Methods

ISYS2001 Lecture 7, Department of Information Systems

Image Source: http://img0.liveinternet.ru/images/attach/c/5/85/779/85779218_g6hR29OI.jpg

Last Week

- Use files for data storage
- Use the OpenFileDialog Control
- Use the SaveFileDialog Control
- Generate Random Numbers
- Create a Load event handler

Today you will be able to…

- Introduction to Methods
- void Methods
- Passing Arguments to Methods
- Passing Arguments by Reference
- Value-Returning Methods

Introduction to Methods

- Methods can be used to break a complex program into small, manageable pieces
  - This approach is known as divide and conquer
  - In general terms, breaking down a program to smaller units of code, such as methods, is known as modularization
- Two types of methods are:
  - A void method simply executes a group of statements and then terminates
  - A value-returning method returns a value to the statement that called it

Example

- Using one long sequence of statement to perform a task
- Using method to divide and conquer a problem

```
Namespace Example
{
 public partial class Form1 : Form
 {
  private void myButton_Click(object sender, EventArgs e)
  {
    statement;
    statement;
    statement;
    statement;
    ………….
  }
 }
}
```

```
Namespace Example
{
 public partial class Form1 : Form
 {
  private void myButton_Click(object sender, EventArgs e)
  {
    Method2();
    Method3();
    ………….
  }

  private void Method2();
  {
    Statements;
  }

 private void Method3();
  {
    Statements;
  }
 }
}
```

void Methods

- A void method simply executes the statement it contains and then terminates. It does not return any value to the statement that called it
- To create a method you write its definitions
- A method definition has two parts:
    - header: the method header appears at the beginning of a method definition to indicate access mode, return type, and method name
    - body: the method body is a collection of statements that are performed when the method is executed

The Method Header

- The book separates a method header into four parts :
  - Access modifier: keywords that defines the access control
    - private: a private method can be called only by code inside the same class as the method
    - public: a public method can be called by code that is outside the class.
  - Return type: specifies whether or not a method returns a value
  - Method name: the identifier of the method; must be unique in a given program. This book uses Pascal case (aka camelCase)
  - Parentheses: A method's name is always followed by a pair of parentheses

Access modifier **Return type** **Method name** **Parentheses**

```
private void DisplayMessage()
{
  MessageBox.Show("This is the DisplayMessage method.");
}
```

Declaring Method Inside a Class

- Methods usually belong to a class
- All Visual C# methods typically belong to applications’ default Form1 class
- In this book, methods are created inside the Form1 class

```
using System;
using …..

namespace Example
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        // your method definition will appear here inside Form1 class
    }
}
```

Calling a Method

- A method executes when it is called
- Event handlers are called when specific events take place. Yet, methods are executed by method call statements.
- A method call statement is the name of the method followed by a pair of parentheses.

```
private void goButton_Click(object sender, EventArgs e)
{
  MessageBox.Show(“This is the goButton_Click method.”);
  DisplayMessage();
}

private DisplayMessage()
{
  MessageBox.Show(“This is the DisplayMessage method.”);
}
```

Concept of Return Point

- When calling a method the system needs to know where the program should return after the method ends
- The system saves the memory address of the location called return point to which it should return
- The system jumps to the method and executes the statements in its body
- When the method ends, the system jumps back to the return point and resumes execution

Top-Down Design

- To modularize a program, programmers commonly use a technique known as top-down design
- It breaks down an algorithm to methods
- The process is performed in the following manner:
  - The overall task that the program is to perform is broken down into a series of subtasks
  - Each subtask is examined to determine whether it can be further broken down into more subtasks. This step is repeated until no more subtasks can be identified
  - Once all the subtasks have been identified, they are written in code

Passing Arguments to Methods

- An argument is any piece of data that is passed into a method when the method is called In the following, the statement calls the MessageBox.Show method and passes the string “Hello” as an argument:

```
MessageBox.Show(“Hello”);
```

- A parameter is a variable that receives an argument that is passed into a method. In the following, value is an int parameter:

```
private void DisplayValue(int value)
{
  MessageBox.Show(value.ToString());
}
```

An example of a call to DisplayValue method with 5 as parameter is:

```
DisplayValue(5);
```

Contents of Variables as Arguments

- You can pass the contents of variables as arguments. For example:

```
private void DisplayValue(int value)
{
  MessageBox.Show(value.ToString());
}

int x = 5;
DisplayValue(x);
DisplayValue(x * 4);
```

- value is an in parameter in DisplayValue method
- In this example, x is an int variable with a value 5. Its contents are passed as argument.
- The expression x * 4 also produces an int results, which can also be passed as argument
- Another example is:

    DisplayValue(int.Parse(“700”));

Argument and Parameter Data Type Compatibility

- An argument’s data type must be assignment compatible with the receiving parameter’s data type
- Basically,
  - You can pass only string arguments into string parameter
  - You can pass int arguments into int parameters, but you cannot pass double or decimal arguments into int parameters
  - You can pass either double or int arguments to double parameters, but you cannot pass decimal values to double parameters
  - You can pass either decimal or int arguments to decimal parameters, but you cannot pass double arguments into decimal parameters

Passing Multiple Arguments

- You can pass more than one argument to a method

```
private void showButton1_Click(object sender, EventArgs e)
{
  ShowMax(5, 10);
}

private void showButton2_Click(object sender, EventArgs e)
{
  int value1 = 2;
  int value2 = 3;
  ShowMax(value1, value2);
}

private void ShowMax(int num1, int num2) { }
```

Named Arguments

- C# allows you to specify which parameter an argument should be passed into. The syntax is:

  parameterName : value
- An argument that is written using this syntax is known as a named argument

```
private void showButton_Click(object sender, EventArgs e)
{
  showName(lastName : “Smith”, firstName : “Suzanne”);
}
private void ShowName(string firstName, string lastName)
{
  MessageBox.Show(firstName + “ “ + lastNmae);
}
```

- Notice that you get the same result if the call statement is:

```
showName(“Suzanne”, “Smith”);
```

Default Arguments

- C# allows you to provide a default argument for a method parameter

```
private void ShowTax(decimal price, decimal taxRate = 0.07m)
{
 decimal tax = price * taxRate;
}
```

- The value of taxRate is defaulted to 0.07m. You can simply call the method by passing only the price

```
showTax(100.0m);
```

- You can also override the default argument

```
showTax(100.0m, 0.08m);
```

Passing Arguments by Reference

- A reference parameter is a special type of parameter that does not receive a copy of the argument's value
- It becomes a reference to the argument that was passed into it
- When an argument is passed by reference to a method, the method can change the value of the argument in the calling part of the program
- In C#, you declare a reference parameter by writing the ref keyword before the parameter variable's data type

```
private void SetToZero(ref int number)
{
  number =0;
}
```

- To call a method that has a reference parameter, you also use the keyword ref before the argument

```
int myVar = 99;
SetToZero(ref myVar);
```

Using Output Parameters

- An output parameter works like a reference parameter with the following differences:

  An argument does not have to be a value before it is passed into an output parameter

  A method that has an output parameter must be the output parameter to some value before it finishes executing

- In C#, you declare an output parameter by writing the out keyword before the parameter variable's data type

```
private void SetToZero(out in number)
{
 number = 0;
}
```

- To call a method that has a output parameter, you also use the keyword out before the argument

```
int myVar;
SetToZero(out myVar);
```

Value-Returning Methods

- A value-returning method is a method that returns a value to the part of the program that called it
- A value-returning method is like a void method in the following ways:

    It contains a group of statements that performs a specific task

    When you want to execute the method, you call it
- The .NET Framework provide many value-returning methods, for example, the int.Parse method that accepts a string and returns an int value

$$\texttt{int number = }\underbrace{\texttt{int.Parse(}\overset{\text{argument}}{\downarrow}\texttt{“100”)}}_{\text{Method call}}\texttt{;}$$

Write Your Own Value-Returning Functions

- In C# the generic format is:

```
AccessModifier DataType MethodName(ParameterList)
{
  statement(s);
  return expression;
}
```

- AccessModifier: private or public
- DataType: int, double, decimal, string, and Boolean
- MethodName: the identifier of the method; must be unique in a program
- ParameterList: an optional list of parameter
- Expression: can be any value, variable, or expression that has a value

The Return Statement

- There must be a return statement inside the method which is usually the last statement of the method. This return statement is used to return a value to the statement that called the method. For example,

```
private int sum(int num1, int num2)
{
  return num1 + num2;
}
```

- Notice that the returned value and the method's type must match. In the above example, the method is an int method, so it can only return int value

Sample Codes

```
// int type
private int Sum(int num1, int num2)
{
  return num1 + num2;
}

// double type
private double Sum(double num1, double num2)
{
  return num1 + num2;
}

// decimal type
private decimal Sum(decimal num1, decimal num2)
{
  return num1 + num2;
}
```

Return Values to Variables

- A value-returning method returns a value with specific type. However, the method no longer keeps the value once it is returned.
- You can declare a variable to hold the returned value to use the value over and over again

```
int combinedAge = Sum (userAge, friendAge);

private int Sum(int num1, int num2)
{
  return num1 + num2;
}
```

- After execution, the value is kept in combinedAge variable

Boolean Methods

- A Boolean method returns either true or false. You can use a Boolean method to test a condition

```
private bool IsEven(int number)
{
   bool numberIsEven;
   if (number % 2 == 0)
   {
      numberIsEven = true;
   }
   else
   {
      numberIsEven = false;
   }
   return numberIsEven;
}
```

With this code, an int value assigned to the number parameter will be evaluated by the if statement
The return statement will return either true or false

Using Modulus Operator in Boolean Expressions

- The book discusses the use of modulus operator to determine if a whole number is odd or even

```
number % 2
```

- The modulus operator is a useful tool to write Boolean expression. When a number modulus 2, there are only two possible outcomes: 0 and 1

```
switch (number % 2)
{
 case 0: numberIsEven = true; break;
 case 1: numberIsEven = false; break;
 // default is not needed in this case
}
```

Returning a String from a Method

- String is a primitive data type. A C# value-returning method can return a string to the statement that called it. For example,

```
private string FullName(string first, string middle, string last)
{
   return first + “ “ + middle + “ “ + last;
}
```

- A sample statement to call it is:

```
FullName(“Lynn”,”Alisha”,”McCormick”);
```

Can You…

- Introduction to Methods
- void Methods
- Passing Arguments to Methods
- Passing Arguments by Reference
- Value-Returning Methods

Next Week

- Value Types and Reference Types
- Array Basics
- Working with Files and Arrays
- Passing Arrays as Arguments to Methods