Chapter 2

First C# Programs

EVALUATION

 Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

First C# Programs

Objectives

After completing this unit you will be able to:

- **Write a basic "Hello, World" program in C#.**
- **Compile and run C# programs in your local development environment.**
- **Describe the basic structure of C# programs.**
- **Describe how related C# classes can be grouped into namespaces.**
- **Use variables and simple expressions in C# programs.**
- **Write C# programs that can perform simple calculations.**
- **Perform simple input and output in C#.**
- **Describe objects and classes in C#.**
- **Use an input wrapper class to perform input in C#.**

EVALUATION

Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Hello, World

- **Whenever learning a new programming language, a good first step is to write and run a simple program that will display a single line of text.**
  - Such a program demonstrates the basic structure of the language, including output.
  - You must learn the pragmatics of compiling and running the program.
- **Here is "Hello, World" in C#:**
  - See **Demos\Hello\Hello.cs**, backed up in **Chap02\Hello**.

```
// Hello.cs

class Hello
{
    public static int Main(string[] args)
    {
        System.Console.WriteLine(
            "Hello, World");
        return 0;
    }
}
```

EVALUATION

 Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Program Structure

```
// Hello.cs

class Hello
{
    ...
}
```

- **Every C# program has at least one *class*.**
  - A class is the foundation of C#'s support for object-oriented programming.
  - A class encapsulates data (represented by **variables**) and behavior (represented by **methods**).
  - All of the code defining the class (its variables and methods) will be contained between the curly braces.
  - We will discuss classes in detail later.
- **Note the *comment* at the beginning of the program.**
  - A line beginning with a double slash is present only for documentation purposes and is ignored by the compiler.
- **C# files have the extension *.cs*.**

EVALUATION

Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Program Structure (Cont'd)

```
// Hello.cs

class Hello
{
    public static int Main(string[] args)
    {
        ...
        return 0;
    }
}
```

- **Every C# program has a distinguished class that has a method whose name must be *Main*.**
  - Note the capitalization!
  - The method should be **public** and **static**.
  - An **int** exit code can be returned to the operating system. Use **void** if you do not return an exit code.

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

  - Command line arguments are passed as an array of strings.
  - You may omit the command line arguments.

```
public static void Main()
```

  - The runtime will call this **Main** method – it is the entry point for the program.
  - All of the code for the **Main** method will be between the curly braces.

EVALUATION

 Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Program Structure (Cont'd)

```
// Hello.cs

class Hello
{
    public static int Main(string[] args)
    {
        System.Console.WriteLine(
            "Hello, World");
        return 0;
    }
}
```

- **Every method in C# has one or more *statements*.**
- **A statement is terminated by a semicolon.**
  - A statement may be spread out over several lines.
- **The *Console* class provides support for standard output and standard input.**
  - The method **WriteLine** displays a string, followed by a new line.

EVALUATION

 Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Namespaces

- **Much of the standard functionality in C# is provided through many classes in the .NET Framework.**
- **Related classes are grouped into *namespaces*.**
- **The fully-qualified name of a class is specified by the namespace, followed by a dot, followed by the class name.**

```
System.Console
```

- **A *using* statement allows a class to be referred to by its class name alone.**

```
// Hello2.cs

using System;

class Hello
{
    public static void Main()
    {
        Console.WriteLine("Hello, World");
    }
}
```

- **Note that in C# it is not necessary for the file name to be the same as the name of the class containing the *Main* method.**
- **This version of the program also illustrates a *Main()* method with no command-line arguments and void return type.**

EVALUATION

 Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Exercise

- **Take a few minutes to add two more lines of code to your program.**
  - Print out the phrase “My name is XXXX” (use your own name).
  - Then print out “Goodbye.”
- **Save your file, compile the program, and run it.**
  - Your output should be something like this:

```
Hello, World
My name is Bob
Goodbye
```

EVALUATION

Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Answer

```
// Hello3.cs

using System;

class Hello
{
    public static void Main()
    {
        Console.WriteLine("Hello, World");
        Console.WriteLine("My name is Bob");
        Console.WriteLine("Goodbye");
    }
}
```

EVALUATION

Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Variables

- **In C#, you can define *variables* to hold data.**
- **Variables represent storage locations in memory.**
- **In C#, variables are of a specific data *type*.**
  - Some common types are **int** for integers and **double** for floating point numbers.
  - You must declare variables before you can use them.
- **A variable declaration reserves memory space for the variable and may optionally specify an initial value.**

```
int kilo = 1024;      // reserves space and assigns
                      // an intial value
int mega;             // reserves space but does
                      // not initialize
```

  - If a variable is not initialized in its declaration, it should be assigned prior to being used.

```
int kilo;
kilo = 1024;
// Now you may use kilo
```

EVALUATION

Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Expressions

- **You can combine variables and constants (or "literals") via *operators* to form *expressions*.**
- **Examples of operators include the standard arithmetic operators:**

```
+        addition
-        subtraction
*        multiplication
/        division
```

- **Here are some examples of expressions:**

```
kilo * 1024
(fahrenheit - 32) * 5 / 9
3.1416 * radius * radius
```

EVALUATION

 Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Assignment

- **You can assign a value to a variable by using the = symbol.**
  - On the left hand side is a variable.
  - On the right hand side is an expression.
  - The expression is evaluated and its value is assigned to the variable on the left.
  - Assignment is a statement and must be terminated by a semicolon.

```
mega = kilo * 1024;
celsius = (fahrenheit - 32) * 5 / 9;
area = 3.1416 * radius * radius;
```

- **Note that the same variable can be used on both sides of an assignment statement.**

```
int item = 5;
int total = 30;
total = total + item;
```

  - The expression **total + item** evaluates to 35, using the old value of **total**, and this value is assigned to **total**, creating a new value.

EVALUATION

Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Calculations Using C#

- **You can easily use C# to perform calculations by adding code to the *Main* method of a C# class.**
  - Declare whatever variables you need.
  - Create expressions and assign values to your variables.
  - Print out the answer using **Console.WriteLine()**.
- **You can easily perform labeled output, relying on two features of C#:**
  - The operator + performs concatenation for **string** data.
  - There is an automatic, implicit conversion available that converts numeric data to string data when required.
  - Hence this code ...

```
int total = 35;
System.Console.WriteLine("The total is " + total);
```

  - ... will produce this output:

```
The total is 35
```

EVALUATION

 Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Sample Program

- **This program will convert temperature from Fahrenheit to Celsius.**
  - See **Convert\Step1**.

```
// Convert.cs - Step 1
//
// Program converts a hardcoded temperature in
// Fahrenheit to Celsius

using System;

class Convert
{
    public static void Main(string[] args)
    {
        int fahr = 86;
        int celsius = (fahr - 32) * 5 / 9;
        Console.WriteLine("fahrenheit = " + fahr);
        Console.WriteLine("celsius = " + celsius);
    }
}
```

EVALUATION

 Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

More about Output in C#

- **The *Console* class in the *System* namespace supports two simple methods for performing output:**
  - **WriteLine()** writes out a string followed by a new line.
  - **Write()** writes out just the string without the new line.

```
int x = 24;
int y = 5;
int z = x * y;
Console.Write("Product of " + x + " and " + y);
Console.WriteLine(" is " + z);
Console.WriteLine("The product is {0}", z);
```

  - The output is all on one line:

```
Product of 24 and 5 is 120
```

- **A more convenient way to build up an output string is to use *placeholders* {0}, {1}, etc.**
  - An equivalent way to do the output shown above is:

```
Console.WriteLine("Product of {0} and {1} is {2}",
   x, y, z);
```

  - The program **OutputDemo** illustrates the output operations just discussed.
  - Later in the course we will see how to control formatting of output, and occasionally in examples we will throw in some simple use of formatting.

EVALUATION

Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Input in C#

- **Our first Convert program is not too useful, because the Fahrenheit temperature is hard-coded.**
  - To convert a different temperature, you would have to edit the source file and recompile.
- **What we really want to do is allow the user of the program to enter a value at runtime for the Fahrenheit temperature.**
- **Although simple console input in C# is fairly easy, we can make it even easier using object-oriented programming.**
  - We can encapsulate or "wrap" the details of input in a class.
  - It will be easy to use the wrapper class.

EVALUATION

Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

More about Classes

- **Although we will discuss classes in more detail later, there is a little more you need to know now.**
- **A class can be thought of as a template for creating objects.**
  - An **object** is an instance of a **class**.
- **A class specifies data and behavior.**
  - The data is different for each object instance.
- **In C#, you instantiate a class by using the *new* keyword.**

```
InputWrapper iw = new InputWrapper();
```

  - This code creates the object instance **iw** of the **InputWrapper** class.

EVALUATION

Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

InputWrapper Class

- **The *InputWrapper* class "wraps" interactive input for several basic data types.**
  - The supported data types are **int**, **double**, **decimal** and **string**.
  - Methods **getInt**, **getDouble**, **getDecimal** and **getString** are provided.
  - A prompt string is passed as an input parameter.
  - See files **InputWrapper.cs** in directory **TestInputWrapper,** which implements the class, and **TestInputWrapper.cs,** which tests the class.
- **Although the code is quite short, it is a little complex, involving a number of different methods from different .NET Framework classes.**
- **However, you do not need to be familiar with the implementation of *InputWrapper* in order to use it.**
  - That is the beauty of "encapsulation"—complex functionality can be hidden by an easy-to-use interface.

EVALUATION

 Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Echo Program

- **We illustrate interactive input by a simple "echo" program.**
  - The program prompts the user for a name, and then prints out a personalized greeting.
  - See **Echo**.
- **This directory has two files, each defining a class.**
  - **InputWrapper.cs** defines the wrapper class. There is no **Main** method in this class.
  - **EchoName.cs** has a class **Echo,** with a **Main** method.

```
// EchoName.cs
//
// Prompts user to enter name and then
// prints out greeting using name

using System;

class Echo
{
 public static void Main(string[] args)
 {
   InputWrapper iw = new InputWrapper();
   string name = iw.getString("Enter your name: ");
   Console.WriteLine("Hello, " + name);
 }
}
```

EVALUATION

 Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Using InputWrapper

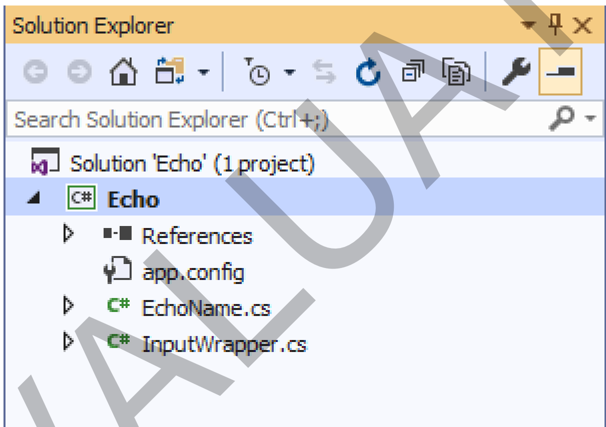
- **The bolded statements illustrate how to use the *InputWrapper* class.**
  - Instantiate an **InputWrapper** object **iw** by using **new**.
  - Prompt to obtain input data by calling the appropriate **getXXX** method.

EVALUATION

Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Multiple Files in Visual Studio

- **It is very easy to work with Visual Studio projects that have multiple files.**
- **As an example, open the Visual Studio *solution* that is specified by the file *Chap02\Echo\Echo.sln*.**
- **A solution can contain one or more *projects*.**
  - In this case there is a single project file **Echo.csproj**.
- **You can see all the files in a solution through the *Solution Explorer*.**



- **When you build the solution via the toolbar button , you will build all the projects in the solution.**
  - You can also build just the current project via the toolbar button .

EVALUATION

Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

The .NET Framework

- **The .NET Framework has a very large class library (several thousand classes).**
- **To make all of this functionality more manageable, the classes are partitioned into *namespaces*.**
- **The root namespace is *System*, which directly contains many useful classes, among them:**
  - **Console** provides access to standard input, output and error streams for I/O.
  - **Convert** provides conversions among base data types.
  - **Math** provides mathematical constants and functions.
- **This courses uses .NET Framework 4.7.2, the latest released version of the .NET Framework at the time Visual Studio 2019 was released.**

EVALUATION

Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

The .NET Framework (Cont’d)

- **Underneath *System*, there are other namespaces, among them:**
  - **System.Data** contains classes constituting the ADO.NET architecture for accessing databases.
  - **System.Xml** provides standards-based support for processing XML.
  - **System.Drawing** contains classes providing GDI+ graphics functionality.
  - **System.Windows.Forms** provides support for creating applications with rich Windows-based interfaces.
  - **System.Web** provides support for browser/server communication.
  - **System.IO** provides support for reading and writing with streams and files. Both synchronous and asynchronous I/O are supported.
  - **System.Net** provides support for several standard network protocols.

EVALUATION

Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Lab 2

C# Programs for Calculation

In this lab you modify or implement several C# programs to perform calculations. You need to perform input (through a wrapper class), perform a calculation, and output the result. Do as many of these exercises as time permits. If you have extra time, do some of the optional experiments suggested in some of the exercises, or make up some experiments on your own.

Detailed instructions are contained in the Lab 2 write-up at the end of the chapter.

Suggested time: 30 minutes

EVALUATION

Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Summary

- **Every C# application has a class with a method *Main*, which is the entry point into the application.**
- **The *System* class includes methods for performing output, such as *WriteLine*.**
- **Expressions in C# are formed from literals, variables and operators.**
- **With the assignment statement, you can assign a value computed by an expression to a variable.**
- **Input in C# is a little more complicated than output, but you can use a wrapper class that encapsulates the required C# classes and presents a simple programming interface.**
- **The .NET Framework has a large class library that is partitioned into namespaces.**

EVALUATION

Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Lab 2

C# Programs for Calculation

Introduction

In this lab, you modify or implement several C# programs to perform calculations. You need to perform input (through a wrapper class), perform a calculation, and output the result. Do as many of these exercises as time permits. If you have extra time, do some of the optional experiments suggested in some of the exercises, or make up some experiments on your own.

Suggested Time: 30 minutes

Root Directory: **OIC\CSharp**

Directories:

Labs\Lab2\Convert	(Exercise 1 work)
Chap02\Convert\Step1	(Backup of Exercise 1 starter files)
Chap02\Convert\Step2	(Answer to Exercise 1)
Chap02\TestInputWrapper	(**InputWrapper** class)
Labs\Lab2	(Exercise 2 work)
Chap02\Circle	(Exercise 2 answer)

Exercise 1. Fahrenheit to Celsius Conversion

Examine the code of the starter program. Build and run. Notice that the Fahrenheit temperature to be converted is hard-coded. Modify the program to prompt the user for a Fahrenheit temperature, read in the value entered by the user, and print out the result. Make use of the wrapper class **InputWrapper** that was discussed in this chapter.

The starter program uses **int** as the data type for temperatures. An optional experiment is to use **double** as the data type. Could you input the Fahrenheit temperature as an **int** and calculate the Celsius temperature as a **double**?

Exercise 2. Calculate the Area of a Circle

Use Visual Studio to create an empty C# project **Circle** in the **Lab2** folder. This will create the folder **Circle**. Add a new file **Circle.cs** to your project, where you will place your program code.

Write a C# program to calculate the area of a circle. Prompt the user for the radius, read in the value entered by the user, calculate the area of the circle, and print out the result. For pi, use the approximation 3.1416. What is an appropriate data type to use for radius and area?

EVALUATION

 Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

As an optional experiment, use the class **Math** (in the namespace **System**) for a more accurate value of pi.

Another optional experiment is to capture the output data in a file. This is easy to do at the command line, by using the **>** to "redirect" output to a file.

```
Circle > output.txt
```

When using this technique, the prompts are written to the output file and not displayed on the screen. Hence you need to know what to type! If you type "10" for the radius, your output file should look like:

```
radius: Using, 3.1416, area = 314.16
Using, Math.PI, area = 314.15926535897933
```

EVALUATION

Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved

Copyright © 2019 Object Innovations Enterprises, LLC
All Rights Reserved