1. Create a class named 'Member' having the following members:
Data members
1 - Name
2 - Age
3 - Phone number
4 - Address
5 - Salary
It also has a method named 'printSalary' which prints the salary of the members.
Two classes 'Employee' and 'Manager' inherits the 'Member' class. The 'Employee' and 'Manager' classes have data members 'specialization' and 'department' respectively. Now, assign name, age, phone number, address and salary to an employee and a manager by making an object of both of these classes and print the same.

```
class Member {
    String name;
    int age;
    String phoneNumber;
    String address;
    double salary;

    Member(String name, int age, String phoneNumber, String address, double salary) {
        this.name = name;
        this.age = age;
        this.phoneNumber = phoneNumber;
        this.address = address;
        this.salary = salary;
    }

    void printDetails() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
        System.out.println("Phone Number: " + phoneNumber);
        System.out.println("Address: " + address);
        printSalary();
    }

    void printSalary() {
        System.out.println("Salary: " + salary);
    }
}

class Employee extends Member {
    String specialization;
    String department;

    Employee(String name, int age, String phoneNumber, String address, double salary, String specialization)
{
        super(name, age, phoneNumber, address, salary);
        this.specialization = specialization;

    }

    @Override
    void printDetails() {
        super.printDetails();
```

```
        System.out.println("Specialization: " + specialization);
    }
}

class Manager extends Member {
    String department;

    Manager(String name, int age, String phoneNumber, String address, double salary, String department) {
        super(name, age, phoneNumber, address, salary);
        this.department = department;
    }

    @Override
    void printDetails() {
        super.printDetails();
        System.out.println("Department: " + department);
    }
}

public class Main {
    public static void main(String[] args) {

        Employee employeeObj = new Employee("Shobika", 25, "6789123455", "123 Main St", 50000,
"Software Development");

        Manager managerObj = new Manager("Priya", 35, "9876543210", "456 Oak St", 70000, "Human
Resources");


        System.out.println("Employee Details:");
        employeeObj.printDetails();
        System.out.println();

        System.out.println("Manager Details:");
        managerObj.printDetails();
    }
}
```

```
Employee Details:
Name: Shobika
Age: 25
Phone Number: 6789123455
Address: 123 Main St
Salary: 50000.0
Specialization: Software Development

Manager Details:
Name: Priya
Age: 35
Phone Number: 9876543210
Address: 456 Oak St
Salary: 70000.0
Department: Human Resources
```

2. You are developing a banking application in Java. Design a class hierarchy that represents different account types such as SavingsAccount, CheckingAccount, and LoanAccount.

Each account should have basic functionality like deposit, withdraw, and check balance. Ensure that your design follows appropriate use of interfaces and inheritance.

```
interface AccountOperations {
   void deposit(double amount);

   void withdraw(double amount);

   double checkBalance();
}

// Base class representing a generic account
class Account implements AccountOperations {
   protected double balance;

   @Override
   public void deposit(double amount) {
      balance += amount;
      System.out.println("Deposited: " + amount);
   }

   @Override
   public void withdraw(double amount) {
      if (amount <= balance) {
         balance -= amount;
         System.out.println("Withdrawn: " + amount);
      } else {
         System.out.println("Insufficient funds. Withdrawal failed.");
      }
   }

   @Override
   public double checkBalance() {
      System.out.println("Current Balance: " + balance);
      return balance;
   }
}


class SavingsAccount extends Account {
   private double interestRate;

   // Constructor
   public SavingsAccount(double initialBalance, double interestRate) {
      this.balance = initialBalance;
      this.interestRate = interestRate;
   }

   // Additional method for calculating and adding interest
   public void addInterest() {
```

```
        double interest = balance * interestRate;
        deposit(interest);
        System.out.println("Interest added: " + interest);
    }
}


class CheckingAccount extends Account {
    private double overdraftLimit;

    // Constructor
    public CheckingAccount(double initialBalance, double overdraftLimit) {
        this.balance = initialBalance;
        this.overdraftLimit = overdraftLimit;
    }


    @Override
    public void withdraw(double amount) {
        double availableFunds = balance + overdraftLimit;
        if (amount <= availableFunds) {
            balance -= amount;
            System.out.println("Withdrawn: " + amount);
        } else {
            System.out.println("Insufficient funds. Withdrawal failed.");
        }
    }
}

// LoanAccount class inheriting from Account
class LoanAccount extends Account {
    private double interestRate;


    public LoanAccount(double initialBalance, double interestRate) {
        this.balance = initialBalance;
        this.interestRate = interestRate;
    }


    public void deductInterest() {
        double interest = balance * interestRate;
        withdraw(interest);
        System.out.println("Interest deducted: " + interest);
    }
}

// Main class for testing
public class Main {
    public static void main(String[] args) {
        // Test SavingsAccount
        SavingsAccount savingsAccount = new SavingsAccount(1000, 0.05);
        System.out.println("SavingsAccount :");
        savingsAccount.deposit(500);
```

```
        savingsAccount.checkBalance();
        savingsAccount.addInterest();
        savingsAccount.checkBalance();
        System.out.println();

        // Test CheckingAccount
        CheckingAccount checkingAccount = new CheckingAccount(2000, 500);
        System.out.println("CheckingAccount :");
        checkingAccount.withdraw(1500);
        checkingAccount.checkBalance();
        System.out.println();

        // Test LoanAccount
        LoanAccount loanAccount = new LoanAccount(5000, 0.1);
        System.out.println("LoanAccount :");
        loanAccount.deductInterest();
        loanAccount.checkBalance();
        System.out.println();
    }
}
```

```
SavingsAccount :
Deposited: 500.0
Current Balance: 1500.0
Deposited: 75.0
Interest added: 75.0
Current Balance: 1575.0

CheckingAccount :
Withdrawn: 1500.0
Current Balance: 500.0

LoanAccount :
Withdrawn: 500.0
Interest deducted: 500.0
Current Balance: 4500.0


Process finished with exit code 0
```

3. You are tasked with designing a university enrollment system in Java. Implement a class hierarchy that includes a base class Person and two subclasses, Student and Professor and a Course class. Each class should have the necessary attributes. Each course should have a list of prerequisites and enrolled students.
Your tasks are as follows:
i) Students should only be enrolled if they have completed all the required prerequisites. In the course class, include logic for enrolling students.
ii) Display enrolled students in a particular with relevant information.

```
import java.util.ArrayList;
import java.util.List;

class Person {
    protected String name;
    protected int age;


    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }


    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}

class Student extends Person {
    private List<Course> enrolledCourses;


    public Student(String name, int age) {
        super(name, age);
        enrolledCourses = new ArrayList<>();
    }


    public List<Course> getEnrolledCourses() {
        return enrolledCourses;
    }


    public void enrollCourse(Course course) {
        if (course.arePrerequisitesMet(this)) {
            enrolledCourses.add(course);
            System.out.println(name + " has been enrolled in the course: " + course.getCourseName());
        } else {
            System.out.println(name + " cannot be enrolled in the course: " + course.getCourseName() +
                    ". Prerequisites not met.");
        }
    }
}

class Professor extends Person {
    private String department;


    public Professor(String name, int age, String department) {
```

```
        super(name, age);
        this.department = department;
    }


    public String getDepartment() {
        return department;
    }
}

class Course {
    private String courseName;
    private List<Course> prerequisites;
    private List<Student> enrolledStudents;


    public Course(String courseName) {
        this.courseName = courseName;
        prerequisites = new ArrayList<>();
        enrolledStudents = new ArrayList<>();
    }


    public String getCourseName() {
        return courseName;
    }


    public void addPrerequisite(Course prerequisite) {
        prerequisites.add(prerequisite);
    }


    public void enrollStudent(Student student) {
        if (!enrolledStudents.contains(student)) {
            enrolledStudents.add(student);
            System.out.println("Student " + student.getName() + " enrolled in course: " + courseName);
        } else {
            System.out.println("Student " + student.getName() + " is already enrolled in course: " +
courseName);
        }
    }


    public boolean arePrerequisitesMet(Student student) {
        for (Course prerequisite : prerequisites) {
            if (!student.getEnrolledCourses().contains(prerequisite)) {
                return false;
            }
        }
        return true;
    }
```

```
    public void displayEnrolledStudents() {
        System.out.println("Enrolled students in course " + courseName + ":");
        for (Student student : enrolledStudents) {
            System.out.println("Name: " + student.getName() + ", Age: " + student.getAge());
        }
    }
}

public class Main {
    public static void main(String[] args) {
        // Create students
        Student student1 = new Student("Shobika", 20);
        Student student2 = new Student("Priya", 22);

        // Create courses
        Course math101 = new Course("Math 101");
        Course physics101 = new Course("Physics 101");
        Course advancedMath = new Course("Advanced Math");


        advancedMath.addPrerequisite(math101);
        advancedMath.addPrerequisite(physics101);


        math101.enrollStudent(student1);
        physics101.enrollStudent(student1);
        advancedMath.enrollStudent(student1);

        physics101.enrollStudent(student2);


        math101.displayEnrolledStudents();
        physics101.displayEnrolledStudents();
        advancedMath.displayEnrolledStudents();
    }
}
```

```
Student Shobika enrolled in course: Math 101
Student Shobika enrolled in course: Physics 101
Student Shobika enrolled in course: Advanced Math
Student Priya enrolled in course: Physics 101
Enrolled students in course Math 101:
Name: Shobika, Age: 20
Enrolled students in course Physics 101:
Name: Shobika, Age: 20
Name: Priya, Age: 22
Enrolled students in course Advanced Math:
Name: Shobika, Age: 20

Process finished with exit code 0
```