11

Object-Oriented Programming

11.1 Classes and Objects

noun A word used to denote or name a person, place, thing, quality, or act.

verb That part of speech that expresses existence, action, or occurrence.

adjective Any of a class of words used to modify a noun or other substantive by limiting, qualifying, or specifying.

The American Heritage Dictionary of the English Language

"A good rule of thumb for object-oriented programming is that classes are the nouns in your analysis of the problem. The methods in your object correspond to verbs that the noun does. The properties are the adjectives that describe the noun."

Gary Cornell & David Jezak

Practical experience in the financial, scientific, engineering, and software design industries has revealed some difficulties with traditional program design methodologies. As programs grow in size and become more complex, and as the number of programmers working on the same project increases, the number of dependencies and interrelationships throughout the code increases exponentially. A small change made by one programmer in one place may have many effects, both intended and unintended, in many other places. The effects of this change may ripple throughout the entire program, requiring the rewriting of a great deal of code along the way.

A partial solution to this problem is "data hiding" where, within a unit, as much implementation detail as possible is hidden. Data hiding is an important principle underlying object-oriented programming. An object is an encapsulation of data and procedures that act on the data. A programmer using an object is concerned only with the tasks that the object can perform and the parameters used by these tasks. The details of the data structures and procedures are hidden within the object.

VideoNote
Classes and objects

Two types of objects will be of concern to us: **control objects** and **code objects**. Examples of control objects are text boxes, list boxes, buttons, and all the other controls that can be created from the Toolbox. So far, most of our programs have contained a single class block beginning with a line such as "Public Class frmName" and ending with the line "End Class." A code object is a specific instance of a user-defined type, called a **class**, which is defined similarly to a structure, but in a separate class block of the form

```
Class ClassName
  statements
End Class
```

Each class block is delineated in the Code Editor by an elongated left bracket appearing to the left of the block. Both control objects and class objects have properties, methods, and events. The main differences are that control objects are predefined and have physical manifestations, whereas the programmer must create the class blocks for code objects. In this section, when we use the word "object" without a qualifier, we mean "code object."

Whenever you double-click on the TextBox icon in the Toolbox, a new text box is created. Although each text box is a separate entity, they all have the same properties, methods, and events. Each text box is said to be an **instance** of the class TextBox. In some sense, the TextBox icon in the Toolbox is a template or blueprint for creating text boxes. When you look at the Properties window for a text box, the drop-down list box at the top of the window reads something like "TextBox1 System.Windows.Forms.TextBox." TextBox1 is the name of the control object and it is said to be an instance of the class "TextBox." You can't set properties or invoke methods of the TextBox class; you can only set properties or invoke methods of the specific text boxes that are instances of the class. The analogy is often made between a class and a cookie

cutter. The cookie cutter is used to create cookies that you can eat, but you can't eat the cookie cutter.

Object-oriented programs are populated with objects that hold data, have properties, respond to methods, and raise events. (The generation of events will be discussed in the next section.) Six examples of objects are as follows:

1. In a professor's program to assign and display semester grades, a student object might hold a single student's name, social security number, midterm grade, and final exam grade. A CalcSemGrade method might calculate the student's semester grade. Events might be raised when improper data are passed to the object.
2. In a payroll program, an employee object might hold an employee's name, hourly wage, and hours worked. A CalculatePay method would tell the object to calculate the wages for the current pay period.
3. In a checking account program, a check register object might have methods that record and total the checks written during a certain month, a deposit slip object might record and total the deposits made during a certain month, and an account object might keep a running total of the balance in the account. The account object would raise an event to alert the bank when the balance got too low.
4. In a bookstore inventory program, a textbook object might hold the name, author, quantity in stock, and wholesale price of an individual textbook. A CalculateRetailPrice method might instruct the textbook object to calculate the selling price of the textbook. An event could be raised when the book went out of stock.
5. In a game program, an airplane object might hold the location of an airplane. At any time, the program could tell the object to display the airplane at its current location or to drop a bomb. An event could be raised each time a bomb was released so that the program could determine if anything was hit.
6. In a card game program, a card object might hold the denomination and suit of a specific card. An IdentifyCard method might return a string such as "Ace of Spades." A deck-of-cards object might consist of an array of card objects and a ShuffleDeck method that thoroughly shuffled the deck. A Shuffling event might indicate the progress of the shuffle.

An important object-oriented term is **class**. A class is a template from which objects are created. The class specifies the properties and methods that will be common to all objects that are instances of that class. Classes are formulated in class blocks. An object, which is an instance of a class, can be created in a program with a pair of statements of the form

```
Dim objectName As ClassName
objectName = New ClassName(arg1, arg2, ...)
```

The first of these two lines of code declares what type of object the variable will refer to. The actual object does not exist until it is created with the New keyword, as done in the second line. This is known as creating an **instance** of an object and is where an object is actually created from its class. After this second line of code executes, the object is then ready for use. The first line can appear either in the Declarations section of a program (to declare a class-level variable) or inside a procedure (to declare a local variable). The instantiation line can appear only in a procedure; however, any object variable can be instantiated when declared (as either class level or local) by using the single line

```
Dim objectName As New ClassName(arg1, arg2, ...)
```

In a program, properties, methods, and events of the object are accessed with statements of the form shown in the following table:

TASK	STATEMENT
Assign a value to a property	*objectName.propertyName = value*
Assign the value of a property to a variable	*varName = objectName.propertyName*
Carry out a method	*objectName.methodName(arg1, ...)*
Raise an event	**RaiseEvent** *eventName*

The program in Example 1 uses a class named Student to calculate and display a student's semester grade. The information stored by an object of the type Student consists of a student's name, social security number, and grades on two exams (midterm and final). This data is stored in variables declared with the statements

```
Private m_name As String        'Name
Private m_ssn As String         'Social security number
Private m_midterm As Double     'Numerical grade on midterm exam
Private m_final As Double       'Numerical grade on final exam
```

The word Private guarantees that the variables cannot be accessed directly from outside the object. In object-oriented programming terminology, these variables are called **member variables** (or **instance variables**). We will follow the common convention of beginning the name of each member variable with the prefix "m_". Each of these variables is used to hold the value of a property. However, instead of being accessed directly, each member variable is accessed indirectly with a **property block**. For instance, the following property block consists of a Get property procedure to retrieve (or *read*) the value of the Name property and a Set property procedure to assign (or *write*) the value of the Name property:

```
Public Property Name() As String
  Get
    Return m_name
  End Get
  Set(ByVal value As String)
    m_name = value
  End Set
End Property
```

In a property block, additional code can be added after the Get and Set statements to validate the data before they are returned or stored. The word Public allows the property to be accessed from outside the code for the Student class block. For instance, the Name property can be accessed by code in the form's class block. On the other hand, since the member variables were declared as Private, they cannot be accessed directly from code in the form's block. They can be accessed only through Property procedures that allow values to be checked and perhaps modified. Also, a Property procedure is able to take other steps necessitated by a change in the value of a member variable.

A property block needn't contain both Get and Set property procedures. For instance, the block

```
Public WriteOnly Property Midterm() As Double
  Set(ByVal value As double)
    m_midterm = value
  End Set
End Property
```

specifies the Midterm property as "write only." This property could be specified to be "read only" with the block

```
Public ReadOnly Property Midterm() As Double
  Get
    Return m_midterm
  End Get
End Property
```

Methods are constructed with Sub or Function procedures. A Function procedure is used when the method returns a value; otherwise a Sub procedure will suffice. For instance, the method CalcSemGrade, which is used to calculate a student's semester grade, is created as follows:

```
Function CalcSemGrade() As String
  Dim grade As Double
  grade = (m_midterm + m_final) / 2
  grade = Math.Round(grade) 'Round the grade.
  Select Case grade
    Case Is >= 90
      Return "A"
    Case Is >= 80
      Return "B"
    Case Is >= 70
      Return "C"
    Case Is >= 60
      Return "D"
    Case Else
      Return "F"
  End Select
End Function
```

An object of the type Student is declared in the form's code with a pair of statements such as

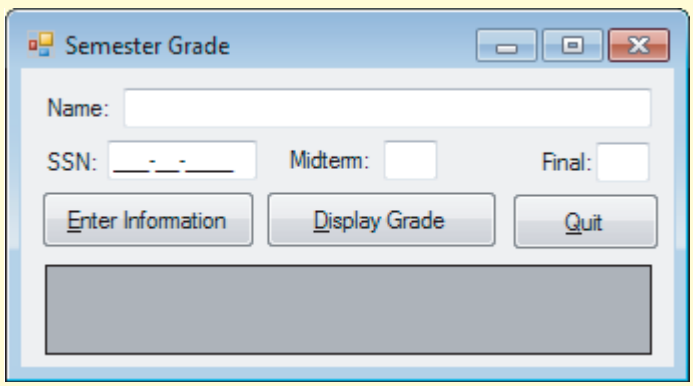
```
Dim pupil As Student     'Declare pupil as an object of type Student
pupil = New Student()    'Create an instance of type Student
```

After these two statements are executed, properties and methods can be utilized with statements such as

```
pupil.Name = "Adams, Al"                 'Assign a value to m_name
txtBox.text = pupil.Name                 'Display the student's name
lstBox.Items.Add(pupil.CalcSemGrade)     'Display semester grade
```

The first statement calls the Set property procedure for the Name property, the second statement calls the Get property procedure for the Name property, and the third statement calls the method CalcSemGrade.

Example 1 The following program uses the class Student to calculate and display a student's semester grade. The structure Person in frmGrades is used by the btnDisplay_Click procedure to place information into the DataGridView control.



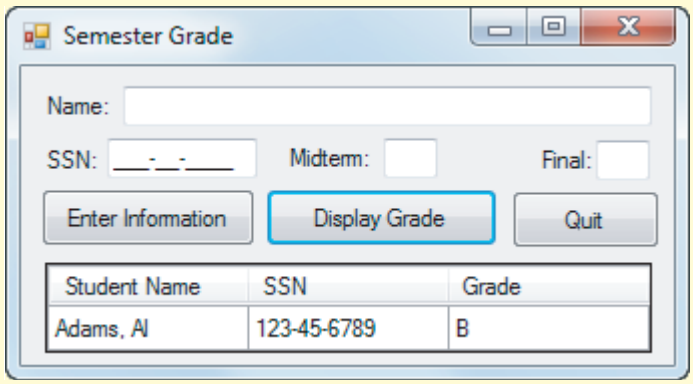
OBJECT	PROPERTY	SETTING
frmGrades	Text	Semester Grade
lblName	Text	Name:
txtName		
lblSSN	Text	SSN:
mtbSSN	Mask	000-00-0000
lblMidterm	Text	Midterm:
txtMidterm		
lblFinal	Text	Final:
txtFinal		
btnEnter	Text	&Enter Information
btnDisplay	Text	&Display Grade
btnQuit	Text	&Quit
dgvGrades	RowHeadersVisible	False

```
Public Class frmGrades
  Dim pupil As Student  'pupil is an object of class Student

  Structure Person  'for use in btnDisplay_Click
    Dim name As String
    Dim socSecNum As String
    Dim semGrade As String
  End Structure

  Private Sub btnEnter_Click(...) Handles btnEnter.Click
    pupil = New Student()    'Create an instance of Student.
    'Read the values stored in the text boxes.
    pupil.Name = txtName.Text
    pupil.SocSecNum = mtbSSN.Text
    pupil.Midterm = CDbl(txtMidterm.Text)
    pupil.Final = CDbl(txtFinal.Text)
    'Clear text boxes and list box
    txtName.Clear()
    mtbSSN.Clear()
    txtMidterm.Clear()
    txtFinal.Clear()
    'Notify user that grades for the student have been recorded.
    MessageBox.Show("Student Recorded.")
  End Sub

  Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    Dim persons(0) As Person
    persons(0).name = pupil.Name
    persons(0).socSecNum = pupil.SocSecNum
    persons(0).semGrade = pupil.CalcSemGrade
    Dim query = From someone In persons
                Select someone.name, someone.socSecNum, someone.semGrade
    dgvGrades.DataSource = query.ToList
    dgvGrades.CurrentCell = Nothing
    dgvGrades.Columns("name").HeaderText = "Student Name"
    dgvGrades.Columns("socSecNum").HeaderText = "SSN"
    dgvGrades.Columns("semGrade").HeaderText = "Grade"
  End Sub
```

```
  Private Sub btnQuit_Click(...) Handles btnQuit.Click
    Me.Close()
  End Sub
End Class      'frmGrades

Class Student
  Private m_name As String        'Name
  Private m_ssn As String         'Social security number
  Private m_midterm As Double     'Numerical grade on midterm exam
  Private m_final As Double       'Numerical grade on final exam

  Public Property Name() As String
    Get
      Return m_name
    End Get
    Set(ByVal value As String)
      m_name = value
    End Set
  End Property

  Public Property SocSecNum() As String
    Get
      Return m_ssn
    End Get
    Set(ByVal value As String)
      m_ssn = value
    End Set
  End Property

  Public WriteOnly Property Midterm() As Double
    Set(ByVal value As Double)
      m_midterm = value
    End Set
  End Property

  Public WriteOnly Property Final() As Double
    Set(ByVal value As Double)
      m_final = value
    End Set
  End Property

  Function CalcSemGrade() As String
    Dim grade As Double
    grade = (m_midterm + m_final) / 2
    grade = Math.Round(grade)  'Round the grade.
    Select Case grade
      Case Is >= 90
        Return "A"
      Case Is >= 80
        Return "B"
      Case Is >= 70
        Return "C"
      Case Is >= 60
        Return "D"
```

```
        Case Else
          Return "F"
      End Select
    End Function
End Class       'Student
```

[Run, enter the data for a student (such as "Adams, Al", "123-45-6789", "82", "87"), click on the *Enter Information* button to send the data to the object, and click on the *Display Grade* button to display the student's name, social security number, and semester grade.]



In summary, the following six steps are used to create a class:

1. Identify a *thing* in your program that is to become an object.
2. Determine the properties and methods that you would like the object to have. (As a rule of thumb, properties should access data, and methods should perform operations.)
3. A class will serve as a template for the object. The code for the class is placed in a class block of the form

```
Class ClassName
  statements
End Class
```

4. For each of the properties in Step 2, declare a private member variable with a statement of the form

```
Private variableName As DataType
```

   Member variables can be preceded with the keyword Public, which allows direct access to the member variables from the code in the form. However, this is considered poor programming practice. By using Set property procedures to update the data, we can enforce constraints and carry out validation.

5. For each of the member variables in Step 4, create a Property block with Get and/or Set procedures to retrieve and assign values of the variable. The general forms of the procedures are

```
Public Property PropertyName() As DataType
  Get
    (Possibly additional code)
    Return variableName
  End Get
  Set(ByVal value As DataType)
    (Possibly additional code)
```

```
    variableName = value
  End Set
End Property
```

In the Get or Set code, additional code can be added to prevent the object from storing or returning invalid or corrupted data. For example, an If block could be added to only allow valid social security numbers, alerting the user in the event of an invalid number.

6. For each method in Step 2, create a Sub procedure or Function procedure to carry out the task.

Example 2 The following modification of the program in Example 1 calculates semester grades for students who have registered on a "Pass/Fail" basis. We create a new class, named PFStudent, with the same member variables and property procedures as the class Student. The only change needed in the class block occurs in the CalcSemGrade method. The new code for this method is

```
Function CalcSemGrade() As String
  Dim grade As Double
  grade = (m_midterm + m_final) / 2
  grade = Math.Round(grade)  'Round the grade.
  If grade >= 60 Then
    Return "Pass"
  Else
    Return "Fail"
  End If
End Function
```

The only change needed in the form's code is to replace the two occurrences of *Student* with *PFStudent*. When the program is run with the same input as in Example 1, the output will be

```
Adams, Al    123-45-6789  Pass
```

Object Constructors

Each class has a special method called a **constructor** that is always invoked when an object is instantiated. The constructor takes zero or more arguments, and the code inside the procedure block performs any tasks needed for initializing an object. It is often used to set default values for member variables and to create other objects associated with this object. The first line of the constructor for a class has the form

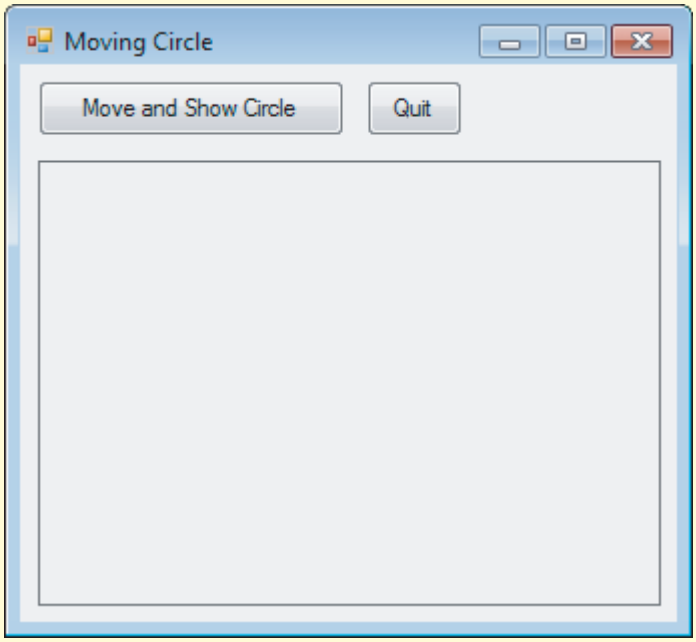
```
Public Sub New(ByVal par1 As DataType1, ByVal par2 As DataType2, ...)
```

The graphical program in Example 3 illustrates the use of a constructor to specify the size and initial placement of a circle. This task involves pixels. To get a feel for how big a pixel is, the initial size of the form when you create a new project is 300 pixels by 300 pixels. Section 9.4 explains how graphics are created inside a picture box with the Graphics object gr = picBox.CreateGraphics. In Example 3, the statement

```
gr.DrawEllipse(Pens.Black, Xcoord, Ycoord, Diameter, Diameter)
```

draws a circle inside a picture box, where Xcoord and Ycoord are the distances (in pixels) of the circle from the left side and top of the picture box.

Example 3 The following program contains a Circle object. The object keeps track of the location and diameter of the circle. (The location is specified by two numbers, called the coordinates, giving the distance from the left side and top of the picture box. Distances and the diameter are measured in pixels.) A Show method displays the circle, and a Move method adds 20 pixels to each coordinate of the circle. Initially, the (unseen) circle is located at the upper-left corner of the picture box and has a diameter of 40. The form has a button captioned *Move and Show Circle* that invokes both methods. Notice that the Xcoord, Ycoord, and Diameter properties, rather than the member variables, appear in the methods.

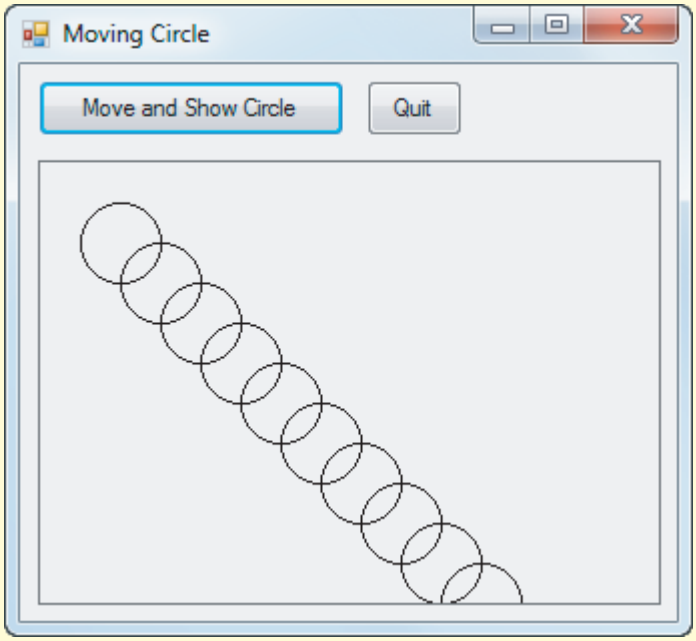


OBJECT	PROPERTY	SETTING
frmCircle	Text	Moving Circle
btnMove	Text	Move and Show Circle
btnQuit	Text	Quit
picCircle		

```
Public Class frmCircle
  Dim round As New Circle()

  Private Sub btnMove_Click(...) Handles btnMove.Click
    round.Move(20)
    round.Show(picCircle.CreateGraphics)
  End Sub

  Private Sub btnQuit_Click(...) Handles btnQuit.Click
    Me.Close()
  End Sub
End Class      'frmCircle

Class Circle
  Private m_x As Integer  'Dist from left side of picture box to circle
  Private m_y As Integer  'Distance from top of picture box to the circle
  Private m_d As Integer  'Diameter of circle

  Public Sub New()
    'Set the initial location of the circle to the upper-left
    'corner of the picture box, and set its diameter to 40.
    Xcoord = 0
    Ycoord = 0
    Diameter = 40
  End Sub
```

```
  Public Property Xcoord() As Integer
    Get
      Return m_x
    End Get
    Set(ByVal value As Integer)
      m_x = value
    End Set
  End Property

  Public Property Ycoord() As Integer
    Get
      Return m_y
    End Get
    Set(ByVal value As Integer)
      m_y = value
    End Set
  End Property

  Public Property Diameter() As Integer
    Get
      Return m_d
    End Get
    Set(ByVal value As Integer)
      m_d = value
    End Set
  End Property

  Sub Show(ByVal gr As Graphics)
    'Draw a circle with the given graphics context.
    gr.DrawEllipse(Pens.Black, Xcoord, Ycoord, Diameter, Diameter)
  End Sub

  Sub Move(ByVal distance As Integer)
    Xcoord += distance
    Ycoord += distance
  End Sub
End Class      'Circle
```

[Run, and click on the *Move* button ten times.]



When the line that instantiates an object contains arguments, the values of these arguments are passed to the object's New procedure. For instance, in Example 3, the first line typed in the form's code can be changed to

```
Dim round As New Circle(0, 0, 40)
```

and the New procedure for the Circle class can be changed to

```
Public Sub New(ByVal x As Integer, ByVal y As Integer,
               ByVal d As Integer)
  'Set the initial location of the circle to x pixels from
  'the left side and y pixels from the top of the picture box.
  'Set the diameter to d pixels.
  Xcoord = x
  Ycoord = y
  Diameter = d
End Sub
```

Auto-Implemented Properties

Some property blocks are very clear-cut in that they do not contain ReadOnly or WriteOnly keywords and have no additional code in their Get or Set blocks. Such property blocks can use the new-to-VB2010 **auto-implemented properties** feature. This feature allows you to reduce a clear-cut property block to just its header, and to omit declaring a member variable for the property. Visual Basic automatically creates hidden Get and Set procedures and a hidden member variable. The name of the member variable is the property name preceded by an underscore character. For example, if you declare an auto-implemented property named SocSecNum, the member variable will be named _SocSecNum. We will use auto-implemented properties in the remainder of this chapter.

Practice Problems 11.1

1. Which of the following analogies is out of place?

   (a) class : object
   (b) sewing pattern : garment
   (c) blueprint : house
   (d) programmer : program
   (e) cookie cutter : cookie

2. In Example 1, suppose that the first five lines of the event procedure btnEnter_Click are replaced with

   ```
   Private Sub btnEnter_Click(...) Handles btnEnter.Click
     Dim ssn As String = "123-45-6789"    'Social security Number
     'Create an instance of Student.
     pupil = New Student(ssn)
     pupil.Name = txtName.Text
   ```

   Create a New procedure and revise the SocSecNum property block for the Student class to be consistent with the last line in the preceding code.

EXERCISES 11.1

Exercises 1 through 14 refer to the class Student from Example 1. When applicable, assume that *pupil* is an instance of the class.

1. What will be the effect if the *Midterm* property block is changed to the following?

```
Public WriteOnly Property Midterm() As Double
  Set(ByVal value As Double)
    Select Case value
      Case Is < 0
        m_midterm = 0
      Case Is > 100
        m_midterm = 100
      Case Else
        m_midterm = value
    End Select
  End Set
End Property
```

2. What will be the effect if the *Midterm* property block is changed to the following?

```
Public WriteOnly Property Midterm() As Double
  Set(ByVal value As Double)
    m_midterm = value + 10
  End Set
End Property
```

3. Modify the class block for *Student* so that the following statement will display the student's midterm grade:

```
MessageBox.Show(CStr(pupil.Midterm))
```

4. Modify the class block for *Student* so that the student's semester average can be displayed with a statement of the form

```
MessageBox.Show(CStr(pupil.Average))
```

5. In the class block for Student, why can't the third line of the CalcSemGrade method be written as follows?

```
grade = (Midterm + Final) / 2
```

6. Write code for the class block that sets the two grades to 10 whenever an instance of the class is created.

7. What is the effect of adding the following code to the class block?

```
Public Sub New()
  SocSecNum = "999-99-9999"
End Sub
```

In Exercises 8 through 14, determine the errors in the given form code.

8.
```
Dim scholar As Student

Private Sub btnGo_Click(...) Handles btnGo.Click
  Dim firstName as String
  scholar.Name = "Warren"
  firstName = scholar.Name
End Sub
```

9.
```
Dim scholar As Student

Private Sub btnGo_Click(...) Handles btnGo.Click
  Dim nom as String
  scholar = Student()
```

```
  scholar.Name = "Peace, Warren"
  nom = scholar.Name
End Sub
```

10.
```
Dim scholar As Student

Private Sub btnGo_Click(...) Handles btnGo.Click
  Dim nom as String
  scholar = New Student()
  m_name = "Peace, Warren"
  nom = scholar.Name
End Sub
```

11.
```
Dim scholar As Student

Private Sub btnGo_Click(...) Handles btnGo.Click
  Dim nom As String
  scholar = New Student()
  scholar.Name = "Peace, Warren"
  nom = m_name
End Sub
```

12.
```
Dim scholar As Student

Private Sub btnGo_Click(...) Handles btnGo.Click
  Dim grade As String
  scholar = New Student()
  scholar.CalcSemGrade = "A"
  grade = scholar.CalcSemGrade()
End Sub
```

13.
```
Dim pupil, scholar As Student

Private Sub btnGo_Click(...) Handles btnGo.Click
  scholar = New Student()
  pupil = New Student()
  scholar.Midterm = 89
  pupil.Midterm = scholar.Midterm
  lstGrades.Items.Add(pupil.Midterm)
End Sub
```

14.
```
Dim scholar As Student
scholar = New Student()

Private Sub btnGo_Click(...) Handles btnGo.Click
  scholar.Name = "Transmission, Manuel"
End Sub
```

15. In the following program, determine the output displayed in the list box when the button is clicked on:

```
Public Class frmCountry
  Dim nation As New Country("Canada", "Ottawa")

  Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    nation.Population = 31
    lstBox.Items.Add("Country: " & nation.Name)
    lstBox.Items.Add("Capital: " & nation.Capital)
```

```
    lstBox.Items.Add("Pop: " & nation.Population & " million")
  End Sub
End Class      'frmCountry

Class Country
  Private m_name As String
  Private m_capital As String
  Private m_population As Double

  Sub New(ByVal name As String, ByVal capital As String)
    m_name = name
    m_capital = capital
  End Sub

  Public ReadOnly Property Name() As String
    Get
      Return m_name
    End Get
  End Property

  Public ReadOnly Property Capital() As String
    Get
      Return m_capital
    End Get
  End Property

  Public Property Population() As Double
    Get
      Return m_population
    End Get
    Set(ByVal value As Double)
      m_population = value
    End Set
  End Property
End Class      'Country
```

Exercises 16 through 18 refer to the class *Circle*.

16. Enhance the program in Example 3 so that the Get and Set property procedures of the Xcoord and Ycoord properties are used by the form code.

17. Modify Example 3 so that the circle originally has its location at the lower-right corner of the picture box and moves diagonally upward each time *btnMove* is clicked on.

18. Modify the form code of Example 3 so that each time *btnMove* is clicked on, the distance moved (in pixels) is a randomly selected number from 0 to 40.

19. Write the code for a class called Square. The class should have three properties—Length, Perimeter, and Area—with their obvious meanings. When a value is assigned to one of the properties, the values of the other two should be recalculated automatically. When the following form code is executed, the numbers 5 and 20 should be displayed in the text boxes:

```
Dim poly As Square

Private Sub btnGo_Click(...) Handles btnGo.Click
  poly = New Square()
  poly.Area = 25
```

```
    txtLength.Text = CStr(poly.Length)
    txtPerimeter.Text = CStr(poly.Perimeter)
End Sub
```

20. Modify the class Square in the previous exercise so that all squares will have lengths between 1 and 10. For instance, the statement `poly.Area = 0.5` should result in a square of length 1, and the statement `poly.Area = 200` should result in a square with each side having length 10.

21. Write the code for a class called PairOfDice. A Random object should be used to obtain the value for each die. When the following form code is executed, three numbers (such as 3, 4, and 7) should be displayed in the text boxes.

```
Dim cubes As PairOfDice

Private Sub btnGo_Click(...) Handles btnGo.Click
    cubes = New PairOfDice()
    cubes.Roll()
    txtOne.Text = CStr(cubes.Die1)
    txtTwo.Text = CStr(cubes.Die2)
    txtSum.Text = CStr(cubes.SumOfFaces)
End Sub
```

22. Write a program to roll a pair of dice 1000 times, and display the number of times that the sum of the two faces is 7. The program should use an instance of the class PairOfDice discussed in the previous exercise.

23. Write the code for a class called College. The class should have properties Name, NumStudents, and NumFaculty. The method SFRatio should compute the student–faculty ratio. When the following form code is executed, the number 12.4 should be displayed in the text box:

```
Dim school As College

Private Sub btnGo_Click(...) Handles btnGo.Click
    school = New College()
    school.Name = "University of Maryland, College Park"
    school.NumStudents = 36041
    school.NumFaculty = 2896
    txtBox.Text = FormatNumber(school.SFRatio, 1)
End Sub
```

24. Write a program that calculates an employee's pay for a week based on the hourly wage and the number of hours worked. All computations should be performed by an instance of the class Wages.

25. Write a program to implement the cash register in Fig. 11.1. The program should have a class called CashRegister that keeps track of the balance and allows deposits and withdrawals. The class should not permit a negative balance.

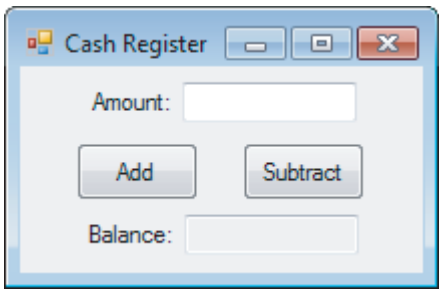


FIGURE 11.1 Form for Exercise 25.

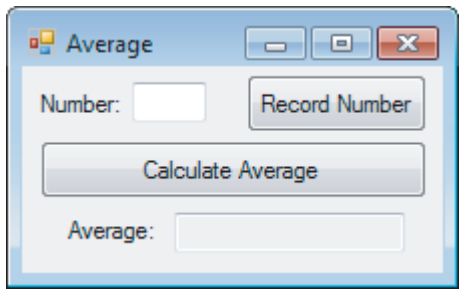


FIGURE 11.2 Form for Exercise 26.

26. Write a program that calculates the average of up to 50 numbers input by the user and stored in an array. See Fig. 11.2. The program should use a class named Statistics and have an AddNumber method that stores numbers into an array one at a time. The class should have a Count property that keeps track of the number of numbers stored and a method called Average that returns the average of the numbers.

27. Write a program that calculates an employee's FICA tax, with all computations performed by an instance of a class FICA. The FICA tax has two components: the social security benefits tax, which in 2009 is 6.2% of the first $106,800 of earnings for the year, and the Medicare tax, which is 1.45% of earnings.

28. Write a program that adds two fractions and displays their sum in reduced form. The program should use a Fraction class that stores the numerator and denominator of a fraction and has a Reduce method that divides each of the numerator and denominator by their greatest common divisor. Exercise 29 of Section 6.1 contains an algorithm for calculating the greatest common divisor of two numbers.

Solutions to Practice Problems 11.1

1. (d) A programmer is not a template for creating a program.
2.
```
Public Sub New(ByVal ssn As String)
    'Assign the value of ssn to the member variable m_ssn.
    m_ssn = ssn
End Sub

Public ReadOnly Property SocSecNum() As String
    Get
        Return m_ssn
    End Get
End Property
```

Note: Since a student's social security number never changes, there is no need to have a Set property procedure for SocSecNum.

11.2 Working with Objects

"An object without an event is like a telephone without a ringer."

Anonymous

Arrays of Objects

VideoNote
Arrays of objects

The elements of an array can have any data type—including a class. The program in Example 1 uses an array of type Student.

Example 1 In the following program, which uses the same form design as Example 1 of the previous section, the user enters four pieces of data about a student into text boxes. When the *Enter Information* button is clicked on, the data are used to create and initialize an appropriate object and the object is added to an array. When the *Display Grades* button is clicked on, the name, social security number, and semester grade for each student in the array are displayed in the grid.

```
Public Class frmGrades
  Dim students(50) As Student
  Dim lastStudentAdded As Integer = -1    'Position in array of student
  '                                        most recently added
```