Microsoft Visual C# 2010

Fourth Edition

Chapter 1

A First Program Using C#

Objectives

- Learn about programming
- Learn about procedural and object-oriented programming
- Learn about the features of object-oriented programming languages
- Learn about the C# programming language
- Write a C# program that produces output
- Learn how to select identifiers to use within your programs

Objectives (cont'd.)

- Improve programs by adding comments and using the `System` namespace
- Write and compile a C# program using the command prompt and using Visual Studio

Programming

- Computer **program**
  - Set of instructions that tells a computer what to do
  - Also called software
- Software comes in two broad categories
  - **System software**
  - **Application software**
- **Machine language**
  - Expressed as a series of 1s and 0s
    - 1s represent switches that are on, and 0s represent switches that are off

Programming (cont'd.)

- **High-level programming languages**
  - Use reasonable terms such as “read,” “write,” or “add”
    - Instead of the sequence of on/off switches that perform these tasks
  - Allows you to assign reasonable names to areas of computer memory
  - Has its own **syntax** (rules of the language)
- **Compiler**
  - Translates high-level language statements into machine code

Programming (cont'd.)

- Programming **logic**
  - Involves executing the various statements and procedures in the correct order
    - To produce the desired results
- **Debugging**
  - Process of removing all syntax and logical errors from the program

Procedural and Object-Oriented Programming

- **Procedural program**
  - Create and name computer memory locations that can hold values (**variables**)
  - Write a series of steps or operations to manipulate those values
- **Identifier**
  - A one-word name used to reference a variable
- **Procedures** or **methods**
  - Logical units that group individual operations used in a computer program
  - **Called** or **invoked** by other procedures or methods

Procedural and Object-Oriented Programming (cont'd.)

- **Object-oriented programming**
  - An extension of procedural programming
- **Objects**
  - Similar to concrete objects in the real world
  - Contain their own variables and methods
  - **Attributes** of an object represent its characteristics
  - **State of an object** is the collective value of all its attributes at any point in time
  - **Behaviors of an object** are the things it “does”

Procedural and Object-Oriented Programming (cont'd.)

- Originally used for two types of applications
  - **Computer simulations**
  - **Graphical user interfaces (GUIs)**

Features of Object-Oriented Programming Languages

- **Classes**
  - A category of objects or a type of object
  - Describes the attributes and methods of every object that is an **instance**, or example, of that class
- **Objects**
  - An instance of a class
- **Encapsulation**
  - Technique of packaging an object's attributes and methods into a cohesive unit; undivided entity
  - Using a **black box**

Features of Object-Oriented Programming Languages (cont'd.)

- **Interface**
  - Interaction between a method and an object
- **Inheritance**
  - Provides the ability to extend a class to create a more specific class
- **Polymorphism**
  - Describes the ability to create methods that act appropriately depending on the context

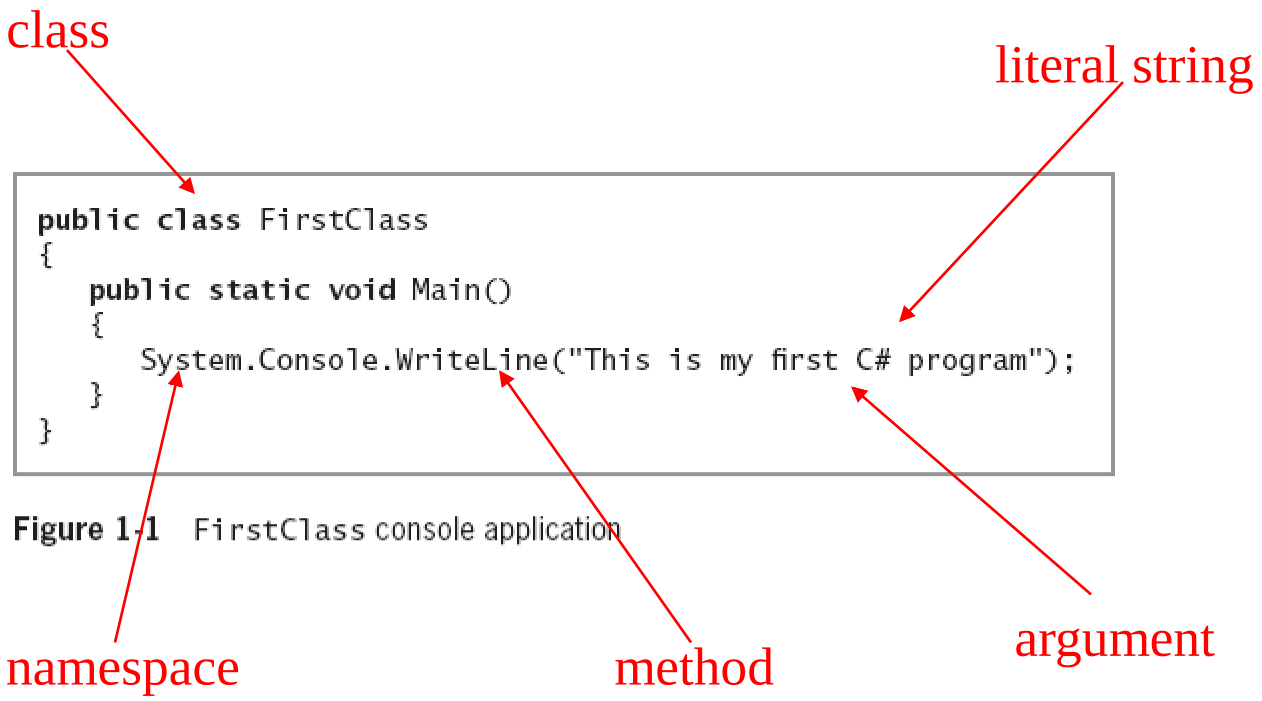
The C# Programming Language

- Developed as an object-oriented and component-oriented language
- Part of Microsoft Visual Studio 2010
- Allows every piece of data to be treated as an object and to consistently employ the principles of object-oriented programming
- Contains a GUI interface that makes it similar to Visual Basic

The C# Programming Language (cont'd.)

- Modeled after the C++ programming language
  - However, eliminates some of the most difficult features to understand in C++
- Very similar to Java
  - In C#, simple data types are objects

Writing a C# Program that Produces Output



```
public class FirstClass
{
    public static void Main()
    {
        System.Console.WriteLine("This is my first C# program");
    }
}
```

Figure 1-1 FirstClass console application

Writing a C# Program that Produces Output (cont'd.)

- **Namespace**
  - Provides a way to group similar classes
- C# method parts
  - **Method header**
    - Includes the method name and information about what will pass into and be returned from a method
  - **Method body**
    - Contained within a pair of curly braces and includes all the instructions executed by the method

Writing a C# Program that Produces Output (cont'd.)

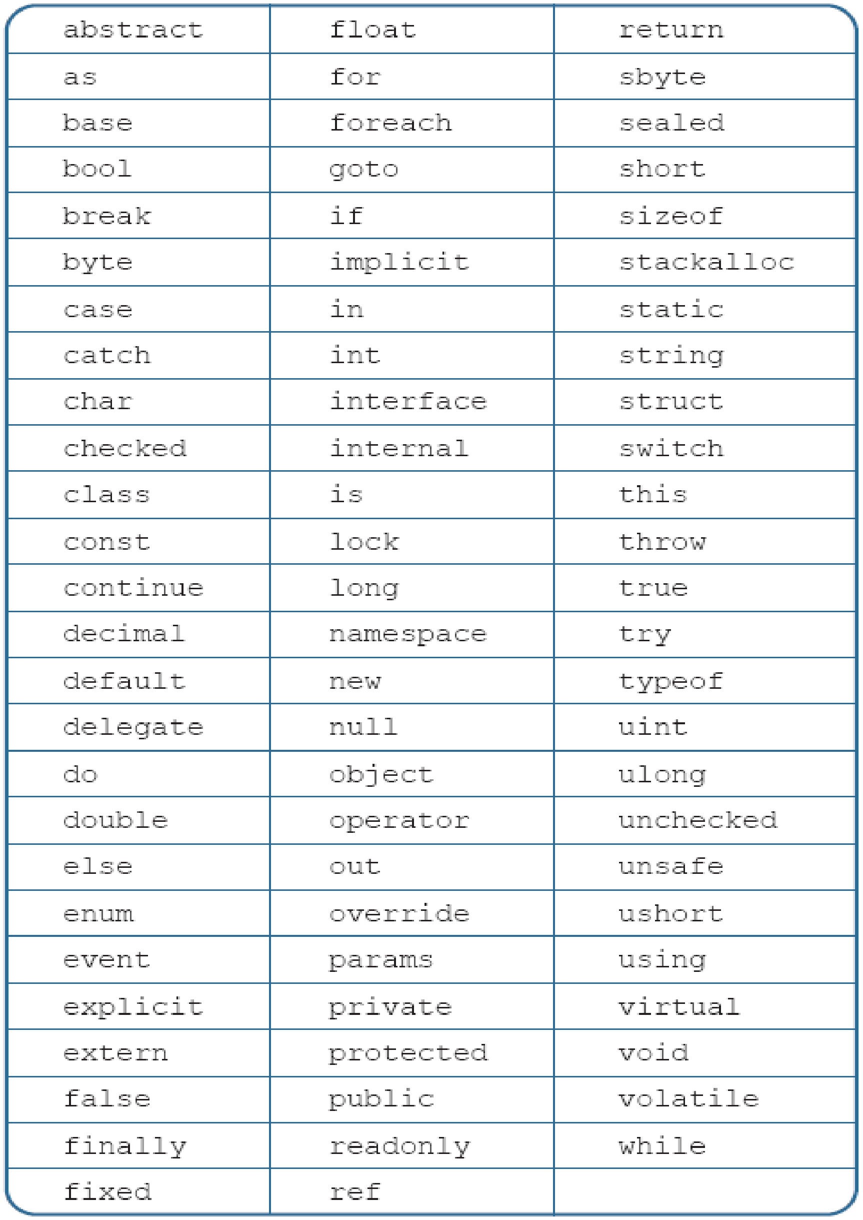
- **Whitespace**
  - Any combination of spaces, tabs, and carriage returns (blank lines)
  - Organizes your code and makes it easier to read
- **Access modifier**
  - Defines the circumstances under which the method can be accessed
- **Keywords**
  - Predefined and reserved identifiers that have special meaning to the compiler

Writing a C# Program that Produces Output (cont'd.)

- The name of the method is `Main()`
  - Every application must have a `Main()` method
  - Classes with a `Main()` method are called **application classes**; others are **non-application classes**
- The method returns nothing as indicated by the keyword **`void`**

Selecting Identifiers

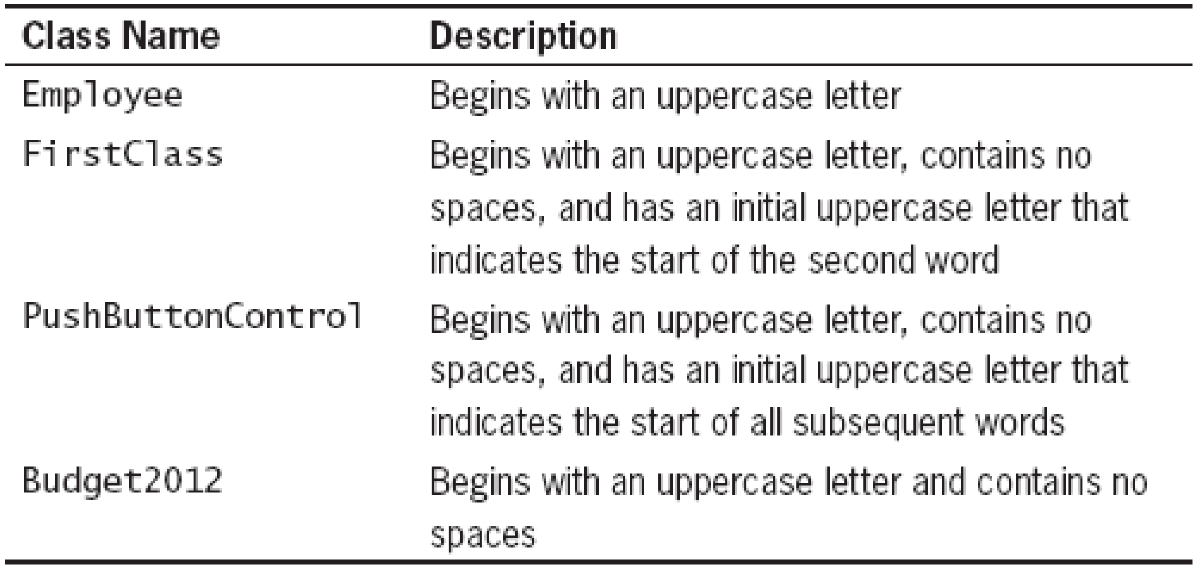
- Requirements
  - Must begin with an underscore, at sign (@), or letter
    - Letters include foreign-alphabet letters
  - Can contain only letters, digits, underscores, and the at sign
    - Not special characters such as #, $, or &
  - Cannot be a C# reserved keyword



abstract	float	return
as	for	sbyte
base	foreach	sealed
bool	goto	short
break	if	sizeof
byte	implicit	stackalloc
case	in	static
catch	int	string
char	interface	struct
checked	internal	switch
class	is	this
const	lock	throw
continue	long	true
decimal	namespace	try
default	new	typeof
delegate	null	uint
do	object	ulong
double	operator	unchecked
else	out	unsafe
enum	override	ushort
event	params	using
explicit	private	virtual
extern	protected	void
false	public	volatile
finally	readonly	while
fixed	ref	

Table 1-1 C# reserved keywords

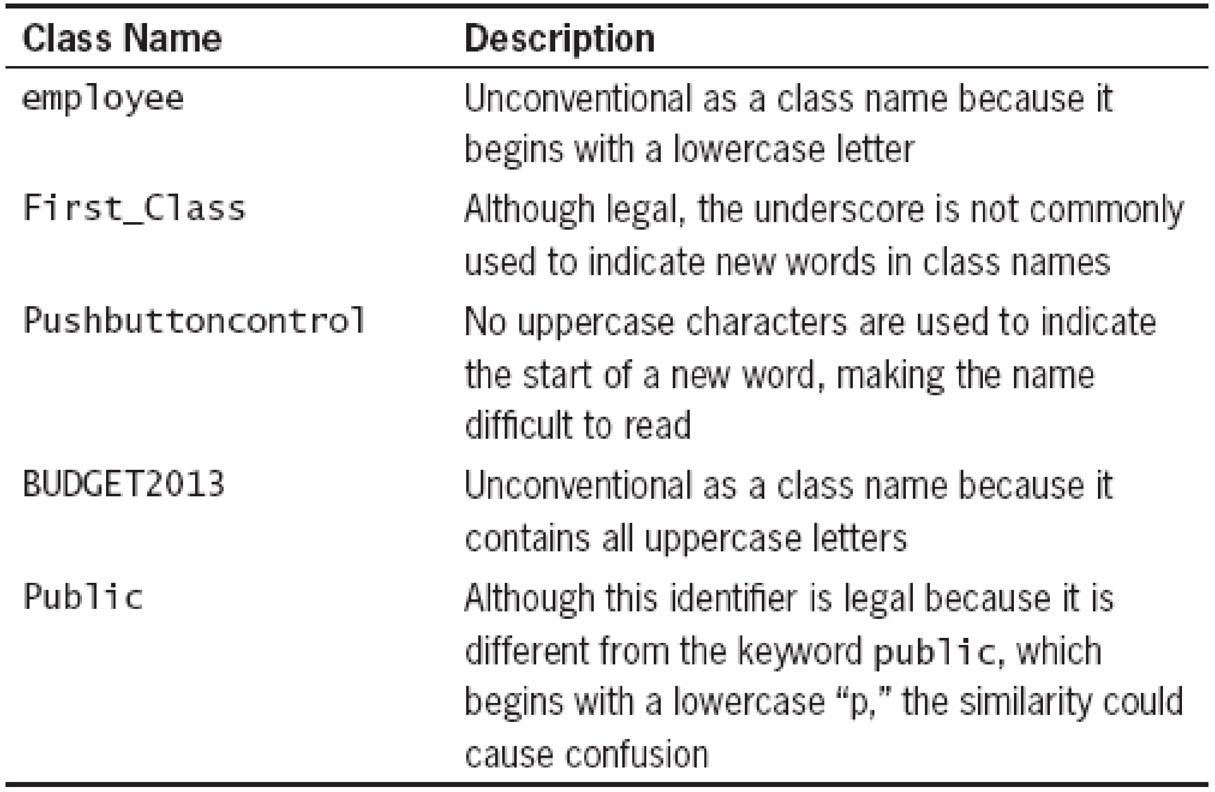
Selecting Identifiers (cont'd.)



Class Name	Description
`Employee`	Begins with an uppercase letter
`FirstClass`	Begins with an uppercase letter, contains no spaces, and has an initial uppercase letter that indicates the start of the second word
`PushButtonControl`	Begins with an uppercase letter, contains no spaces, and has an initial uppercase letter that indicates the start of all subsequent words
`Budget2012`	Begins with an uppercase letter and contains no spaces

Table 1-2 Some valid and conventional class names in C#

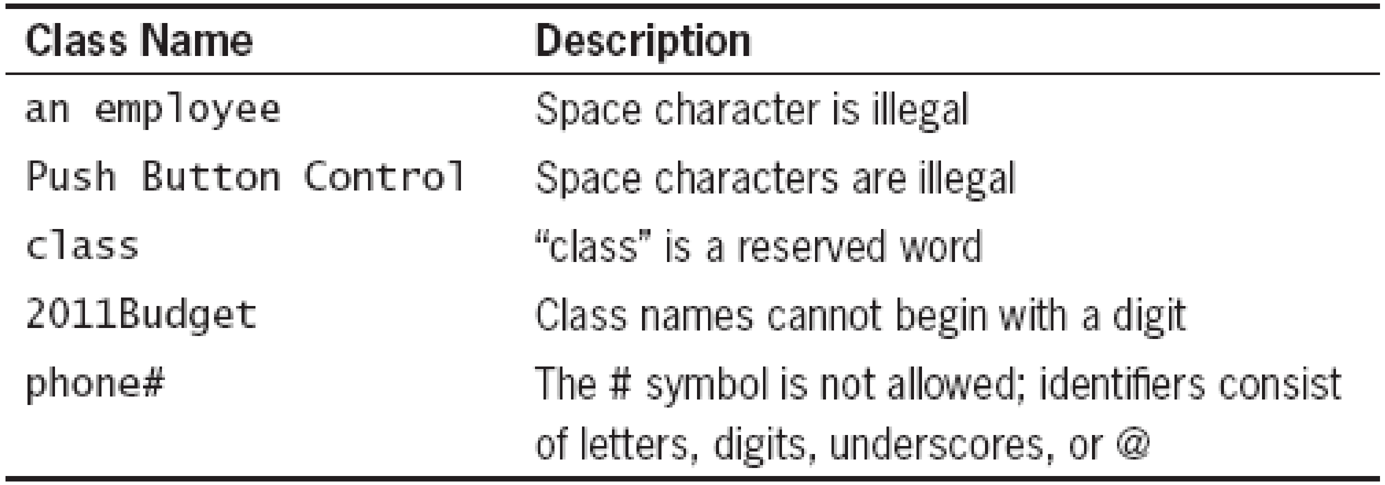
Selecting Identifiers (cont'd.)



Class Name	Description
`employee`	Unconventional as a class name because it begins with a lowercase letter
`First_Class`	Although legal, the underscore is not commonly used to indicate new words in class names
`Pushbuttoncontrol`	No uppercase characters are used to indicate the start of a new word, making the name difficult to read
`BUDGET2013`	Unconventional as a class name because it contains all uppercase letters
`Public`	Although this identifier is legal because it is different from the keyword `public`, which begins with a lowercase "p," the similarity could cause confusion

Table 1-3 Some unconventional (though legal) class names in C#

Selecting Identifiers (cont'd.)



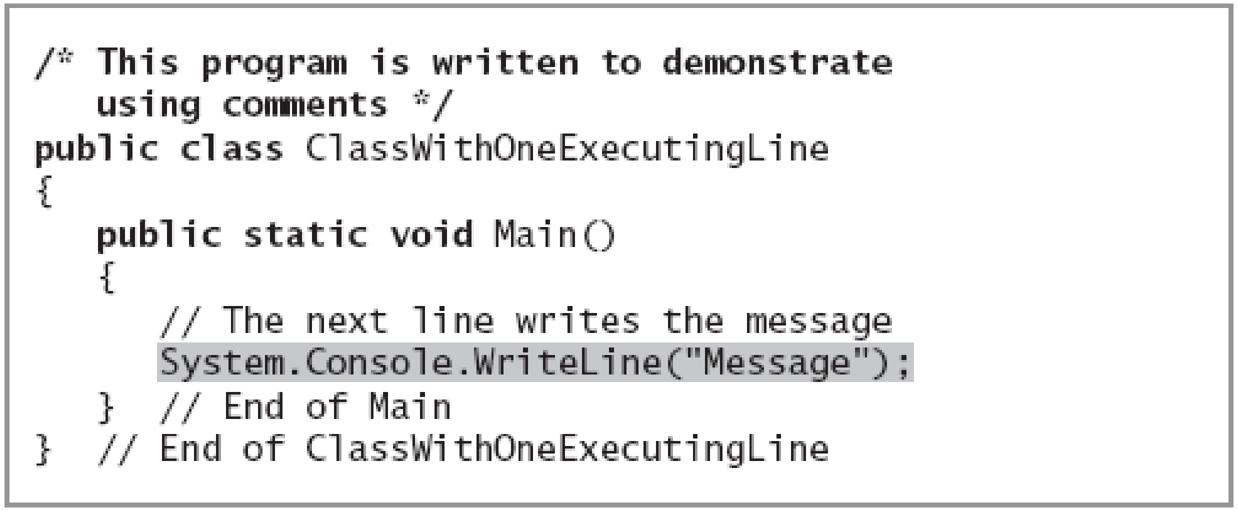
Class Name	Description
`an employee`	Space character is illegal
`Push Button Control`	Space characters are illegal
`class`	"class" is a reserved word
`2011Budget`	Class names cannot begin with a digit
`phone#`	The # symbol is not allowed; identifiers consist of letters, digits, underscores, or @

Table 1-4 Some illegal class names in C#

Improving Programs by Adding Program Comments

- **Program comments**
  - Nonexecuting statements that document a program
- **Comment out**
  - Turn a statement into a comment
- Types of comments in C#
  - Line comments
  - Block comments
  - XML-documentation format comments

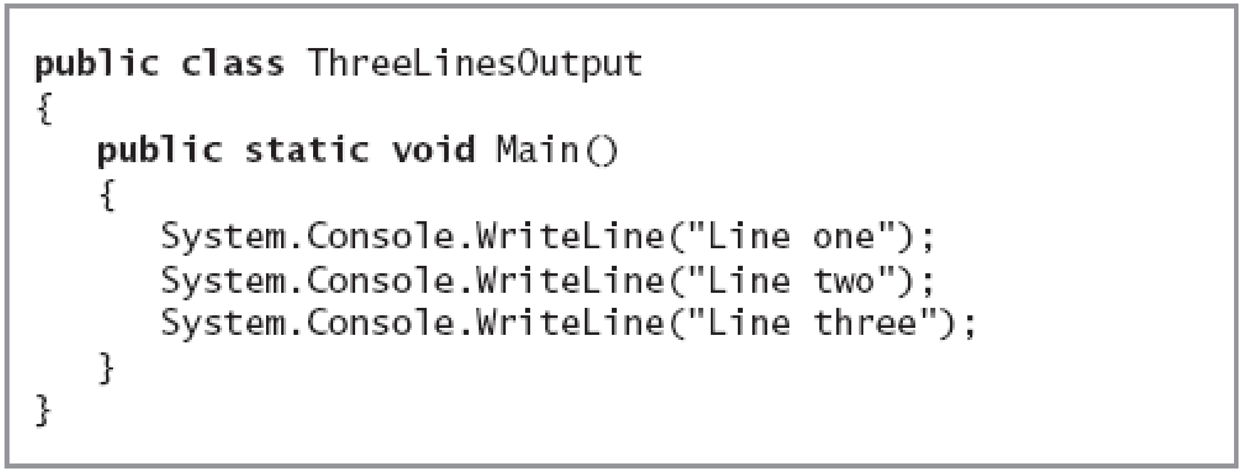
Adding Program Comments (cont'd.)



```
/* This program is written to demonstrate
   using comments */
public class ClassWithOneExecutingLine
{
   public static void Main()
   {
      // The next line writes the message
      System.Console.WriteLine("Message");
   }  // End of Main
}  // End of ClassWithOneExecutingLine
```

Figure 1-4 Using comments within a program

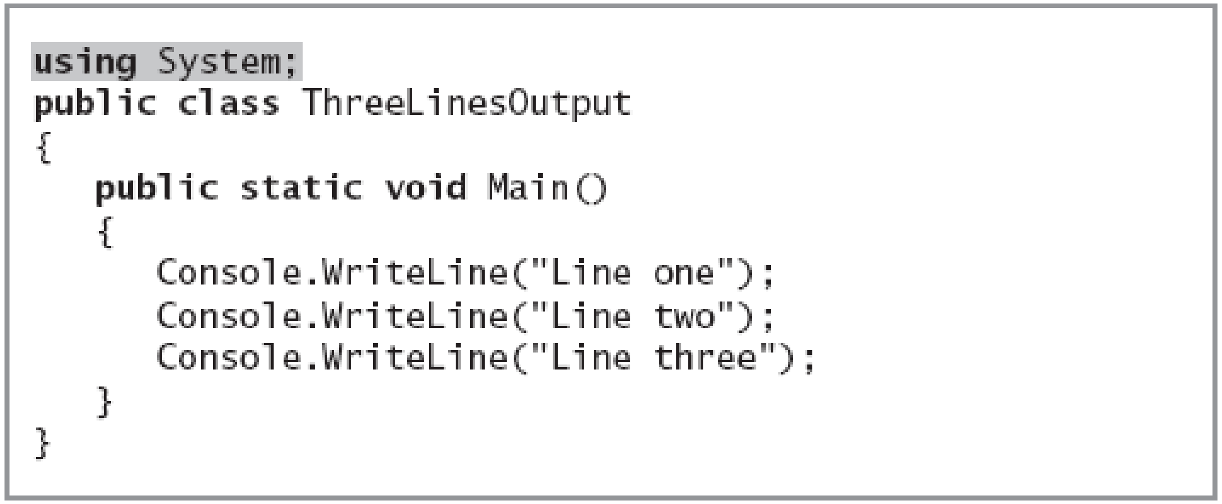
Using the `System` Namespace



```
public class ThreeLinesOutput
{
   public static void Main()
   {
      System.Console.WriteLine("Line one");
      System.Console.WriteLine("Line two");
      System.Console.WriteLine("Line three");
   }
}
```

Figure 1-5 A program that produces three lines of output

Using the `System` Namespace (cont'd.)



```
using System;
public class ThreeLinesOutput
{
   public static void Main()
   {
      Console.WriteLine("Line one");
      Console.WriteLine("Line two");
      Console.WriteLine("Line three");
   }
}
```

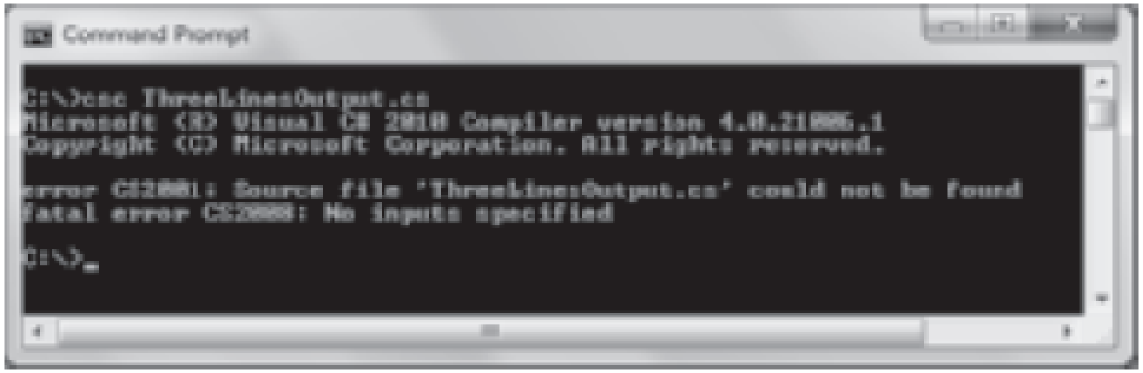
Figure 1-7 A program that produces three lines of output with a `using System` clause

Writing and Compiling a C# Program

- Steps for viewing a program output
  - Compile **source code** into **intermediate language (IL)**
  - C# **just in time (JIT)** compiler translates the intermediate code into executable statements
- You can use either of two ways to compile
  - The **command line**
  - The **Integrated Development Environment (IDE)**

Compiling Code from the Command Prompt

- What to do if you receive an operating system error message
  - Command `csc`
    - Stands for “C Sharp compiler”

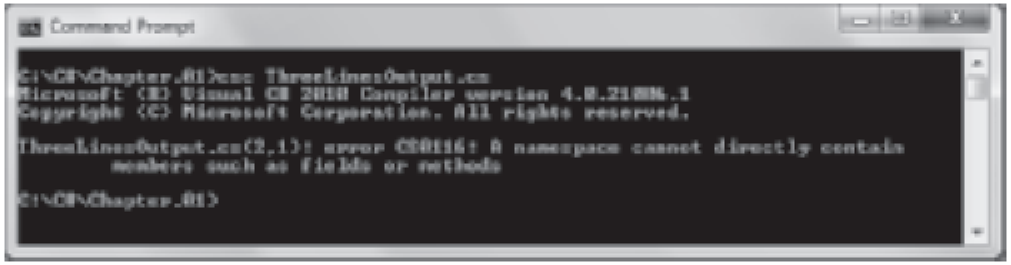


```
C:\>csc ThreeLinesOutput.cs
Microsoft (R) Visual C# 2010 Compiler version 4.0.21006.1
Copyright (C) Microsoft Corporation. All rights reserved.

error CS2001: Source file 'ThreeLinesOutput.cs' could not be found
fatal error CS2008: No inputs specified

C:\>_
```

Figure 1-8 Attempt to compile a program from the root directory at the command line, and error message received

Compiling Code from the Command Prompt (cont'd.)

- What to do if you receive a programming language error message
  - Program error messages start with the program name
  - Followed by the line number and position within the line of the error



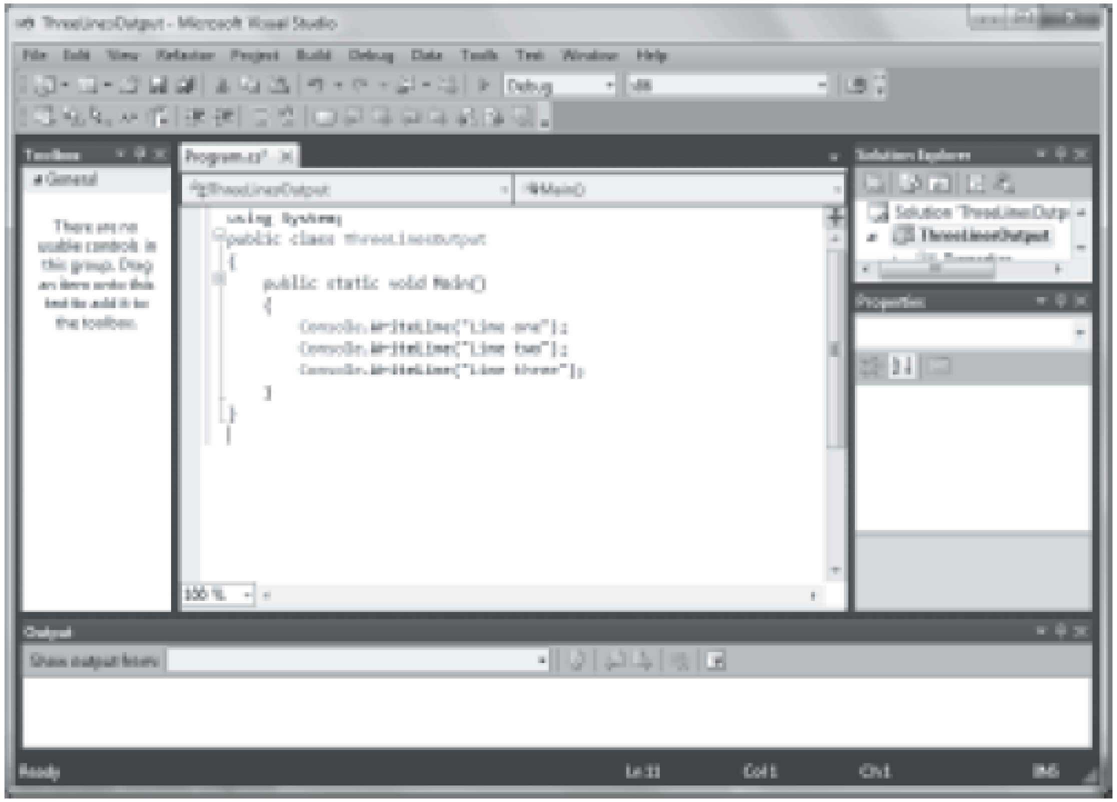
```
C:\C#\Chapter.01>csc ThreeLinesOutput.cs
Microsoft (R) Visual C# 2010 Compiler version 4.0.21006.1
Copyright (C) Microsoft Corporation. All rights reserved.

ThreeLinesOutput.cs(2,1): error CS0116: A namespace cannot directly contain
        members such as fields or methods

C:\C#\Chapter.01>
```

Figure 1-9 Error message generated when "public" is mistyped in the `ThreeLinesOutput` program

Compiling Code from within the Visual Studio IDE

- Advantages of using the Visual Studio IDE
  - Some of the code you need is already created for you
  - The code is displayed in color
  - You can double-click an error message and the cursor will move to the line of code that contains the error
  - Other debugging tools are available

Compiling Code from within the Visual Studio IDE (cont'd.)

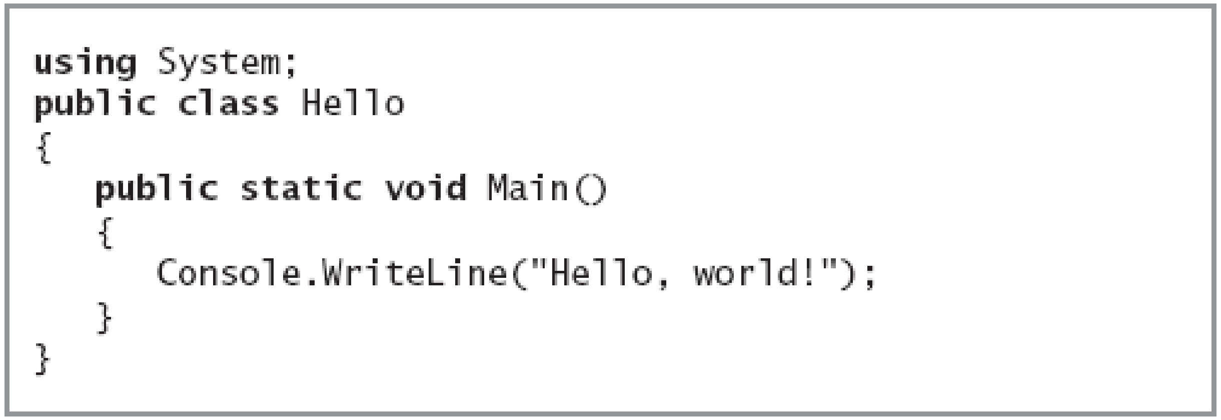

ThreeLinesOutput - Microsoft Visual Studio
File Edit View Refactor Project Build Debug Data Tools Test Window Help
Debug
Toolbox
General
There are no usable controls in this group. Drag an item onto this text to add it to the toolbox.
Program.cs*
ThreeLinesOutput
Main()
Solution Explorer
Solution 'ThreeLinesOutput'
ThreeLinesOutput
Properties
100 %
Output
Show output from:
Ready
Ln 11
Col 1
Ch 1
INS

```
using System;
public class ThreeLinesOutput
{
    public static void Main()
    {
        Console.WriteLine("Line one");
        Console.WriteLine("Line two");
        Console.WriteLine("Line three");
    }
}
```

Figure 1-10 `ThreeLinesOutput` program as it appears in Visual Studio

You Do It

- Enter your first C# program into a text editor so you can execute it
- Use any text editor to write the following code and save it as Hello.cs



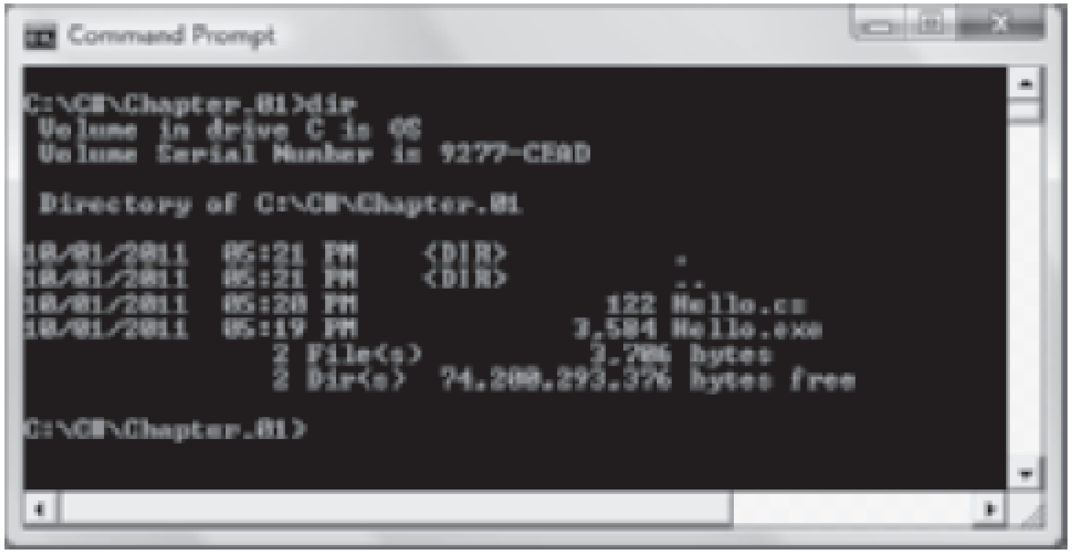
```
using System;
public class Hello
{
   public static void Main()
   {
      Console.WriteLine("Hello, world!");
   }
}
```

Figure 1-11 The Hello class

Compiling and Executing a Program from the Command Line

- Type the command that compiles your program:

```
csc Hello.cs
```

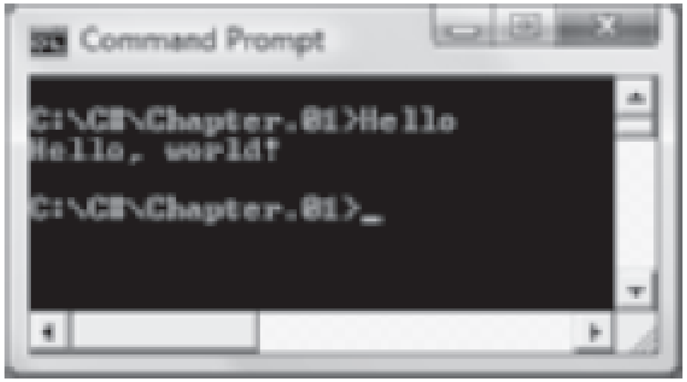


```
C:\C#\Chapter.01>dir
 Volume in drive C is OS
 Volume Serial Number is 9277-CEAD

 Directory of C:\C#\Chapter.01

10/01/2011  05:21 PM    <DIR>          .
10/01/2011  05:21 PM    <DIR>          ..
10/01/2011  05:20 PM               122 Hello.cs
10/01/2011  05:19 PM             3,584 Hello.exe
               2 File(s)          3,706 bytes
               2 Dir(s)  74,200,293,376 bytes free

C:\C#\Chapter.01>
```

Figure 1-12 Directory of Chapter.01 folder after compiling Hello.cs

Compiling and Executing a Program from the Command Line (cont'd.)

- Execute the program Hello.exe



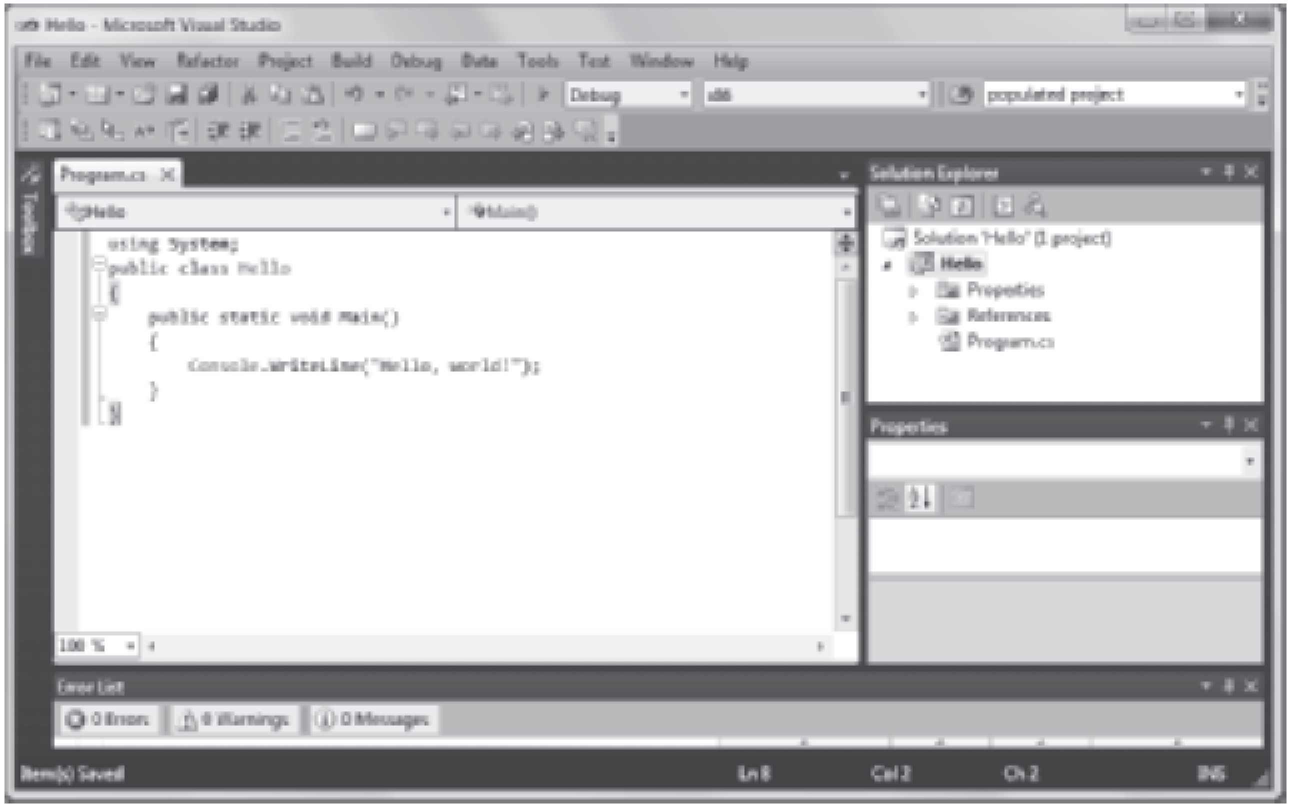
```
C:\C#\Chapter.01>Hello
Hello, world!

C:\C#\Chapter.01>_
```

Figure 1-13 Output of the `Hello` application

Compiling and Executing a Program Using the Visual Studio IDE

- Steps
  - Create a new project (console application)
  - Enter the project name
  - Write your program using the editor
  - To compile the program, click **Build** on the menu bar, and then click **Build Solution**
    - As an alternative, you can press **F6**
  - Click **Debug** on the menu bar and then click **Start Without Debugging**

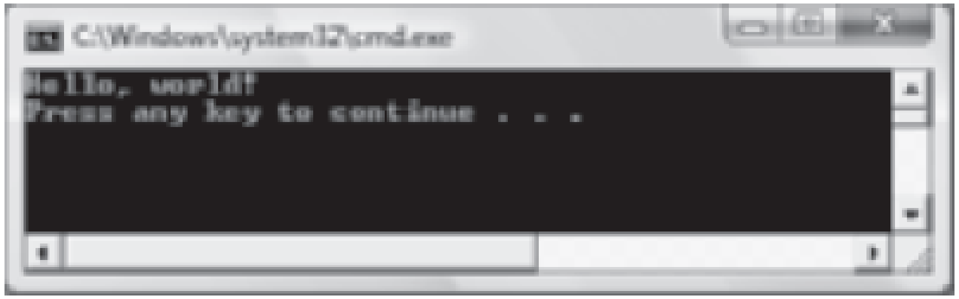
Compiling and Executing a Program Using the Visual Studio IDE (cont'd.)



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

Figure 1-18 The `Hello` application in the IDE

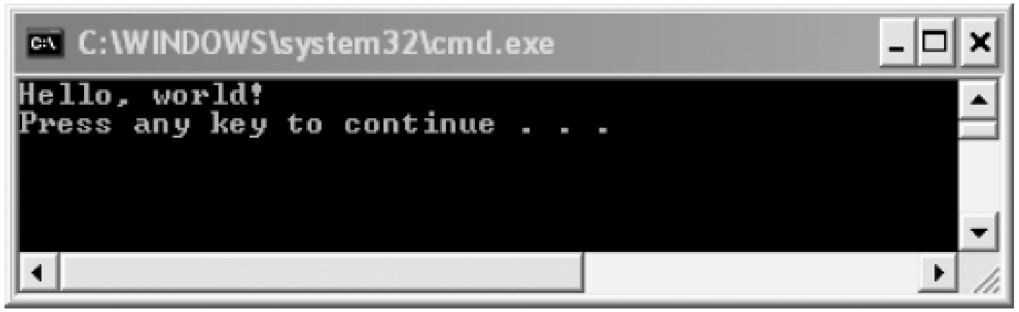
Compiling and Executing a Program Using the Visual Studio IDE (cont'd.)



```
Hello, world!
Press any key to continue . . .
```

Figure 1-19 Output of the `Hello` application in Visual Studio

Compiling and Executing a Program Using the Visual Studio IDE (cont'd.)



```
Hello, world!
Press any key to continue . . .
```

Figure 1-20 Output of the `Hello` application in Visual Studio

Deciding Which Method to Use

- Advantage of using the command line
  - Saves disk space
- Advantages of using the Visual Studio IDE
  - Automatic sentence completion
  - Words are displayed using different colors based on their category
  - Code automatically generated by the IDE is very helpful when writing a GUI

Adding Comments to a Program

- Line comment example

```
// Filename Hello.cs
// Written by <your name>
// Written on <today’s date>
```

- Block comment example

```
/* This program demonstrates the use of
the WriteLine() method to print the
message Hello, world! */
```

Summary

- A computer program is a set of instructions that tell a computer what to do
- Understand differences between procedural programming and object-oriented programming
- Objects are instances of classes and are made up of attributes and methods
- The C# programming language is an object-oriented and component-oriented language
- `System.Console.WriteLine()` method
  - Produces a line of console output

Summary (cont'd.)

- You can define a C# class or variable by using any name or identifier
- Comments are nonexecuting statements that you add to document a program
  - Or to disable statements when you test a program
- Use namespaces to improve programs
- To create a C# program, you can use the Microsoft Visual Studio environment