3

Introduction to Visual Basic Programming

Comment is free, but facts are sacred.

– **C. P. Scott**

The creditor hath a better memory than the debtor.

– **James Howell**

When faced with a decision, I always ask, "What would be the most fun?"

– **Peggy Walker**

Equality, in a social sense, may be divided into that of condition and that of rights.

– **James Fenimore Cooper**

OBJECTIVES

In this chapter you will learn:

- To write simple Visual Basic programs using code rather than visual programming.
- To write statements that input data from the keyboard and output data to the screen.
- To declare and use data of various types.
- To store and retrieve data from memory.
- To use arithmetic operators to perform calculations.

OBJECTIVES

- To use the precedence of arithmetic operators to determine the order in which operators are applied.
- To write decision-making statements.
- To use equality and relational operators to compare operands.
- To use message dialogs to display messages.

Outline

3.1 Introduction

- **Console applications** do not have a graphical user interface.
- There are several types of Visual Basic projects; the console application is one of the simplest.
- The application's output appears in the **Console window** or a Windows **Command Prompt**.

Outline

Welcome1.vb

Comments improve code readability.

```
1  ' Fig. 3.1: Welcome1.vb
2  ' Simple Visual Basic program.
3
4  Module Welcome1
5
6     Sub Main()
7
8        Console.WriteLine("Welcome to Visual Basic!")
9
10    End Sub ' Main
11
12 End Module ' Welcome1
```

```
Welcome to Visual Basic!
```

Fig. 3.1 | Simple Visual Basic program.

3.2 Displaying a Line of Text (Cont.)

- A **single-quote character** (**'**) starts a **comment**.
- Comments improve code readability.
- The Visual Basic compiler ignores comments.
- Console applications consist of pieces called **modules**.

3.2 Displaying a Line of Text (Cont.)

- `Module` is a **keyword** reserved for use by Visual Basic.
  - A complete list of keywords is presented in Fig. 3.2.

Visual Basic keywords and contextual keywords				
AddHandler	AddressOf	Alias	And	AndAlso
As	Boolean	ByRef	Byte	ByVal
Call	Case	Catch	CBool	CByte
CChar	CDate	CDbl	CDec	Char
CInt	Class	CLng	CObj	Const
Continue	CSByte	CShort	CSng	CStr
CType	CUInt	CULng	CUShort	Date
Decimal	Declare	Default	Delegate	Dim
DirectCast	Do	Double	Each	Else
ElseIf	End	Enum	Erase	Error
Event	Exit	False	Finally	For

Fig. 3.2 | Keywords and contextual keywords in Visual Basic. (Part 1 of 3.)

3.2 Displaying a Line of Text (Cont.)

Visual Basic keywords and contextual keywords				
Friend	Function	Get	GetType	GetXmlNamespace
Global	GoTo	Handles	If	Implements
Imports	In	Inherits	Integer	Interface
Is	IsNot	Lib	Like	Long
Loop	Me	Mod	Module	MustInherit
MustOverride	MyBase	MyClass	Namespace	Narrowing
New	Next	Not	Nothing	NotInheritable
NotOverridable	Object	Of	On	Operator
Option	Optional	Or	OrElse	Overloads
Overridable	Overrides	ParamArray	Partial	Private
Property	Protected	Public	RaiseEvent	ReadOnly
ReDim	REM	RemoveHandler	Resume	Return
SByte	Select	Set	Shadows	Shared
Short	Single	Static	Step	Stop
SByte	Select	Set	Shadows	Shared
String	Structure	Sub	SyncLock	Then

Fig. 3.2 | Keywords and contextual keywords in Visual Basic. (Part 2 of 3.)

3.2 Displaying a Line of Text (Cont.)

Visual Basic keywords and contextual keywords				
Throw	To	True	Try	TryCast
TypeOf	UInteger	ULong	UShort	Using
When	While	Widening	With	WithEvents
WriteOnly	Xor			
Contextual keywords				
Aggregate	Ansi	Assembly	Auto	Binary
Compare	Custom	Distinct	Equals	Explicit
From	Group By	Group Join	Into	IsFalse
IsTrue	Join	Key	Let	Mid
Off	Order By	Preserve	Skip	Skip While
Strict	Take	Take While	Text	Unicode
Until	Where			
The following are retained as keywords, although they are no longer supported in Visual Basic 2008				
EndIf	GoSub	Variant	Wend	

Fig. 3.2 | Keywords and contextual keywords in Visual Basic. (Part 3 of 3.)

3.2 Displaying a Line of Text (Cont.)

Common Programming Error 3.1

You cannot use a keyword as an identifier, so it is an error, for example, use them as a Module name. The Visual Basic compiler helps you locate such errors in your programs. Contextual keywords may be used as identifiers, but this is not recommended.

3.2 Displaying a Line of Text (Cont.)

- The name of the `Module` is an **identifier**.
  - Can consist of letters, digits and underscores (_).
  - Cannot begin with a digit or contain spaces.

- Keywords and identifiers are not **case sensitive**.

Good Programming Practice 3.1

Use whitespace to enhance program readability.

3.2 Displaying a Line of Text (Cont.)

- Console applications begin executing at `Main`, the **entry point of the program**.
- `Sub` begins the **body of the method declaration** (the code that will be executed).
- `End Sub` closes the method declarations.

3.2 Displaying a Line of Text (Cont.)

Good Programming Practice 3.2

Indent the entire body of each method declaration one additional “level” of indentation. This emphasizes the structure of the method, improving its readability.

3.2 Displaying a Line of Text (Cont.)

- Characters in double quotes are called **strings**.
- The entire line including `Console.WriteLine` is called a **statement**.

- `Console.WriteLine` contains two identifiers separated by the **dot separator** (`.`).
  - The identifier to the right of the dot is the **method name**.
  - The identifier to the left of the dot is the **class name**.
  - This is known as a **method call**.

3.3 Creating Your First Program in Visual Basic Express

- **File > New Project...** to display the **New Project** dialog (Fig. 3.3).
  - Choose Console Application.
  - For **Name**, enter `Welcome1`, then click **OK**.

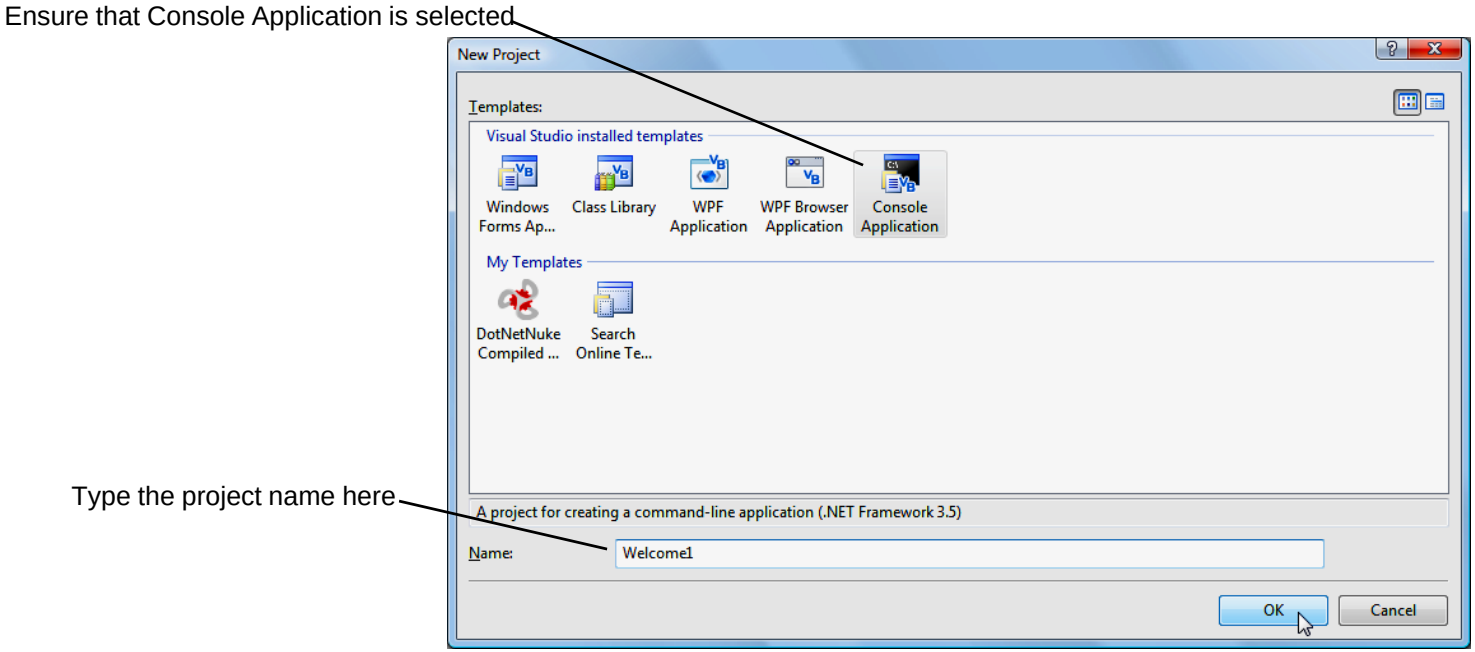


Fig. 3.3 | Creating a **Console Application** with the **New Project** dialog.

3.3 Creating Your First Program in Visual Basic Express (Cont.)

- The IDE now contains the open console application (Fig. 3.4).
- The code coloring scheme is called **syntax-color highlighting**.

Editor window
(type your program code here)

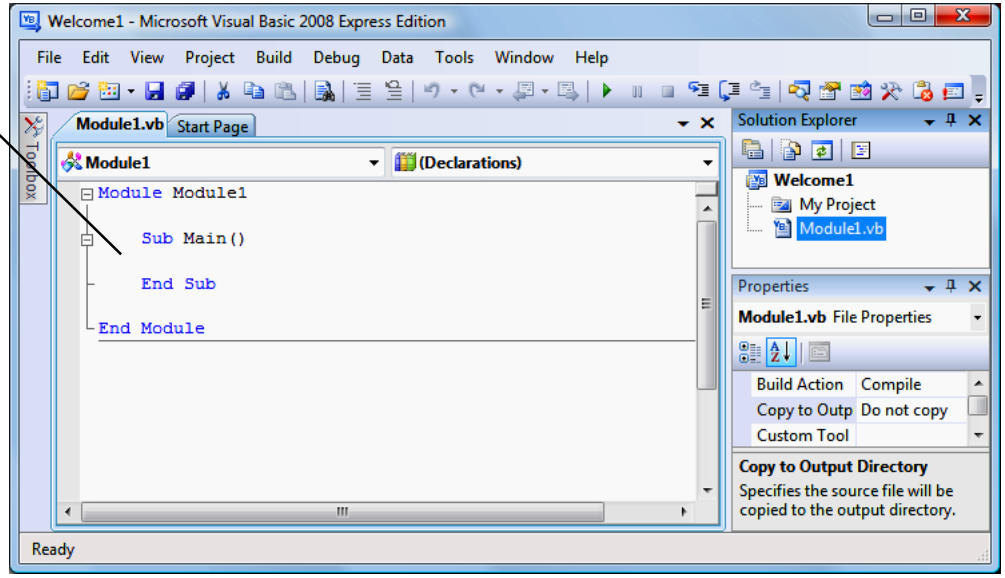


Fig. 3.4 | IDE with an open console application.

3.3 Creating Your First Program in Visual Basic Express (Cont.)

- Select **Tools > Options...**.
  - Ensure the **Show all settings** check box (Fig. 3.5) is unchecked.
  - Expand the **Text Editor Basic** category, and select **Editor**.
  - Under **Interaction**, check the **Line Numbers** check box.

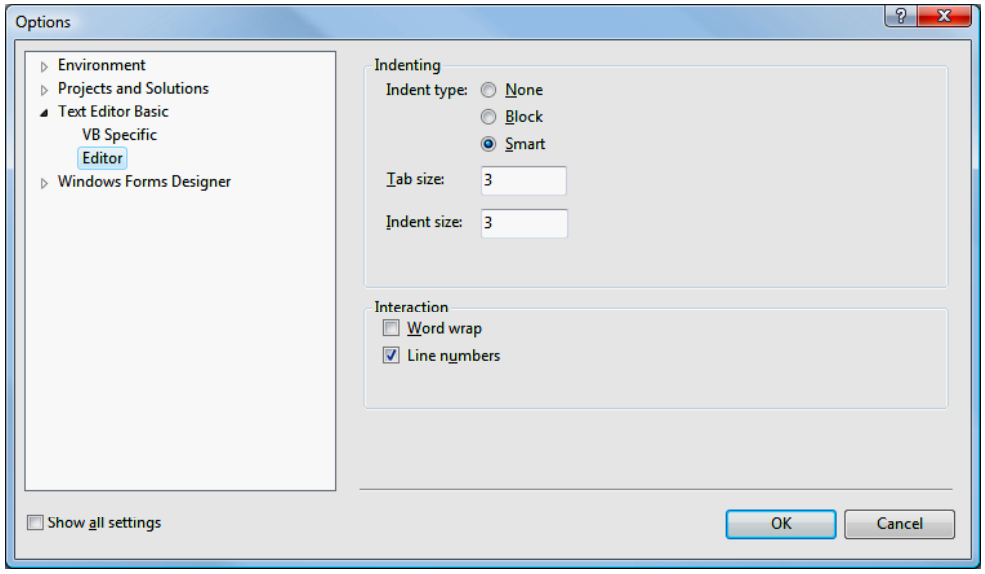


Fig. 3.5 | Modifying the IDE settings.

3.3 Creating Your First Program in Visual Basic Express (Cont.)

- Click `Module1.vb` in the **Solution Explorer** window to display its properties (Fig. 3.6).

- Change the **File Name** to `Welcome1.vb`.

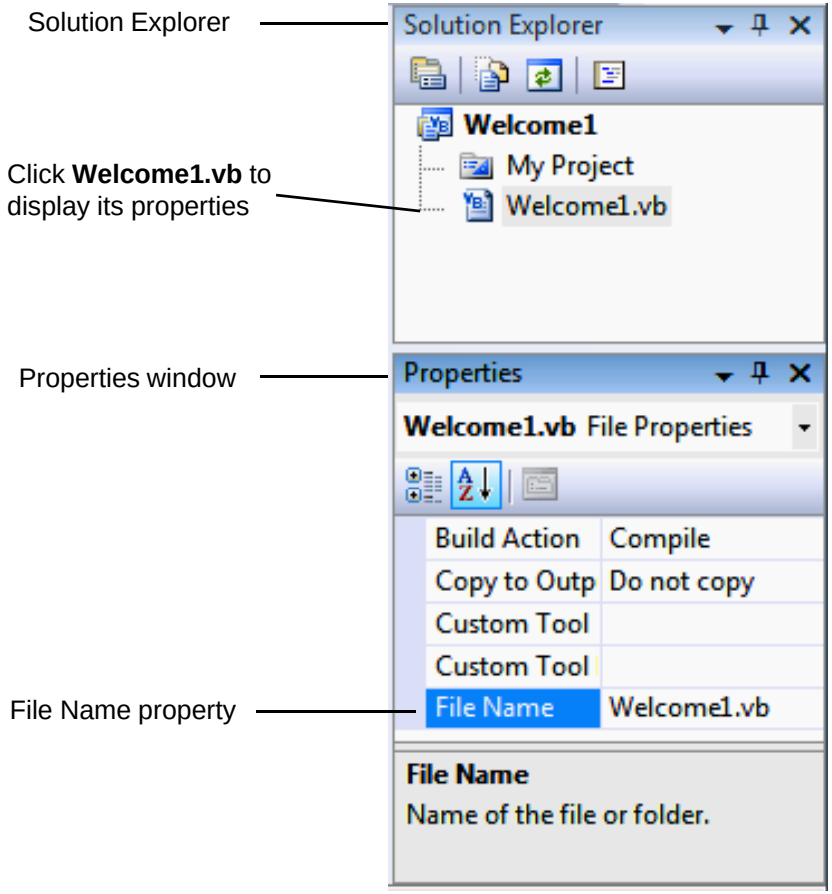


Fig. 3.6 | Renaming the program file in the **Properties** window.

3.3 Creating Your First Program in Visual Basic Express (Cont.)

- Each Visual Basic project has a **startup object**.
- In the **Solution Explorer** window, double click the **My Project** item.
- Select **Welcome1** from the **Startup object** drop-down list (Fig. 3.7).

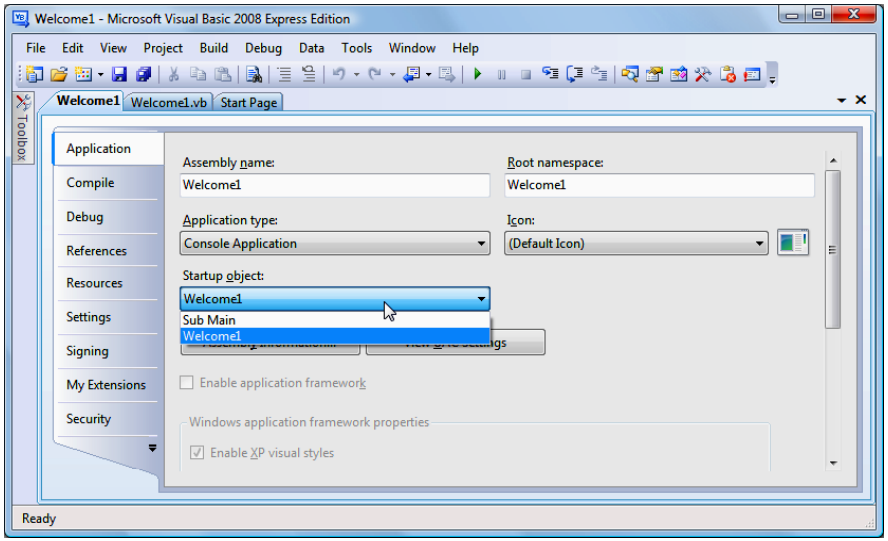


Fig. 3.7 | Setting the startup object.

3.3 Creating Your First Program in Visual Basic Express (Cont.)

- Click the **`Welcome1.vb`** tab in the IDE to view the editor.
- ***IntelliSense*** (Fig. 3.8) lists elements that start with the same characters you've typed so far.

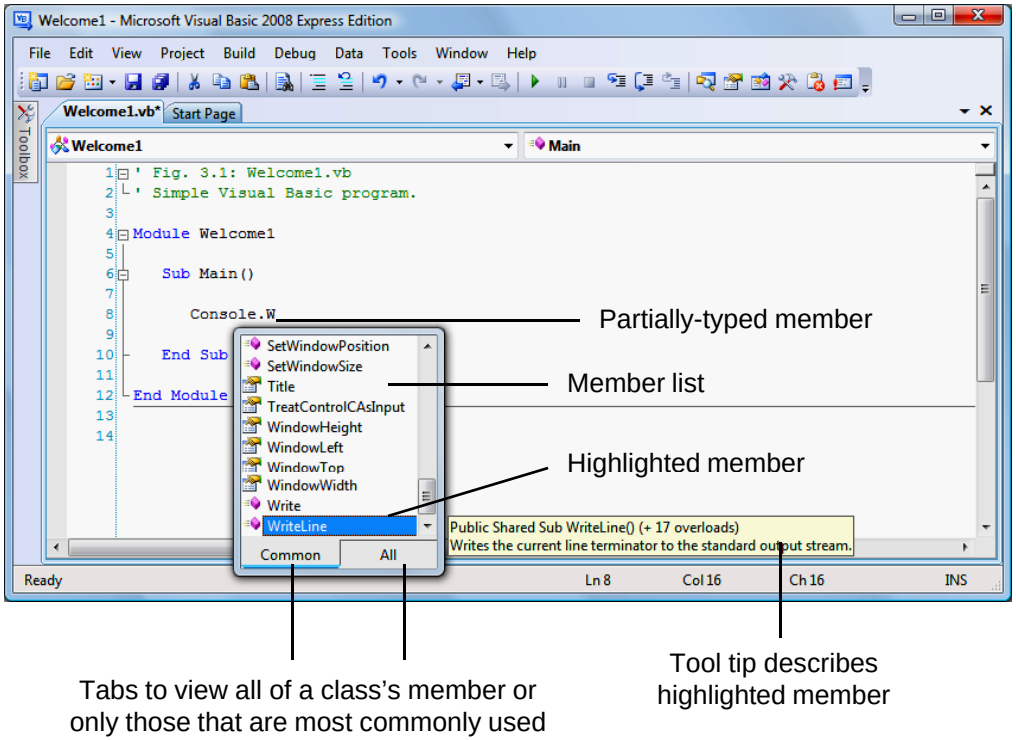


Fig. 3.8 | *IntelliSense* feature of Visual Basic Express.

3.3 Creating Your First Program in Visual Basic Express (Cont.)

- When you type `(` after `Console.WriteLine`, the ***Parameter Info*** window is displayed (Fig. 3.9).
  - This contains information about possible method parameters.
  - Arrows scroll through overloaded versions of the method.

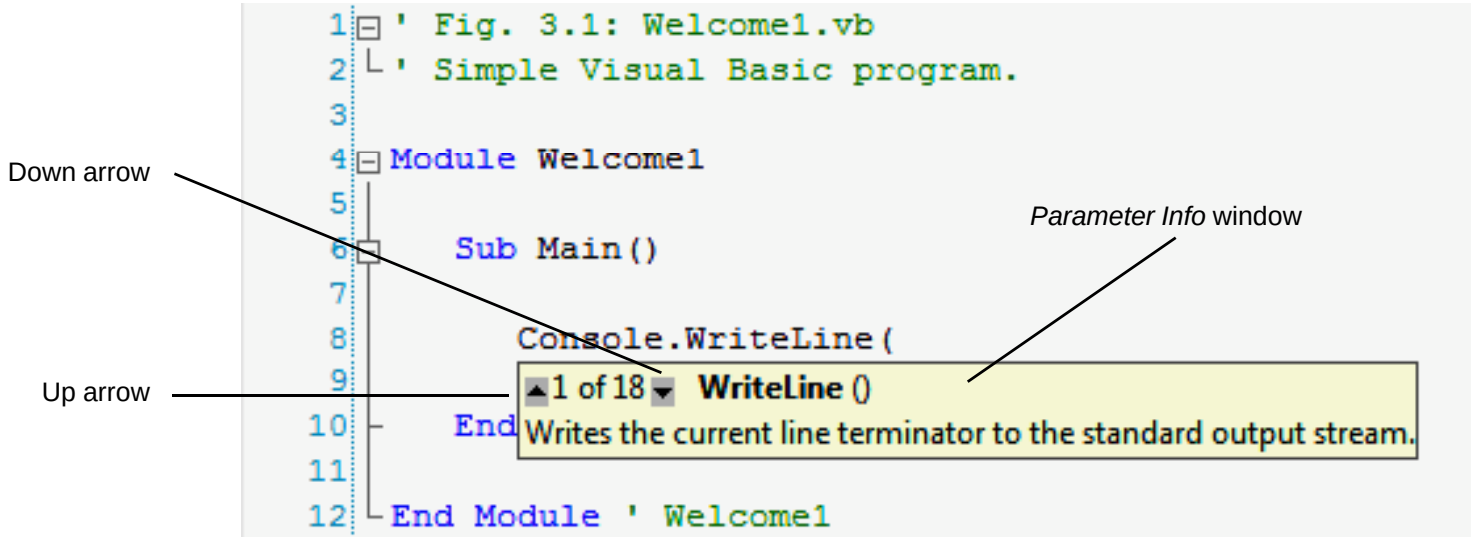


Fig. 3.9 | *Parameter Info* window.

3.3 Creating Your First Program in Visual Basic Express (Cont.)

- **File > Save All** to display the **Save Project** dialog (Fig. 3.10).

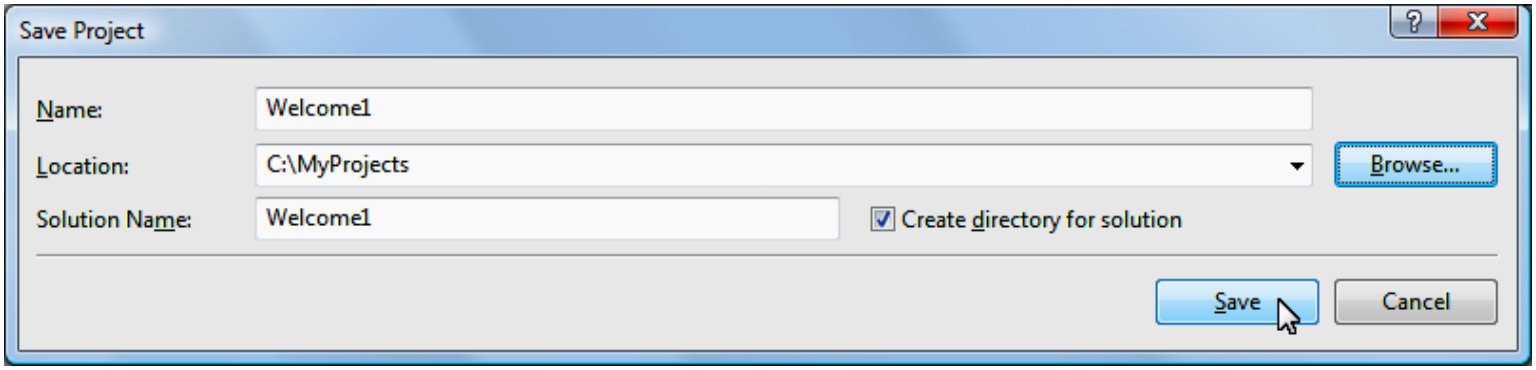


Fig. 3.10 | **Save Project** dialog.

3.3 Creating Your First Program in Visual Basic Express (Cont.)

- Select **Build > Build Welcome1**.
- Select **Debug > Start Debugging**.
- To enable the window to remain on the screen, type *Ctrl* + *F5* (Fig. 3.11).

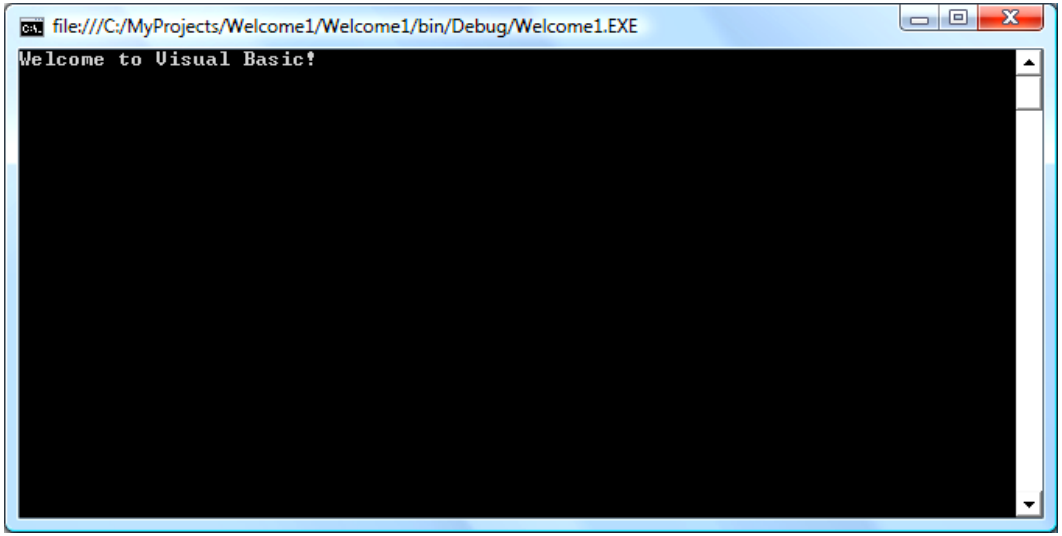


Fig. 3.11 | Executing the program shown in Fig. 3.1.

3.3 Creating Your First Program in Visual Basic Express (Cont.)

Running the Program from the ***Command Prompt***

- Click the Windows **Start** button ( ), then select **All Programs > Accessories > Command Prompt** (Fig. 3.12).
- **Command Prompt** windows normally have black backgrounds and white text.

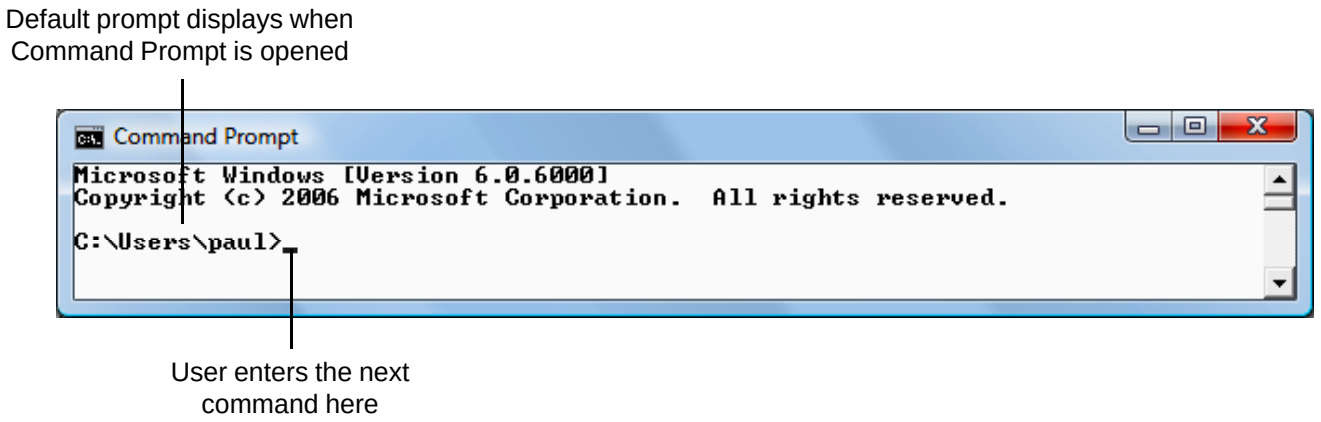


Fig. 3.12 | Executing the program shown in Fig. 3.1 from a **Command Prompt**.

3.3 Creating Your First Program in Visual Basic Express (Cont.)

- Enter the command `cd` followed by the directory where the application's `.exe` file is located.
- Enter the name of the `.exe` file to run the application.

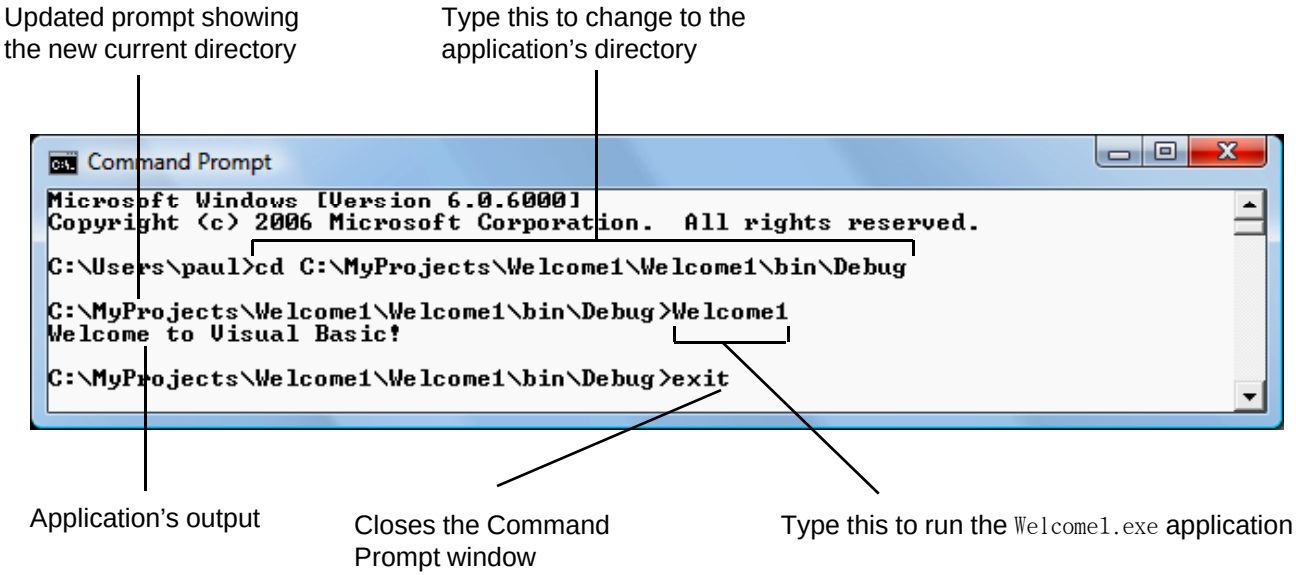


Fig. 3.13 | Executing the program shown in Fig. 3.1 from a **Command Prompt**.

3.3 Creating Your First Program in Visual Basic Express (Cont.)

- When you mistype a line of code, the IDE may generate a **syntax error**.
- The IDE underlines the error in blue and provides a description in the **Error List window**.
- Select **View > Error List** to view this window (Fig. 3.14).

3.3 Creating Your First Program in Visual Basic Express (Cont.)

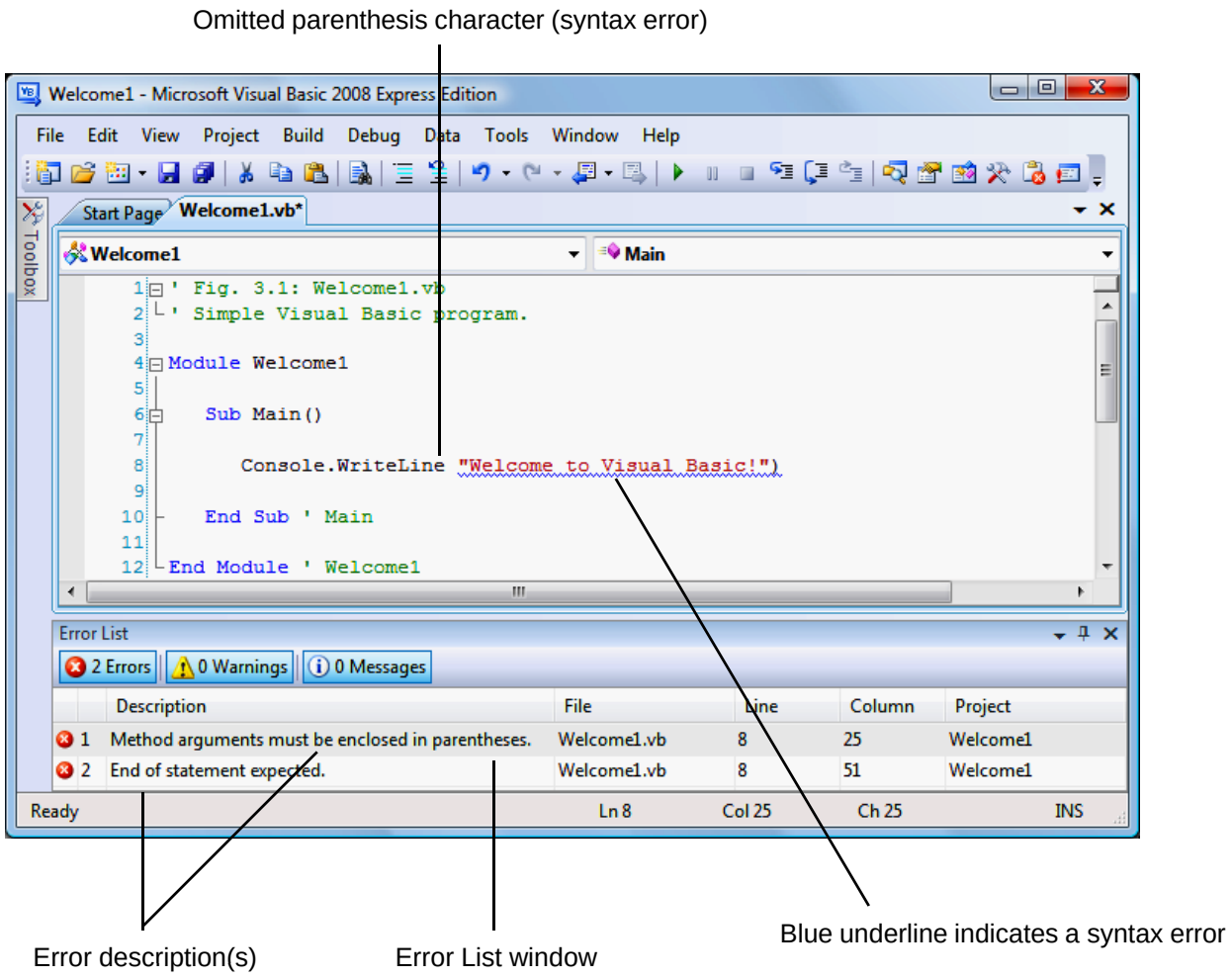


Fig. 3.14 | Syntax error indicated by the IDE.

3.3 Creating Your First Program in Visual Basic Express (Cont.)

Error-Prevention Tip 3.1

One syntax error can lead to multiple entries in the `Error List` window. Each error that you address could eliminate several subsequent error messages. So, when you see a particular error you know how to fix, correct it—this may make the other errors disappear.

Outline

- `Console` method `Write` positions the output to the right of the last character displayed (Fig. 3.15).

`Welcome2.vb`

```
1  ' Fig. 3.15: Welcome2.vb
2  ' Displaying a line of text with multiple statements.
3
4  Module Welcome2
5
6     Sub Main()
7
8        Console.Write("Welcome to ")
9        Console.WriteLine("Visual Basic!")
10
11    End Sub ' Main
12
13 End Module ' Welcome2
```

Output does not move to a new line.

Output appears after the last character displayed with `Write`.

```
Welcome to Visual Basic!
```

Fig. 3.15 | Displaying a line of text with multiple statements.

- **Declarations** begin with keyword `Dim` (Fig. 3.16).
  - number1, number2 and total are the names of **variables** of type `Integer`.

Outline

Addition.vb

(1 of 2)

```
1  ' Fig. 3.16: Addition.vb
2  ' Addition program.
3
4  Module Addition
5
6     Sub Main()
7
8        ' variables used in the addition calculation
9        Dim number1 As Integer
10       Dim number2 As Integer
11       Dim total As Integer
12
13       ' prompt for and read the first number from the user
14       Console.Write("Please enter the first integer: ")
15       number1 = Console.ReadLine()
```

Declaring **variables** of type `Integer`

The user is prompted to enter information.

`ReadLine` obtains a value entered by the user.

Fig. 3.16 | Addition program that adds two integers entered by the user. (Part 1 of 2.)

Outline

Addition.vb

(2 of 2)

```
16
17       ' prompt for and read the second number from the user
18       Console.Write("Please enter the second integer: ")
19       number2 = Console.ReadLine()
20
21       total = number1 + number2 ' add the numbers
22
23       Console.WriteLine("The sum is " & total) ' display the sum
24
25    End Sub ' Main
26
27 End Module ' Addition
```

```
Please enter the first integer: 45
Please enter the second integer: 72
The sum is 117
```

Fig. 3.16 | Addition program that adds two integers entered by the user. (Part 2 of 2.)

3.5 Adding Integers (Cont.)

- Types already defined in Visual Basic are keywords known as **primitive types** (Fig. 3.17).

Primitive Types				
`Boolean`	`Byte`	`Char`	`Date`	`Decimal`
`Double`	`Integer`	`Long`	`SByte`	`Short`
`Single`	`String`	`UInteger`	`ULong`	`UShort`

Fig. 3.17 | Primitive Types in Visual Basic

3.5 Adding Integers (Cont.)

Good Programming Practice 3.3

Choosing meaningful variable names helps a program to be "self-documenting"—the program can be understood by others without the use of documentation manuals or excessive comments.

Good Programming Practice 3.4

A common convention is to have the first word in a variable-name identifier begin with a lowercase letter. Every word in the name after the first word should begin with a uppercase letter. Using these conventions helps make your programs more readable.

Good Programming Practice 3.5

Declaring each variable on a separate line allows for easy insertion of an end-of-line comment next to each declaration. We follow this convention.

3.5 Adding Integers (Cont.)

- A **prompt** directs the user to take a specific action.
- `ReadLine` causes the program to pause and wait for user input.
- The number is assigned to variable `number1` by an **assignment**, `=`.

Good Programming Practice 3.6

The Visual Basic IDE places a space on either side of a binary operator to make the operator stand out and improve the readability of the statement.

3.5 Adding Integers (Cont.)

- If the user types a non-integer value, a **run-time error** occurs.
- An error message is displayed (Fig. 3.18) if you ran the application using **Debug > Start Debugging**.

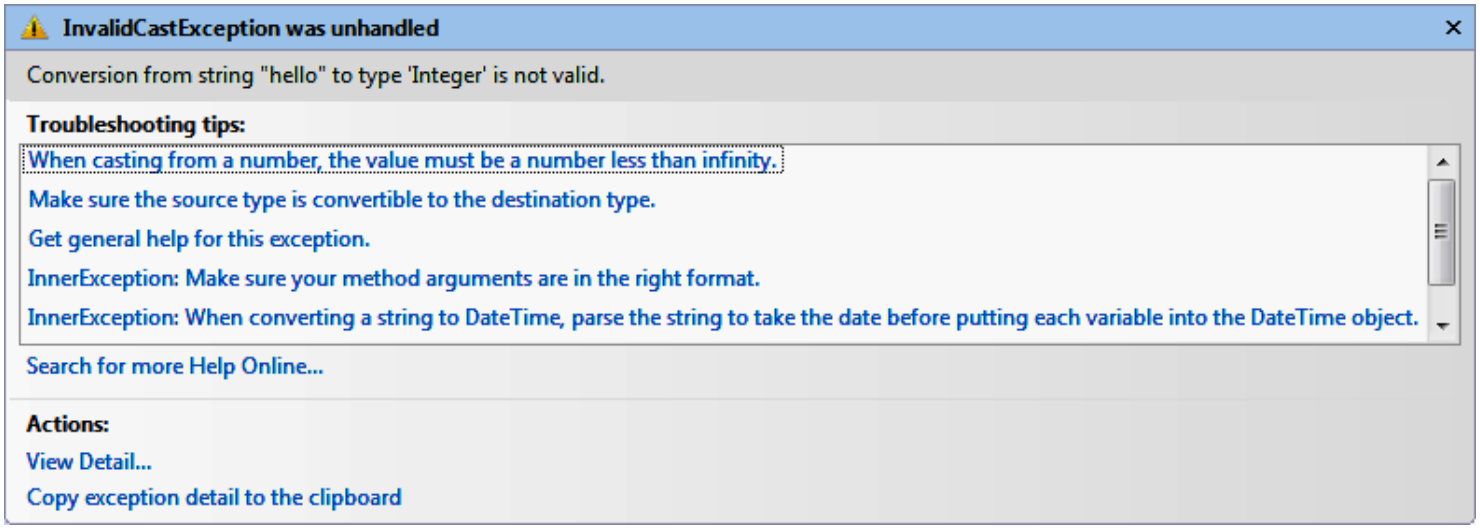


Fig. 3.18 | Dialog displaying a run-time error.

3.5 Adding Integers (Cont.)

- The **string concatenation operator**, **&**, is used to combine values into strings.

```
"The sum is " & total
```

- The string concatenation operator is called a **binary operator**.

Good Programming Practice 3.7

Follow a method's End Sub with an end-of-line comment containing the name of the method that the End Sub terminates.

3.6 Memory Concepts

- Variable names correspond to locations in memory.

```
number1 = Console.ReadLine()
```

- Input data is placed into a memory location to which the name `number1` has been assigned (Fig. 3.19).

Fig. 3.19 | Memory location showing name and value of variable `number1`.

3.6 Memory Concepts (Cont.)

- Whenever a value is placed in a memory location, this value replaces the value previously stored in that location.
- Suppose that the user enters `72`:

```
number2 = Console.ReadLine()
```

- The `Integer` value `72` is placed into location `number2`, and memory appears (Fig. 3.20)

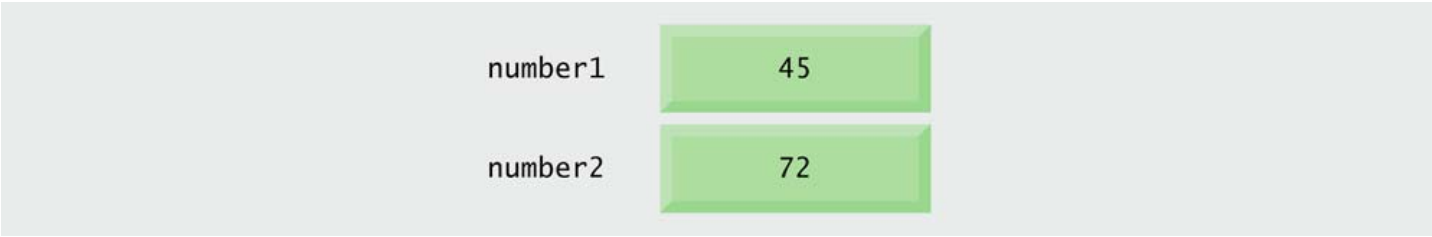


Fig. 3.20 | Memory locations after values for variables `number1` and `number2` have been input.

3.6 Memory Concepts (Cont.)

- The program adds `number1` and `number2` and places their total into variable `total`.

```
total = number1 + number2
```

- After total is calculated, memory appears (Fig. 3.21).
- The values of number1 and number2 appear exactly as they did before the calculation.

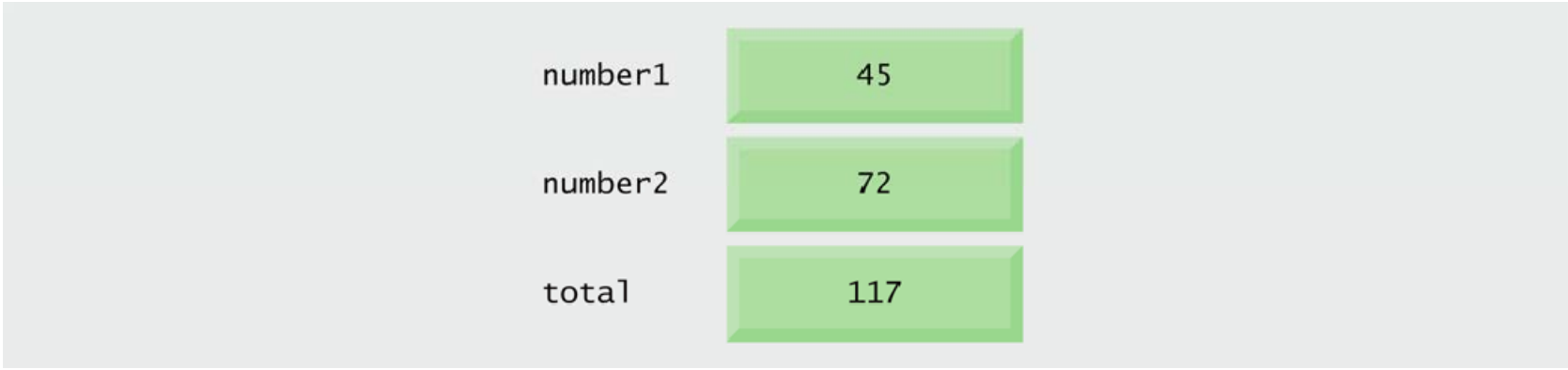


Fig. 3.21 | Memory locations after an addition operation.

3.7 Arithmetic

- **Arithmetic operators** are summarized in Fig. 3.22.
- Some of the symbols are not used in algebra.

Visual Basic operation	Arithmetic operator	Algebraic expression	Visual Basic expression
Addition	`+`	$f + 7$	`f + 7`
Subtraction	`-`	$p - c$	`p - c`
Multiplication	`*`	bm	`b * m`
Division (floating point)	`/`	x/y or $\frac{x}{y}$ or $x \div y$	`x / y`
Division (integer)	`\`	none	`v \ u`
Modulus	`Mod`	$r \bmod s$	`r Mod s`
Exponentiation	`^`	q^p	`q ^ p`
Unary minus	`-`	$-e$	`-e`
Unary plus	`+`	$+g$	`+g`

Fig. 3.22 | Arithmetic operators.

3.7 Arithmetic (Cont.)

- Integer division takes two `Integer` operands and yields an `Integer` result.
  - **Floating-point number** operands are rounded to the nearest whole number.
  - Any fractional part in the result is discarded—not rounded.
- The `Mod` operator yields the remainder after division.
- Expressions such as $\frac{a}{b}$ must be written as `a / b` to appear in a straight line.

3.7 Arithmetic (Cont.)

Common Programming Error 3.2

Using the integer division operator (\) when the floating-point division operator (/) is expected can lead to incorrect results.

Error-Prevention Tip 3.2

Ensure that each integer division operator has only integer operands.

3.7 Arithmetic (Cont.)

Operator(s)	Operation	Order of evaluation (precedence)
^	Exponentiation	Evaluated first. If there are several such operators, they are evaluated from left to right.
+, –	Sign operations	Evaluated second. If there are several such operators, they are evaluated from left to right.
*, /	Multiplication and Division	Evaluated third. If there are several such operators, they are evaluated from left to right.
\	Integer division	Evaluated fourth. If there are several such operators, they are evaluated from left to right.
Mod	Modulus	Evaluated fifth. If there are several such operators, they are evaluated from left to right.
+, –	Addition and Subtraction	Evaluated sixth. If there are several such operators, they are evaluated from left to right.

Fig. 3.23 | Precedence of arithmetic operators.

3.7 Arithmetic (Cont.)

- Consider several expressions with the rules of operator precedence:

Algebra:

$$m = \frac{a + b + c + d + e}{5}$$

Visual Basic:

```
m = ( a + b + c + d + e ) / 5
```

- Parentheses are required because floating-point division has higher precedence than addition.

$$a + b + c + d + \frac{e}{5}$$

3.7 Arithmetic (Cont.)

- The following is the equation of a straight line:

Algebra: $y = mx + b$

Visual Basic:
```
y = m * x + b
```

- No parentheses are required because multiplication is applied first.

3.7 Arithmetic (Cont.)

- The circled numbers under the statement indicate the order.

Algebra: $z = pr \bmod q + w/x - y$

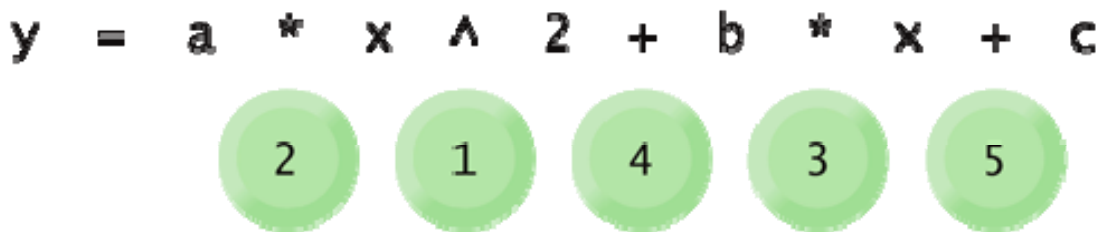
Visual Basic:

```
z = p * r Mod q + w / x - y
```

1 3 4 2 5

3.7 Arithmetic (Cont.)

- Consider how $y = ax^2 + bx + c$ is evaluated:.



3.7 Arithmetic (Cont.)

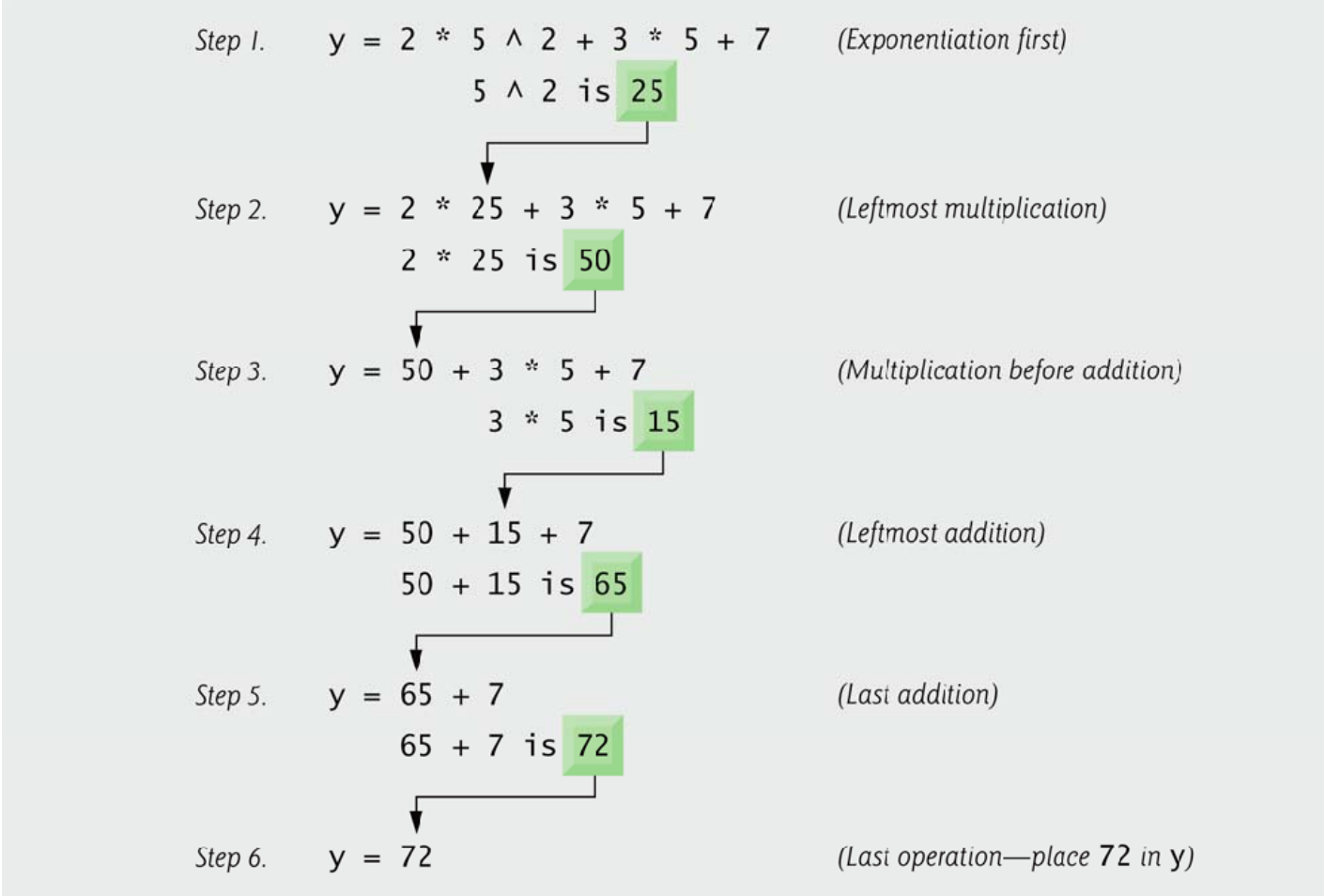


Fig. 3.24 | Order in which a second-degree polynomial is evaluated.

3.7 Arithmetic (Cont.)

Good Programming Practice 3.8

Redundant parentheses can make complex expressions easier to read.

Error-Prevention Tip 3.3

When you are uncertain about the order of evaluation in a complex expression, use parentheses to force the order, as you would in an algebraic expression. Doing so can help avoid subtle bugs

3.8 Decision Making: Equality and Relational Operators

- The `If...Then` statement allows a program to make a decision based on the truth or falsity of a **condition.**
  - If the condition is met, the statement in the `If...Then` statement's body executes.
  - Conditions can be formed by using **equality operators** and **relational operators**.

3.8 Decision Making: Equality and Relational Operators (Cont.)

- The equality and relational operators are summarized in Fig. 3.25.

Standard algebraic equality operator or relational operator	Visual Basic equality or relational operator	Example of Visual Basic condition	Meaning of Visual Basic condition
Equality operators			
=	=	x = y	x is equal to y
≠	<>	x <> y	x is not equal to y
Relational operators			
>	>	x > y	x is greater than y
<	<	x < y	x is less than y
≥	>=	x >= y	x is greater than or equal to y
≤	<=	x <= y	x is less than or equal to y

Fig. 3.25 | Equality and relational operators.

3.8 Decision Making: Equality and Relational Operators (Cont.)

Common Programming Error 3.3

It is a syntax error to reverse the symbols in the operators <>, >= and <= (as in ><, =>, =<). The Visual Basic IDE fixes these errors as you type.

Outline

- The code of Fig. 3.26 compares two numbers.

Comparison.vb

(1 of 4)

```
1  ' Fig. 3.26: Comparison.vb
2  ' Using equality and relational operators.
3
4  Module Comparison
5
6     Sub Main()
7
8        ' declare Integer variables for user input
9        Dim number1 As Integer
10       Dim number2 As Integer
11
12       ' read first number from user
13       Console.Write("Please enter first integer: ")
14       number1 = Console.ReadLine()
15
```

Input assigned to `Integer` variable `number1`

Fig. 3.26 | Performing comparisons with equality and relational operators. (Part 1 of 4.)

Outline

Comparison.vb

(2 of 4)

```
16        ' read second number from user
17        Console.Write("Please enter second integer: ")
18        number2 = Console.ReadLine()
19
20        If number1 = number2 Then ' number1 is equal to number2
21           Console.WriteLine(number1 & " = " & number2)
22        End If
23
24        If number1 <> number2 Then ' number1 is not equal to number2
25           Console.WriteLine(number1 & " <> " & number2)
26        End If
27
28        If number1 < number2 Then ' number1 is less than number2
29           Console.WriteLine(number1 & " < " & number2)
30        End If
31
32        If number1 > number2 Then ' number1 is greater than number2
33           Console.WriteLine(number1 & " > " & number2)
34        End If
```

Comparing `number1` and `number2` for equality

Fig. 3.26 | Performing comparisons with equality and relational operators. (Part 2 of 4.)

Outline

Comparison.vb

(3 of 4)

```
35
36       ' number1 is less than or equal to number2
37       If number1 <= number2 Then
38          Console.WriteLine(number1 & " <= " & number2)
39       End If
40
41       ' number1 is greater than or equal to number2
42       If number1 >= number2 Then
43          Console.WriteLine(number1 & " >= " & number2)
44       End If
45
46    End Sub ' Main
47
48 End Module ' Comparison
```

```
Please enter first integer: 1000
Please enter second integer: 2000
1000 <> 2000
1000 < 2000
1000 <= 2000
```

(continued on next page...)

Fig. 3.26 | Performing comparisons with equality and relational operators. (Part 3 of 4.)

Outline

(continued from previous page…)

```
Please enter first integer: 515
Please enter second integer: 49
515 <> 49
515 > 49
515 >= 49
```

```
Please enter first integer: 333
Please enter second integer: 333
333 = 333
333 <= 333
333 >= 333
```

Comparison.vb

(4 of 4)

Fig. 3.26 | Performing comparisons with equality and relational operators. (Part 4 of 4.)

Good Programming Practice 3.9

Visual Basic indents the statements in the body of an `If…Then` statement to emphasize the body statements and enhance program readability. You should also follow this convention when programming in other languages.

3.8 Decision Making: Equality and Relational Operators (Cont.)

- Figure 3.27 shows operators displayed in decreasing order of precedence.

Operators	Type
^	exponentiation
+ -	sign operations (unary)
* /	multiplication and floating-point division
\	`Integer` division
`Mod`	modulus
+ -	addition and subtraction (binary)
= <> < <= > >=	equality and relational

Fig. 3.27 | Precedence of the operators introduced in this chapter.

Outline

- **Message dialogs** are windows that display messages to the user.
- Class `MessageBox` is used to create message dialogs (Fig. 3.28).

SquareRoot.vb

(1 of 2)

```
1  ' Fig. 3.28: SquareRoot.vb
2  ' Displaying the square root of 2 in a dialog.
3
4  Imports System.Windows.Forms ' Namespace containing class MessageBox
5
6  Module SquareRoot
7
8     Sub Main()
9
10       Dim root As Double = Math.Sqrt(2) ' calculate the square root of 2
```

The `Sqrt` method of class `Math` computes the square root.

Fig. 3.28 | Displaying text in a message dialog. (Part 1 of 2.)

Outline

SquareRoot.vb

(2 of 2)

```
11
12       ' display the results in a message dialog
13       MessageBox.Show("The square root of 2 is " & root, _
14          "The Square Root of 2")
15
16    End Sub ' Main
17
18 End Module ' SquareRoot
```

Using the **line-continuation character**

Method `Show` displays the message dialog

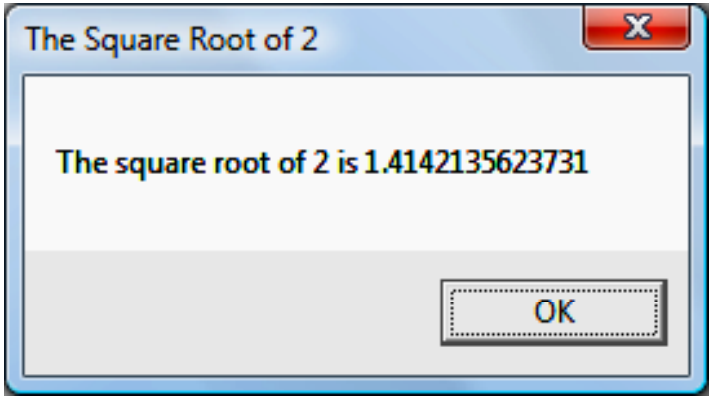


Fig. 3.28 | Displaying text in a message dialog. (Part 2 of 2.)

3.9 Using a Message Dialog to Display a Message (Cont.)

- .NET Framework Class Library classes are grouped into **namespaces**.
- An **`Imports` statement** enables features from another namespace.
- The `Sqrt` method of class `Math` computes the square root.
- Statements may be split over several lines using the line-continuation character, `_` .

3.9 Using a Message Dialog to Display a Message (Cont.)

Common Programming Error 3.4

Splitting a statement over several lines without including the line-continuation character is usually a syntax error.

Common Programming Error 3.5

Failure to precede the line-continuation character (_) with at least one whitespace character is a syntax error.

Common Programming Error 3.6

Placing anything, including comments, on the same line after a line-continuation character is a syntax error.

3.9 Using a Message Dialog to Display a Message (Cont.)

Common Programming Error 3.7

Splitting a statement in the middle of an identifier or string is a syntax error.

Good Programming Practice 3.10

If a single statement must be split across lines, choose breaking points that make sense, such as after a comma in a comma-separated list or after an operator in a lengthy expression. If a statement is split across two or more lines, indent all subsequent lines with one level of indentation.

Good Programming Practice 3.11

Visual Basic places a space after each comma in a method's argument list to make method calls more readable.

3.9 Using a Message Dialog to Display a Message (Cont.)

Analyzing the ***`MessageBox`***

- The message dialog (Fig. 3.29) gives a message to the user.

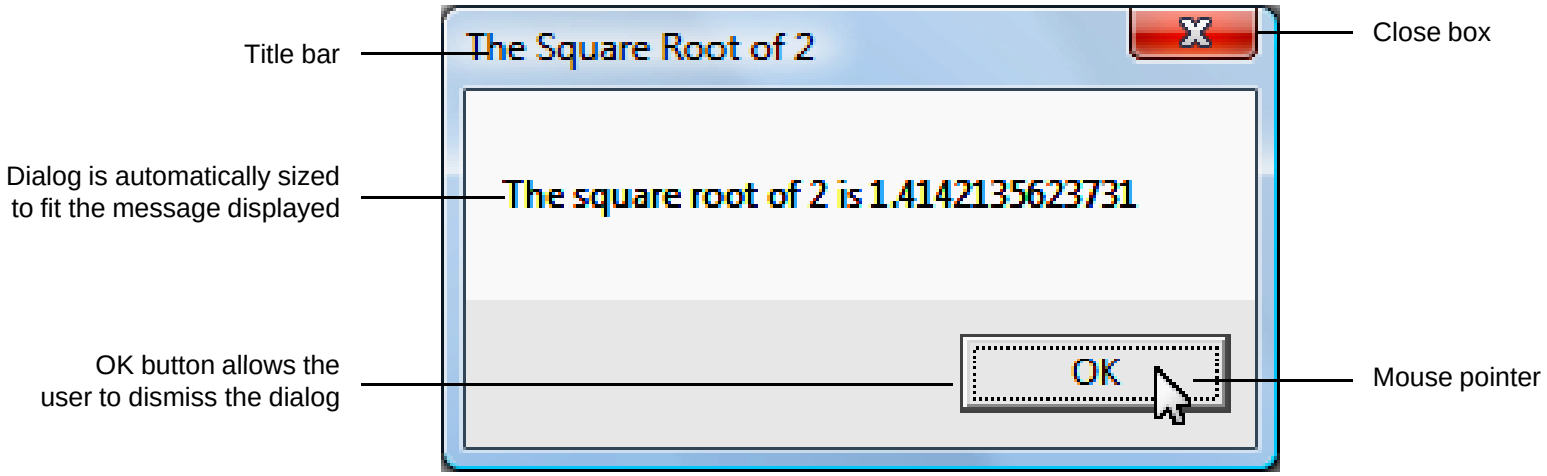


Fig. 3.29 | Message dialog displayed by calling `MessageBox.Show`.

3.9 Using a Message Dialog to Display a Message (Cont.)

- Some classes in the .NET Framework must be added to the project.
- These classes are located in an **assembly** file, that has a `.dll` (**dynamic link library**) file extension.
- Select **Help > Index** (Fig. 3.30).
- Type the class name `MessageBox` in the **Look for:** box, and **filter** by **.NET Framework**.

3.9 Using a Message Dialog to Display a Message (Cont.)

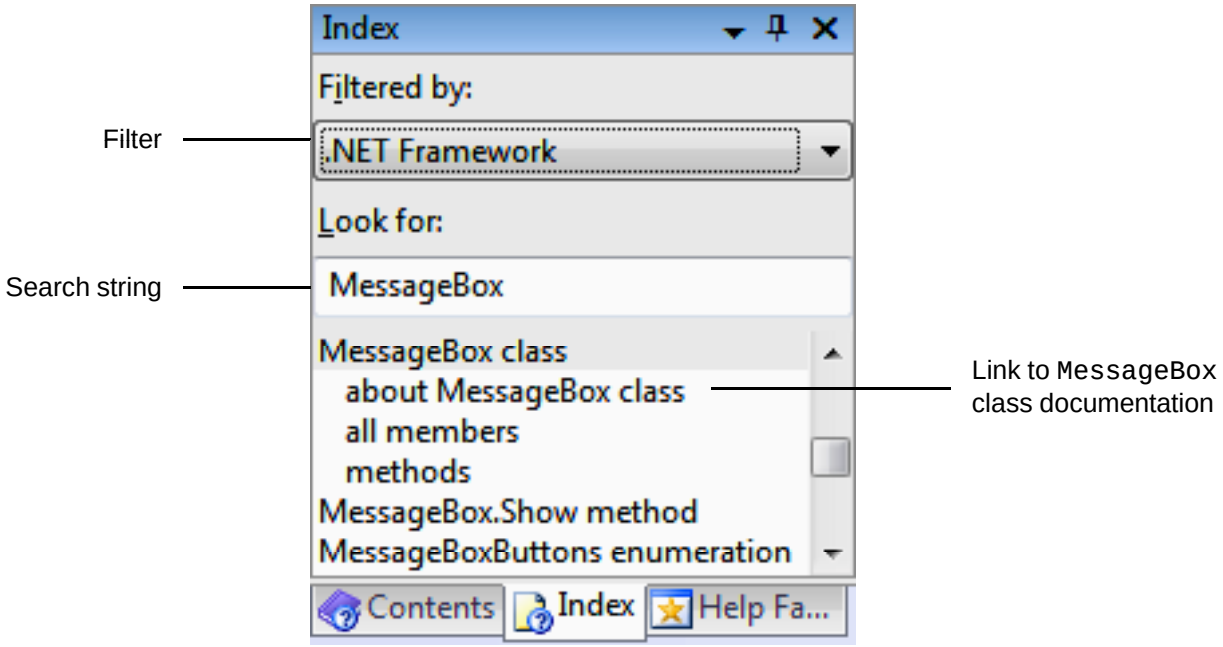


Fig. 3.30 | Obtaining documentation for a class by using the **Index** dialog.

3.9 Using a Message Dialog to Display a Message (Cont.)

- Click the **about MessageBox class** link (Fig. 3.31).
- The documentation lists the assembly that contains the class: `System.Windows.Forms.dll`.

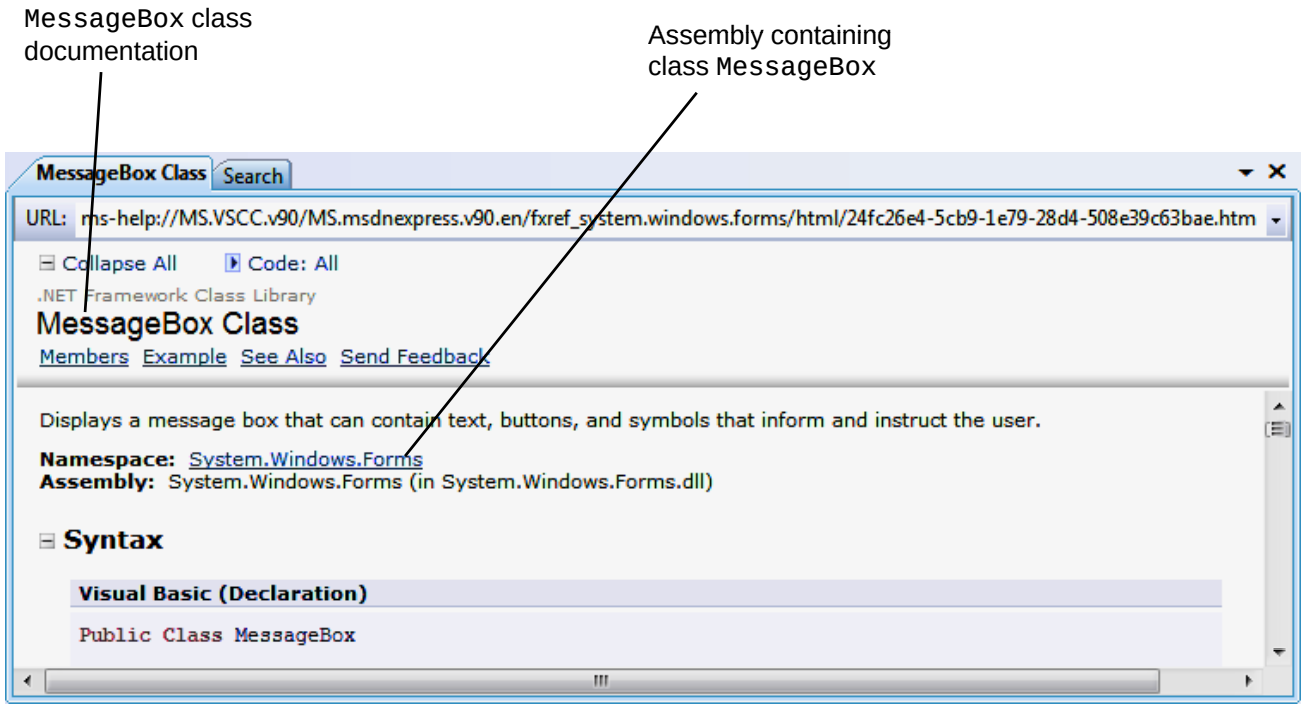


Fig. 3.31 | Documentation for the `MessageBox` class.

3.9 Using a Message Dialog to Display a Message (Cont.)

- **Add a reference** to this assembly to use class `MessageBox`.
  - Click the **Show All Files** button of the **Solution Explorer** (Fig. 3.32).
  - Expand the **References** folder.

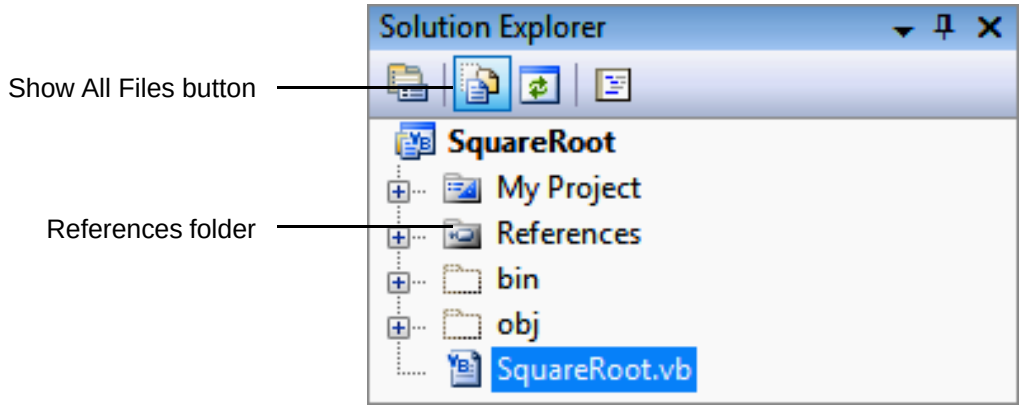


Fig. 3.32 | Viewing a project's references.

3.9 Using a Message Dialog to Display a Message (Cont.)

- Select **Project > Add Reference...** and select **Add Reference** to display the **Add Reference dialog (Fig. 3.33).**
- In the **.NET** tab, select `System.Windows.Forms.dll`.

a) Add **Reference** dialog displayed when you select Project > Add Reference...

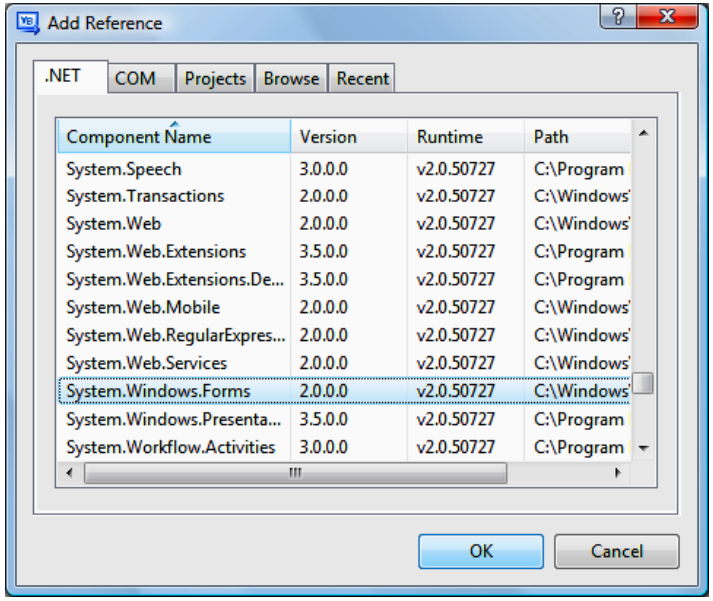


Fig. 3.33 | Adding an assembly reference to a project in the Visual Basic 2008 Express IDE. (Part 1 of 2.)

3.9 Using a Message Dialog to Display a Message (Cont.)

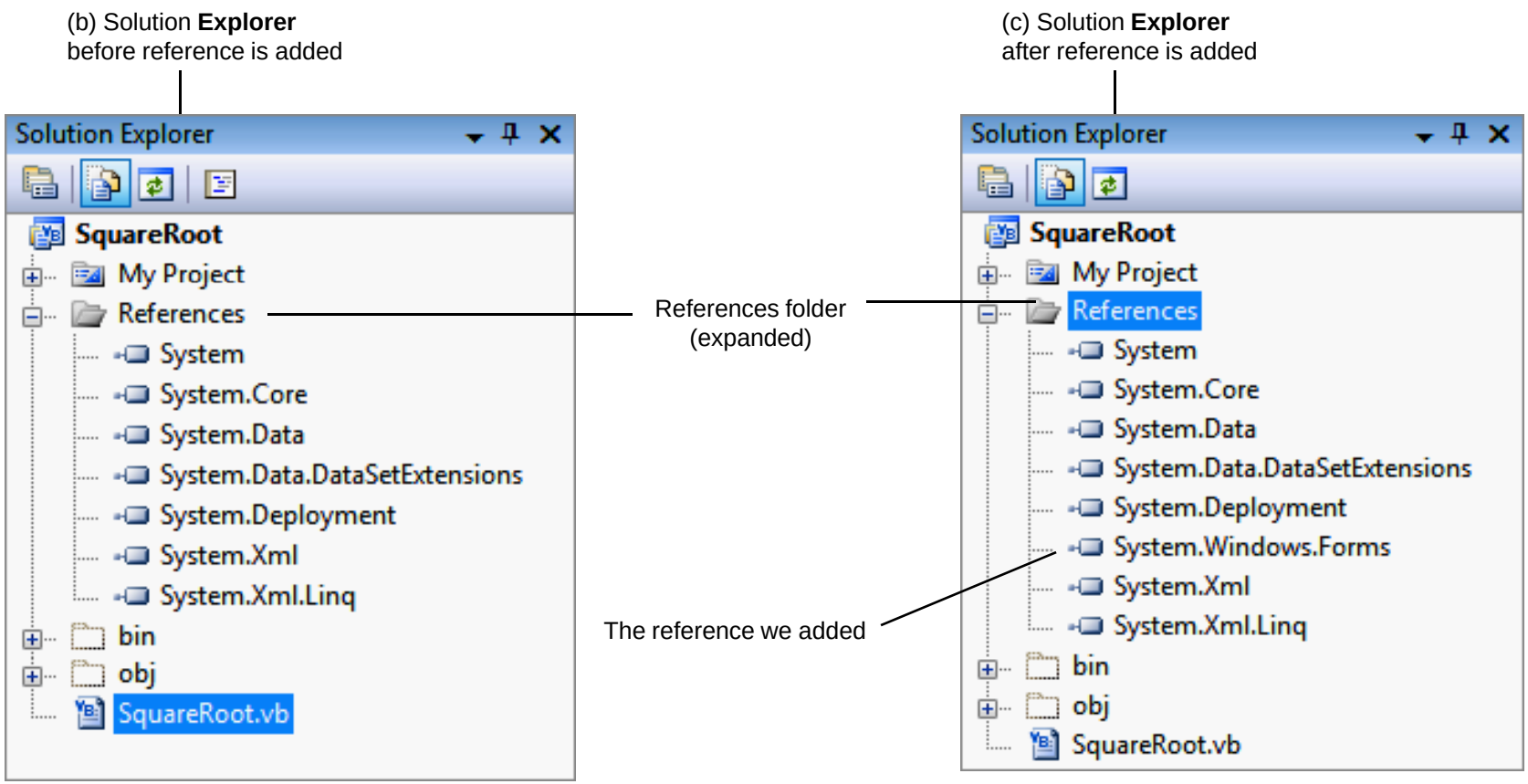


Fig. 3.33 | Adding an assembly reference to a project in the Visual Basic 2008 Express IDE. (Part 2 of 2.)

3.9 Using a Message Dialog to Display a Message (Cont.)

Common Programming Error 3.8

Including a namespace with the `Imports` statement without adding a reference to the proper assembly results in a compilation error.

3.9 Using a Message Dialog to Display a Message (Cont.)

- Figure 3.34 is a Mozilla Firefox browser window with several **GUI components**.

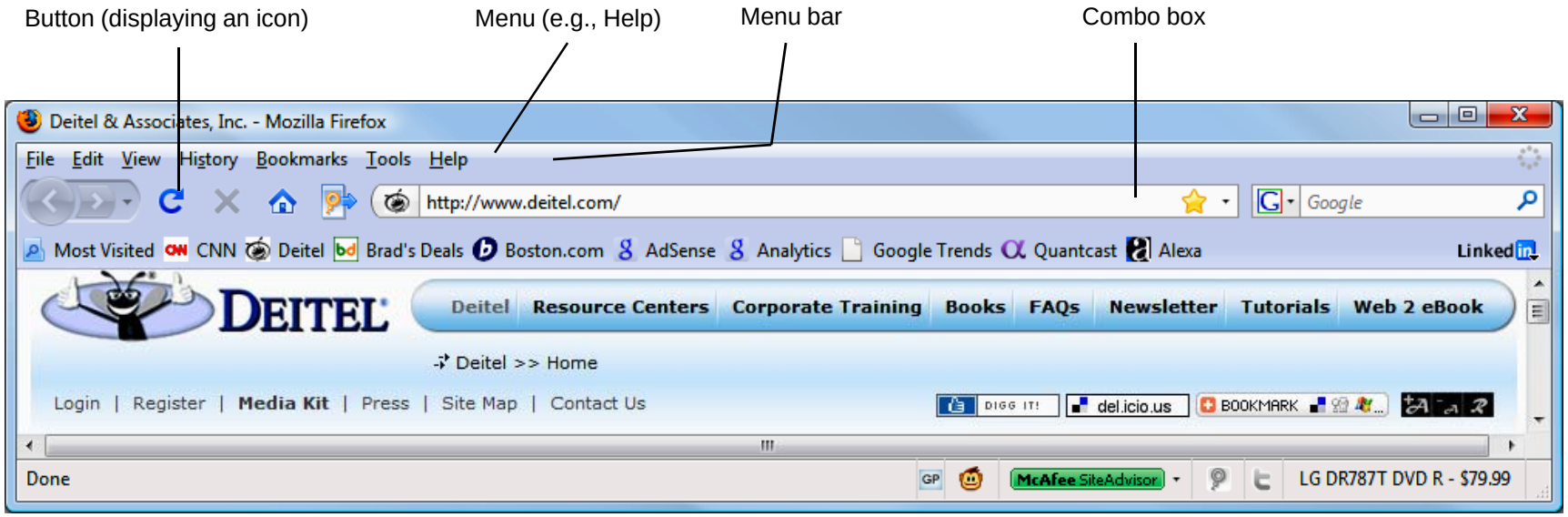


Fig. 3.34 | Mozilla Firefox window with GUI components.

3.10 Software Engineering Case Study: Examining the ATM Requirements Document

- A **requirements document** specifies the system's purpose.
- A bank intends to install a new ATM (Fig. 3.35).

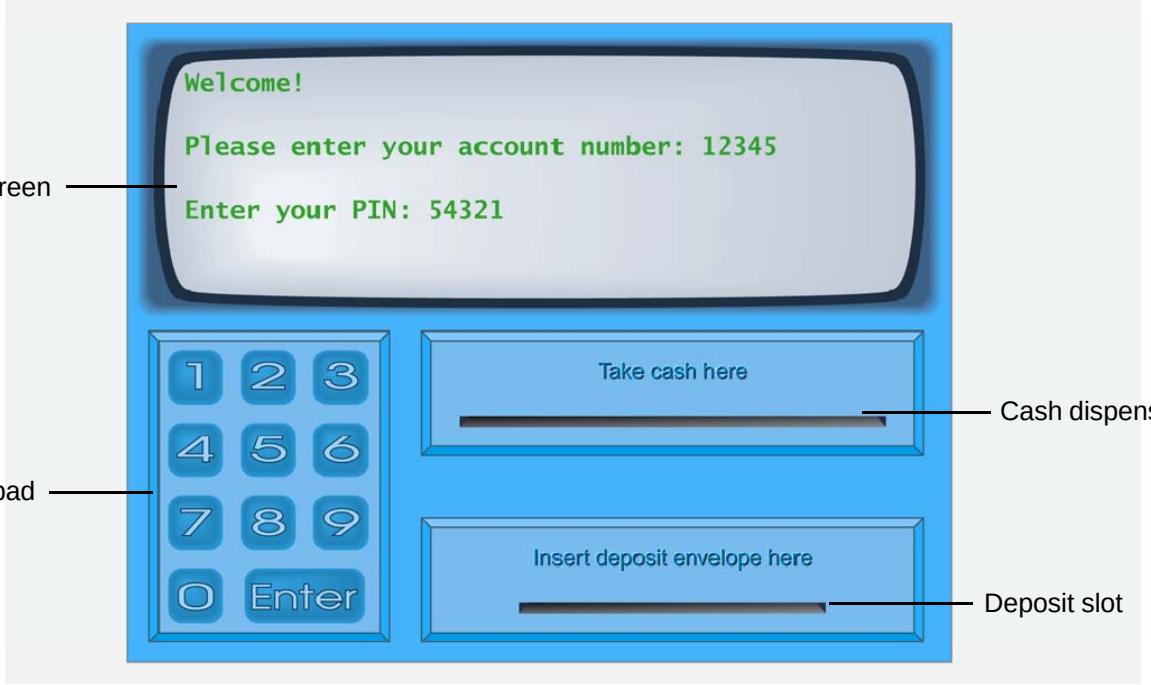


Fig. 3.35 | Automated teller machine user interface.

3.10 Software Engineering Case Study: Examining the ATM Requirements Document (Cont.)

- The software will simulate the functionality of the hardware devices.
  - The screen prompts the user to enter an account number.
  - The screen prompts the user to enter the PIN associated with the specified account number.
  - If the user enters valid input, the screen displays the main menu.

3.10 Software Engineering Case Study: Examining the ATM Requirements Document (Cont.)

- The main menu (Fig. 3.36) displays a numbered option for each of the three types of transaction.

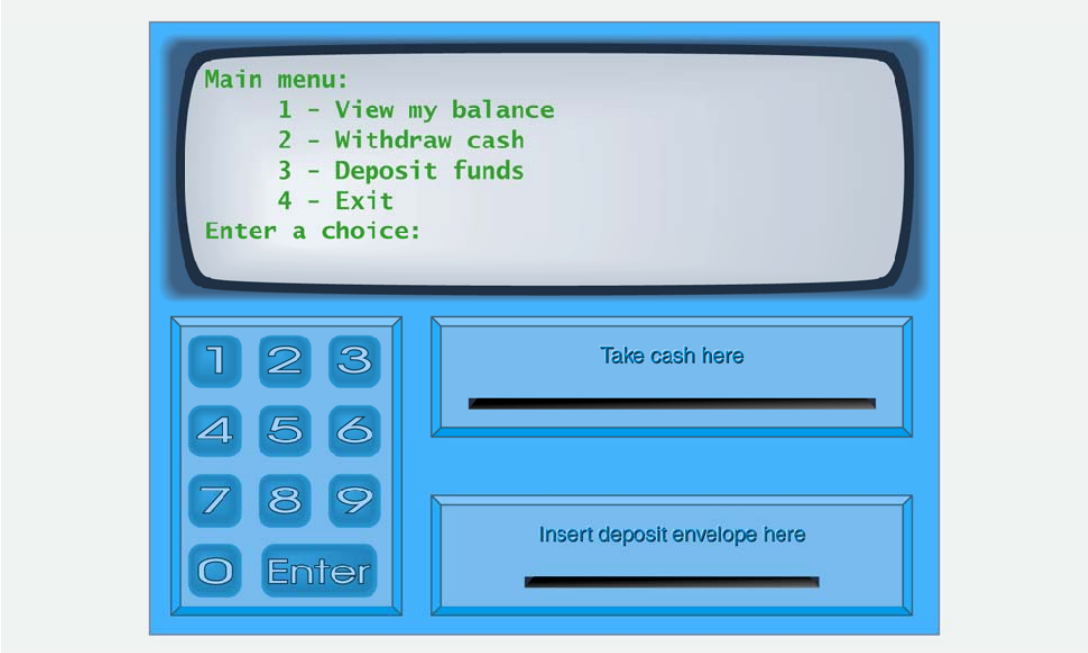


Fig. 3.36 | ATM main menu.

3.10 Software Engineering Case Study: Examining the ATM Requirements Document (Cont.)

- If the user enters 1, the screen obtains the user's account balance from the bank's database.
- The user enters 2 to make a withdrawal (Fig. 3.37).

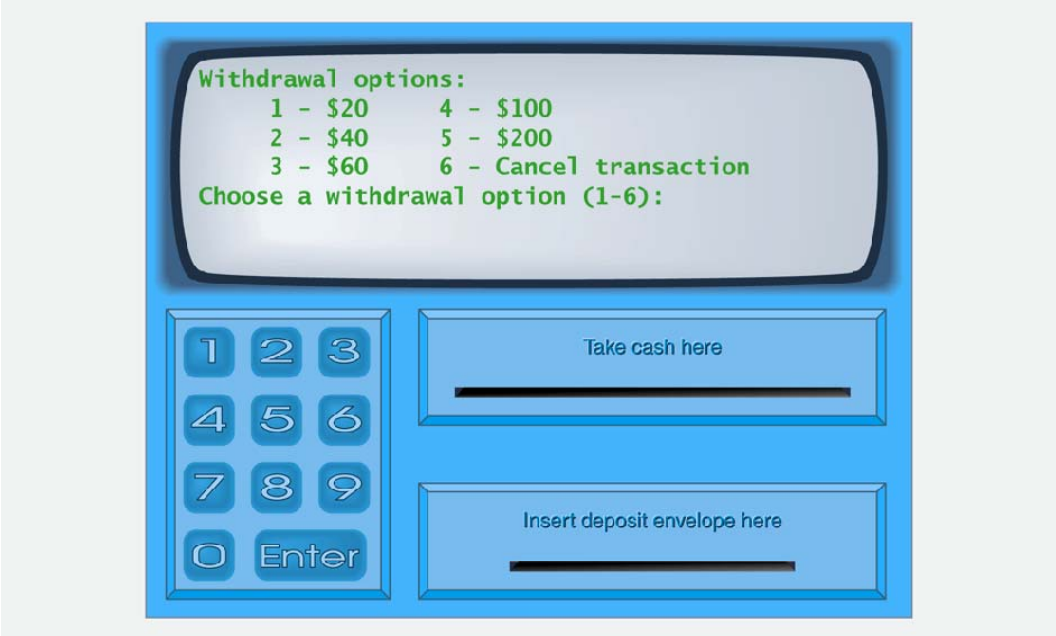


Fig. 3.37 | ATM withdrawal menu.

3.10 Software Engineering Case Study: Examining the ATM Requirements Document (Cont.)

- When the user enters 3 to make a deposit:
  - The screen prompts the user to enter an amount.
  - The user is told to insert a deposit envelope.
  - If the deposit slot receives a deposit envelope, the ATM credits the user's account balance.

3.10 Software Engineering Case Study: Examining the ATM Requirements Document (Cont.)

- **Requirements gathering** might include interviews with potential users and specialists.
- The **software life cycle** specifies the stages from the time it is conceived to the time at which it is retired from use.
  - **Waterfall models** perform each stage once in succession
  - **Iterative models** may repeat one or more stages several times throughout a product's life cycle.

3.10 Software Engineering Case Study: Examining the ATM Requirements Document (Cont.)

- A **use case diagram** (Fig. 3.38) models the interactions between a system's clients and the system.
- The stick figure represents the role of an **actor**, which interacts with the system.

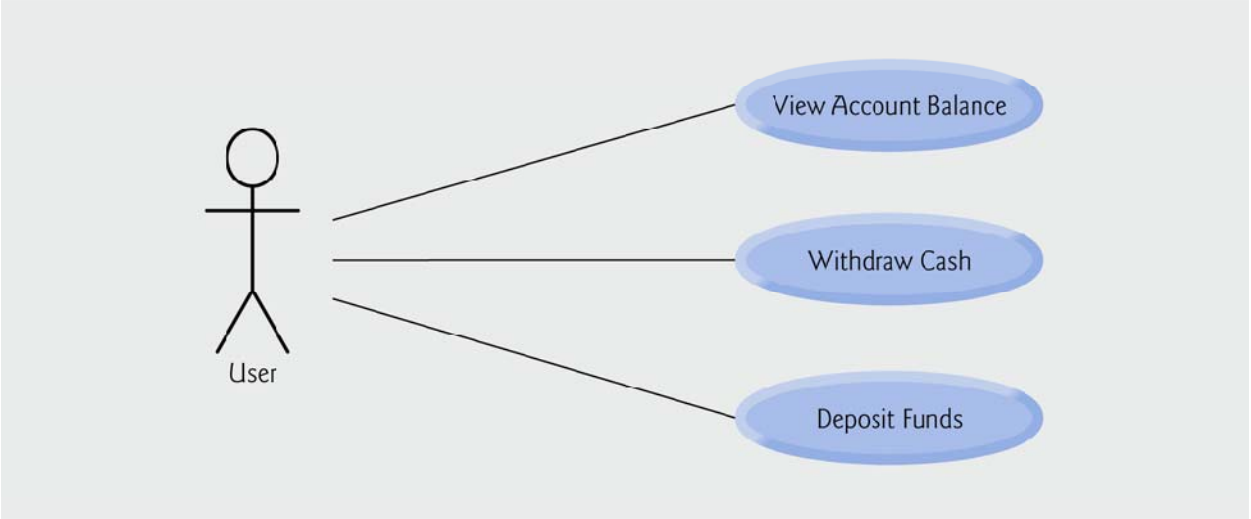


Fig. 3.38 | Use case diagram for the ATM system from the user's perspective.

3.10 Software Engineering Case Study: Examining the ATM Requirements Document (Cont.)

- A **system** is a set of components that interact to solve a problem.
  - **System structure** describes the system's objects and their interrelationships.
  - **System behavior** describes how the system changes as its objects interact with one another.