ITP281
E-Business & Project

C# Programming Fundamentals

Objectives

- Learn the fundamentals of C# programming and how to apply them in creating Web-based ASP.NET applications.
- This topic assumes you are already familiar with Java programming, and will focus on the features that are unique or different in C#.
- For a more thorough examination of the C# programming language, refer to the C# Programming Guide.

Built-in Types

C# Type	.NET Framework Type
bool	System.Boolean
byte	System.Byte
sbyte	System.SByte
char	System.Char
decimal	System.Decimal
double	System.Double
float	System.Single
int	System.Int32

Built-in Types, cont.

C# Type	.NET Framework Type
uint	System.UInt32
long	System.Int64
ulong	System.UInt64
object	System.Object
short	System.Int16
ushort	System.UInt16
string	System.String

Strings

- In C#, the **string** keyword is an alias for String. Therefore, **String** and **string** are equivalent, and you can use whichever naming convention you prefer.
- Similar to the String class in Java, the C# String class provides many methods for safely creating, manipulating, and comparing strings.
- In addition, the C# language overloads some operators to simplify common string operations.

The ToString() Method

- The ToString() method converts an object or built-in type to its string representation so that it is suitable for display.
- For example, to display an integer value as a string in a web page:

```
int Total = 0;
.
.
.
this.Label1.Text = Total.ToString();
```

Arrays

- Arrays in C# are very similar to arrays in Java, in that they are objects with properties and methods, and are declared in the same way. Eg:

```
int[] myArray = new int[5];
int[] myArray2 = { 1, 2, 3, 4, 5 };
```

- Typical code to iterate an array:

```
int Total = 0;

for (int i = 0; i < myArray.Length; i++)
{
     Total += myArray[i];
}
```

Using the foreach Loop

- C# also has the foreach statement, which provides a simple, clean way to iterate through the elements of an array:

```
int[] myArray = { 1, 2, 3, 4, 5 };
int Total = 0;
foreach (int n in myArray)
{
    Total += n;
}
```

Classes

- As with Java, classes in C# are declared by using the class keyword, as shown in the following example:

```
public class Customer
{
    // Fields, properties, methods and
    // events go here...
}
```

- Objects are created by using the new keyword followed by the name of the class that the object will be based on:

```
Customer object1 = new Customer();
```

Class Inheritance

- To inherit from a base class in C#, you append a colon and the base class name (the colon replaces the **extends** keyword in Java):

```
public class Manager : Employee
{
    // Employee fields, properties, methods
    // and events are inherited
    // New Manager fields, properties,
    // methods and events go here...
}
```

Access Modifiers

- Similar to Java, access modifiers are used when declaring classes, properties and methods to specify accessibility and enable encapsulation:
  - public - The type or member can be accessed by any other code in the same assembly or another assembly that references it.
  - private - The type or member can be accessed only by code in the same class or struct.
  - protected - The type or member can be accessed only by code in the same class or struct, or in a class that is derived from that class.
  - internal - The type or member can be accessed by any code in the same assembly, but not from another assembly.

Namespaces

- Namespaces are used heavily in C# programming to organize classes and avoid naming conflicts.
- In the following example, **System** is a namespace and **Console** is a class in that namespace:

```
System.Console.WriteLine("Hello World!");
```

Namespaces

- The **using** keyword can be used so that the complete name is not required, as in the following Example:

```
using System;
.
.
Console.WriteLine("Hello");
```

Namespaces

- You can also declare your own namespaces to help you control the scope of class and method names in larger programming projects:

```
namespace SampleNamespace
{
    class SampleClass
    {
            public void SampleMethod()
            {
                    System.Console.WriteLine(
                    "SampleMethod inside
                    SampleNamespace");
            }
    }
}
```

Session Variables (ASP.NET)

- Due to the stateless nature of the Web, a problem that often comes up is persistence of values between page reloads.
- In C#, we may utilize the HttpSessionState object to persist values within a browser session. Eg:

```
int click_count = Convert.ToInt32(Session["ClickCount"]);
click_count++;
this.Label1.Text = click_count.ToString();
Session["ClickCount"] = click_count.ToString();
```

Session Variables

- There are some issues with using session variables in this way:
  - Session state can expire (by default, after 20 minutes of inactivity), and the information that you store there can be lost.
  - Only strings can be stored as session variables, so conversion is often necessary.
- For more robust options, see ASP.NET State Management Recommendations.

Summary

- In this topic, we have covered the basics of C# programming, as related to Java, which shares the same syntax.
- There are many advanced features of C# which we have not covered, such as Interfaces, Indexers, Delegates, Iterators, etc., but for most web application development purposes, the basics will be sufficient. To learn more, visit the C# Programming Guide.

~The End~

Thank You!