

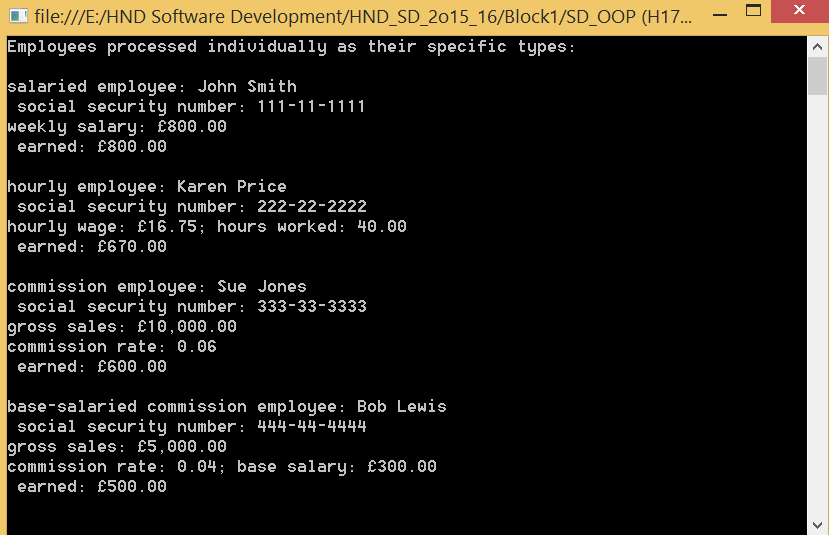
Systems Development: Object Oriented Programming

(H171 35)

Case Study: Payroll system using polymorphism

Step 6 – create a test app

* We are now going to test our Employee hierarchy (by default this will be in the Program.cs file)
* Create an object of each of the 4 concrete classes – SalariedEmployee, HourlyEmployee, CommissionEmployee, and BasePlusCommissionEmployee
* The app will first manipulate these objects via variables of each objects own type (we will then manipulate them polymorphically using an array of Employee variables)
  + Display each employee’s earnings (from the 4 objects created)
    - i.e. Console.WriteLine("{0}\n earned: {1:C}\n", salariedEmployee, salariedEmployee.Earnings());



* Now we are going to manipulate the objects polymorphically
  + Create a 4 element Employee array
    - i.e. Employee[] employees = new Employee[4];
  + Initialise the array with Employees of derived types i.e. the 4 objects created
    - e.g. employees[0] = salariedEmployee;
    - we can assign the references of a derived class object to a base class variable due to the “is a” relationship. A salaried Employee is a Employee.
    - We are treating our employees generically
  + Use a foreach loop to iterate through the array *employees* and invoke methods ToString and Earnings with Employee variable currentEmployee, which is assigned the reference to a different Employee during each iteration
    - All calls to virtual methods ToString and Earnings are resolved at execution time, based on the type of object to which currentEmployee refers (this is known as dynamic binding or late binding). It is coded *generically* in the app, but will be treated specifically at run-time
    - As we process each element, we need to determine if it is of type BasePlusCommissionEmployee as they are to be awarded and additional 10% of their base salary in this month’s salary.

foreach(Employee currentEmployee in employees)

{

Console.WriteLine(currentEmployee); // invokes ToString in derived class

// we need to determine if the elemement is a BasePlusCommissionEmployee

// as they are to be awarded an additional 10% of their base salary in this

// month's pay

if (currentEmployee is BasePlusCommissionEmployee)

{

// downcast Employee reference to BasePlusCommissionEmployee reference

BasePlusCommissionEmployee employee =

(BasePlusCommissionEmployee)currentEmployee;

employee.BaseSalary \*= 1.10M;

Console.WriteLine("new base salary with 10% increase is: {0:C}",

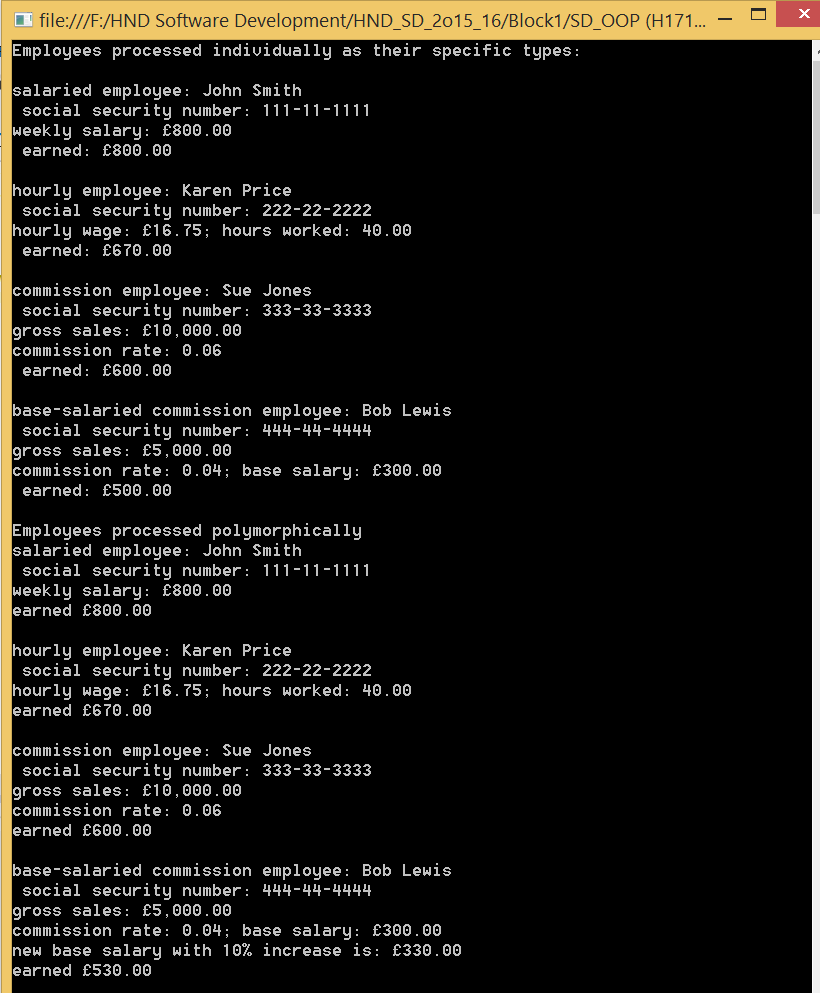
employee.BaseSalary);

}

Console.WriteLine("earned {0:C} \n", currentEmployee.Earnings());

} // end foreach

* We use the *is* operator to determine whether a particular Employee object’s type is BasePlusCommissionEmployee. This is true if the object referenced by currentEmployee *is a* BasePlusCommissionEmployee
* We need to *downcast* currentEmployee from type Employee to type BasePlusCommissonEmployee (casting is only allowed when there is an “is a” relationship) in order to get access to the derived class’s property BaseSalary so we can update the instance variable baseSalary
* attempting to invoke a derived class only method directly on a base class reference is a compilation error i.e. we cannot update the baseSalary for the currentEmployee as Employee does not have a baseSalary or the associated property, therefore we need to treat the object as it’s actual employee type(BasePlusCommissionEmployee) so we can access all of the data
* Using BasePlusCommissionEmployee variable employee, we can access the derived-class-only property BaseSalary to retrieve and update the employee’s base salary with the 10% rise
* In the Console.WriteLine we invoke method Earnings on currentEmployee, which calls the appropriate derived-class object’s Earnings method polymorphically.



Information point

When *downcasting* an object, an InvalidCastException occurs if at execution time the object does not have an *is a* relationship with the type specified in the cast operator. An object can be cast only to its own type or to the type of one of its base classes. You can avoid a potential InvalidCastException by using the **as** operator to perform a downcast rather than a cast operator e.g.

BasePlusCommissionEmployee employee = currentEmployee as BasePlusCommissionEmployee;

employee is assigned a reference to an object that *is a* BasePlusCommissionEmployee, or the value null if currentEmployee is not a BasePlusCommissionEmployee. You can then compare employee with null to determine whether the cast succeeded.

Additional Activity

Every object knows its own type and can access this information through method GetType, which all classes inherit from class *object*. Method GetType returns an object of class Type (of namespace System), which contains information about the objects type, including its class name, the names of its methods, and the name of its base class.

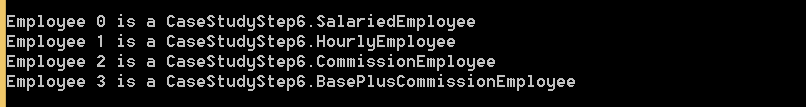
Add the following code to the application to view the types of each object:

for(int j=0; j<employees.Length; j++)

{

Console.WriteLine("Employee {0} is a {1} ", j, employees[j].GetType() );

}



The method GetType is invoked on the object to get its runtime class. Then the method ToString is implicitly invoked on the object returned by GetType.

Information point

Summary of the allowed assignments between base-class and derived-class variables:

Derived-class objects can be treated as if they were base class objects. However, the derived class object can have additional derived class only members. For this reason, assigning a base class reference to a derived class variable is NOT allowed (without an explicit cast), as such an assignment would leave the derived class members undefined for a base class object.

We have discussed 4 ways to assign base class and derived class references to variables of base class and derived class types:

1 – assigning a base class reference to a base class variable – straight forward

2 – assigning a derived class reference to a derived class variable – straight forward

3 – assigning a derived class reference to a base class variable – this is safe because the derived class object “is an” object of it’s base class. However this reference can be used to refer only to base class members. If this code refers to derived class only members through the base class variable, the compiler reports an error

4 – 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 or must be converted using the *as* operator. At execution time, if the object to which the reference refers is NOT a derived class, an exception will occur (unless you use the as operator). The *is* operator can be used to ensure that such a cast is performed only if the object is a derived class