Lab Week 10

Employee Example

Exercise 9.14 (Employee Hierarchy) In this chapter, you studied an inheritance hierarchy in which class BasePlusCommissionEmployee inherited from class CommissionEmployee. However, not all types of employees are CommissionEmployees. In this exercise, you'll create a more general Employee superclass that *factors out* the attributes and behaviors in class CommissionEmployee that are common to all Employees. The common attributes and behaviors for all Employees are firstName, lastName, socialSecurityNumber, getFirstName, getLastName, getSocialSecurityNumber and a portion of method toString. Create a new superclass Employee that contains these instance variables and methods and a constructor. Next, rewrite class CommissionEmployee from Section 9.4.5 as a subclass of Employee. Class CommissionEmployee should contain only the instance variables and methods that are not declared in superclass Employee. Class CommissionEmployee's constructor should invoke class Employee's constructor and CommissionEmployee's toString method should invoke Employee's toString method. Once you've completed these modifications, run the CommissionEmployeeTest and BasePlusCommissionEmployeeTest apps using these new classes to ensure that the apps still display the same results for a CommissionEmployee object and BasePlusCommissionEmployee object, respectively.

Exercise 9.15 (Creating a New Subclass of Employee) Other types of Employees might include Salaried-Employees who get paid a fixed weekly salary, PieceWorkers who get paid by the number of pieces they produce or HourlyEmployees who get paid an hourly wage with time-and-a-half—1.5 times the hourly wage—for hours worked over 40 hours. Create class HourlyEmployee that inherits from class Employee (Exercise 9.14) and has instance variable hours (a double) that represents the hours worked, instance variable wage (a double) that represents the wages per hour, a constructor that takes as arguments a first name, a last name, a social security number, an hourly wage and the number of hours worked, *set* and *get* methods for manipulating the hours and wage, an earnings method to calculate an HourlyEmployee's earnings based on the hours worked and a toString method that returns the HourlyEmployee's String representation. Method setWage should ensure that wage is nonnegative, and setHours should ensure that the value of hours is between 0 and 168 (the total number of hours in a week). Use class HourlyEmployee in a test program that's similar to the one in Fig. 9.5.