JAVA PRACTICAL_6

1. (Composition)Write a java program that contains classes *Date*, *Employee* and *EmployeeTest*.Class Date declares instance variables *month*, *day* and *year* to represent a date. Create a constructor that receives three int parameters for initializing the objects. Provide a *toString* method to obtain the object's String representation. Class Employee has instance variables *firstName*, *lastName*, *birthDate* and *hireDate*. Members firstName and lastName are references to String objects. Members birthDate and hireDate are references to Date objects. Create Employee constructor that takes four parameters representing the first name, last name, birth date and hire date for initializing the objects. Provide a toString to returns a String containing the employee's name and the String representations of the two Date objects. Class EmployeeTest creates two Date objects to represent an Employee's birthday and hire date, respectively. It creates an Employee's object and initializes its instance variables by passing to the constructor two Strings (representing the Employee's first and last names) and two Date objects (representing the birthday and hire date).

2. (Composition)Modify question 1 to perform validation of the month and day. Validate the month—if it's out-of-range, display error report. Validate the day, if the day is incorrect based on the number of days in the particular month (except February 29th which requires special testing for leap years), display error report. Perform the leap year testing for February, if the month is February and the day is 29 and the year is not a leap year, display error report.

3. (Single Inheritance) Write a java app that contains classes Polygon, Rectangle, and myMain. Class Polygon declares instance variables height and width and defines a method setValues for setting height and width values supplied by user. Class rectangle extends class Poygon and defines method area for calculating and returning area of a rectangle. Class myMain defines main method, creates an object and calls other methods for demonstrating their capabilities.

4. (Multilevel Inheritance) Write a java app that contains classes EmployeeA, EmployeeB, EmployeeC and EmployeesTest. Class EmployeeA should declare float variable *salary* initialized with an arbitrary value. Class EmployeeB declares float variable *bonusB* initialized with an arbitrary value and extends class EmployeeA. Class EmployeeC declares float variable *bonusC* initialized with an arbitrary value and extends class EmployeeB. Class EmployeesTest should define main method, create object of class EmployeeC and display the earning of each employee.

5. (Hierarchical Inheritance) Write a java app that contains classes Polygon, Rectangle, Rectangle and myMain. Class Polygon declares instance variables height and width and defines a method setValues for setting height and width values supplied by user. Class Rectangle defines method areaR for calculating and returning area of a rectangle and extends class Poygon. Class Triangle defines method areaT for calculating and returning area of a triangle and extends class Polygon. Class myMain defines main method, creates Rectangle and Triangle's objects and calls the methods for demonstrating their capabilities.

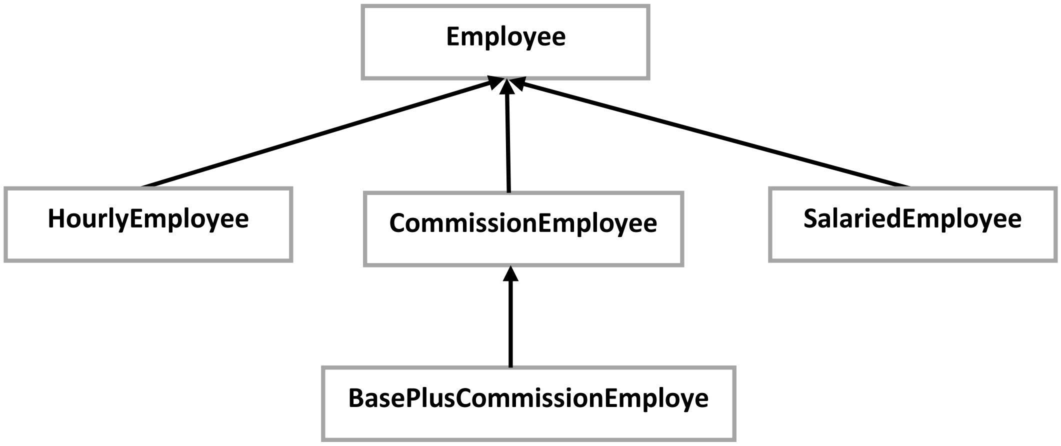
6. (Abstraction and Overriding) Write a java app that contains classes *Polygon*, *Rectangle*, *Triangle* and *myMain*. Class *Polygon* declares instance variables *height* and *width* and defines a method *setValues* for setting height and width values supplied by user. It should also define abstract method *area*. Class *Rectangle* extends class *Polygo*n and implements method *area* for calculating and returning area of a rectangle. Class *Triangle* extends class *Polygon* and implements method *area* for calculating and returning area of a triangle. Class *myMain* defines main method, creates Rectangle and Triangle's objects and calls the methods for demonstrating their capabilities.

7. (Hierarchical Inheritance) Write a java app that contains classes Employee, Programmer, Accountant and MyMain. Class Employee should declare double variable *salary* initialized with an arbitrary value. Class Programmer declares double variable *bonusP* initialized with an arbitrary value and extends class Employee. Class Accountant declares double variable *bonusA* initialized with an arbitrary value and extends class Employee. Class MyMain should define main methdod, create objects of classes Programmer and Accountant and display the earning of each employee.

8. In this question we use an inheritance hierarchy containing types of employees in a company's payroll application to understand the relationship between a superclass and its subclass. In this company, commission employees (who will be represented as objects of a superclass) are paid a percentage of their sales, while base-salaried commission employees (who will be represented as objects of a subclass) receive a base salary plus a percentage of their sales. We divide our problem into five parts.
    a. Declare class CommissionEmployee, which directly inherits from class Object and declares as private instance variables a first name, last name, social security number, commission rate and gross (i.e., total) sales amount.
    b. Declare class BasePlusCommissionEmployee, which also directly inherits from class Object and declares as private instance variables a first name, last name, social security number, commission rate, gross sales amount and base salary. You create this class by writing every line of code the class requires. What problem did you note regarding relationship between class BasePlusCommissionEmployee and CommissionEmployee?
    c. Declare a new BasePlusCommissionEmployee class that extends class CommissionEmployee (i.e., a BasePlusCommissionEmployee is a CommissionEmployee who also has a base salary). Why the code is not running?
    d. Change CommissionEmployee's instance variables to protected. Now BasePlusCommissionEmployee subclass can access that data directly. What is tha drawbacks of using protected instance variables?
    e. Set the CommissionEmployee instance variables back to private to enforce good software engineering. Then show how the BasePlusCommissionEmployee subclass can use CommissionEmployee's public methods to manipulate (in a controlled manner) the private instance variables inherited from CommissionEmployee. What are advantages of using private instance variables in this case?

9. (Abstraction)A company pays its employees on a weekly basis. The employees are of four types: Salaried employees are paid a fixed weekly salary regardless of the number of hours

worked, hourly employees are paid by the hour and receive overtime pay (i.e., 1.5 times their hourly salary rate) for all hours worked in excess of 40 hours, commission employees are paid a percentage of their sales and base-salaried-commission-employees receive a base salary plus a percentage of their sales. For the current pay period, the company has decided to reward salaried-commission employees by adding 10% to their base salaries. The company wants you to write an application that performs its payroll calculations. Use an abstract superclass Employee that includes firstName, lastName, socialSecurityNumber, getFirstName, getLastName, getSocialSecurityNumber, toString, constructor and an abstract method earning. The classes that extend Employee are SalariedEmployee, CommissionEmployee and HourlyEmployee. Class BasePlusCommissionEmployee which extends CommissionEmployee represents the last employee type. The inheritance hierarchy is represented on the figure below.


Employee
HourlyEmployee
CommissionEmployee
SalariedEmployee
BasePlusCommissionEmploye