TABLE 11.2 2009 Federal income tax withheld for a married person paid weekly.

Adjusted Weekly Income	Income Tax Withheld
$0 to $303	$0
Over $303 to $ 470	10% of amount over $303
Over $470 to $1,455	$16.70 + 15% of amount over $470
Over $1,455 to $2,272	$164.45 + 25% of amount over $1,455
Over $2,272 to $4,165	$368.70 + 28% of amount over $2,272
Over $4,165 to $7,321	$898.74 + 33% of amount over $4,165
Over $7,321	$1,940.22 + 35% of amount over $7,321

Solutions to Practice Problems 11.2

1.
```
Public Property SocSecNum() As String
  Get
    Return m_ssn
  End Get
  Set(ByVal value As String)
    If value.Length = 11 Then
      m_ssn = value
    Else
      RaiseEvent ImproperSSN(value.Length, m_name)
    End If
  End Set
End Property
```

2.
```
Public Event ImproperSSN(ByVal length As Integer,
                         ByVal studentName As String)
```

3.
```
Private Sub pupil_ImproperSSN(ByVal length As Integer,
              ByVal studentName As string) Handles pupil.ImproperSSN
  MessageBox.Show("The social security number entered for " &
        studentName & " consisted of " & length &
        " characters. Reenter the data for " & studentName & ".")
End Sub
```

4. The statement

```
Dim pupil As Student
```

must be changed to

```
Dim WithEvents pupil As Student
```

VideoNote
Inheritance

11.3 Inheritance

The three relationships between classes are "use," "containment," and "inheritance." One class **uses** another class if it manipulates objects of that class. We say that class A **contains** class B when a member variable of class A makes use of an object of type class B. Section 11.2 presents examples of use and containment.

Inheritance is a process by which one class (the **child** or **derived** class) inherits the properties, methods, and events of another class (the **parent** or **base** class). The child has access to all of its parent's properties, methods and events as well as to all of its own. If the parent is itself a

child, then it and its children have access to all of its parent's properties, methods and events. Consider the classes shown in Fig. 11.6. All three children inherit Property A and Sub B from their parent. Child2 and Child3 have an additional event and a property, respectively. GrandChild1 has access to Property A, Sub B, and Event C from its parent and adds Function E and Sub F. The collection of a parent class along with its descendants is called a **hierarchy**.

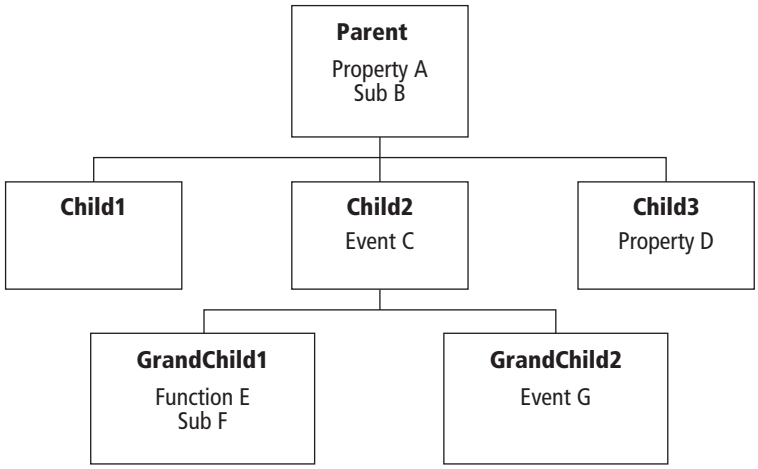


FIGURE 11.6 Example of inheritance hierarchy.

There are two main benefits gained by using inheritance: First, it allows two or more classes to share some common features yet differentiate themselves on others. Second, it supports code reusability by avoiding the extra effort required to maintain duplicate code in multiple classes. For these reasons, inheritance is one of the most powerful tools of object-oriented programming. Considerable work goes into planning and defining the member variables and methods of the parent class. The child classes are beneficiaries of this effort.

Just as structured programming requires the ability to break complex problems into simpler subproblems, object-oriented programming requires the skill to identify useful hierarchies of classes and derived classes. Software engineers are still working on the guidelines for when and how to establish hierarchies. One useful criterion is the **ISA test**: If one class *is a* more specific case of another class, the first class should be derived from the second class.

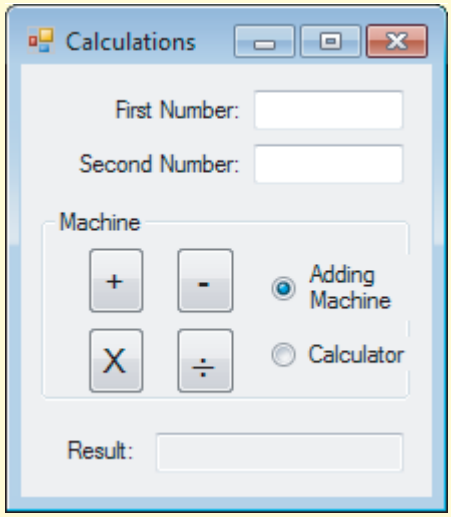
The Visual Basic keyword Inherits identifies the parent of a class. The code used to define the class Parent and its child class Child2 as illustrated in Fig. 11.6 is

```
Class Parent
  Public Property A
    'Property Get and Set blocks
  End Property

  Sub B()
    'Code for Sub procedure B
  End Sub
End Class

Class Child2
  Inherits Parent
  Event C()
End Class
```

As Child2 is itself a parent, its child GrandChild1 can be declared using a similar statement:

```
Class GrandChild1
  Inherits Child2

  Function E()
    'Code for function E
  End Function

  Sub F()
    'Code for Sub procedure F
  End Sub
End Class
```

Example 1 In the following program, the user is presented with a basic adding machine. The Calculator class implements the Multiply and Divide methods and inherits the FirstNumber and SecondNumber properties and the Add and Subtract methods from its AddingMachine parent. When the *Adding Machine* radio button is selected, the user may add or subtract two numbers using an AddingMachine object. When the *Calculator* radio button is selected, the user may add, subtract, multiply, or divide two numbers using a Calculator object. Notice that the multiply and divide buttons are hidden when the Adding Machine is selected, and how the Click event procedures for the *btnAdd* and *btnSubtract* buttons examine the state of the radio button to determine which machine to use.



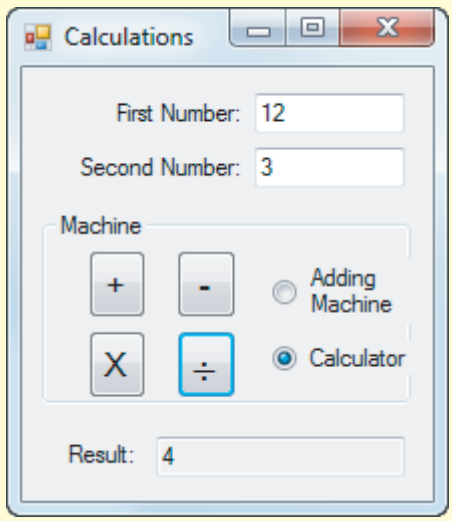
OBJECT	PROPERTY	SETTING
frmCalculate	Text	Calculations
lblNumber1	Text	First Number:
txtNumber1		
lblNumber2	Text	Second Number:
txtNumber2		
lblResult	Text	Result:
txtResult	ReadOnly	True
grpMachine	Text	Machine
radAddingMachine	Text	Adding Machine
	Checked	True
radCalculator	Text	Calculator
btnAdd	Text	+
btnSubtract	Text	−
btnMultiply	Text	×
btnDivide	Font	Symbol
	Text	¸ (Cedilla)

```
Public Class frmCalculate
  'Create both machines.
  Dim adder As New AddingMachine()
  Dim calc As New Calculator()

  Private Sub radAddingMachine_CheckedChanged(...) Handles _
                      radAddingMachine.CheckedChanged
    'Hide the multiply and divide functionality.
    btnMultiply.Visible = False
    btnDivide.Visible = False
  End Sub
```

```
    Private Sub radCalculator_CheckedChanged(...) Handles
                            radCalculator.CheckedChanged
      'Show the multiply and divide functionality.
      btnMultiply.Visible = True
      btnDivide.Visible = True
    End Sub

    Private Sub btnAdd_Click(...) Handles btnAdd.Click
      'Add two numbers.
      If radAddingMachine.Checked Then
        'If adding machine selected, use it to get the result.
        adder.FirstNumber = CDbl(txtNumber1.Text)
        adder.SecondNumber = CDbl(txtNumber2.Text)
        txtResult.Text = CStr(adder.Add)
      Else
        'If calculator selected, use it to get the result.
        calc.FirstNumber = CDbl(txtNumber1.Text)
        calc.SecondNumber = CDbl(txtNumber2.Text)
        txtResult.Text = CStr(calc.Add)
      End If
    End Sub

    Private Sub btnSubtract_Click(...) Handles btnSubtract.Click
      'Subtract two numbers.
      If radAddingMachine.Checked Then
        'If adding machine selected, use it to get the result.
        adder.FirstNumber = CDbl(txtNumber1.Text)
        adder.SecondNumber = CDbl(txtNumber2.Text)
        txtResult.Text = CStr(adder.Subtract)
      Else
        'If calculator selected, use it to get the result.
        calc.FirstNumber = CDbl(txtNumber1.Text)
        calc.SecondNumber = CDbl(txtNumber2.Text)
        txtResult.Text = CStr(calc.Subtract)
      End If
    End Sub

    Private Sub btnMultiply_Click(...) Handles btnMultiply.Click
      'Multiply two numbers.
      calc.FirstNumber = CDbl(txtNumber1.Text)
      calc.SecondNumber = CDbl(txtNumber2.Text)
      txtResult.Text = CStr(calc.Multiply)
    End Sub

    Private Sub btnDivide_Click(...) Handles btnDivide.Click
      'Divide two numbers.
      calc.FirstNumber = CDbl(txtNumber1.Text)
      calc.SecondNumber = CDbl(txtNumber2.Text)
      txtResult.Text = CStr(calc.Divide)
    End Sub
  End Class      'frmCalculate

  Class AddingMachine

    Public Property FirstNumber() As Double
```

```
  Public Property SecondNumber() As Double

  Function Add() As Double
    Return FirstNumber + SecondNumber
  End Function

  Function Subtract() As Double
    Return FirstNumber – SecondNumber
  End Function
End Class       'AddingMachine

Class Calculator
  Inherits AddingMachine
  'Calculator inherits properties FirstNumber and SecondNumber
  'and functions Add() and Subtract().

  Function Multiply() As Double
    Return FirstNumber * SecondNumber
  End Function

  Function Divide() As Double
    Return FirstNumber / SecondNumber
  End Function
End Class       'Calculator
```

[Run, type in 12 and 3, and click on the + and − buttons. Click on the *Calculator* radio button, and click on the +, −, ×, and ÷ buttons.]



Polymorphism and Overriding

The set of properties, methods, and events for a class is called the class **interface**. In essence, the interface of a class defines how it should behave. The interfaces of the classes AddingMachine and Calculator used in Example 1 are shown in Table 11.3.

Consider the classes used in Examples 1 and 2 of Section 11.1. Both Student and PFStudent have the same interface, even though they carry out the task of computing a semester grade differently. See Table 11.4.

TABLE 11.3 Interfaces used in Example 1.

	AddingMachine	Calculator
Properties	FirstNumber	FirstNumber
	SecondNumber	SecondNumber
Methods	Add	Add
	Subtract	Subtract
		Multiply
		Divide
Events	(none)	(none)

TABLE 11.4 Interfaces used in Examples 1 and 2 in Section 11.1.

	Student	PFStudent
Properties	Name	Name
	SocSecNum	SocSecNum
	Midterm	Midterm
	Final	Final
Methods	CalcSemGrade	CalcSemGrade
Events	(none)	(none)

If a programmer wants to write a program that manipulates objects from these two classes, he or she need only know how to use the interface. The programmer need not be concerned with what specific implementation of that interface is being used. The object will then behave according to its specific implementation.

The programmer need only be aware of the CalcSemGrade method and needn't be concerned about its implementation. The feature that two classes can have methods that are named the same and have essentially the same purpose, but different implementations, is called **polymorphism**.

A programmer may employ polymorphism in three easy steps. First, the properties, methods, and events that make up an interface are defined. Second, a parent class is created that performs the functionality dictated by the interface. Finally, a child class inherits the parent and overrides the methods that require different implementation than the parent. The keyword **Overridable** is used to designate the parent's methods that can be overridden, and the keyword **Overrides** is used to designate the child's methods that are doing the overriding.

There are situations where a child class needs to access the parent class's implementation of a method that the child is overriding. Visual Basic provides the keyword **MyBase** to support this functionality.

Consider the code from Example 1 of Section 11.1. To employ polymorphism, the keyword Overridable is inserted into the header of the CalcSemGrade method in the Student class:

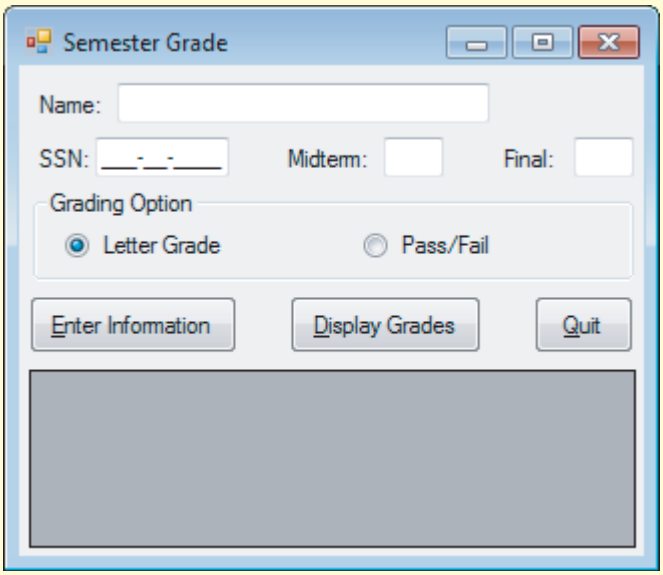
```
Overridable Function CalcSemGrade() As String
```

The PFStudent class inherits all of the properties and methods from its parent, overriding the CalcSemGrade method as follows:

```
Class PFStudent
  Inherits Student
```

```
  Overrides Function CalcSemGrade() As String
    'The student's grade for the semester
    If MyBase.CalcSemGrade = "F" Then
      Return "Fail"
    Else
      Return "Pass"
    End If
  End Function
End Class       'PFStudent
```

Example 2 In the following program, the user can enter student information and display the semester grades for the class. The PFStudent class inherits all of the properties from its parent Student, but overrides the CalcSemGrade method with its own implementation. The btnEnter_Click event procedure stores an element created by either class into the *students* array. However, the btnDisplay_Click event procedure does not need to know which elements are from which class, thus demonstrating polymorphism. ***Note:*** In the sixth line of the btn_Enter event procedure, the statement `pupil = New PFStudent()` is valid, since, due to inheritance, every PFStudent *is a* Student.



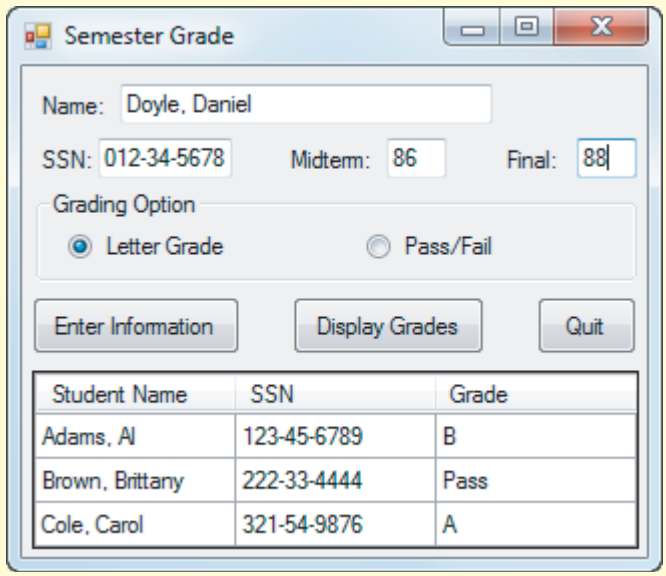
OBJECT	PROPERTY	SETTING
frmGrades	Text	Semester Grade
lblName	Text	Name:
txtName		
lblSSN	Text	SSN:
mtbSSN	Mask	000-00-0000
lblMidterm	Text	Midterm:
txtMidterm		
lblFinal	Text	Final:
txtFinal		
grpGradingOption	Text	Grading Option
radLetterGrade	Text	Letter Grade
	Checked	True
radPassFail	Text	Pass/Fail
btnEnter	Text	&Enter Information
btnDisplay	Text	&Display Grades
btnQuit	Text	&Quit
dgvGrades		

```
Public Class frmGrades
  Dim students(50) As Student             'Stores the class
  Dim lastStudentAdded As Integer = -1    'Last student added to students()

  Private Sub btnEnter_Click(...) Handles btnEnter.Click
    'Stores a student into the array.
    Dim pupil As Student
    'Create the appropriate object depending upon the radio button.
    If radPassFail.Checked Then
      pupil = New PFStudent()
    Else
      pupil = New Student()
    End If
    'Store the values in the text boxes into the object.
    pupil.Name = txtName.Text
```

```
        pupil.SocSecNum = mtbSSN.Text
        pupil.Midterm = CDbl(txtMidterm.Text)
        pupil.Final = CDbl(txtFinal.Text)
        'Add the student to the array.
        lastStudentAdded += 1
        students(lastStudentAdded) = pupil
        'Clear text boxes and list box.
        txtName.Clear()
        mtbSSN.Clear()
        txtMidterm.Clear()
        txtFinal.Clear()
        MessageBox.Show("Student #" & lastStudentAdded + 1 &
                        " recorded.")
        txtName.Focus()
      End Sub

      Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
        ReDim Preserve students(lastStudentAdded)
        Dim query = From pupil In students
                    Select pupil.Name, pupil.SocSecNum, pupil.CalcSemGrade
        dgvGrades.DataSource = query.ToList
        dgvGrades.CurrentCell = Nothing
        dgvGrades.Columns("Name").HeaderText = "Student Name"
        dgvGrades.Columns("SocSecNum").HeaderText = "SSN"
        dgvGrades.Columns("CalcSemGrade").HeaderText = "Grade"
        ReDim Preserve students(50)
        txtName.focus()
      End Sub

      Private Sub btnQuit_Click(...) Handles btnQuit.Click
        'Quit the program
        Me.Close()
      End Sub
    End Class      'frmGrades

    Class Student
      'Member variables to hold the property values
      Private m_midterm As Double
      Private m_final As Double

      Public Property Name() As String

      Public Property SocSecNum() As String

      Public WriteOnly Property Midterm() As Double
        'The student's score on the midterm exam
        Set(ByVal value As Double)
          m_midterm = value
        End Set
      End Property

      Public WriteOnly Property Final() As Double
        'The student's score on the final exam
        Set(ByVal value As Double)
          m_final = value
        End Set
      End Property
```

```
  Overridable Function CalcSemGrade() As String
    'The student's grade for the semester
    Dim grade As Double
    'The grade is based upon average of the midterm and final exams.
    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

Class PFStudent
  Inherits Student

  Overrides Function CalcSemGrade() As String
    'The student's grade for the semester
    If MyBase.CalcSemGrade = "F" Then
      Return "Fail"
    Else
      Return "Pass"
    End If
  End Function
End Class      'PFStudent
```

[Enter the data and click on the *Enter Information* button for three students. Then click on the *Display Grades* button, and finally enter the data for another student.]

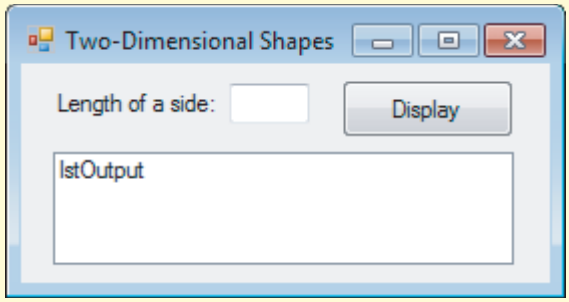


Example 2 employs inheritance and overriding to provide functionality to one child class. If a program contains two or more children of a class, however, the technique of overriding can lead to confusing programs. Visual Basic provides a cleaner design through the use of abstract classes.

Abstract Properties, Methods, and Classes

Sometimes you want to insist that each child of a class have a certain property or method that it must implement for its own use. Such a property or method is said to be **abstract** and is declared with the keyword **MustOverride**. An **abstract** property or method consists of just a header with no code following it. It has no corresponding **`End Property`**, **`End Sub`**, or **`End Function`** statement. Its class is called an **abstract base class** and must be declared with the keyword **MustInherit**. Abstract classes cannot be instantiated; only their children can be instantiated.

Example 3 The following program calculates the area of several regular two-dimensional shapes, given the length of one side. (A regular shape is a shape whose sides have identical length and whose interior angles are identical.) The abstract parent class Shape implements the Length property and declares the Name and Area functions as MustOverride. Notice that methods declared with MustOverride do not have any implementation code. Each child class inherits the property from the parent and implements the two functions. The btnDisplay_Click event procedure uses polymorphism to set the shapes' length and display the shapes' names and areas.



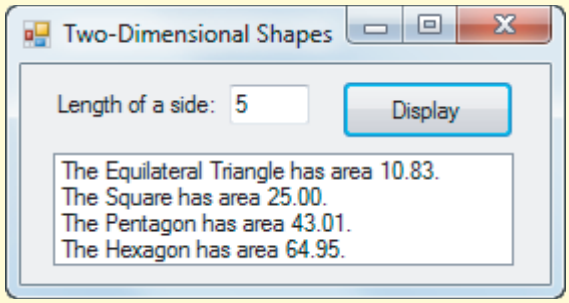
OBJECT	PROPERTY	SETTING
frmShapes	Text	Two-Dimensional Shapes
lblLength	Text	Length of a side:
txtLength		
btnDisplay	Text	Display
lstOutput		

```
Public Class frmShapes
  'Declare shape array.
  Dim shape(3) As Shape

  Private Sub frmShapes_Load(...) Handles MyBase.Load
    'Populate the array with shapes.
    shape(0) = New EquilateralTriangle()
    shape(1) = New Square()
    shape(2) = New Pentagon()
    shape(3) = New Hexagon()
  End Sub

  Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
    Dim length As Double
    'Set lengths of all shapes.
    length = CDbl(txtLength.Text)
    For i As Integer = 0 To 3
      shape(i).Length = length
    Next
```

```
    'Display results.
    lstOutput.Items.Clear()
    For i As Integer = 0 To 3
      lstOutput.Items.Add("The " & shape(i).Name & " has area " &
                          FormatNumber(shape(i).Area)) & "."
    Next
  End Sub
End Class      'frmShapes

MustInherit Class Shape
  Public Property Length() As Double

  MustOverride Function Name() As String
  'Returns the name of the shape.

  MustOverride Function Area() As Double
  'Returns the area of the shape.
End Class      'Shape

Class EquilateralTriangle
  Inherits Shape

  Overrides Function Name() As String
    'The name of this shape
    Return "Equilateral Triangle"
  End Function

  Overrides Function Area() As Double
    'Formula for the area of an equilateral triangle
    Return Length * Length * Math.Sqrt(3) / 4
  End Function
End Class      'EquilateralTriangle

Class Square
  Inherits Shape

  Overrides Function Name() As String
    'The name of this shape
    Return "Square"
  End Function

  Overrides Function Area() As Double
    'Formula for the area of a square
    Return Length * Length
  End Function
End Class      'Square

Class Pentagon
  Inherits Shape

  Overrides Function Name() As String
    'The name of this shape
    Return "Pentagon"
  End Function
```

```
  Overrides Function Area() As Double
    'Formula for the area of a pentagon
    Return Length * Length * Math.Sqrt(25 + (10 * Math.Sqrt(5))) / 4
  End Function
End Class      'Pentagon

Class Hexagon
  Inherits Shape

  Overrides Function Name() As String
    'The name of this shape
    Return "Hexagon"
  End Function

  Overrides Function Area() As Double
    'Formula for the area of a hexagon
    Return Length * Length * 3 * Math.Sqrt(3) / 2
  End Function
End Class      'Hexagon
```

[Run the program, enter 5, and click on the *Display* button.]



Comments

1. Visual Basic uses inheritance in every Windows application that is written. Examination of any program's code reveals that the form's class inherits from the .NET framework class System.Windows.Forms.Form.
2. In Example 2, the btnDisplay_Click event procedure does not need to know which elements of the Student array are instances of the Student class and which are instances of the PFStudent class. In some situations, however, the program may want to know this. Visual Basic provides the expression `TypeOf...Is` to test if an instance was created from a particular class (or from the class' parents, grandparents, etc.) For example, the following procedure counts the number of pass/fail students in the *students* array:

```
Sub CountPassFail()
  Dim query = From student In students
              Where TypeOf (student) Is PFStudent
              Select student
  Dim numPF = query.Count
  MessageBox.Show("There are " & numPF & " pass/fail students out of " &
                  lastStudentAdded + 1 & " students in the class.")
End Sub
```

3. Child classes do not have access to the parent's Private member variables.

Practice Problems 11.3

1. In the class AddingMachine of Example 1, the Add function could have been defined with

```
Function Add() As Double
  Return _FirstNumber + _SecondNumber
End Function
```

Explain why the Multiply function of the class Calculator cannot be defined with

```
Function Multiply() As Double
  Return _FirstNumber * _SecondNumber
End Function
```

2. Consider the hierarchy of classes shown below. What value is assigned to the variable *phrase* by the following two lines of code?

```
  Dim mammal As New Mammals()
  Dim phrase As String = mammal.Msg

Class Animals
  Overridable Function Msg() As String
    Return "Can move"
  End Function
End Class

Class Vertebrates
  Inherits Animals
  Overrides Function Msg() As String
    Return MyBase.Msg & " " & "Has a backbone"
  End Function
End Class

Class Mammals
  Inherits Vertebrates
  Overrides Function Msg() As String
    Return MyBase.Msg & " " & "Nurtures young with mother's milk"
  End Function
End Class

Class Arthropods
  Inherits Animals
  Overrides Function Msg() As String
    Return MyBase.Msg & " " & "Has jointed limbs and no backbone"
  End Function
End Class
```

EXERCISES 11.3

In Exercises 1 through 4, identify the output of the code that uses the following two classes:

```
Class Square
  Overridable Function Result(ByVal num As Double) As Double
    Return num * num
  End Function
End Class
```

```
Class Cube
  Inherits Square

  Overrides Function Result(ByVal num As Double) As Double
    Return num * num * num
  End Function
End Class
```

1. ```
   Dim sq As Square = New Square()
   txtOutput.Text = CStr(sq.Result(2))
   ```

2. ```
   Dim cb As Cube = New Cube()
   txtOutput.Text = CStr(cb.Result(2))
   ```

3. ```
   Dim m As Square = New Square()
   Dim n As Cube = New Cube()
   txtOutput.Text = CStr(m.Result(n.Result(2)))
   ```

4. ```
   Dim m As Square = New Cube()
   txtOutput.Text = CStr(m.Result(2))
   ```

5. Consider the class hierarchy in the second practice problem. What value is assigned to the variable *phrase* by the following two lines of code?

   ```
   Dim anthropod As New Arthropods()
   Dim phrase As String = arthropod.Msg
   ```

6. Consider the class hierarchy in the second practice problem. What value is assigned to the variable *phrase* by the following two lines of code?

   ```
   Dim vertebrate As New Vertebrates()
   Dim phrase As String = vertebrate.Msg
   ```

In Exercises 7 through 16, identify the errors in the code.

7. ```
   Class Hello
     Function Hi() As String
       Return "hi!"
     End Function
   End Class

   Class Greetings
     Overrides Hello
     Function GoodBye() As String
       Return "goodbye"
     End Function
   End Class
   ```

8. ```
   Class Hello
     Function Hi() As String
       Return "hi!"
     End Function
   End Class
   ```

```
Class Greetings
  Inherits Hi()

  Function GoodBye() As String
    Return "goodbye"
  End Function
End Class
```

9.
```
Class Hello
  Function Hi() As String
    Return "hi!"
  End Function
End Class

Class Aussie
  Inherits Hello

  Function Hi() As String
    Return "G'day mate!"
  End Function
End Class
```

10.
```
Class Hello
  Function Hi() As String
    Return "hi!"
  End Function
End Class

Class WithIt
  Inherits Hello

  Overrides Function Hi() As String
    Return "Hey"
  End Function
End Class
```

11.
```
Class Hello
  Overridable Function Hi() As String
    Return "hi!"
  End Function
End Class

Class Cowboy
  Inherits Hello

  Function Hi() As String
    Return "howdy!"
  End Function
End Class
```

12.
```
Class Hello
  MustOverride Function Hi() As String
    Return "hi!"
  End Function
End Class
```

```
Class DragRacer
  Inherits Hello

  Overrides Function Hi() As String
    Return "Start your engines!"
  End Function
End Class
```

13.
```
Class Hello
  MustInherit Function Hi() As String
End Class

Class Gentleman
  Inherits Hello

  Overrides Function Hi() As String
    Return "Good day"
  End Function
End Class
```

14.
```
Class Hello
  MustOverride Function Hi() As String
End Class

Class Euro
  Inherits Hello

  Overrides Function Hi() As String
    Return "Caio"
  End Function
End Class
```

15.
```
MustOverride Class Hello
  MustOverride Function Hi() As String
End Class

Class Southerner
  Inherits Hello

  Overrides Function Hi() As String
    Return "Hi y'all"
  End Function
End Class
```

16.
```
MustInherit Class Hello
  MustOverride Function Hi() As String
End Class

Class NorthEasterner
  Inherits Hello

  Overrides Function Hi(ByVal name As String) As String
    Return "How ya doin', " & name
  End Function
End Class
```

17. Expand Example 1 to use a class ScientificCalculator that is derived from the class Calculator and has an exponentiation button in addition to the four arithmetic buttons.

18. Rewrite Example 2 so that the class Student has an abstract method CalcSemGrade and two derived classes called LGStudent (LG stands for "Letter Grade") and PFStudent.

19. Consider the class CashRegister from Exercise 25 of Section 11.1. Create a derived class called FastTrackRegister that could be used at a toll booth to collect money from vehicles and keep track of the number of vehicles processed. Write a program using the class and having the form in Fig. 11.7. One dollar should be collected from each car and two dollars from each truck.

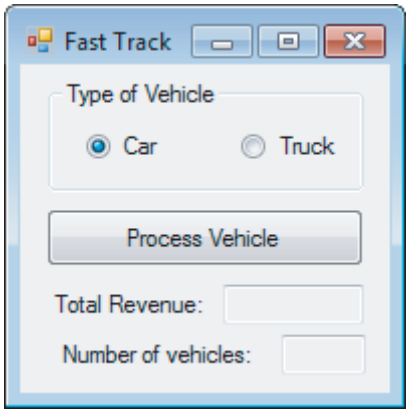


FIGURE 11.7 **Form for Exercise 19.**

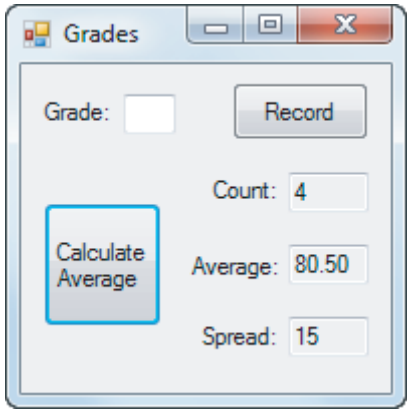


FIGURE 11.8 **Sample output for Exercise 20.**

VideoNote
Student registration (Homework)

20. Consider the class Statistics from Exercise 26 of Section 11.1. Create a derived class called CompleteStats that also provides a Spread function and an event called NewSpread. This event should be raised whenever the spread changes. (The *spread* is the difference between the highest and the lowest grades.) Write a program that uses the classes to analyze up to 50 exam grades input by the user. The program should display the number of grades and the current spread at all times. When the *Calculate Average* button is clicked on, the program should display the average of the grades. A sample output is shown in Fig. 11.8.

21. Write a program that keeps track of a bookstore's inventory. The store orders both trade books and textbooks from publishers. The program should define an abstract class Book that contains the MustOverride property Price, and the ordinary properties Quantity, Name, and Cost. The Textbook and Tradebook classes should be derived from the class Book and should override property Price by adding a markup. (Assume that the markup is 40% for a trade book and 20% for a textbook.) The program should accept input from the user on book orders and display the following statistics: total number of books, number of textbooks, total cost of the orders, and total value of the inventory. (The value of the inventory is the amount of money that the bookstore can make if it sells all of the books in stock.) A sample output is shown in Fig. 11.9.

22. Write a program that records the weekly payroll of a department that hires both salaried and hourly employees. The program should accept user input and display the number of employees, the number of salaried employees, the total payroll, and the average number of hours worked. The abstract class Employee should contain Name and Rate properties. (The Rate text box should be filled in with the weekly salary for salaried workers and the hourly wage for hourly workers.) The Salaried and Hourly classes should inherit the Employee

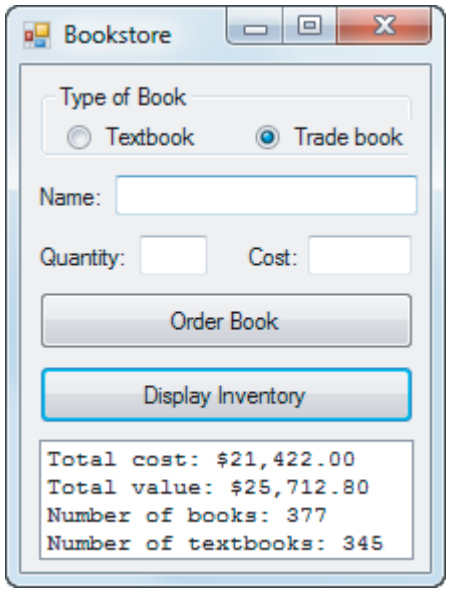


FIGURE 11.9 Sample output for Exercise 21.

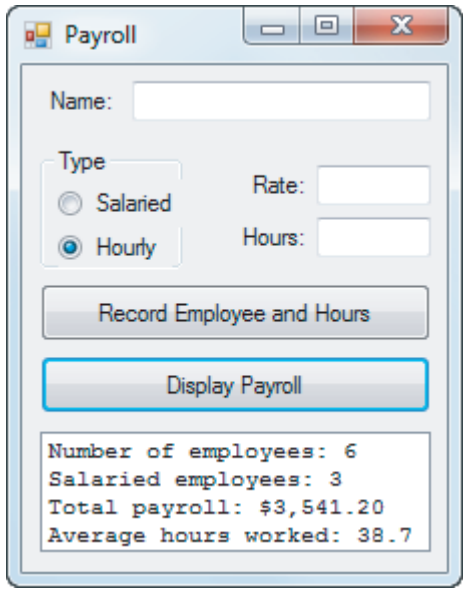


FIGURE 11.10 Sample output for Exercise 22.

class and override the method GrossPay that accepts the number of hours worked as a parameter. A sample output is shown in Fig. 11.10. (***Hint:*** Use an array of a structure that holds the employee object and the number of hours worked during the week.)

Solutions to Practice Problems 11.3

1. While the derived class Calculator has access to the Properties and Methods of the base class AddingMachine, it does not have access to its Private member variables.
2. The string "Can move Has a backbone Nurtures young with mother's milk"

CHAPTER 11 SUMMARY

1. An *object* is an entity that stores data, has methods that manipulate the data, and can raise events. A *class* is a template from which objects are created. A *method* specifies the way in which an object's data are manipulated. An *event* is a message sent by an object to signal the occurrence of a condition.
2. Each class is defined in a separate block of code starting with Class *ClassName* and ending with End Class. Data are stored in member variables and accessed by procedures called properties.
3. A property routine contains a Get block to retrieve the value of a member variable or a Set block to assign a value to a member variable. These procedures can also be used to enforce constraints and carry out validation.
4. Visual Basic automatically invokes a New procedure when an object is created.
5. An object variable is declared with a statement of the form `Dim objectName As ClassName`, and the object is created with a statement of the form `objectName = New ClassName(arg1, arg2, ...)`. These two statements are often combined into the single statement `Dim objectName As New ClassName(arg1, arg2, ...)`.
6. *Auto-implemented properties* enable you to quickly specify a property of a class without having to write code to Get and Set the property.