FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERIG
Department of Computer Engineering

Experiment 6- Based on Inheritance

1. **Course Details:**

Academic Year	2023 - 24	Estimated Time	Experiment No. 6– 02 Hours
Course & Semester	S.E. (COMP) – Sem. III	Subject Name	Skill based lab Course-OOP with Java
Module No.	04	Chapter Title	Inheritance
Experiment Type	Software Performance	Subject Code	CSL304

Name of Student	Mark Lopes	**Roll No.**	9913
Date of Performance:	20/09/2023	**Date of Submission:**	27/09/2023
CO Mapping	**CSL304.4: Implement the concept of inheritance, exception handling and multithreading**		

Timeline (2)	Preparedness (2)	Effort (3)	Result (3)	Total (10)

Problem statement:

1)

PART-A

Since we now have a basic understanding of inheritance and IS-A test, let’s now put them into practice through this simple exercise. In this Experiments, we will implement a simple class hierarchy for users of a **Hospital Management System** (***HMS***). A HMS can help Hospitals to track everything from user *registration* to *patient treatment* to *rooms allotted to in-patient*s and so on. In this experiment, let’s restrict ourselves to implementing only different types of Users. The exercise will be further extended with a ***Billing Component*** with polymorphism.

Users can be of three types: *Patient*, *Doctor*, and *Nurse*. Doctor & Nurse are Staff members (i.e., Doctor IS-A Staff & Nurse IS-A Staff) and all of them (i.e., Patient, Doctor, Nurse, and

Staff) are of type User. Below are details of each class along with information about their instance variables including their data types indicated in parenthesis.

User: id (long), firstName (String), lastName (String), gender (String), email (String)

Patient: patientId (long), insured (boolean)

Staff: staffId (long), yearsOfExperience (int), description (String), salary (double)

Doctor: doctorId (long), specialization (String)

Nurse: nurseId (long)

As part of this experiment, you would implement the above classes. Classes should reflect the **inheritance relationship**. Classes should include the **getters** & **setters** for each of the fields. Make sure to follow correct naming convention for getters & setters as one of the fields is a boolean . Note that for something like doctorId, getter would be named as getDoctorId ('D' in CAPS) and setter would be named as setDoctorId ('D' in CAPS) and similar convention applies for all other methods. Also, do notice how the variables have been named (e.g., yearsOfExperience) as discussed in naming conventions lecture. Finally, make sure to use private access modifiers for all fields and getters & setters would be public (remember ***Information Hiding*** principle!).

PART-B

Let's now extend the HMS system that was implemented in the previous part with **Billing Component**, which would be responsible for billing a patient. Patient & User classes are needed for this exercise. In this exercise, we will put ***polymorphism*** concept & ***instanceof operator*** into practice. This exercise should give you a better feel on how nicely object-oriented programming models real-world scenarios.

When a patient is being billed after treatment, hospitals would apply any *insurance* that the patient may be having. That way the insurance company would pay a part of the medical bill and the remaining will be paid by patient. Insurance policies vary from country to country.

Let's assume we have the following four classes that represent four insurance plans and a patient can buy one of them. Platinum plan has the highest coverage of 0.9, i.e., it covers 90% of the total medical expenses and the patient would pay the remaining 10%. So, if the total medical expense was ₹1000, then the insurance company would pay ₹900 (₹1000 * 0.9) while the patient would pay the remaining ₹100. The coverage offered by GoldPlan, SilverPlan, and BronzePlan are 0.8, 0.7, and 0.6 respectively. All of the below four classes would ***extend*** another class called **HealthInsurancePlan**. Coverage is indicated by a *double* field named 'coverage' and would be part of HealthInsurancePlan class. Corresponding getter (getCoverage) & setter (setCoverage) should also be provided for 'coverage'. Making 'coverage' and its getter & setter part of HealthInsurancePlan and not part of its sub-classes would help avoid code duplication (a benefit of inheritance). However, specific sub-classes (e.g., PlatinumPlan) would set the appropriate value for the coverage field and they can do it from their constructors.

PlatinumPlan

GoldPlan

SilverPlan

BronzePlan

Next, you need to add a new field called ‘insurancePlan’, which is of type HealthInsurancePlan to the **Patient** class (implemented in previous Part) and leave this field uninitialized, i.e., gets a default of null. This field would indicate the insurance plan that a patient has. So, it indicates a ***HAS-A*** relationship, This field 'insurancePlan' would be set by setter (setInsurancePlan) and also has corresponding getter (getInsurancePlan). getter would have a return type of HealthInsurancePlan while setter would have a parameter whose type is HealthInsurancePlan. As you can see, setInsurancePlan(HealthInsurancePlan) would give us the polymorphism benefit as we can pass an instance of any of the above 4 classes.

Next, you would implement the billing logic, which goes into a new class called **Billing.java**, which would have a single *static* method called ***computePaymentAmount.*** Input to this method is a Patient object & 'amount', which is a double value indicating the amount the patient is billed *before* applying insurance. The method returns a *double[]* and its first element would have the amount that the insurance company would pay while second element would have the amount that patient has to pay. As mentioned above, you need to make use of 'coverage' of the insurance plan. Note that some patients may not have any insurance plan in which case 'patient.getInsurancePlan()’ would return a null and you need to have the necessary logic in place to ensure that it does not lead to a NullPointerException. Once the patient's part is computed, additional discount should be applied on the patient's part depending on their insurance plan, which is as follows:

PlatinumPlan: ₹5000 discount

GoldPlan: ₹4000 discount

SilverPlan: ₹3000 discount

BronzePlan: ₹2500 discount

If the patient does not have any insurance plan, then a discount of ₹2000 is applied.

HINT: You can use instanceof operator to identify the correct insurance plan and then you can apply the appropriate discount for that particular insurance plan.

Finally, HealthInsurancePlan class has a field called 'offeredBy' which is of type InsuranceBrand (code provided with exercise). You can ignore both this field and the InsuranceBrand class as they are not relevant to this exercise and will come into play in one of the later exercises. In real-world, a health insurance policy is offered by some company (e.g., in India it would be a company like Star Health) and this field represents that company.

```
public class InsuranceBrand {

        private long id;
        private String name;

                public long getId() {

                return id;

        }

        public void setId(long id) {

                this.id = id;

        }

        public String getName() {

                return name;

        }

        public void setName(String name) {

                this.name = name;

        }

}
```

PART-C

This is a very simple exercise, and it would enable us to use ***abstract classes*** & ***methods***. In this exercise, we will extend the HMS system. Earlier we introduced a *billing component* to bill the patients, which was dependent on their insurance plan (***platinum/gold/silver/bronze***). Let's now build something for staff members. Like patients, staff members would also need health insurance. So, as employees of the Hospital, every month staff members would be paying a **premium** (i.e., some amount of money) towards their health insurance. In this exercise we would compute this **monthly premium**, which would be dependent on the chosen insurance plan (i.e., ***platinum/gold/silver/bronze***) along with **monthly salary** of the staff member. For platinum it would be 8% of the salary, for gold it would be 7% of salary, for silver it would be 6% of salary and for bronze it would be 5% of salary.
Implementation specifics:

Move insured & insurancePlan fields from Patient.java to User.java. Earlier, these fields were specific to Patient. But, now since we are talking about staff members also having health insurance and since both Staff & Patient are sub-classes of User, we can now move these two fields along with their getters and setters into User class.

Let's introduce monthly premium calculation logic in classes corresponding to different insurance plans as the logic is dependent on insurance plan chosen in addition to salary. So, let's introduce the following ***abstract*** method in the super class HealthInsurancePlan and this method would be overridden in specific sub-classes.

public abstract double computeMonthlyPremium(double salary);

You would also need InsuranceBrand.java from previous exercise and there is no change in that class though.

PART-D

In this exercise, we will make use of interfaces and will extend the earlier exercise where we used abstract classes & methods to incorporate logic to compute monthly health insurance premium for staff members. There we used only *salary* information of the User. Now, let's extend it to also include two more parameters: (i) ***age*** and (ii) ***whether or not the user smokes***. Rules for these parameters can vary across insurance plans (***platinum/gold/silver/bronze***) and from one insurance company to another. Let's assume that we have only one insurance company called **Star Health** and its rules for the different plans are as below.

Platinum Plan: If age > 55 then add 2000 to premium. If user smokes then add 1000

Gold Plan: If age > 55 then add 1500 to premium. If user smokes then add 900

Silver Plan: If age > 55 then add 1000 to premium. If user smokes then add 800

Bronze Plan: If age > 55 then add 500 to premium. If user smokes then add 700

Essentially, premium increases if the user is over 55 years of age. Similarly, if user smokes, then his/her premium would also be increased. Note that age & smoking are independent here, i.e., if age < 55 and user smokes, then we increase the premium only for smoking.

As we can see the above rules are specific to **Star Health** which IS-A type of Insurance Brand. Recall from earlier exercises that we already have a class called InsuranceBrand. So, the implementation for the above rules would be provided in **Star Health** in a method called ***computeMonthlyPremium***. Let's also assume that InsuranceBrand is very generic and can be used across class hierarchies. In this case, it may be better to represent it as an interface. So, we can change it from a class to interface and it would declare only a single method computeMonthlyPremium,

Also, since we are making InsuranceBrand an interface, its existing fields (id & name) and the corresponding getters & setters can be moved into the subclass **Star Health**.

public double ***computeMonthlyPremium***(HealthInsurancePlan insurancePlan, int age, boolean smoking);

Recall that in the previous exercise, monthly premium was computed using only salary & insurance plan using a similarly named method ***computeMonthlyPremium(double)***, which was implemented by individual insurance plan classes

(PlatinumPlan/GoldPlan/SilverPlan/BronzePlan). This method should now additionally take the two new parameters 'age' & 'smoking' too.

getOfferedBy() is from the super class and it would return an instance of InsuranceBrand. Notice, that we are passing 'this' reference to computeMonthlyPremium as it is expecting a **HealthInsurancePlan** instance and in this case it would be an instance of PlatinumPlan.

Finally, add age (int) & smoking (boolean) fields in User.java and make sure setters & getters are added with correct conventions. Although, these fields are added in User class, they really do not serve any purpose in this exercise as computeMonthlyPremium above takes age & smoking values separately.

Overall just to summarize, we have two class hierarchies here: (i) an abstract class HealthInsurancePlan and the specific insurance plans as its subtypes (ii) an interface InsuranceBrand and its subtypes (Star Health). HealthInsurancePlan defines both state & behavior while InsuranceBrand defines just the behavior as it should be.

CODE:

// Main.java

```java
public class Main {
    public static void main(String[] args) {
        // Create HealthInsurancePlan objects
        HealthInsurancePlan platinumPlan = new PlatinumPlan();
        HealthInsurancePlan goldPlan = new GoldPlan();
        HealthInsurancePlan silverPlan = new SilverPlan();
        HealthInsurancePlan bronzePlan = new BronzePlan();

        // Create a Patient with insurance plan
        Patient patientWithInsurance = new Patient();
        patientWithInsurance.setPatientId(1);
        patientWithInsurance.setInsured(true);
        patientWithInsurance.setInsurancePlan(platinumPlan);

        // Create a Patient without insurance plan
        Patient patientWithoutInsurance = new Patient();
        patientWithoutInsurance.setPatientId(2);
        patientWithoutInsurance.setInsured(false);

        // Set patient age and smoking status
        int patientAge = 40; // Set the age of the patient
        boolean patientSmokes = true; // Set the smoking status of the
patient

        // Calculate payment amount for patients with age and smoking status
        double[] paymentWithInsurance =
Billing.computePaymentAmount(patientWithInsurance, 500000, patientAge,
patientSmokes);
        double[] paymentWithoutInsurance =
Billing.computePaymentAmount(patientWithoutInsurance, 80000, patientAge,
patientSmokes);

        // Print payment details
        System.out.println("Patient 1 (with insurance) Payment Details:");
        System.out.println("Insurance Pay: " + paymentWithInsurance[0]);
        System.out.println("Patient Pay: " + paymentWithInsurance[1]);

        System.out.println("\nPatient 2 (without insurance) Payment
Details:");
        System.out.println("Insurance Pay: " + paymentWithoutInsurance[0]);
        System.out.println("Patient Pay: " + paymentWithoutInsurance[1]);

        // Create a Doctor
        Doctor doctor = new Doctor();
        doctor.setDoctorId(1);
        doctor.setFirstName("Vivian");
        doctor.setLastName("Ludrick");
        doctor.setGender("Male");
        doctor.setEmail("dummy@gmail.com");
        doctor.setYearsOfExperience(5);
        doctor.setSalary(110000.0);
        doctor.setSpecialization("Surgeon");

        // Create a Nurse
        Nurse nurse = new Nurse();
        nurse.setNurseId(2);
        nurse.setFirstName("Jonathan");
        nurse.setLastName("Gomes");
        nurse.setGender("Male");
        nurse.setEmail("anothermail@gmail.com");
        nurse.setYearsOfExperience(18);
        nurse.setSalary(100000.0);

        // Print Doctor and Nurse details
        System.out.println("\nDoctor Details:");
        System.out.println("Doctor ID: " + doctor.getDoctorId());
        System.out.println("Doctor Name: " + doctor.getFirstName() + " " +
doctor.getLastName());
        System.out.println("Specialization: " +
doctor.getSpecialization());
        System.out.println("Salary: " + doctor.getSalary());
        System.out.println("\nNurse Details:");
        System.out.println("Nurse ID: " + nurse.getNurseId());
        System.out.println("Nurse Name: " + nurse.getFirstName() + " " +
nurse.getLastName());
        System.out.println("Salary: " + nurse.getSalary());
    }
}
```

// User.java

```
public class User
{
  private long id;
  private String firstName;
  private String lastName;
  private String gender;
  private String email;

  public long getUserId()
  {
    return id;
  }

  public void setUserId(long id)
  {
    this.id = id;
  }

  public String getFirstName()
  {
    return firstName;
  }

  public void setFirstName(String firstName)
  {
    this.firstName = firstName;
  }

  public String getLastName()
  {
    return lastName;
  }

  public void setLastName(String lastName)
  {
```

```java
    this.lastName = lastName;
  }

  public String getGender()
  {
    return gender;
  }

  public void setGender(String gender)
  {
    this.gender = gender;
  }

  public String getEmail()
  {
    return email;
  }

  public void setEmail(String email)
  {
    this.email = email;
  }

}
```

// Staff.java

```java
public class Staff extends User {
    private long staffId;
    private int yearsOfExperience;
    private double salary;

    public long getStaffId() {
        return staffId;
    }

    public void setStaffId(long staffId) {
        this.staffId = staffId;
    }

    public int getYearsOfExperience() {
        return yearsOfExperience;
    }
    public void setYearsOfExperience(int yearsOfExperience) {
        this.yearsOfExperience = yearsOfExperience;
    }



    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    public HealthInsurancePlan getInsurancePlan() {
        return null;
    }
}
```

// Patient.java

```java
public class Patient extends User {
    private long patientId;
    private boolean insured;
    private HealthInsurancePlan insurancePlan;

    public long getPatientId() {
        return patientId;
    }

    public void setPatientId(long patientId) {
        this.patientId = patientId;
    }

    public boolean isInsured() {
        return insured;
    }

    public void setInsured(boolean insured) {
        this.insured = insured;
    }

    public HealthInsurancePlan getInsurancePlan() {
        return insurancePlan;
    }
    public void setInsurancePlan(HealthInsurancePlan insurancePlan) {
        this.insurancePlan = insurancePlan;
    }
}
```

// Doctor.java

```java
public class Doctor extends Staff {
    private long doctorId;
    private String specialization;
    private int yearsOfExperience;

    private double salary;

    public long getDoctorId() {
        return doctorId;
    }

    public void setDoctorId(long doctorId) {
        this.doctorId = doctorId;
    }

    public String getSpecialization() {
        return specialization;
    }

    public void setSpecialization(String specialization) {
        this.specialization = specialization;
    }

    public int getYearsOfExperience() {
        return yearsOfExperience;
    }

    public void setYearsOfExperience(int yearsOfExperience) {
        this.yearsOfExperience = yearsOfExperience;
    }


    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    public void setInsurancePlan(HealthInsurancePlan platinumPlan) {
    }
}
```

// Nurse.java

```java
public class Nurse extends Staff {
    private long nurseId;
    private int yearsOfExperience;

    private double salary;

    public long getNurseId() {
        return nurseId;
    }

    public void setNurseId(long nurseId) {
        this.nurseId = nurseId;
    }

    public int getYearsOfExperience() {
        return yearsOfExperience;
    }

    public void setYearsOfExperience(int yearsOfExperience) {
        this.yearsOfExperience = yearsOfExperience;
    }


    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        this.salary = salary;
    }

    public void setInsurancePlan(HealthInsurancePlan silverPlan) {
    }
}
```

// HealthInsurancePlan.java

```java
public class HealthInsurancePlan {
    private double coverage;

    public HealthInsurancePlan(double coverage) {
        this.coverage = coverage;
    }

    public double getCoverage() {
        return coverage;
    }

    public void setCoverage(double coverage) {
        this.coverage = coverage;
    }

    // Default implementation of the monthly premium calculation method
    public double computeMonthlyPremium(double salary, int age, boolean
smoking) {
        return 0.0;
    }
}
```

// BronzePlan.java

```java
public class BronzePlan extends HealthInsurancePlan {
    public BronzePlan() {
        super(0.6); // Set coverage for BronzePlan
    }

    public double computeMonthlyPremium(double salary, int age, boolean
smoking) {
        // Calculate premium based on age and smoking status
        double basePremium = salary * 0.08; // 8% of salary
        if (age > 55) {
            basePremium += 2000; // Additional premium for age over 55
        }
        if (smoking) {
            basePremium += 1000; // Additional premium for smoking
        }
        return basePremium;
    }
}
```

// SilverPlan.java

```java
public class SilverPlan extends HealthInsurancePlan {
    public SilverPlan() {
        super(0.7); // Set coverage for SilverPlan
    }

    public double computeMonthlyPremium(double salary, int age, boolean
smoking) {
        // Calculate premium based on age and smoking status
        double basePremium = salary * 0.08; // 8% of salary
        if (age > 55) {
            basePremium += 2000; // Additional premium for age over 55
        }
        if (smoking) {
            basePremium += 1000; // Additional premium for smoking
        }
        return basePremium;
    }
}
```

// GoldPlan.java

```java
public class GoldPlan extends HealthInsurancePlan {
    public GoldPlan() {
        super(0.8); // Set coverage for GoldPlan
    }

    public double computeMonthlyPremium(double salary, int age, boolean
smoking) {
        // Calculate premium based on age and smoking status
        double basePremium = salary * 0.08; // 8% of salary
        if (age > 55) {
            basePremium += 2000; // Additional premium for age over 55
        }
        if (smoking) {
            basePremium += 1000; // Additional premium for smoking
        }
        return basePremium;
    }
}
```

// PlatinumPlan.java

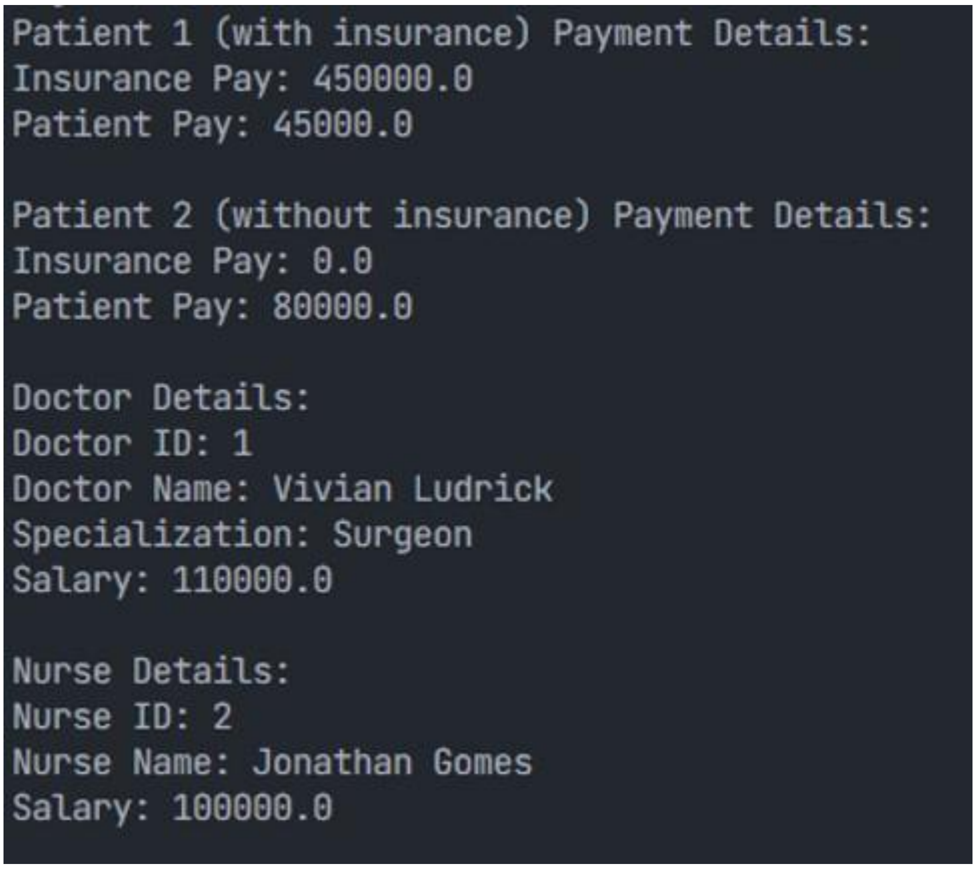
```java
public class PlatinumPlan extends HealthInsurancePlan {
    public PlatinumPlan() {
        super(0.9); // Set coverage for PlatinumPlan
    }
    public double computeMonthlyPremium(double salary, int age, boolean
smoking) {
        // Calculate premium based on age and smoking status
        double basePremium = salary * 0.08; // 8% of salary
        if (age > 55) {
            basePremium += 2000; // Additional premium for age over 55
        }
        if (smoking) {
            basePremium += 1000; // Additional premium for smoking
        }
        return basePremium;
    }
}
```

// HealthCareSystem.java

```java
public class HealthCareSystem {
    private Patient[] patients;
    private Doctor[] doctors;
    private Nurse[] nurses;
    private Staff[] staffList;
    private int patientCount;
    private int doctorCount;
    private int nurseCount;
    private int staffCount;

    public HealthCareSystem(int maxPatients, int maxDoctors, int maxNurses,
int maxStaff) {
        patients = new Patient[maxPatients];
        doctors = new Doctor[maxDoctors];
        nurses = new Nurse[maxNurses];
        staffList = new Staff[maxStaff];
        patientCount = 0;
        doctorCount = 0;
        nurseCount = 0;
        staffCount = 0;
    }

    public void addPatient(Patient patient) {
        if (patientCount < patients.length) {
            patients[patientCount] = patient;
            patientCount++;
        } else {
            System.out.println("Patient capacity reached. Cannot add more
patients.");
```

```java
        }
    }

    public void addDoctor(Doctor doctor) {
        if (doctorCount < doctors.length) {
            doctors[doctorCount] = doctor;
            doctorCount++;
        } else {
            System.out.println("Doctor capacity reached. Cannot add more
doctors.");
        }
    }

    public void addNurse(Nurse nurse) {
        if (nurseCount < nurses.length) {
            nurses[nurseCount] = nurse;
            nurseCount++;
        } else {
            System.out.println("Nurse capacity reached. Cannot add more
nurses.");
        }
    }

    public void addStaff(Staff staff) {
        if (staffCount < staffList.length) {
            staffList[staffCount] = staff;
            staffCount++;
        } else {
            System.out.println("Staff capacity reached. Cannot add more
staff.");
        }
    }

    public Patient getPatientById(long patientId) {
        for (int i = 0; i < patientCount; i++) {
            if (patients[i].getPatientId() == patientId) {
                return patients[i];
            }
        }
        return null; // Patient not found
    }
}
```

OUTPUT:



```
Patient 1 (with insurance) Payment Details:
Insurance Pay: 450000.0
Patient Pay: 45000.0

Patient 2 (without insurance) Payment Details:
Insurance Pay: 0.0
Patient Pay: 80000.0

Doctor Details:
Doctor ID: 1
Doctor Name: Vivian Ludrick
Specialization: Surgeon
Salary: 110000.0

Nurse Details:
Nurse ID: 2
Nurse Name: Jonathan Gomes
Salary: 100000.0
```

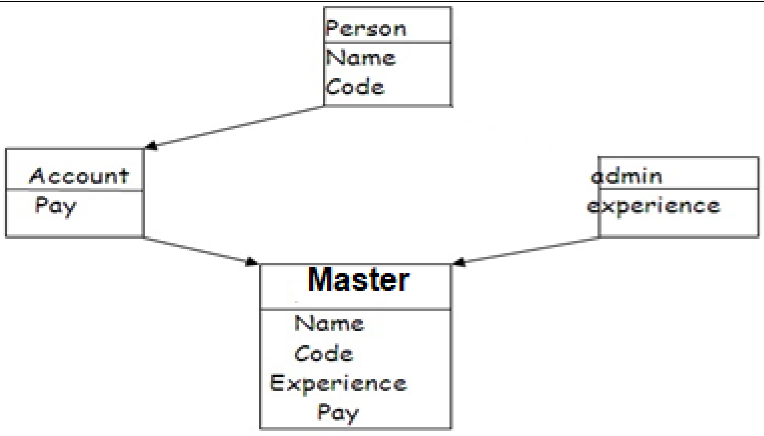
2)

a)

A publishing company that markets both book and audiocassette versions of its works. Create a class **publication** that stores the title (a string) and price (type float) of a publication. From this class derive two classes: **book**, which adds a page count (type int), and **tape**, which adds a playing time in minutes (type float). Each of these three classes should have a getdata() function to get its data from the user at the keyboard, and a putdata() function to display its data. Write a main() program to test the book and tape classes by creating instances of them, asking the user to fill in data with getdata(), and then displaying the data with putdata().

b)

Consider a class network of the following figure. The class master derives information from both account and admin classes which in turn derives information from the class person. Define all the four classes and write a program to create, update and display the information contained in master objects.



c)

Create an abstract class **'Instrument'** which is having the abstract function **play().**
Create three more sub classes from Instrument which is **'Piano', ' Flute' , 'Guitar'**.
Override the play( ) method inside all three classes printing a message,
"Piano is playing tan tan tan tan " for Piano class
"Flute is playing toot toot toot toot" for Flute class
"Guitar is playing tin tin tin " for Guitar class

You must not allow the user to declare an object of Instrument class. Create an array of 10 Instruments. Assign different type of instrument to Instrument reference. Check for the polymorphic behavior of the play method. Use the **instanceof** operator to print that which object stored at which index of instrument array.