```
1   // Member class
class Member {
    // Data members
    String name;
    int age;
    String phoneNumber;
    String address;
    double salary;

    // Constructor to initialize data members
    public 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;
    }

    // Method to print salary
    public void printSalary() {
        System.out.println("Salary: $" + salary);
    }
}

// Employee class inheriting from Member
class Employee extends Member {
    // Additional data member
    String specialization;

    // Constructor to initialize data members of Employee class
```

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

// Manager class inheriting from Member
class Manager extends Member {
    // Additional data member
    String department;

    // Constructor to initialize data members of Manager class
    public Manager(String name, int age, String phoneNumber, String address, double salary, String department) {
        super(name, age, phoneNumber, address, salary);
        this.department = department;
    }
}

// Main class
public class Main {
    public static void main(String[] args) {
        // Creating an object of Employee class
        Employee employee = new Employee("John Doe", 25, "1234567890", "123 Main St", 50000, "Software Developer");

        // Creating an object of Manager class
        Manager manager = new Manager("Jane Smith", 35, "9876543210", "456 Park Ave", 80000, "HR");

        // Printing details for Employee
```

```
        System.out.println("Employee Details:");
        System.out.println("Name: " + employee.name);
        System.out.println("Age: " + employee.age);
        System.out.println("Phone Number: " + employee.phoneNumber);
        System.out.println("Address: " + employee.address);
        employee.printSalary();  // Using the printSalary method from Member class
        System.out.println("Specialization: " + employee.specialization);

        System.out.println();  // Adding a line break for better readability

        // Printing details for Manager
        System.out.println("Manager Details:");
        System.out.println("Name: " + manager.name);
        System.out.println("Age: " + manager.age);
        System.out.println("Phone Number: " + manager.phoneNumber);
        System.out.println("Address: " + manager.address);
        manager.printSalary();  // Using the printSalary method from Member class
        System.out.println("Department: " + manager.department);
    }
}
2, // Interface for common account functionalities
interface Account {
    void deposit(double amount);
    void withdraw(double amount);
    double checkBalance();
}

// Base class representing a generic bank account
abstract class BankAccount implements Account {
    protected double balance;
```

```
    public BankAccount(double initialBalance) {
        this.balance = initialBalance;
    }

    @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.");
        }
    }

    @Override
    public double checkBalance() {
        return balance;
    }
}

// SavingsAccount class inheriting from BankAccount
class SavingsAccount extends BankAccount {
    private double interestRate;

    public SavingsAccount(double initialBalance, double interestRate) {
```

```
        super(initialBalance);
        this.interestRate = interestRate;
    }

    public void applyInterest() {
        double interest = balance * interestRate;
        deposit(interest);
        System.out.println("Interest applied: $" + interest);
    }
}

// CheckingAccount class inheriting from BankAccount
class CheckingAccount extends BankAccount {
    private double overdraftLimit;

    public CheckingAccount(double initialBalance, double overdraftLimit) {
        super(initialBalance);
        this.overdraftLimit = overdraftLimit;
    }

    @Override
    public void withdraw(double amount) {
        if (amount <= balance + overdraftLimit) {
            balance -= amount;
            System.out.println("Withdrawn: $" + amount);
        } else {
            System.out.println("Exceeded overdraft limit.");
        }
    }
}
```

```
// LoanAccount class inheriting from BankAccount
class LoanAccount extends BankAccount {
    private double interestRate;

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

    @Override
    public void deposit(double amount) {
        double effectiveAmount = amount * (1 - interestRate);
        super.deposit(effectiveAmount);
        System.out.println("Effective amount deposited: $" + effectiveAmount);
    }
}

// Main class for testing
public class Main {
    public static void main(String[] args) {
        SavingsAccount savingsAccount = new SavingsAccount(1000, 0.05);
        CheckingAccount checkingAccount = new CheckingAccount(2000, 500);
        LoanAccount loanAccount = new LoanAccount(5000, 0.1);

        System.out.println("Initial Balances:");
        System.out.println("Savings Account: $" + savingsAccount.checkBalance());
        System.out.println("Checking Account: $" + checkingAccount.checkBalance());
        System.out.println("Loan Account: $" + loanAccount.checkBalance());

        savingsAccount.deposit(200);
        savingsAccount.applyInterest();
```

```
        checkingAccount.withdraw(1500);

        loanAccount.deposit(1000);

        System.out.println("Final Balances:");
        System.out.println("Savings Account: $" + savingsAccount.checkBalance());
        System.out.println("Checking Account: $" + checkingAccount.checkBalance());
        System.out.println("Loan Account: $" + loanAccount.checkBalance());
    }
}
```

3, import java.util.ArrayList;
import java.util.List;

```
class Person {
    private String name;
    private 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 String studentId;

    public Student(String name, int age, String studentId) {
        super(name, age);
        this.studentId = studentId;
    }

    public String getStudentId() {
        return studentId;
    }
}

class Professor extends Person {
    private String employeeId;

    public Professor(String name, int age, String employeeId) {
        super(name, age);
        this.employeeId = employeeId;
    }

    public String getEmployeeId() {
        return employeeId;
    }
}

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

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

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

    public void enrollStudent(Student student) {
        if (hasCompletedPrerequisites(student)) {
            enrolledStudents.add(student);
            System.out.println("Enrolled student: " + student.getName() + " in course: " + courseName);
        } else {
            System.out.println("Student " + student.getName() + " does not meet prerequisites for
course: " + courseName);
        }
    }

    private boolean hasCompletedPrerequisites(Student student) {
        for (Course prerequisite : prerequisites) {
            if (!student.hasCompletedCourse(prerequisite)) {
                return false;
            }
        }
        return true;
    }

    public void displayEnrolledStudents() {
```

```
        System.out.println("Enrolled students in course: " + courseName);
        for (Student student : enrolledStudents) {
            System.out.println("Student ID: " + student.getStudentId() + ", Name: " + student.getName());
        }
    }
}

public class UniversityEnrollmentSystem {
    public static void main(String[] args) {
        Student student1 = new Student("Alice", 20, "S101");
        Student student2 = new Student("Bob", 22, "S102");

        Course math101 = new Course("Math 101");
        Course physics101 = new Course("Physics 101");
        Course computerScience101 = new Course("Computer Science 101");

        // Set prerequisites for Computer Science 101
        computerScience101.addPrerequisite(math101);
        computerScience101.addPrerequisite(physics101);

        // Enroll students in courses
        computerScience101.enrollStudent(student1);  // Student1 has prerequisites, so enrollment is successful
        computerScience101.enrollStudent(student2);  // Student2 does not meet prerequisites, so enrollment fails

        // Display enrolled students in Computer Science 101
        computerScience101.displayEnrolledStudents();
    }
}
```