COIS1020H: Programming for Computing Systems

Chapter 7
Using Methods

Understanding Methods and Implementation Hiding

- **Method**
  - Encapsulated series of statements that perform a task
  - Using a method is called **invoking** or **calling**

```
using System;
public class HelloClass
{
   public static void Main()
   {
      Console.WriteLine("Hello");
   }
}
```

Figure 7-1 The HelloClass program

2

Understanding Implementation Hiding

- **Implementation hiding**
  - An important principle of object-oriented programming
  - Keeps the details of a method's operations hidden
- Only concern is the way you **interface** or interact with the method
  - Program does not need to know how method works

3

Writing Methods with No Parameters and No Return Value

- Major reasons to create a method
  - Code will remain short and easy to follow
  - A method is easily *reusable*
- A method must include:
  - **Method declaration** (also called a **method header**)
  - Opening curly brace {
  - **Method body**
  - Closing curly brace }

4

Method Example

```
using System;
public class HelloClass
{
   public static void Main()
   {
      ShowWelcomeMessage();
      Console.WriteLine("Hello");
   }
   public static void ShowWelcomeMessage()
   {
      Console.WriteLine("Welcome");
      Console.WriteLine("It's a pleasure to serve you");
      Console.WriteLine("Enjoy the program");
   }
}
```

Figure 7-4 `HelloClass` program with `Main()` method calling the `ShowWelcomeMessage()` method

5

Writing Methods with No Parameters and No Return Value (cont'd.)

- Method declaration
  - Defines the rules for using the method and contains:
    - Optional declared accessibility
    - Optional `static` modifier
    - Return type for the method
    - Method name, or identifier
    - Opening parenthesis
    - Optional list of method parameters
    - Closing parenthesis

  Example:

```
public static void ShowWelcomeMessage()
```

6

Writing Methods with No Parameters and No Return Value (cont'd.)

- Optional declared **accessibility**
  - Limits how other methods can use your method
  - Possible access values
    - **public** – unlimited access to the method
    - **private** – limits access to the class that contains method
  - Not going to cover but also
    - ***Protected internal***
    - ***Protected***
    - ***Internal***

7

Writing Methods with No Parameters and No Return Value (cont'd.)

- You can declare a method to be **static** or **non-static**
  - Methods are non-static by default
- `static` method
  - Can be called without referring to an object
    - Instead, you refer to the class
  - Cannot call non-static methods
    - Non-static methods can call static ones (later)

8

Writing Methods with No Parameters and No Return Value (cont'd.)

- Every method has a **return type**
  - Indicates what kind of value the method will return to any other method that calls it
  - If a method does not return a value, its return type is `void`
- Method name must be a legal C# identifier

- **Parameter to a method**
  - Variable that holds data passed to a method when it is called (**interface**)

9

Writing Methods with No Parameters and No Return Value (cont'd.)

```
using System;
public class HelloClass
{
   public static void Main()
   {
      ShowWelcomeMessage();
      Console.WriteLine("Hello");
   }
   public static void ShowWelcomeMessage()
   {
      Console.WriteLine("Welcome");
      Console.WriteLine("It's a pleasure to serve you");
      Console.WriteLine("Enjoy the program");
   }
}
```

```
Welcome
It's a pleasure to serve you
Enjoy the program
Hello
```

Figure 7-4 `HelloClass` program with `Main()` method calling the `ShowWelcomeMessage()` method

10

Writing Methods That Requires a Parameter

- You need: - Type of the parameter
  - A local identifier (name) for the parameter
- **Local variable**
  - Declared within a method
- **Formal parameter**
  - Parameter in the method header that accepts a value
- **Actual parameters**
  - Arguments within a method *call or invocation*

Writing Methods That Requires a Parameter

Formal Parameter

```
public static void DisplaySalesTax(double saleAmount)
{
   double tax;
   const double RATE = 0.07;
   tax = saleAmount * RATE;
   Console.WriteLine("The tax on {0} is {1}",
      saleAmount.ToString("C"), tax.ToString("C"));
}
```

Figure 7-6 The `DisplaySalesTax()` method

Writing Methods That Requires a Parameter

```
using System;
public class UseTaxMethod
{
   public static void Main()
   {
      double myPurchase = 12.99;
      DisplaySalesTax(myPurchase);
      DisplaySalesTax(35.67);
   }
   public static void DisplaySalesTax(double saleAmount)
   {
      double tax;
      const double RATE = 0.07;
      tax = saleAmount * RATE;
      Console.WriteLine("The tax on {0} is {1}",
         saleAmount.ToString("C"), tax.ToString("C"));
   }
}
```

The tax on $12.99 is $0.91
The tax on $35.67 Is $$2.50

Actual Parameters

Figure 7-7 Complete program using the `DisplaySalesTax()` method two times

13

Writing Methods That Require Multiple Arguments

- Methods can take any number of parameters
- When you call the method, arguments must match
  - In both number and type

```
public static void DisplaySalesTax(double saleAmount, double taxRate)
{
   double tax;
   tax = saleAmount * taxRate;
   Console.WriteLine("The tax on {0} at {1} is {2}",
      saleAmount.ToString("C"),
      taxRate.ToString("P"), tax.ToString("C"));
}
```

Figure 7-9 The `DisplaySalesTax()` method that takes two arguments

  - To invoke it, in `Main()` use

```
DisplaySalesTax(200, 0.10);
```

14

Writing a Method That Returns a Value

- A method can return, at most, one value through its name to a method that calls it
  - The data type of the return value is the same as the data type of the Method
- **`return` statement**
  - Causes a value to be sent back to the calling method
- You are not required to use a returned value
  - it can be stored in a variable or used directly

15

Writing a Method That Returns a Value (cont'd.)

```
public static double CalcPay(double hours, double rate)
{
   double gross;
   gross = hours * rate;
   return gross;
}
```

Figure 7-10 The `CalcPay()` method

- The data type of the return value `gross` is `double` which matches the return type of the method `CalcPay`
- In `Main()`

```
Result = CalcPay(35, 12.50);  // or
Console.WriteLine("The gross pay is {0:C},
                  CalcPay(35, 12.50));
```

16

Writing a Method That Returns a Value (cont'd.)

```
using System;
public class UseCalcPay
{
   public static void Main()
   {
      double myHours = 37.5;
      double myRate = 12.75;
      double grossPay;
      grossPay = CalcPay(myHours, myRate);
      Console.WriteLine("I worked {0} hours at {1} per hour",
         myHours, myRate);
      Console.WriteLine("My gross pay is {0}",
         grossPay.ToString("C"));
   }
   public static double CalcPay(double hours, double rate)
   {
      double gross;
      gross = hours * rate;
      return gross;
   }
}
```

```
I worked 37.5 hours at 12.75 per hour
My gross pay is $478.13
```

Figure 7-11 Program using the `CalcPay()` method

Nested Method Calls

- A **nested** method call is when a method calls another method
  - Method calls placed inside other method calls

```
// This program demonstrates nested method calls
using System;
public class DeepAndDeeper
{
  public static void Main()
  {
    Console.WriteLine("I am starting in Main.");
    Deep();
    Console.WriteLine("NowI am back in Main.");
}
```

```
I am starting in Main
I am now in Deep
I am in Deeper
Now I am back in Deep
Now I am back in Main
```

```
 // Deep displays message and calls Deeper
 public static void Deep()
 {
    Console.WriteLine("I am now in Deep.");
    Deeper();
    Console.WriteLine("Now I am back in Deep.");
 }

 // The Deeper method displays a message
 public static void Deeper()
 {
    Console.WriteLine("I am now in Deeper.");
 }
}
```

Alternate Ways to Write a `Main()` Method Header

- Conventional way

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

- Passing command-line arguments

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

- Some programmers prefer to write `Main()` method headers with a return type of `int` instead of `void`
  - Last statement in the `Main()` method must be a return statement

19

Summary

- Method is a series of statements that perform a task
- Some methods require passed-in information called arguments or parameters
- You write methods to make programs easier to understand and so that you can easily reuse them
- Object-oriented programs hide their methods' implementation
- You can pass multiple arguments to a method

20

Summary (cont'd.)

- The return type for a method can be any type used in the C# programming language
- You might see different `Main()` method headers in other books or in programs written by others