Module 1

Review of Visual C# Syntax

Module Overview

- Overview of Writing Application by Using Visual C#
- Data Types, Operators, and Expressions
- Visual C# Programming Language Constructs

Lesson 1: Overview of Writing Application by Using Visual C#

- What Is the .NET Framework?
- Key Features of Visual Studio 2017
- Templates in Visual Studio 2017
- Creating a .NET Framework Application
- Overview of XAML

What Is the .NET Framework?

- CLR
  - Robust and secure environment for your managed code
  - Memory management
  - Multithreading
- Class library
  - Foundation of common functionality
  - Extensible
- Development frameworks
  - WPF
  - Universal Windows Platform
  - ASP.NET

Key Features of Visual Studio 2017

- Intuitive IDE
- Rapid application development
- Server and data access
- IIS Express
- Debugging features
- Error handling
- Help and documentation

Templates in Visual Studio 2017

- Console Application
- WPF Application
- Universal Windows Platform (UWP)
- Class Library
- ASP.NET Web Application
- ASP.NET MVC 4 Application
- WCF Service Application

Creating a .NET Framework Application

1. In Visual Studio, on the **File** menu, point to **New**, and then click **Project**.
2. In the **New Project dialog** box, choose a template, location, name, and then click **OK**.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleApplication1
{
   class Program
   {
       static void Main(string[] args) { }
   }
}
```

Overview of XAML

- XML-based language for declaring UIs
- Uses elements to define controls
- Uses attributes to define properties of controls

```
<Label Content="Name:" />

<TextBox Text="" Height="23" Width="120" />

<Button Content="Click Me!" Width="75" />
```

Lesson 2: Data Types, Operators, and Expressions

- What are Data Types?
- Expressions and Operators in Visual C#
- Declaring and Assigning Variables
- Accessing Type Members
- Casting Between Data Types
- Manipulating Strings

What are Data Types?

- int – whole numbers
- long – whole numbers (bigger range)
- float – floating-point numbers
- double - double precision
- decimal - monetary values
- char - single character
- bool - Boolean
- DateTime - moments in time
- string - sequence of characters

Expressions and Operators in Visual C#

Example expressions:

- + operator

```
a + 1
```

- / operator

```
5 / 2
```

- + and – operators

```
a + b - 2
```

- + operator (string concatenation)

```
"ApplicationName: " + appName.ToString()
```

Declaring and Assigning Variables

- Declaring variables:

```
int price;
// OR
int price, tax;
```

- Assigning variables:

```
price = 10;
// OR
int price = 10;
```

- Implicitly typed variables:

```
var price = 20;
```

- Instantiating object variables by using the **new** operator

```
ServiceConfiguration config = new ServiceConfiguration();
```

Accessing Type Members

- Invoke instance members

```
<instanceName>.<memberName>
```

- Example:

```
var config = new ServiceConfiguration();

// Invoke the LoadConfiguration method.
config.LoadConfiguration();

// Get the value from the ApplicationName property.
var applicationName = config.ApplicationName;

// Set the .DatabaseServerName property.
config.DatabaseServerName = "78.45.81.23";

// Invoke the SaveConfiguration method.
config.SaveConfiguration();
```

Casting Between Data Types

- Implicit conversion:

```
int a = 4;
long b = 5;
b = a;
```

- Explicit conversion:

```
int a = (int) b;
```

- **System.Convert** conversion:

```
string possibleInt = "1234";
int count = Convert.ToInt32(possibleInt);
```

Manipulating Strings

- Concatenating strings

```
StringBuilder address = new StringBuilder();
address.Append("23");
address.Append(", Main Street");
address.Append(", Buffalo");
string concatenatedAddress = address.ToString();
```

- Validating strings

```
var textToTest = "hell0 w0rld";
var regularExpression = "\\d";

var result = Regex.IsMatch(textToTest, regularExpression,
RegexOptions.None);

if (result)
{
   // Text matched expression.
}
```

Lesson 3: Visual C# Programming Language Constructs

- Implementing Conditional Logic
- Implementing Iteration Logic
- Creating and Using Arrays
- Referencing Namespaces
- Using Breakpoints in Visual Studio 2017
- Demonstration: Developing the Class Enrollment Application Lab

Implementing Conditional Logic

- **if** statements

```
if (response == "connection_failed") {. . .}
else if (response == "connection_error") {. . .}
else { }
```

- **select** statements

```
switch (response)
{
   case "connection_failed":
      . . .
      break;
   case "connection_success":
      . . .
      break;
   default:
      . . .
      break;
}
```

Implementing Iteration Logic

- **for** loop

```
for (int i = 0 ; i < 10; i++) { ... }
```

- **foreach** loop

```
string[] names = new string[10];
foreach (string name in names) { ... }
```

- **while** loop

```
bool dataToEnter = CheckIfUserWantsToEnterData();
while (dataToEnter)
{
    ...
    dataToEnter = CheckIfUserHasMoreData();
}
```

- **do** loop

```
do
{
    ...
    moreDataToEnter = CheckIfUserHasMoreData();
} while (moreDataToEnter);
```

Creating and Using Arrays

- C# supports:
  - Single-dimensional arrays
  - Multidimensional arrays
  - Jagged arrays
- Creating an array:

```
int[] arrayName = new int[10];
```

- Accessing data in an array:
  - By index

```
int result = arrayName[2];
```

  - In a loop

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

Referencing Namespaces

- Use namespaces to organize classes into a logically related hierarchy
- .NET class library includes:
  - **System.Windows**
  - **System.Data**
  - **System.Web**
- Define your own namespaces:

```
namespace FourthCoffee.Console
{
   class Program {. . .}
```

- Use namespaces:
  - Add reference to containing library
  - Add **using** directive to code file

Using Breakpoints in Visual Studio 2017

- Breakpoints enable you to view and modify the contents of variables:
  - Immediate Window
  - Autos, Locals, and Watch panes
- Debug menu and toolbar functions enable you to:
  - Start and stop debugging
  - Enter break mode
  - Restart the application
  - Step through code

Demonstration: Developing the Class Enrollment Application Lab

In this demonstration, you will learn about the tasks that you will perform in the lab for this module

Lab: Developing the Class Enrollment Application

- Exercise 1: Implementing Edit Functionality for the Students List
- Exercise 2: Implementing Insert Functionality for the Students List
- Exercise 3: Implementing Delete Functionality for the Students List
- Exercise 4: Displaying a Student's Age

Estimated Time: 70 minutes

Lab Scenario

You are a Visual C# developer working for a software development company that is writing applications for The School of Fine Arts, an elementary school for gifted children.

The school administrators require an application that they can use to enroll students in a class. The application must enable an administrator to add and remove students from classes, as well as to update the details of students.

You have been asked to write the code that implements the business logic for the application.

During the labs for the first two modules in this course, you will write code for this class enrollment application.

When The School of Fine Arts ask you to extend the application functionality, you realize that you will need to test proof of concept and obtain client feedback before writing the final application, so in the lab for Module 3, you will begin developing a prototype application and continue with this until then end of Module 8.
In the lab for Module 9, after gaining signoff for the final application, you will develop the user interface for the production version of the application, which you will work on for the remainder of the course.

Module Review and Takeaways

- Review Questions