Chapter 3 – Introduction to Visual Basic Programming

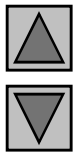
Outline

3.1 Introduction

- In this chapter we introduce
  - Visual Basic programming
    - We present examples that illustrate several important features of the language
  - Console applications
    - Applications that contain only text output
    - Output is displayed in a command window

3.2 Simple Program: Printing a Line of Text

- Simple program that displays a line of text
- When the program is run
  - output appears in a command window
- It illustrates important Visual Basic features
  - Comments
  - Modules
  - Sub procedures



Outline

Welcome1.vb

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

Visual Basic console applications consist of pieces called modules

indicates that the remainder of the line is a comment

The **Main** p
point of the
in all consol

The **Console.WriteLine** statement displays text output to the console

Program Output

```
Welcome to Visual Basic!
```

- A few Good Programming Practices
  - Comments
    - Every program should begin with one or more comments
  - Modules
    - Begin each module with **mod** to make modules easier to identify
  - Procedures
    - Indent the entire body of each procedure definition one “level” of indentation

3.2 Simple Program: Printing a Line of Text

- Now a short step-by-step explanation of how to create and run this program using the features of Visual Studio .NET IDE…

3.2 Simple Program: Printing a Line of Text

1. Create the console application
   - Select **File > New > Project...**
   - In the left pane, select **Visual Basic Projects**
   - In the right pane, select **Console Application**
   - Name the project **`Welcome1`**
   - Specify the desired location
2. Change the name of the program file
   - Click **`Module1.vb`** in the **Solution Explorer** window
   - In the **Properties** window, change the **File Name** property to **`Welcome1.vb`**

3.2 Simple Program: Printing a Line of Text

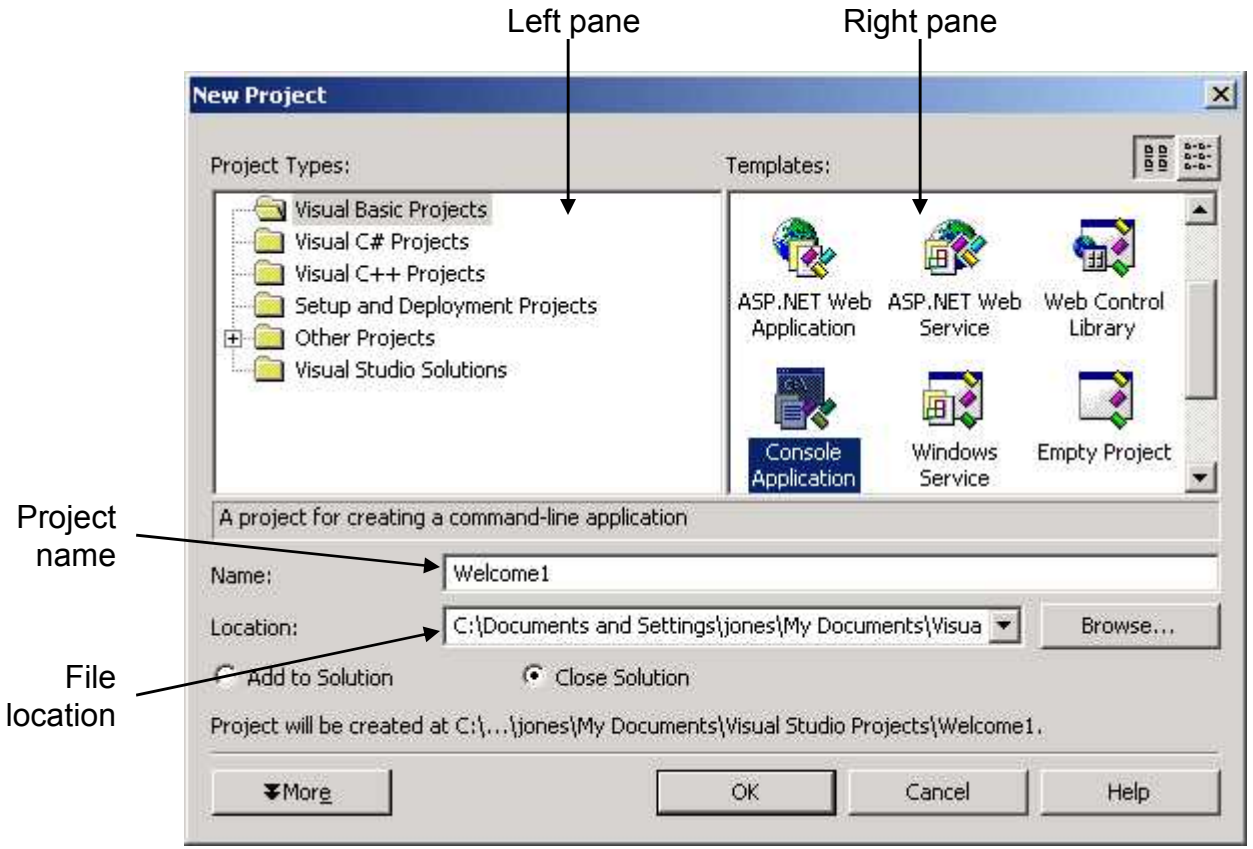


Fig. 3.2 Creating a Console Application with the New Project dialog.

3.2 Simple Program: Printing a Line of Text

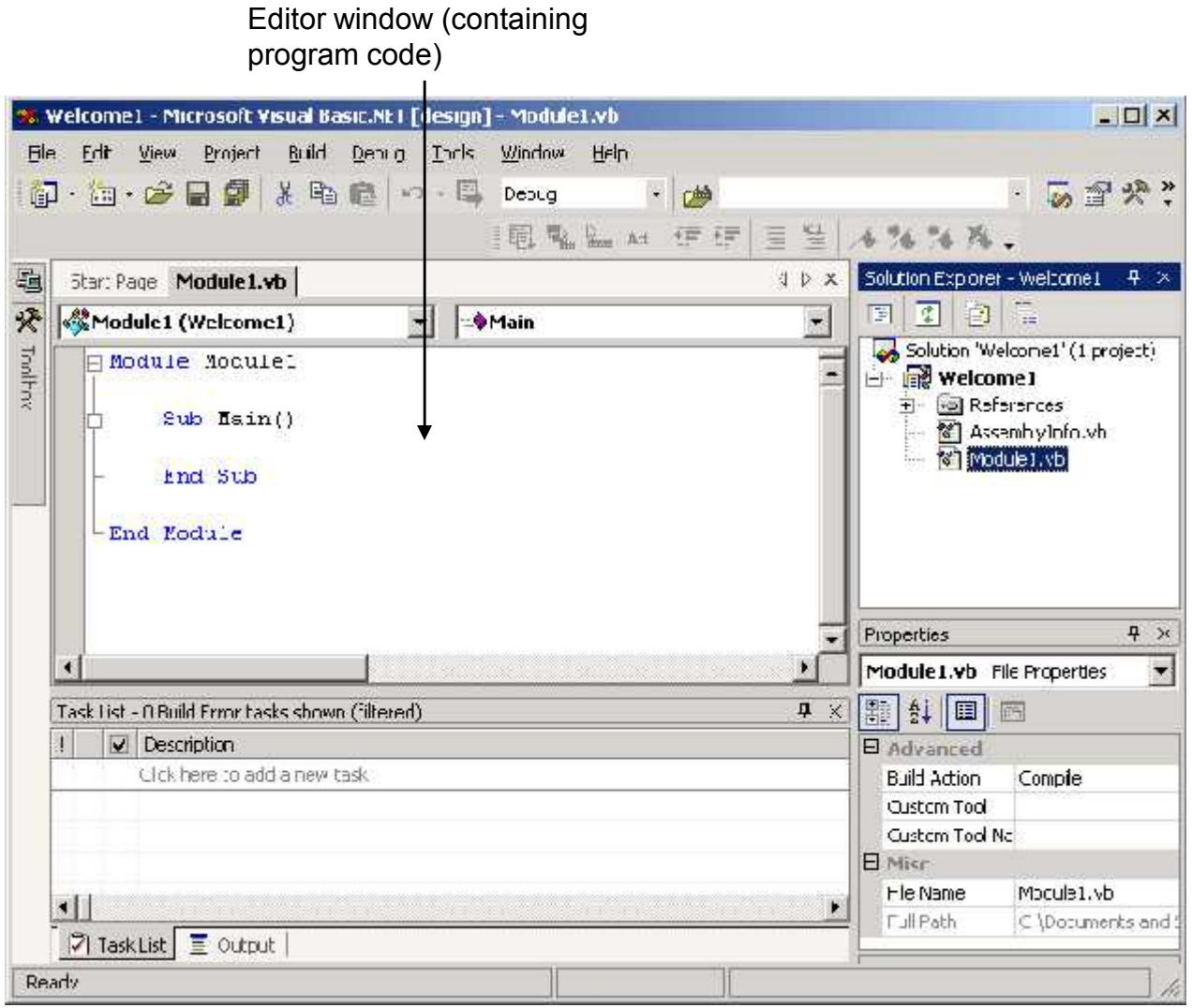


Fig. 3.3 IDE with an open console application.

3.2 Simple Program: Printing a Line of Text

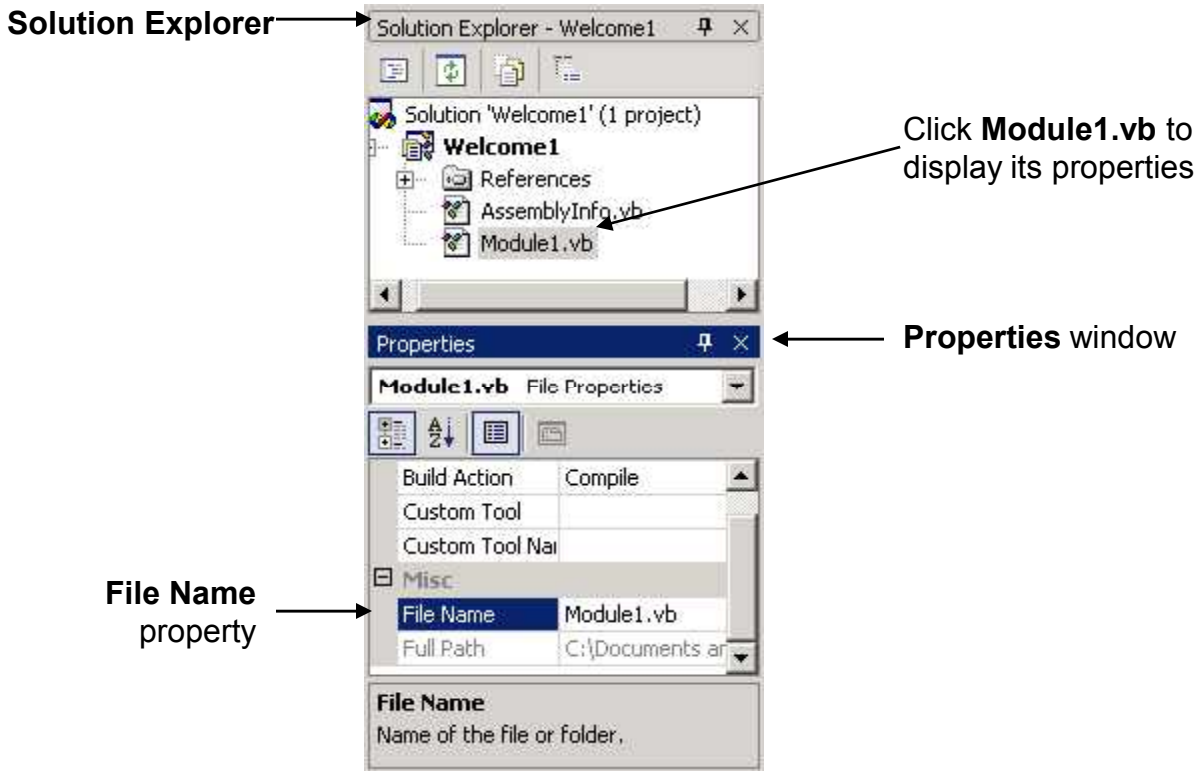


Fig. 3.4 Renaming the program file in the Properties window.

3.2 Simple Program: Printing a Line of Text

3. Change the name of the module
   - Module names must be modified in the editor window
   - Replace the identifier **`Module1`** with **`modFirstWelcome`**
4. Writing code
   - Type the code contained in line 7 of Fig. 3.1 between **`Sub Main()`** and **`End Sub`**
     - Note that after typing the class name and the dot operator the IntelliSense is displayed. It lists a class's members.
     - Note that when typing the text between the parenthesis (parameter), the Parameter Info and Parameter List windows are displayed

3.2 Simple Program: Printing a Line of Text

5. Run the program
   - To compile, select **Build > Build Solution**
     - This creates a new file, named **`Welcome1.exe`**
   - To run, select **Debug > Start Without Debugging**

3.2 Simple Program: Printing a Line of Text

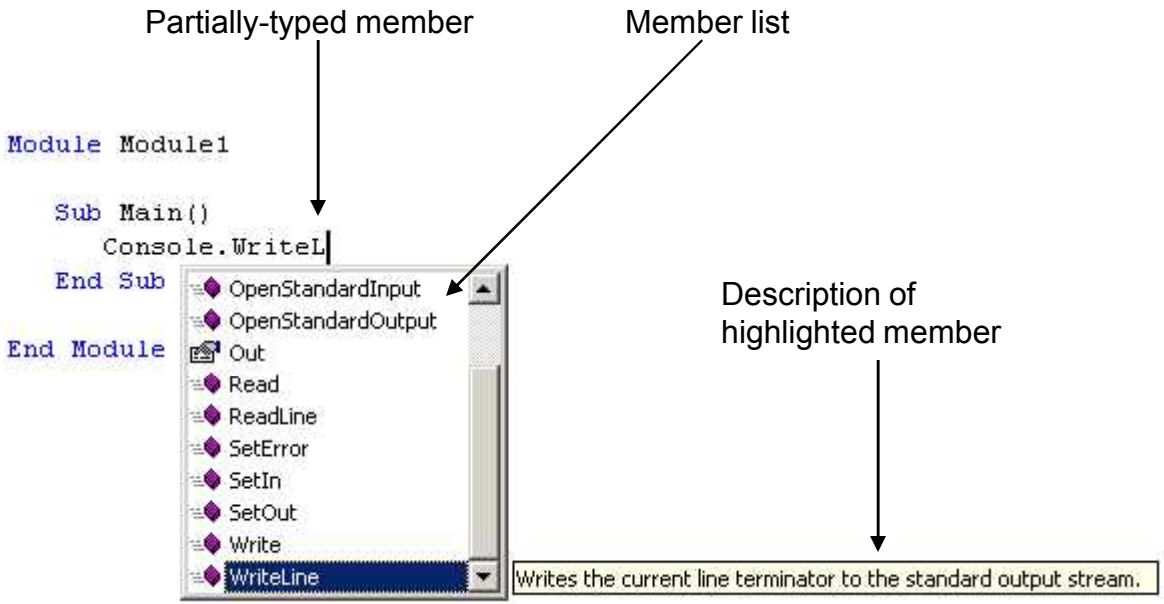


Fig. 3.5 IntelliSense feature of the Visual Studio .NET IDE.

3.2 Simple Program: Printing a Line of Text

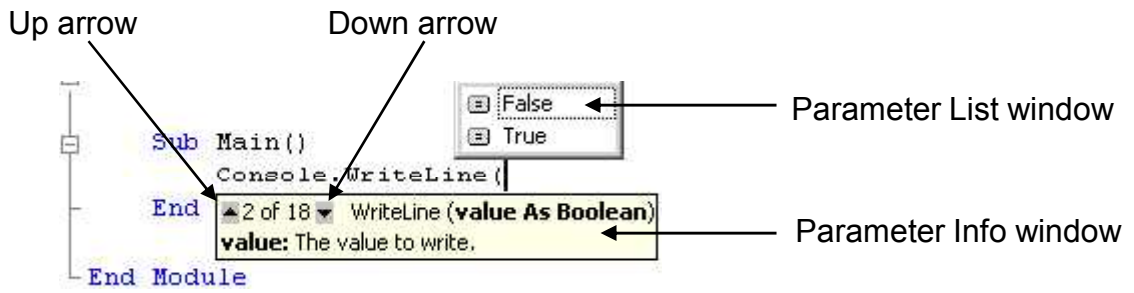


Fig. 3.6 Parameter Info and Parameter List windows.

3.2 Simple Program: Printing a Line of Text

Command window prompts the user to press a key after the program terminates

Fig. 3.7 Executing the program shown in Fig. 3.1.

3.2 Simple Program: Printing a Line of Text

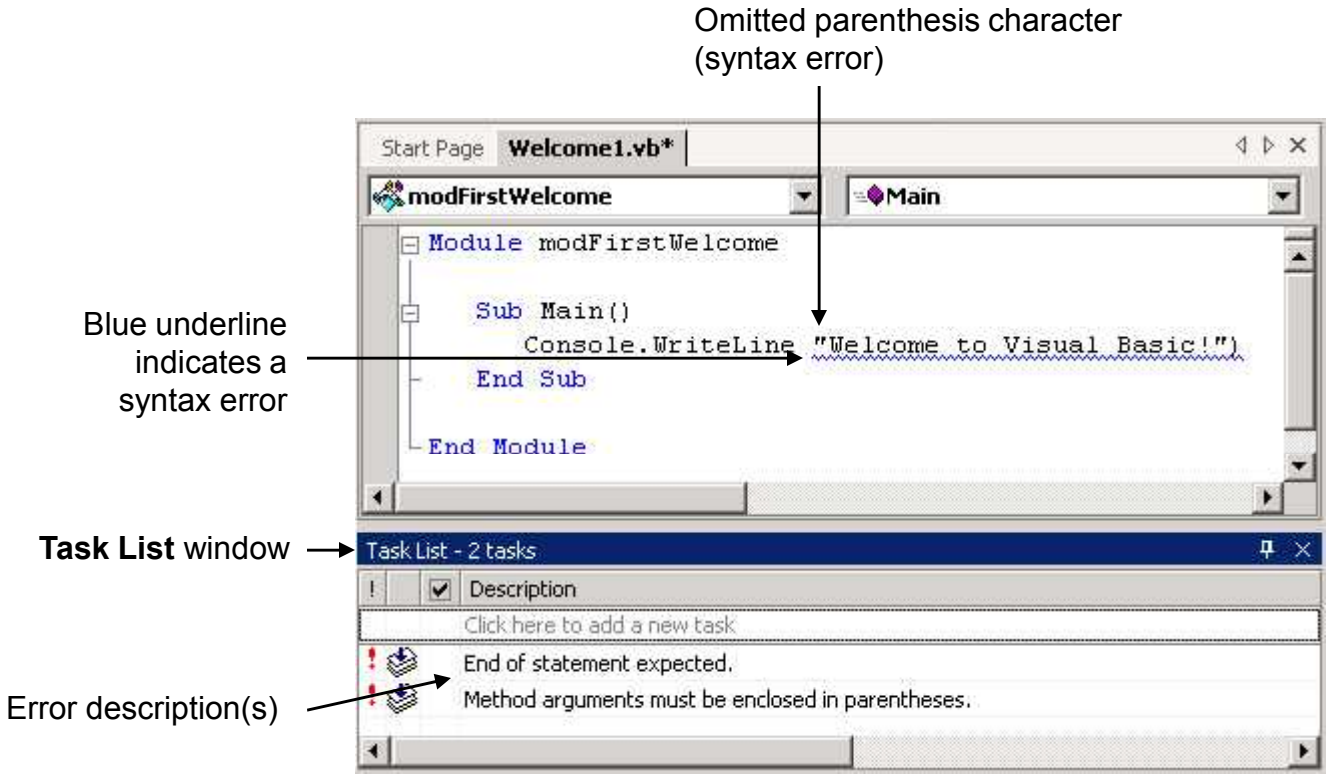
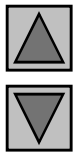


Fig. 3.8 IDE indicating a syntax error.



Outline

```
1   ' Fig. 3.9: Welcome2.vb
2   ' Writing line of text with
3
4   Module modSecondWelcome
5
6      Sub Main()
7         Console.Write("Welcome to ")
8         Console.WriteLine("Visual Basic!")
9      End Sub ' Main
11
12  End Module ' modSecondWelcome
```

Method **`Write`** does not position the output cursor at the beginning of the next line

Method **`WriteLine`** positions the output cursor at the beginning of the next line

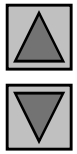
`Welcome2.vb`

```
Welcome to Visual Basic!
```

Program Output

3.3 Another Simple Program: Adding Integers

- User input two integers
  - Whole numbers
- Program computes the sum
- Display result



Outline

Addition.vb

```
1  ' Fig. 3.10: Addition.vb
2  ' Addition program.
3
4  Module modAddition
5
6    Sub Main()
7
8      ' variables for storing user input
9      Dim firstNumber, secondNumber As String
10
11     ' variables used in addition calculation
12     Dim number1, number2, sumOfNumbers As Intege
13
14     ' read first number from user
15     Console.Write("Please enter the first integer: ")
16     firstNumber = Console.ReadLi
17
18     ' read second number from user
19     Console.Write("Please enter the second integer: ")
20        secondNumber = Console.ReadLine()
21
22     ' convert input values to Integers
23     number1 = firstNumber
24     number2 = secondNumber
25
26     sumOfNumbers = number1 + number2 ' add numbers
27
28     ' display results
29     Console.WriteLine("The sum is {0}", sumOfNum
30
31     End Sub ' Main
32
33   End Module ' modAddition
```

These variables store strings of characters

Declarations begin with keyword **Dim**

First value entered by user is assigned to variable **firstNumber**

These variables sto

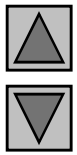
Method **ReadLine** causes program to pause and wait for user input

Implicit conversion from **String** to **Integer**

Sums integers and assigns result to variable **sumOfNumbers**

Format indicates that the argument after the string will be evaluated and incorporated into the string

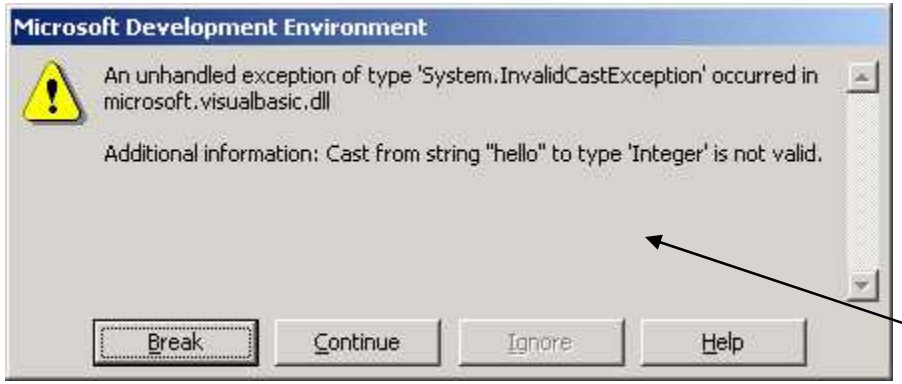
```
Please enter the first integer: 45
Please enter the second integer: 72
The sum is 117
```



Outline

Addition.vb

3.3 Another Simple Program: Adding Integers



If the user types a non-integer value, such as "**`hello`**," a run-time error occurs

Fig. 3.11 Dialog displaying a run-time error.

3.4 Memory Concepts

- Variables
  - correspond to actual locations in the computer's memory
  - Every variable has a
    - Name
    - Type
    - Size
    - value
  - A value placed in a memory location replaces the value previously stored
    - The previous value is destroyed
  - When value is read from a memory location, it is not destroyed

3.4 Memory Concepts

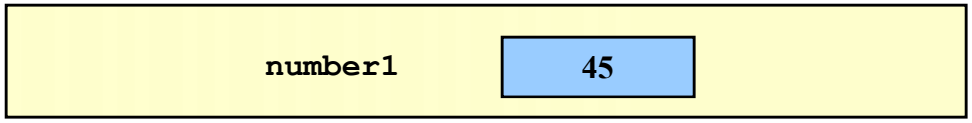


Fig. 3.12 Memory location showing name and value of variable **number1**.

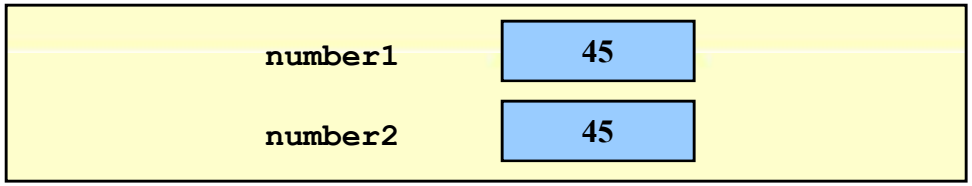


Fig. 3.13 Memory locations after values for variables **number1** and **number2** have been input.

3.5 Arithmetic

- Arithmetic operators
  - Visual Basic use various special symbols not used in algebra
    - Asterisk (**`*`**), keyword **`Mod`**
  - Binary operators
    - Operates using two operands
      - **`sum + value`**
  - Unary operators
    - Operators that take only one operand
      - **`+9`**, **`-19`**

3.5 Arithmetic

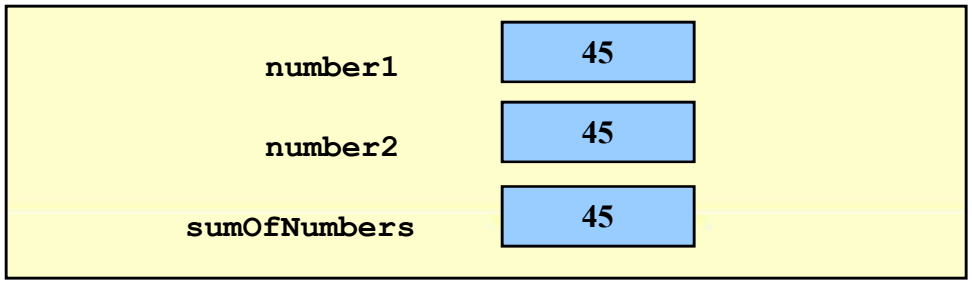


Fig. 3.14 Memory locations after an addition operation.

3.5 Arithmetic

- Integer division
  - Uses the backslash, ****
  - **7 \ 4** evaluates to **1**
- Floating-point division
  - Uses the forward slash, **/**
  - **7 / 4** evaluates to **1.75**
- Modulus operator, **Mod**
  - Yields the remainder after **Integer** division
  - **7 Mod 4** yields **3**

3.5 Arithmetic

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 (float)	/	x / y or <Anchor10> or $\frac{x}{y}$	`x / y`
Division (Integer)	\	none	`v \ u`
Modulus	%	$r \bmod s$	`r Mod s`
Exponentiation	^	q^p	`q ^ p`
Unary Negative	-	–e	`-e`
Unary Positive	+	+g	+g

Fig. 3.14 Arithmetic operators.

Fig. 3.14 Arithmetic Operators.

3.5 Arithmetic

- Rules of operator precedence
  1. Operators in expressions contained within parentheses
  2. Exponentiation
  3. Unary positive and negative
  4. Multiplication and floating-point division
  5. **`Integer`** division
  6. Modulus operations
  7. Addition and subtraction operations

3.5 Arithmetic

Operator(s)	Operation	Order of evaluation (precedence)
( )	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. If there are several pairs of parentheses "on the same level" (i.e., not nested), they are evaluated from left to right.
^	Exponentiation	Evaluated second. If there are several such operators, they are evaluated from left to right.
+, –	Sign operations	Evaluated third. If there are several such operators, they are evaluated from left to right.
*, /	Multiplication and Division	Evaluated fourth. If there are several such operators, they are evaluated from left to right.
\	Integer division	Evaluated fifth. If there are several such operators, they are evaluated from left to right.
Mod	Modulus	Evaluated sixth. If there are several such operators, they are evaluated from left to right.
+, –	Addition and Subtraction	Evaluated last. If there are several such operators, they are evaluated from left to right.
Fig. 3.15 Precedence of arithmetic operators.		

Fig. 3.15 Precedence of arithmetic operators.

3.5 Arithmetic

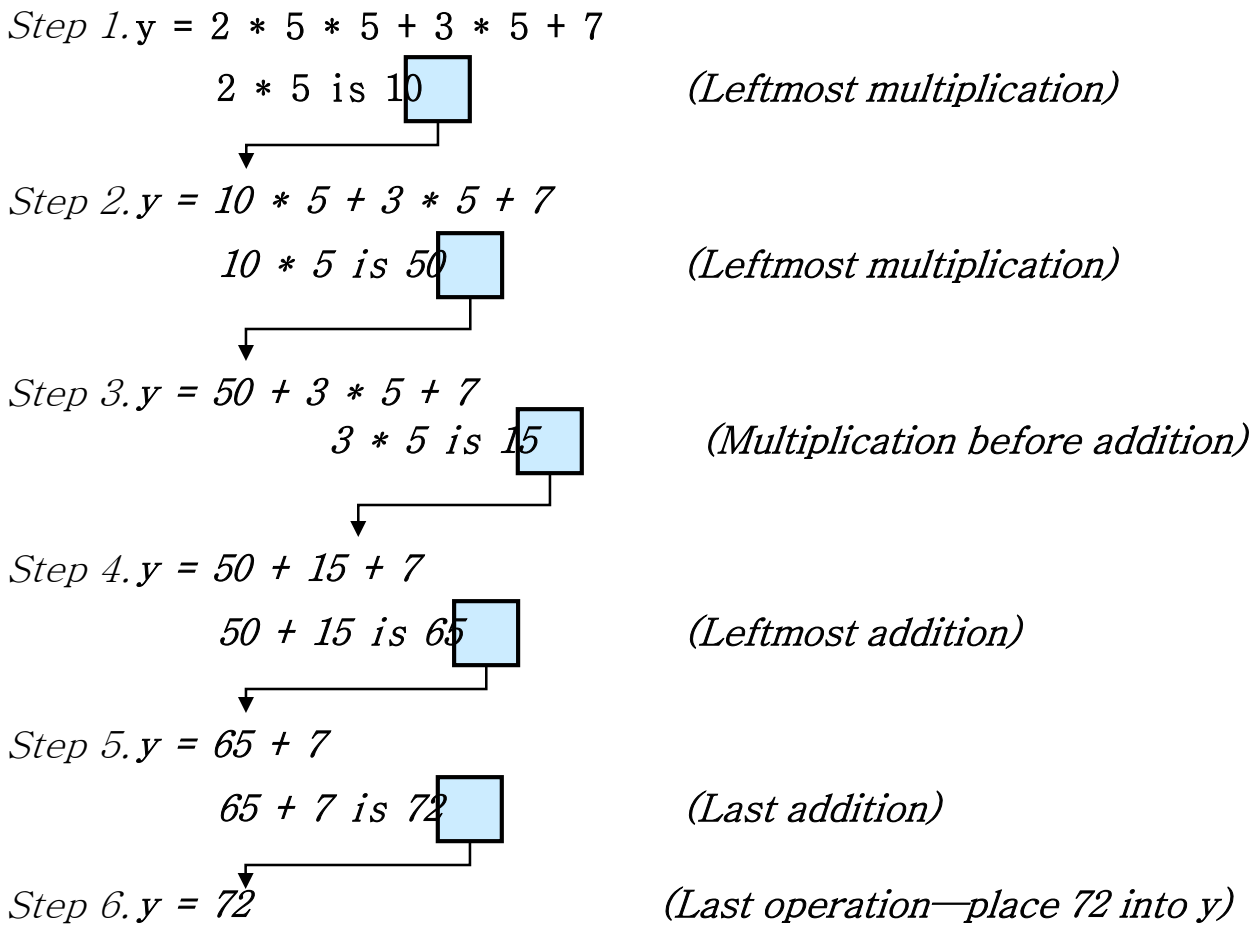


Fig. 3.16 Order in which a second-degree polynomial is evaluated.

3.6 Decision Making: Equality and Relational Operators

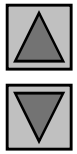
- **`If`/`Then`** structure
  - Allows a program to make decision based on the truth or falsity of some expression
  - Condition
    - The expression in an **`If`/`Then`** structure
  - If the condition is true, the statement in the body of the structure executes
  - Conditions can be formed by using
    - Equality operators
    - Relational operators

3.6 Decision Making: Equality and Relational Operators

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. 3.17 Equality and relational operators.

Fig. 3.17 Equality and relational operators.



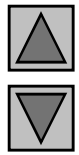
Outline

Comparison.vb

```
1  ' Fig. 3.19: Comparison.vb
2  ' Using equality and relational operators.
3
4  Module modComparison
5
6     Sub Main()
7
8        ' declare Integer variables for user input
9        Dim number1, number2 As Integer
10
11       ' read first number from u
12       Console.Write("Please ente
13       number1 = Console.ReadLine()
14
15       ' read second number from user
16       Console.Write("Please enter second integer: ")
17       number2 = Console.ReadLine()
18
19       If (number1 = number2) Then
20          Console.WriteLine("{0} = {1}", number1, number2)
21       End If
22
23       If (number1 <> number2) Then
24          Console.WriteLine("{0} <> {1}", number1, number2)
25       End If
26
27       If (number1 < number2) Then
28          Console.WriteLine("{0} < {1}", number1, number2)
29       End If
30
31       If (number1 > number2) Then
32          Console.WriteLine("{0} > {1}", number1, number2)
33       End If
```

Variables of the same type may be declared in one declaration

The If/Then structure compares the values of number1 and number2 for equality



Outline

```
34
35       If (number1 <= number2) Then
36          Console.WriteLine("{0} <= {1}", number1, number2)
37       End If
38
39       If (number1 >= number2) Then
40          Console.WriteLine("{0} >= {1}", number1, number2)
41       End If
42
43    End Sub ' Main
44
45 End Module ' modComparison
```

Comparison.vb

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

```
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
```

Program Output

3.6 Decision Making: Equality and Relational Operators

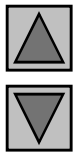
Operators	Associativity	Type
( )	left to right	parentheses
^	left to right	exponentiation
* /	left to right	multiplicative
\	left to right	integer division
Mod	left to right	modulus
+ -	left to right	additive
= <> < <= > >=	left to right	equality and relational

Fig. 3.19 Precedence and associativity of operators introduced in this chapter.

Fig. 3.19 Precedence and associativity of operators introduced in this chapter.

3.7 Using a Dialog to Display a Message

- Dialogs
  - Windows that typically display messages to the user
  - Visual Basic provides class **`MessageBox`** for creating dialogs



Outline

SquareRoot.vb

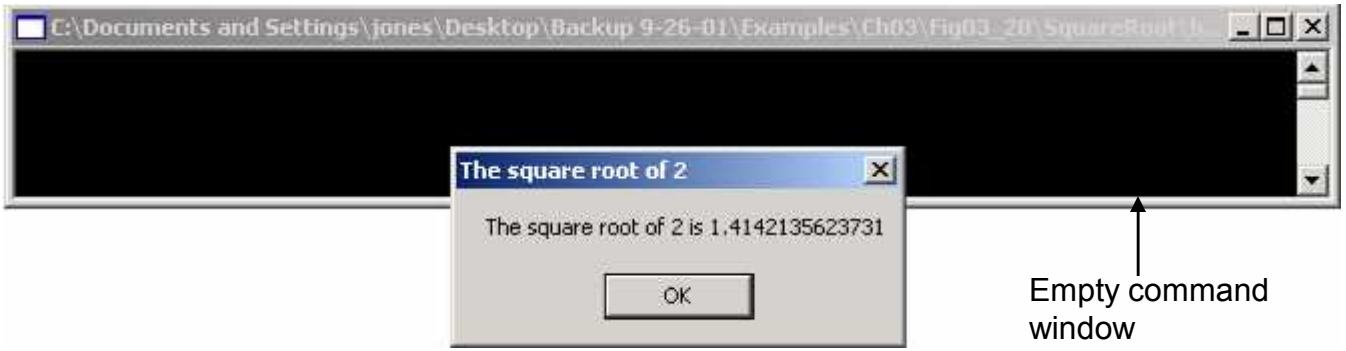
```
1 ' Fig. 3.20: SquareRoot.vb
2 ' Displaying square root of 2 in dialog.
3
4 Imports System.Windows.Forms ' Namespace containing MessageBox
5
6 Module modSquareRoot
7
8    Sub Main()
9
10       ' Calculate square root of 2
11       Dim root As Double = Math.Sqrt(2)
12
13       ' Display results in dialog
14       MessageBox.Show("The squa
15          "The Square Root of 2"
16    End Sub ' Main
17
18 End Module ' modThirdWelcome
```

Sqrt method of the **Math** class is called to compute the square root of 2

Method **Show** of class **MessageBox**

The **Double** data type stores floating-point numbers

Line-continuation character



Program Output

3.7 Using a Dialog to Display a Message

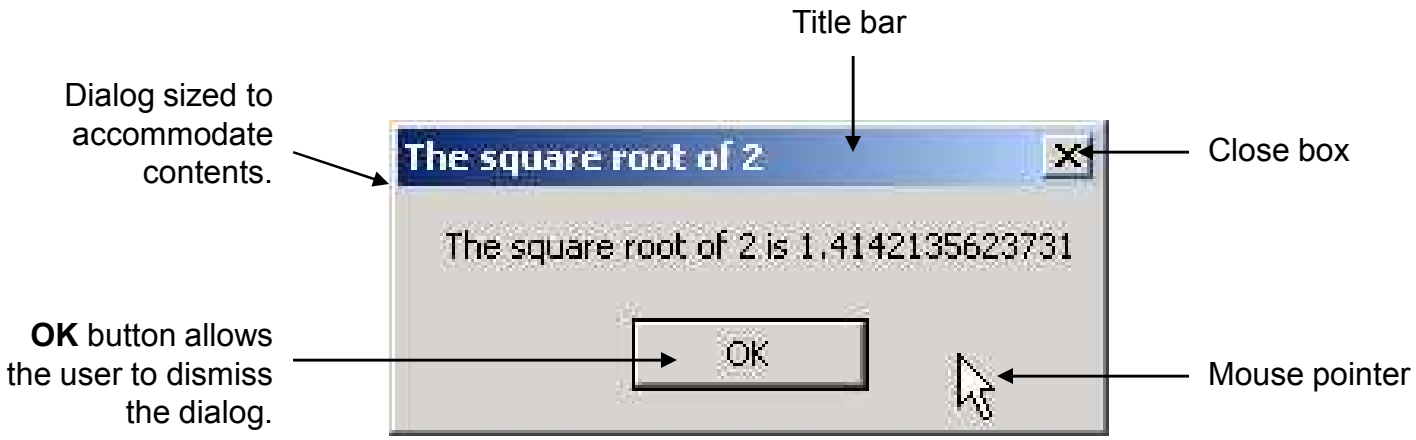


Fig. 3.21 Dialog displayed by calling MessageBox.Show.

3.7 Using a Dialog to Display a Message

- Assembly
  - File that contain many classes provided by Visual Basic
  - These files have a **`.dll`** (or dynamic link library) extension.
  - Example
    - Class **`MessageBox`** is located in assembly **`System.Windows.Forms.dll`**
- MSDN Documentation
  - Information about the assembly that we need can be found in the MSDN documentation
  - Select **Help > Index**… to display the **Index** dialog

3.7 Using a Dialog to Display a Message

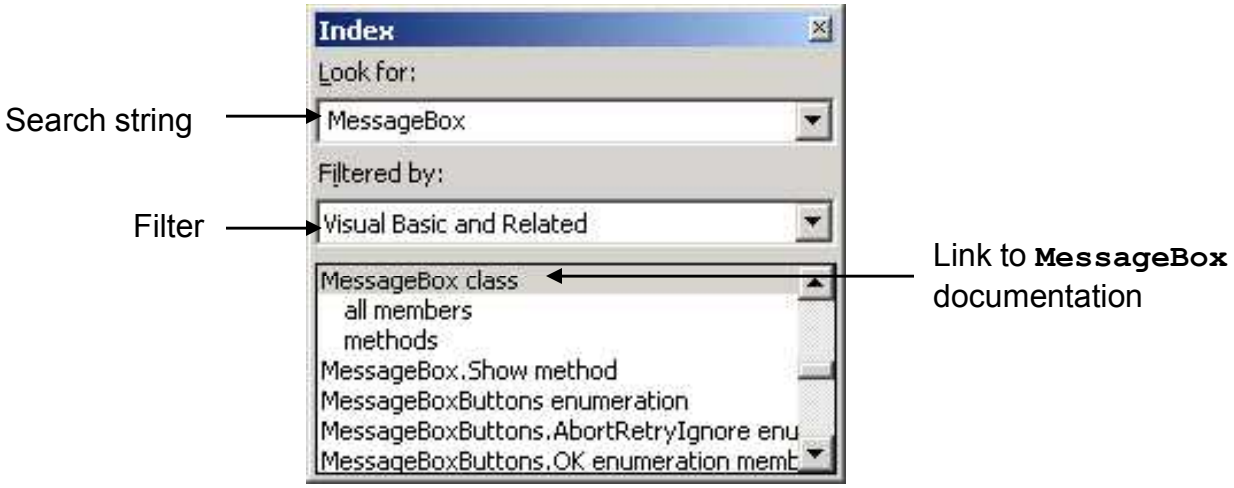


Fig. 3.22 Obtaining documentation for a class by using the Index dialog.

3.7 Using a Dialog to Display a Message

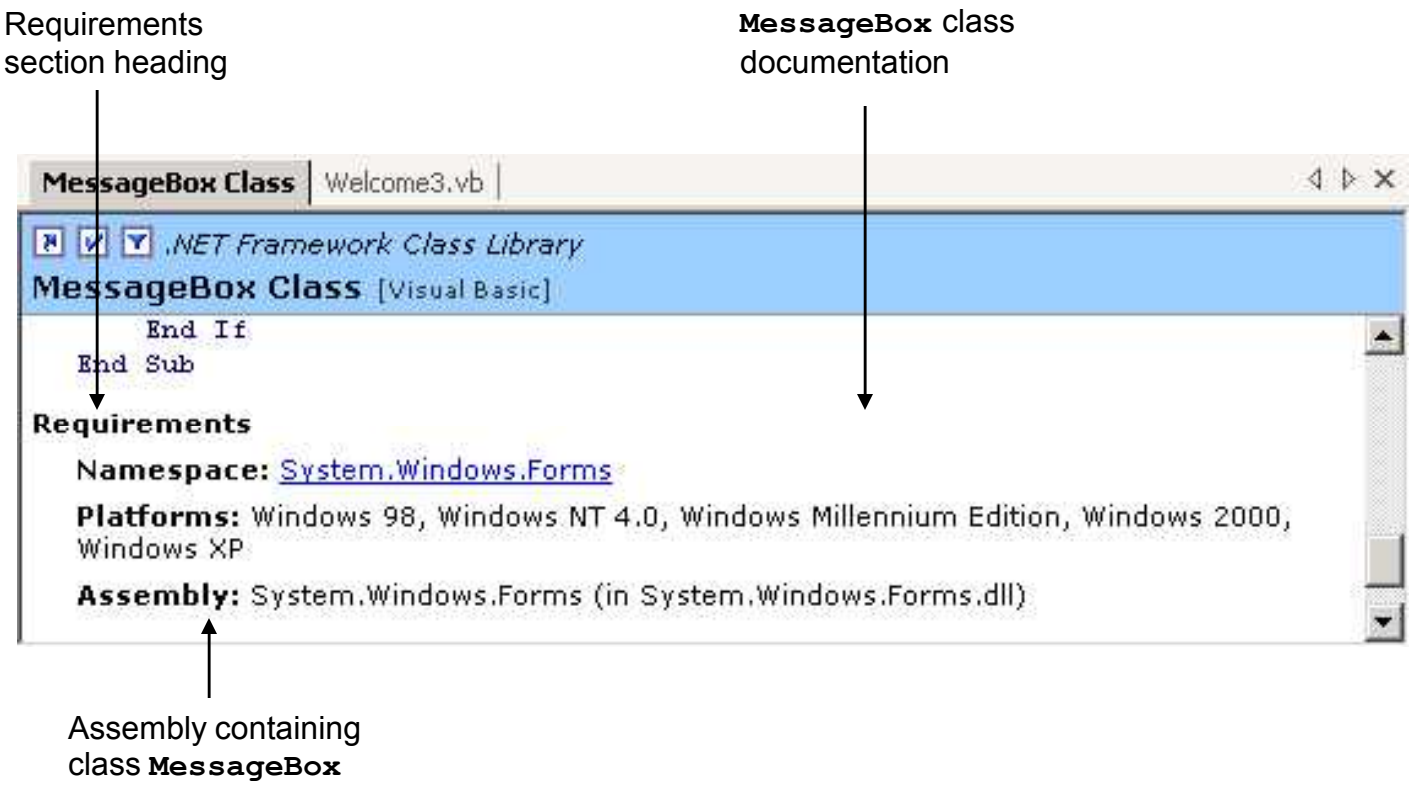


Fig. 3.23 Documentation for the MessageBox class.

3.7 Using a Dialog to Display a Message

- Reference
  - It is necessary to add a reference to the assembly if you wish to use its classes
  - Example
    - To use class **`MessageBox`** it is necessary to add a reference to **`System.Windows.Forms`**
- **`Imports`**
  - Forgetting to add an **`Imports`** statement for a referenced assembly is a syntax error

3.7 Using a Dialog to Display a Message

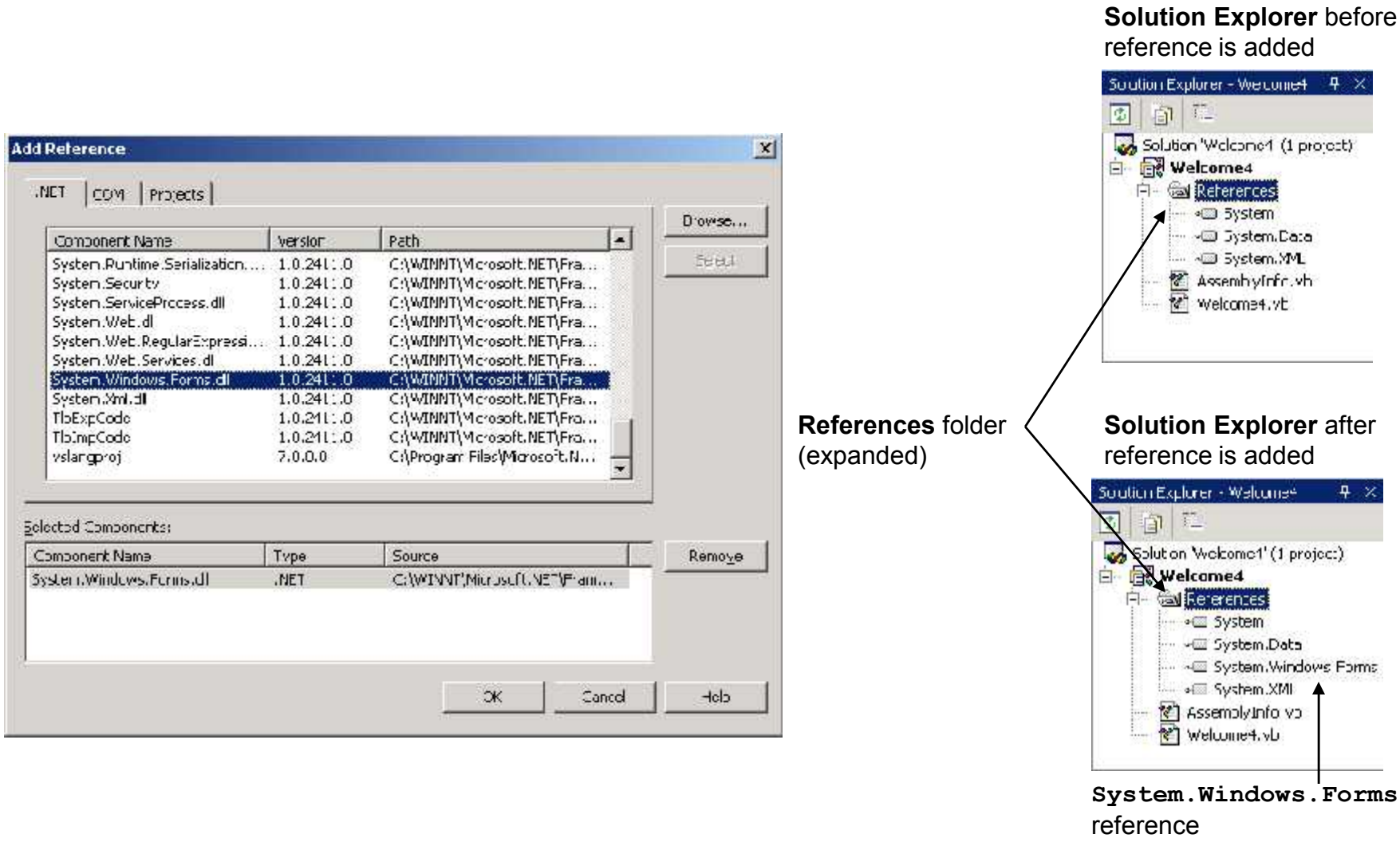


Fig. 3.24 Adding a reference to an assembly in the Visual Studio .NET IDE.

3.7 Using a Dialog to Display a Message

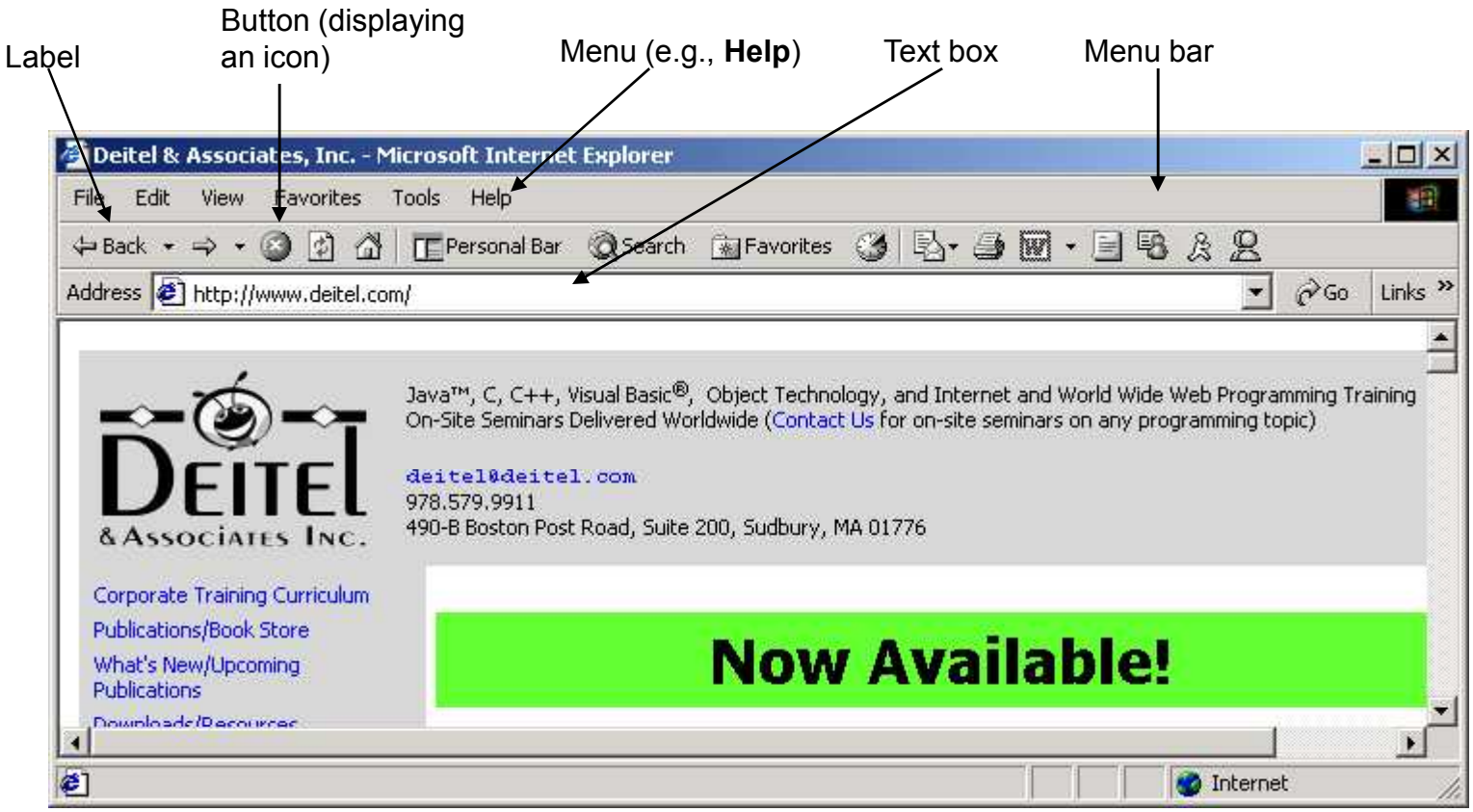


Fig. 3.25 Internet Explorer window with GUI components.