Determine a Memory Location's Scope and Lifetime

Besides a ***name***, a ***data type***, and an ***initial value***, every variable also has a ***scope*** and a ***lifetime***. The scope indicates where the declared memory location can be used in an application's code, and the lifetime indicates how long the variable remains in the computer's main memory.

The scope and lifetime are determined by where you declare the memory location in your code. Memory locations declared in a procedure have procedure scope and are called ***procedure-level variables***. These variables can be used only within the procedure that contains their declaration statement, and only after their declaration statement. They are removed from the computer's main memory when the procedure ends. In other words, a procedure level variable has the same lifetime as the procedure that declares it .

Memory locations declared in a form class's declarations section, but outside of any procedures, have class scope and are referred to as ***class-level variables***. The form class's declarations section is the area between the form's Public Class and End Class clauses in the Code Editor window. However, class-level declaration statements typically appear immediately after the Public Class <formname> clause .

Class-level variables can be used by all of the procedures in the class that contains their declaration statement. In addition, they have the same lifetime as the application, which means they retain their values and remain in the computer's main memory until the application ends.

Use Procedure-Level Variables

Procedure-level variables are typically declared at the beginning of a procedure, and they can be used only after their declaration statement within the procedure. They remain in the computer's main memory only while the procedure is running, and they are removed from memory when the procedure ends. As mentioned earlier, most of the variables in your applications will be procedure-level variables. This is because fewer unintentional errors

occur in applications when the variables are declared using the minimum scope needed, which usually is procedure scope.

The Commission Calculator application illustrates the use of procedure-level variables. As the interface shown in Figure 2-18 indicates, the application displays the amount of a salesperson's commission. The commission is calculated by multiplying the salesperson's sales by the appropriate commission rate: either 8% or 10%.

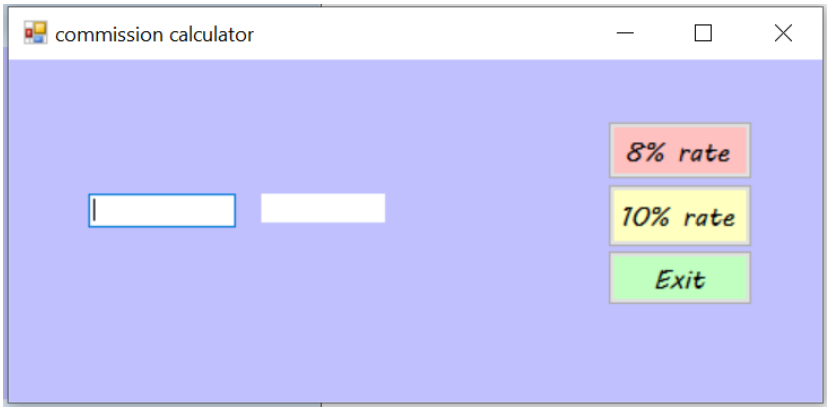

Figure 2-18 User interface for the Commission Calculator application

Figure 2-19 shows the Click event procedures for the 8% rate and 10% rate buttons. When each procedure ends, its procedure-level variables are removed from the computer's memory. The variables will be created again the next time the user clicks the button.

```
1   Public Class Form1
2       Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
3           Dim sales As Double
4           Dim comm As Double
5           sales = TextBox1.Text
6           comm = sales * 0.08
7           Label1.Text = comm
8           Label1.BackColor = Button1.BackColor
9       End Sub
10
11      Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
12          Dim sales As Double
13          Dim comm As Double
14          sales = TextBox1.Text
15          comm = sales * 0.1
16          Label1.Text = comm
17          Label1.BackColor = Button2.BackColor
18      End Sub
19
20      Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
21          Close()
22      End Sub
23  End Class
```

Figure 2-19 Click event procedures using procedure-level variables

Notice that both procedures in Figure 2-19 declare a variable named *sales*. When you use the same name to declare a variable in more than one procedure, each procedure creates its own variable when the procedure is invoked. Each procedure also destroys its own variable when the procedure ends. So, although both procedures declare a variable named *sales*, each sales

variable will refer to a different section in the computer's main memory, and each will be both created and destroyed independently from the other.

Use a Class-Level Variable

Class-level variables are declared immediately after the Public Class clause in the Code Editor window, and they can be used by any of the procedures entered in the window. Class-level variables retain their values and remain in the computer's main memory until the application ends. Figure 2-20 shows the syntax and examples of declaring a class-level variable. As the figure indicates, class-level variables are declared using the Private keyword.

Declaring a Class-Level Variable

Syntax
Private *variableName* **As** *dataType* [= *initialValue*]

Note: Class-level variables are declared immediately after the Public Class clause, and they must be declared outside of any procedure.

Examples
```
Private Quantity As Integer
Private Sales As Decimal
Private IsInsured As Boolean = True
```

Figure 2-20 Syntax and examples of declaring class-level variables

You can use a class-level variable when two or more procedures in the same form need access to the same variable. You can also use a class-level variable when a procedure needs to retain a variable's value even after the procedure ends. This use of a class-level variable is illustrated in the Total Scores Accumulator application, which calculates and displays the total of the scores entered by the user. The application's interface is shown in Figure 2-21.

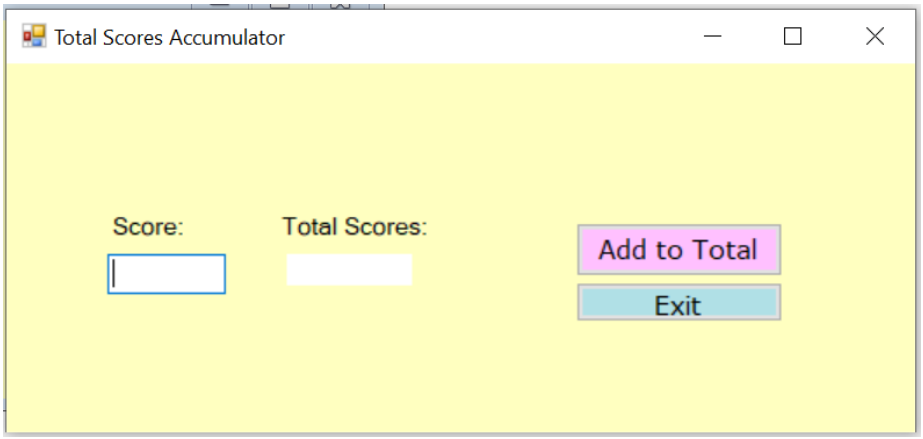

Figure 2-21 User interface for the Total Scores Accumulator application

Figure 2-22 shows most of the application's code, which uses a class-level variable named *total* to accumulate (add together) the *scores* entered by the user.

```
1  Public Class Form1
2      Private total As Double
3      Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
4          Dim score As Double
5          score = TextBox1.Text
6          total = total + score
7          Label1.Text = total
8      End Sub
9
10     Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
11         Close()
12     End Sub
13 End Class
```

Figure 2-22 The application's code using a class-level variable

When the user starts the application, the computer processes the ***Private total As Double*** statement first. The statement creates and initializes (to 0) the class-level total variable. The variable is created and initialized only once, when the application starts. It remains in the computer's main memory until the application ends. Each time the user clicks the Add to total button, the Dim statement on Line 4 in the button1_Click procedure creates and initializes a procedure-level variable named score. The statement in Line 5 stores the content of TextBox1.Text property to the *score* variable. The assignment statement on Line 6 adds the contents of the procedure-level score variable to the contents of the class-level total variable. At this point, the total variable contains the sum of all of the scores entered so far. The assignment statement on Line 7 assigns the total variable's value to the Label1.Text property. When the procedure ends, the computer removes the procedure level score variable from its main memory. However, it does not remove the class-level total variable. The total variable is removed from the computer's memory only when the application ends. As mentioned earlier, a class-level variable can be accessed by any of the procedures entered in its Code Editor window. As a result, using class-level variables can lead to unexpected results when one of the procedures makes an inadvertent or incorrect change to the variable's value. Tracking down the errors in an application's code becomes more complicated as the number of procedures having access to the same variable increases. Therefore, the use of class-level variables should be minimized. Always keep in mind that fewer unintentional errors occur when an application's variables are declared using the minimum scope needed, which usually is procedure scope. Rather than using a class-level variable to accumulate values, you can use a static variable.

Use a Static Variable

A static variable is a procedure-level variable that remains in memory and also retains its value even when its declaring procedure ends. Like a class-level variable, a static variable is not removed from the computer's main memory until the application ends. However, unlike a class-level variable, a static variable can be used only by the procedure in which it is declared. In other words, a static variable has a narrower (or more restrictive) scope than does a class-level variable. As mentioned earlier, many unintentional errors in your code can be avoided by simply declaring the variables using the minimum scope needed. Figure 2-23 shows the syntax and examples of declaring static variables. Keep in mind that the Static keyword can be used only in a procedure.

Declaring a Static Variable

Syntax
Static *variableName* **As** *dataType* [= *initialValue*]

Note: Static variables are declared in a procedure.

Examples
```
Static Total As Double
Static Count As Integer = 1
```

Figure 2-23 Syntax and examples of declaring static variables

The Total Scores Accumulator application from the previous section used a class-level variable to accumulate the scores entered by the user. Rather than using a class-level variable for that purpose, you can also use a static variable, as shown in the code in Figure 2-24.

```
1  Public Class Form1
2      Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
3          Dim score As Double
4          Static total As Double
5          score = TextBox1.Text
6          total = total + score
7          Label1.Text = total
8      End Sub
9
10     Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
11         Close()
12     End Sub
13 End Class
```

Figure 2-24 The application's code using a static variable

The first time the user clicks the Add to total button, the button's procedure creates and initializes (to 0) a procedure-level variable named score and a static variable named total. The assignment statement on Line 5 stores the content of TextBox1.Text property in the *score* variable. The assignment statement on Line 6 adds the contents of the score variable to the contents of the static total variable. The assignment statement on Line 7 assigns the total variable's value to the Label1.Text property. When the procedure ends, the computer removes the variable declared using the Dim keyword(score) from its main memory. But it does not remove the variable declared using the Static keyword (total). Each subsequent time the user clicks the Add to total button, the computer recreates and reinitializes the score variable. However, it does not recreate or reinitialize the static total variable because that variable, as well as its current value, is still in the computer's memory. After recreating and reinitializing the score variable, the computer processes the remaining instructions contained in the button's procedure. Here again, each time the procedure ends, the score variable is removed from the computer's main memory. The total variable is removed only when the application ends.