7a

Object-Oriented Programming: Inheritance

OBJECTIVES

In this lecture you will learn:

- How inheritance promotes software reusability.
- The concepts of base classes and derived classes.
- To create a derived class that inherits attributes and behaviors from a base class.
- To use access modifier `protected` to give derived class methods access to base class members.
- To access base class members with `base`.
- How constructors are used in inheritance hierarchies.
- The methods of class `object`, the direct or indirect base class of all classes.

Outline

Outline

10.1 Introduction

Inheritance

- Software reusability
- Create new class from existing class
  - Absorb existing class's data and behaviors
  - Enhance with new capabilities
- Derived class extends base class
  - Derived class
    - More specialized group of objects
    - Behaviors inherited from base class
      - Can customize
    - Additional behaviors

10.1 Introduction (Cont.)

Class hierarchy

- Direct base class
  - Inherited explicitly (one level up hierarchy)
- Indirect base class
  - Inherited two or more levels up hierarchy
- Single inheritance
  - Inherits from one base class
- Multiple inheritance
  - Inherits from multiple base classes
    - C# does not support multiple inheritance

10.2 Base Classes and Derived Classes

- **Base classes and derived classes**
  - **Object of one class "is an" object of another class**
    - **Example: Rectangle is a quadrilateral.**
      - **Class `Rectangle` inherits from class `Quadrilateral`**
      - **`Quadrilateral`: base class**
      - **`Rectangle`: derived class**
  - **Base class typically represents larger set of objects than derived classes**
    - **Example:**
      - **Base class: `Vehicle`**
        - **Cars, trucks, boats, bicycles, …**
      - **Derived class: `Car`**
        - **Smaller, more-specific subset of vehicles**

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

Fig. 10.1 | Inheritance examples.

10.2 Base Classes and Derived Classes (Cont.)

- **Inheritance hierarchy**
  - **Inheritance relationships: tree-like hierarchy structure**
  - **Each class becomes**
    - **Base class**
      - **Supply members to other classes**

    OR

    - **Derived class**
      - **Inherit members from other classes**

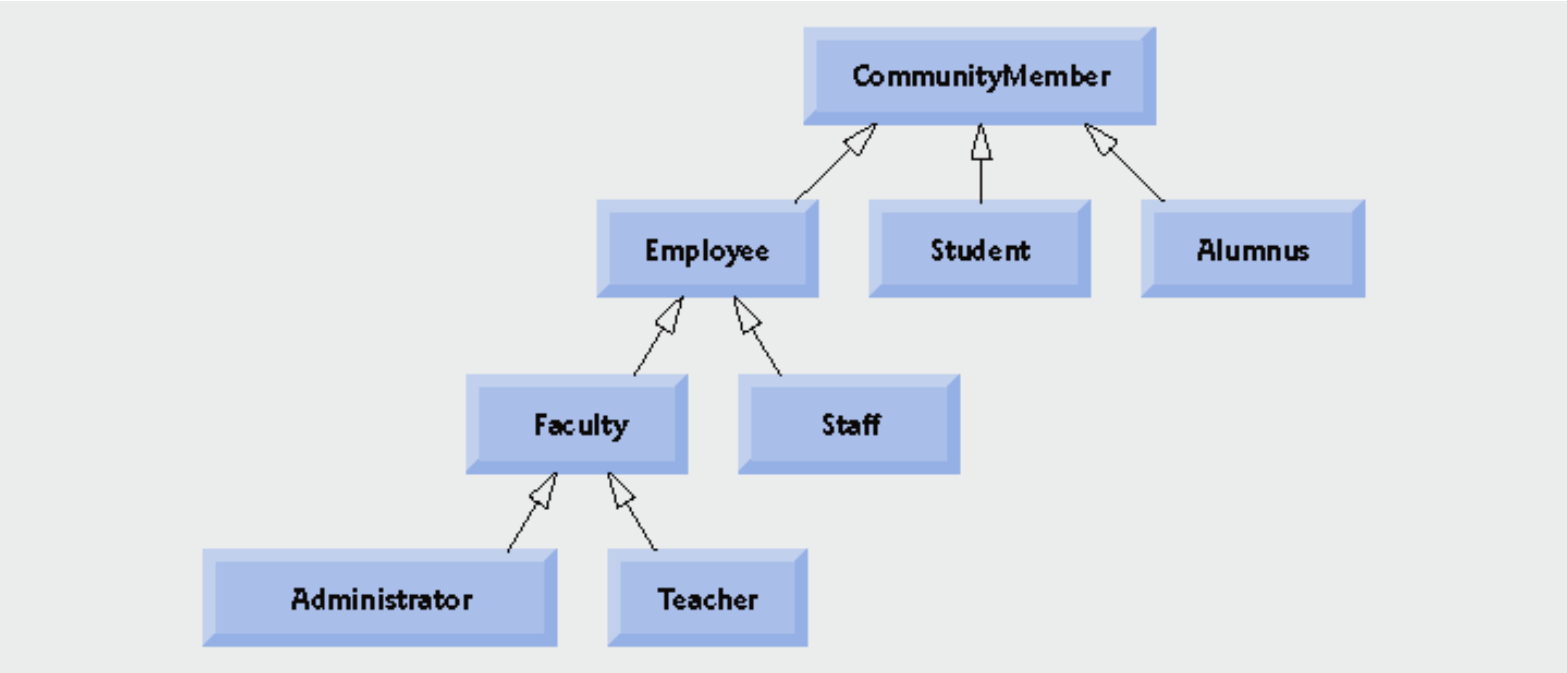

Fig. 10.2 | UML class diagram showing an inheritance hierarchy for university `CommunityMembers`.

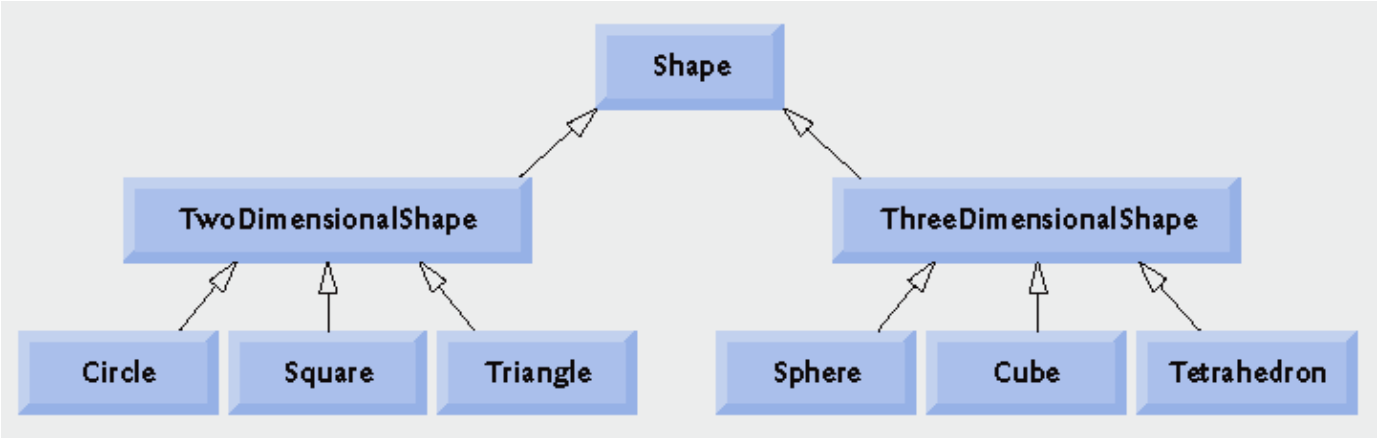

Fig. 10.3 | UML class diagram showing an inheritance hierarchy for Shapes.

10.3 protected Members

- `protected` access
  - Intermediate level of protection between `public` and `private`
  - `protected` members accessible by
    - Base class members
    - Derived class members
  - `protected internal` members accessible by
    - Same features as `protected`
    - Class members in the same assembly also have accessibility
  - Derived class access to base class member
    - Keyword `base` and a dot (.)

Software Engineering Observation 10.1

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 10.2

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 base 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.

10.4 Relationship between Base Classes and Derived Classes

Base class and derived class relationship

- **Example:** `CommissionEmployee/BasePlusCommissionEmployee` **inheritance hierarchy**
  - `CommissionEmployee`
    - **First name, last name, SSN, commission rate, gross sale amount**
  - `BasePlusCommissionEmployee`
    - **First name, last name, SSN, commission rate, gross sale amount**
    - **Base salary**

10.4.1 Creating and Using a `CommissionEmployee` Class

- Class `CommissionEmployee`
  - Extends class `object`
    - Colon "`:`"
    - Every class in C# extends an existing class
      - Except `object`
    - Every class inherits `object`'s methods
    - New class implicitly extends `object`
    - To override base class method use keyword `override` with the same signature (Overridden method must be declared `virtual`)
    - Constructors are not inherited
      - The first task of any derived class constructor is to call its direct base class's constructor

```
// Fig. 10.4: CommissionEmployee.cs
// CommissionEmployee class represents a commission employee.
public class CommissionEmployee : object
{
   private string firstName;
   private string lastName;
   private string socialSecurityNumber;
   private decimal grossSales; // gross weekly sales
   private decimal commissionRate; // commission percentage

   // 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
```

Class `CommissionEmployee` extends class `object`

Declare `private` instance variables

Implicit call to `object` constructor

Initialize instance variables

Invoke properties `GrossSales` and `CommissionRate` to validate data

`CommissionEmployee.cs`

(1 of 4)

```
31
32    // read-only property that gets commission employee's last name
33    public string LastName
34    {
35       get
36       {
37          return lastName;
38       } // end get
39    } // end property LastName
40
41    // read-only property that gets
42    // commission employee's social security number
43    public string SocialSecurityNumber
44    {
45       get
46       {
47          return socialSecurityNumber;
48       } // end get
49    } // end property SocialSecurityNumber
```

Outline

CommissionEmployee.cs

(2 of 4)

Outline

CommissionEmployee.cs

(3 of 4)

```
   // property that gets and sets commission employee's gross sales
   public decimal GrossSales
   {
      get
      {
         return grossSales;
      } // end get
      set
      {
         grossSales = ( value < 0 ) ? 0 : value;
      } // end set
   } // end property GrossSales

   // property that gets and sets commission employee's commission rate
   public decimal CommissionRate
   {
      get
      {
         return commissionRate;
      } // end get
      set
      {
         commissionRate = ( value > 0 && value < 1 ) ? value : 0;
      } // end set
   } // end property CommissionRate
```

```
76
77     // calculate commission employee's pay
78     public decimal Earnings()
79     {
80        return commissionRate * grossSales;
81     } // end method Earnings
82
83     // return string representation of CommissionEmployee object
84     public override string ToString()
85     {
86        return string.Format(
87           "{0}: {1} {2}\n{3}: {4}\n{5}: {6:C}\n{7}: {8:F2}",
88           "commission employee", FirstName, LastName,
89           "social security number", SocialSecurityNumber,
90           "gross sales", GrossSales, "commission rate", CommissionRate );
91     } // end method ToString
92  } // end class CommissionEmployee
```

Calculate earnings

Override method `ToString` of class `object`

Outline

`CommissionEmployee.cs`

(4 of 4)

Software Engineering Observation 10.3

The compiler sets the base class of a class to `object` when the class declaration does not explicitly extend a base class.

Common Programming Error 10.1

It is a compilation error to override a method with a different access modifier. Notice that overriding a method with a more restrictive access modifier would break the is-a relationship. If a `public` method could be overridden as a `protected` or `private` method, the derived class objects would not be able to respond to the same method calls as base class objects. Once a method is declared in a base class, the method must have the same access modifier for all that class's direct and indirect derived classes.

Outline

CommissionEmployeeTest.cs

(1 of 2)

```
// Fig. 10.5: 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( "{0} {1}", "First name is",
         employee.FirstName );
      Console.WriteLine( "{0} {1}", "Last name is",
         employee.LastName );
      Console.WriteLine( "{0} {1}", "Social security number is",
         employee.SocialSecurityNumber );
      Console.WriteLine( "{0} {1:C}", "Gross sales are",
         employee.GrossSales );
      Console.WriteLine( "{0} {1:F2}", "Commission rate is",
         employee.CommissionRate );
      Console.WriteLine( "{0} {1:C}", "Earnings are",
         employee.Earnings() );
```

Instantiate `CommissionEmployee` object

Use `CommissionEmployee`'s properties to retrieve and change the object's instance variable values

Outline

```
28
29        employee.GrossSales = 5000.00M; // set gross sales
30        employee.CommissionRate = .1M; // set commission rate
31
32        Console.WriteLine( "\n{0}:\n\n{1}",
33           "Updated employee information obtained by ToString", employee );
34        Console.WriteLine( "earnings: {0:C}", employee.Earnings() );
35     } // end Main
36  } // 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
```

Implicitly call the object's `ToString` method

CommissionEmployeeTest.cs

(2 of 2)

10.4.2 Creating a `BasePlusCommissionEmployee` Class without Using Inheritance

- Class `BasePlusCommissionEmployee`
  - Much of the code is similar to `CommissionEmployee`
    - `private` instance variables
    - `public` methods
    - Constructor
    - Properties
  - Additions
    - `private` instance variable `baseSalary`
    - `BaseSalary` property

Outline

BasePlusCommissionEmployee.cs

(1 of 5)

```
// Fig. 10.6: BasePlusCommissionEmployee.cs
// BasePlusCommissionEmployee class represents an employee that receives
// a base salary in addition to a commission.
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
```

Add instance variable `baseSalary`

Use property `BaseSalary` to validate data

```
25
26    // read-only property that gets
27    // base-salaried commission employee's first name
28    public string FirstName
29    {
30       get
31       {
32          return firstName;
33       } // end get
34    } // end property FirstName
35
36    // read-only property that gets
37    // base-salaried commission employee's last name
38    public string LastName
39    {
40       get
41       {
42          return lastName;
43       } // end get
44    } // end property LastName
45
46    // read-only property that gets
47    // base-salaried commission employee's social security number
48    public string SocialSecurityNumber
49    {
50       get
51       {
52          return socialSecurityNumber;
53       } // end get
54    } // end property SocialSecurityNumber
```

Outline

BasePlusCommissionEmployee.cs

(2 of 5)

```
55
56    // property that gets and sets
57    // base-salaried commission employee's gross sales
58    public decimal GrossSales
59    {
60       get
61       {
62          return grossSales;
63       } // end get
64       set
65       {
66          grossSales = ( value < 0 ) ? 0 : value;
67       } // end set
68    } // end property GrossSales
69
70    // property that gets and sets
71    // base-salaried commission employee's commission rate
72    public decimal CommissionRate
73    {
74       get
75       {
76          return commissionRate;
77       } // end get
78       set
79       {
80          commissionRate = ( value > 0 && value < 1 ) ? value : 0;
81       } // end set
82    } // end property CommissionRate
```

Outline

BasePlusCommissionEmployee.cs

(3 of 5)

Outline

BasePlusCommissionEmployee.cs

(4 of 5)

```
83
84    // property that gets and sets
85    // base-salaried commission employee's base salary
86    public decimal BaseSalary
87    {
88       get
89       {
90          return baseSalary;
91       } // end get
92       set
93       {
94          baseSalary = ( value < 0 ) ? 0 : value;
95       } // end set
96    } // end property BaseSalary
97
98    // calculate earnings
99    public decimal Earnings()
100   {
101      return BaseSalary + ( CommissionRate * GrossSales );
102   } // end method earnings
```

Validates data and sets instance variable `baseSalary`

Update method `Earnings` to calculate the earnings of a base-salaried commission employee

Outline

```
   // 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
```

Update method `ToString` to display base salary

BasePlusCommissionEmployee.cs

(5 of 5)

Outline

```
// Fig. 10.7: 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 base-salaried commission employee data
      Console.WriteLine(
         "Employee information obtained by properties and methods: \n" );
      Console.WriteLine( "{0} {1}", "First name is",
         employee.FirstName );
      Console.WriteLine( "{0} {1}", "Last name is",
         employee.LastName );
      Console.WriteLine( "{0} {1}", "Social security number is",
         employee.SocialSecurityNumber );
      Console.WriteLine( "{0} {1:C}", "Gross sales are",
         employee.GrossSales );
      Console.WriteLine( "{0} {1:F2}", "Commission rate is",
         employee.CommissionRate );
      Console.WriteLine( "{0} {1:C}", "Earnings are",
         employee.Earnings() );
      Console.WriteLine( "{0} {1:C}", "Base salary is",
         employee.BaseSalary );
```

Instantiate `BasePlusCommissionEmployee` object

Use `BasePluCommissionEmployee`'s properties to retrieve and change the object's instance variable values

BasePlusCommissionEmployeeTest.cs

(1 of 2)

Outline

```
31
32        employee.BaseSalary = 1000.00M; // set base salary
33
34        Console.WriteLine( "\n{0}:\n\n{1}",
35           "Updated employee information obtained by ToString", employee );
36        Console.WriteLine( "earnings: {0:C}", employee.Earnings() );
37     } // end Main
38  } // 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
```

Implicitly call the object's `ToString` method

BasePlusCommissionEmployeeTest.cs

(2 of 2)

Error-Prevention Tip 10.1

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 10.4

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 only to make the changes 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.

10.4.3 Creating a `CommissionEmployee-BasePlusCommiionEmployee` Inheritance Hierarchy

- Class `BasePlusCommissionEmployee2`
  - Extends class `CommissionEmployee`
  - Is a `CommissionEmployee`
  - Has instance variable `baseSalary`
  - Inherits `public` and `protected` members
  - Constructor not inherited

Outline

```
// Fig. 10.8: BasePlusCommissionEmployee2.cs
// BasePlusCommissionEmployee2 inherits from class CommissionEmployee.
public class BasePlusCommissionEmployee2 : CommissionEmployee
{
   private decimal baseSalary; // base salary per week

   // six-parameter derived class constructor
   // with call to base class CommissionEmployee constructor
   public BasePlusCommissionEmployee2( 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 BasePlusCommissionEmployee2 constructor

   // property that gets and sets
   // base-salaried commission employee's base salary
   public decimal BaseSalary
   {
      get
      {
         return baseSalary;
      } // end get
      set
      {
         baseSalary = ( value < 0 ) ? 0 : value;
      } // end set
   } // end property BaseSalary
```

Class `BasePluCommissionEmployee2` is a derived class of `CommissionEmployee`

Invoke the base class constructor using the base class constructor call syntax

BasePlusCommissionEmployee2.cs

(1 of 2)

Outline

BasePlusCommissionEmployee2.cs

```
   // 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 BasePlusCommissionEmployee2
   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 BasePlusCommissionEmployee2
```

Compiler generates errors because base class's instance variable are `private`

Error List

1 Error | 0 Warnings | 0 Messages

	Description	File	Line	Column	Project
1	'BasePlusCommissionEmployee2.Earnings()': cannot override inherited member 'CommissionEmployee.Earnings()' because it is not marked virtual, abstract, or override	BasePlusComr	31	28	BasePlusCommissionEmployee2

Error List

7 Errors | 0 Warnings | 0 Messages

	Description	File	Line	Column	Project
1	'CommissionEmployee.commissionRate' is inaccessible due to its protection level	BasePlusCc	34	29	BasePlusCommission Employee2
2	'CommissionEmployee.grossSales' is inaccessible due to its protection level	BasePlusCc	34	46	BasePlusCommission Employee2
3	'CommissionEmployee.firstName' is inaccessible due to its protection level	BasePlusCc	43	47	BasePlusCommission Employee2
4	'CommissionEmployee.lastName' is inaccessible due to its protection level	BasePlusCc	43	58	BasePlusCommission Employee2
5	'CommissionEmployee.socialSecurityNumber' is inaccessible due to its protection level	BasePlusCc	44	36	BasePlusCommission Employee2
6	'CommissionEmployee.grossSales' is inaccessible due to its protection level	BasePlusCc	45	25	BasePlusCommission Employee2
7	'CommissionEmployee.commissionRate' is inaccessible due to its protection level	BasePlusCc	45	56	BasePlusCommission Employee2

Fig. 10.9 | Compilation errors generated by `BasePlusCommissionEmployee2` (Fig. 10.8) after declaring the `Earnings` method in Fig. 10.4 with keyword `virtual`.

Common Programming Error 10.2

A compilation error occurs if a derived class constructor calls one of its base class constructors with arguments that do not match the number and types of parameters specified in one of the base class constructor declarations.

10.4.4 `CommissionEmployee-BasePlusCommissionEmployee` Inheritance Hierarchy Using `protected` Instance Variables

- Use `protected` instance variables
  - Enable class `BasePlusCommissionEmployee` to directly access base class instance variables
  - Base class's `protected` members are inherited by all derived classes of that base class

Outline

```
// Fig. 10.10: CommissionEmployee2.cs
// CommissionEmployee2 with protected instance variables.
public class CommissionEmployee2
{
   protected string firstName;
   protected string lastName;
   protected string socialSecurityNumber;
   protected decimal grossSales; // gross weekly sales
   protected decimal commissionRate; // commission percentage

   // five-parameter constructor
   public CommissionEmployee2( 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 CommissionEmployee2 constructor

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

Declare `protected` instance variables

CommissionEmployee2.cs

(1 of 4)

```

   // 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
```

Outline

Commission Employee2.cs

(2 of 4)

```
50
51    // property that gets and sets commission employee's gross sales
52    public decimal GrossSales
53    {
54       get
55       {
56          return grossSales;
57       } // end get
58       set
59       {
60          grossSales = ( value < 0 ) ? 0 : value;
61       } // end set
62    } // end property GrossSales
63
64    // property that gets and sets commission employee's commission rate
65    public decimal CommissionRate
66    {
67       get
68       {
69          return commissionRate;
70       } // end get
71       set
72       {
73          commissionRate = ( value > 0 && value < 1 ) ? value : 0;
74       } // end set
75    } // end property CommissionRate
```

Outline

CommissionEmployee2.cs

(3 of 4)

```
   // calculate commission employee's pay
   public virtual 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 CommissionEmployee2
```

Mark `Earnings` as **`virtual`** so the derived class can override the method

Outline

`CommissionEmployee2.cs`

(4 of 4)

Outline

BasePlusCommissionEmployee3.cs

(1 of 2)

```
// Fig. 10.11: BasePlusCommissionEmployee3.cs
// BasePlusCommissionEmployee3 inherits from CommissionEmployee2 and has
// access to CommissionEmployee2's protected members.
public class BasePlusCommissionEmployee3 : CommissionEmployee2
{
   private decimal baseSalary; // base salary per week

   // six-parameter derived class constructor
   // with call to base class CommissionEmployee constructor
   public BasePlusCommissionEmployee3( 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 BasePlusCommissionEmployee3 constructor

   // property that gets and sets
   // base-salaried commission employee's base salary
   public decimal BaseSalary
   {
      get
      {
         return baseSalary;
      } // end get
      set
      {
         baseSalary = ( value < 0 ) ? 0 : value;
      } // end set
   } // end property BaseSalary
```

Must call base class's constructor

Outline

BasePlusCommissionEmployee3.cs

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

   // return string representation of BasePlusCommissionEmployee3
   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 BasePlusCommissionEmployee3
```

Overrides base class's `Earnings` method

Directly access base class's `protected` instance variables

Outline

```
// Fig. 10.12: BasePlusCommissionEmployeeTest3.cs
// Testing class BasePlusCommissionEmployee3.
using System;

public class BasePlusCommissionEmployeeTest3
{
   public static void Main( string[] args )
   {
      // instantiate BasePlusCommissionEmployee3 object
      BasePlusCommissionEmployee3 basePlusCommissionEmployee =
         new BasePlusCommissionEmployee3( "Bob", "Lewis",
         "333-33-3333", 5000.00M, .04M, 300.00M );

      // display base-salaried commission employee data
      Console.WriteLine(
         "Employee information obtained by properties and methods: \n" );
      Console.WriteLine( "{0} {1}", "First name is",
         basePlusCommissionEmployee.FirstName );
      Console.WriteLine( "{0} {1}", "Last name is",
         basePlusCommissionEmployee.LastName );
      Console.WriteLine( "{0} {1}", "Social security number is",
         basePlusCommissionEmployee.SocialSecurityNumber );
      Console.WriteLine( "{0} {1:C}", "Gross sales are",
         basePlusCommissionEmployee.GrossSales );
      Console.WriteLine( "{0} {1:F2}", "Commission rate is",
         basePlusCommissionEmployee.CommissionRate );
      Console.WriteLine( "{0} {1:C}", "Earnings are",
         basePlusCommissionEmployee.Earnings() );
      Console.WriteLine( "{0} {1:C}", "Base salary is",
         basePlusCommissionEmployee.BaseSalary );
```

Instantiate `BasePlusCommissionEmployee3` object

Use `BasePluCommissionEmployee3`'s properties to retrieve and change the object's instance variable values

BasePlusCommission EmployeeTest3.cs

(1 of 2)

Outline

BasePlusCommissionEmployeeTest3.cs

(2 of 2)

```
      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 BasePlusCommissionEmployeeTest3
```

Implicitly call the object's `ToString` method

```
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
```

10.4.4 `CommissionEmployee-BasePlusCommissionEmployee` Inheritance Hierarchy Using protected Instance Variables (Cont.)

- Using `protected` instance variables
  - Advantages
    - Derived classes can modify values directly
    - Slight increase in performance
      - Avoid set/get accessors call overhead
  - Disadvantages
    - No validity checking
      - Derived class can assign illegal value
    - Implementation dependent
      - Derived class methods more likely dependent on base class implementation
      - Derived class implementation changes may result in derived class modifications
        - Fragile (brittle) software

Software Engineering Observation 10.5

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.

10.4.5 `CommissionEmployee-BasePlusCommissionEmployee` Inheritance Hierarchy Using `private` Instance Variables

- **Reexamine hierarchy**
  - **Use the best software engineering practice**
    - **Declare instance variables as `private`**
    - **Provide `public` *get* and *set* accessors**
    - **Use *get* accessor to obtain values of instance variables**

Outline

```
// Fig. 10.13: CommissionEmployee3.cs
// CommissionEmployee3 class represents a commission employee.
public class CommissionEmployee3
{
   private string firstName;
   private string lastName;
   private string socialSecurityNumber;
   private decimal grossSales; // gross weekly sales
   private decimal commissionRate; // commission percentage

   // five-parameter constructor
   public CommissionEmployee3( 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 CommissionEmployee3 constructor

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

Declare `private` instance variables

CommissionEmployee3.cs

(1 of 2)

```
31
32    // read-only property that gets commission employee's last name
33    public string LastName
34    {
35       get
36       {
37          return lastName;
38       } // end get
39    } // end property LastName
40
41    // read-only property that gets
42    // commission employee's social security number
43    public string SocialSecurityNumber
44    {
45       get
46       {
47          return socialSecurityNumber;
48       } // end get
49    } // end property SocialSecurityNumber
```

Outline

Commission Employee3.cs

(2 of 4)

```
50
51    // property that gets and sets commission employee's gross sales
52    public decimal GrossSales
53    {
54       get
55       {
56          return grossSales;
57       } // end get
58       set
59       {
60          grossSales = ( value < 0 ) ? 0 : value;
61       } // end set
62    } // end property GrossSales
63
64    // property that gets and sets commission employee's commission rate
65    public decimal CommissionRate
66    {
67       get
68       {
69          return commissionRate;
70       } // end get
71       set
72       {
73          commissionRate = ( value > 0 && value < 1 ) ? value : 0;
74       } // end set
75    } // end property CommissionRate
```


CommissionEmployee3.cs

(3 of 4)

Outline

```
   // calculate commission employee's pay
   public virtual 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 CommissionEmployee3
```

CommissionEmployee3.cs

Use properties to obtain the values of instance variables

Outline

```
// Fig. 10.14: BasePlusCommissionEmployee4.cs
// BasePlusCommissionEmployee4 inherits from CommissionEmployee3 and has
// access to CommissionEmployee3's private data via
// its public properties.
public class BasePlusCommissionEmployee4 : CommissionEmployee3
{
   private decimal baseSalary; // base salary per week

   // six-parameter derived class constructor
   // with call to base class CommissionEmployee3 constructor
   public BasePlusCommissionEmployee4( 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 BasePlusCommissionEmployee4 constructor

   // property that gets and sets
   // base-salaried commission employee's base salary
   public decimal BaseSalary
   {
      get
      {
         return baseSalary;
      } // end get
      set
      {
         baseSalary = ( value < 0 ) ? 0 : value;
      } // end set
   } // end property BaseSalary
```

Inherits from `CommissionEmployee3`

BasePlusCommissionEmployee4.cs

(1 of 2)

Outline

```
31
32    // calculate earnings
33    public override decimal Earnings()
34    {
35       return BaseSalary + base.Earnings();
36    } // end method Earnings
37
38    // return string representation of BasePlusCommissionEmployee4
39    public override string ToString()
40    {
41       return string.Format( "{0} {1}\n{2}: {3:C}",
42          "base-salaried", base.ToString(), "base salary", BaseSalary );
43    } // end method ToString
44 } // end class BasePlusCommissionEmployee4
```

BasePlusCommissionEmployee4.cs

Invoke an overridden base class method from a derived class

Use properties to obtain the values of instance variables

Invoke an overridden base class method from a derived class

Common Programming Error 10.3

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 `base` and the dot (.) operator when referencing the base class's method causes the derived class method to call itself, creating an error called infinite recursion. Recursion, used correctly, is a powerful capability, as you learned in Section 7.13, Recursion.

Common Programming Error 10.4

The use of "chained" `base` references to refer to a member (a method, property or variable) several levels up the hierarchy—as in `base.base.Earnings()`—is a compilation error.

Outline

BasePlusCommissionEmployeeTest4.cs

(1 of 2)

```
// Fig. 10.15: BasePlusCommissionEmployeeTest4.cs
// Testing class BasePlusCommissionEmployee4.
using System;

public class BasePlusCommissionEmployeeTest4
{
   public static void Main( string[] args )
   {
      // instantiate BasePlusCommissionEmployee3 object
      BasePlusCommissionEmployee4 employee =
         new BasePlusCommissionEmployee4( "Bob", "Lewis",
         "333-33-3333", 5000.00M, .04M, 300.00M );

      // display base-salaried commission employee data
      Console.WriteLine(
         "Employee information obtained by properties and methods: \n" );
      Console.WriteLine( "{0} {1}", "First name is",
         employee.FirstName );
      Console.WriteLine( "{0} {1}", "Last name is",
         employee.LastName );
      Console.WriteLine( "{0} {1}", "Social security number is",
         employee.SocialSecurityNumber );
      Console.WriteLine( "{0} {1:C}", "Gross sales are",
         employee.GrossSales );
      Console.WriteLine( "{0} {1:F2}", "Commission rate is",
         employee.CommissionRate );
      Console.WriteLine( "{0} {1:C}", "Earnings are",
         employee.Earnings() );
      Console.WriteLine( "{0} {1:C}", "Base salary is",
         employee.BaseSalary );
```

Create `BasePlusCommissionEmployee4` object.

Use inherited properties to access inherited `private` instance variables

Use `BasePlusCommissionEmployee4` properties to access `private` instance variable.

```
31
32        employee.BaseSalary = 1000.00M; // set base salary
33
34        Console.WriteLine( "\n{0}:\n\n{1}",
35           "Updated employee information obtained by ToString", employee );
36        Console.WriteLine( "earnings: {0:C}", employee.Earnings() );
37     } // end Main
38 } // end class BasePlusCommissionEmployeeTest4
```

```
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
```


BasePlusCommissionEmployeeTest4.cs

10.5 Constructors in Derived Classes

- **Instantiating derived class object**
  - **Chain of constructor calls**
    - **Derived class constructor invokes base class constructor**
      - **Implicitly or explicitly**
    - **Base of inheritance hierarchy**
      - **Last constructor called in chain is `object`'s constructor**
      - **Original derived class constructor's body finishes executing last**
      - **Example: `CommissionEmployee3-BasePlusCommissionEmployee4` hierarchy**
        - **`CommissionEmployee3` constructor called second last (last is `Object` constructor)**
        - **`CommissionEmployee3` constructor's body finishes execution second (first is `Object` constructor's body)**

Software Engineering Observation 10.6

When an application creates a derived class object, the derived class constructor immediately calls the base class constructor (explicitly, via `base`, or implicitly). The base class constructor's body executes to initialize the base class's instance variables that are part of the derived class object, then the derived class constructor's body executes to initialize the derived class-only instance variables. Even if a constructor does not assign a value to an instance variable, the variable is still initialized to its default value (i.e., `0` for simple numeric types, `false` for `bools` and `null` for references).

Outline

CommissionEmployee4.cs

(1 of 4)

```
// Fig. 10.16: CommissionEmployee4.cs
// CommissionEmployee4 class represents a commission employee.
using System;

public class CommissionEmployee4
{
   private string firstName;
   private string lastName;
   private string socialSecurityNumber;
   private decimal grossSales; // gross weekly sales
   private decimal commissionRate; // commission percentage

   // five-parameter constructor
   public CommissionEmployee4( 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

      Console.WriteLine( "\nCommissionEmployee4 constructor:\n" + this );
   } // end five-parameter CommissionEmployee4 constructor
```

Constructor outputs message to demonstrate method call order.

```

   // 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
```

Outline

Commission Employee4.cs

(2 of 4)

```
54
55    // property that gets and sets commission employee's gross sales
56    public decimal GrossSales
57    {
58       get
59       {
60          return grossSales;
61       } // end get
62       set
63       {
64          grossSales = ( value < 0 ) ? 0 : value;
65       } // end set
66    } // end property GrossSales
67
68    // property that gets and sets commission employee's commission rate
69    public decimal CommissionRate
70    {
71       get
72       {
73          return commissionRate;
74       } // end get
75       set
76       {
77          commissionRate = ( value > 0 && value < 1 ) ? value : 0;
78       } // end set
79    } // end property CommissionRate
```


CommissionEmployee4.cs

(3 of 4)

```
   // calculate commission employee's pay
   public virtual 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 CommissionEmployee4
```

Outline

Commission Employee4.cs

(4 of 4)

Outline

BasePlusCommissionEmployee5.cs

(1 of 2)

```
// Fig. 10.17: BasePlusCommissionEmployee5.cs
// BasePlusCommissionEmployee5 class declaration.
using System;

public class BasePlusCommissionEmployee5 : CommissionEmployee4
{
   private decimal baseSalary; // base salary per week

   // six-parameter derived class constructor
   // with call to base class CommissionEmployee4 constructor
   public BasePlusCommissionEmployee5( 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

      Console.WriteLine(
         "\nBasePlusCommissionEmployee5 constructor:\n" + this );
   } // end six-parameter BasePlusCommissionEmployee5 constructor

   // property that gets and sets
   // base-salaried commission employee's base salary
   public decimal BaseSalary
   {
      get
      {
         return baseSalary;
      } // end get
```

Constructor outputs message to demonstrate method call order.

```
      set
      {
         baseSalary = ( value < 0 ) ? 0 : value;
      } // end set
   } // end property BaseSalary

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

   // return string representation of BasePlusCommissionEmployee5
   public override string ToString()
   {
      return string.Format( "{0} {1}\n{2}: {3:C}",
         "base-salaried", base.ToString(), "base salary", BaseSalary );
   } // end method ToString
} // end class BasePlusCommissionEmployee5
```

Outline

BasePlusCommissionEmployee5.cs

(2 of 2)

Outline

```
// Fig. 10.18: ConstructorTest.cs
// Display order in which base class and derived class constructors
// are called.
using System;

public class ConstructorTest
{
   public static void Main( string[] args )
   {
      CommissionEmployee4 employee1 = new CommissionEmployee4( "Bob",
         "Lewis", "333-33-3333", 5000.00M, .04M );

      Console.WriteLine();
      BasePlusCommissionEmployee5 employee2 =
         new BasePlusCommissionEmployee5( "Lisa", "Jones",
         "555-55-5555", 2000.00M, .06M, 800.00M );

      Console.WriteLine();
      BasePlusCommissionEmployee5 employee3 =
         new BasePlusCommissionEmployee5( "Mark", "Sands",
         "888-88-8888", 8000.00M, .15M, 2000.00M );
   } // end Main
} // end class ConstructorTest
```

Instantiate `CommissionEmployee4` object

Instantiate two `BasePlusCommissionEmployee5` objects to demonstrate order of derived class and base class constructor method calls.

ConstructorTest.cs

(1 of 2)

```
CommissionEmployee4 constructor:
commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: $5,000.00
commission rate: 0.04

CommissionEmployee4 constructor:
base-salaried commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: $2,000.00
commission rate: 0.06
base salary: $0.00

BasePlusCommissionEmployee5 constructor:
base-salaried commission employee: Lisa Jones
social security number: 555-55-5555
gross sales: $2,000.00
commission rate: 0.06
base salary: $800.00

CommissionEmployee4 constructor:
base-salaried commission employee: Mark Sands
social security number: 888-88-8888
gross sales: $8,000.00
commission rate: 0.15
base salary: $0.00

BasePlusCommissionEmployee5 constructor:
base-salaried commission employee: Mark Sands
social security number: 888-88-8888
gross sales: $8,000.00
commission rate: 0.15
base salary: $2,000.00
```


ConstructorTest.cs

10.6 Software Engineering with Inheritance

- **Customizing existing software**
  - **Inherit from existing classes**
    - **Include additional members**
    - **Redefine base class members**
    - **No direct access to base class's source code**
      - **Link to object code**
  - **Independent software vendors (ISVs)**
    - **Develop proprietary code for sale/license**
      - **Available in object-code format**
    - **Users derive new classes**
      - **Without accessing ISV proprietary source code**

Software Engineering Observation 10.7

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 understand how the class is implemented. They may, for example, want to ensure that they are extending a class that performs well and is implemented securely.

Software Engineering Observation 10.8

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 members 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.

Software Engineering Observation 10.9

Declaring a derived class does not affect its base class's source code. Inheritance preserves the integrity of the base class.

Software Engineering Observation 10.10

Just as designers of non-object-oriented systems should avoid method proliferation, designers of object-oriented systems should avoid class proliferation. Such proliferation creates management problems and can hinder software reusability, because in a huge class library it becomes difficult for a client to locate the most appropriate classes. The alternative is to create fewer classes that provide more substantial functionality, but such classes might prove cumbersome.

Performance Tip 10.1

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

10.7 Class `object`

- Class `object` methods
  - `Equals`
  - `Finalize`
  - `GetHashCode`
  - `GetType`
  - `MemberwiseClone`
  - `ReferenceEquals`
  - `ToString`

Method	Description
`Equals`	This method compares two objects for equality and returns true if they are equal and false otherwise. The method takes any object as an argument. When objects of a particular class must be compared for equality, the class should override method Equals to compare the contents of the two objects. The method's implementation should meet the following requirements: • It should return `false` if the argument is null. • It should return true if an object is compared to itself, as in `object1.Equals( object1 )`. • It should return true only if both `object1.Equals( object2 )` and `object2.Equals( object1 )` would return true. • For three objects, if `object1.Equals( object2 )` returns true and `object2.Equals( object3 )` returns true, then `object1.Equals( object3 )` should also return true. • A class that overrides the method `Equals` should also override the method `GetHashCode` to ensure that equal objects have identical hashcodes. The default Equals implementation determines only whether two references *refer to the same object* in memory.
`Finalize`	This method cannot be explicitly declared or called. When a class contains a destructor, the compiler implicitly renames it to override the `protected` method `Finalize`, which is called only by the garbage collector before it reclaims an object's memory. The garbage collector is not guaranteed to reclaim an object, thus it is not guaranteed that an object's `Finalize` method will execute. When a derived class's `Finalize` method executes, it performs its task, then invokes the base class's `Finalize` method. `Finalize`'s default implementation is a placeholder that simply invokes the base class's `Finalize` method.

Fig. 10.19 | `Object` methods that are inherited directly or indirectly by all classes. (Part 1 of 2)

Method	Description
`GetHashCode`	A hashtable is a data structure that relates one object, called the key, to another object, called the value. We discuss `Hashtable` in Chapter 27, Collections. When initially inserting a value into a hashtable, the key's `GetHashCode` method is called. The hashcode value returned is used by the hashtable to determine the location at which to insert the corresponding value. The key's hashcode is also used by the hashtable to locate the key's corresponding value.
`GetType`	Every object knows its own type at execution time. Method `GetType` (used in Section 11.5) returns an object of class `Type` (namespace `System`) that contains information about the object's type, such as its class name (obtained from `Type` property `FullName`).
`MemberwiseClone`	This protected method, which takes no arguments and returns an object reference, makes a copy of the object on which it is called. The implementation of this method performs a shallow copy—instance variable values in one object are copied into another object of the same type. For reference types, only the references are copied.
`ReferenceEquals`	This static method takes two object arguments and returns true if two objects are the same instance or if they are null references. Otherwise, it returns false.
`ToString`	This method (introduced in Section 7.4) returns a string representation of an object. The default implementation of this method returns the namespace followed by a dot and the class name of the object's class.

Fig. 10.19 | `Object` methods that are inherited directly or indirectly by all classes. (Part 2 of 2)

10.8 Summary

1. **A derived type which inherits from a base type implicitly has all inheritable members of the base type.** If a base type has a member M, then a derived type has a member M as well.
2. The following are not inherited
   - constructors and destructors
   - **`private`** members(*)
   - Static members
   - explicitly implemented methods of interfaces (**)

(*) **`private`** members are **not directly accessible** outside the base class. Therefore, the **`private`** members of a base class are **inheritable** only when the derived class falls implicitly or explicitly in the accessibility domain of the base class.

10.8 Summary

A `private` member is implicitly inside the accessible domain of the base class when it is used inside a method or a property accessible outside the base class.

`private` datamembers in the base class are implicitly accessible in the derived class by calling a constructor of the base class, through properties or methods accessible outside the base class.

A `private` member is explicitly inside the accessible domain of the base class when the derived class is embedded inside the base class.

```
1  class A
2  {
3    private int x;
4    private class B : A
5    {
6        void M() { x = 1;}
7    }
8  }
```

Compare with

```
1  class A
2  {
3    private int x;
4    private class B
5    {
6        void M() {
7            A a = new A();
8            a.x = 1;
9    }
10 }
```

Правила за писане на конструктори на класове в йерархия на наследственост

1. **Пишем базовия клас на йерархията от наследственост по правилата за моделиране на клас, дадени в предишната лекция в следната последователност**
   - **A. `private` клас данни**
   - **B. SET и GET свойства / индексатори за всички клас данни**
   - **C. Конструктор за общо ползване (извиква set свойство за валидиране на данните)**
   - **D. Конструктор по подразбиране (извиква конструктора за общо ползване)**
   - **E. Конструктор за копиране (извиква конструктора за общо ползване)**
   - **F. Всички останали клас методи**
   - **G. `string ToString()` метод**

Правила за писане базов клас- декларираме на данните

```
// Fig. 9.15a: CommissionEmployee4.java
// CommissionEmployee4 class represents a commission employee.

public class CommissionEmployee4
{
   private string firstName;
   private string lastName;
   private string socialSecurityNumber;
   private double grossSales; // gross weekly sales
   private double commissionRate; // commission percentage
```

Правила за писане базов клас- SET и GET методи

```
// first name property
  public string FirstName{
     set  {
              firstName = (value!=null)? value:"";
           } // end   set
     get  {
           return firstName;
          } // end  get
   }
  // last name property
  public string LastName{
     set  {
              lastName = (value!=null)? value:"";
           } // end   set
     get  {
           return lastName;
          } // end  get
   }
  // …  and so …. And so on…
```

Правила за писане базов клас-конструктор за общо ползване

```
private string firstName;
private string lastName;
private string socialSecurityNumber;
private double grossSales;          // gross weekly sales
private double commissionRate;      // commission percentage
// five-argument constructor
public CommissionEmployee4( String first, String last, String ssn,
                                                double sales, double rate )
{
   // implicit call to Object constructor occurs here
   firstName = first;
   lastName = last;
   socialSecurityNumber = ssn;
   setGrossSales( sales ); // validate and store gross sales
   setCommissionRate( rate ); // validate and store commission rate

   System.out.printf(
      "\nCommissionEmployee4 constructor:\n%s\n", this );
} // end five-argument CommissionEmployee4 constructor
```

Правила за писане базов клас-конструктори по подразбиране и за копиране

```
// default constructor
 public CommissionEmployee4( ): this("", "", "", 0.0, 0.0)
 {

 } // end five-argument CommissionEmployee4 constructor

 // copy  constructor
 public CommissionEmployee4(CommissionEmployee4 c )
 : this(c.firstName, c.lastName, c.socialSecurityNumber,
                                c.grossSales,c.commissionRate);
 {
 } // end five-argument CommissionEmployee4 constructor
```

Правила за писане базов клас- други методи на класа и *toString()* метода

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

// return String representation of CommissionEmployee4 object
  public override string ToString()
  {
     return String.Format("{0}:{1} {2}\n{3}:{4}\n{5}: {6:F2}\n{7}: {8:F2} ",
        "commission employee", FirstName, LastName,
        "social security number", SocialSecurityNumber,
        "gross sales", GrossSales),
        "commission rate", CommissionRate );
  } // end method toString
```

Правила за писане на конструктори на класове в йерархия на наследственост

2. **Пишем всеки от производните класове на йерархията от наследственост по следните правила в следната последователност**
   - **A. Декларира всички `private` клас данни, които са различни от онаследените**
   - **B. SET и GET свойства за всички клас данни, които са различни от онаследените**
   - **C. Конструктор за общо ползване**
     - **a. Извиква ЯВНО конструкторът за общо ползване на директния базов клас и инициализира ВСИЧКИ онаследени данни**
     - **b. извиква set методите за данните, които са различни от онаследените**
   - **D. Конструктор по подразбиране (извиква конструктора за общо ползване на текущия клас, задава стойности по подразбиране за всички данни – онаследени и тези, дефинирани в текущия клас)**
   - **E. Конструктор за копиране (извиква конструктора за общо ползване на текущия клас , използва GET методи за онаследени клас данни)**
   - **F. Всички останали клас методи**
   - **G. `string ToString()` метод**

Правила за писане производен клас- - декларираме на новите данни

```
// Fig. 9.16a: BasePlusCommissionEmployee5.java
// Modified BasePlusCommissionEmployee5 class declaration.

public class BasePlusCommissionEmployee5 : CommissionEmployee4
{
   // Тук не се декларират отново данните, които са онаследени!
   private double baseSalary; // base salary per week
```

Правила за писане производен клас SET и GET свойства само за новите данни

```
// base salary property
  public double BaseSalary
  {
     set
     {
        baseSalary = ( value < 0.0 ) ? 0.0 : value;
     } // end set
     get
     {
        return baseSalary;
      } // end   get
   }
```

Правила за писане производен клас Конструктор за общо ползване

```
// six-argument constructor- инициализира ВСИЧКИ данни
 public BasePlusCommissionEmployee5( String first, String last,
    String ssn, double sales, double rate, double salary )
                        : base( first, last, ssn, sales, rate )
                       // инициализира онаследените
   {
       // следва инициализация на всички данни, които не са онаследени
        BaseSalary = salary ; // validate and store base salary
    } // end six-argument BasePlusCommissionEmployee5 constructor
```

Правила за писане производен клас Конструктори по подразбиране и за копиране

```
// default constructor
public BasePlusCommissionEmployee5( )
 : this("", "", "", 0.0, 0.0, 0.0)
 {

 } // end six-argument BasePlusCommissionEmployee5 constructor

  // default constructor
public BasePlusCommissionEmployee5( BasePlusCommissionEmployee5 b)
 : this(  b.FirstName, b.LastName,
          b.SocialSecurityNumber,
          b.GrossSales, b.CommissionRate, b.baseSalary)
 {

 } // end six-argument BasePlusCommissionEmployee5 constructor
```

Правила за писане производен клас другиметоди на класа и *ToString()* метода

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

 // return string representation of BasePlusCommissionEmployee5
 public override string ToString()
 {
    return String.format( “{0} {1}\n{2}: {3:F2}”, "base-salaried",
       base.ToString(), "base salary",  BaseSalary );
 } // end method ToString
```