Object-Oriented Programming: Inheritance

Source : [BOOK] How to programming C#

Introduction

Inheritance lets you **save time** during app development by reusing proven and debugged high-quality software.

The existing class from which a new class inherits members is called the **base-class**, and the new class is the **derived-class**.

Each derived class can **become the base class** for future derived classes.

Introduction

Continue

A derived class normally **adds its own fields** and methods => it's more **specific** than its base class.

The **direct-base-class** is the base class from which the derived class explicitly inherits. An **indirect-base-class** is any class above the direct base class in the **class hierarchy**, which defines the inheritance relationships among classes.

Introduction

Continue

The class hierarchy begins with class **object** (*System.Object* in the Framework Class Library), which every class **directly or indirectly extends (or "inherits from")**.

C# supports only **single inheritance**.

Is-a and has-a relationship

Continue

Is-a represents inheritance. In an is-a relationship, an object of a derived class can also be **treated as an object of its base class**.

For example, a car is a vehicle, and a boat is a vehicle.

By contrast, ***has-a*** represents **composition**. In a has-a relationship, an object contains as **members references to other objects**.

For example, a car has a steering wheel, and a car object has a reference to a steering-wheel object.

Base Classes and Derived Classes

In geometry, a **rectangle is a quadrilateral** (as are squares, trapezoids, and etc) => class **Rectangle** can be said to **inherit from class Quadrilateral**.

A **rectangle is a specific type of quadrilateral**, but it's incorrect to claim that every quadrilateral is a rectangle.

base classes tend to be more general, and **derived classes tend to be more specific**.

Base Classes and Derived Classes

Continue

Base class	Derived classes
Student	GraduateStudent, UndergraduateStudent
Shape	Circle, Triangle, Rectangle
Loan	CarLoan, HomeImprovementLoan, MortgageLoan
Employee	Faculty, Staff, HourlyWorker, CommissionWorker
BankAccount	CheckingAccount, SavingsAccount

Fig. 11.1 | Inheritance examples.

Base Classes and Derived Classes

Continue

The set of objects represented by a base class is typically **larger** than the set of objects represented by any of its derived classes

For example: Car and Vehicle

Base Classes and Derived Classes

Continue

Inheritance relationships form **treelike hierarchical structures**.

A class becomes either a base class, **supplying members** to other classes, or a derived class, **inheriting** its members from another class.

Sometimes, a class is **both a base and a derived class**.

Base Classes and Derived Classes

Continue

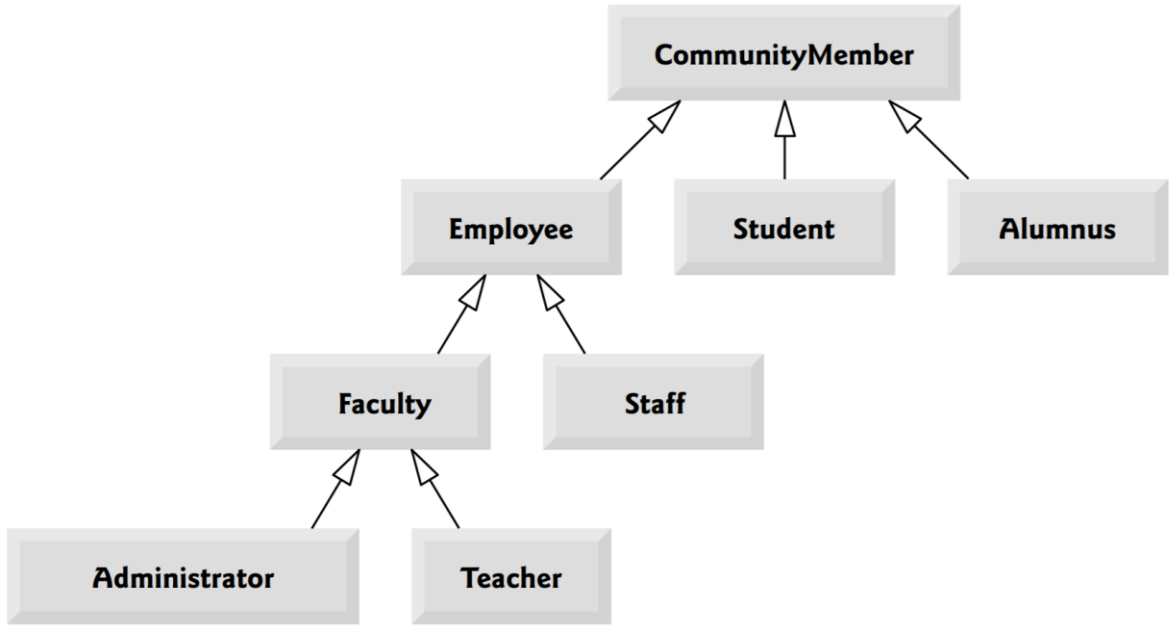


Fig. 11.2 | UML class diagram showing an inheritance hierarchy for university `CommunityMembers`.

Base Classes and Derived Classes

Continue

Each arrow with a hollow **triangular arrowhead** in the hierarchy diagram represents an is-a relationship.

As we follow the arrows, we can state, for instance, that “an **Employee is a *CommunityMember***” and “a **Teacher is a Faculty member**.”

CommunityMember is the **direct base class** of Employee, Student and Alumnus and is an **indirect base class** of all the other classes in the diagram.

Starting from the bottom, you can **follow the arrows** and apply the is-a relationship up to the topmost base class. For example, an Administrator is a Faculty member, is an Employee and is a *CommunityMember*.

Base Classes and Derived Classes

Continue

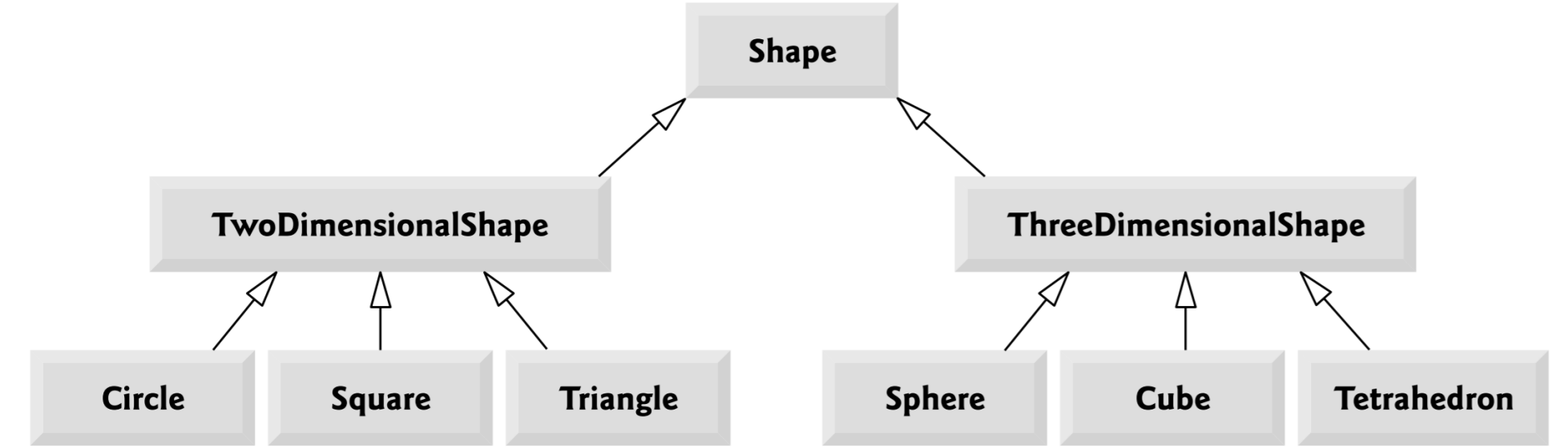


Fig. 11.3 | UML class diagram showing an inheritance hierarchy for Shapes.

Base Classes and Derived Classes

Continue

Now consider the **Shape hierarchy** in Fig. 11.3, which begins with **base class Shape**. This class is **extended** by derived classes *TwoDimensionalShape* and *ThreeDimensionalShape—a* Shape is either a *TwoDimensionalShape* or a *ThreeDimensionalShape*. The third level of this hierarchy contains **specific *TwoDimensionalShapes* and *ThreeDimensionalShapes***. We can follow the arrows from the bottom to the topmost base class in this hierarchy to identify the is-a relationships. For instance, **a Triangle is a *TwoDimensionalShape* and is a Shape**, while a Sphere is a *ThreeDimensionalShape* and is a Shape. This hierarchy could contain many other classes. For example, ellipses and trapezoids also are *TwoDimensionalShapes*.

Base Classes and Derived Classes

Continue

A derived class can **customize methods** it inherits from its base class. In such cases, the derived class can **override (redefine) the base-class method** with an appropriate implementation, as we'll see often in the chapter's code examples.

Protected Members

Using protected access offers an **intermediate level of access** between public and private. A base class's protected members can be **accessed** by members of that base class and by members of its **derived classes**.

All non-private base-class members **retain their original access** modifier when they become members of the derived class—public members of the base class become public members of the derived class, and protected members of the base class become protected members of the derived class.

Software Engineering Observation

Properties and methods of a derived class **cannot directly access private members** of the base class.

A derived class can change the state of private base-class fields only **through non-private methods and properties provided in the base class**.

Software Engineering Observation

Declaring **private fields** in a base class helps you **test, debug and correctly modify systems**. If a derived class could access its base class's private fields, classes that inherit from that derived class could access the fields as well. This would **propagate access to what should be private** fields, and the benefits of information hiding would be lost.

Relationship between Base Classes and Derived Classes

Inheritance hierarchy containing **types of employees** in a company's payroll app to discuss the **relationship** between a base class and its derived classes.

base-salaried commission employee, Commission Employee, Object

We divide our discussion of the relationship between commission employees and base-salaried commission employees into **five examples**:

Relationship between Base Classes and Derived Classes

Continue

1. The first creates **class *CommissionEmployee***, which **directly inherits from class object** and declares as **private instance variables** a first name, last name, social security number, commission rate and gross (i.e., total) sales amount.

2. The second declares class ***BasePlusCommissionEmployee***, which also directly inherits from **object** and declares as **private instance variables** a first name, last name, social security number, commission rate, gross sales amount and base salary.

Relationship between Base Classes and Derived Classes

Continue

3. The third declares a separate **BasePlusCommissionEmployee class that extends class CommissionEmployee** (i.e., a BasePlusCommissionEmployee is a CommissionEmployee who also has a base salary). We show that **base-class methods** must be explicitly declared **virtual** if they're to be overridden by methods in derived classes.

Relationship between Base Classes and Derived Classes

Continue

4. The fourth shows that if base class *CommissionEmployee's* instance variables are declared as **protected**, a BasePlusCommissionEmployee class that inherits from class CommissionEmployee can access that data directly. For this purpose, we declare class CommissionEmployee with protected instance variables.

5. The fifth example demonstrates best practice by **setting the CommissionEmployee instance variables back to private** in class CommissionEmployee to enforce good software engineering. Then we show how a separate BasePlusCommissionEmployee class, which inherits from class CommissionEmployee, can **use *CommissionEmployee's* public methods to manipulate** *CommissionEmployee's* private instance variables.

```
// Fig. 11.4: CommissionEmployee.cs
// CommissionEmployee class represents a commission employee.
using System;

public class CommissionEmployee : Object
{
   private string firstName;
   private string lastName;
   private string socialSecurityNumber;
   private decimal grossSales; // gross weekly sales
   private decimal commissionRate; // commission percentage
```

1.1 Creating and Using a CommissionEmployee Class

Every class in C# (except object) **extends an existing class**.

The colon (:) followed by class name object at the end of the declaration header indicates that class **CommissionEmployee extends (i.e., inherits from) object class** (System.Object in the Framework Class Library).

Because class CommissionEmployee extends class object, class CommissionEmployee inherits the **methods** of class object—class object has no fields. Every C# class **directly or indirectly inherits object's methods**. If a class does not specify that it inherits from another class, the new class **implicitly** inherits from object => you typically do not include ": object" in your code

```
// five-parameter constructor
public CommissionEmployee( string first, string last, string ssn,
   decimal sales, decimal rate )
{
   // implicit call to object constructor occurs here
   firstName = first;
   lastName = last;
   socialSecurityNumber = ssn;
   GrossSales = sales; // validate gross sales via property
   CommissionRate = rate; // validate commission rate via property
} // end five-parameter CommissionEmployee constructor
```

```
// read-only property that gets commission employee's first name
   public string FirstName
   {
      get
      {
         return firstName;
      } // end get
   } // end property FirstName

   // read-only property that gets commission employee's last name
   public string LastName
   {
      get
      {
         return lastName;
      } // end get
   } // end property LastName
```

```
// read-only property that gets
   // commission employee's social security number
   public string SocialSecurityNumber
   {
      get
      {
         return socialSecurityNumber;
      } // end get
   } // end property SocialSecurityNumber
```

```
// property that gets and sets commission employee's commission rate
   public decimal CommissionRate
   {
      get
      {
         return commissionRate;
      } // end get
      set
      {
         if ( value > 0 && value < 1 )
            commissionRate = value;
         else
            throw new ArgumentOutOfRangeException( "CommissionRate",
               value, "CommissionRate must be > 0 and < 1" );
      } // end set
   } // end property CommissionRate
```

```
// calculate commission employee's pay
   public decimal Earnings()
   {
      return commissionRate * grossSales;
   } // end method Earnings

   // return string representation of CommissionEmployee object
   public override string ToString()
   {
      return string.Format(
         "{0}: {1} {2}\n{3}: {4}\n{5}: {6:C}\n{7}: {8:F2}",
         "commission employee", FirstName, LastName,
         "social security number", SocialSecurityNumber,
         "gross sales", GrossSales, "commission rate", CommissionRate );
   } // end method ToString
} // end class CommissionEmployee
```

Common Programming Error

It’s a **compilation error to override a method with one that has a different access modifier** => The method **must have the same access modifier** for all that class’s direct and indirect derived classes.

```
// Fig. 11.5: CommissionEmployeeTest.cs
// Testing class CommissionEmployee.
using System;

public class CommissionEmployeeTest
{
   public static void Main( string[] args )
   {
      // instantiate CommissionEmployee object
      CommissionEmployee employee = new CommissionEmployee( "Sue",
         "Jones", "222-22-2222", 10000.00M, .06M );
```

```
// display commission employee data
      Console.WriteLine(
         "Employee information obtained by properties and methods: \n" );
      Console.WriteLine( "First name is {0}", employee.FirstName );
      Console.WriteLine( "Last name is {0}", employee.LastName );
      Console.WriteLine( "Social security number is {0}",
         employee.SocialSecurityNumber );
      Console.WriteLine( "Gross sales are {0:C}", employee.GrossSales );
      Console.WriteLine( "Commission rate is {0:F2}",
         employee.CommissionRate );
      Console.WriteLine( "Earnings are {0:C}", employee.Earnings() );

      employee.GrossSales = 5000.00M; // set gross sales
      employee.CommissionRate = .1M; // set commission rate

      Console.WriteLine( "\n{0}:\n\n{1}",
         "Updated employee information obtained by ToString", employee );
      Console.WriteLine( "earnings: {0:C}", employee.Earnings() );
   } // end Main
} // end class CommissionEmployeeTest
```

```
Employee information obtained by properties and methods:

First name is Sue
Last name is Jones
Social security number is 222-22-2222
Gross sales are $10,000.00
Commission rate is 0.06
Earnings are $600.00

Updated employee information obtained by ToString:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: $5,000.00
commission rate: 0.10
earnings: $500.00
```

1.2 Creating a BasePlusCommissionEmployee Class without Using Inheritance

Second part of our introduction to inheritance by declaring and testing the (**completely new and independent**) class BasePlusCommissionEmployee, which contains a first name, last name, social security number, gross sales amount, commission rate and base salary.

Class *BasePlusCommissionEmployee's* **public services** include a BasePlusCommissionEmployee constructor , methods Earnings and ToString and **public properties** for the class's private instance variables *firstName*, *lastName*, *socialSecurityNumber*, *grossSales*, *commissionRate* and *baseSalary*

Note the **similarity** between this class and class CommissionEmployee

```
// Fig. 11.6: BasePlusCommissionEmployee.cs
// BasePlusCommissionEmployee class represents an employee that receives
// a base salary in addition to a commission.
using System;

public class BasePlusCommissionEmployee
{
   private string firstName;
   private string lastName;
   private string socialSecurityNumber;
   private decimal grossSales; // gross weekly sales
   private decimal commissionRate; // commission percentage
   private decimal baseSalary; // base salary per week
```

```
// six-parameter constructor
public BasePlusCommissionEmployee( string first, string last,
   string ssn, decimal sales, decimal rate, decimal salary )
{
   // implicit call to object constructor occurs here
   firstName = first;
   lastName = last;
   socialSecurityNumber = ssn;
   GrossSales = sales; // validate gross sales via property
   CommissionRate = rate; // validate commission rate via property
   BaseSalary = salary; // validate base salary via property
} // end six-parameter BasePlusCommissionEmployee constructor
```

```
// read-only property that gets
   // BasePlusCommissionEmployee's first name
   public string FirstName
   {
      get
      {
         return firstName;
      } // end get
   } // end property FirstName

   // read-only property that gets
   // BasePlusCommissionEmployee's last name
   public string LastName
   {
      get
      {
         return lastName;
      } // end get
   } // end property LastName
```

```
// read-only property that gets
// BasePlusCommissionEmployee's social security number
public string SocialSecurityNumber
{
   get
   {
      return socialSecurityNumber;
   } // end get
} // end property SocialSecurityNumber
```

```
// property that gets and sets
   // BasePlusCommissionEmployee's gross sales
   public decimal GrossSales
   {
      get
      {
         return grossSales;
      } // end get
      set
      {
         if ( value >= 0 )
            grossSales = value;
         else
            throw new ArgumentOutOfRangeException(
               "GrossSales", value, "GrossSales must be >= 0" );
      } // end set
   } // end property GrossSales
```

```
// property that gets and sets
   // BasePlusCommissionEmployee's commission rate
   public decimal CommissionRate
   {
      get
      {
         return commissionRate;
      } // end get
      set
      {
         if ( value > 0 && value < 1 )
            commissionRate = value;
         else
            throw new ArgumentOutOfRangeException( "CommissionRate",
               value, "CommissionRate must be > 0 and < 1" );
      } // end set
   } // end property CommissionRate
```

```
// property that gets and sets
   // BasePlusCommissionEmployee's base salary
   public decimal BaseSalary
   {
      get
      {
         return baseSalary;
      } // end get
      set
      {
         if ( value >= 0 )
            baseSalary = value;
         else
            throw new ArgumentOutOfRangeException( "BaseSalary",
               value, "BaseSalary must be >= 0" );
      } // end set
   } // end property BaseSalary
```

```
// calculate earnings
   public decimal Earnings()
   {
      return baseSalary + ( commissionRate * grossSales );
   } // end method Earnings

   // return string representation of BasePlusCommissionEmployee
   public override string ToString()
   {
      return string.Format(
         "{0}: {1} {2}\n{3}: {4}\n{5}: {6:C}\n{7}: {8:F2}\n{9}: {10:C}",
         "base-salaried commission employee", firstName, lastName,
         "social security number", socialSecurityNumber,
         "gross sales", grossSales, "commission rate", commissionRate,
         "base salary", baseSalary );
   } // end method ToString
} // end class BasePlusCommissionEmployee
```

```
// Fig. 11.7: BasePlusCommissionEmployeeTest.cs
// Testing class BasePlusCommissionEmployee.
using System;

public class BasePlusCommissionEmployeeTest
{
   public static void Main( string[] args )
   {
      // instantiate BasePlusCommissionEmployee object
      BasePlusCommissionEmployee employee =
         new BasePlusCommissionEmployee( "Bob", "Lewis",
         "333-33-3333", 5000.00M, .04M, 300.00M );
```

```
// display BasePlusCommissionEmployee's data
      Console.WriteLine(
         "Employee information obtained by properties and methods: \n" );
      Console.WriteLine( "First name is {0}", employee.FirstName );
      Console.WriteLine( "Last name is {0}", employee.LastName );
      Console.WriteLine( "Social security number is {0}",
         employee.SocialSecurityNumber );
      Console.WriteLine( "Gross sales are {0:C}", employee.GrossSales );
      Console.WriteLine( "Commission rate is {0:F2}",
         employee.CommissionRate );
      Console.WriteLine( "Earnings are {0:C}", employee.Earnings() );
      Console.WriteLine( "Base salary is {0:C}", employee.BaseSalary );

      employee.BaseSalary = 1000.00M; // set base salary

      Console.WriteLine( "\n{0}:\n\n{1}",
         "Updated employee information obtained by ToString", employee );
      Console.WriteLine( "earnings: {0:C}", employee.Earnings() );
   } // end Main
} // end class BasePlusCommissionEmployeeTest
```

```
Employee information obtained by properties and methods:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales are $5,000.00
Commission rate is 0.04
Earnings are $500.00
Base salary is $300.00

Updated employee information obtained by ToString:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: $5,000.00
commission rate: 0.04
base salary: $1,000.00
earnings: $1,200.00
```

Much of the code for class BasePlusCommissionEmployee is **similar**, **if not identical**, to the code for class CommissionEmployee.

For example, in class BasePlusCommissionEmployee, private instance variables firstName and lastName and properties FirstName and LastName are **identical** to those of class CommissionEmployee. Classes CommissionEmployee and BasePlusCommissionEmployee also both contain private instance variables **socialSecurityNumber, commissionRate and grossSales**, as well as **properties** to manipulate these variables.

In addition, the BasePlusCommissionEmployee **constructor is almost identical** to that of class CommissionEmployee, except that BasePlusCommissionEmployee's constructor also sets the **baseSalary**.

The other **additions** to class BasePlusCommissionEmployee are private instance variable **baseSalary** and property **BaseSalary**.

Class BasePlusCommissionEmployee's **Earnings** method is nearly identical to that of class CommissionEmployee, except that BasePlusCommissionEmployee's also **adds the baseSalary**.

Similarly, class BasePlusCommissionEmployee's **ToString method is nearly identical** to that of class CommissionEmployee, except that BasePlusCommissionEmployee's ToString also formats the value of instance variable baseSalary as currency.

We literally **copied the code from class CommissionEmployee** and pasted it into class BasePlusCommissionEmployee, then modified class BasePlusCommissionEmployee to include a base salary and methods and properties that manipulate the base salary. This "**copy-and-paste**" approach is often **error prone and time consuming**. Worse yet, it can **spread many physical copies of the same code throughout a system, creating a code-maintenance nightmare**

=> **Inheritance** can solve this problem

Error-Prevention Tip

Copying and pasting code from one class to another can **spread errors across multiple source-code files**. To avoid duplicating code (and possibly errors) in situations where you want one class to “absorb” the members of another class, **use inheritance rather than the “copy-and-paste”** approach.

Software Engineering Observation

With inheritance, the **common members of all the classes** in the hierarchy are declared in a **base class**. When changes are required for these common features, you need to make the **changes only in the base class**—derived classes then inherit the changes. Without inheritance, changes would need to be made to all the source-code files that contain a copy of the code in question.

1.3 Creating a CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy

BasePlusCommissionEmployee extends class **CommissionEmployee**.

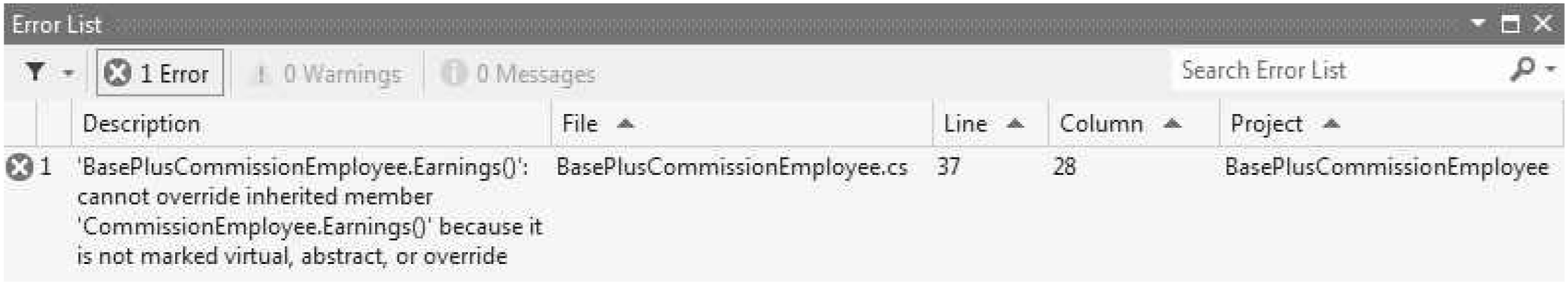
A BasePlusCommissionEmployee object is a CommissionEmployee, but class BasePlusCommissionEmployee also has instance variable **baseSalary**.

The **constructor** of class CommissionEmployee is **not inherited**.

```
// Fig. 11.8: BasePlusCommissionEmployee.cs
// BasePlusCommissionEmployee inherits from class CommissionEmployee.
using System;

public class BasePlusCommissionEmployee : CommissionEmployee
{
   private decimal baseSalary; // base salary per week

   // six-parameter derived class constructor
   // with call to base class CommissionEmployee constructor
   public BasePlusCommissionEmployee( string first, string last,
      string ssn, decimal sales, decimal rate, decimal salary )
      : base( first, last, ssn, sales, rate )
   {
      BaseSalary = salary; // validate base salary via property
   } // end six-parameter BasePlusCommissionEmployee constructor
```

```
// property that gets and sets
   // BasePlusCommissionEmployee's base salary
   public decimal BaseSalary
   {
      get
      {
         return baseSalary;
      } // end get
      set
      {
         if ( value >= 0 )
            baseSalary = value;
         else
            throw new ArgumentOutOfRangeException( "BaseSalary",
               value, "BaseSalary must be >= 0" );
      } // end set
   } // end property BaseSalary
```

```
// calculate earnings
   public override decimal Earnings()
   {
      // not allowed: commissionRate and grossSales private in base class
      return baseSalary + ( commissionRate * grossSales );
   } // end method Earnings

   // return string representation of BasePlusCommissionEmployee
   public override string ToString()
   {
      // not allowed: attempts to access private base class members
      return string.Format(
         "{0}: {1} {2}\n{3}: {4}\n{5}: {6:C}\n{7}: {8:F2}\n{9}: {10:C}",
         "base-salaried commission employee", firstName, lastName,
         "social security number", socialSecurityNumber,
         "gross sales", grossSales, "commission rate", commissionRate,
         "base salary", baseSalary );
   } // end method ToString
} // end class BasePlusCommissionEmployee
```


Error List
1 Error
0 Warnings
0 Messages
Search Error List
Description
File
Line
Column
Project
1
'BasePlusCommissionEmployee.Earnings()': cannot override inherited member 'CommissionEmployee.Earnings()' because it is not marked virtual, abstract, or override
BasePlusCommissionEmployee.cs
37
28
BasePlusCommissionEmployee

A Derived Class's Constructor Must Call Its Base Class's Constructor

Each derived-class constructor **must implicitly or explicitly call its base-class constructor** to ensure that the instance variables inherited from the base class are **initialized properly**.

In the **header of BasePlusCommissionEmployee's six-parameter** constructor invokes the CommissionEmployee's five-parameter constructor by using a constructor initializer with keyword **base to invoke the base-class constructor**.

A Derived Class's Constructor Must Call Its Base Class's Constructor

Continue

If BasePlusCommissionEmployee's constructor did not invoke CommissionEmployee's constructor **explicitly**, C# would attempt to invoke class CommissionEmployee's **parameterless or default constructor implicitly**—but the class does not have such a constructor, so the compiler would issue an **error**.

When a base class contains a parameter less constructor, you can use **base()** in the constructor initializer to call that constructor explicitly, but this is rarely done.

BasePlusCommissionEmployee Method Earnings

Compilation error indicating that we cannot **override** the base class's Earnings method because it was not explicitly "marked **virtual, abstract, or override**."

The **virtual and abstract** keywords indicate that a base-class method **can be overridden** in derived classes.

BasePlusCommissionEmployee Method Earnings

Continue

Adding **keyword virtual** to method Earnings' declaration and **recompile** => **other compilation errors**

Why? because base class CommissionEmployee's instance variables **commissionRate and grossSales are private**—derived class BasePlusCommissionEmployee's methods are not allowed to access base class CommissionEmployee's private instance variables.

Additional errors at BasePlusCommissionEmployee's **ToString method** for the same reason.

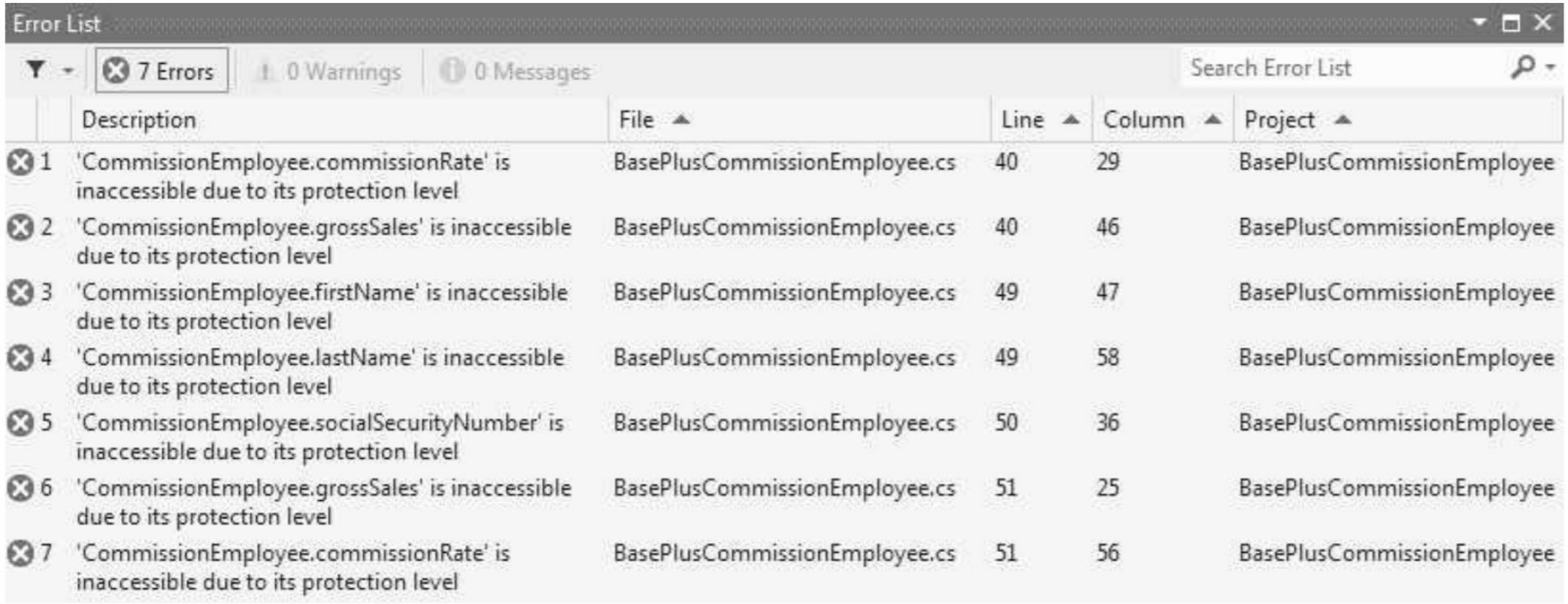


Fig. 11.9 | Compilation errors generated by `BasePlusCommissionEmployee` (Fig. 11.8) after declaring the `Earnings` method in Fig. 11.4 with keyword `virtual`.

1.4 CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy Using protected Instance Variables

To enable derived class to directly **access base-class instance variables** firstName, lastName, socialSecurityNumber, grossSales and commissionRate, we can declare those members as **protected** in the base class.

A base class's protected members are **inherited by all derived classes** of that base class.

Modify CommissionEmployee: declaring its instance variables firstName, lastName, socialSecurityNumber, grossSales and commissionRate as **protected rather than private**.

`public virtual decimal Earnings() =>` BasePlusCommissionEmployee **can override** the method

public vs. protected Data

Declaring **public** instance variables is **poor** software engineering, because it allows unrestricted access to the instance variables, greatly **increasing the chance of errors**.

With **protected** instance variables, the **derived class gets access** to the instance variables, but **classes that are not derived from the base class cannot access** its variables directly.

BasePlusCommissionEmployee.cs

```
// Fig. 11.10: BasePlusCommissionEmployee.cs
// BasePlusCommissionEmployee inherits from CommissionEmployee and has
// access to CommissionEmployee's protected members.
using System;

public class BasePlusCommissionEmployee : CommissionEmployee
{
   private decimal baseSalary; // base salary per week

   // six-parameter derived class constructor
   // with call to base class CommissionEmployee constructor
   public BasePlusCommissionEmployee( string first, string last,
      string ssn, decimal sales, decimal rate, decimal salary )
      : base( first, last, ssn, sales, rate )
   {
      BaseSalary = salary; // validate base salary via property
   } // end six-parameter BasePlusCommissionEmployee constructor
```

BasePlusCommissionEmployee.cs

```
        // property that gets and sets
        // BasePlusCommissionEmployee's base salary
        public decimal BaseSalary
        {
            get
            {
                return baseSalary;
            } // end get
            set
            {
                if ( value >= 0 )
                    baseSalary = value;
                else
                    throw new ArgumentOutOfRangeException( "BaseSalary",
                        value, "BaseSalary must be >= 0" );
            } // end set
        } // end property BaseSalary
```

BasePlusCommissionEmployee.cs

```
    // calculate earnings
    public override decimal Earnings()
    {
       return baseSalary + ( commissionRate * grossSales );
    } // end method Earnings

    // return string representation of BasePlusCommissionEmployee
    public override string ToString()
    {
       return string.Format(
          "{0}: {1} {2}\n{3}: {4}\n{5}: {6:C}\n{7}: {8:F2}\n{9}: {10:C}",
          "base-salaried commission employee", firstName, lastName,
          "social security number", socialSecurityNumber,
          "gross sales", grossSales, "commission rate", commissionRate,
          "base salary", baseSalary );
    } // end method ToString
} // end class BasePlusCommissionEmployee
```

BasePlusCommissionEmployeeTest.cs

```
// Fig. 11.11: BasePlusCommissionEmployeeTest.cs
// Testing class BasePlusCommissionEmployee.
using System;

public class BasePlusCommissionEmployeeTest
{
   public static void Main( string[] args )
   {
      // instantiate BasePlusCommissionEmployee object
      BasePlusCommissionEmployee basePlusCommissionEmployee =
         new BasePlusCommissionEmployee( "Bob", "Lewis",
         "333-33-3333", 5000.00M, .04M, 300.00M );
```

BasePlusCommissionEmployeeTest.cs

```
// display BasePlusCommissionEmployee's data
Console.WriteLine(
   "Employee information obtained by properties and methods: \n" );
Console.WriteLine( "First name is {0}",
   basePlusCommissionEmployee.FirstName );
Console.WriteLine( "Last name is {0}",
   basePlusCommissionEmployee.LastName );
Console.WriteLine( "Social security number is {0}",
   basePlusCommissionEmployee.SocialSecurityNumber );
Console.WriteLine( "Gross sales are {0:C}",
   basePlusCommissionEmployee.GrossSales );
Console.WriteLine( "Commission rate is {0:F2}",
   basePlusCommissionEmployee.CommissionRate );
Console.WriteLine( "Earnings are {0:C}",
   basePlusCommissionEmployee.Earnings() );
Console.WriteLine( "Base salary is {0:C}",
   basePlusCommissionEmployee.BaseSalary );
```

BasePlusCommissionEmployeeTest.cs

```
        basePlusCommissionEmployee.BaseSalary = 1000.00M; // set base salary

        Console.WriteLine( "\n{0}:\n\n{1}",
           "Updated employee information obtained by ToString",
           basePlusCommissionEmployee );
        Console.WriteLine( "earnings: {0:C}",
           basePlusCommissionEmployee.Earnings() );
     } // end Main
  } // end class BasePlusCommissionEmployeeTest
```

```
Employee information obtained by properties and methods:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales are $5,000.00
Commission rate is 0.04
Earnings are $500.00
Base salary is $300.00

Updated employee information obtained by ToString:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: $5,000.00
commission rate: 0.04
base salary: $1,000.00
earnings: $1,200.00
```

Problems with protected Instance Variables

Inheriting protected instance variables enables you to **directly access** the variables in the derived class **without invoking the set or get accessors** of the corresponding property => **violating encapsulation**.

In most cases, it's better to **use private instance variables** to encourage proper software engineering. Your code will be easier to **maintain, modify and debug**.

Problems with protected Instance Variables

Continue

Potential problems of protected instance variables

- A derived-class object can **assign an invalid value to the variable**. For example negative grossSales, … .
- With protected instance variables in the base class, we may need to **modify all the derived classes** of the base class if the **base-class implementation changes.** For example, changing firstName and lastName to first and last
  - In such a case, the software is said to be **fragile** or brittle, because a small change in the base class can **"break" derived-class implementation**.

Software Engineering Observation

You should be able to **change the base-class implementation** while still providing the same services to the derived classes

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 implementation**s.

1.5 CommissionEmployee–BasePlusCommissionEmployee Inheritance Hierarchy Using private Instance Variables

reexamine our hierarchy once more using the **best software engineering practices**. C

lass CommissionEmployee (Fig. 11.12) declares instance variables firstName, lastName, socialSecurityNumber, grossSales and commissionRate as **private** and **provides public properties** FirstName, LastName, SocialSecurityNumber, GrossSales and CommissionRate for manipulating these values.

Methods Earnings and ToString **use the class's properties** to obtain the values of its instance variables.

If we **decide to change the instance-variable names**, the Earnings and ToString declarations will not require modification—only the **bodies of the properties** that directly manipulate the instance variables will need to change.

These changes occur solely **within the base class**—no changes to the derived class are needed. **Localizing the effects of changes** like this is a good software engineering practice.

Derived class BasePlusCommissionEmployee (Fig. 11.13) inherits from CommissionEmployee's and can **access the private base-class members via the inherited public properties**.

CommissionEmployee.cs

```
// Fig. 11.12: CommissionEmployee.cs
// CommissionEmployee class represents a commission employee.
using System;

public class CommissionEmployee
{
   private string firstName;
   private string lastName;
   private string socialSecurityNumber;
   private decimal grossSales; // gross weekly sales
   private decimal commissionRate; // commission percentage
```

CommissionEmployee.cs

```
// five-parameter constructor
   public CommissionEmployee( string first, string last, string ssn,
      decimal sales, decimal rate )
   {
      // implicit call to object constructor occurs here
      firstName = first;
      lastName = last;
      socialSecurityNumber = ssn;
      GrossSales = sales; // validate gross sales via property
      CommissionRate = rate; // validate commission rate via property
   } // end five-parameter CommissionEmployee constructor
```

CommissionEmployee.cs

```
   // read-only property that gets commission employee's first name
      public string FirstName
      {
         get
         {
            return firstName;
         } // end get
      } // end property FirstName

      // read-only property that gets commission employee's last name
      public string LastName
      {
         get
         {
            return lastName;
         } // end get
      } // end property LastName
```

CommissionEmployee.cs

```
// read-only property that gets
// commission employee's social security number
public string SocialSecurityNumber
{
   get { return socialSecurityNumber; } // end get
}
public decimal GrossSales
{
   get { return grossSales; } // end get
   set {
          if ( value >= 0 )
             grossSales = value;
        else
             throw new ArgumentOutOfRangeException(
                        "GrossSales", value, "GrossSales must be >= 0" );
   } // end set
} // end property GrossSales
```

CommissionEmployee.cs

```
public decimal CommissionRate
{
   get
   { return commissionRate; } // end get
   set
   {
      if ( value > 0 && value < 1 )
         commissionRate = value;
      else
         throw new ArgumentOutOfRangeException( "CommissionRate",
            value, "CommissionRate must be > 0 and < 1" );
   } // end set
} // end property CommissionRate

public virtual decimal Earnings()
{
   return CommissionRate * GrossSales;
} // end method Earnings
```

CommissionEmployee.cs

```
      // return string representation of CommissionEmployee object
         public override string ToString()
         {
            return string.Format(
               "{0}: {1} {2}\n{3}: {4}\n{5}: {6:C}\n{7}: {8:F2}",
               "commission employee", FirstName, LastName,
               "social security number", SocialSecurityNumber,
               "gross sales", GrossSales, "commission rate",
               CommissionRate );
         } // end method ToString
      } // end class CommissionEmployee
```

BasePlusCommissionEmployee.cs

```
// Fig. 11.13: BasePlusCommissionEmployee.cs
// BasePlusCommissionEmployee inherits from CommissionEmployee and has
// access to CommissionEmployee's private data via
// its public properties.
using System;

public class BasePlusCommissionEmployee : CommissionEmployee
{
   private decimal baseSalary; // base salary per week

   // six-parameter derived class constructor
   // with call to base class CommissionEmployee constructor
   public BasePlusCommissionEmployee( string first, string last,
      string ssn, decimal sales, decimal rate, decimal salary )
      : base( first, last, ssn, sales, rate )
   {
      BaseSalary = salary; // validate base salary via property
   } // end six-parameter BasePlusCommissionEmployee constructor
```

BasePlusCommissionEmployee.cs

```
    // property that gets and sets
    // BasePlusCommissionEmployee's base salary
    public decimal BaseSalary
    {
       get
       {
          return baseSalary;
       } // end get
       set
       {
          if ( value >= 0 )
             baseSalary = value;
          else
             throw new ArgumentOutOfRangeException( "BaseSalary",
                value, "BaseSalary must be >= 0" );
       } // end set
    } // end property BaseSalary
```

BasePlusCommissionEmployee.cs

```
   // calculate earnings
   public override decimal Earnings()
   {
      return BaseSalary + base.Earnings();
   } // end method Earnings

   // return string representation of BasePlusCommissionEmployee
   public override string ToString()
   {
      return string.Format( "base-salaried {0}\nbase salary: {1:C}",
         base.ToString(), BaseSalary );
   } // end method ToString
} // end class BasePlusCommissionEmployee
```

BasePlusCommissionEmployee Method Earnings

Class BasePlusCommissionEmployee's Earnings method obtains the portion of the employee's earnings based on commission alone by **calling CommissionEmployee's Earnings** method with the expression **base.Earnings()**, then **adds the base salary** to this value.

Invoke an overridden base-class method from a derived class: place the keyword **base** and the member access **(.)** operator before the base-class method name.

=> **good software engineering** practice, by having BasePlusCommissionEmployee's Earnings method invoke CommissionEmployee's Earnings method to calculate part of a BasePlusCommissionEmployee object's earnings, we **avoid duplicating the code and reduce code-maintenance problems**.

Common Programming Error

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 base and the member access (.) operator when referencing the base class's method from the derived-class version causes the derived-class method to call itself, creating infinite recursion.

BasePlusCommissionEmployee Method ToString

Similarly, BasePlusCommissionEmployee's ToString method (Fig. 11.13) overrides CommissionEmployee's (Fig. 11.12) to return a string representation that's appropriate for a base-salaried commission employee. The new version creates part of BasePlusCommissionEmployee string representation (i.e., the string "commission employee" and the values of CommissionEmployee's private instance variables) by calling CommissionEmployee's ToString method with the expression base.ToString() (Fig. 11.13). The derived class's ToString method then outputs the remainder of the object's string representation (i.e., the base salary).

BasePlusCommissionEmployeeTest.cs

```
// Fig. 11.14: BasePlusCommissionEmployeeTest.cs
// Testing class BasePlusCommissionEmployee.
using System;

public class BasePlusCommissionEmployeeTest
{
   public static void Main( string[] args )
   {
      // instantiate BasePlusCommissionEmployee object
      BasePlusCommissionEmployee employee =
         new BasePlusCommissionEmployee( "Bob", "Lewis",
         "333-33-3333", 5000.00M, .04M, 300.00M );
```

BasePlusCommissionEmployeeTest.cs

```
      // display BasePlusCommissionEmployee's data
      Console.WriteLine(
         "Employee information obtained by properties and methods: \n" );
      Console.WriteLine( "First name is {0}", employee.FirstName );
      Console.WriteLine( "Last name is {0}", employee.LastName );
      Console.WriteLine( "Social security number is {0}",
         employee.SocialSecurityNumber );
      Console.WriteLine( "Gross sales are {0:C}", employee.GrossSales );
      Console.WriteLine( "Commission rate is {0:F2}",
         employee.CommissionRate );
      Console.WriteLine( "Earnings are {0:C}", employee.Earnings() );
      Console.WriteLine( "Base salary is {0:C}", employee.BaseSalary );

      employee.BaseSalary = 1000.00M; // set base salary

      Console.WriteLine( "\n{0}:\n\n{1}",
         "Updated employee information obtained by ToString", employee );
      Console.WriteLine( "earnings: {0:C}", employee.Earnings() );
   } // end Main
} // end class BasePlusCommissionEmployeeTest
```

```
Employee information obtained by properties and methods:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales are $5,000.00
Commission rate is 0.04
Earnings are $500.00
Base salary is $300.00

Updated employee information obtained by ToString:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: $5,000.00
commission rate: 0.04
base salary: $1,000.00
earnings: $1,200.00
```