INTERNATIONAL EDITION

Simply Visual Basic® 2010

An App-Driven Approach

FOURTH EDITION

Paul Deitel • Harvey Deitel • Abbey Deitel

ALWAYS LEARNING

PEARSON

Simply Visual Basic® 2010

Fourth Edition

10.13 ***(Arithmetic Calculator App)*** Write an app that allows the user to enter a series of numbers and manipulate them. The app should provide users with the option of adding or multiplying the numbers. Users should enter each number in a `TextBox`. After entering the number, the user clicks a `Button`, and the number is inserted in a `ListBox`. The GUI should behave as in Fig. 10.20.

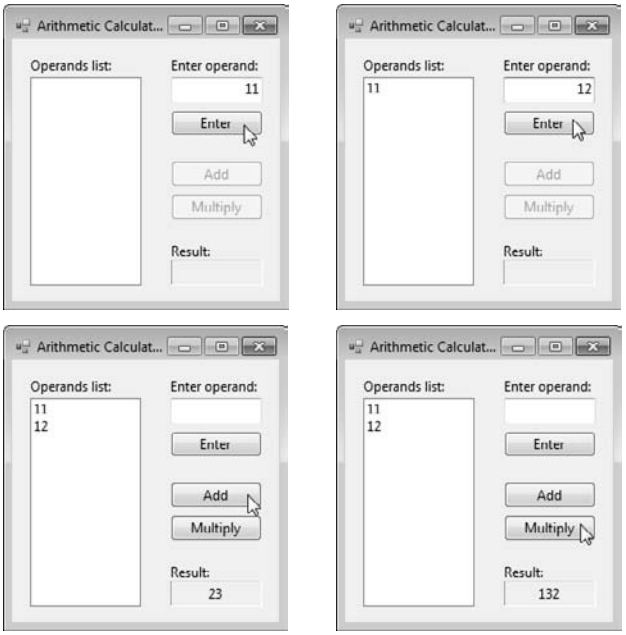

Figure 10.20 **Arithmetic Calculator** app.

a) ***Copying the template to your working directory.*** Copy the directory `C:\Examples\ch10\Exercises\ArithmeticCalculator` to your `C:\SimplyVB2010` directory.

b) ***Opening the app's template file.*** Double click `ArithmeticCalculator.sln` in the `ArithmeticCalculator` directory to open the app.

c) ***Adding a `ListBox` to display the entered numbers.*** Add a `ListBox`. Place and size it as in Fig. 10.20.

d) ***Creating an event handler for the Enter `Button`.*** Create the `Click` event handler for the **Enter** `Button`. If the result of a previous calculation is displayed, this event handler should clear the result, clear the `ListBox` and disable the addition and multiplication `Buttons`. It should then insert the current number in the **Operands list:** `ListBox`. When the `ListBox` contains at least two numbers, the event handler should then enable the addition and multiplication `Buttons`.

e) ***Summing the values in the `ListBox`.*** Define the `Click` event handler for the **Add** `Button`. This event handler should compute the sum of all the values in the **Operands list:** `ListBox` and display the result in `resultLabel`.

f) ***Multiplying the values in the `ListBox`.*** Define the `Click` event handler for the **Multiply** `Button`. This event handler should compute the product of all the values in the **Operands list:** `ListBox` and display the result in `resultLabel`.

g) ***Running the app.*** Select **Debug > Start Debugging** to run your app. Enter two values, then click the **Add** and **Multiply** `Buttons`. Verify that the results displayed are correct. Also, make sure that the **Add** and **Multiply** `Buttons` are not enabled until two values have been entered. Enter a new value and verify that the previous result and the `ListBox` are cleared. Enter two more values, then click the **Add** and **Multiply** `Buttons`. Verify that the results displayed are correct.

h) ***Closing the app.*** Close your running app by clicking its close box.

i) ***Closing the IDE.*** Close the Visual Basic IDE by clicking its close box.

What does this code do?

10.14 What is the result of the following code?

```
Dim y As Integer
Dim x As Integer
Dim mysteryValue As Integer

x = 1
mysteryValue = 0

Do
   y = x ^ 2
   displayListBox.Items.Add(y)
   mysteryValue += 1
   x += 1
Loop While x <= 10

resultLabel.Text = mysteryValue
```

What's wrong with this code?

10.15 Find the error(s) in the following code. This code should add 10 to the value in `y` and store it in `z`. It then should reduce the value of `y` by one and repeat until `y` is less than 10. Last, `resultLabel` should display the final value of `z`.

```
Dim y As Integer = 10
Dim z As Integer = 2

Do
   z = y + 10
Loop Until y < 10

y -= 1

resultLabel.Text = z
```

Using the Debugger

10.16 ***(Factorial App)*** The **Factorial** app calculates the factorial of an integer entered by the user. The factorial of an integer is the product of the integers from 1 to that number. For example, the factorial of 3 is 6 ($1 \times 2 \times 3$). Copy the **Factorial** app from `C:\Examples\ch10\Exercises\Factorial` to your working directory. While testing the app, you noticed that it did not execute correctly. Use the debugger to find and correct the logic error(s) in the app. Figure 10.21 displays the correct output for the **Factorial** app.

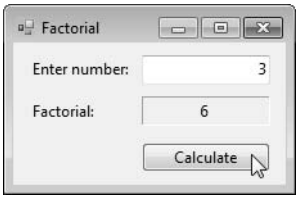

Figure 10.21 Correct output for the **Factorial** app.

Programming Challenge

10.17 ***(Restaurant Bill App)*** Develop an app that calculates a restaurant bill. The user should be able to enter the item ordered, the quantity of the item ordered and the price per item. When the user clicks the **Add Item** `Button`, your app should display the number ordered, the item ordered and the price per unit in three `ListBoxes`, as shown in Fig. 10.22. When the user clicks the **Total Bill** `Button`, the app should calculate the total cost. For each entry in the `ListBox`, multiply the cost of each item by the number of items ordered.

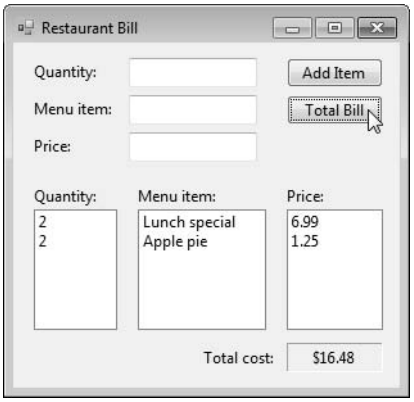

Figure 10.22 **Restaurant Bill** app's `Form`.

11

Interest Calculator App

Introducing the `For...Next` Repetition Statement and `NumericUpDown` Control

Objectives

In this chapter, you'll:

- Execute statements repeatedly with the `For`...`Next` repetition statement.
- Use local type inference to infer a variable's type from its initializer value.
- Obtain user input with the `NumericUpDown` control.
- Display information, using a multiline `TextBox`.
- Use type `String`.

Outline

As you learned in Chapters 9 and 10, apps are often required to repeat actions. Using a `Do` repetition statement allowed you to specify a condition and test it either before entering the loop or after executing the body of the loop. In the **Car Payment Calculator** app and the **Class Average** app, a counter was used to determine the number of times the loop should iterate. In fact, the use of counters in repetition statements is so common in apps that Visual Basic provides an additional control statement specially designed for such cases—the `For`...`Next` repetition statement. In this chapter, you'll use this repetition statement to create an **Interest Calculator** app.

11.1 Test-Driving the Interest Calculator App

The **Interest Calculator** app calculates the amount of money in your savings account. You'll begin with a certain amount of money and will be paid interest for a period of time. The user specifies the principal amount (the initial amount of money in the account), the interest rate and the number of years for which interest will be calculated. The app then displays the results. This app must meet the following requirements:

App Requirements

You're considering investing $1,000.00 in a savings account that yields 2% interest compounded annually, and you want to forecast how your investment will grow. Assuming that you leave all interest on deposit, calculate and display the amount of money in the account at the end of each year over a period of n *years. To compute these amounts, use the following formula:*

$a = p\,(1 + r)^n$

where

p is the original amount of money invested (the principal)
r is the annual interest rate (for example, .02 is equivalent to 2%)
n is the number of years
a is the amount on deposit at the end of the *n*th year.

You'll begin by test-driving the completed app. Then you'll learn the additional Visual Basic capabilities needed to create your own version of this app.

Test-Driving the Interest Calculator App

1. ***Opening the completed app.*** Open `C:\Examples\ch11\CompletedApp\InterestCalculator` to locate the **Interest Calculator** app. Double click `InterestCalculator.sln` to open the app in the IDE.
2. ***Running the Interest Calculator app.*** Select **Debug > Start Debugging** to run the app (Fig. 11.1).

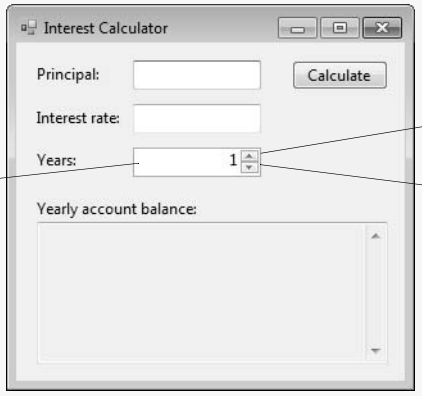

Figure 11.1 Completed **Interest Calculator** app.

3. ***Providing a principal value.*** Once the app is running, provide a value in the **Principal:** `TextBox`. Input 1000, as specified in the problem statement.
4. ***Providing an interest-rate value.*** Next, type a value in the **Interest Rate:** `TextBox`. We specified the interest rate 2% in the problem statement, so enter 2 in the **Interest Rate:** `TextBox`.
5. ***Providing the duration of the investment.*** Now, choose the number of years for which you want to calculate the amount in the savings account. In this case, select 10 by entering it using the keyboard or by clicking the up arrow in the **Years:** `NumericUpDown` control repeatedly until the value reads 10.
6. ***Calculating the amount.*** After you input the necessary information, click the **Calculate** `Button`. The amount of money in your account at the end of each year during a period of 10 years displays in the multiline `TextBox`. The app should look similar to Fig. 11.2.

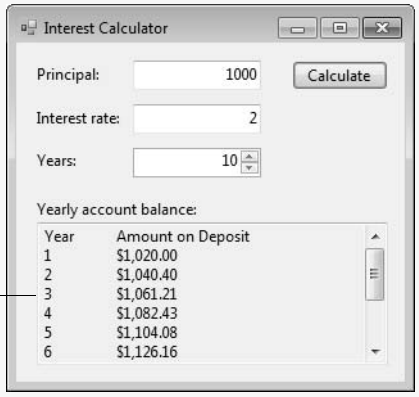

Figure 11.2 Output of completed **Interest Calculator** app.

7. ***Closing the app.*** Close your running app by clicking its close box.
8. ***Closing the IDE.*** Close the Visual Basic IDE by clicking its close box.

11.2 Essentials of Counter-Controlled Repetition

In Chapters 9 and 10, you learned how to use counter-controlled repetition. Its four essential elements are:

1. the *name* of a *control variable* (or loop counter) that's used to determine whether the loop continues to iterate
2. the *initial value* of the control variable
3. the *increment* (or *decrement*) by which the control variable is modified during each iteration of the loop (that is, each time the loop is performed)
4. the *condition* that tests for the *final value* of the control variable (to determine whether looping should continue).

The example in Fig. 11.3 uses the four elements of counter-controlled repetition. This `Do While…Loop` statement is similar to the **Car Payment Calculator** app's loop in Chapter 9.

```
1  Dim years As Integer = 2 ' control variable
2
3  Do While years <= 5
4     months = 12 * years ' calculate payment period
5
6     ' calculate payment value
7     monthlyPayment =
8        Pmt(monthlyInterest, months, -loanAmount)
9
10    ' display payment value
11    paymentsListBox.Items.Add(months & ControlChars.Tab &
12       ControlChars.Tab & String.Format("{0:C}", monthlyPayment))
13
14    years += 1 ' increment counter
15 Loop
```

Figure 11.3 Counter-controlled repetition example.

Recall that the **Car Payment Calculator** app calculates and displays monthly car payments over periods of two to five years. The declaration in line 1 *names* the control variable (`years`) and indicates that it's of data type `Integer`. This declaration initializes the variable to an *initial value* of `2`.

Consider the `Do While…Loop` statement (lines 3–15). Line 4 uses the `years` variable to calculate the number of months over which car payments are to be made. Lines 7–8 use the `Pmt` function to determine the monthly payment for the car. This value depends on the monthly interest rate, the duration of the loan in months and the loan amount. Lines 11–12 display the amount in a `ListBox`. Line 14 increments the control variable `years` by `1` for each iteration of the loop. The condition in the `Do While…Loop` statement (line 3) tests whether the value of the control variable is less than or equal to `5`, meaning that `5` is the *final value* for which the condition is true. The body of this `Do While…Loop` is performed even when the control variable is `5`. The loop terminates when the control variable exceeds `5` (that is, when `years` has a value of `6`).

SELF-REVIEW

1. Counter-controlled repetition _____ the control variable after each iteration.
   a) increments b) initializes
   c) decrements d) Either a or c

2. What aspect of the control variable determines whether looping should continue?
   a) name b) initial value
   c) type d) final value

Answers: 1) d. 2) d.