Employee.Age(DOB) & " years old.")
```

این روتین ابتدا با کلمهٔ کلیدی New یک شیء جدید بنام Employee از کلاس Person می‌سازد. تا اینجا متغیرهایی زیادی در این کتاب تعریف کردیم، اما این اولین باریست که متغیر ما از نوعیست که خودمان آنرا ساخته‌ایم! در دستورات بعدی مقدار موجود در جعبه متن‌های TextBox1 و TextBox2 را بترتیب در خاصیتهای FirstName ، و LastName ذخیره کرده، و سپس با متد Age سن کارمند را بر اساس تاریخ انتخاب شده در کنترل DateTimePicker1 محاسبه کرده‌ایم. در پایان هم، همهٔ این اطلاعات را در یک MsgBox نمایش داده‌ایم. می‌بینید طرز کار با خواص و متدهای شیء جدید Person چقدر ساده است!

٤ برنامه را اجرا کنید. (کُد کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap17\person class ببینید.)

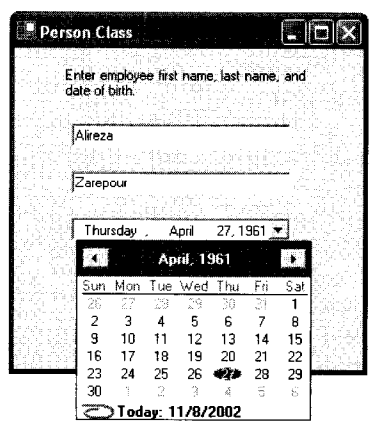
٥ نام و نام خانوادگی خود را بترتیب در جعبه متن‌های FirstName و LastName وارد کنید.

٦ لیست کنترل تاریخ-وقت-گزین را باز کرده، و تاریخ تولد خود را انتخاب کنید.

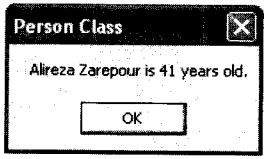
نکته

برای حرکت سریع در کنترل تاریخ-وقت-گزین، آن را باز کرده، و از دکمه‌های ◀ و ◀ استفاده کنید. به کمک این دکمه‌ها می‌توانید بسرعت سال و ماه موردنظر را یافته، و سپس روز را انتخاب کنید.

به شکل زیر نگاه کنید:



۷ روی دکمهٔ Display Record کلیک کنید. با این کار نام و نام خانوادگی شما در شیء Employee ذخیره شده، و بعد از محاسبهٔ سن، این اطلاعات بصورت یک پیام نمایش داده می‌شود:



۸ OK را کلیک کرده، و چند بار دیگر برنامه را (با اطلاعات متفاوت) امتحان کنید.

۹ در پایان، برنامه را ببندید.

یک گام فراتر: وراثت کلاس پایه

همانطور که از فرمها می‌توان ارث برد، از کلاس Person هم می‌توان ارث برد. برای این منظور باید از دستور Inherits استفاده کرده، و کلاس پایه را به کلاس جدید اضافه کنیم. برای متمایز کردن کلاس جدید از کلاس پایه هم می‌توانیم خواص و متدهای جدیدی به آن اضافه کنیم.

در تمرین زیر یک کلاس جدید بنام Teacher به پروژهٔ My Person Class اضافه خواهیم کرد. این کلاس خواص FirstName و LastName ، و متد Age را از کلاس Person ارث می‌برد، و خود خاصیت جدیدی دارد بنام Grade ، که پایهٔ تحصیلی معلم جدید را در خود ذخیره می‌کند.

استفاده از کلمهٔ کلیدی Inherits

۱ کلاس Person.vb را در کاوشگر راه‌حل انتخاب کرده، و دکمهٔ View Code را کلیک کنید.

۲ همانطور که قبلاً هم گفتم، یک ماژول کلاس می‌تواند شامل چندین کلاس باشد، که هر یک از آنها در یک بلوک Public Class...End Class تعریف می‌شوند. برای تعریف کلاس جدید Teacher ، به انتهای کلاس Person (بعد از دستور End Class) رفته، و آنجا کلیک کنید.

۳ دستورات زیر را وارد کنید (شما فقط دستوراتی را که با فونت ضخیم مشخص شده‌اند، بنویسید؛ ویژوال استودیو بقیه دستورات را اضافه خواهد کرد):

```
Public Class Teacher
    Inherits Person
    Private Level As Short

    Public Property Grade() As Short
        Get
            Return Level
        End Get
        Set(ByVal Value As Short)
            Level = Value
        End Set
    End Property
End Class
```

دستور Inherits در ابتدای کلاس Teacher باعث می‌شود تا این کلاس تمام خواص، متدها و متغیرهای کلاس Person را به ارث ببرد. اگر کلاس Person در فایل دیگری قرار داشت، باید فضای نام آنرا هم مشخص می‌کردیم (درست همانطور که هنگام کار با کلاسهای چارچوب .NET عمل می‌کنیم). اکنون اجازه دهید از این کلاس در روال Button1_Click استفاده کنیم.

٤ در ادیتور کُد به روال رویداد Button1_Click بروید.

٥ این روال را مانند زیر تغییر دهید:

```
Dim Employee As New Teacher()
Dim DOB As Date

Employee.FirstName = TextBox1.Text
Employee.LastName = TextBox2.Text
DOB = DateTimePicker1.Value.Date
Employee.Grade = InputBox("What grade do you teach?")

MsgBox(Employee.FirstName & " " & Employee.LastName _
    & " teaches grade " & Employee.Grade)
```

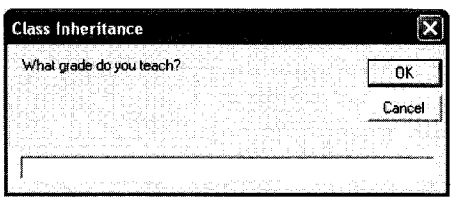
در اینجا فرمول محاسبهٔ سن کارمند را حذف کرده، و بجای آن پایهٔ تحصیلی که وی تدریس می‌کند، را نمایش می‌دهیم.

٦ برنامه را اجرا کنید. (کُد کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap17\class inheritance ببینید.)

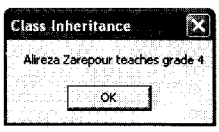
٧ نام و نام خانوادگی خود را بترتیب در جعبه متن‌های FirstName و LastName وارد کنید.

٨ در کنترل تاریخ-وقت-گزین، تاریخ تولد خود را انتخاب کنید.

٩ روی دکمهٔ Display Record کلیک کنید. با این کار نام و نام خانوادگی شما در شیء Employee ذخیره شده، و یک دیالوگ برای گرفتن پایهٔ تحصیلی باز می‌شود:



١٠ عدد 3 را وارد کرده، و OK را کلیک کنید. برنامه این عدد را در خاصیت Grade شیء Employee ذخیره کرده، و سپس این اطلاعات را بصورت یک پیام نمایش می‌دهد:



۱۱ کمی دیگر با برنامه کار کرده، و سپس آنرا ببندید.

در قلمرو برنامه‌نویسی شیءگرا – OOP

اگر با همین مختصر به برنامه‌نویسی شیءگرا علاقمند شده‌اید، ویژوال بیسیک .NET را بسیار مهیج خواهید یافت - ویژوال بیسیک .NET اولین ویرایش این زبان است، که می‌توان آنرا یک زبان برنامه‌نویسی شیءگرای واقعی دانست. با ویژوال بیسیک .NET می‌توانید برای کلاسهای جدید رویداد (event) هم تعریف کنید؛ به خواصی که ایجاد می‌کنید، مقدار پیش‌فرض بدهید؛ و یا از امکانات چندشکلی (polymorphism) آن بهره ببرید. برای کسب اطلاعات بیشتر در این زمینه‌ها می‌توانید به سیستم کمک ویژوال بیسیک، و یا کتابهای پیشرفتهٔ ویژوال بیسیک .NET مراجعه کنید (پیوست ب را ببینید).

مرجع سریع فصل ۱۷

برای ...	انجام دهید
وراثت از یک فرم موجود	فرمان Project|Add Inherited Form را انتخاب کرده، و بعد از وارد کردن نام فرم جدید و کلیک کردن دکمهٔ Open ، با Inheritance Picker فرمی را که می‌خواهید از آن ارث ببرید، انتخاب کنید. (توجه: برای ارث بردن از یک فرم، باید قبلاً آنرا بصورت EXE یا DLL کامپایل کرده باشید. اگر این فرم در پروژهٔ فعلی نیست، باید بصورت DLL کامپایل شده باشد.)
تغییر دادن فرمهای به ارث رسیده	با این فرمها می‌توانید درست مثل فرمهای معمولی کار کنید؛ البته نمی‌توانید خواص اشیاء به ارث رسیدهٔ فرم را ست کنید.
ایجاد یک کلاس پایه	فرمان Project|Add Class را انتخاب کرده، و بعد از وارد کردن نام کلاس جدید و کلیک کردن دکمهٔ Open ، کُد موردنیاز را در ماژول کلاس بنویسید.
تعریف متغیرهای کلاس	برای تعریف متغیرهای خصوصی در کلاس، از کلمهٔ کلیدی Private استفاده کنید: `Private strVar1 As String`

ایجاد یک خاصیت جدید	از دستور Public Property استفاده کنید:
	`Public Property FirstName() As String`
	`  Get`
	`  Return Name1`
	`  End Get`
	`  Set(ByVal Value As String)`
	`  Name1 = Value`
	`  End Set`
	`End Property`

برای ...	انجام دهید

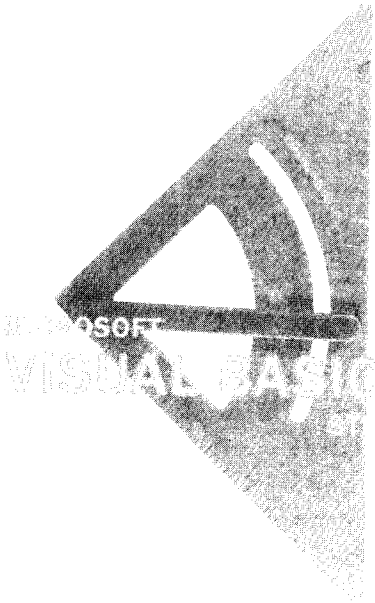
ایجاد یک متد جدید	یک تابع یا سابروتین در ماژول کلاس تعریف کنید:
	`Public Function Age(ByVal Birthday As Date) As Integer`
	`  Return Int(Now.Subtract(Birthday).Days / 365.25)`
	`End Function`

تعریف یک شیء از کلاس جدید	با کلمهٔ کلیدی New متغیری از کلاس موردنظر تعریف کنید: `Dim Employee As New Person()`
ست کردن خواص متغیر شیء	از روش معمولی ست کردن خواص اشیاء استفاده کنید: `Employee.FirstName = TextBox1.Text`

وراثت از یک کلاس پایه	کلاس جدیدی ساخته، و با دستور Inherits کلاس پایه را به آن منضم کنید:
	`Public Class Teacher`
	`  Inherits Person`
	`  Private Level As Short`
	
	`  Public Property Grade() As Short`
	`  Get`
	`  Return Level`
	`  End Get`
	`  Set(ByVal Value As Short)`
	`  Level = Value`
	`  End Set`
	`  End Property`
	`End Class`

فصل

۱۸



چاپ و کار با چاپگرها

در این فصل یاد می گیرید چگونه :

- ✔ در برنامه‌های ویژوال بیسیک، گرافیک چاپ کنید.
- ✔ در برنامه‌های ویژوال بیسیک، متن چاپ کنید.
- ✔ سندهای چندصفحه‌ای را چاپ کنید.
- ✔ از دیالوگهای Print ، Page Setup ، و Print Preview استفاده کنید.

در این فصل توجه خود را به یکی از جنبه‌های پیشرفته در برنامه‌نویسی ویندوز، یعنی چاپ، معطوف خواهیم کرد. ویژوال بیسیک .NET برای چاپ یک کلاس ویژه بنام PrintDocument دارد، که با اشیاء، خواص و متدهای متعدد خود امکانات کاملی در این زمینه ارائه می‌کند. در این فصل خواهید دید که چگونه می‌توان در برنامه‌های ویژوال بیسیک .NET متن و گرافیک چاپ کرد، سندهای چندصفحه‌ای را مدیریت کرد، و با دیالوگهای چاپ (مانند Print ، Page Setup ، و Print Preview) کار کرد. برخی از کُدهایی که در این فصل خواهید دید، مستقیماً در برنامه‌های واقعی قابل استفاده هستند، و در واقع این فصل یکی از عملی‌ترین فصلهای کتاب است.

تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- در ویژوال بیسیک ۶ برای چاپ عمدتاً از خواص و متدهای شیء Printer استفاده می‌کردیم. این وظیفه (به همراه وظایف متعدد دیگر) در ویژوال بیسیک .NET بر عهدهٔ کلاس PrintDocument گذاشته شده است.
- در ویژوال بیسیک ۶ فقط یک دیالوگ اختصاصی، بنام Print ، برای چاپ وجود داشت - که خود جزء دیالوگهای کنترل اکتیوایکس CommonDialog بود؛ در ویژوال بیسیک .NET دیالوگهای متعددی برای عملیات چاپ وجود دارد، از جمله PrintDialog ، PageSetupDialog ، و PrintPreviewDialog.
- برای چاپ یک سند چندصفحه‌ای در ویژوال بیسیک .NET ، باید روال خاصی بنام PrintPage بنویسید و آنرا صفحه به صفحه چاپ کنید. البته پیچیدگیهای این فرآیند با استفاده از توابع فضای نام System.Drawing.Printing تا حد زیادی ساده شده است.

استفاده از کلاس PrintDocument

در بسیاری از برنامه‌های ویندوز امکان چاپ سندهای ایجاد شده وجود دارد، و شاید شما هم بخواهید چنین امکانی را به برنامهٔ خود اضافه کنید. این یکی از زمینه‌هائیست که ویژوال بیسیک .NET پیشرفت قابل‌توجهی نسبت به ویژوال بیسیک ۶ کرده است، و البته هیچ پیشرفتی بدون هزینه نیست: چاپ در ویژوال بیسیک .NET قدری مشکلتر از قبل شده، و نیازمند مقدمات فراوانی است، که به نوع سند و مقدار آن بستگی دارد. برای هر کار چاپی که بخواهید در ویژوال بیسیک .NET انجام دهید، باید کنترلی بنام PrintDocument را به برنامهٔ خود اضافه کنید (و البته این کار هم از طریق کُدنویسی عملی است).

کلاس PrintDocument اشیاء متعدد و مفیدی برای چاپ گرافیک و متن دارد، از جمله شیء PrinterSettings (تنظیمات پیش‌فرض چاپگر)، شیء PageSettings (تنظیمات پیش‌فرض چاپ برای یک صفحهٔ خاص)، و شیء PrintPageEventArgs (اطلاعات صفحهٔ در دست چاپ). کلاس PrintDocument در فضای نام System.Drawing.Printing قرار دارد. با آنکه با اضافه کردن کنترل PrintDocument به فرم برنامه، تعدادی از اشیاء کلاس PrintDocument بطور خودکار به برنامه اضافه می‌شوند، اما برای استفادهٔ کامل از این کلاس باید دستور زیر را در بالای پنجرهٔ کُد فرم وارد کنید:

```
Imports System.Drawing.Printing
```

در تمرین زیر طرز چاپ یک فایل گرافیکی را با استفاده از کلاس PrintDocument خواهید دید.

استفاده از کلاس PrintDocument

۱ در محیط ویژوال استودیو .NET ، یک پروژهٔ جدید Visual Basic Windows Application بنام **My Print Graphics** در پوشهٔ c:\vbnetsbs\chap18 ایجاد کنید.

۲ یک برچسب نسبتاً پهن در بالای فرم برنامه رسم کنید.

۳ زیر برچسب فوق، یک جعبه متن قرار دهید؛ کاربر نام فایل موردنظر را در این جعبه متن وارد می‌کند.

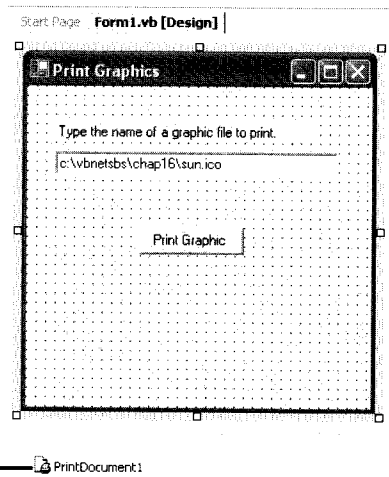
٤ یک دکمه زیر جعبه متن رسم کنید؛ این دکمه کار چاپ را انجام خواهد داد.

۵ در برگهٔ Windows Forms جعبه ابزار ویژوال بیسیک، کنترل PrintDocument را پیدا کرده، و روی آن دو-کلیک کنید. کنترل PrintDocument (مانند تایمر) جزء کنترلهای نامرئی ویژوال بیسیک است، و به سینی اجزاء اضافه خواهد شد.

۶ خواص فرم برنامه و کنترلهای روی آنرا با توجه به جدول زیر ست کنید:

شیء	خاصیت	مقدار
Form1	Text	"Print Graphics"
Button1	Text	"Print Graphic"
Label1	Text	"Type the name of a graphic file to print"
TextBox1	Text	"c:\vbnetsbs\chap16\sun.ico"

فرم برنامه را در شکل زیر ملاحظه می‌کنید:



حال باید کُد لازم برای چاپ فایلهای گرافیکی (بیت‌مپ، آیکون، متافایل، فایلهای JPEG و غیره) را بنویسیم.

۷ روی دکمهٔ Print Graphic دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد باز شود.

۸ دستور زیر را در بالای فرم بنویسید:

```
Imports System.Drawing.Printing
```

با این دستور به اطلاعات شیء PrintPageEventArgs دسترسی خواهید داشت؛ سایر اشیاء کلاس PrintDocument با اضافه شدن کنترل PrintDocument به فرم تعریف خواهند شد.

۹ به روال Button1_Click برگردید، و دستورات زیر را در آن بنویسید:

```
' Print using an error handler to catch problems
Try
    AddHandler PrintDocument1.PrintPage, AddressOf Me.PrintGraphic
    PrintDocument1.Print()      'print graphic
Catch ex As Exception      'catch printing exception
    MessageBox.Show("Sorry--there is a problem printing", ex.ToString())
End Try
```

در اینجا از یک دستور جدید بنام AddHandler استفاده کرده‌ایم؛ این دستور می‌گوید که برای اجرای متد چاپ گرافیک (PrintDocument1.PrintPage) باید از روال PrintGraphic استفاده شود. قبلاً با روالهای کنترل خطا در این کتاب آشنا شده‌اید - روال مجری رویداد (event handler) شباهت زیادی با روال کنترل خطا دارد، که در مورد اتفاقاتی که ذاتاً خطا نیستند، ولی اهمیت اساسی در برنامه دارند، بکار برده می‌شود. در اینجا اتفاق مهم همان چاپ سند است؛ و اطلاعات تکمیلی (مانند سندی که باید چاپ شود، تنظیمات کاغذ و چاپگر) از کلاس PrintDocument می‌آیند. محل این روال (که PrintGraphic نام دارد، بعداً آنرا خواهیم نوشت) نیز توسط عملگر AddressOf مشخص می‌شود.

در خط سوم با متد PrintDocument1.Print() فرمان چاپ را به روال PrintGraphic می‌فرستیم. برای مقابله با هر گونه خطای چاپ (که اتفاقاً بسیار هم پیش می‌آید) کل این کُد را در یک بلوک Try...Catch قرار داده‌ایم. در اینجا از روشی متفاوت با فصل ۹ استفاده کرده‌ایم: متغیر ex (که از نوع Exception است) اطلاعات خطای رُخ داده را برمی‌گرداند.

۱۰ به بالای کُد فرم (زیر دستور "Windows Form Designer generated code") رفته، و سابروتین زیر را در آنجا وارد کنید:

```
'Sub for printing graphic
Private Sub PrintGraphic(ByVal sender As Object, ByVal ev As PrintPageEventArgs)
    ' Create the graphic using DrawImage
    ev.Graphics.DrawImage(Image.FromFile(TextBox1.Text), _
          ev.Graphics.VisibleClipBounds)
    ' Specify that this is the last page to print
    ev.HasMorePages = False
End Sub
```

این روتینی است که رویداد ایجاد شده توسط متد PrintDocument1.Print() را اجرا می‌کند (در اینجا، این سابروتین را در ماژول فرم تعریف کرده‌ایم، اما می‌توانید آنرا در یک ماژول استاندارد نیز

قرار دهید). آرگومان ورودی یعنی ev (که یک شیء PrintPageEventArgs است) تمام اطلاعات لازم را به این سابروتین می‌آورد.

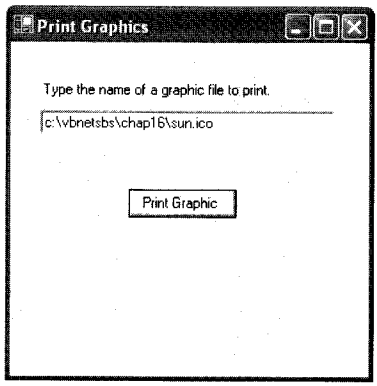
چاپ گرافیک با متد Graphics.DrawImage (که آرگومان اول آن نام فایل موردنظر است) انجام می‌شود. آخرین دستور، یعنی ev.HasMorePages = False ، نیز می‌گوید که این کار چاپی فقط یک صفحه دارد.

۱۱ با کلیک کردن دکمهٔ Save All ، برنامه را ذخیره کنید (کُد کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap18\print graphics بیابید).

قبل از اجرای برنامهٔ Print Graphics ، خوبست که چند فایل گرافیکی را در یک پوشه کپی کرده، و نام و مسیر آنها را یادداشت کنید.

اجرای برنامهٔ Print Graphics

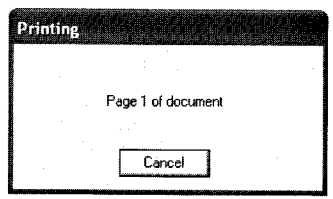
۱ برنامه را اجرا کنید:



۲ چاپگر را روشن کرده و کاغذ در آن قرار دهید، تا آمادهٔ کار (online) شود.

۳ وقتی مطمئن شدید که فایل sun.ico در پوشهٔ c:\vbnetsbs\chap16 وجود دارد، دکمهٔ Print Graphic را کلیک کنید. اگر می‌خواهید فایل دیگری را چاپ کنید، نام و مسیر کامل آنرا وارد کنید.

متد DrawImage تصویر را آنقدر بزرگ می‌کند که تمام صفحهٔ کاغذ را بپوشاند، و سپس آنرا به چاپگر می‌فرستد. (برای آشنایی بیشتر با متد DrawImage و آرگومانهای آن، به سیستم کمک ویژوال بیسیک مراجعه کنید.) اگر دقت کنید، برای چند لحظه پیام زیر را هم خواهید دید:



این دیالوگ هم جزء کلاس PrintDocument است، و به برنامه ظاهری حرفه‌ای می‌دهد.

٤ تصاویر دیگری را با برنامهٔ Print Graphics چاپ کنید.

٥ در پایان، برنامه را ببندید.

خوب، برای شروع کار بد نبود!

چاپ متن

با کنترل PrintDocument آشنا شدید، و دیدید که چگونه می‌توان با آن گرافیک چاپ کرد. اکنون می‌خواهیم از تکنیک مشابهی برای چاپ متن یک جعبه متن استفاده کنیم. در تمرین زیر برای چاپ متن از کلاس PrintDocument استفاده خواهیم کرد، ولی اینکار را از طریق کُد (بدون کنترل PrintDocument) انجام می‌دهیم.

توجه

در برنامهٔ زیر، کل متن چاپ شده کمتر از یک صفحه است. برای چاپ مطالب چندصفحه‌ای به کُد بیشتری نیاز داریم، که در قسمتهای آینده خواهید دید.

استفاده از متد Graphics.DrawString برای چاپ متن

١ پروژهٔ قبلی را با فرمان File|Close Solution بسته، و یک پروژهٔ جدید Visual Basic Windows Application بنام **My Print Text** در پوشهٔ c:\vbnetsbs\chap18 ایجاد کنید.

٢ یک برچسب نسبتاً پهن در بالای فرم برنامه رسم کنید.

٣ زیر برچسب فوق، یک جعبه متن قرار دهید؛ کاربر متنی راکه باید چاپ شود، در این جعبه متن وارد می‌کند.

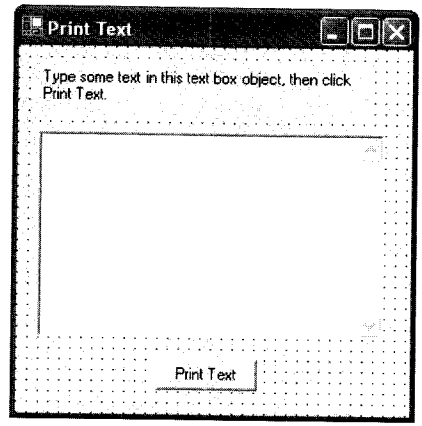
٤ خاصیت Multiline این جعبه متن را True کرده، و آنرا بقدری بزرگ کنید که بتوان چند خط متن در آن نوشت.

٥ یک دکمه زیر جعبه متن رسم کنید.

٦ خواص فرم برنامه و کنترلهای روی آنرا با توجه به جدول زیر ست کنید:

شیء	خاصیت	مقدار
Form1	Text	"Print Text"
Button1	Text	"Print Text"
Label1	Text	"Type some text in this text box object, then click Print Text"
TextBox1	ScrollBars	Vertical
	Text	(خالی)

فرم برنامه را در شکل زیر ملاحظه می‌کنید:



حال باید کُد لازم برای چاپ محتویات این جعبه متن را بنویسیم.

۷ روی دکمهٔ Print Text دو-کلیک کنید، تا روال رویداد Button1_Click در ادیتور کُد باز شود.

۸ دستور زیر را در بالای فرم بنویسید:

```
Imports System.Drawing.Printing
```

با این دستور به اشیاء کلاس PrintDocument دسترسی خواهیم داشت.

۹ به روال Button1_Click برگردید، و کُد زیر را در آن بنویسید:

```
' Print using an error handler to catch problems
Try
    ' Declare PrintDoc variable of type PrintDocument
    Dim PrintDoc As New PrintDocument()
    AddHandler PrintDoc.PrintPage, AddressOf Me.PrintText
    PrintDoc.Print()      'print text
Catch ex As Exception     'catch printing exception
    MessageBox.Show("Sorry--there is a problem printing", ex.ToString())
End Try
```

(تغییرات کُد فوق نسبت به برنامهٔ Print Graphics با فونت ضخیم مشخص شده است.) در اینجا بجای استفاده از کنترل PrintDocument ، با دستور Dim یک شیء بنام PrintDoc از کلاس PrintDocument تعریف کرده‌ایم؛ از اینجا به بعد بجای کنترل PrintDocument از متغیر PrintDoc استفاده خواهیم کرد. (البته نام سابروتین چاپ را هم به PrintText تغییر داده‌ایم.)

۱۰ به بالای فرم (زیر دستور "Windows Form Designer generated code") رفته، و سابروتین زیر را در آنجا وارد کنید:

```
'Sub for printing text
Private Sub PrintText(ByVal sender As Object, ByVal ev As PrintPageEventArgs)
    'Use DrawString to create text in a Graphics object
    ev.Graphics.DrawString(TextBox1.Text, New Font("Arial", 11, _
        FontStyle.Regular), Brushes.Black, 120, 120)
    ' Specify that this is the last page to print
    ev.HasMorePages = False
End Sub
```

این روتینی است که رویداد ایجاد شده توسط متد PrintDoc.Print() را اجرا می‌کند (باز هم تغییرات با فونت ضخیم مشخص شده است). همانطور که می‌بینید، برای چاپ متن از متد جدیدی بنام Graphics.DrawString استفاده کرده‌ایم. این متد به اطلاعات بیشتری نیاز دارد، از جمله نام فونت، نوع، رنگ و اندازهٔ آن، و محل دقیق چاپ متن روی کاغذ. (باز هم مانند قبل، با دستور ev.HasMorePages = False ، اعلام کرده‌ایم که این کار چاپی فقط یک صفحه دارد.)

۱۱ با کلیک کردن دکمهٔ Save All ، برنامه را ذخیره کنید (کُد کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap18\print text بیابید).

اکنون وقت اجرای برنامهٔ Print Text است.

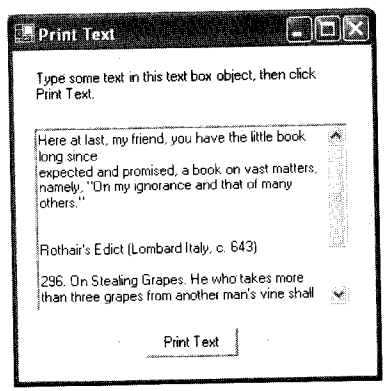
اجرای برنامهٔ Print Text

۱ برنامه را اجرا کنید.

۲ چاپگر را روشن کرده و کاغذ در آن قرار دهید، تا آمادهٔ کار (online) شود.

۳ چند خط متن در جعبه متن بنویسید.

این برنامه از شکستن خودکار خطوط بلند (line wrapping) پشتیبانی نمی‌کند، بنابراین اگر طول خط خیلی زیاد باشد، به احتمال زیاد از حاشیهٔ راست کاغذ بیرون خواهد زد. در شکل زیر نمونه‌ای از متن نوشته شده در جعبه متن TextBox1 را ملاحظه می‌کنید:



٤ دکمهٔ Print Text را کلیک کنید، تا برنامه متن را چاپ کند.

۵ در صورت تمایل، متون دیگری را با برنامهٔ Print Text چاپ کنید.

۶ در پایان هم، برنامه را ببندید.

چاپ فایلهای متنی چندصفحه‌ای

تکنیکهای چاپی که تا اینجا دیدید، برای کارهای ساده مفید هستند، اما چند محدودیت مهم نیز دارند. اول اینکه، شیء PrintDocument برخلاف جعبه متن از **شکستن خودکار خطوط** (line wrapping) پشتیبانی نمی‌کند - بعبارت دیگر، اگر طول خط خیلی زیاد باشد، هنگام چاپ از حاشیهٔ راست کاغذ بیرون خواهد زد. اگر فایل متنی دارید که در پایان هر خط آن کاراکتر سرخط (carriage return) ندارد، باید خودتان طول خطهای چاپی را از طریق کُد برنامه کنترل کنید.

دوم اینکه، برنامهٔ Print Text نمی‌تواند بیش از یک صفحه متن چاپ کند. در واقع، این برنامه حتی نمی‌داند صفحه چیست - متد PrintDoc.Print فقط متن را به چاپگر می‌فرستد؛ اگر مقدار این متن از یک صفحهٔ چاپی بیشتر باشد، بقیهٔ آن اصلاً چاپ نخواهد شد. برای چاپ متنهای بیش از یک صفحه، باید خودتان آنرا به صفحات متعدد تقسیم کرده، و هر صفحه را جداگانه چاپ کنید.

کار سختی است؟ ناامید نشوید! ویژوال بیسیک مکانیزمهایی دارد که در این کار (شکستن خطوط بلند، و تقسیم متن به صفحات چاپی) ما را یاری خواهند کرد. اولین مکانیزم، رویداد PrintPage است، که بعد از چاپ شدن یک صفحه روی می‌دهد. این رویداد یک آرگومان از نوع PrintPageEventArgs می‌گیرد، که ابعاد و مشخصات صفحهٔ چاپ شده را در خود دارد. دومین مکانیزم، متد Graphics.MeasureString است؛ این متد می‌تواند تعیین کند که در یک کادر مستطیلی مشخص (صفحهٔ چاپی) چند خط و کاراکتر جای می‌گیرد. با استفاده از این مکانیزمها (و چند تای دیگر)، چاپ سندهای چندصفحه‌ای به کاری نسبتاً ساده تبدیل خواهد شد.

در تمرین زیر برنامه‌ای می‌نویسیم بنام Print File ، که یک فایل متنی را باز کرده و آنرا چاپ می‌کند. در این تمرین با طرز استفاده از کنترلهای RichTextBox ، PrintDialog و OpenFileDialog نیز آشنا خواهید شد. کنترل RichTextBox یک جعبه متن پیشرفته است، که علاوه بر متن ساده، می‌تواند فرمت آن (فونت، اندازه و رنگ) را هم نمایش دهد. کنترل PrintDialog یک دیالوگ Print باز کرده، و به شما امکان می‌دهد تا تنظیمات چاپ را انجام دهید. (با کنترل OpenFileDialog هم در فصل ۴ آشنا شدید.) با وجود ظاهر سادهٔ این برنامه، از تکنیکهای آن می‌توانید در پروژه‌های پیشرفته‌تر هم استفاده کنید.

مدیریت کارهای چاپی

۱ پروژهٔ قبلی را با فرمان File|Close Solution بسته، و یک پروژهٔ جدید Visual Basic Windows Application بنام **My Print File** در پوشهٔ c:\vbnetsbs\chap18 ایجاد کنید.

۲ دو دکمه در بالای فرم برنامه (و متمایل به سمت چپ) رسم کنید.

۳ کنترل RichTextBox را از جعبه ابزار ویژوال بیسیک انتخاب کرده، و با آن یک جعبه متن که تقریباً

نیمهٔ پائینی فرم را بپوشاند، رسم کنید.

٤ روی کنترل OpenFileDialog دو-کلیک کنید، تا یک دیالوگ «باز کردن فایل» به سینی اجزاء اضافه شود (از این دیالوگ برای باز کردن فایل متن استفاده خواهیم کرد).

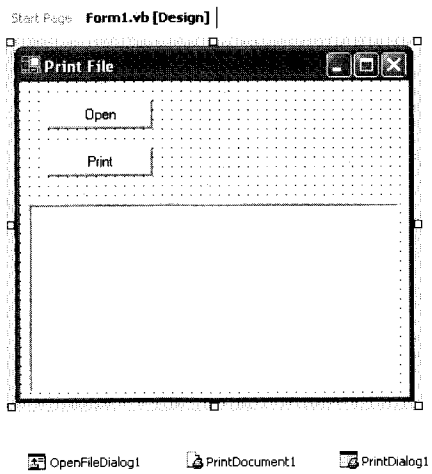
٥ روی کنترل PrintDocument دو-کلیک کنید، تا یک شیء PrintDocument به سینی اجزاء اضافه شود.

٦ روی کنترل PrintDialog دو-کلیک کنید، تا یک دیالوگ «چاپ» به سینی اجزاء اضافه شود (از این دیالوگ برای باز کردن دیالوگ چاپ استفاده خواهیم کرد).

۷ خواص فرم برنامه و کنترلهای روی آنرا با توجه به جدول زیر ست کنید:

شیء	خاصیت	مقدار
Form1	Text	"Print File"
Button1	Name	btnOpen
	Text	"Open"
Button2	Name	btnPrint
	Enabled	False
	Text	"Print"
RichTextBox1	Text	(خالی)

فرم برنامه را در شکل زیر ملاحظه می‌کنید:



اکنون نوبت نوشتن کُد لازم برای باز کردن فایل متنی و چاپ آن است.

۸ روی دکمهٔ Open دو-کلیک کنید، تا روال رویداد btnOpen_Click در ادیتور کُد باز شود.

۹ به بالای فرم رفته، و دستورات زیر را آنجا بنویسید:

```
Imports System.IO     'for FileSystem class
Imports System.Drawing.Printing
```

دستورات فوق کتابخانه‌های کلاس FileSystem و چاپ را در اختیار برنامه می‌گذارند.

۱۰ متغیرهای زیر را بعد از خط "Windows Form Designer generated code" تعریف کنید:

```
Private PrintPageSettings As New PageSettings()
Private StringToPrint As String
Private PrintFont As New Font("Arial", 10)
```

اینها اطلاعات مهمی هستند، که برای چاپ فایل به آنها نیاز داریم.

۱۱ به روال btnOpen_Click برگردید، و کُد زیر را در آن بنویسید:

```
Dim FilePath As String
'Display Open dialog box and select text file
OpenFileDialog1.Filter = "Text files (*.txt)|*.txt"
OpenFileDialog1.ShowDialog()
'If Cancel button not selected, load FilePath variable
If OpenFileDialog1.FileName <> "" Then
    FilePath = OpenFileDialog1.FileName
    Try
        'Read text file and load into RichTextBox1
        Dim MyFileStream As New FileStream(FilePath, FileMode.Open)
        RichTextBox1.LoadFile(MyFileStream, RichTextBoxStreamType.PlainText)
        MyFileStream.Close()
        'Initialize string to print
        StringToPrint = RichTextBox1.Text
        'Enable Print button
        btnPrint.Enabled = True
    Catch ex As Exception
        'display error messages if they appear
        MessageBox.Show(ex.Message)
    End Try
End If
```

وقتی کاربر دکمهٔ Open را کلیک می‌کند، این روال یک دیالوگ «باز کردن فایل» نشان می‌دهد که فقط فایلهای متنی (*.txt) را فیلتر می‌کند. نام و مسیر فایلی که کاربر انتخاب می‌کند، در متغیر FilePath ذخیره می‌شود. سپس، این فایل در یک بلوک Try...Catch باز شده، و در کنترل RichTextBox1 نوشته می‌شود (توجه کنید که برای سهولت در خواندن فایل متنی از یک استریم FileStream استفاده کرده‌ایم). و در پایان، دکمهٔ btnPrint فعال می‌شود تا کاربر بتواند فایل را چاپ کند (کاری که در یک روتین دیگر انجام خواهد شد).

در مرحلهٔ بعد کُد لازم برای نمایش دیالوگ چاپ، و چاپ صفحه به صفحهٔ فایل را می‌نویسیم.

چاپ فایل

۱ به فرم برنامه برگردید، و روی دکمهٔ Print دو-کلیک کنید تا روال رویداد btnPrint_Click در ادیتور کُد باز شود.

۲ کُد زیر را در این روال بنویسید:

```
Try
    'Specify current page settings
    PrintDocument1.DefaultPageSettings = PrintPageSettings
    'Specify document for print dialog box and show
    StringToPrint = RichTextBox1.Text
    PrintDialog1.Document = PrintDocument1
    Dim result As DialogResult = PrintDialog1.ShowDialog()
    'If click OK, print document to printer
    If result = DialogResult.OK Then
        PrintDocument1.Print()
    End If
Catch ex As Exception
    'Display error message
    MessageBox.Show(ex.Message)
End Try
```

این روتین تنظیمات پیش‌فرض صفحه را برای چاپگر انجام می‌دهد، و محتویات کنترل RichTextBox1 را در متغیر StringToPrint می‌نویسد (تا اگر کاربر تغییری در متن داده باشد، آن تغییرات نیز چاپ شود). پس از آن دیالوگ «چاپ» را باز می‌کند، تا کاربر بتواند مشخصات کار چاپی (نوع چاپگر، تعداد کپی‌ها، چاپ در فایل، و مانند آن) را بدلخواه خود تنظیم کند. چاپ زمانی انجام می‌شود، که کاربر دکمهٔ OK را کلیک کرده باشد:

```
PrintDocument1.Print()
```

۳ مجدداً به فرم برنامه برگردید، و روی شیء PrintDocument1 در سینی اجزاء دو-کلیک کنید، تا روال رویداد PrintDocument1_PrintPage در ادیتور کُد باز شود.

۴ دستورات زیر را در این روال بنویسید:

```
Dim numChars As Integer
Dim numLines As Integer
Dim stringForPage As String
Dim strFormat As New StringFormat()
'Based on page setup, define drawable rectangle on page
Dim rectDraw As New RectangleF(e.MarginBounds.Left, e.MarginBounds.Top, _
        e.MarginBounds.Width, e.MarginBounds.Height)
'Define area to determine how much text can fit on a page
'Make height one line shorter to ensure text doesn't clip
Dim sizeMeasure As New SizeF(e.MarginBounds.Width, _
        e.MarginBounds.Height - PrintFont.GetHeight(e.Graphics))
```

```
'When drawing long strings, break between words
strFormat.Trimming = StringTrimming.Word
'Compute how many chars and lines can fit based on sizeMeasure
e.Graphics.MeasureString(StringToPrint, PrintFont, _
        sizeMeasure, strFormat, numChars, numLines)
'Compute string that will fit on a page
stringForPage = StringToPrint.Substring(0, numChars)
'Print string on current page
e.Graphics.DrawString(stringForPage, PrintFont, _
        Brushes.Black, rectDraw, strFormat)
'If there is more text, indicate there are more pages
If numChars < StringToPrint.Length Then
    'Subtract text from string that has been printed
    StringToPrint = StringToPrint.Substring(numChars)
    e.HasMorePages = True
Else
    e.HasMorePages = False
    'All text has been printed, so restore string
    StringToPrint = RichTextBox1.Text
End If
```


در واقع، این روتین است که چاپ را انجام می‌دهد، و این کار را با تعیین دقیق ناحیهٔ چاپ (یا مستطیل چاپ) بر اساس تنظیمات دیالوگ PageSetup انجام می‌دهد. هر متنی که در این مستطیل جا بگیرد، بصورت عادی چاپ خواهد شد؛ اگر متن از این مستطیل بیرون بزند، باید در خط یا صفحهٔ بعد چاپ شود. برای تعیین ناحیهٔ چاپ، یک متغیر بنام rectDraw از کلاس RectangleF تعریف کرده‌ایم. برای تعیین خطوطی که از حاشیهٔ راست کاغذ بیرون می‌زنند، نیز از متغیر strFormat و متد Trimming کمک گرفته‌ایم (چاپ متن هم با متد DrawString ، که قبلاً آنرا دیده‌اید، انجام می‌شود). اگر بعد از چاپ یک صفحه هنوز چیزی از متن باقی مانده باشد، خاصیت HasMorePages را به True ست می‌کنیم، تا لزوم ادامهٔ چاپ مشخص شود.

۵ با کلیک کردن دکمهٔ Save All ، برنامه را ذخیره کنید (کُد کامل این پروژه را می‌ترانید در پوشهٔ c:\vbnetsbs\chap18\print file بیابید).

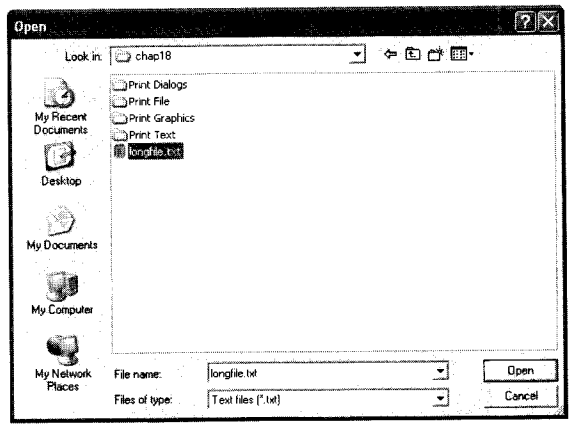
مثل اینکه این برنامه خیلی طولانی شد! اما اکنون آماده‌ایم تا آن را اجرا کرده، و فایلهای متنی چندصفحه‌ای چاپ کنیم.

اجرای برنامهٔ Print File

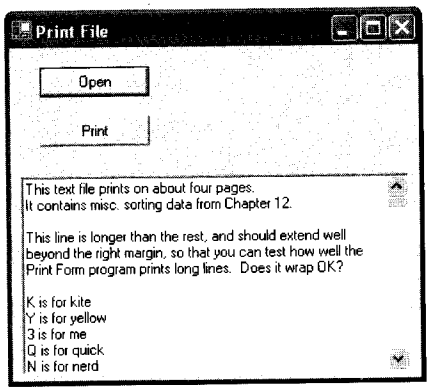
۱ برنامه را اجرا کنید. توجه کنید که در شروع برنامه (و قبل از باز کردن فایل) دکمهٔ Print غیرفعال است.

۲ دکمهٔ Open را کلیک کنید، تا دیالوگ «باز کردن فایل» ظاهر شود.

۳ به پوشهٔ c:\vbnetsbs\chap18 بروید، و فایل longfile.txt را انتخاب کنید:

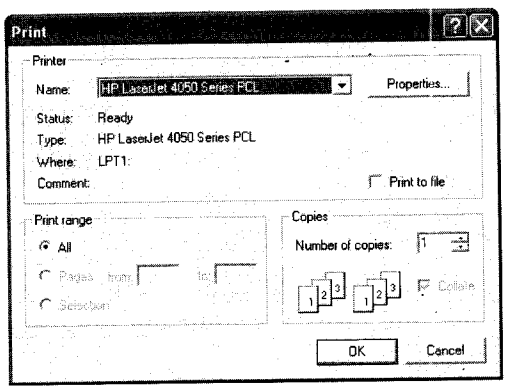


۴ دکمهٔ Open را کلیک کنید؛ بعد از بار شدن فایل در کنترل RichTextBox ، دکمهٔ Print نیز فعال می‌شود. فایل longfile.txt چند خط بسیار بلند دارد، که از حاشیهٔ راست کاغذ بیرون می‌زنند، و آنقدر بزرگ است که چند صفحهٔ چاپی (در حالت عادی، چهار صفحه) خواهد شد:



۵ بعد از آنکه چاپگر را آمادهٔ کار (online) کردید، دکمهٔ Print را کلیک کنید.

۶ با این کار، ویژوال بیسیک دیالوگ «چاپ» را - که آنرا در بسیاری از برنامه‌های ویندوز دیده‌اید - ظاهر خواهد کرد:



۷ برای چاپ فایل، دکمهٔ OK را کلیک کنید. برنامه صفحات چاپی را به چاپگر فرستاده، و بعد از چند لحظه چاپ شروع خواهد شد (دیالوگ تعداد صفحات را نیز برای چند لحظه خواهید دید).

۸ در پایان، با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

یک گام فراتر: دیالوگ‌های پیش‌نمایش چاپ و تنظیم صفحه

برنامهٔ Print File آمادهٔ چاپ کردن فایلهای متنی است، اما هنوز با امکانات چاپی برنامه‌های ویندوز تفاوت زیادی دارد. برای تکمیل این برنامه می‌توانید امکانات دیگری از قبیل پیش‌نمایش چاپ و تنظیم صفحه به آن اضافه کنید. در تمرین زیر کنترلهای PrintPreviewDialog و PageSetupDialog را به برنامهٔ Print File اضافه کرده، و به کمک آنها امکانات فوق را در اختیار کاربر قرار می‌دهیم.

اضافه کردن کنترلهای PrintPreviewDialog و PageSetupDialog

۱ اگر تمرین قبل را دنبال نکرده‌اید، پروژهٔ Print File را از پوشهٔ c:\vbnetsbs\chap18\print file باز کنید.

۲ فرم برنامه را باز کرده، و دو دکمهٔ دیگر در قسمت بالای آن قرار دهید.

۳ روی کنترل PrintPreviewDialog در برگهٔ Windows Forms جعبه ابزار ویژوال بیسیک دو-کلیک کنید، تا یک شیء پیش‌نمایش چاپ به سینی اجزاء اضافه شود.

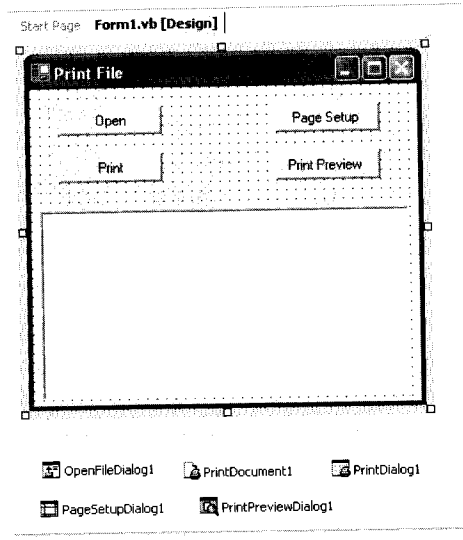
٤ روی کنترل PageSetupDialog در برگهٔ Windows Forms جعبه ابزار ویژوال بیسیک دو-کلیک کنید، تا یک شیء تنظیم صفحه به سینی اجزاء اضافه شود.

اگر اشیاء در سینی اجزاء روی هم افتاده‌اند، در آن راست-کلیک کرده و با انتخاب فرمان Line Up Icons آنها را مرتب کنید.

۵ خواص دکمه‌های جدید را با توجه به جدول زیر ست کنید:

شیء	خاصیت	مقدار
Button1	Name	btnSetup
	Enabled	False
	Text	"Page Setup"
Button2	Name	btnPreview
	Enabled	False
	Text	"Print Preview"

فرم برنامه را در شکل زیر می‌بینید:



۶ روی دکمهٔ Page Setup دو-کلیک کنید، تا روال رویداد btnSetup_Click در ادیتور کُد باز شود.

۷ کُد زیر را در این روال بنویسید:

```
Try
    'Load page settings and display page setup dialog box
    PageSetupDialog1.PageSettings = PrintPageSettings
    PageSetupDialog1.ShowDialog()
Catch ex As Exception
    'Display error message
    MessageBox.Show(ex.Message)
End Try
```

باز کردن دیالوگ PageSetupDialog بسیار ساده است، چون متغیر PrintPageSettings قبلاً در بالای فرم برنامه تعریف شده است. این متغیر اطلاعات مربوط به صفحهٔ فعلی (از قبیل جهت کاغذ، حاشیه‌ها، و غیره) را در خود دارد، و وقتی از آن استفاده کنید، دیالوگ تنظیم صفحه با مقادیر از پیش تنظیم شده باز می‌شود. این کار باعث شده تا بلوک Try...Catch فوق بسیار ساده باشد.

۸ به فرم برنامه برگردید، و روی دکمهٔ Print Preview دو-کلیک کنید تا روال رویداد btnPreview_Click در ادیتور کُد باز شود.

۹ کُد زیر را در این روال بنویسید:

```
Try
    'Specify current page settings
    PrintDocument1.DefaultPageSettings = PrintPageSettings
    'Specify document for print preview dialog box and show
    StringToPrint = RichTextBox1.Text
```

```
    PrintPreviewDialog1.Document = PrintDocument1
    PrintPreviewDialog1.ShowDialog()
Catch ex As Exception
    'Display error message
    MessageBox.Show(ex.Message)
End Try
```

در اینجا هم ابتدا مقدار متغیر PrintPageSettings به خاصیت DefaultPageSettings شیء PrintDocument1 نسبت داده شده، و محتویات کنترل RichTextBox1 در متغیر StringToPrint نوشته می‌شود. پس از آن دیالوگ PrintPreviewDialog با متد ShowDialog باز می‌شود. دیالوگ PrintPreviewDialog برای پیش‌نمایش چاپی سند مشخص شده از اطلاعات DefaultPageSettings استفاده می‌کند.

کُد قبلی برنامه هم به چند دستکاری کوچک نیاز دارد.

۱۰ در ادیتور کُد به روال رویداد btnOpen_Click بروید. در این روال علاوه بر دکمهٔ Print ، باید دکمه‌های جدید را هم فعال کنیم.

۱۱ در روال btnOpen_Click ، به بعد از دستور btnPrint.Enabled = True بروید.

۱۲ دستورات زیر را بعد از خط فوق اضافه کنید:

```
btnSetup.Enabled = True
btnPreview.Enabled = True
```

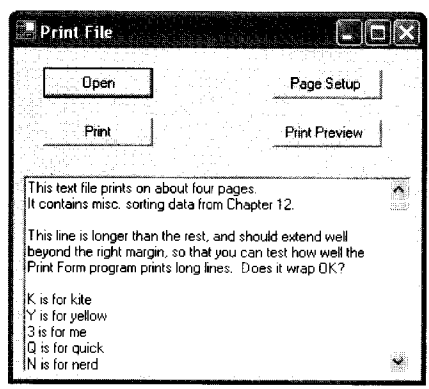
حالا برنامه از هر جهت آماده است.

۱۳ با کلیک کردن دکمهٔ Save All ، برنامه را ذخیره کنید (کُد کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap18\print dialogs بیابید).

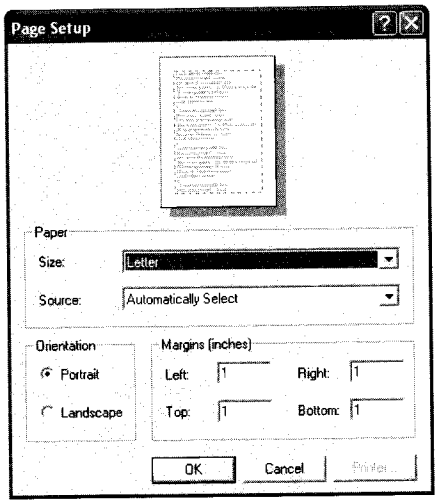
تست پیش‌نمایش چاپ و تنظیم صفحه

۱ برنامه را اجرا کنید؛ در این وضعیت فقط دکمهٔ Open فعال است، که با آن می‌توانید فایل را باز کنید.

۲ دکمهٔ Open را کلیک کرده، و فایل longfile.txt را از پوشهٔ c:\vbnetsbs\chap18 باز کنید. اکنون سه دکمهٔ دیگر نیز فعال شده‌اند:



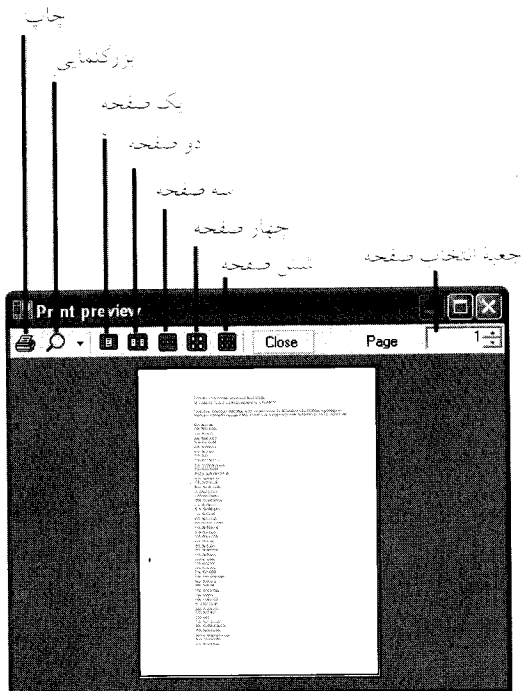
۳ دکمهٔ Page Setup را کلیک کنید، تا دیالوگ تنظیم صفحه باز شود:



در دیالوگ Page Setup گزینه‌های متعددی وجود دارد، که با آنها می‌توانید اندازهٔ کاغذ، جهت چاپ (افقی یا عمودی)، منبع کاغذ و حاشیه‌ها (بالا، پائین، چپ، و راست) را انتخاب و تنظیم کنید.

٤ در فیلد حاشیهٔ چپ کاغذ (Left Margin) عدد 2 را وارد کرده، و OK را کلیک کنید (اعداد حاشیه بر حسب اینچ هستند).

۵ دکمهٔ Print Preview را کلیک کنید، تا دیالوگ پیش‌نمایش چاپ باز شود:

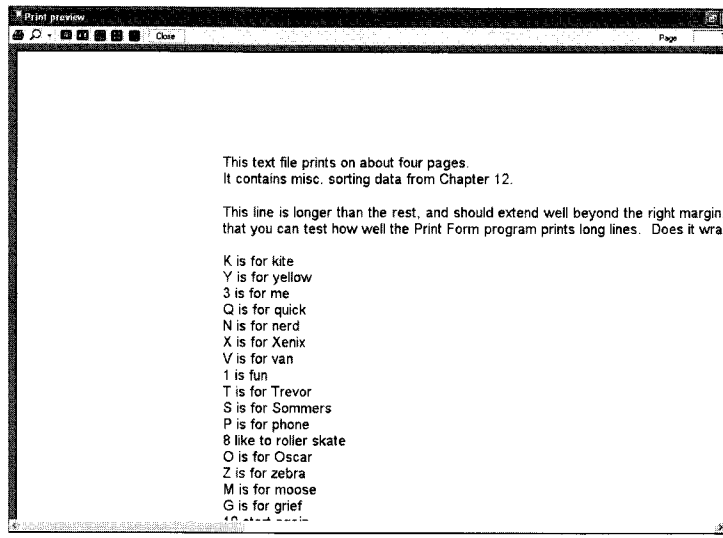


دیالوگ پیش‌نمایش چاپ دارای دکمه‌ها و گزینه‌های متعددیست، که اگر با برنامه‌هایی مانند Word یا Excel کار کرده باشید، آنها را بخوبی می‌شناسید. توجه کنید که همهٔ این امکانات را بدون هیچگونه کُدنویسی بدست آورده‌ایم!

۶ روی دکمهٔ Four Pages کلیک کنید، تا هر چهار صفحهٔ فایل longfile.txt را با هم ببینید.

۷ با کلیک کردن دکمهٔ Maximize ، پنجرهٔ Print Preview را به حالت حداکثر در آورید.

۸ روی پیکان کوچک کنار دکمهٔ Zoom کلیک کرده، و بزرگنمایی 150% را انتخاب کنید:



۹ روی دکمهٔ One Page کلیک کرده، و سپس به کمک جعبهٔ انتخاب صفحه، صفحات بعدی را مشاهده کنید. همانطور که می‌بینید، دیالوگ پیش‌نمایش چاپ بسیار خیره‌کننده است - و همهٔ اینها را فقط با چند خط کُد بدست آورده‌ایم.

۱۰ ابتدا دیالوگ Print Preview ، و سپس برنامه را ببندید.

مرجع سریع فصل ۱۸

برای ...	انجام دهید
آماده کردن برنامه برای چاپ	با دستور زیر کلاس PrintDocument را به برنامه اضافه کنید: `Imports System.Drawing.Printing`

اضافه کردن یک روتین چاپ	از دستور AddHandler و عملگر AddressOf استفاده کنید:
	`AddHandler PrintDocument1.PrintPage, _`
	`AddressOf Me.PrintGraphic`

ایجاد یک شیء PrintDocument	کنترل PrintDocument را به پروژه اضافه کرده، و سپس یک متغیر از آن تعریف کنید: `Dim PrintDoc As New PrintDocument1()`

چاپ گرافیک	از متد Graphics.DrawImage استفاده کنید:
	`ev.Graphics.DrawImage(Image.FromFile(TextBox1.Text), _`
	`ev.Graphics.VisibleClipBounds`

چاپ متن	از متد Graphics.DrawString استفاده کنید:
	`ev.Graphics.DrawString(TextBox1.Text, New Font("Arial", _`
	`11, FontStyle.Regular), Brushes.Black, 120, 120)`

فراخوانی روتین چاپ	از متد Print شیء PrintDocument استفاده کنید: `PrintDocument1.Print()`
چاپ سندهای چندصفحه‌ای	برای رویداد PrintPage یک روتین چاپ نوشته، و در آن با استفاده از آرگومان PrintPageEventArgs ابعاد صفحه، طول خطوط چاپی، و تعداد کاراکترهایی که در یک صفحه جا می‌گیرند، را تعیین کنید. تا زمانیکه هنوز چیزی برای چاپ وجود دارد، خاصیت HasMorePages را True نگه دارید.

باز کردن یک فایل متنی با کلاس FileStrem و بار کردن آن در شیء RichTextBox	یک متغیر از نوع FileStream تعریف کرده، و بعد از دادن مسیر و نام فایل به آن، استریم را در کنترل RichTextBox بار کرده، و سپس آنرا ببندید:
	`Imports System.IO 'at the top of the form`
	`...`
	`Dim MyFileStream As New FileStream( _`
	`FilePath, FileMode.Open)`
	`RichTextBox1.LoadFile(MyFileStream, _`
	`RichTextBoxStreamType.PlainText)`
	`MyFileStream.Close()`

نمایش دیالوگهای چاپ	از کنترلهای PrintDialog ، PrintPreviewDialog و PageSetupDialog که در جعبه ابزار ویژوال بیسیک قرار دارند، استفاده کنید.

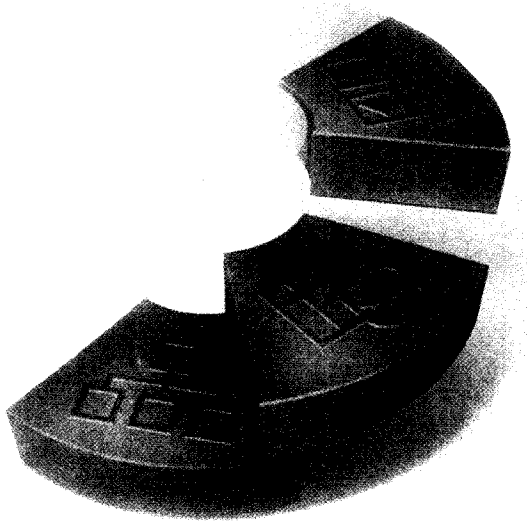
VISUAL BASIC .NET

آموزش گام به گام

۵

بخش

برنامه‌نویسی پایگاه‌داده



فصل

۱۹

آشنایی با ADO.NET

در این فصل یاد می گیرید چگونه :

- ✔ برای برقراری اتصال به یک پایگاه داده، از کاوشگر میزبان (Server Explorer) استفاده کنید.
- ✔ برای استخراج اطلاعات خاص از پایگاه داده، یک آداپتور داده (data adapter) ایجاد کنید.
- ✔ برای نمایش جدولهای پایگاه داده، یک دیتاست (dataset) ایجاد کنید.
- ✔ برای نمایش اطلاعات پایگاه داده و حرکت در آن، از کنترلهای جعبه متن، برچسب، و دکمه استفاده کنید.

در فصل ۵ دیدید که چگونه می‌توان اطلاعات را در متغیرها ذخیره کرد. در این فصل با استاندارد جدید میکروسافت در ذخیره و بازیابی اطلاعات در پایگاه‌های داده (که ADO.NET نام دارد) آشنا می‌شوید. ویژوال بیسیک .NET یکی از بهترین ابزارهای موجود برای ایجاد برنامه‌های پایگاه داده است، و اگر چنین قصدی دارید می‌توانید بلافاصله کار خود را شروع کنید.

در این فصل نحوهٔ ارتباط با پایگاه‌های داده از طریق ADO.NET را خواهید دید. خواهید دید که، چگونه می‌توان به کمک کاوشگر میزبان به یک پایگاه دادهٔ Access متصل شد، اتصال ایجاد شده را با استفاده از دیالوگ خواص لینک داده (Data Link Properties) پیکربندی کرد، به کمک آداپتور داده جدولهای پایگاه داده را انتخاب کرد، و برای نمایش اطلاعات منتخب از دیتاست استفاده کرد. و بالاخره بعد از برداشتن این قدمهای اولیه، به کمک کنترلهای جعبه متن، برچسب و دکمه اطلاعات استخراج شده را روی فرم برنامه نمایش خواهیم داد.

تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- در ویژوال بیسیک .NET تکنولوژی جدید ADO.NET (که بر +ADO متکیست) جایگزین تکنولوژیهای قدیمی RDO و ADO شده است.
- ADO.NET مدل دادهٔ استاندارد در تمام برنامه‌های ویژوال استودیو .NET (ویژوال بیسیک .NET ، ویژوال سی++ .NET ، و ویژوال سی# .NET) است.
- در ویژوال بیسیک .NET کنترلهای Data و ADO Data دیگر وجود ندارند، و برای نمایش اطلاعات پایگاه داده در برنامه باید از آداپتور داده و دیتاست استفاده کنید.
- ویژوال بیسیک ۶ اطلاعات پایگاه داده را از طریق شیء رکوردست نمایش می‌داد. در ویژوال بیسیک .NET این کار بر عهدهٔ شیء دیتاست (که تصویریست از یک جدول پایگاه داده) گذاشته شده است.
- فرمت داخلی داده‌های ADO.NET همان فرمت معروف وب یعنی XML است. بدین ترتیب، استفاده از این مدل در برنامه‌های طراحی شده برای وب بسیار ساده‌تر شده است.

برنامه‌نویسی پایگاه داده با ADO.NET

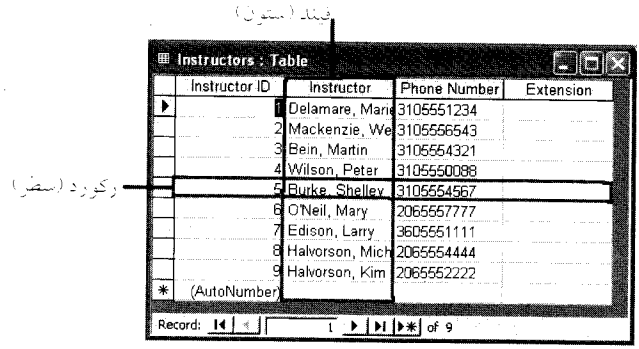
پایگاه داده مجموعه‌ایست از اطلاعات سازماندهی شده در یک فایل. امروزه برنامه‌های بسیار زیادی برای ایجاد یک پایگاه داده وجود دارد، که برخی از معروفترین آنها عبارتند از: Microsoft Access ، Microsoft FoxPro ، Microsoft SQL Server ، Btrieve ، Paradox ، و Oracle . روش دیگری نیز برای ذخیره کردن اطلاعات سازماندهی شده وجود دارد، که بطور خاص برای تبادل اطلاعات روی اینترنت طراحی شده ولی امروزه محبوبیت فراگیری پیدا کرده، و آن فرمت XML (eXtensible Markup Language) است.

پایگاه داده یکی از فراگیرترین مباحث برنامه‌نویسی است، چون کمتر برنامه‌ای را می‌توان پیدا کرد که به نوعی با اطلاعات سروکار نداشته باشد. در واقع، اطلاعات - آدرس مشتریان، موجودی انبار، وضعیت حسابهای بانکی، اطلاعات پرسنلی کارمندان و هزاران آیتم دیگر - برای دنیای صنعت یا تجارت بمثابهٔ خون در رگهای انسان است.

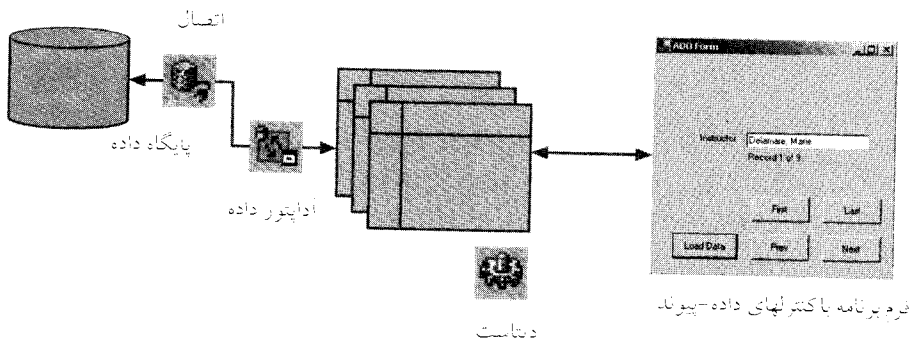
ویژوال بیسیک .NET یک برنامهٔ اختصاصی پایگاه داده نیست، اما در نمایش، آنالیز و پردازش اطلاعات موجود در پایگاه داده تبحر فراوانی دارد. ویرایشهای قبلی ویژوال بیسیک از مدلهای متنوعی برای کار با پایگاه‌های داده استفاده می‌کردند، اما ویژوال بیسیک .NET با مدل جدید ADO.NET از فرمتهای پایگاه دادهٔ بیشتری پشتیبانی می‌کند. بویژه، ADO.NET برای کار روی اینترنت طراحی شده، و این به معنای آنست که می‌توانید براحتی برنامه‌های پایگاه دادهٔ خود را برای کار روی وب بنویسید. برای رسیدن به این هدف، میکروسافت از XML بعنوان فرمت داخلی داده‌ها در ADO.NET استفاده کرده است.

آشنایی با اصطلاحات پایگاه داده

برای کار با پایگاه داده و ADO.NET باید برخی از اصطلاحات فنی آنرا بدانید. به کوچکترین واحد اطلاعاتی که می‌توان در یک پایگاه داده ذخیره کرد، **فیلد** (field) گفته می‌شود. برای مثال، نام، نام خانوادگی، آدرس، شماره تلفن، و شغل می‌توانند فیلدهای یک پایگاه دادهٔ کوچک باشند. به مجموعه اطلاعات مربوط به یک فرد یا شرکت خاص **رکورد** (record) می‌گویند. از به هم پیوستن چند رکورد نیز، یک **جدول** (table) بوجود می‌آید. در برخی از مراجع و کتب، به فیلد و رکورد بترتیب **ستون** (column) و **سطر** (row) گفته می‌شود (شکل زیر را ببینید).



در ADO.NET اشیاء متعددی برای برقراری یک ارتباط با پایگاه داده و بازیابی اطلاعات از آن دخیل هستند، که آنها را در شکل زیر می‌بینید:



ابتدا یک اتصال (connection) به پایگاه داده برقرار می‌شود، که اطلاعات مربوط به آن را برمی‌گرداند. سپس یک آداپتور داده (data adapter) ایجاد می‌شود، که مدیریت بازیابی اطلاعات از پایگاه داده و یا نوشتن داده‌های جدید در آنرا بر عهده دارد. پس از آن اطلاعات جدول (یا جدولهای) پایگاه داده در یک دیتاست (dataset) بازیابی می‌شود (در این روش، شما با یک کپی اطلاعات پایگاه داده سروکار دارید، نه خود آن). برای نمایش اطلاعات موجود در این دیتاست می‌توان از کنترلهای ویژوال بیسیک استفاده کرد.

کار با یک پایگاه دادهٔ Access

در این قسمت با طرز استفاده از تکنولوژی ADO.NET در برنامه‌های ویژوال بیسیک آشنا می‌شوید. برای

شروع، به کمک کاوشگر میزبان (Server Explorer) یک اتصال به پایگاه داده‌ای بنام Students.mdb (که با Microsoft Access ایجاد شده) می‌سازیم. پایگاه دادهٔ Students.mdb دارای جدولهای متعددیست که به کمک آنها می‌توان اطلاعات مربوط به دانشجویان یک مؤسسهٔ آموزشی را مدیریت کرد. در تمرین زیر طرز ایجاد یک دیتاست از جدولهای این پایگاه داده، و نمایش اطلاعات آن در ویژوال بیسیک را خواهید دید.

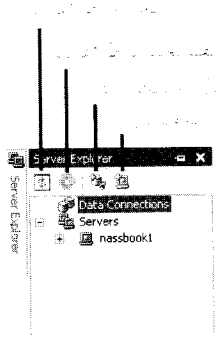
نکته

برای اجرای برنامهٔ زیر نیازی نیست که Access را در کامپیوتر خود داشته باشید. ویژوال استودیو و ADO.NET تمام ابزارهای لازم برای پشتیبانی از فرمت پایگاه دادهٔ Access (و بسیاری از فرمتهای دیگر) را دارند.

برقراری اتصال به پایگاه داده

۱ در محیط ویژوال استودیو .NET، یک پروژهٔ جدید Visual Basic Windows Application بنام **My ADO Form** در پوشهٔ c:\vbnetsbs\chap19 ایجاد کنید.

۲ از منوی View فرمان Server Explorer را انتخاب کنید، تا پنجرهٔ کاوشگر میزبان باز شود:



نکته

بسته به ویرایش ویژوال استودیو که با آن کار می‌کنید، ممکنست گزینهٔ «اتصال به میزبان» را در اختیار نداشته باشید، که البته در این تمرین نیازی به آن نداریم.

کاوشگر میزبان یک ابزار گرافیکی است، که به شما اجازه می‌دهد تا به انواع پایگاه داده (محلی، روی شبکه، و یا روی اینترنت) متصل شوید. با این ابزار می‌توانید مشخصات و ساختار پایگاه دادهٔ موردنظر (جدول‌ها، فیلدها، و رکوردها) را ارزیابی کنید، و یا وارد آنها شوید (logon).

۳ برای پردازش اطلاعات یک پایگاه داده، باید به آن متصل شوید؛ پس - در کاوشگر میزبان، روی دکمهٔ Connect to Database کلیک کنید. با این کار پنجرهٔ خواص لینک داده

(Data Link Properties) باز می‌شود، که می‌توانید در آن اطلاعات مربوط به پایگاه دادهٔ موردنظر (فرمت، محل و کلمهٔ رمز پایگاه داده، و از این قبیل) را وارد کنید.

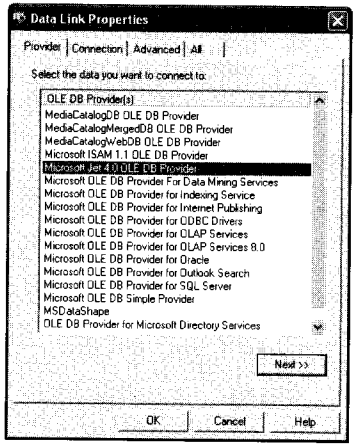
نکته

برای باز کردن «پنجرهٔ خواص لینک داده»، می‌توانید از فرمان Tools|Connect to Database نیز استفاده کنید.

٤ به برگهٔ Provider بروید.

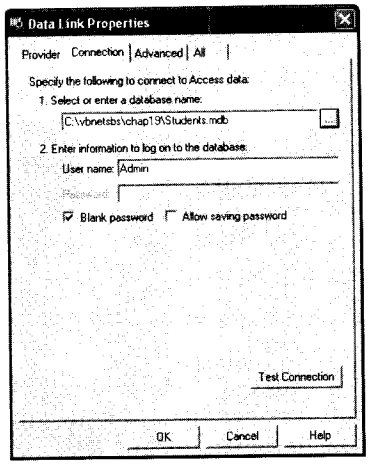
رابط (provider) یک واسطهٔ نرم‌افزاریست که می‌داند چگونه باید به پایگاه داده متصل شده، و اطلاعات موجود در آنرا استخراج کند. OLE DB و SQL دو تا از مهمترین رابط‌های موجود در ویژوال استودیو .NET هستند، ولی برای بسیاری از فرمتهای معروف پایگاه داده نیز در حال حاضر رابط مناسب در بازار موجود است. در این تمرین از رابط Microsoft Jet 4.0 OLE DB Provider (که برای اتصال به پایگاه‌های دادهٔ Access طراحی شده) استفاده خواهیم کرد.

٥ روی آیتم Microsoft Jet 4.0 OLE DB Provider کلیک و آنرا انتخاب کنید (شکل زیر را ببینید).



٦ برای رفتن به برگهٔ Connection ، روی دکمهٔ Next کلیک کنید. از آنجائیکه فرمت Jet OLE DB را انتخاب کرده‌اید، در قسمت بعد باید مشخصات پایگاه دادهٔ Access را وارد کنید.

٧ روی دکمهٔ ... کنار فیلد Select or enter a database name کلیک کرده، و بعد از انتخاب پایگاه دادهٔ Students.mdb (که در پوشهٔ c:\vbnetsbs\chap19 قرار دارد)، Open را کلیک کنید (شکل زیر را ببینید).



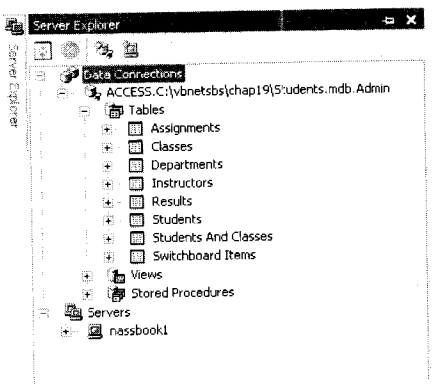
نکته

اگر بخواهید می‌توانید از پایگاه دادهٔ دیگری استفاده کنید، اما مراحل بعد را نیز باید متناسب با آن تغییر دهید.

۸ روی دکمهٔ Test Connection کلیک کنید. با این کار ویژوال استودیو سعی می‌کند به پایگاه دادهٔ Students.mdb متصل شود، و اگر پیام Test Connection Succeeded ظاهر شد، یعنی همه چیز درست است. (اگر در این تست ویژوال استودیو پیام خطا داد، باحتمال زیاد مشکل در نوع رابطی است که انتخاب کرده‌اید).

۹ ابتدا دکمهٔ OK در پیام Test Connection Succeeded ، و سپس دکمهٔ OK در پنجرهٔ Data Link Properties را کلیک کنید. به گره (آیتم) جدیدی که در کاوشگر میزبان ایجاد می‌شود، دقت کنید.

۱۰ در کاوشگر میزبان، بترتیب، گره‌های Data Connections ، ACCESS و Tables را باز کنید. (برای باز کردن یک گره، کافیست روی علامت + کنار آن کلیک کنید.) همانطور که می‌بینید، کاوشگر میزبان ساختار پایگاه دادهٔ Students.mdb و اشیاء آن (جدول‌ها، فیلدها و غیره) را نشان می‌دهد.



ایجاد آداپتور داده

بعد از اتصال به پایگاه داده، باید یک آداپتور داده بسازیم تا بتوانیم اطلاعات موردنظر را از آن بازیابی کنیم. آداپتور داده، در واقع، اطلاعاتی را که باید از پایگاه داده خوانده شود، تعریف می‌کند. آداپتور داده (که می‌توان آنرا نوعی فیلتر دانست) برای بازیابی اطلاعات ضروریست، چون ممکنست یک پایگاه داده از جدولهای متعدد و بسیار بزرگی تشکیل شده باشد.

برای ایجاد یک آداپتور داده در ویژوال استودیو روشهای مختلفی وجود دارد، که ساده‌ترین آنها کشیدن آیکون جدول موردنظر از پنجرهٔ کاوشگر میزبان و انداختن آن روی فرم برنامه است (که با این کار یک شیء آداپتور داده در سینی اجزاء ظاهر می‌شود). روش دوم -که در اینجا از آن استفاده خواهیم کرد- ابزاریست که «جادوگر پیکربندی آداپتور داده» (Data Adapter Configuratin Wizard) نام دارد. این روش به شما اجازه می‌دهد تا بر انتخاب داده‌ها کنترل کاملی داشته باشید.

استفاده از کنترل OleDbDataAdapter

۱ در جعبه ابزار ویژوال بیسیک، روی برگهٔ Data کلیک کنید. در این برگه کنترلهایی را خواهید دید که برای بازیابی اطلاعات از منابع داده بکار می‌آیند. همانطور که می‌بینید، دیگر از کنترلهای Data و ADO Data خبری نیست، و برای دسترسی به منابع داده باید از آداپتورهای داده و دیتاست‌ها استفاده کنید.

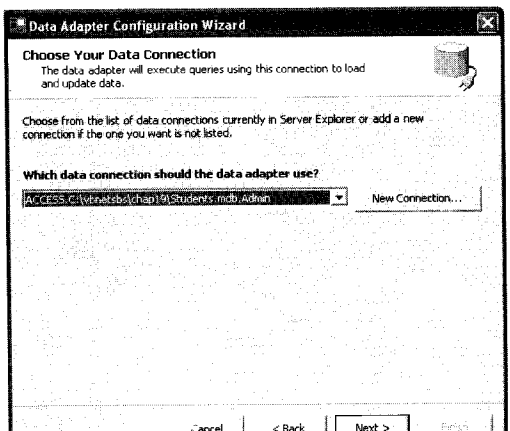
نکته

برای برقراری اتصال به پایگاه داده از کنترلهای OleDbConnection و SqlConnection نیز می‌توانید استفاده کنید. اما، از آنجائیکه قبلاً این کار را انجام داده‌ایم، در اینجا نیازی به استفاده از آنها نیست.

۲ کنترل OleDbDataAdapter را گرفته، و روی فرم برنامه بکشید (این کنترل برای کار با فرمتهای Access/Jet -و بسیّاری از فرمتهای دیگر- طراحی شده است).

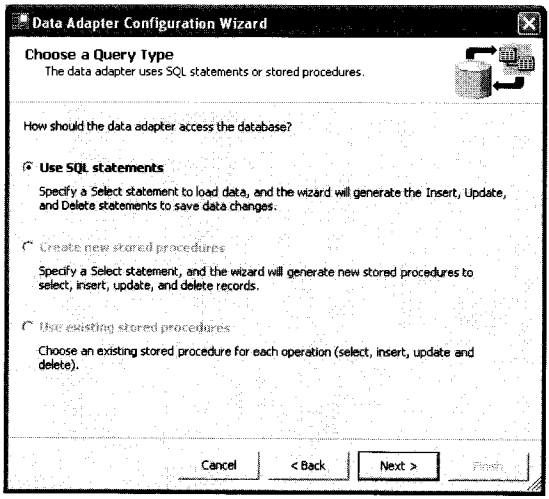
به محض رها کردن کنترل OleDbDataAdapter روی فرم برنامه، ویژوال استودیو «جادوگر پیکربندی آداپتور داده» را شروع می‌کند.

۳ توضیحات صفحهٔ اول را (که دربارهٔ آداپتور داده است) بخوانید، و سپس دکمهٔ Next را کلیک کنید. در صفحهٔ دوم باید نام یک اتصال معتبر را وارد کنید:



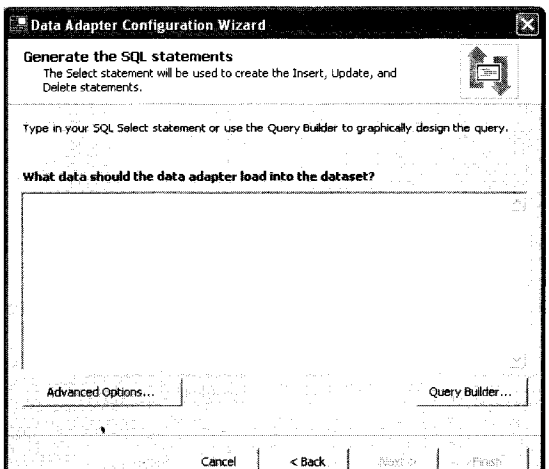
از آنجائیکه قبلاً یک اتصال به پایگاه دادهٔ Access ایجاد کرده‌ایم، نام آن را در لیست Which data connection should the data adapter use? خواهید دید. (اگر قبلاً چنین اتصالی را نساخته‌اید، می‌توانید به کمک دکمهٔ New Connection این کار را انجام دهید.)

٤ Next را کلیک کنید؛ در این مرحله باید نوع اطلاعاتی را که می‌خواهید این آداپتور داده بازیابی کند، مشخص کنید:

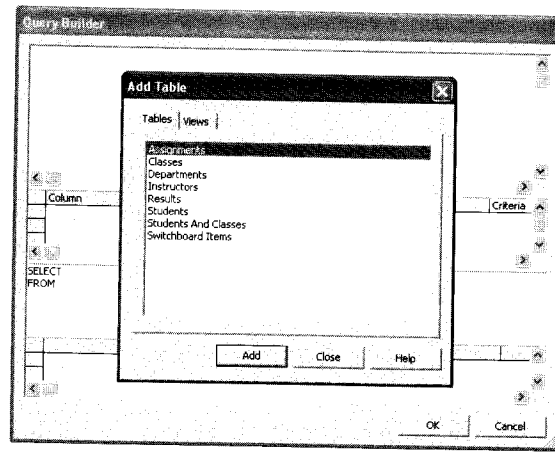


اولین گزینه، دستور SQL ، اجازه می‌دهد تا برای انتخاب و فیلتر کردن اطلاعات یک دستور SQL SELECT ایجاد کنید. اگر قبلاً تجربهٔ برنامه‌نویسی پایگاه داده داشته باشید، براحتی می‌توانید یک دستور SQL SELECT بنویسید؛ اما اگر چنین نیست، می‌توانید در مرحلهٔ بعدی از Query Bulider کمک بگیرید (و این همان کاریست که ما در این تمرین خواهیم کرد).

٥ Next را کلیک کنید؛ در این مرحله باید برای فیلتر کردن اطلاعات موردنظر، یک دستور SQL SELECT بنویسید:



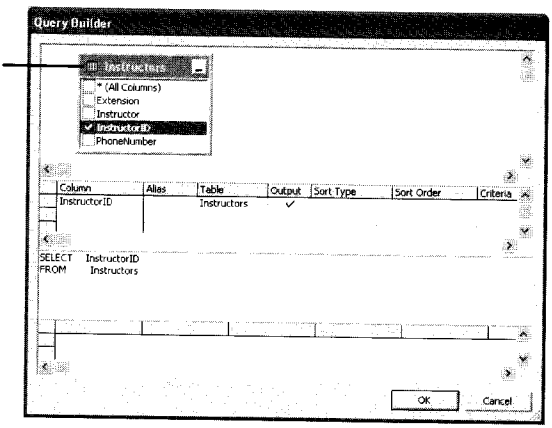
۶ دکمهٔ Query Builder را کلیک کنید؛ Query Builder ابزاریست که به شما اجازه می‌دهد تا یک جستجوی SQL SELECT را براحتی ایجاد کنید:



برای نوشتن یک دستور SELECT ، ابتدا باید یکی از جدولهای پایگاه دادهٔ Students.mdb را انتخاب کنید.

۷ جدول Instructors را انتخاب کرده، و ابتدا دکمهٔ Add و بعد از آن Close را کلیک کنید. با این کار، Query Builder فیلدها (ستونها) ی جدول Instructors را نشان خواهد داد.

۸ در پنجرهٔ جدول Instructors ، روی علامت چک کنار فیلد Instructor کلیک کنید. با این کار، Query Builder یک دستور SELECT برای استخراج این فیلد از جدول Instructors ایجاد خواهد کرد:

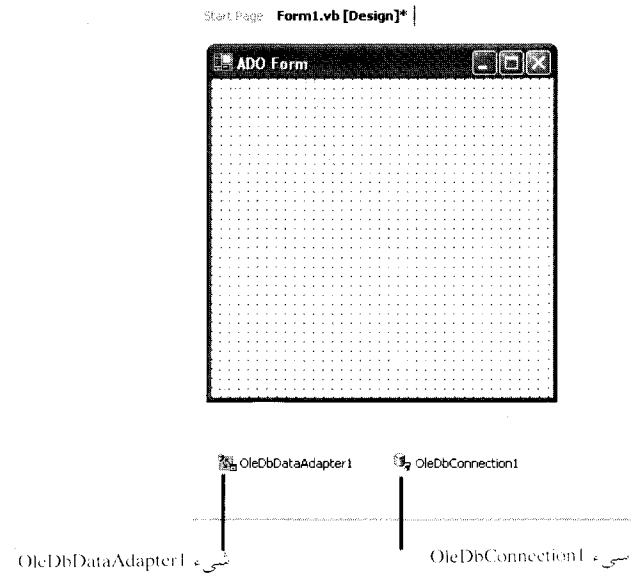


در این تمرین، ما فقط یک فیلد را از یک جدول استخراج می‌کنیم؛ اما شما می‌توانید این کار را با هر تعداد فیلد (و از هر تعداد جدول) که می‌خواهید انجام دهید.

۹ دکمهٔ OK را کلیک کنید، تا دستور SELECT ایجاد شده، و به «جادوگر پیکربندی آداپتور داده» برگردید.

۱۰ برای پایان عملیات، دکمهٔ Finish را کلیک کنید، تا شیء آداپتور داده ایجاد شود.

توجه کنید که ویژوال استودیو علاوه بر شیء OleDbDataAdapter1 ، یک کنترل OleDbConnection1 نیز به سینی اجزاء اضافه کرده است:



کار با دیتاست

قدم بعدی در برنامه‌نویسی ADO.NET ، ایجاد شیء دیتاست (dataset) برای نمایش فیزیکی داده‌ها در برنامه است. دیتاست مجموعه‌ایست از اطلاعات استخراج شده از یک (یا چند) جدول پایگاه داده (که یکی از منابع دادهٔ آن می‌تواند دستور SQL SELECT باشد). بر خلاف شیء رکوردست (recordset) در ویرایشهای قبلی ویژوال بیسیک، دیتاست فقط یک کپی از اطلاعات پایگاه داده است، و دستکاری آن باعث تغییر در پایگاه دادهٔ اصلی نخواهد شد (برای اعمال تغییرات انجام شده در پایگاه داده، باید از دستور خاصی استفاده کنید).

در تمرین زیر یک دیتاست برای نمایش فیلد Instructor جدول Instructors پایگاه دادهٔ Students.mdb ایجاد خواهیم کرد. همانطور که خواهید دید، اگر آداپتور داده بدرستی پیکربندی شده باشد، ایجاد دیتاست کار ساده‌ایست.

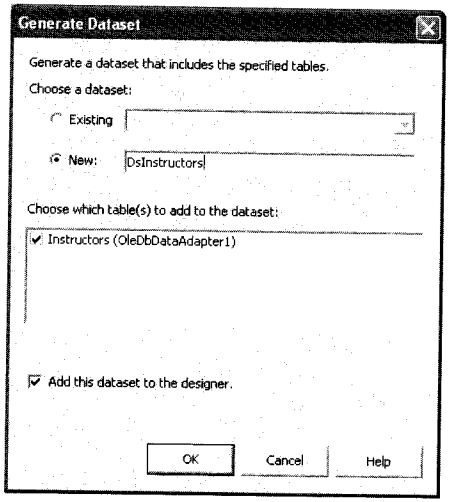
ایجاد دیتاست فیلد Instructor

۱ روی فرم برنامه کلیک کنید، تا فعال شود.

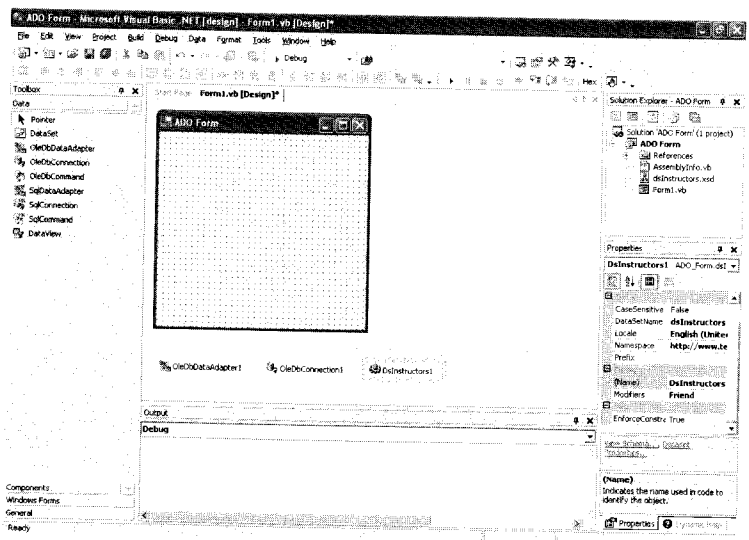
۲ از منوی Data فرمان Generate Dataset را انتخاب کنید، تا پنجرهٔ «ایجاد دیتاست» باز شود.

۳ در فیلد New ، نام دیتاست جدید را **DsInstructors** وارد کنید.

٤ دقت کنید که جعبه چک Add this dataset to the designer فعال باشد، تا ویژوال استودیو این دیتاست را به سینی اجزاء اضافه کند:



۵ OK را کلیک کنید، تا دیتاست DsInstructors ایجاد و به سینی اجزاء برنامه اضافه شود:



اگر دقت کنید، فایلی بنام DsInstructors.xsd نیز به کاوشگر راه‌حل اضافه شده است. این یک فایل اِسکِمای XML (XML schema) است، که جدولها، فیلدها، انواع داده، و سایر اطلاعات دیتاست را توصیف می‌کند. اگر دیتاستی درست کنید که دارای این اِسکِما (فایل xsd.) باشد، از مزایای زیادی

(مانند تکمیل خودکار دستورات در ادیتور کُد، و دسترسی به اطلاعات تکمیلی دربارهٔ جدولها و فیلدهای آن) برخوردار خواهید شد.

بعد از ایجاد دیتاست، آماده‌ایم تا رکوردهای جدول Instructors را روی فرم برنامه نمایش دهیم.

نمایش اطلاعات پایگاه داده با کنترل‌های پیوندی

بعد از این همه مقدمات، بالاخره آماده‌ایم تا اطلاعات پایگاه داده را روی فرم برنامه نمایش دهیم - و این قسمت مهیج کار است. اما چگونه باید آنرا انجام دهیم؟ آیا باید برنامه‌ای مثل Access را از اول بنویسیم، چون می‌خواهیم یک جدول را نشان دهیم؟ خوشبختانه ویژوال بیسیک راه ساده‌تری برای این کار دارد. اما علاوه بر نمایش اطلاعات هر رکورد، باید راهی برای حرکت بین رکوردهای مختلف نیز فراهم آوریم. اینکه کاربر بتواند اطلاعات رکوردها را دستکاری کند یا خیر، یکی دیگر از تصمیماتی است که باید بگیریم.

اغلب کنترلهایی که در جعبه ابزار ویژوال بیسیک می‌بینید، توانایی نمایش اطلاعات فیلدهای پایگاه داده را دارند. در اصطلاح ویژوال بیسیک، به این قبیل کنترلها **کنترل داده‌پیوند** (data-bound control) - یا بطور خلاصه، **کنترل پیوندی** - گفته می‌شود. برای اینکه چنین کنترلی به یک منبع اطلاعات پیوند بخورد، باید خواص DataBindings آنرا به فیلدهای موردنظر در دیتاست مرتبط نمود. چند تا از کنترلهای ویژوال بیسیک که بویژه برای نمایش اطلاعات پایگاه داده بهینه شده‌اند، عبارتند از: جعبه متن (TextBox) ، جعبه ترکیبی (ComboBox) ، جعبه لیست (ListBox) ، جعبه چک (CheckBox) ، دکمهٔ رادیویی (RadioButton) ، جعبه تصویر (PictureBox) ، و شبکهٔ داده (DataGrid) . کنترل شبکهٔ داده یکی از کنترلهائیست که اختصاصاً برای نمایش جدولهای پایگاه داده طراحی شده است؛ فصل آینده را بطور کامل به کار با این کنترل اختصاص داده‌ام.

در تمرین زیر، برای نمایش اطلاعات جدول Instructors پایگاه دادهٔ Students.mdb از یک جعبه متن استفاده خواهیم کرد.

نمایش اطلاعات با جعبه متن

۱ یک جعبه متن نسبتاً بزرگ (که جای کافی برای نشان دادن نام و نام خانوادگی یک معلم فرضی داشته باشد) در وسط فرم برنامه رسم کنید.

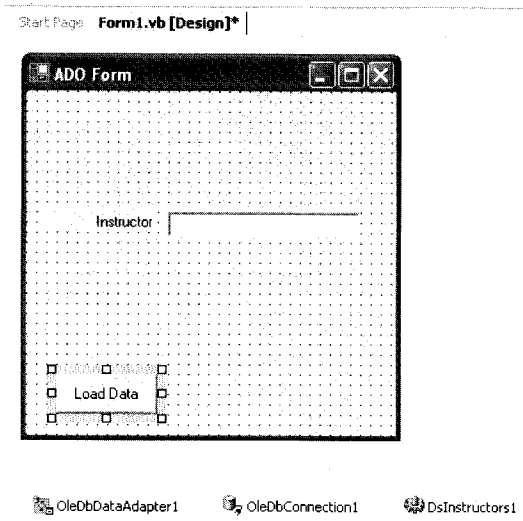
۲ سمت چپ جعبه متن فوق، یک برچسب قرار دهید.

۳ یک دکمه هم در گوشهٔ چپ-پائین فرم قرار دهید.

٤ خواص فرم برنامه و کنترلهای روی آنرا با توجه به جدول زیر ست کنید:

شیء	خاصیت	مقدار
Form1	Text	"ADO Form"
Button1	Name	btnLoad
	Text	"Load Data"
Label1	Name	lblInstructor
	Text	"Instructor"
	TextAlign	MiddleRight
TextBox1	Name	txtInstructor
	Text	(خالی)

فرم برنامه را در شکل زیر ملاحظه می‌کنید:

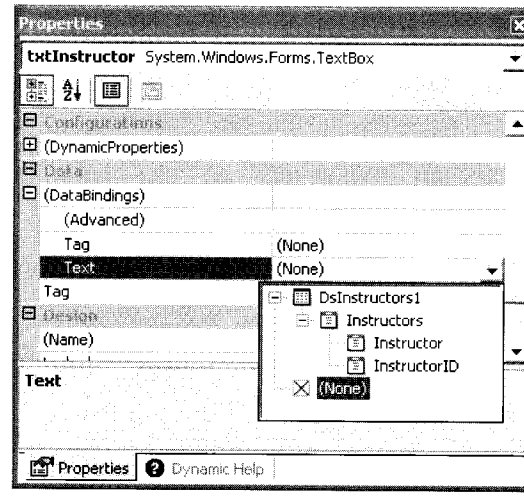


اکنون باید پیوند بین جعبه متن txtInstructor و فیلد Instructor را برقرار کنیم.

۵ روی جعبه متن txtInstructor کلیک کرده، و سپس پنجرهٔ خواص را باز کنید.

۶ در پنجرهٔ خواص، ابتدا گروه (DataBindings) ، و سپس لیست خاصیت Text را باز کنید. در این لیست تمام منابع اطلاعاتی که می‌توان اطلاعات آنها را نمایش داد، دیده می‌شوند. اگر مراحل قبلی این تمرین را بدرستی انجام داده باشید، در این لیست دیتاست DsInstructors1 را خواهید دید.

۷ روی علامت + کنار دیتاست DsInstructors1 کلیک کنید، تا جدول زیر آن باز شود:



۸ روی فیلد Instructor (نه آیکون کنار آن) کلیک و آنرا انتخاب کنید. با این کار، جعبه متن txtInstructor به فیلد Instructor پیوند می‌خورد.

اکنون نوبت نوشتن کُد بار کردن اطلاعات دیتاست DsInstructors1 در جعبه متن txtInstructor است.

۹ پنجرهٔ خواص را به حالت قبل برگردانید، روی دکمهٔ Load Data دو-کلیک کنید، و کُد زیر را در روال رویداد btnLoad_Click بنویسید:

```
DsInstructors1.Clear()
OleDbDataAdapter1.Fill(DsInstructors1)
```

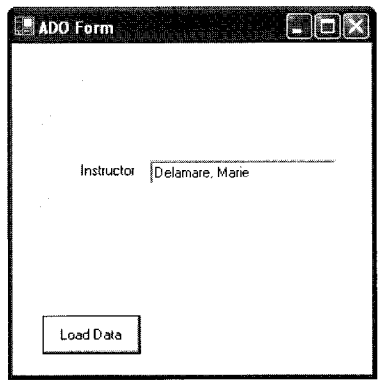
این کُد ابتدا اطلاعات موجود در شیء DsInstructors1 را پاک کرده (تا اگر دکمهٔ Load Data بیش از یک بار کلیک شد، اطلاعات تکراری در دیتاست DsInstructors1 وجود نداشته باشد)، و سپس آداپتور داده را با اطلاعات جدید پُر می‌کند. با این کار، اطلاعات بطور خودکار به جعبه متن txtInstructor منتقل خواهند شد.

نکته

این کُد را در روال Form1_Load نیز می‌توانستیم بنویسیم، تا به محض اجرای برنامه اطلاعات در جعبه‌متن بار شده و نمایش داده شود.

۱۰ برنامه را اجرا کنید.

۱۱ دکمهٔ Load Data را کلیک کنید. با این کار نام اولین معلم پایگاه دادهٔ Students.mdb در جعبه متن نشان داده خواهد شد:



۱۲ با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

برنامهٔ ADO Form در حال حاضر فقط رکورد اول را نشان می‌دهد، و نمی‌توانید سایر رکوردها را ببینید؛ لازمست فکری برای این مشکل هم بکنیم!

حرکت در دیتاست

ADO.NET برای کنترل حرکت در میان رکوردهای یک دیتاست از شیئی بنام CurrencyManager استفاده می‌کند. هر دیتاست یک CurrencyManager خاص خود دارد، و تمام CurrencyManager های یک فرم در مجموعه‌ای بنام BindingContext جمع می‌شوند.

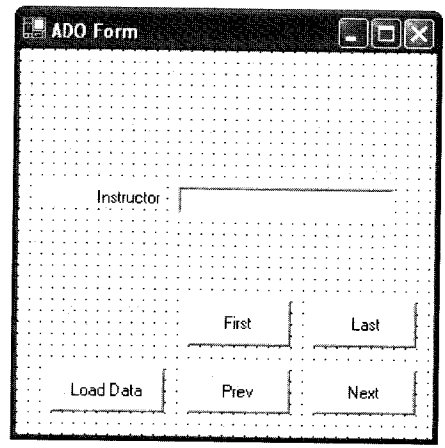
در تمرین زیر، چهار دکمه بنامهای First (رفتن به اولین رکورد دیتاست)، Last (رفتن به آخرین رکورد دیتاست)، Prev (رفتن به رکورد بعدی دیتاست)، و Next (رفتن به رکورد قبلی دیتاست) به فرم ADO Form اضافه می‌کنیم، که امکان حرکت در میان رکوردها را در اختیار کاربر می‌گذارند.

اضافه کردن دکمه‌های First ، Last ، Prev ، و Next

۱ به فرم برنامهٔ ADO Form برگردید، و چهار دکمه روی آن قرار دهید (شکل زیر را ببینید).

۲ خواص این دکمه‌ها را با توجه به جدول زیر ست کنید:

شیء	خاصیت	مقدار
Button1	Name	btnFirst
	Text	"First"
Button2	Name	btnLast
	Text	"Last"
Button3	Name	btnPrev
	Text	"Prev"
Button4	Name	btnNext
	Text	"Next"



۳ روی دکمهٔ First دو-کلیک کنید، تا روال رویداد btnFirst_Click در ادیتور کُد باز شود.

۴ دستور زیر را در این روال بنویسید:

```
Me.BindingContext(DsInstructors1, "Instructors").Position = 0
```

این دستور برای نمایش اولین رکورد جدول Instructors در دیتاست DsInstructors1 ، از شیء BindingContext استفاده کرده است. همانطور که حدس زده‌اید، شماره‌گذاری رکوردهای دیتاست نیز (مانند آرایه‌ها) از 0 شروع می‌شود. (کلمهٔ کلیدی Me نیز به فرم جاری اشاره می‌کند.)

۵ به فرم برنامه برگردید، و روی دکمهٔ Last دو-کلیک کنید، تا روال رویداد btnLast_Click در ادیتور کُد باز شود.

۶ دستور زیر را در این روال بنویسید:

```
Me.BindingContext(DsInstructors1, "Instructors").Position = _
        Me.BindingContext(DsInstructors1, "Instructors").Count - 1
```

این دستور شبیه روال btnFirst_Click است، با این تفاوت که برای رفتن به آخرین رکورد دیتاست، ابتدا تعداد کل رکوردهای دیتاست را خوانده (خاصیت Count)، و سپس یکی از آن کم می‌کند، تا شمارهٔ آخرین رکورد بدست آید (بیاد دارید که شمارهٔ اولین رکورد 0 است).

۷ به فرم برنامه برگردید، و روی دکمهٔ Prev دو-کلیک کنید، تا روال رویداد btnPrev_Click در ادیتور کُد باز شود.

۸ دستور زیر را در این روال بنویسید:

```
Me.BindingContext(DsInstructors1, "Instructors").Position -= 1
```

در اینجا برای رفتن به رکورد قبلی دیتاست، یکی از شمارهٔ رکورد فعلی کم کرده‌ایم. اما اگر در اولین رکورد دیتاست باشیم، و این دکمه را کلیک کنیم، چه اتفاقی می‌افتد (رکورد 1- چه معنایی می‌دهد)؟ اگر نگران هستید که با این کار خطایی روی دهد، باید بگویم جای نگرانی نیست، چون ADO.NET بهیچوجه اجازه نمی‌دهد از مرزهای دیتاست تخطی کنید.

۹ به فرم برنامه برگردید، و روی دکمهٔ Next دو-کلیک کنید، تا روال رویداد btnNext_Click در ادیتور کُد باز شود.

۱۰ دستور زیر را در این روال بنویسید:

```
Me.BindingContext(DsInstructors1, "Instructors").Position += 1
```

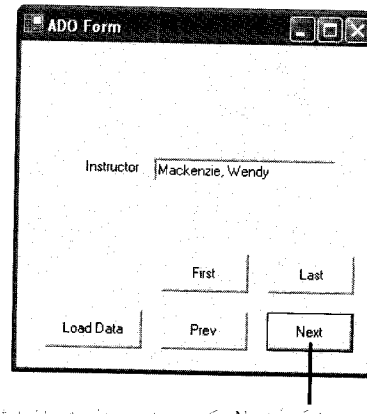
این دستور شبیه روال btnPrev_Click است، با این تفاوت که برای رفتن به رکورد بعدی دیتاست، یکی به شمارهٔ رکورد فعلی اضافه می‌کند.

این هم از دکمه‌های حرکت در دیتاست؛ اکنون

۱۱ برنامه را اجرا کنید.

۱۲ دکمهٔ Load Data را کلیک کنید، تا اولین رکورد دیتاست در جعبه متن نشان داده شود.

۱۳ برای رفتن به رکورد بعدی، دکمهٔ Next را کلیک کنید:



دکمهٔ Next رکورد بعدی دیتاست را نمایش می‌دهد

۱۴ دکمهٔ Prev را کلیک کنید، تا دوباره به اولین رکورد برگردید.

۱۵ چند بار روی دکمهٔ Next کلیک کرده، و رکوردهای مختلف را مشاهده کنید.

۱۶ دکمهٔ First را کلیک کنید، تا دوباره به اولین رکورد برگردید.

۱۷ دکمهٔ Last را کلیک کنید، تا به آخرین رکورد دیتاست بروید.

توجه کنید که اگر در آخرین رکورد باشید، و Next را کلیک کنید (و یا اگر در اولین رکورد باشید، و Prev را کلیک کنید)، هیچ خطایی رُخ نخواهد داد: شیء BindingContext خود با اینگونه خطاها مقابله خواهد کرد.

۱۸ با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

یک گام فراتر: نمایش شمارهٔ رکورد جاری

علاوه بر حرکت در میان رکوردهای دیتاست، شاید بخواهید در هر لحظه موقعیت کاربر را در رکوردها به وی نشان دهید. شمارهٔ رکورد فعلی دیتاست همیشه در خاصیت Position شیء BindingContext ذخیره می‌شود، که می‌توان برای این منظور از آن استفاده کرد.

در تمرین زیر، روالی بنام Count ایجاد خواهیم کرد، که در هر لحظه موقعیت کاربر در رکوردهای دیتاست را در یک برچسب نمایش می‌دهد.

نمایش موقعیت کاربر در دیتاست

۱ به فرم برنامهٔ ADO Form برگردید، و یک برچسب نسبتاً عریض زیر جعبه متن رسم کنید.

۲ خاصیت Name این برچسب را به **lblCount** ست کنید.

۳ خاصیت Text برچسب lblCount را به **Record 0 of 0** ست کنید.

٤ با کلیک کردن دکمهٔ View Code در کاوشگر راه‌حل، ادیتور کُد را باز کنید.

٥ به بالای پنجرهٔ کُد، زیر دستور "Windows Form Designer generated code" بروید.

٦ با نوشتن دستورات زیر، سابروتینی بنام Count ایجاد کنید:

```
Private Sub Count()
    Dim Records, Current As Integer
    Records = Me.BindingContext(DsInstructors1, "Instructors").Count
    Current = Me.BindingContext(DsInstructors1, "Instructors").Position + 1
    lblCount.Text = "Record " & Current.ToString & " of " & Records.ToString
End Sub
```

سابروتین Count مقدار خاصیت‌های BindingContext.Count و BindingContext.Position را بترتیب در دو متغیر بنامهای Records و Current می‌نویسد (علت اضافه کردن 1 به خاصیت Position اینست که شماره گذاری رکوردهای دیتاست از 0 شروع می‌شود)؛ سپس از این دو مقدار برای به روز در آوردن خاصیت Text برچسب lblCount استفاده می‌کند. فرمت این عبارت بصورت "Record *x* of *y*" است، که در آن *x* شمارهٔ رکورد جاری، و *y* تعداد کل رکوردهای دیتاست است.

پس از آن بایستی سابروتین Count را در تمام روالهای برنامه (که تعداد آنها پنج تاست) فراخوانی کنیم، تا برچسب lblCount همیشه به روز باشد.

٧ به روال رویداد btnFirst_Click بروید، و دستور زیر را به انتهای آن اضافه کنید:

```
Count()
```

اکنون این روال به شکل زیر است:

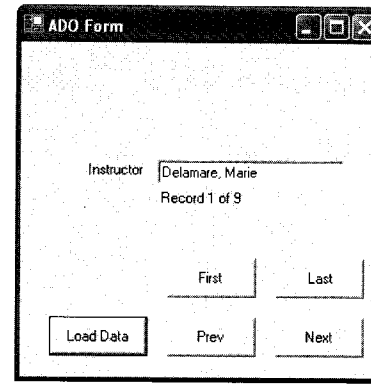
```
Private Sub btnFirst_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles btnFirst.Click
        Me.BindingContext(DsInstructors1, "Instructors").Position = 0
        Count()
End Sub
```

٨ این کار را برای تمام روالهای btnLast_Click ، btnPrev_Click ، btnNext_Click و btnLoad_Click تکرار کنید.

به همین سادگی!

٩ برنامه را اجرا کنید. (کُد کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap19\ado form بیابید.)

١٠ دکمهٔ Load Data را کلیک کنید، تا اولین رکورد دیتاست در جعبه متن نشان داده شود. همانطور که می‌بینید، شمارهٔ رکورد فعلی و تعداد کل رکوردها (بصورت "Record 1 of 9") زیر جعبه متن دیده می‌شود:



۱۱ چند بار روی دکمهٔ Next کلیک کرده، تا ببینید این برچسب همزمان با تغییر رکورد به روز در می‌آید.

۱۲ برای تست عملکرد سابروتین Count ، با دکمه‌های First ، Prev و Last نیز کار کنید.

۱۳ با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

این هم از ADO.NET ! با اینکه در مثالهای این فصل از یک پایگاه دادهٔ Access استفاده کردیم، اما روش کار برای سایر انواع پایگاه داده (حتی آنهایی که روی اینترنت قرار دارند) نیز یکسان است، که علت آن در ساختار توزیع‌یافتهٔ ADO.NET نهفته است. البته اگر می‌خواهید برای اینترنت برنامه بنویسید، باید از **فرمهای وب** (که در فصل ۲۲ به آن خواهیم پرداخت) استفاده کنید - اما در هر حال، تکنیکهای کار با پایگاه داده تفاوت چندانی نخواهد کرد.

مرجع سریع فصل ۱۹

برای ...	انجام دهید
برقراری اتصال به یک پایگاه داده	با فرمان View|Server Explorer پنجرهٔ کاوشگر میزبان را باز کرده، و بعد از کلیک کردن آیکون Connect to Database ، پایگاه دادهٔ موردنظر را انتخاب کنید.
ایجاد یک آداپتور داده	به برگهٔ Data در جعبه ابزار ویژوال بیسیک بروید، و یک کنترل OleDbDataAdapter یا SqlDataAdapter روی فرم برنامه کشیده، و سپس به کمک «جادوگر پیکربندی آداپتور داده» آنرا پیکربندی کنید.
ایجاد یک دیتاست	فرمان Data|Generate Dataset را اجرا کرده، و نام دیتاست را وارد کنید. (مطمئن شوید که جعبه چک Add this dataset to the designer فعال است.)
برقراری پیوند بین کنترل‌ها و دیتاست	بعد از قرار دادن کنترلهای موردنظر روی فرم، خاصیت Text گروه DataBindings هر کنترل را به فیلد دلخواه در دیتاست ست کنید.
پُر کردن دیتاست با داده و نمایش آنها در کنترلهای پیوندی	دستور زیر را در محلی که داده‌ها باید در دیتاست قرار گیرند، بنویسید: OleDbDataAdapter1.Fill(DsInstructors1)
حرکت در رکوردهای دیتاست	کنترلهای لازم را روی فرم برنامه قرار داده، و در رویداد Click آنها از خاصیت BindingContext.Position استفاده کنید. برای مثال، دستور زیر رکورد بعدی دیتاست را نمایش می‌دهد: Me.BindingContext(DsInstructors1, "Instructors") _ .Position += 1

فصل

۲۰

نمایش داده‌ها با کنترل شبکهٔ داده (DataGrid)

در این فصل یاد می گیرید چگونه :

- ✔ برای نمایش رکوردهای یک پایگاه داده از کنترل شبکهٔ داده (DataGrid) استفاده کنید.
- ✔ رکوردهای پایگاه داده را بر مبنای فیلدها (ستونها) مرتب کنید.
- ✔ فرمت و رنگ سلولهای کنترل شبکهٔ داده را تغییر دهید.
- ✔ تغییرات اِعمال شده در سلولهای شبکهٔ داده را در پایگاه داده بنویسید.

در فصل قبل با اصول اولیهٔ برنامه‌نویسی ADO.NET آشنا شدید، و دیدید که چگونه می‌توان اطلاعات یک پایگاه دادهٔ Access را بازیابی کرد و نمایش داد. برنامه‌ای که نوشتیم، توانایی حرکت در میان رکوردهای پایگاه داده را نیز داشت. در این فصل به کار با کلاسها و اشیاء ADO.NET ادامه می‌دهیم، و خواهید دید که چگونه می‌توان به کمک کنترل شبکهٔ داده تعداد زیادی از رکوردها و فیلدهای یک پایگاه داده را به یکباره روی فرم برنامه نمایش داد.

تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- ویژوال بیسیک ۶ انواع مختلفی از کنترلهای شبکه‌ای برای نمایش داده‌ها داشت، از جمله FlexGrid ، Hierarchical FlexGrid ، و DataGrid . در ویژوال بیسیک .NET فقط یک کنترل برای نمایش صفحه‌ای رکوردهای پایگاه داده وجود دارد، که نام آن DataGrid است.
- کنترل DataGrid ویژوال بیسیک .NET تفاوت زیادی با همتای خود در ویژوال بیسیک ۶ دارد. مهمترین تفاوت آنها اینست که در این کنترل دیگر دستورات بازیابی اطلاعات وجود ندارد، چون این وظیفه به لایه‌های پائینتر (آداپتور داده و دیتاست) محول شده است. بسیاری از خواص و متدهای این کنترل نیز فرق کرده است، که برای اطلاع از این تغییرات می‌توانید به سیستم کمک ویژوال بیسیک .NET مراجعه کنید.

استفاده از کنترل DataGrid برای نمایش رکوردهای پایگاه داده

کنترل شبکهٔ داده (DataGrid) اطلاعات پایگاه داده را بصورت جدول (همانطور که در برنامه‌های Microsoft Access یا Microsoft Excel دیده‌اید) نمایش می‌دهد. کنترل شبکهٔ داده می‌تواند هر نوع اطلاعاتی را که ذاتاً بصورت جدول باشد، نمایش دهد، ولی ما در این فصل از آن برای نمایش اطلاعات پایگاه دادهٔ Students.mdb (که در فصل ۱۹ آنرا دیدید) استفاده خواهیم کرد. ابتدا این کنترل را با متن سادهٔ اطلاعات پایگاه داده پُر می‌کنیم، و بعد از آن سراغ فرمت کردن شبکهٔ داده خواهیم رفت. در ادامهٔ نحوهٔ مرتب کردن رکوردها، و ذخیره کردن تغییرات اِعمال شده در پایگاه داده را نیز یاد خواهید گرفت.

کنترل شبکهٔ داده برای ارتباط (پیوند) با پایگاه داده پایگاه داده از خواص DataSource و DataMember استفاده می‌کند. (البته این دو خاصیت فقط بعد از ایجاد آداپتور داده و دیتاست دارای اطلاعات معتبر خواهند شد.) بعد از آن که شبکهٔ داده به منبع داده متصل شد، با متد Fill شیء آداپتور داده می‌توان داده‌ها را در آن نوشت:

```
OleDbDataAdapter1.Fill(DsInstructors1)
```

در تمرین زیر، جدول Instructors پایگاه دادهٔ Students.mdb را در یک کنترل شبکهٔ داده نمایش می‌دهیم.

برقراری اتصال به جدول Instructors

۱ در محیط ویژوال استودیو .NET ، یک پروژهٔ جدید Visual Basic Windows Application بنام **My DataGrid Sample** در پوشهٔ c:\vbnetsbs\chap20 ایجاد کنید.

۲ از منوی View فرمان Server Explorer را انتخاب کنید، تا پنجرهٔ کاوشگر میزبان باز شود. اگر تازه تمرین فصل ۱۹ را تمام کرده‌اید، هنوز می‌توانید اتصال به پایگاه دادهٔ Students.mdb را در گره Data Connections ببینید. (اگر در کنار این اتصال یک X قرمز می‌بینید، یعنی اتصال فعلاً قطع است؛ برای برقراری مجدد این اتصال، کافیست روی گره ACCESS کلیک کنید.)

توجه

اگر تمرین فصل قبل را انجام داده‌اید، و اتصال به پایگاه دادهٔ Students.mdb را ایجاد کرده‌اید، دیگر نیازی نیست مراحل ۳ تا ۷ را انجام دهید (به مرحلهٔ ۸ بروید).

۳ در کاوشگر میزبان، روی دکمهٔ Connect to Database کلیک کنید، تا پنجرهٔ خواص لینک داده (Data Link Properties) باز شود.

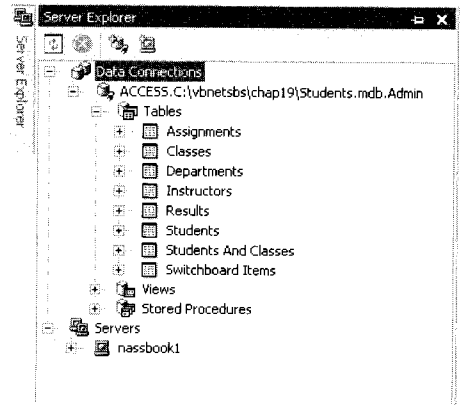
٤ به برگهٔ Provider بروید، و آیتم Microsoft Jet 4.0 OLE DB Provider را انتخاب کنید.

۵ روی دکمهٔ Next کلیک کنید، تا به برگهٔ Connection بروید. در اینجا باید نام و محل پایگاه دادهٔ موردنظر را وارد کنید.

۶ روی دکمهٔ ... کنار فیلد Select or enter a database name کلیک کرده، و بعد از انتخاب پایگاه دادهٔ Students.mdb (که در پوشهٔ c:\vbnetsbs\chap19 قرار دارد)، Open را کلیک کنید.

۷ دکمهٔ OK پنجرهٔ Data Link Properties را کلیک کنید. به گره (آیتم) جدیدی که در کاوشگر میزبان ایجاد می‌شود، دقت کنید.

۸ در کاوشگر میزبان، بترتیب، گره‌های Data Connections ، ACCESS و Tables را باز کنید. (برای باز کردن یک گره، کافیست روی علامت + کنار آن کلیک کنید.) همانطور که می‌بینید، کاوشگر میزبان ساختار پایگاه دادهٔ Students.mdb و اشیاء آن (جدول‌ها، فیلدها و غیره) را نشان می‌دهد:



۹ در برگهٔ Data جعبه ابزار ویژوال بیسیک، کنترل OleDbDataAdapter را گرفته، و روی فرم برنامه بکشید؛ به محض رها کردن کنترل OleDbDataAdapter روی فرم برنامه، ویژوال استودیو «جادوگر پیکربندی آداپتور داده» را شروع می‌کند.

۱۰ در صفحهٔ اول دکمهٔ Next را کلیک کنید.

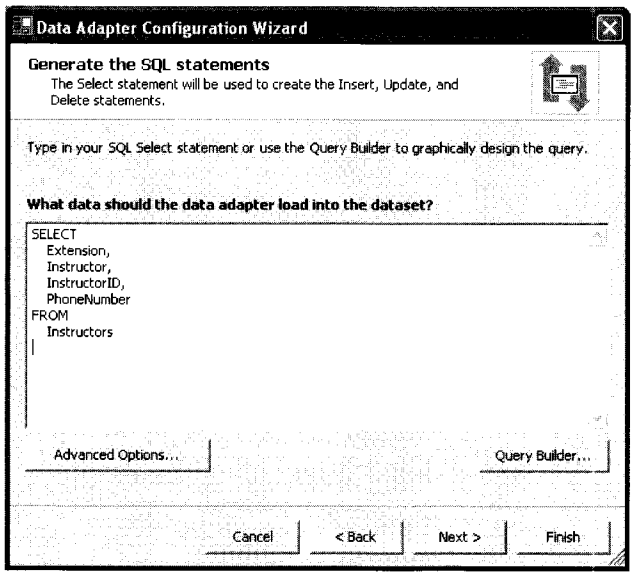
۱۱ مطمئن شوید که پایگاه دادهٔ Students.mdb در این پنجره انتخاب شده، سپس Next را کلیک کنید.

۱۲ در پنجرهٔ بعدی اولین گزینه، یعنی SQL Statements ، را انتخاب کرده، و Next را کلیک کنید. با این انتخاب در پنجرهٔ بعد باید یک دستور SQL SELECT وارد کنید.

۱۳ دستور SELECT زیر را در قسمت What data should the data adapter load into the dataset? وارد کنید:

```
SELECT
    Extension,
    Instructor,
    InstructorID,
    PhoneNumber
FROM
    Instructors
```

این دستور فیلدهای Extension ، Instructor ، InstructorID ، و PhoneNumber از جدول Instructors پایگاه دادهٔ Students.mdb را در آداپتور داده بار می‌کند:



۱۴ دکمهٔ Next را کلیک کنید، تا به صفحهٔ پایانی جادوگر برسید. دقت کنید که تمام دستورات SQL لازم (SELECT ، INSERT ، UPDATE ، و DELETE) بدرستی ایجاد شده باشند.

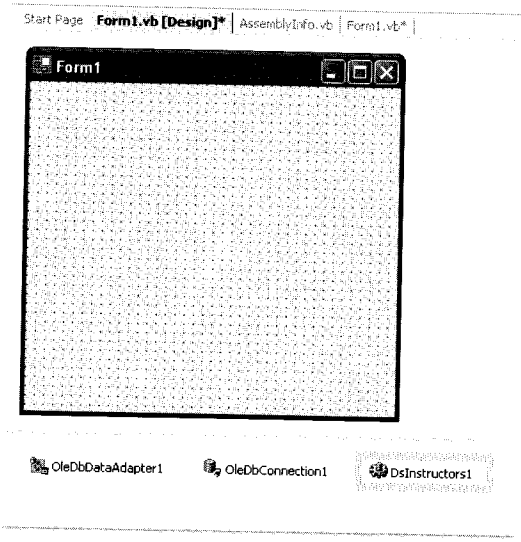
۱۵ برای پایان عملیات، دکمهٔ Finish را کلیک کنید.

اکنون باید شیء دیتاست را ایجاد کنیم.

۱۶ روی فرم برنامه کلیک کنید تا فعال شود، و سپس از منوی Data فرمان Generate Dataset را انتخاب کنید، تا پنجرهٔ «ایجاد دیتاست» باز شود.

۱۷ در فیلد New ، نام دیتاست جدید را **DsInstructors** وارد کنید. همچنین دقت کنید که جعبه چک Add this dataset to the designer فعال باشد، تا ویژوال استودیو این دیتاست را به سینی اجزاء اضافه کند.

۱۸ OK را کلیک کنید، تا دیتاست DsInstructors1 ایجاد و به سینی اجزاء برنامه اضافه شود:



اکنون آماده‌ایم تا رکوردهای جدول Instructors را در کنترل شبکهٔ داده نمایش دهیم.

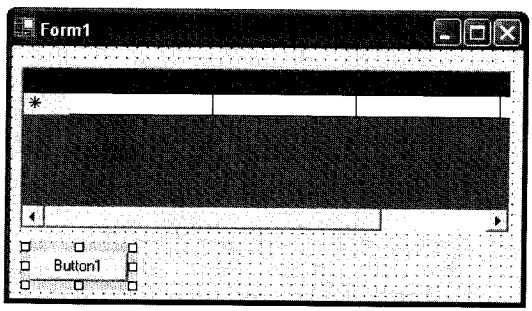
ایجاد شیء شبکهٔ داده

۱ فرم برنامه را آنقدر بزرگ کنید، تا جای تقریباً کافی برای نمایش چهار ستون و ده سطر اطلاعات داشته باشد.

۲ در برگهٔ Windows Forms جعبه ابزار ویژوال بیسیک، روی کنترل DataGrid کلیک کنید.

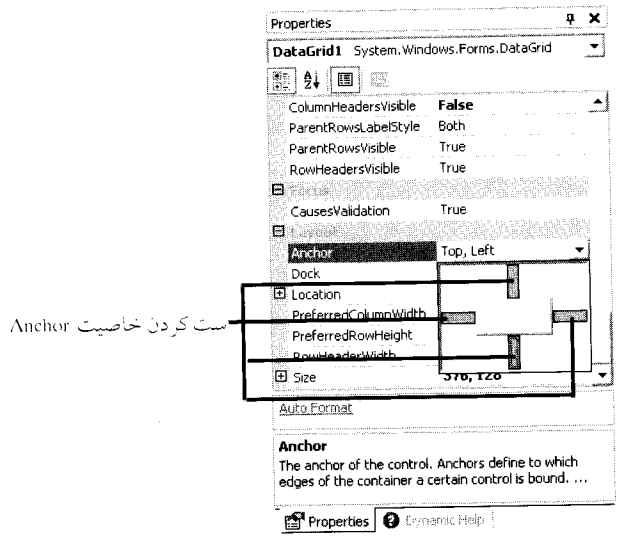
۳ یک کنترل شبکهٔ دادهٔ بزرگ روی فرم رسم کنید.

٤ یک دکمه زیر کنترل شبکهٔ داده قرار دهید (شکل زیر را نگاه کنید).



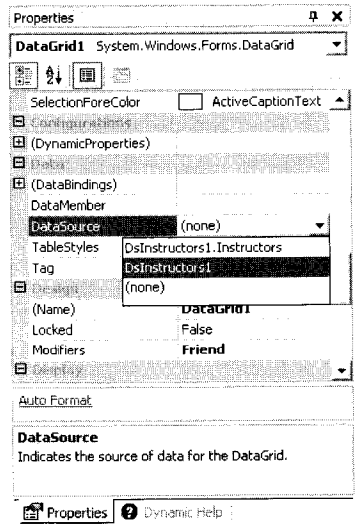
۵ کنترل شبکهٔ دادهٔ را انتخاب کرده، سپس پنجرهٔ خواص را باز کنید.

۶ خاصیت Anchor شبکهٔ دادهٔ را به هر چهار لبهٔ فرم ست کنید:



۷ خاصیت DataSource شبکهٔ دادهٔ را انتخاب کنید.

۸ از لیست مقابل این خاصیت، دیتاست DsInstructors1 (نه DsInstructors1.Instructors) را انتخاب کنید (شکل زیر را ببینید).

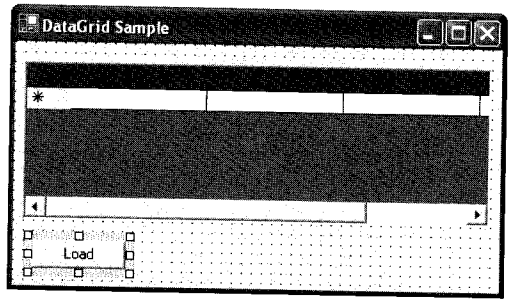


۹ خاصیت DataMember شبکهٔ دادهٔ را انتخاب کرده، و از لیست مقابل آن آیتم Instructors را انتخاب کنید. به محض تعیین خاصیت DataMember ، فیلدهای مشخص شده در دیتاست

Instructors در بالای کنترل شبکهٔ داده ظاهر می‌شوند (یک سطر خالی هم در زیر آن باز خواهد شد). البته بار شدن اطلاعات تا زمان اجرای برنامه انجام نخواهد شد.

۱۰ فرم برنامه را انتخاب کرده، و خاصیت Text آنرا به **DataGrid Sample** ست کنید.

۱۱ دکمهٔ Button1 را انتخاب کنید. خاصیت‌های Anchor ، Name و Text این کنترل را بترتیب به **(Bottom, Left)** ، **btnLoad** و **Load** ست کنید:



۱۲ روی دکمهٔ Load دو-کلیک کنید، تا روال رویداد btnLoad_Click در ادیتور کُد باز شود.

۱۳ کُد زیر را در این روال بنویسید:

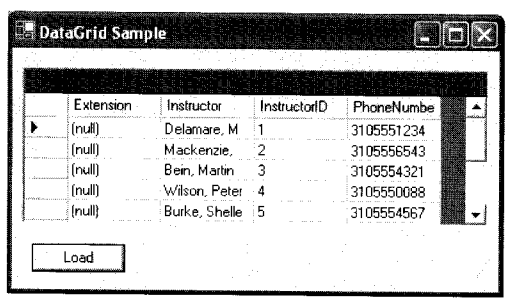
```
DsInstructors1.Clear()
OleDbDataAdapter1.Fill(DsInstructors1)
```

این کُد ابتدا اطلاعات موجود در شیء DsInstructors1 را پاک کرده (تا اگر دکمهٔ Load بیش از یک بار کلیک شد، اطلاعات تکراری در دیتاست DsInstructors1 وجود نداشته باشد)، و سپس آداپتور داده را با اطلاعات جدید پُر می‌کند.

۱۴ با کلیک کردن دکمهٔ Save All ، پروژه را ذخیره کنید.

۱۵ برنامه را اجرا کنید.

۱۶ دکمهٔ Load را کلیک کنید، تا اطلاعات دیتاست در شبکهٔ داده بار شود:



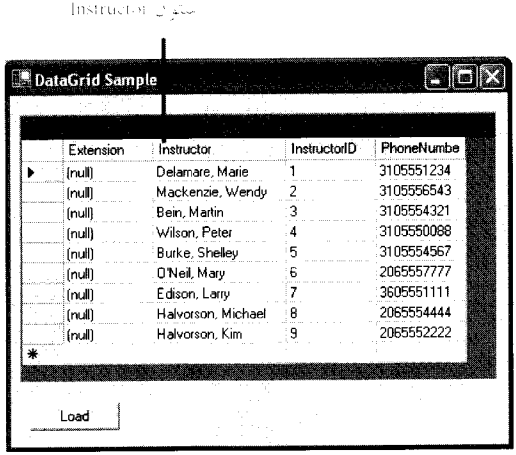
هر سطر شبکهٔ داده یک رکورد جدول Instructors را نمایش می‌دهد. در این مثال ما تمام فیلدهای جدول را نشان داده‌ایم، اما می‌توانید با تغییر دادن دستور SELECT آنها را محدود کنید. دقت کنید

که ترتیب نمایش فیلدها (ستونها) در شبکهٔ داده با ترتیب نوشتن آنها در دستور SELECT یکسان است (برای تغییر دادن این ترتیب، می‌توانید نام فیلدها را در دستور SELECT جابجا کنید). همچنین توجه کنید که، چون تعداد رکوردها از ارتفاع شبکهٔ داده بیشتر است، یک میلهٔ لغزشی (Scroll Bar) در سمت راست آن ظاهر شده است.

۱۷ برای دیدن بقیهٔ رکوردها، از میلهٔ لغزشی استفاده کنید.

۱۸ فرم برنامه را بزرگ کنید. از آنجائیکه خاصیت Anchor شبکهٔ داده را ست کرده‌ایم، با بزرگ شدن فرم این کنترل هم بزرگ می‌شود، بطوریکه می‌توانید تمام رکوردها را در آن واحد ببینید.

اگر پهنای یکی از ستونها کم است و اطلاعات آن فیلد بطور کامل دیده نمی‌شود، می‌توانید با کشیدن خط فاصل عمودی آن فیلد به سمت راست، پهنای آنرا بیشتر کنید. در شکل زیر پهنای ستون Instructor را زیاد کرده‌ایم:



یکی از مزایای بسیار جالب کنترل شبکهٔ داده اینست که، می‌توان با آن اطلاعات را بطور خودکار مرتب کرد.

۱۹ روی عنوان ستون Instructor کلیک کنید، تا اطلاعات این ستون بر حسب حروف الفبا مرتب شود. مثلث کوچکی که در سمت راست این ستون ظاهر می‌شود، نشان می‌دهد که رکوردها بر اساس این ستون مرتب شده‌اند (اگر نوک این مثلث رو به بالا باشد، یعنی مرتب‌سازی بصورت صعودی – A-Z – انجام شده؛ اگر نوک این مثلث رو به پائین باشد، یعنی مرتب‌سازی بصورت نزولی – Z-A – انجام شده است). هر بار که روی این مثلث کلیک کنید، جهت آن (و ترتیب مرتب‌سازی) عوض خواهد شد.

نکته

کنترل DataGrid فقط وقتی اجازهٔ مرتب کردن داده‌ها را می‌دهد، که خاصیت AllowSorting آن به True ست شده باشد.

۲۰ چند بار دیگر روی ستون Instructor کلیک کرده، و جهت مرتب‌سازی را عوض کنید.

۲۱ روی عنوان ستونهای دیگر (مانند InstructorID یا PhoneNumber) کلیک کنید، تا رکوردها بر حسب این ستونها مرتب شوند.

۲۲ با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

فرمت کردن سلولهای شبکهٔ داده

بسیاری از خواص ظاهری سلولهای شبکهٔ داده را می‌توان کنترل کرد. مثلاً، می‌توانید ارتفاع و پهنای پیش‌فرض سلولها را تغییر داده، عنوان ستونها را حذف کرده و یا رنگ سلولها را تغییر دهید، و یا رنگ خطوط شبکه را عوض کنید. در تمرین زیر با روش فرمت کردن سلولهای شبکهٔ داده و اطلاعات آن آشنا می‌شوید.

ست کردن خواص ظاهری شبکهٔ داده

۱ فرم برنامه را باز کرده، و بعد از انتخاب کنترل شبکهٔ داده، پنجرهٔ خواص را باز کنید.

۲ خاصیت PreferredColumnWidth را به **110** ست کنید (واحد این خاصیت پیکسل است).

۳ خاصیت ColumnHeadersVisible را به **False** ست کنید. با این کار نام فیلدها از بالای ستونها ناپدید خواهد شد. (این کار بویژه وقتی سودمند است که نام فیلدها، که اغلب هم بدرستی انتخاب نمی‌شوند، اطلاعات خاصی برای کاربر در بر نداشته باشد.)

٤ لیست مقابل خاصیت BackColor را باز کرده، و بعد از رفتن به برگهٔ Custom ، رنگ زرد را برای زمینهٔ سلولهای شبکهٔ داده انتخاب کنید.

انتخاب این خاصیت باعث می‌شود تا رنگ سطرها بصورت یک در میان بین سفید و رنگ انتخاب شده عوض شود، و یک طرح راه راه بوجود آورد -که برای تمایز بین سطرها بسیار مفید است. (توجه: رنگی که در اطراف و زیر سلولهای شبکهٔ داده دیده می‌شود، مربوط به خاصیت BackgroundColor است.) رنگ پیش‌فرض فونت سلولها سیاه است -اگر رنگ زمینهٔ سلولها را عوض می‌کنید، دقت کنید که رنگ فونت با آن متناسب باشد.

نکته

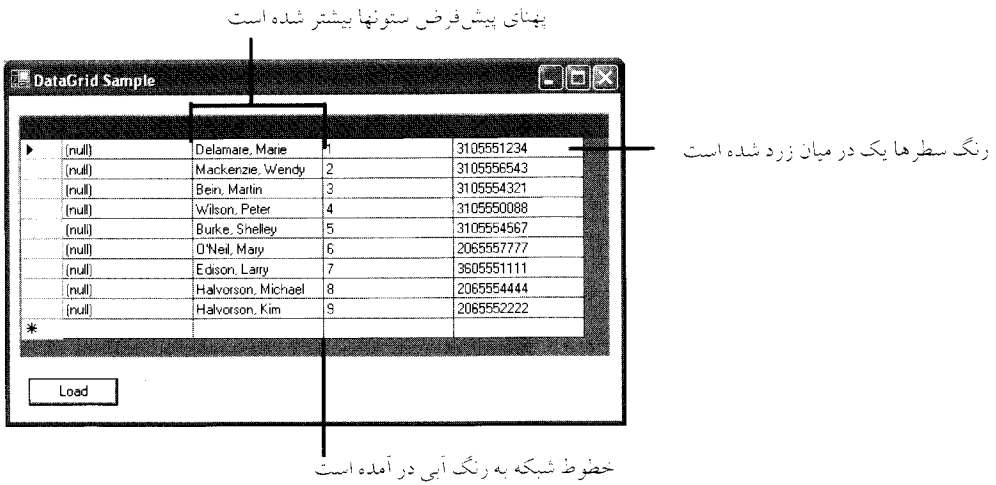
برای تغییر دادن رنگ زمینهٔ سلولهای عنوان، از خاصیت HeaderBackColor استفاده کنید.

۵ لیست مقابل خاصیت GridLineColor را باز کرده، و بعد از رفتن به برگهٔ Custom ، رنگ آبی را برای خطوط شبکهٔ داده انتخاب کنید. سعی کنید این رنگ با رنگ زمینهٔ سلولها متناسب باشد.

۶ برنامه را اجرا کنید.

۷ دکمهٔ Load را کلیک کنید، تا اطلاعات دیتاست در شبکهٔ داده بار شود.

۸ فرم برنامه را بزرگ کنید، تا بتوانید تمام رکوردها را در آن واحد ببینید:



▶	(null)	Delamare, Marie	1	3105551234
	(null)	Mackenzie, Wendy	2	3105556543
	(null)	Bein, Martin	3	3105554321
	(null)	Wilson, Peter	4	3105550088
	(null)	Burke, Shelley	5	3105554567
	(null)	O'Neil, Mary	6	2065557777
	(null)	Edison, Larry	7	3605551111
	(null)	Halvorson, Michael	8	2065554444
	(null)	Halvorson, Kim	9	2065552222
*				

همانطور که می‌بینید، عنوان ستونها ناپدید شده، ولی در عین حال پهنای هر ستون بیشتر شده است. همچنین به رنگ زرد و سفید سطرها (که بصورت یک در میان عوض می‌شود)، و رنگ آبی خطوط شبکه توجه کنید. (متأسفانه این رنگها را در کتاب نمی‌بینید، اما روی صفحهٔ مانیتور خواهید دید!)

۹ با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

در پنجرهٔ خواص کنترل شبکهٔ داده خواص بسیار زیادی را خواهید دید، که به کمک آنها می‌توانید ظاهر آنرا مطابق با سلیقه و نیاز خود تغییر دهید.

یک گام فراتر: به روز در آوردن پایگاه دادهٔ اصلی

همانطور که قبلاً هم گفته‌ام، شیء دیتاست در واقع فقط یک کپی از اطلاعات پایگاه داده است، و تغییراتی که در آن می‌دهید مستقیماً در پایگاه داده نوشته نخواهد شد (با این تمهید، طراحان ویژوال استودیو پایگاه داده را از خطر تغییرات سهوی در امان نگه داشته‌اند). کنترل شبکهٔ داده هم از این قاعده مستثنی نیست، و اگر می‌خواهید تغییرات اعمال شده در آن در پایگاه داده منعکس شود، باید آداپتور داده را وادار به این کار کنید.

در تمرین زیر، خاصیت ReadOnly شبکهٔ داده را بررسی خواهیم کرد، خاصیتی که تغییر داده‌های شبکهٔ داده را مجاز یا ممنوع می‌کند؛ با متد Update (که تغییرات انجام شده در شبکهٔ داده را در پایگاه داده می‌نویسد) نیز آشنا خواهید شد.

نوشتن تغییرات در پایگاه داده

۱ فرم برنامه را باز کرده، و بعد از انتخاب کنترل شبکهٔ داده، پنجرهٔ خواص را باز کنید.

۲ خاصیت ReadOnly را انتخاب کنید.

اگر این خاصیت False باشد (مقدار پیش‌فرض)، کاربر می‌تواند اطلاعات موجود در شبکهٔ داده را

دستکاری کند. ولی اگر میل ندارید به کاربر اجازهٔ تغییر دادن اطلاعات را بدهید، این خاصیت را True کنید. از آنجائیکه می‌خواهیم روش نوشتن تغییرات در پایگاه دادهٔ اصلی را بررسی کنیم، این خاصیت را False نگه دارید.

۳ یک دکمهٔ جدید کنار دکمهٔ Load (و سمت راست آن) رسم کنید. این دکمه‌ایست که تغییرات DataGrid را در پایگاه دادهٔ Students.mdb خواهد نوشت.

٤ خاصیت‌های Anchor ، Name و Text این دکمه را بترتیب به **(Bottom, Left)** ، **btnUpdate** و **Update** ست کنید.

۵ روی دکمهٔ Update دو-کلیک کنید، تا روال رویداد btnUpdate_Click در ادیتور کُد باز شود. دستورات زیر را در این روال بنویسید:

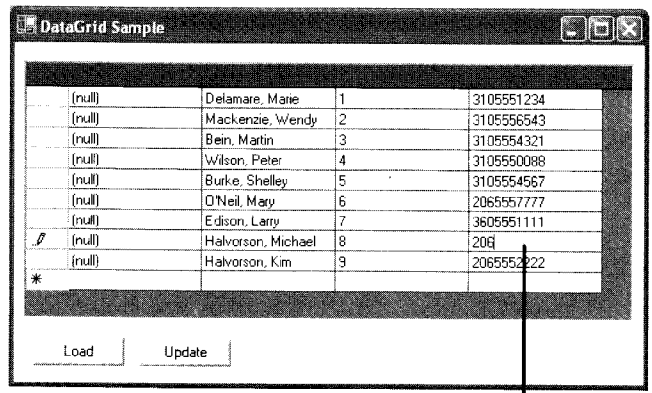
```
Try
    OleDbDataAdapter1.Update(DsInstructors1)
Catch ex As Exception
    MsgBox(ex.ToString)
End Try
```

این کُد با استفاده از متد Update ، آداپتور دادهٔ OleDbDataAdapter1 را وادار می‌کند تا محتویات (جدید) دیتاست DsInstructors1 را در پایگاه دادهٔ اصلی بنویسد.

۶ برنامه را اجرا کنید. (کُد کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap20\datagrid sample بیابید.)

۷ دکمهٔ Load را کلیک کنید، تا اطلاعات جدول Instructors پایگاه دادهٔ Students.mdb در شبکهٔ داده بار شود.

۸ فرم برنامه را بزرگ کنید، تا بتوانید تمام رکوردهای جدول را در آن واحد ببینید. یکی از سلولهای شبکهٔ داده را با ماوس انتخاب کرده، کلید Delete را بزنید، و سپس مقدار جدیدی وارد کنید. دقت کنید که در حین تغییر دادن محتویات سلول، یک آیکون مداد در سمت چپ آن سطر ظاهر می‌شود، که نشان می‌دهد این رکورد در حال دستکاری است:



سلول تغییر کرده در شبکهٔ داده

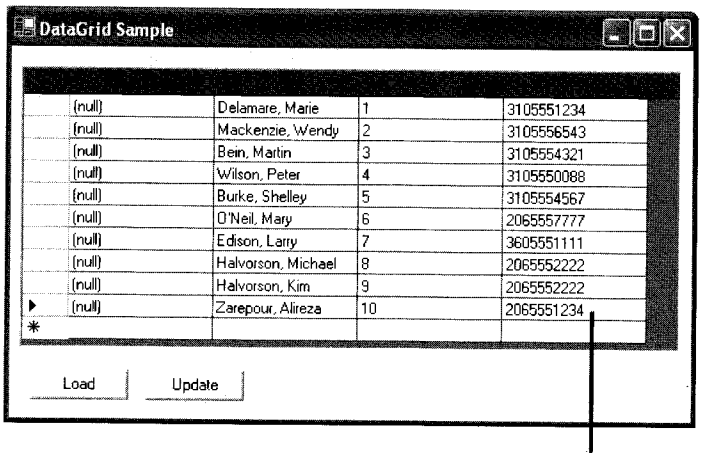
برای ثبت این تغییرات در دیتاست DsInstructors1 ، روی یک سلول دیگر کلیک کنید.

۹ دکمهٔ Update را کلیک کنید، تا تغییرات انجام شده از آداپتور داده به پایگاه دادهٔ Students.mdb منتقل شده، و در آنجا ثبت شود.

۱۰ در یکی از سلولهای سطر آخر، سطری که علامت ستاره (*) دارد، کلیک کنید.

۱۱ سلولهای این سطر را پُر کنید؛ اکنون سطر جدیدی در شبکهٔ داده ایجاد شده است.

۱۲ دکمهٔ Update را کلیک کنید، تا این رکورد جدید برای همیشه در پایگاه‌دادهٔ Students.mdb نوشته شود.



سطر جدید در شبکهٔ داده

۱۳ با کلیک کردن روی آیکون ▶ در سمت چپ این سطر، آنرا انتخاب کرده و کلید Delete را بزنید: سطر از شبکهٔ داده حذف می‌شود.

۱۴ دکمهٔ Load را کلیک کنید: سطر حذف شده دوباره ظاهر می‌شود! چرا؟ مگر همین چند لحظه پیش آنرا حذف نکردیم؟ عملی که ما انجام دادیم، در واقع حذف سطر مزبور از دیتاست DsInstructors1 بود، ولی این رکورد همچنان در پایگاه دادهٔ Students.mdb باقی مانده است. برای حذف کامل این سطر باید دکمهٔ Update را هم کلیک می‌کردیم، تا تغییرات در پایگاه دادهٔ اصلی نیز نوشته می‌شد.

۱۵ کمی با برنامه کار کنید و سپس، با کلیک کردن دکمهٔ Close ، آنرا ببندید.

مرجع سریع فصل ۲۰

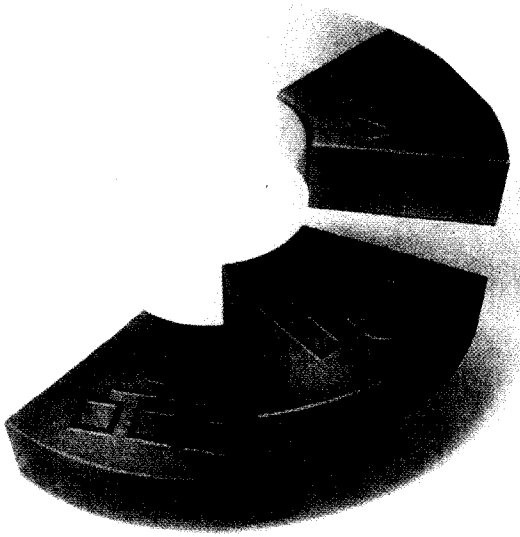
برای ...	انجام دهید
قرار دادن یک شبکهٔ داده روی فرم	از برگهٔ Windows Forms جعبه ابزار ویژوال بیسیک، کنترل DataGrid را انتخاب کرده، و یک شبکهٔ داده روی فرم برنامه رسم کنید.
اتصال شبکهٔ داده به یک دیتاست	خاصیت DataSource کنترل DataGrid را به دیتاست موردنظر، و خاصیت DataMember آنرا به یکی از جدولهای این دیتاست ست کنید.
پُر کردن شبکهٔ داده با اطلاعات دیتاست	از متد Fill آداپتور داده استفاده کنید: OleDbDataAdapter1.Fill(DsInstructors1)
مرتب کردن رکوردهای شبکهٔ داده	روی عنوان ستون موردنظر کلیک کنید، تا رکوردها بر حسب آن فیلد مرتب شوند.
عوض کردن جهت مرتب‌سازی	یک بار دیگر روی عنوان ستون موردنظر کلیک کنید، تا جهت داده‌ها مرتب‌سازی (A-Z یا Z-A) عوض شود.
جلوگیری از مرتب شدن داده‌ها	خاصیت AllowSorting کنترل شبکهٔ داده را False کنید.
تغییر دادن پهنای پیش‌فرض ستونهای شبکهٔ داده	خاصیت PreferredColumnWidth کنترل شبکهٔ داده را به مقدار موردنظر ست کنید.
تغییر دادن رنگ زمینهٔ سلولهای شبکهٔ داده	خاصیت BackColor کنترل شبکهٔ داده را به رنگ موردنظر ست کنید.
جلوگیری از تغییر در شبکهٔ داده	خاصیت ReadOnly کنترل شبکهٔ داده را True کنید.
نوشتن تغییرات اِعمال شده در شبکهٔ داده در پایگاه داده	برای نوشتن اطلاعات دیتاست موردنظر در پایگاه دادهٔ اصلی، از متد Update آداپتور داده استفاده کنید: OleDbDataAdapter1.Update(DsInstructors1)

آموزش گام به گام

بخش ۶

VISUAL BASIC .NET

برنامه‌نویسی اینترنت



فصل

۲۱

نمایش فایلهای HTML با اینترنت اکسپلورِر

در این فصل یاد می گیرید چگونه :

- ✔ مدل شیء اینترنت اکسپلورر (Microsoft Internet Explorer) را بررسی کنید.
- ✔ فایلهای HTML را در برنامهٔ خود ببینید.
- ✔ از رویدادهای اینترنت اکسپلورر استفاده کنید.

در فصل ۳ دیدید که چگونه می‌توان با استفاده از کنترل برچسب‌لینک و به کمک متد System.Diagnostics.Process.Start یک صفحهٔ وب را باز کرد. در این بخش (که شامل دو فصل است) توجه خود را روی ارتباط با اینترنت در برنامه‌های ویژوال بیسیک .NET متمرکز خواهیم کرد. در این فصل با رویدادها، خواص و متدهای برنامهٔ اینترنت اکسپلورر آشنا می‌شوید، یاد می‌گیرید که چگونه با کنترلهای Web Forms ویژوال بیسیک یک برنامهٔ وب بنویسید، و فایلهای HTML را نمایش دهید.

برای این منظور از خواص، متدها و رویدادهای شیء اینترنت اکسپلورر (که در تمام کامپیوترهایی که ویندوز روی آنها نصب شده، وجود دارد) استفاده خواهیم کرد. مزیت استفاده از این شیء آنست که برای نمایش فایلهای HTML (حتی پیچیده‌ترین آنها)، نیازی نیست یک کاوشگر وب (Web Browser) را از نو بنویسیم.

تازه‌های ویژوال بیسیک .NET

اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- ویژوال بیسیک ۶ با اینترنت اکسپلورر ۴ به بازار آمد، و ویژوال بیسیک .NET با اینترنت اکسپلورر ۶. اما این دو ویرایش سازگاری زیادی با هم دارند؛ یعنی براحتی می‌توانید برنامه‌های ویژوال بیسیک ۶ را که از این شیء استفاده می‌کنند، با ویژوال بیسیک .NET کامپایل کنید.
- از آنجائیکه کتابخانهٔ Microsoft Internet Controls - فایل SHDocVw.dll - از تکنولوژی COM استفاده می‌کند، ویژوال بیسیک .NET یک لفاف (wrapper) ویژه به دور آن ایجاد می‌کند، تا کلاسها و انواع دادهٔ این کتابخانه را در اختیار شما بگذارد.

آشنایی با مدل شیء اینترنت اکسپلورر

میکروسافت اینترنت اکسپلورر یک کاوشگر همه کاره است، که با آن می‌توانید فایلهای HTML را براحتی مشاهده کنید. این برنامه بگونه‌ای طراحی شده، که می‌توان از آن بصورت برنامه‌ای مستقل، و یا بصورت یک شیء در دل برنامه‌های دیگر استفاده کرد. برای این منظور، اینترنت اکسپلورر خواص، متدها و رویدادهای خود را بصورت یک کلکسیون در اختیار دیگران می‌گذارد. برای بررسی امکانات این شیء می‌توانید از کاوشگر شیء (Object Browser) ویژوال استودیو استفاده کنید.

شیء اینترنت اکسپلورر جزء ابزارهای ویژوال بیسیک .NET نیست، بلکه آن یک کتابخانهٔ COM است که در تمام سیستمهایی که برنامهٔ اینترنت اکسپلورر در آنها نصب شده (و این یعنی، تقریباً تمام کامپیوترهایی که سیستم عامل آنها ویندوز است)، یافت می‌شود.

بسیار مهم

در تمرین‌های این فصل از اینترنت اکسپلورر ۶ (ویرایشی که با ویژوال بیسیک .NET می‌آید) استفاده خواهیم کرد. با اینکه ویژگیهای اصلی این برنامه در ویرایشهای مختلف تا حد زیادی یکسان باقی مانده است، اما برای پرهیز از هرگونه مشکل، ویرایش اینترنت اکسپلورر سیستم خود را (از طریق فرمان Help|About این برنامه) چک کنید. اگر ویرایش آن ۶ نیست، به کمک کاوشگر شیء ویژوال استودیو مطمئن شوید که خواص، متدها و رویدادهای مورد استفاده در تمرینهای این فصل را داشته باشد.

اضافه کردن ارجاع اینترنت اکسپلورر

اولین قدم در استفاده از شیء اینترنت اکسپلورر، اضافه کردن رفرنس (reference) کتابخانهٔ آن به برنامه است؛ در تمرین زیر روش این کار را ملاحظه خواهید کرد.

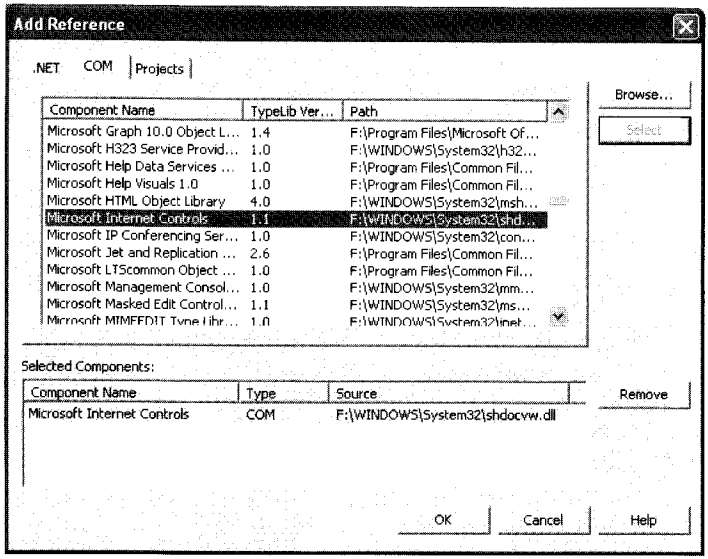
ضمیمه کردن شیء اینترنت اکسپلورر به پروژه

۱ در محیط ویژوال استودیو .NET، یک پروژهٔ جدید Visual Basic Windows Application بنام **My Explorer Objects** در پوشهٔ c:\vbnetsbs\chap21 ایجاد کنید.

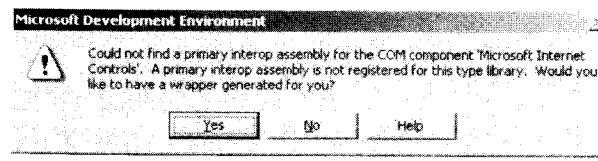
۲ از منوی Project فرمان Add Reference را اجرا کنید.

۳ در دیالوگ Add Reference ، به برگهٔ COM بروید.

٤ آیتم Microsoft Internet Controls را انتخاب کرده، و سپس دکمهٔ Select را کلیک کنید (شکل زیر را ببینید):



۵ دکمهٔ OK را کلیک کنید. از آنجائیکه ویژوال استودیو .NET دیگر بطور مستقیم از کتابخانه‌های COM پشتیبانی نمی‌کند، لازمست یک لفاف برای آن بوجود آورد - پیام زیر هم گویای همین مطلب است:



۶ اگر این پیام را دیدید، دکمهٔ Yes را کلیک کنید، تا ویژوال استودیو لفاف کتابخانهٔ Microsoft Internet Controls را برایتان ایجاد و به پروژه اضافه کند.

بررسی شیء اینترنت اکسپلورر

اجازه دهید قبل از استفاده از شیء اینترنت اکسپلورر در برنامه، کمی بیشتر با خواص، متدها و رویدادهای آن آشنا شویم. شیء اینترنت اکسپلورر یکی از اشیاء کلاس InternetExplorer است، که خود جزء کتابخانهٔ Microsoft Internet Controls (فایل SHDocVw.dll) می‌باشد. در کلاس InternetExplorer خواص،

متدها و رویدادهای متعددی برای نمایش فایلهای HTML وجود دارد. همانطور که در فصل ۱۳ دیدید، کاوشگر شیء ابزاریست مناسب برای کسب اطلاعات دربارهٔ کتابخانه‌هائیست که جزء ویژوال بیسیک نیستند؛ و کتابخانهٔ Microsoft Internet Controls نیز از همین دسته است.

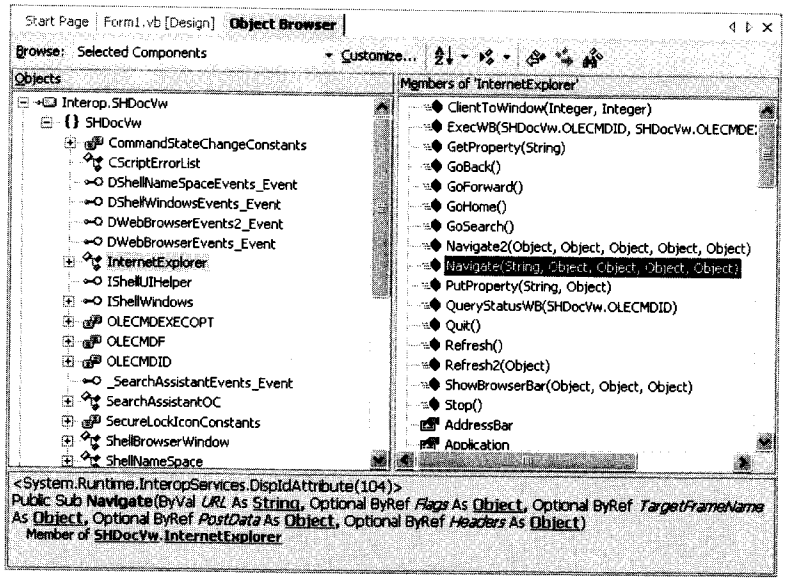
استفاده از کاوشگر شیء برای بررسی شیء اینترنت اکسپلورر

۱ از منوی View|Other Windows فرمان Object Browser را انتخاب کنید، تا پنجرهٔ کاوشگر شیء باز شود (کلید F2 هم همان کار را می‌کند).

۲ در قاب Objects ، ابتدا علامت + کنار Interop.SHDocVw ، و سپس علامت + کنار SHDocVw را کلیک کنید. با این کار تمام اشیاء کتابخانهٔ Microsoft Internet Controls ظاهر خواهند شد:

۳ شیء InternetExplorer را انتخاب کنید (این لیست بر حسب حروف الفبا مرتب شده است). با این کار، تمام خواص و متدهایی را، که این شیء در معرض دید دیگران می‌گذارد، در قاب Members خواهید دید.

۴ در لیست Members ، متد Navigate را انتخاب کنید. متد Navigate صفحهٔ مشخص شده در آرگومان *URL* را (که می‌تواند در همان سیستم، و یا روی اینترنت باشد) باز می‌کند. آرگومان *Flags* مشخص می‌کند که آیا این صفحه باید در لیست تاریخچهٔ اینترنت اکسپلورر یا کاشهٔ آن نوشته شود، یا خیر. آرگومانهای *TargetFrameName* ، *PostData* ، و *Headers* نحوهٔ باز شدن و نمایش فایل HTML را کنترل می‌کنند. تنها آرگومان اجباری این متد، آرگومان *URL* است؛ سایر آرگومانها را می‌توانید حذف کنید (شکل زیر را ببینید):



۵ در لیست Members ، خاصیت LocationURL را انتخاب کنید. خاصیت LocationURL نام و مسیر فایل HTML را، که در حال حاضر در اینترنت اکسپلورر دیده می‌شود، برمی‌گرداند.

۶ کمی دیگر در کاوشگر شیء بگردید، و خواص و متدهای اینترنت اکسپلورر را بررسی کنید.

۷ در پایان، با کلیک کردن دکمهٔ Close ، کاوشگر شیء را ببندید.

۸ با کلیک کردن دکمهٔ Save All ، پروژه را ذخیره کنید.

نمایش فایلهای HTML

نمایش فایلهای HTML در شیء اینترنت اکسپلورر بسیار ساده است، و فقط با چند خط کُد میسر می‌شود. ابتدا باید با استفاده از کلمهٔ کلیدی New یک متغیر از شیء اینترنت اکسپلورر تعریف کرده، و سپس خاصیت Visible این شیء را True کنید. پس از آن نوبت به بار کردن صفحهٔ HTML با استفاده از متد Navigate است:

```
Dim Explorer As SHDocVw.InternetExplorer
Explorer = New SHDocVw.InternetExplorer()
Explorer.Visible = True
Explorer.Navigate("http://www.microsoft.com")
```

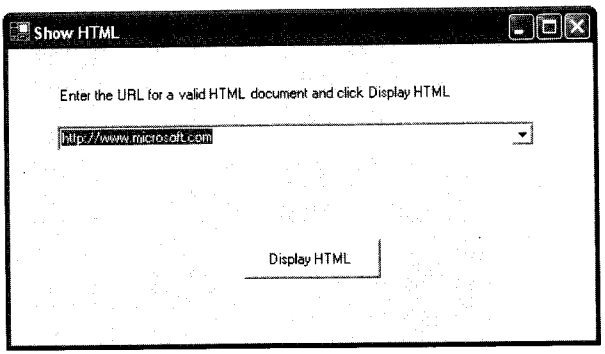
اگر می‌خواهید از این شیء اکسپلورر در تمام فرم یا برنامه استفاده کنید، باید آنرا در بالای فرم یا در یک ماژول استاندارد تعریف کنید. در برنامهٔ زیر، که Show HTML نام دارد، با همین تکنیک صفحهٔ HTML را که از یک لیست انتخاب می‌شود، باز می‌کنیم.

اجرای برنامهٔ Show HTML

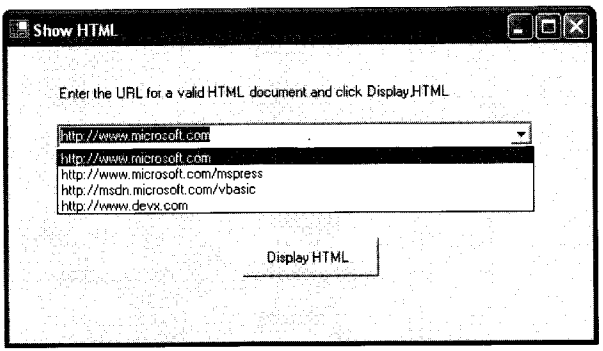
۱ با فرمان File|Close Solution ، پروژهٔ My Explorer Objects را ببندید.

۲ پروژهٔ Show HTML را، که در پوشهٔ c:\vbnetsbs\chap21\show html قرار دارد، باز کنید.

۳ برنامه را اجرا کنید، تا فرم اصلی Show HTML باز شود:



٤ با کلیک کردن پیکان سمت راست جعبهٔ ترکیبی، لیستی از آدرسهای وب مرتبط با برنامه‌نویسی ویژوال بیسیک باز می‌شود:

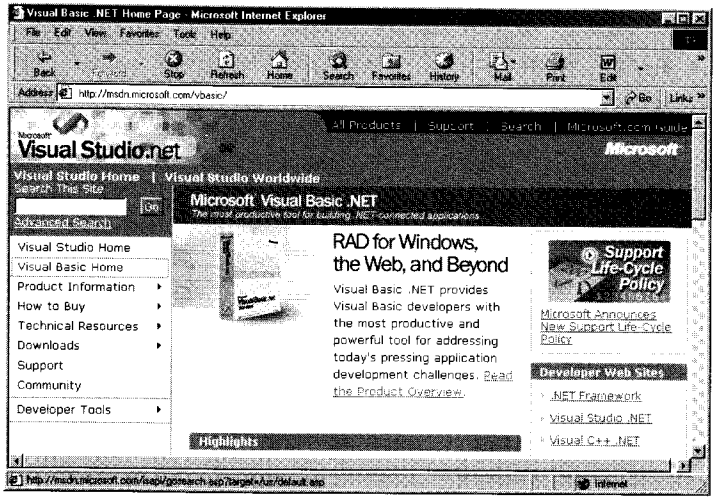


آدرسهایی که در اینجا می‌بینید، عبارتند از:

آدرس اینترنت	توضیح
http://www.microsoft.com	صفحهٔ اصلی شرکت میکروسافت
http://www.microsoft.com/mspress	صفحهٔ اصلی انتشارات میکروسافت
http://msdn.microsoft.com/vbasic	صفحهٔ اصلی برنامه‌نویسان ویژوال بیسیک میکروسافت
http://www.devx.com	مرجعی برای برنامه‌نویسان ویژوال بیسیک .NET

۵ صفحهٔ اصلی برنامه‌نویسان ویژوال بیسیک میکروسافت (http://msdn.microsoft.com/vbasic) را انتخاب کنید.

۶ دکمهٔ Display HTML را کلیک کنید. با این کار ویژوال بیسیک اینترنت اکسپلورر را باز کرده، و پس از چند لحظه (که به نوع ارتباط شما با اینترنت بستگی دارد)، صفحهٔ http://msdn.microsoft.com/vbasic را نشان می‌دهد (احتمالاً آنچه شما در کامپیوتر خود می‌بینید، با شکل زیر قدری متفاوت خواهد بود، چون محتویات این صفحه تقریباً هر روز تغییر می‌کند):



۷ اینترنت اکسپلورر را به حالت حداکثر در آورید، و روی یکی از لینکهای آن کلیک کنید. این صفحه یکی از بهترین منابع اینترنت برای دستیابی به آخرین اطلاعات ویژوال بیسیک است.

۸ پس از کمی گردش در این صفحه، اینترنت اکسپلورر را ببندید (اگر با پیام قطع ارتباط با اینترنت مواجه شدید، پاسخ منفی بدهید.)

۹ دوباره به برنامهٔ Show HTML برگردید (این برنامه هنوز در حال اجراست).

۱۰ روی جعبهٔ ترکیبی کلیک کرده، و بعد از پاک کردن آدرس داخل آن، آدرسی را که خود می‌خواهید وارد کنید. دکمهٔ Display HTML را کلیک کنید. (علاوه بر آدرسهای اینترنت، فایلهای HTML موجود در کامپیوتر خود را هم می‌توانید وارد کنید.)

۱۱ بعد از بازدید از چند صفحه، با کلیک کردن دکمهٔ Close ، برنامه را ببندید.

اکنون اجازه دهید نگاهی به کُد برنامهٔ Show HTML بیندازیم.

یک گام فراتر: پاسخ به رویدادهای اینترنت اکسپلورر

برای کنترل بهتر و مؤثرتر شیء اینترنت اکسپلورر، می‌توانیم از رویدادهای آن بهره بگیریم. همانطور که از فصلهای قبل بیاد دارید، هر شیء ویژوال بیسیک تعدادی رویداد دارد که می‌توان به کمک آنها رفتار شیء را کنترل کرد. برخی از مهمترین رویدادهای شیء اینترنت اکسپلورر عبارتند از: NavigateComplete2 ، DownloadBegin ، DownloadComplete ، DocumentComplete ، TitleChange ، و OnQuit .

اگر می‌خواهید در برنامهٔ خود از رویدادهای شیء اینترنت اکسپلورر استفاده کنید، باید تغییر کوچکی در دستور تعریف متغیر آن بدهید، چون رویدادهای اشیاء COM بطور خودکار در لیست Method Name ادیتور کُد ظاهر نمی‌شوند. برای اضافه کردن این قبیل رویدادها به ادیتور کُد باید از کلمهٔ کلیدی WinEvents استفاده کنید. تغییری که باید در تعریف شیء اینترنت اکسپلورر بدهیم، چنین است:

```
Public WinEvents Explorer As SHDocVw.InternetExplorer
```

بدین ترتیب، شیء Explorer در لیست Class Name و رویدادهای آن در لیست Method Name ادیتور کُد ظاهر خواهند شد.

در تمرین زیر، با استفاده از رویدادهای شیء اینترنت اکسپلورر، آدرسی راکه کاربر مستقیماً در جعبهٔ ترکیبی می‌نویسد، به لیست آن اضافه خواهیم کرد.

استفاده از رویداد NavigateComplete2

۱ پروژهٔ Show HTML را باز کنید (این پروژه در پوشهٔ c:\vbnetsbs\chap21\show html قرار دارد).

۲ به قسمت تعریف متغیرهای عمومی در ادیتور کُد (زیر عبارت "Windows Form Designer generated code") بروید.

۳ تعریف متغیر Explorer را مانند زیر اصلاح کنید:

```
Public WinEvents Explorer As SHDocVw.InternetExplorer
```

٤ کرسر را به خط دیگری ببرید، تا ویژوال بیسیک تغییر انجام شده را تشخیص دهد.

۵ لیست Class Name را باز کرده، و شیء Explorer را انتخاب کنید.

۶ از لیست Method Name ، رویداد NavigateComplete2 را انتخاب کنید، تا روال رویداد Explorer_NavigateComplete2 (به همراه تمام پارامترهایش) در ادیتور کُد باز شود.

۷ دستورات زیر را در این روال بنویسید:

```
ComboBox1.Items.Add(Explorer.LocationURL)
```

اکنون روال Explorer_NavigateComplete2 باید شبیه زیر باشد:

```
Private Sub Explorer_NavigateComplete2(ByVal pDisp As Object, _
        ByRef URL As Object) Handles Explorer_NavigateComplete2
    ComboBox1.Items.Add(Explorer.LocationURL)
End Sub
```

رویداد NavigateComplete2 زمانی رُخ می‌دهد، که اینترنت اکسپلورر یک صفحهٔ وب را با موفقیت بار کرده باشد؛ آدرسهای اشتباه باعث روی دادن این رویداد نخواهند شد. به همین دلیل وقوع این رویداد می‌تواند نشانهٔ خوبی برای موفقیت عملیات نمایش صفحهٔ وب باشد. در کُد بالا، با استفاده از خاصیت LocationURL تاریخچه‌ای از صفحات وب بازدید شده بوجود آورده‌ایم.

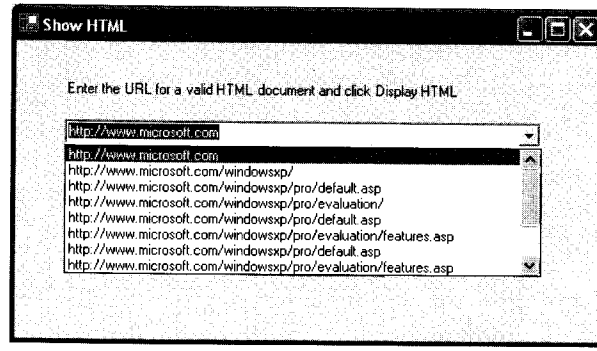
۸ با کلیک کردن دکمهٔ Save All ، پروژه را ذخیره کنید.

۹ برنامه را اجرا کنید.

۱۰ یکی از آدرسهای موجود در جعبهٔ ترکیبی را انتخاب کرده، و سپس دکمهٔ Display HTML را کلیک کنید.

۱۱ بعد از باز شدن این صفحه، چند تا از لینکهای آنرا کلیک کرده، و به آن صفحات بروید.

۱۲ به برنامهٔ Show HTML برگردید، و لیست جعبهٔ ترکیبی را باز کنید. همانطور که می‌بینید، تمام صفحاتی که آنها را بازدید کرده‌اید، به این لیست اضافه شده‌اند:



۱۳ کمی بیشتر با برنامه کار کرده، و سپس آنرا ببندید.

همانطور که دیدید، کار با شیء اینترنت اکسپلورر کاری ساده و در عین حال هیجان‌انگیز است.

مرجع سریع فصل ۲۱

برای ...	انجام دهید
اضافه کردن ارجاع کتابخانهٔ اینترنت اکسپلورر به برنامه	فرمان Project|Add Reference را اجرا کرده، و به برگهٔ COM بروید. بعد از انتخاب کتابخانهٔ Microsoft Internet Controls و کلیک کردن دکمهٔ Select ، با کلیک کردن OK این دیالوگ را ببندید.
بررسی مدل شیء اینترنت اکسپلورر	کاوشگر شیء را باز کرده (کلید F2)، و بترتیب شاخه‌های Interop.SHDocVw و SHDocVw را باز کنید. بعد از انتخاب شیء InternetExplorer در قاب سمت چپ، خواص، متدها و رویدادهای آنرا در قاب سمت راست بررسی کنید.

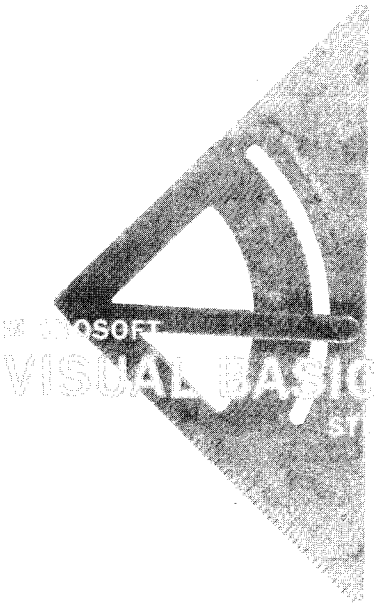
اجرای اینترنت اکسپلورر در برنامه	متغیری از نوع SHDocVw.InternetExplorer تعریف کرده، و پس از ایجاد آن با کلمهٔ کلیدی New ، خاصیت Visible آنرا به True ست کنید:
	`Dim Explorer As SHDocVw.InternetExplorer`
	`Explorer = New SHDocVw.InternetExplorer()`
	`Explorer.Visible = True`

نمایش یک سایت وب با شیء اینترنت اکسپلورر	از متد Navigate شیء اینترنت اکسپلورر استفاده کنید: `Explorer.Navigate("http://www.microsoft.com")`

استفاده از رویدادهای یک شیء COM (مانند اینترنت اکسپلورر)	برای تعریف این شیء از کلمهٔ کلیدی WinEvents استفاده کنید:
	`Public WinEvents Explorer As _`
	`SHDocVw.InternetExplorer`

فصل

۲۲



ایجاد برنامه‌های تعاملی وب با استفاده از فرمهای وب

در این فصل یاد می گیرید چگونه :

- ✔ یک برنامهٔ وب ایجاد کنید.
- ✔ از طراح فرمهای وب (Web Forms Designer) استفاده کنید.
- ✔ صفحهٔ وب را فرمت کنید.
- ✔ کنترلهای وب را به برنامه اضافه کنید.
- ✔ یک صفحهٔ HTML بسازید.
- ✔ برای برقراری ارتباط بین صفحات برنامهٔ وب، از کنترل هایپرلینک (HyperLink) استفاده کنید.

در فصل قبل با اصول اولیهٔ نمایش فایلهای HTML و استفاده از شیء اینترنت اکسپلورر در برنامه‌های ویژوال بیسیک .NET آشنا شدید. در این فصل خواهید دید که چگونه می‌توان از طراح فرمهای وب برای ایجاد برنامه‌های وب استفاده کرد. فرمهای وب مدل جدید برنامه‌نویسی برای اینترنت است، که به تکنولوژی ASP.NET متکی می‌باشد. فرمهای وب جایگزین DHTML و کلاسهای وب (WebClasses) در ویژوال بیسیک ۶ شده است، و با فرمهای ویندوز (که تا اینجای کتاب از آنها استفاده کرده‌ایم) تفاوت اساسی دارد. البته در اینجا مجال کافی برای توضیح کامل دربارهٔ فرمهای وب و ASP.NET وجود ندارد، اما تفاوت این دو بیشتر در لایه‌های زیرین است، تا شکل و شمایل و طرز استفاده از آنها؛ تجربه‌ای که در زمینهٔ فرمهای ویندوز دارید، به شما اجازه می‌دهد تا بلافاصله برنامه‌نویسی با فرمهای وب را شروع کنید (حتی اگر از قبل هیچ آشنایی با برنامه‌نویسی اینترنت و HTML نداشته باشید).

تازه‌های ویژوال بیسیک .NET

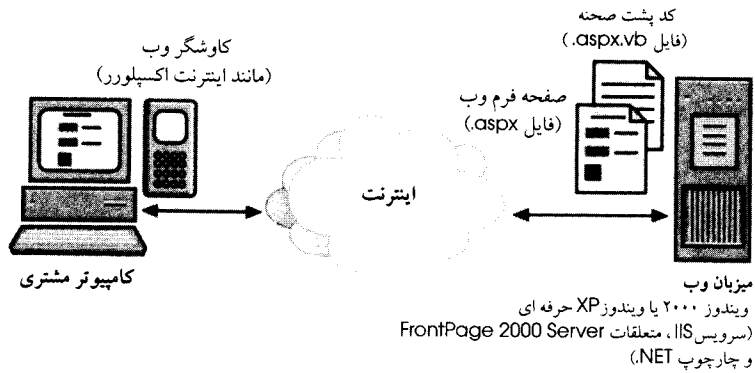
اگر تجربهٔ برنامه‌نویسی با ویژوال بیسیک ۶ را داشته باشید، متوجه ویژگیهای جدیدی در ویژوال بیسیک .NET خواهید شد، که برخی از آنها عبارتند از:

- مدل برنامه‌نویسی فرمهای وب زیرمجموعه‌ای از ASP.NET است. فرمهای وب و طراح فرمهای وب جایگزین کلاسهای وب (WebClasses) و طراح صفحات DHTML در ویژوال بیسیک ۶ شده‌اند.
- طراح فرمهای ویندوز و طراح فرمهای وب، علیرغم تفاوتهای بنیادین، از نظر ظاهر و طرز کار بسیار شبیه یکدیگرند. طراح فرمهای وب جزئی از ویژوال استودیو .NET است، و به همین دلیل در تمام زبانهای آن می‌توان از این طراح استفاده کرد.
- فرمهای وب برای نمایش در کاوشگرهای وب (مانند اینترنت اکسپلورر) طراحی می‌شوند. کنترلهای یک فرم وب در کامپیوتر مشتری برنامه (client) دیده می‌شوند، ولی عملکرد واقعی آنها در همان میزبان وب (Web server) ، که برنامه روی آن قرار دارد، باقی می‌ماند.
- با وجود یکسان بودن نام بسیاری از کنترلهای وب با کنترلهای ویندوز، آنها واقعاً یکی نیستند. برای مثال، کنترلهای وب بجای خاصیت Name خاصیتی بنام ID دارند.

آشنایی با ASP.NET

ASP.NET جدیدترین پلاتفرم برنامه‌نویسی وب میکروسافت است. با اینکه ASP.NET شباهتهای زیادی با سلف خود، یعنی ASP (Active Server Pages) دارد، اما برای انطباق با چارچوب .NET بطور کامل بازنویسی شده است. فرمهای وب (Web Forms) ، در واقع، ابزار طراحی صفحات وب (یا بعبارت دیگر، برنامه‌های وب) در ASP.NET است. اهداف برنامه‌های وب با برنامه‌های ویندوز یکسان است (گرفتن اطلاعات از کاربر، پردازش آنها، و برگرداندن نتیجه به وی)، اما آنها این کار را در یک میزبان وب (Web server) انجام می‌دهند، و فقط قسمتهای ظاهری کار در کامپیوتر مشتری (client) صورت می‌گیرد. این استراتژی اجازه می‌دهد تا برنامه‌ها و منابع نرم‌افزاری (و سخت‌افزاری) در یک نقطه متمرکز شوند، در حالیکه کاربران واقعی آنها در تمام دنیا پراکنده‌اند.

برای نوشتن یک برنامهٔ وب (Web application) در ویژوال بیسیک .NET ، ابتدا باید یک پروژهٔ ASP.NET Web Application ایجاد کرده، و سپس فرم (یا فرمهای) آنرا به کمک طراح فرمهای وب طراحی کنید. هر فرم وب دو بخش دارد: صفحه‌ای که دیده می‌شود، و فایل کُد پشت صحنه (code-behind file) . این تمایز بین بخش ظاهری و کُد اجرایی برنامه، درست شبیه همان چیزیست که در برنامه‌های ویندوز هم دیده بودید. البته کُد هر دو بخش ظاهری و اجرایی را می‌توان در یک فایل aspx. ذخیره کرد، ولی معمولاً رسم بر اینست که بخش ظاهری فرم وب در یک فایل aspx. ، و کُد پشت صحنهٔ آن در یک فایل aspx.vb. ذخیره شوند. در شکل زیر ارتباط بین فایلها و کامپیوترها را در یک برنامهٔ ASP.NET Web Application ملاحظه می‌کنید:



یک برنامهٔ وب، علاوه بر فرمهای وب، فایلهای دیگری نیز می‌تواند داشته باشد، که مهمترین آنها عبارتند از: ماژول کُد (فایل vb.)، صفحهٔ HTML (فایل htm.)، اطلاعات پیکربندی برنامه (فایل Web.config)، و اطلاعات عمومی برنامهٔ وب (فایل Global.asax). برای مدیریت همهٔ این اجزاء متنوع، می‌توانید از طراح فرمهای وب و کاوشگر راه‌حل کمک بگیرید.

فرمهای وب یا فرمهای ویندوز؟

تفاوتهای اساسی بین فرمهای وب و فرمهای ویندوز چیست؟ اول از همه اینکه استاندارد برنامه‌نویسی بین این دو متفاوت است: فرمهای ویندوز از پنجره‌های معمولی برای ارتباط با کاربر استفاده می‌کنند، در حالیکه این نقش در فرمهای وب بر عهدهٔ صفحات وب است (که باید از طریق یک کاوشگر وب آنها را دید).

کنترلهایی که روی یک فرم وب قرار می‌گیرند، علیرغم شباهت ظاهری با کنترلهای ویندوز، با آنها فرق دارند. کنترلهایی که در برنامه‌های ASP.NET Web Application مورد استفاده قرار می‌گیرند، در برگهٔ Web Forms و HTML جعبه ابزار ویژوال استودیو قرار دارند. این کنترلها خواص، متدها و رویدادهایی دارند، که خاص خود آنهاست.

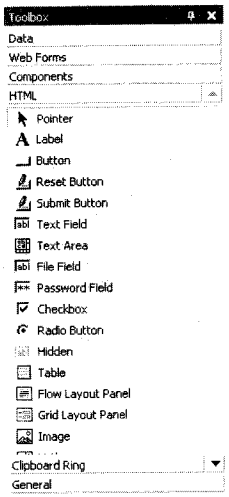
کنترلهای Web Forms به کنترلهای میزبان (server controls) معروفند، چون روی میزبان وب قرار می‌گیرند و کُد آنها همانجا اجرا می‌شود. علامت مشخصهٔ این کنترلها یک مثلث کوچک سبز رنگ (▶) است که در مرحلهٔ طراحی برنامه در بالای آنها دیده می‌شود. اما کنترلهای HTML ذاتاً کنترلهای مشتری (client controls) هستند، چون کُد آنها به کامپیوتر مشتری فرستاده شده و در همانجا اجرا می‌شود. البته می‌توان یک کنترل HTML را بصورت کنترل میزبان هم در آورد؛ برای این کار کافیست در طراح فرمهای وب روی کنترل HTML موردنظر راست-کلیک کرده، و فرمان Run as server control را انتخاب کنید - و یا صفت Runat آنرا به Server ست کنید. در یک برنامهٔ وب می‌توان از کنترلهای Web Forms، کنترلهای HTML، و یا ترکیبی از آنها استفاده کرد.

کنترلهای HTML

کنترلهای HTML نسبتاً قدیمی‌تر هستند، و بدلیل آن که کاملاً بر اساس استانداردهای HTML نوشته شده‌اند، در تمام کاوشگرهای وب می‌توان آنها را بکار گرفت. اگر قبلاً برنامه‌نویسی HTML کرده باشید، با این کنترلها (که Button، Text Field و Checkbox معروفترین آنها هستند) آشنائید. با اینکه کنترلهای

HTML مزیت مشترک بودن بین تمام کاوشگرهای وب را دارند، اما بدلیل آن که ذاتاً روی کامپیوتر مشتری اجرا می‌شوند (مگر اینکه آنها را بصورت کنترلهای میزبان در آورده باشید)، توانایی حفظ اطلاعات حالت (state information) را ندارند. بعبارت دیگر، هر بار که یک صفحهٔ وب را بار کنید، این کنترلها اطلاعات قبلی خود را از دست خواهند داد. در زیر کنترلهای HTML جعبه ابزار ویژوال استودیو را ملاحظه می‌کنید:

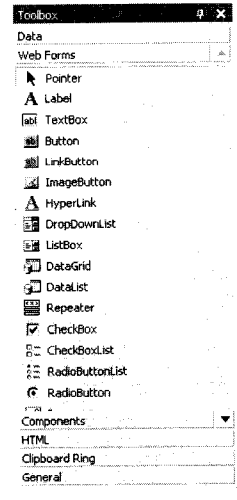
کنترلهای HTML



کنترلهای Web Forms

کنترلهای Web Forms جدیدترین محصولات ویژوال استودیو .NET هستند، که قابلیتهای بسیار بیشتری نسبت به کنترلهای HTML دارند. کنترلهای Web Forms بسیار شبیه کنترلهای ویندوز هستند، که بسیاری از آنها حتی با کنترلهای ویندوز همنام هستند، و خواص، متدها و رویدادهای مشترکی دارند. علاوه بر کنترلهای ساده‌ای مانند Button ، TextBox و Label ، کنترلهای پیچیده‌تری از قبیل DataGrid ، Calendar و RequiredFieldValidator را هم در جعبه ابزار ویژوال استودیو خواهید یافت. در زیر کنترلهای Web Forms جعبه ابزار ویژوال استودیو را ملاحظه می‌کنید:

کنترلهای Web Forms



پشتیبانی از انواع کاوشگرهای وب

آیا استفاده از این تکنولوژیهای جدید به معنای آنست که همهٔ کاربران ما باید از آخرین ویرایش کاوشگرهای وب استفاده کنند؟ و اگر چنین ارتقایی امکانپذیر نباشد، چه باید کرد؟

خوشبختانه چنین الزامی وجود ندارد. شیء DOCUMENT در ویژوال استودیو .NET دارای خاصیتی بنام targetSchema است، که اجازه می‌دهد تا نوع و ویرایش خاصی از کاوشگرهای وب را هدف قرار دهیم. گزینه‌های خاصیت targetSchema عبارتند از: Internet Explorer 3.02/Navigator 3.0 ، Internet Explorer 5.0 ، و Navigator 4.0 ؛ گزینهٔ پیش‌فرض Internet Explorer 5.0 است. مقدار خاصیت targetSchema بر کُد HTML خروجی ویژوال استودیو (و امکانات آن) تأثیر می‌گذارد. برای مثال، اگر به این خاصیت مقدار Internet Explorer 3.02/Navigator 3.0 بدهید، و سپس خاصیت pageLayout را به GridLayout ست کنید، جدولهای HTML خروجی بجای CSS از اشیاء Positioning استفاده خواهند کرد.

بیش از این دربارهٔ خاصیت targetSchema توضیح نمی‌دهم، اما بعداً در همین فصل از pageLayout استفاده خواهیم کرد. (برای کسب اطلاعات بیشتر دربارهٔ خاصیت targetSchema به سیستم کمک ویژوال استودیو مراجعه کنید.)

آشنایی با برنامه‌های وب

بهترین راه آشنایی با ASP.NET و برنامه‌های وب، نوشتن یک برنامهٔ عملی است، که در تمرینهای این فصل انجام خواهیم داد. این برنامه یک ماشین محاسبهٔ اقساط وام اتومبیل است، که صفحهٔ دومی نیز بعنوان راهنمایی و کمک دارد. اما قبل از آن باید از پیکربندی صحیح ویژوال استودیو برای برنامه‌نویسی ASP.NET مطمئن شویم، و سپس به سراغ نوشتن برنامهٔ وب و طراحی صفحات آن برویم.

نصب نرم‌افزارهای لازم برای برنامه‌نویسی ASP.NET

قبل از شروع به نوشتن برنامهٔ وب، باید مطمئن شویم که فایلها و نرم‌افزارهای لازم برای برنامه‌نویسی ASP.NET روی کامپیوتر وجود دارند. برنامه‌های ASP.NET به سه عنصر نرم‌افزاری نیاز دارند: یک میزبان وب - سرویس Internet Information Services (IIS) که باید روی ویندوز ۲۰۰۰ یا ویندوز XP ویرایش حرفه‌ای نصب شده باشد؛ متعلقات FrontPage 2000 Server Extensions ؛ و کتابخانه‌های چارچوب .NET. باید مطمئن شوید که به این برنامه‌ها و سرویسها (خواه بصورت محلی و یا روی شبکه) دسترسی دارید.

بسیار مهم

ویرایش خانگی ویندوز XP از IIS و FrontPage 2000 Server Extensions پشتیبانی نمی‌کند، و این بدان معناست که روی چنین کامپیوتری نمی‌توانید برنامه‌های ASP.NET را بصورت محلی ایجاد کنید. در این فصل فرض کرده‌ایم که شما ویندوز ۲۰۰۰ یا ویندوز XP ویرایش حرفه‌ای دارید، و میزبان وب روی همان کامپیوتری است که برنامه‌ها را در آن می‌نویسید.

این کارها معمولاً بر عهدهٔ Windows Component Update Wizard است، که قبل از نصب ویژوال استودیو .NET زمینه‌های لازم را فراهم می‌آورد. اما اگر در هنگام نصب ویژوال استودیو .NET این

برنامه‌ها را نصب نکرده‌اید، قبل از شروع برنامه‌نویسی با ASP.NET باید آنها را نصب کنید.

توجه

میکروسافت اکیداً توصیه می‌کند که IIS و FrontPage 2000 Server Extensions را قبل از ویژوال استودیو .NET نصب کنید، چون چارچوب .NET باید خود را در رجیستر IIS کند. اگر IIS و FrontPage 2000 Server Extensions را بعد از چارچوب .NET نصب کنید، باید طی مراحلی که در زیر خواهید دید، آنرا مجدداً پیکربندی کنید.

برای نصب IIS و FrontPage 2000 Server Extensions مراحل زیر را دنبال کنید:

نصب IIS و FrontPage 2000 Server Extensions

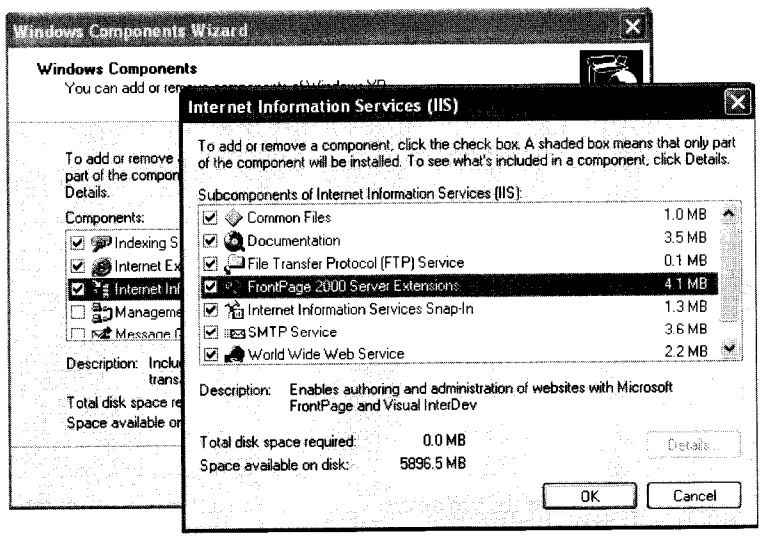
۱ منوی Start|Settings ویندوز را باز کرده، و روی Control Panel کلیک کنید.

۲ روی آیکون Add/Remove Programs دو-کلیک کنید.

۳ در دیالوگ Add/Remove Programs ، دکمهٔ Add/Remove Windows Components را کلیک کنید.

٤ در دیالوگ Add/Remove Windows Components ، آیتم Internet Information Services (IIS) را انتخاب کرده، و دکمهٔ Details را کلیک کنید.

۵ اگر آیتمهای FrontPage 2000 Server Extensions و World Wide Web Server انتخاب نشده‌اند، آنها را انتخاب کنید:



۶ دکمهٔ OK را کلیک کنید.

۷ Next را کلیک کرده، و مراحل بعدی را دنبال کنید. (احتمالاً به CD ویندوز ۲۰۰۰ یا XP هم نیاز پیدا خواهید کرد؛ آنرا دم دست داشته باشید.)

اگر IIS و FrontPage 2000 Server Extensions را بعد از چارچوب .NET نصب کرده‌اید، با طی مراحل زیر آنرا مجدداً پیکربندی کنید:

پیکربندی مجدد چارچوب .NET

۱ اگر ویژوال استودیو .NET را از روی CD نصب کرده‌اید، CD موسوم به Windows Component Update – و اگر از روی DVD نصب کرده‌اید، DVD ویژوال استودیو .NET – را درون درایو قرار دهید. (اگر از CD استفاده می‌کنید، پیامی که می‌گوید «دیسک شمارهٔ ۱ را در درایو قرار دهید»، نادیده بگیرید.)

۲ از منوی Start ویندوز، آیتم Run را کلیک کنید، تا دیالوگ Run باز شود.

۳ اگر از CD استفاده می‌کنید، فرمان زیر را در یک خط در فیلد Open بنویسید (بجای *<CDdrive>* نام درایو CD کامپیوتر خود را وارد کنید):

```
<CDdrive>:\dotNetFramework\dotnetfx.exe /t:c:\temp
/c:"msiexec.exe /fvecms c:\temp\netfx.msi"
```

اگر از DVD استفاده می‌کنید، فرمان زیر را در یک خط در فیلد Open بنویسید (بجای *<DVDdrive>* نام درایو DVD کامپیوتر خود را وارد کنید):

```
<DVDdrive>:\wcu\dotNetFramework\dotnetfx.exe /t:c:\temp
/c:"msiexec.exe /fvecms c:\temp\netfx.msi"
```

٤ دکمهٔ OK را کلیک کنید.

۵ در پاسخ به پیامی که می‌پرسد: «آیا میل دارید نرم‌افزار Microsoft .NET Framework را نصب کنید؟»، Yes را کلیک کنید.

بعد از پایان مراحل فوق، آماده‌اید تا برنامه‌نویسی ASP.NET را شروع کنید.

توجه

از آنجائیکه روش فوق طریقهٔ صحیح نصب ویژوال استودیو .NET نیست، شاید حتی بعد از پیکربندی مجدد چارچوب .NET ، باز هم در اجرای برنامه‌های وب به مشکلاتی برخورد کنید. اگر با چنین وضعیتی مواجه شدید، به فایلهای Setup\WebServer.htm و Setup\WebServerInfo.htm در CD یا DVD نصب ویژوال استودیو .NET (یا سرفصل "Troubleshooting Web Projects" در سیستم کمک ویژوال استودیو) مراجعه کنید.

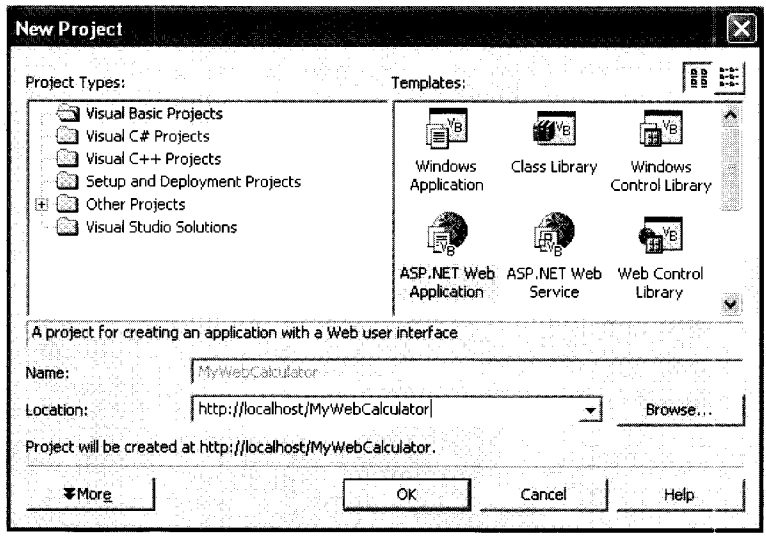
ایجاد یک برنامهٔ وب

۱ در محیط ویژوال استودیو .NET ، دیالوگ «پروژهٔ جدید» را باز کنید – فرمان File|New|Project .

۲ در این دیالوگ، ابتدا آیتم Visual Basic Projects را در قسمت Project Types انتخاب کرده، و سپس در قسمت Templates روی آیکون ASP.NET Web Application کلیک کنید.

نامگذاری پروژه‌های وب قدری با برنامه‌های ویندوز متفاوت است: بجای ایجاد یک پوشهٔ معمولی در هارد دیسک، باید محلی را در یک میزبان وب برای ایجاد برنامه انتخاب کنید (دقت کنید که فیلد Name غیرفعال، و بجای آن فیلد Location فعال شده است). میزبان وب پیش‌فرض، یعنی http://localhost ، به میزبان وبی که روی همان کامپیوتر است، اشاره می‌کند. اگر می‌خواهید از میزبان وب یک کامپیوتر دیگر استفاده کنید، باید قبلاً چارچوب .NET. (و فایلهای پشتیبانی لازم) را روی آن نصب کرده باشید. همانطور که می‌بینید، میزبان وب نه با نام درایو و پوشه، بلکه با یک URL (آدرس وب) مشخص می‌شود.

۳ از آنجائیکه فرض کرده‌ایم که میزبان وب ما روی همان کامپیوتر است، آدرس **http://localhost/MyWebCalculator** را در فیلد Location وارد کنید:

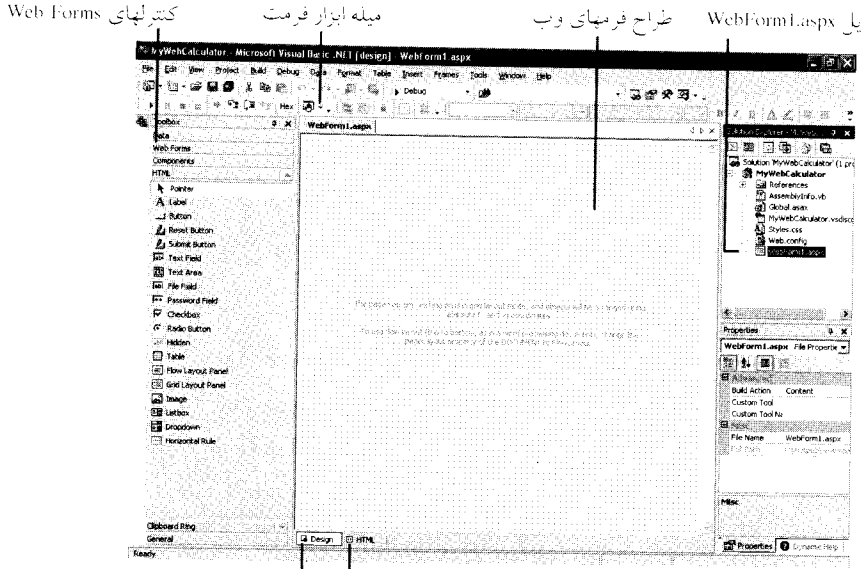


٤ OK را کلیک کنید.

توجه

اگر در این مرحله با خطای ایجاد پروژهٔ وب برخورد کردید، کامپیوتر شما برای برنامه‌نویسی ASP.NET آماده نیست. به قسمتهای قبلی همین فصل مراجعه کرده، و نرم‌افزارهای لازم را نصب کنید.

با این کار، ویژوال استودیو فایلهای لازم برای برنامهٔ وب (WebForm1.aspx و WebForm1.aspx.vb) را ایجاد خواهد کرد (شکل زیر را ببینید):



فرمهای وب، برخلاف فرمهای ویندوز، سفید رنگ (با نقاط شبکهٔ سیاه) هستند. در پائین طراح فرمهای وب هم دو برگه می‌بینید، بنامهای HTML (کُد HTML فرم وب) و Design (طراحی ظاهری فرم وب) -در برگهٔ Design می‌توانید ظاهر فرم را، آنطور که در کاوشگر وب دیده خواهد شد، ببینید. طراحی فرم وب در دو حالت می‌تواند صورت گیرد: حالت طرح شبکه‌ای (grid layout mode) و حالت طرح جریانی (flow layout mode) . این حالتها نحوهٔ قرار گرفتن اشیاء در صفحهٔ وب را کنترل می‌کنند (یک جمله در زمینهٔ فرم وب نشان می‌دهد که در کدام حالت قرار دارید).

در برگهٔ HTML می‌توانید کُد HTML فرم وب را مستقیماً ادیت کنید. اگر قبلاً با نرم‌افزارهای Microsoft Visual InterDev و Microsoft FrontPage کار کرده‌اید، شاید بتوانید از این کُدها سر در بیاورید، و یا حتی آنها را دستکاری کنید.

در میله ابزار فرمت ویژوال استودیو تغییراتی می‌بینید، که به طراحی فرمهای وب مربوط هستند. در برگهٔ Web Forms جعبه ابزار هم کنترلهای ASP.NET را ملاحظه می‌کنید.

استفاده از طراح فرمهای وب

بر خلاف فرمهای ویندوز، برای اضافه کردن مستقیم متن به فرم وب باید در حالت طرح جریانی باشید. وقتی در این حالت هستید، می‌توانید متن را مستقیماً وارد کرده، و آنرا فرمت کنید (در حالت طرح شبکه‌ای، برای این کار باید از کنترل Label استفاده کنید). در تمرین قسمت آینده با این تکنیک آشنا خواهید شد.

اضافه کردن متن به صفحهٔ وب

۱ صفحهٔ وب WebForm1 را در طراح فرمهای وب فعال کرده، و پنجرهٔ خواص را باز کنید. این شیء DOCUMENT نام دارد، و برای وارد کردن متن در آن باید آنرا به حالت طرح جریانی ببرید.

۲ خاصیت pageLayout شیء DOCUMENT را به FlowLayout ست کنید؛ با این کار، ویژوال استودیو نقاط شبکهٔ صفحهٔ وب را حذف می‌کند.

۳ دوباره روی صفحهٔ WebForm1 کلیک کنید، تا یک کرسر متن در بالای آن ظاهر شود.

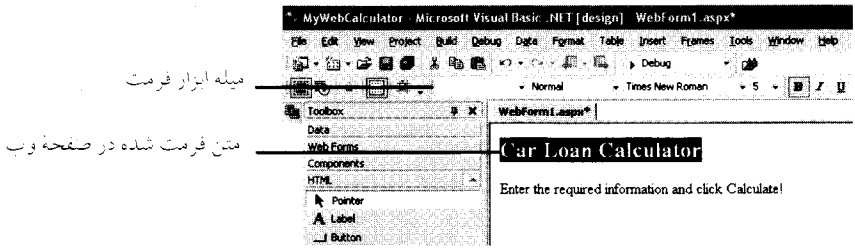
٤ جملهٔ **Car Loan Calculator** را نوشته، و Enter را بزنید. این جمله درست به همان صورتی که نوشته‌اید، در کاوشگر وب دیده خواهد شد.

۵ در زیر عنوان برنامه، جملهٔ **Enter the required information and click Calculate!** را وارد کنید.

اکنون باید فرمت عنوان برنامه را با استفاده از امکانات میله ابزار فرمت عوض کنیم.

۶ جملهٔ Car Loan Calculator را انتخاب کنید.

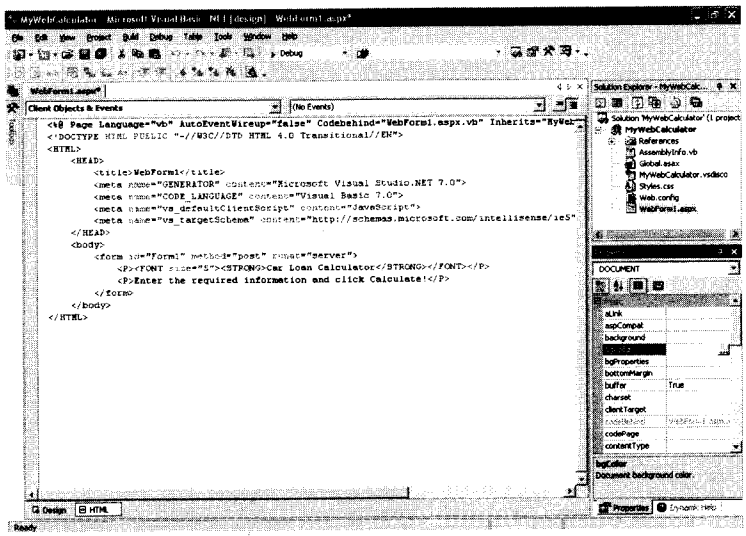
۷ دکمهٔ Bold را در میله ابزار فرمت کلیک کرده، و اندازهٔ فونت آنرا به 5 ست کنید. (اندازهٔ فونت در صفحات وب عددی بین 1 تا 7 است؛ اندازهٔ 5 تقریباً معادل فونت 18 است.) شکل زیر را ببینید:



اکنون اجازه دهید کُد HTML این صفحه را بررسی کنیم.

بررسی کُد HTML صفحهٔ WebForm1

۱ در قسمت پائین طراح فرمهای وب، روی برگهٔ Design کلیک کنید. در این برگهٔ کُد HTML صفحهٔ WebForm1 را می‌بینید (برای بهتر دیدن این صفحه، جعبه ابزار را ببندید):



در اینجا تمام کُد HTML صفحهٔ وب WebForm1 را می‌بینید. همانطور که می‌بینید، این کُد بسیار کوتاه است، و در خط اول آن تمام اطلاعات لازم دربارهٔ این صفحهٔ وب (حتی نام فایل کُد پشت صحنهٔ آن) آورده شده است. متن‌هایی که ما وارد کردیم (و فرمت آنها)، را در قسمت <body> این فایل می‌بینید.

نکته

نمای HTML یک ادیتور تمام عیار است، و می‌توانید کُد صفحهٔ وب را مستقیماً دستکاری کنید. البته برای این کار باید آشنایی کافی با زبان HTML و برچسب‌های آن داشته باشید.

۲ به نمای Design برگردید (و اگر جعبه ابزار را بسته‌اید، آنرا باز کنید).

۳ شیء DOCUMENT را از لیست Object پنجرهٔ خواص انتخاب کنید.

اکنون که اضافه کردن متن را تمام کردیم، می‌توانیم به حالت طرح شبکه‌ای برگردیم؛ پس:

٤ خاصیت pageLayout شیء DOCUMENT را به GridLayout ست کنید.

طرح شبکه‌ای یا طرح جریانی؟

تفاوت طرح شبکه‌ای با طرح جریانی چیست؟ و اصولاً چرا دو طرح وجود دارد؟ هر یک از طرح‌ها مزایا و معایب خاص خود را دارند، ولی در کل آنها دو روش متفاوت برای کنترل طراحی کنترلها روی صفحهٔ وب (و آنچه که در کاوشگر وب دیده خواهد شد) هستند. حالت طرح شبکه‌ای به شما اجازه می‌دهد تا اندازه، مکان (و حتی روی هم افتادن کنترلها) را دقیقاً کنترل کنید. اما این حالت کُد HTML پیچیده‌تری تولید می‌کند، که ممکنست با کاوشگرهای قدیمی‌تر سازگاری نداشته باشد. اگر می‌خواهید برنامه‌تان در تمام کاوشگرها بدرستی دیده شود، توصیه می‌کنم خاصیت pageLayout را به FlowLayout ، و خاصیت targetSchema را به Internet Explorer 3.02/Navigator 3.0 ست کنید.

اضافه کردن کنترلهای وب به برنامه

حال وقت آنست که کنترلهای مورد نیاز را به صفحهٔ وب Car Loan Calculator اضافه کنیم. این کنترلها در برگهٔ Web Forms قرار دارند، و بسیار شبیه کنترلهای همنام خود در فرمهای ویندوز هستند (البته تفاوتهایی هم دارند، که به آنها اشاره خواهم کرد).

استفاده از کنترلهای TextBox ، Label و Button

۱ به برگهٔ Web Forms جعبه ابزار بروید، و مطمئن شوید که فرم وب در حالت طرح شبکه‌ای قرار دارد (نشانهٔ بودن در این حالت، نقاط سیاه رنگ شبکه روی فرم است).

۲ کنترل TextBox را انتخاب کرده، و یک جعبه متن زیر خطوط نوشته شده در تمرین قبل رسم کنید. همانطور که گفتم، کار با کنترلهای وب در حالت طرح شبکه‌ای درست مثل کنترلهای معمولی ویندوز است.

آیکون سبز رنگ کوچکی که در بالای این کنترل دیده می‌شود، نشانهٔ آنست که کُد این کنترل روی میزبان وب اجرا می‌شود.

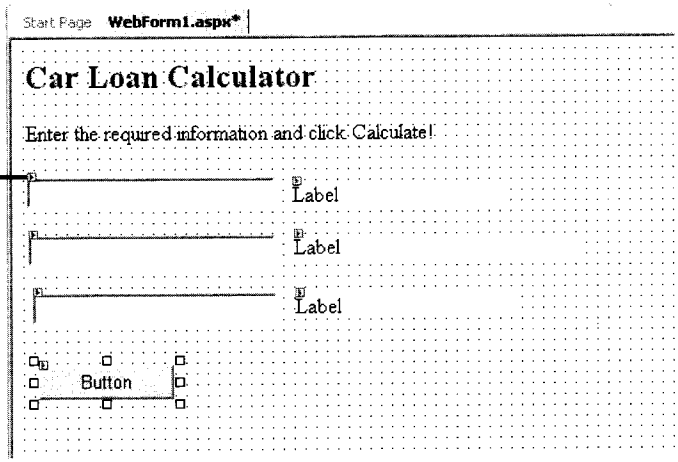
۳ دو جعبه متن دیگر زیر قبلی رسم کنید.

برای مشخص کردن وظیفهٔ این جعبه‌متن‌ها هم از سه برچسب استفاده خواهیم کرد.

٤ کنترل Label را انتخاب کرده، و یک برچسب سمت راست جعبه متن اولی بکشید.

۵ دو برچسب دیگر نیز سمت راست جعبه‌متن‌های دوم و سوم رسم کنید.

۶ کنترل Button را انتخاب کرده، و یک دکمه زیر جعبه‌متن سوم رسم کنید. در شکل زیر فرم وب را تا بدینجا ملاحظه می‌کنید:

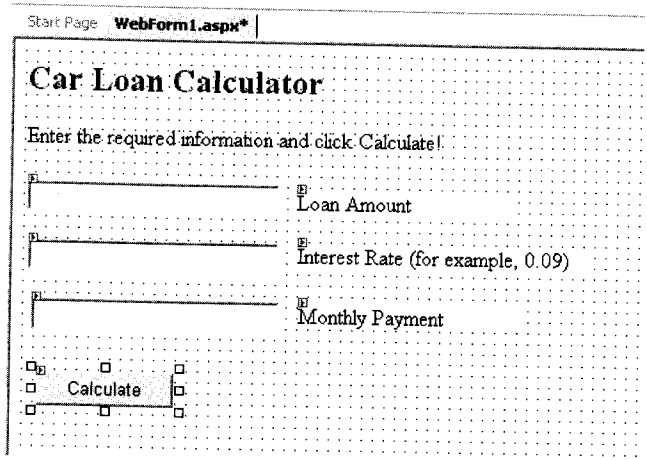


اکنون باید چند خاصیت این کنترلها را تغییر دهیم. در این قسمت متوجه یکی از تفاوتهای کنترلهای وب و ویندوز خواهید شد: کنترلهای وب بجای خاصیت Name خاصیتی بنام ID دارند.

۷ خواص کنترلهای وب را مانند زیر ست کنید:

شیء	خاصیت	مقدار
TextBox1	ID	txtAmount
TextBox2	ID	txtInterest
TextBox3	ID	txtPayment
Label1	ID	lblAmount
	Text	"Loan Amount"
Label2	ID	lblInterest
	Text	"Interest Rate (for example, 0.09)"
Label3	ID	lblPayment
	Text	"Monthly Payment"
Button1	ID	btnCalculate
	Text	"Calculate"

بعد از این تغییرات، صفحهٔ وب WebForm1 را در شکل زیر می‌بینید:



نوشتن روال رویداد برای کنترلهای وب

روش نوشتن روال رویداد برای کنترلهای وب هیچ تفاوتی با کنترلهای ویندوز ندارد: روی کنترل موردنظر دو-کلیک کنید، و دستورات را در ادیتور کُد بنویسید. تفاوت این دو در اینست که کُد صفحهٔ وب در میزبان وب اجرا می‌شود، نه روی کامپیوتری که این صفحه در آن دیده می‌شود. در واقع، وقتی کاربر دکمه‌ای را کلیک می‌کند، این رویداد کلیک به میزبان وب فرستاده شده، و میزبان وب پس از اجرای کُد مربوطه، نتیجهٔ کار را به کاوشگر پس می‌فرستد (می‌بینید که طرز کار کنترلهای وب چقدر با کنترلهای ویندوز فرق دارد!).

در تمرین زیر، برای کلیک شدن دکمهٔ btnCalculate یک روال رویداد می‌نویسیم.

ایجاد روال رویداد btnCalculate_Click

۱ در صفحهٔ وب WebForm1 ، روی دکمهٔ Calculate دو-کلیک کنید، تا فایل کُد پشت صحنه (فایل WebForm1.aspx.vb) باز شده، و روال رویداد btnCalculate_Click در ادیتور کُد ظاهر شود.

۲ دستورات زیر را در این روال بنویسید:

```
Dim LoanPayment As Single
'Use Pmt function to determine payment for 36 month loan
LoanPayment = Pmt(txtInterest.Text / 12, 36, txtAmount.Text)
)"00.0$" ,)tnemyaPnaoL(sbA(tamroF = txeT.tnemyaPtxt
```

در این کُد برای محاسبهٔ اقساط وام از تابع Pmt (که یکی از توابع داخلی ویژوال بیسیک است) استفاده کرده‌ایم. این تابع اقساط وامی با مقدار txtAmount.Text و بهرهٔ txtInterest.Text را در یک دورهٔ سه ساله (۳۶ ماه) محاسبه می‌کند. مقدار برگشتی این تابع در یک متغیر اعشاری بنام LoanPayment ذخیره شده، و پس از فرمت شدن در جعبه‌متن txtPayment نوشته می‌شود. از آنجائیکه تابع Pmt (به نشانهٔ مقروض بودن) یک عدد منفی برمی‌گرداند، با تابع Abs (قدر مطلق) آنرا به عددی مثبت تبدیل کرده‌ایم.

همانطور که می‌بینید، کُدنویسی در صفحات وب بسیار شبیه برنامه‌های معمولی ویژوال بیسیک است. کُد فوق به یک دستور Imports نیز نیاز دارد، که باید آنرا بالای ادیتور کُد بنویسیم.

۳ به بالای ادیتور کُد رفته، و دستور زیر را بعنوان اولین خط بنویسید:

```
Imports System.Math
```

علت اضافه کردن این دستور، وجود تابع Abs (از اعضای کتابخانهٔ System.Math) است.

٤ با کلیک کردن دکمهٔ Save All ، برنامه را ذخیره کنید.

برنامهٔ Car Loan Payment به همین سادگی نوشته شد! اکنون اجازه دهید آنرا اجرا کنیم.

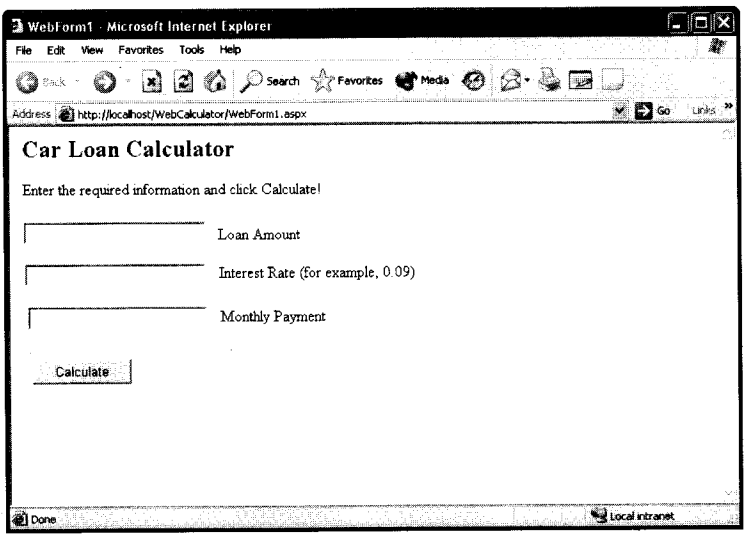
اجرای برنامهٔ وب

۱ برنامه را اجرا کنید.

توجه

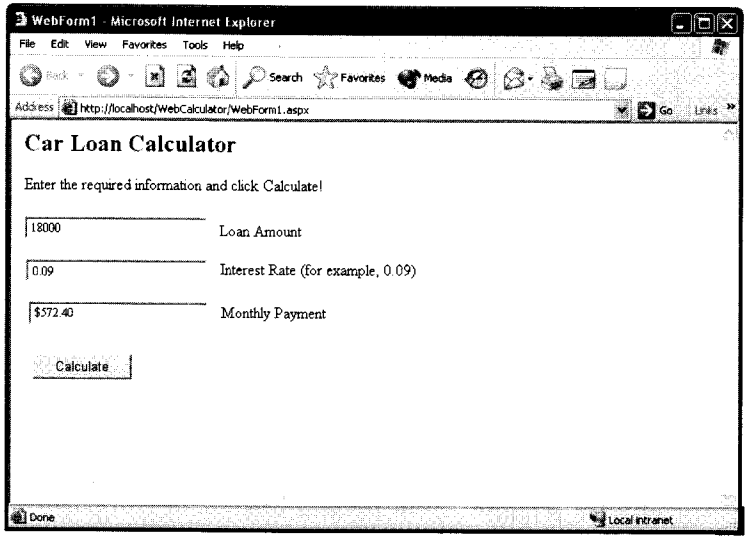
اگر پیامی دیدید که می‌گفت «میزبان وب از دیباگ کردن برنامه‌های ASP.NET پشتیبانی نمی‌کند»، به معنای آنست که IIS ، FrontPage 2000 Server Extensions و چارچوب .NET. بدرستی نصب و پیکربندی نشده‌اند. برای برطرف کردن این مشکل به ابتدای همین فصل، قسمت «نصب نرم‌افزارهای لازم برای برنامه‌نویسی ASP.NET »، مراجعه کنید.

ویژوال بیسیک بعد از کامپایل کردن پروژه، آنرا در اینترنت اکسپلورر اجرا می‌کند:



۲ در جعبه‌متن‌های Loan Amount و Interest Rate بترتیب اعداد **18000** و **0.09** را وارد کنید. در اینجا می‌خواهیم اقساط بازپرداخت وامی به مبلغ 18,000 دلار، با بهرهٔ 9% را در یک دورهٔ ۳۶ ماهه محاسبه کنیم.

۳ دکمهٔ Calculate را کلیک کنید. برنامه بعد از محاسبهٔ اقساط این وام، مبلغ اقساط ماهانه را (که 572.40 دلار است) در جعبه‌متن Monthly Payment نمایش می‌دهد:



۴ اینترنت اکسپلورر را ببندید؛ با این کار برنامهٔ وب ما نیز بسته خواهد شد.

همانطور که دیدید، نوشتن برنامهٔ وب با ویژوال بیسیک بسیار ساده است، و تنها تفاوت آن با یک برنامهٔ ویندوز اینست که در کاوشگر وب (و در واقع روی میزبان وب) اجرا می‌شود. برای اجرای صحیح (و در واقع توزیع) برنامهٔ وب هم، کافیست فایلهای aspx. (و سایر فایلهای لازم) را روی یک میزبان وب کپی کنید.

تعیین اعتبار فیلدهای ورودی در فرمهای وب

برنامهٔ وبی که نوشتیم، ساده و مفید است، ولی یک اشکال کوچک دارد: اگر کاربر یکی از اعداد را وارد نکند (و یا درست وارد نکند)، چه اتفاقی خواهد افتاد؟ برای اجتناب از این قبیل مشکلات، باید از یک یا چند کنترل تعیین‌اعتبارکننده (validator control) در برنامه استفاده کنیم. این کنترلها که در برگهٔ Web Forms جعبه ابزار قرار دارند، می‌توانند فیلدهای ورودی مهم را به روشهای مختلف تعیین اعتبار کنند. برای مثال، کنترل RequiredFieldValidator اطمینان حاصل می‌کند که کاربر در یک فیلد مقدار وارد کرده است؛ یا، کنترل RangeValidator اطمینان حاصل می‌کند که عدد وارد شده در محدودهٔ صحیح قرار داشته باشد.

یک گام فراتر: ایجاد لینک به یک صفحهٔ وب دیگر

اگر برنامهٔ وب بیش از یک صفحه داشته باشد، برای رفتن از یک صفحه به صفحهٔ دیگر می‌توان از کنترلی بنام HyperLink (هایپرلینک) که در برگهٔ Web Forms جعبه ابزار ویژوال بیسیک قرار دارد، استفاده کرد. این کنترل یک خاصیت بنام Text دارد، که در واقع همان عبارت یا جمله‌ایست که کاربر روی آن کلیک می‌کند، و

خاصیت دیگری بنام NavigateUrl ، که آدرس صفحه‌ایست که باید باز شود.

در تمرین زیر، یک صفحهٔ HTML دیگر (بعنوان صفحهٔ Help برنامه) با ویژوال استودیو ساخته، و با یک کنترل هایپرلینک امکان دسترسی به آن را برای کاربر صفحهٔ WebCalculator فراهم خواهیم آورد.

ایجاد یک صفحهٔ HTML

۱ از منوی Project فرمان Add HTML page را اجرا کنید، تا دیالوگ «اضافه کردن آیتم جدید» باز شود.

۲ در فیلد Name نام این صفحه را **WebCalculatorHelp.htm** گذاشته، و Open را کلیک کنید. آیتم جدیدی به کاوشگر راه‌حل اضافه شده، و یک صفحهٔ خالی در «طراح HTML» باز می‌شود.

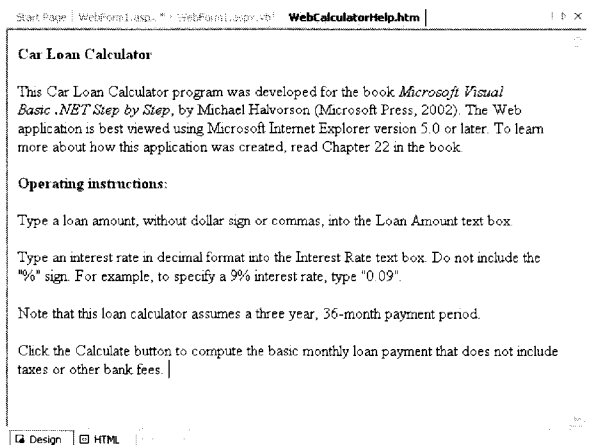
توجه کنید که برگهٔ Web Forms دیگر در جعبه ابزار دیده نمی‌شود، چون صفحات HTML از این نوع کنترلها پشتیبانی نمی‌کنند.

۳ صفحهٔ HTML را فعال کرده، و خاصیت pageLayout آنرا به FlowLayout ست کنید (با این کار، نقاط سیاه شبکه ناپدید می‌شوند).

٤ روی صفحهٔ HTML کلیک کنید تا کرسر چشمک‌زن متن در آن ظاهر شود، سپس متن زیر را در آن بنویسید:

Car Loan Calculator
This Car Loan Calculator program was developed for the book Microsoft Visual Basic .NET Step by Step, by Michael Halvorson (Microsoft Press, 2002). The Web application is best viewed using Microsoft Internet Explorer version 5.0 or later. To learn more about how this application was created, read Chapter 22 in the book.
Operating instructions:
Type a loan amount, without dollar sign or commas, into the Loan Amount text box.
Type an interest rate in decimal format into the Interest Rate text box. Do not include the "%" sign. For example, to specify a 9% interest rate, type "0.09".
Note that this loan calculator assumes a three year, 36-month payment period.
Click the Calculate button to compute the basic monthly loan payment that does not include taxes or other bank fees.

۵ با استفاده از میله ابزار فرمت، این متن را مانند شکل زیر فرمت کنید:



اکنون باید با استفاده از کنترل هایپرلینک ارتباط صفحهٔ WebCalculator را با این صفحه برقرار کنیم.

استفاده از کنترل هایپرلینک

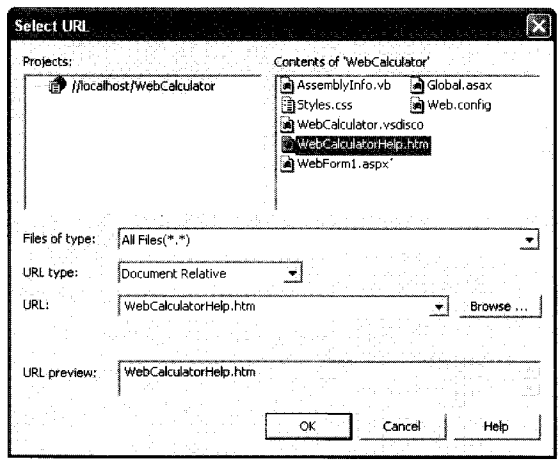
۱ به صفحهٔ WebForm1 (فایل WebForm1.aspx) برگردید.

۲ کنترل HyperLink را از برگهٔ Web Forms جعبه ابزار ویژوال بیسیک انتخاب کرده، و یک هایپرلینک سمت راست دکمهٔ Calculate رسم کنید.

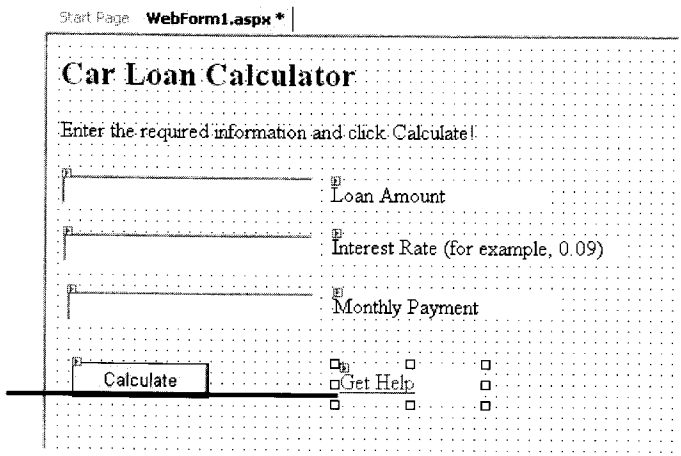
۳ خاصیت Text این کنترل را به **Get Help** ست کنید.

٤ در پنجرهٔ خواص، خاصیت NavigateUrl را انتخاب کرده، و روی دکمهٔ ... سمت راست آن کلیک کنید، تا دیالوگ «انتخاب URL» باز شود.

۵ در قاب سمت راست، فایل WebCalculatorHelp.htm را انتخاب کنید:



۶ OK را کلیک کنید (شکل زیر را ببینید):



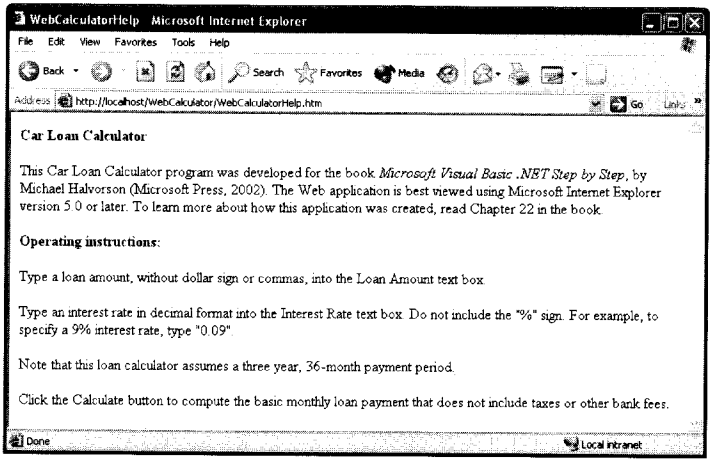
این هم از لینک به صفحهٔ دوم. اکنون آماده‌ایم برنامه را اجرا کنیم.

۷ با کلیک کردن دکمهٔ Save All ، پروژه را ذخیره کنید.

۸ برنامه را اجرا کنید. (کُد کامل این پروژه را می‌توانید در پوشهٔ c:\vbnetsbs\chap22\webcalculator پیدا کنید.)

۹ برنامه را مانند قبل تست کنید.

۱۰ اکنون روی دکمهٔ Get Help کلیک کنید، تا (پس از چند لحظه) صفحهٔ دوم در اینترنت اکسپلورر باز شود:



۱۱ پس از خواندن متن این صفحه، دکمهٔ Back اینترنت اکسپلورر را کلیک کنید، تا به صفحهٔ WebCalculator برگردید.

۱۲ کمی بیشتر با برنامه کار کرده، و سپس اینترنت اکسپلورر را ببندید.

با اینکه ASP.NET قابلیتهای زیادی دارد که دربارهٔ آنها صحبت نکردیم، اما با همین یک مثال ساده هم توانستید به قدرت و سادگی آن در ایجاد برنامه‌های وب پی ببرید.

به پایان آمد این دفتر، اما حکایت ویژوال بیسیک .NET همچنان باقیست! اکنون با مهارتهایی که از این کتاب بدست آورده‌اید، آماده‌اید تا قدم به دنیای برنامه‌نویسی بگذارید. فراموش نکنید که تجربه بهترین معلم است (البته نقش کتابهای خوب را هم نباید نادیده گرفت). **موفق باشید!**

مرجع سریع فصل ۲۲

برای ...	انجام دهید
ایجاد یک برنامهٔ وب	در دیالوگ «پروژهٔ جدید» آیتم ASP.NET Web Application راانتخاب کرده، و نام پروژه را وارد کنید.
وارد کردن متن در صفحهٔ وب یا HTML	ابتدا خاصیت pageLayout شیء DOCUMENT را به FlowLayout ست کرده، سپس روی صفحه کلیک کرده، و متن را وارد کنید.
فرمت کردن متن در صفحهٔ وب یا HTML	متن را در حالت FlowLayout انتخاب کرده، و با ابزارهای میله ابزار فرمت آنرا فرمت کنید.
دیدن کُد HTML صفحهٔ وب یا صفحهٔ HTML	روی برگهٔ HTML در پائین «طراح فرمهای وب» کلیک کنید.
نمایش نقاط شبکه روی فرم وب	خاصیت pageLayout شیء DOCUMENT را به GridLayout ست کنید.
اضافه کردن کنترل به صفحهٔ وب	کنترل موردنظر را از برگهٔ Web Forms جعبه ابزار ویژوال بیسیک انتخاب کرده، و روی فرم رسم کنید.
تغییر دادن نام کنترلهای وب	از خاصیت ID شیء موردنظر استفاده کنید.
نوشتن روال رویداد برای کنترلهای وب	روی شیء موردنظر دو-کلیک کنید، تا فایل کُد پشت صحنه باز شود،و سپس دستورات لازم را در آن بنویسید.
تعیین اعتبار داده‌های ورودی	از کنترلهای «تعیین‌اعتبارکننده» استفاده کنید.
ایجاد یک صفحهٔ HTML	پس از اجرای فرمان Project|Add HTML page ، نام صفحه را وارد کرده، Open را کلیک کنید.
ایجاد لینک به صفحات وب دیگر NavigateUrl	کنترل هایپرلینک را به صفحهٔ مبدأ اضافه کرده، و خاصیت آنرا به آدرس صفحهٔ مقصد ست کنید.

نمایه

آ

ب

پ

ت

ث،ج

چ

ح

خ

د

ر

س

ش

ص

ط

ع

ف،ق

ک

گ، ل

م

ن

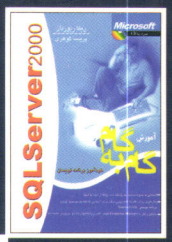
و

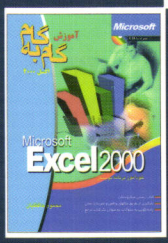
آموزش گام به گام

ویژوال بیسیک .NET

با مطالعه این کتاب نحوه انجام کارهای زیر را خواهید آموخت

- استفاده از ابزارهای ویژوال بیسیک .NET
- کار با کتابخانه‌های ویژوال بیسیک .NET
- دیباگ کردن برنامه‌های ویژوال بیسیک .NET
- نوشتن تابع و سابروتین
- استفاده از آرایه و کلکسیون
- اتوماسیون برنامه‌های آفیس
- ساخت کلاسهای پایه
- برنامه‌نویسی پایگاه‌داده با ADO.NET
- برنامه‌نویسی اینترنت با ASP.NET





همراه با یک CD-ROM
که حاوی متن کامل همه مثالها
و برنامه های این کتاب است.

شابک: ۰-۶۸-۵۸۰۱-۹۶۴
9679645801603

تهران- میدان انقلاب خ اردیبهشت بن بست مبین شماره ۲۳۷
تلفکس: ۶۹۵۳۸۸۳ تلفن: ۶۴۱۲۳۸۵ ص.پ. ۸۶۳-۱۳۱۴۵