CGT 215 Lecture 6

Methods: A Deeper Look

Divide and Conquer

- Experience has shown that the best way to develop and maintain a large application is to construct it from small, simple pieces.
- This technique is called ***divide and conquer***

Methods

- Called ***functions*** or ***procedures*** in other programming languages
- Methods allow you to modularize an application by separating its tasks into self-contained units.
- You have created many methods already.
- These methods are sometimes referred to as ***user-defined methods***.

Motivations for Modularizing an Application

- One is "divide and conquer"
  - Makes applications more manageable by constructing it as small, simple pieces
- Another is ***software reusability***
  - Existing methods can be used as building blocks to create new applications.
- Another reason is to avoid repeating code
  - Dividing an application into meaningful methods makes the application easier to debug and maintain.

Software Engineering Observation 7.1

- Don't try to "reinvent the wheel." when possible, reuse Framework Class Library classes and methods.

Software Engineering Observation 7.2

- To promote software reusability, every method should be limited to performing a single, well-defined task, and the name of the method should express that task effectively. Such methods make applications easier to write, debug, maintain, and modify.

Software Engineering Observation 7.3

- If you cannot choose a concise name that expresses a method's task, your method might be attempting to perform too many diverse tasks. It is usually best to break such a method into several smaller methods.

Error-Prevention Tip 7.1

- A small method that performs one task is easier to test and debug than a larger method that performs many tasks.

static methods

- Although most methods execute on specific objects in response to method calls, this is not always the case.
- Sometimes a method performs a task that does not depend on the contents of any object.
- Such a method applies to the class in which it is declared as a whole and is known as a ***static method***.

static methods

- Math.sqrt()
  - sqrt() is a static method of the Math class
  - You do *not* need to create an instance of the Math class in order to call sqrt()
- Console.WriteLine()
  - WriteLine() is a static method of the Console class
  - You do *not* need to create an instance of the Console class in order to call WriteLine()

const variables

- A constant is declared with the keyword const – its value cannot be changed after the constant is declared.
  - Pi is a constant – its value never changes
- Most const variables are by default static, unless declared inside of a method.

static variables

- Each instance of an object of a class has *separate* instance of the variables.
- This is *not* the case with ***static variables***
- When objects of a class containing static variables are created, all the objects of that class *share one copy* of the class's ***static variables***.

Why is method Main declared static?

- Method Main is sometimes called the application's ***entry point***.
- Declaring Main as static allows the execution environment to invoke Main without creating an instance of the class.
  - public static void Main(string args[])

Where do you declare static?

- Before the return type of a method:
  - public **static** void AddListItemMethod()
  - public **static** int GetValue()
- Before the class name of a variable declaration:
  - private **static** string name;
  - private **static** bool isTrue;

Math class

Method	Description	Example

`Abs( x )`	absolute value of *x*	`Abs( 23.7 ) is 23.7`
		`Abs( 0.0 ) is 0.0`
		`Abs( -23.7 ) is 23.7`

`Ceiling( x )`	rounds *x* to the smallest integer not less than *x*	`Ceiling( 9.2 ) is 10.0` `Ceiling( -9.8 ) is -9.0`
`Cos( x )`	trigonometric cosine of *x* (*x* in radians)	`Cos( 0.0 ) is 1.0`
`Exp( x )`	exponential method e^x	`Exp( 1.0 ) is 2.71828` `Exp( 2.0 ) is 7.38906`
`Floor( x )`	rounds *x* to the largest integer not greater than *x*	`Floor( 9.2 ) is 9.0` `Floor( -9.8 ) is -10.0`
`Log( x )`	natural logarithm of *x* (base *e*)	`Log( Math.E ) is 1.0` `Log( Math.E * Math.E ) is 2.0`
`Max( x, y )`	larger value of *x* and *y*	`Max( 2.3, 12.7 ) is 12.7` `Max( -2.3, -12.7 ) is -2.3`
`Min( x, y )`	smaller value of *x* and *y*	`Min( 2.3, 12.7 ) is 2.3` `Min( -2.3, -12.7 ) is -12.7`

Math class

Method	Description	Example
`Pow( x, y )`	*x* raised to the power *y* (i.e., x^y)	`Pow( 2.0, 7.0 ) is 128.0` `Pow( 9.0, 0.5 ) is 3.0`
`Sin( x )`	trigonometric sine of *x* (*x* in radians)	`Sin( 0.0 ) is 0.0`
`Sqrt( x )`	square root of *x*	`Sqrt( 900.0 ) is 30.0`
`Tan( x )`	trigonometric tangent of *x* (*x* in radians)	`Tan( 0.0 ) is 0.0`

Declaring a method

- You need:
  - Access modifier
  - (optional) static
  - Return type
  - Name
  - (optional) Parameters
    - public static void Main(string args[])
    - public void SetGender(string gender)
    - public string GetName()

Methods with multiple parameters

- When a method has more than one parameter, the parameters are specified as a comma-separated list.
  - Notice: each parameter has a type – at type is required for each parameter

```
public double FindMax( double x, double y, double z )
{
    …
}
```

Calling methods

- You must provide the correct number of parameters in each method call
- If a method is declared with 3 parameters, you must pass 3 parameters in.
- Declaration:
  - public double FindMax(**double x, double y, double z**)
- Calling it:
  - FindMax(3.2, 4.5, 1.3);
  - FindMax(); //would be an error
  - FindMax(2.3, 1.2); //would be an error
  - FindMax(2.2, 3.3, 4.4, 5.5); //would be an error

String Concatenation (revisited)

- Use the + symbol to concatenate strings together.
- "hello " + "there" creates the string "hello there"
- "Maximum is: " + result
  - Another example of concatenation

Common Errors

- It is a syntax error to break a string literal across multiple lines in an application:
  - Wrong:
    - "hello
      how are you"

Common Errors

- Confusing the + operator used for string concatenation with the + operator used for addition can lead to strange results.
  - Example:
    - "y + 2 = " + y + 2
      - Results in the string "y + 2 = 52", ***not*** "y + 2 = 7"
    - "y + 2 = " + (y + 2)
      - Results in the string "y + 2 = 7"

Common Errors

- Declaring a method outside the body of a class declaration or inside the body of another method is a syntax error.
- Omitting the return type in a method declaration is a syntax error.

Common Errors

- Re-declaring a method parameter as a local variable in the method's body is a compilation error.
- Forgetting to return a value from a method that should return one is a compilation error.

Conversion Types

Type	Conversion types
bool	no possible implicit conversions to other simple types
byte	ushort, short, uint, int, ulong, long, decimal, float or double
char	ushort, int, uint, long, ulong, decimal, float or double
decimal	no possible implicit conversions to other simple types
double	no possible implicit conversions to other simple types
float	double
int	long, decimal, float or double
long	decimal, float or double
sbyte	short, int, long, decimal, float or double
short	int, long, decimal, float or double
uint	ulong, long, decimal, float or double
ulong	decimal, float or double
ushort	uint, int, ulong, long, decimal, float or double

.NET namespaces

Namespace	Description
System.Windows.Forms	Contains the classes required to create and manipulate GUIs. (Various classes in this namespace are discussed in Chapter 14, Graphical User Interfaces with Windows Forms: Part 1, and Chapter 15, Graphical User Interfaces with Windows Forms: Part 2.)
System.Windows.Controls System.Windows.Input System.Windows.Media System.Windows.Shapes	Contain the classes of the Windows Presentation Foundation for GUIs, 2-D and 3-D graphics, multimedia and animation. (You'll learn more about these namespaces in Chapter 16, GUI with Windows Presentation Foundation, Chapter 17, WPF Graphics and Multimedia and Chapter 24, Silverlight, Rich Internet Applications and Multimedia.)
System.Linq	Contains the classes that support Language Integrated Query (LINQ). (You'll learn more about this namespace in Chapter 9, Introduction to LINQ and Generic Collections, and several other chapters throughout the book.)
System.Data System.Data.Linq	Contain the classes for manipulating data in databases (i.e., organized collections of data), including support for LINQ to SQL. (You'll learn more about these namespaces in Chapter 21, Databases and LINQ to SQL.)
System.IO	Contains the classes that enable programs to input and output data. (You'll learn more about this namespace in Chapter 19, Files and Streams.)

.NET namespaces

Namespace	Description
System.Web	Contains the classes used for creating and maintaining web applications, which are accessible over the Internet. (You'll learn more about this namespace in Chapter 22, ASP.NET 3.5 and ASP.NET AJAX.)
System.Xml.Linq	Contains the classes that support Language Integrated Query (LINQ) for XML documents. (You'll learn more about this namespace in Chapter 20, XML and LINQ to XML, and several other chapters throughout the book.)
System.Xml	Contains the classes for creating and manipulating XML data. Data can be read from or written to XML files. (You'll learn more about this namespace in Chapter 20.)
System.Collections System.Collections.Generic	Contain the classes that define data structures for maintaining collections of data. (You'll learn more about these namespaces in Chapter 28, Collections.)
System.Text	Contains the classes that enable programs to manipulate characters and strings. (You'll learn more about this namespace in Chapter 18, Strings, Characters and Regular Expressions.)

using statements

- To include a namespace from the .NET Framework Class Library, use the ***using*** statement.
  - using System.Windows.Forms;
  - using System.Data;
  - using System.IO;
  - using System.Windows.Media;

Scope of declarations

- The scope of a parameter declaration is: the body of the method in which the declaration appears.
- The scope of a local-variable declaration is: from the point at which the declaration appears to the end of the block containing the declaration.

Scope of declarations

- The scope of a local-variable declaration that appears in the initialization section of a *for* statement's header is: the body of the for statement and the other expressions in the header.

Scope of declarations

- The scope of a method, property, or field of a class is the entire body of the class.
- This enables non-static methods and properties of a class to use any of the class's fields, methods, and properties, regardless of the order in which they are declared.
- Similarly, static methods and properties can use any of the static members of the class.

Method Overloading

- Methods of the same name can be declared in the same class, as long as they have different sets of parameters (determined by the number, types, and order of the parameters)
- This is called ***method overloading***.
- When an ***overloaded method*** is called, the C# compiler selects the appropriate method by examining the number, types and order of the arguments in the call.

Overloaded methods

- Method *calls* cannot be distinquished by return type. Therefore, the return type is not included as one of the attributes that define method overloading.

Examples

- The order of parameter types is important.
  - public void Method1(int a, float b, string c)
  - public void Method1(float a, int b, string c)
  - public void Method1(string a, float b, int c)
- Example of calling each of the above methods:
  - Method1(3, 4.5, "hi there");
  - Method1(4.5, 3, "hi there");
  - Method1("hi there", 4.5, 3);

Examples

- The number of parameters is important
  - public void MyMethod(string a)
  - public void MyMethod(string a, string b)
  - public void MyMethod(string a, string b, string c)
  - public void MyMethod(string a, string b, string c, string d)
- Example of calling each of the above methods:
  - MyMethod("here", "is", "some", "text");
  - MyMethod("here", "is", "some");
  - MyMethod("here", "is");
  - MyMethod("here");