**Chapter 10**

**Exercise 10.12 (Payroll System Modification)**

Create a payroll system to include private instance variable birthDate in class Employee. Create a class Date to represent an employee’s birthday. Add get methods to class Date. Assume that payroll is processed once per month. Create an array of Employee variables to store references to the various employee objects. In a loop, calculate the payroll for each Employee (polymorphically), and add a $100.00 bonus to the person’s payroll amount if the current month is the one in which the Employee’s birthday occurs.

**Exercise 10.13 (Project: Shape Hierarchy)**

Implement the Shape hierarchy each TwoDimensionalShape should contain method getArea to calculate the area of the two-dimensional shape. Each ThreeDimensionalShape should have methods getArea and getVolume to calculate the surface area and volume, respectively, of the three-dimensional shape. Create a program that uses an array of Shape references to objects of each concrete class in the hierarchy. The program should print a text description of the object to which each array element refers. Also, in the loop that processes all the shapes in the array, determine whether each shape is a TwoDimensionalShape or a ThreeDimensionalShape. If it’s a TwoDimensionalShape, display its area. If it’s a ThreeDimensionalShape, display its area and volume.

**Exercise 10.14 (Payroll System Modification)**

Modify the payroll system to include an additional Employee subclass PieceWorker that represents an employee whose pay is based on the number of pieces of merchandise produced. Class PieceWorker should contain private instance variables wage (to store the employee’s wage per piece) and pieces (to store the number of pieces produced). Provide a concrete implementation of method earnings in class PieceWorker that calculates the employee’s earnings by multiplying the number of pieces produced by the wage per piece. Create an array of Employee variables to store references to objects of each concrete class in the new Employee hierarchy. For each Employee, display its String representation and earnings.

**Exercise 10.15 (Accounts Payable System Modification)**

To modify the accounts payable application as described in your exercise, we need to follow a structured approach. The overall task is to integrate classes from the payroll application into the existing accounts payable system, making necessary changes to the class hierarchy and adding functionality that correctly processes invoices and payroll objects.

Here’s how we can approach each of the outlined:

1. **Modify HourlyEmployee and CommissionEmployee to implement Payable**

Modify the HourlyEmployee and CommissionEmployee classes so that they extend Employee and implement the Payable interface. We also need to change their method from earnings to getPaymentAmount, as required by the Payable interface.

1. **Modify BasePlusCommissionEmployee to extend CommissionEmployee**

Now, we modify BasePlusCommissionEmployee to extend the version of CommissionEmployee created in part (a), and implement the logic to adjust the base salary by 10% if the employee is a BasePlusCommissionEmployee.

1. **Modify PayableInterfaceTest to process two Invoices, one SalariedEmployee, one HourlyEmployee, one CommissionEmployee, and one BasePlusCommissionEmployee**

Now we need to modify the PayableInterfaceTest to process these objects polymorphically. We’ll also include logic to increase the base salary of the BasePlusCommissionEmployee by 10% before printing the payment amount.

**Additional Modifications:**

The Invoice class should implement the Payable interface and provide the getPaymentAmount method. For simplicity, here's an example of the Invoice class:

**Exercise 10.16 (Accounts Payable System Modification)**

Modify class Employee to implement interface Payable and declare method getPaymentAmount to invoke method earnings. Method getPaymentAmount would then be inherited by the subclasses in the Employee hierarchy. When getPaymentAmount is called for a particular subclass object, it polymorphically invokes the appropriate earnings method for that subclass and do not modify any of class Employee’s subclasses.

**Making a Difference**

**Exercise 10.17 (CarbonFootprint Interface: Polymorphism)**

Using interfaces, as you learned in this chapter, you can specify similar behaviors for possibly disparate classes. Governments and companies worldwide are becoming increasingly concerned with carbon footprints (annual releases of carbon dioxide into the atmosphere) from buildings burning various types of fuels for heat, vehicles burning fuels for power, and the like. Many scientists blame these greenhouse gases for the phenomenon called global warming. Create three small classes unrelated by inheritance—classes Building, Car and Bicycle. Give each class some unique appropriate attributes and behaviors that it does not have in common with other classes. Write an interface CarbonFootprint with a getCarbonFootprint method. Have each of your classes implement that interface, so that its getCarbonFootprint method calculates an appropriate carbon footprint for that class (check out a few websites that explain how to calculate carbon footprints). Write an application that creates objects of each of the three classes, places references to those objects in ArrayList, then iterates through the ArrayList, polymorphically invoking each object’s getCarbonFootprint method. For each object, print some identifying information and the object’s carbon footprint.