

DOT NET Framework

Unit – 1 Module – 1

Beginning ASP.NET 4.5 in C#

Matthew MacDonald

Pages 9 – 16

Introduction

- **DOT NET Framework** is a software framework developed by Microsoft that runs primarily on Microsoft Windows.
- It includes the following
 - *Class library named Framework Class Library (FCL)*
 - *Programs written for .NET Framework execute in a software environment named Common Language Runtime(CLR), an application that provides services such as security, memory management, and exception handling.*

.NET Framework Architecture



DOT NET Languages

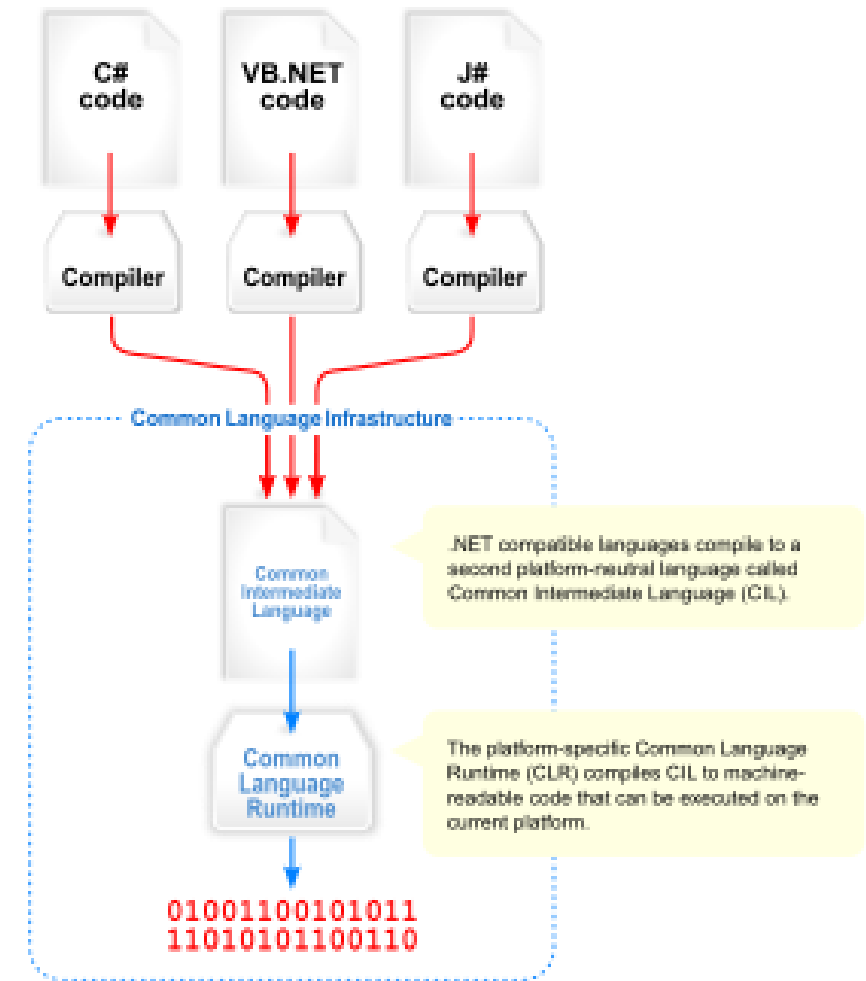
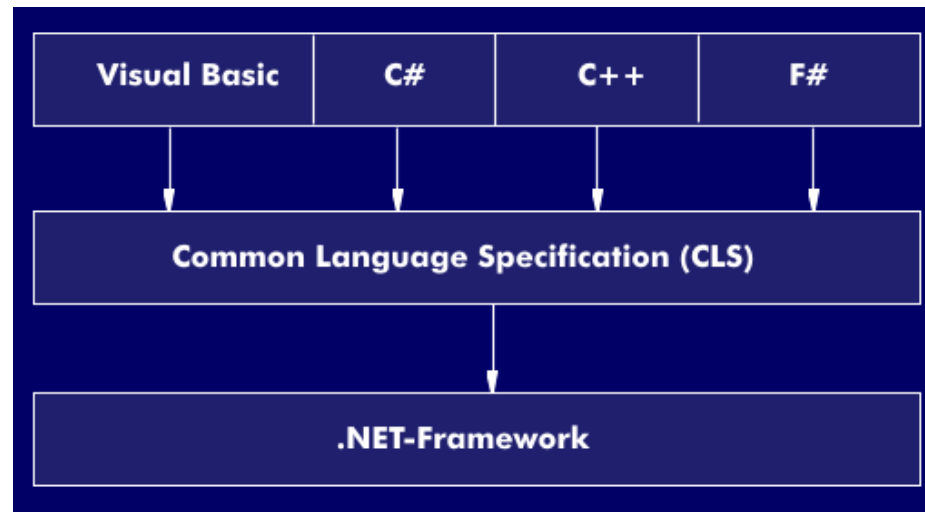
- Both **VB and C#** use the .NET class library and are supported by the CLR. In fact, almost any block of C#code can be translated, line by line, into an equivalent block of VB code (and vice versa).
- A developer who has learned one .NET language can move quickly and efficiently to another.
- In short, both VB and C# are elegant, modern languages that are ideal for creating the next generation of web applications.

Intermediate Language

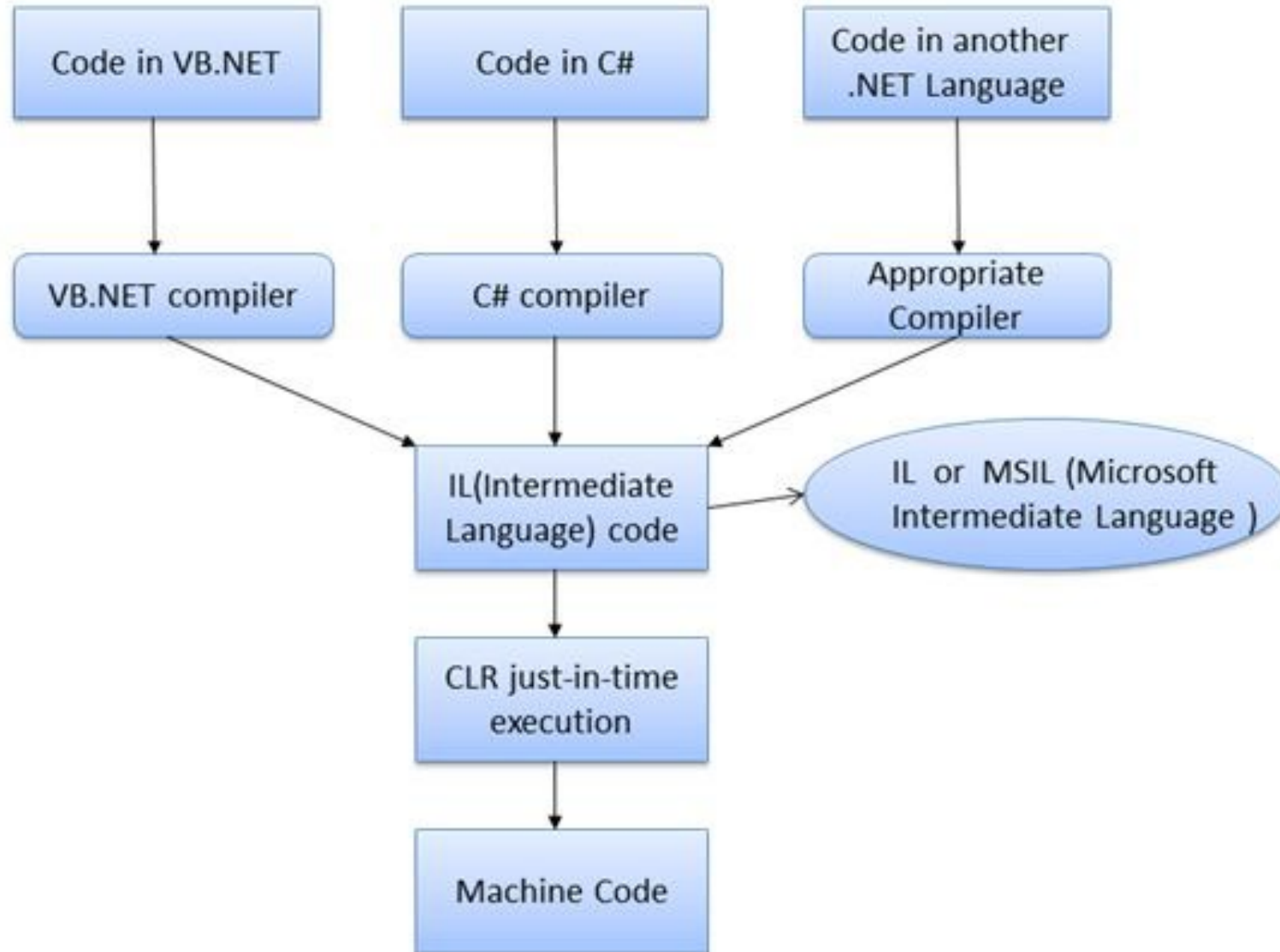
- All the .NET languages are compiled into another lower-level language before the code is executed. This lower-level language is the Common Intermediate Language (CIL, or just IL).
- The CLR, the engine of .NET, uses only IL code. Because all .NET languages are designed based on IL, they all have profound similarities. This is the reason that the VB and C# languages provide essentially the same features and performance.
- The .NET Framework formalizes this compatibility with something called the Common Language Specification (CLS). Essentially, the CLS is a contract that, if respected, guarantees that a component written in one .NET language can be used in all the others. One part of the CLS is the common type system (CTS), which defines the rules for data types such as strings, numbers, and arrays that are shared in all .NET languages. The CLS also defines object oriented ingredients such as classes, methods, events, and quite a bit more.
- For the most part, .NET developers don't need to think about how the CLS works, even though they rely on it every day.
- Every EXE or DLL file that you build with a .NET language contains IL code. This is the file you deploy to other computers. In the case of a web application, you deploy your compiled code to a live web server.

Common Language Specification

- A **Common Language Specification (CLS)** is a document that says how computer programs can be turned into Common Intermediate Language (CIL) code. When several languages use the same bytecode, different parts of a program can be written in different languages.
- Microsoft uses a Common Language Specification for their .NET Framework.
- Microsoft has defined CLS which are nothing but guidelines for languages to follow so that it can communicate with other .NET languages in a seamless manner.



Compilation in .NET



The Common Language Runtime

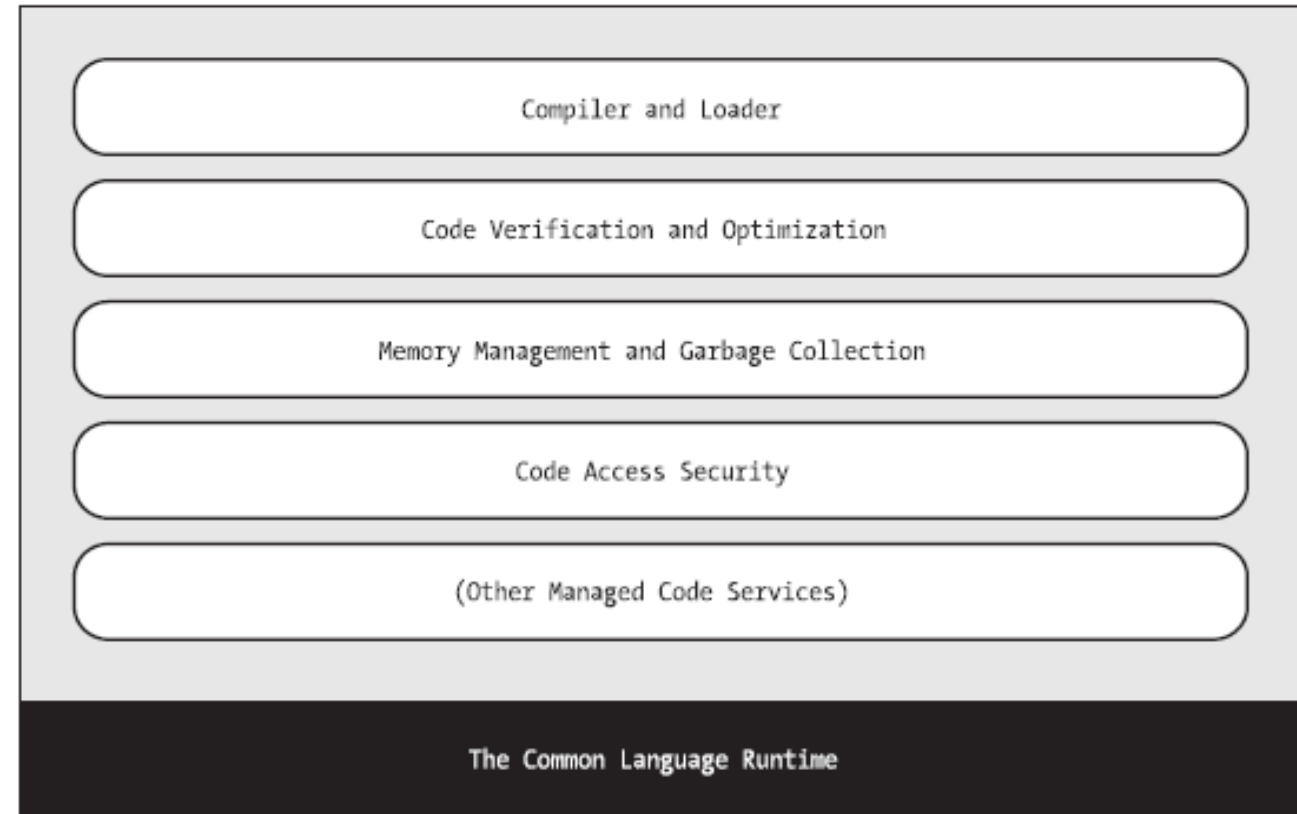
The CLR is the engine that supports all the .NET languages. Many modern languages use runtimes.

These runtimes may provide libraries used by the language, or they may have the additional responsibility of executing the code (as with Java).

Not only does the CLR execute code, it also provides a whole set of related services such as code verification, optimization, and object management.

All .NET code runs inside the CLR - running a Windows application or a web service.

For example, when a client requests an ASP.NET web page, the ASP.NET service runs inside the CLR environment, executes your code, and creates a final HTML page to send to the client.



The implications of the CLR are wide-ranging:

Deep language integration: VB and C#, like all .NET languages, compile to IL. This is far more than mere language compatibility; it's language *integration*.

Side-by-side execution: The CLR also has the ability to load more than one version of a component at a time. In other words, you can update a component many times, and the correct version will be loaded and used for each application. As a side effect, multiple versions of the .NET Framework can be installed

Fewer errors: Whole categories of errors are impossible with the CLR. For example, the CLR prevents many memory mistakes that are possible with lower-level languages such as C++.

Performance: A typical ASP.NET application is much faster than a comparable ASP application, because ASP.NET code is compiled to machine code before it's executed. With high-volume web applications, the potential bottlenecks are rarely processor-related but are usually tied to the speed of an external resource such as a database or the web server's file system.

Code transparency If you distribute a compiled application or component, other programmers may have an easier time determining how your code works. This isn't much of an issue for ASP.NET applications, which aren't distributed but are hosted on a secure web server.

Questionable cross-platform support: No one is entirely sure whether .NET will ever be adopted for use on other operating systems and platforms. However, .NET will probably never have the wide reach of a language such as Java because it incorporates too many different platform-specific and operating system-specific technologies and features.

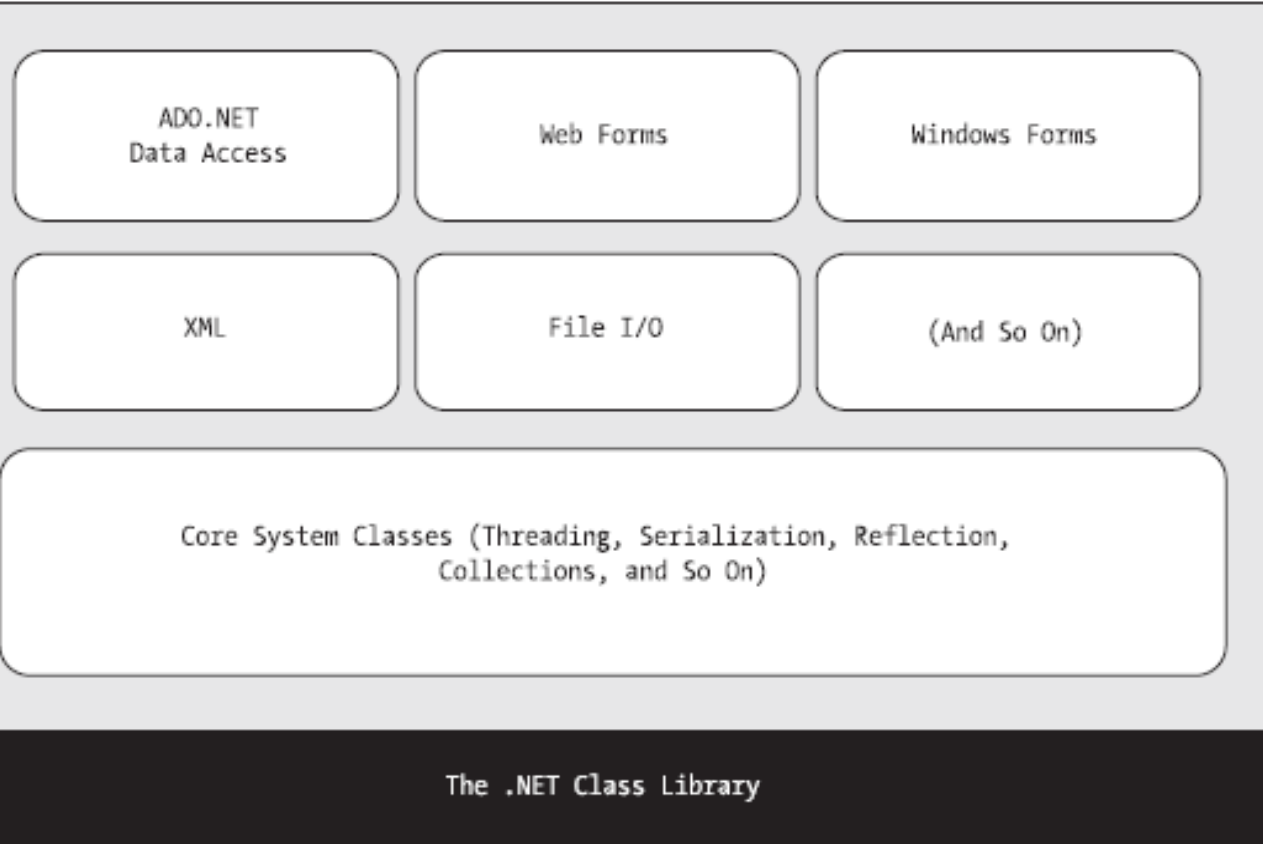
The .NET Class Library

The .NET class library is a giant repository of classes that provide prefabricated functionality for everything from reading an XML file to sending an e-mail message.

Any .NET language can use the .NET class library's features by interacting with the right objects. This helps encourage consistency among different .NET languages and removes the need to install numerous components on your computer or web server.

Some parts of the class library are meant for desktop applications with the Windows interface and some are targeted directly at web development.

There are some classes that can be used in various programming scenarios and aren't specific to web or Windows development. These include the base set of classes that define common variable types and the classes for data access.



DOT NET Framework

Assignment – 1

Refer <https://docs.microsoft.com/en-us/dotnet/framework/get-started/overview> and answer the following questions

- Objectives of DOT NET Framework
- Write a note on the following terms
 - Class Library
 - CLS-Compliant
- Features of CLR

Refer <https://docs.microsoft.com/en-us/dotnet/standard/common-type-system> and answer the following questions

- What are the activities do CTS perform ?
- What are the types and categories of data types supported by CTS ?

C# Basics

Unit – 1 Module – 2

Beginning ASP.NET 4.5 in C#

Matthew MacDonald

Chapter 2 & 3

Points to Remember

- Case-Sensitive
- Statements are terminated with a semi-colon
- Control Statements and loops should be enclosed with { }
- Comments – Single-Line (//) and Multi-Line (/* */)
- Escaped Characters - \n, \t, \\, \"

Data Types

C# Name	VB Name	.NET Type Name	Contains
byte	Byte	Byte	An integer from 0 to 255.
short	Short	Int16	An integer from -32,768 to 32,767.
int	Integer	Int32	An integer from -2,147,483,648 to 2,147,483,647.
long	Long	Int64	An integer from about -9.2e18 to 9.2e18.
float	Single	Single	A single-precision floating point number from approximately -3.4e38 to 3.4e38 (for big numbers) or -1.5e-45 to 1.5e-45 (for small fractional numbers).
double	Double	Double	A double-precision floating point number from approximately -1.8e308 to 1.8e308 (for big numbers) or -5.0e-324 to 5.0e-324 (for small fractional numbers).
decimal	Decimal	Decimal	A 128-bit fixed-point fractional number that supports up to 28 significant digits.
char	Char	Char	A single 16-bit Unicode character.
string	String	String	A variable-length series of Unicode characters.
bool	Boolean	Boolean	A true or false value.
*	Date	DateTime	Represents any date and time from 12:00:00 AM, January 1 of the year 1 in the Gregorian calendar, to 11:59:59 PM, December 31 of the year 9999. Time values can resolve values to 100 nano-second increments. Internally, this data type is stored as a 64-bit integer.
*	*	TimeSpan	Represents a period of time, as in ten seconds or three days. The smallest possible interval is 1 <i>tick</i> (100 nanoseconds).
object	Object	Object	The ultimate base class of all .NET types. Can contain any data type or object.

Arrays and ArrayLists

- 1D Array – `string[] stringArray = new string[4];`
`string[] stringArray = {"1", "2", "3", "4"};`
- 2D Array – `int[,] intArray = new int[2, 4];`
`int[,] intArray = {{1, 2}, {3, 4}, {5, 6}, {7, 8}};`
- ArrayList
- C# arrays do not support redimensioning. This means that once you create an array, you can't change its size. If you need a dynamic array-like list, you can use one of the collection classes provided to all .NET languages
- through the .NET class library. One of the simplest collection classes that .NET offers is the ArrayList, which always allows dynamic resizing. Here's a snippet of C# code that uses an ArrayList:
- `ArrayList dynamicList = new ArrayList();`
- `// The ArrayList is not strongly typed, so you can add any data type`
- `dynamicList.Add("one");`
- `dynamicList.Add("two");`
- `dynamicList.Add("three");`
- `// Retrieve the first string. Notice that the object must be converted to a string, because there's no way for .NET to be certain what it is.`
- `string item = Convert.ToString(dynamicList[0]);`

TypeCasting & String Type

```
double myValue;  
myValue = Math.Sqrt(81);           // myValue = 9.0  
myValue = Math.Round(42.889, 2);   // myValue = 42.89  
myValue = Math.Abs(-10);           // myValue = 10.0  
myValue = Math.Log(24.212);        // myValue = 3.18.. (and so on)  
myValue = Math.PI;                 // myValue = 3.14.. (and so on)
```

```
string countString = "10";
```

```
// Convert the string "10" to the numeric value 10.  
int count = Convert.ToInt32(countString);
```

```
// Convert the numeric value 10 into the string "10".  
countString = Convert.ToString(count);
```

```
string myString = "This is a test string";  
myString = myString.Trim();           // = "This is a test string"  
myString = myString.Substring(0, 4);  // = "This"  
myString = myString.ToUpper();        // = "THIS"  
myString = myString.Replace("IS", "AT"); // = "THAT"
```

```
int length = myString.Length;         // = 4
```


Operator	Description	Example
+	Addition	$1 + 1 = 2$
-	Subtraction (and to indicate negative numbers)	$5 - 2 = 3$
*	Multiplication	$2 * 5 = 10$
/	Division	$5.0 / 2 = 2.5$
%	Gets the remainder left after integer division	$7 \% 3 = 1$

Operator	Description
==	Equal to.
!=	Not equal to.
<	Less than.
>	Greater than.
<=	Less than or equal to.
>=	Greater than or equal to.
&&	Logical and (evaluates to true only if both expressions are true). If the first expression is false, the second expression is not evaluated.
	Logical or (evaluates to true if either expression is true). If the first expression is true, the second expression is not evaluated.

Member	Description
Length	Returns the number of characters in the string (as an integer).
ToUpper() and ToLower()	Returns a copy of the string with all the characters changed to uppercase or lowercase characters.
Trim(), TrimEnd(), and TrimStart()	Removes spaces or some other characters from either (or both) ends of a string.
Insert()	Puts another string inside a string at a specified (zero-based) index position. For example, Insert(1, "pre") adds the string <i>pre</i> after the first character of the current string.
Remove()	Removes a specified number of strings from a specified position. For example, Remove(0, 1) removes the first character.
Replace()	Replaces a specified substring with another string. For example, Replace("a", "b") changes all <i>a</i> characters in a string into <i>b</i> characters.
Substring()	Extracts a portion of a string of the specified length at the specified location (as a new string). For example, Substring(0, 2) retrieves the first two characters.
StartsWith() and EndsWith()	Determines whether a string starts or ends with a specified substring. For example, StartsWith("pre") will return either true or false, depending on whether the string begins with the letters <i>pre</i> in lowercase.
IndexOf() and LastIndexOf()	Finds the zero-based position of a substring in a string. This returns only the first match and can start at the end or beginning. You can also use overloaded versions of these methods that accept a parameter that specifies the position to start the search.
Split()	Divides a string into an array of substrings delimited by a specific substring. For example, with Split(".") you could chop a paragraph into an array of sentence strings.
Join()	Fuses an array of strings into a new string. You can also specify a separator that will be inserted between each element.

Table 2-4. Useful DateTime Members

Member	Description
Now	Gets the current date and time. You can also use the <code>UtcNow</code> property to take the current computer's time zone into account. <code>UtcNow</code> gets the time as a <i>coordinated universal time</i> (UTC). Assuming your computer is correctly configured, this corresponds to the current time in the Western European (UTC+0) time zone.
Today	Gets the current date and leaves time set to 00:00:00.
Year, Date, Month, Day, Hour, Minute, Second, and Millisecond	Returns one part of the <code>DateTime</code> object as an integer. For example, <code>Month</code> will return 12 for any day in December.
DayOfWeek	Returns an enumerated value that indicates the day of the week for this <code>DateTime</code> , using the <code>DayOfWeek</code> enumeration. For example, if the date falls on Sunday, this will return <code>DayOfWeek.Sunday</code> .
Add() and Subtract()	Adds or subtracts a <code>TimeSpan</code> from the <code>DateTime</code> .
AddYears(), AddMonths(), AddDays(), AddHours(), AddMinutes(), AddSeconds(), AddMilliseconds()	Adds an integer that represents a number of years, months, and so on, and returns a new <code>DateTime</code> . You can use a negative integer to perform a date subtraction.
DaysInMonth()	Returns the number of days in the specified month in the specified year.
IsLeapYear()	Returns true or false depending on whether the specified year is a leap year.
ToString()	Returns a string representation of the current <code>DateTime</code> object. You can also use an overloaded version of this method that allows you to specify a parameter with a format string.

DateTime

- `DateTime dob = new DateTime(1974, 7, 10, 7, 10, 24);`
- `Console.WriteLine("Day:{0}", dob.Day);`
- `Console.WriteLine("Month:{0}", dob.Month);`
- `Console.WriteLine("Year:{0}", dob.Year);`
- `Console.WriteLine("Hour:{0}", dob.Hour);`
- `Console.WriteLine("Minute:{0}", dob.Minute);`
- `Console.WriteLine("Second:{0}", dob.Second);`
- `Console.WriteLine("Millisecond:{0}", dob.Millisecond);`
- `dob.AddYears(2);`
- `dob.AddDays(12);`
- `dob.AddHours(4);`
- `dob.AddMinutes(15);`
- `dob.AddSeconds(45);`
- `dob.AddMilliseconds(200);`

DateTime and TimeSpan Objects

```
DateTime dob = new DateTime(2000, 10, 20, 12, 15, 45);
DateTime subDate = new DateTime(2000, 2, 6, 13, 5, 15);
    // TimeSpan with 10 days, 2 hrs, 30 mins, 45 seconds, and 100 milliseconds
TimeSpan ts = new TimeSpan(10, 2, 30, 45, 100);
    // Subtract a DateTime
TimeSpan diff1 = dob.Subtract(subDate);
Console.WriteLine(diff1.ToString());
    // Subtract a TimeSpan
DateTime diff2 = dob.Subtract(ts);
Console.WriteLine(diff2.ToString());
    // Subtract 10 Days
DateTime daysSubtracted = new DateTime(dob.Year, dob.Month, dob.Day - 10);
Console.WriteLine(daysSubtracted.ToString());
    // Subtract hours, minutes, and seconds
DateTime hms = new DateTime(dob.Year, dob.Month, dob.Day, dob.Hour - 1, dob.Minute - 15, dob.Second - 15);
Console.WriteLine(hms.ToString());
```

Comparing two dates

```
DateTime firstDate = new DateTime(2002, 10, 22);  
DateTime secondDate = new DateTime(2009, 8, 11);  
int result = DateTime.Compare(firstDate, secondDate);  
  
if (result < 0)  
    Console.WriteLine("First date is earlier");  
else if (result == 0)  
    Console.WriteLine("Both dates are same");  
else  
    Console.WriteLine("First date is later");
```

Date Formatting

- `Console.WriteLine(dob.ToString("r"));`

```
d: 10/22/2002
D: Tuesday, October 22, 2002
f: Tuesday, October 22, 2002 12:00 AM
F: Tuesday, October 22, 2002 12:00:00 AM
g: 10/22/2002 12:00 AM
G: 10/22/2002 12:00:00 AM
m: October 22
M: October 22
```

```
t: 12:00 AM
T: 12:00:00 AM
```

```
y: October, 2002
Y: October, 2002
```

Table 2-6. *Useful Array Members*

Member	Description
Length	Returns an integer that represents the total number of elements in all dimensions of an array. For example, a 3×3 array has a length of 9.
GetLowerBound() and GetUpperBound()	Determines the dimensions of an array. As with just about everything in .NET, you start counting at zero (which represents the first dimension).
Clear()	Empties part or all of an array's contents, depending on the index values that you supply. The elements revert to their initial empty values (such as 0 for numbers).
IndexOf() and LastIndexOf()	Searches a one-dimensional array for a specified value and returns the index number. You cannot use this with multidimensional arrays.
Sort()	Sorts a one-dimensional array made up of comparable data such as strings or numbers.
Reverse()	Reverses a one-dimensional array so that its elements are backward, from last to first.

Control Statements and Loops

```
if (myNumber > 10)
{
    // Do something.
}
else if (myString == "hello")
{
    // Do something.
}
else
{
    // Do something.
}
```

```
switch (myNumber)
{
    case 1:
        // Do something.
        break;
    case 2:
        // Do something.
        break;
    default:
        // Do something.
        break;
}
```

```
for (int i = 0; i < 10; i++)
{
    // This code executes ten times.
    System.Diagnostics.Debug.Write(i);
}
```

```
int i = 0;
while (i < 10)
{
    i += 1;
    // This code executes ten times.
}
```

```
string[] stringArray = {"one", "two", "three"};

foreach (string element in stringArray)
{
    // This code loops three times, with the element variable set to
    // "one", then "two", and then "three".
    System.Diagnostics.Debug.Write(element + " ");
}
```

```
int i = 0;
do
{
    i += 1;
    // This code executes ten times.
}
while (i < 10);
```

Methods

```
// This method doesn't return any information.
void MyMethodNoReturnedData()
{
    // Code goes here.
}

// This method returns an integer.
int MyMethodReturnsData()
{
    // As an example, return the number 10.
    return 10;
}
```

```
private int AddNumbers(int number1, int number2)
{
    return number1 + number2;
}
```

```
private decimal GetProductPrice(int ID)
{
    // Code here.
}

private decimal GetProductPrice(string name)
{
    // Code here.
}

// And so on...
```

```
// Get price by product ID (the first version).
price = GetProductPrice(1001);

// Get price by product name (the second version).
price = GetProductPrice("DVD Player");
```

Classes

```
public class MyClass           public class Product           Product saleProduct = new Product();
{                               {                               // Optionally you could do this in two steps:
    // Class code goes here.   private string name;   // Product saleProduct;
                               private decimal price;         // saleProduct = new Product();
                               private string imageUrl;
                               }
                               // Now release the object from memory.
                               saleProduct = null;
```

Table 3-1. Accessibility Keywords

Keyword	Accessibility
public	Can be accessed by any class
private	Can be accessed only by members inside the current class
internal	Can be accessed by members in any of the classes in the current assembly (the compiled code file)
protected	Can be accessed by members in the current class or in any class that inherits from this class
protected internal	Can be accessed by members in the current application (as with internal) <i>and</i> by the members in any class that inherits from this class

```

using System;
namespace demo
{
    class Student
    {
    private string code = "N.A";
    private string name = "not known";
    private int age = 0;
    // Declare a Code property of type string:
    public string Code
    {
        get
            { return code; }

        set
            { code = value; }
    }
    // Declare a Name property of type string:
    public string Name
    {
        get
            { return name; }

        set { name = value; }
    }
    // Declare a Age property of type int:
    public int Age
    {
        get
            { return age; }

        set
            { age = value; }
    }
    public override string ToString()
    {
        return "Code = " + Code + ", Name = " + Name + ", Age
= " + Age; }
    }
}

```

```

class ExampleDemo
{
    public static void Main()
    {
        // Create a new Student object:
        Student s = new Student();
        // Setting code, name and the age of the student
        s.Code = "001";
        s.Name = "Zara";
        s.Age = 9;
        Console.WriteLine("Student Info: {0}", s);
        //let us increase age
        s.Age += 1;
        Console.WriteLine("Student Info: {0}", s);
        Console.ReadKey();
    }
}

```

Adding Properties

```
public class Product
{
    private string name;
    private decimal price;
    private string imageUrl;
    public string Name
    {
        get
        { return name; }
        set
        { name = value; }
    }

    public decimal Price
    {
        get
        { return price; }
        set
        { price = value; }
    }

    public string ImageUrl
    {
        get
        { return imageUrl; }
        set
        { imageUrl = value; }
    }
}
```

```
Product saleProduct = new Product();
saleProduct.Name = "Kitchen Garbage";
saleProduct.Price = 49.99M;
saleProduct.ImageUrl = "http://mysite/garbage.png";
```

```
public class Product
{
    // (Variables and properties omitted for clarity.)

    public string GetHtml()
    {
        string htmlString;
        htmlString = "<h1>" + name + "</h1><br />";
        htmlString += "<h3>Costs: " + price.ToString() + "</h3><br />";
        htmlString += "<img src='" + imageUrl + "' />";
        return htmlString;
    }
}
```

```
public Product(string name, decimal price)
{
    // Set the two properties in the class.
    Name = name;
    Price = price;
}
```



Some more Concepts of DOT NET



Casting Objects - I

Student s = new Student(12,"abc"); //Student is a Base class
Sy sy = new Sy(13, "SYStudent",'D'); // sy is a derived class
from Student

Console.WriteLine("Type Casting - 1");
Student s1 = new Student(555, "newbase1");
Console.WriteLine("Before casting");
s1.printStudent();
Console.WriteLine("After casting");
s1=sy;
s1.printStudent();



Output -1

```
Type Casting - 1
Before casting
ConsoleApplication2.Student
555
newbase1
After casting
ConsoleApplication2.Sy
13
SYStudent
_
```




Casting Objects - II

```
Console.WriteLine("Type Casting - 3");  
Sy s3 = new Sy(23, "newderived", 'D');  
Student s4 = new Student(54, "newbase");  
Console.WriteLine("Before casting");  
s4.printStudent();  
s4 = (Student)s3;  
Console.WriteLine("After casting");  
s4.printStudent();  
  
Console.ReadKey();
```



Output- 2

```
Type Casting - 3  
Before casting  
ConsoleApplication2.Student  
54  
newbase  
After casting  
ConsoleApplication2.Sy  
23  
newderived  
_
```



Casting Objects - III

```
Console.WriteLine("Type Casting - 3");  
Sy sy2 = new Sy(555, "newderived1", 'D');  
Console.WriteLine("Before casting");  
sy2.printStudent();  
Console.WriteLine("After casting");  
sy2 = s;  
sy2.printStudent();  
Console.ReadKey();
```



Output -3

- Compile Error. (Cannot implicitly convert Student to Sy)




Casting Objects - IV

```
Console.WriteLine("Type Casting - 4");  
    Student s = new Student(12,"abc");  
    Sy sy2 = new Sy(555, "newderived1", 'D');  
    Console.WriteLine("Before casting");  
    sy2.printStudent();  
    Console.WriteLine("After casting");  
    sy2 = (Sy)s;  
  
    sy2.printStudent();  
    Console.ReadKey();
```



Output -4

- Run time Error. (Invalid Cast Exception)



```
static void Main(string[] args)
{
    sy sy1 = new sy();
    student s1 = sy1;
    s1.stream = "CS";
}
```

```
static void Main(string[] args)
{
    student s1 = new sy();
    sy sy1 = (sy)s1;
    sy1.stream = "CS";
}
```

```
static void Main(string[] args)
{

    student s1 = new student();
    sy sy1 = (sy)s1;

}
```




ASP.NET File Types

File Name

► Ends with **.aspx**

These are ASP.NET web pages. They contain the user interface and, optionally, the underlying application code. Users request or navigate directly to one of these pages to start your web application.

► Ends with **.ascx**

These are ASP.NET user controls. User controls are similar to web pages, except that the user can't access these files directly. Instead, they must be hosted inside an ASP.NET web page. User controls allow you to develop a small piece of user interface and reuse it in as many web forms as you want without repetitive code.

► **web.config**

This is the configuration file for your ASP.NET application. It includes settings for customizing security, state management, memory management, and much more.

► **global.asax**

This is the global application file. You can use this file to define global variables (variables that can be accessed from any web page in the web application) and react to global events (such as when a web application first starts).

► Ends with **.cs**

These are code-behind files that contain C# code. They allow you to separate the application logic from the user interface of a web page. We'll introduce the code-behind model in this chapter and use it extensively in this book.



ASP.NET Folders

➤ **App_Browsers**

Contains .browser files that ASP.NET uses to identify the browsers that are using your application and determine their capabilities. Usually, browser information is standardized across the entire web server, and you don't need to use this folder.

➤ **App_Code**

Contains source code files that are dynamically compiled for use in your application.

➤ **App_GlobalResources**

Stores global resources that are accessible to every page in the web application. This directory is used in localization scenarios, when you need to have a website in more than one language.

➤ **App_LocalResources**

Serves the same purpose as App_GlobalResources, except these resources are accessible to a specific page only.

➤ **App_WebReferences**

Stores references to web services, which are remote code routines that a web application can call over a network or the Internet.

➤ **App_Data**

Stores data, including SQL Server Express database files

➤ **App_Themes**

Stores the themes that are used to standardize and reuse formatting in your web application.

➤ **Bin**

Contains all the compiled .NET components (DLLs) that the ASP.NET web application uses. For example, if you develop a custom component for accessing a database

Access Specifiers in C#

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication3
{
    public class BaseClass
    {
        protected internal int prot_intValue = 0;
        public int publicValue = 12;
        internal int internalValue=100;
        private int privateValue=500;
        protected int protectedValue=600;
        void display()
        {
            //all are accessible
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            var baseObject = new BaseClass();
            //internalValue,publicValue,prot_intValue are accessible;
            //private and protected are not accessible
        }
    }
    class derivedsameassembly : BaseClass
    {
        void display()
        {
            //internalValue,protectedValue,publicValue are accessible;
            //prot_intValue is accessible;
            //private is not accessible
        }
    }
}

```

```

namespace oop
{
    class diffnamespace1
    {
        void display()
        {
            var baseObject = new ConsoleApplication3.BaseClass();
            baseObject.
            //internalValue,publicValue,prot_intValue are accessible;
            //protectedValue,private are not accessible
        }
    }
    class diffnamespace2:ConsoleApplication3.BaseClass
    {
        void display()
        {
            //internalValue,publicValue are accessible;
            //prot_intValue, protectedValue, are accessible;
            //private is not accessible
        }
    }
}

```

```
using System.Text;
using ConsoleApplication3;
namespace ConsoleApplication4
{
    class Program
    {
        static void Main(string[] args)
        {
            var baseObject = new BaseClass();
            baseObject.publicValue = 1000;
            //only publicValue is accessible

        }
    }
    class DerivedClass : BaseClass
    {
        void display()
        {

            var derivedObject = new DerivedClass();
            //prot_intValue,protectedValue,publicValue are accessible
            //private and internal are not accessible

        }
    }
}
```

Demo C# Program

Program to check if a number is even or odd

```
using System;
public class Exercise2
{
    public static void Main()
    {
        int num1, rem1;
        Console.Write("\n\n");
        Console.Write("Check whether a number is even or odd :\n");
        Console.Write("-----");
        Console.Write("\n\n");
        Console.Write("Input an integer : ");
        num1= Convert.ToInt32(Console.ReadLine());
        rem1 = num1 % 2;
        if (rem1 == 0)
            Console.WriteLine("{0} is an even integer.\n",num1);
        else
            Console.WriteLine("{0} is an odd integer.\n",num1);
    }
}
```

DOT NET

Server Controls

- Created and configured as *objects*.
- Run on the web server
- Automatically provide their own HTML output
- State is maintained and events are triggered

HTML server controls:

- Server-based equivalents for standard HTML elements.
- Useful when migrating ordinary HTML pages or classic ASP pages to ASP.NET, because they require the fewest changes.

Web controls:

- Similar to the HTML server controls, but they provide
 - richer object model
 - variety of properties for style and formatting details.
 - More events
- Web controls also feature some user interface elements that have no direct HTML equivalent, such as the GridView, Calendar, and validation controls.

HTML Server Controls

HTML server controls provide an object interface for standard HTML elements. They provide three key features:

- ***They generate their own interface***: You set properties in code, and the underlying HTML tag is created automatically when the page is rendered and sent to the client.
- ***They retain their state***: Because the Web is stateless, ordinary web pages need to do a lot of work to store information between requests. HTML server controls handle this task automatically. For example, if the user selects an item in a list box, that item remains selected the next time the page is generated. Or if your code changes the text in a button, the new text sticks the next time the page is posted back to the web server.
- ***They fire server-side events***: For example, buttons fire an event when clicked, text boxes fire an event when the text they contain is modified, and so on. Your code can respond to these events, just like ordinary controls in a Windows application. If a given event doesn't occur, the event handler won't be executed.

Control - Most Important Properties

- `HtmlAnchor` - `HRef`, `Target`
- `HtmlImage` - `Src`, `Alt`, `Width`, `Height`
- `HtmlInputCheckBox` - `Checked`
- `HtmlInputRadioButton` - `Checked`
- `HtmlInputText` - `Value`
- `HtmlSelect` - `Items` (collection)
- `HtmlGenericControl` - `InnerText` `InnerHtml`

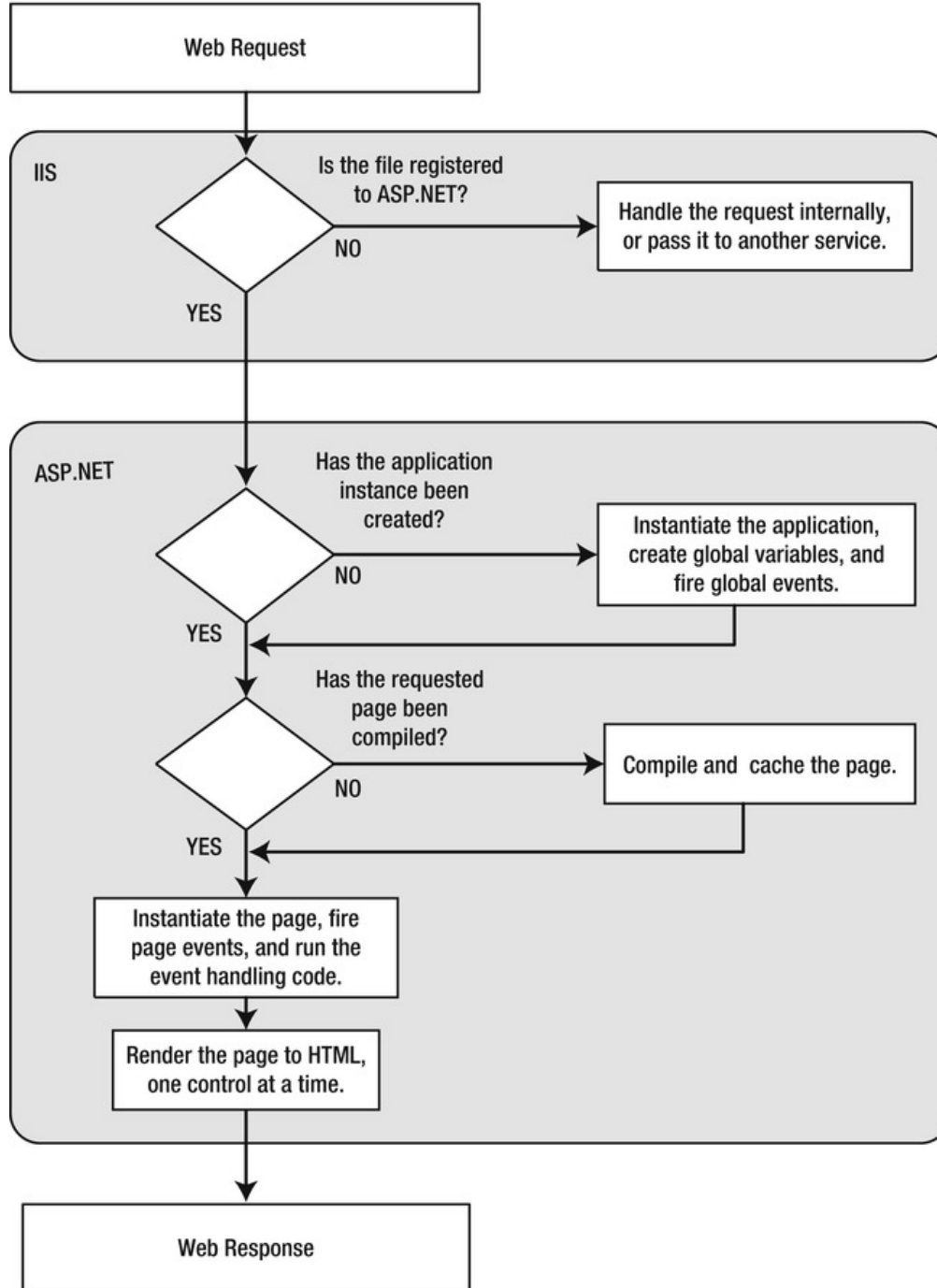
Sample Code

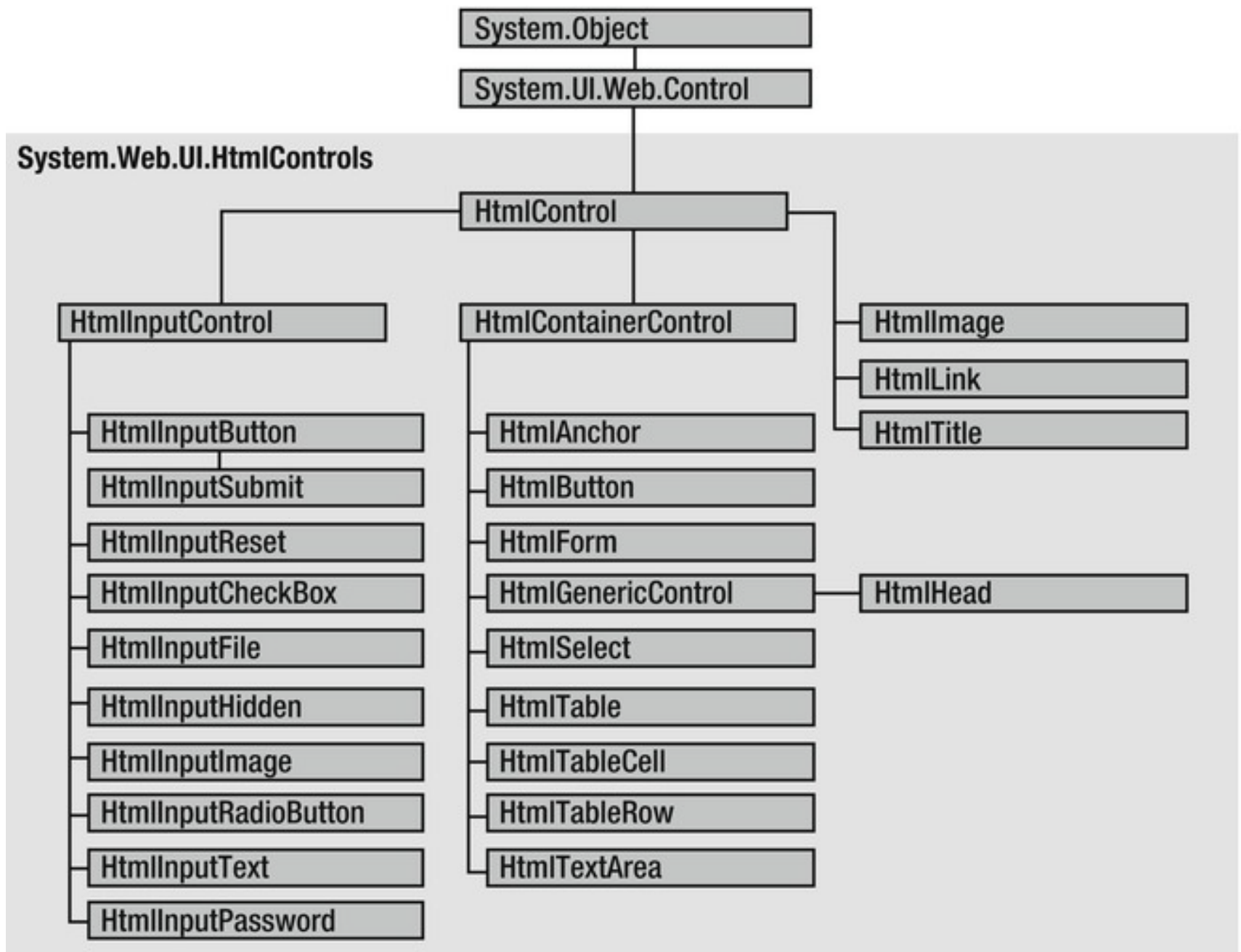
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
public partial class CurrencyConverter : System.Web.UI.Page
{
    protected void Convert_ServerClick(object sender, EventArgs e)
    {
        decimal USAmount = Decimal.Parse(US.Value);
        decimal euroAmount = USAmount * 0.85 M;
        Result.InnerText = USAmount.ToString() + " U.S. dollars = ";
        Result.InnerText += euroAmount.ToString() + " Euros.";
    }
}
```

- It starts with several *using* statements. This provides access to all the important namespaces. This is a typical first step in any code-behind file.
- The page class is defined with the partial keyword. That's because your class code is merged with another code file that you never see. This extra code, which ASP.NET generates automatically, defines all the server controls that are used on the page. This allows you to access them by name in your code.
- The page defines a single event handler. You'll notice that the event handler accepts two parameters (sender and e). It allows your code to identify the control that sent the event (through the sender parameter) and retrieve any other information that may be associated with the event (through the e parameter).
- The event handler is connected to the control event by using the OnServerClick attribute in the <input> tag for the button.

Behind the Scenes

1. The request for the page is sent to the web server. The request is sent to the built-in test server.
2. The web server determines that the .aspx file extension is registered with ASP.NET. If the file extension belonged to another service (as it would for .html files, for example), ASP.NET would never get involved.
3. If this is the first time a page in this application has been requested, ASP.NET automatically creates the application domain. It also compiles all the web page code for optimum performance, and caches the compiled files. If this task has already been performed, ASP.NET will reuse the compiled version of the page.
4. The compiled CurrencyConverter.aspx page acts like a miniature program. It starts firing events (most notably, the Page.Load event). However, you haven't created an event handler for that event, so no code runs. At this stage, everything is working together as a set of in-memory .NET objects.
5. When the code is finished, ASP.NET asks every control in the web page to render itself into the corresponding HTML markup.
6. The final page is sent to the user, and the application ends.





HTML Control Events

- **Event Controls That Provide It**
- ServerClick - HtmlAnchor, HtmlButton, HtmlInputButton, HtmlInputImage, HtmlInputReset
- ServerChange - HtmlInputText, HtmlInputCheckBox, HtmlInputRadioButton, HtmlSelect

- **HtmlControl Properties**
- **Attributes** - Provides a collection of all the attributes that are set in the control tag, and their values. Rather than reading or setting an attribute through the Attributes, it's better to use the corresponding property in the control class.
- **Controls** - Provides a collection of all the controls contained inside the current control. (Forexample, a <div> server control could contain an <input> server control.) Each object is provided as a generic System.Web.UI.Control object so that you may need to cast thereference to access control-specific properties.
- **Disabled** - Disables the control when set to true, thereby ensuring that the user cannot interact with it, and its events will not be fired.
- **EnableViewState** - Disables the automatic state management for this control when set to false. In this case, the control will be reset to the properties and formatting specified in the control tag every time the page is posted back. If this is set to true (the default), the control stores its state in a hidden input field on the page, thereby ensuring that any changes you make in code are remembered.
- **Page** - Provides a reference to the web page that contains this control as a System.Web.UI.Page object.
- **Parent** - Provides a reference to the control that contains this control. If the control is placed directly on the page (rather than inside another control), it will return a reference to the page object.
- **Style** - Provides a collection of CSS style properties that can be used to format the control.
- **TagName** - Indicates the name of the underlying HTML element (for example, img or div).
- **Visible** - Hides the control when set to false and will not be rendered to the final HTML page that is sent to the client.

WebForm1.aspx.cs

WebForm1.aspx* X

Client Objects & Events

(No Events)

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="WebForm1.aspx.cs" Inherits="WebApplication2.WebForm1" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
<script language="javascript" type="text/javascript">
    function Submit1_onclick() {
        document.getElementById("Text1").value = document.getElementById("Text1").attributes.length;
    }
</script>
</head>
<body>
<form id="form1" runat="server">
<div>

    <input id="Text1" type="text" size="100" class="t1" runat="server" />
    <input id="Submit1" type="submit"
        value="submit" onclick="return Submit1_onclick()" /><br />
    <br />
</div>
</form>
</body>
</html>
```

100 %

Design Split Source

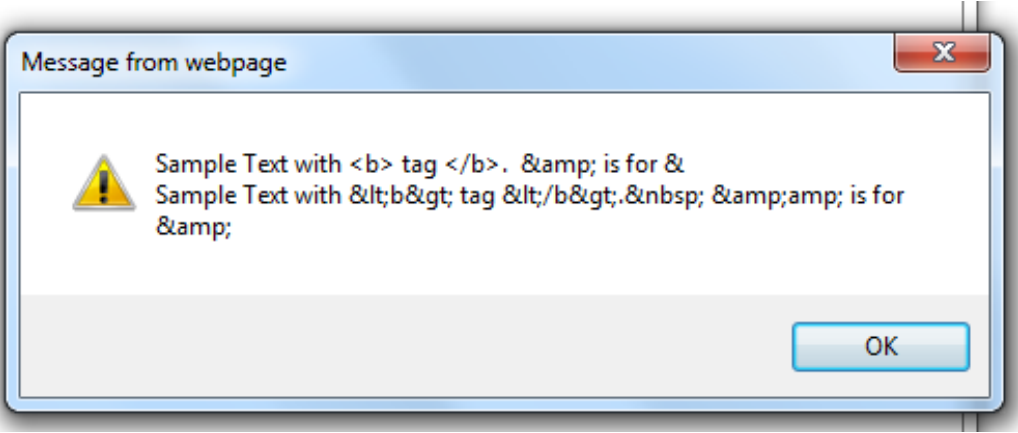
HtmlInputControl Properties

- **Type** - Provides the type of input control.
- **Value** - Returns the contents of the control as a string.

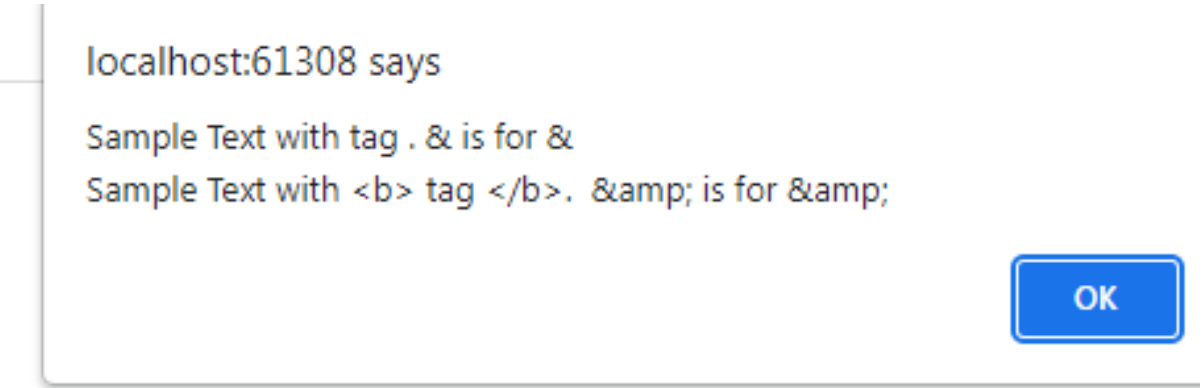
HtmlContainerControl Properties

- **InnerText** - The text content between the opening and closing tags of the control. Special characters will be automatically converted to HTML entities and displayed as text (for example, the less-than character (<) will be converted to < and will be displayed as < in the web page).
- **InnerHtml** - The HTML content between the opening and closing tags of the control. Special characters that are set through this property will not be converted to the equivalent HTML entities.

```
document.getElementById("div2").innerText = "Sample Text with <b> tag </b>. &amp; is for &";
minnerText = document.getElementById("div2").innerText;
minnerHtml = document.getElementById("div2").innerHTML;
alert(minnerText + "\n" + minnerHtml);
```



```
document.getElementById("div2").innerHTML = "Sample Text with <b> tag </b>. &amp; is for &";
```



```
$("#p1").html("Sample Text with <b> tag (html property);  
$("#p2").text("Sample Text with <b> tag (text property);
```

Sample Text with **tag (html property)**

Sample Text with tag (text property)

Page Properties

Property	Description
IsPostBack	This Boolean property indicates whether this is the first time the page is being run (false) or whether the page is being resubmitted in response to a control event, typically with stored view state information (true).
EnableViewState	When set to false, this overrides the EnableViewState property of the contained controls, thereby ensuring that no controls will maintain state information.
Application	This collection holds information that's shared between all users in your website
Session	This collection holds information for a single user, so it can be used in different pages.
Cache	This collection allows you to store objects that are time-consuming to create so they can be reused in other pages or for other clients.
Request	This refers to an HttpRequest object that contains information about the current web request.
Response	This refers to an HttpResponse object that represents the response ASP.NET will send to the user's browser.
Server	This refers to an HttpServerUtility object that allows you to perform a few miscellaneous tasks. For example, it allows you to encode text so that it's safe to place it in a URL or in the HTML markup of your page.
User	If the user has been authenticated, this property will be initialized with user information.

EnableViewState and ViewStateMode

- The ViewStateMode property of a page or a control has an effect only if the EnableViewState property is set to true.
- The default value of the ViewStateMode property for a page is Enabled. The default value of the ViewStateMode property for a Web server control in a page is Inherit.
- You can use the ViewStateMode property to enable view state for an individual control even if view state is disabled for the page.

- Disabled - will disable the viewstate for that page or control(i.e. if the page level property is disabled and control level property is enabled, view state will work for the control).
- Enabled - will enable the viewstate for that page or control(i.e. if the page level property is enabled and control level property is disabled, view state will not work for the control).
- Inherit - will inherit the page viewstate property and apply it to the control viewstate property.

- When EnableViewState of Page is set to True

EnableViewState-ViewStateMode	Status
True – Inherit	Content Preserved
True – Enabled	Content preserved
True - Disabled	Content is not preserved
False – Inherit	Content is not preserved
False – Enabled	Content is not preserved
False – Disabled	Content is not preserved

- When EnableViewState of Page is set to False, Content Preserved irrespective of control's EnableViewState / ViewStateMode

Sending the User to a New Page

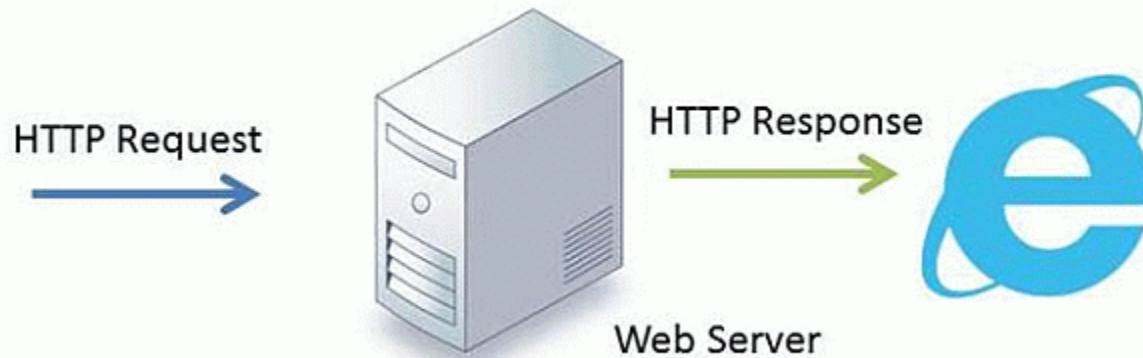
- Click ` here` to go to newpage.aspx
- `Response.Redirect("newpage.aspx");`
- `Server.Transfer("newpage.aspx");`

Response.Redirect()	Server.Transfer()
ASP.NET immediately stops processing the page and sends a redirect message back to the browser. When the browser receives the redirect message, it sends a request for the new page.	Doesn't involve the browser. Instead of sending a redirect message back to the browser, ASP.NET simply starts processing the new page as though the user had originally requested that page.
browser shows the new URL	browser will still show the original URL.
used to redirect a user to an external websites	used only on sites running on the same server
the previous page is removed from server memory and loads a new page in memory	previous page also exists in server memory



`Response.Redirect("newpage.aspx");`

`Response.Redirect` sends an HTTP request to the browser, then the browser sends that request to the web server, then the web server delivers a response to the web browser.



`Server.Transfer("newpage.aspx");`

`Server.Transfer` sends a request directly to the web server and the web server delivers the response to the browser.

Working with HTMLEncoding

```
protected void Button1_Click1(object sender, EventArgs e)
{
    Label1.Text = "This <text> is without Server.HtmlEncode <br>";
    Label1.Text += Server.HtmlEncode("This <text> is with Server.HtmlEncode");
    Label1.Text += "<br>";
    Label1.Text += Server.HtmlEncode("<b> tag is used to make the text ");
    Label1.Text += "<b>bold</b>";
}
```

This is without Server.HtmlEncode
This <text> is with Server.HtmlEncode
 tag is used to make the text **bold**

Server.HtmlEncode Demo

`$("#p1").html("Sample Text with tag (html property);`
`$("#p2").text("Sample Text with tag (text property);`

Sample Text with **tag (html property)**

Sample Text with tag (text property)

Context Object

- HttpContext object will hold information about the current http request.
- HttpContext object will be constructed newly for every request given to an ASP.Net application
- This holds current request specific information like Request, Response, Server, Session, Cache, User and etc.

```
using System;
public partial class UserRegister : System.Web.UI.Page
{
    protected void btn_Detail_Click(object sender, EventArgs e)
    {
        Context.Items.Add("Name", "Dev");
        Context.Items.Add("Email", "dev.mehta@gmail.com");
        Server.Transfer("UserDetail.aspx");
    }
}
```

```
using System;
public partial class UserDetail : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        try
        {
            Response.Write( Context.Items["Name"].ToString(), Context.Items["Email"].ToString());
        }
        catch (NullReferenceException ex)
        {
            Response.Write(ex.Message);
        }
    }
}
```

Using Application Events

- By using application events, code can be written that runs every time a request is received, no matter what page is being requested.
- Basic ASP.NET features such as session state and authentication use application events to plug into the ASP.NET processing pipeline.
- code-behind for a web form is not used for this. Instead, we need the help of another ingredient: the `global.asax` file.

global.asax File

- The global.asax file allows you to write code that responds to global application events. These events fire at various points during the lifetime of a web application, including when the application domain is first created

Table 5-11. *Basic Application Events*

Event-Handling Method	Description
Application_Start()	Occurs when the application starts, which is the first time it receives a request from any user. It doesn't occur on subsequent requests. This event is commonly used to create or cache some initial information that will be reused later.
Application_End()	Occurs when the application is shutting down, generally because the web server is being restarted. You can insert cleanup code here.
Application_BeginRequest()	Occurs with each request the application receives, just before the page code is executed.
Application_EndRequest()	Occurs with each request the application receives, just after the page code is executed.
Session_Start()	Occurs whenever a new user request is received and a session is started. Sessions are discussed in detail in Chapter 8.
Session_End()	Occurs when a session times out or is programmatically ended. This event is raised only if you are using in-process session-state storage (the InProc mode, not the StateServer or SQLServer modes).
Application_Error()	Occurs in response to an unhandled error. You can find more information about error handling in Chapter 7.

```
protected void Application_Start(object sender, EventArgs e)
{
    Application["appCtr"] = 0;
    Application["noOfUsers"] = 0;
}
protected void Session_Start(object sender, EventArgs e)
{
    Application.Lock();
    Application["noOfUsers"] = (int)Application["noOfUsers"] + 1;
    Application.Unlock();
}
protected void Session_End(object sender, EventArgs e)
{
    Application.Lock();

    Application["noOfUsers"] = (int)Application["noOfUsers"] - 1;
    Application.Unlock();
}
protected void Application_BeginRequest(object sender, EventArgs e)
{
    Response.Write(" Application BeginRequest method is triggered<hr/><br>");
    Application.Lock();
    Application["appCtr"] = (int)Application["appCtr"] + 1;
    Application.Unlock();
}
protected void Application_EndRequest(object sender, EventArgs e)
{
    Response.Write(" <hr/>This page was served at " + DateTime.Now.ToString() + "<br>");
    Response.Write("No of Visitors - " + Application["appCtr"] + "<br>");
    Response.Write("No of Users Currently Active - " + Application["noOfUsers"]);
}
```

Point to note

- The Session_End event doesn't fire when the browser is closed, it fires when the server hasn't gotten a request from the user in a specific time period (by default 20 minutes). That means that if you use Session_End to remove users, they will stay in the chat for 20 minutes after they have closed the browser.*

```
<sessionState timeout="1" mode="InProc"/>
```