# MS .NET, C# & ASP .NET

lecture
2011/2012

# Literature

# MS .NET – what is it?

- Managed environment placed between user program and operating system
- Pros:
  - safety,
  - many languages,
  - easier management,
  - simplification of application design/coding
- Cons:
  - speed,
  - resource consumption,
  - problems on non-MS platforms (Mono, Grasshopper)

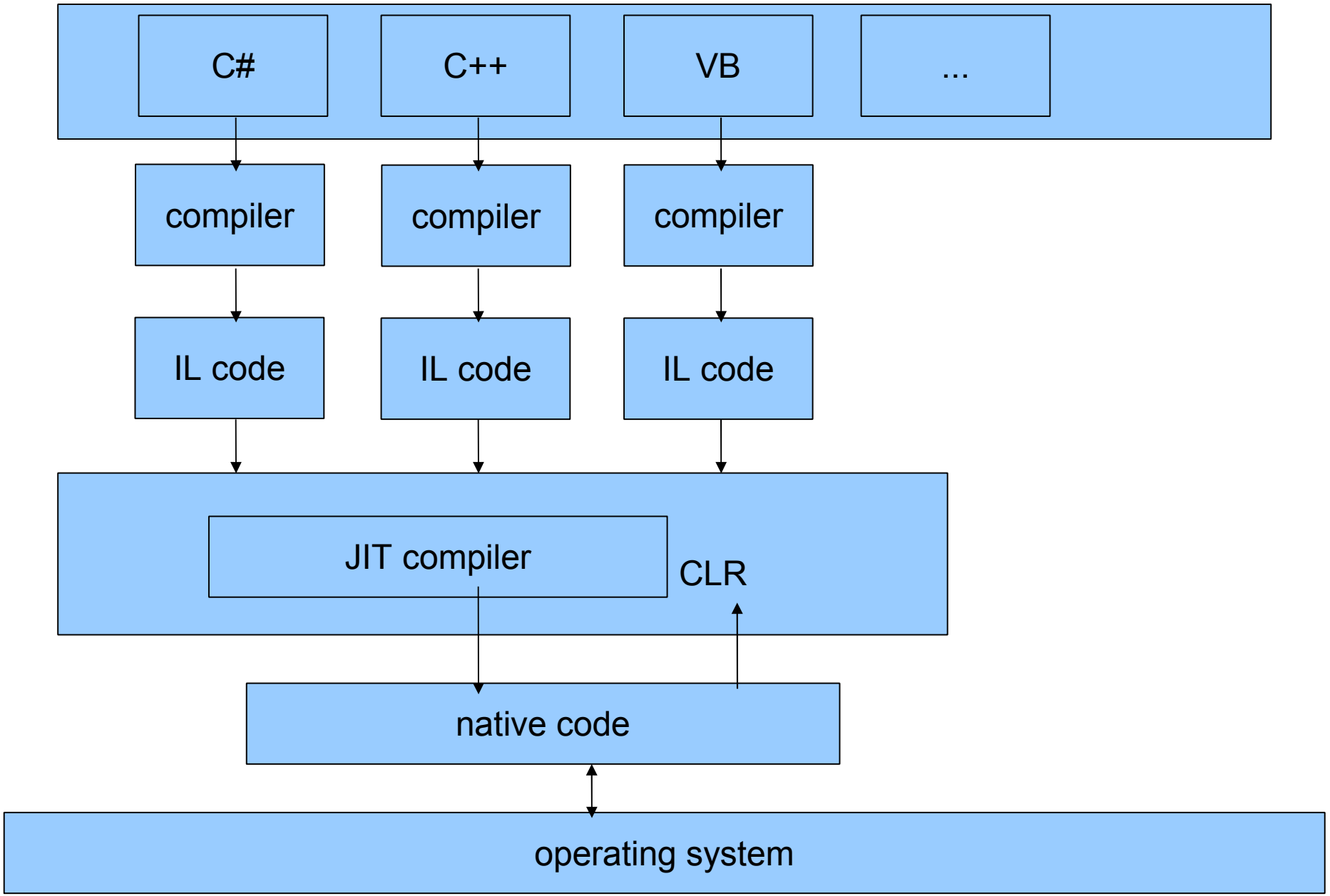
# MS .NET components

- Common Language Runtime
- .NET Framework class library

# C# - its position

- C# is a language created for effective use of .NET features
- Uses concepts similar to C++ and Java

# .NET concept


C#
C++
VB
...
compiler
compiler
compiler
IL code
IL code
IL code
JIT compiler
CLR
native code
operating system

# Safety

- Memory management (e.g., accessing unallocated memory)
- Types safety (e.g., conversion to incompatible types)
- Variable initialisation, stack management, function pointers management
- Verification of MSIL code (memory access, types compatibility)
- Permission control for code, not user

# Multilingual

- C#, C++ (C++/CLI), VB, F#
- Elements standardising language constructs: Common Language Specification, Common Type System
- Translation into MS Intermediate Language
- Ability to mix modules written in different languages

# Management

- Different versions of components may co-exist (solves *DLL hell* problem)
- Built-in mechanisms for module identification
- Metadata and self-describing code – modules contain information about required components, CLR checks if everything needed is available

# Simpler application creation

- Many useful predefined objects
- Garbage collector
- Better exception handling
- Every class derives from *Object*
- Run time type information
- New instructions and constructs

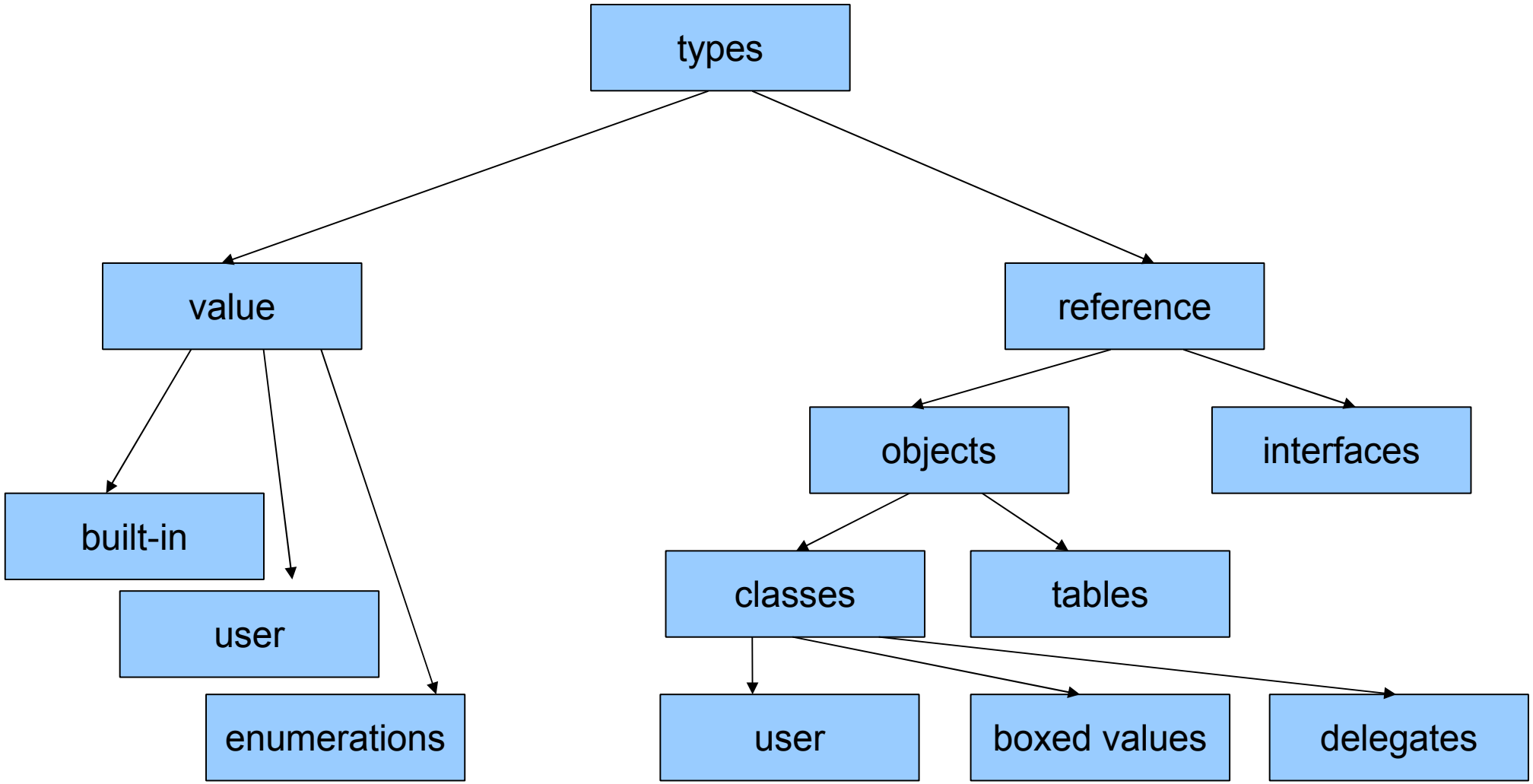
# Speed and resource utilisation

- JIT versus code generation during installation
- CLR module
- Garbage collection
- Boxing and unboxing

# Other platforms

- Problems with migration to other platforms...
- Possible solutions:
  - Native implementation of .NET on other platforms (Mono)
  - Intermediate layer, e.g., translating IL code into Java bytecode
  - Problems of WWW servers (e.g., mod_mono for Apache)

# Common Type System



# Values vs. references

- Small
- Allocated on the stack
- Copied when passed to a function
- Compared bitwise
- Defined using keyword `struct`
- Derived from `System.ValueType`

- Can be large
- Allocated on the heap
- Function receives a pointer
- Compared by addresses
- Defined using keyword `class`
- Derived from `System.Object`

# Built-in value types

- Logical values
- Alphanumeric characters (Unicode)
- Integer values (signed and unsigned)
- Floating point values (32 i 64bit)
- Hardware-dependent

# Recommended naming convention

- Not all languages are case-sensitive
- Pascal convention should be used, variable names excepted (function parameters, class fields, local variables), where Camel convention should be used
- Hungarian notation should not be used
- DLL libraries corresponding to modules should be named Company.Component.dll
- Namespaces should be named Company.Product.ProductElement

# Recommended naming convention

- Do not prefix class names (exception: interfaces)
- Append base class name to the derived class name

# C# conventions

- No pointers
- Access to fields and methods always using . (dot)
- Namespaces in place of header files
- No global elements
- `Main` function is a static method of a class

# Modification of values passed to methods – `ref` & `out`

- Allow modification of passed variables
- `out` allows to pass uninitialised varaible

# Modification of passed values

```
using System;
public class MyClass
{
   public static void TestRef(ref char i)
   {
      i = 'b';
   }

   public static void TestNoRef(char i)
   {
      i = 'c';
   }

   public static void Main()
   {

      char i = 'a';     // initialisation
      TestRef(ref i);   // pass as ref
      TestNoRef(i);
   }
}
```

# Modification of passed values

```
using System;
public class MyClass
{
    public static int TestOut(out char i)
    {
        i = 'b';
        return -1;
    }

    public static void Main()
    {
        char i; //no need to initialise
        Console.WriteLine(TestOut(out i));
        Console.WriteLine(i);
    }
}
```

# User value types – `struct`

- Passed by value
- Cannot take part in inheritance
- Cannot have a default constructor
- Fields initialised to 0 (only if *not* created using `new`)

Created as

```
type_name object_name;
```

or

```
type_name object_name = new
 type_name(par1, par2, ..., parN);
```

# Struct – example

```
struct Point
{
  public Point(int x, int y)
  {
    this.x = x;
    this.y = y;
  }

  public int x;
  public int y;
}

Point pt1; //inicjalised to zeros
Point pt2 = new Point(12, 28);
```

# Enumeration – `enum`

- Based on integers (`int` by default, other can be used – except `char`)
- By default starts with 0 and is incremented by one for every element
- Can be used as bit flags
- Derived from `System.Enum`

# enum – example 1

```
enum LineStyle
{
   solid,
   dotted,
   dashed
}
enum LineStyle : byte
{
   solid = 0,
   dotted = 8,
   dashed = 16
}
enum LineStyle
{
   solid = 0,
   dotted = solid + 8,
   dashed = dotted * 2
}
```

# `enum` – example 2

```
[Flags]
enum LineStyle
{
   thick = 0x00000001,
   dotted = 0x00000002,
   dashed = 0x00000004
}
```

# User classes – `class`

- Allows inheritance
- No multiple inheritance, but a class may implement many interfaces
- Class definition may be split among many files (`partial`)

Objects may be created only as

```
type_name object_name = new
type_name(par1, par2, ..., parN);
```

# Access modifiers

- Class level:
  - `public`
  - `internal`
- Member level
  - `public`
  - `internal protected`
  - `internal`
  - `protected`
  - `private`
- Class access further restricts member access

# Constructors

- Instance – called when the object is created
- Static – used to initialise the class (not the object) before any object is created and before static memebers are referenced

# Instance constructors

- If no default constructor is provided, compiler generates generic implementation. Fields are initialised to 0
- Base class constructor may be called using keyword `base`
- Another constructor in the same class can be called using keyword `this`
- Field initialisation can be done in constructor or in the field declaration
- Procate – prohibits creation of objects (unless public constructor is available)

# Instance constructor – example

```
public Cylinder(double r, double h)
: base(r, h)
{
}

public Point(): this(0,20)
{
}


public class MyClass
{
  public int counter = 100;
}
```

# Static constructors

- No access modifiers or arguments
- Cannot be called explicitly
- User has no influence on when the constructor will be called

# Static constructors – example

```
class MyClass
{
    // Static constructor:
    static MyClass()
    {
        Console.WriteLine("Static constructor");
    }

    public static void MyMethod()
    {
        Console.WriteLine("MyMethod invoked.");
    }
}
```

# Destructors

- Work on different principle than in C++, called during garbage collection
- Equivalent to `Finalize` method
- `Finalize` method can be written only in languages without destructors
- Destructor always calls `Finalize` in the base class
- User cannot influence the moment or order of calling `Finalize`, it may be not called at all
- In order to 'manually release resources' `Dispose` method should be used (Idisposable interface)

# Destructors

```
~MyClass()
{
    // Perform some cleanup operations here.
}

protected override void Finalize()
{
    try
    {
        // Perform some cleanup operations here.
    }
    finally
    {
        base.Finalize();
    }
}
```

# Static elements – `static`

- Methods (including constructors)
- Fields

# Constants

- `const` – only built-in types, written as literals – value must be known during compilation. Set in declaration
- `readonly` – value may be set in constructor or in declaration and cannot be changed after that. In different constructors different values may be used.

# Constants – example

```
class MyClass
{
    public int x;
    public readonly int y = 25; // declaration
    public readonly int z;
    public const int v = 5;

    public MyClass()
    {
        z = 24;   // constructor
    }

    public MyClass(int p1, int p2, int p3)
    {
        x = p1;
        y = p2;
        z = p3;
    }
}
```

# Inheritance (1)

- No multiple inheritance, but calss may implement multiple interfaces
- Virtual methods in derived classes overriding methods in the base class have to use `override` or `new` keyword
- Non-virtual methods in derived classes overriding methods in the base class must use `new` keyword
- Abstract classes can be created by `abstract` keyword (abstract class does not have to have abstract methods, but abstract methods can be placed only in an abstract class)

# Inheritance (2)

- `sealed` keyword prohibits creating derived classes
- `sealed` keyword prohibits overriding a method
- Base class methods may be called using `base` keyword

# Inheritance – example 1

```
class Square
{
    public double x;
    public Square(double x)
    {
        this.x = x;
    }
    public virtual double Area()
    {
        return x*x;
    }
}

class Cube: Square
{
    public Cube(double x): base(x){}
    public override double Area()
    {
        return (6*(base.Area()));
    }
}
```

# Inheritance – example 2

```
class A
{
    public void F() {}
    public virtual void G() {}
}
class B: A
{
    new public void F() {}
    public override void G() {}
}
```

# Inheritance – example 3

```
using System;
class A
{
   public virtual void F() { Console.WriteLine("A.F"); }
}
class B: A
{
   public override void F() { Console.WriteLine("B.F"); }
}
class C: B
{
   new public virtual void F() { Console.WriteLine("C.F"); }
}
class D: C
{
   public override void F() { Console.WriteLine("D.F"); }
}
class Test
{
   static void Main() {
      D d = new D();
      A a = d;
      B b = d;
      C c = d;
      a.F();          B.F
      b.F();          B.F
      c.F();          D.F
      d.F();          D.F
   }
}
```

# Inheritance – example 4

```
abstract class A
{
   public abstract void F();
}
abstract class B: A
{
   public void G() {}
}
class C: B
{
   public override void F() {
      // actual implementation of F
   }
}
```

# Inheritance – example 5

```
using System;
class A
{
    public virtual void F() {}
    public virtual void G() {}
}
class B: A
{
    sealed override public void F() {
        Console.WriteLine("B.F");
    }
    override public void G() {
        Console.WriteLine("B.G");
    }
}
class C: B
{
    override public void G() {
        Console.WriteLine("C.G");
    }
}
```

# Inheritance – example 6

```
sealed class MyClass
{
   MyClass(){}
}

//error
class MyNewClass : MyClass
{
}
```

# Interfaces

- Similar to abstract class with all memebers `abstract`
- A class may implement more than one interface
- Structures may also implement interfaces
- Interfaces may derive from other interfaces
- Class implementing interface must define all methods of this interface

# Interfaces – example 1

```
public class DiagramObject
{
   public DiagramObject(){}
}

interface IScalable
{
   void ScaleX(float factor);
   void ScaleY(float factor);
}

public class TextObject: DiagramObject, IScalable
{
   public void ScaleX(float factor)
   {

   }
   public void ScaleY(float factor)
   {
   }
}
```

# Interfaces – example 2

```
TextObject text = new TextObject(“bla”);
text.ScaleX(0.5F)
IScalable scalable = (IScalable) text;
scalable.ScaleY(0.5F)
```

# Interfaces – example 3

```
interface IList
{
   int Count { get; set; }
}
interface ICounter
{
   void Count(int i);
}
interface IListCounter: IList, ICounter {}
class C
{
   void Test(IListCounter x) {
      x.Count(1);                    // ?
      x.Count = 1;                 // ?
      ((IList)x).Count = 1;        // ?
      ((ICounter)x).Count(1);        // ?
   }
}
```

# Explicit interface implementation

- When two interfaces declare functions with the same name, yet implementations in a class that implements these interfaces must be separate
- In case of hiding interface implementation
- Cannot have access modifiers

# Explicit interface implementation – example 1

```
interface IFoo
{
   void Execute();
}

interface IBar
{
   void Execute();
}

class Tester : IFoo, IBar
{
   public void Execute(){}
}                          void IFoo.Execute()
                           {
                              //IFoo code
                           }
                           void IBar.Execute()
                           {
                              //IBar code
                           }
```

# Explicit interface implementation – example 2

```
Tester tester = new Tester();

tester.Execute() //error

IFoo iFoo = (IFoo)tester;
iFoo.Execute();

IBar iBar = (IBar)tester;
iBar.Execute();
```

# Type information

- `typeof` operator returns object's type
- `Object.GetType()function` returns object's type
- `is` operator checks whether an object can be converted to a type
- `as` operator performs such conversion, if it is not possible, returns `null`

# Type information – example 1

```
using System;
using System.Reflection;

public class MyClass
{
    public int intI;
    public void MyMeth()
    {
    }

    public static void Main()
    {
        Type t1 = typeof(MyClass);

        MyClass mc = new MyClass();
        Type t2 = mc.GetType();
    }
}
```

# Type information – example 2

```
class Class1
{
}
class Class2
{
}

public class IsTest
{
    public static void Test (object o)
    {
        Class1 a;
        Class2 b;
        if (o is Class1)
        {
            Console.WriteLine ("o is Class1 type");
            a = (Class1)o;
            // do something with a
        }else if (o is Class2)
        {
            Console.WriteLine ("o is Class2 type");
            b = (Class2)o;
            // do something with b
        }else
        {
            Console.WriteLine ("o is other type.");
        }
    }
}
```

# Type information – example 3

```
using System;
class MyClass1
{
}
class MyClass2
{
}

public class IsTest
{
    public static void Main()
    {
        object [] myObjects = new object[6];
        myObjects[0] = new MyClass1();
        myObjects[1] = new MyClass2();
        myObjects[2] = "hello";
        myObjects[3] = 123;
        myObjects[4] = 123.4;
        myObjects[5] = null;

        for (int i=0; i<myObjects.Length; ++i)
        {
            string s = myObjects[i] as string;
            Console.Write ("{0}:", i);
            if (s != null)
                Console.WriteLine ( "'" + s + "'" );
            else
                Console.WriteLine ( "is not text" );
        }
    }
}
```

# Control statements

- `if`, `while`, `do...while` and `for` statements require `bool` expressions
- In `switch` statement every block must end with `break` or `goto case`
- `foreach` statement can be used to iterate through elements. Should not be used for modifications. If iterating variable is a value type, it is read-only.

# `foreach` – example 1

```
int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9};
foreach (int element in numbers)
{
    System.Console.WriteLine(element);
}
```

# `foreach` – example 2

```
foreach (DiagramObject d in dArray)
{
   if (d is IScalable)
   {
      IScalable scalable = (IScalable)d;
      scalable.ScaleX(0.5F);
   }
}
```

# Error handling

- Basic mechanism for error handling are exceptions
- All exception classes derive from `System.Exception`
- Multiple `catch` sections are allowed, `catch` Wit no parameters catches all exceptions
- Caught exception may be further thrown using `throw`, including the ona caught in a section with no parameter
- `finally` section allows executing sttements immediately after `try` block, including the situation when an exception is thrown

# Error handling – example 1

```
class MyClass
{
   public static void Main()
   {
      MyClass x = new MyClass();
      try
      {
         string s = null;
         x.MyFn(s);
      }

      // Narrow:
      catch (ArgumentNullException e)
      {
      }

      // Broader:
      catch (Exception e)
      {
      }

   }

   public void MyFn(string s)
   {
      if (s == null)
         throw new ArgumentNullException();
   }
}
```

# Error handling – example 2

```
public static void Main()
{
    int x;
    try
    {
        x = 123;    //    wrong
        // ...
    }
    catch
    {
        // ...
    }
    Console.Write(x);    // error
}
```

# Error handling – example 3

```
object o2 = null;
try
{
    int i2 = (int) o2;
}
catch (InvalidCastException e)
{
    // some code...
    throw (e);
}
catch
{
    // some code...
    throw;
}
```

# Error handling – example 4

```
public class EHClass
{
    public static void Main ()
    {
        try
        {
            // some code...
            throw new NullReferenceException();
        }
        finally
        {
            // always executed
        }
        catch(NullReferenceException e)
        {
            // narrow
        }
        catch
        {
            // broad
        }

    }
}
```

# Operators (1)

- Built-in operators for `int`, `uint`, `long`, `ulong`, `float`, `double` and `decimal`
- Operators can be overloaded. Overloadable are:
  unary:
  `+  -  !  ~  ++  --  true  false`
  binary:
  `+  -  *  /  %  &  |  ^  <<  >>`
  relations (in pairs):
  `==  !=  >  <  >=  <=`
- Cannot be overloaded:
  `&&  ||` (but use `&  |  true` and `false`)
  `[]` (but indexers can be defined)
  `()` (but conversion operators can be defined)
  `+= -= *= /= %= &= |= ^= <<= >>=` (but use overloadable operators)
  `= . ?: -> new is as sizeof typeof`

# Operators (2)

- Operator must be a public, static method
- At least one argument must be of the same type as the class in which it is defined

# Operators – example

```
public class IntVector
{
   public IntVector(int length) {}
   public int Length {}
   public int this[int index] {}
   public static IntVector operator ++(IntVector iv) {
      for (int i = 0; i < iv.Length; i++)
         iv[i] = iv[i] + 1;
      return iv;
   }
}
class Test
{
   static void Main() {
      IntVector iv1 = new IntVector(4);  // 4 x 0
      IntVector iv2;
      iv2 = iv1++;   // ???
      iv2 = ++iv1;   // ???
   }
}
```

# Operators – example

```
public class IntVector
{
   public IntVector(int length) {}
   public int Length {}
   public int this[int index] {}
   public static IntVector operator ++(IntVector iv) {
      IntVector temp = new IntVector(iv.Length);
      for (int i = 0; i < iv.Length; i++)
         temp[i] = iv[i] + 1;
      return temp;
   }
}
class Test
{
   static void Main() {
      IntVector iv1 = new IntVector(4);  // 4 x 0
      IntVector iv2;
      iv2 = iv1++;   // iv2 4 x 0, iv1 4 x 1
      iv2 = ++iv1;   // iv2 4 x 2, iv1 4 x 2
   }
}
```

# Conversions

- Implicit – for values: no loss of range (but possible loss of precision). Also to the base class and implemented interface.
- Explicit – possible loss of range
- User:
  - Public static method
  - Parameter or return value of the same type as the type in which it is declared
  - In order to determine whether the conversion is to be explicit or implicit, `explicit` and `implicit` keywords are used

# Conversions – example

```
using System;
public struct Digit
{
    byte value;
    public Digit(byte value)
    {
        if (value < 0 || value > 9)
          throw new ArgumentException();
        this.value = value;
    }
    public static implicit operator byte(Digit d)
    {
        return d.value;
    }
    public static explicit operator Digit(byte b)
    {
        return new Digit(b);
    }
}
```

# Arrays

- Arrays can be one-dimensional, multi-dimensional, jagged. The placement in memory is not determined
- In case of arrays of reference types, objects are not created during creation of the array
- Inherit from System.Array, which gives methods such as `Sort`, `IndexOf`, `LastIndexOf`, `BinarySearch` i `Reverse`
- Conversion from one table to another is possible when bot have the same number of dimensions, both are arrays of reference types and there are appropriate conversions between these types

# Arrays – example

```
int[] store = new int[50];
string[] names = new string[50];

int[][] a = new int[100][];
for (int i = 0; i < 100; i++) a[i] = new int[5];

int[,] b = new int[100, 5];
```

# Strings

- `System.String`
- Is immutable
- Supports many operations, e.g., `Compare`, `StartsWith`, `EndsWith`, `IndexOf`, `LastIndexOf`, `Concat`, `Insert`, `Remove`, `Replace`, `Split`, `Substring`, `ToLower`, `ToUpper` etc.
- Contains always Unicode characters, methods are provided for conversions from/to other encodings
- Can be formatted using static `Format` method or class `StringBuilder`

# Strings – `Format` method

- `public static string Format(string format, params object[] args);`

`format` argument contains formatting elements:
`{index[,alignment][:formatString]}`
where:
`index` – index of object to format
`alignment` – optional field indicating minimal width, negative for left align, positive for right align
`formatString` – optional argument (for built-in types or types implementing `IFormattable`)

# Strings – `formatString` for numbers

- syntax `Axx` where A can be:
  C - currency
  D - integer
  E - exponential
  F – fixed point
  G - optimal
  N – with thousand separation
  P - percent
  R – ensuring ability to convert back
  X - hexadecimal
  Small letters are also acceptable

  xx – precision, meaning depends on A

# Strings – example 1

```
String s = String.Format("Brad's dog has {0,-8:G} fleas.", 42);
```

Brad's dog has 42_____ fleas.

```
String.Format(
"The identity of {0} has been accessed {1} times.",
                                    Identity.Name,
                                    nAccesses);
```

# Strings - `StringBuilder`

- Ability to construct a string by modification of an exisiting one (is mutable)

# StringBuilder - example

```
string s = “I will not buy this record.”;
char[] separators = new char[]{' '};
StringBuilder sb = new StringBuilder();
int number = 1;
foreach (string sub in s.Split(separators))
{
   sb.AppendFormat(“{0}: {1} “, number++, sub);
}
Console.WriteLine(“{0}”, sb);
```

1: I 2: will 3: not 4: buy 5: this 6: record.

# Properties

- Visible externally as fields, in fact are methods. Usually used for 'intelligent' access to a private field
- May allow reading, writing o both
- May be virtual anf abstract
- In derived classes the whole property is overridden (it is not possible to override only the reading part or only the writing part)
- May be static

# Properties – example 1

```
public abstrat class DrawingObject
{
   public abstract string Name
   {
      get;
   }
}
class Circle : DrawingObject
{
   string name = "Circle";
   public override string Name
   {
      get
      {
         return (name);
      }
   }
}
class Test
{
   public static void Main()
   {
      DrawingObject d = new Circle();
      Console.WriteLine("Name: {0}", d.Name);
   }
}
```

# Properties – example 2

```
public class BaseClass
{
    private string name;
    public string Name{
        get{
            return name;
        }
        set{
            name = value;
        }
    }
}
public class DerivedClass : BaseClass
{
    private string name;
    public new string Name{
        get{
            return name;
        }
        set{
            name = value;
        }
    }
}
```

# Indexers

- Allow accessing the object as an array
- May simulate multidimensional arrays
- Index can be expressed using various types (eg., strings)
- Work similarily to properties
- Donot have a separate keyword – are declared using `this` keyword

# Indexers - example

```
class Grid
{
    const int NumRows = 26;
    const int NumCols = 10;
    int[,] cells = new int[NumRows, NumCols];
    public int this[char c, int col] {
        get {
            c = Char.ToUpper(c);
            if (c < 'A' || c > 'Z') {
                throw new ArgumentException();
            }
            if (col < 0 || col >= NumCols) {
                throw new IndexOutOfRangeException();
            }
            return cells[c - 'A', col];
        }
        set {
            c = Char.ToUpper(c);
            if (c < 'A' || c > 'Z') {
                throw new ArgumentException();
            }
            if (col < 0 || col >= NumCols) {
                throw new IndexOutOfRangeException();
            }
            cells[c - 'A', col] = value;
        }
    }
}
```

# Attributes 1

- Aloow passing additional information, eg., for commenting code or modification of keyword behaviour
- Can be extracted during runtime
- Are placed in squared parentheses before the target declaration
- Attribute arguments can be recognised by position or name
- Many attributes may be specified, they are placed in one parentheses pair and separated with comma, or in separate pairs

# Attribute 2

- It is possible to explicitly state the attribute target:

```
[SomeAttr] int Method1( string s ) // method
[method: SomeAttr] int Method1( string s ) //
method
[return: SomeAttr] int Method1( string s ) //
return value
```

- Available modifiers: `assembly`, `module`, `class`, `struct`, `interface`, `enum`, `delegate`, `method`, `parameter`, `field`, `property – indexer`, `property – get`, `property – set`, `event – field`, `event – property`, `event – add`, `event – remove`
- User attributes can be created by deriving from `System.Attribute`

# Attribute 3

- User attribute class must specify the possible usage target using `AttributeUsage` attribute and `AttributeTargets` enum, having following values: `Assembly`, `Module`, `Class`, `Struct`, `Enum`, `Constructor`, `Method`, `Property`, `Field`, `Event`, `Interface`, `Parameter`, `Return`, `Delegate`, `All`, `ClassMemebers`
- `AllowMultiple` parameter indicates the attribute can be applied more than once to the same target

# Attributes - example

```
using System;
[AttributeUsage(AttributeTargets.Class|AttributeTargets.Struct,
   AllowMultiple=true)]
public class Author : Attribute
{
   public Author(string name)
   {
      this.name = name; version = 1.0;
   }
   public double version;
   string name;
   public string GetName()
   {
      return name;
   }
}

[Author("H. Ackerman")]
class FirstClass
{
   /*...*/
}

class SecondClass  // no Author attribute
{
   /*...*/
}

[Author("H. Ackerman"), Author("M. Knott", version=1.1)]
class Steerage
{
   /*...*/
}
```

# Attributes – example

```
class AuthorInfo
{
   public static void Main()
   {
      PrintAuthorInfo(typeof(FirstClass));
      PrintAuthorInfo(typeof(SecondClass));
      PrintAuthorInfo(typeof(Steerage));
   }
   public static void PrintAuthorInfo(Type t)
   {
      Console.WriteLine("Author information for {0}", t);
      Attribute[] attrs = Attribute.GetCustomAttributes(t);
      foreach(Attribute attr in attrs)
      {
         if (attr is Author)
         {
            Author a = (Author)attr;
            Console.WriteLine("   {0}, version {1:f}",
                a.GetName(), a.version);
         }
      }
   }
}
```

# Delegates 1

- Substitute for function pointers
- Derived from System.Delegate
- May point to:
  - Static method
  - Object and non-static method
  - Other delegate
- May have access modifiers
- May have `new` keyword
- May point to a method iff it has the same number of parameters, the parameters are of the same types, in the same order, with the same modifiers and the return value is of the same type

# Delegates 2

- Delegates can be combined using `+`, `+=`, `-`, `-=` operators
- Calling a single delegate is identical to calling a method. When calling a chain of delegates following rules are obeyed:
  - Delegates are called in sequence
  - Every delegate receives the same parameters, but if `ref` or `out` parameters are present, changes caused by one method are visible in the next one
  - Return value is the return value of the last delegate
  - Unhandled exception on one method prohibits calling the following delegates

# Delegates – example 1

```
delegate int D1(int i, double d);
class A
{
    public static int M1(int a, double b) {...}
}
class B
{
    delegate int D2(int c, double d);
    public static int M1(int f, double g) {...}
    public static void M2(int k, double l) {...}
    public static int M3(int g) {...}
    public static void M4(int g) {...}

     void test()
     {
          D1 del1;
          D2 del2;
          del1 = A.M1;
          del2 = A.M1;
          del1 = M1;
          del2 = M1;
          del1 = M2;  //error
          int x = del2(2, 6);

     }
}
```

# Delegates – example 2

```
delegate void D(int x);
class C
{
   public static void M1(int i) {...}
   public static void M2(int i) {...}
}
class Test
{
   static void Main() {
      D cd1 = new D(C.M1); // M1
      D cd2 = new D(C.M2); // M2
      D cd3 = cd1 + cd2;   // M1 + M2
      D cd4 = cd3 + cd1;   // M1 + M2 + M1
      D cd5 = cd4 + cd3;   // M1 + M2 + M1 + M1 + M2
   }
}
```

# Delegates – example 3

```
delegate void D(int x);
class C
{
   public static void M1(int i) {...}
   public void M2(int i) {...}
}
class Test
{
   static void Main() {
      D cd1 = new D(C.M1);  // static method
      Test t = new C();
      D cd2 = new D(t.M2);  // object method
      D cd3 = new D(cd2);   // delegate
   }
}
```

# Delegates – example 4

```
delegate void D(int x);
class C{
   public static void M1(int i) {/*...*/}
   public static void M2(int i) {/*...*/}
   public void M3(int i) {/*...*/}
}
class Test{
   static void Main() {
      D cd1 = new D(C.M1);
      cd1(-1);                  // M1
      D cd2 = new D(C.M2);
      cd2(-2);                  // M2
      D cd3 = cd1 + cd2;
      cd3(10);                  // M1 and M2
      cd3 += cd1;
      cd3(20);                  // M1, M2 and M1
      C c = new C();
      D cd4 = new D(c.M3);
      cd3 += cd4;
      cd3(30);                  // M1, M2, M1 and M3
      cd3 -= cd1;               // removes the last M1
      cd3(40);                  // M1, M2 and M3
      cd3 -= cd4;
      cd3(50);                  // M1 and M2
}
```

# Delegates – example 5

```
public delegate void Del(string message);

public static void DelegateMethod(string message)
{
    System.Console.WriteLine(message);
}

Del handler = DelegateMethod;

handler("Hello World");


public void MethodWithCallback(int param1,
   int param2, Del callback)
{
    callback("The number is: " +
     (param1 + param2).ToString());
}

MethodWithCallback(1, 2, handler);
```

# Events 1

- Indicate something happened
- Declared using delegates
- Steps:
  - Declaration of a delegate. It defines arguments passed to the method handling event. Various arguments are possible, but .NET specification requires `Object sender` and `EventArgs e`. A built-in delegate `EventHandler` can be used
  - Event declaration – similar to field declaration of delegate type, but preceded with word `event`
  - Event calling – just like calling a delegate. Must ensure event is not `null`. Calling is only possible from the class declaring the event.

# Events 2

- Attaching to event: from outside event looks like a field, but the only possible oppperations are $+=$ i $-=$

# Events – example 1

```
public delegate void ChangedEventHandler(object sender, EventArgs e);
public class ListWithChangedEvent: ArrayList
{
   public event ChangedEventHandler Changed;
   protected virtual void OnChanged(EventArgs e)
   {
      if (Changed != null)
         Changed(this, e);
   }
   public override int Add(object value)
   {
      int i = base.Add(value);
      OnChanged(EventArgs.Empty);
      return i;
   }
}
class EventListener
{
   private ListWithChangedEvent List;
   public EventListener(ListWithChangedEvent list)
   {
      List = list;
      List.Changed += new ChangedEventHandler(ListChanged);
   }
   private void ListChanged(object sender, EventArgs e)
   {
      Console.WriteLine("This is called when the event fires.");
   }
   public void Detach()
   {
      List.Changed -= new ChangedEventHandler(ListChanged);
      List = null;
   }
}
```

# Events – example 1

```
    class Test
    {
        // Test the ListWithChangedEvent class.
        public static void Main()
        {
        // Create a new list.
        ListWithChangedEvent list = new ListWithChangedEvent();

        // Create a class that listens to the list's change event.
        EventListener listener = new EventListener(list);

        // Add and remove items from the list.
        list.Add("item 1");
        list.Clear();
        listener.Detach();
        }
    }
}
```

# Events – example 2

```
public class ListWithChangedEvent: ArrayList
{
   public event EventHandler Changed;
   protected virtual void OnChanged(EventArgs e)
   {
      if (Changed != null)
         Changed(this,e);
   }
   public override int Add(object value)
   {
      int i = base.Add(value);
      OnChanged(EventArgs.Empty);
      return i;
   }
}
class EventListener
{
   private ListWithChangedEvent List;
   public EventListener(ListWithChangedEvent list)
   {
      List = list;
      List.Changed += new EventHandler(ListChanged);
   }
   private void ListChanged(object sender, EventArgs e)
   {
      Console.WriteLine("This is called when the event fires.");
   }
   public void Detach()
   {
      List.Changed -= new EventHandler(ListChanged);
      List = null;
   }
}
```

# class Object

- Methods:
  - `public virtual bool Equals(object);`
  - `public virtual int GetHashCode();`
  - `public virtual string ToString();`

# Object.Equals

- Default implementation checks references
- Should be used to compare values
- Cannot throw exeptions
- GetHashCode should also be implemented
- Must obey some rules:
  - x.Equals(x) always returns true
  - x.Equals(y) returns the same as y.Equals(x)
  - (x.Equals(y) && y.Equals(z)) returns true iff x.Equals(z) returns true
  - x.Equals(null) returns false

# Equality operators for Object class

```
bool operator ==(object x, object
  y);
bool operator !=(object x, object
  y);
```

- Default implementation compares references
- Should not throw exceptions
- Should be overloaded in pair

# operator == - example

```
using System;
class Test
{
   public static void Main()
   {
      // Value equality: True
      Console.WriteLine((2 + 2) == 4);

      // Reference equality: different objects, the same value: False
      object s = 1;
      object t = 1;
      Console.WriteLine(s == t);

      // string
      string a = "hello";
      string b = String.Copy(a);
      string c = "hello";

      // reference equality for object and literal: True
      Console.WriteLine(a == b);

      // reference equality;
      // a is reference to literal, b to object: False
      Console.WriteLine((object)a == (object)b);

      // reference equality; literals are identical, so they are packed
      //into one: True
      Console.WriteLine((object)a == (object)c);
   }
}
```

True
False
True
False
True

# Object.ToString

- Default implementation returns type name
- In derived classes usually used to return object value

# class Convert

- Allows conversions between built-in types
- Some conversions (eg., from/to `Char`) are not possible and result in `InvalidCastException`
- Exception is not thrown at precision loss, but is thrown at overflow (`System.OverflowException`)
- Methods have syntax `ToTypeName`

# class Math

- System.Object
  - System.Math
- Fields:
  - `public const double E;`
  - `public const double PI;`
- Methods:
  - `Abs, Cos, Exp, Log, Log10, Min, Max, Pow, Sqrt` etc.
  - Methods are static

# class ValueType

- System.Object
  - System.ValueType
- Value type object can be wrapped so that it looks like a reference type object (boxing). Wrapped object can be unwrapped (unboxing).

# boxing - example

```
using System;
class TestBoxing
{
   public static void Main()
   {
      int i = 123;
      object o = i;  // implicit boxing
      i = 456;       // i changes value
      Console.WriteLine
        ("The value-type value = {0}", i);
      Console.WriteLine
        ("The object-type value = {0}", o);
   }
}
```

The value-type value = 456
The object-type value = 123

# unboxing - example

```
using System;
public class UnboxingTest
{
   public static void Main()
   {
      int intI = 123;

      // Boxing
      object o = intI;

      try
      {
         int intJ = (short) o; // should be int intJ = (int) o;
         Console.WriteLine("Unboxing OK.");
      }

      catch (InvalidCastException e)
      {
         Console.WriteLine("{0} Error: Incorrect unboxing.",e);
      }
   }
}
```

# System.Collections

- Classes:
  - ArrayList
  - Hashtable
  - Queue
  - SortedList
  - Stack
- Interfaces:
  - ICollection
  - IComparer
  - IDictionary
  - IEnumerable
  - IEnumerator
  - IList
- Structures:
  - DictionaryEntry

# System.IO

- Classes:
  - BinaryReader
  - BinaryWriter
  - BufferedStream
  - Directory
  - DirectoryInfo
  - File
  - FileInfo
  - FileStream
  - Path
  - Stream
  - StreamReader
  - StreamWriter
  - StringReader
  - StringWriter
  - TextReader
  - TextWriter
  - exceptions

# Exceptions

- System.Object
  - System.Exception
    - System.SystemException
      - System.IO.IOException
        - System.IO.DirectoryNotFoundException
        - System.IO.EndOfStreamException
        - System.IO.FileLoadException
        - System.IO.FileNotFoundException
        - System.IO.PathTooLongException

# MS ASP.NET

http://samples.gotdotnet.com/quickstart/aspplus/

# Features

- WWW application environment
  - Compiled code
  - based on Common Language Runtime
  - Configured using text files
  - Optimised for multiprocessor servers
  - Extendable
  - Safe
  - Support for C#, VB i JS#

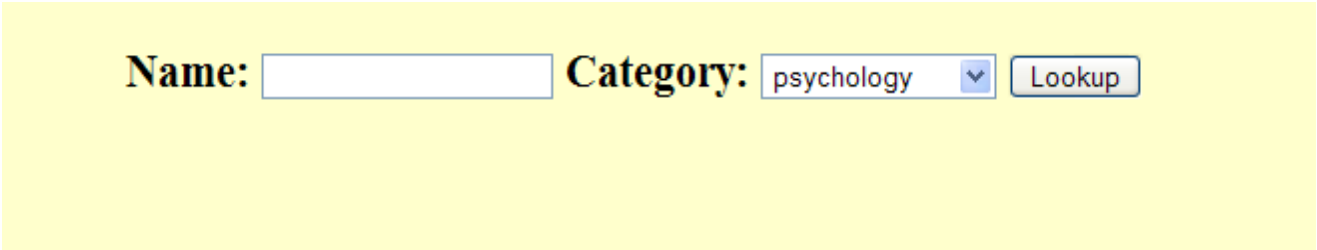
# Web Forms

- Basic way for designing UI:
  - Extension of ASP
  - Ability to create reusable controls
  - WYSIWYG editor support

# Web Forms

- Text files with .aspx extension, hosted on IIS
- Compiled into a .NET class
- Simplest example: HTML file with .aspx extension

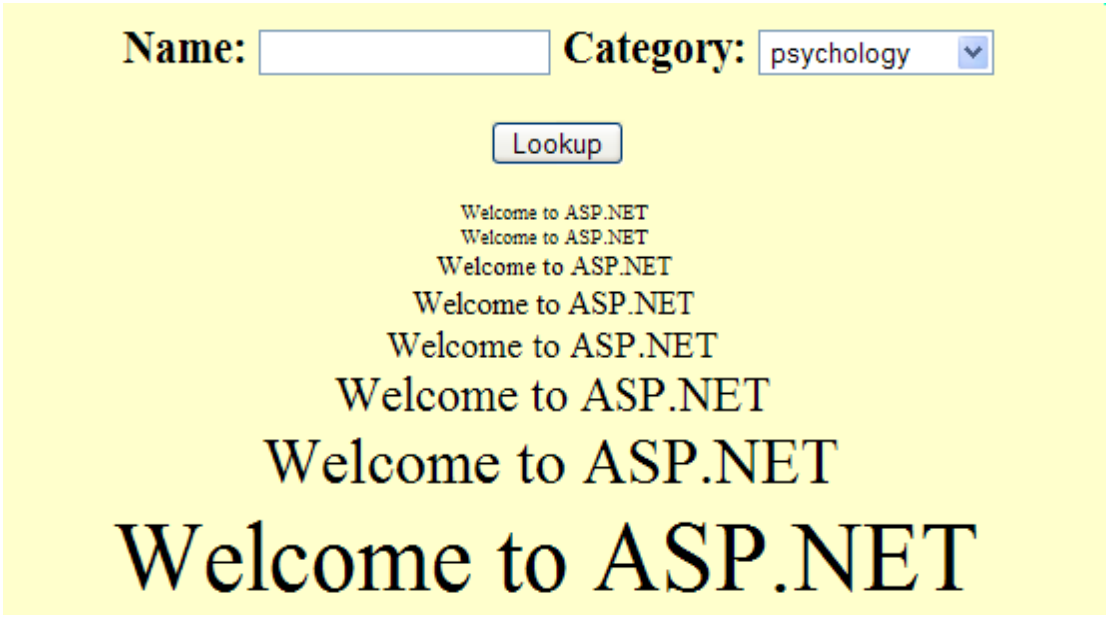
```
<html>
    <head>
        <link rel="stylesheet" href="intro.css">
    </head>

    <body>
        <center>
        <form action="intro1.aspx" method="post">
            <h3> Name: <input id="Name" type=text>
            Category:  <select id="Category" size=1>
                            <option>psychology</option>
                            <option>business</option>
                            <option>popular_comp</option>
                       </select>
            <input type=submit value="Lookup">
            </h3>
        </form>
        </center>
    </body>
</html>
```



# <% %> blocks

- Blocks surrounded with <% %> can be mixed with HTML, they contain code compiled by the server

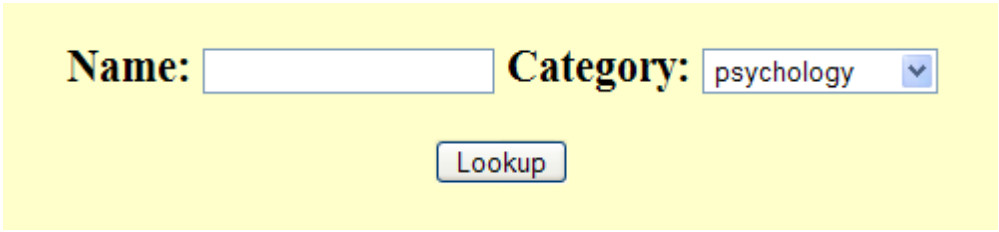
```
<%@ Page Language="C#"%>

<html>
    <head>
        <link rel="stylesheet"href="intro.css">
    </head>

    <body>
        <center>
        <form action="intro2.aspx" method="post">
            <h3> Name: <input id="Name" type=text>

            Category:  <select id="Category" size=1>
                           <option>psychology</option>
                           <option>business</option>
                           <option>popular_comp</option>
                       </select>
            </h3>
            <input type=submit value="Lookup">
            <p>
            <% for (int i=0; i <8; i++) { %>
                <font size="<%=i%>"> Welcome to ASP.NET </font> <br>
            <% }%>
        </form>
        </center>
    </body>
</html>
```



# Server controls

- Alternative for <% %> blocks
- Declared using special tags or HTML tags with `runat=”server”` attribute
- Contained in System.Web.UI.HtmlControls and System.Web.UI.WebControls

```
<%@ Page Language="C#"%>

<html>
    <head>
       <link rel="stylesheet"href="intro.css">
    </head>

    <body>
        <center>
        <form action="intro4.aspx" method="post" runat=server>
            <h3> Name: <asp:textbox id="Name" runat="server"/>
            Category:  <asp:dropdownlist id="Category" runat=server>
                          <asp:listitem>psychology</asp:listitem>
                          <asp:listitem>business</asp:listitem>
                          <asp:listitem>popular_comp</asp:listitem>
                       </asp:dropdownlist>
            </h3>
            <asp:button text="Lookup" runat="server"/>
        </form>
        </center>
    </body>
</html>
```



# Types of server controls

- HTML controls – HTML elements accessible for application running on server. Object model closely related to HTML element
- Web controls – advanced functionality, not directly related to a single HTML element
- Validators – allows testing user data against some rules
- User controls – pages containing other controls or user classes

# HTML controls

- HTML elements containing attributes thanks to which they are visible to the server (standard HTML elements are not visible)
- Each HTML element can be transformed into a HTML control by adding runat=”server” and ID attributes
- Control is visible as an object
- Inherits from System.Web.UI.HtmlControls.HtmlControl

```
html>
   <script language="C#" runat="server">

       void Page_Load(Object Src, EventArgs E) {
           Message.Text = "You last accessed this page at: " +
            DateTime.Now;
       }

   </script>

   <body>
      <h3><font face="Verdana">Manipulating Server Controls</font></h3

      This sample demonstrates how to manipulate
      the &lt;asp:label&gt; server control within
      the Page_Load event to output the current time.

      <p>
      <hr>

      <asp:label id="Message" font-size="24" font-bold="true"
          runat=server/>

   </body>
/html>
```

**Manipulating Server Controls**

This sample demonstrates how to manipulate the <asp:label> server control within the Page_Load event to output the current time.

---

**You last accessed this page at: 12/16/2005 10:28:13 AM**

# HTML controls features

- Handle events on the server side
- May handle events on the client side
- Retain the state during transmission to/from server
- Can work with validators
- Support for CSS (requires browser support)

```
tml>
  <script language="C#" runat="server">
      void EnterBtn_Click(Object Src, EventArgs E) {
          Message.Text = "Hi " + Name.Text + ", welcome to ASP.NET!";
      }
  </script>

  <body>
     <h3><font face="Verdana">Handling Control Action Events</font></h
     <p>
     This sample demonstrates how to access a &lt;asp:textbox&gt;
     server control within the "Click" event of a &lt;asp:button&gt;,
     and use its content to modify the text of a &lt;asp:label&gt;.
     <p>
     <hr>

     <form action="controls3.aspx" runat=server>
        <font face="Verdana">
           Please enter your name:
                <asp:textbox id="Name" runat=server/>
                <asp:button text="Enter" Onclick="EnterBtn_Click"
                    runat=server/>

           <p>
           <asp:label id="Message"  runat=server/>

        </font>
     </form>
  </body>
html>
```

**Handling Control Action Events**

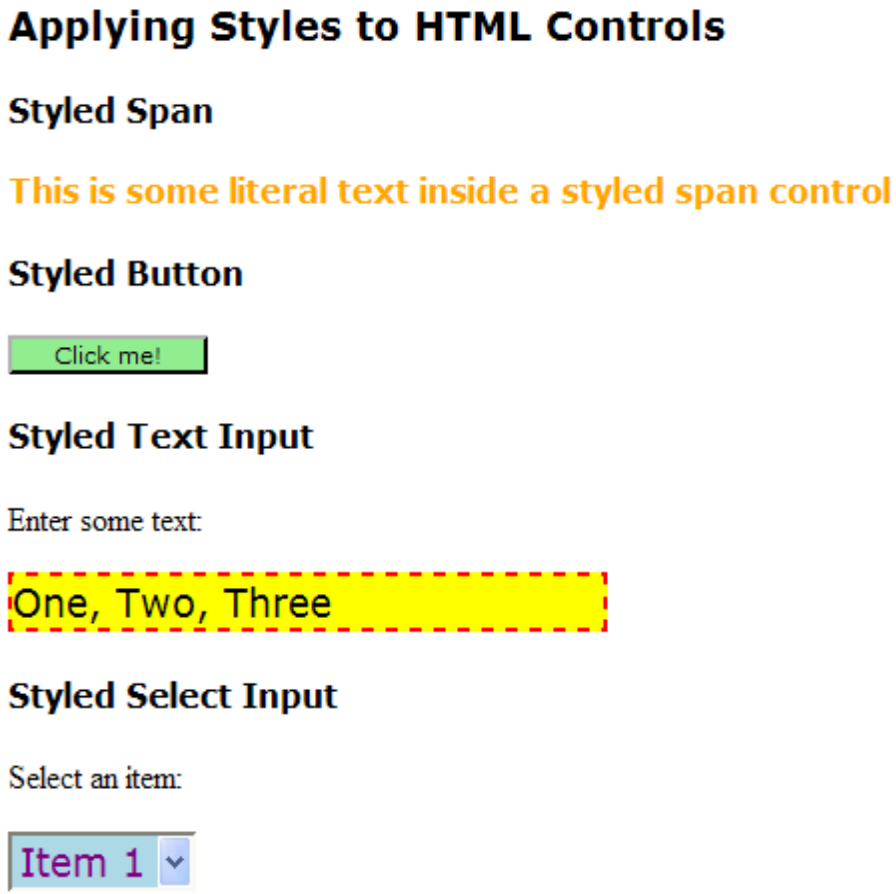
This sample demonstrates how to access a <asp:textbox> server control within the "Click" event of a <asp:button>, and use its content to modify the text of a <asp:label>.

---

Please enter your name: John Enter

Hi John, welcome to ASP.NET!

```
%@ Page Language="C#" %>
html>
body>-->
 <h3><font face="verdana">Applying Styles to HTML Controls</font></h3>
 <p><font face="verdana"><h4>Styled Span</h4></font><p>
 <span style="font: 12pt verdana; color:orange;font-weight:700"
  runat="server">This is some literal text inside a styled span contro
 </span>

 <p><font face="verdana"><h4>Styled Button</h4></font><p>
 <button style="font: 8pt verdana;background-color:lightgreen;
  border-color:black;width:100" runat="server">Click me!</button>

 <p><font face="verdana"><h4>Styled Text Input</h4></font><p>

 Enter some text: <p>
 <input type="text" value="One, Two,
   Three" style="font: 14pt verdana;
   background-color:yellow;
   border-style:dashed;
   border-color:red;
   width:300;" runat="server"/>

 <p><font face="verdana"><h4>
   Styled Select Input</h4></font><p>

 Select an item: <p>
 <select style="font: 14pt verdana;
  background-color:lightblue;
  color:purple;" runat="server">
   <option>Item 1</option>
   <option>Item 2</option>
   <option>Item 3</option>
 </select>
/body>
/html>
```



# Adding HTML controls to a webpage

- Using WYSIWYG editor
- Directly in HTML
- Programmatically:
  - Insert PlaceHolder or Panel control
  - Create desired control object, set its properties
  - Add to Controls collection, PlaceHolder or Panel control
  - Control can be added in any position of the collection, but only adding at the end always works correctly

```
<body>
   <form runat="server">
      <h3>PlaceHolder Example</h3>

      <asp:PlaceHolder id="PlaceHolder1"
           runat="server"/>
   </form>
</body>
```

---

```
private void DropDownList1_SelectedIndexChanged(object sender,
   System.EventArgs e)
{
   // Get the number of labels to create.
   int numlabels = System.Convert.ToInt32(DropDownList1.
      SelectedItem.Text);
   for (int i=1; i<=numlabels; i++)
   {
      Label myLabel = new Label();
      // Set the label's Text and ID properties.
      myLabel.Text = "Label" + i.ToString();
      myLabel.ID = "Label" + i.ToString();
      PlaceHolder1.Controls.Add(myLabel);
      // Add a spacer in the form of an HTML <BR> element.
      PlaceHolder1.Controls.Add(new LiteralControl("<br>"));
   }
}
```

# System.Web.UI.Control

- System.Object
  System.Web.UI.Control

- Properties:
  - `public virtual ControlCollection Controls {get;}`
  - `protected virtual HttpContext Context {get;}`
  - `protected EventHandlerList Events {get;}`
  - `public virtual string ID {get; set;}`
  - `public virtual Page Page {get; set;}`
  - `public virtual Control Parent {get;}`
  - `public virtual bool Visible {get; set;}`

```
rivate void Button1_Click(object sender, EventArgs MyEventArgs)

    // Find control on page.
    Control myControl1 = FindControl("TextBox2");
    if(myControl1!=null)
    {
       // Get control's parent.
       Control myControl2 = myControl1.Parent;
       Response.Write("Parent of the text box is : " + myControl2.ID)
    }
    else
    {
       Response.Write("Control not found");
    }
```

# System.Web.UI.Control

- Methods:
  - `protected virtual object SaveViewState();`
  - `protected virtual void LoadViewState(object savedState);`
  - `protected virtual void Render(HtmlTextWriter writer);`
  - `protected virtual void OnInit(EventArgs e);`
  - `protected virtual void OnLoad(EventArgs e);`
  - `protected virtual void OnUnload(EventArgs e);`
- Events:
  - `public event EventHandler Init;`
  - `public event EventHandler Load;`
  - `public event EventHandler Unload;`

```
<%@ Page language="c#" Codebehind="WebForm1.aspx.cs"
 AutoEventWireup="false" Inherits="WebApplication5.WebForm1" %>
<%@ Register TagPrefix="UserCtrlSample" Namespace="WebApplication5"
 Assembly="WebApplication5" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN" >
<HTML>
    <HEAD>
        <title>WebForm1</title>
        <meta content="Microsoft Visual Studio .NET 7.1"
          name="GENERATOR">
        <meta content="C#" name="CODE_LANGUAGE">
        <meta content="JavaScript" name="vs_defaultClientScript">
        <meta content="http://schemas.microsoft.com/intellisense/ie5"
          name="vs_targetSchema">
    </HEAD>
    <body MS_POSITIONING="GridLayout">
        <form id="Form1" method="post" runat="server">
            <USERCTRLSAMPLE:MYCTRL id="Control1" runat="server">
             </USERCTRLSAMPLE:MYCTRL></form>
    </body>
</HTML>
```

```
namespace WebApplication5{
    public class WebForm1 : System.Web.UI.Page{
        protected WebApplication5.MyCtrl Control1;
        TextWriter myFile = File.CreateText
            (@"C:\Inetpub\wwwroot\NewTextFile.txt");
        private void Page_Load(object sender, System.EventArgs e){
            Control1.file = myFile;
            myFile.WriteLine("Page has loaded.");}
        private void Page_Unload(object sender, System.EventArgs e)    {
            myFile.WriteLine("Page was unloaded.");
            myFile.Close();}
        override protected void OnInit(EventArgs e){
            InitializeComponent();
            base.OnInit(e);}
        private void InitializeComponent(){
            this.Load += new System.EventHandler(this.Page_Load);
            this.Unload += new System.EventHandler(this.Page_Unload);}
    }
    public class MyCtrl : Control{
        public TextWriter file;
        public MyCtrl(){
            Load += new EventHandler(MyControl_Load);
            Unload += new EventHandler(MyControl_Unload);}
        protected override void Render(HtmlTextWriter output){
            output.Write("{0} {1} {2}","<H2>",
            "Welcome to Control Development!", "</H2>");}
        void MyControl_Load(object sender,EventArgs e){
            file.WriteLine("Custom control has loaded.");}
        void MyControl_Unload(object sender,EventArgs e){
            file.WriteLine("Custom control was unloaded.");}
    }
}
```

```
Page has loaded.
Custom control has loaded.
Custom control was unloaded.
Page was unloaded.
```

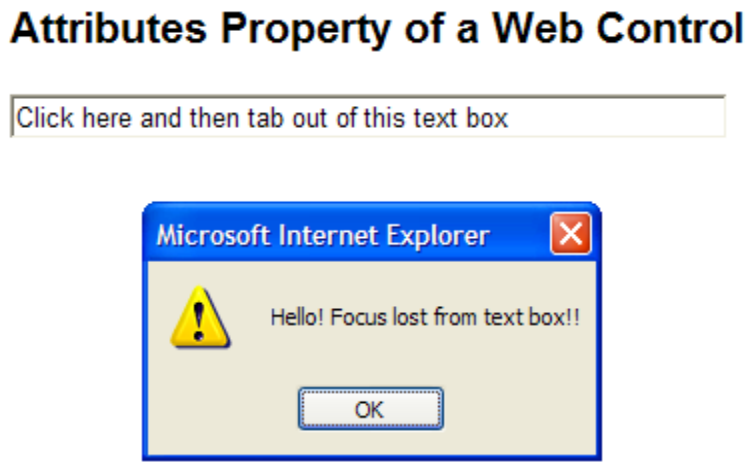
**Welcome to Control Development!**

```
protected override object SaveViewState()
{  // Change Text Property of Label when this function is invoked.
   if(HasControls() && (Page.IsPostBack))
   {
      ((Label)(Controls[0])).Text = "Custom Control Has Saved State";
   }
   // Save State as a cumulative array of objects.
   object baseState = base.SaveViewState();
   string userText = UserText;
   string passwordText = PasswordText;
   object[] allStates = new object[3];
   allStates[0] = baseState;
   allStates[1] = userText;
   allStates[2] = PasswordText;
   return allStates;
}
 protected override void LoadViewState(object savedState)
 {
    if (savedState != null)
    {
       // Load State from the array of objects that was saved at ;
       // SavedViewState.
       object[] myState = (object[])savedState;
       if (myState[0] != null)
          base.LoadViewState(myState[0]);
       if (myState[1] != null)
          UserText = (string)myState[1];
       if (myState[2] != null)
          PasswordText = (string)myState[2];
    }
 }
```

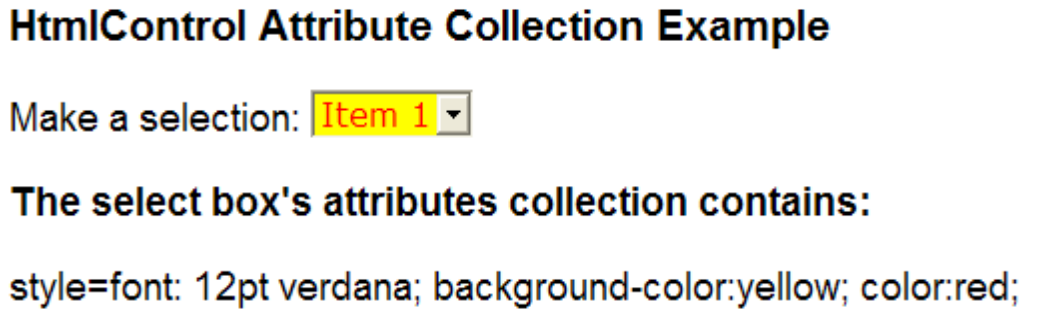
# System.Web.UI.HtmlControls.HtmlControl

- System.Object
  System.Web.UI.Control
    **System.Web.UI.HtmlControls.HtmlControl**
      System.Web.UI.HtmlControls.HtmlContainerControl
      System.Web.UI.HtmlControls.HtmlImage
      System.Web.UI.HtmlControls.HtmlInputControl
- `public abstract class HtmlControl : Control, IAttributeAccessor`
- Properties:
  - `public AttributeCollection Attributes {get;}`
  - `public bool Disabled {get; set;}`
  - `public CssStyleCollection Style {get;}`
  - `public virtual string TagName {get;}`

```
<html>
<head>
   <script language="C#" runat="server">

      void Page_Load(Object sender, EventArgs e) {
         TextBox1.Attributes["onblur"]="javascript:alert('Hello!
           Focus lost from text box!!');";
      }
   </script>

</head>
<body>
   <h3>Attributes Property of a Web Control</h3>
<form runat="server">

   <asp:TextBox id="TextBox1" columns=54
    Text="Click here and then tab out of this text box"
    runat="server"/>

</form>
</body>
</html>
```



```
%@ Page Language="C#" AutoEventWireup="True" %>
html>
script language="C#" runat="server">
  void Page_Load(Object sender, EventArgs e){
     Message.InnerHtml = "<h4>The select box's attributes collection
      contains:</h4>";
     IEnumerator keys = Select.Attributes.Keys.GetEnumerator();
     while (keys.MoveNext()){
        String key = (String)keys.Current;
        Message.InnerHtml += key +"="+ Select.Attributes[key] + "<br>"
     }
  }
/script>

body>
  <h3>HtmlControl Attribute Collection Example</h3>
  Make a selection:
  <select id="Select"
          style="font: 12pt verdana;
                 background-color:yellow;
                 color:red;"
          runat="server">
     <option>Item 1</option>
     <option>Item 2</option>
     <option>Item 3</option>
  </select>
  <p>
  <span id="Message" MaintainState="false" runat="server" />
/body>
/html>
```



```
%@ Page Language="C#" AutoEventWireup="True" %>
html>
script language="C#" runat="server">
  void Page_Load(Object sender, EventArgs e){
     Message.InnerHtml = "<h4>The select box's style collection
      contains:</h4>";
     IEnumerator keys = Select.Style.Keys.GetEnumerator();
     while (keys.MoveNext()){
        String key = (String)keys.Current;
        Message.InnerHtml += key +"="+ Select.Style[key] + "<br>";
     }
  }
/script>

body>
  <h3>HtmlControl Style Collection Example</h3>
  Make a selection:
  <select id="Select"
          style="font: 12pt verdana;
                 background-color:yellow;
                 color:red;"
          runat="server">
     <option>Item 1</option>
     <option>Item 2</option>
     <option>Item 3</option>
  </select>
  <p>
  <span id="Message" MaintainState="false" runat="server" />
/body>
/html>
```

**HtmlControl Style Collection Example**

Make a selection: Item 1

**The select box's style collection contains:**

font=12pt verdana
background-color=yellow
color=red

- HtmlAnchor <a>
- HtmlButton <button>
- HtmlForm <form>
- HtmlGenericControl
- HtmlImage <img>
- HtmlInputButton <input type= button>, <input type= submit>, <input type= reset>
- HtmlInputCheckBox <input type= checkbox>
- HtmlInputFile <input type= file>
- HtmlInputHidden <input type=hidden>
- HtmlInputImage <input type= image>
- HtmlInputRadioButton <input type= radio>
- HtmlInputText <input type= text>, <input type= password>
- HtmlSelect <select>
- HtmlTable <table>
- HtmlTableCell <td>, <th>  w HtmlTableRow.
- HtmlTableCellCollection
- HtmlTableRow <tr>
- HtmlTableRowCollection
- HtmlTextArea <textarea>

# Web controls

- Advanced functionality
- Browser detection
- Ability to define parateters using templates
- Ability to delay sending data to server
- Ability to bubble events

```
<html>
  <head>
    <script language="C#" runat="server">
      void Button1_Click(object Source, EventArgs e){
          String s = "Selected items:<br>";
          for (int i=0; i < Check1.Items.Count; i++){
              if ( Check1.Items[ i ].Selected ){
                  s = s + Check1.Items[i].Text;
                  s = s + "<br>";
              }
          }
          Label1.Text = s;
      }
      void chkLayout_CheckedChanged(Object sender, EventArgs e){
          if (chkLayout.Checked == true){
              Check1.RepeatLayout = RepeatLayout.Table;
          }else {
              Check1.RepeatLayout = RepeatLayout.Flow;
          }
      }
      void chkDirection_CheckedChanged(Object sender, EventArgs e) {
          if (chkDirection.Checked == true) {
              Check1.RepeatDirection = RepeatDirection.Horizontal;
          }else{
              Check1.RepeatDirection = RepeatDirection.Vertical;
          }
      }
    </script>
  </head>
```

```
<body>
    <h3><font face="Verdana">CheckBoxList Example</font></h3>
    <form runat=server>
        <asp:CheckBoxList id=Check1 runat="server">
            <asp:ListItem>Item 1</asp:ListItem>
            <asp:ListItem>Item 2</asp:ListItem>
            <asp:ListItem>Item 3</asp:ListItem>
            <asp:ListItem>Item 4</asp:ListItem>
            <asp:ListItem>Item 5</asp:ListItem>
            <asp:ListItem>Item 6</asp:ListItem>
        </asp:CheckBoxList>
        <p>
        <asp:CheckBox id=chkLayout
         OnCheckedChanged="chkLayout_CheckedChanged"
         Text="Display Table Layout" Checked=true AutoPostBack="true"
         runat="server" />
        <br>
        <asp:CheckBox id=chkDirection
         OnCheckedChanged="chkDirection_CheckedChanged"
         Text="Display Horizontally" AutoPostBack="true"
         runat="server" />
        <p>
        <asp:Button id=Button1 Text="Submit" onclick="Button1_Click"
         runat="server"/>
        <p>
        <asp:Label id=Label1 font-name="Verdana" font-size="8pt"
         runat="server"/>
    </form>
</body>
</html>
```

## CheckBoxList Example

- [x] Item 1
- [ ] Item 2
- [x] Item 3
- [ ] Item 4
- [ ] Item 5
- [ ] Item 6

- [x] Display Table Layout
- [ ] Display Horizontally

Submit

Selected items:
Item 1
Item 3

## CheckBoxList Example

- [x] Item 1 [ ] Item 2 [x] Item 3 [ ] Item 4 [ ] Item 5 [ ] Item 6

- [x] Display Table Layout
- [x] Display Horizontally

Submit

Selected items:
Item 1
Item 3

```
%@ Import Namespace="System.Data" %>

html>
script language="C#" runat="server">
    ICollection CreateDataSource() {
        DataTable dt = new DataTable();
        DataRow dr;
        dt.Columns.Add(new DataColumn("IntegerValue",typeof(Int32)));
        dt.Columns.Add(new DataColumn("StringValue",typeof(string)));
        dt.Columns.Add(new DataColumn("DateTimeValue",typeof(DateTime)));
        dt.Columns.Add(new DataColumn("BoolValue",typeof(bool)));
        dt.Columns.Add(new DataColumn("CurrencyValue",typeof(double)));
        for (int i = 0; i < 9; i++) {
            dr = dt.NewRow();
            dr[0] = i;
            dr[1] = "Item " + i.ToString();
            dr[2] = DateTime.Now;
            dr[3] = (i % 2 != 0) ? true : false;
            dr[4] = 1.23 * (i+1);
            dt.Rows.Add(dr);
        }
        DataView dv = new DataView(dt);
        return dv;
    }

    void Page_Load(Object sender, EventArgs e) {
        MyDataGrid.DataSource = CreateDataSource();
        MyDataGrid.DataBind();
    }

/script>
```

```
<body>

    <h3><font face="Verdana">Simple DataGrid Example</font></h3>

    <form runat=server>

      <ASP:DataGrid id="MyDataGrid" runat="server"
        BorderColor="black"
        BorderWidth="1"
        GridLines="Both"
        CellPadding="3"
        CellSpacing="0"
        Font-Name="Verdana"
        Font-Size="8pt"
        HeaderStyle-BackColor="#aaaadd"
      />

  </form>

</body>
</html>
```

**Simple DataGrid Example**

| IntegerValue | StringValue | DateTimeValue | BoolValue | CurrencyValue |
|---|---|---|---|---|
| 1 | Item 1 | 12/17/2005 3:20:00 PM | True | 2.46 |
| 2 | Item 2 | 12/17/2005 3:20:00 PM | False | 3.69 |
| 3 | Item 3 | 12/17/2005 3:20:00 PM | True | 4.92 |
| 4 | Item 4 | 12/17/2005 3:20:00 PM | False | 6.15 |
| 5 | Item 5 | 12/17/2005 3:20:00 PM | True | 7.38 |
| 6 | Item 6 | 12/17/2005 3:20:00 PM | False | 8.61 |
| 7 | Item 7 | 12/17/2005 3:20:00 PM | True | 9.84 |
| 8 | Item 8 | 12/17/2005 3:20:00 PM | False | 11.07 |
| 9 | Item 9 | 12/17/2005 3:20:00 PM | True | 12.3 |

# Validators

- HtmlInputText - Value
- HtmlTextArea - Value
- HtmlSelect - Value
- HtmlInputFile - Value
- TextBox - Text
- ListBox - SelectedItem.Value
- DropDownList - SelectedItem.Value
- RadioButtonList - SelectedItem.Value

```
<html>
<head>
    <script language="C#" runat=server>

        void ValidateBtn_Click(Object Sender, EventArgs E) {

            if (Page.IsValid == true) {
                lblOutput.Text = "Page is Valid!";
            }
            else {
                lblOutput.Text = "Some of the required fields are empty
            }
        }

   </script>

</head>
```

```
body>
h3><font face="Verdana">Simple RequiredField Validator Sample</font>
/h3><p><form runat="server">
   <table bgcolor="#eeeeee" cellpadding=10>
   <tr valign="top"><td colspan=3>
     <asp:Label ID="lblOutput" Text="Fill in the required fields below
      ForeColor="red" Font-Name="Verdana" Font-Size="10"
      runat=server /><br></td></tr>
   <tr><td colspan=3>
     <font face=Verdana size=2><b>Credit Card Information</b></font>
     </td></tr>
   <tr><td align=right>
       <font face=Verdana size=2>Card Type:</font>
     </td><td>
       <ASP:RadioButtonList id=RadioButtonList1 RepeatLayout="Flow"
            runat=server>
           <asp:ListItem>MasterCard</asp:ListItem>
           <asp:ListItem>Visa</asp:ListItem>
       </ASP:RadioButtonList>
     </td><td align=middle rowspan=1>
       <asp:RequiredFieldValidator id="RequiredFieldValidator1"
           ControlToValidate="RadioButtonList1"
           Display="Static"
           InitialValue="" Width="100%" runat=server>
           *
       </asp:RequiredFieldValidator>
     </td></tr>
   <tr><td align=right>
       <font face=Verdana size=2>Card Number:</font>
     </td><td>
       <ASP:TextBox id=TextBox1 runat=server />
     </td><td>
       <asp:RequiredFieldValidator id="RequiredFieldValidator2"
           ControlToValidate="TextBox1"
           Display="Static"
           Width="100%" runat=server>
           *
       </asp:RequiredFieldValidator>
     </td></tr>
```

```
<tr><td align=right>
        <font face=Verdana size=2>Expiration Date:</font>
      </td><td>
        <ASP:DropDownList id=DropDownList1 runat=server>
            <asp:ListItem></asp:ListItem>
            <asp:ListItem >06/00</asp:ListItem>
            <asp:ListItem >07/00</asp:ListItem>
            <asp:ListItem >08/00</asp:ListItem>
            <asp:ListItem >09/00</asp:ListItem>
            <asp:ListItem >10/00</asp:ListItem>
            <asp:ListItem >11/00</asp:ListItem>
            <asp:ListItem >01/01</asp:ListItem>
            <asp:ListItem >02/01</asp:ListItem>
            <asp:ListItem >03/01</asp:ListItem>
            <asp:ListItem >04/01</asp:ListItem>
            <asp:ListItem >05/01</asp:ListItem>
            <asp:ListItem >06/01</asp:ListItem>
            <asp:ListItem >07/01</asp:ListItem>
            <asp:ListItem >08/01</asp:ListItem>
            <asp:ListItem >09/01</asp:ListItem>
            <asp:ListItem >10/01</asp:ListItem>
            <asp:ListItem >11/01</asp:ListItem>
            <asp:ListItem >12/01</asp:ListItem>
        </ASP:DropDownList>
      </td><td>
        <asp:RequiredFieldValidator id="RequiredFieldValidator3"
          ControlToValidate="DropDownList1"
          Display="Static"
          InitialValue="" Width="100%" runat=server>
          *
        </asp:RequiredFieldValidator>
      </td><td></tr><tr><td></td><td>
        <ASP:Button id=Button1 text="Validate"
         OnClick="ValidateBtn_Click" runat=server />
      </td><td></td></tr></table>
</form>
</body>
</html>
```

## Simple RequiredField Validator Sample

Some of the required fields are empty

**Credit Card Information**

Card Type: ○ MasterCard ○ Visa *

Card Number: *

Expiration Date: *

Validate

## Simple RequiredField Validator Sample

Some of the required fields are empty

**Credit Card Information**

Card Type: ◉ MasterCard ○ Visa

Card Number: 123456

Expiration Date: *

Validate

## Simple RequiredField Validator Sample

Page is Valid!

**Credit Card Information**

Card Type: ◉ MasterCard ○ Visa

Card Number: 123456

Expiration Date: 06/01

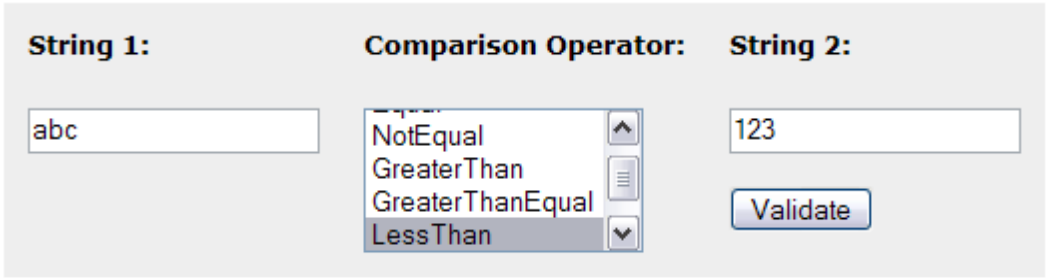
Validate

```
<%@ Page clienttarget=downlevel %>

<html>
<head>
    <script language="C#" runat="server">

        void Button1_OnSubmit(Object sender, EventArgs e)
{

            if (Page.IsValid) {
               lblOutput.Text = "Result: Valid!";
            }
            else {
               lblOutput.Text = "Result: Not valid!";
            }
        }

        void lstOperator_SelectedIndexChanged(Object
sender, EventArgs e) {

            comp1.Operator = (ValidationCompareOperator)
lstOperator.SelectedIndex;
            comp1.Validate();
        }

   </script>

</head>
```

```
<body>
    <h3><font face="Verdana">CompareValidator Example</font></h3>
    <p>Type a value in each textbox, select a comparison operator,
      then click "Validate" to test.</p>
    <form runat=server><table bgcolor="#eeeeee" cellpadding=10>
      <tr valign="top"><td>
            <h5><font face="Verdana">String 1:</font></h5>
            <asp:TextBox id="txtComp" runat="server"></asp:TextBox>
        </td><td>
            <h5><font face="Verdana">Comparison Operator:</font></h5>
            <asp:ListBox id="lstOperator" OnSelectedIndexChanged=
             "lstOperator_SelectedIndexChanged" runat="server">
                    <asp:ListItem Selected Value="Equal" >Equal
                      </asp:ListItem>
                    <asp:ListItem Value="NotEqual" >NotEqual
                      </asp:ListItem>
                    <asp:ListItem Value="GreaterThan" >GreaterThan
                      </asp:ListItem>
                    <asp:ListItem Value="GreaterThanEqual" >
                      GreaterThanEqual</asp:ListItem>
                    <asp:ListItem Value="LessThan" >LessThan
                      </asp:ListItem>
                    <asp:ListItem Value="LessThanEqual" >LessThanEqual
                      </asp:ListItem>
            </asp:ListBox>
        </td><td>
            <h5><font face="Verdana">String 2:</font></h5>
            <asp:TextBox id="txtCompTo" runat="server"></asp:TextBox><p
            <asp:Button runat=server Text="Validate" ID="Button1"
             onclick="Button1_OnSubmit" />
        </td></tr></table>
      <asp:CompareValidator id="comp1" ControlToValidate="txtComp"
         ControlToCompare = "txtCompTo" Type="String" runat="server"/>
      <br>
      <asp:Label ID="lblOutput" Font-Name="verdana" Font-Size="10pt"
         runat="server"/>
    </form>
</body>
</html>
```

## CompareValidator Example

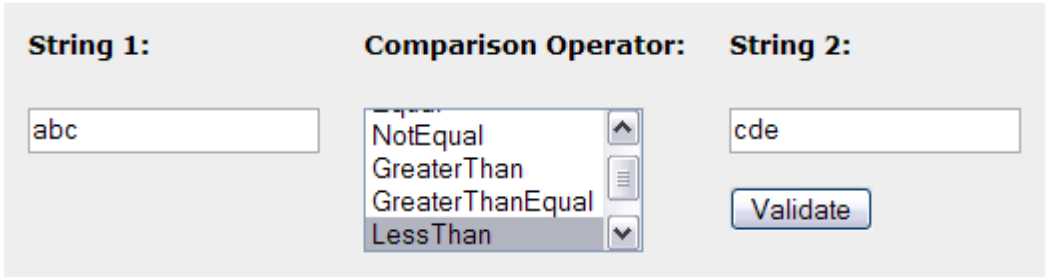
Type a value in each textbox, select a comparison operator, then click "Validate" to test.




Result: Not valid!

## CompareValidator Example

Type a value in each textbox, select a comparison operator, then click "Validate" to test.




Result: Valid!

# Other validations

- Range
- Using regular expressions
- Using user functions

# User controls

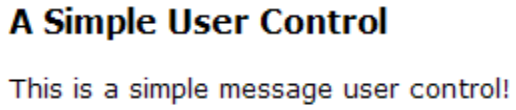
- A way to reuse ASP.NET pages
- Are of `System.Web.UI.UserControl` type
- Usually created as an ascx file
- Included using `Register` directive

Pagelet1.aspx

```
<%@ Page Language="C#" %>
<%@ Register TagPrefix="Acme" TagName="Message" Src="pagelet1.ascx" %>

<html>
<body style="font: 10pt verdana">

  <h3>A Simple User Control</h3>

  <Acme:Message runat="server"/>

</body>
</html>
```

Pagelet1.ascx

```
This is a simple message user control!
```



## Pagelet2.aspx

```
%@ Register TagPrefix="Acme" TagName="Message" Src="pagelet2.ascx" %>
html>
 <script language="C#" runat="server">

     void SubmitBtn_Click(Object sender, EventArgs E) {
         MyMessage.Text = "Message text changed!";
         MyMessage.Color = "red";
     }

 </script>

body style="font: 10pt verdana">
 <h3>A Simple User Control w/ Properties</h3>
 <form runat="server">
   <Acme:Message id="MyMessage" Text="This is a custom message!"
     Color="blue" runat="server"/><p>
   <asp:button text="Change Properties" OnClick="SubmitBtn_Click"
     runat=server/>
 </form>
/body>
/html>
```

## Pagelet2.ascx

```
script language="C#" runat="server">

 public String Color = "blue";
 public String Text = "This is a simple message user control!";

/script>

span id="Message" style="color:<%=Color%>"><%=Text%></span>
```

**A Simple User Control w/ Properties**

This is a custom message!

Change Properties

**A Simple User Control w/ Properties**

Message text changed!

Change Properties

# Services

- Contained in .asmx file
- Exported methods have [WebMethod] attribute
- Ability to create WSDL document for the client
- Client can create a wrapper class using for example WSDL.exe tool

```
<%@ WebService Language="C#" Class="MathService" %>

using System;
using System.Web.Services;

public class MathService : WebService {

    [WebMethod]
    public float Add(float a, float b)
    {
        return a + b;
    }

    [WebMethod]
    public float Subtract(float a, float b)
    {
        return a - b;
    }

    [WebMethod]
    public float Multiply(float a, float b)
    {
        return a * b;
    }

    [WebMethod]
    public float Divide(float a, float b)
    {
        if (b==0) return -1;
        return a / b;
    }

}
```

# MathService

The following operations are supported. For a formal definition, please review the Service Description.

- Multiply
- Divide
- Add
- Subtract

---

**This web service is using http://tempuri.org/ as its default namespace.**

**Recommendation: Change the default namespace before the XML Web service is made public.**

Each XML Web service needs a unique namespace in order for client applications to distinguish it from other services on the Web. http://tempuri.org/ is available for XML Web services that are under development, but published XML Web services should use a more permanent namespace.

Your XML Web service should be identified by a namespace that you control. For example, you can use your company's Internet domain name as part of the namespace. Although many XML Web service namespaces look like URLs, they need not point to actual resources on the Web. (XML Web service namespaces are URIs.)

For XML Web services creating using ASP.NET, the default namespace can be changed using the WebService attribute's Namespace property. The WebService attribute is an attribute applied to the class that contains the XML Web service methods. Below is a code example that sets the namespace to "http://microsoft.com/webservices/":

C#

```
[WebService(Namespace="http://microsoft.com/webservices/")]
public class MyWebService {
    // implementation
}
```

Visual Basic.NET

```
<WebService(Namespace:="http://microsoft.com/webservices/")> Public Class MyWebService
    ' implementation
End Class
```

For more details on XML namespaces, see the W3C recommendation on Namespaces in XML.

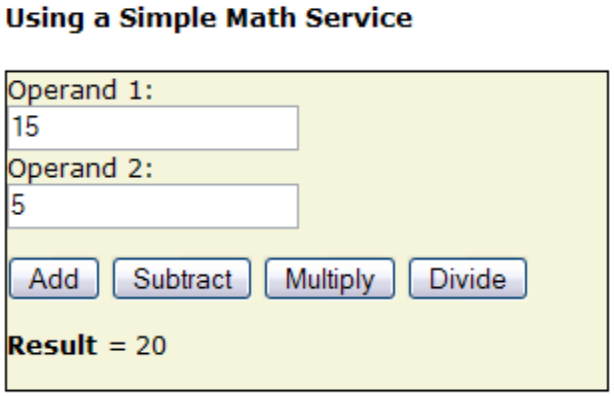
For more details on WSDL, see the WSDL Specification.

For more details on URIs, see RFC 2396.

```
//------------------------------------------------------------------------------
// <autogenerated>
//     This code was generated by a tool.
//     Runtime Version: 1.0.3705.0
//
//     Changes to this file may cause incorrect behavior and will be lost if
//     the code is regenerated.
// </autogenerated>
//------------------------------------------------------------------------------

//
// This source code was auto-generated by wsdl, Version=1.0.3705.0.
//
namespace MathService {
    using System.Diagnostics;
    using System.Xml.Serialization;
    using System;
    using System.Web.Services.Protocols;
    using System.ComponentModel;
    using System.Web.Services;


    /// <remarks/>
    [System.Diagnostics.DebuggerStepThroughAttribute()]
    [System.ComponentModel.DesignerCategoryAttribute("code")]
    [System.Web.Services.WebServiceBindingAttribute(Name="MathServiceSoap",
    Namespace="http://tempuri.org/")]
    public class MathService : System.Web.Services.Protocols.SoapHttpClientProtocol {

        /// <remarks/>
        public MathService() {
            this.Url = "http://localhost/quickstart/aspplus/samples/services/MathService/CS
             /MathService.a" + "smx";
        }

        /// <remarks/>
        [System.Web.Services.Protocols.SoapDocumentMethodAttribute("http://tempuri.org/Add",
        RequestNamespace="http://tempuri.org/", ResponseNamespace="http://tempuri.org/",
        Use=System.Web.Services.Description.SoapBindingUse.Literal,
        ParameterStyle=System.Web.Services.Protocols.SoapParameterStyle.Wrapped)]
        public System.Single Add(System.Single a, System.Single b) {
            object[] results = this.Invoke("Add", new object[] {
                        a,
                        b});
            return ((System.Single)(results[0]));
        }

        /// <remarks/>
        public System.IAsyncResult BeginAdd(System.Single a, System.Single b,
        System.AsyncCallback callback, object asyncState) {
            return this.BeginInvoke("Add", new object[] {
                        a,
                        b}, callback, asyncState);
        }

        /// <remarks/>
        public System.Single EndAdd(System.IAsyncResult asyncResult) {
            object[] results = this.EndInvoke(asyncResult);
            return ((System.Single)(results[0]));
        }
```

```
<%@ Import Namespace="MathService" %>

<html>

<script language="C#" runat="server">

  float operand1 = 0;
  float operand2 = 0;

  public void Submit_Click(Object sender, EventArgs E)
  {
      try
      {
        operand1 = float.Parse(Operand1.Text);
        operand2 = float.Parse(Operand2.Text);
      }
      catch (Exception) { /* ignored */ }

      MathService service = new MathService();

        switch (((Control)sender).ID)
        {
          case "Add"      : Result.Text = "<b>Result</b> = " +
            service.Add(operand1, operand2).ToString(); break;
          case "Subtract" : Result.Text = "<b>Result</b> = " +
            service.Subtract(operand1, operand2).ToString(); break;
          case "Multiply" : Result.Text = "<b>Result</b> = " +
            service.Multiply(operand1, operand2).ToString(); break;
          case "Divide"   : Result.Text = "<b>Result</b> = " +
            service.Divide(operand1, operand2).ToString(); break;
        }
  }

</script>
```

```
<body style="font: 10pt verdana">

  <h4>Using a Simple Math Service </h4>

  <form runat="server">
  <div style="padding:15,15,15,15;background-color:beige;width:300;
   border-color:black;border-width:1;border-style:solid">

    Operand 1: <br><asp:TextBox id="Operand1" Text="15"
      runat="server"/><br>
    Operand 2: <br><asp:TextBox id="Operand2" Text="5"
      runat="server"/><p>

    <input type="submit" id="Add" value="Add"
      OnServerClick="Submit_Click" runat="server">
    <input type="submit" id="Subtract" value="Subtract"
      OnServerClick="Submit_Click" runat="server">
    <input type="submit" id="Multiply" value="Multiply"
      OnServerClick="Submit_Click" runat="server">
    <input type="submit" id="Divide" value="Divide"
      OnServerClick="Submit_Click" runat="server">
    <p>
    <asp:Label id="Result" runat="server"/>
  </div>

  </form>

</body>
</html>
```



# ASP.NET applications

- Everything that is contained in a folder (and its subfolders) on the server
- Optional Global.aspx file defining HttpApplication class extensions
- Ability to provide code for `Application_Start`, `Application_End`, `Session_Start`, `Application_Error`, `Session_End`
- Ability to store objects in Application property (also accessible from a page)

Global.aspx

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.IO" %>

<script language="C#" runat="server">

    void Application_Start(Object sender, EventArgs e) {

        DataSet ds = new DataSet();

        FileStream fs = new FileStream(
          Server.MapPath("schemadata.xml"),
          FileMode.Open,FileAccess.Read);
        StreamReader reader = new StreamReader(fs);
        ds.ReadXml(reader);
        fs.Close();

        DataView view = new DataView(ds.Tables[0]);
        Application["Source"] = view;
    }

</script>
```

Application2.aspx

```
<%@ Import Namespace="System.Data" %>

<html>

  <script language="C#" runat="server">

      void Page_Load(Object Src, EventArgs E ) {

          DataView Source = (DataView)(Application["Source"]);
          MySpan.Controls.Add(new LiteralControl
             (Source.Table.TableName));
          MyDataGrid.DataSource = Source;
          MyDataGrid.DataBind();
      }

  </script>

<body>
  <h3><font face="Verdana">Reading Data in Application_OnStart
   </font></h3>
  <h4><font face="Verdana">XML Data for Table: <asp:PlaceHolder
   runat="server" id="MySpan"/></font></h4>

  <ASP:DataGrid id="MyDataGrid" runat="server"
    Width="900"
    BackColor="#ccccff"
    BorderColor="black"
    ShowFooter="false"
    CellPadding=3
    CellSpacing="0"
    Font-Name="Verdana"
    Font-Size="8pt"
    HeaderStyle-BackColor="#aaaadd"
    EnableViewState="false"
  />
</body>
</html>
```

# Reading Data in Application_OnStart

## XML Data for Table: Table

| ProductID | CategoryID | ProductName | ProductDescription | UnitPrice | ImagePath | ServingSize | Servings | Quantity | MinOnHand | MaxOnHand | Manufacturer |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1001 | 1 | Chocolate City Milk | Chocolate City Milk Description | 2 | /quickstart/aspplus/images/milk5.gif | 8 fl oz (240 mL) | 8 | 0 | 0 | 0 | Chocolate City |
| 1002 | 1 | Bessie Brand 2% Milk | Bessie Brand 2% Milk Description | 1.19 | /quickstart/aspplus/images/milk1.gif | 8 fl oz (240 mL) | 8 | 0 | 0 | 0 | Milk Factory |
| 1003 | 1 | Funny Farms Milk | Funny Farms Whole Milk Description | 1.29 | /quickstart/aspplus/images/milk4.gif | 8 fl oz (240 mL) | 10 | 0 | 0 | 0 | Funny Farms |
| 2001 | 2 | Fruity Pops | Fruity Pops Description | 4.07 | /quickstart/aspplus/images/cereal7.gif | 3/4 cup (30 g) | 17 | 0 | 0 | 0 | River Mills |
| 2002 | 2 | U.F.O.s Cereal | U.F.O.s Cereal Description | 3.34 | /quickstart/aspplus/images/cereal3.gif | 1 cup (30 g) | 10 | 0 | 0 | 0 | Acme Harvesters |
| 2003 | 2 | Healthy Grains | Healthy Grains Cereal Description | 3.78 | /quickstart/aspplus/images/cereal1.gif | 3/4 cup (30 g) | 17 | 0 | 0 | 0 | All Natural Co. |
| 2004 | 2 | Super Sugar Strike | Super Sugar Strike Description | 4.17 | /quickstart/aspplus/images/cereal6.gif | 3/4 cup (30 g) | 17 | 0 | 0 | 0 | Capitol Cereals |
| 3001 | 3 | Purple Rain | Brown Barrel Root Beer Description | 1.1 | /quickstart/aspplus/images/soda5.gif | 4 fl oz (120 mL) | 8 | 0 | 0 | 0 | BrainFade, Inc. |
| 3002 | 3 | Extreme Orange | Bargain Cola Description | 0.89 | /quickstart/aspplus/images/soda6.gif | 6 fl oz (180 mL) | 6 | 0 | 0 | 0 | SuperX Beverages |
| 3003 | 3 | Kona Diet Cola | Super Red Pop Soda Description | 1.1 | /quickstart/aspplus/images/soda7.gif | 4 fl oz (120 mL) | 10 | 0 | 0 | 0 | Kona Kola Co. |
| 3004 | 3 | Fizzy Fizzing Drink | Lemon Lime Quencher Description | 1.05 | /quickstart/aspplus/images/soda8.gif | 6 fl oz (180 mL) | 5 | 0 | 0 | 0 | Sparkle Co. |
| 1005 | 1 | Marigold Whole Milk | Marigold Whole Milk Description | 1.39 | /quickstart/aspplus/images/milk6.gif | 8 fl oz (240 mL) | 8 | 0 | 0 | 0 | Marigold Meadows |

# Application state - session

Global.aspx

```
<script language="C#" runat="server">

    void Session_Start(Object sender, EventArgs e) {
      Session["BackColor"] = "beige";
      Session["ForeColor"] = "black";
      Session["LinkColor"] = "blue";
      Session["FontSize"] = "8pt";
      Session["FontName"] = "verdana";
    }
</script>
```

Session1.aspx

```
<html>
  <script language="C#" runat="server">
      String GetStyle(String key) {
        return Session[key].ToString();
      }
  </script>

  <style>
    body {
      font: <%=GetStyle("FontSize")%> <%=GetStyle("FontName")%>;
      background-color: <%=GetStyle("BackColor")%>;
    }
    a { color: <%=GetStyle("LinkColor")%> }
  </style>
  <body style="color:<%=GetStyle("ForeColor")%>">
    <h3><font face="Verdana">Storing Volatile Data in Session State
      </font></h3>
    <b><a href="customize.aspx">Customize This Page</a></b><p>
    Imagine some content here ...<br>
  </body>
</html>
```

customize.aspx

```
<html>

  <script language="C#" runat="server">

      void Page_Load(Object sender, EventArgs E) {

          if (!Page.IsPostBack) {
            ViewState["Referer"] = Request.Headers["Referer"];
            BackColor.Value = (String)Session["BackColor"];
            ForeColor.Value = (String)Session["ForeColor"];
            LinkColor.Value = (String)Session["LinkColor"];
            FontSize.Value = (String)Session["FontSize"];
            FontName.Value = (String)Session["FontName"];
          }
      }

      void Submit_Click(Object sender, EventArgs E) {
        Session["BackColor"] = BackColor.Value;
        Session["ForeColor"] = ForeColor.Value;
        Session["LinkColor"] = LinkColor.Value;
        Session["FontSize"]  = FontSize.Value;
        Session["FontName"]  = FontName.Value;

        if ( ViewState["Referer"] != null ){
          Response.Redirect(ViewState["Referer"].ToString());
        }
      }

      void Cancel_Click(Object sender, EventArgs E) {
        if ( ViewState["Referer"] != null ){
          Response.Redirect(ViewState["Referer"].ToString());
        }
      }

      String GetStyle(String key) {
        return Session[key].ToString();
      }
  </script>
```

# Application state - session

**Storing Volatile Data in Session State**

Customize This Page

Imagine some content here ...
Imagine some content here ...
Imagine some content here ...
Imagine some content here ...
Imagine some content here ...
Imagine some content here ...
Imagine some content here ...
Imagine some content here ...

**Customize This Page**

**Select Your Preferences:**

| | |
|---|---|
| Background Color: | beige |
| Foreground Color: | black |
| Hyperlink Color: | blue |
| Font Size: | 8pt |
| Font Name: | verdana |

Cancel Submit

# Cookies1.aspx Application state - cookies

```
<html>
  <script language="C#" runat="server">
      void Page_Load(Object sender, EventArgs E) {
        if (Request.Cookies["preferences1"] == null) {
          HttpCookie cookie = new HttpCookie("preferences1");
          cookie.Values.Add("ForeColor","black");
          cookie.Values.Add("BackColor","beige");
          cookie.Values.Add("LinkColor","blue");
          cookie.Values.Add("FontSize","8pt");
          cookie.Values.Add("FontName","Verdana");
          Response.AppendCookie(cookie);
        }
      }
      protected String GetStyle(String key) {
        HttpCookie cookie = Request.Cookies["preferences1"];
        if (cookie != null) {
          switch (key) {
            case "ForeColor" :
              return cookie.Values["ForeColor"]; break;
            case "BackColor" :
              return cookie.Values["BackColor"]; break;
            case "LinkColor" :
              return cookie.Values["LinkColor"]; break;
            case "FontSize" :
              return cookie.Values["FontSize"]; break;
            case "FontName" :
              return cookie.Values["FontName"]; break;
          }
        }
        return "";
      }
  </script>
```

```
  <style>

    body {
      font: <%=GetStyle("FontSize")%> <%=GetStyle("FontName")%>;
      background-color: <%=GetStyle("BackColor")%>;
    }

    a { color: <%=GetStyle("LinkColor")%> }

  </style>

  <body style="color:<%=GetStyle("ForeColor")%>">

    <h3><font face="Verdana">Storing Volatile Data with
      Client-Side Cookies</font></h3>

    <b><a href="customize.aspx">Customize This Page</a></b><p>

    Imagine some content here ...<br>

  </body>

</html>
```

Customize.aspx

```
<html>
  <script language="C#" runat="server">
      void Page_Load(Object sender, EventArgs E) {
        if (!IsPostBack){
          HttpCookie cookie = Request.Cookies["preferences1"];
          ViewState["Referer"] = Request.Headers["Referer"];

          if ( cookie != null ){
            BackColor.Value = (String)cookie.Values["BackColor"];
            ForeColor.Value = (String)cookie.Values["ForeColor"];
            LinkColor.Value = (String)cookie.Values["LinkColor"];
            FontSize.Value = (String)cookie.Values["FontSize"];
            FontName.Value = (String)cookie.Values["FontName"];
          }
        }
      }

      void Submit_Click(Object sender, EventArgs E) {

        HttpCookie cookie = new HttpCookie("preferences1");
        cookie.Values.Add("ForeColor",ForeColor.Value);
        cookie.Values.Add("BackColor",BackColor.Value);
        cookie.Values.Add("LinkColor",LinkColor.Value);
        cookie.Values.Add("FontSize",FontSize.Value);
        cookie.Values.Add("FontName",FontName.Value);
        Response.AppendCookie(cookie);

        if ( ViewState["Referer"] != null ){
          Response.Redirect(ViewState["Referer"].ToString());
        }
      }

      void Cancel_Click(Object sender, EventArgs E) {
        if ( ViewState["Referer"] != null ){
          Response.Redirect(ViewState["Referer"].ToString());
        }
      }
```

# Page class

- System.Object
  - System.Web.UI.Control
    - System.Web.UI.TemplateControl
      - System.Web.UI.Page
- `public class Page : TemplateControl, IHttpHandler`
- Related to .aspx files
- Access to properties such as
  - Application
  - ErrorPage
  - IsPostBack
  - IsValid
  - Request
  - Response
  - Session

# HttpRequest, HttpResponse classes

- System.Object
  - System.Web.HttpRequest
- `public sealed class HttpRequest`
- System.Object
  - System.Web.HttpResponse
- `public sealed class HttpResponse`
- Properties:
  - `public TextWriter Output {get;}`
  - `public Stream OutputStream {get;}`
  - `public string ContentType {get; set;}`
  - `public Encoding ContentEncoding {get; set;}`
- Methods:
  - `public void Redirect(string url);`
  - `public void End();`
  - `public void WriteFile(string filename);`

```
rivate void Page_Load(object sender, System.EventArgs e)
  {
        //Set the appropriate ContentType.
    Response.ContentType = "Application/pdf";
        //Get the physical path to the file.
    string FilePath = MapPath("acrobat.pdf");
        //Write the file directly to the HTTP content output strea
    Response.WriteFile(FilePath);
    Response.End();
  }
```

# Configuration

- Configuration files are placed with the application
- They are XML files
- Configuration changes are automatically detected, no need to restart
- Config file is named `web.config` and configures the folder it is placed in (exception: `machine.config` – for the server as a whole)
- Configurations are inherited in subfolders

# User configuration parameters

- Placed in `appSettings` section
- Written as key-value pairs
- Can be accessed using `ConfigurationSettings` class:
  - static `AppSettings` property
  - `NameValueCollection` collection, which contain key-value pairs and can be accessed using iterators or keys

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<html>

<script language="C#" runat="server">

    void Page_Load(Object Src, EventArgs E ) {

        String dsn = ConfigurationSettings.AppSettings["pubs"];

        SqlConnection myConnection = new SqlConnection(dsn);
        SqlDataAdapter myCommand = new SqlDataAdapter
          ("select * from Authors", myConnection);

        DataSet ds = new DataSet();
        myCommand.Fill(ds, "Authors");

        MyDataGrid.DataSource=new DataView(ds.Tables[0]);
        MyDataGrid.DataBind();
    }

</script>

<body>


</body>
</html>
```

web.config

```
<configuration>
  <appSettings>
    <add key="pubs" value="server=(local)\NetSDK;database=pubs;
      Integrated Security=SSPI" />
  </appSettings>
</configuration>
```