Chapter 12
OOP: Polymorphism, Interfaces and Operator Overloading

Visual C# 2012 How to Program

OBJECTIVES

In this chapter you'll learn:

- How polymorphism enables you to "program in the general" and make systems extensible.
- To use overridden methods to effect polymorphism.
- To create abstract classes and methods.
- To determine an object's type at execution time.
- To create `sealed` methods and classes.
- To declare and implement interfaces.
- To overload operators to enable them to manipulate objects.

12.1 Introduction
12.2 Polymorphism Examples
12.3 Demonstrating Polymorphic Behavior
12.4 Abstract Classes and Methods
12.5 Case Study: Payroll System Using Polymorphism
12.5.1 Creating Abstract Base Class `Employee`
12.5.2 Creating Concrete Derived Class `SalariedEmployee`
12.5.3 Creating Concrete Derived Class `HourlyEmployee`
12.5.4 Creating Concrete Derived Class `CommissionEmployee`
12.5.5 Creating Indirect Concrete Derived Class `BasePlusCommission-Employee`
12.5.6 Polymorphic Processing, Operator `is` and Downcasting
12.5.7 Summary of the Allowed Assignments Between Base-Class and Derived-Class Variables

12.1 Introduction

- **Polymorphism** enables you to write apps that process objects that share the same base class in a class hierarchy as if they were all objects of the base class.
- Polymorphism promotes extensibility.

12.2 Polymorphism Examples

- If class `Rectangle` is derived from class `Quadrilateral`, then a `Rectangle` is a more specific version of a `Quadrilateral`.
- Any operation that can be performed on a `Quadrilateral` object can also be performed on a `Rectangle` object.
- These operations also can be performed on other `Quadrilateral`s, such as `Square`s, `Parallelogram`s and `Trapezoid`s.
- The polymorphism occurs when an app invokes a method through a base-class variable.

12.2 Polymorphism Examples (Cont.)

- As another example, suppose we design a video game that manipulates objects of many different types, including objects of classes `Martian`, `Venusian`, `Plutonian`, `SpaceShip` and `LaserBeam`.
- Each class inherits from the common base class `SpaceObject`, which contains method `Draw`.
- A screen-manager app maintains a collection (e.g., a `SpaceObject` array) of references to objects of the various classes.
- To refresh the screen, the screen manager periodically sends each object the same message—namely, `Draw`, while object responds in a unique way.

Software Engineering Observation 12.1

Polymorphism promotes extensibility: Software that invokes polymorphic behavior is independent of the object types to which messages are sent. New object types that can respond to existing method calls can be incorporated into a system without requiring modification of the base system. Only client code that instantiates new objects must be modified to accommodate new types.

12.3 Demonstrating Polymorphic Behavior

- In a method call on an object, the type of the *actual referenced object*, not the type of the *reference*, determines which method is called.
- An object of a derived class can be treated as an object of its base class.
- A base-class object is not an object of any of its derived classes.
- The *is-a* relationship applies from a derived class to its direct and indirect base classes, but not vice versa.

12.3 Demonstrating Polymorphic Behavior (Cont.)

- The compiler allows the assignment of a base-class reference to a derived-class variable *if* we explicitly cast the base-class reference to the derived-class type.
- If an app needs to perform a derived-class-specific operation on a derived-class object referenced by a base-class variable, the app must first cast the base-class reference to a derived-class reference through a technique known as downcasting. This enables the app to invoke derived-class methods that are not in the base class.
- Fig. 12.1 demonstrates three ways to use base-class and derived-class variables.

```
1  // Fig. 12.1: PolymorphismTest.cs
2  // Assigning base-class and derived-class references to base-class and
3  // derived-class variables.
4  using System;
5
6  public class PolymorphismTest
7  {
8     public static void Main( string[] args )
9     {
10       // assign base-class reference to base-class variable
11       CommissionEmployee commissionEmployee = new CommissionEmployee(
12          "Sue", "Jones", "222-22-2222", 10000.00M, .06M );
13
14       // assign derived-class reference to derived-class variable
15       BasePlusCommissionEmployee basePlusCommissionEmployee =
16          new BasePlusCommissionEmployee( "Bob", "Lewis",
17          "333-33-3333", 5000.00M, .04M, 300.00M );
18
```

Fig. 12.1 | Assigning base-class and derived-class references to base-class and derived-class variables. (Part 1 of 5.)

```
19          // invoke ToString and Earnings on base-class object
20          // using base-class variable
21          Console.WriteLine( "{0} {1}:\n\n{2}\n{3}: {4:C}\n",
22             "Call CommissionEmployee's ToString and Earnings methods ",
23             "with base-class reference to base class object",
24             commissionEmployee.ToString(),
25             "earnings", commissionEmployee.Earnings() );
26
27          // invoke ToString and Earnings on derived-class object
28          // using derived-class variable
29          Console.WriteLine( "{0} {1}:\n\n{2}\n{3}: {4:C}\n",
30             "Call BasePlusCommissionEmployee's ToString and Earnings ",
31             "methods with derived class reference to derived-class object",
32             basePlusCommissionEmployee.ToString(),
33             "earnings", basePlusCommissionEmployee.Earnings() );
34
```

Fig. 12.1 | Assigning base-class and derived-class references to base-class and derived-class variables. (Part 2 of 5.)

```
35          // invoke ToString and Earnings on derived-class object
36          // using base-class variable
37          CommissionEmployee commissionEmployee2 =
38             basePlusCommissionEmployee;
39          Console.WriteLine( "{0} {1}:\n\n{2}\n{3}: {4:C}",
40             "Call BasePlusCommissionEmployee's ToString and Earnings ",
41             "with base class reference to derived-class object",
42             commissionEmployee2.ToString(), "earnings",
43             commissionEmployee2.Earnings() );
44       } // end Main
45    } // end class PolymorphismTest
```

Fig. 12.1 | Assigning base-class and derived-class references to base-class and derived-class variables. (Part 3 of 5.)

```
Call CommissionEmployee's ToString and Earnings methods with base class ref-
erence to base class object:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: $10,000.00
commission rate: 0.06
earnings: $600.00

Call BasePlusCommissionEmployee's ToString and Earnings methods with derived
class reference to derived class object:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: $5,000.00
commission rate: 0.04
base salary: $300.00
earnings: $500.00
```

Fig. 12.1 | Assigning base-class and derived-class references to base-class and derived-class variables. (Part 4 of 5.)

```
Call BasePlusCommissionEmployee's ToString and Earnings methods with base
class reference to derived class object:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: $5,000.00
commission rate: 0.04
base salary: $300.00
earnings: $500.00
```

Fig. 12.1 | Assigning base-class and derived-class references to base-class and derived-class variables. (Part 5 of 5.)

12.3 Demonstrating Polymorphic Behavior (Cont.)

- When the compiler encounters a `virtual` method call made through a variable, the compiler checks the *variable's* class type to determines if the method can be called.
- At execution time, *the type of the object to which the variable refers* determines the actual method to use.

12.4 Abstract Classes and Methods

- **Abstract classes**, or **abstract base classes** cannot be used to instantiate objects.
- Abstract base classes are too general to create real objects—they specify only what is common among derived classes.

Purpose of an Abstract Class

- Classes that can be used to instantiate objects are called **concrete classes**.
- Concrete classes provide the specifics that make it reasonable to instantiate objects.

12.4 Abstract Classes and Methods (Cont.)

Creating an Abstract Class

- An abstract class normally contains one or more **abstract methods**, which have the keyword `abstract` in their declaration.
- A class that contains abstract methods must be declared as an abstract class even if it contains concrete (non-`abstract`) methods.
- *Abstract methods do not provide implementations.*

12.4 Abstract Classes and Methods (Cont.)

Abstract Properties

- Abstract property declarations have the form:

```
public abstract PropertyType MyProperty
{
   get;
   set;
} // end abstract property
```

- An abstract property omits implementations for the `get` accessor and/or the `set` accessor.
- Concrete derived classes must provide implementations for *every* accessor declared in the abstract property.

12.4 Abstract Classes and Methods (Cont.)

Constructors and `static` Methods Cannot be Abstract

- Constructors and `static` methods *cannot* be declared `abstract`.

Software Engineering Observation 12.2

An abstract class declares common attributes and behaviors of the various classes that inherit from it, either directly or indirectly, in a class hierarchy. An abstract class typically contains one or more abstract methods or properties that concrete derived classes must override. The instance variables, concrete methods and concrete properties of an abstract class are subject to the normal rules of inheritance.

Common Programming Error 12.1

Attempting to instantiate an object of an abstract class is a compilation error.

Common Programming Error 12.2

Failure to implement a base class's abstract methods and properties in a derived class is a compilation error unless the derived class is also declared `abstract`.

12.4 Abstract Classes and Methods (Cont.)

- We can use abstract base classes to declare variables that can hold references to objects of any concrete classes derived from those abstract classes.
- You can use such variables to manipulate derived-class objects polymorphically and to invoke `static` methods declared in those abstract base classes.

12.5 Case Study: Payroll System Using Polymorphism

- In this section, we create an enhanced employee hierarchy to solve the following problem:
  - *A company pays its employees on a weekly basis. The employees are of four types: Salaried employees are paid a fixed weekly salary regardless of the number of hours worked, hourly employees are paid by the hour and receive "time-and-a-half" overtime pay for all hours worked in excess of 40 hours, commission employees are paid a percentage of their sales, and salaried-commission employees receive a base salary plus a percentage of their sales. For the current pay period, the company has decided to reward salaried-commission employees by adding 10% to their base salaries. The company wants to implement an app that performs its payroll calculations polymorphically.*

12.5 Case Study: Payroll System Using Polymorphism (Cont.)

- We use `abstract` class `Employee` to represent the general concept of an employee.
- `SalariedEmployee`, `CommissionEmployee` and `HourlyEmployee` extend `Employee`.
- Class `BasePlusCommissionEmployee`—which extends `CommissionEmployee`—represents the last employee type.

12.5 Case Study: Payroll System Using Polymorphism (Cont.)

- The UML class diagram in Fig. 12.2 shows the inheritance hierarchy for our polymorphic employee payroll app.

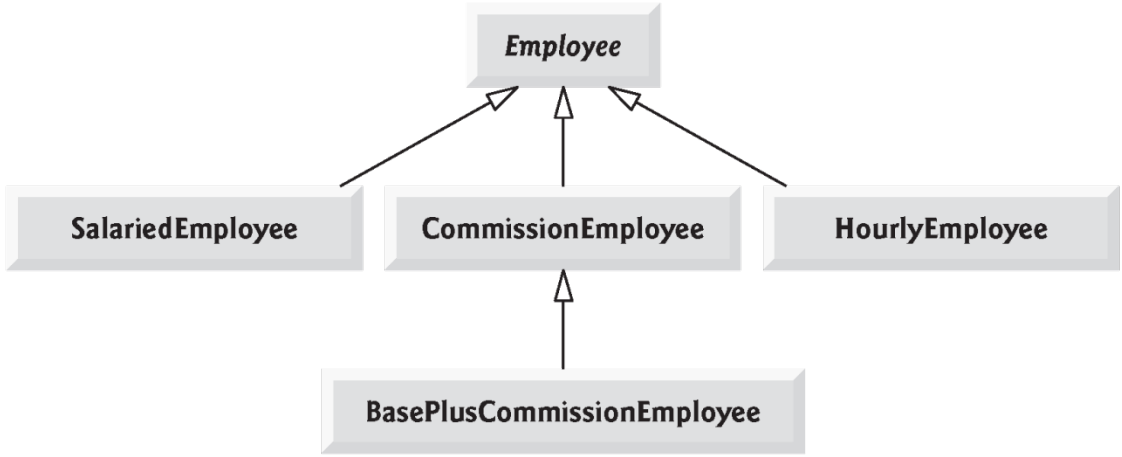

Fig. 12.2 | `Employee` hierarchy UML class diagram.

12.5.1 Creating Abstract Base Class `Employee`

- Class `Employee` provides methods `Earnings` and `ToString`, in addition to the properties that manipulate `Employee`'s instance variables.
- Each earnings calculation depends on the employee's class, so we declare `Earnings` as `abstract`.
- The app iterates through the array and calls method `Earnings` for each `Employee` object. These method calls are processed polymorphically.
- Each derived class overrides method `ToString` to create a string representation of an object of that class.

12.5 Case Study: Payroll System Using Polymorphism (Cont.)

- The diagram in Fig. 12.3 shows each of the five classes in the hierarchy down the left side and methods `Earnings` and `ToString` across the top.
- The `Employee` class's declaration is shown in Fig. 12.4.

	Earnings	ToString
Employee	abstract	*firstName lastName* social security number: *SSN*
Salaried-Employee	weeklySalary	salaried employee: *firstName lastName* social security number: *SSN* weekly salary: *weeklysalary*
Hourly-Employee	*If hours <= 40* wage * hours *If hours > 40* 40 * wage + (hours - 40) * wage * 1.5	hourly employee: *firstName lastName* social security number: *SSN* hourly wage: *wage* hours worked: *hours*
Commission-Employee	commissionRate * grossSales	commission employee: *firstName lastName* social security number: *SSN* gross sales: *grossSales* commission rate: *commissionRate*
BasePlus-Commission-Employee	(commissionRate * grossSales) + baseSalary	base salaried commission employee: *firstName lastName* social security number: *SSN* gross sales: *grossSales* commission rate: *commissionRate* base salary: *baseSalary*

```
1  // Fig. 12.4: Employee.cs
2  // Employee abstract base class.
3  public abstract class Employee
4  {
5     // read-only property that gets employee's first name
6     public string FirstName { get; private set; }
7
8     // read-only property that gets employee's last name
9     public string LastName { get; private set; }
10
11    // read-only property that gets employee's social security number
12    public string SocialSecurityNumber { get; private set; }
13
14    // three-parameter constructor
15    public Employee( string first, string last, string ssn )
16    {
17       FirstName = first;
18       LastName = last;
19       SocialSecurityNumber = ssn;
20    } // end three-parameter Employee constructor
21
```

Fig. 12.4 | Employee abstract base class. (Part 1 of 2.)

```
22      // return string representation of Employee object, using properties
23      public override string ToString()
24      {
25         return string.Format( "{0} {1}\nsocial security number: {2}",
26            FirstName, LastName, SocialSecurityNumber );
27      } // end method ToString
28
29      // abstract method overridden by derived classes
30      public abstract decimal Earnings(); // no implementation here
31   } // end abstract class Employee
```

Fig. 12.4 | `Employee` abstract base class. (Part 2 of 2.)

12.5.2 Creating Concrete Derived Class `SalariedEmployee`

- The Salaried`Employee` class's declaration is shown in Fig. 12.5.

```
1  // Fig. 12.5: SalariedEmployee.cs
2  // SalariedEmployee class that extends Employee.
3  using System;
4
5  public class SalariedEmployee : Employee
6  {
7     private decimal weeklySalary;
8
9     // four-parameter constructor
10    public SalariedEmployee( string first, string last, string ssn,
11       decimal salary ) : base( first, last, ssn )
12    {
13       WeeklySalary = salary; // validate salary via property
14    } // end four-parameter SalariedEmployee constructor
15
```

Fig. 12.5 | SalariedEmployee class that extends Employee. (Part 1 of 3.)

```
16      // property that gets and sets salaried employee's salary
17      public decimal WeeklySalary
18      {
19         get
20         {
21            return weeklySalary;
22         } // end get
23         set
24         {
25            if ( value >= 0 ) // validation
26               weeklySalary = value;
27            else
28               throw new ArgumentOutOfRangeException( "WeeklySalary",
29                  value, "WeeklySalary must be >= 0" );
30         } // end set
31      } // end property WeeklySalary
32
33      // calculate earnings; override abstract method Earnings in Employee
34      public override decimal Earnings()
35      {
36         return WeeklySalary;
37      } // end method Earnings
38
```

Fig. 12.5 | SalariedEmployee class that extends Employee. (Part 2 of 3.)

```
39      // return string representation of SalariedEmployee object
40      public override string ToString()
41      {
42         return string.Format( "salaried employee: {0}\n{1}: {2:C}",
43            base.ToString(), "weekly salary", WeeklySalary );
44      } // end method ToString
45   } // end class SalariedEmployee
```

Fig. 12.5 | SalariedEmployee class that extends Employee. (Part 3 of 3.)

12.5.3 Creating Concrete Derived Class `HourlyEmployee`

- The Hourly`Employee` class's declaration is shown in Fig. 12.6.

```
1  // Fig. 12.6: HourlyEmployee.cs
2  // HourlyEmployee class that extends Employee.
3  using System;
4
5  public class HourlyEmployee : Employee
6  {
7     private decimal wage; // wage per hour
8     private decimal hours; // hours worked for the week
9
10    // five-parameter constructor
11    public HourlyEmployee( string first, string last, string ssn,
12       decimal hourlyWage, decimal hoursWorked )
13       : base( first, last, ssn )
14    {
15       Wage = hourlyWage; // validate hourly wage via property
16       Hours = hoursWorked; // validate hours worked via property
17    } // end five-parameter HourlyEmployee constructor
18
```

Fig. 12.6 | `HourlyEmployee` class that extends `Employee`. (Part 1 of 4.)

```
19      // property that gets and sets hourly employee's wage
20      public decimal Wage
21      {
22         get
23         {
24            return wage;
25         } // end get
26         set
27         {
28            if ( value >= 0 ) // validation
29               wage = value;
30            else
31               throw new ArgumentOutOfRangeException( "Wage",
32                  value, "Wage must be >= 0" );
33         } // end set
34      } // end property Wage
35
```

Fig. 12.6 | `HourlyEmployee` class that extends `Employee`. (Part 2 of 4.)

```
36      // property that gets and sets hourly employee's hours
37      public decimal Hours
38      {
39         get
40         {
41            return hours;
42         } // end get
43         set
44         {
45            if ( value >= 0 && value <= 168 ) // validation
46               hours = value;
47            else
48               throw new ArgumentOutOfRangeException( "Hours",
49                  value, "Hours must be >= 0 and <= 168" );
50         } // end set
51      } // end property Hours
52
53      // calculate earnings; override Employee's abstract method Earnings
54      public override decimal Earnings()
55      {
56         if ( Hours <= 40 ) // no overtime
57            return Wage * Hours;
58         else
59            return ( 40 * Wage ) + ( ( Hours - 40 ) * Wage * 1.5M );
60      } // end method Earnings
```

Fig. 12.6 | HourlyEmployee class that extends Employee. (Part 3 of 4.)

```
61
62      // return string representation of HourlyEmployee object
63      public override string ToString()
64      {
65         return string.Format(
66            "hourly employee: {0}\n{1}: {2:C}; {3}: {4:F2}",
67            base.ToString(), "hourly wage", Wage, "hours worked", Hours );
68      } // end method ToString
69   } // end class HourlyEmployee
```

Fig. 12.6 | `HourlyEmployee` class that extends `Employee`. (Part 4 of 4.)

12.5.4 Creating Concrete Derived Class `CommissionEmployee`

- The Commission`Employee` class's declaration is shown in Fig. 12.7.

```
1  // Fig. 12.7: CommissionEmployee.cs
2  // CommissionEmployee class that extends Employee.
3  using System;
4
5  public class CommissionEmployee : Employee
6  {
7     private decimal grossSales; // gross weekly sales
8     private decimal commissionRate; // commission percentage
9
10    // five-parameter constructor
11    public CommissionEmployee( string first, string last, string ssn,
12       decimal sales, decimal rate ) : base( first, last, ssn )
13    {
14       GrossSales = sales; // validate gross sales via property
15       CommissionRate = rate; // validate commission rate via property
16    } // end five-parameter CommissionEmployee constructor
17
```

Fig. 12.7 | CommissionEmployee class that extends Employee. (Part 1 of 4.)

```
18      // property that gets and sets commission employee's gross sales
19      public decimal GrossSales
20      {
21         get
22         {
23            return grossSales;
24         } // end get
25         set
26         {
27            if ( value >= 0 )
28               grossSales = value;
29            else
30               throw new ArgumentOutOfRangeException(
31                  "GrossSales", value, "GrossSales must be >= 0" );
32         } // end set
33      } // end property GrossSales
34
```

Fig. 12.7 | CommissionEmployee class that extends Employee. (Part 2 of 4.)

```
35      // property that gets and sets commission employee's commission rate
36      public decimal CommissionRate
37      {
38         get
39         {
40            return commissionRate;
41         } // end get
42         set
43         {
44            if ( value > 0 && value < 1 )
45               commissionRate = value;
46            else
47               throw new ArgumentOutOfRangeException( "CommissionRate",
48                  value, "CommissionRate must be > 0 and < 1" );
49         } // end set
50      } // end property CommissionRate
51
52      // calculate earnings; override abstract method Earnings in Employee
53      public override decimal Earnings()
54      {
55         return CommissionRate * GrossSales;
56      } // end method Earnings
57
```

Fig. 12.7 | CommissionEmployee class that extends Employee. (Part 3 of 4.)

```
58      // return string representation of CommissionEmployee object
59      public override string ToString()
60      {
61         return string.Format( "{0}: {1}\n{2}: {3:C}\n{4}: {5:F2}",
62            "commission employee", base.ToString(),
63            "gross sales", GrossSales, "commission rate", CommissionRate );
64      } // end method ToString
65   } // end class CommissionEmployee
```

Fig. 12.7 | CommissionEmployee class that extends Employee. (Part 4 of 4.)

12.5.5 Creating Indirect Concrete Derived Class `BasePlusCommissionEmployee`

- Class `BasePlusCommissionEmployee` (Fig. 12.8) extends class `CommissionEmployee` and therefore is an indirect derived class of class `Employee`.

```
1  // Fig. 12.8: BasePlusCommissionEmployee.cs
2  // BasePlusCommissionEmployee class that extends CommissionEmployee.
3  using System;
4
5  public class BasePlusCommissionEmployee : CommissionEmployee
6  {
7     private decimal baseSalary; // base salary per week
8
9     // six-parameter constructor
10    public BasePlusCommissionEmployee( string first, string last,
11       string ssn, decimal sales, decimal rate, decimal salary )
12       : base( first, last, ssn, sales, rate )
13    {
14       BaseSalary = salary; // validate base salary via property
15    } // end six-parameter BasePlusCommissionEmployee constructor
16
```

Fig. 12.8 | BasePlusCommissionEmployee class that extends CommissionEmployee. (Part 1 of 3.)

```
17      // property that gets and sets
18      // base-salaried commission employee's base salary
19      public decimal BaseSalary
20      {
21         get
22         {
23            return baseSalary;
24         } // end get
25         set
26         {
27            if ( value >= 0 )
28               baseSalary = value;
29            else
30               throw new ArgumentOutOfRangeException( "BaseSalary",
31                  value, "BaseSalary must be >= 0" );
32         } // end set
33      } // end property BaseSalary
34
35      // calculate earnings; override method Earnings in CommissionEmployee
36      public override decimal Earnings()
37      {
38         return BaseSalary + base.Earnings();
39      } // end method Earnings
```

Fig. 12.8 | BasePlusCommissionEmployee class that extends CommissionEmployee. (Part 2 of 3.)

```
40
41      // return string representation of BasePlusCommissionEmployee object
42      public override string ToString()
43      {
44         return string.Format( "base-salaried {0}; base salary: {1:C}",
45            base.ToString(), BaseSalary );
46      } // end method ToString
47   } // end class BasePlusCommissionEmployee
```

Fig. 12.8 | BasePlusCommissionEmployee class that extends CommissionEmployee. (Part 3 of 3.)

12.5.6 Polymorphic Processing, Operator `is` and Downcasting

- The app in Fig. 12.9 tests our `Employee` hierarchy.

```
1  // Fig. 12.9: PayrollSystemTest.cs
2  // Employee hierarchy test app.
3  using System;
4
5  public class PayrollSystemTest
6  {
7     public static void Main( string[] args )
8     {
9        // create derived-class objects
10       SalariedEmployee salariedEmployee =
11          new SalariedEmployee( "John", "Smith", "111-11-1111", 800.00M );
12       HourlyEmployee hourlyEmployee =
13          new HourlyEmployee( "Karen", "Price",
14          "222-22-2222", 16.75M, 40.0M );
15       CommissionEmployee commissionEmployee =
16          new CommissionEmployee( "Sue", "Jones",
17          "333-33-3333", 10000.00M, .06M );
18       BasePlusCommissionEmployee basePlusCommissionEmployee =
19          new BasePlusCommissionEmployee( "Bob", "Lewis",
20          "444-44-4444", 5000.00M, .04M, 300.00M );
21
22       Console.WriteLine( "Employees processed individually:\n" );
23
```

Fig. 12.9 | Employee hierarchy test app. (Part 1 of 6.)

```
24          Console.WriteLine( "{0}\nearned: {1:C}\n",
25             salariedEmployee, salariedEmployee.Earnings() );
26          Console.WriteLine( "{0}\nearned: {1:C}\n",
27             hourlyEmployee, hourlyEmployee.Earnings() );
28          Console.WriteLine( "{0}\nearned: {1:C}\n",
29             commissionEmployee, commissionEmployee.Earnings() );
30          Console.WriteLine( "{0}\nearned: {1:C}\n",
31             basePlusCommissionEmployee,
32             basePlusCommissionEmployee.Earnings() );
33
34          // create four-element Employee array
35          Employee[] employees = new Employee[ 4 ];
36
37          // initialize array with Employees of derived types
38          employees[ 0 ] = salariedEmployee;
39          employees[ 1 ] = hourlyEmployee;
40          employees[ 2 ] = commissionEmployee;
41          employees[ 3 ] = basePlusCommissionEmployee;
42
43          Console.WriteLine( "Employees processed polymorphically:\n" );
44
45          // generically process each element in array employees
46          foreach ( Employee currentEmployee in employees )
47          {
48             Console.WriteLine( currentEmployee ); // invokes ToString
```

Fig. 12.9 | Employee hierarchy test app. (Part 2 of 6.)

```
49
50          // determine whether element is a BasePlusCommissionEmployee
51          if ( currentEmployee is BasePlusCommissionEmployee )
52          {
53             // downcast Employee reference to
54             // BasePlusCommissionEmployee reference
55             BasePlusCommissionEmployee employee =
56                ( BasePlusCommissionEmployee ) currentEmployee;
57
58             employee.BaseSalary *= 1.10M;
59             Console.WriteLine(
60                "new base salary with 10% increase is: {0:C}",
61                employee.BaseSalary );
62          } // end if
63
64          Console.WriteLine(
65             "earned {0:C}\n", currentEmployee.Earnings() );
66       } // end foreach
67
68       // get type name of each object in employees array
69       for ( int j = 0; j < employees.Length; j++ )
70          Console.WriteLine( "Employee {0} is a {1}", j,
71             employees[ j ].GetType() );
72    } // end Main
73 } // end class PayrollSystemTest
```

Fig. 12.9 | Employee hierarchy test app. (Part 3 of 6.)

```
Employees processed individually:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: $800.00
earned: $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: $16.75; hours worked: 40.00
earned: $670.00

commission employee: Sue Jones
social security number: 333-33-3333
gross sales: $10,000.00
commission rate: 0.06
earned: $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: $5,000.00
commission rate: 0.04; base salary: $300.00
earned: $500.00
```

Fig. 12.9 | Employee hierarchy test app. (Part 4 of 6.)

```
Employees processed polymorphically:

salaried employee: John Smith
social security number: 111-11-1111
weekly salary: $800.00
earned $800.00

hourly employee: Karen Price
social security number: 222-22-2222
hourly wage: $16.75; hours worked: 40.00
earned $670.00
```

Fig. 12.9 | Employee hierarchy test app. (Part 5 of 6.)

```
commission employee: Sue Jones
social security number: 333-33-3333
gross sales: $10,000.00
commission rate: 0.06
earned $600.00

base-salaried commission employee: Bob Lewis
social security number: 444-44-4444
gross sales: $5,000.00
commission rate: 0.04; base salary: $300.00
new base salary with 10% increase is: $330.00
earned $530.00

Employee 0 is a SalariedEmployee
Employee 1 is a HourlyEmployee
Employee 2 is a CommissionEmployee
Employee 3 is a BasePlusCommissionEmployee
```

Fig. 12.9 | Employee hierarchy test app. (Part 6 of 6.)

Common Programming Error 12.3

Assigning a base-class variable to a derived-class variable (without an explicit downcast) is a compilation error.

Software Engineering Observation 12.3

If at execution time the reference to a derived-class object has been assigned to a variable of one of its direct or indirect base classes, it's acceptable to cast the reference stored in that base-class variable back to a reference of the derived-class type. Before performing such a cast, use the `is` operator to ensure that the object is indeed an object of an appropriate derived-class type.

12.5.6 Polymorphic Processing, Operator `is` and Downcasting (Cont.)

- You can avoid a potential `InvalidCastException` by using the **as** operator to perform a downcast rather than a cast operator.
  - If the downcast is invalid, the expression will be null instead of throwing an exception.
- Method `GetType` returns an object of class Type (of namespace `System`), which contains information about the object's type, including its class name, the names of its methods, and the name of its base class.
- The `Type` class's `ToString` method returns the class name.

12.5.6 Polymorphic Processing, Operator `is` and Downcasting (Cont.)

12.5.7 Summary of the Allowed Assignments Between Base-Class and Derived-Class Variables

- Assigning a base-class reference to a base-class variable is straightforward.
- Assigning a derived-class reference to a derived-class variable is straightforward.
- Assigning a derived-class reference to a base-class variable is safe, because the derived-class object *is an* object of its base class. However, this reference can be used to refer only to base-class members.
- Attempting to assign a base-class reference to a derived-class variable is a compilation error. To avoid this error, the base-class reference must be cast to a derived-class type explicitly.