The statement in Example 1 selects all of the records in the dataset and assigns the records to the `records` variable. The statement in Example 2 performs the same task; however, the records are assigned in ascending order by the Platform field. If you are sorting records in ascending order, you do not need to include the keyword `Ascending` in the Order By clause because `Ascending` is the default. The statement in Example 3 assigns only the new video game records to the `records` variable. The statement in Example 4 performs the same task; however, the records are assigned in descending order by the Price field. The statement in Example 5 uses the Like operator and the asterisk pattern-matching character to select only records whose Title field begins with the letter M. You learned about the Like operator and its pattern-matching characters in Chapter 23.

The Red Dragon Games Application

You will use the tblGames table and LINQ to code the Red Dragon Games application. The application will display records whose Title field begins with one or more characters entered by the user. It will also allow the user to display only the new video game records or only the used video game records. The user will also be able to calculate and display the average price of a video game.

To begin coding the Red Dragon Games application:

1. Start Visual Studio 2012. Open the **Red Dragon Solution (Red Dragon Solution.sln)** file contained in the ClearlyVB2012\Chap25\Red Dragon Solution folder. If necessary, open the designer window. See Figure 25-3.

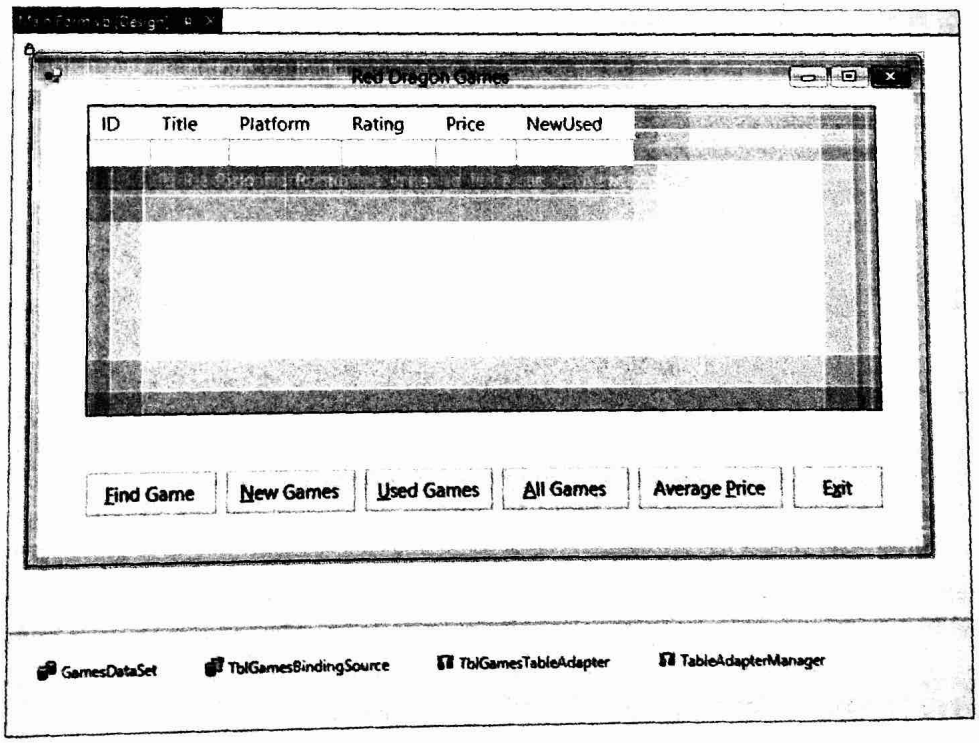

Figure 25-3 User interface for the Red Dragon Games application

2. Open the Code Editor window and locate the btnFind_Click procedure. First, you will use the InputBox function to prompt the user to either enter one or more characters or leave the input area empty. Click the **blank line** below the `' get user input` comment, and then enter the following assignment statement:

   strSearch = InputBox(strPROMPT, "Find Game").ToUpper

3. Now you will enter the LINQ statement to select the appropriate records. Click the **blank line** below the `' select the appropriate records` comment and then enter the following LINQ statement:

   ```
   Dim records = From game In GamesDataSet.tblGames
                 Where game.Title.ToUpper Like strSearch & "*"
                 Select game
   ```

   Important note: If a jagged line appears below `records` in the LINQ statement, click TOOLS on the menu bar and then click Options. Expand the Projects and Solutions node and then click VB Defaults. If necessary, change Option Strict and Option Infer to Off and On, respectively.

The LINQ statement merely selects the records and assigns them to the `records` variable. To actually view the records, you need to assign the variable's contents to the DataSource property of a BindingSource object. The syntax for doing this is shown in Figure 25-4 along with an example of using the syntax.

Assigning a LINQ Variable's Contents to a BindingSource Object

Basic syntax

bindingSource.**DataSource** = *variableName*.**AsDataView**

Example

```
TblGamesBindingSource.DataSource = records.AsDataView
```

assigns the contents of the `records` variable to the TblGamesBindingSource object

Figure 25-4 Assigning a LINQ variable's contents to a BindingSource object

To complete the btnFind_Click procedure:

1. Click the **blank line** below the `' display the records` comment and then enter the following assignment statement:

   TblGamesBindingSource.DataSource = records.AsDataView

Figure 25-5 shows the code entered in the Find Game button's Click event procedure.

```
Private Sub btnFind_Click(sender As Object,
e As EventArgs) Handles btnFind.Click
    ' displays records with titles
    ' beginning with user's entry


    Const strPROMPT As String =
        "Enter one or more characters in the " &
        "title. " &
        "Leave blank to retrieve all records."
    Dim strSearch As String

    ' get user input
    strSearch = InputBox(strPROMPT, "Find Game").ToUpper

    ' select the appropriate records
    Dim records = From game In GamesDataSet.tblGames
                  Where game.Title.ToUpper Like strSearch & "*"
                  Select game

    ' display the records
    TblGamesBindingSource.DataSource = records.AsDataView

End Sub
```

Figure 25-5 Find Game button's Click event procedure

To test the btnFind_Click procedure:

1. Save the solution and then start the application. First, you will select all of the records whose Title field begins with the letter L. Click the **Find Game** button. Type **l** in the Find Game dialog box and then press **Enter**. The records whose title begins with the letter L appear in the DataGridView control. See Figure 25-6.

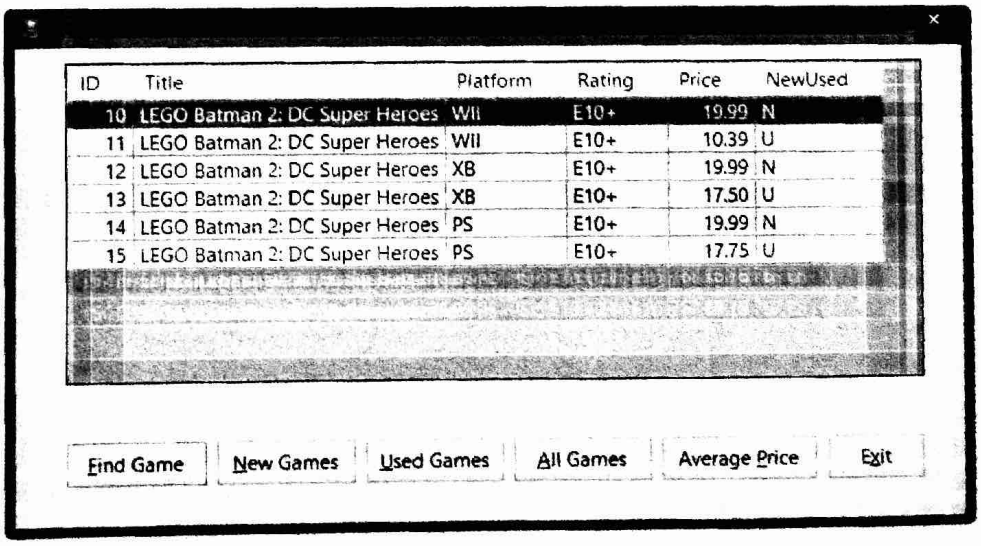

Figure 25-6 Records whose Title field begins with the letter L

2. Now you will display the records whose Title field begins with the word "The". Click the **Find Game** button. Type **the** and then press the **Spacebar**. Press **Enter**. The four records whose title begins with the word "The" appear in the DataGridView control.
3. You can use the Find Game button to display all of the records in the dataset. Click the **Find Game** button and then click the **OK** button. The 35 records appear in the DataGridView control.
4. Click the **Exit** button.

In the next set of steps, you will code the Click event procedures for the New Games, Used Games, and All Games buttons.

To code the three buttons' Click event procedures:

1. Locate the btnNew_Click procedure. The procedure should display only the new video game records. Click the **blank line** above the End Sub clause and then enter the two statements shown in Figure 25-7.

```
Private Sub btnNew_Click(sender As Object,
e As EventArgs) Handles btnNew.Click
' displays only new game records in the dataset

    Dim records = From game In GamesDataSet.tblGames
                  Where game.NewUsed.ToUpper = "N"
                  Select game

    TblGamesBindingSource.DataSource = records.AsDataView
End Sub
```

enter these statements

Figure 25-7 New Games button's Click event procedure

2. Save the solution and then start the application. Click the **New Games** button. Only the 21 records that have N in the NewUsed field appear in the DataGridView control. See Figure 25-8. (You will need to scroll the control to view all of the new records.)

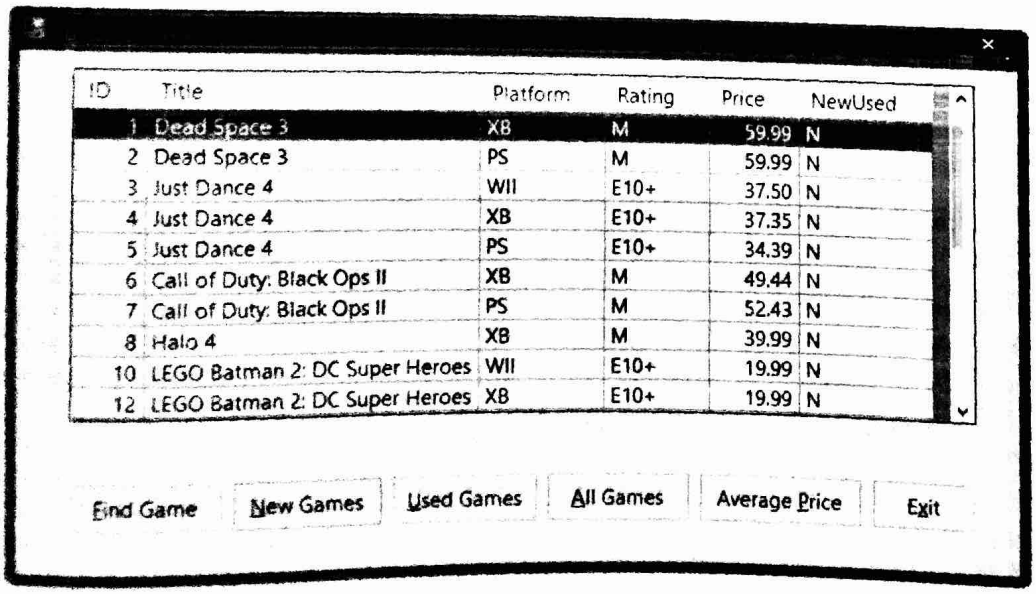

Figure 25-8 New video game records

3. Click the **Exit** button.

4. Locate the btnUsed_Click procedure. The procedure should display only the used video game records. Click the **blank line** above the End Sub clause and then enter the two statements shown in Figure 25-9.

enter these statements

```
Private Sub btnUsed_Click(sender As Object,
e As EventArgs) Handles btnUsed.Click
    ' displays only used game records in the dataset

    Dim records = From game In GamesDataSet.tblGames
                  Where game.NewUsed.ToUpper = "U"
                  Select game

    TblGamesBindingSource.DataSource = records.AsDataView
End Sub
```

Figure 25-9 Used Games button's Click event procedure

5. Save the solution and then start the application. Click the **Used Games** button. Only the 14 records that have U in the NewUsed field appear in the DataGridView control. (You will need to scroll the control to view all of the used records.) See Figure 25-10.

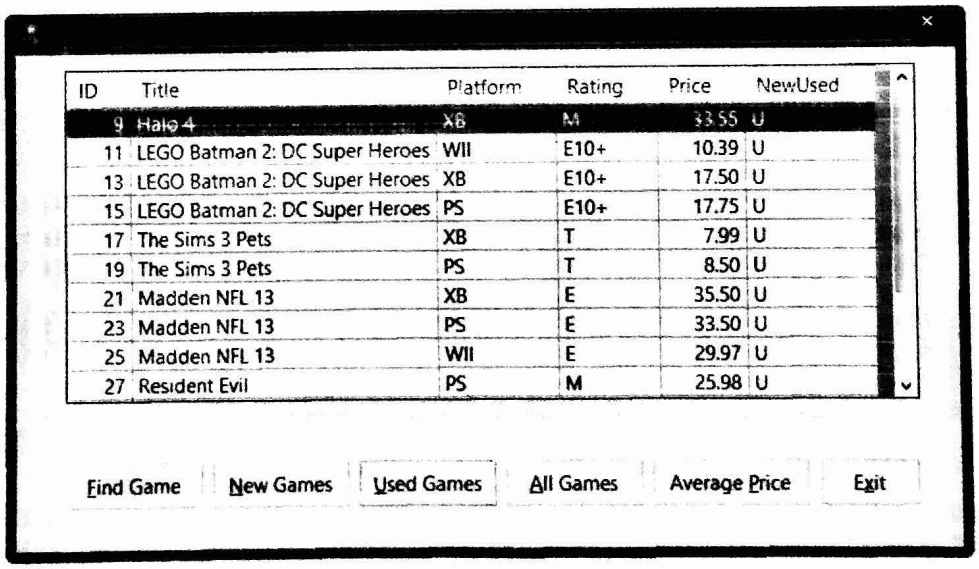

Figure 25-10 Used video game records

6. Click the **Exit** button.

7. Locate the btnAll_Click procedure. The procedure should display all of the records. Click the **blank line** above the End Sub clause and then enter the two statements shown in Figure 25-11.

```
Private Sub btnAll_Click(sender As Object,
e As EventArgs) Handles btnAll.Click
    ' displays all of the records in the dataset

    Dim records = From game In GamesDataSet.tblGames
                  Select game

    TblGamesBindingSource.DataSource = records.AsDataView
End Sub
```

enter these statements

Figure 25-11 All Games button's Click event procedure

8. Save the solution and then start the application. Click the **Used Games** button to display the 14 used records, and then click the **All Games** button to display the 35 records contained in the dataset. (You will need to scroll the control to view all of the records.) Click the **Exit** button.

Mini-Quiz 25-1

See Appendix B for the answers.

1. Complete the Where clause in the following statement, which should select only records whose LastName field begins with an uppercase letter A:

```
Dim records = From name In NamesDataSet.tblNames
              Where _____
              Select name
```

2. Complete the following statement, which should arrange the records in descending order by the LastName field:

```
Dim records = From name In NamesDataSet.tblNames
              _____
              Select name
```

3. What does LINQ stand for?
4. In a LINQ statement, which clause is used to arrange the selected records in a particular order?
   a. Order
   b. Order By
   c. Sort
   d. Where
5. In a LINQ statement, which clause contains a condition?
   a. Order
   b. Order By
   c. Sort
   d. Where

and then start and test the application. Close the Code Editor window and then close the solution.

INTERMEDIATE

8. In this exercise, you code a different version of the Red Dragon Games application from the chapter. Open the Red Dragon Solution (Red Dragon Solution.sln) file contained in the ClearlyVB2012\Chap25\Red Dragon Solution-CheckBoxes folder. The Display button's Click event procedure should display only the video games for the specific ESRB rating(s) chosen by the user. (Hint: You can use a logical operator in the Where clause.) Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

ADVANCED

9. Open the MusicBox Solution (MusicBox Solution.sln) file contained in the ClearlyVB2012\Chap25\MusicBox Solution folder. The application is connected to the MusicBox database stored in the MusicBox.accdb file. The database contains a table named tblBox. The table contains 10 records, each composed of four text fields.

   a. Start the application to view the records contained in the dataset, and then stop the application.
   b. Open the Code Editor window. Code the btnAll_Click procedure so that it displays all the records.
   c. Code the btnShape_Click procedure so that it displays the records for music boxes having the shape selected by the user.
   d. Code the btnSource_Click procedure so that it displays the records for music boxes either received as gifts or purchased by the user.
   e. Code the btnCount_Click procedure to display (in a message box) the number of music boxes in the dataset.
   f. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

ADVANCED

10. Open the Trips Solution (Trips Solution.sln) file contained in the ClearlyVB2012\Chap25\Trips Solution folder. The application is connected to a Microsoft Access database named Trips, which keeps track of a person's business and pleasure trips. The database contains one table named tblTrips. The table has 10 records. Each record has the following four text fields: TripDate, Origin, Destination, and BusinessPleasure. The application should allow the user to display the number of trips from a specific origin to a specific destination, such as from Chicago to Atlanta. (Hint: You can use a logical operator in the Where clause.) It should also allow the user to display (in a message box) either the total number of business trips or the total number of pleasure trips.

    a. Open the Code Editor window and code the application.
    b. Save the solution and then start the application. Use the application to answer the following questions:

       How many trips were made from Nashville to Chicago?
       How many trips were made from Atlanta to Los Angeles?
       How many business trips were taken?
       How many pleasure trips were taken?

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

ADVANCED

11. In this exercise, you code a different version of the Red Dragon Games application from the chapter. Open the Red Dragon Solution (Red Dragon Solution.sln) file contained in the ClearlyVB2012\Chap25\Red Dragon Solution-ADVANCED folder. The Display button's Click event procedure should display only the video games for the specific ESRB rating and platform chosen by the user. (Hint: You can use a logical operator in the Where clause.) Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

Class Statement

Syntax
Public Class *className*
    attributes section
    behaviors section
End Class

Adding a class file to an open project
1. Click PROJECT on the menu bar and then click Add Class. The Add New Item dialog box opens with Class selected in the middle column.
2. Type the name of the class followed by a period and the letters vb in the Name box, and then click the Add button.

Figure 26-1 Syntax of the Class statement

The Bonnette Pool & Spa Depot Application

You will use the Class statement in the Bonnette Pool & Spa Depot application from Chapter 21. As you may remember, the application determines the amount of water required to fill a rectangular pool. To accomplish this task, the application first calculates the volume of the pool. You calculate the volume by multiplying the pool's length by its width and then multiplying the result by the pool's depth. Assuming the length, width, and depth are measured in feet, this gives you the volume in cubic feet. To determine the number of gallons of water, you multiply the number of cubic feet by 7.48 because there are 7.48 gallons in one cubic foot.

In Chapter 21, you used a structure to group together the pool's length, width, and depth measurements. Recall that it's logical to group the three items because each represents one of the three dimensions of a rectangular pool. Most of the application's code from Chapter 21 is shown in Figure 26-2. In this chapter, you will modify the code to use a class rather than a structure.

form's Declarations section

receives a structure variable *by value*

declares a structure variable

assigns the input data to the structure variable

passes the structure variable to the GetGallons function

```
Structure Dimensions
    Public dblLength As Double
    Public dblWidth As Double
    Public dblDepth As Double
End Structure

Public Function GetGallons(ByVal pool As Dimensions) As Double
    ' calculates and returns the number of gallons

    Const dblGAL_PER_CUBIC_FOOT As Double = 7.48

    Return pool.dblLength * pool.dblWidth *
        pool.dblDepth * dblGAL_PER_CUBIC_FOOT
End Function

Private Sub btnCalc_Click(sender As Object,
e As EventArgs) Handles btnCalc.Click
    ' displays the number of gallons

    Dim poolSize As Dimensions
    Dim dblGallons As Double

    Double.TryParse(txtLength.Text, poolSize.dblLength)
    Double.TryParse(txtWidth.Text, poolSize.dblWidth)
    Double.TryParse(txtDepth.Text, poolSize.dblDepth)

    dblGallons = GetGallons(poolSize)
    lblGallons.Text = dblGallons.ToString("N0")
End Sub
```

Figure 26-2 Code for the Bonnette Pool & Spa Depot application (with a structure)

To begin creating the class:

1. Start Visual Studio 2012. Open the **Bonnette Solution (Bonnette Solution.sln)** file contained in the ClearlyVB2012\Chap26\Bonnette Solution folder. If necessary, open the designer window. See Figure 26-3. (The image in the picture box was downloaded from the Open Clip Art Library at *http://openclipart.org*.)

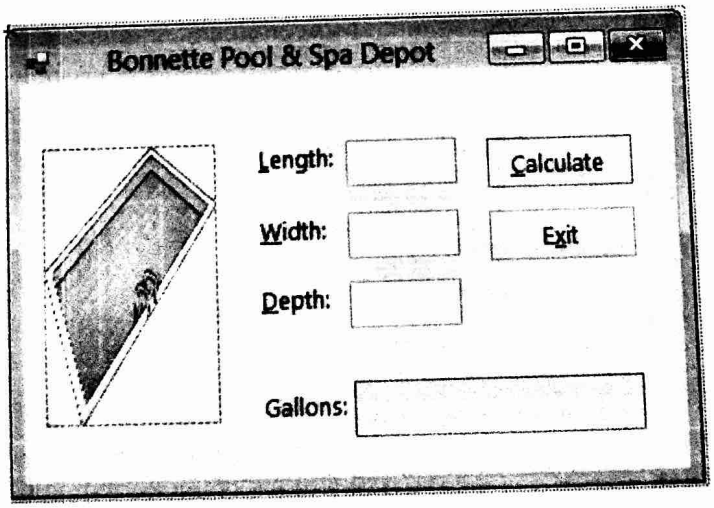

Figure 26-3 Bonnette Pool & Spa Depot interface

OpenClipArt.org/laobc

2. If necessary, permanently display the Solution Explorer window.
3. First, you will add a class file to the project. Click **PROJECT** on the menu bar and then click **Add Class**. The Add New Item dialog box opens with Class selected in the middle column. Change the filename in the Name box to **RectangularPool.vb** and then click the **Add** button. The computer adds the RectangularPool.vb file to the project. It also opens the file, which contains the Class statement, in a separate window.
4. Click the **blank line** below the Public Class clause. See Figure 26-4.

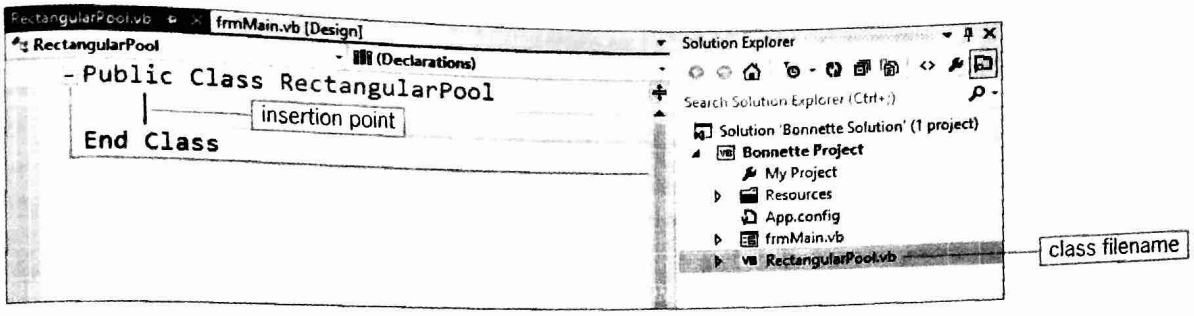

Figure 26-4 Class statement entered in the RectangularPool.vb window

In the context of OOP, a RectangularPool object has three attributes: a length, a width, and a depth. As mentioned earlier, the attributes are represented in the Class statement by variables and Property procedures. The variables appear first in the Class statement, followed by the Property procedures. In most cases, the variables are declared using the keyword `Private`. A class's Private variables can be used only by instructions within the class itself. The naming convention for a class's Private variables is to use the underscore as the first character in the name and then camel case for the remainder of the name. Following this naming convention, you will use the names `_dblLength`, `_dblWidth`, and `_dblDepth` for the Private variables in the RectangularPool class.

To include Private variables in the class:

1. Auto-hide the Solution Explorer window and then enter the following three declaration statements. Press **Enter** twice after typing the last declaration statement.

```
Private _dblLength As Double
Private _dblWidth As Double
Private _dblDepth As Double
```

Who Owns That Property?

When an application instantiates an object, only the Public members of the object's class are visible (available) to the application. Using OOP terminology, the Public members are "exposed" to the application, whereas the Private members are "hidden" from the application. In this case, the `_dblLength`, `_dblWidth`, and `_dblDepth` variables will be hidden from any application that contains an instance of the RectangularPool class. For an application to assign data to or retrieve data from a Private variable in a class, it must use a Public property. In other words, an application cannot directly refer to a Private variable in a class. Instead, it must refer to the variable indirectly, through the use of a Public property.

You create a Public property using a **Property procedure** whose procedure header begins with the keyword `Public`. A Public Property procedure creates a property that is visible to any application that contains an instance of the class. The basic syntax of a Public property

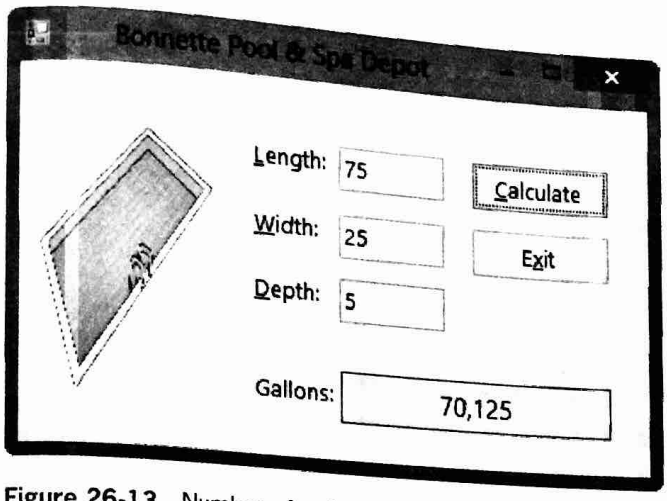

Figure 26-13 Number of gallons displayed in the interface
OpenClipArt.org/laobc

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

At this point, the advantage of creating a class and instantiating objects—in other words, the advantage of object-oriented programming—may not be apparent. After all, modifying the Bonnette Pool & Spa Depot application to include a class (rather than a structure) required many more lines of code. The real advantage of object-oriented programming is the ability to reuse a class in a different way or in a different application. In the next section, you will use the RectangularPool class in the Pool Supplies application.

Pool Supplies Application

Pool Supplies sells a water clarifier designed to combat a common problem in swimming pools: cloudy water. The company recommends one ounce of SoClear clarifier per 5000 gallons of water. The manager of Pool Supplies wants an application that calculates the number of gallons of water contained in a pool and the required number of ounces of clarifier. You can code this application using the RectangularPool class that you created earlier in this chapter. The RectangularPool.vb file is contained in the ClearlyVB2012\Chap26\Bonnette Solution\Bonnette Project folder.

To code the Pool Supplies application:

1. Use Windows to copy the RectangularPool.vb file from the ClearlyVB2012\Chap26\Bonnette Solution\Bonnette Project folder to the ClearlyVB2012\Chap26\Pool Supplies Solution\Pool Supplies Project folder.
2. Open the **Pool Supplies Solution** (**Pool Supplies Solution.sln**) file contained in the ClearlyVB2012\Chap26\Pool Supplies Solution folder. If necessary, open the designer window. See Figure 26-14.

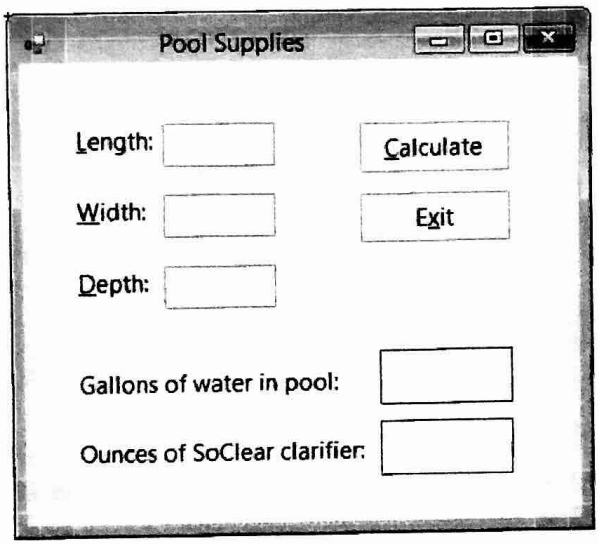

Figure 26-14 Pool Supplies interface

3. First, you will add the RectangularPool.vb file to the project. Click **PROJECT** on the menu bar and then click **Add Existing Item** to open the Add Existing Item dialog box. Click **RectangularPool.vb** in the Pool Supplies Project folder and then click the **Add** button. Temporarily display the Solution Explorer window to verify that it contains the RectangularPool.vb filename.

4. Open the Code Editor window and locate the btnCalc_Click procedure. First, you will instantiate a RectangularPool object. Click the **blank line** above the procedure's End Sub clause and then enter the following declaration statement:

```
Dim pool As New RectangularPool
```

5. Now you will declare variables to store the number of gallons of water and the number of ounces of clarifier. Enter the following two declaration statements. Press **Enter** twice after typing the second declaration statement.

```
Dim dblGallons As Double
Dim dblOunces As Double
```

6. Next, you will assign the input items to the RectangularPool object's properties. Enter the following three statements. Press **Enter** twice after typing the third statement.

```
Double.TryParse(txtLength.Text, pool.Length)
Double.TryParse(txtWidth.Text, pool.Width)
Double.TryParse(txtDepth.Text, pool.Depth)
```

7. Now you will use the RectangularPool object's GetGallons method to get the number of gallons of water. Enter the following assignment statement:

```
dblGallons = pool.GetGallons
```

8. You calculate the number of ounces of clarifier by dividing the number of gallons of water by 5000. Type the following assignment statement and then press **Enter** twice:

```
dblOunces = dblGallons / 5000
```

9. Finally, you will display the calculated amounts in the interface. Enter the following assignment statements:

```
lblGallons.Text = dblGallons.ToString("N0")
lblClarifier.Text = dblOunces.ToString("N1")
```

Figure 26-15 shows the code entered in the btnCalc_Click procedure.

```
Private Sub btnCalc_Click(sender As Object,
e As EventArgs) Handles btnCalc.Click
    ' calculates and displays the number of gallons
    ' in a pool and the number of ounces of
    ' clarifier needed

    Dim pool As New RectangularPool
    Dim dblGallons As Double
    Dim dblOunces As Double

    Double.TryParse(txtLength.Text, pool.Length)
    Double.TryParse(txtWidth.Text, pool.Width)
    Double.TryParse(txtDepth.Text, pool.Depth)

    dblGallons = pool.GetGallons
    dblOunces = dblGallons / 5000

    lblGallons.Text = dblGallons.ToString("N0")
    lblClarifier.Text = dblOunces.ToString("N1")

End Sub
```

Figure 26-15 btnCalc_Click procedure in the Pool Supplies application

To test the Pool Supplies application's code:

1. Save the solution and then start the application. Type **75**, **25**, and **5** in the Length, Width, and Depth boxes, respectively. Click the **Calculate** button. The interface shows that the pool contains 70,125 gallons of water, which requires 14.0 ounces of clarifier. See Figure 26-16.

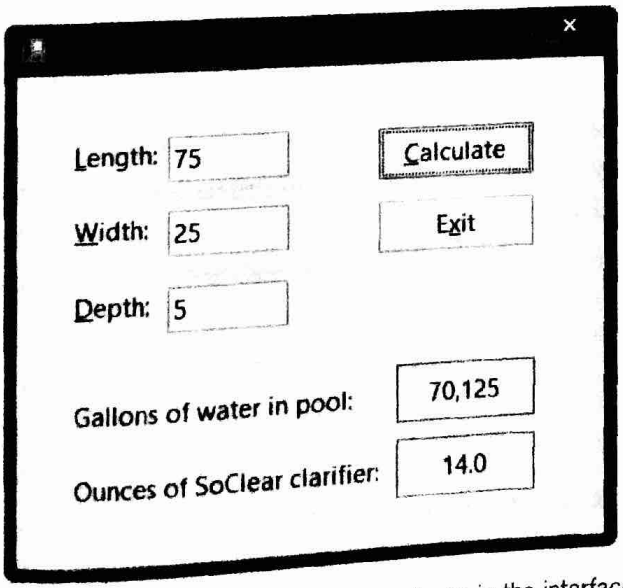

Figure 26-16 Calculated amounts shown in the interface

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

INTERMEDIATE

5. Open the Grade Solution (Grade Solution.sln) file contained in the ClearlyVB2012\Chap26\Grade Solution folder.
   a. Add a new class file named CourseGrade.vb to the project. The CourseGrade class should have three attributes: the scores for three tests. Each test score will be an integer. The class should also have two behaviors: the default constructor and a method that determines and returns the letter grade. The letter grade is based on the total score, as indicated in Figure 26-17. Code the CourseGrade class. Save the solution and then close the CourseGrade.vb window.
   b. Open the form's Code Editor window. Code the btnDisplay_Click procedure. Each test score should be from 0 through 100 only. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

Total score	Grade
270 – 300	A
240 – 269	B
210 – 239	C
180 – 209	D
Less than 180	F

Figure 26-17 Information for Exercise 5

INTERMEDIATE

6. In this exercise, you modify the RectangularPool class from the chapter. Use Windows to make a copy of the Bonnette Solution folder. Save the copy in the ClearlyVB2012\Chap26 folder. Rename the copy Bonnette Solution–INTERMEDIATE. Open the Bonnette Solution (Bonnette Solution.sln) file contained in the Bonnette Solution–INTERMEDIATE folder. Open the designer window. Open the RectangularPool.vb file. Change the GetVolume function to a Sub procedure. Save the solution and then start and test the application. Close the RectangularPool.vb window and then close the solution.

INTERMEDIATE

7. Open the Circle Area Solution (Circle Area Solution.sln) file contained in the ClearlyVB2012\Chap26\Circle Area Solution folder.
   a. Add a new class file named Circle.vb to the project. The Circle class should contain one attribute: the circle's radius. It should also contain a default constructor and a method that calculates and returns the circle's area. Use the following formula to calculate the area: 3.141592 * $radius^2$. Code the Circle class. Save the solution and then close the Circle.vb window.
   b. Open the form's Code Editor window. The btnCalc_Click procedure should display the circle's area, using the radius entered by the user. Display the area with two decimal places. Code the procedure.
   c. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

INTERMEDIATE

8. Open the Carpet Haven Solution (Carpet Haven Solution.sln) file contained in the ClearlyVB2012\Chap26\Carpet Haven Solution folder (The image in the picture box was downloaded from the Open Clip Art Library at *http://openclipart.org*.)
   a. Add a new class file named Rectangle.vb to the project. The Rectangle class should contain two attributes: the rectangle's length and its width. It should also contain a default constructor and a method that calculates and returns the rectangle's area. Code the Rectangle class. Save the solution and then close the Rectangle.vb window.

b. Open the form's Code Editor window. The btnCalc_Click procedure should display the required number of square yards needed to carpet a room and the carpet cost. Code the procedure.
c. Save the solution and then start and test the application. (Hint: If the room's length and width measurements are 9 feet and 8.5 feet, respectively, the number of square yards is 8.5. If the price per square yard is $9.50, the carpet cost is $80.75.) Close the Code Editor window and then close the solution.

ADVANCED

9. Shelly Jones, the manager of Pennington Book Store, wants an application that calculates and displays the total amount a customer owes. The interface for this application is shown in Figure 26-18. A customer can purchase one or more books at either the same price or different prices. The application should keep a running total of the amount the customer owes, and display the total in the Total due box. For example, a customer might purchase two books at $6 and three books at $10. To calculate the total due, Shelly will need to enter 2 in the Quantity box and 6 in the Price box, and then click the Add to Sale button. The Total due box should display $12.00. To complete the order, Shelly will need to enter 3 in the Quantity box and 10 in the Price box, and then click the Add to Sale button. The Total due box should display $42.00. Before calculating the next customer's order, Shelly will need to click the New Order button.

a. Create a Visual Basic Windows application. Use the following names for the solution and project, respectively: Pennington Solution and Pennington Project. Save the application in the ClearlyVB2012\Chap26 folder. Change the name of the form file on your disk to frmMain.vb. If necessary, change the form's name to frmMain.
b. Create the interface shown in Figure 26-18.
c. Add a class file named BookSale.vb to the project. The BookSale class should have three attributes: a quantity, a price, and a total due. It also should have two behaviors: the default constructor and a method that keeps a running total of the amount the customer owes. Code the BookSale class. Save the solution and then close the BookSale.vb window.
d. Open the form's Code Editor window and code the application.
e. Save the solution and then start and test the application. Close the Code Editor window and then close the solution.

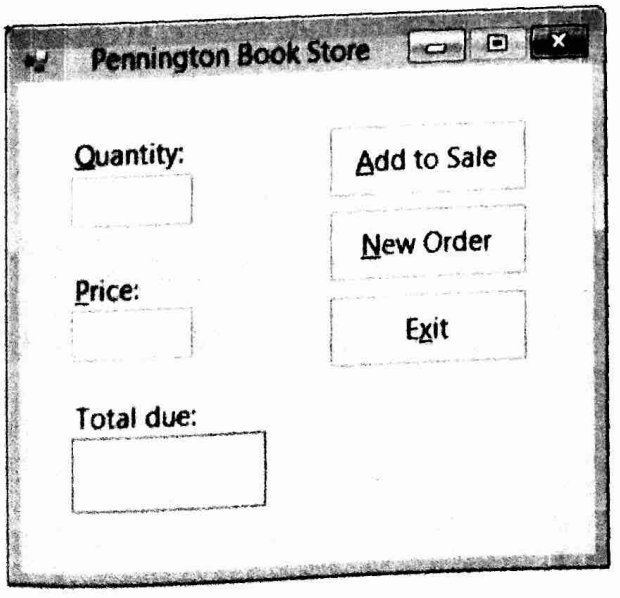

Figure 26-18 Interface for Exercise 9