

Systems Development: Object Oriented Programming

(H171 35)

Case Study: Accounts Payable using an Interface

Step 4 – using interface IPayable to process Invoices and Employees polymorphically

* When a class implements an interface, the same *is-a* relationship provided by inheritance applies. Class Employee implements IPayable, so we can say that an Employee *is an* IPayable, as are any classes that extend Employee
  + e.g. SalariedEmployee objects are IPayable objects
* an object of a class that implements an interface may be thought of as an object of the interface type
* objects of any classes derived from the class that implements the interface can also be thought of as objects of the interface type
* **therefore…just as we can assign a reference of a SalariedEmployee object to a base-class Employee variable, we can assign the reference of a SalariedEmployee object to an interface IPayable variable**

Information Point

Inheritance and interfaces are similar in their implementation of the “is-a” relationship. An object of a class that implements an interface may be thought of as an object of that interface type. An object of any derived classes of a class that implements an interface also can be thought of as an object of the interface type.

The *is-a* relationship that exists between base classes and derived classes, and between interfaces and the classes that implement them, holds when passing an object to a method. When a method parameter receives an argument of a base class or interface type, the method polymorphically processes the object received as an argument.

IPayable can be used to process a set of Invoices and Employees polymorphically in a single app.

* Set up an array of 4 IPayable objects
  + IPayable[] payableObjects = new IPayable[4];
* Assign the references of Invoice objects to the first two elements of payableObjects
  + payableObjects[0] = new Invoice("01234", "seat", 2, 375.00M);
  + payableObjects[1] = new Invoice("56789", "tyre", 4, 79.95M);
* Assign the references of SalariedEmployee objects to the remaining two elements of payableObjects
  + payableObjects[3] = new SalariedEmployee("John", "Smith", "111-11-1111", 800.00M);
  + payableObjects[4] = new SalariedEmployee("Lisa", "Barnes", "888-88-8888", 1200.00M);

Information Point

These assignments are allowed because an Invoice is an IPayable, a SalariedEmployee is an Employee and an Employee is an IPayable

* Use a foreach statement to process each IPayable object in payableObjects polmorphically displaying the object as a string along with the payment due

foreach(var currentPayable in payableObjects)

{

//output currentPayable and its appropriate payment amount

Console.WriteLine("{0} \n payment due: {1:C}\n",

currentPayable, currentPayable.GetPaymentAmount());

}

Information Point

Console.WriteLine implicitly invokes method ToString off an IPayable interface reference, even though ToString is not declared in interface IPayable – all references refer to objects that extend object and therefore have a ToString method.

**var** is an implicit type. It aliases any type. The aliased type is determined by the C# compiler. This is a C# feature called **implicitly typed local variables**, which enables the compiler to *infer* a local variable’s type based on the context in which it’s used. This has no performance penalty.