Chapter 12

OOP: Polymorphism, Interfaces and Operator Overloading

12.2 Polymorphism Examples

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

By example: *Polymorphic Employee Inheritance Hierarchy*

- Simple payroll application that **polymorphically** calculates the weekly pay of several different types of employees using each employee's `Earnings` method.
- The earnings of each type of employee are calculated in a specific way.
- Two new classes – `SalariedEmployee` (for people paid a fixed weekly salary) and `HourlyEmployee` (for people paid an hourly salary and "time-and-a-half" for overtime).

By example: *Polymorphic Employee Inheritance Hierarchy* cont

- *common set of functionality* for all the classes in the updated hierarchy in an "abstract" class, **`Employee`**, from which classes **`SalariedEmployee`**, **`HourlyEmployee`** and **`CommissionEmployee`** *inherit directly* and class **`BasePlusCommissionEmployee`** *inherits* ***indirectly***.
- => The correct earnings calculation is performed due to C#'s polymorphic capabilities.

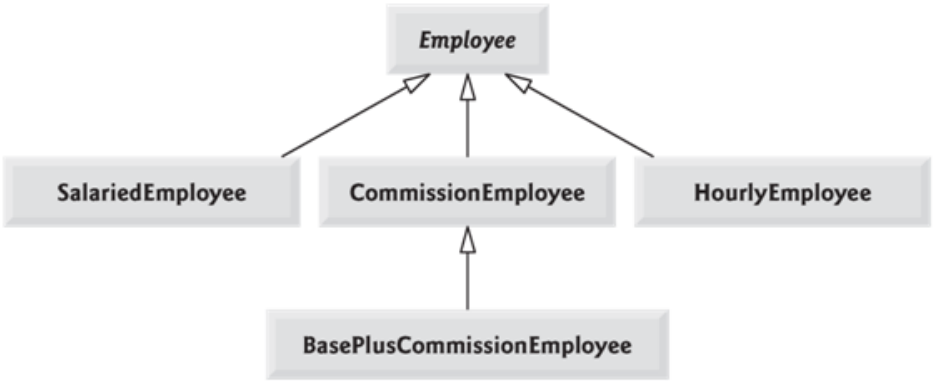

Fig. 12.2 | Employee hierarchy UML class diagram.

12.3 Demonstrating Polymorphic Behavior

```
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 4.)

```
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 4.)

```
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 4.)

```
Call CommissionEmployee's ToString and Earnings methods with base class refer-
ence 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

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
```

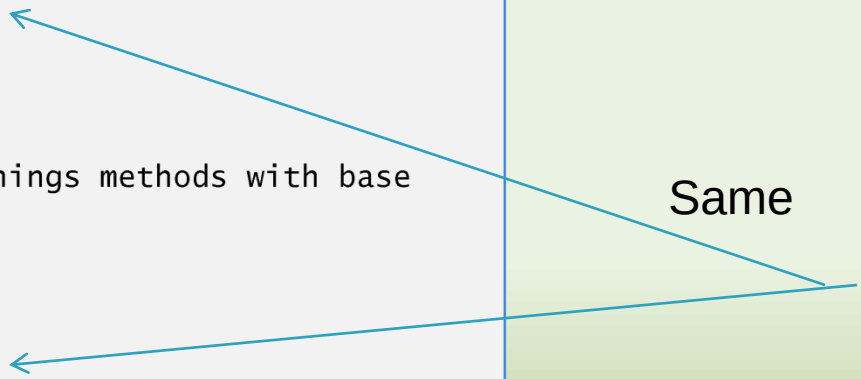

Fig. 12.1 | Assigning base-class and derived-class references to base-class and derived-class variables. (Part 4 of 4.)

Payroll processing

- Types of employees
  - Salaried Employees : fixed weekly salary
  - Hourly Employees: paid by the hour and receives 1.5 times overtime pay for hours worked in excess of 40 hours
  - Commission Employees
  - Salaried-commission Employees
- For the current pay period, the company has decided to reward salaried-commission employees by adding 10% to their base salary.

12.5 Case Study: Payroll System Using Polymorphism

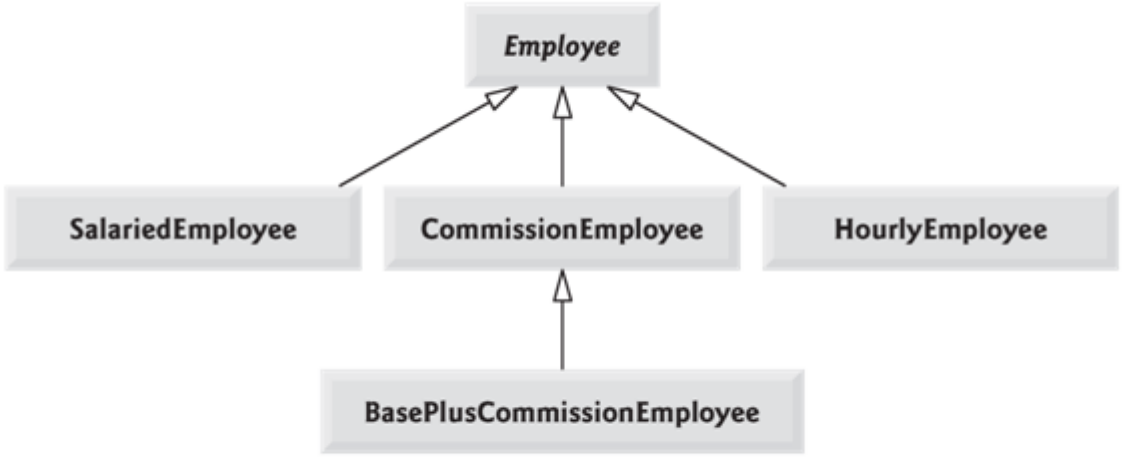

Fig. 12.2 | Employee hierarchy UML class diagram.

	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*

Fig. 12.3 | Polymorphic interface for the `Employee` hierarchy classes.

```
// Fig. 12.4: Employee.cs
// Employee abstract base class.
public abstract class Employee
{
   public string FirstName { get; }
   public string LastName { get; }
   public string SocialSecurityNumber { get; }

   // three-parameter constructor
   public Employee(string firstName, string lastName,
      string socialSecurityNumber)
   {
      FirstName = firstName;
      LastName = lastName;
      SocialSecurityNumber = socialSecurityNumber;
   }
   public override string ToString() => $"{FirstName} {LastName}\n“
+ $"social security number: {SocialSecurityNumber}";

   // abstract method overridden by derived classes
   public abstract decimal Earnings(); // no implementation here
}
```

```
// Fig. 12.5: SalariedEmployee.cs
// SalariedEmployee class that extends Employee.
public class SalariedEmployee : Employee
{
   private decimal weeklySalary;
   // four-parameter constructor
   public SalariedEmployee(string firstName, string lastName,
      string socialSecurityNumber, decimal weeklySalary)
      : base(firstName, lastName, socialSecurityNumber)
   {
      WeeklySalary = weeklySalary; // validate salary via property
   }

   // property that gets and sets salaried employee's salary
   public decimal WeeklySalary
   {
      get
      {
         return weeklySalary;
      }
      set
      {
         if (value < 0) // validation
         {
            throw new ArgumentOutOfRangeException(nameof(value),
               value, $"{nameof(WeeklySalary)} must be >= 0");
         }
         weeklySalary = value;
   }
}
```

```
 // calculate earnings; override abstract method Earnings in
Employee
   public override decimal Earnings() => WeeklySalary;

   // return string representation of SalariedEmployee
object
   public override string ToString() =>
      $"salaried employee: {base.ToString()}\n" +
      $"weekly salary: {WeeklySalary:C}";

}
```

```
// Fig. 12.6: HourlyEmployee.cs
public class HourlyEmployee : Employee
{
   private decimal wage; // wage per hour
   private decimal hours; // hours worked for the week
   public HourlyEmployee(string firstName, string lastName,
      string socialSecurityNumber, decimal hourlyWage, decimal hoursWorked)
      : base(firstName, lastName, socialSecurityNumber)
   {
      Wage = hourlyWage; // validate hourly wage
      Hours = hoursWorked; // validate hours worked
   }
   // property that gets and sets hourly employee's wage
   public decimal Wage
   {
      get
      {
         return wage;
      }
      set
      {
         if (value < 0) // validation
            throw new ArgumentOutOfRangeException(nameof(value),
               value, $"{nameof(Wage)} must be >= 0");
         wage = value;
   }
}
```

```
 // property that gets and sets hourly employee's hours
   public decimal Hours
   {
      get{ return hours; }
      set
      {
         if (value < 0 || value > 168) // validation
         {
            throw new ArgumentOutOfRangeException(nameof(value),
               value, $"{nameof(Hours)} must be >= 0 and <= 168");
         }
         hours = value;
      }
   }

   // calculate earnings; override Employee's abstract method Earnings
   public override decimal Earnings()
   {
      if (Hours <= 40) // no overtime
         return Wage * Hours;
      else
         return (40 * Wage) + ((Hours - 40) * Wage * 1.5M);
   }
   // return string representation of HourlyEmployee object
   public override string ToString() =>
      $"hourly employee: {base.ToString()}\n" +
      $"hourly wage: {Wage:C}\nhours worked: {Hours:F2}";
}
```

```
// Fig. 12.7: CommissionEmployee.cs
public class CommissionEmployee : Employee
{
   private decimal grossSales; // gross weekly sales
   private decimal commissionRate; // commission percentage

   // five-parameter constructor
   public CommissionEmployee(string firstName, string lastName,
      string socialSecurityNumber, decimal grossSales,
      decimal commissionRate)
      : base(firstName, lastName, socialSecurityNumber)
   {
      GrossSales = grossSales; // validates gross sales
      CommissionRate = commissionRate; // validates commission rate
   }
   public decimal GrossSales
   {
      get
      {
         return grossSales;
      }
      set
      {
         if (value < 0) // validation
            throw new ArgumentOutOfRangeException(nameof(value),
               value, $"{nameof(GrossSales)} must be >= 0");
         grossSales = value;
      }
   }
```

```
   public decimal CommissionRate
   {
      get
      {
         return commissionRate;
      }
      set
      {
         if (value <= 0 || value >= 1) // validation
         {
            throw new ArgumentOutOfRangeException(nameof(value),
               value, $"{nameof(CommissionRate)} must be > 0 and < 1");
         }
         commissionRate = value;
      }
   }

   // calculate earnings; override abstract method Earnings in Employee
   public override decimal Earnings() => CommissionRate * GrossSales;

   // return string representation of CommissionEmployee object
   public override string ToString() =>
      $"commission employee: {base.ToString()}\n" +
      $"gross sales: {GrossSales:C}\n" +
      $"commission rate: {CommissionRate:F2}";
}
```

```
public class BasePlusCommissionEmployee : CommissionEmployee
{
   private decimal baseSalary; // base salary per week
   public BasePlusCommissionEmployee(string firstName, string lastName,
      string socialSecurityNumber, decimal grossSales,
      decimal commissionRate, decimal baseSalary)
      : base(firstName, lastName, socialSecurityNumber, grossSales, commissionRate)
   {
      BaseSalary = baseSalary; // validates base salary
   }
   public decimal BaseSalary
   {
      get
      {
         return baseSalary;
      }
      set
      {
         if (value < 0) // validation
            throw new ArgumentOutOfRangeException(nameof(value),
               value, $"{nameof(BaseSalary)} must be >= 0");
         baseSalary = value;
      }
   }
   public override decimal Earnings() => BaseSalary + base.Earnings();

   // return string representation of BasePlusCommissionEmployee
   public override string ToString() =>
      $"base-salaried {base.ToString()}\nbase salary: {BaseSalary:C}";
}
```

```
// Fig. 12.9: PayrollSystemTest.cs
using System.Collections.Generic;
class PayrollSystemTest
{
   static void Main()
   {
      // create derived-class objects
      var salariedEmployee = new SalariedEmployee("John", "Smith",
          "111-11-1111", 800.00M);
      var hourlyEmployee = new HourlyEmployee("Karen", "Price",
         "222-22-2222", 16.75M, 40.0M);
      var commissionEmployee = new CommissionEmployee("Sue", "Jones",
         "333-33-3333", 10000.00M, .06M);
      var basePlusCommissionEmployee =
         new BasePlusCommissionEmployee("Bob", "Lewis",
         "444-44-4444", 5000.00M, .04M, 300.00M);

      Console.WriteLine("Employees processed individually:\n");

      Console.WriteLine($"{salariedEmployee}\nearned: " +
          $"{salariedEmployee.Earnings():C}\n");
      Console.WriteLine(
         $"{hourlyEmployee}\nearned: {hourlyEmployee.Earnings():C}\n");
      Console.WriteLine($"{commissionEmployee}\nearned: " +
          $"{commissionEmployee.Earnings():C}\n");
      Console.WriteLine($"{basePlusCommissionEmployee}\nearned: " +
          $"{basePlusCommissionEmployee.Earnings():C}\n");
```

```
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 application. (Part 5 of 6.)

```
var employees = new List<Employee>() {salariedEmployee,
        hourlyEmployee, commissionEmployee, basePlusCommissionEmployee};
     Console.WriteLine("Employees processed polymorphically:\n");

     // generically process each element in employees
     foreach (var currentEmployee in employees)
     {
        Console.WriteLine(currentEmployee); // invokes ToString

        // determine whether element is a BasePlusCommissionEmployee
        if (currentEmployee is BasePlusCommissionEmployee)
        {
           // downcast Employee reference to
           // BasePlusCommissionEmployee reference
           var employee = (BasePlusCommissionEmployee)currentEmployee;

           employee.BaseSalary *= 1.10M;
           Console.WriteLine("new base salary with 10% increase is: " +
               $"{employee.BaseSalary:C}");
         }

        Console.WriteLine($"earned: {currentEmployee.Earnings():C}\n");
     }
```

GetType

```
for (int j = 0; j < employees.Count; j++)
{
     Console.WriteLine(
      $"Employee {j} is a {employees[j].GetType()}");
}
```

```
1  // Fig. 12.9: PayrollSystemTest.cs
2  // Employee hierarchy test application.
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 application. (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
```

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

```
45          // generically process each element in array employees
46          foreach ( Employee currentEmployee in employees )
47          {
48             Console.WriteLine( currentEmployee ); // invokes ToString
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
```

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

```
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 application. (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

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 application. (Part 6 of 6.)

Sealed methods and classes

- Only methods declared virtual, override or abstract can be overridden
- A sealed method in a base class cannot be overridden in a derived class
- Implicitly sealed methods
  - Private methods
  - Static methods
- A derived class method declared both override and sealed can override a base-class method, but cannot be overridden in derived classes further down in the hierarchy

```
public class BaseClass
{
    public virtual void Display(){ Console.WriteLine("Virtual method");}
}
public class DerivedClass : BaseClass
{
    public override sealed void Display()
    {
        Console.WriteLine("Sealed method");
    }
}
public class ThirdClass : DerivedClass
{
    public override void Display()
    {
        Console.WriteLine(“ThirdClass"); // Here we try again to override display
method which is not possible and will give error
    }
}
```

Sealed methods and classes

- Sealed class cannot be a base class
- All methods in a sealed class are implicitly sealed

```
sealed class SealedClass {

    public int Add(int a, int b)
    {
        return a + b;
    }

}
```

Interfaces

- Define and standardize the ways in which objects can interact with one another
- Specify what operations an object must permit client code to perform
  - Doesn't specify how they're performed
- Enable different types to have is-a relationships for polymorphic processing

Implementing an interface

- Members are implicitly public and abstract
- : notation
- After that: the base class, comma separated list of all interfaces implemented by the class

Interface vs. abstract class

- Interface often is used in place of and abstract class when there is no default implementation to inherit
  - No default method implementation

IPayable hierarchy

- Use for generalized payment system
  - Pay invoices and employees
  - Invoice is not and employee , employee is not an invoice
  - => both are payable
  - Give is-a relationship to different types

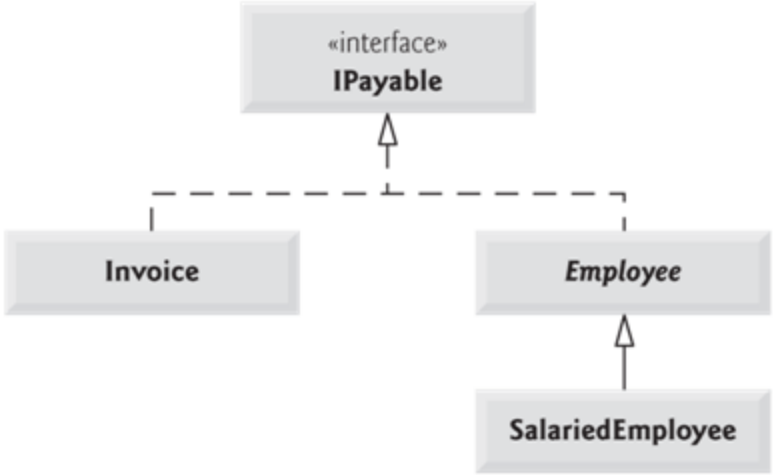

Fig. 12.10 | `IPayable` interface and class hierarchy UML class diagram.

```
// Fig. 12.11: IPayable.cs
// IPayable interface declaration.
public interface IPayable
{
   decimal GetPaymentAmount(); // calculate
//payment; no implementation
   //string Version { get; set; }
}
```

```
public class Invoice : Object, IPayable
{
   public string PartNumber { get; }
   public string PartDescription { get; }
   private int quantity;
   private decimal pricePerItem;
   public Invoice(string partNumber, string partDescription, int quantity,
      decimal pricePerItem)
   {
      PartNumber = partNumber;
      PartDescription = partDescription;
      Quantity = quantity; // validate quantity
      PricePerItem = pricePerItem; // validate price per item
   }
   public int Quantity
   {
      get
      {
         return quantity;
      }
      set
      {
         if (value < 0) // validation
            throw new ArgumentOutOfRangeException(nameof(value),
               value, $"{nameof(Quantity)} must be >= 0");
         quantity = value;
      }
   }
```

```
public decimal PricePerItem
{
   get
   {
         return pricePerItem;
   }
   set
   {
         if (value < 0) // validation
            throw new ArgumentOutOfRangeException(nameof(value),
               value, $"{nameof(PricePerItem)} must be >= 0");
         pricePerItem = value;
   }
}

   // return string representation of Invoice object
   public override string ToString() =>
      $"invoice:\npart number: {PartNumber} ({PartDescription})\n" +
      $"quantity: {Quantity}\nprice per item: {PricePerItem:C}";

   // method required to carry out contract with interface IPayable
   public decimal GetPaymentAmount() => Quantity * PricePerItem;
}
```

```
// Fig. 12.13: Employee.cs
public abstract class Employee : IPayable
{
   public string FirstName { get; }
   public string LastName { get; }
   public string SocialSecurityNumber { get; }

   // three-parameter constructor
   public Employee(string firstName, string lastName,
      string socialSecurityNumber)
   {
      FirstName = firstName;
      LastName = lastName;
      SocialSecurityNumber = socialSecurityNumber;
   }
   public override string ToString() => $"{FirstName} {LastName}\n" +
      $"social security number: {SocialSecurityNumber}";

   public abstract decimal Earnings(); // no implementation here

   public decimal GetPaymentAmount() => Earnings();
}
```

```
public class SalariedEmployee : Employee
{
   private decimal weeklySalary;
   public SalariedEmployee(string firstName, string lastName,
      string socialSecurityNumber, decimal weeklySalary)
      : base(firstName, lastName, socialSecurityNumber)
   {
      WeeklySalary = weeklySalary; // validate salary via property
   }
   public decimal WeeklySalary
   {
      get
      {
         return weeklySalary;
      }
      set
      {
         if (value < 0) // validation
            throw new ArgumentOutOfRangeException(nameof(value),
               value, $"{nameof(WeeklySalary)} must be >= 0");
         weeklySalary = value;
      }
   }
   public override decimal Earnings() => WeeklySalary;
   public override string ToString() =>
      $"salaried employee: {base.ToString()}\n" +
      $"weekly salary: {WeeklySalary:C}";
}
```

```
// Fig. 12.14: PayableInterfaceTest.cs
class PayableInterfaceTest
{
   static void Main()
   {
      var payableObjects = new List<IPayable>() {
         new Invoice("01234", "seat", 2, 375.00M),
         new Invoice("56789", "tire", 4, 79.95M),
         new SalariedEmployee("John", "Smith", "111-11-1111", 800.00M),
         new SalariedEmployee("Lisa", "Barnes", "888-88-8888", 1200.00M)};

      Console.WriteLine(
         "Invoices and Employees processed polymorphically:\n");
      // generically process each element in payableObjects
      foreach (var payable in payableObjects)
      {
         // output payable and its appropriate payment amount
         Console.WriteLine($"{payable}");
         Console.WriteLine(
            $"payment due: {payable.GetPaymentAmount():C}\n");
      }
   }
}
```

```
Invoices and Employees processed polymorphically:

invoice:
part number: 01234 (seat)
quantity: 2
price per item: $375.00
payment due: $750.00

invoice:
part number: 56789 (tire)
quantity: 4
price per item: $79.95
payment due: $319.80

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

salaried employee: Lisa Barnes
social security number: 888-88-8888
weekly salary: $1,200.00
payment due: $1,200.00
```

Fig. 12.15 | Tests interface `IPayable` with disparate classes. (Part 3 of 3.)

```
1  // Fig. 12.15: PayableInterfaceTest.cs
2  // Tests interface IPayable with disparate classes.
3  using System;
4
5  public class PayableInterfaceTest
6  {
7     public static void Main( string[] args )
8     {
9        // create four-element IPayable array
10       IPayable[] payableObjects = new IPayable[ 4 ];
11
12       // populate array with objects that implement IPayable
13       payableObjects[ 0 ] = new Invoice( "01234", "seat", 2, 375.00M );
14       payableObjects[ 1 ] = new Invoice( "56789", "tire", 4, 79.95M );
15       payableObjects[ 2 ] = new SalariedEmployee( "John", "Smith",
16          "111-11-1111", 800.00M );
17       payableObjects[ 3 ] = new SalariedEmployee( "Lisa", "Barnes",
18          "888-88-8888", 1200.00M );
19
20       Console.WriteLine(
21          "Invoices and Employees processed polymorphically:\n" );
22
```

Fig. 12.15 | Tests interface `IPayable` with disparate classes. (Part 1 of 3.)

```
23          // generically process each element in array payableObjects
24          foreach ( var currentPayable in payableObjects )
25          {
26             // output currentPayable and its appropriate payment amount
27             Console.WriteLine( "payment due {0}: {1:C}\n",
28                currentPayable, currentPayable.GetPaymentAmount() );
29          } // end foreach
30       } // end Main
31    } // end class PayableInterfaceTest
```

Fig. 12.15 | Tests interface `IPayable` with disparate classes. (Part 2 of 3.)

structs

- Often in programs we have small classes that serve as collections of related variables stored in memory
- Here we require no inheritance or polymorphism.
- => we can use structs instead of classes

```
struct position
{
        public int x;
        public int y;
        public int dir;
}


static void Main()
{
        position p;
        p.x = 1;
        p.y = 1;
        p.dir=4;
}
```

Common .Net interfaces

- Icomparable
- CompareTo method
  - Negative integer return value => if the caller is less than the argument
  - Return 0 => Equal to
  - Negative integer return value => greater than

Common .Net interfaces

- IDisposable
- Implemented by classes that must provide an explicit mechanism for releasing resources.
- Dispose method that can be called to explicitly release resources that are explicitly associated with an object

Common .Net interfaces

- IEnumarable
- Iterating through the elements of a collection
- foreach uses an IEnumarable object to iterate through elements