مقدمه‌ای بر

ویژوال استودیو

و

C

تألیف: HANS-PETTER HALVORSEN

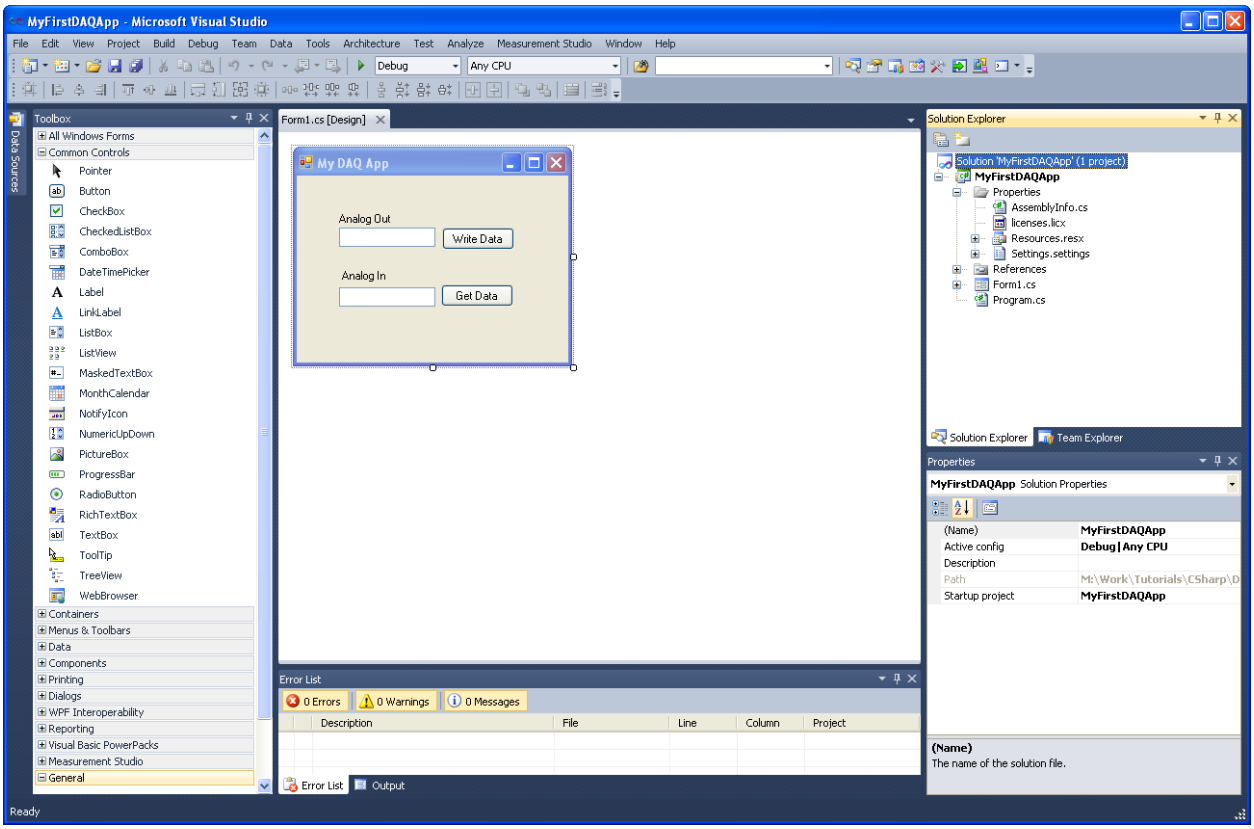
ترجمه: سید نورالدین رفیعی طباطبائی

کالج دانشگاهی جنوب شرق نروژ

مقدمه بر ویژوال استودیو و C#

هانس - پیتر هاورسن، ۲۶ سپتامبر ۲۰۱۶

(ترجمه و ویرایش: سید نورالدین رفیعی)



http://home.hit.no/~hansha

فهرست مندرجات

۱. مقدمه

در این خودآموز ما ویژوال استودیو و C# را می‌بینیم. C# زبان برنامه نویسی است، در حالی که ویژوال استودیو محیط توسعه می‌باشد.

پیوندهای مفید:

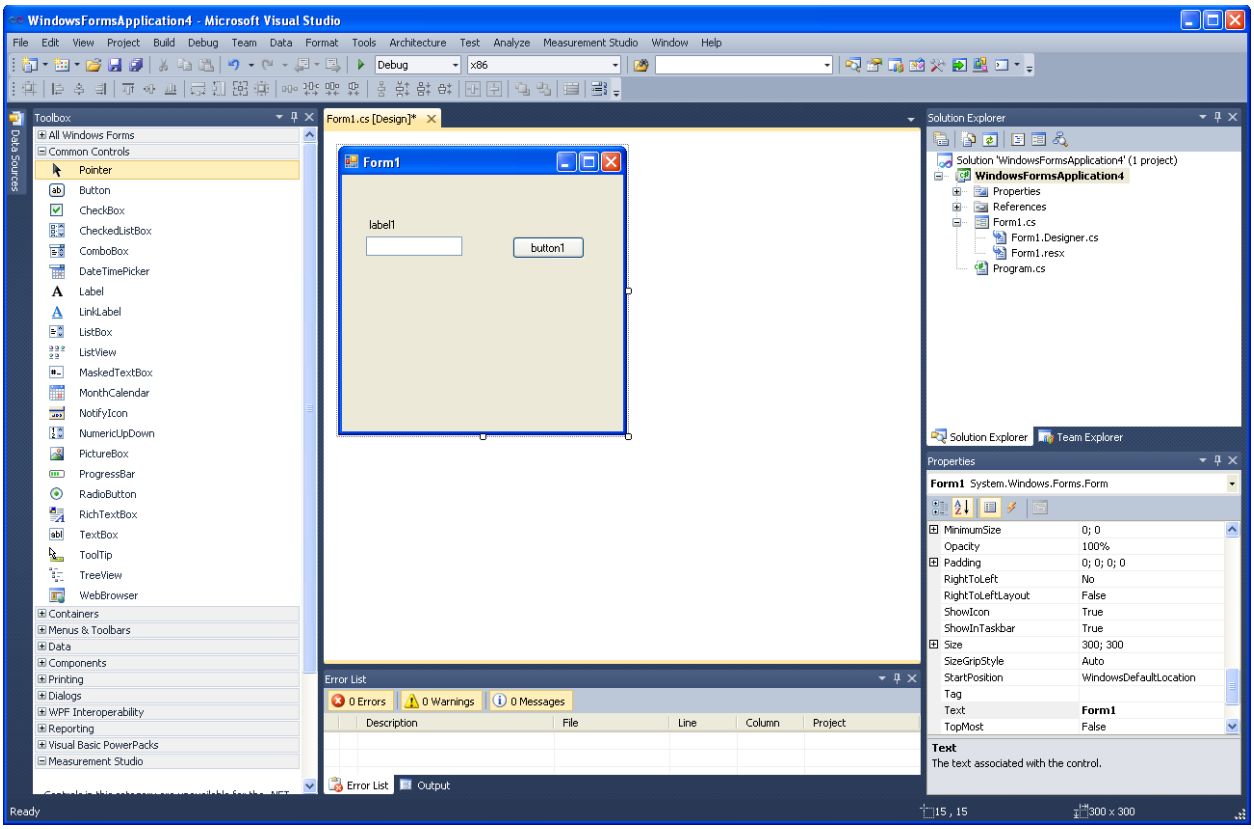
ویژوال استودیو: http://msdn.microsoft.com/en-us/library/dd831853.aspx

C#: http://msdn.microsoft.com/en-us/library/kx37x362.aspx

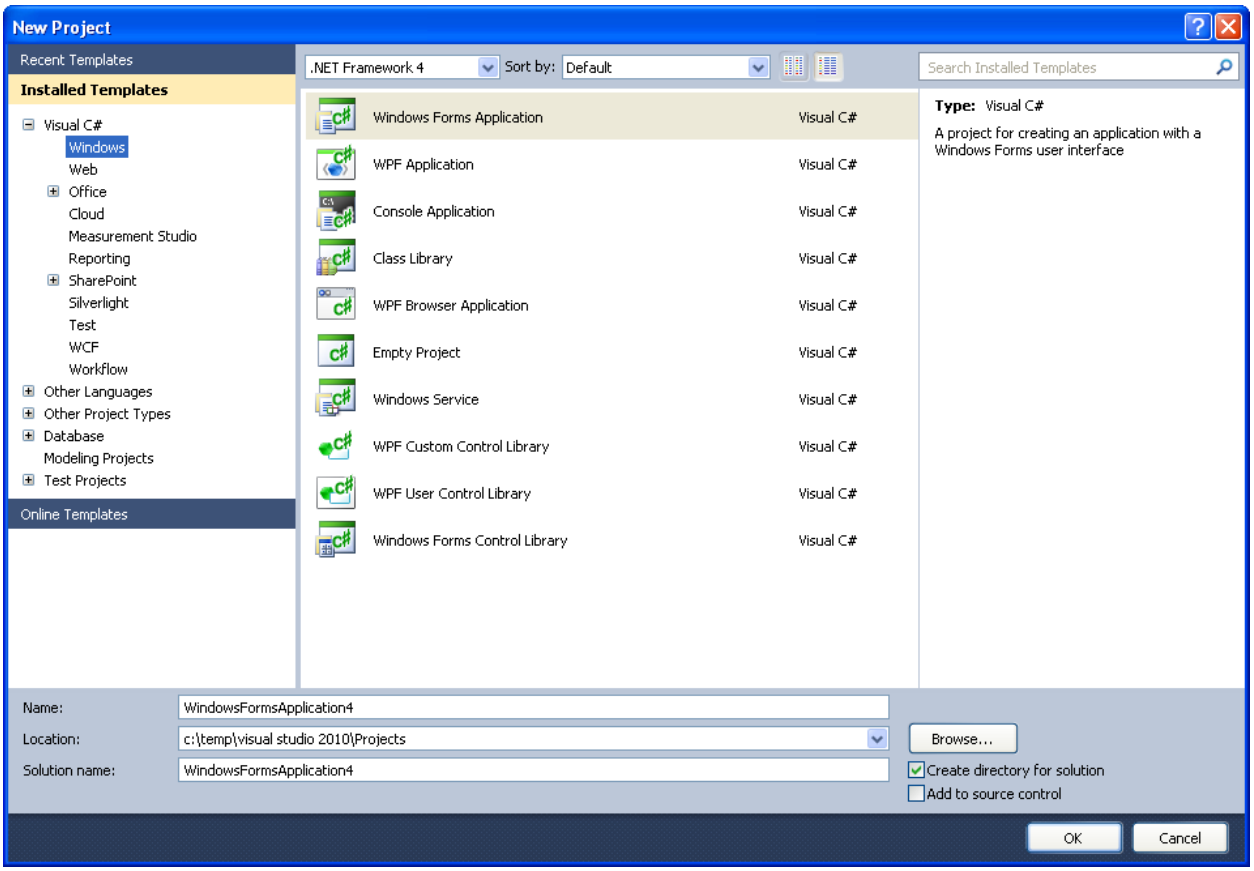
۱.۱ ویژوال استودیو

میکروسافت ویژوال استودیو، یک محیط توسعهٔ یکپارچه (IDE) از میکروسافت است. می‌تواند برای توسعهٔ برنامه‌های واسط گرافیکی و کنسول کاربر، همراه با برنامه‌های فرم ویندوز، وب‌سایت‌ها، برنامه‌های تحت وب، و سرویس‌های وب، استفاده شود؛ هم در کد اصلی و هم با کد مدیریت شده برای تمام پلات‌فرم‌های {پایگاه‌های} پشتیبانی شده توسط میکروسافت ویندوز، ویندوز فون، ویندوز CE، چارچوب .NET و میکروسافت سیلورلایت.

در زیر محیط توسعهٔ یکپارچه (IDE) را در ویژوال استودیو می‌بینیم:



پروژه‌های جدید از پنجرهٔ New Project ساخته می‌شوند:



C# .۱.۲

C# تلفظ می‌شود «سی شارپ». یک زبان برنامه‌نویسی شیء گرا و بخشی از خانوادهٔ .NET. از میکروسافت است. بسیار شبیه به C++ و جاوا می‌باشد. توسط میکروسافت توسعه یافته و فقط بر پلات‌فرم ویندوز کار می‌کند.

.۱.۳ چارچوب .NET

چارچوب .NET. (که تلفظ می‌شود «دات نت») یک چارچوب نرم‌افزاری است که اصولاً بر میکروسافت ویندوز اجرا می‌گردد. حاوی یک کتابخانهٔ بزرگ است و چندین زبان برنامه‌نویسی را پشتیبانی می‌کند که به این زبان‌ها قابلیت همکاری با یکدیگر را می‌دهد (هر زبان می‌تواند کدی که در زبان‌های دیگر نوشته شده را استفاده کند). کتابخانهٔ .NET. برای تمام زبان‌های برنامه‌نویسی که .NET. را پشتیبانی می‌کنند قابل استفاده است. برنامه‌های نوشته شده برای چارچوب .NET. در یک محیط نرم‌افزاری که زبان مشترک زمان اجرا

(CLR) خوانده می‌شود اجرا می‌گردند {در بعضی از متون معروف است به «زمان اجرای زبان مشترک»}. یک ماشین مجازی برنامه که سرویس‌های مهم از قبیل امنیت، مدیریت حافظه، و استثناءگردانی را فراهم می‌کند. کتابخانهٔ کلاس و CLR با هم چارچوب .NET را تشکیل می‌دهند.

۱.۴. برنامه‌نویسی شیءگرا

برنامه‌نویسی شیءگرا (OOP) یک الگوی زبان برنامه‌نویسی است که پیرامون «اشیاء» بجای «اعمال» سازمان داده شده و داده‌ها بجای منطق. از لحاظ تاریخی، قبلاً یک برنامه بعنوان روالی منطقی دیده می‌شده است که داده‌های ورودی را می‌گیرد، آنها را پردازش می‌کند، و داده‌های خروجی را تولید می‌نماید.

نخستین گام در OOP شناسایی تمام اشیائی است که می‌خواهید اداره کنید و چگونگی ارتباط آنها با یکدیگر، و یک تمرین که اغلب به مدل‌سازی داده‌ها معروف است. یک بار که یک شیء را شناسایی کردید، آن را بعنوان یک کلاس از اشیاء تعمیم می‌دهید؛ و نوع داده‌هایی که در بر دارد و هر ترتیب منطقی که می‌تواند آن را اداره کند را تعیین می‌نمایید. هر ترتیب منطقی مشخص بعنوان یک متد {method} شناخته می‌شود. یک نمونهٔ حقیقی از یک کلاس بعنوان یک «شیء» یا یک «نمونهٔ کلاس» نامیده می‌شود. شیء یا نمونهٔ کلاس، آن چیزیست که در کامپیوتر اجرا می‌نمایید. متدهای آن دستورات کامپیوتر را تدارک می‌بینند و مشخصه‌های شیء کلاس داده‌های مربوطه را تهیه می‌کنند. شما با اشیاء ارتباط برقرار می‌کنید – و آنها با یکدیگر.

خصیصه‌های مهم OOP اینها هستند:

- کلاس‌ها و اشیاء
- توارث {وراثت = Inheritance}
- پولی‌مورفیسم {چند ریختی = Polymorphism}
- کپسوله‌سازی {Encapsulation}

سیمولا {Simula} نخستین زبان برنامه‌نویسی شیء گرا بود. در سالهای دههٔ ۱۶۶۰ توسط کیستیان نیگارد از نروژ ساخته شد.

جاوا، پیتون، C++، ویژوال بیسیک دات نت و C# امروزه زبانهای OOP رایج هستند.

از آنجا که اشیاء از نوع سیمولا در C++، جاوا و C# دوباره پیاده‌سازی می‌شوند تأثیر سیمولا اغلب درک می‌شود. سازندهٔ C++ (۱۹۷۹)، برایان استراستراپ (از دانمارک)، اعتراف دارد که سیمولا بیشترین تأثیر را بر وی در ساختن C++ داشته است.

۲. ویژوال استودیو

۲.۱. مقدمه

صفحهٔ خانهٔ ویژوال استودیو: http://www.microsoft.com/visualstudio

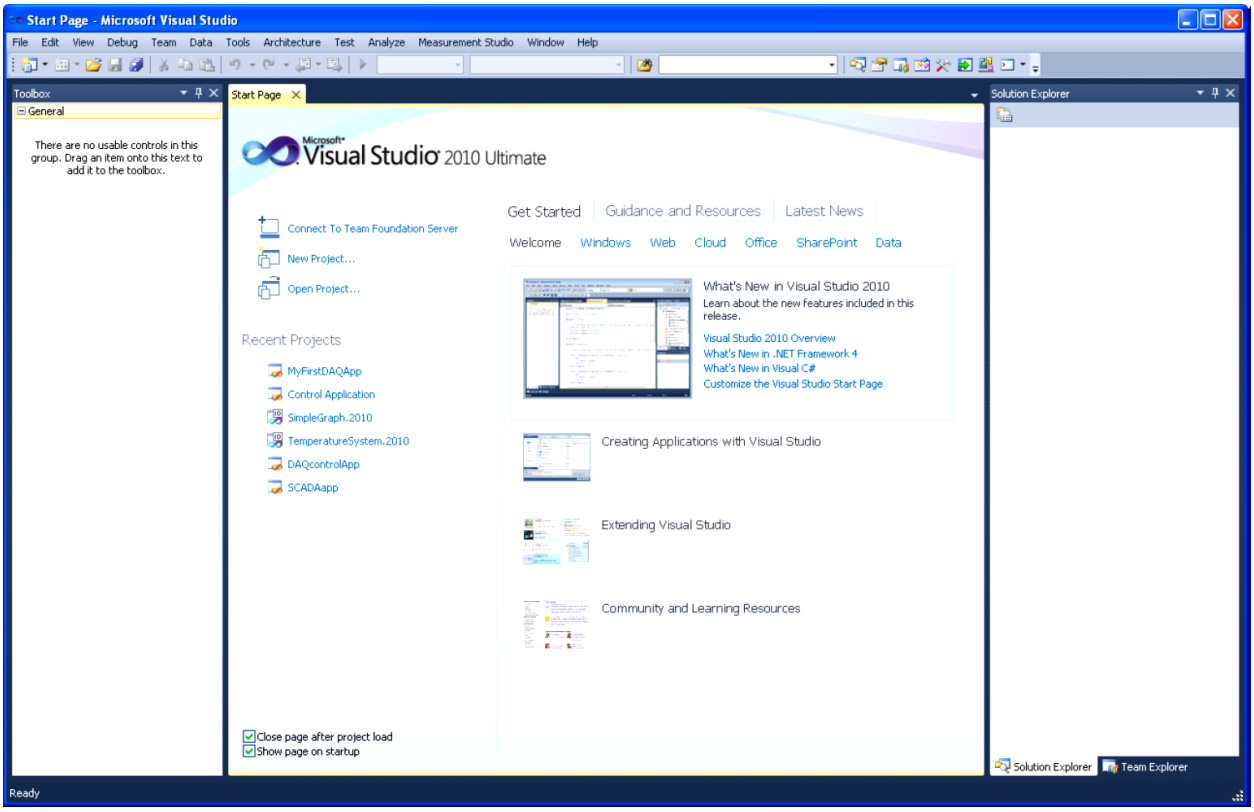
ویرایش‌های گوناگونی از ویژوال استودیو وجود دارد، از قبیل ویژوال استودیو اکسپرس (رایگان)، ویژوال استودیو حرفه‌ای، ویژوال استودیو پرمیوم و ویژوال استودیو اولتیمیت.

۲.۲. شروع

۲.۱.۱. محیط توسعهٔ یکپارچه (IDE)

خانوادهٔ محصول ویژوال استودیو، یک محیط توسعهٔ یکپارچه (IDE) به اشتراک می‌گذارد که از عناصر متعددی تشکیل یافته است: نوار منو، نوار ابزار استاندارد، پنجره‌های ابزاری مختلف که لنگر شده‌اند یا در سمت چپ، پایین، و راست مخفی شده‌اند، و همچنین فضای ویرایشی. پنجره‌های ابزار، منوها، و نوارهای ابزار بسته به نوع پروژه یا فایلی که بر روی آن کار می‌کنید موجود می‌باشند.

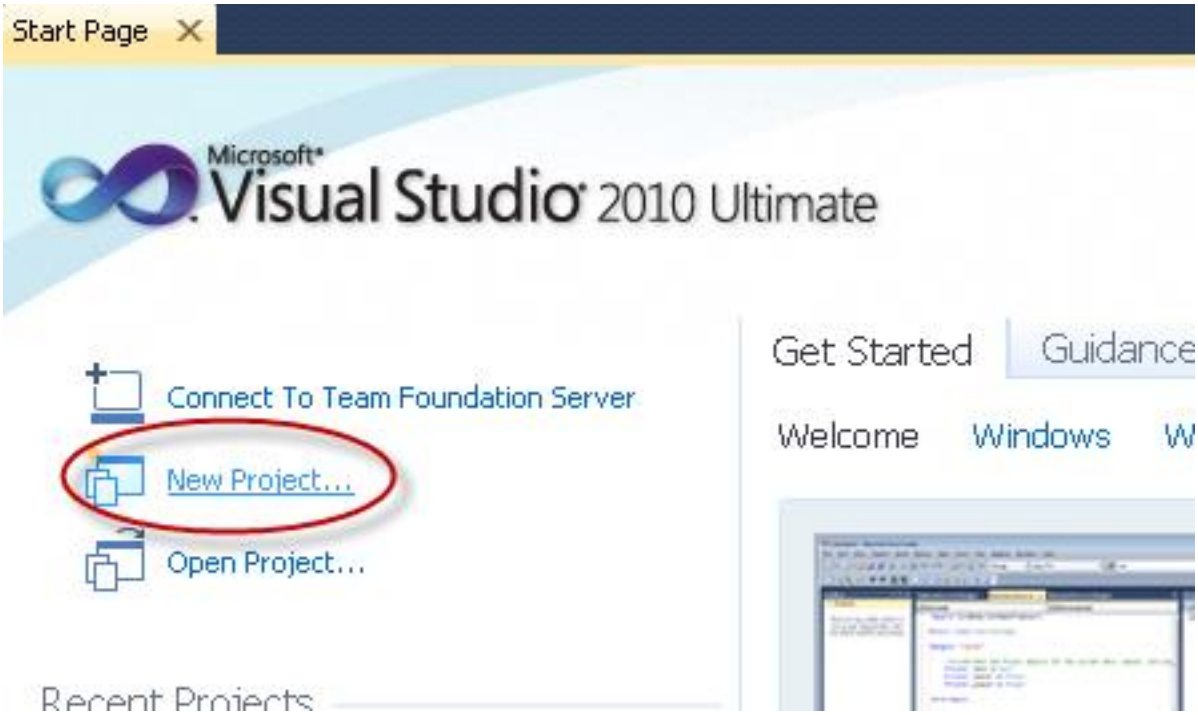
در زیر ما IDE ویژوال استودیو (محیط توسعهٔ یکپارچه) را می‌بینیم:



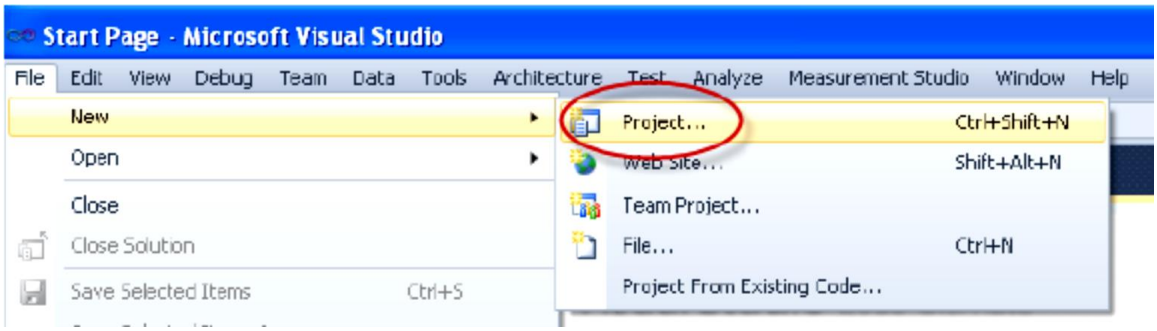
۲.۲.۲. پروژهٔ جدید

نخستین چیزی که هنگام ساختن یک برنامهٔ جدید باید انجام دهید، ایجاد یک پروژهٔ جدید است.

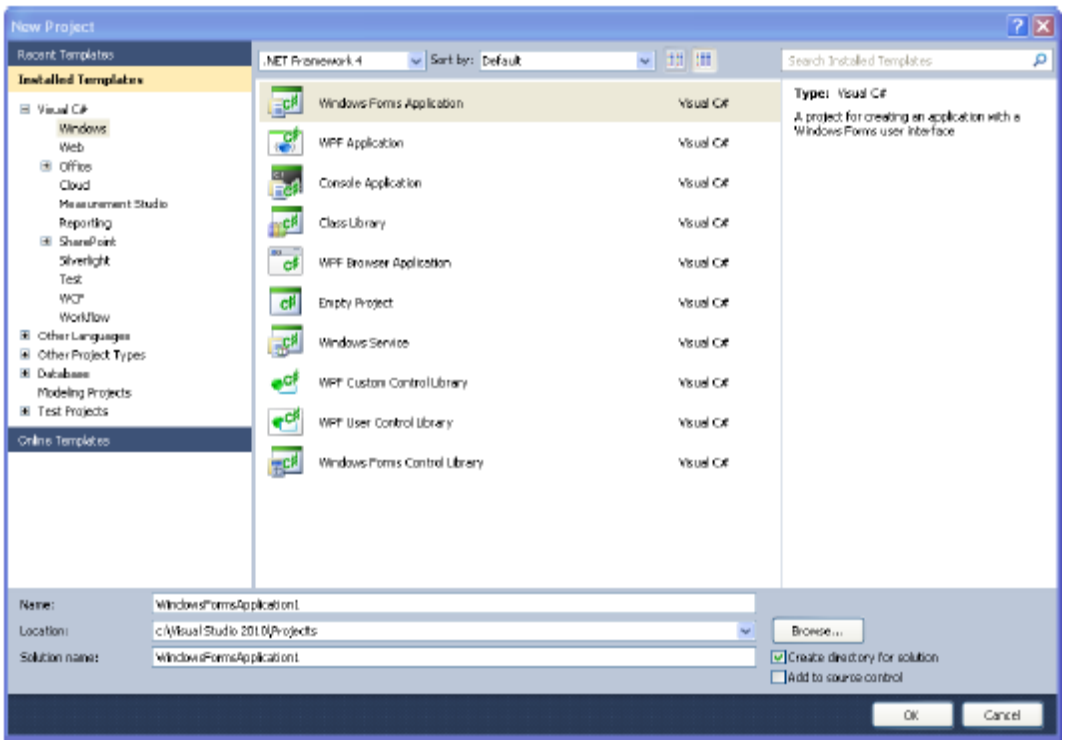
این کار را می‌توان از صفحهٔ آغاز انجام داد:



یا از منوی فایل:



سپس پنجرهٔ New Project ظاهر می‌گردد:



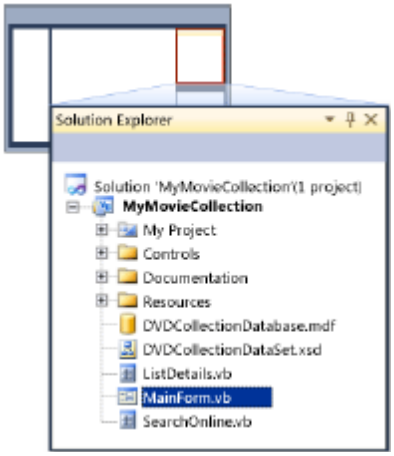
در این پنجره، الگوی مناسب را براساس آن نوع برنامه‌ای که می‌خواهید بسازیم انتخاب می‌نماییم، و یک نام و محلی برای پروژه و راه حل.

رایج‌ترین برنامه‌ها عبارتند از:

- برنامهٔ فرم ویندوز
- برنامهٔ کنسول
- برنامهٔ WPF
- برنامهٔ ASP .NET Web
- برنامهٔ Silverlight

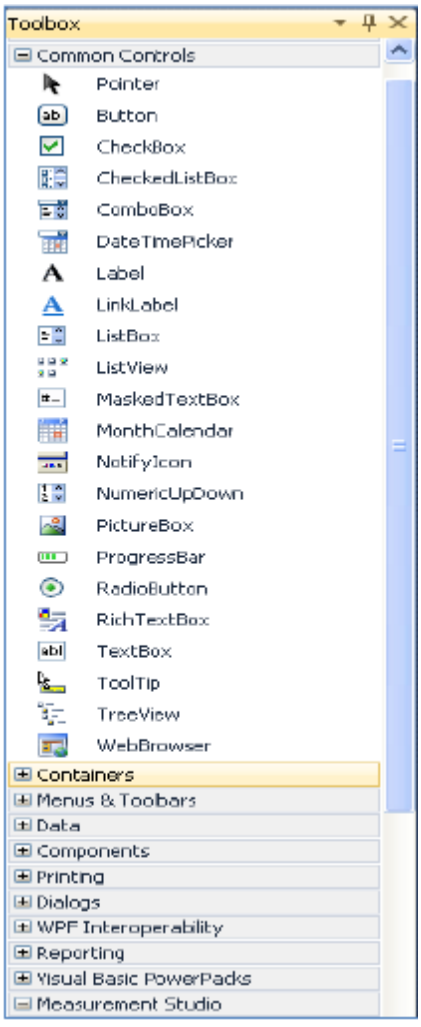
۲.۲.۳. کاوشگر راه حل {Solution Explorer}

راه حل‌ها و پروژه‌ها شامل مواردی هستند که مراجع، ارتباطات داده‌ها، پوشه‌ها، و فایل‌هایی که برای ایجاد برنامه نیاز داریم را می‌نمایانند. یک ظرف راه حل {solution container} می‌تواند شامل پروژه‌های متعدد باشد، و یک ظرف پروژه بطور نمونه حاوی چند مورد است.



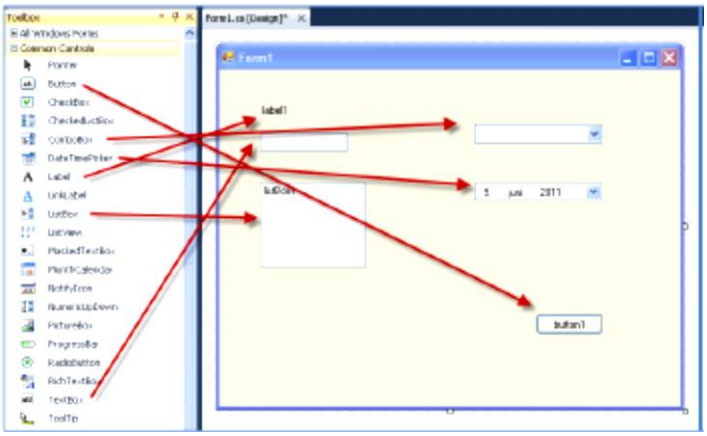
۲.۲.۴. جعبه ابزار {Toolbox}

جعبه ابزار شامل تمام کنترل‌های لازم و دیگر چیزهاست که ما برای ایجاد رابط کاربر نیاز داریم. شکل زیر را نگاه کنید.



جعبه ابزار حاوی تمام کنترل‌ها، و بقیهٔ چیزهاییست که ما می‌توانیم در رابط کاربری خود استفاده کنیم.

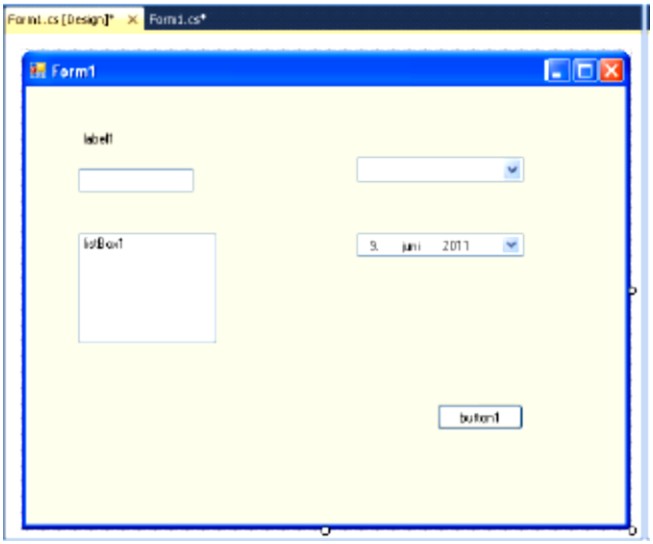
در جهت استفاده از آنها در رابط کاربری خود، فقط کافیست آنها را بکشیم و در Form رها کنیم، همانطور که در زیر نشان داده شده:



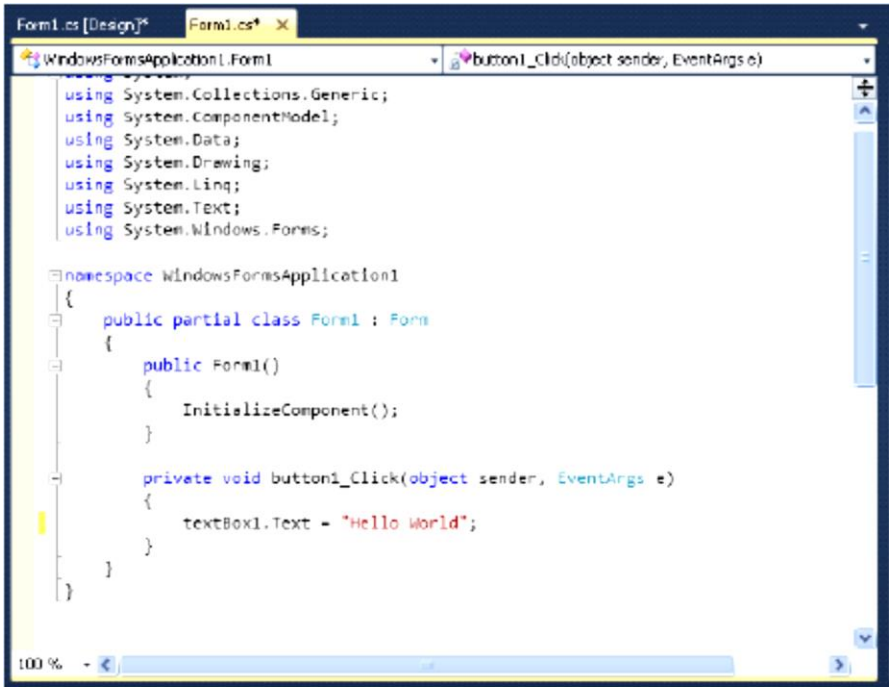
۲.۲.۵. ویرایشگرها و طراح‌ها

ویژوال استودیو ابزارهای طراحی و ویرایشگرهای متعددی دارد.

طراح رابط کاربری گرافیکی:



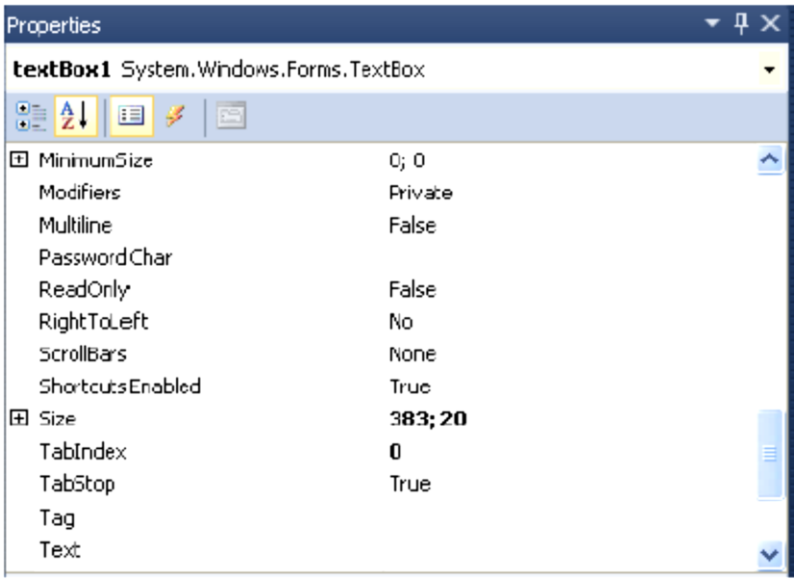
ویرایشگر کد:



۲.۲.۶. پنجرهٔ خصوصیات

هر کنترل که بر رابط کاربری خود داریم خصوصیت‌های زیادی دارد که می‌توانیم تنظیم کنیم.

این کار در پنجرهٔ خصوصیات انجام می‌پذیرد:

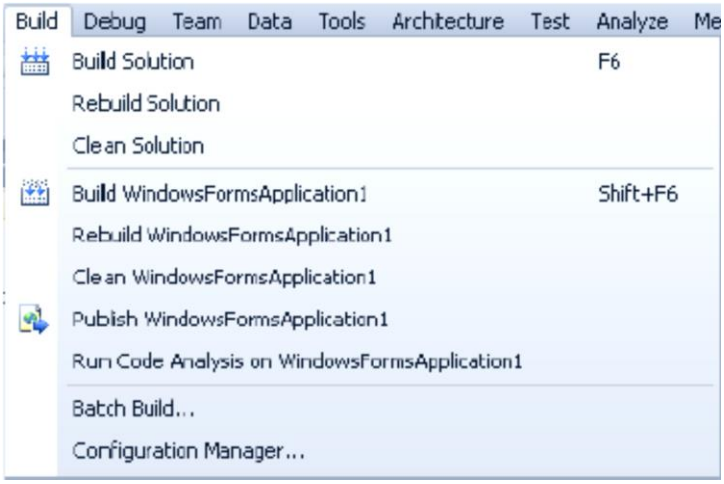


۲.۲.۷. ابزارهای ساخت و اشکال‌زدایی

در ویژوال استودیو ما ابزارهای اشکال‌زدایی و ساخت بسیاری داریم.

منوی ساخت

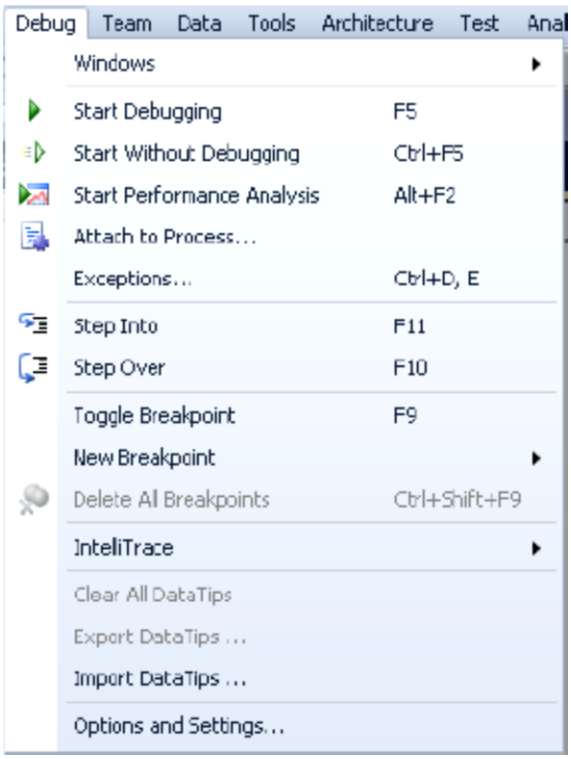
در زیر ما منوی **ساخت** را ملاحظه می‌کنیم:



ابزاری که بیشتر از همه استفاده شده Build Solution است (کلید میانبر: F6).

منوی اشکال زدایی

در زیر منوی اشکال‌زدا را می‌بینیم:



ابزاری که بیشتر از همه استفاده می‌شود Start Debugging است (کلید میانبر: F5).

۳. برنامه نویسی ویندوز

۳.۱ مقدمه

هنگام ایجاد برنامه‌های ویندوز مقدماتی، ما می‌توانیم بین دو مورد زیر انتخاب می‌کنیم:

- برنامهٔ فرم‌های ویندوز
- برنامه‌های WPF (اساس نمایش ویندوز)

فرم‌های ویندوز روش استاندارد ایجاد برنامه‌های ویندوز می‌باشد و خیلی وقت است که وجود دارد، حتی قبل از این که .NET 1.0 (2002) معرفی شود. WPF یک رویکرد جدید در ایجاد برنامه‌های ویندوز است و با .NET Framework 3.0 (2006) معرفی شد.

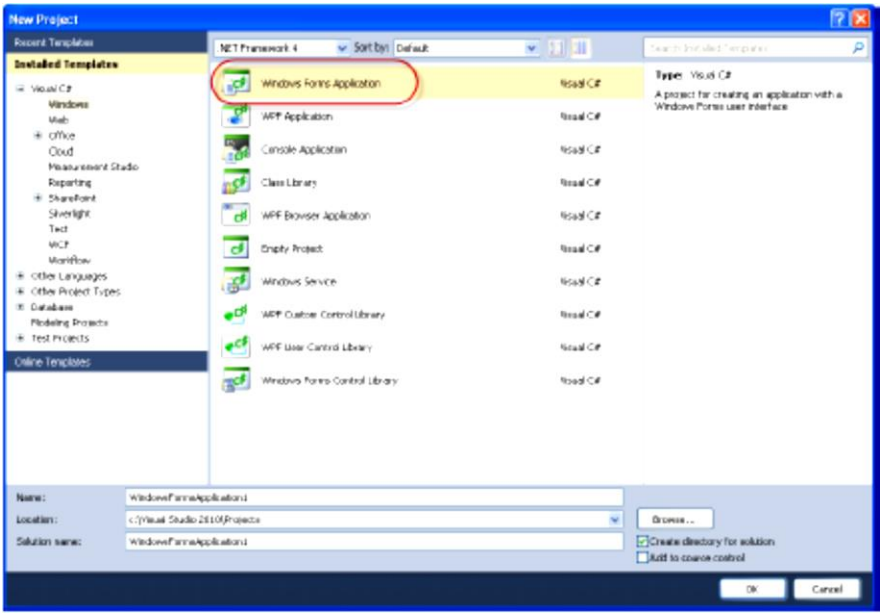
مثلاً محیط توسعهٔ یکپارچه ویژوال استودیو {Visual Studio IDE} بکمک WPF کلاً بازنویسی شده است.

برنامه‌های WPF و برنامهٔ فرم‌های ویندوز در زیر با جزئیات بیشتر شرح داده خواهند شد.

۳.۱.۱ فرم‌های ویندوز

فرم‌های ویندوز روش استاندارد ایجاد برنامه‌های ویندوز است.

Windows Forms Application را در پنجرهٔ New Project انتخاب کنید:



WPF .۳.۱.۲

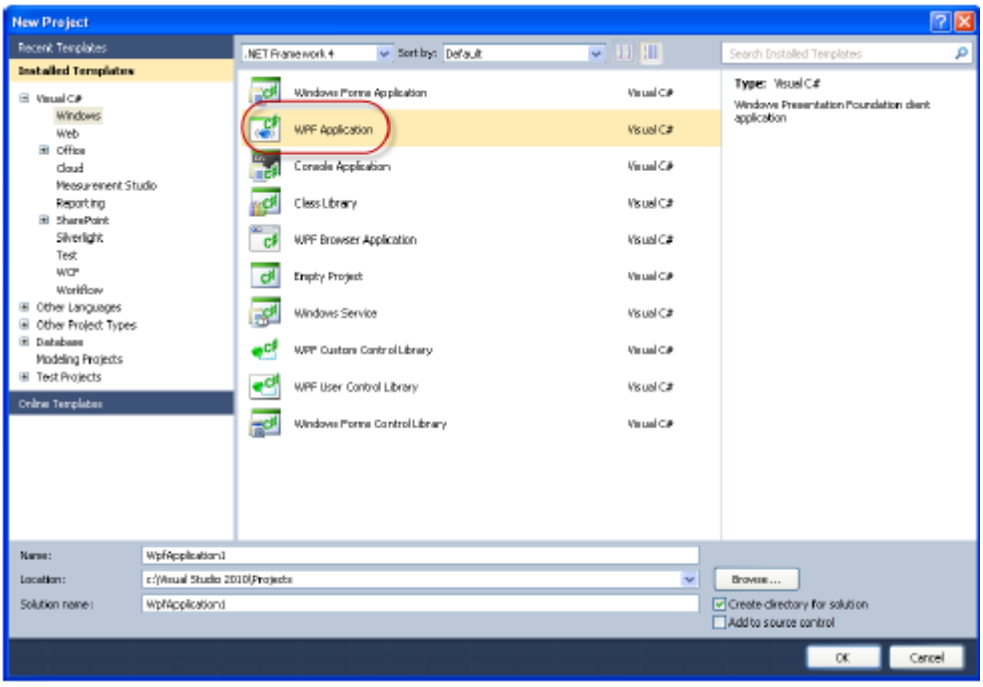
اساس نمایش ویندوز {Windows Presentation Foundation} (یا WPF) توسعه یافته توسط میکروسافت، یک زیرسیستم گرافیکی نرم‌افزار کامپیوتری برای ترجمهٔ رابطهای کاربر در برنامه‌های برمبنای ویندوز است. WPF در نظر گرفته شده تا اجرای فرم‌های مرسوم ویندوز را بعهده گیرد.

رابط کاربری گرافیکی در WPF با استفاده از **XAML** (زبان نشانه‌گذاری کاربردی توسعه‌پذیر) طراحی شده است.

:XAML

در پی موفقیت زبان‌های نشانه‌گذاری برای گسترش وب، WPF یک زبان موسوم به زبان نشانه‌گذاری کاربردی توسعه‌پذیر (XAML) معرفی می‌کند که بر مبنای XML است. XAML بعنوان یک روش کارآمدتر از ایجاد رابطهای کاربری کاربردی طراحی شده است.

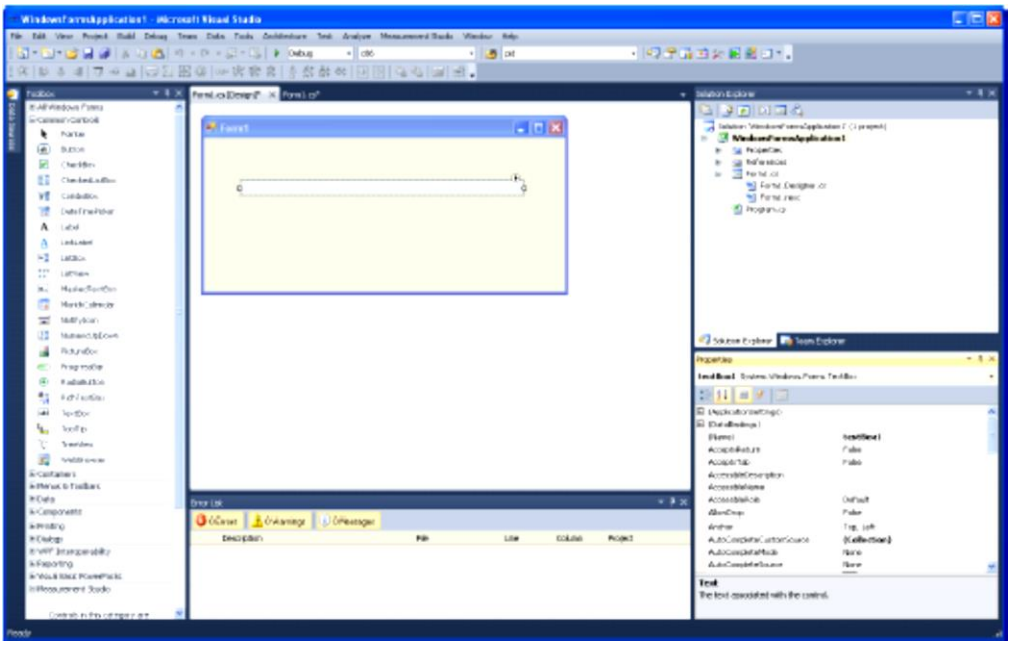
در پنجرهٔ New Project گزینهٔ WPF Application را انتخاب کنید:



Hello World .۳.۲

ما با ایجاد برنامهٔ مرسوم Hello world بکمک فرم‌های ویندوز شروع می‌کنیم. برنامه در زیر آمده است:

محیط ویژوال استودیو به شکل زیر دیده می‌شود:



در این پروژه ما از یک TextBox ساده استفاده می‌کنیم (textBox1) و وقتی برنامه را شروع می‌نماییم متن Hello World در TextBox نوشته می‌شود.

کد بصورت زیر است:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
      public partial class Form1 : Form
      {
            public Form1()
            {
```

```
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            textBox1.Text = "Hello World";
        }
    }
}
```

۴. شروع با C#

C# یک زبان برنامه‌نویسی شیء-گرا است.

برنامه نویسی شیء-گرا (OOP) یک الگوی برنامه‌نویسی با استفاده از «اشیاء» است - ساختارهای داده از فیلدهای داده و متدها تشکیل می‌شوند همراه با برهم‌کنش آنها با یکدیگر {تعامل بین آنها} - تا برنامه‌های کامپیوتری را طراحی کنند. روش‌های برنامه‌نویسی ممکن است ویژگی‌هایی چون خلاصه‌سازی داده‌ها، کپسوله‌سازی، پیام‌رسانی، پولی‌مورفیسم {چند ریختی}، و توارث را شامل باشند.

۴.۱. مقدمه

در این فصل ما با موارد اصلی آغاز می‌کنیم که تمام زبان‌های برنامه‌نویسی دارند:

- انواع داده‌ای و متغیرها
- جریان کنترل: if-else و غیره
- حلقه‌ها: حلقه‌های While، حلقه‌های For، و غیره.

بعد موارد زیر را معرفی می‌نماییم:

- فضاهای نام
- کلاس‌ها
- حوزه‌های داده
- متدها
- خصوصیات

در فصل بعد بیشتر به عمق آنچه که برنامه‌نویسی شیء-گرا هست می‌رویم و مباحث مهم OOP زیر را معرفی می‌کنیم:

- توارث
- پولی‌مورفیسم {چندریختی}
- کپسوله‌سازی

تذکر! C# نسبت به کوچکی و بزرگی حروف حساس است.

۴.۲. انواع داده‌ای و متغیرها

متغیرها در واقع مکان‌های ذخیره برای داده‌ها هستند. می‌توانیم داده‌ها را در آنها قرار دهیم و محتوی آنها را بصورت بخشی از یک عبارت C# بازیابی کنیم. تفسیر داده‌ها در یک متغیر از طریق Types کنترل می‌شود.

انواع سادهٔ داده‌ای C# عبارتند است از:

- نوع بولی
- انواع عددی: اعداد صحیح، اعداد با ممیز شناور، اعشاری
- انواع رشته‌های حرفی

۴.۲.۱. انواع بولی

انواع بولی با استفاده از کلیدواژهٔ bool تعریف شده‌اند. دو مقدار دارند: true یا false. در سایر زبان‌ها، مثل C یا C++، وقتی که 0 یعنی false و هرچیز دیگر یعنی true شرطهای بولی می‌توانند ادا شوند. اما، در C# تنها مقادیری که یک شرط بولی را ادا می‌کنند true و false می‌باشند، که کلیدواژه‌های رسمی هستند.

مثال:

```
bool content = true;
bool noContent = false;
```

۴.۲.۲ انواع عددی: اعداد صحیح، با ممیز شناور، اعشاری

مثال:

```
int i=35;
long y=654654;
```

```
float x;
double y;
decimal z;
```

۴.۲.۳ انواع رشته‌ای

مثال:

```
string myString=”Hei på deg”;
```

کاراکترهای خاص که ممکن است در رشته‌ها استفاده شوند:

Escape Sequence	Meaning
\'	Single Quote
\"	Double Quote
\\	Backslash
\0	Null, not the same as the C# *null* value
\a	Bell
\b	Backspace
\f	form Feed
\n	Newline
\r	Carriage Return
\t	Horizontal Tab
\v	Vertical Tab

۴.۲.۴. آرایه‌ها

مثال:

```
int[] myInts = { 5, 10, 15 };
```

۴.۳. روند کنترل

توانایی کنترل جریان داده‌ها در برنامه در هر زبان برنامه‌نویسی اهمیت دارد.

مهمترین شیوه‌ها این دو هستند:

- دستور if
- دستور switch

۴.۳.۱ دستور if

دستور if احتمالاً پرکاربردترین مکانیزم بکار رفته برای کنترل جریان داده در برنامه است.

یک دستور if به شما اجازه می‌دهد مسیرهای منطقی متفاوتی، بسته به شرایط داده شده، اتخاذ نمایید. وقتی شرط از لحاظ منطق بولی درست {true} ارزیابی شود، یک بلوک کد برای آن شرط درست اجرا خواهد شد. شما اختیار یک دستور if واحد، چند دستور else if متعدد، و یک دستور else دلخواه را دارید.

مثال:

```
bool myTest;
myTest=false;
if (myTest==false)
     MessageBox.Show("Hello");
```

اگر ما بیش از یک خط کد داشته باشیم که باید اجرا شود، ناچاریم که از آکولاد استفاده کنیم، مثلاً:

```
bool myTest;
myTest=false;
if (myTest == false)
{
      MessageBox.Show("Hello1");
      MessageBox.Show("Hello2");
}
```

برای منطق پیچیده‌تر از if ... else استفاده می‌کنیم.

مثال:

```
bool myTest;
myTest=true;
if (myTest == false)
{
      MessageBox.Show("Hello1");
}
else
{
      MessageBox.Show("Hello2");
}
```

یا شما می‌توانید از جملات **if ... else** تو در تو استفاده کنید.

مثال:

```
int myTest;
myTest=2;
if (myTest == 1)
{
      MessageBox.Show("Hello1");
}
else if (myTest == 2)
{
      MessageBox.Show("Hello2");
}
else
{
      MessageBox.Show("Hello3");
}
```

۴.۳.۲. دستور switch

یک شکل دیگر از دستور انتخابی، دستور switch است، که مجموعه‌ای از دستورات منطقی را بسته به مقدار یک پارامتر داده شده اجرا می‌نماید. انواع مقادیری که یک دستور switch بر آن عمل می‌کند می‌توانند بولی، شمارشی، انواع صحیح، و رشته‌ها باشند.

مثال:

```
switch (myTest)
{
	case 1:
	MessageBox.Show("Hello1");
	break;
	case 2:
	MessageBox.Show("Hello2");
	break;
	default:
	MessageBox.Show("Hello3");
	break;
}
```

۴.۴. حلقه‌ها

در C# ما انواع مختلف حلقه را داریم:

- حلقهٔ While
- حلقهٔ do
- حلقهٔ for
- حلقهٔ foreach

۴.۴.۱. حلقهٔ While

یک حلقهٔ While شرط را بررسی می‌کند و سپس تا وقتی که شرط سنجیده شده با یک مقدار بولی، صحیح است، به اجرای یک قطعه از کد ادامه می‌دهد.

مثال:

```
int myInt = 0;
while (myInt < 10)
{
	MessageBox.Show("Inside Loop: " + myInt.ToString());
	myInt++;
}
MessageBox.Show("Outside Loop: " + myInt.ToString());
```

۴.۴.۲. حلقهٔ do

یک حلقهٔ do شبیه به حلقهٔ while است، با این تفاوت که شرط را در پایان حلقه بررسی می‌کند. یعنی اجرای حداقل یک بار حلقهٔ do حتمی است. از دیگر سو، حلقهٔ while عبارت بولی خود را در آغاز می‌سنجد

و معمولاً هیچ تضمینی نیست که دستورات درون حلقه اجرا شوند، مگر شما کد را برنامه‌ریزی کنید که صریحاً چنین کند.

مثال:

```
int myInt = 0;
do
{
      MessageBox.Show("Inside Loop: " + myInt.ToString());
      myInt++;
} while (myInt < 10);
MessageBox.Show("Outside Loop: " + myInt.ToString());
```

۴.۴.۳. حلقهٔ for

یک حلقهٔ for مثل حلقهٔ while کار می‌کند، بجز این که حلقهٔ for شامل ارزش‌دهی آغازی و تغییر شرط است. حلقه‌های for وقتی مناسبند که شما می‌دانید دقیقاً چند بار می‌خواهید دستورات داخل حلقه اجرا شوند.

مثال:

```
for (int i = 0; i < 10; i++)
{
      MessageBox.Show("Inside Loop: " + myInt.ToString());
      myInt++;
}
MessageBox.Show("Outside Loop: " + myInt.ToString());
```

۴.۴.۴. حلقهٔ foreach

یک حلقهٔ foreach برای تکرار مواردی از یک فهرست بکار می‌رود. بر آرایه‌ها یا مجموعه‌ها عمل می‌کند.

مثال:

```
string[] names = { "Elvis", "Beatles", "Eagles", "Rolling Stones" };
foreach (string person in names)
{
      MessageBox.Show(person);
}
```

۴.۵. متدها

متدها بسیار مفیدند، زیرا به شما اجازه می‌دهند منطق خود را درون واحدهای مختلف مجزا سازید. می‌توانید اطلاعات را به متد صادر کنید، آن را وادار سازید یک یا چند دستور را اجرا کند، و یک مقدار بازگشتی را حاصل نماید. توانایی صدور عملگرها و بازگشت مقادیر، اختیاری است و بستگی دارد به آنچه که می‌خواهید متد انجام دهد.

متدها مثل توابع، روال‌ها {پروسیجرها procedure} یا زیرروال‌ها {سابروتین subroutine} هستند که در زبان‌های برنامه‌نویسی دیگر استفاده می‌شوند. تفاوت اینست که یک متد همیشه بخشی از یک کلاس است.

مثال:

```
public void ShowCarColor(string color)
{
      MessageBox.Show("My Car is: " + color);
}
```

ما دربارهٔ متدها در بخش کلاس‌ها که در پی می‌آید بیشتر خواهیم آموخت.

دو نوع متد داریم:

- متدهای استاتیک
- متدهای غیر استاتیک (متد نمونه)

متدهای استاتیک به همهٔ کلاس تعلق دارند، ولی متدهای غیراستاتیک به هرنمونهٔ ایجاد شده از کلاس تعلق دارند.

۴.۵.۱ متدهای غیراستاتیک

مثال:

کلاس را تعریف می‌کنیم:

```
class Car
{
      //Nonstatic/Instance Method
      public void SetColor(string color)
      {
            MessageBox.Show("My Car is: " + color);
      }
}
```

سپس آن را استفاده می‌نماییم:

```
Car myCar = new Car(); //We create an Instance of the Class
myCar.SetColor("blue"); //We call the Method
```

۴.۵.۲. متدهای استاتیک

کلاس را تعریف می‌کنیم:

```
class Boat
{
      //Static Method
      public static void SetColor(string color)
      {
            MessageBox.Show("My Boat is: " + color);
```

```
        }
}
```

سپس آن را استفاده می‌نماییم:

```
Boat.SetColor("green");
```

بنابراین، نیاز به ایجاد یک شیء/معرفی کلاس پیش از استفاده از متد استاتیک نداریم.

۴.۶. فضاهای نام

فضاهای نام اجزای برنامهٔ C# هستند که طراحی شده‌اند تا به ما در سازمان‌دهی برنامه‌ها کمک کنند. در اجتناب از تداخل اسامی بین دو مجموعه از کد نیز مساعدت می‌نمایند. پیاده‌سازی فضاهای نام در کد عادت خوبیست؛ زیرا وقتی می‌خواهیم قسمتی از کد خود را دوباره استفاده کنیم ما را از مشکلات بعدی می‌رهاند.

فضای نامی که می‌خواهیم استفاده کنیم را در بالای کد مشخص می‌نماییم.

مثال:

وقتی یک برنامهٔ فرم‌های ویندوز می‌سازیم، فضاهای نام زیر بطور پیش‌فرض لحاظ خواهند شد:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
…
```

فقط کافیست وقتی نیاز داریم موارد بیشتر را اضافه کنیم.

مطالب بیشتر دربارهٔ فضاهای نام بعداً می‌آیند.

۴.۷. کلاس‌ها

نخستین گام در OOP شناسایی تمام اشیائی است که می‌خواهیم اداره کنیم و این که آنها چگونه با یکدیگر ارتباط دارند. تمرینی که اغلب به مدل‌سازی داده‌ها معروف است. یکبار که یک شیء را شناسایی کردیم، آن را بعنوان کلاسی از اشیاء تعمیم می‌دهیم و نوع داده‌هایی را که در بر دارد و هر ترتیب منطقی که می‌تواند

آن را اداده کند تعیین می‌نماییم. هر ترتیب منطقی مجزا بعنوان یک متد شناخته می‌شود. یک نمونهٔ حقیقی از یک کلاس، یک «شیء» یا «یک نمونه از کلاس» خوانده می‌شود. نمونهٔ کلاس یا شیء آن چیزیست که در رایانه اجرا می‌کنیم. متدهای آن دستورات رایانه را فراهم می‌کنند و مشخصه‌های شیء کلاس، داده‌های مرتبط را تهیه می‌کنند. ما با اشیاء رابطه برقرار می‌کنیم و آنها با یکدیگر ارتباط برقرار می‌نمایند.

همه چیز در C# بر مبنای کلاس است. کلاس‌ها با استفاده از کلیدواژهٔ class شناسانده می‌شوند و در پی آن نام کلاس می‌آید و یک مجموعه از اعضای کلاس که در آکولاد محصور شده‌اند.

یک کلاس بطور عادی حاوی متدها، فیلدها {Fields = حوزه‌ها} و خواص است.

هر کلاس یک سازنده دارد، که هرگاه یک نمونه از یک کلاس ایجاد می‌گردد فراخوانده می‌شود. مقصود سازنده‌ها اینست که وقتی یک نمونهٔ کلاس ایجاد می‌شود به اعضای آن مقدار اولیه بدهند. سازنده‌ها مقادیر بازگشتی ندارند و همیشه همان نام کلاس را دارند.

مثال:

ما کلاس زیر را تعریف می‌کنیم:

```
class Car
{
	public string color; //Field
	//Method
	public void ShowCarColor()
	{
		MessageBox.Show("My Car is: " + color);
	}
}
```

سپس می‌توانیم آن را استفاده کنیم:

```
Car myCar = new Car(); //We create an Instance of the Class
myCar.color = "blue"; //We set a value for the color Field
myCar.ShowCarColor(); //We call the Method
```

۴.۷.۱. سازنده

مقصود سازنده، دادن مقدار اولیه به اعضای کلاس است وقتی یک نمونه از کلاس ایجاد شد.

مثال:

ما می‌توانیم یک سازنده را برای ایجاد یک رنگ «پیش‌فرض» برای خودروی خودمان استفاده کنیم.

کلاس را تعیین می‌کنیم:

```
class Car
{
	public string color; //Field
	//Constructor - Used to initialize the Class
	public Car()
	{
		color="green";
	}
	//Method
	public void ShowCarColor()
	{
		MessageBox.Show("My Car is: " + color);
	}
}
```

سپس می‌توانیم آن را استفاده کنیم

```
Car myCar = new Car(); //We create an Instance of the Class
myCar.ShowCarColor(); //We call the Method
myCar.color = "blue"; //We set a value for the color Field
myCar.ShowCarColor(); //We call the Method
```

نتیجه به این صورت است:

مثال:

همچنین می‌توانیم به این صورت عمل کنیم:

```
class Car
{
	public string color; //Field
	//Constructor - Used to initialize the Class
	public Car(string initColor)
	{
		color = initColor;
	}
	//Method
	public void ShowCarColor()
	{
		MessageBox.Show("My Car is: " + color);
	}
}
```

سپس از آن استفاده می‌کنیم:

```
Car myCar = new Car("green"); //We create an Instance of the Class
myCar.ShowCarColor(); //We call the Method
```

ویژگی جدید: مقدار دهی اولیه بدون یک سازنده:

در C# 4.0 می‌توانیم بشکل زیر عمل نماییم:

کلاس را تعریف می‌کنیم (بدون هیچ سازنده‌ای):

```
class Car
{
	public string color; //Field
	public string model; //Field
	//Method
	public void ShowCarColor()
	{
		MessageBox.Show("My Car Color is: " + color);
		MessageBox.Show("My Car Model is: " + model);
	}
}
```

سپس به این صورت عمل می‌کنیم:

```
Car myCar = new Car {color="white", model="2004"};
myCar.ShowCarColor(); //We call the Method
```

۴.۸. ویژگی‌ها

ویژگی‌ها، فرصت حفاظت از یک فیلد در یک کلاس را با خواندن و نوشتن بر آن از طریق ویژگی می‌دهند. در دیگر زبان‌ها این کار اغلب با برنامه‌هایی انجام می‌شود که گیرنده و گذارندهٔ اختصاصی را پیاده می‌کنند. ویژگی‌های C# این نوع از حفاظت را مقدور می‌سازد در حالی که به شمار اجازهٔ دستیابی به ویژگی را نیز می‌دهد؛ درست مثل این که آن، یک فیلد بوده است.

مثال:

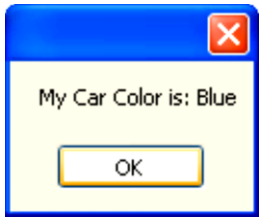
ما یک کلاس با خواص و یک متد تعریف می‌کنیم:

```
class Car
{
      public string Name {get; set;}
      public string Color {get; set;}
      public void ShowCarProperties()
      {
            MessageBox.Show("My Car Name is: " + Name);
            MessageBox.Show("My Car Color is: " + Color);
      }
}
```

سپس می‌توانیم آن را استفاده می‌کنیم:

```
Car myCar = new Car();
myCar.Name="Volvo";
myCar.Color="Blue";
myCar.ShowCarProperties();
```

نتایج چنین خواهد بود:



تنها اعضای کلاس که تا اینجا دیده‌ایم فیلدها، ویژگی‌ها، متدها، و سازنده‌ها بوده‌اند. اینجا یک فهرست کامل از انواع عضو است که می‌توانیم در کلاس خود داشته باشیم:

- **سازنده‌ها**
- تخریب‌کننده‌ها (عکس سازنده‌ها)
- **فیلدها**
- **متدها**
- **خواص**
- شاخص سازها {ایندکسرها = Indexers}
- نماینده‌ها
- **رویدادها**
- کلاس‌های تو در تو

۴.۹. قراردادهای نام‌گذاری

قراردادهای نام‌گذاری گوناگونی برای تصریح کردن متغیرها، کلاس‌ها و متدها و غیره وجود دارد.

نمادسازی Camel: {Camel = کلک، سار، شتر}

ما برای **متغیرها** و **پارامترها/آرگومان‌ها** معمولاً از «نمادسازی Camel» استفاده می‌کنیم.

مثال:

```
string myCar;
int number;
string backColor;
```

← در حالت Camel، حرف نخست یک شناسه در حالت حروف کوچک است و نخستین حرف هر مورد الحاق شدهٔ بعدی در حالت حرف بزرگ است.

نشان‌گذاری پاسکال:

برای **کلاس‌ها**، **متدها** و **خواص**، ما بطور عادی از «نشان‌گذاری پاسکال» استفاده می‌کنیم.

مثال:

```
class Car
```

```
        {
        void ShowCarColor()
        {
                …
        }
}
```

← در حالت پاسکال نخستین حرف در یک شناسه و نخستین حرف هر مورد الحاق شدهٔ بعدی به حات حروف بزرگ می‌باشند.

برای فضاهای نام ما از حالت پاسکال و یک نقطهٔ جداکننده استفاده می‌کنیم.

مثال:

```
System.Drawing
System.Collections.Generics
```

کنترل‌ها:

ما برای کنترل‌ها بر رابط کاربری خود یا از «نشان‌گذاری پاسکال» استفاده می‌کنیم یا از «نشان‌گذاری مجارستانی»، ولی فقط یکی از آنها را رعایت می‌کنیم!

مثال:

«نشان‌گذاری پاسکال»:

```
LoginName
LoginPassword
```

«نشان‌گذاری مجارستانی»:

```
txtName
txtPassword
lblName
btnCancel
```

که «txt» یعنی آن یک کنترل متن است، «lbl» کنترل برچسب است، «btn» یک کنترل دکمه است، وغیره.

سرنام‌ها: {acronyms = حروف مخفف}

وضعیت سرنام بستگی دارد به طول سرنام. تمام سرنام‌ها حداقل دو حرف طول دارند. اگر یک سرنام دقیقاً دو حرف باشد، بعنوان یک سرنام کوتاه طرح می‌شود. یک سرنام با سه حرف یا بیشتر یک سرنام بلند است.

در کل، نباید از اختصارات یا سرنام‌ها استفاده کنیم. این کار خوانایی نام‌های ما را کم می‌کند. بهمین نحو، دشوار است بدانیم چه زمانی درست است که فرض کنیم یک سرنام زیاد برسمیت شناخته شده است.

ولی وقتی مجبوریم، قواعد به این صورت می‌باشند:

مثال‌های سرنام کوتاه (دو کاراکتر):

`DBRate`

یک ویژگی بنام DBRate مثالی از سرنام کوتاه (DB) است که بعنوان نخستین واژهٔ یک شناسهٔ حالت پاسکال استفاده شده است.

`ioChannel`

یک پارامتر بنام ioChannel مثالی از سرنام کوتاه (IO) استفاده شده بعنوان نخستین واژهٔ یک شناسهٔ حالت Camel است.

مثال‌های سرنام بلند (سه کاراکتر یا بیشتر)

`XmlWriter`

یک کلاس بنام XmlWriter مثالی از یک سرنام بلند استفاده شده بعنوان نخستین واژهٔ یک شناسهٔ حالت پاسکال است.

`htmlReader`

یک پارامتر بنام htmlReader مثالی از یک سرنام بلند استفاده شده بعنوان نخستین واژهٔ یک شناسهٔ حالت Camel است.

۵- بیشتر در مورد روش‌های شیء-گرا

در این فصل روش‌های شیءگرای زیر را معرفی می‌کنیم:

- توارث {وراثت = Inheritance}
- پولی‌مورفیسم {چند ریختی = Polymorphism}
- کپسوله سازی {Encapsulation}

۵.۱. توارث

توارث یکی از مفاهیم اولیهٔ برنامه‌نویسی شیء-گرا می‌باشد. اجازه می‌دهد از کد موجود دوباره استفاده کنیم.

با بکارگیری مؤثر از تکرار، می‌توانیم در برنامه‌نویسی در وقت صرفه جویی کنیم.

مثال:

ما کلاس پایه را تعریف می‌کنیم:

```
class Car
{
	public void SetColor(string color)
	{
		MessageBox.Show("My Car is: " + color);
	}
}
```

سپس یک کلاس جدید که از کلاس پایه ارث می‌برد را تعریف می‌کنیم:

```
class Volvo : Car
{
	//In this simple Example this class does nothing!
}
```

بعد، شروع به استفاده از کلاس می‌کنیم:

```
Car myCar = new Car();
myCar.SetColor("blue");

Volvo myVolvo = new Volvo();
myVolvo.SetColor("green");
```

چنانکه می‌بینیم می‌توانیم متد SetColor() را استفاده کنیم که در کلاس پایه تعریف شده است.

۵.۲. پولی‌مورفیسم

یکی دیگر از مفاهیم اولیهٔ برنامه‌نویسی شیء-گرا پولی‌مورفیسم است. اجازه می‌دهد متدهای کلاس مشتق شده را طی زمان اجرا از طریق یک مرجع کلاس پایه احضار کنیم.

مثال:

ما با یک کلاس پایه شروع می‌کنیم:

```
class Car
{
	public virtual void CarType()
	{
		MessageBox.Show("I am a Car");
	}
}
```

تعدیل‌کنندهٔ مجازی به کلاس‌های مشتق شده اشاره دارد که آنها می‌توانند این متد را باطل کنند.

سپس ۳ کلاس جدید ایجاد می‌کنیم که از کلاس پایه مشتق شده‌اند:

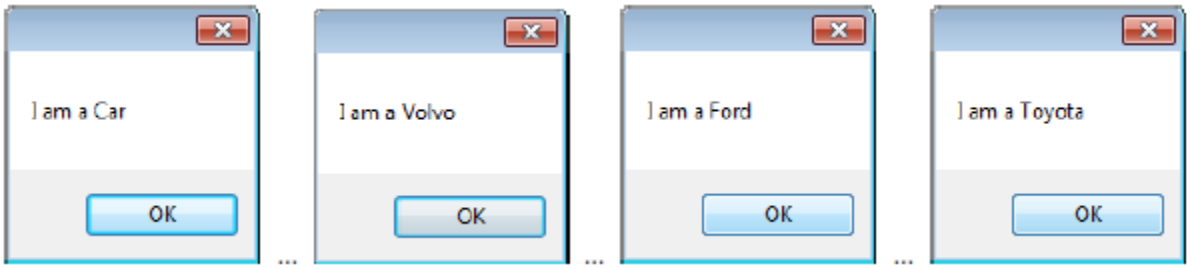
```
class Volvo : Car
{
	public override void CarType()
	{
		MessageBox.Show("I am a Volvo");
	}
}
class Ford : Car
{
	public override void CarType()
	{
		MessageBox.Show("I am a Ford");
	}
}
class Toyota : Car
{
	public override void CarType()
	{
		MessageBox.Show("I am a Toyota");
	}
}
```

این سه کلاس، کلاس Car را به ارث می‌برند. هر کلاس یک متد CarType() دارد و هر متد CarType() یک تعدیل‌کنندهٔ ابطال دارد. تعدیل‌کنندهٔ ابطال اجازه می‌دهد یک متد، متد مجازی پایهٔ خود را در زمان اجرا ابطال کند.

سپس می‌توانیم آن را استفاده کنیم:

```
Car[] car = new Car[4];
car[0] = new Car();
car[1] = new Volvo();
car[2] = new Ford();
car[3] = new Toyota();
foreach (Car carmodel in car)
{
      carmodel.CarType();
}
```

نتیجه این است:



← این، پولی‌مورفیسم است.

۵.۳. کپسوله سازی

کپسوله‌سازی یعنی این که نمایش داخلی یک شیء از دید خارج از تعریف شیء بطور کلی پنهان گردد. فقط متدهای دارای شیء می‌توانند مستقیماً فیلدهای آن را بطور نمونه بازبینی یا اداره کنند.

ما می‌توانیم اجازه‌های دسترسی مختلفی بر کلاس‌ها و متدها تعیین کنیم:

Access Modifier	Description (who can access)
private	Only members within the same type. (default for type members)
protected	Only derived types or members of the same type.
internal	Only code within the same assembly. Can also be code external to object as long as it is in the same assembly. (default for types)
protected internal	Either code from derived type or code in the same assembly. Combination of protected OR internal.
public	Any code. No inheritance, external type, or external assembly restrictions.

۶. استثناء گردانی

در برنامه نویسی خطا و استثناءگردانی بسیار مهم است. C# مکانیسم توکار و آماده‌ای برای کار با این مسئله دارد. این مکانیسم بر مبنای کلیدواژه‌های *try*، *catch*، *throw*، و *finally* است.

استثنائات خطاهای پیش‌بینی نشده هستند که در برنامه‌های ما رخ می‌دهند. بیشتر اوقات، می‌توانیم، و باید، خطاهای برنامه را یافته و رفع کنیم. برای مثال، تأیید اعتبار ورودی کاربر، چک کردن شیءهای تهی، و وارسی این که آیا مقادیر بازگشتی از متدها آن چیزیست که ما انتظار داریم، مثال‌هایی هستند از رفع خطای استاندارد خوب که باید همیشه انجام دهیم.

ولی، زمان‌هایی هست که نمی‌دانیم خطا رخ می‌دهد. برای مثال، نمی‌توانیم پیش‌بینی کنیم چه زمانی یک خطای I/O دریافت می‌کنیم، حافظهٔ سیستم کم می‌آید، یا با یک خطای پایگاه داده‌ها مواجه می‌شویم. اینها معمولاً بعید هستند، ولی هنوز امکان رخ دادنشان هست، و ما می‌خواهیم وقتی رخ می‌دهند قادر باشیم به آنها رسیدگی کنیم. این آن جاییست که استثناءگردانی به میان می‌آید.

وقتی استثنائات رخ می‌دهند، به آنها گفته می‌شود: «پرتاب شو». C# کلیدواژه‌های *try*، *catch*، *throw*، و *finally* را استفاده می‌کند. اینطور عمل می‌نماید: یک متد سعی خواهد کرد قسمتی از کد را اجرا کند. اگر کد یک مسئله بیابد، یک نشانهٔ خطا *پرتاب* می‌کند، که کد ما می‌تواند بگیرد، و بدون توجه به آنچه رخ می‌دهد، *نهایتاً* یک بلاک کد ویژه را در پایان اجرا می‌نماید.

ترکیب {نحو = syntax} آن بصورت زیر است:

```
MyMethod()
{
      try
      {
        ..................................... //Do Something that can cause an Exception
      }
      catch
      {
        ..................................... //Handle Exceptions
      }
      finally
      {
```

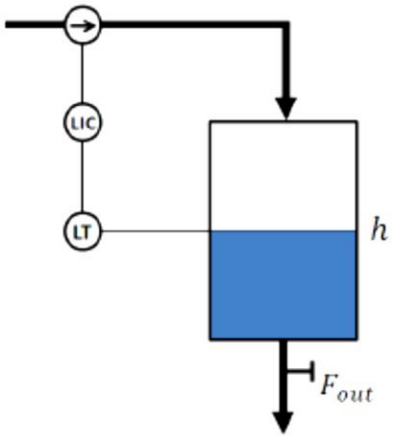
```
                                        //Clean Up
        }
}
```

مثال:

```
public void WriteDaqData(double analogDataOut)
        {
            Task analogOutTask = new Task();
            AOChannel myAOChannel;
            try
            {
                  myAOChannel = analogOutTask.AOChannels.CreateVoltageChannel(
                        aoChannel,
                        "myAOChannel",
                        0,
                        5,
                        AOVoltageUnits.Volts
                        );
                  AnalogSingleChannelWriter writer = new
                        AnalogSingleChannelWriter(analogOutTask.Stream);
                  writer.WriteSingleSample(true, analogDataOut);
            }
            catch (Exception e)
            {
                  string errorMessage;
                  errorMessage = e.Message.ToString();
            }
            finally
            {
                  analogOutTask.Stop();
            }
      }
```

۷. مثال فرم‌های ویندوز

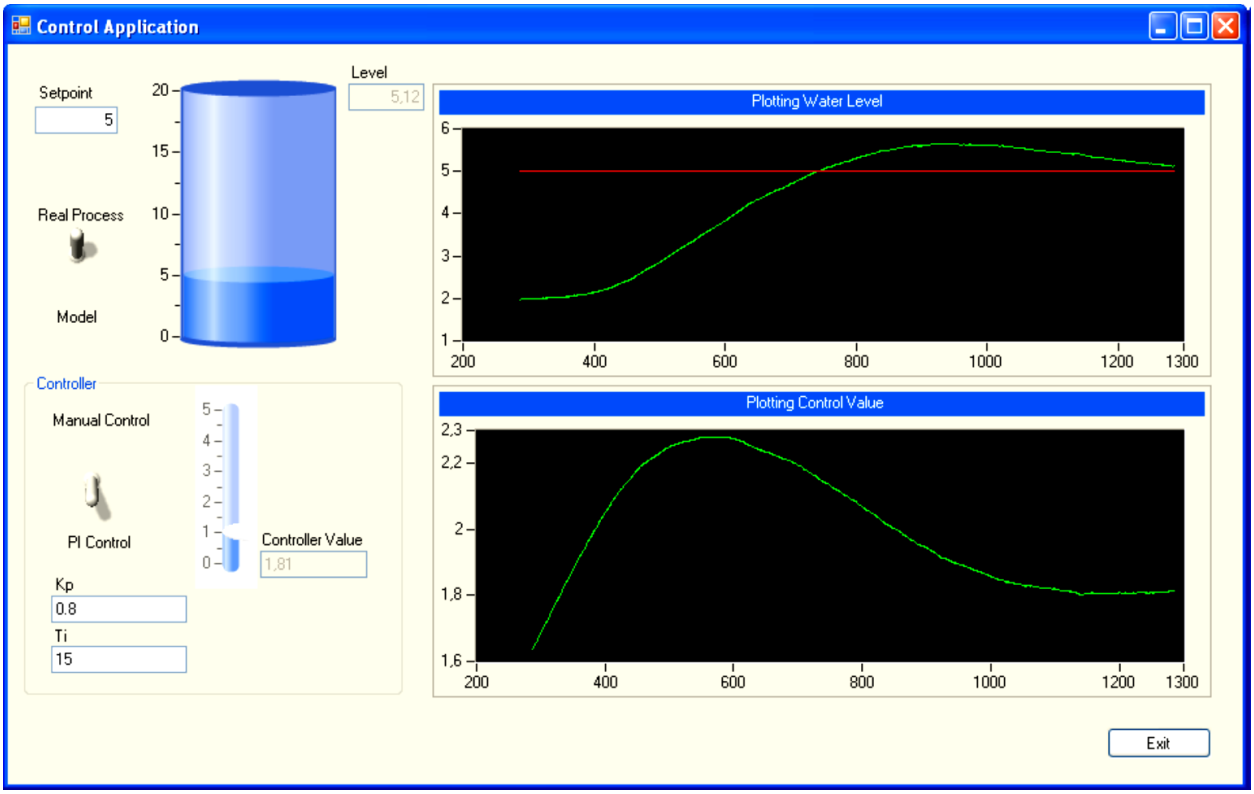
در این فصل ما یک مثال بزرگ را مرور می‌نماییم. در این مثال یک برنامه در ویژوال استودیو ایجاد می‌کنیم که فرآیند تانک آب زیر را کنترل می‌نماید:



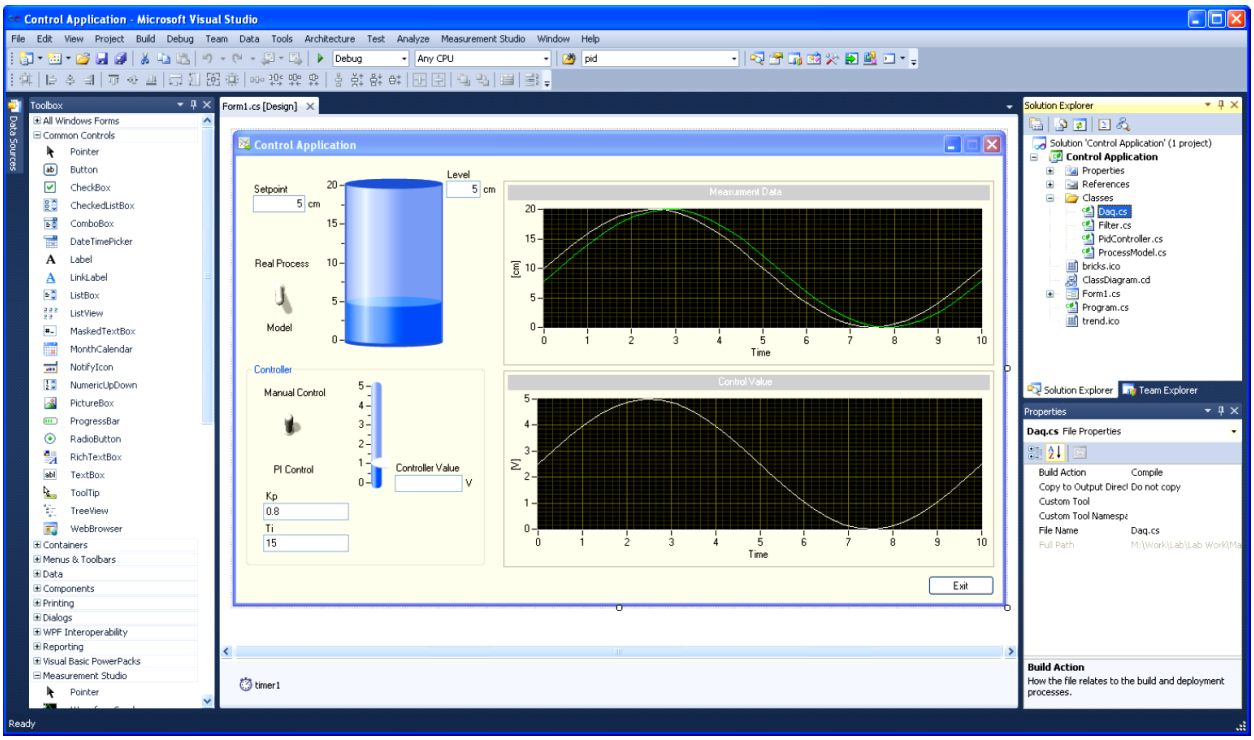
مقصود کنترل سطح در تانک آب است. در جهت ارتباط با فرآیند فیزیکی، یک دستگاه NI USB-6008 DAQ استفاده می‌کنیم:

ما تمام جزئیات در کد را نشان نمی‌دهیم، ولی بر ساختار تمرکز می‌نماییم.

در زیر رابط کاربری که در ویژوال استودیو ایجاد کرده‌ایم را می‌بینیم:

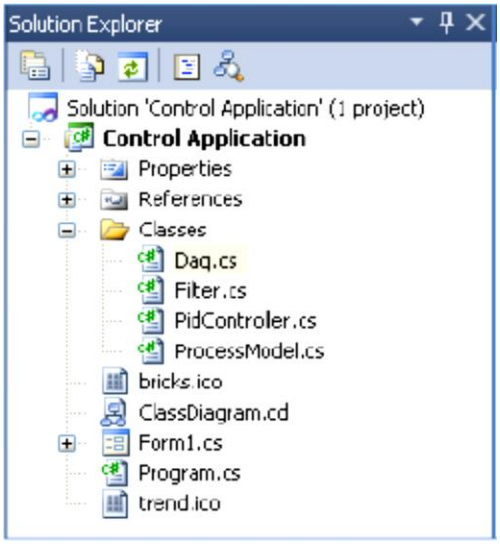


در زیر پروژهٔ ویژوال استودیو را می‌بینیم:

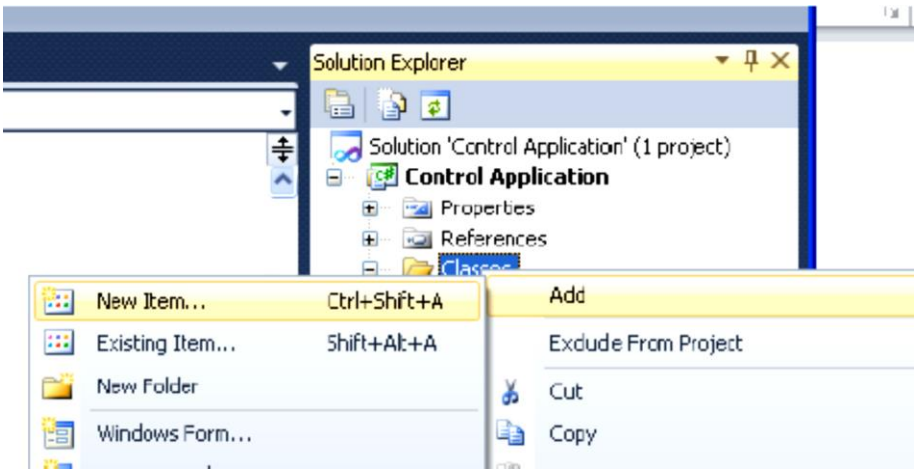


ما با تعریف کلاس‌هایی که در برنامهٔ خود نیاز داریم آغاز می‌کنیم.

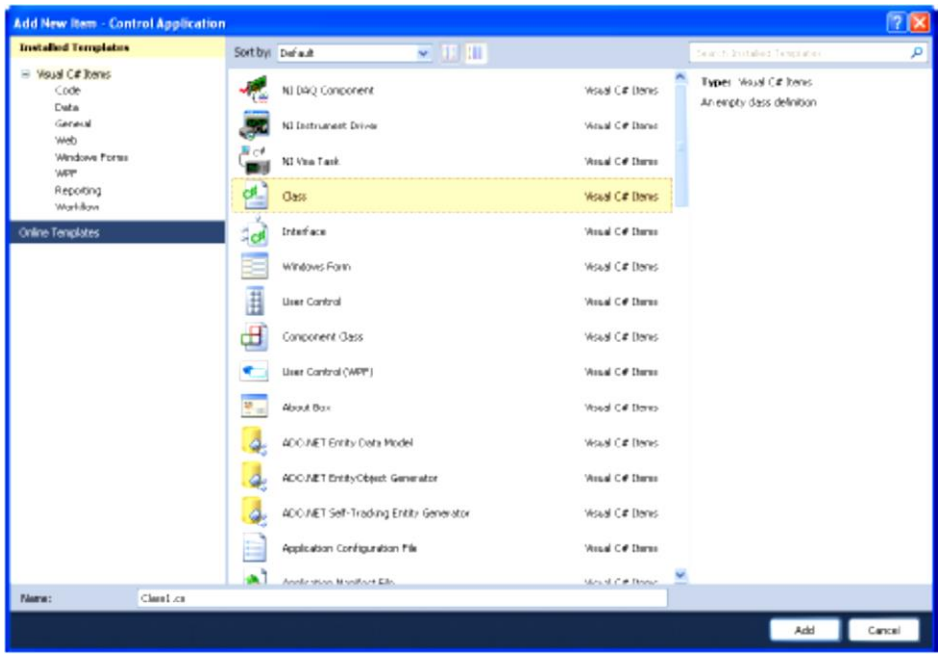
در Solution Explorer یک فولدر «Classes» تعریف می‌کنیم که در آن تمام کلاس‌هایی را که ایجاد کرده‌ایم می‌گذاریم:



برای ایجاد کلاس‌های جدید بر Solution Explorer کلیک راست می‌کنیم و Add-New Item... را انتخاب می‌نماییم:



سپس گزینهٔ Class را در پنجرهٔ Add New Item انتخاب می‌کنیم:



کلاس‌ها:

ما با کلاس Daq آغاز می‌کنیم که حاوی منطق برای خواندن و نوشتن بر دستگاه DAQ است:

```
//Include necessary Namespaces
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using NationalInstruments;
using NationalInstruments.DAQmx;
namespace Tuc.Control //We define a useful namespace
{
      public class DaqData
      {
            public string aiChannel;
            public string aoChannel;
            //Constructor
            public DaqData(                              )
            {

                  ----------------------------------------
            }
            //Method
            public double ReadDaqData()
            {

                  ----------------------------------------
            }
            //Method
            public void WriteDaqData(                                  )
            {

                  ----------------------------------------
            }
      }
}
```

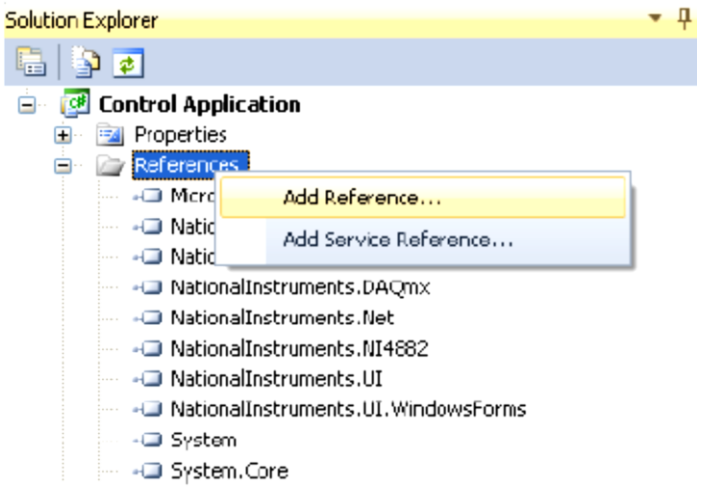
شیوهٔ خوبی است که برای هر کلاس جدیدی که تعریف می‌کنیم یک فایل ایجاد نماییم. سپس با تعریف یک فضای نام بامعنی شروع می‌کنیم. بعد، کلاس خود را با متدها، ویژگی‌ها و فیلدهای لازم تعریف می‌نماییم.

همچنین نیاز داریم فضاهای نام ضروری که کلاس ما احتیاج دارد را لحاظ کنیم. در این مثال برخی از اسمبلی‌های ۳.طرفی از National Instruments را لحاظ کرده‌ایم:

```
using NationalInstruments;
using NationalInstruments.DAQmx;
```

این اسمبلی‌ها حاوی درایور {گرداننده} برای دستگاه NI USB-6008 DAQ که استفاده می‌کنیم هستند.

اسمبلی‌هایی که استفاده کرده‌ایم باید به فولدر References در Solution Explorer اضافه شوند:



برنامهٔ اصلی: {Main Application}

در برنامهٔ اصلی ما با لحاظ نمودن فضای نام خود شروع می‌کنیم:

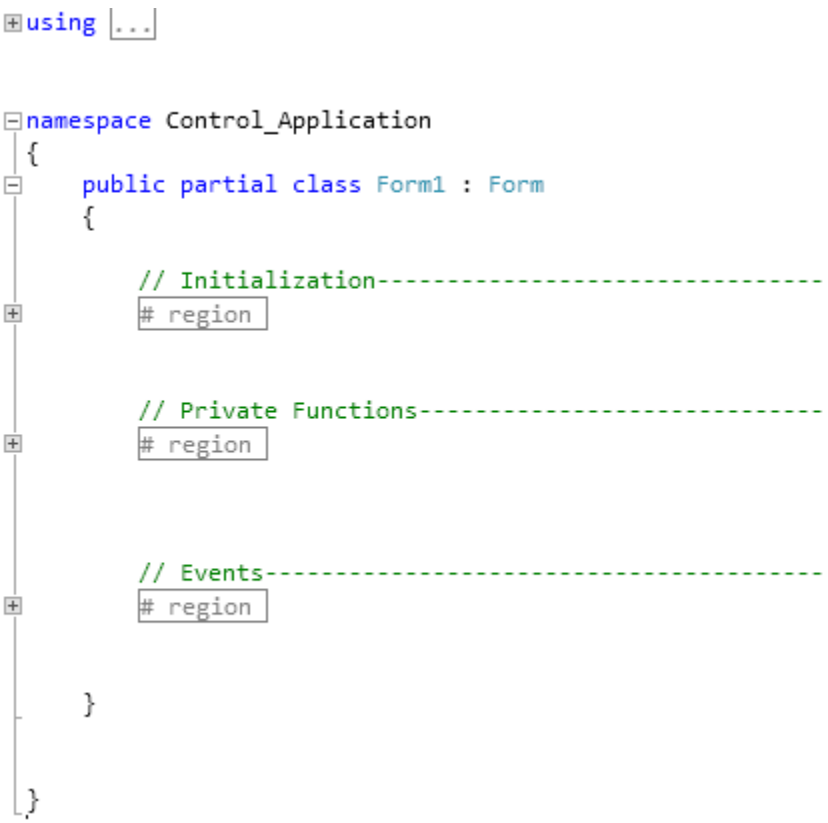
```
using NationalInstruments;
using NationalInstruments.UI;
using NationalInstruments.UI.WindowsForms;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using Tuc.Control;
namespace Control_Application
{
      public partial class Form1 : Form
```

```
        {
        }
}
```

همچنین ایدهٔ خوبی است که نواحی مختلفی ایجاد کنیم تا کد ما بهتر سازمان داده شود. این امر را می‌توان به این صورت انجام داد:

```
//Comment
# region
                                        //Your Code
# endregion
```

با این روش برنامهٔ اصلی ما ممکن است چنین به نظر آید:



۸. برنامه نویسی وب

۸.۱. مقدمه

امروزه بیشتر برنامه‌ها بر اینترنت تمرکز دارند، جایی که برنامه‌ها می‌توانند در یک مرورگر وب استاندارد دیده شوند. صفحات وب استاتیک براساس HTML و CSS می‌باشند. درجهت ایجاد برنامه‌های پیشرفته‌تر، نیاز به ابزارهای قدرتمندتر داریم.

ابزارها و چارچوبهای مهم برای پدیدآوری صفحات وب داینامیک {پویا} عبارتند از:

- ASP.NET
- AJAX/ ASP.NET AJAX
- JavaScript
- Silverlight

این ابزارها و چارچوبها در زیر شرح داده خواهند شد.

برای جزئیات بیشتر خودآموز ASP.NET and Web Programming را ببینید.

۸.۲. HTML

HTML که نمایندهٔ Hyper Text Markup Language {به معنای «زبان نشانه‌گذاری فرامتن} است، زبان نشانه‌گذاری غالب برای صفحات وب است. HTML بلوک‌های بنای پایهٔ صفحات وب است.

HTML در فرم اجزاء HTML شامل تگ‌ها، در میان علامت زاویه (مثل <html>)، بین محتوای صفحهٔ وب نوشته شده است. تگ‌های HTML بطور عادی بصورت جفت می‌آیند مثل <h1> و </h1>. تگ اول در یک جفت، تگ شروع است، تگ دوم، تگ پایان است (آنها را تگ‌های افتتاح و تگ‌های اختتام نیز می‌نامند). در طراحان وب می‌توانند در میان این تگ‌ها متن، جداول، اشکال و غیره را درج کنند.

۸.۳ مرورگرهای وب

مقصود یک مرورگر وب خواندن اسناد HTML و نوشتن آنها در میان صفحات وب دیداری یا شنیداری است. مرورگر تگ‌های HTML را نشان نمی‌دهد، ولی در تفسیر مندرجات صفحه از تگ‌ها استفاده می‌کند.

امروزه ما این مرورگرهای اصلی وب را داریم:

- مرورگر اینترنت (توسط میکروسافت) {اینترنت اکسپلورر}
- فایرفاکس (توسط موزیلا)
- کروم (توسط گوگل)
- سافاری (توسط اَپل)
- اُپرا (توسط اُپرا از نروژ)

CSS ۸.۴

مرورگر وب می‌تواند به برگه‌های تعریف آبشاری (CSS) {Cascading Style Sheets} نیز ارجاع نماید تا صفحه‌آرایی و ظهور متن و سایر چیزها را تعیین کند.

W3C، نگاهدارندهٔ هردوی استانداردهای HTML و CSS

JavaScript ۸.۵

JavaScript یک زبان برنامه‌نویسی شیء-گرا است که اساساً برای ایجاد صفحات وب داینامیک استفاده می‌شود. اصولاً JavaScript در فرم JavaScript سمت سرویس گیرنده استفاده شده، که بعنوان بخشی از یک مرورگر وب برای عرضهٔ رابطهای کاربری تسهیل شده و وب‌سایت‌های داینامیک اجرا شده است.

ASP.NET ۸.۶

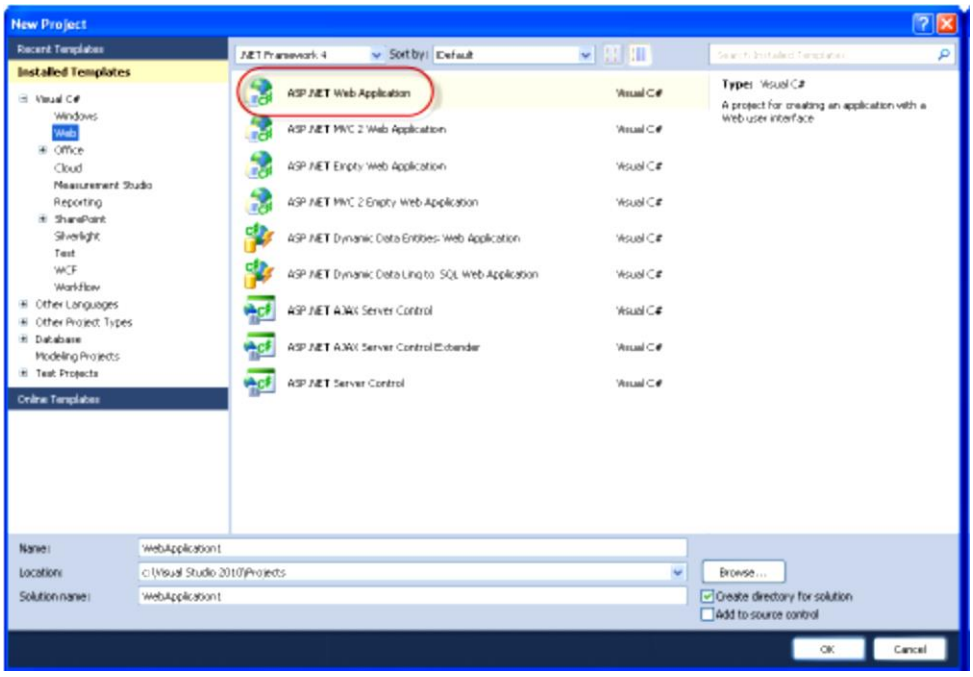
ASP.NET یک چارچوب وب توسعه یافته توسط میکروسافت است برای این که به برنامه‌نویسان اجازه دهد وب سایت‌های داینامیک، برنامه‌های وب و سرویس‌های وب بسازند.

ASP.NET جزء بستهٔ ویژوال استودیو است.

اول در ژانویهٔ ۲۰۰۲ با ویرایش 1.0 از چارچوب .NET منتشر شد، و جایگزین فن‌آوری صفحات سرور فعال (ASP) میکروسافت {Microsoft's Active Server Pages} شد. ASP.NET بر زبان مشترک زمان اجرا (CLR) مبتنی است{Common Language Runtime}، که به برنامه نویس اجازه می‌دهد کد ASP.NET را با استفاده از هر زبان .NET. پشتیبانی شده مثل C# و VB.NET بنویسند.

صفحات وب ASP.NET، (که رسماً بعنوان فرم‌های وب شناخته می‌شوند)، بلاک ساخت اصلی برای توسعهٔ برنامه می‌باشند. فرم‌های وب در فایل‌ها با پسوند aspx. قرار گرفته‌اند.

ASP.NET Application را در پنجرهٔ New Project انتخاب کنید:



۸.۷ AJAX/ ASP.NET AJAX

AJAX یک سرنام برای Asynchronous JavaScript and XML است. AJAX یک گروه از متدهای توسعهٔ وب مرتبط با هم است استفاده شده بر سمت کاربر در ایجاد برنامه‌های محاوره‌ای وب. با AJAX

برنامه‌های وب می‌توانند بطور ناهمگام داده‌ها را به یک سرور بفرستند و از آن دریافت کنند (در پس‌زمینه) بدون تداخل با نمایش و رفتار صفحهٔ موجود.

ASP.NET AJAX مجموعه‌ای از تعمیم‌ها به ASP.NET است که توسط میکروسافت برای انجام عاملیت AJAX توسعه یافته.

۸.۸. Silverlight {سیلورلایت}

میکروسافت سیلورلایت یک چارچوب برنامه برای نوشتن و اجرای پلاگین‌های مرورگر یا سایر برنامه‌های توانمند اینترنتی است، با ویژگی‌ها و مقاصدی شبیه به Adobe Flash. محیط زمان اجرا برای سیلورلایت بعنوان یک پلاگین برای بیشتر مرورگرها در دسترس است. سیلورلایت همچنین یکی از دو پلات‌فرم توسعهٔ برنامه برای ویندوز فون ۷/۸ است {Windows Phone 7/8}.

آخرین ویرایش سیلورلایت ۵.۰ است {Silverlight 5.0}.

سیلورلایت براساس WPF است، بنابراین در برنامه‌های سیلورلایت رابطهای کاربری به زبان نشانه‌گذاری کاربردی قابل توسعه (XAML) شناسانده شده‌اند و با استفاده از یک زیرمجموعه از چارچوب .NET. برنامه نویسی شده‌اند.

۹. برنامه نویسی پایگاه داده‌ها

بیشتر برنامه‌های امروز از یک پایگاه داده‌های باطنی استفاده می‌کنند تا داده‌های مهم را ذخیره نمایند، مثلاً Facebook، Twitter، و غیره.

ما به منظور استفاده از پایگاه داده‌ها در برنامه‌های خود نیاز داریم زبان پرس و جوی ساخت‌یافته (SQL) را بدانیم. برای اطلاعات بیشتر در مورد SQL خودآموز زیر را ببینید:

Structured Query Language (SQL)

بعلاوه لازم است دربارهٔ سیستم‌های پایگاه داده آگاه باشیم. ما انواع مختلفی از سیستم‌های پایگاه داده و بسیاری از فروشندگان گوناگون داریم. از آنجا که این خودآموز دربارهٔ ویژوال استودیو و C# است، ما از **Microsoft SQL Server** استفاده می‌کنیم. برای اطلاعات بیشتر دربارهٔ سیستم‌های پایگاه داده در کل و مخصوصاً SQL Server خودآموز زیر را ببینید:

Introduction to Database Systems

خودآموزها از http://home.hit.no/~hansha در دسترس می‌باشند.

۹.۱ ADO.NET

ADO.NET (ActiveX Data Object for .NET) مجموعه‌ای از مؤلفه‌های نرم‌افزاری کامپیوتر است که برنامه‌نویسان می‌توانند برای دست‌یابی به داده‌ها و خدمات استفاده کنند. بخشی از کتابخانهٔ کلاس پایه است که با چارچوب Microsoft .NET لحاظ شده است. عموماً توسط برنامه‌نویسان برای دسترسی و اصلاح داده‌های ذخیره شده در سیستم‌های پایگاه دادهٔ رابطه‌ای استفاده شده، و از طریق آن می‌توان به داده‌ها در منابع غیررابطه‌ای نیز دسترسی پیدا کرد.

Hans-Petter Halvorsen, M.Sc.

E-mail: hans.p.halvorsen@hit.no
Blog: http://home.hit.no/~hansha/

کالج دانشگاهی جنوب شرق نروژ

www.usn.no

ترجمه و ویرایش: سید نورالدین رفیعی طباطبائی

اصفهان، فروردین ۱۳۹۶ خورشیدی.