11

Introduction to Visual Basic Programming: Inheritance
Say not you know another entirely, 
till you have divided an inheritance with him.

– Johann Kasper Lavater

This method is to define as the number 
of a class the class of all classes similar 
to the given class.

– Bertrand Russell
**Good as it is to inherit a library, it is better to collect one.**

– Augustine Birrell

**Save base authority from others’ books.**

– William Shakespeare
OBJECTIVES

In this chapter you will learn:

- What inheritance is and how it promotes software reusability.
- The notions of base classes and derived classes.
- To use keyword `inherited` to create a class that inherits attributes and behaviors from another class.
- To use the access modifier `Protected` in a base class to give derived class methods access to base class members.
OBJECTIVES

- To access base class members from a derived class with `MyBase`.
- How constructors are used in inheritance hierarchies.
- To access the current object with `Me` and `MyClass`.
- The methods of class `Object`—the direct or indirect base class of all classes in Visual Basic.
11.1 Introduction
11.2 Base Classes and Derived Classes
11.3 Protected Members
11.4 Relationship between Base Classes and Derived Classes
11.5 Constructors in Derived Classes
11.6 Software Engineering with Inheritance
11.7 Class Object
11.8 Friend Members
11.1 Introduction

• In inheritance, a new class is created by extending an existing class’s members.
• The existing class is called the base class, and the new class is the derived class.
• An indirect base class is inherited from two or more levels up in the class hierarchy.
11.1 Introduction (Cont.)

• Every class in Visual Basic extends (or “inherits from”) Object.

• An is-a relationship represents inheritance. For example, a car is a vehicle.

• By contrast, the has-a relationship represents composition. For example, a car has a steering wheel.
11.2 Base Classes and Derived Classes

- Figure 11.1 lists several simple examples of base classes and derived classes.

<table>
<thead>
<tr>
<th>Base class</th>
<th>Derived classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Student</td>
<td>GraduateStudent, UndergraduateStudent</td>
</tr>
<tr>
<td>Shape</td>
<td>Circle, Triangle, Rectangle</td>
</tr>
<tr>
<td>Loan</td>
<td>CarLoan, HomeImprovementLoan, MortgageLoan</td>
</tr>
<tr>
<td>Employee</td>
<td>Faculty, Staff</td>
</tr>
<tr>
<td>BankAccount</td>
<td>CheckingAccount, SavingsAccount</td>
</tr>
</tbody>
</table>

Fig. 11.1 | Inheritance examples.
11.2 Base Classes and Derived Classes (Cont.)

• An **inheritance hierarchy** shows the relationship between classes.

![Inheritance hierarchy for university Community Members.](image)

**Fig. 11.2** | Inheritance hierarchy for university Community Members.
11.2 Base Classes and Derived Classes (Cont.)

**Fig. 11.3** | Inheritance hierarchy for *Shapes*. 
11.3 Protected Members

- Protected members can be accessed only by the class and its derived classes.

- Public and Protected members retain their original access modifiers in derived classes.

- When a derived class method overrides a method, the base class version can be accessed from the derived class with keyword MyBase.
11.3 Protected Members (Cont.)

Software Engineering Observation 11.1

Derived class methods cannot directly access Private members of their base class. A derived class can change the state of Private base class instance variables only through non-Private methods provided in the base class and inherited by the derived class.

Software Engineering Observation 11.2

Declaring Private instance variables helps programmers test, debug and correctly modify systems. If a derived class could access its base class’s Private instance variables, classes that inherit from that derived class could access the instance variables as well. This would propagate access to what should be Private instance variables, and the benefits of information hiding would be lost.
• Commission Employees are paid a percentage of their sales (Fig. 11.4).

```vbnet
Public Class CommissionEmployee
    Inherits Object ' optional
    Private firstNameValue As String ' first name
    Private lastNameValue As String ' last name
    Private socialSecurityNumberValue As String ' social security number
    Private grossSalesValue As Decimal ' gross weekly sales
    Private commissionRateValue As Double ' commission percentage

    ' five-argument constructor
    Public Sub New(ByVal first As String, ByVal last As String, ByVal ssn As String, ByVal sales As Decimal, ByVal rate As Double)
        ' implicit call to Object constructor occurs here
        FirstName = first
        LastName = last
        SocialSecurityNumber = ssn
        GrossSales = sales ' validate and store gross sales
        CommissionRate = rate ' validate and store commission rate
    End Sub ' New
```

Fig. 11.4 | CommissionEmployee class represents an employee paid a percentage of gross sales. (Part 1 of 5.)
' property FirstName
Public Property FirstName() As String
    Get
        Return firstNameValue
    End Get
    Set(ByVal first As String)
        firstNameValue = first ' no validation
    End Set
End Property ' FirstName

' property LastName
Public Property LastName() As String
    Get
        Return lastNameValue
    End Get
    Set(ByVal last As String)
        lastNameValue = last ' no validation
    End Set
End Property ' LastName

Fig. 11.4 | CommissionEmployee class represents an employee paid a percentage of gross sales. (Part 2 of 5.)
' property SocialSecurityNumber
Public Property SocialSecurityNumber() As String
    Get
        Return socialSecurityNumberValue
    End Get
    Set(ByVal ssn As String)
        socialSecurityNumberValue = ssn ' no validation
    End Set
End Property ' SocialSecurityNumber

' property GrossSales
Public Property GrossSales() As Decimal
    Get
        Return grossSalesValue
    End Get
End Property ' GrossSales

Fig. 11.4 | CommissionEmployee class represents an employee paid a percentage of gross sales. (Part 3 of 5.)
Fig. 11.4 | CommissionEmployee class represents an employee paid a percentage of gross sales. (Part 4 of 5.)
Fig. 11.4 | CommissionEmployee class represents an employee paid a percentage of gross sales. (Part 5 of 5.)
Common Programming Error 11.1
It is a compilation error to attempt to override a method that is not declared `Overridable`.

Common Programming Error 11.2
It is a compilation error to override a method with a method that has a different access modifier than the method being overridden.
Module CommissionEmployeeTest

Sub Main()

' instantiate CommissionEmployee object
Dim employee As New CommissionEmployee("Sue", "Jones", 222-22-2222, 10000D, 0.06)

' get commission employee data
Console.WriteLine("Employee information obtained by properties:")
& vbCrLf & "First name is " & employee.FirstName & vbCrLf & "Last name is " & employee.LastName & vbCrLf & "Social Security Number is " & employee.SocialSecurityNumber"

Console.WriteLine("Gross sales is {0:C}", employee.GrossSales)
Console.WriteLine("Commission rate is {0:F}", employee.CommissionRate)
employee.GrossSales = 500D ' set gross sales
employee.CommissionRate = 0.1 ' set commission rate to 10%

' get new employee information
Console.WriteLine(vbCrLf & 
  "Updated employee information obtained by ToString: " & _
  vbCrLf & employee.ToString() & vbCrLf)

' display the employee's earnings
Console.WriteLine("Employee's earnings: {0:C}", _
  employee.CalculateEarnings())

End Sub ' Main
End Module ' CommissionEmployeeTest

Employee information obtained by properties:
First name is Sue
Last name is Jones
Social Security Number is 222-22-2222
Gross sales is $10,000.00
Commission rate is 0.06

Updated employee information obtained by ToString:
commission employee: Sue Jones
social security number: 222-22-2222
gross sales: $500.00
commission rate: 0.10

Employee's earnings: $50.00

Fig. 11.5 | CommissionEmployee class test program. (Part 2 of 2.)
• **BasePlusCommissionEmployee** is a separate class with a similar function (Fig. 11.6).

```vbnet
' Fig. 11.6: BasePlusCommissionEmployee.vb
' BasePlusCommissionEmployee class represents an employee that receives
' a base salary in addition to a commission.
Public Class BasePlusCommissionEmployee
    Private firstNameValue As String ' first name
    Private lastNameValue As String ' last name
    Private socialSecurityNumberValue As String ' social security number
    Private grossSalesValue As Decimal ' gross weekly sales
    Private commissionRateValue As Double ' commission percentage
    Private baseSalaryValue As Decimal ' base salary per week

    ' six-argument constructor
    Public Sub New(ByVal first As String, ByVal last As String, _
                    ByVal ssn As String, ByVal sales As Decimal, _
                    ByVal rate As Double, ByVal salary As Decimal)
    Dim firstName As String = first
    Dim lastName As String = last
    Dim ssnValue As String = ssn
    Dim grossSales As Decimal = sales
    Dim commissionRate As Double = rate
    Dim baseSalary As Decimal = salary

    Private Sub New(
        ByVal firstName As String, ByVal lastName As String, _
        ByVal ssnValue As String, ByVal grossSales As Decimal, _
        ByVal commissionRate As Double, ByVal baseSalary As Decimal)
    ' six-argument constructor
    Public Sub New(ByVal firstName As String, ByVal lastName As String, _
                    ByVal ssnValue As String, ByVal grossSales As Decimal, _
                    ByVal commissionRate As Double, ByVal baseSalary As Decimal)
```

A BasePlusCommissionEmployee has an additional baseSalaryValue variable.

The BasePlusCommissionEmployee constructor has an additional salary parameter.

**Fig. 11.6** | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part 1 of 7.)
' implicit call to Object constructor occurs here

FirstName = first
LastName = last
SocialSecurityNumber = ssn
GrossSales = sales ' validate and store gross sales
CommissionRate = rate ' validate and store commission rate
BaseSalary = salary ' validate and store base salary

End Sub ' New

' property FirstName
Public Property FirstName() As String
   Get
       Return firstNameValue
   End Get

   Set(ByVal first As String)
       firstNameValue = first ' no validation
   End Set

End Property ' FirstName

Fig. 11.6 | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part 2 of 7.)
' property LastName
Public Property LastName() As String
    Get
        Return lastNameValue
    End Get
    Set(ByVal last As String)
        lastNameValue = last ' no validation
    End Set
End Property ' LastName

' property SocialSecurityNumber
Public Property SocialSecurityNumber() As String
    Get
        Return socialSecurityNumberValue
    End Get
    Set(ByVal ssn As String)
        socialSecurityNumberValue = ssn ' no validation
    End Set
End Property ' SocialSecurityNumber

Fig. 11.6 | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part 3 of 7.)
Fig. 11.6 | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part 4 of 7.)
Fig. 11.6 | BasePlusCommissionEmployee class represents an employee who receives a base salary in addition to a commission. (Part 5 of 7.)
The additional `BaseSalary` property validates its input.

Calculates a `BasePlusCommissionEmployee` class represents an employee who receives a base salary in addition to a commission. (Part 6 of 7.)
Fig. 11.6 | The `BasePlusCommissionEmployee` class represents an employee who receives a base salary in addition to a commission. (Part 7 of 7.)
• Figure 11.7 tests class

BasePlusCommissionEmployee.

• The BasePlusCommissionEmployee constructor is *almost* identical to that of class CommissionEmployee.

```vbnet
Module BasePlusCommissionEmployeeTest
    Sub Main()
        ' instantiate BasePlusCommissionEmployee object
        Dim employee As New BasePlusCommissionEmployee(  
            "Bob", "Lewis", "333-33-3333", 5000D, 0.04, 300D)

        ' get base-salaried commission employee data
        Console.WriteLine("Employee information obtained by properties:"  
            & vbCrLf & "First name is " & employee.FirstName & vbCrLf &  
            "Social Security Number is " & employee.SocialSecurityNumber)

        Console.WriteLine("Gross sales is {0:C}", employee.GrossSales)
        Console.WriteLine("Commission rate is {0:F}", _
```

**Fig. 11.7 |** BasePlusCommissionEmployee test program. (Part 1 of 2.)
Employee information obtained by properties:
First name is Bob
Last name is Lewis
Social Security Number is 333-33-3333
Gross sales is $5,000.00
Commission rate is 0.04
Base salary is $300.00

Updated employee information obtained by ToString:
base-plus-commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: $5,000.00
commission rate: 0.04
base salary: $1,000.00

Employee's earnings: $1,200.00

Fig. 11.7 | BasePlusCommissionEmployee test program. (Part 2 of 2.)
Software Engineering Observation 11.3

Copying and pasting code from one class to another can spread errors among multiple source code files.

Software Engineering Observation 11.4

With inheritance, the common instance variables and methods of all the classes in the hierarchy are declared in a base class. When changes are required for these common features, software developers need to make the changes only in the base class—derived classes then inherit the changes. Without inheritance, the changes would need to be made to all the source code files that contain copies of the code in question.
• Method `CalculateEarnings` is now declared `Overridable` (Fig. 11.8).

```vbnet
' Fig. 11.8: CommissionEmployee.vb
' CommissionEmployee class represents a commission employee.
Public Class CommissionEmployee
  Inherits Object ' optional

  Private firstNameValue As String ' first name
  Private lastNameValue As String ' last name
  Private socialSecurityNumberValue As String ' social security number
  Private grossSalesValue As Decimal ' gross weekly sales
  Private commissionRateValue As Double ' commission percentage

  ' five-argument constructor
  Public Sub New(ByVal first As String, ByVal last As String, _
                  ByVal ssn As String, ByVal sales As Decimal, ByVal rate As Double)
    ' implicit call to Object constructor occurs here
    FirstName = first
    LastName = last
    SocialSecurityNumber = ssn
    GrossSales = sales ' validate and store gross sales
    CommissionRate = rate ' validate and store commission rate
  End Sub ' New
```
' property FirstName
Public Property FirstName() As String
    Get
        Return firstNameValue
    End Get

    Set(ByVal first As String)
        firstNameValue = first ' no validation
    End Set
End Property ' FirstName

' property LastName
Public Property LastName() As String
    Get
        Return lastNameValue
    End Get

    Set(ByVal last As String)
        lastNameValue = last ' no validation
    End Set
End Property ' LastName

Fig. 11.8 | CommissionEmployee class with Overridable method CalculateEarnings. (Part 2 of 6.)
' property SocialSecurityNumber
Public Property SocialSecurityNumber() As String
    Get
        Return socialSecurityNumberValue
    End Get
    Set(ByVal ssn As String)
        socialSecurityNumberValue = ssn ' no validation
    End Set
End Property ' SocialSecurityNumber

' property GrossSales
Public Property GrossSales() As Decimal
    Get
        Return grossSalesValue
    End Get
End Property ' GrossSales

Fig. 11.8 | CommissionEmployee class with Overridable method CalculateEarnings. (Part 3 of 6.)
Set(ByVal sales As Decimal)
If sales < 0D Then ' validate gross sales
    grossSalesValue = 0D
Else
    grossSalesValue = sales
End If
End Set
End Property ' GrossSales
'

property CommissionRate
Public Property CommissionRate() As Decimal
    Get
    Return commissionRateValue
End Get

Fig. 11.8 | CommissionEmployee class with Overridable method CalculateEarnings. (Part 4 of 6.)
Set(ByVal rate As Decimal)
    If rate > 0.0 AndAlso rate < 1.0 Then ' validate rate
        commissionRateValue = rate
    Else
        commissionRateValue = 0.0
    End If
End Set
End Property ' CommissionRate

' calculate earnings
Public Overridable Function CalculateEarnings() As Decimal
    Return Convert.ToDecimal(commissionRateValue) * grossSalesValue
End Function ' CalculateEarnings

Fig. 11.8 CommissionEmployee class with Overridable method CalculateEarnings. (Part 5 of 6.)
' return String representation of CommissionEmployee object
Public Overrides Function ToString() As String
    Return ("commission employee: " & firstNameValue & " " & _
        lastNameValue & vbNewLine & "social security number: " & _
        socialSecurityNumberValue & vbNewLine & "gross sales: " & _
        String.Format("{0:C}", grossSalesValue) & vbNewLine & _
        "commission rate: " & String.Format("{0:F}", _
        commissionRateValue))
End Function 'ToString
End Class 'CommissionEmployee

Fig. 11.8 | CommissionEmployee class with Overridable method CalculateEarnings. (Part 6 of 6.)
BasePlusCommissionEmployee (Fig. 11.9) now inherits from CommissionEmployee.

```vbnet
Public Class BasePlusCommissionEmployee
    Inherits CommissionEmployee

    Private baseSalaryValue As Decimal ' base salary per week

    ' six-argument constructor
    Public Sub New(ByVal first As String, ByVal last As String, _
                    ByVal ssn As String, ByVal sales As Decimal, _
                    ByVal rate As Double, ByVal salary As Decimal)
        MyBase.New(first, last, ssn, sales, rate)
        BaseSalary = salary ' validate and store base salary
    End Sub ' New

    ' property BaseSalary
    Public Property BaseSalary() As Decimal
        Get
            Return baseSalaryValue
        End Get
    End Property

Fig. 11.9 | Private base class members cannot be accessed in a derived class. (Part 1 of 3.)
```
```vbnet
23  Set(ByVal salary As Decimal)
24      If salary < 0D Then ' validate base salary
25          baseSalaryValue = 0D
26      Else
27          baseSalaryValue = salary
28      End If
29  End Set
30  End Property ' BaseSalary
31
32  ' calculate earnings
33  Public Overrides Function CalculateEarnings() As Decimal
34      ' not allowed: attempts to access private base class members
35      Return baseSalaryValue + (  
36          Convert.ToDecimal(commissionRateValue) * grossSalesValue)
37  End Function ' CalculateEarnings
```

**Fig. 11.9** | *Private* base class members cannot be accessed in a derived class. (Part 2 of 3.)
`return String representation of BasePlusCommissionEmployee object`

```vbnet
Public Overrides Function ToString() As String
' not allowed: attempts to access private base class members
    Return ("base-plus-commission employee: " & firstNameValue & 
        " " & lastNameValue & vbCrLf & "social security number: " & 
        socialSecurityNumberValue & vbCrLf & "gross sales: " & 
        String.Format("(0:C)", grossSalesValue) & vbCrLf & 
        "commission rate: " & String.Format("(0:F)", 
        commissionRateValue) & vbCrLf & "base salary: " & 
        String.Format("(0:C)", baseSalaryValue))
End Function ' ToString
```

Fig. 11.9 | Private base class members cannot be accessed in a derived class. (Part 3 of 3.)
Class CommissionEmployee (Fig. 11.10) declares its instance variables as Protected rather than Private.

Derived classes gain access to these instance variables.

```vbnet
Public Class CommissionEmployee
    Inherits Object ' optional

    Protected firstNameValue As String ' first name
    Protected lastNameValue As String ' last name
    Protected socialSecurityNumberValue As String ' social security number
    Protected grossSalesValue As Decimal ' gross weekly sales
    Protected commissionRateValue As Double ' commission percentage

    ' five-argument constructor
    Public Sub New(ByVal first As String, ByVal last As String, ByVal ssn As String, ByVal sales As Decimal, ByVal rate As Double)
        ' implicit call to Object constructor occurs here
        FirstName = first
    End Sub
End Class
```

Fig. 11.10 | CommissionEmployee class with Protected instance variables. (Part 1 of 6.)
LastName = last
SocialSecurityNumber = ssn
GrossSales = sales ' validate and store gross sales
CommissionRate = rate ' validate and store commission rate
End Sub ' New

' property FirstName
Public Property FirstName() As String
    Get
        Return firstNameValue
    End Get

    Set(ByVal first As String)
        firstNameValue = first ' no validation
    End Set
End Property ' FirstName

' property LastName
Public Property LastName() As String
    Get
        Return lastNameValue
    End Get

    Set(ByVal last As String)
        lastNameValue = last ' no validation
    End Set
End Property ' LastName

Fig. 11.10 | CommissionEmployee class with Protected instance variables. (Part 2 of 6.)
' property SocialSecurityNumber
Public Property SocialSecurityNumber() As String
    Get
        Return socialSecurityNumberValue
    End Get
Set(ByVal ssn As String)
    socialSecurityNumberValue = ssn ' no validation
End Set
End Property ' SocialSecurityNumber

' property GrossSales
Public Property GrossSales() As Decimal
    Get
        Return grossSalesValue
    End Get
End Property ' GrossSales

Fig. 11.10 | CommissionEmployee class with Protected instance variables. (Part 3 of 6.)
44

Outline

Fig. 11.10 | CommissionEmployee class with protected instance variables. (Part 4 of 6.)
```vbnet
Set(ByVal rate As Double)
    If rate > 0.0 AndAlso rate < 1.0 Then ' validate rate
        commissionRateValue = rate
    Else
        commissionRateValue = 0.0
    End If
End Set
End Property ' CommissionRate

' calculate earnings
Public Overridable Function CalculateEarnings() As Decimal
    Return Convert.ToDecimal(commissionRateValue) * grossSalesValue
End Function ' CalculateEarnings
```

**Fig. 11.10** | CommissionEmployee class with protected instance variables. (Part 5 of 6.)
Fig. 11.10 | CommissionEmployee class with Protected instance variables. (Part 6 of 6.)
• Modify `BasePlusCommissionEmployee` (Fig. 11.9) so that it inherits from `CommissionEmployee`.

```vbnet
Public Class BasePlusCommissionEmployee
    Inherits CommissionEmployee

    Private baseSalaryValue As Decimal ' base salary per week

    ' six-argument constructor
    Public Sub New(ByVal first As String, ByVal last As String, ByVal ssn As String, ByVal sales As Decimal, ByVal rate As Double, ByVal salary As Decimal)
        MyBase.New(first, last, ssn, sales, rate)
        BaseSalary = salary ' validate and store base salary
    End Sub
```

© 2009 Pearson Education, Inc. All rights reserved.

`BasePlusCommissionEmployee` inherits Protected instance variables from `CommissionEmployee`. (Part 1 of 3.)
' property BaseSalary
Public Property BaseSalary() As Decimal
    Get
        Return baseSalaryValue
    End Get
    
Set(ByVal salary As Decimal)
If salary < 0D Then ' validate base salary
    baseSalaryValue = 0D
Else
    baseSalaryValue = salary
End If
End Set
End Property ' BaseSalary

' calculate earnings
Public Overrides Function CalculateEarnings() As Decimal
    Return baseSalaryValue + ( Convert.ToDecimal(commissionRateValue) * grossSalesValue)
End Function ' CalculateEarnings

Fig. 11.11 | BasePlusCommissionEmployee inherits instance variables from CommissionEmployee. (Part 2 of 3.)
Fig. 11.11 | BasePlusCommissionEmployee inherits Protected instance variables from CommissionEmployee. (Part 3 of 3.)
Figure 11.12 tests the modified classes.

```vbnet
Module BasePlusCommissionEmployeeTest
    Sub Main()
        ' instantiate BasePlusCommissionEmployee object
        Dim employee As New BasePlusCommissionEmployee(
            "Bob", "Lewis", "333-33-3333", 5000D, 0.04, 300D)

        ' get base-salaried commission employee data
        Console.WriteLine("Employee information obtained by properties:" & vbCrLf & "First name is " & employee.FirstName & vbCrLf & "Last name is " & employee.LastName & vbCrLf & "Social Security Number is " & employee.SocialSecurityNumber)
        Console.WriteLine("Gross sales is {0:C}", employee.GrossSales)
        Console.WriteLine("Commission rate is {0:F}", employee.CommissionRate)
        Console.WriteLine("Base salary is {0:C}", employee.BaseSalary)
    End Sub
End Module
```

**Fig. 11.12** | Protected base class members inherited into derived class BasePlusCommissionEmployee. (Part 1 of 2.)
employee.BaseSalary = 1000D ' set base salary

' get new employee information
Console.WriteLine(vbNewLine & _
    "Updated employee information obtained by ToString: " & _
    vbNewLine & employee.ToString() & vbCrLf)

' display the employee's earnings
Console.WriteLine( Employee's earnings: {0:C} & _
    employee.CalculateEarnings())
End Sub ' Main
End Module ' BasePlusCommissionEmployeeTest

Employee Information obtained by properties:
First name is Bob
Last name is Lewis
Social Security Number is 333-33-3333
Gross sales is $5,000.00
Commission rate is 0.04
Base salary is $300.00

Updated employee information obtained by ToString:
base-plus-commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: $5,000.00
commission rate: 0.04
base salary: $1,000.00

Employee's earnings: $1,200.00
11.4 Relationship between Base Classes and Derived Classes (Cont.)

• Using `Protected` instance variables creates potential problems.
  – The derived class can set an inherited variable to an invalid value.
  – Derived classes should depend only on the base class services and not on data implementation.
  – We should be able to change the base class implementation while still providing the same services.

Software Engineering Observation 11.5

Use the `Protected` access modifier on a method when a base class is to provide the method to its derived classes but not to other clients.
• Class **CommissionEmployee** (Fig. 11.13) now declares **Private** instance variables and provides **Public** properties.

```vbnet
Public Class CommissionEmployee
    Private firstNameValue As String ' first name
    Private lastNameValue As String ' last name
    Private socialSecurityNumberValue As String ' social security number
    Private grossSalesValue As Decimal ' gross weekly sales
    Private commissionRateValue As Double ' commission percentage

    ' five-argument constructor
    Public Sub New(ByVal first As String, ByVal last As String, ByVal ssn As String, ByVal sales As Decimal, ByVal rate As Double)
        ' implicit call to Object constructor occurs here
        FirstName = first
        LastName = last
        SocialSecurityNumber = ssn
        GrossSales = sales ' validate and store gross sales
        CommissionRate = rate ' validate and store commission rate
    End Sub ' New

Fig. 11.13 | CommissionEmployee class uses properties to manipulate its **Private** instance variables. (Part 1 of 6.)
```

© 2009 Pearson Education, Inc. All rights reserved.
' property FirstName
Public Property FirstName() As String
    Get
        Return firstNameValue
    End Get

    Set(ByVal first As String)
        firstNameValue = first ' no validation
    End Set
End Property ' FirstName

' property LastName
Public Property LastName() As String
    Get
        Return lastNameValue
    End Get

    Set(ByVal last As String)
        lastNameValue = last ' no validation
    End Set
End Property ' LastName

Fig. 11.13 | CommissionEmployee class uses properties to manipulate its Private instance variables. (Part 2 of 6.)
Fig. 11.13  |  CommissionEmployee class uses properties to manipulate its Private instance variables. (Part 3 of 6.)
CommisionEmployee.vb

(4 of 6)

Fig. 11.13 | CommissionEmployee class uses properties to manipulate its Private instance variables. (Part 4 of 6.)
Set(ByVal rate As Double)
    If rate > 0.0 And Also rate < 1.0 Then ' validate rate
        commissionRateValue = rate
    Else
        commissionRateValue = 0.0
    End If
End Set ' CommissionRate

' calculate earnings
Public Overridable Function CalculateEarnings() As Decimal
    Return Convert.ToDecimal(CommissionRate) * GrossSales
End Function

Fig. 11.13 | CommissionEmployee class uses properties to manipulate its instance variables. (Part 5 of 6.)
Private instance variables.

Fig. 11.13  |  CommissionEmployee class uses properties to manipulate its Private instance variables. (Part 6 of 6.)
Software Engineering Observation 11.6

Declaring base class instance variables Private (as opposed to Protected) enables the base class implementation of these instance variables to change without affecting derived class implementations.

Error-Prevention Tip 11.1

When possible, do not include Protected instance variables in a base class. Instead, include non-Private properties and methods that carefully access Private instance variables. This will ensure that objects of the derived classes of this base class maintain consistent states of the base class instance variables.
Derived class **BasePlusCommissionEmployee** (Fig. 11.14) inherits **CommissionEmployee**’s non-private properties and methods.

```vbnet
' Fig. 11.14: BasePlusCommissionEmployee.vb
' BasePlusCommissionEmployee inherits from class CommissionEmployee.
Public Class BasePlusCommissionEmployee
  Inherits CommissionEmployee

  Private baseSalaryValue As Decimal ' base salary per week

  ' six-argument constructor
  Public Sub New(ByVal first As String, ByVal last As String, _
                  ByVal ssn As String, ByVal sales As Decimal, _
                  ByVal rate As Double, ByVal salary As Decimal)
    MyBase.New(first, last, ssn, sales, rate)
    BaseSalary = salary ' validate and store base salary
  End Sub ' New
```

**Fig. 11.14** | BasePlusCommissionEmployee class inherits CommissionEmployee, which provides only instance variables. (Part 1 of 3.)
Adding base salary to a CommissionEmployee's earnings.

```vbnet
' property BaseSalary
Public Property BaseSalary() As Decimal
    Get
        Return baseSalaryValue
    End Get

    Set(ByVal salary As Decimal)
        If salary < 0D Then ' validate base salary
            baseSalaryValue = 0D
        Else
            baseSalaryValue = salary
        End If
    End Set
End Property ' BaseSalary

' calculate earnings
Public Overrides Function CalculateEarnings() As Decimal
    Return BaseSalary + MyBase.CalculateEarnings()
End Function ' CalculateEarnings
```

Fig. 11.14 | BasePlusCommissionEmployee class Inherits CommissionEmployee, which provides only instance variables. (Part 2 of 3.)
```
37 ' return String representation of BasePlusCommissionEmployee object
38 Public Overrides Function ToString() As String
39     Return ("base-plus-" & MyBase.ToString() & vbCrLf & _
40     "base salary:" & String.Format("{0:C}", BaseSalary))
41 End Function ' ToString
42 End Class ' BasePlusCommissionEmployee
```

**Fig. 11.14** BasePlusCommissionEmployee class inherits CommissionEmployee, which provides only instance variables. (Part 3 of 3.)

Adding the base salary property to the CommissionEmployee’s output.
Common Programming Error 11.3

When a base class method is overridden in a derived class, the derived class version often calls the base class version to do a portion of the work. Failure to prefix the base class method name with the keyword `MyBase` and a dot (.) separator causes the derived class method to call itself, usually resulting in infinite recursion.
• Figure 11.15 performs a test of the BasePlusCommissionEmployee object.

**Fig. 11.15** | Base class Private instance variables are accessible to a derived class via the Public or Protected properties and methods inherited by the derived class. (Part 1 of 2.)
Employee.BaseSalary = 1000D ' set base salary

' get new employee information
Console.WriteLine(vbNewLine & _
    "Updated employee information obtained by ToString: " & _
    vbCrLf & employee.ToString() & vbCrLf)

' display the employee's earnings
Console.WriteLine("Employee's earnings: {0:C}"
    & vbCrLf & employee.CalculateEarnings())

End Sub ' Main
End Module ' BasePlusCommissionEmployeeTest

Employee information obtained by properties:
First name is Bob
Last name is Lewis
Social Security Number is 333-33-3333
Gross sales is $5,000.00
Commission rate is 0.04
Base salary is $300.00

Updated employee information obtained by ToString:
base-plus-commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: $5,000.00
commission rate: 0.04
base salary: $1,000.00

Employee's earnings: $1,200.00
• Instantiating a derived class object begins a chain of constructor calls.

• Our next example redeclares class `CommissionEmployee` (Fig. 11.16) with the constructor printing a message.

```vbnet
Public Class CommissionEmployee
    Private firstNameValue As String ' first name
    Private lastNameValue As String ' last name
    Private socialSecurityNumberValue As String ' social security number
    Private grossSalesValue As Decimal ' gross weekly sales
    Private commissionRateValue As Double ' commission percentage

    ' five-argument constructor
    Public Sub New(ByVal first As String, ByVal last As String, ByVal ssn As String, ByVal sales As Decimal, ByVal rate As Double)
        ' implicit call to Object constructor occurs here
        FirstName = first
        LastName = last
        SocialSecurityNumber = ssn
        GrossSales = sales ' validate and store gross sales
        CommissionRate = rate ' validate and store commission rate
    End Sub
End Class
```

Fig. 11.16 | CommissionEmployee’s constructor outputs text. (Part 1 of 6.)
When this class’s constructor executes, the object’s information is printed.

Outputting `Me` implicitly invokes the `ToString` method.

---

**Fig. 11.16 | CommissionEmployee’s constructor outputs text. (Part 2 of 6.)**
property SocialSecurityNumber
Public Property SocialSecurityNumber() As String
    Get
        Return socialSecurityNumberValue
    End Get
    Set(ByVal ssn As String)
        socialSecurityNumberValue = ssn ' no validation
    End Set
End Property ' SocialSecurityNumber

property GrossSales
Public Property GrossSales() As Decimal
    Get
        Return grossSalesValue
    End Get
End Property ' GrossSales

Fig. 11.16 | CommissionEmployee's constructor outputs text. (Part 3 of 6.)
```
Set(ByVal sales As Decimal)
If sales < 0D Then ' validate gross sales
    grossSalesValue = 0D
Else
    grossSalesValue = sales
End If
End Set
End Property ' GrossSales

' property CommissionRate
Public Property CommissionRate() As Double
    Get
    Return commissionRateValue
End Get
```

**Fig. 11.16** | *CommissionEmployee*'s constructor outputs text. (Part 4 of 6.)
Set(ByVal rate As Double)
    If rate > 0.0 AndAlso rate < 1.0 Then ' validate rate
        commissionRateValue = rate
    Else
        commissionRateValue = 0.0
    End If
End Set
End Property ' CommissionRate

' calculate earnings
Public Overridable Function CalculateEarnings() As Decimal
    Return Convert.ToDecimal(CommissionRate) * GrossSales
End Function ' CalculateEarnings
Fig. 11.16 | CommissionEmployee’s constructor outputs text. (Part 6 of 6.)
The `BasePlusCommissionEmployee` (Fig. 11.17) constructor also outputs text when invoked.

```vbnet
Public Class BasePlusCommissionEmployee
    Inherits CommissionEmployee

    Private baseSalaryValue As Decimal ' base salary per week

    ' six-argument constructor
    Public Sub New(ByVal first As String, ByVal last As String, _
                    ByVal ssn As String, ByVal sales As Decimal, _
                    ByVal rate As Double, ByVal salary As Decimal)
        MyBase.New(first, last, ssn, sales, rate)
        BaseSalary = salary ' validate and store base salary
        Console.WriteLine(vbNewLine & _
                          "BasePlusCommissionEmployee constructor: {0}", Me)
    End Sub ' New
```

When this class’s constructor executes, the object’s information is printed.
Fig. 11.17 BasePlusCommissionEmployee's constructor outputs text. (Part 2 of 3.)
```
35 ' calculate earnings
36 Public Overrides Function CalculateEarnings() As Decimal
37 Return BaseSalary + MyBase.CalculateEarnings()
38 End Function ' CalculateEarnings
39
40 ' return String representation of BasePlusCommissionEmployee object
41 Public Overrides Function ToString() As String
42 Return "base-plus-" & MyBase.ToString() & vbNewLine & _
43 "base salary: " & String.Format("{0:C}", BaseSalary)
44 End Function ' ToString
45
46 End Class ' BasePlusCommissionEmployee
```

**Fig. 11.17** | BasePlusCommissionEmployee’s constructor outputs text. (Part 3 of 3.)
• Figure 11.18 demonstrates the order in which constructors are called.

• `BasePlusCommissionEmployee`’s version of the `ToString` method is called from `CommissionEmployee`’s constructor, so the base salary is not yet initialized in the second output.

```vbnet
Module Constructor
    Sub Main()
        Dim employee1 As New CommissionEmployee(  
            "Bob", "Lewis", "333-33-3333", 5000D, 0.04)
        Console.WriteLine()
        Dim employee2 As New BasePlusCommissionEmployee(  
            "Lisa", "Jones", "555-55-5555", 2000D, 0.06, 800D)
        End Sub ' Main
    End Module ' Constructor
```

**Fig. 11.18** | Constructor call order using `Me`. (Part 1 of 2.)
CommissionEmployee constructor:
commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: $5,000.00
commission rate: 0.04

CommissionEmployee constructor:
base-plus-commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: $2,000.00
commission rate: 0.06
base salary: $0.00

BasePlusCommissionEmployee constructor:
base-plus-commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: $2,000.00
commission rate: 0.06
base salary: $800.00

Fig. 11.18 | Constructor call order using Me. (Part 2 of 2.)
11.5 Constructors in Derived Classes (Cont.)

• A method call with MyClass always invokes the version of the method defined in that particular class.

• Replace Me in line 21 with


MyClass.ToString()


• CommissionEmployee’s ToString will be executed by the class’s constructor.
11.5 Constructors in Derived Classes (Cont.)

- Now execute the program. The output is shown in Fig. 11.19.

```java
CommissionEmployee constructor:
commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: $5,000.00
commission rate: 0.04

CommissionEmployee constructor:
commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: $2,000.00
commission rate: 0.06

BasePlusCommissionEmployee constructor:
base-plusCommission employee: Lisa Jones
social security number: 555-55-5555
gross sales: $2,000.00
commission rate: 0.06
base salary: $800.00
```

Fig. 11.19 | Constructor call order using MyClass.
11.6 Software Engineering with Inheritance

• Extending classes to meet our needs by including additional members and by overriding base class members is a useful practice.

• Independent software vendors (ISVs) develop proprietary classes for sale or license and make them available to users in MSIL format.
Software Engineering Observation 11.7

Despite the fact that inheriting from a class does not require access to the class’s source code, developers often insist on seeing the source code to see how the class is implemented. They want to ensure that they are extending a solid class that performs well and is implemented securely.
At the design stage in an object-oriented system, the designer often finds that certain classes are closely related. The designer should “factor out” common instance variables and methods and place them in a base class. Then the designer should use inheritance to develop derived classes, specializing them with capabilities beyond those inherited from the base class.

Performance Tip 11.1

If derived classes are larger than they need to be (i.e., contain too much functionality), memory and processing resources may be wasted. Extend the base class that contains the functionality that is closest to what you need.
11.6 Software Engineering with Inheritance (Cont.)

Performance Tip 11.1

If derived classes are larger than they need to be (i.e., contain too much functionality), memory and processing resources may be wasted. Extend the base class that contains the functionality that is closest to what you need.
11.7 Class Object

- All classes inherit seven methods from class `Object` (Fig. 11.20).

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Equals</code></td>
<td>Compares two objects for equality; returns <code>True</code> if they are equal and <code>False</code> otherwise.</td>
</tr>
<tr>
<td><code>Finalize</code></td>
<td>This <code>Protected</code> method is called by the garbage collector on an object just before the garbage collector reclaims the object’s memory.</td>
</tr>
<tr>
<td><code>GetHashCode</code></td>
<td>A hashtable is a data structure that relates one object, called the key, to another object, called the value.</td>
</tr>
</tbody>
</table>

*Fig. 11.20 | `Object` methods that are inherited by all classes. (Part 1 of 2.)*
### 11.7 Class Object (Cont.)

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GetType</strong></td>
<td>Returns an object of class <code>Type</code> that contains information about the object’s type.</td>
</tr>
<tr>
<td><strong>MemberwiseClone</strong></td>
<td>This <strong>Protected</strong> method makes a copy of the object on which it is called. The implementation of this method performs a <strong>shallow copy</strong>—instance variable values in one object are copied into another object of the same type.</td>
</tr>
<tr>
<td><strong>ReferenceEquals</strong></td>
<td>This <strong>Shared</strong> method returns <code>True</code> if two objects are the same instance.</td>
</tr>
<tr>
<td><strong>ToString</strong></td>
<td>This method returns a <code>String</code> representation of an object.</td>
</tr>
</tbody>
</table>

**Fig. 11.20** | **Object** methods that are inherited by all classes. (Part 2 of 2.)
11.8 Friend Members

• A class’s Friend members can be accessed only by code in the same assembly.
• To access a non-Shared Friend member within the same assembly, create an object of the class that declares the Friend member.
• You can access a Friend member that is Shared via the name of the class that declares the Friend member.
• Protected Friend members are accessible both from code in the same assembly and by subclasses.