The last assignment statement in the btnCalc_Click procedure displays the new price (formatted with a dollar sign and two decimal places) in the interface. Finally, the computer processes the procedure's End Sub clause. When the procedure ends, the computer removes the `dblPrice` variable from its internal memory.

To test the Price Calculator application:

1. Save the solution and then start the application. Type **8.99** in the Current price box and then click the **Calculate** button. $9.44 appears in the New price box, as shown in Figure 17-11. (Recall that you can use the Alt key to show/hide the access keys.)

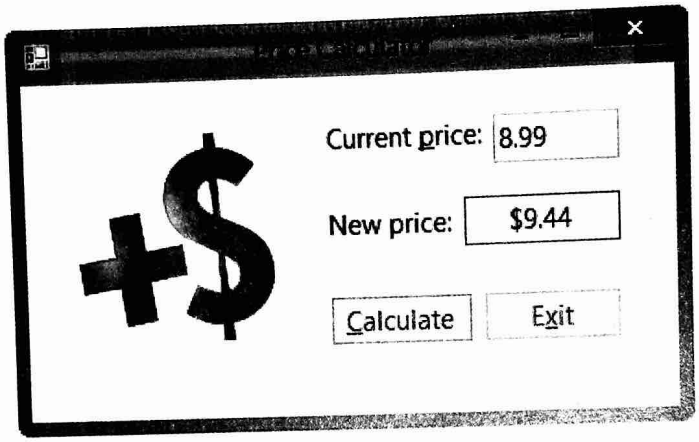

Figure 17-11 New price shown in the interface
OpenClipArt.org/baroquon/Jos Buivenga

2. Click the **Exit** button. Close the Code Editor window and then close the solution.

Revisiting the Concert Tickets Application

Figure 17-12 shows the interface for the Concert Tickets application from Chapter 16. As you may remember, you coded the application using an independent Sub procedure to assign the discount amount. Figure 17-13 shows the code entered in the AssignDiscount and btnCalc_Click procedures in Chapter 16. The Call statement in the event procedure passes three items of information to the AssignDiscount procedure: two *by value* and one *by reference*. The `intTickets` and `decSubtotal` variables are passed *by value* because the AssignDiscount procedure needs to know, but not change, the information contained in those variables. The `decDiscount` variable, on the other hand, is passed *by reference* because the AssignDiscount procedure needs to set the variable's value.

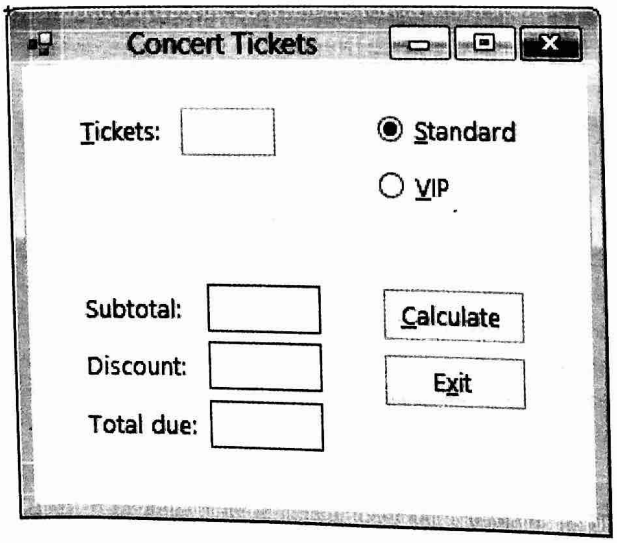

Figure 17-12 Interface for the Concert Tickets application

```
Private Sub AssignDiscount(ByVal intNum As Integer,
                           ByVal decSub As Decimal,
                           ByRef decDisc As Decimal)
    Select Case intNum
        Case Is >= 6
            decDisc = decSub * 0.1
        Case Is >= 4
            decDisc = decSub * 0.02
        Case Else
            decDisc = 0
    End Select
End Sub

Private Sub btnCalc_Click(sender As Object,
e As EventArgs) Handles btnCalc.Click
    ' displays the subtotal, discount, and total due

    Const decSTANDARD As Decimal = 32.75
    Const decVIP As Decimal = 75
    Dim intTickets As Integer
    Dim decSubtotal As Decimal
    Dim decTotalDue As Decimal
    Dim decDiscount As Decimal

    Integer.TryParse(txtTickets.Text, intTickets)

    ' calculate subtotal
    If radStandard.Checked = True Then
        decSubtotal = intTickets * decSTANDARD
    Else
        decSubtotal = intTickets * decVIP
    End If

    ' assign discount
    Call AssignDiscount(intTickets, decSubtotal, decDiscount)

    ' calculate total due
    decTotalDue = decSubtotal - decDiscount

    ' display subtotal, discount, and total due
    lblSubtotal.Text = decSubtotal.ToString("N2")
    lblDiscount.Text = decDiscount.ToString("N2")
    lblTotalDue.Text = decTotalDue.ToString("N2")
End Sub
```

AssignDiscount Sub procedure

btnCalc_Click procedure

Call statement

Figure 17-13 AssignDiscount and btnCalc_Click procedures

Instead of using an independent Sub procedure to assign the discount amount, you can use a function. Consider the changes you will need to make to the code shown in Figure 17-13 in order to do this. You will make the changes in the following set of steps.

To modify the Concert Tickets application's code to use a function:

1. Open the **Concert Solution** (**Concert Solution.sln**) file contained in the ClearlyVB2012\Chap17\Concert Solution folder. If necessary, open the designer window.

2. Open the Code Editor window and locate the btnCalc_Click procedure. The Call statement will need to be replaced with a statement that invokes the AssignDiscount function (rather than the AssignDiscount Sub procedure). Like the independent Sub procedure, the function will need the statement to pass the values stored in the `intTickets` and `decSubtotal` variables because those values are used to determine the discount amount. However, it will not need the statement to pass the address of the `decDiscount` variable; this is because the statement itself will store the discount amount in that variable. Replace the Call statement with the following assignment statement, and then click **another line** within the procedure. (Don't be concerned about the jagged line that appears below a portion of the statement. It will disappear when you complete Step 4 below.)

   decDiscount = AssignDiscount(intTickets, decSubtotal)

3. Locate the AssignDiscount Sub procedure. Change the `Sub` keyword in the procedure header to **Function** and then click **another line** within the function. The Code Editor automatically changes the `Sub` keyword in the footer to the `Function` keyword. (Don't be concerned about the jagged line that appears below the function footer. It will disappear when you complete Step 7 below.)

4. Next, delete `, ByRef decDisc As Decimal` from the function header. (Be sure to delete the comma, but don't delete the ending parenthesis.) Now click **another line** within the function. A jagged line appears below each occurrence of `decDisc` in the function. The jagged line indicates that the `decDisc` variable has not been declared.

5. In order to use the `decDisc` variable, the function will need to declare it in a Dim statement. Insert a blank line above the Select Case clause. Type the following Dim statement and then press **Enter**. When you press Enter, the Code Editor removes the jagged line below each occurrence of `decDisc`.

   Dim decDisc As Decimal

6. Recall that the data type of the function's return value is specified at the end of the function header. The AssignDiscount function returns a Decimal value. Click **after the)** in the function header, press the **Spacebar**, and then type **As Decimal**. Click **another line** within the function.

7. Finally, you need to tell the function to return the discount amount to the statement that invoked it. The discount amount is stored in the `decDisc` variable. Insert a blank line below the End Select clause. Type the following Return statement, and then click **another line** in the Code Editor window:

   Return decDisc

Figure 17-14 shows the AssignDiscount function and btnCalc_Click procedure. The modified lines of code are shaded in the figure.

```
Private Function AssignDiscount(ByVal intNum As Integer,
                                ByVal decSub As Decimal) As Decimal
    Dim decDisc As Decimal

    Select Case intNum
        Case Is >= 6
            decDisc = decSub * 0.1
        Case Is >= 4
            decDisc = decSub * 0.02
        Case Else
            decDisc = 0
    End Select
    Return decDisc
End Function

Private Sub btnCalc_Click(sender As Object,
e As EventArgs) Handles btnCalc.Click
    ' displays the subtotal, discount, and total due

    Const decSTANDARD As Decimal = 32.75
    Const decVIP As Decimal = 75
    Dim intTickets As Integer
    Dim decSubtotal As Decimal
    Dim decDiscount As Decimal
    Dim decTotalDue As Decimal

    Integer.TryParse(txtTickets.Text, intTickets)

    ' calculate subtotal
    If radStandard.Checked = True Then
        decSubtotal = intTickets * decSTANDARD
    Else
        decSubtotal = intTickets * decVIP
    End If

    ' assign discount
    decDiscount = AssignDiscount(intTickets, decSubtotal)

    ' calculate total due
    decTotalDue = decSubtotal - decDiscount

    ' display subtotal, discount, and total due
    lblSubtotal.Text = decSubtotal.ToString("N2")
    lblDiscount.Text = decDiscount.ToString("N2")
    lblTotalDue.Text = decTotalDue.ToString("N2")
End Sub
```

AssignDiscount function

btnCalc_Click procedure

invokes the function and assigns its return value to a variable

Figure 17-14 AssignDiscount function and btnCalc_Click procedure

To test the modified Concert Tickets application:

1. Save the solution and then start the application. Type **4** in the Tickets box and then click the **Calculate** button. The correct subtotal, discount, and total due appear in the interface, as shown in Figure 17-15.

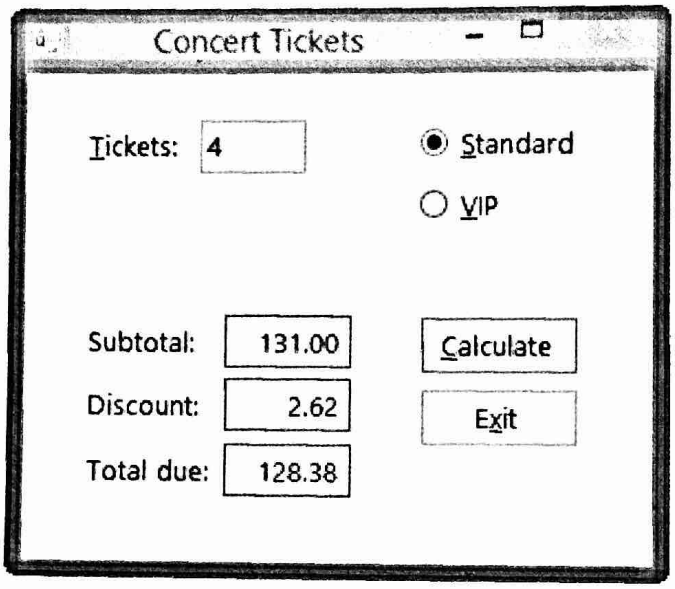

Figure 17-15 Calculated amounts shown in the interface

2. Change the number of tickets to **2**. Click the **VIP** radio button and then click the **Calculate** button. The numbers 150.00, 0.00, and 150.00 appear in the Subtotal, Discount, and Total due boxes, respectively.

3. Change the number of tickets to **6**. Click the **Standard** radio button and then click the **Calculate** button. The numbers 196.50, 19.65, and 176.85 appear in the Subtotal, Discount, and Total due boxes, respectively.

4. Click the **Exit** button. Close the Code Editor window and then close the solution.

Which Way Is Better?

Now that you've seen two different ways of assigning the discount in the Concert Tickets application—one using an independent Sub procedure and the other using a function—you may be wondering whether one way is better than the other. Comparing the Sub procedure header in Figure 17-13 with the function header in Figure 17-14, you will notice that only the Sub procedure is passed a memory location *by reference*. This is because the Sub procedure is responsible for assigning a value to the memory location. After the computer processes the Sub procedure header, the memory location has two different names (`decDiscount` and `decDisc`) and can be accessed by two different procedures (the AssignDiscount Sub procedure and the btnCalc_Click procedure). Allowing more than one procedure to change the contents of a memory location can lead to subtle errors that are difficult to find, especially in large applications. As you learned in Chapter 8, fewer unintentional errors occur in applications when memory locations have the minimum scope needed. Therefore, using a function to assign the discount is the better way to code the Concert Tickets application. Most programmers pass a variable *by reference* only when a procedure needs to produce more than one result. For example, a procedure may need to return the number of hours an employee worked and also indicate whether the hours are valid.

Ch17-Functions

For more examples of functions, see the Function ures section in the antMore.pdf file.

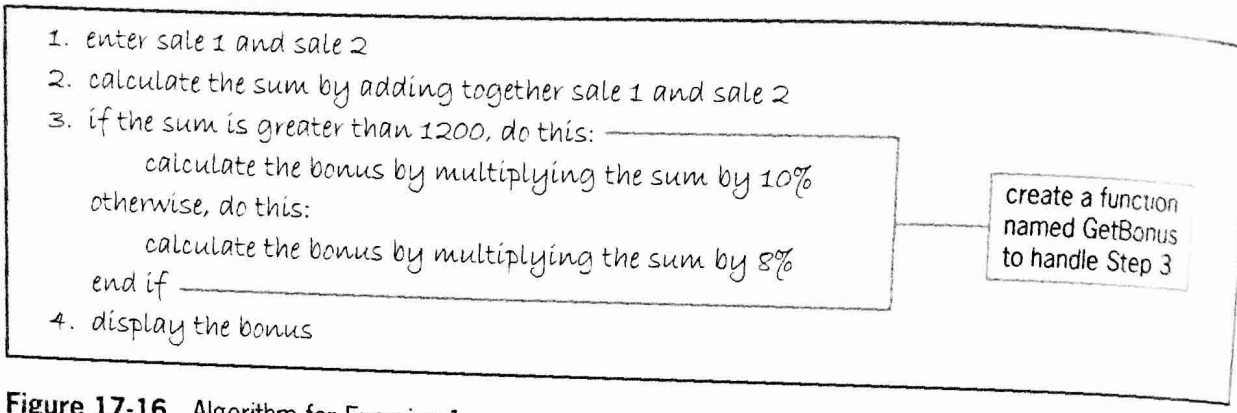

Figure 17-16 Algorithm for Exercise 1

TRY THIS

2. In this exercise, you modify the application from Exercise 1. Use Windows to make a copy of the Bonus Solution-TRY THIS 1 folder. Save the copy in the ClearlyVB2012\Chap17 folder. Rename the copy Bonus Solution-TRY THIS 2. Open the Bonus Solution (Bonus Solution.sln) file contained in the Bonus Solution-TRY THIS 2 folder. Open the designer window and then change the form's Text property to Bonus Solution-TRY THIS 2. Open the Code Editor window. The btnCalc_Click procedure should use a function to add together the two sales amounts and then return the sum. Create the GetSum function and then modify the event procedure's comments and code. Save the solution and then start and test the application. Close the Code Editor window and then close the solution. (See Appendix B for the answer.)

MODIFY THIS

3. In this exercise, you modify the Price Calculator application coded in the chapter. Use Windows to make a copy of the Price Solution folder. Save the copy in the ClearlyVB2012\Chap17 folder. Rename the copy Price Solution-MODIFY THIS. Open the Price Solution (Price Solution.sln) file contained in the Price Solution-MODIFY THIS folder. Open the designer and Code Editor windows. Modify the code so that it uses an independent Sub procedure (rather than a function) to calculate the new price. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

MODIFY THIS

4. In this exercise, you modify the Favorite Title application from Chapter 16. Open the Favorite Title Solution (Favorite Title Solution.sln) file contained in the ClearlyVB2012\Chap17\Favorite Title Solution folder. Open the designer and Code Editor windows. Change the DisplayMsg Sub procedure to a function named GetMsg. The function should return the appropriate message to the btnDisplay_Click procedure, which should display the return value in the lblMsg control. Make the necessary modifications to the code and comments. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

INTRODUCTORY

5. Open the Grade Solution (Grade Solution.sln) file contained in the ClearlyVB2012\Chap17\Grade Solution folder. The application should display a letter grade, which is based on the average of three test scores. If the average is at least 90, the grade is A. If the average is at least 80 but less than 90, the grade is B. If the average is at least 70 but less than 80, the grade is C. If the average is at least 60 but less than 70, the grade is D. If the average is below 60, the grade is F. Code the application, using a function to return the appropriate letter grade. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

Using an Element in a One-Dimensional Array

Example 1
```
Dim strCities() As String = {"Paris", "Rome", "Lisbon"}
strCities(0) = "Madrid"
```
assigns the string "Madrid" to the first element in the `strCities` array, replacing the string "Paris"

Example 2
```
Private intSalaries() As Integer = {25000, 35000, 50000, 23000}
intSalaries(3) = intSalaries(3) + 2000
```
adds 2000 to the contents of the last element in the `intSalaries` array (23000) and then assigns the result (25000) to the element

Example 3
```
Dim decSales(10) As Decimal
Decimal.TryParse(txtSales.Text, decSales(2))
lblSales.Text = decSales(2).ToString("C2")
```
assigns the value returned by the TryParse method to the third element in the `decSales` array and then displays the value (formatted with a dollar sign and two decimal places) in the lblSales control

Figure 18-3 Examples of using an element in a one-dimensional array

The procedures you code in this chapter will demonstrate some of the ways one-dimensional arrays are used in an application. In most applications, the values stored in an array come from a file on the computer's disk and are assigned to the array after it is declared. However, so that you can follow the code and its results more easily, the applications in this chapter use the array's declaration statement to populate the array with the appropriate values.

Mini-Quiz 18-1

See Appendix B for the answers.

1. Write a Visual Basic statement that declares a 20-element one-dimensional Integer array named `intQuantities`.
2. What is the highest subscript in the `intQuantities` array from Question 1?
3. Write a Visual Basic statement that assigns the number 7 to the fourth element in the `intQuantities` array.

Airlines Application

The Airlines application will store the names of four airlines in a procedure-level, one-dimensional String array named `strAirlines`. It then will display the contents of the array in three label controls named lblOriginal, lblAscending, and lblDescending. In the lblOriginal control, the names will appear in the same order as in the array. In the lblAscending and lblDescending controls, the names will appear in ascending and descending order, respectively.

To begin coding the Airlines application:

1. Start Visual Studio 2012. Open the **Airlines Solution (Airlines Solution.sln)** file contained in the ClearlyVB2012\Chap18\Airlines Solution folder. If necessary, open the designer window. (The image in the picture box was downloaded from the Open Clip Art Library at *http://openclipart.org*.)

2. Open the Code Editor window and locate the btnDisplay_Click procedure. First, you will declare and initialize the `strAirlines` array. Click the **blank line** below the `' procedure-level array` comment. Enter the following declaration statement:

   ```
   Dim strAirlines() As String =
       {"Delta", "Southwest", "United", "American"}
   ```

3. Now you will clear the contents of the three label controls. Click the **blank line** below the `' clear labels` comment. Enter the following three assignment statements. Press **Enter** twice after typing the last assignment statement.

   ```
   lblOriginal.Text = String.Empty
   lblAscending.Text = String.Empty
   lblDescending.Text = String.Empty
   ```

Next, you will display the contents of the array in the lblOriginal control. You can accomplish this task using a loop to access each element in the array, beginning with the element whose subscript is 0 and ending with the element whose subscript is 3. Because you want the loop instructions processed a precise number of times—in this case, from 0 to 3—you will code the loop using the For...Next statement. As you learned in Chapter 14, the For...Next statement provides a more convenient way to code a counter-controlled loop because it takes care of initializing and updating the counter variable, as well as evaluating the loop condition.

Figure 18-4 shows two versions of the code for displaying the contents of the `strAirlines` array in the lblOriginal control. Only the *endValue* in the For clause is different in each version. In Version 1, the endValue is 3, which represents the highest subscript in the array. In Version 2, the endValue is the expression `strAirlines.Length - 1`. The expression uses the array's **Length property**, which contains an integer that represents the number of elements in the array. In this case, the Length property contains the number 4. To determine the highest subscript in the array, you simply subtract the number 1 from the value stored in the array's Length property, as shown in the expression. Although both versions of code in Figure 18-4 will work, Version 2's code is the preferred one for several reasons. First, if you use the Length property, you won't have to count the number of array elements yourself. Second, the Length property is helpful when you don't know the exact number of array elements, which may be the case when the array values come from a file. Finally, if the number of array elements changes in the future, the Length property's value will adjust automatically.

endValue

```
Version 1
For intSub As Integer = 0 To 3
    lblOriginal.Text = lblOriginal.Text &
        strAirlines(intSub) & ControlChars.NewLine
Next intSub

Version 2                                    endValue
For intSub As Integer = 0 To strAirlines.Length - 1
    lblOriginal.Text = lblOriginal.Text &
        strAirlines(intSub) & ControlChars.NewLine
Next intSub
```

Figure 18-4 Two versions of the

To continue coding the btnDisplay_Click procedure:

1. Click the **blank line** below the `' display array in original order` comment. Enter the following lines of code:

```
For intSub As Integer = 0 To strAirlines.Length - 1
    lblOriginal.Text = lblOriginal.Text &
        strAirlines(intSub) & ControlChars.NewLine
Next intSub
```

2. Save the solution and then start the application. Click the **Display** button. The names of the airlines appear in the Original order box. As Figure 18-5 shows, the names appear in the order in which they are stored in the array. (Recall that you can use the Alt key to show/hide the access keys.)

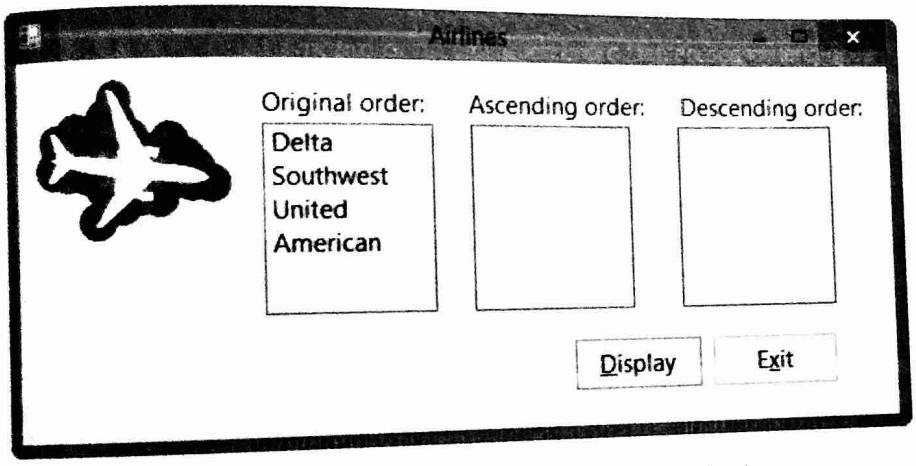

Figure 18-5 Array contents displayed in the Original order box

OpenClipArt.org/molumen

3. Click the **Exit** button.

Before displaying the array values in the Ascending order and Descending order boxes, you need to arrange the values in the appropriate order. Arranging data in a specific order is called **sorting**. Visual Basic provides the **Array.Sort method** for sorting the values in a one-dimensional array in ascending order. When a numeric array is sorted in ascending order, the first element in the array contains the smallest number and the last element contains the largest number. When a String array is sorted in ascending order, the strings appear in alphabetical order. To sort the array values in descending order, you first use the Array.Sort method to sort the values in ascending order and then use the **Array.Reverse method** to reverse the order of the values. When a numeric array is sorted in descending order, the first element in the array contains the largest number and the last element contains the smallest number. The strings in a String array, on the other hand, appear in descending alphabetical order. Figure 18-6 shows the syntax of both methods and includes examples of using the methods.

Array.Sort and Array.Reverse Methods

Syntax
Array.Sort(*arrayName*)
Array.Reverse(*arrayName*)

Example 1
```
Dim intNums() As Integer = {1, 3, 10, 2, 4, 23}
Array.Sort(intNums)
```
sorts the values in the `intNums` array in ascending order as follows: 1, 2, 3, 4, 10, 23

Example 2
```
Dim intNums() As Integer = {1, 3, 10, 2, 4, 23}
Array.Reverse(intNums)
```
reverses the order of the values in the `intNums` array as follows: 23, 4, 2, 10, 3, 1

Example 3
```
Dim intNums() As Integer = {1, 3, 10, 2, 4, 23}
Array.Sort(intNums)
Array.Reverse(intNums)
```
sorts the values in the `intNums` array in descending order as follows: 23, 10, 4, 3, 2, 1

Figure 18-6 Syntax and examples of the Array.Sort and Array.Reverse methods

To complete the btnDisplay_Click procedure:

1. Insert two blank lines below the `Next intSub` clause. Enter the following comm lines of code:

```
' display array in ascending order
Array.Sort(strAirlines)
For intSub As Integer = 0 To strAirlines.Length - 1
    lblAscending.Text = lblAscending.Text &
        strAirlines(intSub) & ControlChars.NewLine
Next intSub
```

2. Insert two blank lines below the `Next intSub` clause from the previous step. following comment and lines of code:

```
' display array in descending order
Array.Sort(strAirlines)
Array.Reverse(strAirlines)
For intSub As Integer = 0 To strAirlines.Length - 1
    lblDescending.Text = lblDescending.Text &
        strAirlines(intSub) & ControlChars.NewLine
Next intSub
```

Figure 18-7 shows the code entered in the btnDi

Modified Airlines Application

In the Airlines application coded in the previous section, the `strAirlines` array was declared and initialized in the btnDisplay_Click procedure. Because the array was a procedure-level array, it remained in the computer's internal memory only while the procedure was being processed; it was removed from memory when the procedure ended. As a result, the array will need to be recreated each time the user selects the Display button in the interface. Using a procedure-level array is fine when the array is very small, like the `strAirlines` array, which contains only four elements. However, having the computer recreate a large array every time a button is clicked is very inefficient. A better approach is to create (declare) a class-level array.

You declare a class-level array using the `Private` keyword, as shown earlier in Figure 18-2. You enter the declaration statement in the form's Declarations section, which begins with the Public Class clause and ends with the End Class clause in the Code Editor window. Memory locations (arrays, simple variables, and named constants) declared in the form's Declarations section have **class scope**, which means they can be used by any of the procedures in the form's Code Editor window. Memory locations with class scope remain in the computer's internal memory until the application ends.

Figure 18-9 shows two examples of declaring and assigning values to a class-level array. In Example 1, the array values are provided in the array's declaration statement, which is entered in the form's Declarations section. In Example 2, the array is declared in the form's Declarations section, but its values are assigned in the form's Load event procedure. A form's **Load event** occurs when the application is started and the form is displayed the first time.

Declaring and Assigning Values to a Class-Level Array

form's Declarations section

Example 1

```
Private strAirlines() As String =
    {"Delta", "Southwest", "United", "American"}
```

form's Declarations section

Example 2

```
Private strAirlines(3) As String

Private Sub frmMain_Load(sender As Object,
e As EventArgs) Handles Me.Load
    ' fills array with values

    strAirlines(0) = "Delta"
    strAirlines(1) = "Southwest"
    strAirlines(2) = "United"
    strAirlines(3) = "American"
End Sub
```

you must provide the highestSubscript argument

frmMain_Load procedure

Figure 18-9 Class-level array examples

To use a class-level array in the Airlines

3. Click the **blank line** below the Public Class clause and then press **Enter** to insert another blank line. Enter the following comment and declaration statement:

```
' class-level array
Private strAirlines() As String =
    {"Delta", "Southwest", "United", "American"}
```

4. Locate the btnDisplay_Click procedure. Delete the `' procedure-level array` comment and the array declaration statement.

5. Save the solution and then start the application. The code in the form's Declarations section is processed first. The code declares and initializes the `strAirlines` array. The array will remain in memory until the application ends.

6. Click the **Display** button. The button's Click event procedure displays the array values in their original, ascending, and descending order, as shown earlier in Figure 18-8.

7. Click the **Exit** button. When the application ends, the computer removes the `strAirlines` array from its internal memory.

8. Close the Code Editor window and then close the solution.

Figure 18-10 shows the code entered in the form's Declarations section and the btnDisplay_Click procedure.

```
' class-level array                                          form's Declarations
Private strAirlines() As String =                            section
    {"Delta", "Southwest", "United", "American"}

Private Sub btnDisplay_Click(sender As Object,               btnDisplay_Click
e As EventArgs) Handles btnDisplay.Click                     procedure
    ' displays array in original, ascending,
    ' and descending order

    ' clear labels
    lblOriginal.Text = String.Empty
    lblAscending.Text = String.Empty
    lblDescending.Text = String.Empty

    ' display array in original order
    For intSub As Integer = 0 To strAirlines.Length - 1
        lblOriginal.Text = lblOriginal.Text &
            strAirlines(intSub) & ControlChars.NewLine
    Next intSub

    ' display array in ascending order
    Array.Sort(strAirlines)
    For intSub As Integer = 0 To strAirlines.Length - 1
        lblAscending.Text = lblAscending.Text &
            strAirlines(intSub) & ControlChars.NewLine
    Next intSub

    ' display array in descending order
    Array.Sort(strAirlines)
    Array.Reverse(strAirlines)
    For intSub As Integer = 0 To strAirlines.Length - 1
        lblDescending.Text = lblDescending.Text &
            strAirlines(intSub) & ControlChars.NewLine
    Next intSub
End Sub
```

Figure 18-10 Code entered in the modified Airlines application

Salary Application

The Salary application will store six salary amounts in a one-dimensional Integer array named `intSalaries`. Each salary amount corresponds to a salary code; the valid codes are the numbers 1 through 6. Code 1's salary is stored in the `intSalaries(0)` element in the array, code 2's salary in the `intSalaries(1)` element, and so on. Notice that the code is one number more than the subscript of its corresponding salary in the array. After storing the salary amounts in the array, the application will prompt the user to enter a salary code. It then will display the amount associated with the code. Figure 18-11 shows the planning information for the application.

```
Output:      salary amount

Processing:  six-element one-dimensional array of salary amounts (class-level)
             array subscript

Input:       salary code (1 through 6)

Algorithm:
1. enter the salary code
2. calculate the subscript of the array element associated with the salary code
   by subtracting 1 from the salary code
3. use the array subscript to display the associated salary amount from the array
```

Figure 18-11 Planning information for the Salary application

To code and then test the Salary application:

1. Open the **Salary Solution (Salary Solution.sln)** file contained in the ClearlyVB2012\Chap18\Salary Solution folder. If necessary, open the designer window. (The image in the picture box was downloaded from the Open Clip Art Library at *http://openclipart.org*.)
2. Open the Code Editor window. First, you will declare a class-level array that will store the six salary amounts. Click the **blank line** below the `' class-level array` comment and then enter the following declaration statement:

   Private intSalaries(5) As Integer
3. Now you will have the form's Load event procedure assign values to the array. Click the **Class Name** list arrow and then click **(frmMain Events)** in the list. Click the **Method Name** list arrow and then click **Load** in the list. The code template for the form's Load event procedure appears in the Code Editor window. Type the following comment and then press **Enter** twice:

   ' fills array with values
4. Enter the following assignment statements:

   intSalaries(0) = 45000
   intSalaries(1) = 33700
   intSalaries(2) = 25600
   intSalaries(3) = 22500
   intSalaries(4) = 70000
   intSalaries(5) = 75000

5. Open the code template for the btnDisplay_Click procedure. Type the following comment and then press **Enter** twice:

```
' displays the salary amount associated with a code
```

6. First, you will declare the procedure's variables. Enter the following declaration statements. Press **Enter** twice after typing the last declaration statement.

```
Dim intCode As Integer
Dim intSub As Integer
```

7. The first step in the algorithm is to enter the salary code. The user enters the code in the txtCode control. You will use the TryParse method to convert the code to the Integer data type, storing the result in the `intCode` variable. Enter the following TryParse method:

```
Integer.TryParse(txtCode.Text, intCode)
```

8. Step 2 in the algorithm calculates the subscript of the array element associated with the salary code. This is accomplished by subtracting the number 1 from the code. Enter the following comment and assignment statement:

```
' calculate the appropriate subscript
intSub = intCode - 1
```

9. The last step in the algorithm uses the array subscript to display the appropriate salary amount from the array. Enter the following comment and assignment statement:

```
' display the salary amount from the array
lblSalary.Text = intSalaries(intSub).ToString("C0")
```

10. Save the solution and then start the application. Type **2** in the Code box and then click the **Display Salary** button. The salary amount stored in the `intSalaries(1)` element appears in the Salary box, as shown in Figure 18-12.

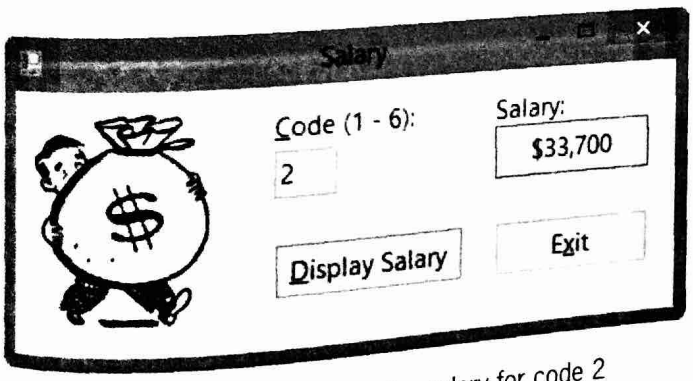

Figure 18-12 Interface showing the salary for code 2

OpenClipArt.org/johnny_automatic

11. Now you will observe the result of entering an invalid salary code. Replace the 2 in the Code box with **8** and then click the **Display Salary** button. A run time error occurs. As a result, the Code Editor highlights the statement where the error was encountered. The Error Correction window indicates that the index (which is another term for subscript) is outside the bounds of the array.

12. Place your mouse pointer on `intSub` in the highlighted statement, as shown in Figure 18-13. The variable contains the number 7, which is one number less than the salary code you entered. The valid subscripts for the array, however, are the numbers 0 through 5 only.

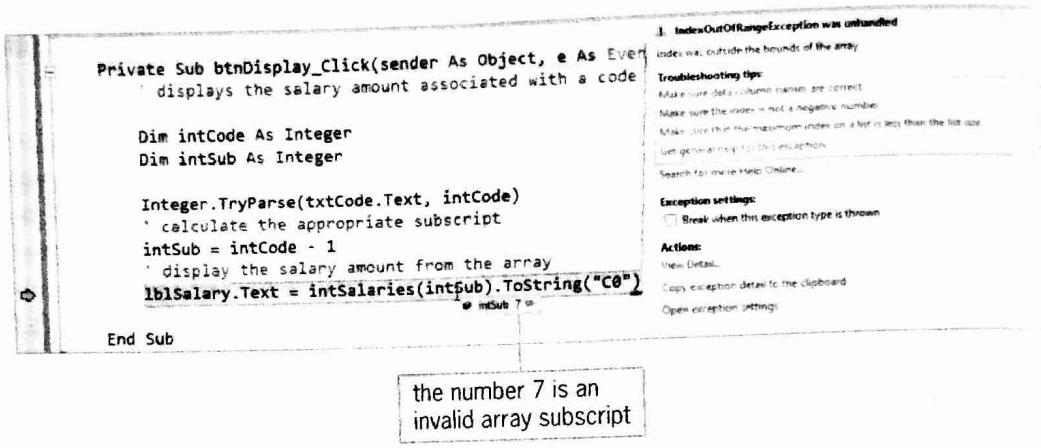

Figure 18-13 Result of the run time error caused by an invalid array subscript

13. Click **DEBUG** on the menu bar and then click **Stop Debugging**.

Before attempting to access an array element, a procedure should verify that the subscript being used is valid. This can be accomplished using a selection structure whose condition verifies that the subscript is within an acceptable range for the array. The acceptable range is a number that is greater than or equal to 0 but less than the number of array elements. If the subscript is not in the acceptable range, the procedure should not try to access the element because doing so results in a run time error. Figure 18-14 shows the modified algorithm for the Salary application. Notice that Step 3 now validates the array subscript.

```
Output:      salary amount

Processing:  six-element one-dimensional array of salary amounts (class-level)
             array subscript

Input:       salary code (1 through 6)

Algorithm:
1. enter the salary code
2. calculate the subscript of the array element associated with the salary code
   by subtracting 1 from the salary code
3. if the array subscript is greater than or equal to 0 but less than the
   number of array elements, do this:
        use the array subscript to display the associated salary amount from the array
   otherwise, do this:
        display a message stating the valid salary codes
   end if
```

Figure 18-14 Modified algorithm for the Salary application

To modify and then test the btnDisplay_Click procedure:

1. Insert a blank line above the last comment and then enter the selection structure shown in Figure 18-15. (You will need to move the last comment and assignment statement, which you entered in Step 9 in the previous set of steps, into the selection structure's true path.)

```
Private Sub btnDisplay_Click(sender As Object,
e As EventArgs) Handles btnDisplay.Click
    ' displays the salary amount associated with a code

    Dim intCode As Integer
    Dim intSub As Integer

    Integer.TryParse(txtCode.Text, intCode)
    ' calculate the appropriate subscript
    intSub = intCode - 1
    If intSub >= 0 AndAlso intSub < intSalaries.Length Then
        ' display the salary amount from the array
        lblSalary.Text = intSalaries(intSub).ToString("C0")
    Else
        MessageBox.Show("Valid salary codes are 1 through 6.",
                        "Salary", MessageBoxButtons.OK,
                        MessageBoxIcon.Information)
    End If

End Sub
```

enter this selection structure

Figure 18-15 Modified btnDisplay_Click procedure

2. Save the solution and then start the application. Type **3** in the Code box and then click the **Display Salary** button. $25,600 appears in the Salary box.
3. Replace the 3 in the Code box with **9** and then click the **Display Salary** button. The message "Valid salary codes are 1 through 6." appears in a message box. Close the message box.
4. On your own, test the application using salary codes of 2 through 6. Also test it by clicking the Display Salary button when the Code box is empty.
5. When you are finished testing the application, click the **Exit** button. Close the Code Editor window and then close the solution.

States Application

The States application stores the names of nine states in a one-dimensional String array named strStates. The names are stored in the order each was visited by the user. For example, the user visited Hawaii first, followed by Colorado. Therefore, the strings "Hawaii" and "Colorado" are stored in the `strStates(0)` and `strStates(1)` elements, respectively. The application's interface provides a text box for the user to enter a state name. The application searches for the state name in the array, beginning with the first array element. It then displays a message indicating whether the state name was found. Sample messages include "Hawaii is number 1 in the list of states you visited." and "You did not visit Illinois." Figure 18-16 shows the planning information for the application.

MODIFY THIS

3. In this exercise, you modify the application coded in Exercise 2. Use Windows make a copy of the Grades Solution folder. Save the copy in the ClearlyVB2012\Ch folder. Rename the copy Grades Solution–MODIFY THIS. Open the Grades tion (Grades Solution.sln) file contained in the Grades Solution–MODIFY THIS f Open the designer and Code Editor windows. Change the For...Next stateme n the btnCount_Click procedure to a Do...Loop statement. Save the solution and t start and test the application. Close the Code Editor window and then close the so on.

INTRODUCTORY

4. Open the Tea Emporium Solution (Tea Emporium Solution.sln) file contained the Tea Emporium Solution folder. Open the Code Editor window. The btnDisplay Click procedure should display the total of the values stored in the `decPounds` array. Code the procedure. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

INTRODUCTORY

5. Open the NumDays Solution (NumDays Solution.sln) file contained in the ClearlyVB2012\Chap18\NumDays Solution folder. (The image in the picture box was downloaded from the Open Clip Art Library at *http://openclipart.org*.) Declare a class-level, 12-element one-dimensional array that contains the number of days in each month; use 28 for February. The txtMonth control should accept only numbers and the Backspace key; code the appropriate procedure. The btnDisplay_Click procedure should display the number of days corresponding to the month number entered by the user. For example, if the user enters the number 1, the procedure should display 31 because there are 31 days in January. The procedure should display an appropriate message in a message box when the user enters an invalid month number. Code the procedure. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

INTRODUCTORY

6. Open the Update Prices Solution (Update Prices Solution.sln) file contained in the ClearlyVB2012\Chap18\Update Prices Solution folder. The Update button's Click event procedure should display the contents of the `dblPrices` array in the lblOriginal control. It then should increase each price by $2 and display the updated prices in the lblUpdated control. Display the original and updated prices with two decimal places. Code the procedure. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

INTRODUCTORY

7. Open the Scores Solution (Scores Solution.sln) file contained in the ClearlyVB2012\Chap18\Scores Solution folder. Declare a class-level, 20-element one-dimensional array containing the following integers: 88, 72, 99, 20, 66, 95, 99, 100, 72, 88, 78, 45, 57, 89, 85, 78, 75, 88, 72, and 88. Open the code template for the btnDisplay control's Click event procedure. The procedure should prompt the user to enter a score from 0 through 100. It then should display (in a message box) the number of students who earned that score. Save the solution and then start the application. How many students earned a score of 72? How many earned a score of 88? How many earned a score of 20? How many earned a score of 99? Close the Code Editor window and then close the solution.

INTRODUCTORY

8. Open the Tips Solution (Tips Solution.sln) file contained in the ClearlyVB2012\Chap18\Tips Solution folder. (The image in the picture box was downloaded from the Open Clip Art Library at *http://openclipart.org*.) Declare a class-level, 20-element one-dimensional array containing the following tip amounts: 101, 95, 67, and 83. The btnForNext_Click procedure should use the For...Next statement to calculate the average tip. The btnDoLoop_Click procedure should use the Do...Loop statement to calculate the average tip. Save the solution and then start the application. Close the Code Editor window and then close the solution.