آموزش مبتدی Microsoft Visual Basic .NET

نویسندگان :

امیر احسانی amir@ehsani.org

حامد بنایی hamed@hamedbanaei.com


NICE LIBRARY

درباره کتاب

چه کسی کتاب را بخواند؟

از آنجا که این کتاب از مفاهیم مقدماتی برنامه نویسی آغاز می شود؛ هر کسی، حتی بدون داشتن آشنایی با زبانهای برنامه نویسی می تواند این کتاب را بخواند. عملا تنها پیش‌نیاز این کتاب داشتن آشنایی با ویندوز است. به هر حال، کسی که می خواهد برنامه نویسی ویندوز را یاد بگیرد باید یک کاربر خوب ویندوز باشد.

در نگارش این کتاب سعی ما بر این بوده است که مبانی ریاضی چندانی لازم نداشته باشد تا برای دانش آموزان دبیرستانی علاقه مند نیز مفید باشد. بنابراین پایه ریاضی بیشتر مفاهیم کتاب در حدی است که یک دانش آموز اول دبیرستان به راحتی آن را بفهمد.

چطور کتاب را بخوانیم؟

ما توصیه می‌کنیم قدم به قدم با خواندن کتاب، کدها را تایپ کنید. یادگرفتن برنامه‌نویسی به هوش و استعداد فوق‌العاده‌ای احتیاج ندارد. فقط باید به اندازه کافی تمرین کنید. به همین دلیل در پایان هر فصل تعدادی تمرین، کارگاه و پروژه آمده است. تمرینها از مفاهیمی هستند که در طول فصل توضیح داده شده‌اند و اگر نتوانستید به آنها پاسخ بدهید بهتر است مفاهیم فصل را مرور کنید. کارگاه‌ها، از مفاهیمی هستند که در طول فصل توضیحی درباره آنها داده نشده است و برای حل آنها باید تلاش بیشتری بکنید و کمی از خلاقیت خودتان استفاده کنید و در ویژوال بیسیک و MSDN به جستجو بپردازید. اما اگر نتوانستید پاسخ را پیدا کنید، نا امید نشوید؛ برای هر کارگاه یک راهنمایی در پایان فصل آمده است و پس از آن پاسخ کارگاه نیز ذکر شده است.

در نهایت اینکه، ما توصیه می کنیم قبل از اینکه از فصل system.io (در نسخه اینترنتی صفحه 131) جلوتر بروید، به مفاهیم پیش از آن مسلط شوید. فصلهای پایانی به نسبت سنگینتر از سایر بخشهای کتاب است.

امیدواریم این کتاب، که حاصل حدود شش ماه تلاش ما است برای شما مفید باشد.

امیر احسانی (amir@ehsani.org)
حامد بنایی (hamed@hamedbanaei.com)

مقدمه ای بر دات نت

برای اینکه بفهمیم چرا دات نت به نیازی اساسی برای برنامه نویسی تبدیل شده است؛ باید بدانیم قبل از دات نت برنامه نویسی چه شرایطی داشته است.

نگاهی به گذشته

IBM PC که در سال 1981 معرفی شد؛ دارای یک Floppy drive ، بدون دیسک سخت و یک نمایشگر سبز فسفری با 128 کیلوبایت رم بود. این کامپیوتر توسط مجله Time (در حالی که عکسش روی جلد مجله بود) به عنوان مرد سال معرفی شد . این کار بسیار عجیبی بود. زیرا روی جلد این مجله فقط جای عکس رهبران و انسانهای مشهور بوده است.

پیشرفت کامپیوتر بقدری سریع بود که تنها 20 سال بعد، کامپیوتر به چنان دستگاه قدرتمندی تبدیل شد که از اینترنت یا وسایل بی سیم برای ارتباط با دیگر کامپیوتر ها استفاده می کرد. در حال حاضر همان طور که مشاهده می کنید سیستم های کامپیوتری (از جمله خدمات اینترنتی) در تمام ابعاد زندگی انسان نفوذ کرده اند. حتی در ایران که جزو کشور های در حال توسعه است و سیستم مخابراتی مناسبی ندارد؛ این امر کاملاً مشاهده می شود. اگرچه هنوز میزان استفاده ما و نحوه آن با کشورهای صنعتی بسیار متفاوت است. نکته جالب در مورد کامپیوتر این است که این علم در همین جا متوقف نشده است و ما تقریباً هر هفته شاهد نوآوری جدیدی در زمینه سخت افزار یا نرم افزار هستیم.

همپای پیشرفت سخت افزار، نرم افزارها و زبانهای برنامه نویسی هم پیشرفت کردند. بطوری که در اوایل دهه 1980 برنامه نویسان از زبان C برای کارهایشان استفاده می کردند. در آن زمان سیستم عامل ها Single Task بودند. یعنی هر برنامه ای که اجرا میشد تمام منابع سیستم عامل را در اختیار میگرفت و تا زمانی که کار آن برنامه تمام نشده بود، برنامه دیگری نمیتوانست اجرا شود.. سیستم عامل MS-DOS (که یک سیستم عامل Single Task بود) در این زمان ارائه شد. MSDOS به دلیل استفاده از assembler سیستم عامل پر سرعت و کم حجمی بود. API[1] های DOS تنها مجموعه ای از وقفه ها بودند.

در اوایل دهه 80 پایه های API های ویندوز با استفاده از زبان C نوشته شد. این API ها مخفیانه طراحی می شدند تا کسی از نحوه کار آنها با خبر نشود. در آن API ها صدها تابع با نام های طولانی و بعضاً مرموز وجود داشت. انتقال به ویندوز برنامه نویسی را دچار تحول اساسی کرد . برنامه نویس ها می بایست برای برنامه های خود طراحی گرافیکی نیز انجام داده و از خواص multi tasking نیز استفاده می کردند.سیستم عاملهای multi task بر خلاف single taskها این قابلیت را دارد که چندین برنامه را بطور هم زمان اجرا کند.

در آن زمان خیلی از برنامه نویسان در برابر استفاده از ویندوز ایستادگی کردند و می گفتند برنامه نویسی در ویندوز کند ، زمان گیر و پیچیده است . شاید وقتی قسمت های بعدی این کتاب را ببینید بگویید برنامه نویسی در ویندوز بسیار کار ساده ای است، اما در آن زمان ابزارهای فعلی وجود نداشت . برای یک برنامه ساده ای که قرار بود در ویندوز اجرا شود و محیط گرافیکی داشته باشد؛ باید هزاران خط کد نوشته میشد.

البته هم اکنون نیز بعضی از برنامه ها برای کنترل دقیق تر از ابزارهای low level[2] تری استفاده می کنند .

به دلیل این تغییرات و راحتی در استفاده ، ویندوز در بسیاری از کامپیوتر های رومیزی به کار گرفته شد. در سال 1991 اولین نسخه ویژوال بیسیک ارائه شد . زبانی ساده ولی کارا برای کسانی که خود را درگیر پیچیدگی های برنامه نویسی C نمی کردند .

ویژوال بیسیک تاریخ جالبی دارد . زبانی که به آرامی رشد کرد و اکنون به یکی از متداول ترین زبانهای دنیا تبدیل شده است . ویژوال بیسیک 3 به ما امکان استفاده از بانک های اطلاعاتی را می داد . ویژوال بیسیک 4 امکان کامپایل کردن برنامه برای سیستم عاملهای 16 بیتی و 32 بیتی را فراهم کرد ، همچنین در این نسخه امکان نوشتن COM یا DLL ایجاد شده بود . در نسخه 5 امکان ساختن ActiveX Control ها محیا شد و در آخر ، در نسخه 6 کل برنامه از ابتدا نوشته شد . کامپایلر[3] که از نسخه 5 از حالت مفسری[4] درآمده بود سرعت بیشتری به برنامه ها داد . همچنین امکان ساختن control ها برای وب و interface و inheritance نیز اضافه شد .

دنیای کامپیوتر بار دیگر نیز متحول شد و این بار اینترنت محور اصلی تغییرات است. نیاز ما به اینترنت حتی با سال گذشته قابل مقایسه نیست . امروزه اکثر برنامه ها از اینترنت استفاده می کنند حتی در استراتژی جدید بعضی شرکت های بزرگ تغییر ایجاد شده و دیگر برنامه ها برای نصب بر روی سی دی ارائه نمی شود و همه چیز web based خواهد بود .

---

[1] Application Programming Interface

[2] سطح پایین: در کامپیوتر ، سطح پایین بودن به معنی نزدیکتر بودن به سخت افزار است.

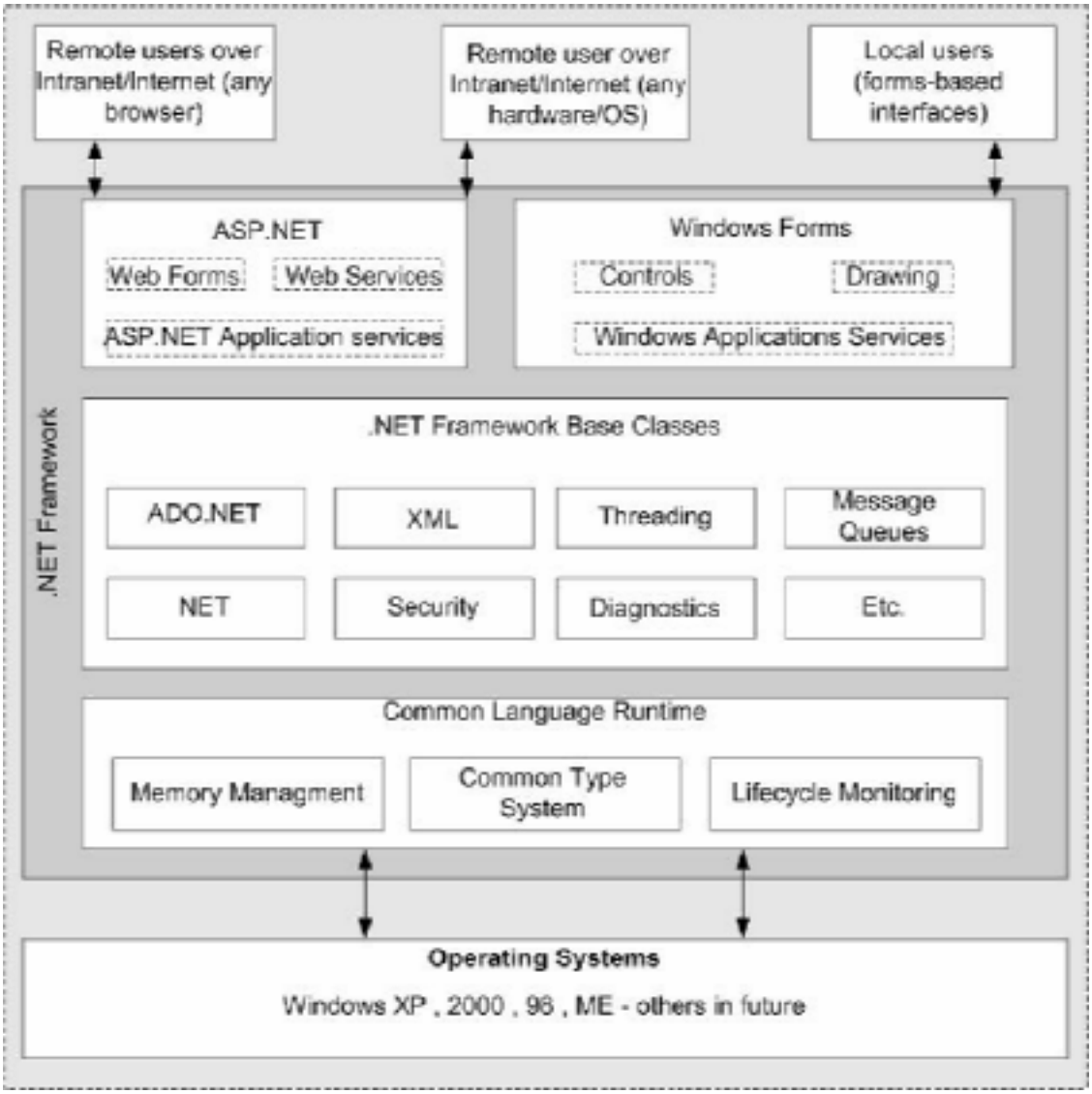
[3] به برنامه ای گفته میشود که برنامه نوشته شده در یک زبان سطح بالا مانند ویژوال بیسیک را به زبان ماشین ترجمه میکند.

[4] برنامه ای که برنامه نوشته شده به یک زبان سطح بالا را بدون ترجمه کردن به زبان ماشن و بصورت خط به خط اجرا میکند.

دنیای .NET
تمام این تغییرات دست به دست هم دادند تا نیازهایی را به وجود آورند که در نهایت منتهی به ایجاد دات نت شد. در دات نت کاربران می توانند اطلاعاتشان را در محیط ، زمان ، مکان یا هر دستگاهی بازیابی کنند . چند سال پیش استفاده از تلفن همراه برای بازیابی اطلاعات کار بسیار دشواری بود در حالی که هم اکنون با استفاده از تکنولوژی دات نت می توانید برای گوشی تلفن همراه خود برنامه بنویسید . حتی مایکروسافت تلاش میکند پای دات نت را به دستگاه های خود پرداز (ATM) نیز باز کند . در حال حاضر پروژه هایی در دست اجراست که برنامه هایی که با دات نت نوشته می شود را بتواند بر روی سیستم عامل های دیگر اجرا کند .
برا اینکه کمی بیشتر با مزایای دات نت آشنا شویم مثالی از یکی از کاربردی ترین امکانات دات نت را ذکر می کنم . فرض کنید برنامه تجاری داریم و شرکتی از آن استفاده می کند که دارای چندین دفتر در سراسر کشور است . پس از مدتی مدیر تصمیم می گیرد حقوق گروه خاصی از کارمندان را 0.5 درصد افزایش دهد . در مدل قدیمی اعمال چنین تغییری می توانست مشکلاتی به بار آورد ، اگر برنامه از DCOM استفاده می کرد مشکل اول پیاده سازی خودِ DCOM بر روی اینترنت و بستر مخابراتی ایران بود و مشکل دوم بروز رسانی آن . در ویندوز همان طور که می دانید DLL ها نصب و در registry کدی برای آنها نوشته می شود . هر DLL یا COM شماره ای مانند 1.1.2.002 برای مشخص کردن نسخه آن است . اکر نسخه جدید DLL ما که حاوی اطلاعات حقوق است را بدون در نظر گرفتن این نکته نصب می کردیم امکان داشت نسخه قبلی و جدید با هم تداخل پیدا کنند و برنامه دیگر کار نکند . در دات نت با استفاده از تکنولوژی به نام وب سرویس این کار به راحتی انجام می شود . کافی است اطلاعات جدید برای استفاده روی سرور قرار گیرد برنامه های دفترهای سراسر کشور یا online از وب سرویس استفاده می کنند یا اطلاعات جدید را از آن گرفته و بصورت Offline آنها را به کار می گیرند .
مشکلی که در بالا برای DLL ها به آن اشاره شد DLL Hell نام دارد ، این مشکل در دات نت رفع گردیده است . DLL هایی که در دات نت ساخته می شوند احتیاج به نصب شدن ندارند . کافی است فقط آن را کپی کنیم برنامه ما هر مقدار هم بزرگ و پیچیده باشد احتیاج به نصب ندارد . کافی است با دستور xcopy آن را کپی کنیم؛ هر چند توصیه می شود برای کاربر نهایی(end user) با استفاده از installer یک نسخه آماده نصب بسازیم زیرا برنامه installer میتواند علاوه بر کپی کردن فایلهای برنامه ما، روی دسکتاپ Icon ایجاد کند و برنامه را به Programs در منوی start اضافه کند. قبل از دات نت اگر ما برنامه ای را uninstall می کردیم امکان داشت DLL هایی را که مربوط به دیگر برنامه ها بود را هم حذف کند و این باعث می شد برنامه های دیگر هم از کار بیافتد. ولی در دات نت هر برنامه می تواند کپی مخصوص خود را داشته باشد و در هنگام uninstall شدن فقط فایلهای مربوط به همان برنامه حذف شود .

توضیحاتی در مورد **.NET Framework**

.NET Framework. برنامه ای است که به عنوان یک لایه جدید روی سیستم عامل قرار می گیرد ، خودِ دات نت از لایه های مختلفی تشکیل شده است .



قلب ِ دات نت CLR یا Common Language Runtime است . CLR مسئول اجرای برنامه هاست . با استفاده از خصوصیات CLR امکان استفاده از برنامه هایی که در یک زبان برنامه نویسی نوشته می شود در زبان دیگر وجود دارد . در دات نت نوع های داده ها توسط CLR یکسان سازی شده اند تا در هنگام استفاده از آنها در زبان های دیگر مشکل پیش نیاید . البته این هدف باعث شد بعضی از انواع داده ها که در ویژوال بیسیک 6 مورد استفاده قرار می گرفت مانند Variant در دات نت حذف شود .
همان طور که در شکل 1-1 مشاهده می کنید در بالاترین سطح کامپایلرهای سی شارپ ، ویژوال بیسیک ، C++ و دیگر زبان ها وجود دارد . در قسمت بعدی CLS یا Common Language Specifications قرار گرفته است . این لایه مسئول بررسی این است که آیا کامپایلر و زبان خصوصیات پایه ای برای CLR یا تبادل اطلاعات با دیگر CLS ها را دارد یا خیر . با این وسیله تضمین می شود که وقتی شرکتهای دیگر زبان یا کامپایلر های دیگری برای دات نت بنویسند محصول آنها با دیگر محصولات هماهنگ خواهد بود .
Web Services مسئول ایجاد محیط کاربری تحت وب (User interface) است. علاوه بر این کنترل ارتباط با وب ، پروتکل های آن ، امنیت و دیگر مسائل مربوط نیز برعهده این قسمت می باشد . در دات نت برنامه ای که برای windows form یا win32 app نوشته شود؛ با کمی تغییر می تواند تحت وب و در web forms اجرا شود .
در مقابل web services قسمت user interfaces قرار دارد . User Interface مسئول برقراری ارتباط در محیط ویندوز با استفاده از windows forms است .
Data و XML لایه های بعدی هستند . دات نت برای انتقال اطلاعات بر روی وب از XML استفاده می کند .

BCL یا Basic Class Library مسئول نگهداری کلاسهای اصلی دات نت است .هر چیزی در دات نت یک کلاس است و همه کلاس ها از کلاس اصلیِ System منشعب می شوند (در فصل های آینده مطالب کاملی در این مورد ذکر می کنیم) . اطلاعات کامل تر در مورد CLR و ساختارش ، JIT Compiler ، Meta Data و ... را به کتاب پیشرفته موکول می کنیم .

آشنایی با محیط ویژوال استدیو دات نت

نصب ویژوال استودیو دات نت

در ایران نسخه های مختلفی از ویژوال استودیو دات نت وجود دارد که بعضی از آنها کامل نیستند. بعضی نسخه ها مثال ها(samples) را ندارند و بعضی دیگر راهنمای MSDN را . لذا در هنگام نصب باید این نکته را در نظر داشته باشید که ممکن است در مراحل نصب بعضی فایلها وجود نداشته باشند و برنامه نصب کننده پیغام های خطای زیادی بدهد . بهتر است در صورت مواجه شدن با چنین حالتی از نصب خارج شده ، دوباره آن را شروع کنید ، ولی این بار فراموش نکنید که فایلهایی را که وجود نداشتند انتخاب نکنید .

اولین مرحله نصب، بروز رسانی سیستم شماست . نصب NET Framework. و IE6 از مراحل اساسی برای دات نت است. البته باید توجه داشته باشید که ویژوال استودیو دات نت بر روی ویندوز های NT نصب می شود و برنامه ای که با دات نت می نویسید بر روی ویندوز 98 به بالا قابل اجرا خواهد بود . دیسک بروز رسانی معمولاً دیسک آخر از سری سی دی هایی است که شما خریداری کردید . وقتی از دیسک اول فایلهای برنامه نصاب کپی شد از شما دیسک Windows Component Update را می خواهد که همان دیسک به روز رسانی ماست . بعد از بروز رسانی و احتمالاً چند بار راه اندازی مجدد سیستم دوباره دیسک اول را از شما می خواهد .

در این مرحله شما برنامه هایی که مورد نیازتان است را از لیست انتخاب می کنید تا نصب شود . اگر دیسک سخت شما جای کافی دارد از Language Tools هر دو زبان ویژوال بیسیک و سی شارپ را انتخاب کنید ، در صورتی که از کامل بودن دیسکهای خود مطمئن هستید اجازه دهید تمامی برنامه هایی که خود برنامه نصب کننده انتخاب کرده است نصب شود . مسیر و مکانی که برای کپی شدن فایلها مورد نظرتان است را در سمت راست می توانید انتخاب کنید . همچنین حجمی که فایلهای انتخاب شده اشغال می کند و فضای خالی پارتیشن ها نیز در سمت راست مشخص است . بعد از انتخاب های مناسب ، بر روی Install Now کلیک کنید . بستگی به میزان فایلهایی که انتخاب کردید زمانی در حدود نیم ساعت طول می کشد تا برنامه نصب شود و شما پیغام Done را ببینید .

مرحله بعدی بروز رسانیِ خود ویژوال استودیو دات نت است . ویژوال استودیو دات نت نیز دارای سرویس پک است که می توانید آن را از سایت مایکروسافت دریافت کنید .

اجرا

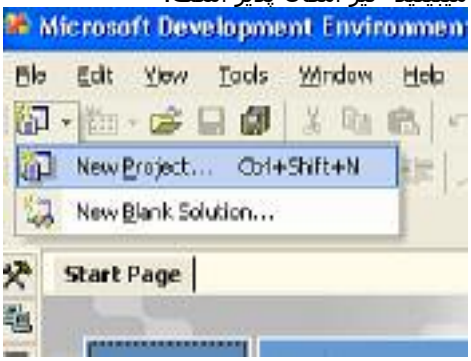
بعد از نصب می توانید ویژوال استودیو را از منوی Programs انتخاب و اجرا کنید . در ویژوال استودیو دات نت بر خلاف نسخه های قبلی تمامی محیط های برنامه نویسی از یک IDE استفاده می کنند و دیگر مانند ویژوال بیسیک 6 و ویژوال سی 6 و InterDev نیست که محیط هایی کاملاً مجزا داشته باشند .

شکل ظاهری ویژوال استودیو را بسته به علاقه ای که دارید می توانید انتخاب کنید به چه صورت باشد .

ما در اینجا همه چیز را همان پیش فرض های ویژوال استودیو دات نت انتخاب می کنیم . البته می توان همه چیز را برای ویژوال بیسیک تغییر داد ، ولی ما معتقدیم در دات نت باید بتوانیم از همه قابلیت ها استفاده کنیم و خودمان را محدود به بیسیک یا سی شارپ نکنیم ، اعمالی هست که در یکی آسان تر از دیگری انجام می شود ، پس بهتر است از قابلیتهای هر دو بهره ببریم.

همان طور که در شکل می بینید سه بخش در صفحه اصلی وجود دارد ، بخش MyProfile نیاز به توضیح اضافه ندارد . بخش Projects قسمتی است که آخرین پروژه های باز شده را لیست کرده است ، همچنین دکمه هایی برای باز کردن پروژه هایی که در لیست وجود ندارند و ساختن پروژه جدید وجود دارد . در بخش Online Resources نیز (در صورتی به اینترنت متصل باشید) امکاناتی نظیر دریافت فایل ، نمونه برنامه ، اخبار و دیگر عناوین وجود دارد .

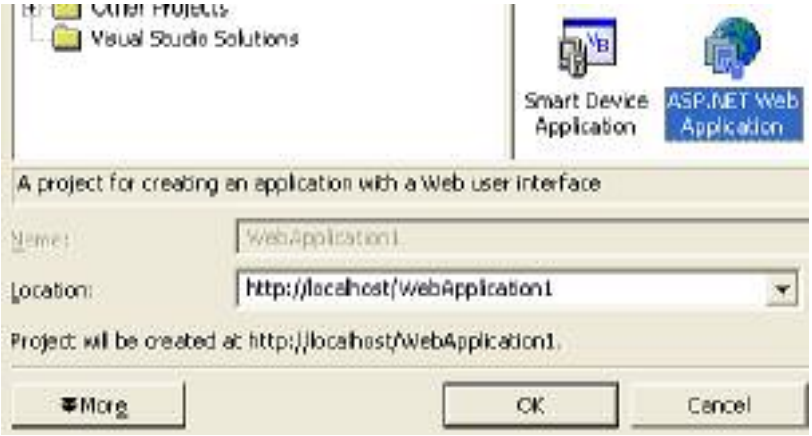
حـالا بـرای شـروع کـار از قسـمت Projects روی New Project کلیـک کنیـد . البتـه همـین کـار را از منـوی File>New>Project هم می توانید انجام دهید این کار از طریق Toolbar - از آیکـونی کـه شـکلش را در زیـر میبینید- نیز امکان پذیر است.



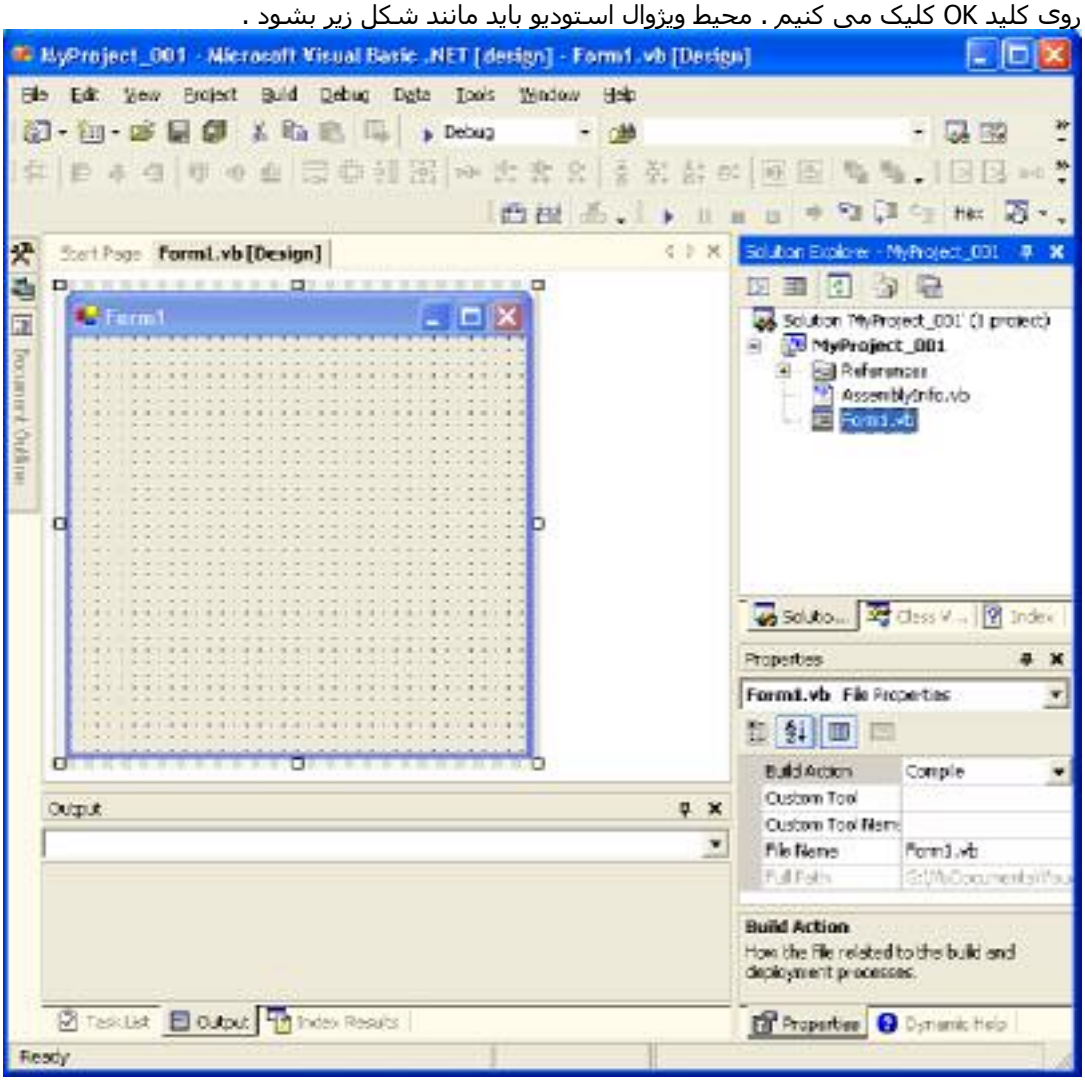
وقتی صفحه New Project باز شود شکلی مانند زیر به شما نمایش داده می شود .

همان طور که در شکل می بینید در سمت چپ نوع کلی پروژه و در سمت راست نوع دقیق آن را می توانید مشخص کنید . انواع کلی مانند پروژه های ویژوال بیسیک ، پروژه های سی شارپ یا پروژه هایی کـه بـرای ساختن برنامه نصاب به کار می روند می باشد . وقتی روی هر نوع کلی کلیـک کنیـد انـواع پـروژه هـای زیـر مجموعه آن را در سمت راست نمایش می دهد . در شکل بالا بعضی از انواع پروژه هـای ویـژوال بیسـیک را می بینید ، مانند ، Windows Application، Class Library ، ASP.NET Web Application یا Smart Device Application و .... .

توجه کنید با کلیک بر روی هر کدام از انواع پروژه ها روش نام گذاری و مکان ذخیره سازی آن نیـز تغییـر مـی کند . در مدل Windows Application شما نام پروژه خود را می نویسید و جعبـه مـتن Location پوشـه کلـی آن را انتخاب می کنید . در شکل زیر مدل ASP.NET را می بینید که در آنجا نحوه نام گذاری متفاوت است .



مسیرهایی که به عنوان مسیر پیش فرض وجود دارد در منوی Tools>Options قابل تغییر است . برای ادامه کار یک Windows Application انتخاب و برای نام از myProject_001 استفاده می کنیم . سپس بر روی کلید OK کلیک می کنیم . محیط ویژوال استودیو باید مانند شکل زیر بشود .



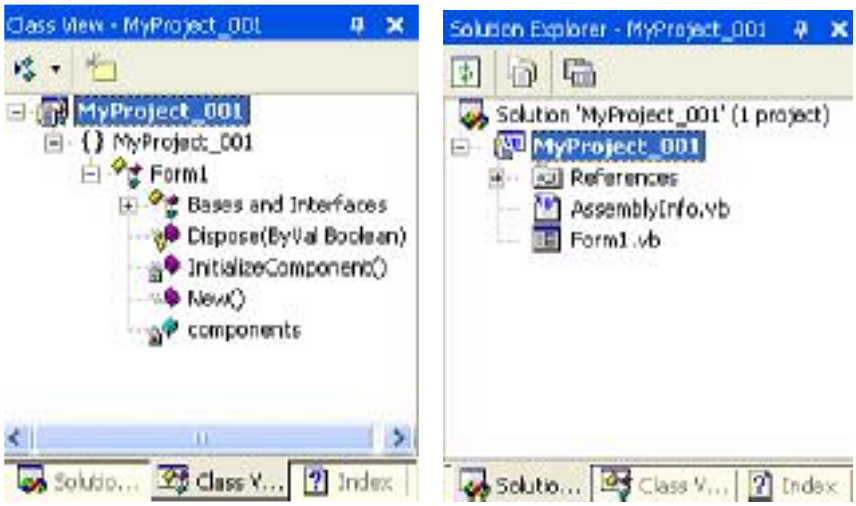
محیط ویژوال استودیو از چند بخش مهم تشکیل شده است .

Solution Explorer

Properties
Toolbox
Server Explorer و محیط کد نویسی و طراحی فرم .

Solution Explorer

این قسمت همان طور که در شکل زیر مشخص است نمایش دهنده کلیه فایل ها و کلاس های پروژه ما است .



در مورد کلاس ها و نحوه ساختن آنها در فصل های آینده مطالبی را خواهیم دید .
در ویژوال استودیو اصطلاحی به نام Solution به معنی "راه کار" وجود دارد؛ هر Solution می تواند حاوی تعدادی پروژه باشد . همان طور که در شکل راست می بینید شاخه اصلی درخت فوق یک Solution است و MyProject_001 یک پروژه یا یک شاخه از آن . با double click بر روی هر کدام از فایلها یا شاخه های درون Solution Explorer محتویات آن فایل یا شاخه نمایش داده می شود .

Properties

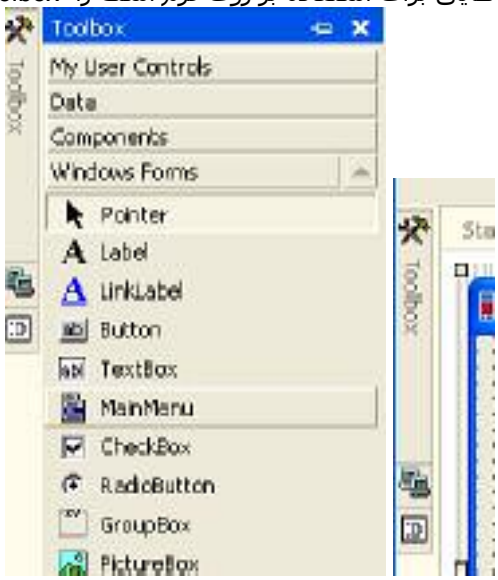
قسمت مهم دیگر پنجره Properties است . Property به معنی خاصیت می باشد (با این مفهوم در بحث OOP بیشتر آشنا می شویم) ، در این پنجره شما خاصیت های هر یک از اجزایی که بر رویش کلیک کرده اید را می بینید . در شکل زیر خواص فرم اصلی که به شکل پیش فرض در پروژه وجود دارد را می بینید .

درباره خواص معروف کنترل ها در بخش های بعدی توضیحاتی خواهید دید ولی در حال حاضر جلوی خاصیت Text عبارت myFirstApp را به جای Form1 می نویسیم و کلید Enter را می زنیم . اگر دقت کنید عنوان فرم ما تغییر کرد .

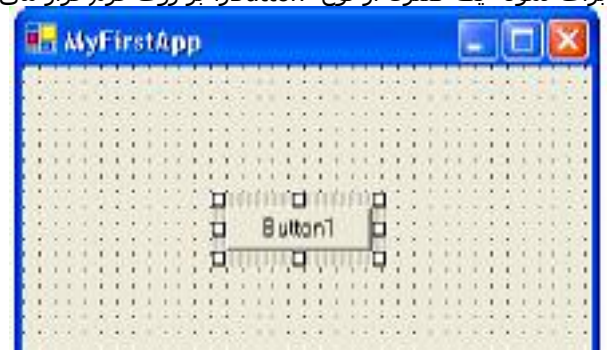
پنجره دیگری که دقیقا در همان مکانی که Properties وجود دارد، قرار گرفته است Dynamic Help است . در صورتی که ویژوال استودیو دات نت را کامل نصب کرده باشید، هنگام برنامه نویسی یا هنگام ساختن شکل ظاهری فرم یا هر چیزی که تایپ می کنید؛ اگر ویژوال استودیو اطلاعاتی در باره آن داشته باشد، در آن پنجره نمایش می دهد . این امکان در صورتی خوب است که کامپیوتر بسیار سریعی داشته باشید زیرا برای هر کلمه ای که تایپ کنید ویژوال استودیو MSDN را جستجو می کند. .

Toolbox

پنجره ای که در بخش کناری سمت چپ قرار دارد و اگر موس را روی آن ببرید فعال می شود و حاوی کنترل هایی برای استفاده بر روی فرم است را Toolbox می نامند .



کنترل هایی که در این منو و tabهای آن وجود دارد را می توانید با drag & drop بر روی فرم قرار دهید . برای نمونه یک کنترل از نوع Buttonرا بر روی فرم قرار می دهیم .

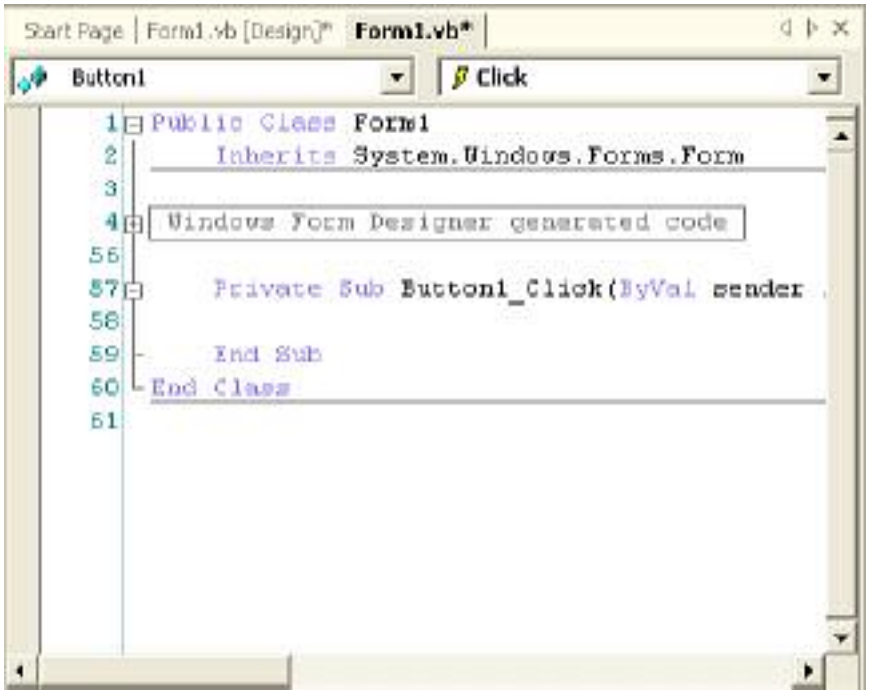


بر روی button یک بار کلیک می کنیم تا خواص آن در پنجره Properties نمایش داده شود . جلوی Text به جایButton1 عبارت Hello World را تایپ می کنیم و کلید Enter را می زنیم .

دو tab دیگر در زیر Toolbox قرار دارد ، اولی Server Explorer است و با دومی در کتاب های ASP.NET آشنا می شوید . در Server Explorer لیست سرور هایی مانند SQLServer که به آنها متصل هستید یا قبلاً متصل شده اید یا دیگر اجزای سرویس دهنده ویندوز وجود دارد . توضیحات تکمیلی در کتاب های پیشرفته تر خواهد بود .

همان طور که مشاهده می کنید همه این پنجره ها در اطراف صفحه به شکلی جاسازی شده اند ، می توانید با drag & drop آنها را از حالت dock خارج کنید تا در وسط یا هر جای دیگر قرار گیرند . البته به شما پیشنهاد می کنیم ترکیب فعلی را تغییر ندهید ، بهترین حالت همین حالتی است که طراحانِ مایکروسافت ایجاد کرده اند .

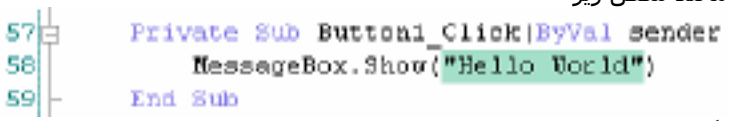
محیطی که در وسط صفحه دیده می شود همان محیط طراحی فرم است که باید شکل ظاهری برنامه خود را نقاشی کنید . می توانید عکس ، منو ، لیست ، جعبه متن و خیلی کنترل های دیگر بر روی آن قرار دهید . برای دیدن محیط کد نویسی دو بار بر روی Button ی که ساختیم کلیک می کنیم تا شکلی مانند زیر ظاهر شود .



اینجا محیط متنی است ، محیطی که بخش اصلی کار ما به عنوان برنامه نویس با آن می باشد . این قسمت از ویژوال استودیو دات نت نیز از شکل tab استفاده می کند ، یعنی همان طور که در بالای شکل می بینید نوشته است Form1.vs[Design] اگر بر روی آن کلیک کنید دوباره محیط طراحی شکل ظاهری فرم نمایان می شود . به همین ترتیب می توانید تعداد زیادی فرم یا فایل باز شده داشته باشید بدون اینکه مزاحم یکدیگر باشند . در مورد اینکه هر یک از بخش های این نوشته ای که می بینید بیانگر چه مطلبی است در فصلهای آینده توضیحاتی خواهید دید ولی در اینجا برای اینکه اولین برنامه مان را نوشته باشیم یک خط کد می نویسیم . بین Private Sub و End Sub ی که در شکل مشخص است این خط را می نویسیم .

```
MessageBox.Show("Hello World")
```

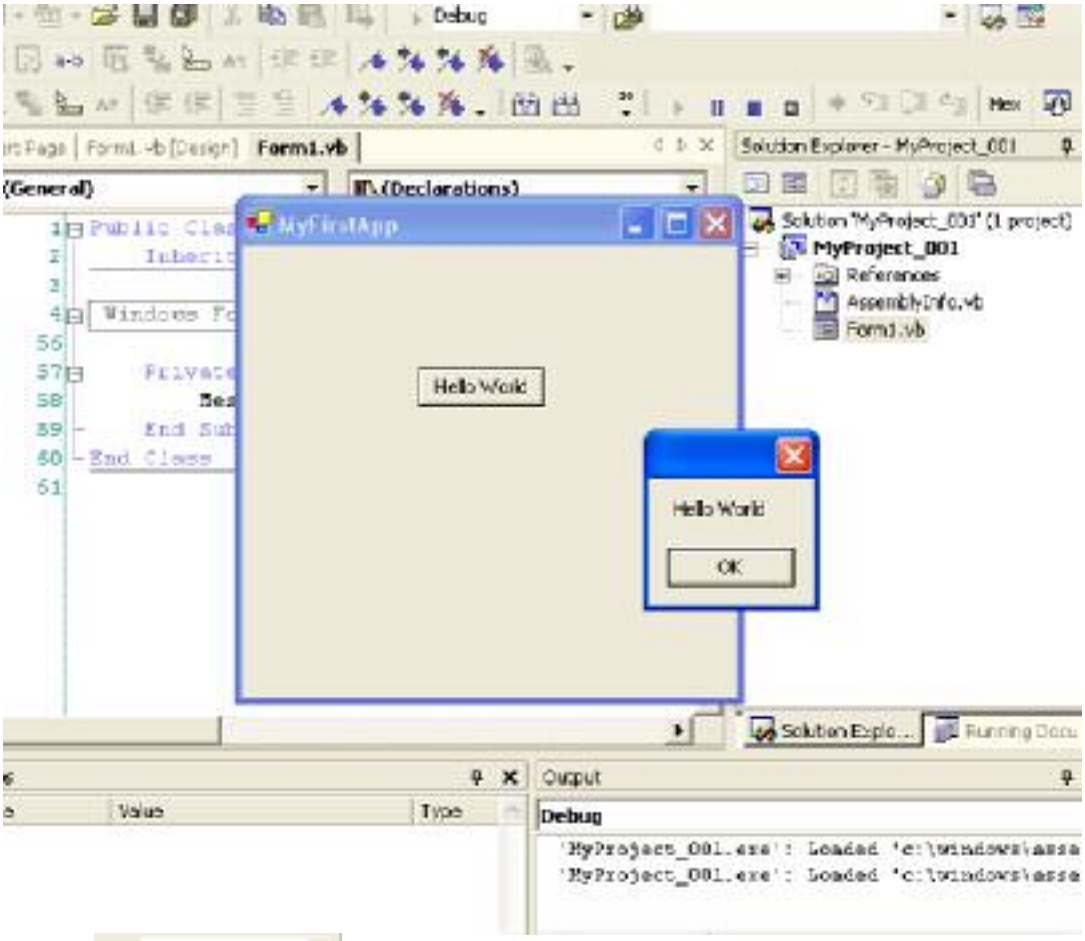
مانند شکل زیر



اگر دقت کرده باشید وقتی بعد از تایپ MessageBox نقطه را تایپ کردیم یک لیست برایمان باز شد و وقتی Show را تایپ کردیم عملاً مانند این بود که Show را از لیست انتخاب کنیم . به این قابلیت پیشرفته IntelliSence می گویند.

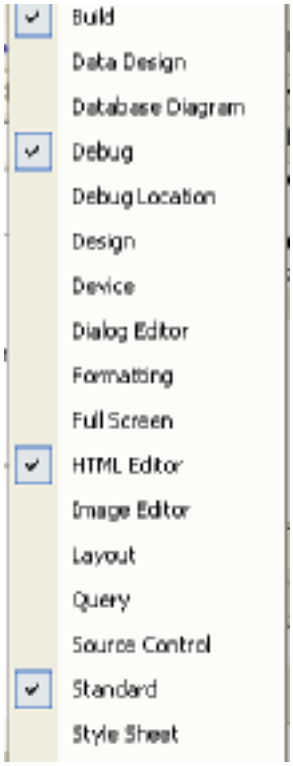
این یکی از مفید ترین تکنولوژی هایی است که مایکروسافت در اختیار برنامه نویسان قرار داده تا تمامی خواص و اجزای اشیاء را به خاطر نسپارند. ویژوال بیسیک بجای ما خواص اشیاء را حفظ میکند، ما فقط کافی است نگاهی به لیست بیندازیم .البته در مورد اینکه در این لیست چه چیزهایی نمایش داده می شود مفصل توضیح خواهیم داد .

حالا اولین برنامه ما حاضر است ، کافی است مانند تمامی نسخه های ویژوال بیسیک کلید F5 را بزنیم . برنامه ما اجرا می شود و وقتی بر روی کلید آن کلیک کنیم شکلی مانند زیر نمایش داده می شود .

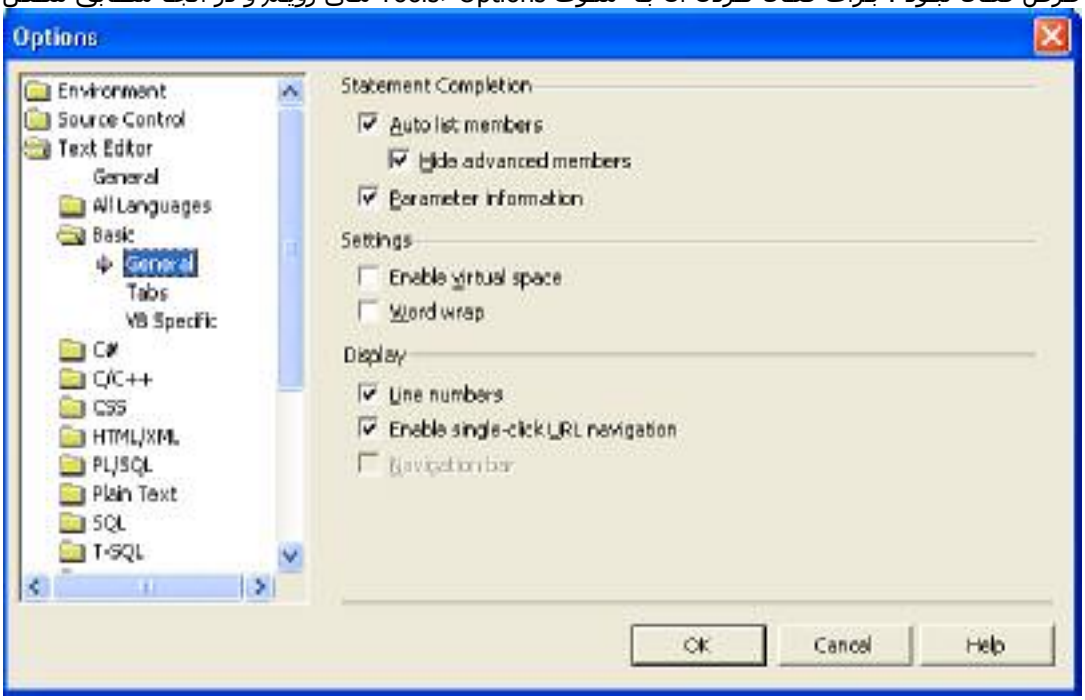


برای اجرا از toolbar شکلی مانند Play را هم می توانید انتخاب کنید :

ویژوال استودیو دارای toolbar های بسیار زیادی است که شما به عنوان کاربر حرفه ای ویندوز با اسامی آنها آشنایی دارید.



یکی از مفید ترین امکانات ویژوال استودیو امکان نمایش شماره خط برنامه است اگر این قابلیت بصورت پیش فرض فعال نبود ، برای فعال کردن آن به منوی Tools>Options می رویم و در آنجا مطابق شکل زیر



در قسمت Display علامت چک را برای Line Numbers فعال می کنیم .

مفاهیم پایه

آشنایی با الگوریتمها

هر برنامه کامپیوتری به منظور حل کردن یک یا چند مساله نوشته می شود. بعضی از مساله ها را میتوان به راحتی با یک فرمول ریاضی حل کرد ولی بسیاری از مساله ها با فرمولهای ریاضی قابل حل نیستند. فرض کنید به ما ده عدد غیر تکراری داده شده و از ما خواسته اند عدد x را در آنها پیدا کنیم. راه حلی که ما استفاده می کنیم این است که ابتدا عدد x را با اولین عنصر از آن ده عدد مقایسه می کنیم، اگر درست بود می گوییم اولین عدد x است و اگر درست نبود عدد x را با دومین عدد مقایسه می کنیم. این کار را آن قدر ادامه می دهیم تا مقدار یکی از اعداد برابر x باشد یا اینکه به انتهای لیست اعداد برسیم. به روش حل مرحله به مرحله مساله، الگوریتم مساله می گویند. که مبتکر آن خوارزمی، دانشمند ایرانی، است.
الگوریتم سه جزء پایه دارد. توالی، انتخاب و تکرار.

توالی

گفتیم الگوریتم روش حل مرحله به مرحله یا قدم به قدم یک مساله است. معمولا هر مرحله از الگوریتم را در یک خط جدا می نویسند و به آن یک دستور می گویند. دستورات بطور متوالی و پشت سر هم اجرا می شوند. برای مثال الگوریتم شکستن یک شیشه(!) بصورت زیر است:

1. یک سنگ از روی زمین پیدا کن
2. سنگ را در دست بگیر
3. شیشه مورد نظر را پیدا کن
4. هدفگیری کن
5. سنگ را بطرف شیشه پرتاب کن
6. پایان

یا الگوریتم اجرا کردن ویژوال بیسیک دات نت بصورت زیر است:

1. نشانگر موس را روی دکمه start ببر.
2. کلیک کن.
3. نشانگر موس را روی programs ببر.
4. کلیک کن.
5. نشانگر موس را روی شاخه Microsoft Visual Studio.Net ببر.
6. کلیک کن.
7. نشانگر موس را روی آیتم Microsoft Visual Studio.Net ببر.
8. کلیک کن.
9. پایان

دستورات یک الگوریتم بطور متوالی و از بالا به پایین اجرا می شوند و هر تغییری که یک دستور در محیط ایجاد کند روی دستورات بعدی تاثیر می گذارد. به الگوریتم زیر توجه کنید:

1. x را برابر 2+3 قرار بده.
2. y را برابر 10 قرار برده.
3. z را برابر x+y قرار بده.
4. z را برابر y-x قرار بده.
5. پایان

برطبق آنچه در مورد توالی آموختیم ابتدا x مقدار 2+3 میگیرد بنابراین مقدار آن برابر 5 میشود. این تغییر مقدار x از مقدار قبلی (که نمیدانیم چند بوده) به مقدار فعلی(عدد 5) میتواند روی دستورات بعدی تاثیر گذار باشد. دستور دوم مقدار قبلی y را به 10 تغییر میدهد. و دستور سوم مقدار x و y را با هم جمع کرده و در z قرار می دهد. منظور از مقدار x و y آخرین مقداری است که در آنها قرار گرفته. پس مقدار x برابر با 5 و y برابر با 10 است. در نتیجه مقدار z برابر با 15 خواهد شد. این همان تاثیر مراحل قبلی روی مرحله بعد از خودش است. خط چهارم نیز از مراحل قبلی تاثیر می گیرد اما دقت کنید که در این مرحله بار دیگر به z مقدار داده می شود پس مقدار فعلی z اهمیتی ندارد (همانطور که در مراحل یک و دو ، مقدار x و y قبل از مقدار دهی اهمیتی نداشت) و z برابر 5 میشود.

انتخاب

یکی دیگر از اجزای پایه یک الگوریتم، انتخاب است. ما در هر مرحله از یک الگوریتم میتوانیم بسته به شرایط تصمیم بگیریم که مرحله بعدی مرحله چندم باشد. الگوریتم روشن کردن کامپیوتر می تواند به شکل زیر باشد.

1. کلید power کامپیوتر را فشار بده.
2. کلید power مونیتور را فشار بده.
3. پایان

در نگاه اول این الگوریتم درست بنظر می رسد. ولی اگر مونیتور یا کامپیوتر روشن باشند، فشار دادن دکمه باعث خاموش شدن آنها می شود و پس الگوریتم درست کار نکرده است. الگوریتم اصلاح شده با استفاده از انتخاب در زیر آمده است :

1. اگر کامپیوتر خاموش است
1-1. کلید power کامپیوتر را فشار بده.
2. اگر مونیتور خاموش است
1-2. کلید power مونیتور را فشار بده.
3. پایان.

اگر شرط مرحله اول برقرار باشد، دستورات زیر مجموعه آن انتخاب می شوند. در غیر اینصورت آن دستورات انتخاب نخواهند شد. و دستور بعدی دستور 2 خواهد بود. برای مرحله دوم نیز شرایط به همین شکل است.

تکرار

سومین جزء پایه از اجزای الگوریتم، تکرار است. با استفاده از تکرار می توانیم یک یا چند مرحله (دستور) از الگوریتم را به تعداد دفعات خاص یا تا وقتی که شرایطی خاص برقرار شود تکرار کنیم. الگوریتم جستجویی که در ابتدای بحث مطرح کردیم در زیر بشکل ساختیافته تری آمده است :

1. x و n_1 تا n_{10} را دریافت کن.
2. i را برابر با 0 قرار بده.
3. کارهای زیر را تا وقتی y با x برابر نیست تکرار کن.
3-1. i را برابر i+1 قرار بده.
3-2. اگر i بزرگتر از 10 است برو به 5.
3-3. y را برابر با n_i قرار بده.
4. i را اعلام کن.
5. پایان

در اینجا مراحل زیر مجموعه 3 تا وقتی که y با x برابر نیست اجرا می شوند.

یک الگوریتم را میتوان مانند این مثالها به زبان فارسی نوشت، یا به زبان انگلیسی یا با یک زبان برنامه نویسی کامپیوتری. وقتی یک الگوریتم به یک زبان برنامه نویسی کامپیوتری بیان شود، به آن یک برنامه کامپیوتری می گوییم.

آشنایی با شی‌گرایی

آشنایی با شی گرایی از این جهت برای ما لازم است که ویژوال بیسیک دات نت یک زبان برنامه نویسی شی گرا است و تمام مفاهیم آن بر مبنای شی گرایی بنا نهاده شده.

چرا شی‌گرایی؟

اواخر دهه 1960 که شی‌گرایی اولین قدمهای خود را برمی داشت؛ زبانهای برنامه نویسی رویه ای مانند pascal، C و Fortran در اوج قدرت بودند. این نسل از زبانهای برنامه‌نویسی بطور ساده به کامپیوتر می‌گویند که چه کاری را انجام دهد. مثلا یک عدد از ورودی بگیر ، آن را 10 برابر کن و سپس نمایش بده. به عبارت دیگر این زبانها (زبانهای رویه‌ای) شامل لیستی از دستورات هستند که بدنبال هم اجرا میشوند.

اما وقتی همه دستورات بدنبال هم نوشته شود حتی بهترین برنامه‌نویسها هم نمی‌توانند از آن سر در بیاورند. پس برای سامان بخشیدن به این وضعیت هر برنامه را به تعدادی زیر برنامه(تابع) تقسیم کردند.

هرچند که این سازماندهی مشکلات را تا حدودی رفع می کند اما در برنامه های بزرگ، آشفتگی‌ها باز هم نمایان می‌شوند. مشکل این روش کجاست؟ در حقیقت مشکل اصلی مربوط به وضعیت داده‌ها در این روش برنامه نویسی است(در یک برنامه که 2 را با 3 جمع می کند، داده ها 2 و 3 هستند و در یک برنامه انبارداری، داده ها کالاهای موجود در انبار) .

کم ارزش بودن داده‌ها

در برنامه نویسی رویه ای (ساختیافته) داده ها بسیار کم ارزشند و تاکید بر روی "انجام یک کار" است. همانطور که گفتیم هر زیر برنامه "کاری انجام می دهد". اما برای ما داده‌ها بیشتر از توابعی که کارهایی بر روی آنها انجام می دهند

اهمیت دارند. برای مثال در یک فروشگاه آنچه بیشترین اهمیت را دارد، کالاها هستند؛ نه کسی که تعداد آنها را می‌شمارد یا کسی که آنها را به مشتری می دهد.

دسترسی به داده‌ها

در برنامه‌نویسی رویه‌ای اگر تعداد زیربرنامه‌هایی(توابعی) که می خواهند به یک داده دسترسی داشته باشند بیشتر از یکی باشد، باید آن داده بصورت سراسری تعریف شود و در اختیار تمام زیر برنامه ها قرار بگیرد. این وضعیت را میتوان به جا گذاشتن اسناد محرمانه در ورزشگاه صدهزار نفری آزادی تشبیه کرد! هرگز نمی توانیم اطمینان داشته باشیم که یک داده سراسری فقط توسط زیربرنامه هایی که ما مجاز میدانیم تغییر داده شود.

عدم شباهت به دنیای واقعی

یک زیربرنامه مشابه چه چیزی در دنیای واقعی است؟ داده‌ها مشابه چه چیزهایی هستند؟ اجازه بدهید در مورد یک مثال خاص صحبت کنیم. فرض کنید می خواهیم برنامه‌ای برای کنترل یک آسانسور بنویسیم. در یک چنین برنامه‌ای چه توابعی داریم؟ چه داده‌هایی داریم؟ در یک فروشگاه چطور؟ جواب دادن به این سوالها واقعا سخت است. چون توابع و داده ها هیچ معادلی در دنیای واقعی ندارند. در حقیقت توابع در برنامه نویسی رویه‌ای فقط یک تکه از برنامه هستند و دلالت بر انجام یک کار دارند. پس نمیتوان برای آنها نمونه‌ای در دنیای واقعی پیدا کرد.

شی گرایی

حال که دیدیم زبانهای برنامه‌نویسی رویه‌ای چه مشکلاتی دارند. میتوانیم به سوالی که در ابتدا پرسیدیم جواب بدهیم: "شی گرایی را به این دلیل انتخاب کرده‌ایم که بتوانیم برنامه خود را ساماندهی کنیم." این ساماندهی منجر به یک نظم منطقی می شود که در نهایت هزینه تولید نرم افزار را کاهش می دهد. شی گرایی تعدادی مفاهیم اولیه دارد که در دنیای واقعی با آنها سر و کار داریم و هیچکدام از آنها برای ما جدید نیستند. مفاهیم ساده‌ای از قبیل: طبقه(طبقه هایی از اشیاء) ، شی، داده ها، رویداد، قابلیت و ...

شی(Object) : بطور غیر رسمی میتوان مفهوم شی در برنامه نویسی شی گرا را همان مفهوم شی در دنیای واقعی دانست. همانطور که در دنیای واقعی اشیایی مانند میز، صندلی، چرخ گوشت، آسانسور و ... وجود دارند. در یک برنامه کامپیوتری هم می توانند وجود داشته باشند. برای مثال برنامه کنترل آسانسور یک شی آسانسور دارد. کمی بعد تعریف رسمی‌تری ارائه می‌کنیم.

کلاس (Class) : در مدرسه یاد گرفتیم که چطور اشیاء را طبقه بندی کنیم. بلوط را در طبقه درختها و گرانیت را در طبقه سنگها قرار میدادیم. مفهوم "کلاس" در برنامه نویسی شی گرا مانند مفهوم طبقه است. البته در مورد طبقه بندی، دنیای واقعی با برنامه‌نویسی شی‌گرا تفاوت کوچکی دارد. در دنیای واقعی اشیایی که وجود دارند را طبقه بندی می‌کنیم. اما در برنامه‌نویسی شی‌گرا اول یک کلاس(طبقه) تعریف می‌کنیم سپس شی را به عنوان نمونه‌ای از آن کلاس(طبقه) ایجاد می‌کنیم(شاید اگر دنیای واقعی را خودمان ایجاد می کردیم آنجا هم از همین روش استفاده می کردیم). به عبارت دیگر یک کلاس می گوید که اشیاء این کلاس(طبقه) چه داده هایی دارند. البته کلاس مشخص نمی کند که این داده ها چه مقدارهایی دارند. همچنین کلاس تعیین میکند که اشیایی که از این کلاس ایجاد می شوند چه متدهایی دارند و چه رویدادهایی می تواند برای آنها اتفاق بیفتد.

ویژوال بیسیک دات نت استاندارد حدود 7000 کلاس دارد، علاوه بر اینها ما میتوانیم کلاسهای مورد نیاز خودمان را از اینترنت دریافت یا خودمان کلاس جدیدی ایجاد کنیم.

داده ها : هر شی اطلاعاتی در اختیار ما قرار می دهد که به آنها داده (data) می گوییم. برای مثال شی صندلی در باره رنگ و وزنش اطلاعاتی به ما می دهد. بعضی از اطلاعات یک شی را می توان تغییر داد؛ می توانیم رنگ یک صندلی را تغییر دهیم. اما بعضی اطلاعات قابل تغییر نیستند. اطلاعات، در برنامه نویسی شی گرا به دو گروه تقسیم می شوند.به یک گروه آن فیلد (Field) و به دیگری خاصیت (Property) می گویند. بطور غیر رسمی از واژه خاصیت (Property) برای فیلدها نیز استفاده می شود.

تذکر : کلاس نمی گوید وزن شی چقدر است. کلاس فقط می گوید که نمونه‌هایی از اشیاء که از روی این کلاس(صندلی) درست می شوند داده ای به نام وزن دارند که یک عدد در خود نگهداری می کند .

متد (Method) : کاری که شی می تواند انجام دهد. توانایی یا قابلیت انجام یک عمل. مثلا شی آدم توانایی راه رفتن، شی آسانسور قابلیت بالا یا پایین رفتن و شی بستنی فروشی توانایی فروختن بستنی دارد. متدها بین نمونه‌های مختلف یک کلاس مشترکند و برخلاف خاصیتها، شی نمی تواند قابلیتهایش را تغییر دهد. در عمل متدها همان زیربرنامه‌های زبانهای رویه‌ای هستند که بطور سازمان یافته‌ای در برنامه‌نویسی شی‌گرا مورد استفاده قرار گرفته‌اند. به متدها تابع عضو هم گفته می شود.

رویداد (event) : رویدادها تغییر وضعیت شی را به ما اعلام می کنند. وقتی در زندگی روزمره می گوییم: "شیشه شکست." مانند این است که در برنامه نویسی شی گرا بگوییم: "رویداد شکستن شیشه فعال شد." وضعیت شیشه تغییر کرده است. وقتی کلید برق را "فشار می دهیم"، وضعیت آن تغییر می کند و رویداد "فشرده شدن" برای آن فعال می شود.

تعریف دقیقتری از شی : شی ترکیبی از داده‌ها، توابع عضو و رویدادها است. به عبارت دیگر: شی یک واحد داده خود شمول است که توابعی که برای کار با آن لازم است درون خودش قرار دارد.

مروی بر مشکلات زبانهای رویه‌ای

آیا واقعا شی‌گرایی مشکلات زبانهای رویه‌ای را حل کرده است؟ مشکل اول ارزش کم داده‌ها بود که این مشکل حل شده است. چون جزء اصلی برنامه‌نویسی شی‌گرا "شی" است که در حقیقت همان داده ما است. مشکل بعدی امنیت دسترسی به داده‌ها بود. در برنامه‌نویسی شی‌گرا هر شی می تواند داده‌های خود را طوری محافظت کند که هیچ شی دیگری جز خودش نتواند آنها را تغییر بدهد یا حتی ببیند! فقط شباهت با دنیای واقعی ماند. این شباهت بقدری زیاد است که ما برای توضیح دادن شی‌گرایی از دنیای واقعی کمک گرفتیم.

مقایسه با میز غذا خوری

رابرت لی‌فور در کتاب برنامه‌نویسی با C++ خودش برای نشان دادن تفاوت برنامه نویسی شی گرا و رویه‌ای از یک میز غذاخوری استفاده کرده است.
یک مهمانی با 30 مهمان(تابعها) را تصور کنید که همه دور یک میز نشته اند و غذا(داده‌ها) بین آنها روی میز است. دست همه مهمانها به همه غذاها میرسد و بخاطر همین هیچوقت از دیگری برای برداشتن غذا کمک نمی خواهند. یکی از مهمانها می خواهد کباب را از آنطرف میز بردارد و بشقابش به بشقاب یکی دیگر می خورد. آستین یکی در خورشت فرو می رود و دیگری بجای اینکه سوپ را در بشقاب خودش بریزد روی دست بقل دستی می ریزد! این وضعیتی است که در برنامه‌های رویه‌ای پیش می‌آید. حال فرض کنید که بجای یک میز 30 نفره از 6 میز 5 نفره استفاده شود. و غذاهای مختلفی روی میزها چیده شود. مهمانها سر میزهایی می‌نشینند که به غذای آن علاقه بیشتری دارند و هرکدام که غذای میز دیگر را خواستند، از یکی از کسانی که سر آن میز نشسته است؛ میخواهند که غذا را به آنها بدهد. این وضعیت شبیه وضعیت برنامه‌نویسی شی‌گرا است.

شی ما چه جزییاتی لازم دارد؟

آیا برای ایجاد یک شی باید همه جزییات آن شی را ایجاد کنیم؟ جواب منفی است. ما از شی فقط چیزهایی را ایجاد می کنیم که به آنها احتیاج داریم. در یک برنامه‌ی فروش بستنی،به یک بستنی فروشی احتیاج داریم. بستنی فروشی طول، عرض، ارتفاع، آدرس، رنگ در، شکل کاشی‌های دیوار، رنگ سقف، تعداد بستنی، تعداد نون بستنی، تعداد کارمند، تعداد و حجم یخچال و هزار و یک داده دیگر دارد. این شما هستید که تشخیص می دهید کدام داده برای شی بستنی فروشی در برنامه لازم و کدام اضافه است. در حقیقت ما قبول میکنیم که دنیای واقعی پیچیده‌تر از آن است که بتوانیم آن را بطور کامل ایجاد کنیم. پس با حذف کردن بعضی بخشهای اضافه (که به کار ما مربوط نیست) به بخشهای باقیمانده بیشتر می‌پردازیم. به داده‌ای که اینگونه خلاصه شده است، داده مجرد یا انتزاعی(abstract) هم گفته می شود. به داده مجرد بعلاوه توابع مورد نیازش یک "نوع داده مجرد(Abstract Data Type)" می گویند. حال می توانیم بگوییم که کلاس یک نوع داده مجرد(انتزاعی) است.

آیا شی باید در دنیای واقعی وجود داشته باشد؟

جواب خیر است. زبان برنامه نویسی هیچ اهمیتی نمی دهد که شیی که ما ایجاد می کنیم در دنیای واقعی وجود دارد یا خیر و هیچ اطلاعی هم در این باره نمی دهد. اگر هم شیی مانند آنچه می خواهید ایجاد کنید در دنیای واقعی وجود دارد لازم نیست حتما شی شما همان مشخصات را داشته باشد. شاید شما دوست داشته باشید شی آدمی با 3 چشم ایجاد کنید. زبان برنامه نویسی هیچ اهمیتی به این موضوع نمی دهد. شما کاملا در این زمینه آزادی عمل دارید.

تذکر : در نهایت اینکه هرچند زبانهای شی‌گرا به شما این قابلیت را می دهند که برنامه‌های شی گرا بنویسید؛ ولی این کاملا به طرز فکر شما بستگی دارد و اینکه تا چه حد شی گرا کار کنید. عملا میتوان در یک زبان شی‌گرا هم کاملا رویه‌ای برنامه نوشت. در این صورت بسیاری از قابلیتهای زبان را بکار نبرده‌اید.

داده های پایه در VB.NET

گفتیم شی نمونه ای از یک کلاس می باشد و از طرفی، شی از داده ها و توابع تشکیل شده است.داده هایی که در ایجاد یک شی استفاده می شوند انواع مختلفی دارند، پرکاربردترین و عمومی ترین این داده ها، داده های پایه هستند. در این فصل با داده های پایه در VB.NETآشنا خواهیم شد و خواهیم دید که VB.Net امکان ایجاد چه داده هایی را به ما میدهد. یا به عبارت دیگر چه نوع داده های پایه ای در VB.NET تعبیه شده است. بعضی از انواع داده اصلی در VB.NET عبارتند از :

نوع داده	برای ذخیره یک	مقدار نمونه
Short	عدد صحیح کوچک	30000
Integer	عدد صحیح بزرگ	123000000
Long	عدد صحیح بسیار بزرگ	1844674407370950
Single	عدد اعشاری	450.4350
Double	عدد اعشاری با دقت مضاعف	7.9228162514264335
Char	کاراکتر یونیکد	"د"
string	رشته از کاراکترهای یونیکد، یک متن	"Visual Basic .NET"
boolean	مقدار "false" یا "true"	true

نوع داده های صحیح

هر داده از نوع داده صحیح می تواند یک عدد صحیح در خود نگه دارد. اعداد صحیح، اعدادی هستند که می توانند مثبت یا منفی باشند ولی نمی توانند اعشار داشته باشند. ویژوال بیسیک سه نوع داده برای ذخیره کردن اعداد صحیح درنظر گرفته است. تفاوت نوع داده های صحیح مختلف با هم در اندازه عددی است که می توانند در خود نگه دارند و حافظه ای که اشغال می کنند است.

نوع داده صحیح	نام دیگر	محدوده متغیر	اندازه
Short	Int16	-32,768 تا 32,767	16bit (2byte)
Integer	Int32	-2,147,483,648 تا 2,147,483,647	32bit (4byte)
Long	Int64	-9,223,372,036,854,775,808 تا 9,223,372,036,854,775,807	64bit (8byte)

اگر بخاطر داشته باشید در فصل قبل گفتیم که اشیاء نمونه هایی از کلاسها هستند. در مورد نسبت متغیرها به نوع داده ها نیز همین رابطه برقرار است. نمی توان یک مقدار را در یک نوع داده ذخیره کرد بلکه باید ابتدا یک متغیر از روی آن نوع داده تعریف کرد و سپس مقدار مورد نظر را در آن متغیر ذخیر کرد. برای اینکه یک نمونه از روی یک نوع درست کنیم از ساختار دستوری مانند زیر استفاده میکنیم :

```
Dim variableName as TypeName
```

دستور Dim به VB میگوید که ما قصد داریم یک متغیر یا شی تعریف کنیم. variableName نام متغیر یا شی ما است و TypeName نشان می دهد که متغیر یا شی ما نمونه ای از کدام نوع داده است. در زیر تعریف یک متغیر به نام sTest1 از نوع short آمده است.

```
Dim sTest1 as Short
```

مثال زیر یک متغیر از نوع integer به نام iVar تعریف می کند.

```
Dim iVar as Integer
```

در یک خط می توان بیش از یک متغیر تعریف کرد، در این صورت متغیرها باید با "," از هم جدا شوند.

```
Dim var1 as Integer, Var2,Var3 as Long
```

در اینجا var1 از نوع Integer تعریف شده و متغیرهای var2 و var3 از نوع Long تعریف شده اند.

متغیر چیست؟

وقتی اجرای برنامه به دستوری مانند دستور بالا می رسد بخشی از حافظه اصلی(بخشی از RAM) کامپیوتر را از سیستم عامل می گیرد و به متغیر ما اختصاص می دهد. برنامه ما می تواند با استفاده از متغیر، مقدار این بخش از حافظه را تغییر دهد یا بخواند. به عبارت دیگر ما از متغیرها برای ذخیره و بازیابی داده ها در حافظه اصلی کامپیوتر استفاده می کنم. پس هرگاه بخواهیم یک عدد صحیح در حافظه کامپیوتر نگه داریم، لازم است که یک متغیر از نوع داده صحیح ایجاد کنیم.

جزییات حافظه ای که باید به متغیر اختصاص داده شود توسط نوع داده تعیین می شود. نوع داده short اعلام می کند که برنامه ما 2 بایت حافظه برای ذخیره یک عدد صحیح که می تواند منفی باشد نیاز دارد.

نامگذاری متغیرها

نام متغیر باید یک شناسه معتبر باشد. شناسه می تواند شامل حروف انگلیسی، فارسی یا اعداد باشد. اما حرف اول آن حتما باید یک کاراکتر غیر عددی باشد. کاراکترهایی که حرف یا عدد نیستند مانند : " * ، & ، @ و..." را نمی توان در شناسه استفاده کرد. فقط کاراکتر (_)Underline می تواند در شناسه استفاده شود، اما این کاراکتر هم نمی تواند به تنهایی یک شناسه باشد.

```
'Dim _ as Integer ' error : Identifier expected
Dim Var_Test as Integer
'Dim 123 as Integer 'error : Identifier expected
```

بجز قواعدی که کامپایلر تعیین کرده است و باید اجرا شوند؛ قواعد دیگری هم برای نامگذاری متغیرها وجود دارد که جنبه اختیاری داشته و بیشتر برای خوانا کردن کد برنامه است.

1. نام متغیر باید مشخص کند که این متغیر برای ذخیره کردن چه داده ای ایجاد شده است. برای مثال اگر یک متغیر میخواهد تعداد کاراکترهای یک متن را نگه دارد، نام CharCount برای آن بسیار مناسبتر از Z یا MX است.
2. از نامهای یک کاراکتری و دو کاراکتری برای متغیرها استفاده نکنید.
3. اگر نام یک متغیر از چند کلمه تشکیل شده است، حرف اول هر کلمه را با حرف بزرگ بنویسید. برای مثال CharCount بجای charcount .
4. میتوانید یک شناسه یک یا چند کاراکتری در ابتدای نام متغیر قرار دهید که نوع آن را مشخص کند. مثلا برای متغیر Integer اول اسم متغیر یک i و برای متغیر short اول اسم متغیر یک s قرار دهید تا به محض دیدن نام متغیر متوجه شوید از چه نوعی است. معمولا این شناسه نوع را با حروف کوچک مینویسند. مثال : iVar ، sCharCount . گاهی که امکان تداخل نام های اولیه وجود دارد می توانید تعداد کاراکتر بیشتری از اسم نوع داده را بکار برید ، مانند intVar یا strName برای متغیری از نوع String .

مقداردهی اولیه

در ویژوال بیسیک متغیر پس از ایجاد شدن، مقدار دهی اولیه می شود. ما میتوانیم مقدار اولیه متغیر را خودمان تعیین کنیم یا اجازه بدهیم VB مقدار پیش فرض را درنظر بگیرد؛ مقدار پیش فرض برای داده های عددی برابر صفر (0) است. مقدار دهی اولیه به یک متغیر بر مبنای ساختار دستوری زیر انجام می شود:

```
Dim variableName as TypeName = variableInitialValue
```

مانند :

```
Dim sTest2 as Short = 100
```

در بخش قبلی مقدار stest1 برابر 0 بود. چون مقدار دهی اولیه نشده بود. اما در اینجا مقدار sTest2 برابر با 100 است. یعنی در آن بخش از حافظه که از سیستم عامل گرفته شده است، در حال حاضر عدد 100 ذخیره شده است.

عملگر جایگزینی

اولین عملگری که برای کار با متغیرها به آن نیاز داریم عملگر جایگزینی(=) است. این عملگر مقدار سمت راست خودش را جایگزین مقدار متغیر سمت چپ می کند. عملوند سمت راست عملگر جایگزینی می تواند عدد ثابت، متغیر یا هر کدی که یک مقدار برگرداند باشد، اما عملوند سمت چپ این عملگر باید یک متغیر(شی) یا خاصیت باشد.

```
Dim v1 as Short = 200
Dim v2 as Short = 100
v1 = 400
```

خط اول و دوم دو متغیر با نامهای v1 و v2 تعریف میکنند و به ترتیب با 200 و 100 مقداردهی اولیه میکنند. خط سوم مقدار سمت راست عملگر جایگزینی را در متغیر سمت چپ جایگزین میکند. پس مقدار v1 برابر با 400 میشود. نوع داده سمت راست عملگر جایگزینی باید با نوع داده سمت چپ آن یکی باشد. به عنوان مثال متغیرهایی که از نوع داده short هستند میتوانند عدد صحیحی بین 32768- تا 32767+ را در خود نگهداری کنند. پس مقدار سمت راست باید یک عدد صحیح در همین بازه باشد. اگر مقداری که قرار است در متغیر جایگزین شود خارج از بازه تعریف شده برای آن نوع داده باشد خطای سرریز (Overflow) رخ میدهد.

تبدیل انواع داده بصورت ضمنی

در بخش قبل تاکید کردیم که نوع داده های دو طرف عملگر جایگزینی باید یکسان باشد. پس نباید بتوانیم یک مقدار short را در یک متغیر Integer کپی کنیم. اما قطعه برنامه زیر از نظر کامپایلر هیچ ایرادی ندارد.

```
Dim intVar as Integer
Dim shrtVar as Short = 100
intVar = shrtVar
```

هرگاه ویژوال بیسیک با دو نوع داده متفاوت در دو سمت عملگر جایگزینی مواجه می شود سعی می کند داده سمت راست را به داده سمت چپ تبدیل کند. یک متغیر از نوع short همیشه قابل تبدیل به متغیری از نوع Integer است بدلیل اینکه تمام مقدارهای مجاز در نوع داده short در Integer هم مجازند. اما برعکس آن همیشه صادق نیست و ممکن است همانطور که در بخش قبل دیدیم خطای سرریز(Overflow) رخ بدهد.

```
Dim intVar as Integer = 4000
Dim shrtVar as Short
shrtVar = intVar ' مقدار دهی معتبر است
intVar = 50000
shrtVar = intVar 'خطای سرریز رخ میدهد
```

این روش تبدیل نوع داده ها بدلیل اینکه صراحتا ذکر نمی کند که تبدیلی در حال انجام است تبدیل نوع داده ضمنی خوانده می شود. تبدیل ضمنی باعث ناخوانا شدن کد برنامه می شود و توصیه می کنیم در حد امکان از آن استفاده نکنید. در فصلهای بعد نوع صریح آن را خواهیم آموخت.

تبدیل های ایمن

```
Short -> Integer -> Long
```

تبدیل هایی که امکان ایجاد خطای سرریز دارند

```
Long -> Integer -> Short
```

بطور کلی داده های صحیح کوچکتر به راحتی به بزرگترها تبدیل می شوند و برعکس آن احتمال خطا دارد.

نوع داده های اعشاری

متغیرهایی که از روی یک نوع داده اعشاری تعریف می شوند می توانند یک عدد اعشاری در خود ذخیره کنند. برای ذخیره اعداد اعشاری در vb.NET دو نوع داده وجود دارد که در جدول زیر مشخصات آنها آمده است :

نوع داده اعشاری	محدوده متغیر	اندازه
Single	از 3.4028235E+38- تا 1.401298E-45- برای اعداد منفی و از 1.401298E-45 تا 3.4028235E+38 برای اعداد مثبت	32bit (4byte)
Double	از 1.79769313486231570E+308- تـــــا - 4.94065645841246544E-324 بـرای اعـداد منفـی و از 4.94065645841246544E-324 تـــــــــــا 1.79769313486231570E+308 برای اعداد مثبت	64bit (8byte)

نوع داده single برای ذخیره اعداد اعشاری با دقت معمولی و نوع double برای ذخیره اعداد اعشاری با دقت مضاعف بکار می رود. یک عدد اعشاری با دقت معمولی(single) نهایتا میتواند 51 رقم اعشار داشته باشد در حالی که عدد اعشاری با دقت مضاعف میتواند تا 341 رقم اعشار داشته باشد. به عبارت ساده تر، برای اعداد اعشاری بزرگ از متغیر double استفاده می شود.

تعریف و مقداردهی اولیه و متغیرهای اعشاری هم مانند همه متغیرها و مانند متغیرهای صحیح است. مقدار اولیه پیشفرض برای داده های اعشاری نیز عدد صفر(0) است.

```
Dim Var1 as Double = 3.14159
Dim Var2 as Single = 3.14
Dim Var3 as Single = 10
Var3 = Var2
```

مقدار Var3 چند است؟

تبدیل ضمنی داده های اعشاری

در تبدیل ضمنی داده های صحیح فقط یک مشکل جدی داشتیم و آن خطای سر ریز بود. اما در مورد داده های اعشاری مشکل دیگری هم وجود دارد؛ وقتی داده اعشاری x میخواهد به نوع داده دیگری تبدیل شود که دقت اعشاری آن (تعداد رقم اعشاری که می تواند داشته باشد) کمتر از تعداد رقم اعشار داده x است،

داده x گرد می شود تا تعداد رقم اعشارش با دقت اعشاری نوع جدید یکسان شود. برای مثال اگر یک عدد double را در متغیر single جایگزین کنید تعداد رقم های اعشار آن به 51 رقم کاهش پیدا می کند.

```
Dim intVar1 as Integer = 3.14  'intVar=3
Dim intVar2 as Integer = 7.65 'intVar=8
```

اگر داده اعشاری به داده صحیح تبدیل شود تمام اعشارش را از دست می دهد. مقدار متغیر صحیح برابر با گرد شده عدد اعشاری خواهد شد.

بطور خلاصه :

تبدیل های زیر بدون خطا انجام می شوند.

```
Short -> Integer -> Long -> Single -> Double
```

و برعکس آن احتمالا سرریزی یا ازدست رفتن دقت اعشاری دارد.

چهار عمل اصلی

در vb.NET می توان از چهار عمل اصلی برای محاسبات ریاضی استفاده کرد. برای جمع و تفریق بترتیب از عملگرهای + و – استفاده می شود و برای ضرب و تقسیم از عملگرهای * و / استفاده می شود.

```
Dim x as Integer = 10
Dim y as Integer = 20
Dim z as Integer
z = x + y 'z=30
z = x * y 'z=200
z = y / x 'z=2
z = x – y 'z=-10
```

در دو طرف هریک از عملگرهای مذکور می تواند یک متغیر، یک عدد یا یک عبارت ریاضی قرار بگیرد. این عملگرها می توانند بصورت متوالی در یک دستور بکار گرفته شوند.

```
z = x + y * 10 – 5
```

عبارت بالا کمی نامفهوم است، سوالی که مطرح می شود این است که ابتدا کدام عمل انجام می شود؟ اول x+y انجام می شود و حاصل آن در 10 ضرب می شود یا اول y ده برابر می شود و سپس 5 از آن کم می شود و بعد با x جمع می شود؟ برای رفع چنین ابهامهایی، در زبان برنامه نویسی برای هر عملگری اولویتی درنظر گرفته شده است.

در جدول زیر هرچه از بالا به پایین برویم اولویت عملگر کمتر می شود.

عملگر
* /
+ -
=

بین عملگرهایی که اولویت برابر دارند (در یک سطر قرار دارند) اولویت با عملگری است که در فرمول ریاضی سمت چپ باشد. با در نظر گرفتن این اطلاعات می توانیم جواب سوالی که پرسیده بودیم را بدهیم. در آن مثال چون اولویت ضرب بیشتر از جمع و جایگزینی است ابتدا y*10 می شود و سپس از بین دو عملگر - و + آن عملگری که سمت چپ قرار دارد فراخوانی می شود یعنی حاصل y*10 با x جمع می شود، سپس از نتیجه 5 واحد کم می شود و در نهایت حاصل در z ذخیره می شود.

اولویت عملگر جایگزینی از همه عملگرها کمتر است. به همین دلیل می توانیم مطمئن باشیم که عمل جایگزینی وقتی انجام می شود که همه محاسبات ما انجام شده است.

به مثالهای زیر توجه کنید :

```
Dim z as Integer
Dim s as Single
z = 1+2+3*4 'z=15
z = 2*5+3*8 'z=34
z = 20/4+8-2*4 'z=5
s = 239 / 4 'z=59.75
z = 239 / 4 'z=60
```

در دو خط آخر حاصل یک تقسیم ابتدا در یک متغیر single و سپس در یک متغر Integer جایگزین شده است. در مورد جایگزین کردن حاصل این تقسیم در متغیر single مشکلی وجود ندارد اما به دلیل اینکه متغیر

z از نوع صحیح است حاصل تقسیم به طور ضمنی به یک عدد صحیح تبدیل می شود و دقت اعشاری آن از دست می رود.
توجه : مقدار سمت راست عملگر تقسیم (مقسوم علیه) نباید صفر باشد.

عملگرهای حسابی دیگر

عملگر تقسیم صحیح "\" : عملوند سمت چپ را به عملوند سمت راست تقسیم می کند و خارج قسمت صحیح آن را برمی گرداند. هر دو عملوند این عملگر باید اعداد صحیح باشند. در صورتی که اعشاری باشند بصورت ضمنی تبدیل به عدد صحیح می شوند.

```
z = 239 \ 4 'z=59
```

عملگر باقیمانده "mod" : باقیمانده تقسیم عملوند سمت چپ به عملوند سمت راست را برمی گرداند.

```
z = 239 mod 4 'z=3
```

عملگر توان "^" : عملوند سمت چپ را به توان عملوند سمت راست می رساند

```
z = 10 ^ 2 'z=100
z = 2 ^ 10 'z=1024
```

عملگر منفی"-" : تفاوت این عملگر با عملگر تفریق این است که عملگر تفریق روی دو عملوند تاثیر می گذارد و عملگر منفی تنها یک عملوند دارد.

```
x = 10
z = -x 'z=-10
```

حال باید جدول اولویت عملگرها را بازنویسی کنیم تا ببینیم وضعیت عملگرهایی که تازه یاد گرفتیم در این جدول چگونه است:

عملگر
^
- (منفی)
* /
\
Mod
- +
=

به مثالهای زیر توجه کنید :

```
z = 10 * 20 - 2 ^ 5 * -2 'z=264
s = 4 ^ -2 's=0.0625
z = 64 ^ 0.5 'z=8
s = 3 ^ 5 \ 2  'z=121
```

پرانتز، بیشترین اولویت

اولویتهایی که زبان برنامه نویسی به عملگرها داده برای ما مشخص می کند که هر عبارت ریاضی چگونه ارزیابی می شود؛ اما این اولویتها همه خواسته های ما را بر آورده نمی کنند. برای مثال ما نمی توانیم حاصل یک عمل جمع را به توان برسانیم چون اولویت توان از جمع بیشتر است. برای حل این مشکل می توانیم از پرانتز استفاده کنیم. اولویت پرانتز از همه عملگرها بیشتر است. بنابراین ابتدا عبارت داخل پرانتز ارزشیابی می شود.

```
z = (2+3)^2 'z=25
z = (3*3)^(1+1) 'z=81
z = -(3*(4/2)) 'z=-6
```

در پرانتزهای تو در تو اولویت از درونی ترین پرانتز به بیرونی ترین پرانتز کاهش می یابد. یعنی بیشترین اولویت با درونی ترین پرانتز و کمترین اولویت با بیرونی ترین پرانتز است.

```
z = 64 ^ (1/2) 'z=8
z = 64 ^ (1/3) 'z=4
```

با استفاده از توانهای کوچکتر از یک و بزرگتر از صفر و پرانتز میتوان به راحتی جذر گرفت.

تمرین :

حاصل عبارتهای زیر را حساب کنید. (بعضی از عبارتها خطا دارند.)

```
Dim s as Single=0
```

```
Dim m as Integer = 0, sh as Short = 0
s = 40 * 2 / 2
m = 2 ^ 15
sh = 2 ^ 15
s = ( 3 * 5 ^ 2 ) - ( 4 ^ ( 1 / 2 ) - 1 ) * -4
sh = 3 \ 4 / 3
s = -3^2
s = 81 ^ - ( 1 / 2 )
```

نوع داده های کاراکتری

تا اینجا فقط انواع داده عددی را بررسی کردیم، اما حالا می خواهیم داده هایی را بررسی کنیم که کاراکتر در خود نگه می دارند. در ویژوال بیسیک برای ذخیره داده های کاراکتری دو نوع داده تعبیه شده است. نوع اول که char نام دارد فقط میتواند یک کاراکتر(یک حرف یا علامت) را در خود ذخیره کند و نوع دوم که string نام دارد می تواند یک متن در خود ذخیره کند؛ این متن می تواند نام یک کتاب، آدرس ای میل یا حتی یک متن چندین صفحه ای باشد.

حال می خواهیم یک متغیر کاراکتری تعریف کنیم و مقدار آن را برابر با کاراکتر(حرف) A قرار دهیم:

```
Dim c as char
c = A
```

خط اولی که نوشتیم به کامپایلر می گوید که فضای کافی برای یک متغیر از نوع char به نام c در حافظه اصلی کامپیوتر بگیرد. تا اینجای کار همه چیز به خیر و خوشی می گذرد اما کامپایلر اصلا از خط دوم خوشش نمی آید! کامپایلر تصور می کند که A نام یک متغیر است و از آنجا که ما این متغیر را تعریف نکرده ایم، به ما اخطار میدهد که A تعریف نشده است (Name 'A' is not declared)

برای اینکه به کامپایلر بگوییم که آنچه نوشته ایم یک کاراکتر است و متغیر (یا هر شناسه دیگری) نیست از کوتیشن (") استفاده می کنیم.

```
c = "A"
```

هر چیزی که بین دو علامت " نوشته شود از نظر کامپایلر یک کاراکتر (یا متن) است.

```
Dim d as char = "ن"
```

بین دو کوتیشن هر کاراکتری می توان قرار داد، ویژوال بیسیک در این مورد اصلاً ما را محدود نمی کند و می توانیم از حروف فارسی یا هر زبان دیگری استفاده کنیم. برای اینکه لیستی از کاراکترها ببینید می توانید برنامه Character Map را اجرا کنید.(از منوی start گزینه run را انتخاب کنید و charmap را تایپ کنید و OK را بزنید.) هر کاراکتری که در لیست این برنامه وجود دارد می توانید به عنوان کاراکتر به متغیر char بدهید.

نوع داده رشته (string)

نوع داده رشته برای ذخیره کردن دنباله ای از کاراکترها(برای مثال یک متن یا یک نام) مورد استفاده قرار می گیرد. هر رشته می تواند تا حدود 2 میلیارد کاراکتر در خود داشته باشد، بنابراین حتی می توان چندصد صفحه از یک کتاب را هم در یک رشته ذخیره کرد.

```
Dim str as String
```

خط بالا یک متغیر از نوع string ، با نام str ایجاد میکند. مقدار دهی اولیه و عملگر جایگزینی برای رشته ها مانند کاراکترها است و رشته ها هم مانند کاراکترها باید در کوتیشن قرار بگیرند.

```
Dim test as String = "شاهنامه فردوسی"
test = "گلستان سعدی"
```

خط اول متغیر test را تعریف میکند و مقدار اولیه آن را برابر با "شاهنامه فردوسی" قرار می دهد و خط دوم مقدار "گلستان سعدی" را در متغیر test جایگزین می کند.

الحاق رشته ها

در ویژوال بیسیک به راحتی می توان دو رشته (string) را با هم الحاق کرد و رشته جدیدی ایجاد کرد. برای این کار می توانیم از عملگر + یا عملگر & استفاده کنیم. نتیجه کار هر دو عملگر یکسان است.

```
Dim s2 as String = "Visual Basic.NET"
```

```
Dim s1 as String = "Microsoft "
Dim s3 as string
s3 = s1 + s2 ‘or s3 = s1 & s2
```

مقدار s3 برابر با رشته "Microsoft Visual Basic.NET" است. عمل الحاق رشته جدیدی ایجاد می کند، سپس رشته اول را در آن جایگزین می کند و در نهایت رشته دوم را به انتهای رشته جدید (که برابر رشته اول است) اضافه کرده و مقدار حاصل را بر می گرداند.

اگر هر دو عملوندِ عملگرهای & و + رشته باشند نتیجه هر دو یکسان است. اما در صورتی که یکی از عملوندها مقدار عددی داشته باشد (نوع داده صحیح یا اعشاری باشد) عملگر + سعی میکند عملگر رشته ای را بطور ضمنی به نوع داده Double تبدیل کند و عملگر جمع عددی را روی آن اعمال کند؛ در صورتی که عملگر & سعی می کند مقدار صحیح را بطور ضمنی به نوع داده رشته تبدیل کند و عمل الحاق رشته ها را انجام دهد.

```
Dim s as String
s = 10 + "15" ‘s="25"
s = 10 & "15" ‘s="1015"
s = "15" + 10 ‘s="25"
s = "15" & 10 ‘s="1510"
```

در صورتی که یک عملوند عددی باشد و یک عملوند رشته ای و عملوند رشته ای قابل تبدیل به نوع double نباشد کامپایلر کار را با یک پیام خطا مبنی بر اینکه نمی تواند رشته مورد نظر شما را به double تبدیل کند به پایان می برد.

```
s = "a" + 10 ‘Cast from string "a" to type 'Double' is not valid.
s = "a" & 10 ‘s="a10"
```

همانطور که دیدید هر رشته به دو کاراکتر " محدود می شود، حال چگونه خود این کاراکتر را در رشته درج کنیم؟

```
Dim s as String = "hamid:"salam""
```

عبارت بالا کامپایلر را ناراحت میکند! و کامپایلر ناراحتی خودش را با یک پیغام خطا به شما ابراز میکند. کامپایلر انتظار دارد دستور شما پس از علامت کوتیشن دوم تمام شده باشد. دستور صحیح بصورت زیر است :

```
Dim s as String = "hamid:""salam"""
```

برای درج کاراکتر " در یک رشته باید دو بار این کاراکتر را پشت سر هم بیاوریم. وقتی کوتیشن وارد رشته می شود تشخیص مرزهای رشته تا حدودی دشوار می شود.

```
Dim s1 as String = "salam"
Dim s2 as String
s2 = "hamid:""" + s1 + """" ‘hamid:"salam"
s2 = "hamid:"" + s1 + """" ‘hamid:" + s1 + ""
```

در خط آخر فقط یک کوتیشن از هشت کوتیشن خط قبلش حذف شده اما نتیجه تفاوت بسیاری دارد. برای اینکه مرزهای رشته ها را راحت تر تشخیص دهید میتوانید رنگ زمینه رشته ها را تغییر دهید. برای این کار وارد محیط کاری vs.NET شوید و از منوی tools گزینه options را انتخاب کنید. سمت چپ فرم بخش Environment و سپس Fonts and Colors را انتخاب کنید. در این بخش می توانید فونت و رنگ اجزاء مختلف محیط کاری را تغییر دهید. از لیست Display Items آیتم String را انتخاب کرده و Item Background را مطابق با سلیقه خودتان تغییر دهید؛ پس از اینکه پنجره را با زدن دکمه Ok ببندید رنگ زمینه رشته ها تغییر می کند.

نوع داده بولی (boolean)

این نوع داده، ساده ترین و کوچکترین نوع داده در ویژوال بیسیک است. متغیرهایی که از این نوع تعریف می شوند فقط می توانند یکی از مقدارهای true یا false را بگیرند؛ و مقدار اولیه پیشفرض برای داده های بولی برابر false است.

```
Dim b as Boolean = true
b = false
```

توجه داشته باشید که کلمه های کلیدی true و false در کوتیشن قرار نمی گیرند. اما در صورتی که شما این مقدارها را در کوتیشین قرار دهید هم کامپایلر از شما ایراد نمی گیرد. چون به طور ضمنی آنها را به boolean تبدیل می کند.

```
b =  "true"
```

توجه داشته باشید که فقط دو رشته "true" و "false" قابل تبدیل به نوع بولی هستند. در صورتی که مقدار یک متغیر بولی را در یک رشته جایگزین کنید. مقدار رشته برابر با یکی از مقدارهای "true" یا "false" خواهد شد.

```
b = true
Dim s as string = b 's = "true"
```

داده بولی را می توان به داده های عددی نیز تبدیل تبدیل کرد. در این صورت مقدار true به -1 و مقدار false به 0 تبدیل خواهد شد.

```
b=false
Dim i as integer = b ' i = 0
```

در تبدیل داده عددی به داده بولی هر مقدار غیر صفر به true و صفر به false تبدیل میشود.

```
b = 5054.3 ' b = true
```

ثابتها

متغیرها را با هم بررسی کردیم، و متوجه شدیم که یک متغیر بخشی از حافظه اصلی است که می توانیم تغییر دهیم. حال می خواهیم به بررسی ثابتها بپردازیم. بطور ساده، یک ثابت بخشی از حافظه اصلی است که در اختیار برنامه ما قرار می گیرد و ما نمی توانیم آن را تغییر بدهیم. یا به عبارت دیگر فقط می توانیم به آن مقدار اولیه بدهیم. ساختار دستوری تعریف یک ثابت بصورت زیر است:

```
Const name [As TypeName] = initValue
```

برای مثال :

```
Const MY_TEST_CONST As Integer = 10
```
کد بالا باعث می شود فضای لازم برای یک Integer از سیستم عامل گرفته شود و مقدار 10 در آن ذخیره شود. از این خط به بعد مقدار MY_TEST_CONST برابر 10 است. به ثابتها نمی توان با استفاده از عملگر جایگزینی مقدار داد. ثابتها وقتی استفاده می شوند که یک مقدار در برنامه ما مفهوم خاصی دارد، مثلا شاید در یک بازی عدد 50 نمایانگر امتیاز لازم برای برنده شدن در بازی باشد. میتوانیم به سادگی همه جا از همین عدد 50 استفاده کنیم. اما به روزی فکر کنید که بخواهیم 50 را به 100 تغییر بدهیم، در آن صورت باید هر جا 50 نوشته شده است به 100 تبدیل کنیم، با توجه به اینکه همه 50 هایی که در برنامه نوشته شده است مربوط به امتیاز لازم برای برنده شدن نیستند شاید بعضی از آنها مربوط به تعداد بازیکن های مجاز برای بازی باشند. برای جلوگیری از این آشفتگی می توانیم برای اعداد و رشته هایی که مفهوم خاصی دارند، ثابتهایی تعریف کنیم و هرجا لازم شد، از ثابتها استفاده کنیم. ثابتها علاوه بر اینکه باعث سهولت تغییر دادن مقادیر می شوند، به خوانا شدن برنامه نیز کمک می کنند.

```
Const WIN_POINT As Short = 50
Const MAX_PLAYER As Short = 50
```
اکنون می توانیم هرجا که به امتیاز لازم برای برد احتیاج داریم از WIN_POINT و هرجا به تعداد بازیکنان مجاز احتیاج داشتیم از MAX_PLAYER استفاده کنیم و اگر روزی خواستیم امتیاز لازم برای برد را به 100 تغییر دهیم کافی است مقدار ثابت مربوط به آن را به 100 تغییر بدهیم.
نوع داده یک ثابت را می توانیم ننویسیم:

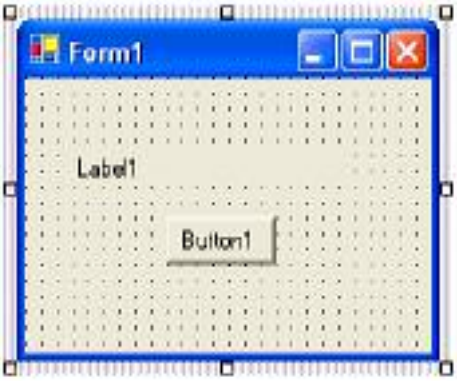
```
Const WIN_POINT = 50
Const MAX_PLAYER = 50
```
در این صورت ویژوال بیسیک با توجه به مقدار اولیه نوع داده را تشخیص می دهد. اما اگر نوع داده را ذکر کرده باشیم مقدار اولیه باید قابل تبدیل به نوع ذکر شده باشد.
نکته : معمولا برای نامگذاری ثابتها از حروف بزرگ استفاده می شود و در صورتی که نام چند کلمه ای باشد، کلمه های مختلف با _ (Underline) از یکدیگر جدا می شوند.

اولین برنامه

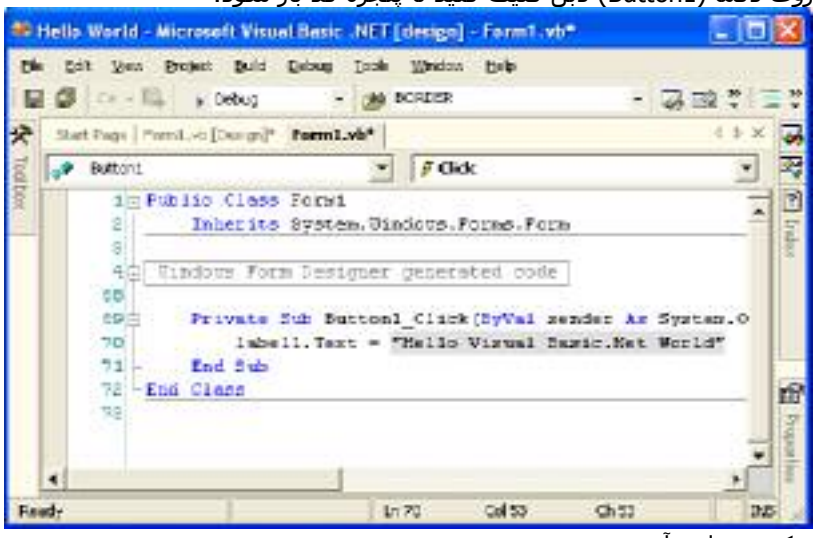
در این فصل می خواهیم اولین برنامه خودمان را با ویژوال بیسیک بنویسم و با رویدادها ، خاصیتها و متدها از نزدیک آشنا شویم.

برنامه کلاسیک Hello World

پروژه جدیدی با نام "Hello World" ایجاد کنید.(اگر طریقه انجام این کار را فراموش کرده اید به فصل 2 مراجعه کنید.) از Toolbox یک Label و یک Command Button بردارید و به فرم اضافه کنید. اینکه هرکدام را کجا قرار دهید دقیقا بستگی به سلیقه شما دارد. فقط آنها را طوری روی فرم قرار دهید که هردو کاملا دیده شوند. حاصل کار شاید شبیه عکس زیر باشد:



روی دکمه(Button1) دبل کلیک کنید تا پنجره کد باز شود.



و کد زیر را در آن بنویسید :

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Label1.Text = "Hello Visual Basic.Net World"
End Sub
```

سپس دکمه F5 را فشار دهید تا برنامه اجرا شود. اگر همه چیز درست پیش برود پنجره ای دقیقا مانند آنچه طراحی کرده بودید نمایش داده خواهد شد. با این تفاوت که نقطه چینهای روی آن محو شده است. اگر برنامه درست اجرا نشده است یک بار دیگر با دقت کدی را که نوشته اید با آنچه در بالا نوشته شده است مقایسه کنید.

پس از اجرای برنامه روی دکمه Button1 کلیک کنید. خواهید دید که متنی که سمت راست عملگر جایگزینی (مساوی) نوشته بودیم همانجایی که Label قرار داشت نشان داده میشود. البته چگونگی نمایش آن تا حدودی بستگی به این دارد که اندازه Label شما چقدر باشد.

اگر Label شما عرضش(Width) بیشتر باشد بطوری که "world" هم در خط اول جا شود این نوشته یک خطی خواهد شد. و اگر هم عرض و هم ارتفاع(Height) آن کم باشد احتمالا بخشهایی از این نوشته دیده نخواهند شد.
پنجره برنامه ای که نوشته اید ببندید تا بار دیگر به محیط ویژوال بیسیک بازگردیم. اگر بدون بستن برنامه در حال اجرا به محیط ویژوال بیسیک برگردید نمیتوانید کد برنامه را تغییر دهید، یا ابزاری به فرم اضافه کنید.

چگونه کار میکند؟

اجازه بدهید کارهایی را که انجام دادیم مرور کنیم؛ ابتدا یک Button و یک Label روی صفحه قرار دادیم. با قرار گرفتن هرکدام از آنها روی فرم ویژوال بیسیک چندین خط کد در برنامه شما اضافه می کند. یک بار دیگر به بخش کد بروید و این بار روی + کنار "Windows Form Designer generated code " کلیک کنید. کدهایی که در این بخش نوشته شده است توسط خود ویژوال بیسیک نوشته می شود و معمولا نیازی به تغییر دادن آنها نیست. ما می خواهیم ببینیم وقتی یک Label روی صفحه قرار می دهیم ویژوال بیسیک چه چیزهایی می نویسد؟ اولین کدی که نوشته می شود

```
Friend WithEvents Label1 As System.Windows.Forms.Label
```

است. اگر تصور کنید که بجای دو کلمه کلیدی اول این خط کلمه کلیدی Dim نوشته شده است متوجه می شوید که بسیار شبیه تعریف کردن متغیر است. در این خط، شی Label1 به عنوان نمونه ای از کلاس Label ساخته می شود. اگر کمی با دقت نگاه کنید می توانید خط های دیگری را که ویژوال بیسیک برای Label1 نوشته است را پیدا کنید. اما ما (حداقل در حال حاضر) نیازی به دانستن مفهوم این کدها نداریم. کنجکاو بودیم که بدانیم Label1 چیست؟ و حالا میدانیم که یک شی از نوع Label است. Button1 هم یک شی از روی کلاس Button است. بار دیگر روی + مذکور (که حالا تبدیل به – شده است) کلیک کنید تا کدهایی که خود ویژوال بیسیک نوشته ناپدید شوند.

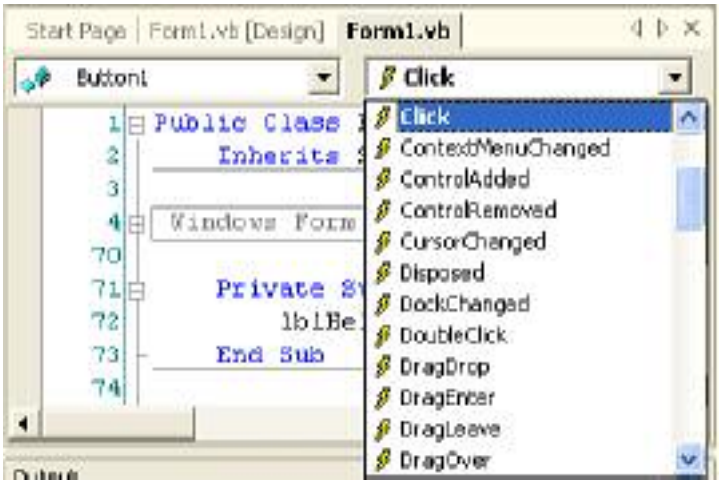
رویداد

رویدادها نقش بسیار مهمی در برنامه هایی که با ویژوال بیسیک نوشته میشوند دارند. وقتی روی دکمه کلیک می کنیم، برای دکمه ، کلیک شدن اتفاق می افتد. سیستم عامل که متوجه کلیک شدن بر روی دکمه می شود پیغامی به برنامه شما می فرستد و جزییات این واقعه را گزارش می کند. و در صورتی که شما بخشی از برنامه را مامور پاسخ دادن به این رویداد کرده باشید، اجرای برنامه به آن بخش از برنامه منتقل خواهد شد.

```
Private Sub Button1_Click(...) Handles Button1.Click

End Sub
```

کد بالا همان دو خطی است که وقتی روی دکمه دبل کلیک کردیم ویژوال بیسیک برای ما نوشته بود(نوشته های درون پرانتز برای کوتاه شدن خط حذف شده). عبارت Handles Button1.click به کامپایلر اعلام می کند که این بخش از برنامه مسوول پاسخ دادن به رویداد click برای شیء button1 است. بدین ترتیب وقتی روی Button1 ، click شود اجرا برنامه به این قسمت منتقل خواهد شد.
هر شی می تواند تعداد زیادی رویداد قابل برنامه نویسی داشته باشد. برای اینکه لیست این رویدادها را ببینید می توانید در بخش بالای "صفحه کد" از combo سمت چپ نام شیی که می خواهید لیست رویدادهایش را ببینید(برای مثال button1) را انتخاب کنید و سپس combo سمت راست را باز کنید تا لیست رویدادها را ببینید. اگر روی یکی از رویدادها در این لیست کلیک کنید، کد تابعی که مامور پاسخگویی به آن رویداد شده است برای شما نوشته می شود(مانند همان کدی که برای click نوشته شده)

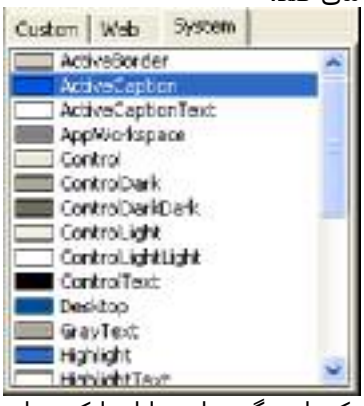


خاصیت

به صفحه طراحی [Design] بازگردید و روی Label1 کلیک کنید. در پنجره Properties می توانید لیستی از خاصیتهای Label1 را ببینید.

حال خاصیت Text را در جدول Properties پیدا کنید و مقدار آن را به دلخواه خودتان تغییر بدهید. خواهید دید که متن نوشته شده در Label1 هم تغییر می کند. خاصیت text از شی Label1 تعیین می کند که شی Label1 چه متنی را نشان بدهد. با تغییر دادن خاصیتها می توانیم ظاهر برنامه را بهتر کنیم، برای مثال خاصیت TextAlign تعیین میکند که نوشته درون Label کجا قرار گیرد. اگر مایلید متن هم از نظر ارتفاع و هم از نظر عرض در وسط label قرار گیرد آنرا به MiddleCenter تغییر دهید.

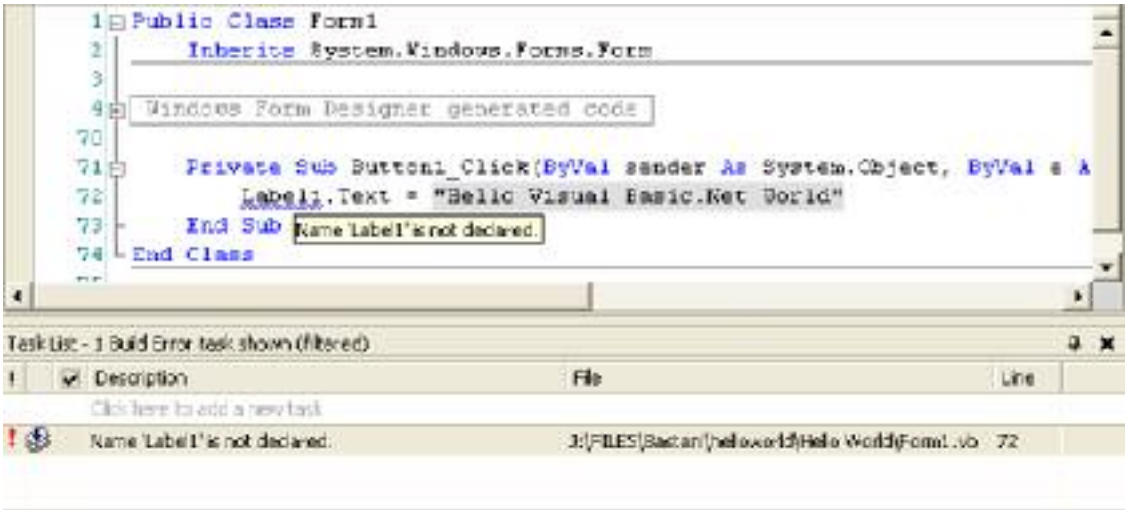
خاصیت Forecolor رنگ نوشته Label را تعیین می کند. پنجره انتخاب رنگ سه زبانه دارد ،اولی Custom است که در آن می توانید هر رنگی را که مایل باشید انتخاب کنید. زبانه دوم web است که رنگهای آن به رنگهایی که در وب استفاده می شود محدود شده و سومی سیستم است. در زبانه سیستم شما میتوانید خاصیت رنگ شی خودتان را به رنگ یکی از اجزاء سیستم عامل نسبت دهید. این رنگها درصورت تغییر کردن رنگهای مورد استفاده سیستم عامل، تغییر خواهند کرد. برای مثال اگر forecolor را در label به ActiveCaption نسبت بدهیم رنگ آن با رنگ میله عنوان (Title bar) پنجره یکی می شود. حال اگر در Control panel\Display\Appearance رنگ میله عنوان در ویندوز را عوض کنید، رنگ نوشته label نیز تغییر می کند.



یک بار دیگر برنامه را اجرا کنید تا نتیجه تغییرات را در زمان اجرا هم ببینید.

پس از اینکه برنامه را بستید به پنجره properties بروید و خاصیت (name) شی Label1 را به lblHello تغییر دهید. خاصیت (name) خاصیتی است که ویژوال بیسیک شی را از روی آن می شناسد. برای کار با متغیرهایی که از روی نوع داده های پایه ساخته شده بودند نیز از "نام" آنها استفاده می کردیم.
اگر در این وضعیت برنامه را اجرا کنید. ویژوال بیسیک به شما خواهد گفت که خطاهایی پیش آمده آیا می خواهید ادامه دهید؟

اگر yes را بزنید برنامه ویژوال بیسیک آخرین نسخه از برنامه شما را که درست کامپایل شده است اجرا خواهد کرد و اگر No را انتخاب کنید اجرا متوقف می شود و میتوانید مشکل را رفع کنید.
بخاطر بیاورید که این مشکل وقتی پیش آمد که ما خاصیت name از شی label1 را تغییر دادیم. در پنجره Task List می توانید جزییات خطایی که رخ داده است ببینید و اگر روی خطا کلیک کنید آن خط از برنامه که باعث ایجاد این خطا بوده است به شما نشان داده خواهد شد.



زیر کلمه خاصی که باعث خطا شده است خط کشیده می شود و با نگه داشتن موس روی آن می توانید از جزییات خطا مطلع شوید.
این بار پیغام خطای ما "Name 'Label1' is not declared" است. در فصل قبل یک بار به پیغامی شبیه این برخورد کرده بودیم. دلیلش این است که کامپایلر به شناسه ای برخورد کرده است که برایش آشنا نیست. همیشه حق با کامپایلر است! ما نام Label1 را تغییر داده ایم و این دقیقا به معنی آن است که در حال حاضر هیچ شیی با نام Label1 وجود ندارد. آن خطی که شی label1 را تعریف کرده بود بخاطر دارید؟ حالا آن خط به خط زیر تغییر کرده است :

```
Friend WithEvents lblHello As System.Windows.Forms.Label
```

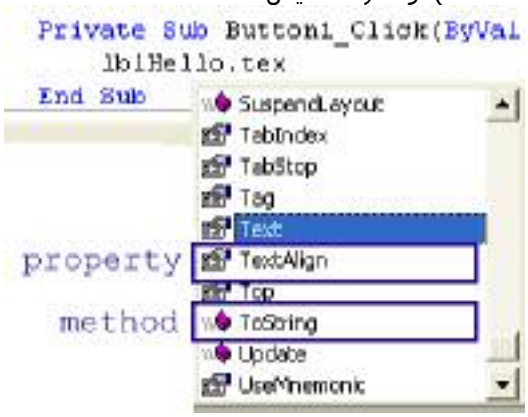
برای اینکه برنامه بار دیگر اجرا شود باید کدی که نوشته بودیم تغییر دهیم و بجای Label1 نام جدید آن یعنی lblHello را بنویسیم.

```
lblHello.Text = "Hello Visual Basic.Net World"
```

بعد از تغییر دادن کد کرسر را از روی خط حرکت دهید، اگر نام را درست نوشته باشید خط زیر نوشته حذف می شود. اکنون می توانید برنامه را بار دیگر اجرا کنید.

عملگر دسترسی به اعضا (.)

برای دسترسی به اعضای(اجزاء) یک شی(متدها و خاصیتها) از عملگر دسترسی به اجزاء (نقطه) استفاده می کنیم. به این صورت که ابتدا نام شی، سپس عملگر نقطه و در آخر نام متد یا خاصیت مورد نظرمان را می نویسم. همانطور که در تصویر زیر می بینید خاصیتها و متدها را میتوان به راحتی (از روی آیکون کنار نامشان) از هم تشخیص داد.



در برنامه Hello World قصد ما این بود که وقتی کاربر روی دکمه کلیک کرد متن شی Label1 تغییر کند. این کار را با کد زیر انجام دادیم.

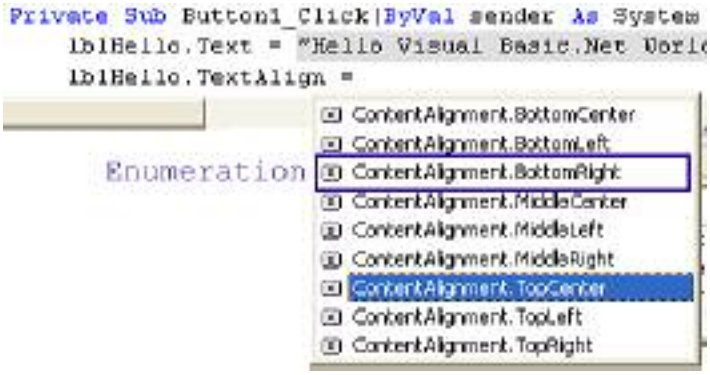
```
lblHello.Text = "Hello Visual Basic.Net World"
```

خاصیت text از شی lblHello یک متغیر رشته ای(string) است. پس می توانیم با عملگر جایگزینی یک مقدار رشته ای را در آن جایگزین کنیم. و این جایگزینی به معنی عوض شدن متن شی lblHello است.

قبلا گفتیم که مقدار هر دو سمت عملگر جایگزینی باید از یک نوع باشد، پس وقتی بخواهیم یک خاصیت را تغییر بدهیم باید بدانیم از چه نوعی است. یک خاصیت میتواند عدد صحیح، عدد اعشاری، رشته یا یک شی باشد؛ بعضی از خاصیتها هم فقط مقدارهای خاصی را قبول میکنند (به این نوع داده ها، نوع داده شمارشی گفته میشود) برای مثال خاصیت TextAlign مقدارهایی از قبیل

ContentAlignment.BottomCenter
ContentAlignment.BottomLeft
ContentAlignment.MiddleLeft

می گیرد. اگر بخاطر سپردن این نامهای طولانی برای شما هم مانند من سخت است میتوانید بعد از نوشتن نام خاصیت کاراکتر مساوی را تایپ کنید تا ویژوال بیسیک لیستی از مقدارهای مجاز برای شما باز کند.



متدها

گفتیم که متدها، توانایی های یک شی هستند. برای مثال اشیایی که از نوع Label هستند توانایی غیب شدن(Hide) و ظاهر شدن(Show) دارند. برای اینکه متدها را تمرین کنیم، دو button دیگر به صفحه اضافه کنید، خاصیت text و Name اولی را به Show و btnShow و دومی را به Hide و btnHide تغییر دهید.

می خواهیم وقتی کاربر روی دکمه btnHide کلیک کرد، lblHello ناپدید شود و وقتی روی btnShow کلیک کرد دوباره ظاهر شود. پس کدی را که باعث ناپدید شدن دکمه می شود باید در تابعی که به کلیک شدن روی دکمه btnHide پاسخ می دهد بنویسیم. پس روی این دکمه دبل کلیک می کنیم و کد مربوط به ناپدید کردن lblHello را در آن می نویسیم.

```
Private Sub btnHide_Click(...) Handles btnHide.Click
    lblHello.Hide()
End Sub
```

کدی که نوشتیم، متد Hide() از شی lblHello را فراخوانی می کند. به عبارت دیگر به lblHello می گوید "غیب شو" و lblHello، خودش را غیب می کند.

با فراخوانی متد show() شی lblHello خودش را ظاهر می کند. این متد را باید در تابع پاسخگوی کدام رویداد بنویسیم؟

```
Private Sub btnShow_Click(...) Handles btnShow.Click
    lblHello.Show()
End Sub
```

درباره متدها در فصلهای بعد بطور مفصل بحث خواهیم کرد.

تمرین

عنوان دکمه اولی که روی فرم قرار دادیم را به عنوانی مناسب تغییر دهید.

عنوان(text) پنجره برنامه را به "Hello World" تغییر دهید.
رنگ زمینه (Backcolor) شی lblHello را به دلخواه خودتان تغییر دهید.
قلم (font) شی lblHello را تغییر دهید.
کد برنامه را طوری تغییر دهید که وقتی روی دکمه button1 کلیک می کنیم نام خودتان در lblHello نمایش داده شود.

کارگاه
برنامه را طوری تغییر دهید که وقتی روی lblHello کلیک می کنیم همه دکمه ها ناپدید شوند.
برنامه را طوری تغییر دهید که دکمه button1 نیز همراه lblHello ناپدید و ظاهر شود.
برنامه را طوری تغییر دهید که وقتی موس روی دکمه button1 می آید عنوان آن به "hello" و وقتی از روی آن کنار میرود عنوانش به "bye" تغییر کند.

راهنمایی کارگاه
اشیایی که از نوع Label هستند، مانند button ها دارای رویداد click هستند. و buttonها هم مانند labelها دارای متدهای hide و show هستند.
button1 همان زمانی باید ناپدید شود که lblHello ناپدید می شود، lblHello چه وقت ناپدید می شود؟ وقتی که روی btnHide کلیک می شود.
در لیست رویدادهای شی button1 دنبال رویدادهای MouseEnter و MouseLeave بگردید.

پاسخ کارگاه
.1
```
Private Sub lblHello_Click(...) Handles lblHello.Click
    Button1.Hide()
    btnHide.Hide()
    btnShow.Hide()
End Sub
```
.2
```
Private Sub btnHide_Click(...) Handles btnHide.Click
    lblHello.Hide()
    Button1.Hide()
End Sub

Private Sub btnShow_Click(...) Handles btnShow.Click
    lblHello.Show()
    Button1.Show()
End Sub
```
.3
```
Private Sub Button1_MouseEnter(...) Handles Button1.MouseEnter
    Button1.Text = "hello"
End Sub

Private Sub Button1_MouseLeave(...) Handles Button1.MouseLeave
    Button1.Text = "bye"
End Sub
```

دستورات شرطی و حلقه ها

در فصل مفاهیم پایه، با مفاهیم "انتخاب" و "تکرار" آشنا شدیم. دستورات شرطی معادل "انتخاب" در الگوریتم هستند و حلقه ها ساختارهایی برای ایجاد تکرار. این فصل را با دستور شرطی(if) آغاز خواهیم کرد و با استفاده از دستورات شرطی یک بازی کوچک خواهیم نوشت. پس از آن به حلقه های تکرار می رسیم و با استفاده از چند مثال ساده ، کار با آنها را یاد خواهیم گرفت.

دستور شرطی (if)

ما در مکالمه های روزمره بارها از جمله های شرطی استفاده می کنیم "اگر هوا خوب باشد، فوتبال بازی می کنیم" یا "اگر به اندازه کافی تمرین کنی، برنامه نویس خوبی خواهی شد." دستور شرطی if دقیقاً همین جمله ها را پیاده سازی می کند. ساختار این دستور (در ساده ترین حالت) به شکل زیر است :

```
If condition Then statement
```

اگر *شرایط برقرار بود* آنگاه *دستورات را اجرا کن*.

دستور if با کلمه کلیدی if شروع می شود و پس از آن یک عبارت شرطی می آید. حاصل عبارت شرطی باید مقدار true یا false داشته باشد (داده بولی) یا مقداری باشد که بطور ضمنی قابل تبدیل به داده بولی است. پس از عبارت شرطی کلمه کلیدی then می آید و در نهایت یک دستور(هر دستوری) قرار می گیرد.

عملگرهای مقایسه ای

پر کاربردترین عبارتهای شرطی، عبارتهایی هستند که در آنها عملگرهای مقایسه ای استفاده شده است. پس ابتدا عملگرهای مقایسه ای را بررسی می کنیم. عملگرهای مقایسه ای دو مقدار را با هم مقایسه می کنند و نتیجه مقایسه را که یک مقدار بولی است بر می گردانند.

عملگر	معادل ریاضی	نوع مقایسه	مثال
=	=	برابری	a = b
<	<	کوچکتر بود عملوند سمت چپ	a < b
>	>	بزرگتر بودن عملوند سمت چپ	a > b
=<	≤	کوچکتر یا مساوی بودن عملوند سمت چپ	a =< b
>=	≥	بزرگتر یا مساوی بودن عملوند سمت چپ	a >= b
<>	≠	برابر نبودن	a <> b

مثال :

```
2 = 2 ' ture
3 = 4 'false
3 <> 4 ' true
2 < 3 ' true
3 < 2 ' false
```

عملگرهای مقایسه ای می توانند کاراکترها و رشته ها را نیز با هم مقایسه کنند.

```
"Visual Basic" = "Visual Basic" 'true
"Microsoft" = "Visual Basic" ' false
"Visual Basic" = "visual basic" 'false
```

همانطور که در مثال آخری می بینید عملگر مقایسه برابری به حروف بزرگ و کوچک حساس است. این حساسیت از آنجا ناشی می شود که برای کامپیوتر هر کاراکتر معرف یک عدد است. برای مثال عدد متناظر a، 61 و عدد متناظر A، 41 است. پس میتوان نتیجه گرفت A از a کوچکتر است.

```
"A" < "a"  ' true
 "b" = "B"  ' false
 "a" < "b"  'ture
"Z" < "a"  'true
```

برای اینکه لیست کاراکترها و عدد متناظرشان در کدپیج unicode را ببینید میتوانید از برنامه Character Map[5] را استفاده کنید. اگر به عدد متناظر "پ" و "د" دقت کنید متوجه میشود که عدد متناظر با "پ" از عدد متناظر "د" بزرگتر است. بنابراین

[5] از منوی start گزینه run را انتخاب کنید و در آنجا charmap.exe را تایپ کنید و ok را بزنید.

“پ” < “د” `false!
متاسفانه از دید کدپیج یونی کد (و به طبع آن ویژوال بیسیک) "گ" ، "چ" ، "پ" و "ژ" از اکثر کاراکترهای فارسی بزرگترند.

بازی حدس زدن عدد

گفتیم که دستور شرطی شامل عبارت شرطی است که باید مقدار بولی داشته باشد، و عملگرهای شرطی را نیز بررسی کردیم. حال می خواهیم برای اینکه بیشتر با دستور شرطی if آشنا شویم یک بازی بنویسیم. البته بازی ما چندان شباهتی به بازیهایی مانند FIFA2003 یا GTA نخواهد داشت! ولی برای آشنایی با دستور if بازی مناسبی است.
قبل از اینکه برنامه نویسی یک برنامه آغاز شود؛ باید جواب یک سوال مشخص شود : "برنامه ما دقیقا چه کاری باید انجام دهد؟" در مورد برنامه کوچکی مثل "بازی حدس زدن عدد"، جواب دادن به این سوال نسبتاً راحت است. ولی در مورد برنامه های بزرگ، ممکن است چندین ماه وقت صرف جواب دادن به این سوال شود.
پس "هرگز تا وقتی مطمئن نشده اید صورت مساله چیست، سعی نکنید آن را حل کنید."
صورت مساله همیشه کامل نیست، برای مثال به توضیح زیر درباره بازی حدس زدن عدد دقت کنید :
بازی حدس زدن عدد، برنامه ای است که یک عدد انتخاب می کند و از کاربر می خواهد که آن عدد را حدس بزند.
اول اینکه در اینجا مشخص شده است که این بازی عددی انتخاب می کند، اما مشخص نشده که این عدد، یک عدد تصادفی است یا یک عدد از بین لیستی از اعداد از پیش تعیین شده. اگر عدد تصادفی است، مشخص نشده که این عدد بین کدام دو عدد است. بین صفر تا صد هزار یا بین منهای هزار تا یک میلیارد؟
دوم اینکه مشخص نشده چطور کاربر عدد را حدس خواهد زد. آیا برنامه باید به کاربر برای رسیدن به جواب کمک کند؟ یا کاربر باید از حس ششمش برای پیدا کردن عدد استفاده کند؟
هرچند که ما اصرار داریم این جزییات مشخص شود، اما بازهم علاقه نداریم همه جزییات مشخص باشد. هرچه جزییات بیشتری مشخص شود، جا برای خلاقیت شما کم می شود. پس سعی می کنیم اینقدر از مساله بدانیم که بتوانیم آنچه مساله می خواهد را ایجاد کنیم؛ ولی همیشه جا برای خلاقیت خودمان نگه می داریم.
صورت مساله بازی حدس زدن عدد:
بازی حدس زدن عدد، برنامه ای است که عدد تصادفی در محدوده ای که کاربر تعیین کرده است(حداقل 100 رقم) انتخاب می کند. و هر بار که کاربر عددی را حدس می زند به کاربر اطلاع می دهد که عددی که انتخاب کرده بزرگتر از عدد انتخاب شده توسط برنامه است یا کوچکتر.
حال می دانیم چه چیز می خواهیم و فضای خالی برای خلاقیت نیز داریم. برای مثال مشخص نشده که برنامه چطور به کاربر اطلاع می دهد. می توانیم از هر روشی که مایل بودیم استفاده کنیم، از نوشتن متن، نشان دادن تصویر یا پخش کردن صدا و...
حال که یک مساله داریم باید ببینیم چطور میتوانیم آن را حل کنیم؟ برای حل کردن یک مساله روشهای زیادی وجود دارد، یکی از آن روشها این است که آن مساله را به اجزاء کوچکتری تقسیم کنیم و سعی کنیم اجزاء کوچکتر را حل کنیم تا مساله اصلی حل شود. مساله ما شامل چند بخش است:

1. دریافت یک محدوده از کاربر (که حداقل شامل 100 رقم باشد)
2. ایجاد عدد تصادفی در محدوده تعیین شده توسط کاربر
3. دریافت یک عدد از کاربر
4. مقایسه عددی که کاربر انتخاب کرده با عدد تصادفی انتخاب شده و اعلام نتیجه مقایسه به کاربر.

پروژه جدیدی ایجاد کنید تا با هم برای هرکدام از این بخشها راه حلی پیدا کنیم. ابتدا ببینیم چطور میتوان یک عدد از کاربر گرفت، چون هم برای بخش اول و هم بخش سوم به آن نیاز داریم. برای گرفتن یک عدد (یا هر داده دیگری) از کاربر، راه های بسیار زیادی وجود دارد که در این کتاب با تعدادی از آنها آشنا می شود. راحت ترین راه، استفاده از شی textbox است. از toolbox یک Textbox، یک Label و یک Botton پیدا کنید و آن را روی فرم قرار دهید. و کد زیر را در رویداد click از شی Button1 بنویسید:

```
Private Sub Button1_Click(…) Handles Button1.Click
    Label1.Text = TextBox1.Text
End Sub
```

طبق آنچه در فصلهای قبل آموختیم این کد وقتی اجرا می شود که رویداد کلیک شدن برای Button1 اتفاق بیفتد؛ و وقتی این اتفاق افتاد خاصیت Text از شی TextBox1 در خاصیت Text از شی Label1 جایگزین می شود. خاصیت Text شی TextBox همان متنی است که در آن نوشته شده و کاربر به راحتی میتواند

آن را تغییر دهد. برنامه را اجرا کنید، و روی Button1 کلیک کنید. متن Textbox1 را تغییر دهید و بار دیگر روی دکمه کلیک کنید.
برنامه را ببندید و دکمه را از روی فرم حذف کنید. و کدی که قبلا در رویداد click از شی Button1 نوشته بودیم را در رویداد TextChanged از شی textbox1 بنویسید.

```
    Private Sub TextBox1_TextChanged(ByVal sender As System.Object,
ByVal e As System.EventArgs) Handles TextBox1.TextChanged
        Label1.Text = TextBox1.Text
    End Sub
```

بار دیگر برنامه را اجرا کنید و متن textbox1 را تغییر دهید. رویداد TextChanged همانطور که از اسمش پیداست وقتی اجرا می شود که متن تغییر کند. با کدی که ما نوشتیم هروقت متن Textbox1 تغییر کند، متن Label1 با متن آن برابر می شود.

توجه : هرگز در بدنه یک رویداد، کاری نکنید که باعث فراخوانی مجدد آن رویداد گردد. برای مثال هیچ وقت نباید متن یک textbox را در بدنه رویداد TextChange همان رویداد تغییر دهید. شاید بد نباشد کد زیر را یک بار اجرا کنید

```
    Private Sub TextBox1_TextChanged(…) Handles TextBox1.TextChanged
        TextBox1.Text = TextBox1.Text + "1"
    End Sub
```

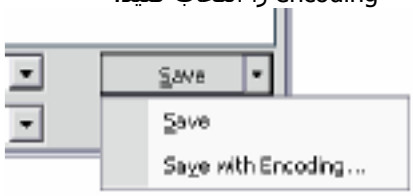
برنامه حتی قبل از اینکه بتواند فرمش را نشان بدهد وارد یک حلقه بینهایت میشود. یک کاراکتر "1" به متن textbox1 اضافه میکند. این کار باعث می شود متن textbox1 تغییر کند و دوباره رویداد صدا می شود و ... این حلقه تا روز قیامت تکرار خواهد شد! برای متوقف کردن برنامه از ترکیب Shift+F5 (stop debugging در منوی debug) استفاده کنید. اگر موفق نشدید ctrl+break را آزمایش کنید.

اکنون می دانیم چطور از کاربر یک عدد بگیریم. برای بخش سوم از مساله نیز می توانیم از همین روش استفاده کنیم، اما خواهید دید که رابط کاربر بهتری برای آن بخش طراحی می کنیم. برای بخش اول ما دو عدد لازم داریم که یکی نمایانگر شروع محدوده عدد و دیگری نمایانگر پایان محدوده است. اشیایی را که برای آزمایش روی صفحه گذاشته بودیم پاک کنید. سپس دو textbox برای گرفتن اطلاعات به فرم اضافه کنید؛ و نام (Name) آنها را به txtStart و txtEnd تغییر دهید. وقتی کاربر این برنامه را اجرا کند با دو textbox مواجه می شود که نمیداند هر کدام از آنها برای چه منظوری آنجا هستند. برای حل این مشکل میتوانیم کنار هرکدام از آنها یک Label قرار دهیم تا درباره وظیفه textbox توضیحی بدهد.

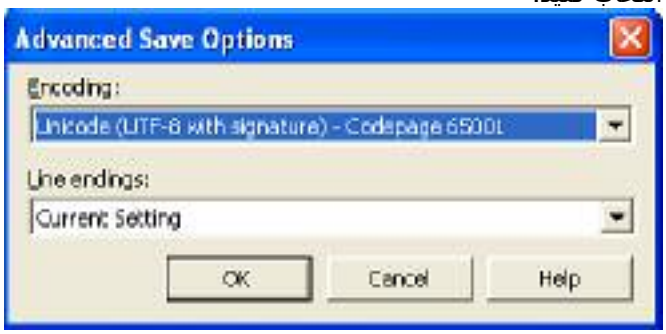
احتمالا برنامه شما تا حدودی شبیه این فرم شده است. ممکن است شما در Labelها متن دیگری نوشته باشید، یا حتی انگلیسی نوشته باشید. اگر شما هم مانند ما فرم خود را فارسی طراحی کرده اید باید دو مطلب جدید هم یاد بگیرید.

1. اکثر اشیاء خاصیتی به نام RightToLeft دارند که اگر برابر False باشد جهت متن در آن شی از چپ به راست و در صورتی که True باشد جهت نوشته از راست به چپ خواهد شد. اگر این خاصیت را برای فرم به true تنظیم کنید، جهت نوشتن در همه اشیایی که به فرم اضافه میشوند بصورت پیش فرض راست به چپ می شود.

2. برای اینکه نوشته های فارسی درست در فایل ذخیره شوند؛ لازم است فایل خود را بصورت Unicode UTF8 ذخیره کنید. برای این کار گزینه save as از منوی فایل را انتخاب کنید و Save with encoding را انتخاب کنید.



از بخش Encoding در فرم Advanced Save Options آیتم Unicode (UTF-8 with signature) را انتخاب کنید.



کار بخش اول مساله تقریبا تمام شده است. حالا می توانیم به بخش دوم بپردازیم. در این بخش باید برنامه یک عدد تصادفی در محدوده تعیین شده توسط کاربر انتخاب کند. انتخاب یک عدد تصادفی در ویژوال بیسیک توسط تابع Rnd() انجام میشود. تابع Rnd() یکی از توابعی است که در در ویژوال بیسیک تعبیه شده است. یک Button (نام آن را btnNew و متن آن را "شروع" بگذارید.) و یک Label (احتمالا Label3) برای آزمایش روی فرم قرار دهید و کد زیر را در رویداد Click از شی Button بنویسید:

```
Private Sub btnNew_Click(…) Handles btnNew.Click
    Randomize()
    label3.Text = Rnd()
End Sub
```

حال اگر روی دکمه کلیک کنید، متن label برابر عددی تصادفی بین صفر و یک میشود. تابع Randomize() که قبل از Rnd استفاده شده است باعث می شود که در هر بار اجرای برنامه دنباله جدیدی از اعداد تصادفی ایجاد شود. در صورتی که از این دستور استفاده نکنید تا وقتی که کامپیوتر restart نشود با هر بار اجرای برنامه، دنباله ای تکراری از اعداد تصادفی انتخاب می شوند.

برای تبدیل کردن عدد تصادفی بین صفر تا یک به عدد تصادفی بین x و y از فرمول زیر استفاده می کنیم :

Int((y - x + 1) * Rnd() + x)

در این فرمول y بزرگترین عدد و x کوچکترین عددی است که احتمال دارد تولید شود. بزرگترین عدد مساله ما txtEnd.Text و کوچکترین عدد آن txtStart.Text است. پس آنها را جایگزین y و x می کنیم.

```
Private Sub btnNew_Click(…) Handles btnNew.Click
    Randomize()
    Label3.Text = Int((txtEnd.Text - txtStart.Text + 1) * Rnd() +
txtStart.Text)
End Sub
```

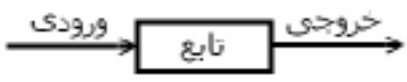
تابحال فقط از تابع هایی استفاده کرده بودیم که بشکل

```
FunctionName ()
```

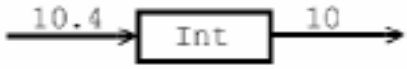
بودند. اما در اینجا از تابعی(تابع int) استفاده کرده ایم که با ساختار قبلی متفاوت است.

```
FucntionName (variableName)
```

در کل توابع میتوانند چندین ورودی و یک خروجی داشته باشند.



تابع، زیر برنامه ای است که تعداد مشخصی(بین صفر تا n) ورودی میگیرد و پس از انجام عملیاتی روی آنها، خروجی را باز میگرداند. برای مثال ورودی تابع Int یک عدد اعشاری است. و خروجی آن بخش غیر اعشاری آن عدد. پس اگر ورودی 10.4 به آن بدهیم خروجی 10 به ما میدهد.




```
output = FunctionName (Input)
```

توابع جزء بسیار مهمی از برنامه نویسی هستند به همین دلیل فصل 11 از کتاب بطور کامل به توابع اختصاص یافته است. اما برای نوشتن بازی حدس زدن عدد، همین معلومات درباره توابع برای ما کافی است.

برنامه را اجرا کنید و در هرکدام از Textbox ها یک عدد بنویسید و دکمه "شروع" را فشار دهید. موفقیت آمیز و راضی کننده بنظر می رسد. اما هنوز مشکلاتی دارد. یکی از آنها این است که کاربر می تواند محدوده ای با کمتر از 100 عدد تعریف کنید. انتظار داریم پس از پایان فصل بتوانید این مشکل را خودتان حل کنید. این مشکل در تمرین 8 این فصل به شما واگذار شده است. مشکل دیگر برنامه این است که اگر کاربر قبل از اینکه عددی وارد کند دکمه "شروع" را بزند برنامه با خطا متوقف می شود. بدلیل اینکه نمی تواند متن TextBox های ما را به عدد تبدیل کند. بهتر است از قبل مقدار پیش فرض مناسبی در Textbox ها قرار دهیم. برای شروع 0 و برای پایان 1000 مناسب به نظر می رسد. همیشه بخاطر داشته باشید که کاربر شما مشغله های بسیار زیادی دارد، سعی کنید برنامه طوری تنظیم شده باشد که کاربر زودتر کارش انجام شود.

اکنون که می توانیم یک عدد تصادفی در محدوده تعیین شده توسط کاربر انتخاب کنیم بهتر است به بخش سوم بپردازیم. می خواهیم برای گرفتن عدد در بخش سوم مساله، رابط کاربری طراحی کنیم که کاربر بتواند با استفاده از موس عدد را وارد کند. برای رسیدن به این منظور رابط کاربری شبیه ماشین حساب کارآمد بنظر می رسد. پس یک Textbox و ده Button به فرم اضافه کنید؛ نام Textbox را txtNumber و نام دکمه ها را btnCalc0 تا btnCalc9 بگذارید. دو دکمه دیگر هم به بخش ماشین حسابی اضافه کنید، یکی برای اینکه کاربر وقتی عددش را انتخاب کرد آن را بزند تا عمل مقایسه و راهنمایی (بخش چهارم) انجام شود و دیگری برای اینکه جعبه متن عدد را پاک کند. نام اولی را معادل btnOK و دومی را معادل btnClear قرار دهید. فرم شما در این مرحله باید شامل اشیایی باشد که در شکل زیر وجود دارد. اما ترتیب قرارگیری آنها روی صفحه تا حدود زیادی بستگی به نظر خودتان دارد.

برای اینکه کلیدهای ماشین حسابی بکار بیافتند باید در رویداد کلیک هر کدام از آنها بنویسیم عدد مربوط به خودشان را به txtNumber اضافه کنند برای مثال در کلیک برای btnCalc1 باید بنویسیم :

```
Private Sub btnCalc1_Click(…) Handles btnCalc1.Click
    txtNumber.Text = txtNumber.Text + "1"
End Sub
```

برای اینکه هر ده دکمه کار کنند میتوانیم این کد را در هر ده دکمه بنویسیم و سپس رشته "1" را به رشته مربوط به آن دکمه تغییر دهیم. معنی دقیق آن 9 بار copy و paste کردن و تغییر دادن یک کاراکتر است. در برنامه نویسی اگر دیدید کاری را میتوان با copy و paste کردن انجام داد، اطمینان داشته باشید که راه بهتری هم وجود دارد. بیایید راه بهتر را پیدا کنیم.

رشته ای که اضافه میکنیم در همه دکمه ها برابر با خاصیت text همان دکمه است. پس میتوانیم بجای آن

رشته از این خاصیت استفاده کنیم :

```
    Private Sub btnCalc1_Click(…) Handles btnCalc1.Click
        txtNumber.Text = txtNumber.Text + btnCalc1.Text
    End Sub
```

از طرف دیگر تابحال چیزی درباره کدی که خود ویژوال بیسیک برای یک event مینویسد نگفته ایم.

```
Private Sub btnCalc1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnCalc1.Click
```

بخش درون پرانتز که تابحال معمولا بخاطر اینکه جای زیادی می گیرد حذف می شد دو شی به ما می دهد. که مقدار شی اولی (sender) برابر همان شیی است که این رویداد برای آن اتفاق افتاده است. یعنی اگر در رویداد کلیک از شی btnCalc1 باشیم، sender برابر با btnCalc1 است و اگر در رویداد TextChanged از شی txtNumber باشیم شی sender برابر با txtNumber خواهد بود. در این صورت میتوانیم در رویداد click از شی btnCalc1 بجای btnCalc1.Text از sender.Text استفاده کنیم :

```
    Private Sub btnCalc1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles btnCalc1.Click
        txtNumber.Text = txtNumber.Text + sender.Text
    End Sub
```

توجه : وقتی بعد از sender عملگر دسترسی به اجزاء (.) را قرار بدهید هیچ لیستی باز نخواهد شد و باید Text را خودتان تایپ کنید.

مقدار دومی که در پرانتز نوشته شده است (e) اطلاعات بیشتری درباره رویداد به ما می دهد. البته درباره رویداد کلیک هیچ اطلاعات اضافه مفیدی وجود ندارد. در فصلهای بعد موارد استفاده آن را خواهید دید.

کد ما در این مرحله بهتر از قبل شده است. حداقل اگر در این مرحله بخواهیم آن را در رویدادهای دیگر اشیاء copy و paste کنیم دیگر نیازی به تغییر ندارد. اما کار را راحت تر از این هم می توان انجام داد. می توانیم به ویژوال بیسیک بگوییم که همه این دکمه ها وقتی رویداد کلیک برایشان اتفاق می افتد باید همین قطعه کد را اجرا کنند. عبارت `Handles btnCalc1.Click` به ویژوال بیسیک میگوید که هر وقت روی btnCalc کلیک شد این قطعه کد را اجرا کند. می توانیم رویدادهای مشابه از اشیاء دیگر را نیز با کاما "," به آن اضافه کنیم :

```
    Private Sub btnCalc1_Click(…) Handles btnCalc0.Click,
btnCalc1.Click, btnCalc2.Click
        txtNumber.Text = txtNumber.Text + sender.Text
    End Sub
```

تغییری که دادیم باعث میشود که هرگاه رویداد کلیک برای یکی از اشیاء btnCalc0، btnCalc1 یا btnCalc2 اتفاق بیفتد این قطعه کد اجرا شود. برنامه را اجرا کنید و روی دکمه های 0 ، 1 و 2 کلیک کنید. بعد از اینکه برنامه خودتان را بستید کد را بازهم تغییر دهید تا سایر دکمه ها هم بکار بیفتند. اگر طولانی شدن خطی که در حال نوشتن آن هستید آزار دهنده است میتوانید با استفاده از زیرخط "_" ادامه آن را در خط بعد بنویسید. این کار فقط جنبه نمایشی دارد و در اجرای دستور تغییری ایجاد نمیکند.

```
    Private Sub btnCalc1_Click(…) _
        Handles btnCalc0.Click, btnCalc1.Click, btnCalc2.Click, _
                btnCalc3.Click, btnCalc4.Click, btnCalc5.Click, _
                btnCalc6.Click, btnCalc7.Click, btnCalc8.Click, _
                btnCalc9.Click
        txtNumber.Text = txtNumber.Text + sender.Text
    End Sub
```

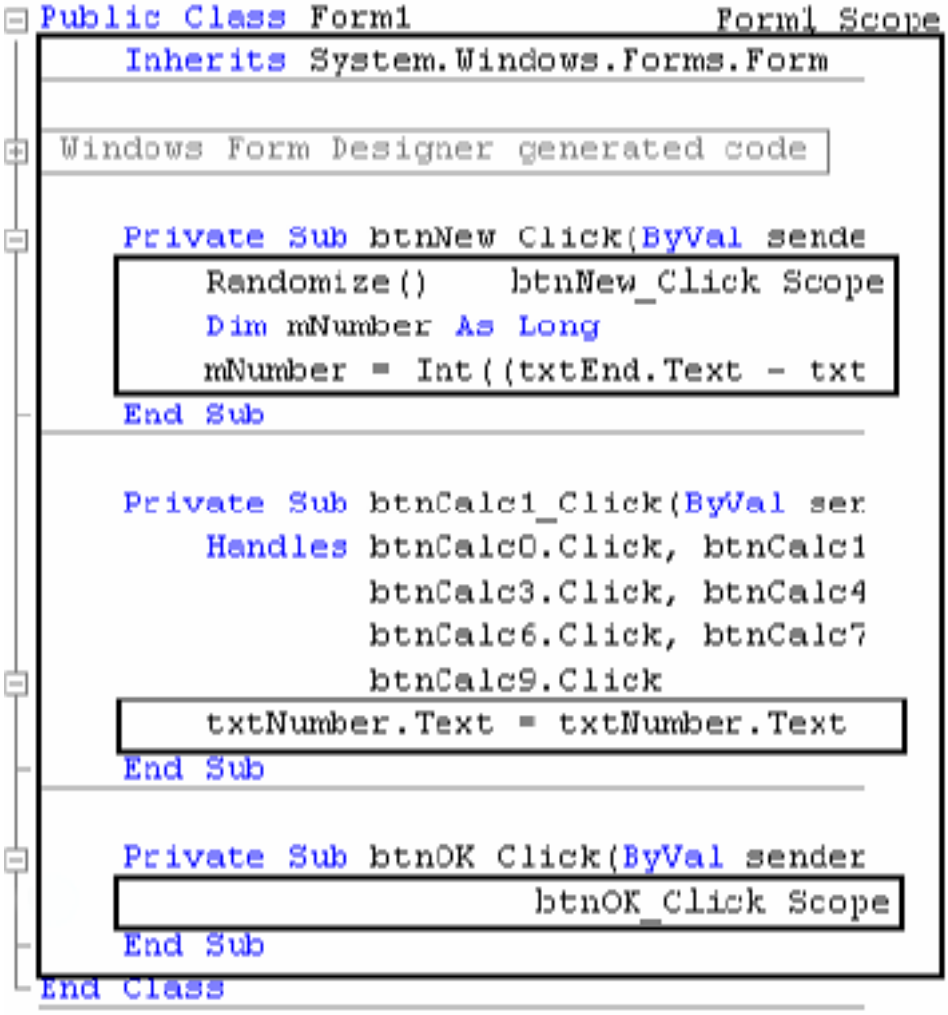
بخش سوم هم تقریبا تمام شده است. اگر میخواهید کاربر نتواند با استفاده از کیبرد در txtNumber عدد وارد کند میتوانید خاصیت ReadOnly این شی را true کنید. در این صورت کاربر مجبور خواهد شد برای وارد کردن عدد از دکمه های ماشین حسابی ما استفاده کند.

بخش چهارم اصلی ترین بخش این برنامه است در این بخش باید عددی را که قبلا انتخاب کرده ایم با عددی که کاربر وارد کرده (txtNumber.Text) مقایسه کنیم. در بخش دوم وقتی عدد را انتخاب کردیم، آن را در یک Label نوشتیم، بدیهی است که ما تمایل نداریم عددی که کاربر باید حدس بزند جلوی چشمش باشد. اگر قرار است آن عدد نمایش داده نشود، نیازی هم به Label نداریم. عدد را میتوان به سادگی در یک متغیر Integer یا Long ذخیره کرد.

```
    Private Sub btnNew_Click(…) Handles btnNew.Click
        Randomize()
        Dim mNumber As Long
```

```
        mNumber = Int((txtEnd.Text - txtStart.Text + 1) * Rnd() +
txtStart.Text)
    End Sub
```

پس از اینکه در رویداد click از شی btnNew عدد را در یک متغیر ذخیره کردیم، باید در رویداد کلیک از شی btnOK آن متغیر را با عددی که کاربر وارد کرده مقایسه می کنیم. اما این کار امکان پذیر نیست. ویژوال بیسیک به ما میگوید که متغیر mNumber تعریف نشده است! ویژوال بیسیک متغیر ما را نمی بیند!
محدوده دید یک متغیر(شی) : هر متغیر(شی) فقط در همان میدانی (scope) دیده می شود که در آن تعریف شده است.
بدنه هر رویداد یک میدان(scope) مجزا است. به همین دلیل اگر شما یک متغیر را در بدنه یک رویداد تعریف کنید رویدادهای دیگر نمی توانند آن را ببینند. یک راه برای اینکه یک متغیر در چند میدان مجزا دیده شود این است که آن را در میدانی بزرگتری که شامل همه آن میدانها می شود تعریف کنیم.



همانطور که در شکل می بینید میدانهای رویدادها جزیی از میدان فرم (Form1 Scope) هستند. پس اگر یک متغیر در میدان فرم تعریف شود در هر چهار میدانی که در تصویر مشخص شده اند دیده خواهد شد.
تعریف متغیر را از رویداد کلیک شی btnNew پاک میکنیم و آن را به خط بعد از Inherits منتقل میکنیم :

```
    Inherits System.Windows.Forms.Form
    Dim mNumber As Long

    Private Sub btnNew_Click(…) Handles btnNew.Click
        Randomize()
        mNumber = Int((txtEnd.Text - txtStart.Text + 1) * Rnd() +
txtStart.Text)
```

```
    End Sub
```

با تعریف کردن متغیر mNumber در میدان فرم، این متغیر برای تمام فرم قابل دیدن می شود.
برای تکمیل اشیاء لازم برای بخش چهارم مساله، یک Label به منظور نشان دادن نتیجه مقایسه به فرم اضافه کنید و نام آن را برابر lblResult قرار دهید. اکنون می توانیم عمل مقایسه را در دکمه btnOK انجام دهیم، برای این کار از سه دستور شرطی ساده استفاده می کنیم :

```
Private Sub btnOK_Click(…) Handles btnOK.Click
    If txtNumber.Text < mNumber Then lblresult.text = "وارد بزرگتری عدد کنید"
    If txtNumber.Text > mNumber Then lblResult.Text = "وارد کوچکتری عدد کنید"
    If txtNumber.Text = mNumber Then lblResult.Text = "شدید برنده شما"
End Sub
```

فرض می کنیم عددی که کاربر وارد کرده (txtNumber.Text) از عدد انتخاب شده توسط برنامه (mNumber) کوچکتر باشد. در این صورت عبارت مقایسه ای (txtNumber.Text < mNumber) مقدار true بازمی گرداند و از آنجا که شرط برقرار بوده است دستوری که بعد از then نوشته شده است اجرا خواهد شد. سپس نوبت به اجرای خط دوم برنامه یعنی دستور if دوم میرسد، عبارت مقایسه ای مقدار false برمی گرداند چون طبق فرض txtNumber.Text کوچکتر از mNumber است پس عبارت بعد از then اجرا نخواهد شد. و اجرای برنامه به خط سوم میرسد. عبارت مقایسه ای شرط سوم هم false برمی گرداند و در نتیجه دستوری بعد از then اجرا نمی شود. پس در نهایت اگر عددی که کاربر وارد می کند از عدد ما کوچکتر باشد پیغام "عدد بزرگتری وارد کنید" نمایش داده می شود.

تبدیل صریح نوع داده :

مقایسه هایی که در این کد انجام شده است بسیار مبهم است. مقایسه یک رشته و یک عدد صحیح بر چه مبنایی انجام می شود؟ کدام یک بطور ضمنی به دیگری تبدیل می شود؟ پاسخ این است که رشته به عدد تبدیل می شود. ولی ما هرگز به شما توصیه نمی کنیم که از چنین مقایسه ای استفاده کنید. بهتر است قبل از اینکه عملگر مقایسه ای عمل مقایسه را انجام دهد، خودمان بطور صریح txtNumber.Text را به عدد صحیح تبدیل کنیم. در .NET. تبدیل های صریح با استفاده از کلاس Convert انجام می شود. در زیر لیستی از متدهای آن آمده است :

نام متد	وظیفه
ToChar	مقدار ورودی را به یک کاراکتر یونیکد تبدیل میکند.
ToString	مقدار ورودی را به رشته معادلش تبدیل میکند.
ToInt16	مقدار ورودی را به Int16 (short) تبدیل میکند.
ToInt32	مقدار ورودی را به Int32 (Integer) تبدیل میکند.
ToInt64	مقدار ورودی را به Int64 (Long) تبدیل میکند.
ToSingle	مقدار ورودی را به عدد اعشاری با دقت معمولی تبدیل میکند.
ToDouble	مقدار ورودی را به عدد اعشاری با دقت مضاعف تبدیل میکند.
ToBoolean	مقدار ورودی را به یک مقدار بولی تبدیل میکند.

بار دیگر به btnOK_Click بازگردیم، می توانیم هرجا که از txtNumber.Text استفاده کرده ایم از Convert.ToInt64(txtNumber.Text) استفاده کنیم. در این صورت مشکل واضح نبودن کد حل خواهد شد. ولی 3 بار عمل تبدیل به Long انجام می شود که باعث می شود منابع ما تا حدودی هرز برود. برای اینکه عمل تبدیل به Long فقط یک بار انجام شود، میتوانیم حاصل تبدیل را در یک متغیر long ذخیره کنیم :

```
Private Sub btnOK_Click(…) Handles btnOK.Click
    Dim tmp As Long = Convert.ToInt64(txtNumber.Text)
    If tmp < mNumber Then lblresult.text = "کنید وارد بزرگتری عدد"
    If tmp > mNumber Then lblResult.Text = "کنید وارد کوچکتری عدد"
    If tmp = mNumber Then lblResult.Text = "شدید برنده شما"
End Sub
```

این کد درست عمل میکند، و به اندازه کافی واضح است. اما کارایی مناسبی ندارد. ما میدانیم که اگر a<b درست باشد، a>b و a=b هر دو غلط خواهند بود. پس هیچ نیازی نیست که این شرطها را برسی کنیم. یا اگر a<b درست نباشد و a>b هم درست نباشد بطور بدیهی a=b دست خواهد بود؛ در این مورد دیگر لازم نیست a و b را مقایسه کنیم. اما برنامه ما در هر حال سه دستور شرطی را اجرا میکند که هرکدام یک عملگر مقایسه ای استفاده می کنند.

دستور شرطی کامل

دستور شرطی می تواند شامل کلمه کلیدی else باشد :

```
If condition Then statement1 Else statement2
```
درصورتی که condition درست (true) باشد، statment1 اجرا خواهد شد در غیر این صورت دستور statment2 اجرا خواهد شد.
لازم به ذکر است که statement1 و statement2 فقط می توانند یک دستور باشند. درصورتی که بیش از یک دستور در بدنه شرط لازم باشد باید از ساختار بلوکی استفاده شود که بصورت زیر است :

```
If condition Then
   statements1
Else
   statements2
End If
```

کلمه کلیدی Else میتواند وجود داشته باشد، یا حذف شود.
ساختار کامل دستور شرطی بصورت زیر است :

```
If condition1 Then
    statements1
ElseIf condition2 Then
    statements2
Else
    statements3
End If
```

در صورتی که condition1 درست (true) باشد دستورات statements1 اجرا خواهند شد و سپس اجرای برنامه به اولین خط بعد از end if منتقل می شود. در صورتی که condition1 درست نباشد و condition2 درست(true) باشد دستورات statements2 اجرا خواهند شد. اگر نه condition1 درست باشد نه condition2 آنگاه statments3 اجرا خواهد شد.
در یک دستور شرطی به هر تعدادی که مایل باشید می توانید از ElseIf استفاده کنید اما فقط استفاده از یک Else مجاز است.
با اطلاعات این جدید می توانیم کارایی برنامه را بالاتر ببریم :

```
If txtNumber.Text < mNumber Then
    lblResult.Text = "کنید وارد بزرگتری عدد"
ElseIf txtNumber.Text > mNumber Then
    lblResult.Text = "کنید وارد کوچکتری عدد"
Else
    lblResult.Text = "شدید برنده شما"
End If
```

در کد جدید حداکثر دو عملگر مقایسه ای اعمال می شود و حداقل یکی. (چرا؟)
میدان (Scope) دستور شرطی :
هر کدام از بلوکهای دستور شرطی (if) یک میدان مجزا است. به مثال زیر توجه کنید :

```
    If a < b Then
        Dim f As Integer
        f = 5
    Else
        f = 2 'Name f is not declared
    End If
```

پس اگر بخواهیم یک متغیر در همه بخشهای یک دستور if قابل دسترسی باشد باید آن را بیرون این دستور تعریف کنیم.

```
    'scope 0
    If a < b Then
        'scope 1
    ElseIf a > b Then
        'scope 2
    Else
        'scope 3
    End If
```

دستورات شرطی تو در تو

دستورات شرطی می توانند به هر تعداد که لازم باشد بصورت تو در تو استفاده شوند :

```
If a < b Then
    If x > y Then

    Else

    End If
Else

End If
```

در اینجا اگر a<b باشد، شرط دوم برسی می شود؛ درغیر این صورت کارهایی که در else دستور شرطی اول آمده انجام خواهد شد.

عملگرهای منطقی

تا بحال دستور شرطی، عملگرهای مقایسه ای را یاد گرفته ایم و دیدیم که چطور می توان دستور شرطی را تو در تو استفاده کرد. اما تا اینجا همه شرطهای ما ساده بوده اند و با یک عملگر مقایسه ای می توانستیم آنها راپیاده سازی کینم. اما عبارتهایی مانند "اگر b بین a و c باشد" را نمی توان با یک عملگر مقایسه ای پیاده سازی کرد. جمله "اگر b بین a و b باشد" را در ریاضی بصورت a < b < c نمایش می دهیم. اما از دید ویژوال بیسیک این عبارت معادل عبارت (a < b) < c می باشد. در این عبارت ابتدا a<b ارزشیابی می شود و حاصل آن یک مقدار بولی (فرض کنید true) می شود، سپس true با c مقایسه می شود! در حالی که ما می خواستیم c با b مقایسه شود!

برای پیاده سازی چنین عبارتهایی از عملگرهای منطقی استفاده می کنیم، مهمترین عملگرهای منطقی عملگرهای AND و OR و Not هستند. ابتدا عملگرهای And و or را بررسی می کینم

عملوند اول	عملوند دوم	AND	OR
false	false	false	false
true	false	false	true
false	true	false	true
true	true	true	true

```
If true And false Then statement اجرا نمیشود statement
If true And true Then statement اجرا میشود statement
If false or true Then statement اجرا میشود statement
```

به زبان ساده می توان گفت حاصل عملگر منطقی And وقتی true است که هر دو عملوند آن true باشند و حاصل عملگر or وقتی true می شود که حداقل یکی از عملگرهای آن true باشد.

با آنچه آموختیم می توانیم عبارت شرطی "اگر b بین a و b باشد" را نیز پیاده سازی کنیم :

```
If a < b And b < c Then
```

دستور بعد از then در صورتی اجرا خواهد شد که هم a<b باشد، هم b<c باشد.

عملگر not کار بسیار ساده ای انجام می دهد. این عملگر فقط یک عملوند می گیرد و اگر مقدار آن عملوند true باشد، آن را به false و اگر false باشد آن را به true تغییر می دهد.

عملوند	Not
false	true
true	false

```
If Not false Then statement اجرا میشود statement
If Not true Then statement اجرا نمیشود statement
```

عملگرهای شرطی را میتوان به هر تعداد که لازم باشد بدنبال هم استفاده کرد :

```
If a < b And b < c Or textbox1.text = "" And w = 83 Or Not x = 10 Then
```

برای اولویت دهی به عملگرهای منطقی هم می توان از پرانتز استفاده کرد، اما در حالت عادی اولویت not بیشتر از And و اولویت And نیز بیشتر از Or است.

دستور شرطی Select

دستور Select یک عبارت دریافت می کند و بسته به مقدار آن عبارت اجرای برنامه را به یکی از زیر بلوکهایش هدایت می کند. ساختار دستوری Select بصورت زیر است:

```
Select Case testexpression
  Case value1
    statements1
  Case value2
    statements2
  Case valueN
    statementsN
  Case Else
    ElseStatements
End Select
```

این دستور ابتدا مقدار testExpression را با Value1 مقایسه می کند. اگر مقدار آنها برابر بود بدنه Case Value1 را اجرا می کند، در غیر این صورت مقدار testExpression را با Value2 مقایسه می کند. اگر testExpression با Value برابر بود پس از اجرای بدنه case مربوطه، اجرای برنامه به اولین خط بعد از End Select منتقل می شود.

: Case Else

در صورتی که testExpression با هیچ کدام از value ها برابر نشود، اگر Case Else وجود داشته باشد؛ بدنه آن اجرا می شود. وظیفه case else در دستور select مانند وظیفه else در دستور if است.

برنامه ای آزمایشی ایجاد کنید و یک Label و یک دکمه و یک TextBox روی آن قرار دهید، سپس کد زیر را برای رویداد کلیک دکمه بنویسید.

```
Select Case Convert.ToInt32(TextBox1.Text)
    Case 1
        Label1.Text = "یک"
    Case 2
        Label1.Text = "دو"
    Case 3
        Label1.Text = "سه"
    Case 4
        Label1.Text = "چهار"
    Case Else
        Label1.Text = "ناشناخته"
End Select
```

اگر متن TextBox1 برابر با عدد 1 باشد، متن Label1 برابر با "یک" می شود و اگر برابر با 2 باشد...
درصورتی که متن TextBox هرچیز بجز اعداد 1 ، 2 ، 3 و 4 باشد متن "ناشناخته" نمایش داده خواهد شد.
در دستور Select از عملگرهای مقایسه ای نیز می توانیم استفاده کنیم:

```
Select Case Convert.ToInt32(TextBox1.Text)
    …
    Case 4
        Label1.Text = "چهار"
    Case Is > 10
        Label1.Text = "ده از بزرگتر"
    Case Is < 1
        Label1.Text = "یک از کوچکتر"
    Case Else
        Label1.Text = "ناشناخته"
End Select
```

بدنه دستور Case Is > 10 در صورتی اجرا می شود که Textbox1.Text از 10 بزرگتر باشد. بطور غیر رسمی می توان گفت در اینجا Is بجای testExpression ، به عنوان عملوند سمت چپ عملگر مقایسه قرار می گیرد.

بجای value میتوانیم از یک بازه نیز استفاده کنیم:

```
Select Case Convert.ToInt32(TextBox1.Text)
    …
    Case 5 To 10
        Label1.Text = "ده و پنج بین"
    Case Else
```

```
            Label1.Text = "ناشناخته"
    End Select
```

در اینجا بدنه Case 5 To 10 در صورتی اجرا خواهد شد که TextBox1.Text مقداری بزرگتر مساوی 5 و کوچکتر مساوی 10 داشته باشد.
و در نهایت اینکه مقادیر دستور case را میتوان با کاما (,) با هم ترکیب کرد.

```
    Select Case Convert.ToInt32(TextBox1.Text)
        Case 1, 2
            Label1.Text = "دو یا یک"
        Case 3
            Label1.Text = "سه"
        …
    End Select
```

بدنه دستور Case 1,2 درصورتی اجرا می شود که textbox1.text یا برابر 1 باشد، یا برابر 2. کاما در دستور select مانند عملگر منطقی OR استفاده می کند.

بحث دستورات شرطی به پایان رسیده است. در ادامه فصل به حلقه های تکرار خواهیم پرداخت.

تمرین:

1. کاربر می تواند قبل از اینکه بازی شروع شود (قبل از زدن دکمه "شروع") نیز عدد انتخاب کند. این امکان را از او بگیرید.
2. متاسفانه فراموش کرده ایم کد مربوط به دکمه btnClear را بنویسیم. شما این کار را انجام دهید.
3. اگر پس از زدن دکمه شروع و قبل از وارد کردن عدد، کاربر دکمه "تایید" را بزند برنامه با خطا متوقف می شود، این مشکل را حل کنید.
4. کاری کنید که کاربر نتواند عددی مانند 000310 وارد کند. به عبارت دیگر، نتواند اول عدد صفر وارد کند.
5. معادل عبارت not x=y بدون استفاده از عملگرهای منطقی چیست؟
6. پس از برنده شدن کاربر نباید بتواند عدد جدیدی انتخاب کند.
7. وقتی کاربر برنده شد به او بگویید که پس از چند حدس ناموفق، سرانجام موفق شده است.
8. کاری کنید که کاربر نتواند محدوده عددی تعیین کند که کمتر از 100 عدد دارد. یا در صورتی که محدوده انتخابی کمتر از 100 عضو داشت خودتان آن را اصلاح کنید.
9. راهی در برنامه تعبیه کنید که بتوانید زودتر برنده شوید! (تقلب)
10. اگر کاربر عددی وارد کرد که خارج از محدوده تعیین شده توسط خودش بود پیغام مناسبی به او نشان دهید.

کارگاه:

1. برنامه را طوری تغییر دهید که وقتی کاربر دکمه "تایید" را میزند عدد نوشته شده در txtNumber انتخاب (select) شود. و وقتی عدد جدیدی وارد میکند عدد قبلی پاک شود.
2. دکمه ای اضافه کنید که با استفاده از آن کاربر بتواند آخرین رقم عددی که وارد کرده پاک کند.

راهنمایی کارگاه :

1. کمی درباره متد ()SelectAll و خاصیت SelectedText از شی txtNumber تحقیق کنید.
2. اگر عدد به 10 تقسیم شود...

حل کارگاه :

1.

```
Private Sub btnOK_Click(…) Handles btnOK.Click
    ...
    txtNumber.SelectAll()
End Sub
Private Sub btnCalc1_Click(…) _
    Handles btnCalc0.Click, …
        txtNumber.SelectedText = sender.Text
End Sub
```

البته اگر تمرین تمرین 4 را حل کرده باشید این تابع کمی تغییر کرده است. جواب کامل را می توانید در CD همراه کتاب پیدا کنید. متد SelectAll تمام متن را select میکند و SelectedText متن select شده را برمی گرداند یا تغییر می دهد. در صورتی که هیچ متنی select نشده باشد و ما SelectedText را تغییر بدهیم مانند این است که همانجایی که cursor قرار دارد تایپ کنیم. پس به الحاق رشته ها نیاز نداریم.

2.

```
Private Sub btnBackspace_Click(…) Handles btnBackspace.Click
    txtNumber.Text = txtNumber.Text \ 10
End Sub
```

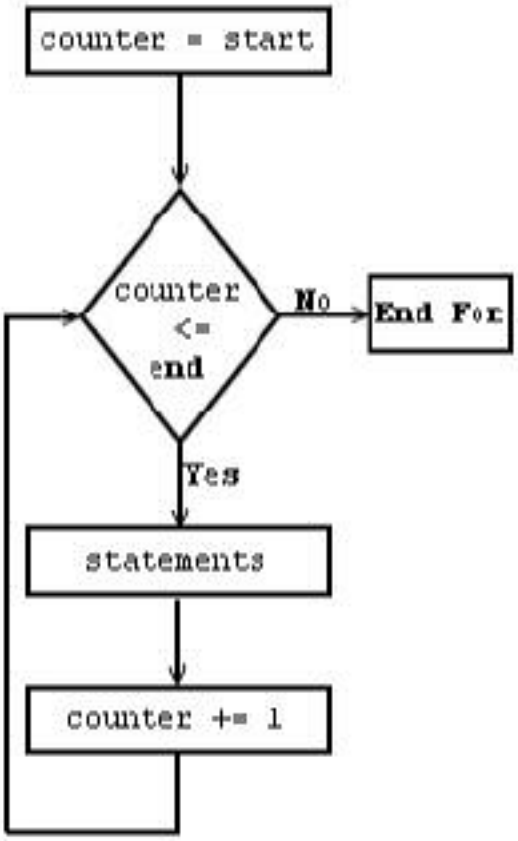
حلقه های تکرار

حلقه های تکرار وقتی استفاده می شوند که می خواهیم دستورات مشخصی، چندین بار اجرا شوند. به عبارت دیگر برای اجرای تکراری دستورات از حلقه های تکرار استفاده می شود.

حلقه for

این حلقه بسیار ساده و در عین حال بسیار پر کاربرد است. ساختار خلاصه شده این دستور بصورت زیر است :

```
For counter = start to end
    Statements
Next
```



الگوریتم حلقه for بصورت زیر است :

1. مقدار start را در counter جایگزین کن.
2. اگر counter <= end آنگاه وارد حلقه شو، درغیر این صورت به خط اول بعد از Next برو.
3. Statements را اجرا کن
4. (کلمه کلیدی Next) یکی به counter اضافه کن و به مرحله 2 بازگرد.

از حلقه for وقتی استفاده می کنیم که می خواهیم دستوراتی را به تعداد دفعات معین تکرار کنیم. برای مثال اگر بخواهیم 10تا از یک کاراکتر را در یک Textbox بنویسیم از حلقه for استفاده می کنیم.

```
Dim i As Integer
For i = 1 To 10
    TextBox1.Text += "A"
Next
```

عبارت TextBox1.Text += "A" معادل TextBox1.Text = TextBox1.Text + "A" است.

پس از اجرا به متن TextBox1 ده کاراکتر "A" اضافه می شود.

در اینجا ما از متغیر i هیچ استفاده ای نکردیم. متغیر شمارنده حلقه for بسیار کارآمد است، در زیر تکه برنامه ای آمده است که میتواند عددهای یک تا ده را در یک جعبه متن بنویسد.

```
Dim i As Integer
For i = 1 To 10
    TextBox1.Text += (i & " ")
Next
```

در اینجا ابتدا متغیر i که شمارنده حلقه است، برابر 1 می شود، سپس بدنه حلقه اجرا می شود که باعث می شود مقدار یک (با یک فاصله) به TextBox1.Text اضافه شود. پس از اجرای بدنه حلقه اجرای دستورات به خط Next میرسد و i برابر 2 می شود، از آنجا که 10=>2 است بدنه حلقه بار دیگر اجرا می شود. این کار تا زمانی انجام میشود که i=10 شود، در این مرحله پس از اینکه بدنه حلقه اجرا شد، i=11 می شود و از آنجا که 11=>10 نیست اجرای برنامه به اولین خط بعد از Next منتقل می شود. و کار حلقه به پایان می رسد.

ساختار کامل حلقه for

ساختار کامل حلقه for بصورت زیر است :

```
For counter [as typename] = start to end [step s]
    [statements]
Next
```

در ساختار دستورات، بخشهایی که بین [ ] قرار می گیرند اختیاری هستند. همانطور که می بینید لازم نیست متغیر شمارنده حلقه قبل از حلقه تعریف شود، بلکه این کار می تواند درون خود دستور انجام شود، مثال قبل را میتوان بصورت زیر نوشت :

```
For i As Integer = 1 To 10
    TextBox1.Text += (i & " ")
Next
```

البته این امکان از جمله تغییراتی است که در Framework نسخه 1.1 داده شده است بنابراین فقط در صورتی می توانید متغیر حلقه را داخل حلقه تعریف کنید که ویژوال استدیو شما نسخه 2003 باشد. در این صورت متغیر i جزو میدان (scope) حلقه for تعریف می شود و بیرون حلقه for قابل دسترسی نیست. لابد متوجه شدید که بدنه حلقه for نیز یک scope است.

step : وقتی حلقه for را توضیح دادیم، در مرحله آخر از یک دور حلقه گفتیم که شمارنده حلقه یکی زیاد می شود. این گفته در مورد ساختاری که بیان کرده بودیم درست است اما بهتر است بگوییم در پایان هر دور اجرای حلقه، شمارنده بعلاوه s می شود که به s ، گام حلقه(step) می گویند. s میتواند هر مقدار عددی دلخواهی باشد. فقط توجه داشته باشید که اگر step را اعشاری تعیین کردید باید شمارنده حلقه نیز اعشاری باشد. در غیر این صورت حلقه تا بی نهایت ادامه پیدا میکند.(چرا؟ راهنمایی: به تبدیل داده ضمنی فکر کنید.)

گام حلقه for می تواند عدد منفی نیز باشد. در این صورت شرط counter<=end به counter>=end تغییر میکند. برای شمارش نزولی از step منفی استفاده می شود :

```
For i As Single = 10 To -10 Step -1
    TextBox1.Text += (i & " ")
Next
```

این حلقه اعداد 10 تا 10- را در TextBox1.Text می نویسد.

برای اینکه بیشتر با for آشنا شوید دو برنامه کوچک خواهیم نوشت.

شمارش کاراکتر

اولین برنامه، برنامه ای است که یک کاراکتر و یک رشته از کاربر می گیرد و تعداد آن کاراکتر در رشته را می شمارد. برای نوشتن این برنامه باید درباره رشته ها بیشتر بدانیم.

اگرچه ما رشته ها جزو داده های پایه برسی کردیم، اما نوع داده رشته (string) در ویژوال بیسیک یک کلاس(class) است و نمونه هایی که از روی آن تعریف می شوند شی هستند. البته این مطلب درباره تمام انواع پایه دیگر در .NET نیز صادق است.

شی رشته خواص و متدهای مختلفی دارد که در طول کتاب با آنها آشنا خواهید شد. در این بخش با دو خاصیت شی رشته آشنا خواهیم شد. خاصیت Length طول رشته را بر میگرداند، این خاصیت فقط خواندنی (ReadOnly) است و نمی توان مقدار آن را با عملگر جایگزینی تغییر داد.

```
Dim s As String = "hello"
Dim l As Integer = s.Length 'l=5
                   's.Length = 0 'Error : Property Length is ReadOnly
```

دومین خاصیتی که در اینجا از آن استفاده خواهیم کرد خاصیت Chars(index) است. این خاصیتی با خاصیتهایی که تابحال دیدیم کمی فرق می کند. خاصیت Chars یک اندیس به عنوان ورودی می گیرد و کاراکتری که دارای آن اندیس است برمی گرداند. با توجه به اینکه این اندیس دهی از عدد صفر شروع میشود char(0) کاراکتر اول رشته و char(1) کاراکتر دوم رشته را باز می گرداند. (این خاصیت هم فقط خواندنی است)

```
Dim s As String = "hello"
Dim c As Char = s.Chars(0) 's="h"
's.Chars(1) = "x" 'Error : Property chars is ReadOnly
```

با توجه به اینکه Length طول رشته را به ما می دهد و اولین اندیس chars صفر است، پس کاراکتر آخر رشته s.chars(s.length-1) خواهد بود.

حال که اطلاعات کافی برای نوشتن برنامه شمارش کاراکتر داریم اشیاء لازم برای این برنامه را روی فرم قرار دهید تا کار را شروع کنیم :

لیست اشیایی که لازم داریم در جدول زیر آمده است :

نوع شی	نام Name	Text
Button	btnCount	بشمار
TextBox	txtChar	

TextBox	txtStr
Label	lblResult

میتوانید دو Label هم برای توصیف TextBoxها در کنار آنها قرار دهید.

برای شمردن کاراکتر ها باید کاراکتری را که در txtChar.Text وارد شده با تک تک کاراکترهای رشته txtStr.Text کنیم؛ و در صورتی که یکی بودند مقدار متغیری که به عنوان شمارنده استفاده می کنیم یکی اضافه کنیم. پس یک کار تکراری داریم و باید از حلقه تکرار استفاده کنیم. تعداد دفعات این تکرار محدود (برابر با طول رشته txtStr.Text) است پس می توانیم از حلقه for استفاده کنیم.

```
    Private Sub btnCount_Click(…) Handles btnCount.Click
        Dim count As Integer = 0
        For i As Integer = 0 To txtStr.Text.Length - 1
            If txtStr.Text.Chars(i) = txtChar.Text.Chars(0) Then count
+= 1
        Next
        lblResult.Text = count
    End Sub
```

ابتدا متغیر count که جواب مساله ما را در خود نگه می دارد، تعریف می شود و مقدار اولیه صفر می گیرد. سپس متغیر i تعریف شده و مقدار 0 می گیرد و اگر (txtStr.Text.Length – 1) کوچکتر از 0 نباشد وارد حلقه می شود. اگر کاراکتر اول از رشته txtChar.Text با کاراکتر iام از رشته txtStr.Text برابر باشد مقدار count را یکی اضافه می کند. این کار تا وقتی شمارنده حلقه for به txtStr.Text.Length – 1 نرسیده ادامه پیدا می کند و در نهایت پس از خروج از حلقه مقدار count در lblResult.Text جایگزین می شود.

سوالی که پیش می آید این است که اگر کاربر در txtChar فقط یک کاراکتر وارد میکند، چه دلیلی دارد که ما برای گرفتن آن کاراکتر از txtChar.Text.Chars(0) استفاده کنیم؟ این کد را به این دلیل نوشتیم که کاربر می تواند چندین کاراکتر نیز در txtChar وارد کند. البته بهتر است که اجازه این کار را به او ندهیم. برای اینکه تعداد کاراکترهای TextBox را محدود کنید می توانید از خاصیت MaxLength آن استفاده کنید. اگر MaxLength را برابر با 1 قرار دهید کاربر فقط می تواند یک کاراکتر وارد کند. در آن صورت نیازی به txtChar.Text.Chars(0) نیست و می توانیم آن را با txtChar.Text عوض کنیم.

تشخیص عدد اول

عدد اول عددی است که به هیچ عددی بجز خودش و یک بخشپذیر نباشد. می خواهیم برنامه ای بنویسیم که عددی از کاربر بگیرد و بگوید این عدد اول است یا خیر. با توجه به صورت مساله ظاهرا باید عددی که از کاربر می گیریم به تمام اعداد حقیقی تقسیم کنیم! ولی ما میدانیم که برای تشخیص اول بودن یک عدد نیازی نیست که آن را به همه اعداد کوچکتر از یک و بزرگتر از خودش تقسیم کنیم. در نتیجه باید عددی که از کاربر می گیریم به همه اعداد بزرگتر از یک و کوچکتر از خود عدد تقسیم کنیم. در اینجا نیز مانند مثال قبلی همه وجنات از آن حکایت میکند که باید یک حلقه for استفاده کنیم. اشیاء مورد نیاز در جدول زیر آمده است.

نوع شی	نام Name	Text
Button	btnPrimeNumber	Is prime?
TextBox	txtNumber	
Label	lblResult	

```
    Private Sub btnPrimeNumber_Click(…) Handles btnPrimeNumber.Click
        For i As Integer = 2 To txtNumber.Text - 1
            If txtNumber.Text Mod i = 0 Then
                lblIsPrime.Text = "نیست اول عدد"
            End If
```

```
        Next
    End Sub
```

این حلقه دقیقا همان کاری را میکند که کمی قبل گفتیم، باقیمانده عددی که کاربر وارد کرده است را به تمام اعداد بزرگتر و مساوی 2 (بزرگتر از 1) و کوچکتر از خود عدد تقسیم می کند و در صورتی که این باقی مانده برابر با صفر بود، به کاربر می گوید که عددی که وارد کرده اول نیست. اما چیزی درباره اول بودن عدد نمی گوید. اگر کمی بی دقتی کنیم ممکن است از کد زیر برای تشخیص اول بودن عدد استفاده کنیم :

```
    For i As Integer = 2 To txtNumber.Text - 1
        If txtNumber.Text Mod i = 0 Then
            lblIsPrime.Text = "نیست اول عدد"
        Else
            lblIsPrime.Text = "است اول عدد"
        End If
    Next
```

فکر میکند این کد درست کار خواهد کرد؟ برنامه را اجرا کنید و آن را با عددهای 3 ، 5 ، 8 ، 15 و هرعدد دیگری که مایلید آزمایش کنید. از دید این برنامه همه اعداد اول هستند! این خطا از آنجا ناشی شده که ما با بی دقتی تعریف مساله را فراموش کرده ایم. صورت مساله میگوید عددی اول است که به هیچ عددی بین یک و خود عدد بخشپذیر نباشد، اما ما به محض اینکه یک عدد پیدا کردیم که عدد کاربر به آن بخشپذیر نیست گفتیم این عدد، عدد اول است. برای اینکه حلقه ها را بهتر درک کنیم از جدولهایی استفاده میکنیم که مقدارهای مهمی که حلقه آنها را تغییر می دهد، به ازای هر مقداری که شمارنده می گیرد، در آن می نویسیم. در جدول زیر فرض میکنیم عددی که کاربر انتخاب کرده 8 است:

شمارنده i	txtNumber.Text mod i	lblIsPrime.Text
2	0	عدد اول نیست
3	2	عدد اول است
4	0	عدد اول نیست
5	3	عدد اول است
6	2	عدد اول است
7	1	عدد اول است

اکنون میدانیم که چه چیزی غلط است، اما هنوز راه حلی برای تشخیص عدد اول نداریم. راه های درست زیادی ممکن است به فکر شما رسیده باشد، یکی از این راه ها این است که ابتدا فرض کنیم که عدد اول است یعنی قبل از حلقه مقدار txtNumber.Text را برابر "عدد اول است" قرار دهیم؛ اگر عدد اول نبود، حلقه مقدار txtNumber.Text را تغییر خواهد داد، اگرنه حتما عدد اول است و مقدار ما هم بی تغییر باقی خواهد ماند.

```
    lblIsPrime.Text = "است اول عدد"
    For i As Integer = 2 To txtNumber.Text - 1
        If txtNumber.Text Mod i = 0 Then
            lblIsPrime.Text = "نیست اول عدد"
        End If
    Next
```

این برنامه درست کار می کند و همه اعداد اول را تشخیص می دهد، ولی آیا همین که برنامه درست کار میکند کافی است؟ برنامه را با عددی مانند 1343000 آزمایش کنید. برای اعداد بزرگ، این برنامه یک فاجعه است! اینکه چطور یک برنامه را کارآمدتر کنیم هیچ راه مشخصی ندارد، در حقیقت سرعت برنامه تا حدود زیادی بستگی به معلومات ، تجربه و هوش برنامه نویس دارد. این ابتکارهای شما است که می تواند یک برنامه را سریعتر و کارآمدتر کند.

همیشه کم کردن تعداد حلقه ها یا کوچکتر کردن تعداد تکرار در هر حلقه؛ سرعت برنامه را بیشتر می کند. در اینجا نیز ما سعی می کنیم با استفاده از همین قاعده کلی سرعت برنامه را بیشتر کنیم.

می دانیم که هیچ کدام از اعداد زوج، بجز دو، عدد اول نیستند. با استفاده از همین حقیقت میتوانیم تعداد تکرار حلقه را نصف کنیم. کافی است زوج بودن عدد را قبل از حلقه آزمایش کنیم و برای اعداد زوج اصلا وارد حلقه نشویم. در این صورت میتوانیم شروع حلقه را 3 قرار دهیم و گام آن را بجای یک، به دو تغییر دهیم:

```
    lblIsPrime.Text = "است اول عدد"
    If txtNumber.Text Mod 2 = 0 And txtNumber.Text > "2" Then
        lblIsPrime.Text = "نیست اول عدد"
    Else
        For i As Integer = 3 To txtNumber.Text - 1 Step 2
```

```
            If txtNumber.Text Mod i = 0 Then
                lblIsPrime.Text = "نیست اول عدد"
            End If
        Next
    End If
```

اگر عددی که کاربر وارد کرده است یک عدد زوج (بخش پذیر بر 2) و بزرگتر از 2 باشد، خیلی سریع به کاربر اعلام می کنیم که عددی که وارد کرده است، اول نیست. در غیر این صورت عدد را به همه عددهای فرد کوچکتر از خودش و بزرگتر مساوی 3 تقسیم می کنیم.
دراین برنامه تکلیف اعداد زوج بزرگتر از 2 و اعداد فرد مشخص شده است اما برای خود 2 چه اتفاقی می افتد؟ وقتی کاربر عدد 2 را وارد کند، مقدار عبارت شرطی ما برابر با false خواهد شد (بخاطر عملگر and و اینکه 2<2 نیست) پس عدد 2 به قسمت else از شرط می رود، اما آنجا هم وارد حلقه نخواهد شد چون برای عدد 2، پایان حلقه (txtNumber.Text-1) کوچکتر از شروع حلقه (3) می شود. به این ترتیب 2 بدون اینکه وارد حلقه شود از حلقه خارج می شود و مقدار lblIsPrime برابر "عدد اول است" باقی می ماند، این همان چیزی است که ما می خواستیم.
باز هم می توانیم سرعت برنامه را بیشتر کنیم. بعد از اینکه فهمیدیم عددی که کاربر وارد کرده به یک عدد غیر از یک و خودش بخشپذیر بوده میتوانیم بگوییم که آن عدد اول نیست و لازم نیست کار حلقه ادامه پیدا کند. با دستور Exit For می توانیم قبل از پایان کار حلقه، از آن خارج شویم :

```
...
If txtNumber.Text Mod i = 0 Then
    lblIsPrime.Text = "نیست اول عدد"
    Exit For
End If
...
```

اکنون می توانیم شرطی که برای اعداد بخشپذیر به 2 اضافه کرده بودیم حذف کنیم و کران پایین حلقه را برابر 2 قرار دهیم، بدون اینکه در سرعت برنامه تغییری ایجاد شود. (چرا؟)
تمرین :

1. با توجه به اینکه "عددی که به هیچ کدام از اعداد بزرگتر از یک و کوچکتر از جزر خودش بخشپذیر نباشد، عدد اول است." کاری کنید که سرعت برنامه "تشخیص عدد اول" بیشتر شود.
2. برنامه را طوری تغییر دهید که برنامه برای اعداد کوچکتر از یک پیام مناسبی به کاربر نشان دهد.

حلقه While

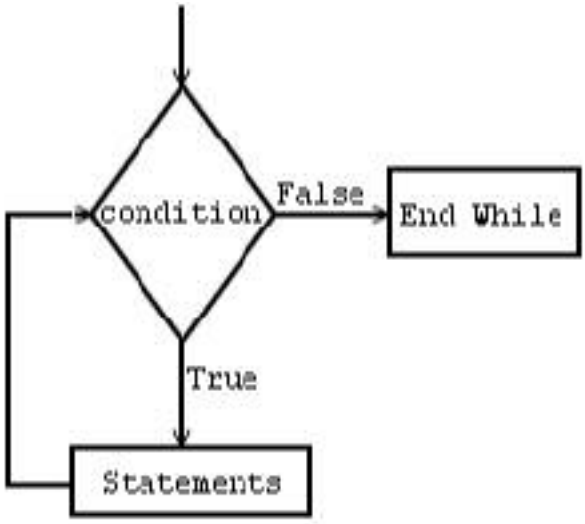
دومین حلقه تکراری که بررسی می کنیم حلقه تکرار while است، حلقه while در مواردی بکار میرود که تعداد تکرار مشخص نیست و بدنه حلقه باید تا زمانی که شرط حلقه برقرار است اجرا شود :

```
While condition
   statements
End While
```

الگوریتم حلقه while بصورت زیر است:

1. اگر شرط (condition) برقرار است وارد حلقه شو، اگر نه از حلقه خارج شو.
2. دستورات را اجرا کن
3. برگرد به 1

توجه داشته باشید که در حلقه while ابتدا شرط حلقه برسی می شود و فقط در صورت درستی شرط، برنامه بدنه حلقه را اجرا خواهد کرد. پس اگر از ابتدا شرط معتبر برقرار نباشد، بدنه حلقه هرگز اجرا نخواهد شد.



مثال : می خواهیم برنامه ای بنویسیم که عددی از کاربر بگیرد و بگوید کوچکترین عددی که بـه 3 بخشـپذیر و از عدد کاربر بزرگتر باشد چند اسـت.
برای حل این مساله میتوانیم فرض کنیم جواب عدد x است و تا وقتی که x کوچکتر از عدد کاربر اسـت آن را بعلاوه 3 کنیم و حاصل جمع را در خود x جایگزین کنیم. برای این کار نمی توانیم از حلقه for استفاده کنـیم، چون تعداد تکرار مشخص نیست. پس از حلقه while استفاده می کنیم:

```
Dim result As Integer = 0
While result <= TextBox1.Text
    result += 3
End While
Label1.Text = result
```

البته روش کارآمدتری هم برای حل این مساله ساده وجود دارد، که نیـازی بـه حلقـه نـدارد. پیـدا کـردن ایـن روش را به شما واگذار می کنیم.
در صورتی که شرط حلقه while همیشـه درسـت باشـد، برنامـه هیچگـاه از حلقـه خـارج نخواهـد شـد و بـه اصطلاح در یک حلقه بینهایت گرفتار خواهد شد :

```
While 1 < 2

End While
```

یا

```
While True

End While
```

دستور exit while در حلقه while همان کاری را می کند که دسـتور exit for در حلقـه for مـی کـرد. یعنـی قبل از اینکه ارزش عبارت شرطی حلقه while برابر false شود، به کار حلقه خاتمه می دهد.
حلقه while را می توان در حالت خاص، مانند حلقه for استفاده کرد :

```
Dim i As Integer = 2
While i <= 10
    i += 1
End While
```

معادل است با :

```
For i As Integer = 2 To 10

Next
```

اما استفاده کردن از حلقه while بجای for از خوانایی برنامه کم می کند.

حلقه Do Loop

این حلقه نیز مانند حلقه while در مواردی استفاده مـی شـود کـه تعـداد تکـرار دسـتورات مشـخص نباشـد.
حلقه do چهار حالت مختلف دارد :

```
Do While condition

Loop
```
(حالت 1)

```
Do

Loop While condition
```
(حالت 2)

```
Do Until condition

Loop
```
(حالت 3)

```
Do

Loop Until condition
```
(حالت 4)

حالت 1 : این حالت دقیقا مانند حلقه while عمل می کند یعنـی ابتـدا شـرط را بررسـی میکنـد و در صـورت درست بودن آن، بدنه حلقه اجرا می شود.
حالت 2 : در این حالت ابتدا بدنه حلقه اجرا می شود و سپس شرط بررسـی مـی شـود و در صـورت برقـرار بودن شرط، حلقه تکرار می شود. بنابراین در این حالت بدنه حلقه حداقل یک بار اجرا می شود.
حالت 3 : در این حالت (بجای کلمه کلیدی while از until استفاده شده) ابتدا شرط بررسی می شـود و در صورتی که شرط برقرار *نبود* بدنه حلقه اجرا می شود. حلقه ای که از کلمه کلیدی until استفاده می کند تا زمانی اجرا می شود که condition=false باشد. به عبـارت دیگـر حلقـه do until آنقـدر اجـرا مـی شـود کـه شرط آن برقرار شود.
حالت 4 : در این حالت ابتدا بدنه حلقه اجرا می شود و سپس در صورتی که شرط برقرار نبـود، حلقـه تکـرار می شود.
دستور Exit Do می تواند قبل از موعد به کار حلقه do loop پایان بدهد.
حلقه do loop را می توان بصورت زیر نیز استفاده کرد :

```
Do

Loop
```

در این صورت باید حتما از دستور exit do برای خروج از حلقه استفاده کنید، در غیر این صورت برنامـه در یـک حلقه بینهایت گرفتار خواهد شد.

مثالهای بیشتر برای حلقه های while و do loop را به فصلهای بعد موکول می کنیم.

پروژه برنامه نویسی :

1. برنامه ای بنویسید که کاربر عددی (در بازه مشخص) انتخاب کند و کامپیوتر با استفاده از راهنمایی کاربر آن عدد را پیدا کند.
2. یک برنامه ماشین حساب بنویسید. رابط کاربر این ماشین حساب باید به کاربر ایـن امکـان را بدهـد که همه کارهایش را با موس انجام دهد.

اولین کلاس

در این فصل مساله "بستنی فروشی" را مطرح می کنم. اما برنامه این مساله -برخلاف مساله های قبلی که طرح کردیم- در همین فصل تکمیل نمی شود. در حقیقت از اینجا تا پایان کتاب ما با این مساله سرو کار داریم. در این فصل اولین کلاس خودمان را ایجاد خواهیم کرد و کار کردن با اشیایی از آن کلاس را به فصل بعد واگذار می کنیم.

مساله بستنی فروشی

آقای محمدی چند مغازه فروش بستنی سنتی دارد. او در این مغازه ها، بستنی زعفرانی سنتی نانی و لیوانی میفروشد. آقای محمدی می خواهد آخر روز بتواند حساب کند هر کدام از مغازه ها چه مقدار بستنی فروخته و چه تعداد از لیوانها و نانها کم شده و چقدر پول دریافت کرده است. ما می خواهیم برای آقای محمدی برنامه ای بنویسم که این کارها را برای او انجام بدهد.

برای ساده شدن کار فرض می کنیم در هر مغازه شخصی مسوول این است که به محض فروش بستنی به آقای محمدی تلفن بزنند و گزارش فروش بدهند. این فرض کار را برای برنامه ما تاحد زیادی ساده می کند؛ چون وظیفه جمع آوری اطلاعات از مغازه ها را از دوش برنامه ما بر می دارد. بنابراین هر وقت که تلفن آقای محمدی زنگ می زند، او باید اطلاعات فروش را از فروشنده گرفته و در برنامه وارد کند و در پایان روز وضعیت فروش هر مغازه را بررسی کند.

اشیاء مساله

اولین قدم در نوشتن برنامه شی گرا این است که اشیاء مساله را شناسایی کنیم. در نگاه اول مساله مغازه های بستنی فروشی آقای محمدی اشیاء بسیار زیادی دارد. بستنی ها، نانها، لیوانها، فروشنده ها، مسوول صندوق، رابط آقای محمدی، بستنی فروشی، خود آقای محمدی و حتی کامپیوتر آقای محمدی! واقعا همه این اشیاء به برنامه ما ارتباط ندارند. ما باید ببینیم درباره کدام شی می خواهیم در برنامه خودمان اطلاعات نگه داریم. این کاملاً به صورت مساله ما مربوط است. اگر آقای محمدی می خواست درباره وضعیت کارمندانش اطلاعات داشته باشد، شاید فروشنده ها شی مورد نظر ما بودند. اما آقای محمدی بیشتر به حساب و کتاب مغازه ها علاقه دارد تا وضعیت فروشندها، پس فروشنده ها شی مورد نظر ما نیستند، صندوقدار و رابط هم وضعیتشان مشابه فروشنده ها است.

مناسبترین شی برای این برنامه، خود بستنی فروشی است. می توانیم هر بستنی فروشی را یک نمونه از کلاس بستنی فروشی درنظر بگیریم که داده هایی مانند تعداد نان، مقدار بستنی، موجودی پول و ... متدهایی مانند فروش بستنی و دریافت پول و ... و رویدادهایی مانند فروخته شدن بستنی، دریافت بستنی از آقای محمدی و ... دارد.

البته یک راه دیگر این است بجای بستنی فروشی، نانها و بستنی ها و پولها را شی درنظر بگیریم. اما با این روش برنامه تا حدودی سخت تر می شود و باید برای توصیف هر مغازه چندین شی درنظر بگیریم. به وضعیتی فکر کنید که بخواهیم یک مغازه بستنی فروشی جدید را به برنامه اضافه کنیم، در حالتی که بستنی فروشی یک شی باشد فقط باید یک شی به برنامه اضافه کنیم، اما اگر هر بستنی فروشی چندین شی مجزا باشد باید چندین شی به برنامه اضافه کنیم. همیشه حالتی که قابلیت توسعه بیشتر و راحت تری داشته باشد مطلوبتر است.

کلاس بستنی فروشی

کلاس بستنی فروشی تعیین میکند که هر نمونه از کلاس بستنی فروشی (شی بستنی فروشی) چه داده ها، متدها(Method) و رویدادهایی(Event) دارد. پیش از اینکه برنامه نویسی را شروع کنیم باید به دقت کلاس بستنی فروشی را بررسی کنیم و اجزاء آن را شناسایی کنیم.

داده ها :

اگرچه یک بستنی فروشی واقعی داده های بسیاری دارد ولی ما (همانطور که در فصل سوم گفتیم) یک دید انتزاعی نسبت به آن داریم و فقط آن داده هایی را درنظر می گیریم که به مساله ما مربوط می شود. برای مثال رنگ کف مغازه اثری در تعداد بستنی ها و سایر داده هایی که برای ما مهم است ندارد؛ بنابراین آن را جزو داده های کلاس بستنی فروشی درنظر نمی گیریم.

داده هایی که برای ما مهم است در زیر آمده است :

- مقدار بستنی موجود : برحسب گرم
- تعداد نان موجود
- تعداد لیوان موجود
- تعداد قاشق موجود
- مقدار پول موجود : بر حسب ریال

ما در اینجا فرض کردیم که مغازه های آقای محمدی بستنی، نان، لیوان و قاشق را از آقای محمدی می گیرند و بنا به درخواست مشتری بستنی را آماده می کنند. به همین دلیل مقدار بستنی و تعداد نان و ... را داده های بستنی فروشی درنظر گرفتیم. اما اگر بستنی فروشی های آقای محمدی بستنی نانی و لیوانی آماده را از آقای محمدی می گرفتند می توانستیم تعداد بستنی نانی و تعداد بستنی لیوانی را اشیاء بستنی فروشی درنظر بگیریم.

متدها (Methods) :
اگر قرار بود بستنی های آماده فروخته شود می توانستیم متدی مانند "فروختن یک بستنی لیوانی" داشته باشیم، اما وقتی بستنی به سفارش مشتری درست می شود ممکن است یکی از مشتریها بستنی بزرگتری بخواهد یا دیگری از گرسنگی نان اضافه بخواهد! پس باید متدهای فروختن نان و بستنی و... جدا از هم باشند. البته ممکن است تعدادی از این فرضیات را برای سادگی کار در نظر نگیریم . اما در مثالهایی که آخرین فصلهای کتاب برای تکمیل این مساله می آوریم از همه این فرضیات استفاده می کنیم .

- فروختن 50 گرم بستنی (کمی بعد این متد را به "فروختن x گرم بستنی" تغییر می دهیم، فعلا برای سادگی فرض کردیم وزن بستنی مضربی از 50 است.)
- فروختن یک نان (هر بستنی حداقل دو نان دارد، پس برای فروختن یک بستنی نانی حداقل دو بار باید این متد فراخوانی شود.)
- فروختن یک لیوان
- فروختن یک قاشق
- گرفتن پول بستنی از مشتری

رویدادها (Events) :
در اینجا هم دید انتزاعی باعث می شود که تعداد رویدادهای کلاس کمتر شود و پیاده سازی آن امکان پذیر گردد. ما می توانیم رویدادهای "فروخته شدن یک قاشق" یا "فروخته شدن یک بستنی" را در نظر نگیریم؛ چون چندان فایده ای برای آقای محمدی ندارند. اما در عوض رویدادهای زیر را برای کلاس درنظر بگیریم:

- تمام شدن بستنی
- تمام شدن نان
- تمام شدن لیوان
- تمام شدن قاشق

ایجاد یک کلاس جدید
اکنون که می دانیم کلاس ما چه چیزهایی لازم دارد می توانیم برنامه نویسی را شروع کنیم. پروژه جدیدی ایجاد کنید و نام مناسبی برای آن تعیین کنید. سپس از منوی Project آیتم Add Class را انتخاب کنید و نام آن را Bastani.vb بگذارید.
کلاس Bastani ایجاد شده است. اگر روی Bastani.vb در Solution Explorer دبل کلیک کنید می توانید کد آن را مشاهده کنید :

```
Public Class bastani

End Class
```

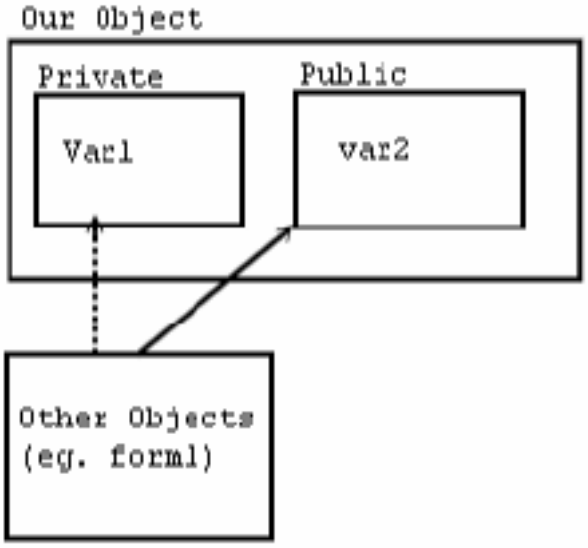
پس از ایجاد کلاس باید داده ها و متدها و رویدادها را ایجاد کنیم.

داده ها :
همانطور که قبل از این گفته شد، داده ها می توانند به یکی از دو صورت فیلد یا خصوصیت باشند. فیلدها ساختار ساده تری دارند. در حقیقت یک فیلد، چیزی بیشتر از یک نمونه ساده از یک نوع داده (مثلا یک متغیر) نیست. کد زیر یک فیلد برای ذخیره کردن تعداد نانها ایجاد میکند.

```
Class bastani
    Public TedadNoon As Integer
End Class
```

همانطور که می بینید برای تعریف متغیر(فیلد)، بجای استفاده از کلمه کلیدی dim از کلمه کلیدی public استفاده کرده ایم. محدوده دید متغیری که با public تعریف شود با متغیری که با dim تعریف شود تفاوت دارد. پیش از این درباره میدان دید (scope) متغیرها چیزهایی یاد گرفتیم، scope هایی که تا بحال دیدیم برنامه را به دو بخش تقسیم می کردند. بخشی که خارج از scope بود نمی توانست متغیر را ببیند و خود scope می توانست آن را ببیند. در مورد توابع و سایر scopeهایی که در توابع ایجاد می شوند بحث میدان دید به همینجا ختم می شود. اما برای classها وضع کمی فرق می کند. ما می توانیم تعیین کنیم که متغیری که تعریف می کنیم بیرون این کلاس دیده شود یا نه. برای این کار از کلمه های کلیدی public و private استفاده می کنیم.

کلمه کلیدی public اعلام می کند که همه اشیاء و کلاسهایی که بیرون از این کلاس وجود دارند می توانند به این متغیر(فیلد) دسترسی داشته باشند.
کلمه کلیدی private اعلام می کند که فقط همین کلاس می تواند به این متغیر(فیلد) دسترسی داشته باشد.



اکنون به منظور اینکه تفاوت public و Private را در عمل ببینیم، ابتدا یک متغیر private به نام test در کلاس bastani ایجاد می کنیم و سپس یک شی از نوع bastani در form برنامه درست می کنیم.
متغیر خصوصی (private) test در زیر تعریف شده است :

```
Class bastani
    Public TedadNan As Integer
    Private test As Integer
End Class
```

ایجاد شی از روی کلاس

تعریف یک شی از روی کلاس دقیقا مانند تعریف هر نمونه داده دیگری است. و با استفاده از ساختار

```
[public | private ] dim varname as vartype
```

تعریف می شود.

```
    Private Sub btnTest_Click(…) Handles btnTest.Click
        Dim t As bastani
    End Sub
```

با ذهنیتی که از کار با داده های پایه داشتیم، بنظر می رسد که اکنون بتوانیم از t استفاده کنیم. برای مثال :

```
    Private Sub btnTest_Click(…) Handles btnTest.Click
        Dim t As bastani
        t.TedadNan = 10
    End Sub
```

ولی اشیاء یک تفاوت عمده با داده های پایه (و بعضی دیگر از انواع داده) دارند. وقتی یک متغیر از نوع داده های پایه ایجاد می کردیم، حافظه مورد نیاز برای آن از سیستم گرفته می شد. ولی خط تعریف شی حافظه مورد نیاز برای آن شی را از سیستم نمی گیرد. ما می توانیم پس از ایجاد شی، با استفاده از کلمه کلیدی new برای شی حافظه بگیریم:

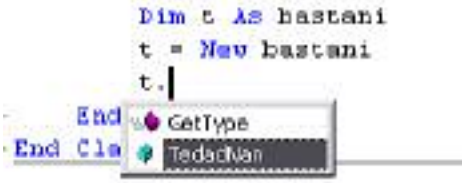
```
    Dim t As bastani
    t = New bastani
    t.TedadNan = 10
```

خط دوم باعث می شود برنامه حافظه ای مطابق با آنچه کلاس bastani تعیین کرده از سیستم عامل بگیرد. البته می توانستیم خط اول و دوم را با هم ترکیب کنیم و به صورت

```
    Dim t As New bastani
```

بنویسیم. که دقیقا معادل همان دو خط اول تکه کد قبلی است.

پس از گرفتن حافظه برای شی t، میتوانیم از آن استفاده کنیم. اگر بعد از t عملگر دسترسی به اجزاء (.) را تایپ کنید خواهید دید که در لیستی که خود vb باز میکند، فیلد خصوصی test وجود ندارد.



ولی به فیلد TedadNan دسترسی کامل داریم، میتوانیم مقدار آن را بخوانیم، یا تغییر بدهیم. برای مثال :

```
t.TedadNan = t.TedadNan * 5 + 2
```

اگرچه با استفاده از یک فیلد می توانیم به راحتی اطلاعات مورد نیازمان را ذخیره کنیم و تغییر بدهیم، اما همین راحتی باعث دردسرهایی هم می شود. همیشه باید فکر کنید که کاربر کلاس (کسی که از کلاسی که شما می نویسید در برنامه خودش استفاده می کند) کسی بجز خود شما است. پس اطلاعات چندانی درباره اینکه چه چیزهایی برای یک کلاس غیر مجاز است ندارد (در عمل برنامه نویس کلاس هم بعد از مدتی فراموش میکند چه چیز مجاز و چه چیز غیر مجاز است). به مثال TedadNan برگردیم، آیا برای فیلد تعداد نان، مقدار غیر مجازی هم وجود دارد؟ پاسخ مثبت است. همه مقادیر کوچکتر از صفر برای ما غیر مجازند. ولی متاسفانه فیلدها به ما این امکان را نمی دهند که داده هایی را که کاربر وارد می کند، محدود کنیم. ما هیچ کنترلی روی کاری که کاربر کلاس با فیلدها می کند نداریم، همین نداشتن کنترل روی داده ها ما را به سمت استفاده از خاصیتها (properties) می کشاند.

خاصیت Property :

ساختار خلاصه شده یک خاصیت (property) بصورت زیر است :

```
[Public|Private] Property PropertyName() As TypeName
    Get

    End Get
    Set(ByVal Value As TypeName)

    End Set
End Property
```

کلمه کلیدی Property به ویژوال بیسیک می گوید که ما قصد داریم یک خاصیت برای کلاسمان تعریف کنیم. مفهوم کلمات کلیدی Public و Private نیز همان است که پیش از این آموختیم. PropertyName نام خاصیت ما است. همانطور که می بینید خاصیت ما به دو بخش Get و Set تقسیم شده است، بخش Get وقتی اجرا می شود که کاربر کلاس می خواهد مقدار خاصیت را بخواند و بخش set وقتی اجرا می شود که کاربر کلاس بخواهد به خاصیت مقدار بدهد، در این صورت مقداری که کاربر می خواهد در property جایگزین کند در متغیر value قرار می گیرد. برای مثال می خواهیم یک خاصیت برای تعداد نان ایجاد کنیم. آنچه تا بحال در کلاس نوشته اید پاک کنید. نام خاصیت ما TedadNan است و نوع داده آن Integer . پس ساختار خاصیت به شکل زیر تغییر می کند :

```
Public Property TedadNan() As Integer
    Get

    End Get
    Set(ByVal Value As Integer)

    End Set
End Property
```

خاصیت ما به خوبی تعریف شده است و می توانیم در کد فرم برنامه از آن استفاده کنیم.

```
Private Sub btnTest_Click(…) Handles btnTest.Click
    Dim t As New bastani
    t.TedadNan = 10
    Dim tmp As Integer = t.TedadNan
End Sub
```

وقتی می نویسیم t.TedadNan=10 بخش Set از پروپرتی ما فراخوانی می شود و Value مقدار 10 میگیرد. اما وقتی می خواهیم مقدار t.TedadNan را بخوانیم، بخش Get از پروپرتی ما فراخوانی می شود و مقدار خاصیت را برمی گرداند. انتظار داریم این مقدار برابر 10 باشد؛ ولی صفر است! دلیلش این است که در

حقیقت پروپرتی به تنهایی هیچ مقداری ذخیره نمی کند. مقدار خاصیت باید در بخش set در یک شی (متغیر) ذخیره شود و در بخش get از آن شی (متغیر) خوانده و برگردانده شود.
برای برگرداندن یک مقدار، از دستور return استفاده می کنیم :

```
Return SomeValue
```

خاصیتی که نوشته بودیم، بشکل زیر تصحیح می کنیم :

```
Private mTedadNan As Integer
Public Property TedadNan() As Integer
    Get
        Return mTedadNan
    End Get
    Set(ByVal Value As Integer)
        mTedadNan = Value
    End Set
End Property
```

در اینجا متغیری با نام mTedadNan تعریف کرده ایم که تعداد نانها را نگهداری کند. (حرف m اول اسم این متغیر مخفف Member است.) در بخش Get با استفاده از دستور return مقدار متغیر mTedadNan را برمی گردانیم و در بخش Set مقدار Value را در متغیر mTedadNan جایگزین می کنیم. اگر بار دیگر به btnTest_Click سر بزنید خواهید دید که این بار عدد 10 در آن ذخیره می شود :

```
Private Sub btnTest_Click(…) Handles btnTest.Click
    Dim t As New bastani
    t.TedadNan = 10
    Dim tmp As Integer = t.TedadNan
    Label1.Text = tmp.ToString
End Sub
```

این بار وقتی اجرای برنامه به خط t.TedadNan = 10 می رسد، بخش set اجرا می شود و مقدار Value که برابر 10 است را در متغیر mTedadNan ذخیره می کند. وقتی هم برنامه می خواهد مقدار t.TedadNan را بخواند، بخش Get فراخوانی شده و دستور Return mTedadNan را اجرا می کند، این دستور مقدار 10 را برمی گرداند، بنابراین مقدار t.TedadNan در زیربرنامه ای که آن را صدا کرده بود (btnTest_Click) برابر با 10 می شود.
یک توضیح اضافه هم در مورد ToString بدهیم، این متد که در بسیاری از اشیاء وجود دارد شی را بصورت صریح به داده رشته ای تبدیل می کند.
گرچه ما موفق شدیم یک خاصیت برای کلاس bastani تعریف کنیم اما هنوزهم کاربر کلاس می تواند هر تغییری در مقدار متغیر ما بدهد، برای مثال میتواند مقدار 10- را به خاصیت بدهد. وقتی با فیلد کار می کردیم نمی توانستیم جلوی این کار را بگیریم، اما در خاصیتها (property) دست ما باز است و می توانیم با یک شرط ساده جلوی تغییرات غیر مجاز را بگیریم:

```
Public Property TedadNan() As Integer
    ...
    Set(ByVal Value As Integer)
        If Value >= 0 Then mTedadNan = Value
    End Set
End Property
```

هنوز هم کاربر میتواند از عبارتی مانند

```
Private Sub btnTest_Click(…) Handles btnTest.Click
    Dim t As New bastani
    t.TedadNan = -10
    ...
End Sub
```

استفاده کند، اما به دلیل اینکه در بخش set این خواصیت شرط بر قرار نمی شود، هیچ تغییری در مقدار متغیر ما ایجاد نمی شود. (آزمایش کنید)
اکنون میتوانیم سایر خاصیتهای این کلاس را نیز تعریف کنیم. در اینجا نام آنها آمده است و تعریف کامل آنها در CD ضمیمه کتاب وجود دارد.

نام	توضیح
MeghdarBastani	مقدار بستنی موجود : برحسب گرم
TedadLivan	تعداد لیوان موجود
TedadGhashogh	تعداد قاشق موجود

MeghdarPool مقدار پول موجود : بر حسب ریال

متدها

تا اینجا گه گاهی درباره زیر برنامه ها چیزهایی گفته ایم و با متدها نیز کمی کار کرده ایم. اما تاکنون خودمان یک متد ایجاد نکرده ایم، ساختار ساده شده یک متد (یک زیربرنامه) بصورت زیر است :

```
[Public | Private] Sub MethodName ()
    Statements
End Sub
```

متد، توانایی انجام کار یک شی است. همانطور که پیش از این گفته بودیم، هر شی یک نوع داده خود شمول است که متدها و خاصیت هایی که برای کار کردن با داده هایش لازم دارد درون خودش قرار دارد. کار با متدها را با متد ForooshNan آغاز می کنیم. این متد قرار است پس از فراخوانی شدن، یکی از تعداد نانها کم کند :

```
Public Sub ForooshNan()
    TedadNan = TedadNan - 1
End Sub
```

اگر در دکمه btnTest کد زیر را بنویسیم :

```
Private Sub btnTest_Click(…) Handles btnTest.Click
    Dim t As New bastani
    t.TedadNan = 10
    t.ForooshNan()
    Label1.Text = t.TedadNan.ToString
End Sub
```

خواهیم دید که Label1.Text برابر 9 می شود. وقتی اجرای برنامه با خط t.FrooshNan() می رسد به متد ForooshNan از شی t مراجعه می کند و دستوراتی که در آن نوشته اجرا می کند. در این متد ما مقدار TedadNan را یکی کم کرده ایم. احتمالا برای خواننده تیزهوش این سوال پیش می آید که چطور از یک پروپرتی استفاده کرده ایم بدون اینکه قبل از آن نام شی صاحب پروپرتی را بنویسیم؟ پاسخ این است که از آنجا که متد ForooshNan برای شی t صدا شده است و TedadNan هم یک پروپرتی از شی t است، میتوانیم بدون ذکر کردن نام شی از پروپرتی استفاده کنیم. به عبارت دیگر وقتی قبل از اسم خاصیت، نام شی نمی آید ویژوال بیسیک "همین شی" را صاحب آن پروپرتی می داند.

اجازه بدهید نگاه دقیق تری به کدی که در متد ForooshNan نوشتیم بیاندازیم. این کد یک بار بخش Get از پروپرتی TedadNan را صدا می کند و سپس از مقدار برگشتی آن یکی کم می کند و با صدا کردن بخش Set از این پروپرتی، مقدار آن را با مقدار جدید برابر می کند. آیا می توانیم کمی این متد را بهینه کنیم؟ پاسخ مثبت است. کد زیر کمی (بسیار بسیار کم) از کد قبلی سریعتر است :

```
Public Sub ForooshNan()
    TedadNan = mTedadNan - 1
End Sub
```

در اینجا برای گرفتن مقدار پروپرتی بجای استفاده از خود پروپرتی، بطور مستقیم از متغیری که پروپرتی مقدارش را در آن ذخیره می کند استفاده کرده ایم. اگرچه کاربر کلاس نمی تواند به متغیر mTedadNan دسترسی داشته باشد (چون private است) اما متدهای خود کلاس (به دلیل عضویت در کلاس) می توانند به این متغیر دسترسی داشته باشند.

اگرچه استفاده از متغیرهای عضو کمی سرعت را بیشتر می کند، ولی معمولا بهتر است از خود پروپرتیها استفاده کنید. مخصوصا در مورد مقدار دهی به پروپرتیها. در اینجا اگر سمت چپ مساوی هم از mTedadNan استفاده کنید، ممکن است تعداد نان کمتر از صفر بشود. در حالی که استفاده کردن از خاصیت TedadNan جلوی این اتفاق را می گیرد.

در نهایت این به شما بستگی دارد که تشخیص بدهید از متغیر عضو استفاده کنید یا از خاصیتها. ما ترجیح می دهیم در اینجا از TedadNan = TedadNan – 1 یا معادل کوتاه تر آن TedadNan -= 1 استفاده کنیم.

```
Public Sub ForooshNan()
    TedadNan -= 1
End Sub
```

متدهای دیگر نیز تا حدود زیادی شبیه همین متد هستند. متد فروش 50 گرم بستنی (ForooshBastani) در زیر آمده است.

```
Public Sub ForooshBastani()
    MeghdarBastani -= 50
End Sub
```

متدهای فروش قاشق و فروش لیوان نیز مانند این دو متد هستند، اما پیاده سازی متد دریافت پول از مشتری به فصلهای بعد موکول می شود.

```
Public Sub ForooshLivan()
    TedadLivan -= 1
End Sub

Public Sub ForooshGhashogh()
    TedadGhashogh -= 1
End Sub
```

رویدادها

ساختار ساده شده اعلان یک رویداد بصورت زیر است :

```
[ Public | Private ] Event EventName ()
```

اعلان رویداد به همین یک خط محدود می شود و شامل هیچ بدنه ای نمی شود (برخلاف آنچه در مورد تعریف متدها داشتیم). خط اعلان فقط به ویژوال بیسیک می گوید که ما یک رویداد به نام EventName داریم و نمی گوید که این رویداد چه موقع اتفاق می افتد. برای مثال اعلان رویداد "تمام شدن نان" در زیر آمده است:

```
Public Event NanTamamShod()
```

این خط، رویداد NanTamamShod را به برنامه ما اضافه می کند. ولی هنوز مشخص نیست که چه زمانی این رویداد فراخوانی می شود. ما می دانیم که نان وقتی تمام می شود که مقدار آن برابر صفر باشد. از کجا می توانیم بفهمیم که مقدار نان برابر صفر شده است؟ مقدار نان وقتی صفر می شود که بخش set از خاصیت TedadNan مقدار 0 را به عنوان Value دریافت کند. پس می توانیم در این بخش، رویداد را فعال کنیم.

```
Public Property TedadNan() As Integer
    Get
        Return mTedadNan
    End Get
    Set(ByVal Value As Integer)
        If Value >= 0 Then mTedadNan = Value
        If mTedadNan = 0 Then RaiseEvent NanTamamShod()
    End Set
End Property
```

دستور RaiseEvent رویدادی را که نامش پس از این دستور آمده است را فعال می کند.

تا اینجا فقط گفته ایم که این رویداد اتفاق افتاده. اینکه با اتفاق افتادن این رویداد، چه کارهایی انجام میشود، به کاربر کلاس بستگی دارد؛ نه برنامه نویس کلاس.

برای روشن شدن طرز کار یک رویداد، مثالی از کد سمت کاربر کلاس می زنیم. فرض کنید کاربر کلاس می خواهد برنامه ای بنویسد که یک دکمه برای کم کردن از تعداد بستنی ها داشته باشد و وقتی بستنی ها تمام شد، روی صفحه پیغامی مبنی بر تمام شدن بستنی ها نمایش داده شود.

برای اینکه بتوانیم از رویدادهای یک شی استفاده کنیم باید کلمه کلیدی WithEvents را قبل از نام آن شی بنویسیم. درضمن وقتی از کلمه کلیدی WithEvents استفاده می کنیم نمی توانیم شی را در یک زیربرنامه تعریف کنیم و باید آن را بیرون زیربرنامه ها تعریف کنیم.

```
Public Class Form1
    Inherits System.Windows.Forms.Form

    Private WithEvents t As New bastani

    Private Sub btnTest_Click(…) Handles btnTest.Click
        t.ForooshNan()
    End Sub

    Private Sub Form1_Load(…) Handles MyBase.Load
        t.TedadNan = 5
    End Sub
End Class
```

در اینجا رویداد Load از شی Form استفاده کرده ایم، این رویداد وقتی Form در حافظه قرار می گیرد و قبل از اینکه نمایش داده شود اتفاق می افتد. پس قبل از اینکه فرم نمایش داده شود، مقدار t.TedadNan برابر

5 میشود و پس از آن هربار که دکمه btnTest کلیک شود، یکی از نانها به فروش می رود. حال ببینیم چطور می توانیم بفهمیم نانها تمام شده اند.
از ComboBox مربوط به Class Name شی t را انتخاب کنید و از ComboBox مربوط به Method Name رویداد NanTamamShod را انتخاب کنید.

و در بدنه آن کد زیر را بنویسد :

```
Private Sub t_NanTamamShod() Handles t.NanTamamShod
    Label1.Text = "شد تمام نان"
End Sub
```

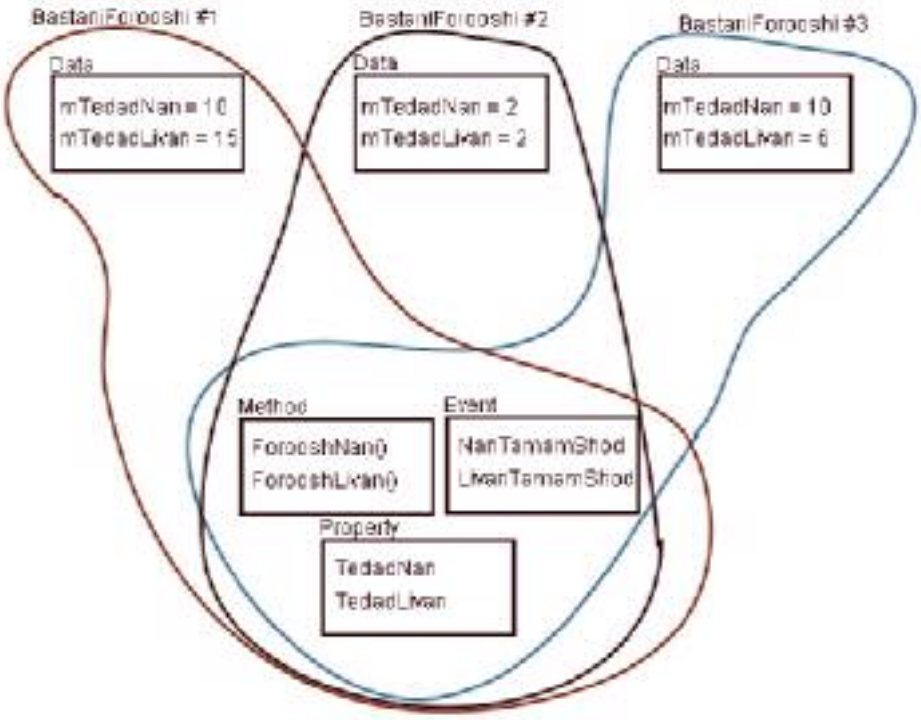
برنامه سمت کاربر کلاس ما تمام شده است. دقیقا همان کاری را انجام می دهد که می خواستیم. اما دقیقاً چه اتفاقی افتاده؟ آیا به شباهت بین متدهایی که در کلاسمان تعریف کردیم و کدی که برای کنترل کردن یک رویداد توسط ویژوال بیسیک نوشته می شود دقت کرده اید؟ اجازه بدهید نگاه دقیق تری به آنچه روی می دهد بیاندازیم.
وقتی کاربر روی دکمه کلیک می کند، متد فروش نان از شی t فراخوانی می شود. این متد به نوبه خود با استفاده از پروپرتی TedadNan یکی از تعداد نانها کم می کند. هر بار که این اتفاق می افتد پروپرتی TedadNan بررسی می کند ببیند آیا تعداد نانها صفر شده است؟ یا خیر. اگر صفر شده بود رویداد NanTamamShod را فعال می کند. شی فرم ما مطلع می شود که رویدادی به اسم NanTamamShod اتفاق افتاده است، به متدهای خودش مراجعه می کند تا ببیند متدی پیدا می کند که وظیفه پاسخگویی به این رویداد را به عهده داشته باید؟ در نتیجه متد t_NanTamamShod را پیدا می کند که در انتهای آن Handles t.NanTamamShod نوشته شده است. پس این متد را فراخوانی می کند. فراخوانی شدن این متد باعث می شود که متن Label1 تغییر کند و ...
پس فهمیدیم که t_NanTamamShod یک متد از کلاس Form1 است که مسوول پاسخگویی به رویداد NanTamamShod از شی t است. به همین صورت، متد btnTest_Click نیز متدی از کلاس Form1 است که مسوول پاسخگویی به رویداد click از شی btnTest است.

نام رویدادهای دیگر کلاس bastani در ادامه آمده است، کد مربوط به آنها را میتوانید در CD ضمیمه پیدا کنید.

نام رویداد	توضیح
BastaniTamamShod	تمام شدن بستنی (بستنی موجود کمتر از 50 گرم است.)
LivanTamamShod	تمام شدن لیوان (لیوان موجود برابر صفر است.)
GhashoghTamamShod	تمام شدن قاشق (قاشق موجود برابر صفر است.)

یک شی واقعا چیست؟

اجازه بدهید ابتدا کمی درباره مفهوم کلاس صحبت کنیم. کلاس شبیه یک دستور پخت غذا است. وقتی ما یک کلاس تعریف می کنیم مانند این است که دستور پخت یک غذا را روی کاغذ نوشته باشیم، هنوز غذایی برای خوردن وجود ندارد. دستور پخت غذا (کلاس) فقط می گوید که چگونه این غذا را بپذیم و اینکه این غذا چه مواد اولیه ای(داده ها) لازم دارد. اما تا وقتی که ما غذایی نپزیم (شی ایجاد نکنیم) چیزی برای خوردن وجود ندارد.
در حقیقت کلاس، دستور ایجاد شی است. کلاس می گوید که برای ایجاد یک شی (برای مثال بستنی فروشی) به چه داده های پایه ای احتیاج داریم و چه متدها و رویدادها و خواصی باید برای آن آماده شود.
شی، نمونه ساخته شده کلاس است. هر نمونه، داده های پایه خودش را دارد، اما خواص و رویدادها و متدها بین همه اشیاء مشترک هستند. با تشبیه به دستور آشپزی می توانیم بگوییم هر بار که از روی دستور آشپزی غذا درست می کنیم مواد اولیه تغییر می کنند، اما کارهای که هنگام پختن انجام می دهیم (متدها) و رویدادهای غذا مشترک هستند.



تمرین :

1. متدهای مربوط به دریافت از آقای محمدی را اضافه کنید. (دریافت بستنی، دریافت نان و ...)
2. آقای محمدی می خواهد وقتی موجودی فروشگاه به 1000000 ریال رسید به فروشگاه برود و پول را بگیرد، یک رویداد مناسب برای آن طراحی کنید.

پروژه برنامه نویسی

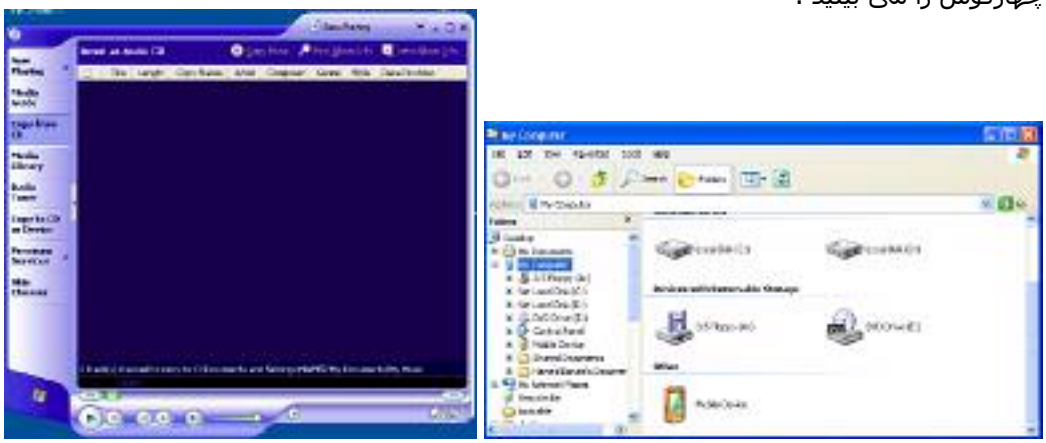
کتابخانه : می خواهیم برنامه ای بنویسیم که مشخصات کتابهای یک کتابخانه را نگهداری کند، بتواند کتابهایی به کتابخانه اضافه کند و کتابهایی نیز از آن حذف کند، اطلاعات آنها را ویرایش کند و بین آنها جستجو کند. در مرحله اول (برای این فصل) داده ها، رویدادها و متدهای شی کتاب را برسی و سپس کلاس کتاب را ایجاد کنید. ما پیشنهاد میکنیم این پروژه را نیز همپای برنامه بستنی فروشی انجام بدهید.

آشنایی با GUI[6] یا رابط گرافیکی کاربر

یکی از برتری های یک سیستم عامل گرافیکی مانند ویندوز نسبت به سیستم عاملهای متنی (مانند DOS) ، وجود رابط کاربر گرافیکی قدرتمند است. ویندوز یکی از مشهور ترین سیستم عاملهای گرافیکی است ؛ امروزه بیش از 90 درصد کامپیوترهای خانگی جهان از این سیستم عامل بهره می برند.
ما به عنوان برنامه نویس ویندوز باید با بعضی از اصول طراحی رابط کاربر برای برنامه های تحت ویندوز آشنایی داشته باشیم. علاوه بر این باید بتوانیم رابط کاربری طراحی کنیم که کاربران در مواجهه با آن دچار سردگمی نشوند . وقتی برنامه ای دارای رابط کاربری (user interface) مناسب باشد کاربران خیلی راحت از آن استفاده می کنند اما اگر شما بهترین الگوریتم ها را به کار برده باشید ولی طراحی ظاهر برنامه مناسب نباشد ، کاربر به راحتی نمی تواند از آن استفاده کند و بعد از مدتی دنبال برنامه دیگری می رود . شاید برای شما هم پیش آمده باشد که بین دو برنامه که یک کار انجام می دهند، برنامه ای را که ظاهر بهتری دارد به برنامه سریعتر ترجیح بدهید. وقتی از رابط کاربر مناسب صحبت می کنیم، منظور ما فقط زیبایی ظاهر برنامه نیست. سهولت استفاده از برنامه نیز بسیار مهم است. به مبحث خوب طراحی شدن ظاهر کاربر پسند بودنِ یک برنامه User Friendly بودنِ آن محصول می گویند .

خاصیتهای فرم

تاثیر گذارترین بخش برنامه پنجره اصلی یا همان Windows Formِ برنامه است . شمای ظاهری آن ، راست به چپ بودن (برای زبانهایی مثل فارسی) و دیگر خصوصیات، در طرز برخورد کاربر با برنامه تاثیر بسزایی دارد . ما در این کتاب درباره چگونگی تغییر شکل کلیِ فرم از چهار ضلعی به اشکال دیگر صحبت نمی کنیم ولی این را هم به عنوان یک امکان در نظر داشته باشید . در شکل زیر تفاوت دو فرم معمولی و غیر چهارگوش را می بینید .



شما قطعاً انواع فرمهای ویندوز را در برنامه های مختلف دیده اید ، بعضی می توانند تمام صفحه را بگیرند ، بعضی در task bar نمایش داده نمی شوند و بعضی دیگر روی همه فرمهای برنامه دیگرِ در حال اجرا قرار می گیرند(Always on top) . برای تعیین این خواص میتوانید از قسمت Property های فرم استفاده کنید .

[6] Graphical User Interface

کاربردی ترین خواص یک فرم را به همراه توضیحات ِ آن در جدول زیر می بینید :

نام	توضیحات
AutoScroll	نوع این خاصیت Boolean است و مشخص کننده این است که اگر اشیاء روی فرم، از محدوده فرم خارج شدند بطور خودکار برای فرم scrollbar ایجاد شود یا خیر.
BackColor و BackgroundImage	رنگ و عکس زمینه فرم را مشخص می کند
FormBorderStyle	مشخص کننده این است که آیا فرم در حالت اجرا قابلیت تغییر اندازه خواهد داشت یا خیر . این خاصیت دارای گزینه های مختلفی مانند Sizable یا FixedSingle و... است .
GridSize	وقتی کنترلی بر روی یک فرم قرار می دهید آن را هر جایی آن نمی توانید جاسازی کنید و فقط می توانید آن را در Stepهای خاصی حرکت دهید ، فاصله آن نقاط را با استفاده از GridSize می توانید تغییر دهید . نقاطی که هنگام طراحی شکل ظاهری برنامه در VS.NET مشاهده می کنید همان نقاط Grid هستند که با فاصله آنها همان مقدار GridSize است و چهار گوشه کنترل روی این نقاط قرار می گیرد .
Icon	شکلکی در بالا و سمت چپ همه برنامه های ویندوز وجود دارد و در TaskBar نیز نمایش داده میشود.
MaximizeBox	این خاصیت از نوع Boolean است . در تمامی برنامه های ویندوز در بالا و سمت راست سه دکمه وجود دارد ، یکی برای بستن برنامه ، دیگری برای اینکه برنامه تمام صفحه را بگیرد و آخری برای این است که برنامه در Taskbar جای بگیرد . اگر MaximizeBox برابر True باشد این دکمه در بالا

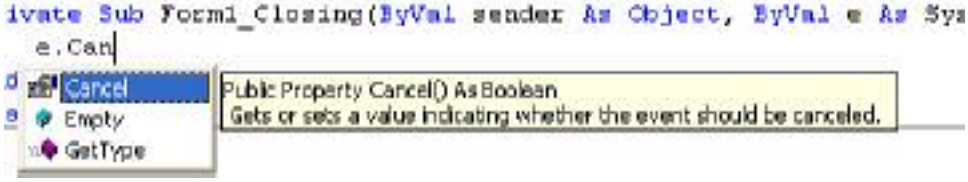
	سمت راست برنامه ما نمایش داده می شود .
MinimizeBox	این خاصیت همانند MaximizeBox است ولی برای دکمه سوم .
RightToLeft	با Yes قرار دادن این خاصیت می توانیم از خواص راست به چپ بودن فرم ها استفاده کنیم که برای زبان فارسی مناسب است .
ShowInTaskbar	نوع این خاصیت Boolean است و True بودن آن باعث می شود برنامه هنگام اجرا در قسمت Taskbar ِ ویندوز نمایش داده شود .
StartPosition	با استفاده از این خاصیت می توان محل قرارگیری برنامه در صفحه نمایشگر را برای زمان اجرا تعیین کرد.
Text	مقدار این خاصیت در Titlebar ِ فرم و در Taskbar نمایش داده می شود .
TopMost	نوع این خاصیت Boolean است و True بودن آن باعث می شود این فرم هنگام اجرا همیشه بر روی تمامی برنامه های در حال اجرا قرار گیرد حتی اگر خود این فرم در برنامه فعال نباشد . بعضی از برنامه ها با آن Always On Top میگویند.
WindowState	این خاصیت سه مقدار ِ Normal ، Maximized و Minimized می تواند بگیرد . حالت Normal وقتی است که همین طرحی در قسمت طراحی شکل ظاهری کشیده اید هنگام اجرا نمایش داده شود. اگر حالت Maximized انتخاب شود برنامه هنگام اجرا تمام صفحه را خواهد گرفت و اگر Minimized انتخاب شود برنامه هنگام اجرا به Taskbar خواهد رفت .

فرم ها به عنوان یک Container مورد استفاده قرار می گیرند یعنی می توان دیگر کنترل ها را بر روی آنها قرار داد . لیست کنترل های موجود (همان طور که در فصل های قبلی ذکر شد) در Toolbox (سمت چپ IDE) قرار دارد . برای استفاده از هر کدام از کنترل ها کافی است آن را با drag & drop روی فرم قرار دهید.

رویدادهای فرم

هنگامی که فرم برای اولین بار می خواهد load شود رویداد onload ِ آن اتفاق می افتد که معمولاً قسمت زیادی از کدهای لازم برای اینکه از پیش اطلاعاتی در فرم نمایش داده شود و مقدار دهی های اولیه بعضی متغیرها و اشیاء را در این سابروتین می نویسند . فرم رویداد دیگری دارد که بسیار شبیه load است ولی می تواند بیش از یکبار نیز اتفاق بیافتد . آن رویداد Activate است ، هر موقع که فرم از حالت فعال خارج شده و دوباره به حالت فعال باز گردد این رویداد اتفاق می افتد .

شاید تا به حال برنامه هایی را دیده باشید که هنگامی که بر روی دکمه X در بالا سمت راست فرم آن کلیک می کنید به جای بسته شدن یا اصلاً عملی اتفاق نمی افتد یا برنامه در قسمت Notify Area در کنار ساعت ِ ویندوز جای می گیرد . در ویژوال بیسیک دات نت چنین کاری را می توان در رویداد Closing انجام داد . این رویداد قبل از بسته شدن فرم اتفاق می افتد . بعضی از رویدادها در دات نت دارای پارامترهای خاص خود هستند که آنها را در خط تعریف رویداد می توانید با شی ِ e ببینید . برای مثال شی ِ e در رویداد ِ Closing دارای خاصیتی به نام Cancel است. اگر مقدار Cancel را برابر Trueقرار دهید فرم شما با استفاده از دکمه X بسته نخواهد شد . هر بار که کاربر سعی کند فرم شما را ببندد رویداد closing فراخوانی می شود و e.Cancel=True جلوی بسته شدن فرم را میگیرد. برای بستن برنامه از دکمه Stop یا منوی Debug\Stop Debugging استفاده کنید.



راه دیگر برای یستن برنامه در این حالت این است که از دستور End در دکمه ای با عنوان خروج استفاده کنید . دستور End برنامه شما را Terminate می کند، به همین دلیل Closing اتفاق نخواهد افتاد.

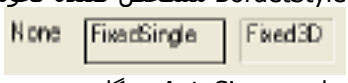
رویدادهای Resize ، Click ، DblClick بین فرم و بعضی از کنترلهای دیگر مشترک هستند . Resize زمانی اتفاق می افتد که اندازه کنترل تغییر کند . رویدادهای Click و dblClick هم بترتیب هنگامی اتفاق می افتد که روی کنترل مورد نظر کلیک یا دبل کلیک شود .

خاصیتهای کنترلها

کنترلها دارای دو دسته خواص هستند ، خواص عمومی و خواص خصوصیِ هر کنترل . برای مثال همه کنترل ها دارای خاصیت Name هستند که نام آن کنترل را مشخص می کند. یا اکثر کنترل ها دارای خاصیت font یا Text هستند. برخی کنترل ها خواص مخصوص به خود دارند مانند Columns در ListView. در ادامه بعضی از کنترل های پر کاربر را معرفی می کنیم .

Label

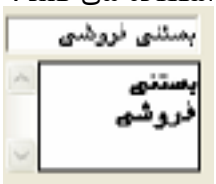
یکی از کاربردی ترین کنترلهایی که در برنامه های ویژوال استفاده می شود Label یا برچسب است . این کنترل برای نمایش متنهایی با طولِ کم به کار می رود . برای مثال در فصل های قبل برای HelloWorld از آن استفاده کردیم.
پر استفاده ترین خواص Label عبارتند از Text ،AutoSize، ForeColor ،Font و BorderStyle .
خاصیت Text متنی را که می خواهید این برچسب نشان دهد را مشخص می کند .
با استفاده از ForeColore رنگ متن را تعیین می کنید .
Font ، نام ، اندازه و دیگر مشخصاتِ فونتِ متن را معین می کند .
BordetStyle مشخص کننده نحوه نمایش دورِ label است ، تفاوت سه مقدار آن را در شکل زیر می بینید .



خاصیت AutoSize هنگامی مفید است که متن درون برچسب متغیر باشد ، در این حالت اگر AutoSize برابر True باشد طولِ Label متناسب با طول متن تغییر می کند .

TextBox

یکی دیگر از کنترل های پرکاربرد، TextBox یا جعبه متن است. از این کنترل برای دریافت یک رشته از کاربر استفاده می کنند .



TextBox به غیر از خواص عمومی این دسته از کنترل ها مانند Name ، Font، BorderStyle ، ForeColor و ... دارای خواص دیگری مانند MultiLine ، ScrollBar، HideSelection ، PasswordChar و MaxLenght است .
در شکل بالا دو نوع TextBox را مشاهده می کنید ، اولی تمامی متنِ خود را در یک خط جا داده است و دومی امکان آمدن به خط های بعدی را دارد. در TextBoxِ دوم مقدار خاصیت MultiLine برابرTrue است . همچنین در TextBox دوم مقدار ScrollBar برابر Vertical است.
اصطلاحاً به حالتی که در فرم کنترلی به عنوان کنترل فعال باشد می گویند قبلاً به آن کنترل focus شده است . مثلاً وقتی روی یک دکمه کلیک می کنید قبل از رویداد کلیک رویداد focusِ آن کنترل رخ می دهد . یا هنگامی که در یک TextBox در حال نوشتن متن هستید ، آن جعبه متن در حالت focus شده قرار دارد .
وقتی در TextBox متنی وجود داشته باشد و قسمتی از آن را انتخاب کنید آن قسمت رنگش تغییر می کند . در این حالت اگر textboxِ مورد نظر از حالت فعال خارج شود برای رنگِ قسمت انتخاب شده دو حالت وجود دارد ، یا رنگش به همان رنگِ حالت انتخاب شده می ماند یا اصلاً نشان داده نمی شود ، خاصیت

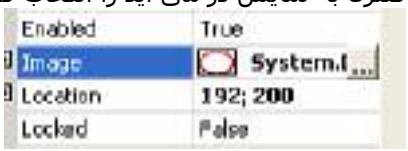
HideSelection اگر برابر Trueباشد در هنگامی که textbox از focus خارج شود باعث می شود که متن انتخاب شده رنگش دیده نشود .
خاصیت PasswordChar : شما حتماً بارها رمز ایمیل خود را در جعبه متن مربوطه وارد کردید ، اما همیشه به جای اینکه کاراکترهای رمز شما نمایش داده شود تعدادی * نمایش داده می شده. برای اینکه جعبه متنی که می سازید حکم اینگونه جعبه متنها را داشته باشد می توانید از خاصیت PasswordChar استفاده کنید. مثلاً مقدار این خاصیت را برابر A یا هر کاراکتر دیگری مانند *قرار دهید ، به ازای هر کاراکتری که هنگام اجرا در جعبه متن تایپ کنید حرف A نمایش داده خواهد شد .
گاهی اوقات لازم است طول متنی را که کاربر می تواند در جعبه متن تایپ کند محدود کنید ، برای این کار از خاصیت MaxLength باید استفاده شود .

Button

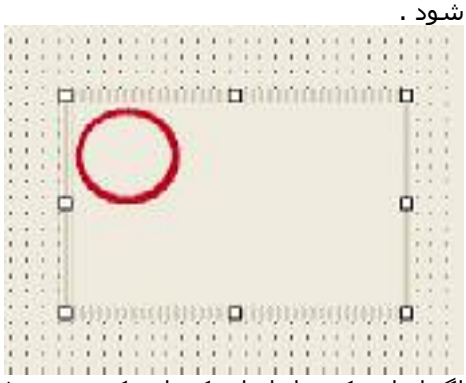
یکی دیگر از کنترل های پر مصرف در تمامی برنامه های ویژوال دکمه یا Button است . معمولاً کاربر برای اینکه کار مورد نظرش را به برنامه بگوید از دکمه های موجود در فرم برنامه استفاده می کند .
Button دارای خواص متعددی است مانند Text که همان متنی است که روی آن نمایش داده می شود.
مهمترین رویدادی که Button دارد رویداد کلیک است که اکثر کارهای عملیاتی آن را در این رویداد می نویسند .

PictureBox

از دیگر کنترل های پرکاربرد در برنامه های ویژوال PictureBox است . از این کنترل برای نمایش تصاویر گرافیکی استفاده می شود. در این کنترل بوسیله خاصیتی به نام Image می توان عکسی که در این کنترل به نمایش در می آید را انتخاب کرد.



وقتی عکس مورد نظر را انتخاب کردید یک نمونه کوچک آن در جلوی خاصیت Image در قسمت خواص نمایش داده می شود ، همچنین در خودِ کنترل picturebox نیز همان تصویر با اندازه واقعی نمایان می شود .



اگر ابعاد عکس از اندازه کنترلی که بر روی فرم قرار دارد کوچک تر باشد (مانند عکس بالا) ، تمام عکس دیده شده و دیگر قسمت های کنترلِ Picturebox خالی می ماند ، اما اگر اندازه PictureBox کوچک تر باشد فقط به اندازه خودِ کنترل از عکس نمایش داده می شود .

PictureBox دارای خاصیتی به نام SizeMode است. شرایطی که در بالا ذکر شد برای مقدارِ Normal در SizeMode است. مقادیر دیگری که خاصیت SizeMode می گیرد عبارتند از :
StretchImage

AutoSize
CenterImage

مقدار StretchImage باعث می شود که ابعاد عکس بدون توجه به تناسبِ خود عکس با ابعاد کنترلِ Picturebox هماهنگ شود .

در مواردی مانند شکل بالا این خاصیت کاملاً تناسب عکس اصلی را خراب می کند .

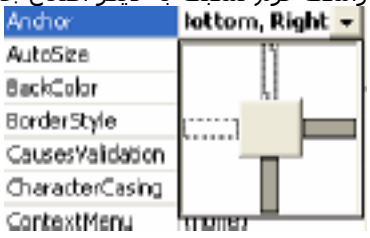
مقدار AutoSize اندازه PictureBox را برابر اندازه عکس مورد نظر می کند .

مقدار آخر ، CenterImage ، همیشه مرکز عکس را روی مرکز PictureBox قرار می دهد ، حتی اگر اندازه PictureBox کوچکتر از اندازه عکس باشد .

سایر خاصیت های این کنترل مشابه دیگر کنترلهاست .

خاصیت Anchor

فرض کنید فرم قابلیت تغییر اندازه داشته باشد ، یعنی با drag کردن یکی از اضلاع آن بتوانیم ابعاد آن را تغییر دهیم . اگر یک Button روی این فرم داشته باشیم و بخواهیم هنگامی که فرم تغییر اندازه می دهد اندازه آن دکمه هم تغییر کند چه کاری باید بکنیم ؟
در ویژوال بیسیک 6 می توانستیم با استفاده از رویداد resize از فرم این کار را تا حدود انجام دهیم ولی در مواردی performance_ مورد نظر را نداشت . در .NET. برای این کار خاصیتی به نام anchor پیش بینی شده است. در این خاصیت شما تعیین می کنید که هر کنترل نسبت به کدام ضلعِ فرم باید فاصله ثابتی داشته باشد . برای مثال در شکل زیر مقدار این خاصیت برابر پایین و راست است ، یعنی هر گاه ضلع پایین یا راست فرم نسبت به دیگر اضلاع جابجا شود کنترلِ مورد نظر ما نیز ابعادش تغییر می کند .



طراحی رابط کاربر برای بستنی فروشی

حال با استفاده از کنترلهای جدیدتری که خواهیم آموخت برنامه بستنی فروشی را با روش های مختلف کامل می کنیم . شاید رابط کاربرهای اولیه چندان هم مطلوب نباشند، اما در نهایت به رابط کاربر مطلوب خواهیم رسید.

RadioButton

کنترل جدیدی که در ابتدا از آن استفاده می کنیم RadioButton است . از RadioButton برای دادن امکان انتخاب بین چند موضوع به کاربر استفاده می شود . فرض کنید برای کاری سه حالت الف ، ب و ج وجود داشته باشد و کاربر شما باید فقط یکی از این سه حالت را انتخاب کند .

در هر لحظه کاربر فقط یکی از این 3 RadioButton را می تواند انتخاب کند و شما از روی انتخاب وی عمل مورد را انجام می دهید .

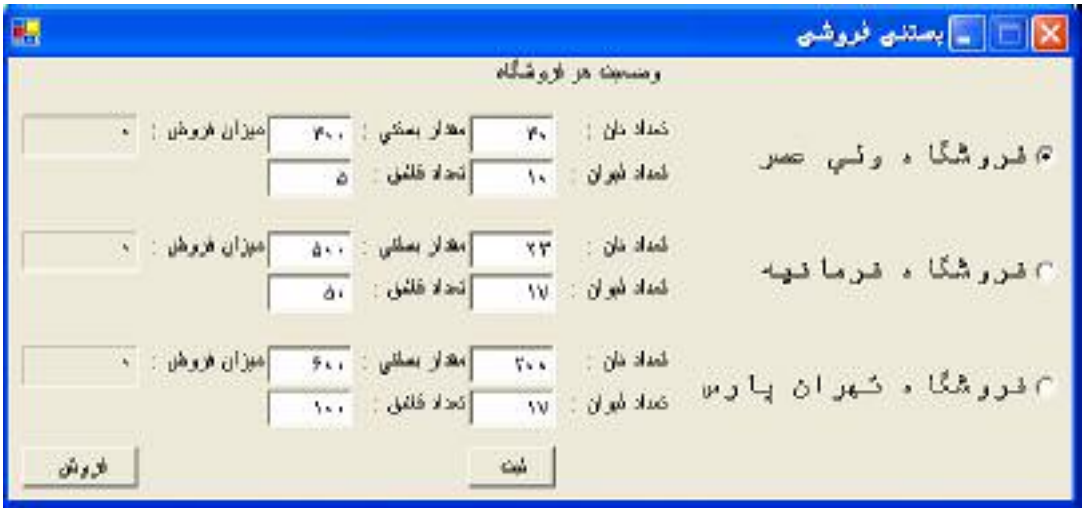
RadioButton دارای خواص مختلفی هستند که کاربردی ترین آنها را در جدول زیر مشاهده می کنید :

نام خاصیت	توضیحات و نحوه استفاده
Text	متنی که در Text قرار می گیرد در RadioButton نمایش داده می شود . `RadioButton1.Text = "الف"`

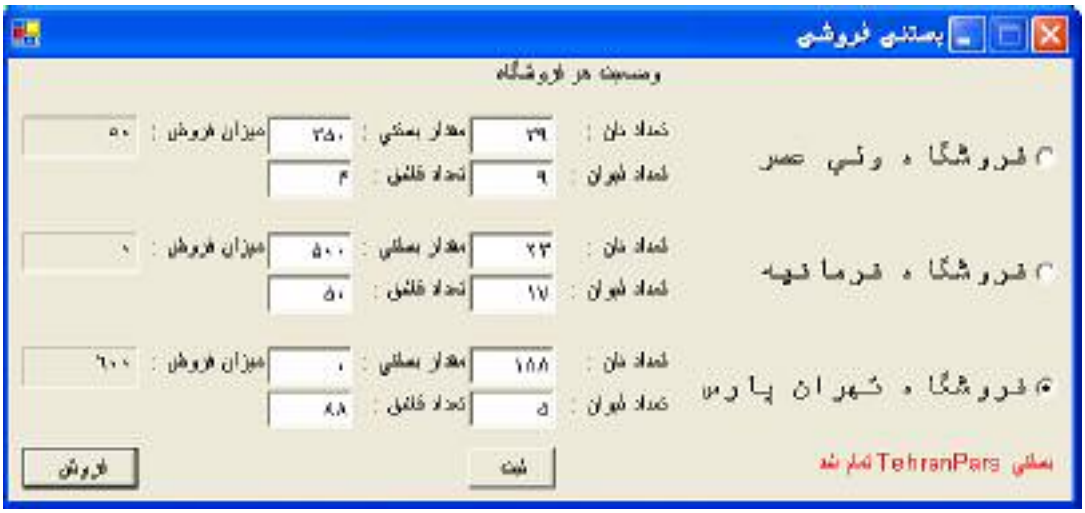
Checked	برای اینکه ببینید آیا یک RadioButton انتخاب شده است یا خیر از این خاصیت استفاده می کند . نوع این خاصیت Boolean است .
	`If RadioButton1.Checked = True Then`
	`'Do Something`
	`End If`

TextAlign	از این خاصیت برای انتخاب محل نمایش Text استفاده می شود . 

البته این کنترل نیز مانند اکثر دیگر کنترلها دارای خاصیت فونت ، RightToLeft و دیگر خواص عمومی است .
در برنامه بستنی فروش از 3 عدد RadioButton برای مشخص کردن 3 فروشگاه استفاده می کنیم ، همچنین از تعدادی TextBox و Label برای ایجاد فرمی مانند فرم زیراستفاده کردیم:



ما باید در این برنامه بتوانیم مقدار اولیه(موجودی) هر فروشگاه را ثبت کنیم؛ بنابراین از 4 عدد TextBox برای هر کدام از فروشگاه ها استفاده کردیم . بعد از پر کردن مقدار اولیه همه لوازم در همه فروشگاه ها بر روی کلید ثبت کلیک می کنیم تا مقادیر به object های ساخته شده برای هر کدام از فروشگاه ها انتقال یابد . کلید دیگری برای فروش ایجاد شده تا با کلیک بر روی آن از فروشگاه انتخاب شده یک بستنی به فروش رود . همچنین یک label به forecolor ِ قرمز برای نمایش پیغام های برنامه در پایین آن ایجاد می کنیم .



سورس این برنامه در سی دی ضمیمه وجود دارد .

برای اینکه برنامه ما به نحوی که در توضیح دادیم کار کند، باید کمی در کلاس ِ بستنی تغییرات اعمال کنیم .

ابتدا یک خاصیت جدید از نوع String با نام Title به کلاس بستنی اضافه می کنیم .

```
Public Property Title() As String
    Get
        Return mObjTitle
    End Get
    Set(ByVal Value As String)
        mObjTitle = Value
    End Set
```

```
    End Property
```

و در قسمت Member Variables این متغیر را تعریف می کنیم .

```
Private mObjTitle As String
```

همچنین تمامی رویدادهایی که در کلاس ایجاد کردیم را مانند زیر ویرایش می کنیم :

```
Public Event NanTamamShod(ByVal Sender As String)
Public Event BastaniTamamShod(ByVal Sender As String)
Public Event LivanTamamShod(ByVal Sender As String)
Public Event GhashoghTamamShod(ByVal Sender As String)
Public Event ZamanTasviyeHesab(ByVal Sender As String)
```

متغیری که این رویدادها بر می گرداند همان نام object ی است که از روی این کلاس ساخته خواهد شد . هر جایی که رویدادی را فراخوانی می کنیم به جای Sender از mObjName که حاوی نام است استفاده می کنیم :

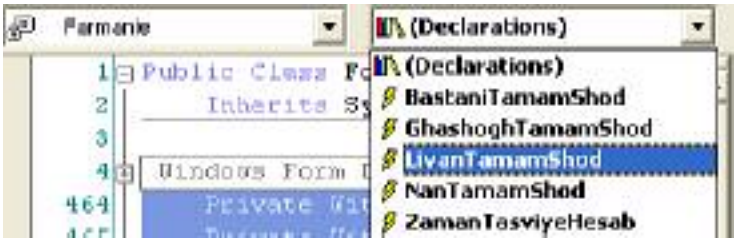
```
RaiseEvent BastaniTamamShod(mObjName)
RaiseEvent NanTamamShod(mObjName)
RaiseEvent GhashoghTamamShod(mObjName)
RaiseEvent LivanTamamShod(mObjName)
RaiseEvent ZamanTasviyeHesab(mObjName)
```

توجه : در ابزارهای ویژوال بیسیک شی Sender شیی است که رویداد برای آن اتفاق افتاده، اما در اینجا نام آن رویداد است. استفاده از این نام صرفاً برای کم کردن کدهایی است که در ادامه خواهیم نوشت .
حال به برنامه اصلی باز می گردیم. در قسمت کد، بعد از محدوده ای که خود ویژوال استودیو کدهایی را برای ساختن فرم نوشته است این خطوط را اضافه می کنیم .

```
Private WithEvents ValiAsr As New bastani
Private WithEvents Farmanie As New bastani
Private WithEvents TehranPars As New bastani

Dim CanSellValiAsr As Boolean = True
Dim CanSellFarmanie As Boolean = True
Dim CanSellTehranPars As Boolean = True
```

در 3 خط اول Object هایی با نام های معین برای فروشگاه های مختلف از روی کلاسِ بستنی ساختیم .



در 3 خط بعدی 3 متغیر از نوع Boolean تعریف کردیم تا مشخص کند که آیا یک فروشگاه هنوز امکان فروختن دارد یا خیر . مثلاً وقتی یک فروشگاه دیگر بستنی برای فروش نداشت کاربر هم نباید بتواند بستنی ی از آن فروشگاه بفروشد .

در سابروتین ِ load ِ فرم از این سه خط برای نام گذاری Objectهایمان استفاده می کنیم .

```
ValiAsr.Title = "ValiAsr"
Farmanie.Title = "Farmanie"
TehranPars.Title = "TehranPars"
```

برای صرفه جویی در کد به جای اینکه برای تمامی رویدادهای تمام Object ها کد بنویسیم برای تمام رویدادهای یک object کد می نویسیم و از آن برای رویدادهای مشابه دیگر Object ها استفاده می کنیم (مشابه کاری که در بازی حدس زدن عدد، برای دکمه های ماشین حسابی کردیم):

```
Private Sub ValiAsr_BastaniTamamShod(ByVal Sender As String) Handles_
ValiAsr.BastaniTamamShod,          Farmanie.BastaniTamamShod,          _
TehranPars.BastaniTamamShod
        lblAlarm.Text = "بستنی " & Sender & " شد تمام"
        MakeDisable(Sender)
End Sub
```

در این مثال به جای اینکه رویداد تمام شدن بستنی را برای تمام object ها بنویسیم یک بار آن را برای ValiAsr نوشتیم . فقط در ادامه ی عبارت Handles دیگر Object هایی که از همان نوع رویداد استفاده می کنند را ذکر کردیم . با این روش هر موقع رویداد تمام شدن بستنی در دیگر فروشگاه ها رخ دهد از همین زیربرنامه اجرا خواهد شد..
در این سابروتین دو کار انجام می شود ، ابتدا در label ی که برای پیغام ها ساخته بودیم پیغام ِ بستنی ِ فروشگاه .... تمام شد نوشته می شود . نام فروشگاه را از همان پارامتری که در تعریف رویداد در کلاس ساختیم می گیرد و به جمله بالا اضافه می کند . دومین عملیاتی که در این سابروتین انجام می شود علامت گذاری فروشگاه مورد نظر برای غیر قابل فروش بودن بستنی از آن است . این کار بوسیله سابروتین MakeDisable انجام می شود . پارامتری که این سابروتین می گیرد نام فروشگاه است .

```
    Private Sub MakeDisable(ByVal Sender As String)
        Select Case Sender
            Case "ValiAsr"
                CanSellValiAsr = False
            Case "Farmanie"
                CanSellFarmanie = False
            Case "TehranPars"
                CanSellTehranPars = False
        End Select
    End Sub
```

در کلید ِ "ثبت" فقط مقدار هر کدام از Textbox ها را به خواص ِ object ِ فروشگاه مورد نظر نسبت می دهیم . این کار به عنوان تمرین به شما واگذار میشود (پاسخ روی CD ضمیمه موجود است)
در کلید فروش باید ببینیم که کاربر با استفاده از RadioButton ها کدام فروشگاه را انتخاب کرده است و آیا این فروشگاه امکان فروش دارد یا خیر . وقتی فروشگاه مناسب انتخاب شد با استفاده از متدهای ایجاد شده اقدام به فروش یک بستنی ، نان و دیگر مواد می کنیم ، سپس مقدار پول (موجودی) را به میزان لازم افزایش داده و در آخر textbox ها را با مقادیر جدید پر می کنیم .

```
If RadioButton1.Checked And CanSellValiAsr Then
            ValiAsr.ForooshBastani()
            ValiAsr.ForooshGhashogh()
            ValiAsr.ForooshLivan()
            ValiAsr.ForooshNan()
```

```
            ValiAsr.MeghdarPool += 50
            txtNoon1.Text = ValiAsr.TedadNan
            lblPrice1.Text = ValiAsr.MeghdarPool
            txtBastani1.Text = ValiAsr.MeghdarBastani
            txtLivan1.Text = ValiAsr.TedadLivan
            txtGhashogh1.Text = ValiAsr.TedadGhashogh

ElseIf RadioButton2.Checked And CanSellFarmanie Then
            Farmanie.ForooshBastani()
            Farmanie.ForooshGhashogh()
            Farmanie.ForooshLivan()
            Farmanie.ForooshNan()
            Farmanie.MeghdarPool += 50
            txtNoon2.Text = Farmanie.TedadNan
            lblPrice2.Text = Farmanie.MeghdarPool
            txtBastani2.Text = Farmanie.MeghdarBastani
            txtLivan2.Text = Farmanie.TedadLivan
            txtGhashogh2.Text = Farmanie.TedadGhashogh
ElseIf RadioButton3.Checked And CanSellTehranPars Then
            TehranPars.ForooshBastani()
            TehranPars.ForooshGhashogh()
            TehranPars.ForooshLivan()
            TehranPars.ForooshNan()
            TehranPars.MeghdarPool += 50
            txtNoon3.Text = TehranPars.TedadNan
            lblPrice3.Text = TehranPars.MeghdarPool
            txtBastani3.Text = TehranPars.MeghdarBastani
            txtLivan3.Text = TehranPars.TedadLivan
            txtGhashogh3.Text = TehranPars.TedadGhashogh
End If
```

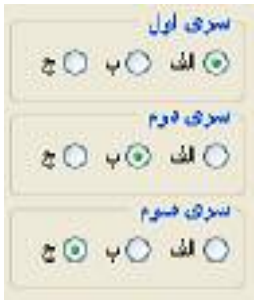
از آنجا که CanSellValiAsr و RadioButton1.Checked (و همزادهایشان) از نوع Boolean هستند، برای فهمیدن درستی(True) آنها نیازی به عملگر مقایسه ای نداریم.

Group Box

وقتی از RadioButton ها در یک فرم به شکلی که در بالا دیدید استفاده کنیم در هر لحظه فقط یکی از آنها امکان انتخاب شدن دارند. برای مثال در شکل زیر سه سری الف و ب و ج داریم که می خواهیم در هر لحظه یکی از هر سری انتخاب شده باشد . ولی در حالت معمولی امکان چنین کاری نیست .

برای حل این مشکل باید از کنترل دیگری به نام GroupBox استفاده کنیم .

GroupBox مانند یک فرم است و می توان بر روی آن کنترل های دیگری را قرار داد. به عبارت دیگر GroupBox یک Container است . در شکل زیر روش به کار گیری GroupBox مشخص شده است . GroupBox خاصیتی به نام Text دارد که متن نمایش داده شده بر روی آن را مشخص می کند .



CheckBox

کنترل بعدی که درباره آن صحبت مـی کنـیم CheckBox اسـت . از ایـن کنتـرل بـر خـلاف RadioButton بـرای مواقعی که انتخاب ها بـه هـم وابسـطه نیسـتند اسـتفاده مـی کننـد. بـه عبـارت دیگـر، RadioButton بـرای انتخاب از بین چند گزینه است (مانند کنکور!) و CheckBox برای انتخاب کـردن یـا انتخـاب نکـردن یـک گزینـه. برای مثال فرض کنید در برنامه بستنی فروش بخواهیم امکان اینکه بـا هربـار فـروش بسـتنی نـانی بفـروش برسد یا نرسد را به کاربر بدهیم می توانیم از یک CheckBox برای نان در هـر یـک از فروشـگاه هـا اسـتفاده کنیم . به شکل زیر توجه کنید :

هر گاه نانِ یک فروشگاه انتخاب شده باشد ، هنگام فروش از تعداد آن کاسته خواهد شـد ولـی در صـورتی که نان انتخاب نشده باشد بدین معنـی اسـت کـه بسـتنی بـدون نـان بـوده و از تعـداد نـان هـای موجـود در فروشگاه نباید عددی کم شود .

CheckBox ماننـد RadioButton دارای دو خاصـیت مهـم Text و Checked اسـت کـه دقیقـاً همـان اعمـال را انجام می دهند . تنها تغییری که در کد نیاز داریم در قسمت فروش است .

این سه خط

```
ValiAsr.FoorooshNan()
Farmanie.FoorooshNan()
TehranPars.FoorooshNan()
```

باید به سه خط زیر تبدیل شوند

```
If chkNan1.Checked Then ValiAsr.FoorooshNan()
```

```
If chkNan2.Checked Then Farmanie.ForooshNan()
If chkNan3.Checked Then TehranPars.ForooshNan()
```

تمرین

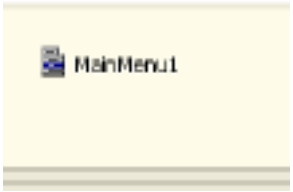
1. امکان فروختن بستنی بدون قاشق و لیوان را به برنامه اضافه کنید.
2. کاری کنید که پیش از "ثبت" کاربر نتواند بستنی بفروشد و با هر بار اجرای برنامه فقط یک بار بتواند عمل ثبت را اجام دهد.
3. دکمه هایی برای فروش یک نان، فروش یک لیوان، 50 گرم بستنی یا یک قاشق به فرم اضافه کنید. (قیمت آنها را خودتان تعیین کنید.)

فرمها و منوها

اگرچه برنامه ما درست کار میکند، اما میتواند بهتر از این هم باشد. میخواهیم با استفاده از کنترلهای حرفه ای تر، رابط کاربر بهتری برای طراحی کنیم. به جای RadioButton برای انتخابِ فروشگاه مورد نظر از ComboBox و به جای textbox هایی که برای دادن مقدار اولیه استفاده می شد، از inputBox و به جای label ی که پیغام ها را درونش می نوشتیم از MessageBox استفاده می کنیم . همچنین با کنترلهایی مثل Menu ، ContextMenu و Tooltip نیز آشنا می شویم . برای نمایش عکسِ بستنی ها از Picturebox و یک فرم جدید استفاده می کنیم .

Main Menu

از آنجا که میخواهیم برنامه را تقریبا از اول بنویسیم، یک پروژه جدید می سازیم و کلاسِ بستنی را در آن اضافه می کنیم(Project\Add Existing Item) . سپس بر روی فرم یک MainMenu قرار می دهیم تا در قسمتی که در شکل زیر می بینید جای گیرد .



در روی فرم نیز شکلی مانند زیر ایجاد می شود بر روی Type Here کلیک کنید و سپس کلمه "فایل" را بنویسید. سپس در زیر شاخه فایل "خروج" را بسازید و در کنار فایل امکانات را. همچنین در زیر شاخه امکانات منوی دیگری به نام "ثبت" ایجاد کنید .

در اکثر برنامه هایی که در ویندوز اجرا می شود اولین منو File است ، بعد از آن منوی Edit سپس View و .... . این نامها به استانداردی تبدیل شده است که در تمامی برنامه ها رعایت می شوند . ما هم باید از این استاندارد تبعیت کنیم تا برنامه خوانا تر و کاربر پسندتری برای کاربران داشته باشیم. مگر اینکه دلیل خوبی برای رعایت نکردن این استاندارد داشته باشیم.
این استاندارد فقط به منوهای اصلی محدود نمیشود و درباره آیتمهای درون آنها نیز صادق است. برای مثال "خروج" همیشه باید آخرین آیتم از اولین منو باشد. و About آخرین آیتم در آخرین منو. برای اینکه این استانداردها را بیاموزیم میتوانیم از برنامه های استاندارد، نکات زیادی بیاموزیم.

Status Bar

بسیاری از برنامه هایی که از یک حدی حجیم تر هستند از کنترلی به نام StatusBar برای نمایش وضعیت خود، به کاربر استفاده می کنند . ما نیز در این برنامه برای نمایش آخرین وضعیت بستنی فروشی ها (جدیدترین رویدادی که اتفاق افتاده است) از StatusBar استفاده می کنیم .

یک statusbar از toolbox بر داشته و بر روی فرم قرار دهید ، نام آن را به mStatusbar (پیشوند m را بیشتر برای member varibleها استفاده کرده ام، البته در کل فرقی نمیکنه ولی اگه را داره یه پیشوند دیگه براش بذار) تغییر دهید و در خواص آن ShowPanels را برابر True قرار دهید . Text آن را نیز پاک کنید تا اطلاعات لازم در طول اجرای برنامه در آن قرار گیرد .

ComboBox

کنترل بعدی که مورد بحث ما است کنترلی با نام ComboBox می باشد ، هنگامی که بر روی این کنترل کلیک شود اطلاعاتش به شکل یک لیست از پایین آن باز شده و نمایش داده می شود . مانند شکل زیر :

((مایکروسافت برا combo از پیشوند cbo استفاده میکنه. رجوع شود به " INFO: Object Hungarian Notation Naming Conventions for VB" در MSDN "))

یک ComboBox با نام cmbStores بر روی فرم قرار دهید و خاصیت DropDownStyle را به DropDownList تغییر دهید تا هنگام اجرا کاربر نتواند مقدار آن را تغییر دهد فقط بتواند مقادیری که در آن از قبل قرار دارد را انتخاب کند .

شکل کلی فرم ما چنین خواهد شد :

در اینجا نیز از GroupBox استفاده کرده ایم، ولی فقط کارکرد ظاهری دارد. و برای این است که کاربر متوجه وابستگی کنترلهای داخل GroupBox به یکدیگر شود.

کلاس Collection

پیش از آنکه به نوشتن کد برای استفاده از ComboBox بپردازیم لازم است با کلاس Collection آشنا شویم. شی Collection شیی است که میتواند مجموعه از اشیاء را در خود نگه دارد. به عبارت دیگر شی Collection یک کلکسیون از اشیاء است. این شی دو خاصیت و دو متد مهم دارد.

خاصیت Count : این خاصیت تعداد اعضای مجموعه ای که Collection نگهداری میکند را به ما میدهد. یعنی تعداد آیتم های مجموعه.

خاصیت Item : این خاصیت میتواند یکی از عضوهای شیی که از نوع Collection ایجاد شده است به ما برگرداند. اگر ورودی عدد Index باشد، خاصیت Item، شیی که در خانه Index+1 ام از Collection است، برمیگرداند(که میتوانیم مقدار آن را بخوانیم یا تغییر دهیم.)

متد Add : با این متد میتوانیم یک Item به Collection اضافه کنیم. پارامتر اول، شیی است که باید به Collection اضافه شود، دومی که پارامتری اختیاری می باشد؛ کلید بازیابی شی است (اگر آن را تنظیم کنیم میتوانیم کلید را بجای Index برای بازیابی به خاصیت Item بدهیم). اگر به این پارامتر مقدار ندهیم کلید آن خالی میماند. پارامتر سوم محل درج شی در Collection را برحسب اینکه قبل یا بعد از کدام شی باشد، تعیین میکند. اگر این پارامتر مقدار نگیرد، شی جدید به انتهای Collection اضافه مشود. در عمل پر کاربرد ترین حالت استفاده از این متد، حالتی است که فقط پارامتر اول مقدار بگیرد.
متد Remove : این متد یک آیتم را از Collection حذف میکند. تنها پارامتر ورودی آن، میتواند Index یا کلید(Key) باشد.

کار با کنترلهای جدید

حال باید کدهای مورد نظر را بنویسیم ، ابتدا کدهایی که در مدل قبلیِ مورد استفاده قرار گرفته بود و در اینجا نیز استفاده می شود را اضافه کنیم .

```
Private WithEvents ValiAsr As New bastani
Private WithEvents Farmanie As New bastani
Private WithEvents TehranPars As New bastani

Dim CanSellValiAsr As Boolean = True
Dim CanSellFarmanie As Boolean = True
Dim CanSellTehranPars As Boolean = True
```

آیتم های منو نیز مانند دکمه ها یک رویداد کلیک بسار پرکاربرد دارند. برای اضافه کردن کدهای رویداد کلیک منو، بر روی آن دبل کلیک می کنیم. کد زیر، کدی است که برای منوی خروج احتیاج داریم :

```
Private Sub MenuItem2_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MenuItem2.Click
        End
End Sub
```

سابروتینِ load در فرم اصلی کمی تغییر می کند .

```
ValiAsr.Name = "ValiAsr"
Farmanie.Name = "Farmanie"
TehranPars.Name = "TehranPars"
```

اضافه کردن آیتم به ComboBox

```
cmbStores.Items.Add("ValiAsr")
cmbStores.Items.Add("Farmanie")
cmbStores.Items.Add("TehranPars")
```

شی Items در ComboBox که دربردارنده آیتمهای موجود در لیست بازشونده است. یک نمونه از کلاس Collection است. برای اضافه کردنِ یک آیتم به ComboBox (همان طور که در بالا می بینید) از متدِ Add در Items استفاده می کنیم(اضافه کردن به Collection) . به عنوان پارامترِ Add متنی را که می خواهیم در ComboBox اضافه شود به Add انتقال می دهیم . خروجیِ Add یک عدد صحیح است که شماره عنصر اضافه شده را بر می گرداند.

کد رویدادها نیز کمی تغییر می کند ، برای مثال کد رویداد تمام شدن بستی را در زیر می بینید :

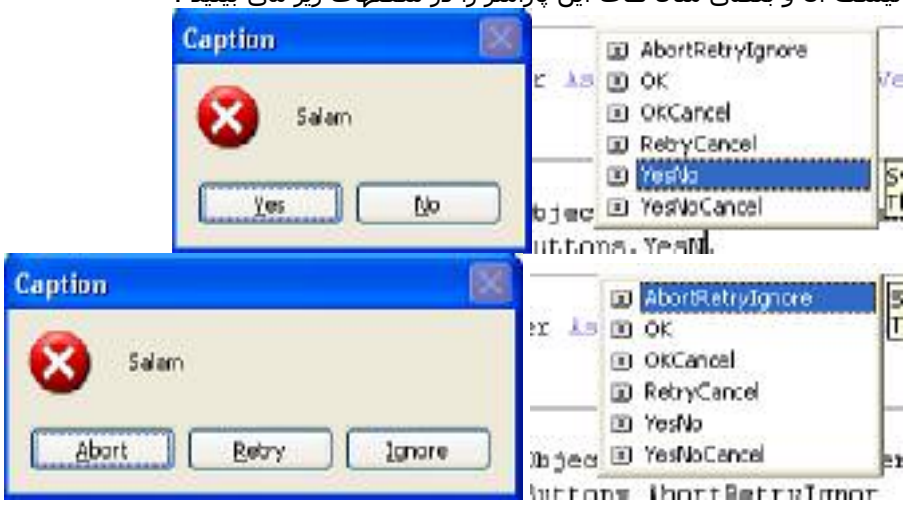
```
MessageBox.Show("بستي " & Sender & " شد تمام")
mStatusBar.Text = "بستي " & Sender & " شد تمام"
MakeDisable(Sender)
```

تابع MakeDisable همانند مدل قبل عملیاتِ مربوط به خود را انجام می دهد .
MessageBox شی ی است که با استفاده از متدِ Show در آن می توانیم متن مورد نظر را به کاربر در یک قالب مشخص نمایش دهیم .

کد زیر پیغام بالا را نمایش داده است :

```
MessageBox.Show("Salam", "Caption", MessageBoxButtons.OK, _
    MessageBoxIcon.Error, MessageBoxDefaultButton.Button1)
```

متدِ Show دارای پارامترهای مختلفی است ، اولین پارامتر متنی است که می خواهید به عنوان پیغام نمایش داده شود؛ در شکل بالا از "Salam" برای این پارامتر استفاده کردیم . دومین پارامتر عنوانِ این پنجره است که در اینجا از "Caption" استفاده شده ، پارامتر سوم نوع کلید هایی است نمایش داده می شود ، لیست آن و بعضی مثال های این پارامتر را در شکلهای زیر می بینید :



پارامتر چهارم نوع Icon ی است که در پنجره ی MessageBox نمایش داده می شود . این Icon نشان دهنده هدف ما از این پیغام برای کاربر است . باید Iconای نمایش بدهیم که واقعا به منظور ما مربوط است. علاوه بر این، باید در استفاده Icon دایره با ضربدر سفید (Error) کمی با خساست عمل می کنیم. چون استفاده زیاد از آن، حساست کاربر را برای پیغامهای بحرانی کم می کند. پس استفاده از این Icon را به مواقع بحرانی محدود می کنم.

پارامتر آخر برای انتخاب دکمه های نمایش داده شده است ، مثلاً در شکل بالا که پنجره ی نمایش داده شده دارای سه دکمه است ، انتخاب ما اولین کلید بوده . دیگر متغیرهای این پارامتر را در شکل زیر می بینید :

البته دقت کنید که متدِ Show در MessageBox دارای 12 overload[7] با پارامترهای مختلف است . متد Show به عنوان مقدار بازگشتی، کلید زده شده را بر می گرداند . به این ترتیب میتوانیم متوجه شویم کاربر روی کدام کلید کلیک کرده. توجه داشته باشید که ما نمیتوانیم برای رویدادهای دکمه های یکی MessageBox برنامه بنویسیم.

هر بار که از ComboBox نام فروشگاه را انتخاب کنیم باید اطلاعات آن را نمایش دهد ، برای این کار از رویدادی به نام SelectedIndexChanged استفاده می کنیم. کد مورد نظر در این سابروتین را در پایین می بینید :

```
    Private Sub cmbStores_SelectedIndexChanged(ByVal sender As
System.Object, ByVal e As System.EventArgs) Handles
cmbStores.SelectedIndexChanged
        Select Case cmbStores.Text
            Case "ValiAsr"
                lblPrice.Text = ValiAsr.MeghdarPool
                lblNoon.Text = ValiAsr.TedadNan
                lblGhashogh.Text = ValiAsr.TedadGhashogh
                lblLivan.Text = ValiAsr.TedadLivan
                lblICECream.Text = ValiAsr.MeghdarBastani
            Case "Farmanie"
                lblPrice.Text = Farmanie.MeghdarPool
                lblNoon.Text = Farmanie.TedadNan
                lblGhashogh.Text = Farmanie.TedadGhashogh
                lblLivan.Text = Farmanie.TedadLivan
                lblICECream.Text = Farmanie.MeghdarBastani
            Case "TehranPars"
                lblPrice.Text = TehranPars.MeghdarPool
                lblNoon.Text = TehranPars.TedadNan
                lblGhashogh.Text = TehranPars.TedadGhashogh
                lblLivan.Text = TehranPars.TedadLivan
                lblICECream.Text = TehranPars.MeghdarBastani
        End Select
    End Sub
```

[7] بار اضافی دادن به زیربرنامه یا تابع، که در فصل مربوط به زیربرنامه ها بررسی می شود.

هر بار که فروشگاه دیگری انتخاب می شود از خاصیت Text (متی که در ComboBox نوشته شده است.) استفاده می کنیم ، سپس با استفاده از Select Case تشخیص می دهیم اطلاعات کدام فروشگاه باید نمایش داده شوند و اطلاعات آن را به label های پیشبینی شده انتقال می دهیم .

برای نوشتن کد های منوی ثبت ، بر روی آن کلیک کنید و سپس کد زیر را برای سابروتین ایجاد شده بنویسید :

```
Select Case cmbStores.Text
   Case "ValiAsr"
      lblPrice.Text = ValiAsr.MeghdarPool
      ValiAsr.TedadNan = InputBox("بنویسید را عصر ولي در موجود نان تعداد
؟")
      lblNoon.Text = ValiAsr.TedadNan
      ValiAsr.TedadGhashogh = InputBox("را عصر ولي در موجود قاشق تعداد
؟ بنویسید")
      lblGhashogh.Text = ValiAsr.TedadGhashogh
      ValiAsr.TedadLivan = InputBox("را عصر ولي در موجود لیوان تعداد
؟ بنویسید")
      lblLivan.Text = ValiAsr.TedadLivan
      ValiAsr.MeghdarBastani = InputBox("را عصر ولي در موجود بستني مقدار
؟ بنویسید")
      lblICECream.Text = ValiAsr.MeghdarBastani
   Case "Farmanie"
      lblPrice.Text = Farmanie.MeghdarPool
      Farmanie.TedadNan = InputBox("را فرمانیه در موجود نان تعداد
؟ بنویسید")
      lblNoon.Text = Farmanie.TedadNan
      Farmanie.TedadGhashogh = InputBox("را فرمانیه در موجود قاشق تعداد
؟ بنویسید")
      lblGhashogh.Text = Farmanie.TedadGhashogh
      Farmanie.TedadLivan = InputBox("را فرمانیه در موجود لیوان تعداد
؟ بنویسید")
      lblLivan.Text = Farmanie.TedadLivan
      Farmanie.MeghdarBastani = InputBox("فرمانیه در موجود بستني مقدار
؟ بنویسید را")
      lblICECream.Text = Farmanie.MeghdarBastani
   Case "TehranPars"
      lblPrice.Text = TehranPars.MeghdarPool
      TehranPars.TedadNan = InputBox("را پارس تهران در موجود نان تعداد
؟ بنویسید")
      lblNoon.Text = TehranPars.TedadNan
      TehranPars.TedadGhashogh = InputBox("تهران در موجود قاشق تعداد
؟ بنویسید را پارس")
      lblGhashogh.Text = TehranPars.TedadGhashogh
      TehranPars.TedadLivan = InputBox("پارس تهران در موجود لیوان تعداد
؟ بنویسید را")
      lblLivan.Text = TehranPars.TedadLivan
      TehranPars.MeghdarBastani = InputBox("تهران در موجود بستني مقدار
؟ بنویسید را پارس")
      lblICECream.Text = TehranPars.MeghdarBastani
End Select
```

در این کد برای اینکه هر متغییر را از کاربر دریافت کنیم ازشی ی به نام InputBox استفاده کردیم .

این شی نیز مانند MessageBox دارای پارامترهای مختلفی است ، در خروجی نیز متنی را که در textbox_ آن می نویسید نمایش می دهد . اشکالی که این روش در برنامه ما دارد این است که برای یک فروشگاه باید چهار بار از Inputbox استفاده کنیم و این روش جالبی نیست . در ادامه روش بهتری با استفاده از Windows Form را خواهیم دید .

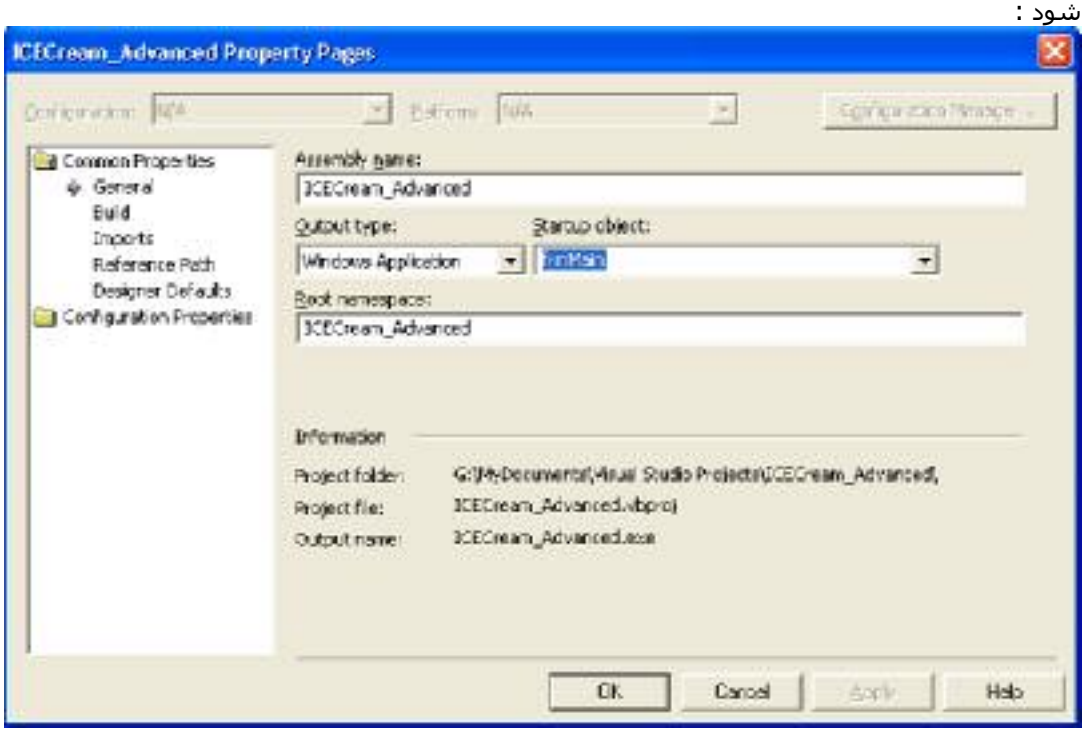
توجه : کد کلید فروش مانند مدل قبلی است و تفاوتی ندارد .

استفاده از Windows Form های جدید

همان طور که در بخش قبلی مشاهده کردید استفاده مکرر از inputbox روش مناسبی برای دریافت چندین داده متوالی نیست . برای اینکه بتوانیم اطلاعات مناسب را دریافت کنیم از یک Windows Form جدید استفاده می کنیم .
تمامی برنامه هایی که تا این فصل ساختیم تنها از یک Windows Form تشکیل شده بودند. دراکثر برنامه های واقعی امکان جا سازی تمام امکانات در یک فرم وجود ندارد و گاهی چند صد فرم در یک برنامه استفاده می شود .
برای اینکه یک فرم جدید به برنامه اضافه کنیم در قسمت Project Explorer روی پروژه کلیک سمت راست موس را می زنیم تا منویی مانند شکل زیر باز شود .

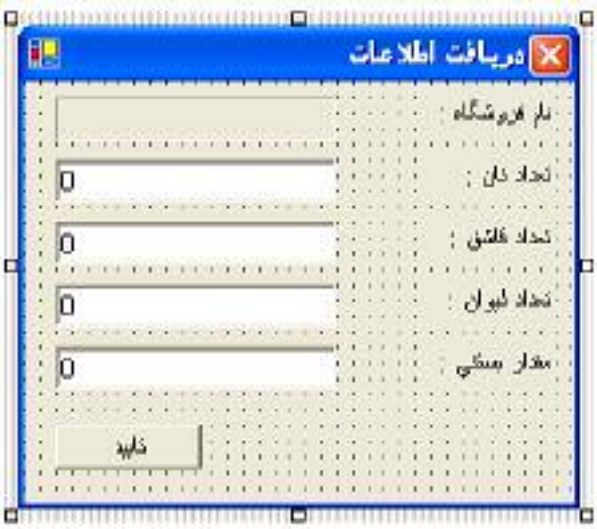
بعد از انتخاب Windows Form پنجره جدیدی باز می شود که در آن نام فرم را می توانیم بنویسیم . وقتی این کار را انجام دادید فرم جدید در قسمت Project Explorer نمایش داده می شود .

در هر پروژه تنها یک فرم می تواند به عنوان فرم ابتدایی انتخاب شود ، فرم ابتدایی فرمی است که هنگام اجرای فایل EXE برنامه نمایش داده می شود . برای تغییر آن روی نام پروژه در Project Explorer کلیک سمت راست موس را بزنید و سپس properties را انتخاب کنید . پنجره ای مانند شکل زیر نمایش داده می شود :



لیست فرم هایی که در پروژه وجود دارد در قسمت Startup Object وجود دارد و از آنجا می توانیم فرم ابتدایی را تغییر دهیم (دراینجا نیازی به این کار نیست).

برای شروع کار با فرم جدیدی که ساختیم روی آن دبل کلیک کنید تا قسمت طراحی ظاهری آن نمایان شود ، سپس کنترلهایی مانند شکل زیر را بر روی آن قرار دهید .



وقتی فرمی را به پروژه اضافه می کنیم، آن فرم به معنای یک کلاس جدید است و مانند تمامی کلاس ها باید یک نمونه از روی آن بسازید . برای مثال می خواهیم فرم جدیدی را که ساختیم نمایش دهیم . نام کلاسِ فرم ما frmInformation است. و میخواهیم وقتی روی آیتم "ثبت" از منوی "امکانات" کلیک شد، نمایش داده شود. پس کد مربوط به نمایش دادن فرم را در این رویداد مینویسیم:

```
Private Sub MenuItem5_Click(…) Handles MenuItem5.Click
    Dim frm As New frmInformation
End Sub
```

بعد از نوشتن این کد frm یک نمونه از کلاسِ frmInformation است و می توان از خواص و متد های آن استفاده کرد . ما می خواهیم که این فرم نمایش داده شود و اطلاعات مورد نظر را درون آن وارد کنیم . با خطی که در بالا دیده اید فرمی به نمایش در نخواهد آمد بلکه فقط حافظه ای برای آن گرفته شده و از روی کلاس آن نمونه سازی می شود. برای نمایش فرم دو روش وجود دارد که با توجه به کاربرد فرم مورد نظر انتخاب می شود . روش اول استفاده از متدِ Show است . وقتی این متد صدا زده شود فرم به نمایش در می آید و به عنوان فرم فعال برنامه در روی دیگر فرمها قرار می گیرد ، اما اگر بر روی دیگر فرم های برنامه کلیک شود این فرم از فعال بودن خارج می شود و فرمی که روی آن کلیک شده فعال می شود. در بعضی حالات این روش ممکن است مشکل ساز شود و ما بخواهیم همیشه فرمِ مورد نظرمان در روی دیگر فرم های برنامه باشد . در آن حالت به جای Show از ShowDialog استفاده می کنیم . دقت کنید که در اینجا همیشه رو بودنِ فرم فقط مخصوص برنامه ما است نه کل سیستم عامل ، برای اینکه فرم در کل برنامه های اجرا شده در سیستم عامل همیشه روی دیگر فرم ها باشد از TopMost باید استفاده کرد. پس ما با استفاده از این خط فرم را می توانیم نمایش دهیم .

```
frm.ShowDialog()
```

اما قبل از این کار می خواهیم در فرم دریافت اطلاعات، نام فروشگاهِ انتخاب شده در نمایش داده شود . اگر در لیست اعضای frm دقت کنید نام کلیه کنترلهای موجود بر روی آن فرم را خواهید دید . در اینجا به راحتی با استفاده از یکی از label هایی که در روی فرم است و برای نام فروشگاه در نظر گرفتیم ، نام آن را به فرم منتقل می کنیم .

```
frm.lblStoreName.Text = cmbStores.Text
```

پس سه خط ما به این ترتیب قرار خواهد گرفت :

```
Dim frm As New frmInformation
frm.lblStoreName.Text = cmbStores.Text
frm.ShowDialog()
```

بعد از اجرای این سه خط فرم نمایش داده می شود. بر روی فرم کلیدی داریم با نام تایید که بعد از کلیک بر روی آن فرم ورود اطلاعات باید بسته شود و اطلاعات وارد شده آن به شیِ فروشگاهِ انتخاب شده انتقال یابد . برای اینکه یک فرم را ببندید می توانید از متدِ Close آن فرم استفاده کنید . در سابروتینِ کلیکِ دکمه ی تایید این یک خط را بنویسید .

```
Close()
```

وقتی این خط اجرا شد کنترل برنامه درست به خطِ بعد از frm.ShowDialog() در سابروتینِ فرا خواننده باز خواهد گشت ، حال باید با توجه به مقدارِ کنترلهای موجود در فرمِ ورود اطلاعات و فروشگاهِ انتخاب شده اطلاعات را انتقال دهیم :

```
Dim frm As New frmInformation
frm.lblStoreName.Text = cmbStores.Text
frm.ShowDialog()
Select Case cmbStores.Text
       Case "ValiAsr"
           lblPrice.Text = ValiAsr.MeghdarPool
           ValiAsr.TedadNan = frm.txtNoon.Text.Trim
           lblNoon.Text = ValiAsr.TedadNan
           ValiAsr.TedadGhashogh = frm.txtGhashogh.Text.Trim
           lblGhashogh.Text = ValiAsr.TedadGhashogh
           ValiAsr.TedadLivan = frm.txtLivan.Text.Trim
           lblLivan.Text = ValiAsr.TedadLivan
           ValiAsr.MeghdarBastani = frm.txtBastani.Text.Trim
           lblICECream.Text = ValiAsr.MeghdarBastani
       Case "Farmanie"
           lblPrice.Text = Farmanie.MeghdarPool
           Farmanie.TedadNan = frm.txtNoon.Text.Trim
           lblNoon.Text = Farmanie.TedadNan
           Farmanie.TedadGhashogh = frm.txtGhashogh.Text.Trim
           lblGhashogh.Text = Farmanie.TedadGhashogh
           Farmanie.TedadLivan = frm.txtLivan.Text.Trim
           lblLivan.Text = Farmanie.TedadLivan
           Farmanie.MeghdarBastani = frm.txtBastani.Text.Trim
           lblICECream.Text = Farmanie.MeghdarBastani
       Case "TehranPars"
           lblPrice.Text = TehranPars.MeghdarPool
           TehranPars.TedadNan = frm.txtNoon.Text.Trim
           lblNoon.Text = TehranPars.TedadNan
           TehranPars.TedadGhashogh = frm.txtGhashogh.Text.Trim
           lblGhashogh.Text = TehranPars.TedadGhashogh
           TehranPars.TedadLivan = frm.txtLivan.Text.Trim
           lblLivan.Text = TehranPars.TedadLivan
           TehranPars.MeghdarBastani = frm.txtBastani.Text.Trim
           lblICECream.Text = TehranPars.MeghdarBastani
End Select
```

در اینجا از یک متد از کلاس String به نام Trim استفاده کرده ایم. این متد فاصله های خالی اضافه را از دو طرف متن حذف میکند و حاصل را به ما برمیگرداند ولی شیی را که برای آن صدا شده تغییر نمیدهد.
همان طور که مشاهده می کنید اطلاعاتی که در کنترلهای فرمِ ورود اطلاعات قرار داشت قابل بازیابی است . بعد از End Select می توانیم حافظه اختصاص داده شده به فرم ورود اطلاعات را کلاً آزاد کنیم . این عملیات بوسیله خط زیر انجام می شود :

```
frm.Dispose()
```

به برنامه هایی از ایندست، که فرمهایی جدا از هم دارد، برنامه SDI[8] گفته میشود. در برابر این نوع برنامه ها، نوع دیگری از برنامه های ویندوز به نام MDI[9] وجود دارد. به نرم افزارهایی مانند MS Word یا VS.NET نگاه کنید ، اکثر فرمهای جدیدی که فراخوانی می کنیم در یک فرم اصلیِ دیگر که مانند parent است قرار می گیرد . به آن فرمِ اصلی MDI Form و به فرم هایی که درون فرم اصلی قرار می گیرند MDI Child Form گفته می شود .

تمرین
اگر در ComboBox هیچ گزینه ای انتخاب نشده باشد، عمل ثبت بی ثمر خواهد بود. این مشکل را حل کنید.

پروژه
برای مساله کتابخانه که فصل قبل مطرح شد، رابط کاربری طراحی کنید که بتواند کتابهایی اضافه یا حذف کند.

[8] Single Document Interface
[9] Multiple Document Interface

توابع و زیربرنامه ها

پیش از این بارها از زیر برنامه ها و توابع استفاده کرده ایم، برای مثال int و rnd دو تا از توابعی هستند که در فصل ششم از آنها استفاده کردیم. با تعریف و استفاده از متدها نیز آشنا شده ایم. متدها زیربرنامه های عضو یک کلاس هستند. به عبارت دیگر به توابع و زیربرنامه هایی که در یک کلاس تعریف شوند، توابع عضو یا زیربرنامه های عضو گفته میشود، که نام دیگر متد است. در این فصل می خواهیم بیشتر با توابع و زیربرنامه ها آشنا شویم.

احضار زیربرنامه

زیربرنامه بخشی از برنامه است که بصورت مجزا تعریف میشود تا سایربخشهای برنامه بتوانند از آن استفاده کنند. برای مثال دربرنامه بستنی فروشی ما یک زیربرنامه عضو ForooshNan در کلاس بستنی تعریف کردیم تا هر بخشی از برنامه که قصد دارد یک فروش بستنی ثبت کند، از آن استفاده کند. استفاده از زیربرنامه ها علاوه بر سازماندهی برنامه، باعث صرفجویی در حافظه کامپیوتر نیز میشود. و جلوی تکرار شدن بی مورد کد زیربرنامه در تمام برنامه را میگیرد.

تعریف زیربرنامه

تمام متدهایی که تابحال تعریف کرده ایم، زیربرنامه های عضو بوده اند. زیربرنامه معمولا وقتی استفاده میشود که میخواهیم کاری انجام شود اما لازم نیست که نتیجه انجام آن کار را به ما برگرداند. برای مثال زیربرنامه عضو ()ForooshNan باید یکی از تعداد نانها کم کند و نیازی نیست مقداری به ما برگرداند. در مورد توابعی چون int و rnd شرایط فرق میکند و ما میخواهیم مقداری را از این توابع دریافت کنیم.

چه وقت باید یک زیربرنامه جدید ایجاد کنیم؟ اگر فکر میکنید بخشی از زیربرنامه یا تابعی که درحال نوشتن آن هستید، در بخشهای دیگر برنامه استفاده خواهد شد، آن بخش را به یک زیربرنامه جدا تبدیل کنید تا بخشهای دیگر برنامه هم بتوانند بدون تکرار کردن کد مشابه از آن استفاده کنند.

به عنوان مثال میخواهیم تمرین یک از فصل شش را حل کنیم. فرض کنید برای حل کردن این تمرین از این روش استفاده میکنیم که در ابتدا همه دکمه ها غیر فعال (پروپرتی enable برابر false) است و پس از زدن دکمه start فعال میشوند. در این صورت میتوانیم زیربرنامه عضوی به اسم EnableAll درست کنیم که وظیفه فعال کردن تمام دکمه ها را به عهده داشته باشد. حال اگر هرجای دیگر از برنامه لازم شود این دکمه ها فعال شوند، فقط کافی است این زیربرنامه صدا شود.

تعریف دقیقا همان تعریفی است که برای متدها گفته بودیم :

```
Private Sub EnableAll()
    btnCalc0.Enabled = True
    btnCalc1.Enabled = True
    btnCalc2.Enabled = True
    btnCalc3.Enabled = True
    btnCalc4.Enabled = True
    btnCalc5.Enabled = True
    btnCalc6.Enabled = True
    btnCalc7.Enabled = True
    btnCalc8.Enabled = True
    btnCalc9.Enabled = True
    btnClear.Enabled = True
    btnBackspace.Enabled = True
    btnOK.Enabled = True
End Sub
```

احتمالا این سوال پیش می آید که چرا متدهایی که در کلاس bastani تعریف کردیم، همگی Public بودند و این متد Private است؟ جواب این است که Private یا Public بودن متدها و زیربرنامه ها بستگی به شما دارد. ما فکر کردیم زیربرنامه عضو EnableAll در هیچ جای دیگر از برنامه بجز همین کلاس (کلاس فرم) نباید استفاده شود، به همین دلیل آن را Private تعریف کردیم.

اکنون زیربرنامه btnNew_Click نیز به شکل زیر تغییر می کند:

```
Private Sub btnNew_Click(…) Handles btnNew.Click
  …
    mNumber = Int((txtEnd.Text - txtStart.Text + 1) * Rnd() +
txtStart.Text)
    EnableAll()
```

```
    …
End Sub
```

تمرین 6 از همان فصل میگوید کاری کنید که کاربر پس از برنده شدن نتواند عدد جدیدی انتخاب کند، برای اینکه جلوی انتخاب عدد جدید را بگیریم میتوانیم همه دکمه ها را غیرفعال (پروپرتی Enabled برابر false) کنیم. میتوانیم برای حل این تمرین نیز یک زیربرنامه جدید ایجاد کنیم، اما پیش از اینکه این کار را انجام دهید؛ کمی صبر کنید. راه بهتری هم هست.

آرگمان و پارامتر

زیربرنامه میتواند مقدارهایی نیز از قطعه برنامه ای که آن را فراخوانی میکند بگیرد. به این مقدارها آرگمانهای زیربرنامه گفته میشود :

```
[Private|Public] Sub SubName (arg As TypeName)

End Sub
```

در اینجا ما یک زیربرنامه (sub) تعریف کرده ایم که یک آرگمان ورودی میگیرد. بخشی از برنامه که میخواهد این تابع را فراخوانی کند باید مقداری از نوع TypeName نیز به این زیربرنامه بدهد. به مقداری که در فراخوانی زیربرنامه یا تابع قرار میگیرد پارامتر گفته میشود.

اجازه بدهید به تمرین 6 از فصل 6 برگردیم. تفاوت EnableAll با زیربرنامه ای که ما نیاز داریم در این است که EnableAll خاصیت Enabled را برای دکمه ها برابر True میکند و برای این تمرین ما به زیربرنامه ای احتیاج داریم که Enabled را False کند. برای حل این مساله کاری میکنیم که زیربرنامه EnableAll مقدار خاصیت Enabled دکمه ها را از زیربرنامه ای که آن را صدا میکند بگیرد. به این ترتیب زیربرنامه EnableAll هم میتواند این مقدار را به true هم به false تغییر دهد. و اینکه به کدام یک تغییر میدهد بستگی به پارامتری دارد که زیربرنامه ای که EnableAll را صدا میکند به آن پاس میدهد.

```
Private Sub EnableAll(ByVal val As Boolean)
    btnCalc0.Enabled = val
    btnCalc1.Enabled = val
    btnCalc2.Enabled = val
    btnCalc3.Enabled = val
    btnCalc4.Enabled = val
    btnCalc5.Enabled = val
    btnCalc6.Enabled = val
    btnCalc7.Enabled = val
    btnCalc8.Enabled = val
    btnCalc9.Enabled = val
    btnClear.Enabled = val
    btnBackspace.Enabled = val
    btnOK.Enabled = val
End Sub
```

زیربرنامه EnableAll مقدار آرگمان val را در خاصیت Enabled همه دکمه ها جایگزین میکند. مقدار val تا زمانی که این زیربرنامه فراخوانی نشود مشخص نمیشود. برای فراخوانی این زیربرنامه، در پرانتز باز و بسته بعد از اسم زیربرنامه ، یک مقدار یا متغیر boolean (یا قابل تبدیل ضمنی شدن به boolean) قرار میدهیم.

```
Private Sub btnNew_Click(…) Handles btnNew.Click
    …
    mNumber = Int((txtEnd.Text - txtStart.Text + 1) * Rnd() + txtStart.Text)
    EnableAll(True)
    …
End Sub
```

وقتی اجرای برنامه به خط EnableAll(True) برسد، مقدار پارامتر val را که برابر True است در آرگمان val در زیربرنامه EnableAll کپی میکند. و سپس آن زیربرنامه را فراخوانی میکند و اجرای برنامه به خط اول زیربرنامه منتقل میشود. وقتی زیربرنامه اجرا میشود مقدار آرگمان val برابر True است، پس خاصیت Enabled همه دکمه ها را به True تغییر میدهد. و در پایان زیربرنامه، اجرای برنامه به خط بعد از دستور فراخوانی زیربرنامه منتقل میشود.

برای حل تمرین 6 از فصل 6 میتوانیم از کد زیر استفاده کنیم :

```
Private Sub btnOK_Click(…) Handles btnOK.Click
    Dim tmp As Long = Convert.ToInt64(txtNumber.Text)
    If tmp < mNumber Then
```

```
            lblResult.Text = "کنید وارد بزرگتری عدد"
        ElseIf tmp > mNumber Then
            lblResult.Text = "کنید وارد کوچکتری عدد"
        Else
            ' کرده تغییر 6 فصل از 7 تمرین از بخشی حل برای
            lblResult.Text = "برنده غلط حدس " & counter & " از پس شما "
."شدید
            EnableAll(False)
        End If
        …
    End Sub
```

اینبار وقتی اجرای برنامه به خط EnableAll(False) میرسد، پارامتر val برابر False است، پس False در آرگمان val کپی میشود و در نتیجه مقدار Enable همه دکمه ها برابر False میشود.

اگر زیربرنامه ما بیشتر از یک آرگمان داشته باشد، میتوانیم آرگمانها را با استفاده از کاما (,) از هم جدا کنیم:

```
[Private|Public] Sub SubName (arg1 As TypeName1, arg2 As TypeName)

End Sub
```

زمان فراخوانی نیز، باید پارامترها با استفاده از کاما از هم جدا شوند.

نکته : آرگمانها میتوانند از هرنوع داده ای باشند، از انواع داده پایه بسیار ساده نظیر Boolean تا اشیاء بزرگی مثل یک دکمه یا یک فرم.

توابع و مقدار برگشتی

تا اینجا یاد گرفتیم که چگونه یک زیربرنامه تعریف کنیم که آرگمان ورودی داشته باشد. در اینجا میخواهیم به بحث توابع بپردازیم. توابع علاوه بر اینکه تمام قابلیتهای یک زیربرنامه(sub) را دارند، میتوانند یک مقدار برگشتی نیز به زیربرنامه یا تابعی که آنها را صدا کرده است برگردانند.

ساختار تعریف یک تابع تا حدود زیادی شبیه تعریف یک زیربرنامه است :

```
[Private|Public] Function SubName (arg As TypeName) [As TypeName]

End Function
```

با این تفاوت که بجای کلمه کلیدی Sub که نمایانگر یک زیربرنامه است از کلمه کلیدی Function که نمایانگر تعریف یک تابع است، استفاده میکنیم. تفاوت دیگر در تعریف یک تابع با تعریف یک زیربرنامه در این است که در تعریف توابع، بعد از پرانتز بسته، میتوانیم با استفاده از کلمه کلیدی As نوع مقدار برگشتی تابع را تعیین کنیم. تابع Add که در زیر تعریف شده است، دو مقدار صحیح X و Y را به عنوان آرگمانهای ورودی دریافت کرده و یک مقدار صحیح برمیگرداند:

```
Private Function Add(ByVal x As Integer, ByVal y As Integer) As Integer

End Function
```

برای برگرداندن یک مقدار به زیربرنامه یا تابعی که تابع ما را صدا کرده است از دستور return استفاده میکنیم:

```
Private Function Add(ByVal x As Integer, ByVal y As Integer) As Integer
    Dim tmp As Integer
    tmp = x + y
    Return tmp
End Function
```

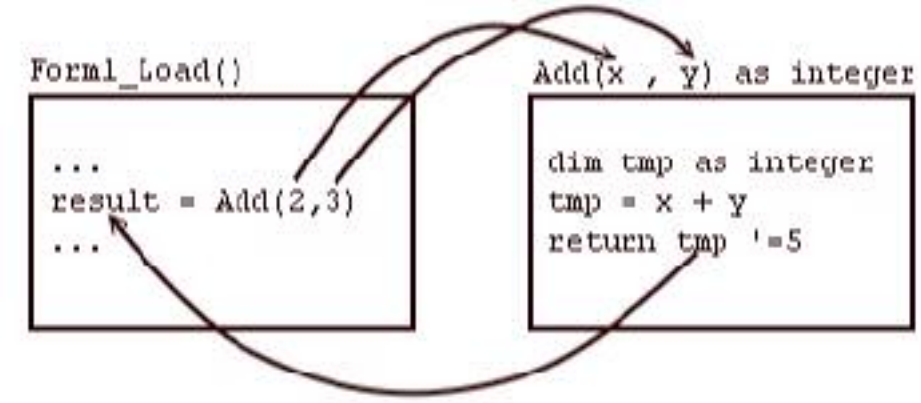
پس از اجرای دستور return اجرای برنامه به خطی که تابع را فراخوانی کرده بود بازمیگردد (حتی اگر return آخرین دستور تابع نباشد) و مقدار بازگشتی را در متغیری که احتمالا در انتظار مقدار بازگشتی تابع است جایگزین میکند.

فراخوانی توابع مانند فراخوانی زیربرنامه ها است. با این تفاوت که میتوانیم برای گرفتن مقدار برگشتی تابع، از عملگر جایگزینی استفاده کنیم. دقت کنید که تابع همیشه سمت راست عملگر جایگزینی قرار میگیرد و متغیری که میخواهد مقدار برگشتی تابع را بگیرد سمت چپ عملگر جایگزینی:

```
Private Sub Form1_Load(…) Handles MyBase.Load
    Dim result As Integer
    result = Add(2, 3)
```

```
    Label1.Text = result
End Sub
```

وقتی برنامه اجرا میشود، Form1_Load اجرا خواهد شد و متغیر result را از نوع integer تعریف میکند. پس از آن اجرای برنامه به خط result = Add(2, 3) میرسد. برای اینکه مقدار result مشخص شود باید تابع add فراخوانی شود. در اینصورت پارامتر اول (2) در آرگمان اول یعنی x و پارامتر دوم (3) در آرگمان دوم، یعنی y جایگزین خواهد شد و اجرای برنامه به خط اول تابع add منتقل میشود، در آنجا متغیری به نام tmp از نوع Integer تعریف میشود و در خط دوم حاصل جمع آرگمانهای ورودی در آن قرار میگیرد. اکنون مقدار tmp برابر 5 است. اجرای برنامه به خط return tmp میرسد و مقدار برگشتی تابع مشخص شده و برگردانده میشود. اجرای برنامه به خط فراخوانی کنند برمیگردد و مقدار برگشتی از تابع (در اینجا 5) در متغیر result جایگزین میشود.



تابع add را میتوانستیم خلاصه تر نیز بنویسیم:

```
Private Function Add(ByVal x As Integer, ByVal y As Integer) As Integer
    Return x + y
End Function
```

در اینجا دیگر از متغیر tmp برای نگهداری حاصل جمع استفاده نکرده ایم و حاصل جمع را مستقیما به دستور return داده ایم. حاصل کار همان است که در بخش قبل داشتیم، اما این کد بیشتر در حافظه صرفه جویی میکند. بطور ساده میتوان گفت این تابع مقدار x و y را با هم جمع کرده و حاصل جمع را برمیگرداند.

آرگمانهای اختیاری

همانطور که میدانید اگر تعداد آرگمانهای یک تابع n باشد ؛ تعداد پارامترهایی هم که به آن فرستاده می شود باید n باشد. اما گاهی ما میخواهیم این امکان را فراهم کنیم که کاربر تابع (کسی که از تابع استفاده میکند) بتواند یک یا چند آرگمان را مقدار دهی نکند. به چنین آرگمانهایی آرگمان اختیاری میگویند. آرگمان اختیاری آرگمانی است که در تعریف تابع برای آن مقدار پیش فرضی درنظر میگیریم و اگر کاربر به آن آرگمان مقدار ندهد، مقدار پیش فرض برای آن درنظر گرفته میشود. ساختار دستوری آن بشکل زیر است:

```
[Private|Public] Sub SubName (Optional arg As TypeName=value)

End Sub
```

کلمه کلیدی Optional به کامپایلر میگوید که این آرگمان اختیاری است و value مقدار پیش فرض آن را تعیین میکند. به عنوان مثال میخواهیم زیربرنامه FrooshBastani از کلاس بستنی را طوری تغییر دهیم که بتواند هر مقدار که کاربر زیربرنامه بخواهد، از بستنی ها کم کند. برای این کار زیربرنامه باید یک آرگمان ورودی داشته باشد، اما اگر ما آرگمان ورودی برای آن تعریف کنیم باید هرجا که این تابع فراخوانی شده است، یک پارامتر نیز به آن انتقال بدهد. از طرفی، بیشتر مواقع بستنیها پنجاه گرمی هستند. پس میتوانیم این زیربرنامه عضو (متد) را بصورت زیر تعریف کنیم :

```
    Public Sub FoorooshBastani(Optional ByVal arg As Integer = 50)
        MeghdarBastani -= arg
    End Sub
```

در اینجا آرگمان ورودی inp از نوع Integer و بصورت اختیاری (Optional) با مقدار پیشفرض 50 تعریف شده است. حال اگر در بخشی از برنامه این متد بصورت FoorooshBastani() صدا شود، مقدار پیشفرض در inp قرار میگیرد و برنامه 50 گرم بستنی از بستنیها کم میکند. اما اگر بصورت FoorooshBastani(400) صدا شود،

مقدار پیشفرض درنظر گرفته نمیشود و مقدار 400 در inp قرارگرفته و از میزان بستنیها 400گرم کم خواهد شد. به همین ترتیب میتوانیم متدی نیز برای دریافت پول از مشتری ایجاد کنیم:

```
Public Sub DaryaftPool(Optional ByVal arg As Integer = 50)
    MeghdarPool += arg
End Sub
```

حال میتوانیم در زیربرنامه کنترل کننده رویداد کلیک از دکمه "فروش"، عبارتهای

```
ValiAsr.MeghdarPool += 50
Farmanie.MeghdarPool += 50
TehranPars.MeghdarPool += 50
```

را بترتیب با فراخوانی متدهای معادل آنها جایگزین کنیم.

```
ValiAsr.DaryaftPool()
Farmanie.DaryaftPool()
TehranPars.DaryaftPool()
```

آنچه درزمان تعریف کردن آرگمانهای اختیاری باید به آن دقت کنید این است که هر آرگمان به شرطی میتواند اختیاری تعریف شود که آرگمان بعدی آن نیز اختیاری باشد.

```
Function Test (Optional ByVal a As integer=2, ByVal b As String) As Long

End Function
```

مثال بالا کار نخواهد کرد. چون a اختیاری تعریف شده است، در حالی که آرگمان بعد از آن بصورت اختیاری تعریف نشده است. اگر بخواهیم a اختیاری باشد یا باید b را هم اختیاری تعریف کنیم یا جای این دو آرگمان را عوض کنیم.

آرگمانهای مقداری و ارجاعی

تابحال درباره کلمه کلیدی ByVal که ویژوال بیسیک قبل از نام آرگمانها مینویسد توضیحی نداده بودیم. این کلمه کلیدی مخفف By Value است و نشان میدهد که مقدار پارامتر ورودی در آرگمان کپی میشود. برای مثال میخواهیم زیربرنامه ای بنویسیم که بتواند مقدار دو متغیر را با هم عوض کند :

```
Private Sub swap(ByVal x As Integer, ByVal y As Integer)
    Dim tmp As Integer
    tmp = x
    x = y
    y = tmp
End Sub
```

این زیربرنامه x و y (از نوع object که میتواند هر نوع داده ای را در خود جای دهد) را به عنوان آرگمانهای ورودی دریافت میکند و مقدار آنها را با هم عوض میکند. ظاهرا مشکلی وجود ندارد، پس میتوانیم برای صدا کردن آن از کد زیر استفاده کنیم:

```
Private Sub Form1_Load(…) Handles MyBase.Load
    Dim a As Integer, b As Integer
    a = 10
    b = 20
    swap(a, b)
    Label1.Text = "a=" & a.ToString & " b=" & b.ToString
End Sub
```

انتظار داریم مقدار a و b با هم عوض شده باشد، ولی درصورتی که این کد را اجرا کنید خواهید دید که مقادیر هیچ تغییری نکرده اند. وقتی اجرای برنامه به swap(a,b) میرسد، دو متغیر جدید در حافظه اصلی کامپیوتر تعریف میکند و نام آنها را x و y قرار داده، مقدار a و b را در آنها کپی میکند. پس وقتی زیربرنامه swap مقدار x و y را با هم عوض میکند، a و b هیچ تغییری نمیکنند. چون swap به این متغیرها دسترسی ندارد و فقط به یک کپی از آنها دسترسی دارد. این مکانیزم به منظور جلوگیری از اعمال تغییرات ناخواسته توسط زیربرنامه ها روی پارامترها درنظر گرفته شده است.

ولی راه حل چیست؟ جواب مساله swap در کلمه کلیدی ByRef است. یکی از قابلیتهای بسیار جالب در توابع و زیربرنامه ها، انتقال پارامتر با استفاده از ارجاع(Reference) است:

```
Private Sub swap(ByRef x As Integer, ByRef y As Integer)
    Dim tmp As Integer
    …
End Sub
```

در اینجا فقط کلمه کلیدی ByVal را با کلمه کلیدی ByRef عوض کرده ایم، و حاصل تابعی است که دو پارامتر ورودی میگیرد و مقدار آنها را با هم عوض میکند.
بطور غیر رسمی میتوان گفت ارجاع (Reference) یک متغیر، نام دیگری برای همان متغیر است. یعنی در اینجا وقتی اجرای برنامه به swap(a,b) میرسد، x و y را به عنوان نامهایی مستعار برای a و b درنظر میگیرد و وقتی زیربرنامه swap مقدار x و y را عوض میکند، مقدار a و b تعویض میشود.
آنچه درواقع اتفاق می افتد این است که آرگمانها (متغیرهای) ارجاعی (مانند x و y) چهار بایت حافظه اشغال میکنند (بدون توجه به نوع داده) که میتواند آدرس یک متغیر دیگر در حافظه را نگهداری کند. وقتی مقادیر a و b را به swap پاس میدهیم، آدرس آنها در حافظه اصلی در x و y کپی میشود. پس از آن وقتی x و y را تغییر بدهیم این تغییرات در آن بخشی از حافظه اعمال میشود که آدرسش در x ویا y ذخیره شده است.

شی و ارجاع

کمی از توابع فاصله بگیرید و تعریف کردن شی از روی یک کلاس را بخاطر بیاورید.

```
Dim test As Bastani
test = new Bastani
```

این کد برای همه ما آشنا است. گفته بودیم خط اول یک شی از نوع بستنی تعریف میکند و خط دوم برای آن حافظه میگیرد. اما دلیل آن را توضیح نداده بودیم. اشیاء در ویژوال بیسیک بصورت ارجاع (Reference) تعریف میشود. یعنی خط تعریف Dim test As Bastani فقط چهار بایت حافظه از حافظه اصلی میگیرد و آن را به عنوان یک ارجاع به یک شی از کلاس بستنی درنظر میگیرد. ولی این ارجاع هنوز شیی را که باید به آن اشاره کند نمیشناسد (شیبه یک کنترل تلویزیون بدون تلویزیون). خط دوم test=new Bastani یک شی از نوع بستنی ایجاد میکند (new Bastani) و آدرس آن را در شی test قرار میدهد. اکنون میتوانیم از شی test استفاده کنیم.
حال سوال این است که چه اتفاقی می افتد اگر یک داده ارجاعی (مانند یک شی) را بصورت ByVal به یک زیربرنامه بدهیم؟ پاسخ این است که تمام اعضای آن شی را میتوان در زیربرنامه تغییر داد اما خود آن را نمیتوان تغییر داد، به مثال زیر که در فرم اصلی برنامه بستنی فروشی (بطور موقت) نوشته شده است دقت کنید :

```
Private Sub EmptyBastani(ByVal b As bastani)
    b.MeghdarBastani = 0
End Sub
Private Sub Form1_Load(…) Handles MyBase.Load
    Dim a As bastani
    a = New bastani
    a.MeghdarBastani = 10
    EmptyBastani(a)
    Label1.Text = "bastani a=" & a.MeghdarBastani.ToString
    …
End Sub
```

اگرچه b یک کپی از a است اما از آنجا که خود a با استفاده از ارجاع به MeghdarBastani دسترسی داشته است، b هم با استفاده از ارجعاع میتواند مقدار آن را تغییر دهد. اما اگر کد زیربرنامه EmptyBastani را بصورت زیر بنویسیم :

```
Private Sub EmptyBastani(ByVal b As bastani)
    Dim m As New bastani
    m.MeghdarBastani = 0
    b = m
End Sub
```

مقدار a در زیربرنامه Form1_Load هیچ تغییری نخواهد کرد. چون فقط کپی a در حافظه تغییر کرده است. در نهایت اینکه کار با ارجاعها اگرچه دست ما را باز میکند ولی دقت بیشتری میطلبد.

باراضافی دادن به توابع و زیربرنامه ها (Overload)

باراضافی دادن به توابع یکی از برتری های بزرگ ویژوال بیسیک دات نت به ویژوال بیسیک 6 است. با استفاده از این قابلیت میتوانیم چند تابع هم اسم داشته باشیم. برای نمونه ای از توابع بار اضافی داده شده میتوانیم به تابع Show از کلاس MessageBox اشاره کنیم.

همانطور که میبینید این تابع 12 حالت مختلف دارد، اما تمام آنها فقط با یک نام قابل دسترسی هستند. این 12 حالت، 12 تابع مجزا و بار اضافی داده شده هستند.
اجازه بدهید مثالی از بستنی فروشی بزنیم. میخواهیم بتوانیم مقدار بستنی فروخته شده را بر حسب کیلوگرم و گرم نیز وارد کنیم. چون گاهی فروشگاه ها عمده فروشی دارند و برایشان سخت است که مقدار آن را بر حسب گرم وارد کنند، ترجیح میدهند آن را بر حسب کیلوگرم و گرم بیان کنند (10 کیلو و 530 گرم)
برای این کار میتوانیم یک زیربرنامه فروش بستنی دیگر (مثلا به اسم ForooshBastaniKilooi) درست کنیم :

```
    Public Sub ForooshBastaniKilooi(ByVal kilo As Integer, ByVal gram
As Integer)
        MeghdarBastani -= kilo * 1000 + gram
    End Sub
```

اگرچه این متد درست کار میکند، ولی ما بای دو نام مختلف را برای کاری که از نظر مفهومی یکی است بخاطر بسپاریم. با استفاده از باراضافی دادن به توابع میتوانیم نام این متد را نیز با متد قبلی یکی کنیم:

```
    Public Overloads Sub ForooshBastani(Optional ByVal inp As Integer =
50)
        MeghdarBastani -= inp
    End Sub

    Public Overloads Sub ForooshBastani(ByVal kilo As Integer, ByVal
gram As Integer)
        MeghdarBastani -= kilo * 1000 + gram
    End Sub
```

توجه داشته باشید که وقتی به چند تابع یا زیربرنامه بار اضافه میدهیم باید در تعداد یا نوع آرگمانهایشان متفاوت باشند. چون کامپایلر از روی همین تفاوت پی میبرد که باید کدام یک از توابع را اجرا کند. حال اگر متد ForooshBastani با یک پارامتر یا بدون پارامتر فراخوانی شود، تابع اولی اجرا خواهد شد. اما اگر دستور فراخوانی دو پارامتر داشته باشد، تابع دومی صدا خواهد شد.
کلمه کلیدی Overloads به کامپایلر میگوید که قصد بار اضافه دادن به این تابع را داریم. اما استفاده از این کلمه کلیدی اجباری نیست. به این معنی که میتوانیم این کلمه کلیدی را برای هیچ کدام از توابع هم اسم در کلاسمان ننویسیم یا برای همه آنها بنویسیم. توجه داشته باشید که نمیتوان برای یکی از توابع هم اسم از این کلمه کلیدی استفاده کرد و برای دیگری استفاده نکرد.

زیربرنامه ها و داده های Shared

داده های Shared
میدانیم که وقتی یک متغیر عضو برای یک کلاس تعریف می کنیم، هر یک از نمونه های آن کلاس متغیر مخصوص خودشان را دارند و مقدار آن متغیر برای هر یک از نمونه های کلاس می تواند متفاوت باشد. داده های Shared داده هایی هستند که به نمونه ای از کلاس وابسته نیستند. این داده ها بین تمام نمونه های کلاس مشترکند و همه آنها میتوانند به داده های Shared دسترسی داشته باشند. هر تغییری که یکی از نمونه های کلاس روی یک داده Shared بدهد، بدلیل مشترک بودن آن داده روی بقیه هم تاثیر میگذارد.
علاوه بر این به داده های Shared هم میتوان با استفاده از نام شی (همانطور که به داده های عضو دسترسی پیدا میکردیم) دسترسی داشت. هم با استفاده از نام خود شی:

```
Public Class A
    Public Shared var As Integer
End Class
Public Class frmMain
    …
    Private Sub frmMain_Load(…) Handles MyBase.Load
        Dim c As New A
        A.var = 10
        MessageBox.Show(c.var)
    End Sub
End Class
```

کلاس A یک داده عمومی Shared به نام var از نوع Integer دارد. خط اول یک نمونه (شی) از A ایجاد کرده است. خط دوم با استفاده از نام کلاس A.var به متغیر Shared دسترسی پیدا کرده و مقدار آنرا برابر 10 کرده است. خط دوم مقدار c.var را نمایش میدهد، که برابر 10 است.

زیربرنامه های Shared

زیربرنامه های Shared زیربرنامه هایی هستند که به نمونه خاصی از کلاس وابسته نیستند و برای صدا کردن آنها نیازی به ایجاد شی نیست. آنها را میتوان هم با استفاده از نام کلاس هم با استفاده از نام شی صدا کرد. توجه داشته باشید که زیربرنامه های Shared فقط میتوانند به داده های Shared دسترسی پیدا کنند و نمیتوانند به داده های عضو دسترسی داشته باشند.

```
Public Class A
    Public Shared var As Integer
    Public mVar As Integer
    Public Shared Sub test()
        ' mVar = 300 ' Syntax Error
        var = 1000
    End Sub
End Class
```

حال میتوانیم زیر برنامه Test را با استفاده از خط دستور A.Test صدا کنیم. یا با استفاده از c.Test که در اینجا c یک نمونه از کلاس A است. به همین ترتیب میتوان توابع و خاصیتهای shared نیز تعریف کرد.

پروژه برنامه نویسی

در ادامه کار بر روی پروژه کتابخانه، توابع برنامه کتابخانه را بررسی کنید و آرگمانهای لازم را تعریف کنید.

آرایه ها

یکی از مشکلاتی که در برنامه بستنی فروشی زنجیره ای آقای محمدی داریم، این است که بعضی از بخشهای برنامه را باید برای هرکدام از مغازه ها تکرار کنیم، برای مثال اکنون که سه مغازه داریم از کد زیر در رویداد SelectedIndexChanged از ComboBox برنامه استفاده کرده ایم:

```
Select Case cmbStores.Text
    Case "ValiAsr"
        lblPrice.Text = ValiAsr.MeghdarPool
        lblNoon.Text = ValiAsr.TedadNan
        lblGhashogh.Text = ValiAsr.TedadGhashogh
        lblLivan.Text = ValiAsr.TedadLivan
        lblICECream.Text = ValiAsr.MeghdarBastani
    Case "Farmanie"
        lblPrice.Text = Farmanie.MeghdarPool
        lblNoon.Text = Farmanie.TedadNan
        lblGhashogh.Text = Farmanie.TedadGhashogh
        lblLivan.Text = Farmanie.TedadLivan
        lblICECream.Text = Farmanie.MeghdarBastani
    Case "TehranPars"
        lblPrice.Text = TehranPars.MeghdarPool
        lblNoon.Text = TehranPars.TedadNan
        lblGhashogh.Text = TehranPars.TedadGhashogh
        lblLivan.Text = TehranPars.TedadLivan
        lblICECream.Text = TehranPars.MeghdarBastani
End Select
```

در دکمه‌ی "فروش" و منوی "ثبت" نیز وضعیت همین طور است. تصور کنید اگر یک بستنی فروشی دیگر به بستنی فروشی ها اضافه شود چه پیش خواهد آمد؟ اگر آقای محمدی بخواهد در همه شهرهای کشور حداقل یک شعبه داشته باشد چه بلایی سر برنامه ما خواهد آمد؟ این مشکلات ما را به سمت استفاده از آرایه ها راهنمایی میکنند.

تعریف آرایه

آرایه برای ذخیره کردن یک مجموعه از مقادیری استفاده میشود که نوع داده یکسانی دارند. با استفاده از آرایه میتوانیم ارجاعی به یک دنباله از متغیرها داشته باشیم که همگی به یک نام شناخته میشوند ولی اندیسهای مجزایی دارند. این قابلیت به ما کمک میکند که کد برنامه خودمان را ساده تر و کوتاه تر کنیم، چون میتوانیم در بسیاری موارد تعداد خطهای زیادی از کد را با یک حلقه جایگزین کنیم.

ساختار دستوری تعریف یک آرایه (در ساده ترین حالت) بصورت زیر است :

Dim *arrayName*(*UpperBound*) As *DataType*

تعریف آرایه تا حدود زیادی شبیه تعریف کردن متغیر است، با این تفاوت که پس از نام متغیر درون پرانتز حد بالایی (کران بالای) آرایه را تعریف میکنیم. کران بالا (UpperBound) بزرگترین اندیس آرایه را مشخص میکند. کوچکترین اندیس آرایه نیز همیشه صفر است.

```
Dim m(10) As Integer
```

کد بالا یک آرایه به نام m از نوع Integer تعریف میکند که میتواند 11 عدد صحیح (Integer) را در خود نگه دارد، اندیس اولین عدد 0 و اندیس آخرین عدد 10 است.

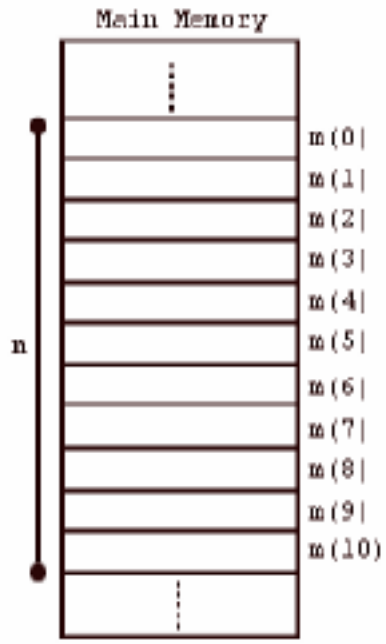
دسترسی به اعضای آرایه

به هریک از اعضای آرایه با استفاده از ساختار زیر میتوان دسترسی پیدا کرد :

arrayName(*Index*)

این کد معادل خانه index+1ام از آرایه arrayName است، برای مثال، کد زیر معادل خانه سوم (چون اولین خانه اندیس صفر دارد) از آرایه است.

m(2)



هر خانه از آرایه مانند یک متغیر معمولی است و میتوانیم تمام کارهایی که با یک متغیر معمولی انجام میدادیم با آن انجام دهیم :

m(2) = 100

Dim x As Integer = m(3)

علاوه بر این وقتی میخواهیم عملی را روی تمام خانه های یک آرایه انجام دهیم میتوانیم از حلقه ها استفاده کنیم، در حالی که برای انجام همان عمل روی تعدادی متغیر منفرد باید از کدهای مجزا برای هریک از متغیر ها استفاده میکردیم. مثال زیر به مقدار هر یک از خانه های آرایه بیست واحد اضافه میکند.

```
Dim m(10) As Integer
Dim i As Integer
For i = 0 To 10
    m(i) += 20 ' m(i) = m(i) + 20
Next
```

در اجرای اول حلقه، مقدار i برابر با صفر میشود و m(i) برابر با m(0) . پس برابر با خانه اول آرایه میشود و مقدارش 20 واحد اضافه میشود. در اجرای دوم حلقه m(i) برابر با m(1) میشود و الی آخر.

مقدار دهی اولیه به آرایه

از آنجا که یه آرایه مجموعه ای از مقادیر است، باید مقداردهی اولیه به آن با مقداردهی به متغیرهای معمولی تفاوت داشته باشد. ساختار دستوری مقداردهی اولیه بصورت زیر است :

Dim *arrayName*() As *DataType* = { *value1* , *value2* , *value3* , ... , *valueN* }

همانطور که از ساختار بالا مشخص است، برای مقدار دهی اولیه به یک آرایه باید کران بالای آن آرایه را در تعریف خالی بگذاریم و پس از DataType علامت مساوی را قرار داده و در نهایت در {} مقدارهای اولیه آرایه را که با کاما از هم تفکیک شده اند بیاوریم. اندازه آرایه از روی تعداد مقدارهای اولیه تعیین میشود. برای مثال :

```
Dim m() As Integer = {5, 20, 53, 12, 3, 1, 3, 4, 5, 6, 9}
```

یا معادل آن :

```
Dim m As Integer() = {5, 20, 53, 12, 3, 1, 3, 4, 5, 6, 9}
```

این دستور آرایه ای یازده عضوی (اندیس صفر تا 10) ایجاد میکند که مقدار اولیه خانه اول آن 5 و خانه آخر 9 میباشد. این روش، خلاصه شده روش زیر است :

```
Dim m() As Integer = New Integer(10) {5, 20, 53, 12, 3, 1, 3, 4, 5, 6,
9}
```

توجه داشته باشید که در صورتی که از این روش استفاده کنید میتوانید سایز آرایه را در پرانتز دوم تعیین کنید، باید مقدار اولیه ها هم به همان تعداد باشند.

آرایه به عنوان شی

در ویژوال بیسیک دات نت آرایه ها نیز شی هستند. بنابراین هر آرایه (همانطور که درباره اشیاء گفته بودیم) یک نوع داده ارجاعی (reference) است. بنابراین زمانی که نام یک آرایه را مساوی نام آرایه دیگری قرار دهیم، فقط ارجاع (reference) آن کپی میشود. به عبارت دیگر، پس از این عمل هر دو نام به یک آرایه اشاره میکنند:

```
Dim m As Integer() = {5, 20, 53, 12, 3, 1, 3, 4, 5, 6, 9}
Dim x As Integer()
x = m
x(4) = 100
MessageBox.Show(m(4))
```

وقتی اجرای برنامه به خط x=m میرسد، آدرسی را که مرجع m است در x نیز کپی میشود و از آن پس m و x به یک آرایه اشاره خواهند کرد. بنابراین پس از اجرای این دستورات خواهیم دید که m(4) برابر 100 شده است.

اگر بخواهیم مقادیر آرایه x با آرایه m برابر شود باید تک تک اعضای m را در x کپی کنیم :

```
Dim m As Integer() = {5, 20, 53, 12, 3, 1, 3, 4, 5, 6, 9}
Dim x(10) As Integer
For i As Integer = 0 To 10
    x(i) = m(i)
Next
```

شی آرایه تعدادی خاصیت و متد نیز دارد که به مرور با آنها آشنا خواهیم شد. مهمترین خاصیت (property) آن خاصیت Length است. این خاصیت طول (اندازه) آرایه را برمیگرداند و فقط خواندنی است. طول آرایه همیشه یکی بیشتر از کران بالای آرایه است (چون اولین اندیس صفر است). در مثال بالا، حلقه for را میتوان بشکل زیر تصحیح کرد:

```
For i As Integer = 0 To m.Length - 1
```

آرایه ای از اشیاء

تعریف کردن یک آرایه از اشیاء نیز مانند تعریف کردن آرایه از نوع داده های پایه است. با این تفاوت که وقتی آرایه ای از اشیاء تعریف میکنیم عملا آرایه ای از ارجاعها (reference) تعریف کرده ایم و هنوز حافظه ای برای آنها گرفته نشده است. بنابراین باید قبل از استفاده از هر یک از خانه های آرایه، با استفاده از کلمه کلیدی new برای آن حافظه لازم را بگیریم:

```
Dim forooshgah(10) As bastani
forooshgah(0) = New bastani
forooshgah(0).MeghdarBastani = 10
```

خط اول یک آرایه یازده عضوی از کلاس bastani ایجاد میکند و خط دوم برای خانه اول آرایه حافظه میگیرد. توجه کنید که وقتی آرایه ای از اشیاء ایجاد میکنیم نمیتوانیم از کلمه کلیدی New در خط تعریف آرایه استفاده کنیم (بر خلاف اشیاء مفرد).

```
'Dim forooshgah(10) As New bastani 'error
```

اگر بخواهیم برای همه اعضای آرایه حافظه بگیریم، میتوانیم از حلقه for استفاده کنیم :

```
Dim forooshgah(10) As bastani
For i As Integer = 0 To forooshgah.Length - 1
    forooshgah(i) = New bastani
Next
```

حال میخواهیم برنامه بستنی فروشی را با استفاده از آرایه ها بهینه کنیم، ولی یک مشکل وجود دارد. مشکل این است که در تعریف آرایه‌ای از اشیاء نمی‌توان از کلمه کلیدی WithEvents استفاده کرد. برای حل این مشکل دو راه داریم، یا اشیاء را بصورت منفرد تعریف کنیم و ارجاع آنها را به آرایه بدهیم:

```
    Private WithEvents ValiAsr As New bastani
    Private WithEvents Farmanie As New bastani
    Private WithEvents TehranPars As New bastani
    Private Forooshgah(2) As bastani
    …

    Private Sub frmMain_Load(…) Handles MyBase.Load
        ValiAsr.Title = "ValiAsr"
        Farmanie.Title = "Farmanie"
```

```
        TehranPars.Title = "TehranPars"
        Forooshgah(0) = ValiAsr
        Forooshgah(1) = Farmanie
        Forooshgah(2) = TehranPars
        …
    End Sub
```

به این ترتیب میتوانیم از رویدادهای کلاس بستنی استفاده کنیم. علاوه بر آن می توانیم تا حدودی از قابلیتهای آرایه نیز استفاده کنیم. اما چرا فقط تا حدودی؟ به این دلیل که برای اضافه کردن هر فروشگاه جدید باید یک متغیر برای آن اضافه کنیم، در تعریف زیربرنامه های کنترل کننده رویدادها نام آن را اضافه کنیم و یکی از خانه های آرایه را به آن نسبت بدهیم. اما زیربرنامه های کنترل کننده رویدادهایی که در ابتدای فصل ذکر کردیم در حد قابل قبولی ساده میشوند.

با توجه به اینکه تمام مشکل ما از کنار هم قرار دادن آرایه ای از اشیاء با رویدادهای آن اشیاء است، راه دیگر این است که از رویدادهای بستنی استفاده نکنیم. در اینصورت برای فهمیدن اینکه بستنی تمام شده است یا خیر، باید از یک دستور شرطی استفاده کنیم (و همچنین برای بقیه خاصیتها) .

اینکه کدام راه را انتخاب کنیم بستگی به شرایط مساله دارد، مثلا در اینجا بستگی به برنامه آقای محمدی برای آینده دارد. اگر قصد ندارد هر هفته یک بستنی فروشی اضافه کند میتوانیم از روش اول استفاده کنیم و اگرنه بهتر است از روش دوم استفاده کنیم. ما در اینجا از روش اول استفاده میکنیم اما بهتر است برای تمرین شما روش دوم را نیز پیاده سازی کنید.

کد رویداد کلیک از شی "فروش" با استفاده از آرایه در زیر آمده است :

```
    Private Sub btnForoosh_Click(…) Handles btnForoosh.Click
        For i As Integer = 0 To Forooshgah.Length - 1
            If cmbStores.Text = Forooshgah(i).Name Then
                Forooshgah(i).FoorooshBastani()
                Forooshgah(i).FoorooshGhashogh()
                Forooshgah(i).FoorooshLivan()
                Forooshgah(i).FoorooshNan()
                Forooshgah(i).DaryaftPool()
                lblNoon.Text = Forooshgah(i).TedadNan
                lblPrice.Text = Forooshgah(i).MeghdarPool
                lblICECream.Text = Forooshgah(i).MeghdarBastani
                lblLivan.Text = Forooshgah(i).TedadLivan
                lblGhashogh.Text = Forooshgah(i).TedadGhashogh
            End If
        Next
    End Sub
```

همانطور که می‌بینید این کد بسیار خلاصه تر از کد قبلی است. حلقه for به تعداد فروشگاه ها تکرار میشود و اگر نام فروشگاه با متن ComboBox برابر بود، عملیات فروش را برای آن خانه از آرایه انجام میدهد. به دلیل اینکه هر خانه از آرایه ارجاعی به یکی از فروشگاه ها است، هر تغییری در هر خانه آرایه، باعث تغییر فروشگاه نظیر آن نیز میشود.

اگر دقت کنید متوجه میشود شرطی که تضمین میکرد فقط وقتی امکان فروش وجود دارد، بستنی فروخته شود در اینجا ذکر نشده است. در صورتی که تمرین مربوط به CanSell از فصل قبل را حل کرده باشید میتوانید اینجا از شرط زیر استفاده کنید. (فرض کردیم نام متد یا خاصیتی که تعیین میکند این شی میتواند بستنی فروشد یا خیر CanSell است.)

```
If cmbStores.Text = Forooshgah(i).Name And Forooshgah(i).CallSell then
    …
End If
```

تغییر کد رویدادهای SelectedIndexChanges و رویداد کلیک منوی "ثبت" نیز مانند تغییر کد دکمه "فروش" است. کد تغییر یافته در CD ضمیمه موجود است.

آرایه به عنوان آرگمان تابع

از آنجا که آرایه نیز یک شی است، پس وضعیت آرایه به عنوان آرگمان یک تابع یا زیربرنامه مانند همان وضعیتی است که در فصل قبل بررسی کردیم. به این معنی که اگر آرایه ByVal به تابع یا زیربرنامه پاس داده شود اعضای آن قابل تغییرند(مثال زیر). اما خودش قابل تغییر نیست.

```
    Private Function Test(ByVal a() As Integer)
        a(1) = 0
```

```
End Function
```

توجه داشته باشید که وقتی میخواهیم یک آرگمان آرایه ای تعریف کنیم، نباید کران بالای آن را تعیین کنیم.

آرایه های چند بعدی

آرایه هایی که تا بحال برسی کردیم، آرایه های یک بعدی بودند. در ویژوال بیسیک میتوانیم آرایه های چند بعدی (حداکثر 32 بعد) نیز تعریف کنیم. یک آرایه دو بعدی بصورت زیر تعریف میشود:

Dim *arrayName*(*size1*, *size2*) As *TypeName*

برای مثال :

Dim m(4,3) As Short

	0	1	2	3
0	m(0,0)	m(0,1)	m(0,2)	m(0,3)
1	m(1,0)	m(1,1)	m(1,2)	m(1,3)
2	m(2,0)	m(2,1)	m(2,2)	m(2,3)
3	m(3,0)	m(3,1)	m(3,2)	m(3,3)
4	m(4,0)	m(4,1)	m(4,2)	m(4,3)

آرایه دو بعدی m را میتوان به عنوان 5 آرایه از آرایه های 4 عضوی درنظر گرفت. مقدار دهی به این آرایه نیز شبیه مقدار دهی به آرایه یک بعدی است :

m(2,1) = 10

برای مقدار دهی اولیه به آرایه های چند بعدی از آکولادهای { } تو در تو استفاده میکنیم:

```
Dim m(,) As Integer = {{2, 3, 4}, {3, 4, 5}}
```

در اینجا m بصورت m(1,2) تعریف خواهد شد. به عبارت دیگر دو آرایه سه عضوی.

آرایه های دو بعدی بیشتر برای ذخیره کردن ماتریسها استفاده میشوند. به عنوان مثال، تابعی که دو ماتریس را جمع کرده و حاصل جمع را برگرداند در زیر آمده است :

```
Private Function MatrixSum(ByVal m1(,) As Integer, _
    ByVal m2(,) As Integer, ByVal l1 As Integer, _
    ByVal l2 As Integer) As Integer(,)

    Dim tmp(l1, l2) As Integer
    For i As Integer = 0 To l1
        For j As Integer = 0 To l2
            tmp(i, j) = m1(i, j) + m2(i, j)
        Next
    Next
    Return tmp
End Function
```

آرگمانهای این تابع دو آرایه دوبعدی (که کران بالای آنها نباید تعریف شود) و دو متغیر صحیح (integer) هستند. متغیرهای صحیح برای تعیین ابعاد ماتریس یا به عبارت دیگر کران بالاهای دو آرایه است. برای دسترسی به تمام اعضای یک آرایه دو بعدی از دو حلقه for تو در تو استفاده میکنیم که یکی اندیس اولی (سطر ماتریس) و دیگری اندیس دوم (ستون ماتریس) را تغییر میدهند. به همین ترتیب برای دسترسی به همه اعضای یک ماتریس N بعدی به N ماتریس تو در تو نیاز داریم.

حاصل جمع دو ماتریس در آرایه tmp ذخیره میشود و در نهایت دستور return آرایه tmp را به زیربرنامه صدا کننده این تابع انتقال میدهد.

دستور Redim

یکی از مشکلات آرایه، ثابت بودن طول آن است. این سوال پیش می آید که اگر ندانیم اندازه آرایه باید چقدر باشد (برای مثال بخواهیم تعداد فروشگاه ها را از خود آقای محمدی بپرسیم) باید چکار کنیم؟ یا اگر وقتی برنامه نویسی ما تمام شده آقای محمدی یک فروشگاه جدید تاسیس کند ما باید چکار کنیم؟ دستور Redim به ما این امکان را میدهد که طول آرایه را تغییر دهیم. این دستور هم میتواند روی آرایه های یک بعدی کار کند هم آرایه های چند بعدی.

ساختار دستور Redim بشکل زیر است :

Redim [preserve] *arrayName* (*size1*, *size2*, ... , *sizeN*)

برای مثال آرایه m یک آرایه به طول 10 است، که میخواهیم طول آنر را به 15 تغییر بدهیم :

```
Dim m(9) As Integer
ReDim m(14)
```

میتوانیم طول آرایه m را در خط تعریف آن مشخص نکنیم و پس از تعریف (هرجا که طول آن برایمان مشخص شد، مثلا زمانی که آن را از کاربر پرسیدیم.) به آن مقدار بدهیم.

```
Dim m() As Integer
…
ReDim m(14)
```

برای آرایه های چند بعدی هم روال کار به همین صورت است.

Redim چگونه کار میکند؟

وقتی که کامپایلر به دستور ReDim میرسد، آرایه جدیدی با ابعاد داده شده ایجاد میکند، به آن مقدار اولیه پیشفرض نوع داده آرایه را میدهد(برای integer مقدار صفر، برای string مقدار "" و...)، سپس ارجاع آرایه قبلی را به آرایه جدید میدهد و آرایه قبلی را بطور کامل از حافظه پاک میکند. این باعث می شود تمام اطلاعات قبلی از حافظه پاک شود.

اگر بخواهیم مقادیری که در آرایه هستند، در آرایه جدید حفظ شوند باید از کلمه کلیدی preserve استفاده کنیم. این کلمه کلیدی به کامپایلر میگوید که پس از ایجاد و مقدار اولیه دهی به آرایه، مقادیر آرایه قبلی را در آرایه جدید کپی کند.

توجه : در صورتی که اندازه آرایه را، بجای زیادکردن کم کنید؛ تعدادی از خانه های آرایه را از دست خواهید داد.

توجه : دستور redim نمیتواند نوع داده یک آرایه را عوض کند.

شی ArrayList

اگرچه با ReDim میتوانیم طول آرایه را تغییر دهیم ولی اگر طول آرایه ما بطور پیوسته تغییر کند، کنترل کردن آن کار هزینه بری خواهد شد. در ویژوال بیسیک دات نت کلاسی به نام ArrayList تعبیه شده است که آرایه ای با طول متغیر است. طول این آرایه بسته به آنچه ما نیاز داشته باشیم تغییر میکند و لازم نیست خودمان را درگیر این کنیم که طول آرایه چقدر باید باشد.

تعریف کردن یک ArrayList مانند تعریف کردن هر شی دیگری است :

```
Dim a As New ArrayList
```

متدها و پروپرتی های مهم آن به شرح زیر است :

Capacity : ظرفیت فعلی ArrayList

Count : تعداد عناصری که در ArrayList قرار دارند. count همیشه کوچکتر یا مساوی capacity است.

Add : یک شی به انتهای آرایه اضافه میکند و در صورتی که count با capacity برابر باشد، capacity را دوبرابر میکند؛ آرایه جدیدی در حافظه ایجاد میکند و مقادیر آرایه قبلی را در آن کپی میکند.

Remove : یک شی به عنوان پارامتر میگیرد و آن را در ArrayList جستجو میکند و اولین تکرار آن را حذف میکند.

RemoveAt : یک اندیس میگیرد و شیی را که در آن اندیس است پاک میکند.

Insert : یک شی و یک اندیس میگیرد و شی را در آن اندیس خاص درج میکند.

Item : یک اندیس میگیرد و شیی را که در آن اندیس قرار دارد برمیگرداند.

خاصیت (property) آرایه ای

اگر به خاصیت Item از شی ArrayList دقت کنید خواهید دید که این خاصیت مانند یک آرایه عمل میکند. ما هم میتوانیم چنین خاصیتی تعریف کنیم:

```
Private mArrayProp(20) As String
Public Property ArrayProp(ByVal index As Integer) As String
    Get
        Return mArrayProp(index)
    End Get
```

```
    Set(ByVal Value As String)
        mArrayProp(index) = Value
    End Set
End Property
```

آنچه در پرانتز بعد از نام خاصیت آمده است آرگمان ورودی خاصیت است (مانند آنچه برای توابع داریم)؛ ما از این آرگمان ورودی برای اینکه یک خاصیت آرایه ای ایجاد کنیم استفاده کرده ایم.
توجه : آرگمانهای یک خاصیت باید ByVal باشند.
اکنون با استفاده از این خاصیت میتوانیم به آرایه mArrayProp بطور غیر مستقیم دسترسی داشته باشیم. این دسترسی غیر مستقیم باعث میشود کاربر کلاس نتواند اندازه آرایه را با استفاده از دستور ReDim تغییر دهد.

شی تایمر Timer

شی تایمر یکی از جالبترین ابزارهای دات نت است. این ابزار برای تکرار کردن یک عمل در فاصله زمانی های مشخص (Interval) بکار میرود. مثلا برای اینکه هر ثانیه یک بار یک عدد روی فرم برنامه نوشته شود. ما برای اینکه کار با این ابزار را یاد بگیریم یک انیمیشن ساده درست خواهیم کرد.

ایجاد تصاویر متحرک

یک فرم به نام frmAnimation به برنامه بستنی درست کنید و دستورات مربوط به نمایش دادن آن فرم را در یک منو در فرم اصلی برنامه بستنی فروشی بنویسید. انیمیشن ما از تعدادی تصویر تشکیل شده که با یک فاصله زمانی مشخص با هم تعویض میشوند. پس ابتدا سه تصویر آماده کنید (برای مثال یک بستنی در حالتهای مختلف یا تصاویر متوالی از راه رفتن یک آدم.) و آنها را در چند PictureBox با نامهای picAnim0 تا picAnim2 قرار بدهید و خاصیت visible آنها را false کنید. سپس PictureBox دیگری با نام picAnimation روی فرم قرار بدهید. برای ایجاد یک انیمیشن باید تصاویری ک در picAnim0 تا picAnim2 هست را به ترتیب در picAnimation نشان بدهیم.

کاری که باید بکنیم این است که ابتدا تصویری که در picAnim0 قرار دارد نشان بدهیم، پس از مدت مشخصی تصویری که در picAnim1 قرار دارد را نشان بدهیم و ... اولین سوال این است که از کجا بفهمیم کدام تصویر باید نمایش داده شود؟ یک پاسخ میتواند کد زیر باشد :

```
Dim x As Integer = 0
Select Case x
    Case 0
        picAnimation.Image = picAnim0.Image
    Case 1
        picAnimation.Image = picAnim1.Image
    Case 2
        picAnimation.Image = picAnim2.Image
End Select
x += 1
```

در اینجا با استفاده از Select Case و کمک گرفتن از یک متغیر، تشخیص میدهیم که کدام تصویر باید در picAnimation نمایش داده شود. در ابتدا x تعریف شده و مقدار صفر میگیرد، پس تصویر اول در picAnimation نمایش داده میشود، سپس یکی به مقدار x اضافه میشود. اما کار زیربرنامه تمام شده است پس در این مرحله دیگر select case اجرا نمیشود که تصمیم بگیرد تصویر عوض شود. شاید فکر کنید اگر یک بار دیگر روی دکمه کلیک کنیم تصویر بعدی نمایش داده شود، چون مقدار فعلی x یک است. اما اینطور نیست. اگر برنامه را اجرا کنید متوجه میشوید که فقط تصویر اول نمایش داده میشود.

طول عمر متغیر

پیش از این مفهوم میدان دید متغیر را بررسی کردیم و متوجه شدیم که با کلمات کلیدی public و private و نیز با تعریف کردن متغیر در scope های مختلف میتوانیم میدان دید یک متغیر را تغییر دهیم. مفهوم دیگری که باید بررسی کنیم مفهوم طول عمر متغیر است. طول عمر متغیر میگوید که متغیر ما تا چه زمانی در حافظه وجود دارد و از چه زمانی مرده و دیگر وجود ندارد. طول عمر نیز مانند میدان دید هم به نوع تعریف و هم به محل تعریف متغیر بستگی داد.

متغیرهای عضوی که در یک کلاس تعریف میشوند (fieldها) با هر شیی که ایجاد میشود متولد میشوند و تا زمانی که شی وجود داشته باشد، وجود دارند. پس متغیرهای هر کدام از اشیاء کلاس بستنی تا زمانی وجود دارند که شی مربوط به آنها وجود داشته باشد. از طرفی ما خود این اشیاء را به عنوان فیلدهای کلاس فرم اصلی برنامه تعریف کرده ایم، که از روی آن فقط یک شی داریم و آن هم شی_فرم اصلی برنامه است. فرم اصلی برنامه از اول تا آخر برنامه وجود دارد، پس (در این حالت خاص) متغیرهایی که در آن تعریف میشوند عمرشان به اندازه عمر برنامه است.

متغیرهای Shared یک کلاس عمرشان با عمر برنامه برابر است (بدون اینکه به تعریف شدن یا نشدن شی از روی کلاس بستگی داشته باشد.) .

اما وضعیت توابع با کلاسها متفاوت است، آنچه گفتیم درباره فیلدها بود و همانطور که میدانید فیلد به متغیرهای عضو کلاس (متغیرهایی که داخل کلاس و بیرون توابع عضو تعریف میشوند) گفته میشود. وقتی یک متغیر را با استفاده از کلمه کلیدی dim در یک تابع تعریف میکنیم، عمر آن متغیر تا زمانی است که تابع درحال اجرا است. یعنی به محض تمام شدن یک بار اجرای تابع، تمام متغیرهایی که تعریف کرده است از حافظه پاک میشوند و اگر بار دیگر تابع اجرا شود، متغیرها دوباره ایجاد میشوند و مقدار اولیه میگیرند.

به همین دلیل است که برنامه ما فقط عکس اول را نشان میدهد. هربار که به x یک واحد اضافه میشود، عمر آن به پایان میرسد و بار دیگر که تابع صدا میشود متغیر مقدار صفر میگیرد.
چاره چیست؟ برای حل کردن این مشکل دو راه داریم، اولی این است که متغیر را بجای اینکه در یک تابع تعریف کنیم، به عنوان داده عضو تعریف کنیم. این کار احتمال دستکاری شدن مقدار متغیر توسط دیگر توابع کلاس را زیاد میکند. راه دیگر این است که با استفاده از کلمه کلیدی static به کامپایلر بگوییم که ما به این متغیر خیلی علاقه داریم و نمیخواهیم شاهد مرگش باشیم!
کلمه کلیدی static به کامپایلر میگوید که طول عمر متغیر ما را به اندازه طول عمر شیی که این تابع در آن تعریف شده است زیاد کند. البته اگر تابعی که در آن متغیر Static را تعریف میکنیم یک تابع Shared باشد، طول عمر متغیر برابر طول عمر برنامه میشود.
کلمه کلیدی Dim در برنامه قبلی را به static تغییر دهید:

```
Static x As Integer = 0
…
```

حال بار دیگر برنامه را اجرا کنید. با هر بار کلیک روی دکمه، یکی از تصاویر نمایش داده میشود. اما وقتی به تصویر آخر میرسد متوقف میشود، برای اینکه پس از نمایش تصویر آخر باز هم تصویر اول را نمایش بدهد و این چرخه ادامه پیدا کند میتوانیم وقتی x بزرگتر از 2 شد، آن را صفر کنیم. همین کار را بدون استفاده از دستور شرطی و با استفاده از عملگر mod نیز میتوان انجام داد.

```
…
End Select
x = (x + 1) Mod 3
```

در اینجا از باقیمانده تقسیم x+1 بر 3 استفاده کرده ایم که همیشه عددی بین 0 و 2 است. پس وقتی x=2 باشد و این خط اجرا شود، x+1 برابر 3 میشود و باقیمانده تقسیم صفر میشود، پس عکس اولی نمایش داده خواهد شد.
تنها مشکل جدی برنامه ما این است که کاربر باید برای نمایش تصویر بعدی روی دکمه کلیک کند، چقدر خوب بود اگر رویدادی داشتیم که هر چند ثانیه یک بار اجرا بشود. شی تایمر دقیقا برای همین کار ساخته شده است. روی شی Timer در toolbox دبل کلیک کنید تا به فرم اضافه شود. این نیز مانند menuها است و روی فرم نمایش داده نمیشود. نام Timer را به tmrAnimation تغییر دهید، سپس خاصیت enabled آن را true کنید. در نهایت روی شی تایمر دبل کلیک کنید تا تابع handeler مربوط به رویداد Tick از شی تایمر ایجاد شود. حال کدی که در دکمه نوشته بودیم بطور کامل کپی کنید و در tmrAnimation_Tick بچسبانید. انیمیشن شما حاضر است! فقط دکمه F5 را بزنید.

خاصیت Interval

خاصیت Interval تعیین میکند که هر چند هزارم ثانیه یک بار رویداد Tick برای تایمر اتفاق بیفتد. به عبارت دیگر رویداد tick از شی تایمر هر Interval هزارم ثانیه یک بار اتفاق می افتد و کدی که در تابع مربوط به آن نوشته شده است اجرا میشود.
توجه : رویداد Tick در صورتی اتفاق می افتد که منابع سیستم آزاد باشند و اجرای قبلی تابع مربوط به این رویداد تمام شده باشد. یعنی اگر Interval برابر 100 باشد، به این معنی است که سعی میکند هر 100 هزارم ثانیه یک بار اجرا شود اما در صورت در دسترس نبودن منابع سیستم امکان دارد چند ثانیه اصلا اجرا نشود!

آرایه ای از pictureboxها

همانطور که اشیای Bastani را در یک آرایه قراردادیم، میتوانیم pictureboxها را نیز در یک آرایه قراردهیم تا برنامه بهینه شود و قابلیت توسعه بیشتری داشته باشد. میتوانیم آرایه ای از PictureBox ایجاد کنیم و هر خانه از آرایه را برابر یکی از picAnimها قرار دهیم:

```
Dim ArrayPic(2) As PictureBox
Private Sub Form1_Load(…) Handles MyBase.Load
    ArrayPic(0) = picAnim0
    ArrayPic(1) = picAnim1
    ArrayPic(2) = picAnim2
End Sub
Private Sub tmrAnimation_Tick(…) Handles tmrAnimation.Tick
    Static x As Integer = 0
    picAnimation.Image = ArrayPic(x).Image
    x = (x + 1) Mod 3
End Sub
```

با توجه به اینکه فقط ارجاع picAnimها در ArrayPic کپی میشـود، بـا ایجـاد ایـن آرایـه فقـط 12 (3*4) بایـت حافظه اضافه از کامپیوتر گرفته میشود. بنابراین این روش حافظه هرز زیادی ایجاد نمیکند.

در نهایت اینکه اگر در نوشتن برنامه درون یک تایمر با مشکل مواجه شدید، بخاطر بیاورید کـه رویـداد Tick از شـی تایمر مانند رویداد click از دکمه ای است که بطور متناوب روی آن کلیک میشود.

پروژه برنامه نویسـی:
برنامه یادآور(Reminder) : برنامه ای بنویسید که لیست کارهایی که کاربر باید انجام دهد با زمان انجام آنها را از کاربر بگیرد. سپس هرگاه که زمان انجام کاری رسید به کاربر اطلاع بدهد.

مختصات و رویدادهای موس و کیبرد

در این فصل میخواهیم یک بازی پرتاب بستنی بنویسیم. هدف ما از نوشتن این بازی آشنایی با مختصات، رویدادهای کیبرد و رویدادهای موس است. پیش از این با رویداد کلیک آشنا شده ایم، در این فصل رویدادهای دیگری از موس را خواهیم آموخت.

بازی پرتاب بستنی

کار روزانه برای آقای محمدی کسالت آور است، گاهی باید ساعتهای متمادی پشت میز بنشیند و به تصویر مونیتور خیره شود بدون اینکه کار مفیدی برای انجام دادن داشته باشد. برای اینکه وقت آقای محمدی پر شود ما یک بازی برای او درست میکنیم که وقتی بیکار میشود بستنی به صورت خودش بکوبد!

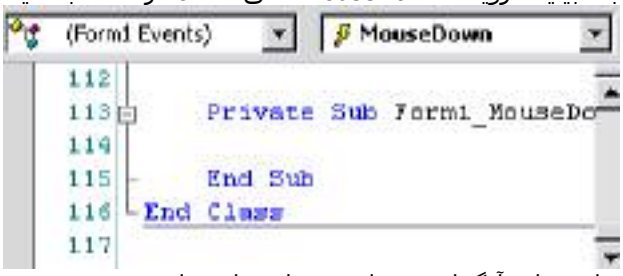
مساله بطور واضحتر به شرح زیر است:

برنامه ای که کاربر باید با استفاده از موس روی تصویری که بصورت تصادفی حرکت میکند کلیک کند. هر بار که کاربر کلیک میکند یک تصویر بستنی به نقطه ای که کاربر روی آن کلیک کرده اضافه میشود (چه روی هدف باشد چه نباشد).

برای شروع کار یک PictureBox به فرم اضافه کنید و خاصیت image آن را به دلخواه خودتان تنظیم کنید و نام آن را به picTarget تغییر دهید. PictureBox دیگری نیز به فرم اضافه کنید و نام آن را به picBastani تغییر دهید و یک عکس بستنی هم در آن قرار دهید (روی CD ضمیمه میتوانید این عکس را پیدا کنید.) هربار که کاربر کلیک میکند (چه روی فرم چه روی picTarget) باید تصویر بستنی در آن نقطه نمایش داده شود. پس مساله اول این است که چطور بفهمیم کاربر روی چه نقطه ای کلیک کرده؟

رویدادهای MouseDown و MouseUp

رویداد کلیک مختصات نقطه ای که کاربر روی آن کلیک کرده به ما نمیگوید. برای فهمیدن مختصات کلیک باید از دیگر رویدادهای موس استفاده کنیم. رویداد MouseDown وقتی اتفاق می افتد که دکمه موس (یکی از دکمه ها) فشار داده شود و پایین برود. رویداد MouseUp وقتی اتفاق می افتد که دکمه موس رها شده و بالا بیاید. رویداد MouseDown شی Form1 را انتخاب کنید.



خاصیتهای آرگمان e در این رویداد عبارتند از :

e.X : فاصله موس از سمت چپ شی (در اینجا فرم) را به ما میدهد.

e.Y : فاصله موس از بالای شی (در اینجا فرم) را به ما میدهد.

e.Button : میگوید کدام دکمه موس فشار داده شده است.

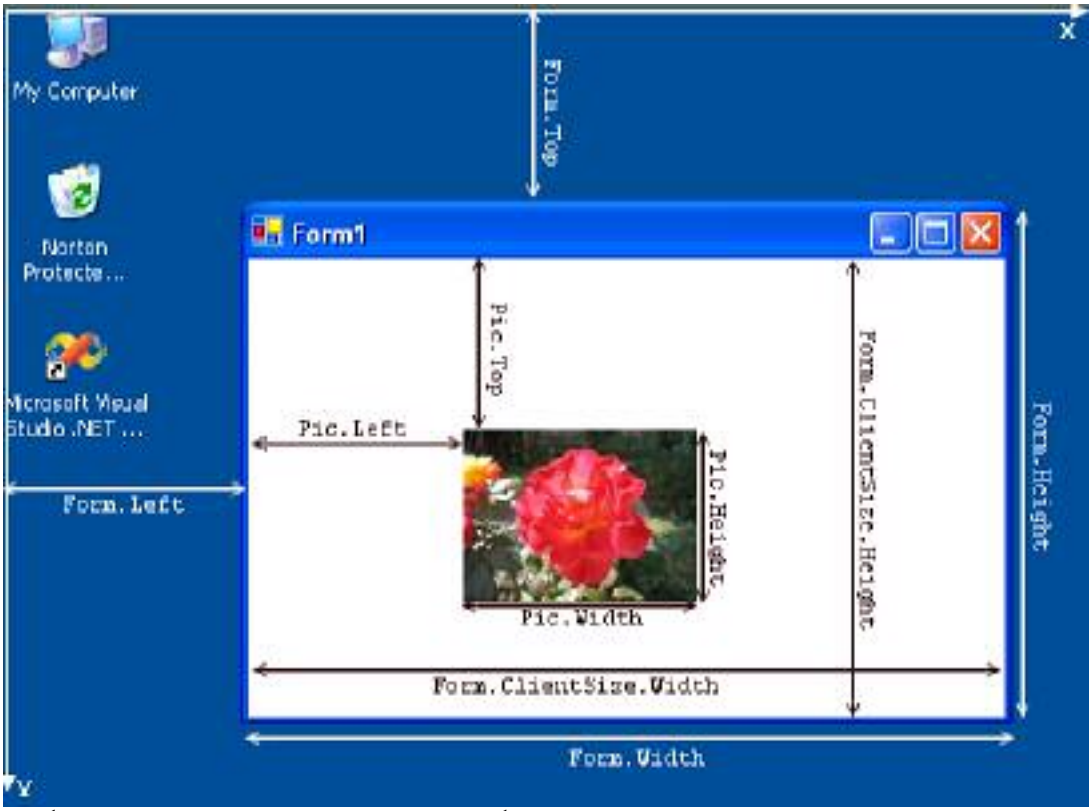
حالا میدانیم که کاربر روی چه نقطه ای کلیک کرده است، بیایید برای آزمایش شی picTarget را به آن نقطه منتقل کنیم:

```
    Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles MyBase.MouseDown
        picTarget.Left = e.X
        picTarget.Top = e.Y
    End Sub
```

مختصات

در کامپیوتر مبداء مختصات گوشه بالا سمت چپ مونیتور است. پس مقدار X از چپ به راست و مقدار Y از بالا به پایین زیاد میشوند و خلاف این جهتها کاهش پیدا میکنند. در ویژوال بیسیک محل قرارگیری هر کنترل با استفاده از دو خاصیت Left و Top بیان میشود. این دو خاصیت بیانگر مختصات نقطه سمت چپ بالای کنترل، نسبت به کنترل پدرشان هستند. یعنی Form1.Left فاصله Form1 از سمت چپ کنترل پدرش (Desktop ویندوز) را بیان میکند. و picTarget.Left فاصله picTarget از سمت چپ کنترل پدرش (Form1) را بیان میکند. Top نیز همین فاصله ها را از بالا نشان میدهد(به شکل مراجعه کنید.) تا اینجا مختصات یک نقطه از کنترل را پیدا کرده ایم، از آنجا که ابزارهای ویژوال بیسیک همه مستطیلی شکل هستند، باید

بتوانیم نقطه سمت راست و پایین آنها را نیز پیدا کنیم تا بدانیم دقیقا کجا قرار گرفته اند. برای این منظور از خاصیتهای Width و Height استفاده میکنیم که اولی عرض کنترل و دومی ارتفاع کنترل را به ما میگویند:



کنترل فرم علاوه بر این مشخصه ها، دو مشخصه مفید دیگر هم دارد، یکی ClientSize.Height و دیگری ClientSize.Width که اولی ارتفاع محیط کاری فرم و دومی عرض محیط کاری فرم را به ما میدهد. همانطور که در شکل میبینید، این دو مقدار کمتر از عرض و ارتفاع فرم هستند. با استفاده از این دو خاصیت میتوانیم فاصله یک کنترل را نسبت به پایین فرم:

Form.ClientSize.Height – Obj.Top – Obj.Height

یا نسبت به سمت راست فرم :

Form.ClientSize. Width – Obj.Left – Obj. Width

بدست آوریم.

ایجاد کنترل جدید

گفتیم که میخواهیم هرجا که کاربر کلیک کرد یک تصویر بستنی ظاهر شود، این کار را به دو بخش تقسیم میکنیم. بخش اول وقتی روی هدف کلیک نکرده باشد و دومی وقتی که کاربر روی هدف کلیک کرده باشد. برای حل کردن بخش اول ابتدا خاصیت Visible از picBastani را false میکنیم و سپس آرایه ای از PictureBox ها ایجاد میکنیم و هربار که کاربر روی فرم کلیک کرد، یک PictureBox به آن نقطه منتقل میکنیم:

```
Dim picFired(9) As PictureBox
```

ولی برای هیچ کدام از این اشیاء حافظه گرفته نشده است. میتوانیم در Form_Load برای همه آنها حافظه بگیریم، یا اینکه برای هرکدام دقیقا همان وقتی حافظه بگیریم که لازم دارند. یعنی هر بار که کاربر روی فرم کلیک کرد برای یکی از اینها حافظه بگیریم. از آنجا که ما بطور پویا این اشیاء را به فرم اضافه کرده ایم باید خواصی از قبیل Left، Top، Width، Height را خودمان مقدار دهی کنیم.

```
Private Sub Form1_MouseDown(…) Handles MyBase.MouseDown
    Static iPicIndex As Integer = 0
    picFired(iPicIndex) = New PictureBox
    picFired(iPicIndex).Image = picBastani.Image
    picFired(iPicIndex).Left = e.X
```

```
        picFired(iPicIndex).Top = e.Y
        picFired(iPicIndex).Height = picBastani.Height
        picFired(iPicIndex).Width = picBastani.Width
        iPicIndex = (iPicIndex + 1) Mod 10
    End Sub
```

در خط اول یک متغیر static تعریف کرده ایم که اندیس picFired بعدی را نگه دارد. خط دوم یک PictureBox جدید ایجاد میکند و خطهای بعد از آن مقداردهی های اولیه را انجام میدهند و شی تازه ایجاد شده را به محل کلیک موس منتقل میکنند. حال برنامه را اجرا کنید و روی فرم کلیک کنید.
هیچ PictureBox ای به فرم اضافه نمیشود! دلیلش چیست؟ دلیل این است که ما نگفته ایم این اشیا درون کدام کنترل قرار دارند، به عبارت دیگر پدر این کنترلها را به کامپایلر معرفی نکرده ایم.

شی Me

برای دسترسی به کنترل جاری از کلمه کلیدی Me استفاده میکنیم. این کلمه کلیدی به کنترلی اشاره میکند که کدِ در حال اجرا ، در آن قرار دارد. برای مثال اگر از این کلمه کلیدی در Form1 استفاده کنید به فرمی که در حال اجرا است اشاره میکند. دقت کنید که برای دسترسی به این شی از Form1 نمیتوانیم استفاده کنیم، چون Form1 نام کلاس است نه نام شی.
در برنامه پرتاب بستنی نیز به کنترل جاری(Me) احتیاج داریم، به این دلیل که میخواهیم آن را به عنوان پدر (Parent) کنترلهایی که ایجاد کرده ایم معرفی کنیم:

```
        ...
        picFired(iPicIndex).Width = picBastani.Width
        picFired(iPicIndex).Parent = Me
        iPicIndex = (iPicIndex + 1) Mod 10
```

حال برنامه را اجرا کنید. و چندین بار روی فرم کلیک کنید. اتفاق عجیبی می افتد! ما فقط 10 pictureBox تعریف کرده بودیم در حالی که هر چند بار که کلیک کنیم PictureBox های جدید روی صفحه ظاهر میشوند. اگر مساله مصرف شدن منابع سیستم نبود حتما از این اتفاق خیلی هم خوشحال میشدیم، ولی هر کدام از این pictureboxها مقداری از حافظه اصلی کامپیوتر را اشغال میکنند. علت چیست؟ بار دیگر به خط زیر نگاه کنید :

```
picFired(iPicIndex) = New PictureBox
```

ما در این خط ارجاع (Refrence) شی PicFired(iPicIndex) را به یک شی جدید تغییر داده ایم ولی چه بر سر آن بخش از حافظه می آید که این شی قبلا به آن ارجاع داشت؟ پاسخ این است که هیچ! آن بخش از حافظه به امان خدا رها میشود. برای همین است که هر بار که کلیک میکنیم یک شی جدید ایجاد میشود. البته خیلی هم نگران آن حافظه هرز نباشید. از آنجا که parent همه آنها فرم برنامه است، وقتی فرم برنامه از بین برود، آنها هم از حافظه خارج میشوند. اما به هرحال برای اینکه جلوی ایجاد بی رویه این اشیاء را بگیریم یک شرط قبل از آن دستور اضافه میکنیم که فقط در صورتی شی جدید ایجاد کند که شی برابر Nothing است یعنی به هیچ جا اشاره نمیکند :

```
If picFired(iPicIndex) Is Nothing Then picFired(iPicIndex) = New
PictureBox
```

بلوک With

وقتی میخواهیم بطور متوالی تعدادی از خواص یا متدهای یکی شی را تغییر دهیم یا فراخوانی کنیم (مانند کد بالا) بلوک with میتواند کد ما را تا حدودی خلاصه تر کند و از تکرار بی دلیل نام شی جلوگیری کند. ساختار این بلوک بشکل زیر است :

With *objName*
  statements
End With

در بخش statements اگر از عملگر دسترسی به اجزاء (.) بدون نام شی استفاده کنیم، آن عملگر به عنوان عملگر دسترسی به اجزاء objName درنظر گرفته میشود، به این ترتیب کدی که پیش از این نوشتیم به این صورت اصلاح میشود :

```
        Static iPicIndex As Integer = 0
        If picFired(iPicIndex) Is Nothing Then picFired(iPicIndex) =
New PictureBox
        With picFired(iPicIndex)
            .Image = picBastani.Image
            .Left = e.X
            .Top = e.Y
```

```
            .Height = picBastani.Height
            .Width = picBastani.Width
            .Parent = Me
        End With
        iPicIndex = (iPicIndex + 1) Mod 10
```

فقط یک مشکل کوچک باقی مانده؛ وقتی روی فرم کلیک میکنیم گوشه سمت چپ تصویر بستنی در نقطه ای که کلیک کرده ایم قرار میگیرد در حالی که ما ترجیح میدهیم وسط تصویر در آنجا قرار بگیرد. پس باید به اندازه نصف عرض تصویر، تصویر را بطرف چپ ببریم، از آنجا که این تغییر مکان خلاف جهت محور X است باید به اندازه نصف عرض تصویر از X موس کم کنیم، برای Y نیز شرایط به همین صورت است.

```
.Left = e.X - picBastani.Width / 2
.Top = e.Y - picBastani.Height / 2
```

ویرایش تصویر

بخش دوم مساله این است که وقتی کاربر روی هدف کلیک میکند، تصویر بستنی روی تصویر هدف نشان داده شود. اگر برای این کار از روش ایجاد کردن اشیاء استفاده کنیم مشکلی پیش خواهد آمد که در مرحله قبل آن را نادیده گرفتیم. وقتی روی این اشیاء جدید کلیک شود رویداد کلیک تصویر هدف (یا فرم) فراخوانی نخواهد شد. در مورد فرم این مساله برای ما اهمیت نداشت ولی در مورد تصویر هدف این برایمان مهم است. بنابراین، بجای اینکه اشیاء جدیدی ایجاد کنیم که Parent آنها picTarget است؛ تصویر (Image) هدف را تغییر میدهیم و یک بستنی به آن اضافه میکنیم.

کلاس Graphics

کلاس Graphics امکانات بسیار زیادی برای کار با تصویر در اختیار ما قرار میدهد. البته ما قصد نداریم به شرح کامل این کلاس بپردازیم و فقط در حد رسم یک تصویر (Image) روی یک تصویر دیگر از این کلاس استفاده خواهیم کرد.

میتوانیم شی از نوع Graphics را نیز مانند سایر اشیاء تعریف کنیم ولی گرفتن حافظه برای یک نمونه از Graphics با اشیایی که تا الان دیدیم متفاوت است. برای این کلاس نمیتوانیم از کلمه کلیدی New استفاده کنیم.

```
Dim g As Graphics
```

در چنین مواردی باید راه دیگری برای ایجاد شی جدید پیدا کنیم. کلاس Graphics یک تابع Shared دارد که میتواند یک شی Graphics از روی یک Image ایجاد کند. میتوانیم با استفاده از این تابع و عملگر مساوی، شی g را مقدار دهی کنیم :

```
g = Graphics.FromImage(picTarget.Image)
```

اکنون هر تغییری که روی شی g بدهیم روی شی picTarget.Image اعمال میشود. با شی گرافیک میتوانیم خط، دایره و سایر اشکال هندسی را رسم کنیم. علاوه بر این میتوانیم یک Image دیگر را روی Image فعلی رسم کنیم.

```
        Dim g As Graphics
        g = Graphics.FromImage(picTarget.Image)
        With picBastani.Image
            Dim x As Integer = e.X - (.Width / 2)
            Dim y As Integer = e.Y - (.Height / 2)
            g.DrawImage(picBastani.Image, x, y, .Width, .Height)
        End With
```

تابع عضو (متد) DrawImage یک تصویر (Image) را از ورودی دریافت میکند و آن را روی شکل فعلی در مختصات x و y (پارامترهای سوم و چهارم) رسم میکند. ابعاد تصویر میتوانند در آرگمانهای چهارم و پنجم بیایند(البته این متد به 30 شکل مختلف تعریف شده است(overload))

در اینجا از widthو height شی picBastani.Image بجای همین خصوصیتها از شی picBastani استفاده کردیم که طول و عرض تصویر را بجای طول و عرض شی picBastani به ما میدهد.

کار این بخش تمام شده است اما تا وقتی که picTarget را Refresh نکنیم تغییرات جدید نشان داده نخواهند شد (مگر اینکه به دلیل دیگری VB مجبور شود آن را دوباره رسم کند.).

```
            …
            g.DrawImage(picBastani.Image, x, y, .Width, .Height)
        End With
        picTarget.Refresh()
```

هدفی که فرار میکند

تنها بخش باقی مانده از این برنامه حرکت تصادفی هدف است. با معلوماتی که داریم و با کمی دقت میتوانیم به راحتی این بخش را انجام دهیم. باید هر چند لحظه یک بار برای picTarget یک x و y تصادفی انتخاب کنیم فقط باید این X و Y را طوری تعیین کنیم که هدف از فرم خارج نشود. اول یک تایمر روی فرم قرار دهید و در رویداد Tick آن کد زیر را بنویسید. اگر فرمول اعداد تصادفی را بخاطر ندارید به فصل 6 مراجعه کنید.

```
Randomize()
picTarget.Left = Int(Rnd() * (ClientSize.Width - picTarget.Width))
picTarget.Top = Int(Rnd() * (ClientSize.Height - picTarget.Height))
```

این کد، برای picTarget.Left عددی بین صفر و ClientSize.Width - picTarget.Width انتخاب میکند. که به این معنی است که این شی هیچ وقت از سمت راست صفحه خارج نخواهد شد. وضع picTarget.Top نیز دقیقا به همین صورت است.

پرتاب بستنی دو نفره

در بازی دو نفره، یک نفر هدف را با کیبرد حرکت میدهد و دیگری آن را با موس به آن بستنی پرت میکند. برای نوشتن این برنامه باید بیشتر با رویدادهای کیبرد آشنا شویم. مهمترین رویدادهای کیبرد رویدادهای KeyDown ، KeyPress وKeyUp هستند. وقتی یک کلید فشار داده میشود ابتدا رویداد KeyDown یک بار فراخوانی میشود، سپس مادامی که کلید فشار داده میشود رویداد KeyPress فراخوانی میشود و در نهایت با رها شدن کلید، رویداد KeyUp فعال میشود. رویداد KeyPress فقط برای کلیدهای کاراکتری اتفاق می افتد و در مورد کلیدهای غیر کاراکتری (مانند کلیدهای کمکی و کلیدهای جهتی) فقط رویدادهای KeyDown و KeyUp فراخوانی میشوند. وقتی یک کلید غیر کاراکتری فشرده میشود مادامی که آن کلید فشار داده میشود رویداد KeyDown فراخوانی میشود.

حرکت دادن هدف

برای حرکت دادن هدف (Target) باید در کدام رویداد برنامه بنویسیم؟ از آنجا که میخواهیم از کلیدهای جهتی (Arrow Keys) استفاده کنیم. بنابراین رویداد KeyPress هرگز فراخوانی نخواهد شد. رویداد KeyUp هم تا زمانی که کلید رها نشده باشد فراخوانی نمیشود، پس در صورتی که از آن استفاده کنیم کاربر باید برای حرکت دادن هدف کلید را بطور متناوب فشار بدهد و رها کند. بنابراین مناسب ترین گزینه رویداد KeyDown است. اما رویداد KeyDown از کدام شی؟ بخاطر داشته باشید که رویدادهای کیبرد فقط به آن شیی میرسند که Focus دارد. برای مثال اگر شما دو TextBox روی فرم داشته باشید وقتی کرسر(cursor) روی یکی از آنها است رویدادهای کیبرد فقط برای آن TextBox فراخوانی میشوند. یا اگر یک دکمه روی فرم داشته باشید، Focus در اختیار آن دکمه خواهد بود و رویدادهای کیبرد برای فرم اجرا نخواهند شد.
اما بدلیل اینکه ما در اینجا فقط چند PictureBox روی صفحه داریم و PictureBoxها نمیتوانند Focus بگیرند.

```
    Private Sub Form1_KeyDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles MyBase.KeyDown

    End Sub
```

با استفاده از این رویداد میتوانیم کدی بنویسیم که وقتی کلیدی فشرده شد اجرا شود، اما از کجا بفهمیم کدام کلید فشرده شده است؟ پاسخ در آرگمان e است. هر کدام از کلیدهای کیبرد کدی دارند که به آن KeyCode میگوییم. e.KeyCode این کد را به ما برمیگرداند:

```
    Private Sub Form1_KeyDown(…) Handles MyBase.KeyDown
        If e.KeyCode = Keys.Left Then
            MessageBox.Show("Arrow Left")
        End If
    End Sub
```

لازم نیست کد همه کلیدها را حفظ کنید. کد آنها در داده شمارشی keys موجود است فقط باید نام کلید مورد نظر خود را بدانید. کد بالا وقتی کلید جهتی چپ فشار داده شود یک MessageBox با متن Arrow Left نمایش میدهد. ما میخواستیم کاری کنیم که وقتی این کلید فشار داده شد هدف به سمت چپ حرکت کند. پس باید وقتی این کلید فشار داده شد از picTarget.Left چند واحد کم کنیم :

```
    Private Sub Form1_KeyDown() Handles MyBase.KeyDown
        If e.KeyCode = Keys.Left Then
            picTarget.Left -= 10
        End If
    End Sub
```

برای اینکه هدف از سمت چپ صفحه خارج نشود میتوانیم شرط را کمی تغییر دهیم :

```
If e.KeyCode = Keys.Left And picTarget.Left >= 10 Then
    picTarget.Left -= 10
End If
```

برای کلیدهای جهتی دیگر نیز باید کد مشابهی نوشته شود.

Alt، Shift و Control

با استفاده از سه خاصیت Alt، Shift و Control در e میتوانیم از وضعیت این سه کلید مطلع شویم، اگر مقدار این متغیرهای Boolean برابر True باشد یعنی آن کلید فشرده شده است و اگر False باید فشرده نشده است.

Handled

در حالت عادی یک رویداد ممکن است توسط توابع مختلفی handle شود. این توابع میتوانند توابع پیشفرض خود زبان برنامه نویسی یا ویندوز باشند. برای مثال اگر شما کلید ترکیبی Alt+F4 را فشار دهید پس از اینکه رویداد KeyDown و KeyUp فراخوانی شدند، تابع دیگری مسوولیت کنترل کردن این رویداد را بر عهده گرفته و فرم شما را میبندند. با تنظیم کردن Handled به True اعلام میکنیم که ما تمام کارهایی که برای این رویداد لازم بود را انجام داده ایم و هیچ نیازی به توابع پیشفرض و یا هر تابع دیگری نیست که این رویداد را کنترل کنند:

```
Private Sub Form1_KeyDown(…) Handles MyBase.KeyDown
    …
    If e.Alt = True And e.KeyCode = Keys.F4 Then e.Handled = True
End Sub
```

در صورتی که کد بالا را به KeyDown فرم اضافه کنید، اگر کلید F4 در حالی که Alt برابر True است (فشارداده شده است.) زده شود، Handled مقدار True میگیرد. بنابراین آنچه بطور پیشفرض باید انجام میشد دیگر انجام نخواهد شد. به عبارت دیگر، این کد کاری میکند که فرم ما با Alt+F4 بسته نشود (البته در صورتی که Focus در اختیار form باشد.)

رویداد KeyPress

گفتیم که رویداد KeyPress فقط برای کلیدهایی کاراکتری فراخوانی میشوند. آرگمان e در تابع Handler این رویداد ساده تر از آرگمان e در تابع Handler رویداد KeyDown است. این آرگمان فقط دو خاصیت Handled و KeyChar دارد. خاصیت Handled دقیقا همان است که در مورد KeyDown و KeyUp گفتیم. و خاصیت KeyChar کاراکتر دکمه ای که فشار داده شده است به ما میدهد. برای فهمیدن اینکه کدام کلید کاراکتری فشرده شده است در حالت کلی مشکلی نداریم، چون میتوانیم KeyChar را با کاراکتر مورد نظر خودمان مقایسه کنیم ولی برای بعضی کاراکترهای خاص مشکل داریم.

کاراکترهای خاص

در مورد کاراکترهای خاص از ثابتهایی استفاده میکنیم که در ویژوال بیسیک تعبیه شده است:

نام کاراکتر	نام ثابت آن	معادل است با
BackSpace	vbBack	Chr(8)
کاراکترهای سرخط و خط بعد	vbCrLf	Chr(13)+Chr(10)
کاراکتر سر خط	vbCr	Chr(13)
کاراکتر خط بعدی	vbLf	Chr(10)
کاراکتر خط جدید	vbCrLf	Chr(13)+Chr(10)
کاراکتر Tab	vbTab	Chr(9)

برای مثال، کد زیر تشخصی میدهد که کاراکتر BackSpace زده شده است یا خیر :

```
Private Sub Form1_KeyPress(…) Handles MyBase.KeyPress
    If e.KeyChar = vbBack Then MessageBox.Show("backspace")
End Sub
```

کارگاه :

هم در برنامه بستنی فروش و هم در برنامه های دیگر مواردی پیش آمده است که بخواهیم با استفاده از یک TextBox یک عدد بگیریم. با توجه به اینکه کاربر میتواند در TextBox ما کاراکترهای غیر عددی نیز وارد کند. ممکن است وقتی متن TextBox را به عدد تبدیل میکنیم خطایی رخ بدهد. قطعه کدی بنویسید که یک TextBox را تبدیل به TextBox ای کند که فقط عدد میگیرد.

راهنمایی :

در رویداد KeyPress از e.KeyChar و e.Handled استفاده کنید.

پاسخ کارگاه :

```
    Private Sub TextBox1_KeyPress(…) Handles txtNoon.KeyPress
        If Not (e.KeyChar >= "0" And e.KeyChar <= "9" Or e.KeyChar =
vbBack) Then
            e.Handled = True
        End If
    End Sub
```

اگر e.Handled برابر true شود، کاراکتری که تایپ شده، در TextBox نوشته نخواهد شد.

پروژه برنامه نویسی

برنامه ای بنویسید که یک PictureBox که در آن تصویر متحرکی نمایش داده میشود، مدام در صفحه حرکت کند و هر وقت با یکی از دیوارهای فرم برخورد کرد با زاویه ای یکسان ولی خلاف جهت بازگردد. (مانند بازتابش نور در برخورد با آینه تخت.)

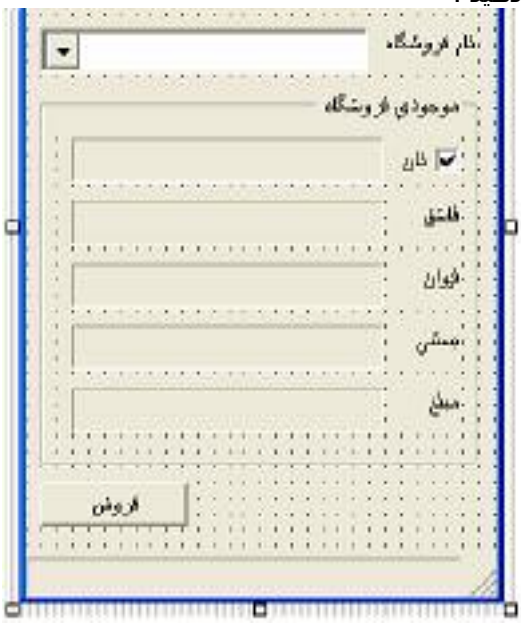
تکمیل و بهبود بستنی

در این فصل رابط کاربر نهایی برنامه بستنی فروشی را خواهیم ساخت . برای این هدف از چند کنترل جدید و کلاس ArrayList (که قبلا با آن آشنا شدیم) استفاده می کنیم. برنامه ای که در این فصل می نویسیم در فصل پایگاه داده ها نیز دوباره مورد استفاده قرار می گیرد.
کنترل های جدیدی که با آنها آشنا خواهیم شد عبارتند از:
ImageList ، Toolbar ، ToolTip و ContextMenu

برای شروع ساختن نسخه نهایی نرم افزار فروش بستنی یک پروژه جدید ایجاد کنید و مشخصات فرم اصلی آن را مانند جدول زیر تنظیم کنید :

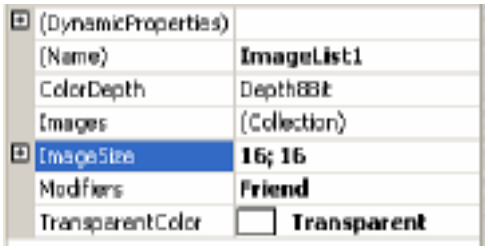
نام خاصیت	مقدار
Name	frmMain
Font	Arial;9pt
FormBorderStyle	FixedSingle
RightToLeft	Yes
StartPosition	CenterScreen
Text	نرم افزار فروش بستنی

مانند فرمی که پیش از این ساخته بودیم کنترلهایی که در شکل زیر دیده می شوند را بر روی فرم قرار دهید .



ImageList

ImageList مانند انبار تصاویر عمل می کند. یعنی میتواند تعدادی تصویر را در خود نگه دارد تا هر کنترلی که می خواهد تصویری نمایش بدهد، از یک یا چند تصویر ذخیره شده در ImageList استفاده کند. یکی از پر کاربرد ترین کنترلهایی که از ImageList استفاده می کند؛ Toolbar است. که در ادامه با آن آشنا خواهید شد.
ImageList از جمعه کنترل هایی است که شکل ویژوال ندارد بنابراین مانند MainMenu در قسمت کنترل های غیر ویژوال قرار می گیرد.



مهمترین خاصیت ImageList، خاصیت Images است. بوسیله این خاصیت می توانید عکسهای مورد نظر را در این کنترل قرار دهید.

برای اضافه کردن تصاویر کافی است بر روی دکمه Add کلیک و سپس نام فایل را انتخاب کنید.

ContextMenu

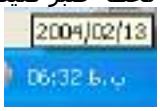
ContextMenu شباهت بسیار زیادی به MainMenu دارد و مانند آن به عنوان منو در قسمت های مختلف برنامه استفاده می شود ، ContextMenu نیز مانند MainMenu میتواند تعدادی آیتم داشته باشد، که هرکدام از این آیتمها یک رویداد کلیک دارند و میتوان برای آنها برنامه نویسی کرد. اما تفاوت ContextMenu با MainMenu در این است که MainMenu بالای فرم نمایش داده می شود. اما ContextMenu می تواند روی اکثر کنترلها نمایش داده شود. بعضی از کنترلهای یک ContextMenu پیشفرض دارند. برای مثال وقتی روی TextBox کلید راست را میزنید منویی شامل آیتمهای Cut، Copy و... نمایش داده میشود.
میخواهیم یک ContextMenu برای دکمه "فروش" ایجاد کنیم(مانند شکل زیر). برای این کار ابتدا باید یک ContextMenu به فرم اضافه کنیم و آیتمهایی را که در شکل زیر نشان داده شده است، در آن اضافه می کنیم. سپس برای اینکه این ContextMenu وقتی کاربر روی دکمه "فروش" کلید راست موس را می زند نمایش داده شود؛ باید خاصیت ContextMenu از شی دکمه "فروش" را برابر با نام ContextMenu1 قرار بدهیم.

در هنگام اجرا ، هرگاه کاربر بر روی دکمه فروش کلید سمت راست موس را بزند منوی مورد نظر نمایش داده می شود .

نکته : ContextMenu در کنترلهایی مانند Toolbar نیز قابل استفاده است.

ToolTip

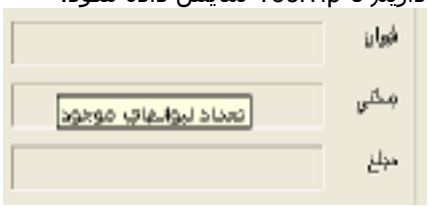
کنترل دیگری مورد استفاده قرار می گیرد ToolTip است. موس خود را بر روی ساعت ویندوز ببرید و چند لحظه صبر کنید. بعد از مدتی خواهید دید که پنجره کوچکی باز شده و تاریخ امروز را می نویسد.



به کنترلی که این کار را انجام می دهد ToolTip می گویند. هر گاه برای کنترلی بر روی فرم ، ToolTip تنظیم شود هنگامی که موس چند لحظه روی آن کنترل باقی بماند؛ متنی که با استفاده از Tooltip برای آن کنترل تنظیم کرده اید نمایش داده می شود. در نسخه های قبلی ویژوال بیسیک، کنترلهای خاصیتی به نام ToolTip داشتند که تعیین می کرد چه متنی در Tooltip آنها نمایش داده شود. ولی در ویژوال بیسیک دات نت این خاصیت تبدیل به یک کنترل مجزا شده.
یک کنترل ToolTip (از ToolBox) به فرم اضافه کنید. کنترل ToolTip متدی به نام SetToolTip دارد. پارامتر اول این متد کنترلی است که می خواهیم به آن ToolTip نسبت بدهیم و پارامتر دوم آن متن ToolTip است. کد زیر را در زیربرنامه Load فرم بنویسید.

```
ToolTip1.SetToolTip(lblNoon, "موجود نانهاي تعداد")
ToolTip1.SetToolTip(lblGhashogh, "موجود قاشقهاي تعداد")
ToolTip1.SetToolTip(lblLivan, "موجود ليوانهاي تعداد")
ToolTip1.SetToolTip(lblIceCream, "موجود بستني مقدار")
ToolTip1.SetToolTip(lblPrice, "فرويشگاه در موجودي ميزان")
ToolTip1.SetToolTip(chkNoon, "ليواني بستني همراه نان فروش براي")
```

حال برنامه را اجرا کرده و موس را روی برچسب هایی(Label) که به آنها ToolTip نسبت داده ایم، نگه می داریم تا ToolTip نمایش داده شود.

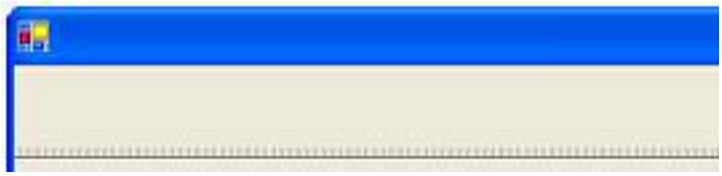


Toolbar

در اکثر برنامه هایی که در ویندوز استفاده می کنید نواری زیر ِ قسمت منو قرار دارد که دکمه هایی با شکل های گوناگون بر روی آن جای دارد. هر کدام از این دکمه ها عملیات مخصوص به خود را انجام می دهد که

البته اکثر آنها در منوی اصلی برنامه نیز قرار دارند. در شکل زیر Toolbar ِ نرم افـزار Word 2003 را مشـاهده می کنید.

یک کنترل ِ Toolbar بر روی فرم قرار دهید. این کنترل در بالای فرم جای می گیرد.

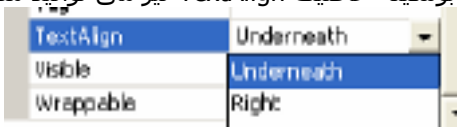


خواصـی کـه در ایـن کنتـرل بیشـتر اسـتفاده مـی شـوند عبارتنـد از Buttons ، Image list، TextAlign و Wrappable .

در خاصیت Buttons می توانید دکمه هایی را که بر روی این کنترل قرار می گیرند را تعریف کنید.

مقدار ImageList نام کنترل ImageList ی است که بر روی فرم قرار دارد.

بوسیله خاصیت TextAlign نیز می توانید مکان ِ قرار گرفتن نوشته هر کدام از دکمه ها را تعیین کنید.

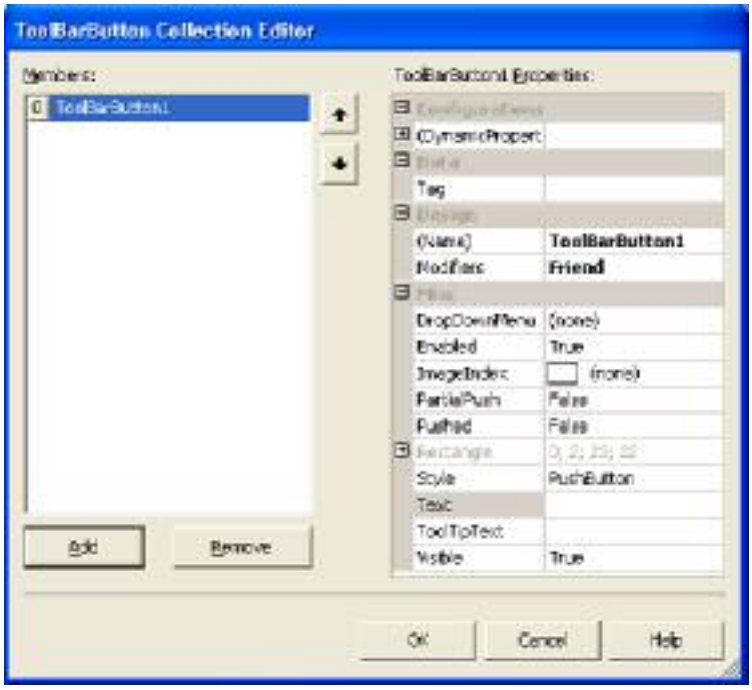


اگر مقدار خاصیت Wrappable برابر True باشد هنگامی که عرض فرم کمتر از عـرض تمـام دکمـه هـا باشـد ارتفاع Toolbar اضافه شده(چند خطی می شود) و دکمه هایی که دیده نمی شوند به خـط پـایینی منتقـل می شوند .

Buttons

بر روی خاصیت Buttons کلیک کنید تا فرم زیر نمایش داده شود .

بر روی دکمه Add کلیک کنید.



هر بار که بر روی دکمه Add کلیک کنید یک دکمه جدید بر روی نوار Toolbar ساخته خواهد شد. مشخصات هر کدام از دکمه ها را بعد از Add کردن در بخش سمت راست می توانیم تغییر دهیم.
متاسفانه شی Toolbar برای دکمه هایش رویداد کلیک مجزا ندارد و برای همه آنها یک رویداد فراخوانی می شود. برای اینکه بتوانیم بفهمیم روی کدام دکمه کلیک شده است، ابتدا به خاصیت Tag هر دکمه عبارتی نسبت میدهیم و سپس در رویداد ButtonClick از Toolbar با استفادا از یک Select Case کلید زده شده را تشخیص می دهیم. در این فیلد می توانیم هر عبارتی که علاقه مند بودیم بنویسیم مثلاً برای دکمه ای که قرار است یک فروشگاه اضافه کند، عبارت "Add" مناسب بنظر می رسد .
DropDownMenu : نام ContextMenu ی است که برای این کنترل در نظر گرفته شده.
ImageIndex : شماره عکسی است که در ImageList برای این دکمه ساخته شده است .
Style : نوع دکمه را مشخص می کند. برای اینکه بتوانید از ContextMenu استفاده کنید می بایست مقدار این خاصیت را برابر DropDownButton قرار دهید.
Text : متنی است که برای این دکمه به نمایش در خواهد آمد.
ToolTipText : متنی است که به عنوان ToolTip برای دکمه در نظر گرفته اید.

بعد از فشردن کلید OK دکمه ها به شکل کامل بر روی Toolbar در فرم نمایش داده می شود.

مشخصات کامل دکمه ها و عکسهای ImageList در سی دی ِ ضمیمه کتاب موجود است.

همانطور که پیش از این گفتیم، کنترل Toolbar برای همه دکمه هایش تنها یک رویداد ButtonClick دارد:

```
Private Sub ToolBar1_ButtonClick(ByVal sender As System.Object, ByVal e
As       System.Windows.Forms.ToolBarButtonClickEventArgs)       Handles
ToolBar1.ButtonClick
```

شی ِ e در این زیربرنامه دارای خاصیتی به نام Button است. این خاصیت (e.Button) همان دکمه ای است که کاربر روی آن کلیک کرده است. پس می توانیم با استفاده از مشخصات دکمه (از جمله Tag) تشخیص بدهیم کاربر روی کدام دکمه کلیک کرده است.

```
    Private Sub ToolBar1_ButtonClick(ByVal sender As System.Object,
ByVal e As System.Windows.Forms.ToolBarButtonClickEventArgs) Handles
ToolBar1.ButtonClick
        Dim e2 As New System.EventArgs
        Select Case e.Button.Tag
            Case "Add"
                MenuAdd_Click(MenuAdd, e2)
            Case "Delete"
                MenuDelete_Click(MenuDelete, e2)
            Case "Register"
                MenuRegister_Click(MenuRegister, e2)
        End Select
```

در کد بالا با استفاده از Select Case و مقدار Tag تشخیص داده ایم کدام دکمه کلیک شده است. از آنجا که وظیفه هرکدام از این دکمه ها معادل وظیفه یکی از آیتمهای منو است؛ هر یک از آنها زیربرنامه کنترل کننده رویداد کلیک منوی معادلشان را فراخوانی میکنند. این زیربرنامه ها دو آرگمان دارند. میدانیم که آرگمان

اول، همان شیی است که رویداد برای آن اتفاق افتاده است، پس باید همان MenuItem ای باشد که می خواهیم زیربرنامه کنترل کننده رویداد کلیک آن فراخوانی شود. آرگمان دوم (دراینجا) از نوع EventArgs است. ما نیز یک شی از این نوع درست می کنیم و به آن زیربرنامه می دهیم.

در این برنامه هنگام شروع اجرا هیچ فروشگاهی در ComboBox وجود ندارد. کاربر باید نام فروشگاه ها را در برنامه ثبت کند. برای نگهداری لیست فروشگاه ها از کلاسی با نام ArrayList که در Collections قرار دارد استفاده می کنیم. شی ی که از روی این کلاس ساخته می شود باید در کلاس ها و فرم های برنامه در دسترس باشد ، به همین دلیل کلاس دیگری با نام mGlobals به برنامه اضافه کرده و شی مذکور را بصورت Public Shared در آن تعریف می کنیم. یکی از روشهایی که می تواند برای مدیریت متغیرهای عمومی به کار رود استفاده از یک کلاس جداگانه با نامی مشابه Global برای آن کلاس است. این روش زمانی که تعداد متغیرهای عمومی برنامه ما زیاد باشد بسیار مفید خواهد بود. ولی استفاده زیاد از این روش می تواند باعث از هم پاشیدگی ساختار شی گرایی برنامه شود. پس در استفاده از اشیاء Public Shared زیاده روی نکنید.

```
Public Class mGlobals
    Public Shared Forooshgah As New Collections.ArrayList
End Class
```

برای اینکه بتوانیم راحت تر اعمال حذف و درج فروشگاه ها را انجام دهیم، و نگران ابعاد آرایه نباشیم، بجای آرایه ساده از یک ArrayList استفاده کرده ایم.
برای اینکه نام فروشگاه ها را در ComboBox بنویسیم از زیربرنامه زیر استفاده می کنیم.

```
    Private Sub FillCombo()
        cmbStores.Items.Clear()
        Dim i As Integer
        For i = 0 To mGlobals.Forooshgah.Count - 1
            cmbStores.Items.Add(mGlobals.Forooshgah(i).Name)
        Next
    End Sub
```

برای اینکه دقدقه اینکه آیا آیتمی را در زمان عملیات اضافه و حذف اشتباهی از combobox حذف کردیم یا نه را نداشته باشیم و همیشه از صحت اطلاعات combo مطمئن باشیم و با توجه به اینکه تعداد داده های درون آن کم است بهتر است هر زمان که تغییری در لیست فروشگاه ها انجام شد اطلاعات درون combo را از نو وارد آن کنیم.

ایجاد فروشگاه جدید

برای اضافه کردن یک فروشگاه، در رویداد کلیک از منوی "امکانات/فروشگاه ها/اضافه کردن" کد زیر را می نویسیم:

```
        Dim str As String = InputBox("نام فروشگاه جدید؟؟")
        If str.Trim <> "" Then
            Dim bastan As New bastani(str.Trim)
            mGlobals.Forooshgah.Add(bastan)
            FillCombo()
        End If
```

ابتدا با استفاده از InputBox نام جدید را از کاربر دریافت کرده ، سپس در صورتی که نام وارد شده توسط کاربر (پس از حذف فضاهای خالی اول و آخر آن) برابر "" نبود. با استفاده از آن نام، شیِ جدیدی از بستنی فروشی ساخته و ارجاع آن را در Forooshgah اضافه می کند. در آخر نیز با استفاده از FillCombo مقادیر جدید را در ComboBox نمایش می دهد.
البته کد بالا را کمی خلاصه تر هم می توان نوشت:

```
        Dim str As String = InputBox("نام فروشگاه جدید؟؟")
        If str.Trim <> "" Then
            mGlobals.Forooshgah.Add( New bastani(str.Trim) )
            FillCombo()
```

```
        End If
```

در اینجا شی bastan تعریف نشده است و عبارت New Bastani(str.Trim) به عنوان پارامتر به متد Add داده شده است. این روش استفاده از New باعث خلاصه شدن کد ما می شود. دستور New سازنده کلاس بستنی را صدا میکند و با استفاده از آن یک شی جدید ایجاد می کند که هیچ نامی ندارد. سپس ارجاع این شی بی نام را به متد Add از شی Forooshgah میدهد.
دیگر کد هایی که در این فرم قرار داده شده تکراری است و می توانید آنها را سی دی ضمیمه ببینید.

فرم حذف فروشگاه

بخش دیگری که قابل ذکر است کدهای مربوط به حذف یک فروشگاه می باشد. برای اینکار فرم جدیدی به پروژه اضافه کرده و کنترلهای مورد نظر(مانند شکل زیر) را بر روی آن قرار می دهیم.

اطلاعات فروشگاه ها را بوسیله کدی مانند FillCombo به ComboBox روی این فرم منتقل می کنیم.

```
        Dim frm As New frmStores
        Dim i As Integer
        frm.cmbStores.Items.Clear()
        For i = 0 To mGlobals.Forooshgah.Count - 1
            frm.cmbStores.Items.Add(mGlobals.Forooshgah(i).Name)
        Next
        frm.ShowDialog()
        FillCombo()
```

سپس آن را نمایش داده و در آخر دوباره اطلاعات ComboBox روی فرم اصلی را پر می کنیم زیرا ممکن است آنها طی عملیات حذف تغییر کرده باشند. البته با استفاده از یک مقدار Boolean می توان این کار را انجام داد بدین صورت که یک مقدار boolean با scope ی معادل publlic در فرمِ حذف اضافه می کنیم ، اگر عمل حذف انجام شد آن را True می کنیم ، در غیر این صورت مقدار پیش فرض آن False است. وقتی از ShowDialg باز گشت در صورت True بودن مقدار آن Boolean از FillCombo استفاده می کنیم.
کدِ رویداد کلیک از دکمه حذف را در زیر می بینید :

```
If  MessageBox.Show("هستید مطمئن آیا ", "فروشي بستني", _
MessageBoxButtons.YesNo) = DialogResult.Yes Then
      Dim i, j As Integer
      If cmbStores.SelectedIndex <> -1 Then
            Dim index As Integer = cmbStores.SelectedIndex
            mGlobals.Forooshgah.RemoveAt(index)
            cmbStores.Items.Clear()
            For i = 0 To mGlobals.Forooshgah.Count - 1
                  cmbStores.Items.Add(mGlobals.Forooshgah(i).Name)
            Next
      End If
End If
```

در خط اول با استفاده از یک MessageBox از کاربر اطمینان وی مبنی بر حذف فروشگاه پرسیده می شود ، در صورت موافق بودن ادامه کدهای درون بلاک If اجرا می شود .
در شرط ifِ دوم اگر از ComboBox مقداری انتخاب شده باشد ، مقدار SelectedIndex مخالف 1- است و باعث می شود وارد If دوم شود. در اینجا مقدار SelectedIndex را گرفته و در Index ذخیره می کند .

بوسیله متد RemoveAt در ComboBox ِItems مقداری را که دارای شماره Index بوده حذف می کنیم و در خط بعد همین کار را برای ArrayList ِ فروشگاه ها انجام می دهیم در آخر نیز اطلاعات ComboBox را دوباره نمایش می دهیم .

سازنده ها و مخرب ها

سازنده ها زیربرنامه عضو (متدهای) خاصی هستند که وظیفه آنها مقدار اولیه دادن به یک شـی اسـت. بـه عبارت دیگر هر کاری که شی در زمان ایجاد شدن باید انجام بدهد به عهده سازنده ها است.

تعریف سازنده (constructor)

تعریف کردن یک سازنده بسیار ساده است. کافی است در کلاسـی کـه میخـواهیم سـازنده بـرایش تعریـف کنیم، زیربرنامه ای (sub) با نام New ایجاد کنیم.

```
Class bastani
    …
    Public Sub New()
        MessageBox.Show("constructor")
    End Sub
    …
End Class
```

حال اگر برنامه را اجرا کنید خواهید دید که به محض ایجاد شدن شی پیغام مذکور نمایش داده خواهـد شـد. وقتی کامپایلر به دستور New bastani() میرسد متـد New از کـلاس bastani را اجـرا میکنـد. البتـه اگـر متـد New تعریف نشده باشد سازنده پیشفرض اجرا میشود و به همه متغیرها مقدار اولیه پیشفرض را میدهد.

توجه : اگر سازنده بصورت Private تعریف شود، نمیتوان بـا اسـتفاده از کلمـه کلیـدی New یـک شـی جدیـد ایجاد کرد.

میتوانیم در سازنده به متغیرها مقدار اولیه بدهیم :

```
    Public Sub New()
        mMeghdarBastani = 0
        mMeghdarPool = 0
        mTedadGhashogh = 0
        mTedadLivan = 0
        mTedadNan = 0
    End Sub
```

این سازنده به تمام متغیرهای ما مقدار صفر میدهد. البته در این مورد کـاری کـه سـازنده ای کـه مـا تعریـف کرده ایم با کار سازنده پیش فرض یکی است. اما در فصلهای بعد خواهیم دید که وظایف بیشتری بـه عهـده این سازنده میگذاریم.

مخرب ها (destructor)

مخرب ها زیربرنامه های عضوی هستند که وظیفه آنهـا از بـین بـردن شـی و برگردانـدن منـابع گرفتـه شـده توسط شی به سیستم است. البته در مورد متغیرهایی که تعریف میکنیم هـیچ جـای نگرانـی نیسـت چـون خود ویژوال بیسیک آنها را به سیستم باز میگرداند. در ویژوال بیسیک مخربها بیشتر وظیفه قطع کردن ارتباط با فایلها و پایگاههای داده را به عهده دارند، که در فصلهای بعد بررسی خواهیم کرد.

تعریف یک مخرب بصورت زیر است :

```
Class bastani
    …
    Protected Overrides Sub Finalize()

    End Sub
    …
End Class
```

از آنجا که توضیح دادن کلمات کلیدی Protected و Overrides نیـاز بـه اطلاعـات در زمینـه ارثبـری دارد و ارث بری جزو مباحث این کتاب نیست از توضیح آنها خودداری میکنیم. البتـه تـا زمـانی کـه ارثبـری را یـاد نگرفتـه باشید استفاده دیگری نیز از این کلمات کلیدی نخواهید کرد، بنابراین میتوانید این مـورد را بـه همـین صـورت حفظ کنید.

مخرب نیز مانند سازنده ها بصورت خودکار اجرا میشود و لازم نیست ما آن را اجرا کنیم. وقتی عمر یک شی به پایان برسد، کامپایلر زیربرنامه Finalize آن شی را اجرا میکند. اینجا نیز مانند سازنده ها، اگر مخرب وجود نداشته باشد مخرب پیشفرض اجرا میشود.

دستور زیر را در مخرب بنویسید و برنامه را اجرا کنید تا ترتیـب اجـرا شـدن مخربهـا و سـازنده هـا را در عمـل ببینید.

```
        MessageBox.Show("destructor")
```

سازنده ها به محض اجرای برنامه اجرا میشوند. چون اشیاء Bastani را به عنوان داده عضو در کلاس frmMain تعریف کرده ایم و به همین دلیل عمر آنها وقتی به پایان می رسد که فرم در حال بسته شدن است، Finalize اشیاء ما نیز همان موقع اجرا میشوند.

سازنده های چند آرگمانی

چقدر بهتر بود که سازنده ما میتوانست آن مقداری را که ما میخواهیم در ابتدای کار به متغیرهای عضو بدهد. برای اینکه مقدار اولیه را از کاربر کلاس بگیریم، میتوانیم از سازنده چند آرگمانی استفاده کنیم. سازنده ها نیز مانند هر زیربرنامه دیگری میتوانند چندین آرگمان داشته باشند:

```
Public Sub New(ByVal nan As Integer, _
    ByVal bas As Integer, ByVal pool As Integer, _
    ByVal livan As Integer, ByVal ghashogh As Integer)

    mMeghdarBastani = bas
    mMeghdarPool = pool
    mTedadGhashogh = ghashogh
    mTedadLivan = livan
    mTedadNan = nan
End Sub
```

آنچه در اینجا تعریف کردیم یک سازنده 5 آرگمانی است که میتواند برای همه داده های ما مقدار اولیه بگیرد. برای استفاده از این سازنده میتوانیم شی را بصورت زیر تعریف کنیم :

```
Dim m As New bastani(20, 10, 0, 5, 8)
```

انتخاب اینکه کدام سازنده باید اجرا شود بسیار شبیه انتخاب از بین توابع بار اضافی داده شده است. کامپایلر از بین سازنده ها بدنبال سازنده ای میگردد که 5 آرگمان عددی داشته باشد. و آن را با مقادیر تعریف شده فراخوانی میکند.

نکته: اگر یک سازنده تعریف کنید، دیگر سازنده پیشفرض اجرا نخواهد شد. پس اگر سازنده‌ی بدون آرگمان تعریف نکرده باشید کاربر مجبور میشود از سازنده چند آرگمانی استفاده کند.

یکی از مشکلات برنامه بستنی فروشی ما این است که ممکن است کاربر فراموش کند نام (Name) شی بستنی را تنظیم کند. ما میتوانیم با حذف کردن سازدنده بدون آرگمان و ایجاد یک سازنده تک آرگمانی که نام شی بستنی را میگیرد، کاربر کلاس را مجبور کنیم نام شی را تنظیم کند.

```
Public Sub New(ByVal name As String)
    mObjName = name
End Sub
```

را اضافه می کنیم :nameبه آرگمانهای سازنده پنج آرگمانی هم آرگمان

```
Public Sub New(ByVal name As String, ByVal nan As Integer, _
ByVal bas As Integer, ByVal pool As Integer, _
ByVal livan As Integer, ByVal ghashogh As Integer)

    mObjName = name
    mMeghdarBastani = bas
    mMeghdarPool = pool
    mTedadGhashogh = ghashogh
    mTedadLivan = livan
    mTedadNan = nan
End Sub
```

پس از این تغییرات ویژوال بیسیک از تعریف اشیاء کلاس بستنی ما ایراد خواهد گرفت چون سازنده بدون آرگمان را پیدا نمی کند. با استفاده از سازنده تک آرگمانی آنها را اصلاح میکنیم:

```
Private WithEvents ValiAsr As New bastani("ValiAsr")
Private WithEvents Farmanie As New bastani("Farmanie")
Private WithEvents TehranPars As New bastani("TehranPars")
Private Forooshgah(2) As bastani
Private Sub frmMain_Load(…) Handles MyBase.Load
    Forooshgah(0) = ValiAsr
    Forooshgah(1) = Farmanie
```

```
        Forooshgah(2) = TehranPars

        cmbStores.Items.Add("ValiAsr")
        cmbStores.Items.Add("Farmanie")
        cmbStores.Items.Add("TehranPars")
    End Sub
```

و در نهایت سه خطی که وظیفه مقدار اولیه دهی به اشیاء Bastani را داشتند، از frmMain_Load حذف می کنیم.

سازنده Shared

سازنده Shared برای مقدار اولیه دادن به متغیرهای Shared استفاده میشود. سازنده های Shared به محض اجرای برنامه اجرا میشوند. بدون .وابستگی به اینکه از روی آن کلاس شیی ایجاد شده است یا نه.

کارگاه

راهی پیدا کنید که با استفاده از آن، کلاس بستنی بتواند از تعداد اشیایی که از رویش ایجاد شده است مطلع شود.

راهنمایی : داده Shared و سازنده غیر Shared

پاسخ کارگاه:

```
    Private Shared Count As Integer
    Shared Sub New()
        Count = 0
    End Sub
    Public Sub New(ByVal name As String)
        Count += 1
        mObjName = name
    End Sub
    Protected Overrides Sub Finalize()
        Count -= 1
    End Sub
```

متغیر Shared بین همه اشیاء مشترک است. ابتدای برنامه، توسط سازنده shared با مقدار 0 مقدار دهی میشود. پس از آن هرگاه که یه شی ایجاد شود یکی به مقدار Count اضافه میشود و هرگاه یک شی از بین برود، یکی از Count کم میشود. البته Count+=1 باید در سازنده 6 آرگمانی (5 آرگمانی سابق) نیز نوشته شود.

پروژه برنامه نویسی

سازنده (های) شی کتاب در پروژه کتابخانه را طراحی و پیاده سازی کنید.

Imports و Namespace

يک Namespace مفهومي انتزاعي است که براي ارتباط دادن يک يا چند کلاس يا ماژول به يکديگر به کار مي رود . اين ارتباط با توجه به مفاهيم آن کلاس ها و نوع کار آنها است. به اين معني که برنامه نويس تشخيص مي دهد کلاس هايي که کار خاصي مي کنند را در يک Namespace قرار دهد. مثلاً کلاسهايي که مربوط به کار با فايل و ورودي يا خروجي است در IO قرار مي گيرد. فرض کنيد تعداد زيادي کلاس در مورد صندلی (Chair) ساخته ايد. مثلاً OneLegChair، ThreeLegChair و FourLegChair و WoodenChair و GoldenChair. براي اينکه همه آنها را يکجا جمع آوري کنيم تا راحت تر به آن دسترسي داشته باشيم آنها را به اين شکل در يک Namespace قرار مي دهيم .

```
Namespace MyChairs
      Public Class OneLegChair
      .
      .
      .
      End Class
      Public Class ThreeLegChair
      .
.
.
End Class
.
.
.
.
End Namespace
```

حال در هر کجا که لازم باشد از يکي از اين کلاس ها استفاده کنيم به اين روش مي توانيم عمل کنيم .

```
MyChair.OneLegChair
```

يکي ديگر از مزايای استفاده از Namespaceها، اطمينان از يکتا بودن کلاس هايمان است. براي مثال وقتي GoldenChair در Namespace_ MyChairs قرار بگيرد ما محدوديتي براي ساختن کلاس ديگري که با همين نام در جاي ديگر برنامه نداريم . آن GoldenChair در يک Namespace ديگر است و goldenChair_ فعلي ما در MyChairs .

علاوه بر اين، Namespaceها با طبقه بندی کردن کلاسها باعث می شوند کار با آنها ساده تر از قبل شود. دليل ديگر استفاده از Namespace ساده بودن برنامه نويسي است . قبلاً با تکنولوژي IntelliSence آشنا شديد . حتماً توجه کرديد که با استفاده از آن تکنولوژي نوشتن برنامه با سرعت بيشتري پيش مي رود اگر تعداد کلاسهاي ما زياد باشد بخاطر سپردن نام آنها سخت مي شود ، اما وقتي درون namespace ها طبقه بندي شده باشد به راحتي با فراخواني نام namespace و يک نقطه بعد از آن مي توانيم نام کلاس مورد نظر خود را بيابيم .

```
Dim m_Chair as New MyChairs.GoldenChairs
```

تمام کلاسهاي دات نت بوسيله namespace ها به شکل درختي طبقه بندي شده اند ، اين روش درختي به ما امکان_ داشتن شاخه مختلف را مي دهد . کلي ترين namespace، System است . کافي است براي ديدن تعداد زير شاخه هاي System با استفاده از IntelliSence شاخه هاي آن را ببينيد . هرگونه کلاسی که براي برنامه خود احتياج داشته باشيد در اين namespace موجود است ، اگر DLL جديدي به منابع پروژه اضافه کنيد با استفاده از system به آن دسترسي خواهيد داشت (عموماً) . System.Data، System.XML ، System.IO، System.Collections و ... نمونه هايي از زير شاخه هاي System هستند که هر کدام از آنها داراي زير شاخه هاي فراواني مي باشند .

در هنگام کار اگر کلاس مورد نظر ما در زير شاخه لايه هاي دوم و سوم به بعد (مانند System.IO.File) باشد؛ استفاده مکرر از اين نام براي برنامه نويس خسته کننده مي شود .براي حل اين مشکل بايد از کلمه کليدی Imports استفاده کنيم. ساختار دستوری Imports بصورت زير است :

```
Imports Namespace.element
```

این دستور باید قبل تعریف کلاس بکار برود. برای مثال :

```
Imports System.IO
Public Class frmMain
…
```

حال در frmMain به جاي عبارت قبل(System.IO.File) مي توانيم از عبارت کوتاه File استفاده کنيم .
البته استفاده نابجا از Imports تمام مزيت namespace را از بين مي برد . اگر چند کلاس در namespace هاي مختلف هم نام باشد و ما همه آن Namespaceها را Import کنيم؛ ممکن است کلاسهای همنام را جابجا استفاده کنيم.
در بعضي از موارد بودنِ نامِ namespace به خوانا بودن برنامه کمک مي کند. براي مثال اگر عبارت Imports System.Windows.Forms را به کار برده باشيد وقتي از کلاس control استفاده مي کنيم ممکن است با کلاس System.Web.UI.Control اشتباه شود . پس براي حل اين مشکل بهتر است در Imports از Imports System و به جاي control از Windows.Forms.Control استفاده کنيم .

Error Handling

در برنامه نویسی امکان رخ دادن سه نوع خطا وجود دارد.
نوع اول مشکلات و اشتباهات در تایپ کلمات و دستورات است که به آن Syntax Error گفته می شود . همان طور که پیش از این دیده اید این نوع ایراد ها در vs.net به سرعت و سهولت رفع می شود زیرا در هنگام تایپ کردن از صحت نوشتاری کدهای خود با علائمی که IDE در زیر خطوطِ نوشته شده قرار میدهد مطلع می شویم .
نوع دوم Runtime Error یا خطاهای زمان اجراست. این گونه خطاها زمانی پیش می آیند که دستور از لحاظ ساختاری (نحوی) درست است؛ ولی وقتی زمان اجرا (Run Time) می رسد، به دلایلی نمی تواند اجرا شود و باعث توقف برنامه میشوند. از آنجا که این گونه خطاها استثناً پیش می آیند و همیشه وجود ندارند، به آنها استثنا (Exception) نیز گفته می شود. مانند تقسیم بر صفر یا وقتی که سعی کنیم "a" را به عدد صحیح تبدیل کنیم.
نوع سوم که رفع آن سخت تر از بقیه انواع ایرادها است، Logical Error یا ایراد منطقی می باشد.این خطا زمانی پیش می آید که ساختار دستوری(نحوی) برنامه درست است، زمان اجرا نیز خطایی پیش نمی آید ولی نتیجه برنامه آن چیزی نیست که ما انتظار داشتیم. برای مثال برنامه ای که قرار است دو عدد را جمع کند، بجای اینکه حاصل 2+3 را 5 برگرداند 32 برمی‌گرداند. این خطا در منطق برنامه و کارایی الگوریتم های آن است. وقتی تمام نیازها را تحلیل و ارتباط بین اجزای برنامه را طراحی می کنیم باید دقت کامل داشته باشیم تا در آینده اشکالی در منطق برنامه شما پیدا نشود.
Error Handling یکی از مهمترین مباحث در هر زبان برنامه نویسی است که در زمینه کنترل کردن خطاهای زمان اجرا (Exceptionها) مطرح می شود. اگر یک برنامه اجازه دهد در هنگام کار ، خطاها آن را مختل کنند کاربر را از گرفتن جواب صحیح از برنامه ناامید می کند و باعث می شود برنامه بلااستفاده بماند.
در net. برای error handling از شیی به نام exception استفاده می شود که از کلاس system.exception مشتق می شود. این شیِ exception برای برنامه نویسان مکانیزمی استاندارد و یکپارچه در کل net. فراهم می کند تا به خطاهای بوجود آمده رسیدگی کنند.
همانطور که گفتیم، Exception ها حالتهای استثنایی هستند که زمان اجرا پیش می آیند و باعث توقف (crash) برنامه می شوند. اجازه بدهید مثال دیگری را نیز بررسی کنیم. فرض کنید کاربر برنامه ما اطلاعات خود را حاضر کرده و دستور ذخیره در یک فلاپی را توسط برنامه ما صادر می کند اما دیسکتی در فلاپی درایو نیست ، این یک Exception یا خطا است که اگر آن را پیش بینی نکرده باشیم می تواند باعث از دست رفتن اطاعات کاربر شود.
شیِ Exception برای بسیاری از namespace ها توسعه یافته است به گونه ای که جوابگوی اطلاعات و محتویات آن namespace باشد. در جدول زیر پر استفاده ترین exception ها را در namespace های مختلف می بینید.

Class	Namespace
ApplicationException	System
SystemException	
VB6Exception	

System.Data	InvalidConstraintException
System.IO	IOException
System.Runtime.InteropServices	COMException
System.Web.Services.Protocols	SoapException
System.XML	XmlException

System Namespace خیلی از استثناهایی که ممکن است در برنامه ما رخ دهد را پوشش می دهد. در جدول زیر تعدادی از این استثناها را به همراه توضیحاتی می بینید.

Class	توضیحات
ArgumentNullException	وقتی به یک پارامتری که نباید Null باشد مقدار Null فرستاده شود
DivideByZeroException	وقتی عددی تقسیم بر صفر بشود
OutofMemoryException	وقتی حافظه کافی برای ادامه عملیات وجود نداشته باشد
Vb6Exception	اگر هنگام استفاده از توابع ویژوال بیسیک 6 خطایی رخ دهد

استاندارد کردن Error Handling

در هنگام نوشتن کد یک برنامه در VB باید توجه خاصی به استاندارد کردن error handling داشته باشیم . اگر error handling ما استاندارد نباشد ممکن است ایرادی خارج از ساختاری که ما برای ایراد طراحی کردیم اتفاق بیفتد و باعث شود برنامه از روال طبیعی خود خارج شود ، در حالی که تمام هدف error handling ایجاد روشی برای بررسی تمامی اتفاق های نا خواسته ای است که در برنامه رخ می هد و این روشها از از دست رفتن اطلاعات کاربر جلوگیری می کنند.
مبحث error handling با روشهایی برای به دام انداختن ایرادهای برنامه در ساختاری که ما می سازیم آغاز می شود. در ساختار برنامه های NET. همیشه امکان رخ دادن ایراد به کلاس فعلی یا برنامه ای که کلاس شما در آن است محدود نمی شود ، بلکه ممکن است در یکی از توابعی که در یک کامپیوتر دیگر اجرا می شود و شما آن را صدا زده اید اتفاق بیفتد. شما به عنوان طراح برنامه باید تصمیم بگیرید که چگونه می خواهید ایرادهای رخ داده را به کاربر نمایش دهید ، توسط یک MessageBox یا ساختن یک سیستم Log گیری. در VB.NET همچنان از روش VB6 برای error که همان On Error است پشتیبانی می شود اما بهتر است به جای استفاده از روش قدیمی ، از روشی که در NET. ایجاد شده ، Try Catch End Try استفاده کنید.

Exception Handler

در این بخش چگونگی استفاده از Exception ، ساختار آن ، خواص و متدهای آن را بررسی می کنیم . در VB.NET برای Exception از ساختاری به شکل

```
Try
Catch [ex As Exception]
Finally
End Try
```

استفاده می شود. کدی ممکن است استثنا(Exception) در آن رخ دهد؛ بین Try و Catch می نویسیم. اگر خطایی در بلوک Try پیش بیاید، اجرای برنامه به بلوک Catch منتقل می شود. بخش Finally قسمتی است که همیشه اجرا می شودچه در بخش try خطایی پیش بیاید، چه پیش نیاید.
برای نمونه از مثال معروف تقسیم بر صفر استفاده می کنیم :

```
Dim a, b, c As Integer
b = 0
a = 5
Try
    c = a / b
Catch
```

```
    MessageBox.Show("است داده رخ ایرادی")
Finally
    MessageBox.Show(Convert.ToString(c))
End Try
```

اگر در تقسیم a/b خطایی رخ بدهد، اجرای برنامه به بخش Catch منتقل میشود و پیغام "ایرادی رخ داده است" نمایش داده خواهد شد. در نهایت بخش Finally اجرا می شود و مقدار C را نمایش میدهد (که در اینجا به دلیل خطایی که رخ میدهد c مقدار نمی گیرد و برابر صفر است) . در اینجا پیغام ما خیلی مشخص نیست، برای همه خطایی که ممکن است پیش بیاید، فقط یک پیغام داریم که آن هم اطلاعات کافی به کاربر نمی دهد. برای جدا کردن خطاهای مختلف از یکدیگر میتوانیم از Catch های متوالی استفاده کنیم، که هر کدام نوعی خطا را کنترل میکنند :

```
Dim a, b, c As Integer
b = 0
a = 5
Try
    c = a / b
Catch error1 As System.OverflowException
    MessageBox.Show("صفر بر تقسیم")
Catch
    MessageBox.Show("است داده رخ ایرادی")
Finally
    MessageBox.Show(Convert.ToString(c))
End Try
```

در این مثال دوبار از catch استفاده کردیم ، اولین catch خطاهایی از نوع OverflowException را کنترل میکند که "تقسیم بر صفر" نیز جزء آنهاست. در حقیقت وقتی خطایی رخ میدهد، برنامه از اولین Catch شروع کرده و بدنبال یک Catch می گردد که نوع خطایی که کنترل میکند، با نوع خطای پیش آمده یکی باشد. در اینجا error1 شیی از نوع OverflowException است که اطلاعاتی (از جمله پیغام خطایی که پیش آمده error1.Message) درباره خطا دارد. دومین Catch همان Catchای است که پیش از این نیز نوشته بودیم، این Catch می تواند برای همه نوع خطایی فراخوانی شود.

Exit Try

این دستور باعث می شود از بلاک try catch خارج شده و در صورت وجود به قسمت finally برود. در مثال ما هر موقع b صفر باشد پیغام خطای overflow می دهد ، بنابراین ما از این حالت چشم پوشی می کنیم :

```
                        Catch error1 As System.OverflowException
    If b = 0 Then
        Exit Try
    End If
    MessageBox.Show("صفر بر تقسیم")
Catch
```

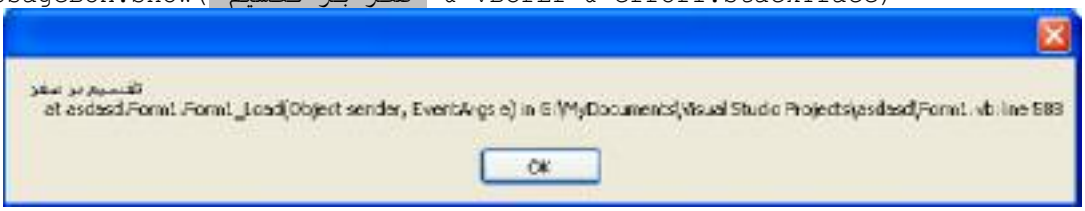
ساختارهای Try Catch را می توان بصورت تو در تو نیز بکار برد. این روش معمولا در حالتهایی استفاده می شود که قسمت catch نیز امکان بوجود آمدن ایراد وجود داشته باشد .

```
Try
    Try
    Catch
        Try
        Catch
        End Try
    End Try
Catch
    Try
    Catch
```

```
        End Try
    End Try
```

روش های دیگری نیز برای پیدا کردن و جلوگیری از ایراد وجود دارد. برای اینکه راحت تر خطی که در آن ایراد رخ داده را بیابیم می توان از دو خاصیت به نامهای source و stacktrack استفاده کرد ، مانند شکل زیر :

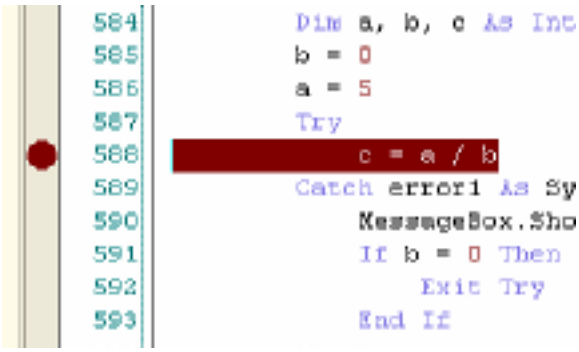
```
MessageBox.Show("صفر بر تقسیم" & VBCrLf & error1.StackTrace)
```



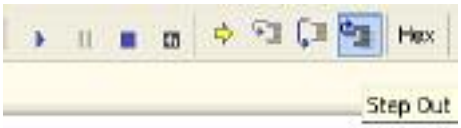
در مثال بالا خطی که در آن ایراد رخ داده است نیز اعلام شده ، البته این امکان فقط در حالت debug قابل استفاده است ، اگر مدِ ویژوال استودیو را به release تغییر دهید دیگر شماره خط را اطلاع نمی دهد .

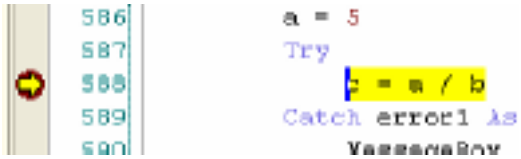
Break Point

گاهی اوقات ممکن است ایرادی در برنامه بوجود آید که به غیر از پیمایش خط به خط راهی برای پیدا کردن و رفع آن وجود نداشته باشد. در ویژوال استودیو شما می توانید از break point برای این هدف استفاده کنید. ابتدا خطی از برنامه را که می خواهید از آنجا به بعد خودتان خط به خط برنامه را بازبینی کنید انتخاب و سپس بر روی IDE در سمت چپ ، روی نوار کنار کدها کلیک کنید تا دایره ای قرمز بر روی آن نمایش داده ، همچنین خط انتخاب شده نیز قرمز می شود. برای حذف این break point می توانید دوباره روی آن دایره قرمز کلیک کنید.



حالا برنامه را اجرا می کنیم، وقتی به خط مورد نظر برسد وارد IDE شده و از آنجا با استفاده از کلید هایی که در شکل زیر می بینید (و shortcut های آنها) می توانید برنامه را به جلو یا گاهی به عقب ببرید و نتیجه اجرای هر خط را ببینید.





دقت کنید که این کار فقط در حالت debug امکان پذیر است و اگر از حالت release استفاده کنیم چنین امکانی وجود ندارد. همچنین دقت کنید که از break point فقط برای خطوطی که اجرا می شوند می توانیم استفاده کنیم ، مثلاً روی خطی که دستور Dim a As Integer در آن نوشته شده، نمی توان BreakPoint گذاشت.
وقتی ویژوال استودیو وارد حالت break بشود می توانید از جدولی که در شکل زیر می بینید و در پایین IDE قرار گرفته برای دستکاری متغیرهای برنامه استفاده کنید.


برای مرور خطوط برنامه سه روش وجود دارد که شکل آنها را در محیط IDE مشاهده می کنید

Step Into : هنگامی که بخواهیم برنامه خط به خط بدون پرش از روی خط خاصی اجرا شود از این کلید استفاده می کنیم . هر بار که روی آن کلیک کنیم یک خط دیگر از برنامه اجرا خواهد شد ، با استفاده از کلید F8 (تو 2003 شده F11) هم می توانیم از این امکان استفاده کنیم. اگر هنگام استفاده از Step Into اجرا به یک تابع برسد، وارد آن شده و خط به خط آن را هم اجرا می کند.
Step Over : این روش وقتی به توابع برسد، وارد آنها نمیشود؛ به همین دلیل زمانی مناسب است که قبلاً تابع را بازبینی کرده باشیم و از بی عیب بودن آن اطمینان داشته باشیم. به این ترتیب وقتی برنامه به تابع برسد آن را اجرا می کند بدون اینکه ما اجرای خط به خط آن را ببینیم .

Step Out : با استفاده از این روش اگر در یک تابع قرار گرفته باشیم به خارج آن منتقل می شویم .

System.IO

IO يكي از Namespace هاي پركاربرد در دات نت مي باشد . كمتر برنامه اي است كه احتياج به ورودي و خروجي نداشته باشد . در اين namespace كه مخفف Input/Output مي باشد كلاسهاي متفاوتي براي كار با File، Directory، Stream و ... وجود دارد كه مثالهايي از بعضي از آنها را خواهيم ديد . قبل از شروع بحث بايد چند مفهوم آشنا شويم .

Stream چيست ؟

Stream يک مفهوم انتزاعي از صفي از بايتهاست. فايلها، اطلاعاتي كه از دستگاه هاي ورودي و خروجي دريافت و يا به آنها ارسال مي شود يا حتي يک Socket در TCP/IP همگی مثالهایی از جریانها (Streams) هستند. كلاس Stream يک كلاس پايه براي تمامي كلاس ها از نوع Stream است ، مانند FileStream كه براي فايل بهينه سازي شده ، يا كلاس StreamWriter كه براي خروجي كاراكترها مي تواند به كار رود. از انواع ديگر stream مي توان به Memory Stream و Network Stream اشاره كرد كه هر كدام از اين كلاس ها خواصي دارند كه با توجه به رسانه انتقال آنها با ديگري تفاوت دارد ، مانند جستجو در Stream . اين كلاس ها بسيار قدرتمند هستند و عملاً برنامه نويسی با آنها نيازی به دانستن جزئيات كار سيستم عامل و دستگاه ندارد .

يكي ديگر از كلاسهاي مهم در IO كلاس File است .از اين كلاس (به علت نداشتن سازنده (Constructor)) نمي توان يک Object ساخت بلكه بايد از امكانات آن به روش static استفاده نمود . كارمان را با يک مثال شروع مي كنيم . فرض كنيد مي خواهيم فايلي با نام HelloWorld.txt در پارتيشن C ايجاد كنيم . كلاس File داراي متدي است كه به عنوان پارامتر مسير را مي گيرد و به عنوان خروجي يک StreamWriter را باز مي گرداند .

```
Imports System.IO
…
Dim MyWriter as StreamWriter = File.AppendText("C:\HelloWorld.txt")
With MyWriter
    .WriteLine("This is line 1")
    .WriteLine("This is line 2")
    .Flush()
    .Close()
End With
```

به همين سادگي با استفاده از متد WriteLine ، متن مورد نظر را در يک خط از فايل مي توانيم بنويسيم . هنگامي كه متد Flush استفاده مي شود تمام اطلاعاتي كه در Stream باقي مانده است به سمت فايل مي رود و در آن قرار مي گيرد . در آخر نيز با استفاده از Close ، Stream بسته مي شود . البته در اينجا متد Close كارِ Flush را نيز انجام مي دهد ولي بهتر است هميشه از Flush استفاده كنيد .

خواندن فايل :

حال كه يک فايل ايجاد و اطلاعاتي را در آن ذخيره كرديم بهتر است ببينيم چگونه مي توانيم آن را بخوانيم . براي خواندن فايل تقريباً به همان كلاسي كه براي نوشتن مورد نياز بود احتياج داريم . اين بار به جاي AppendText از OpenText و به جاي StreamWriter از StreamReader استفاده مي كنيم . متد OpenText به عنوان پارامتر آدرس فايل را مي تواند بگيرد و خروجي يي از نوع StreamReader دارد .

```
Dim MyReader as StreamReader = File.OpenText("C:\HelloWorld.txt")
Dim strText as String = MyReader.ReadToEnd()
MyReader.Close()
MessageBox.Show(StrText)
```

البته از متد Open هم مي توان استفاده كرد . متد Open حالت كلي است و پارامترهاي مختلفي مي تواند داشته باشد . نمونه زير مي تواند جايگزين كد قبلي شود .

```
Dim MyReader as New StreamReader(File.Open("C:\HelloWorld.txt",FileMode.OpenOrCreate))
```

کلاس File متدهاي مختلفي دارد که نمونه ديگر آن متد Copy است . متد کپي فايل اول را به آدرس فايل دوم کپي مي کند . همچنين اين متد داراي يک Overload با پارامتر Boolean براي اينکه آيا بر روی فايل قبلی در صورت وجود overwrite بکند يا خير مي باشد .

```
File.Copy("C:\HelloWorld.txt","C:\HiWorld.txt",True)
```

متد ديگر Exists است که پارامتر آن آدرس فايل و خروجي آن Boolean ي است که نشان دهنده وجود يا عدم وجود فايل مورد نظر است .

```
If File.Exists("C:\MyFile.txt") Then File.Delete("C:\MyFile.txt")
```

کلاس FileInfo

استفاده از کلاس فايل براي همه حالتها مناسب نيست . کلاس FileInfo براي برخي استفاده ها از File مناسب تر است براي مثال وقتي حجم چند فايل را همزمان احتياج داشته باشيد کلاس FileInfo به کار مي آيد . بر خلاف کلاس File که به شما اجازه ساختن نمونه جديد نمي داد شما مي توانيد از کلاس FileInfo براي هر کدام از فايلهايتان يک نمونه جديد بسازيد . با ذکر يک مثال تفاوت اين دو کلاس را بررسي مي کنيم .

```
Dim fInfo as New FileInfo("C:\HelloWorld.txt")
MessageBox.Show(fInfo.Lenght.ToString())
```

در خط اول نمونه ای از کلاس FileInfo ساخته و سپس در خط دوم حجم آن را با استفاده از MessageBox اعلام کرديم.
اين کلاس داراي تعداد زيادي متد و خاصيت است که تقريباً تمامي عملياتي که براي فايلها مورد نياز باشد را پوشش مي دهد .

کلاس DirectoryInfo

يکي ديگر از کلاس هاي مفيد و بسيار کارآمد در IO کلاس DirectoryInfo است . همانطور که از نام اين کلاس مشخص است با استفاده از آن مي توانيد اطلاعاتي در مورد يک Directory بدست آوريد . براي مثال شما مي توانيد فايلها و شاخه هاي درون يک شاخه را پيدا کنيد ، به مثال زير توجه کنيد . در اين مثال ليست تمام دايرکتوري هاي پارتيشن C بدست مي آيد .

```
Dim str as New System.Text.StringBuilder
Private Sub Main()
      WriteFolders("C:\")
      Textbox1.Text = str.ToString()
End Sub

Private Sub WriteFolders(ByVal Path as String)
      Dim FolderList as New DirectoryInfo(Path)
      Dim DirInfo as DirectoryInfo()
      DirInfo = FolderList.GetDirectories()
      Dim DirInf as DirectoryInfo
      For Each DirInf in DirInfo
            WriteFolders(DirInf.Fullname)
            str.Append(DirInf.Fullname & ControlChars.Newline)
      Next
End Sub
```

هنگامی که زیربرنامه Main را صدا بزنیم آن سابروتین WriteFolders را صدا خواهد زد و در پارامتر آن مسیر مورد نظر را انتقال خواهد داد.
همان طور که در فصلهاي قبل ديده ايد اگر متغيري از نوع String تعريف کنيد هر بار که به آن مقدار جديد نسبت دهيم آن Object باز سازي مي شود . در صورتي که تعداد بازسازي کم باشد چندان مشکلي ايجاد نمي شود ولي اگر اين تعداد به چند هزار برسد سرعت را خيلي کاهش مي دهد . معمولاً در کامپيوترهاي موجود به علت داشتن فضاي بسيار زيادِ ديسک ، تعداد فايلها و Folder ها بسيار زياد است . چون ما در اين مثال مي خواهيم ليست دايرکتوري ها را پيدا کنيم لذا بايد تعداد زيادي نام جديد به رشته حاوی نامها اضافه کنيم و اين سرعت را به شدت کاهش مي دهد . راه ديگر براي انجام اين کار استفاده از کلاس StringBuilder و راه ديگر قرار دادن مقدار جديد در Textbox است . راه دوم زمان بيشتري مي گيرد زيرا هر بار بايد يک کنترل ويژوال را تغيير دهد و تغييرات ويژوال زمان زيادی می گيرد.
کلاس StringBuilder يکي از کلاس هاي زير مجموعه namespace ي به نام Text است و با استفاده از متد Appendِ آن مي شود مقدار جديدي به انتهاي رشته ای که نگهداری می کنداضافه کرد .

مثال ديگري از Stream ها

در بحث قبلي کمي با stream ها آشنا شديم . همان طور که گفته شد stream ها انواع مختلفي دارند که کاربرد يک نوع از آنها را در فايلها ديديم در اينجا مي خواهيم يک نمونه ديگر از کاربرد stream ها که در اينترنت است را با يک مثال توضيح دهيم .
وقتي شما در مرورگر خود آدرس يک صفحه را مي نويسيد تقاضاي شما به سرور آن صفحه فرستاده مي شود و در جواب تقاضاي شما رشته اي از اطلاعات از طرف سرور به سمت کامپيوتر فرستاده مي شود . در اين مثال ما مي خواهيم اين عمليات را در برنامه خود پياده سازي کنيم . روشهاي مختلفي براي اين کار وجود دارد ، کد زير يکي از آن روشهاست :

```
Dim m_URL As String = "http://www.cnn.com"
Dim        myHttpWebRequest        As        Net.HttpWebRequest        =
CType(Net.WebRequest.Create(m_URL), Net.HttpWebRequest)
Dim       myHttpWebResponse        As       Net.HttpWebResponse        =
CType(myHttpWebRequest.GetResponse(), Net.HttpWebResponse)
Dim receiveStream As IO.Stream = myHttpWebResponse.GetResponseStream()
Dim encode As System.Text.Encoding
encode = System.Text.Encoding.Default
Dim readStream As New IO.StreamReader(receiveStream, encode)
Dim read(256) As [Char]
Dim count As Integer = readStream.Read(read, 0, 256)
Dim mySite As String = ""
While count > 0
Dim str As New [String](read, 0, count)
mySite += (str)
            count = readStream.Read(read, 0, 256)
End While
MyTextBox.Text = mySite
```

ابتدا شي ي از نوع HTTPWebRequest ساخته می شود سپس اين شي پيامي را براي سرور CNN مي فرستد . سپس اطلاعاتي را که از سرور فرستاده مي شود ، گرفته و در اختيار ReceiveSteam قرار مي دهد و ReadStream با توجه به Encoding متن مورد نظر را مي خواند و در آخر در يک Textbox قرار داده مي شود .

پایگاه داده ها

یکی از مشخصه های اکثر نرم افزارهای کاربردی استفاده از بانک های اطلاعاتی است . در این فصل مفاهیم اساسی بانک های اطلاعاتی ، نحوه کار با MS Access و چگونگی استفاده از بانک Access در دات نت را خواهیم دید .

پایگاه داده ها چیست ؟

تصور کنید شما مسئول ثبت نام یک آموزشگاه هستید و می خواهید اطلاعات هنرآموزهای آنجا را در کامپیوتر ذخیره کنید . یکی از ابتدایی ترین روشهایی که می توانید استفاده کنید ساختن یک فایل متنی برای هر شخص، با توجه به شماره آن هنرجو است . در هر فایل اطلاعات افراد را به هر شکلی که مایل بودید ذخیره می کنید . مادامی که شماره هنرجو را داشته باشید، مشکلی ایجاد نمی شود؛ اما فرض کنید در این لیست طولانی از فایلها بخواهید با استفاده از نام هنرجو، شماره او را پیدا کنید . شما فایلها را بر مبنای شماره آنها ساخته اید در حالی که حالا نام شخص را در دست دارید نه شماره وی . یک راه بسیار ساده برای جبران این مشکل ذخیره اطلاعات بر اساس نام اشخاص است یعنی نام فایل اطلاعات هر شخص نام خودش باشد (فرض می کنیم هیچ نامی تکراری نباشد) . همین روند را ادامه دهید ، یک اطلاع در مورد یک شخص چند بار باید کپی شود شاید بگویید با توجه به حجم بسیار زیاد هارد دیسک های موجود این کار ایرادی نداشته باشد ولی فرض کنید تمام آنهایی که ابتدای شماره تلفنشان 2 است را باید به 22 تغییر دهید . ببینید حجم عملیات چند برابر می شود !!

پس احتمالاً این روش در ذخیره سازی اطلاعات مناسب نیست و کپی اطلاعات یک شخص در چندین فایل کار درستی نمی باشد . می توانیم برای هر شخص یک فایل داشته باشیم ولی در مشخصات آن از یک ساختار مشخص استفاده کنیم . مثلاً خط اول اسم ، خط دوم نام فامیل ، خط سوم تلفن و خط چهارم آدرس ، نام فایل هم شماره او باشد . حال می خواهیم شماره شخص خاصی که نام فامیلش را می دانیم پیدا کنیم . با استفاده از System.IO لیست تمام فایلها را پیدا می کنیم و در خط دوم تک تک آنها دنبال نام فامیل مورد نظر می گردیم . با استفاده از مطالب فصلهای قبلی این کار را به راحتی می توانید انجام دهید . همان طور که در اینجا مشاهده کردید مشکل تعداد زیاد کپی ها از یک اطلاعات حل شد .

چرا نتوانیم آن اطلاعات را در یک فایل داشته باشیم ؟؟ فرض کنید ساختار فایل ما به این شکل باشد که قبل از نام دانشجو در یک خط بالاتر شماره آن ذکر شده باشد . حالا جستجوی ما نیز در یک فایل است و هر خط حاوی اطلاعات هنرجویی است که در خط های قبلی شماره آن ذکر شده . برای مثل به شکل زیر نگاه کنید

```
0000100
Hamed
Banaei
0911202020202
Tehran , Iran
0000101
Amir
Ehsani
0911202020203
Tehran , Iran
```

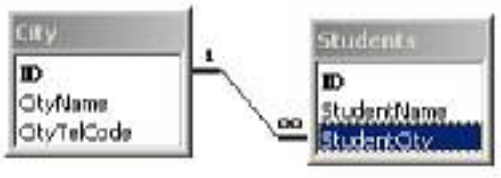
ساختار فوق را با کمی تغییرات می توان به شکل زیر تبدیل کرد

```
7#0000100#5#Hamed#6#Banaei#13#0911202020202#13#Tehran , Iran#
```

این ساختار چی مفهومی دارد ؟ عدد 7 اول یکی شماره هنرجو هفت حرف است ، عدد 5 یعنی نام Hamed پنج حرف است و بقیه هم مانند آن . از کاراکتر # هم برای جدا سازی این قسمت ها از هم استفاده کردیم . با این شکل ذخیره سازی می توانیم اطلاعات هر شخص را با توجه به شکل و فرمت کلی که ایجاد کردیم در هر خط از فایل هنرجو ها قرار دهیم . توجه کنید که انواع این روش ها در یک درس 3 واحدی در رشته نرم افزار به نام ذخیره و بازیابی بررسی می شود . شکل کلی این فرمت همانند یک جدول است که مشخص شده هر خاصیت چه طولی دارد . همچنین به همین روش می توان نوع اطلاعات هر خاصیت را هم مشخص کرد . مثلاً نام ها فقط کاراکتر هستند و شماره هنرجو فقط عدد . در اصطلاح به هر کدام از این خاصیت ها مثل نام ، شماره ، هنرجو و ... یک Field گفته می شود .

```
7          5          6          13          13
```

هر فیلد اگر کاراکتری باشد طول مشخص و اگر عددی باشد از نوع byte تا Double می تواند باشد . البته data type های فیلد ها محدود به این دو حالت نیست و دارای تعداد بیشتری از انواع مختلف می باشد . به فیلد شماره هنرجو فیلد اصلی می گویند . اطلاعات این فیلد تکراری نخواهند بود . اگر بگوییم که اطلاعات شخصی را می خواهیم که شماره مشخص دارد مستقیماً سراغ یک ردیف از جدول می رویم . به این فیلد اصلی ، کلید هم گفته می شود . در اکثر نرم افزارهای ایجاد کننده بانک های اطلاعاتی می توانیم دو یا چند فیلد را به صورت مشترک به عنوان primary key تعریف کنیم . به هر ردیف از این جدول یک record گفته می شود . در یک بانک اطلاعاتی معمولاً تعداد زیادی جدول وجود دارد . مثلاً نام هنرجوها ، نام درس ها و یا نام اساتید که هر کدام دارای تعدادی field و رکورد است .
بانک های اطلاعاتی مفاهیم و مدل های مختلفی دارند که کاربردی ترین آنها مدل رابطه ای است . این مدل پایه قوی در نظریه مجموعه ها در ریاضی دارد و اکثر کارهایی که شما در مجموعه ها در ریاضی انجام می دهید را می توانید در اینجا هم استفاده کنید . در بانک های رابطه ای بین بعضی از فیلدها در یک جدول با فیلدهایی در جدول دیگر می تواند روابطی وجود داشته باشد . برای بیان مفهوم یک مثال می زنم . فرض کنید در همان جدول هنر جو ها 2 فیلد شهر و پیش شماره وجود داشته باشد . مثلاً هنرجویی که از تهران است در فیلد شهر برایش تهران و در فیلد پیش شماره 021 نوشته شده است . گروه دیگری از مشهد هستند و برایشان 0511 ذکر شده و فرض کنید تعداد این رکورد ها 500 عدد باشد . فرض کنید که پیش شماره مشهد از 0511 به 0512 تغییر کند . شما باید اطلاعات 500 رکورد را تغییر دهید این کار بسیار زمان گیر است البته 500 رکورد عدد بسیار کمی است فرض کنید 5 میلیون رکورد برای تغییر وجود داشته باشد! راهی که برای حل این مشکل وجود دارد اضافه کردند جدول دیگری با این ستون هاست نام / کد شهر/ شماره . شماره فیلد یکتای ما برای هر شهراست . به جای این که در هر رکورد هنرجو نام شهر و کد آن ذکر شود فقط شماره آن گفته می شود . حال اگر تغییری لازم باشد شما یک بار آن را در جدول شهر ها اعمال می کنید و نه 500 بار در جدول هنرجو ها . شکل زیر دیاگرامی از این حالت است و مفهوم آن این است که به ازای یک رکورد شهر چندین رکورد در هنرجوها می توانیم داشته باشیم ، به فیلد شهر در جدول هنرجو یک کلید خارجی می گوییم . البته نوع فیلد شهر و شماره باید یکسان باشد زیرا مقدار شماره در فیلد شهر قرار می گیرد .



SQL[10]

یکی از برتری های بانکهای اطلاعاتی به سیستم های مبتنی بر فایل، امکان استفاده از SQL است.
SQL یک فرا زبان است که بوسیله آن پرس و جوهایی (این پرس و جو فقط به معنی سوال و جواب نیست بلکه شامل تغییر دادن نیز می شود.) از بانک اطلاعاتی انجام می دهند . برای مثال وقتی شما می خواهید ببینید نام یک هنرجو با شماره خاص چیست این جمله را باید به SQL بیان کنید . در SQL برای کار با داده ها چهار دستور کلی وجود دارد که عبارتند از : Insert Into، Select ، Update و Delete . ما در دات نت می توانیم این دستورات را با واسطه هایی که در ادامه ذکر خواهد شد در برابر یک بانک اطلاعات بکار گیریم .

دستور Insert با طرح کلی INSERT INTO talbeName (fields) VALUES (,,,,) برای اضافه کردن یک رکورد به جدول مورد نظر است . مثل :

```
INSERT INTO City (CityName) VALUES ('Tehran')
```

دستور پرکاربردی که باید در استفاده از آن مهارت پیدا کرد SELECT است . با دستور SELECT می توانید اطلاعات مورد نظر خود را از بانک فراخوانی کنید . بررسی کامل این دستور خود احتیاج به یک کتاب جداگانه دارد ، ما در اینجا فقط پارامترهای ضروری از آن را توضیح می دهیم . شکل کلی SELECT مانند زیر است :

[10] Structural Query Language

SELECT filedNames FROM tableNames WHERE statements

fieldnames نام فیلدهایی از table های یاد شده جلوی FROM است که می خواهید در خروجی باشد . از * برای زمانی استفاده می کنند که همه فیلدها را در خروجی بخواهیم . بین نام هر فیلد از , استفاده می کنند . همچنین نام فیلدهای خروجی را نیز می توان تغییر با استفاده از کلمه AS تغییر داد .
در خروجی می توان مجموع ، تعداد ، میانگین و بعضی خواص دیگر یک فیلد را نیز به تابع فراخواننده باز گرداند . خطوط زیر مثال هایی از به ترتیب جمع ، تعداد و میانگین است .

```
SELECT SUM(CityCode) FROM City
SELECT COUNT(CityName) FROM City
SELECT AVG(CityCode) FROM City
```

tableNames نام جداولی است که از آنها می خواهیم اطلاعات را دریافت کنیم . در مثال زیر می خواهیم همه اطلاعات شهرها به همراه نام هنرجو ها را استخراج کنیم :

```
SELECT City.*,Students.StudentName FROM City,Student
```

دقت کنید که هنگامی که از چند جدول استفاده می کنیم آنها در هم ضرب دکارتی می شود و نتیجه می توانند جدول بسیار بزرگی شود و کامپیوتر وقت زیادی را صرف ساخت آن بکند .
WHERE statement قسمتی است که در آن شرط مورد نظر را برای خروجی می نویسیم . برای مثال در خط زیر می خواهید تمام نام ها و شماره های شهرهایی را که شماره آنها بزرگتر مساوی 2 است را استخراج کنیم .

```
SELECT CityName,ID FROM City WHERE ID >= 2
```

یا

```
SELECT CityName FROM City WHERE CityName LIKE 'Te%'
```

این دستور تمام شهرهایی را می دهد که اول آنها Te قرار دارد . برای بررسی شط مساوی بودن باید از City = 'Te' استفاده کرد ولی برای اینکه قسمتی از فیلد کاراکتری را بررسی کند باید از Like استفاده کنیم . عبارت LIKE '%Te%' تمام نام شهرهایی را بر می گرداند که در آنها از Te استفاده شده و جای Te در آن مهم نیست .
فرض کنید جدولی داریم مطابق شکل که اطلاعات هنرجوها را نگهداری می کند و نامش Students است .

ID	StudentName	StudentCity
1	Ali	2
2	Hamed	2
3	Amir	3

جدول دیگری هم مطابق شکل زیر حاوی اطلاعات شهرها و نامش Cities است .

ID	CityName
2	Tehran
3	Shiraz

حال می خواهیم نام کسانی را که در شهر تهران زندگی می کنند را پیدا کنیم .

```
SELECT StudentName FROM students,cities WHERE StudentCity = Cities.ID AND CityName = 'Tehran'
```

در این مثال از دو جدول استفاده کردیم . در قسمت WHERE از دو شرط استفاده کردیم و علت وجود AND بین دو شرط همیشه وقتی نتیجه کلی درست است که هر دو شرط درست باشد . البته قسمت شرطی این دستور را می توانستیم با استفاده از SELECT های متداخل نیز بنویسیم بدین شکل

```
SELECT StudentName FROM students WHERE StudentCity IN ( SELECT id FROM Cities WHERE CityName = 'Tehran'
```

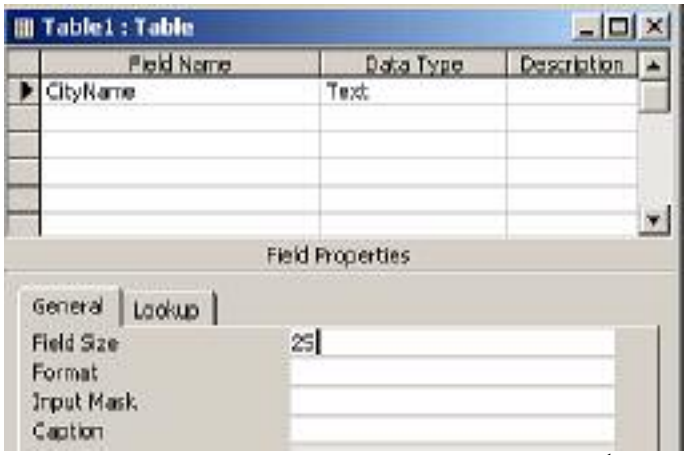
نوشتن SELECT ی که با کمترین بارگزاری روی کامپیوتر نتیجه دلخواه را بدهد بعضاً کار بسیار دشواری است . گاهاً با نوشتن یک SELECT یک کار بزرگ آماری را انجام می دهند . ما در کتاب پیشرفته بیشتر در مورد SELECT صحبت خواهیم کرد .
دو دستور Update و Delete به ندرت مستقیماً در دات نت استفاده می شود لذا آنها را بعداً در کاربرد خواهیم دید .
حال برای اینکه از بحث های تئوری خارج شویم نحوه کار با نرم افزار Access را بررسی می کنیم . شما برای انجام تمرین های این کتاب احتیاج به Access 2000 یا XP یا 2003 دارید .

کار با MS Access

نرم افزار Access یکی از ساده ترین راه ها برای استفاده و ایجاد بانک اطلاعاتی در حد مصارف کوچک است(در پایگاه داده، "کوچک" می تواند به معنی یک پایگاه داده دو گیگابایتی باشد) . برای شروع Access را اجرا کنید و از منوی فایل ، New را زده و سپس Blank Database را انتخاب کنید . مسیری را که می خواهید دیتابیس ذخیره شود انتخاب و نام TestDB را برای آن بنویسید .

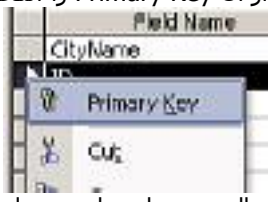
پسوند فایلهایی که Access برای دیتابیس می سازد mdb است . در قسمت tables روی Create Table in Design View کلیک کنید تا پنجره ای مطابق شکل زیر باز شود .

می خواهیم جدول شهر ها را ایجاد کنیم پس احتیاج به 2 فیلد داریم . در قسمت Field Name عبارت CityName را تایپ کنید . در جلوی آن ستون Data Type قرار دارد که انواع داده های مجاز در آن نمایش داده شده است . مناسب ترین نوع داده برای نام شهر Text است . وقتی Text را انتخاب کردید همان طور که در شکل دیده می شود می توانید مشخصات آن Text را تغییر دهید . طول آن را به 25 تغییر دهید زیرا احتمالاً نام هیچ شهری بیش از 25 حرف نباشد .



فیلد دیگر مورد نیاز ID است . ID در اینجا حکم کلید اصلی را دارد . برای این که این فیلد ایجاد شود دو راه وجود دارد .

راه اول استفاده از همین روش که در مورد CityName دیدید . یعنی تایپ ID ، انتخاب AutoNumber برای نوع داده آن و برای اینکه به فیلد کلید تبدیل شود روی دکمه سمت چپ آن کلیک سمت راست می کنید و از آن Primary Key را انتخاب می کنید تا علامت کلید در کنار آن نمایش داده شود .



حال می توانیم این جدول را با نام Cities ذخیره کنیم .

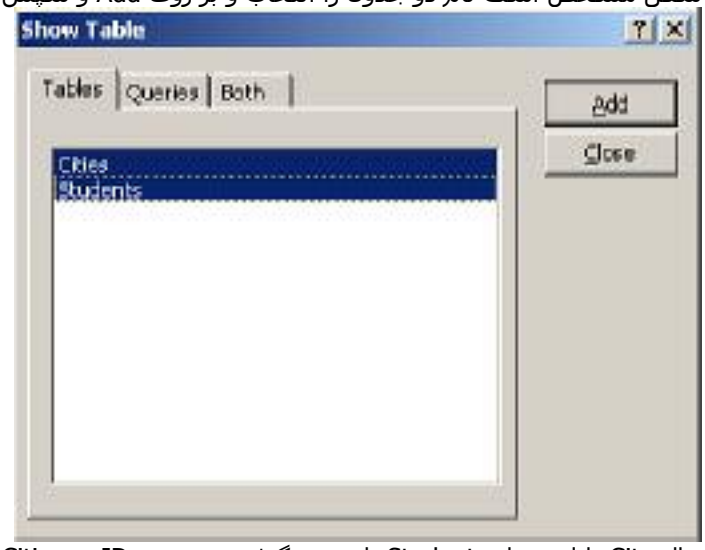
روش دوم این است که وقتی CityName را ایجاد کردید سعی کنید جدول را ذخیره کنیم . برنامه Access از ما سوال می کند که به علت نداشتن کلید اصلی آیا می خواهید خود برنامه این کار را بکند در صورت تایید ، فیلد ID به طور خودکار ساخته خواهد شد .

حال جدول ما همان طور که در شکل دیده می شود به لیست اضافه شده است . روی آن Double click کنید تا برنامه آن را باز کند و بتوانید اطلاعات را وارد آن کنید . شکلی مانند شکل زیر نمایان می شود . در قسمت CityName عبارت Tehran را تایپ کنید مشاهده می کنید که ID هم عدد 1 را به خود می گیرد اگر عبارت shiraz را برای رکورد دوم بنویسید شماره ID برابر 2 خواهد شد .

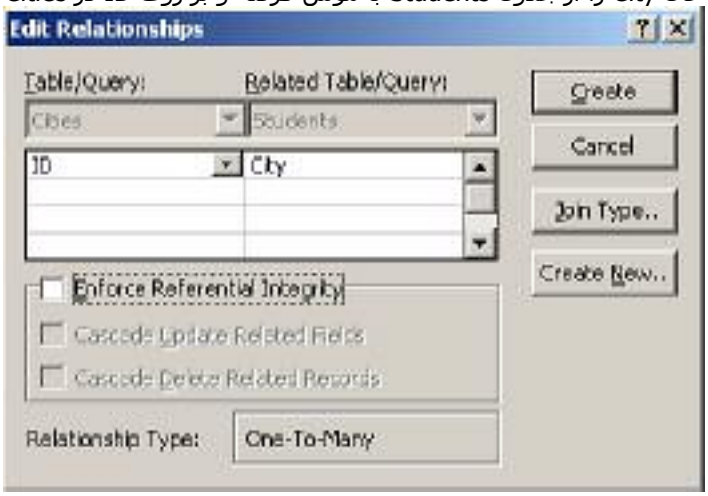
حال باید دومین جدول که همان هنرجوها است را بسازیم ، دوباره روی Create Table in Design View کلیک کنید ، این بار sName را برای نام هنرجو از نوع Text با طول 50 و City را از نوع Number انتخاب کنید . نوع داده Memo برای متنهایی با طول بیشتر از 255 کاراکتر است . برای اینکه ID به جدول ما اضافه شود این بار مستقیماً جدول را با نام Students ذخیره کنید .

حال جدول جدید ما در کنار جدول قبلی قرار گرفته است . قبل از اینکه اطلاعات هنرجو را وارد کنید بهتر است رابطه کلید خارجی را بین دو جدول بر روی فیلد های ID در جدول شهر و City در جدول هنرجو برقرار

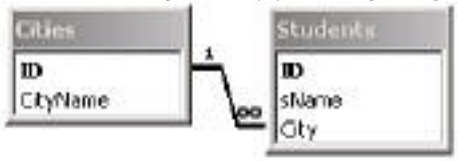
کنید . در این رابطه به ازای یک شهر در جدول شهرها بینهایت هنرجو در آن شهر می تواند وجود داشته باشد ، به عبارت دیگر به ازای یک رکورد در جدول شهر بینهایت رکورد در جدول دانشجو می توان اضافه کرد . برای انجام این کار به منوی Tools بروید و Relationships را انتخاب کنید . در آن قسمت همان طور که در شکل مشخص است نام دو جدول را انتخاب و بر روی Add و سپس بر روی Close کلیک کنید .



حال City را از جدول Students با موس گرفته و بر روی ID در Cities قرار داده تا شکل زیر ظاهر شود .

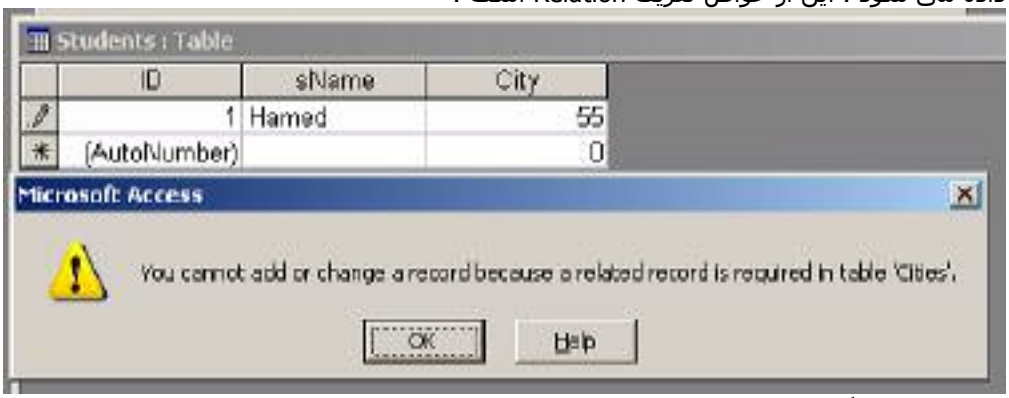


بر روی Enforce Referential Integrity کلیک و سپس روی Cascade Delete Related Record کلیک کنید . شکل جداول مانند زیر می شود .



با استفاده از این تکنیک هر موقع رکوردی از شهرها حذف شود که تعدادی از رکورد های هنرجو ها از آن استفاده کرده بودند آن هنرجو ها هم حذف می شوند . این کار برای حفظ یکپارچگی دیتابیس لازم است در مورد جامعیت یا integrity در کتاب پیشرفته توضیحاتی خواهیم داد . برای اطلاعات بیشتر در این مورد به کتاب دیتابیس آقای روحانی رانکوهی نیز می توانید مراجعه کنید .
پنجره Relationships را close و آن را ذخیره کنید . حال بر روی جدول Students دو بار کلیک کنید تا اطلاعات را وارد کنید . در sname عبارت Hamed و در City عدد 55 را وارد کنید ، حال سعی کنید به رکورد بعدی برای اضافه کردن بروید . Access به شما پیغام خطا می دهد زیرا رکوردی در جدول شهرها با شماره 55 ذخیره

نشده ، حال به جای 55 عدد 1 را که همان Tehran است وارد کنید . این بار به شما اجازه ثبت رکورد جدید داده می شود . این از خواص تعریف Relation است .



حال یک نام دیگر با همان شماره برای شهر مانند Amir و یک اضافه کنید . حالا پنجره Students را ببندید و جدول شهرها را باز کنید . همان طور که در شکل مشخص است در کنار هر کدام از رکورد ها یک علامت + درج شده که اگر بر روی آن کلیک کنید تمامی رکوردهایی را که مرتبط به آن رکورد خاص در شهرهاست نمایش داده می شود .

کار با نرم افزار Access در همین حد برای ما کافی است بخش عمده کار با بانک های اطلاعاتی در خود VB.NET انجام خواهد شد و ما با Access صرفاً فایلهای دیتابیس را می سازیم . گزارش گیری ، اضافه کردن و تغییر داده های همگی در VB.NET انجام می شود .

کار با بانک اطلاعاتی در VB.NET

برای کار با بانک های اطلاعاتی سه راه کلی وجود دارد . راه اول استفاده از wizard هاست که اصلاً توصیه نمی شود زیرا در یک برنامه حرفه ای شما تمام اجزای برنامه را خودتان باید تنظیم کنید . راه دوم استفاده از Data Binding است و راه سوم ساختن همه اجزای لازم توسط خود شما . قبل از آن باید کمی در مورد کلاسها و مفاهیمی که در دات نت برای کار با دیتابیس وجود دارد آشنا شویم .

ADO

به گروهی از DLLها و کلاس ها می گویند که مسئول کار با داده ها هستند . پیش از دات نت نیز ADO وجود داشت ولی در دات نت از ADO.NET استفاده می کنیم که با ADO قبلی تفاوتهایی دارد .

Connection String : برای اتصال به بانک های داده باید نوع و مسیر آن و گاهی username و password و بعضاً اطلاعات بیشتری را مشخص کرد که این اطلاعات را با استفاده از Connection String به اجزای ADO ارسال می کنند . برای مثال این یک نمونه Connection String برای Access است :

```
Provider=Microsoft.jet.oledb.4.0;data source=I:\testdb.mdb;
```

Connection : این کلاس و آبجکت هایی که از آن ساخته می شود مسئول برقراری ارتباط با دیتابیس است برای مثال به خطوط زیر توجه کنید :

```
Dim strCon as String = " Provider=Microsoft.jet.oledb.4.0;data source=I:\testdb.mdb;"
Dim Con as New OleDB.OleDBConnection(strCon)
Con.Open()
```

در خط سوم connection ی که در خط دوم به دیتابیس ایجاد شده باز و آماده کار می شود .
Data Adapter : بوسیله این کلاس می توانیم آبجکتهایی بسازیم که با استفاده از Connection اطلاعات را از دیتابیس دریافت کنند . یکی از پارامترهای Data Adapter در هنگام Construction دستور Select است ، مانند نمونه زیر :

```
Dim strSQL as String = "SELECT * FROM students"
Dim Adapter as New OleDB.OleDBDataAdapter(strSQL,con)
```

Con در اینجا همان است که در توضیحات connection دیدید . اگر overload های Data Adapter را هنگام ساختن نگاه کنید می بینید که می توان مستقیماً Connection String را به جای یک آبجکت Connection به کار برد تا خودِ DataAdapter یک Connection از روی Connection String بسازد .
Command : با استفاده از این کلاس می توان دستورات SQL را اجرا کرد . Command را می توان در انواع مختلف مثلاً برای update ، delete یا insert into استفاده کرد . Command را هم می توان مستقیماً با یک Connection به کاربرد هم آن را به property هایی به نام های مختلف در Data Adapter نسبت داد . مثلاً

```
Dim Comm as New OleDB.OleDBCommand("INSERT INTO students (sname) VALUES ('Ahmad')",Con)
```

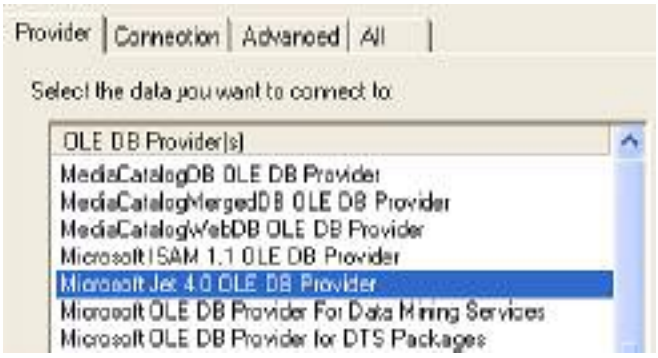
در این حالت از Command و Connection برای اضافه کردن Ahmad در جدول هنرجوها استفاده شده است .

Data Row : هر رکورد می تواند در یک DataRow قرارگیرد . Data Row دارای خواص و متدهای گوناگون می باشد . خاصیت Item حاوی محتوای هر فیلد آن است و متد Delete آن را حذف می کند . هر رکورد دارای یک وضعیت است اعم از دست نخورده ، حذف شده ، ویرایش شده و .... که در مورد هر رکورد از خاصیت RowState آن می توان نوع آن رکورد را فهمید .

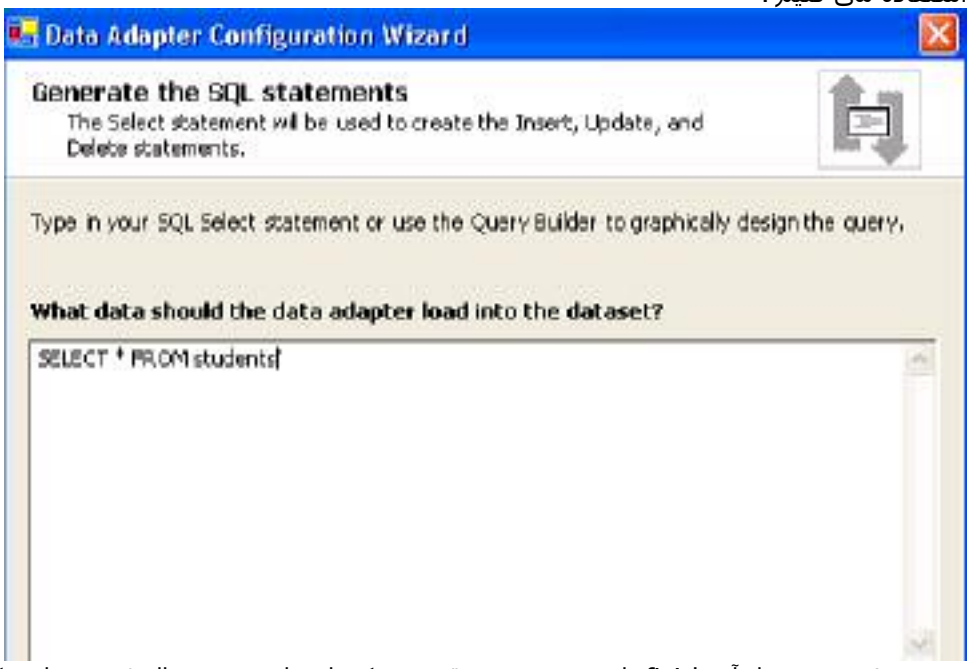
Data Column و Data Row : به ترتیب ستون ها و سطر های یک جدول هستند .
Data Table : همان طور که از نامش مشخص است حاوی کل اطلاعات یک جدول با رکورد های آن می باشد .
Data Set : کلاسی است که می تواند درون خود چند Table با تمام روابط بین آنها را ذخیره کند . معمولاً برای کار با داده ها از متد Fill در Data Adapter و DataSet استفاده می شود . در این مورد در ادامه توضیح داده خواهد شد .

روش Data binding و استفاده از Data Grid

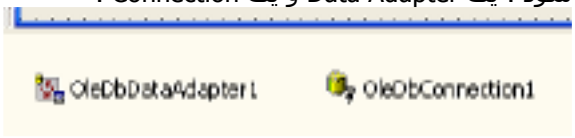
در روش Bind کردن دیتابیس به یک سری از کنترلها متصل می شود و در برابر بعضی از حرکات واکنش می دهد مانند جلو رفتن در رکورد ها .
برای شروع در Toolbox از Data ِtab یک OleDBDataAdapter برداشته و بروی فرم خود قرار دهید . پنجره ای باز می شود تا از شما سوالاتی پرسیده شود . ابتدا باید دیتابیس خود را مشخص کنید . قبل از این کار فایل TestDB را که قبلاً ساخته بودیم را به شاخه bin در همین پروژه کپی کنید . حال مانند شکل بر روی new connection کلیک کنید و از قسمت Provider گزینه Microsoft Jet Oledb 4 را و در قسمت Connection فایل مورد نظر را انتخاب و بر روی OK کلیک کنید .



حالا next را دو بار بزنید ، در اینجا باید Select مورد نظر را بنویسید ، ما از SELECT * FROM students استفاده می کنیم .

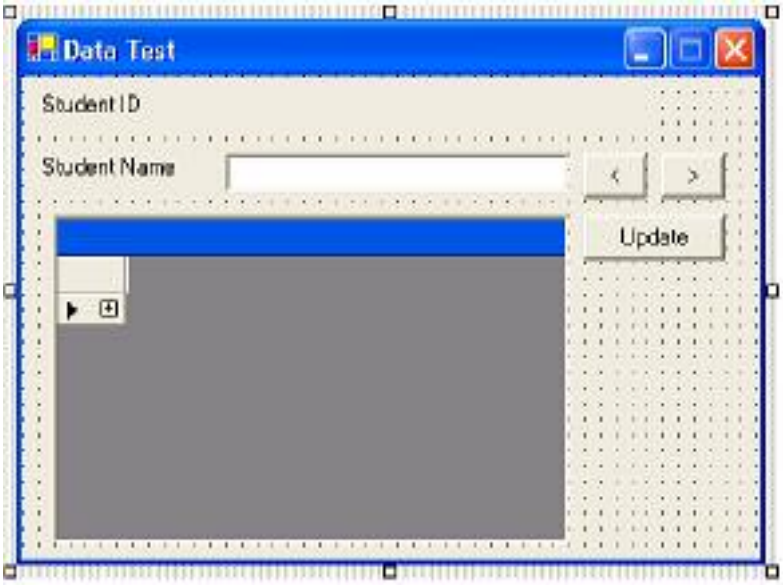


سپس next و بعد از آن finish را بزنید . بر روی قسمت کنترل های غیر ویژوال فرم شما دو کنترل دیده می شود . یک Data Adapter و یک Connection .



بر روی Adapter کلیک سمت راست کنید و سپس Generate Data Set را بزنید .

حال کنترل جدیدی از نوع Data Set در کنار آنها قرار گرفت . اطلاعات مورد نظر ما از طریق Connection و Adapter به Data Set منتقل می شود . فرمی مانند شکل زیر با مشخصات جدول زیر بسازید .

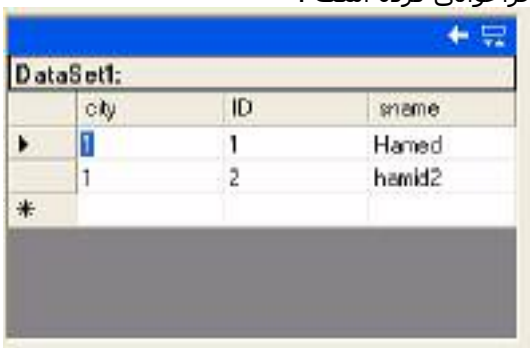


نام کنترل	نوع کنترل	بعضی مشخصات
lblID	Label	Text = "Student ID"
lblStudentName	Label	Text = "Student Name"
txtName	Textbox	
btnPrev	Button	Text = "<"
btnNext	Button	Text = ">"
btnUpdate	Button	Text = "Update"
DataGrid1	Data Grid	DataSource = Dataset11

اکنون قبل از اینکه textbox و label را هم bind کنیم یک بار این کار را با Data Grid آزمایش می کنیم . در سابروتین Load از فرم ، دستور پر کردن Dataseet را بوسیله Adapter می دهیم .

```
OleDbDataAdapter1.Fill(DataSet11, "Students")
```

همان طور که مشخص است پارامتر اول نام DataSet می باشد و پارامتر دوم نام جدول . چون در هر DataSet چندین جدول می توان ذخیره کرد لذا می توان نامی برای هر کدام در نظر گرفت . حال برنامه را اجرا کنید . شکلی مانند شکل زیر خواهید دید . برنامه بدون اینکه شما کد زیادی بنویسید دیتاهای شما را فراخوانی کرده است .



حال سعی می کنیم textbox و label را هم به بانک داده متصل کنیم . سابروتینی با نام PrevRecord با کد زیر را به پروژه اضافه کنید .

```
Private Sub PrevRecord(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnPrev.Click
    intRecordCount = Me.BindingContext(DataSet11, "Students").Count
    intCurrentPos = Me.BindingContext(DataSet11, "Students").Position + 1
```

```
    If intRecordCount <= 1 Then
       btnPrev.Enabled = False
       btnNext.Name = False

       Exit Sub
    End If
    If sender.Equals(btnPrev) Then
       If intCurrentPos > 1 Then
          Me.BindingContext(DataSet11, "Students").Position -= 1
       Else
          btnPrev.Enabled = False
       End If
    End If
    If sender.Equals(btnNext) Then
       If intRecordCount > intCurrentPos Then
          Me.BindingContext(DataSet11, "Students").Position += 1
       Else
          btnNext.Enabled = False
       End If
    End If
    intCurrentPos = Me.BindingContext(DataSet11, "Students").Position + 1
    If intRecordCount = intCurrentPos Then
       btnNext.Enabled = False
    Else
       btnNext.Enabled = True
    End If
    If intCurrentPos = 1 Then
       btnPrev.Enabled = False
    Else
       btnPrev.Enabled = True
    End If
End Sub
```

در load از فرم خطوط زیر را بنویسید .

```
    OleDbDataAdapter1.Fill(DataSet11, "Students")
    txtName.DataBindings.Add("Text", DataSet11, "Students.sName")
    lblID.DataBindings.Add("Text", DataSet11, "Students.ID")
    AddHandler btnPrev.Click, AddressOf PrevRecord
    AddHandler btnNext.Click, AddressOf PrevRecord
```

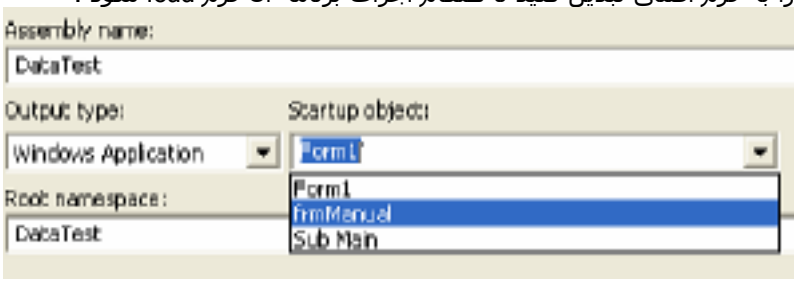
در سابروتین load خط دوم و سوم مسئول bind کردن به بانک هستند . پارامتر اول نام خاصیت متصل شده است ، پارامتر دوم نام DataSet و پارامتر سوم نام جدول و فیلد مورد نظر است . خط چهارم و پنجم هم رویداد Click هر دو Button را به PrevRecord منتقل می کند .
در سابروتین PrevRecord باید تعیین کنیم که اگر به رکورد آخر رسیدم کلید Next غیر فعال شود و همچنین در مورد کلید Prev و اولین رکورد .
حال برنامه را اجرا کنید و با استفاده از دکمه های < و> در دیتاها حرکت کنید ، مشاهده خواهید کرد که اطاعات در label و textbox نمایش داده می شوند . اگر دقت کرده باشید اطلاعات را می توانید تغییر دهید ولی آنها ذخیره نمی شود . برای ذخیره آنها در رویداد Click از btnUpdate این خط را بنویسید :

```
OleDbDataAdapter1.Update(DataSet11, "Students")
```

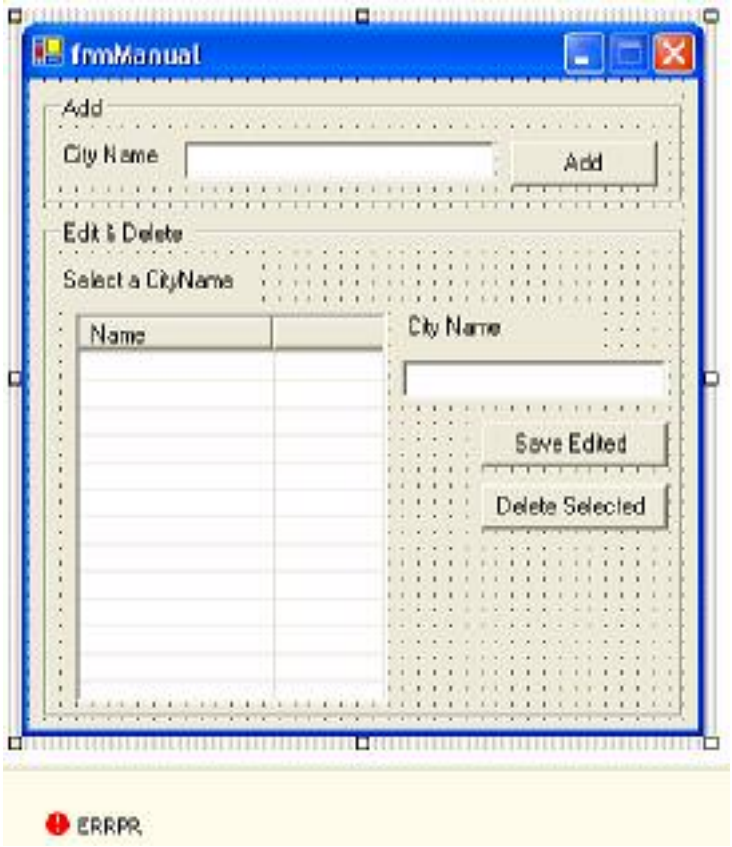
پارامتر اول نام Dataset و پارامتر دوم نام جدولی است که از طریق این Adapter باید update شود . اگر دوباره برنامه را اجرا کنید ودر اطلاعات تغییراتی بدهید بعد از زدن کلید update اطلاعات جدید جایگزین قدیمی ها می شود .
روش بعدی روشی است که خود شما باید همه این کارها را انجام دهید . در این مثال ما روش اضافه کردن ، ویرایش و حذف رکورد ها را بررسی می کنیم .

روش Manual

این روش را با یک مثال بررسی می کنیم . یک فرم دیگر ایجاد کنید و با استفاده از Property های پروژه آن را به فرم اصلی تبدیل کنید تا هنگام اجرای برنامه آن فرم load شود .



فرم را مانند شکل زیر و کنترل های آن را از روی جدول زیر طراحی کنید . با استفاده از group box و جدا سازی بخش های برنامه ، همه آنها را در یک فرم خواهیم داشت .



نام کنترل	نوع کنترل	توضیحات اضافی
btnAdd	Button	Text = "Add"
txtAddName	Textbox	
Groupbox1	GroupBox	Text = "Add"
GroupBox2	GroupBox	Text ="Edit && Delete"
txtEditName	Textbox	
btnSave	Button	Text = "Save Edited"
btnDelete	Button	Text= "Delete Selected"
LV	ListView	**

View = Details
MultiSelect = False
HideSelection = False
FullRowSelect = True
Gridlines = True

ErrorProvider ERRPR

** : در خاصیت Columns یک ستون ایجاد و برای Text آن مقدار Name تایپ کنید .

از ERRPR برای نمایش بعضی از خطاهای کاربر استفاده می کنیم .
کد برنامه را در زیر می بینید .

```
Private strConString As String = "provider=microsoft.jet.oledb.4.0;data source=" & Application.StartupPath & "\testdb.mdb;"
  Dim intSelectedID As Integer
  Private Sub FillListView()
    Try
      LV.Items.Clear()
      Dim Con As New OleDb.OleDbConnection(strConString)
      Con.Open()
      Dim CMD As New OleDb.OleDbCommand("Select * from Cities", Con)
      Dim Reader As OleDb.OleDbDataReader = CMD.ExecuteReader
      Do Until Not Reader.Read
        Dim Itm As New ListViewItem
        Itm.Text = Reader("cityname").trim
        Itm.Tag = Reader("ID")
        LV.Items.Add(Itm)
      Loop
      Reader = Nothing
      CMD.Dispose()
      Con.Close()
      Con.Dispose()
    Catch ex As Exception
      MessageBox.Show(ex.Message)
    End Try
  End Sub
  Private Sub frmManual_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
    FillListView()
  End Sub
  Private Sub btnAdd_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles btnAdd.Click
    Try
      If txtAddName.Text.Trim = "" Then
        ERRPR.SetError(txtAddName, "Fill This Textbox")
        Exit Sub
      End If
      Dim Adapter As New OleDb.OleDbDataAdapter("Select * from Cities", strConString)
      Dim DS As New DataSet
      Adapter.Fill(DS, "Cities")
      Dim dr As DataRow = DS.Tables("Cities").NewRow
      dr("CityName") = txtAddName.Text.Trim
      DS.Tables("Cities").Rows.Add(dr)
      Dim CMD As New OleDb.OleDbCommandBuilder(Adapter)
```

```
            Adapter.Update(DS, "Cities")
            DS.Dispose()
            CMD.Dispose()
            Adapter.Dispose()
            txtAddName.Text = ""
            FillListView()

        Catch ex As Exception
            MessageBox.Show(ex.Message)
        End Try
    End Sub
    Private Sub LV_DoubleClick(ByVal sender As Object, ByVal e As System.EventArgs) Handles
LV.DoubleClick
        Try
            If LV.SelectedItems(0).Text <> "" Then
                txtEditName.Text = LV.SelectedItems(0).Text
                intSelectedID = LV.SelectedItems(0).Tag
            End If
        Catch
        End Try
    End Sub
    Private Sub btnSave_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnSave.Click
        Try
            If txtEditName.Text.Trim = "" Or intSelectedID = 0 Then
                ERRPR.SetError(txtEditName, "Fill This Textbox")
                Exit Sub
            End If
            Dim Adapter As New OleDb.OleDbDataAdapter("Select * from Cities where id = " &
intSelectedID.ToString, strConString)
            Dim DS As New DataSet
            Adapter.Fill(DS, "Cities")
            Dim dr As DataRow = DS.Tables("Cities").Rows(0)
            dr("CityName") = txtEditName.Text.Trim
            Dim CMD As New OleDb.OleDbCommandBuilder(Adapter)
            Adapter.Update(DS, "Cities")
            DS.Dispose()
            CMD.Dispose()
            Adapter.Dispose()
            txtEditName.Text = ""
            intSelectedID = 0
            FillListView()
        Catch ex As Exception
            MessageBox.Show(ex.Message)
        End Try
    End Sub
    Private Sub btnDelete_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
btnDelete.Click
        Try
            If txtEditName.Text.Trim = "" Or intSelectedID = 0 Then
                ERRPR.SetError(txtEditName, "Fill This Textbox")
                Exit Sub
            End If
            If MessageBox.Show("Are you sure ?", Application.ProductName, MessageBoxButtons.YesNo) =
DialogResult.No Then Exit Sub
```

```
        Dim Adapter As New OleDb.OleDbDataAdapter("Select * from Cities where id = " & intSelectedID.ToString, strConString)
        Dim DS As New DataSet
        Adapter.Fill(DS, "Cities")
        DS.Tables("Cities").Rows(0).Delete()
        Dim CMD As New OleDb.OleDbCommandBuilder(Adapter)
        Adapter.Update(DS, "Cities")
        DS.Dispose()
        CMD.Dispose()
        Adapter.Dispose()
        txtEditName.Text = ""
        intSelectedID = 0
        FillListView()
      Catch ex As Exception
        MessageBox.Show(ex.Message)
      End Try
    End Sub
```

Connection String را که چند بار مورد استفاده قرار می گیرد به صورت یک متغیر عمومی در سطح کلاس تعریف شده . برای اینکه مسیر دیتابیس می تواند با جابجایی برنامه تغییر کند پس باید به نحوی مسیر آن پویا باشد . در اینجا فرض شده که دیتابیس در کنار فایل Exe برنامه است و از Application.StartupPath برای دستیابی به آن استفاده شده . متغیر عمومی دیگر intSelectedID است که برای ذخیره شماره رکوردی که از listview انتخاب شده به کار می رود .

انواع فراخوانی اطلاعات از یک دیتابیس

در این مثال دو نوع فراخوانی اطلاعات از دیتابیس داریم ، یک نوع با استفاده از DataReader و دیگری با استفاده از DataSet است . هر کدام از این روش ها محاسن و معایبی دارند .
DataReader : گاهی اوقات شما می خواهید اتصالتان به دیتابیس برقرار باشد و رکوردها را تک تک از دیتابیس فراخوانی کنید . در این حالت شما از حافظه برای نگهداری تمام رکوردها استفاده نمی کنید و این یک حسن برای این روش محسوب می شود و در مقابل آن شما یک Connection را دائماً برای آن با دیتابیس باید داشته باشید .
DataSet : در روش استفاده از DataSet موضوع کاملاً بر عکس DataReader است ، یعنی اتصال با دیتابیس بلافاصله بعد از انتقال دیتا به DataSet قطع می شود ولی تمام رکورد ها در حافظه نگهداری می شود . باید بررسی کرد برای برنامه های شما کدام حالت مفید تر است و یک راه کلی برای تمام موارد نمی توان در نظر گرفت .
تابع FillListView از روش DataReader استفاده و اطلاعات را در ListView اضافه می کند . ابتدا یک connection به دیتابیس ایجاد می کند سپس با استفاده از یک command ، Select_ مورد نظر را نوشته و آن را اجرا می کند ، نتیجه را در یک DataReader قرار می دهد و آن را در یک Loop تا زمانی که به انتها نرسیدیم صدا می زند .
از خاصیت Tag در هر کدام از آیتم های ListView برای ذخیره ID آن رکورد استفاده می کنیم . در کل ، خاصیت Tag برای این گونه موارد در نظر گرفته شده است .
دقت کنید بعد از loop حتماً connection را Close کنید تا بعداً از بروز مشکلات جلوگیری شود . البته در اکثر موارد وقتی یک سابروتین یا تابع به آخرین خط خود رسید تمام Object های موجود در آن از حافظه حذف می شوند ولی مواردی وجود دارد که این کار انجام نمی شود . این موضوع شبیه استفاده از Flush در فایلها است ، اگر شما Flush را صدا نزنید و Close کنید تمام اطلاعات در stream به فایل منتقل خواهد شد ولی بهتر است Flush زده شود !
برای اینکه هنگامی که این فرم نمایش داده می شود اطلاعات مورد نظر ما در ListView وجود داشته باشد ساده ترین راه فراخوانی FillListView در رویداد Load از فرم است .
در رویداد Double Click از ListView کدی نوشته شده است که رکوردی که روی آن double click شده را در بخش ویرایش نمایش می دهد .

اضافه کردن رکورد به دیتابیس

دو روش کلی برای این کار وجود دارد .
استفاده از Command : با استفاده از کلاس Command و Connection و استفاده از دستور INSERT INTO در SQL می توانیم اطلاعات را وارد دیتابیس کنیم ، مانند کد زیر :

```
Dim Con As New OleDb.OleDbConnection(strConString)
Con.Open()
Dim Command As New OleDb.OleDbCommand("INSERT INTO Cities (cityname) VALUES ('" & txtAddName.Text.Trim & "')", Con)
Command.ExecuteNonQuery()
Command.Dispose()
Con.Close()
Con.Dispose()
```

این روش سریعی است ولی بعضاً با فارسی مشکلاتی دارد .

استفاده از DataSet : روش دیگر برای اضافه کردن به دیتابیس استفاده از DataSet و DataAdapter است ، مانند کد زیر :

```
Dim Adapter As New OleDb.OleDbDataAdapter("Select * from Cities", strConString)
Dim DS As New DataSet
Adapter.Fill(DS, "Cities")
Dim dr As DataRow = DS.Tables("Cities").NewRow
dr("CityName") = txtAddName.Text.Trim
DS.Tables("Cities").Rows.Add(dr)
Dim CMD As New OleDb.OleDbCommandBuilder(Adapter)
Adapter.Update(DS, "Cities")
DS.Dispose()
CMD.Dispose()
Adapter.Dispose()
```

در این حالت ابتدا DataSet را با مقادیری که از Select در Adapter بازگشته است پر می کرده سپس یک DataRow جدید با همان مشخصات ِ جدول مورد نظر ساخته ، اطلاعات لازم را وارد آن می کنیم و در نهایت آن را به DataSet اضافه می کنیم و در آخرین مرحله با استفاده از متد update از adapter آن را به دیتابیس برای بروزرسانی بر می گردانیم .
در مدلی که از toolbox یک Adapter بر روی فرم قرار می دادیم خود برنامه برای تمام command های Adapter (که در اینجا Insert آن لازم است) را می ساخت ولی در اینجا خودمان باید این کار را انجام دهیم . برای ساختن این command ها دو را وجود دارد ، راه سخت تر این است که همه Command های مورد نظر را با استفاده از کلاس command بسازیم و آنها را به خاصیت های مختلف آن در Adapter نسبت دهیم . و راه ساده تر استفاده از کلاس Command Builder مانند مثال است .
مشکلی که می تواند در روش DataSet نمایان شود در مرحله دریافت اطلاعات برای DataSet است . در این مرحله اگر از همین * SELECT استفاده شود ، کل دیتابیس فراخوانی می شود و اگر تعداد رکورد ها زیاد باشد این بسیار برنامه را کند می کند . برای این مشکل یک راه ساده این است که در ادامه SELECT از شرطی استفاده کنیم که هیچ وقت درست نباشد ، مانند SELECT * FROM Students WHERE ID = -1 این شرط همیشه اشتباه است و هیچ رکورد بازگردانده نخواهد شد ولی Schema ی جدول به Dataset منتقل می شود .
ویرایش و حذف نیز مانند روش اضافه کردن است با این تفاوت که دیگر Datarow به Dataset اضافه نمی شود و در شرط ِ Select ِ Adapter هم شرط مورد نظر خودمان را اضافه کردیم .

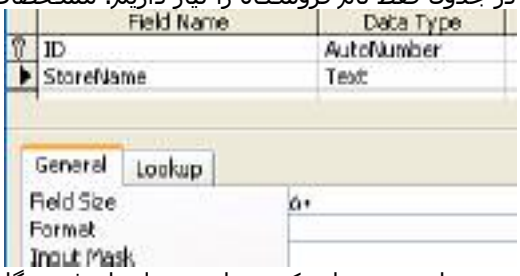
همان طور که مشاهده کردیم یک Dataset دارای خاصیتی به نام Tables است . شما با استفاده از Adapter برای هر جدول می توانید بیش از یک جدول در یک Dataset قرار دهید . برای مثال در این پروژه می توانستیم اطلاعات هنرجوها را نیز با استفاده از یک Adapter دیگر به Dataset منتقل کنیم . شماره این جداول در Tables ِCollection از صفر شروع می شود ، برای فراخوانی هر جدول هم از نام آن و هم از شماره اش می توان استفاده کرد که این شماره بر حسب ترتیبی است که Adapter آن را به Dataset اضافه کرده .
هر جدول در Dataset دارای خاصیتی به نام Rows است که حاوی نام رکوردهای جدولی است که از Adapter دریافت شده . بعضی از خاصیت های مفید Rows را در جدول زیر مشاهده می کنید .

نام	توضیحات
Count	تعداد رکورد های درون آن جدول را بر می گرداند.
.Item	برای دریافت اطلاعات فیلد مورد نظر است .
Columns	اطلاعات در مورد فیلدهای جدول است .

استفاده از بانکهای اطلاعاتی در برنامه بستنی فروش

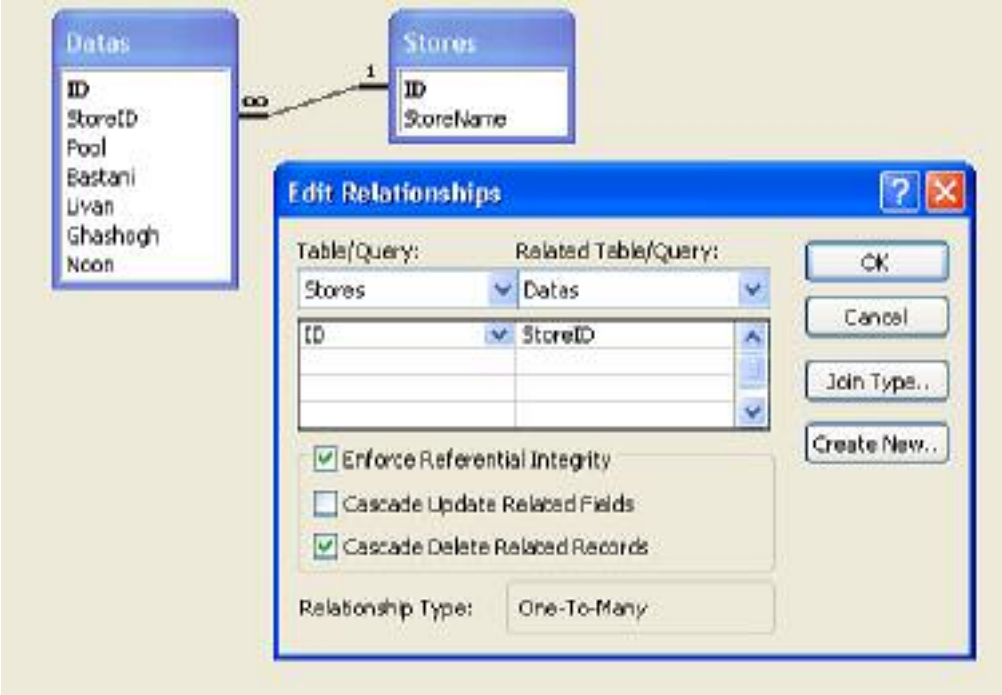
برای اینکه از بانکهای اطلاعاتی برای ذخیره اطلاعات بستنی فروشی استفاده کنیم به غیر از طراحی جداول بانک اطلاعاتی احتیاج به کمی تغییراتی در کد برنامه بستنی فروشی داریم.
یک دیتابیس خالی با نام dbBastani در شاخه Bin از پروژه بستنی فروش ایجاد کنید .
با توجه به مساله بستنی فروشی در این بانک احتیاج به یک جدول داریم که در آن نام فروشگاه و دیگر اطلاعات قرار می گیرد ، اما با یک دوراندیشی فرض می کنیم که بخواهیم تاریخ هر فروش و شخص خریدار را هم برای هر فروش ذخیره کنیم (تمرین برای شما) ، در آن صورت یک جدول کافی نیست زیرا اگر فقط یک جدول داشته باشیم نام فروشگاه به تعداد فروشها تکرار می شود برای رفع این تکرار در یک جدول نام فروشگاه ها و در جدول دیگر اطلاعات مربوط به فروش و موجودی آن ها را ذخیره می کنیم. نام این جدول را Stores قرار دهید .

در جدول فقط نام فروشگاه را نیاز داریم. مشخصات جدول اول را در شکل زیر می بینید :



در جدول دوم تعداد رکورد ها به تعداد نام فروشگاه هاست. نام این جدول را Datas قرار دهید . در شکل زیر مشخصات فیلدهای این جدول را می بینید :

برای اینکه هرگاه یک فروشگاه حذف شد اطلاعات آن نیز حذف شود در قسمت Relation ها یک رابطه بین فیلد کلید در و StoreID برقرار کرده و قسمت Delete آن را فعال می کنیم.



حال باید کدهای لازم برای کار با دیتابیس را به پروژه بستنی فروش اضافه کنیم. روشی که استفاده خواهیم کرد لزوماً پر سرعت ترین روش نخواهد بود بلکه می خواهیم ببینیم چگونه می توان از بانکهای اطلاعاتی در برنامه بستنی فروش و کلاس های آن استفاده کرد.
از کلاس Bastani شروع می کنیم. برای اینکه کل عبارت Connection String را هر بار که مورد نیاز است باز نویسی نکنیم آن را یک بار در کلاس Bastni معرفی می کنیم ، مقدار آن را هم در سابروتین ِ سازنده Shared از فرم کلاس بستنی تعیین می کنیم

```
    Public Shared ConnectionString As String
```

و مقدار اولیه آن

```
Shared Sub New()
ConnectionString = "provider=microsoft.jet.oledb.4.0;data source=" &
Application.StartupPath & "\dbBastani.mdb;"
End Sub
```

تغییرات دیگری که مورد نیاز است در کلاس بستنی است. احتیاج به یک خاصیت دیگر به نام ID برای ذخیره شماره رکورد در دیتابیس داریم. همچنین یک تابع خصوصی به نام GetData و یک زیربرنامه خصوصی به نام SetData . از GetData برای زمانی که در خاصیت ها می خواهیم اطلاعاتی مثل مقدار بستنی یا تعداد لیوان را بدست آوریم استفاده می کنیم ، SetData را نیز در مواقعی که می خواهیم یکی از موارد را بفروشیم صدا می زنیم.
کد GetData را در زیر می بینید :

```
Private Function GetData(ByVal FieldName As String) As Integer
      Try
            Dim strSQL As String = "SELECT " & FieldName & " From Datas
      Where StoreID = " & mID.ToString
            Dim Con As New OleDb.OleDbConnection(ConnectionString)
            Con.Open()
            Dim CMD As New OleDb.OleDbCommand(strSQL, Con)
            Dim reader As OleDb.OleDbDataReader = CMD.ExecuteReader
```

```
            Dim result As Integer
            Do Until Not reader.Read
                result = reader(FieldName)
            Loop
            CMD.Dispose()
            Con.Close()
            Con.Dispose()
            Return result
      Catch e As InvalidCastException
            'hich kari nakon , chon 0 boode meghdar
      Catch ex As Exception
            MessageBox.Show(ex.Message)
      End Try
End Function
```

در قسمت catch از یک catch اختصاصی استفاده کردیم. این catch زمانی اتفاق می افتد که مقداری که reader(FieldName) بر می گرداند DBNull است و می خواهد آن را به result که از نوع صحیح است تبدیل کند. در این حالت یک ایراد رخ می دهد.
روش استفاده از این تابع را در خاصیت لیوان می بینید :

```
Public Property TedadLivan() As Integer
      Get
            mTedadLivan = GetData("livan")
            Return mTedadLivan
      End Get
      Set(ByVal Value As Integer)
            If Value >= 0 Then
                mTedadLivan = Value
                SetData("livan", Value)
            End If
            If mTedadLivan = 0 Then RaiseEvent LivanTamamShod(mObjName)
      End Set
End Property
```

کدِ سابروتینِ SetData را در ادامه می بینید :

```
Private Sub SetData(ByVal FieldName As String, ByVal Value As Integer)
      Try
            Dim Adapter As New OleDb.OleDbDataAdapter("SELECT * FROM
Datas WHERE storeID = " & mID.ToString, ConnectionString)
            Dim DS As New DataSet
            Adapter.Fill(DS, "Datas")
            If DS.Tables("Datas").Rows.Count <> 0 Then
                DS.Tables("Datas").Rows(0)(FieldName) = Value
            Else
                Dim dr As DataRow = DS.Tables("Datas").NewRow
                dr("StoreID") = ID
                dr(FieldName) = Value
                DS.Tables("Datas").Rows.Add(dr)
            End If
            Dim CMD As New OleDb.OleDbCommandBuilder(Adapter)
            Adapter.Update(DS, "Datas")
            DS.Dispose()
            CMD.Dispose()
            Adapter.Dispose()
      Catch ex As Exception
```

```
            MessageBox.Show(ex.Message)
        End Try
End Sub
```

وقتی می خواهیم اطلاعاتی مانند مقدار بستنی را در رکورد یک فروشگاه در Datas ذخیره کنیم اول باید بدانیم که آیا آن فروشگاه در Datas قرار دارد یا خیر. اگر قرار داشت صرفاً مقدار بستنی را تغییر می دهیم ولی اگر قرار نداشت باید آن فروشگاه را در Datas اضافه کرده و سپس مقدار بستنی را برای آن فروشگاه تنظیم کنیم.
روش دیگری که می توانست در اینجا مورد استفاده قرار گیرد این بود که هنگامی که یک فروشگاه را در Stores اضافه می کردیم یک رکورد هم در Datas برای آن فروشگاه اضافه می کردیم.

قسمت بعدی که باید تغییر دهیم در فرم اصلی است. زیربرنامه FillCombo باید اطلاعات از بانک اطاعاتی بخواند ، همچنین FillCombo باید در آخرین خط از سابروتین Load صدا زده شود.

```
Private Sub FillCombo()
        cmbStores.Items.Clear()
        Dim Con As New OleDb.OleDbConnection(Bastani.ConnectionString)
        Con.Open()
        Dim CMD As New OleDb.OleDbCommand("select * from Stores", Con)
        Dim reader As OleDb.OleDbDataReader = CMD.ExecuteReader
        Do Until Not reader.Read
            Dim bastan As New bastani(reader("StoreName"))
            bastan.ID = reader("ID")
            Forooshgah.Add(bastan)
            cmbStores.Items.Add(reader("StoreName"))
        Loop
        reader = Nothing
        CMD.Dispose()
        Con.Close()
        Con.Dispose()
End Sub
```

تغییر بعدی در زیربرنامه ای است که مسئول اضافه کردن اطلاعات به دیتابیس است ، یعنی زیربرنامه ای که قبلاً به Forooshgah اطلاعات را اضافه می کرد .

```
        Dim str As String = InputBox("نام فروشگاه جدید؟؟")
        If str.Trim <> "" Then
            Dim Adapter As New OleDb.OleDbDataAdapter("select * from
Stores where id = -1", Bastani.ConnectionString)
            Dim DS As New DataSet
            Adapter.Fill(DS, "Stores")
            Dim dr As DataRow = DS.Tables("Stores").NewRow
            dr("StoreName") = str
            DS.Tables("Stores").Rows.Add(dr)
            Dim CMD As New OleDb.OleDbCommandBuilder(Adapter)
            Adapter.Update(DS, "Stores")
            DS.Dispose()
            Adapter.Dispose()
            FillCombo()
        End If
```

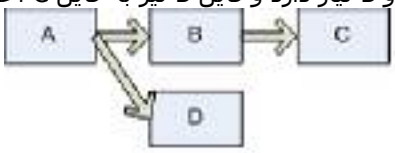
به علت اینکه تمامی زیربرنامه های خواندن را در کلاسِ بستنی نوشتیم دیگر تغییر مهمی در برنامه نیاز ندارد. سورس کامل این برنامه را در سی دی ضمیمه ببینید.

Deployment

بعد از اینکه یک نرم افزار کامپیوتری را طراحی و اجرا کردیم باید بتوانیم آن را به کامپیوتر کاربرانِ آن سیستم انتقال دهیم . به مراحلی که محصولی را از مرحله و محیط ساخت به محیط استفاده کنندگان می برد Deployment می گویند. دات نت در بعضی از جنبه های مراحل Deployment کارها را ساده تر از روش های برنامه نویسی قدیمی کرده است .

Deployment چیست ؟ Deployment در یک نگاه ساده به انتقال نرم افزار ما به کامپیوتر کسانی که از آن استفاده خواهند کرد گفته می شود. Deployment با توجه به نوع برنامه ای که باید انتقال داده شود می تواند تغییرات اساسی بکند ، وقتی برنامه مورد نظر ما یک وب سایت باشد باید فایلهای آن را به وب سرور منتقل کنید و وقتی برنامه ما یک برنامه ویندوزی است باید یک فایل exe و بعضی فایلهای مرتبط با آن انتقال یابد.

معمولاً در deployment با تعداد بسیاری از فایلها سر و کار داریم که همه آنها باید به دقت و با توجه به پیشنیازهایشان نصب شوند. هرگاه فایلی برای اجرای صحیح و کامل به فایل دیگری نیاز داشته باشد. وقتی یک فایل exe به فایلهای DLL مختلف نیاز دارد، می گویند فایل exe دارای تعدادی وابستگی(dependency) است . این ارتباطات می تواند مانند یک زنجیر ادامه دار باشد ، مثلاً فایل A به D و B نیاز دارد و فایل B نیز به فایل C احتیاج دارد.



بنابراین هنگام deployment باید به این ارتباطات نیز توجه کرد تا نرم افزار ما بعد از انتقال به کامپیوتر کاربر نیز درست عمل کند.

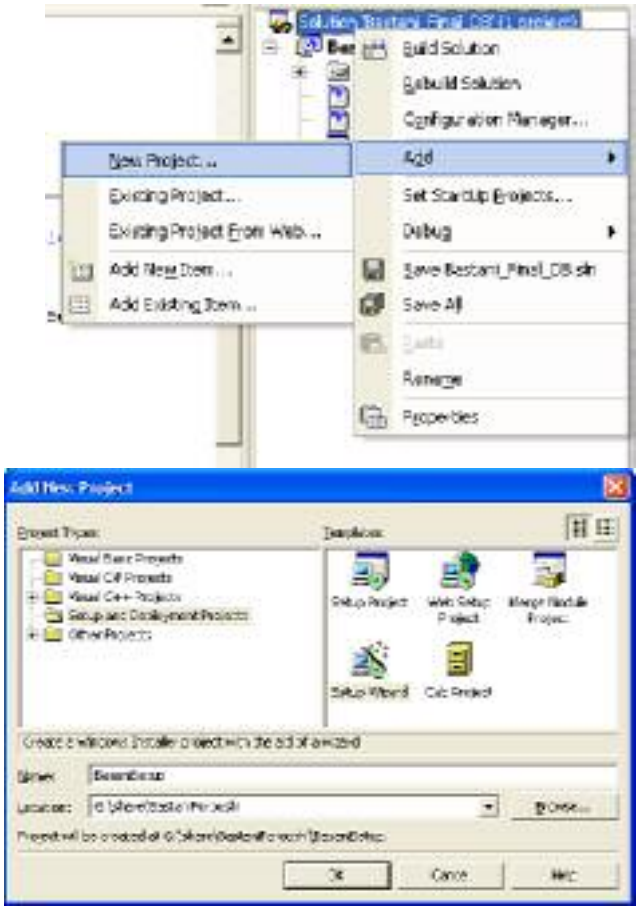
برنامه هایی که در دات نت نوشته می شوند به شکل پیش فرض دارای پیشنیازهای مختلفی هستند که اکثرِ آنها در .NET Framework قرار دارند. هنگامی که شما VS.NET را نصب کردیم این فایلها نیز به شکل خودکار نصب شده اند اما وقتی می خواهیم برنامه خود را به دیگر سیستم ها منتقل کنیم دیگر نمی توانیم VS.NET را همراه آن در هر کامپیوتر نصب کنیم !! ما صرفاً باید .NET Framework که در یک فایل فشرده 23 مگابایتی است؛ به سیستم کاربران خود منتقل کرده و نصب کنیم.

قبل از .NET. هنگام نصب برنامه ها اطلاعاتی در رجیستری ویندوز نیز ذخیره می شد ، هرگاه ما می خواستیم نسخه جدیدی را روی کامپیوتر کاربر نصب کنیم ابتدا می بایست نسخه قدیمی را حذف و سپس نسخه جدید را نصب می کردیم. انجام ندادن این کار، باعث بروز مشکلاتی در اجرای نسخه قبلی و جدید می شد . اما در دات نت احتیاجی به چنین کاری نیست . در حقیقت برای ثبت کامپوننت های (components) موجود در برنامه احتیاجی به رجیستری نداریم. همه آن اطلاعات در فایلی در خود برنامه ذخیره می شود . در نتیجه در دات نت تنها با کپی فایل می توانید برنامه خود را به کامپیوتر دیگری منتقل کنیم. معمولاً در محیط دات نت به این روش انتقال برنامه که احتمالاً مشکل خاصی هم در اجرایش وجود ندارد روش XCopy می گویند. یعنی مانند دستور xcopy در DOS می توان برنامه را انتقال داد.

اما گاهی فقط کپی کردن برنامه کارساز نیست و ما می خواهیم بر روی Desktop ، منوی Programsِ سیستمِ کاربر shortcut هایی به برنامه خود ایجاد کنیم . یا برنامه ما احتیاج به فونت خاصی دارد که باید در شاخه Windows/Fonts قرار گیرد. یا حتی می خواهیم کاربرمان بتواند با یک دکمه تمامی اینها را حذف کند، به عبارت دیگر امکان Uninstall داشته باشد. در چنین حالتی است که xcopy کار خاصی برای ما نمی تواند انجام دهد و حتماً می بایست از installer ها استفاده کنیم. در VS.NET این کار بوسیله ساختن یک فایل از نوع MSI یا Windows Installer انجام می شود. Windows Installer آخرین فناوری مایکروسافت برای نصب برنامه هاست که حتی VS.NET یا Office و اکثر برنامه های دیگر از این تکنولوژی برای نصب استفاده می کنند.

حال برای ادامه کار برای برنامه بستنی فروش می خواهیم یک برنامه Setup بسازیم . البته در این جا، هدف برنامه بستنی فروش نیست و به هیچ وجه محتوای آن اهمیتی ندارد زیرا می خواهیم با روش ساختن برنامه Setup آشنا شویم.

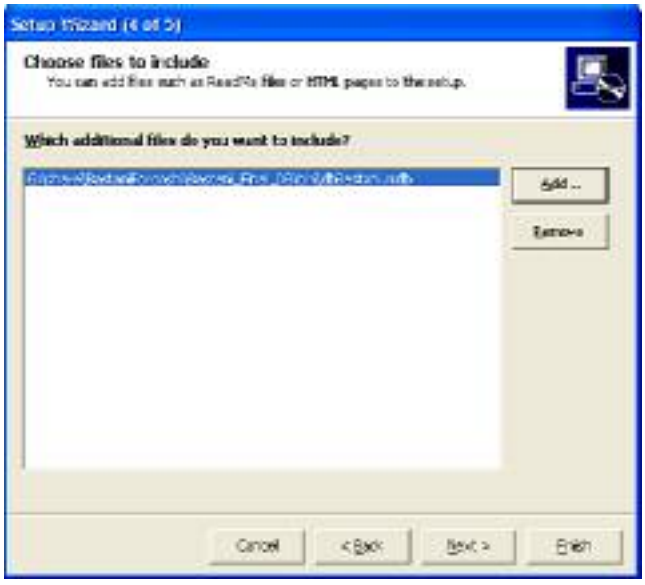
پروژه بستنی فروش را باز کنید. پروژه در قالب کلی تری به نام Solution قرار دارد. Setup ها نیز مانند خود بستنی فروش یک پروژه هستند و باید به solution اضافه شوند. بر روی نام Solution کلیک سمت راست موس را زده و از Add Projects یک Setup wizard را انتخاب کنید .



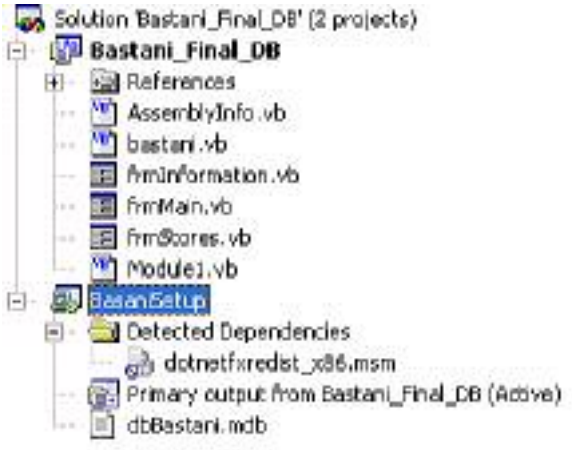
وقتی این پروژه را انتخاب کردید به صورت خودکار Wizard مربوطه نمایش داده می شود.

همانطور که در شکل مشخص است امکانات مختلفی در این Wizard قرار دارد. ما می توانیم با توجه به نوع پروژه خود یکی از آنها را انتخاب کنیم. در اینجا به خاطر اینکه برنامه ما تحت ویندوز است و می خواهیم یک MSI از آن ایجاد کنیم اولین گزینه، Create a setup for a windows application را انتخاب می کنیم. در مرحله بعد Wizard از ما می خواهد فایلهایی را که در خروجی MSI لازم داریم انتخاب کنیم.

در لیست گزینه ها حتی سورس کدهای برنامه نیز وجود دارد ، اما گزینه مورد نظر ما Primary output from Bastani_Final_db است که همان فایل exe مربوط به پروژه بستنی فروشی است. در مرحله بعدی می توانیم فایلهای دیگری به MSI اضافه کنیم ، در اینجا به فایل دیتابیسی که برای برنامه ساختیم نیاز داریم و باید آن را به لیست اضافه کنیم .

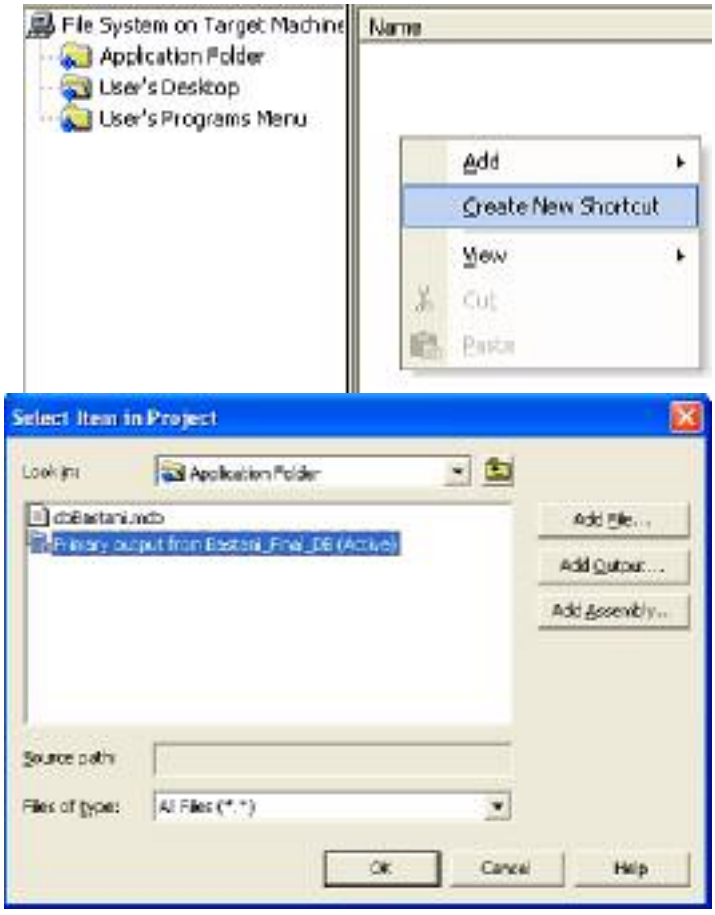


بعد از انتخاب فایلهای اضافه ، Wizard خلاصه ای از کارهایی را که انجام داده است را نشان می دهد و ما برای خاتمه باید بر روی دکمه Finish کلیک کنیم تا Wizard تمامی کارهای باقی مانده را انجام دهد.
بعد از کلیک بر روی Finish کار تمام شده است و پروژه ای با نامی که انتخاب کرده بودیم در Solution بستنی فروشی اضافه می شود .

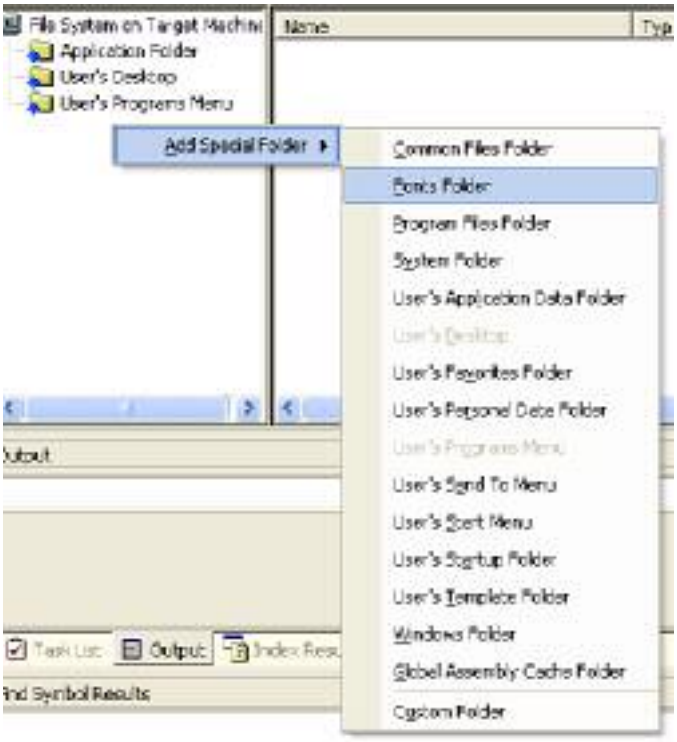


همان طور که در شکل بالا مشخص است فایل دیتابیسی که انتخاب کرده بودیم نیز در لیست فایلهای پروژه Setup قرار گرفته است.
VS.NET مشخصات دیگری از پروژه Setup را نیز نمایش می دهد. در سمت چپ نام شاخه هایی که در کامپیوتر مقصد ساخته خواهد شد و در سمت راست محتوای هر کدام از آن شاخه ها دیده می شود.

شاخه های User's Desktop ، Application Folder و User's Programs Menu سه شاخه ای هستند که همیشه وجود دارند. برای اینکه یک shortcut به برنامه اصلی در Desktop کامپیوتر مقصد ایجاد شود روی شاخه User's desktop کلیک کرده و سپس در سمت راست کلیک سمت راست موس را می زنیم و در Context Menu ای که باز می شود Create new shortcut را انتخاب کنید . سپس خروجی برنامه اصلی را از application folder انتخاب می کنیم تا shortcut مورد نظر ساخته شود ، نام این shortcut را نیز می توانید به سلیقه خود تغییر دهید.

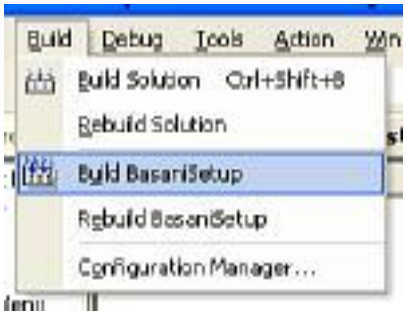


فرض کنید برنامه ما احتیاج به نصب فونت خاصی داشته باشد در آن صورت باید آن فونت را نیز در این فایل MSI قرار دهیم . برای این کار روی قسمت چپ کلیک سمت راست موس را بزنید تا از context menu ی که باز می شود Font Folder را انتخاب کنید .

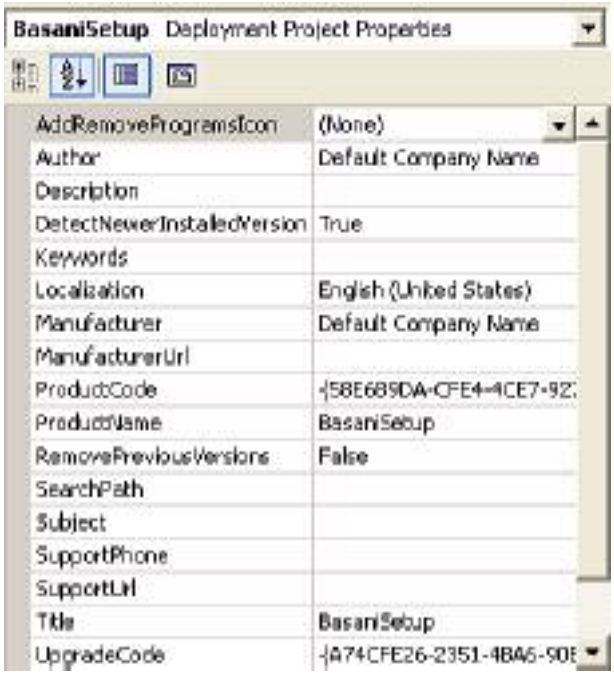


حال هر فونتی را که احتیاج است را با استفاده از این folder به پروژه setup اضافه کنید .

در حال حاضر صرفاً پروژه ای در solution داریم بدون آنکه فایل MSI یی از آن ساخته شده باشد . برای ساخته شدن باید یک بار این پروژه را Build کنیم. بعد از build در مسیر پروژه فایل setup و MSI مورد نظر قرار گرفته است.



خود پروژه Setup دارای خواص مختلفی مانند نام محصول ، نام شرکت ، زبان ، سازنده و .... است که آنها را می توانیم در پنجره خواص تغییر دهیم.



همان گونه که ذکر شد برنامه های دات نت احتیاج به .NET Framework دارند. وقتی یک MSI می سازیم خودِ .NET Framework در آن قرار نمی گیرد بلکه آنرا به شکل جداگانه باید به همراه فایل MSI به کاربر بدهید تا آن را نصب کند.
البته روش دیگر و بهتر این است که کاربر ، خود آخرین نسخه دات نت و دیگر اجزای کلی را از سایت Microsoft.com دریافت کند ولی با توجه به حجم فایلها و نبود بستر مناسب مخابراتی در ایران در زمان نوشتن این متن عملاً امکان اجرای این روش وجود ندارد.
نکته دیگر فایلهای MSI هستند. در نسخه های قدیمی ویندوز ممکن است اصلاً فایلهای MSI معتبر نباشند . در چنین شرایطی فایل Setup ی که در کنار برنامه MSI قرار دارد به کار می آید و به کاربر می گوید که باید آخرین نسخه از برنامه Windows Installer را از سایت Microsoft.com دریافت کند. البته این فایلها برای نصب در شاخه vs.net/common7/tools/deployment قرار دارند و می توانیم آنها را همراه برنامه اصلی به کاربر بدهیم تا در صورت بوجود آمدن مشکل آنها را نصب کند.