Core Java Assignment 4

1. You are tasked with designing a system for a hospital that keeps track of patient information and their medical history. The system should allow doctors and nurses to add new patients, record their medical history, and retrieve patient information as needed.
   To implement this system, you need to create the following classes:
   Patient: This class should have the following fields:
   I. name (String): The name of the patient
   II. age (int): The age of the patient
   III. gender (String): The gender of the patient
   IV. address (String): The address of the patient
   MedicalRecord: This class should have the following fields:
   I. date (Date): The date when the medical record was created
   II. doctorName (String): The name of the doctor who created the medical record
   III. diagnosis (String): The diagnosis given to the patient
   IV. treatment (String): The treatment prescribed to the patient
   PatientRecord: This class should have the following fields:
   I. patient (Patient): An instance of the Patient class
   II. medicalRecords (Array of MedicalRecord): A list of MedicalRecord objects associated with the patient
   Hospital: This class should have the following fields:
   I. patientRecords (Array of PatientRecord): A list of PatientRecord objects
   HospitalManager: This class should have the following methods:
   I. addPatientRecord: Adds a new PatientRecord object to the patientRecords list
   II. addMedicalRecord: Adds a new MedicalRecord object to the medicalRecords list of a given PatientRecord
   III. getPatientRecord: Returns a PatientRecord object for a given patient name
   IV. getPatientMedicalRecords: Returns a list of MedicalRecord objects for a given patient name

2. Suppose you are working on a project to develop a library management system. In this system, there are multiple libraries that contain a large number of books. Each library is managed by a librarian who has to keep track of the books in the library.

   Design a Java program that models this library management system using aggregation. The program should have the following classes:

Book: This class represents a book and contains information such as the title, author, ISBN, and publisher. The class should also have a method to get the number of copies of the book in the library.

Librarian: This class represents a librarian and contains information such as the name and contact details. The class should have methods to add and remove books from the library, search for books by title or author, and get the total number of books in the library.

Library: This class represents a library and contains a list of Book objects. The class should have methods to add and remove books from the library, search for books by title or author, and get the total number of books in the library. The class should also have a Librarian object to represent the librarian who manages the library.

Your program should also implement the following business logic:

I. When a book is added to a library, the number of copies of the book should be increased by 1. If the book already exists in the library, the number of copies should be increased by 1.

II. When a book is removed from a library, the number of copies of the book should be decreased by 1. If the number of copies of the book becomes 0, the book should be removed from the library.

III. The Librarian object should be responsible for managing the library. The librarian should be able to add and remove books from the library, search for books by title or author, and get the total number of books in the library.

IV. The Library object should be responsible for maintaining the list of books in the library and the Librarian object.

You can implement the program using aggregation by having the Library class contain a list of Book objects and a Librarian object.

3. Suppose you are developing a payroll management system for a company that has different types of employees, including full-time employees and part-time employees. You have a superclass called **Employee** that contains common attributes and methods for all types of employees. You also have two subclasses called

FullTimeEmployee and **PartTimeEmployee**, respectively, that inherit from the Employee class.

The **Employee** class has the following attributes:

I. name (String)
II. id (int)
III. salary (double)

The Employee class also has the following methods:

I. calculatePay() - a method that calculates the pay of an employee based on their salary.

The **FullTimeEmployee** class has the following additional attributes:

I. bonus (double)

The FullTimeEmployee class also has the following methods:

I. calculatePay() - a method that calculates the pay of a full-time employee based on their salary and bonus.

The **PartTimeEmployee** class has the following additional attributes:

I. hoursWorked (int)
II. hourlyRate (double)

The PartTimeEmployee class also has the following methods:

I. calculatePay() - a method that calculates the pay of a part-time employee based on their hours worked and hourly rate.

Write a Java program that demonstrates the following:

I. Create an array of Employee objects that contains both FullTimeEmployee and PartTimeEmployee objects.
II. Use dynamic method dispatch to call the calculatePay method on all the Employee objects in the array.
III. Use the instanceof operator to identify the FullTimeEmployee and PartTimeEmployee objects in the array and adjust their pay accordingly. A FullTimeEmployee receives a bonus of 10% of their salary and a PartTimeEmployee receives a 20% bonus if they work more than 40 hours in a week.
IV. Calculate and print the total payroll of the company.

Note: You may assume that a full-time employee works 40 hours per week and a part-time employee does not receive any bonus if they work 40 hours or less in a week.

4. Suppose you are developing a library management system that has different types of books, including reference books, fiction books, and non-fiction books. You have a superclass called **Book** that contains common attributes and methods for all types of books. You also have three subclasses called **ReferenceBook**, **FictionBook**, and **NonFictionBook**, respectively, that inherit from the Book class.

   The **Book** class has the following attributes:
   I. title (String)
   II. author (String)
   III. publisher (String)
   IV. price (double)
   V. numPages (int)

   The Book class also has the following methods:
   I. calculatePrice() - a method that calculates the price of a book based on its price and number of pages.

   The **ReferenceBook** class has the following additional attribute:
   I. edition (int)

   The ReferenceBook class also has the following methods:
   I. calculatePrice() - a method that calculates the price of a reference book based on its price, number of pages, and edition.

   The **FictionBook** class has the following additional attribute:
   I. genre (String)

   The FictionBook class also has the following methods:
   I. calculatePrice() - a method that calculates the price of a fiction book based on its price, number of pages, and genre.

   The **NonFictionBook** class has the following additional attribute:
   I. subject (String)

   The NonFictionBook class also has the following methods:
   I. calculatePrice() - a method that calculates the price of a non-fiction book based on its price, number of pages, and subject.

   Write a Java program that demonstrates the following:
   I. Create an array of Book objects that contains instances of ReferenceBook, FictionBook, and NonFictionBook.
   II. Use dynamic method dispatch to call the calculatePrice and printBookInfo methods on all the Book objects in the array.
   III. Use the instanceof operator to identify the ReferenceBook, FictionBook, and NonFictionBook objects in the array and adjust their price accordingly. A ReferenceBook has a 20% discount if it is an older edition (edition < 5), a FictionBook has a 10% discount if it is a romance novel, and a NonFictionBook has a 15% discount if it is about history.
   IV. Calculate and print the total price of all the books in the array.

   Note: You may assume that all books have the same price per page.

5. Suppose you are developing a sports management system that manages different types of players, including cricket players and football players. You have a superclass called **Player** that contains common attributes and methods for all types

of players. You also have two subclasses called **CricketPlayer** and **FootballPlayer**, respectively, that inherit from the Player class.

The **Player** class has the following attributes:

I. name (String)
II. age (int)
III. team (String)
IV. score (int)

The Player class also has the following methods:

I. play() - a method that simulates playing a game and updates the score of the player.

The **CricketPlayer** class has the following additional attributes:

I. runsScored (int)
II. wicketsTaken (int)

The CricketPlayer class also has the following methods:

I. play() - a method that simulates playing a cricket match and updates the runs scored and wickets taken by the player.

The **FootballPlayer** class has the following additional attributes:

I. goalsScored (int)
II. assists (int)

The FootballPlayer class also has the following methods:

I. play() - a method that simulates playing a football match and updates the goals scored and assists made by the player.

Write a Java program that demonstrates the following:

I. Create an array of Player objects that contains both CricketPlayer and FootballPlayer objects.
II. Use dynamic method dispatch to call the play method on all the Player objects in the array.
III. Use the instanceof operator to identify the CricketPlayer and FootballPlayer objects in the array and adjust their scores accordingly. A CricketPlayer scores an additional 10 runs for every wicket they take, and a FootballPlayer scores an additional 5 goals for every assist they make.
IV. Calculate and print the total score of the team.

Note: You may assume that a cricket match has 50 overs and a football match has 90 minutes.

6. Suppose you are developing a bank account management system that has different types of accounts, including savings accounts and checking accounts. You have a superclass called **Account** that contains common attributes and methods for all types of accounts. You also have two subclasses called **SavingsAccount** and **CheckingAccount**, respectively, that inherit from the Account class.

   The **Account** class has the following attributes:
   I. accountNumber (String)
   II. balance (double)

   The Account class also has the following methods:
   I. deposit(double amount) - a method that adds the specified amount to the account balance.
   II. withdraw(double amount) - a method that subtracts the specified amount from the account balance.

   The **SavingsAccount** class has the following additional attributes:
   I. interestRate (double)

   The SavingsAccount class also has the following methods:
   I. calculateInterest() - a method that calculates the interest earned by the account based on the interest rate and current balance.

   The **CheckingAccount** class has the following additional attributes:
   I. overdraftLimit (double)

   The CheckingAccount class also has the following methods:
   I. withdraw(double amount) - a method that subtracts the specified amount from the account balance, but allows the balance to go negative up to the overdraft limit.

   Write a Java program that demonstrates the following:
   I. Create an array of Account objects that contains both SavingsAccount and CheckingAccount objects.
   II. Use dynamic method dispatch to call the deposit, withdraw, and getBalance methods on all the Account objects in the array.
   III. Use the instanceof operator to identify the SavingsAccount and CheckingAccount objects in the array and adjust their balance accordingly. A SavingsAccount earns interest at a rate of 2% per year and a CheckingAccount is charged a penalty fee of $20 if their balance goes negative.
   IV. Calculate and print the total balance of all the accounts in the array.

Note: You may assume that the interest and penalty fees are only applied once per year and all accounts have been open for exactly one year.

7. Suppose you are developing a student management system for a school that has different types of students, including undergraduate students and graduate students. You have a superclass called **Student** that contains common attributes and methods for all types of students. You also have two subclasses called **UndergraduateStudent** and **GraduateStudent**, respectively, that inherit from the Student class.

   The Student class has the following attributes:
   I. name (String)
   II. id (int)
   III. major (String)
   IV. gpa (double)

   The **Student** class also has the following methods:
   I. calculateTuition() - a method that calculates the tuition of a student based on their major and GPA.

   The **UndergraduateStudent** class has the following additional attributes:
   I. creditHours (int)
   II. tuitionRate (double)

   The UndergraduateStudent class also has the following methods:
   I. calculateTuition() - a method that calculates the tuition of an undergraduate student based on their credit hours and tuition rate.

   The **GraduateStudent** class has the following additional attributes:
   I. researchFee (double)

   The GraduateStudent class also has the following methods:
   I. calculateTuition() - a method that calculates the tuition of a graduate student based on their research fee.

   Write a Java program that demonstrates the following:
   I. Create an array of Student objects that contains both UndergraduateStudent and GraduateStudent objects.
   II. Use dynamic method dispatch to call the calculateTuition and printTuition methods on all the Student objects in the array.
   III. Use the instanceof operator to identify the UndergraduateStudent and GraduateStudent objects in the array and adjust their tuition accordingly. An UndergraduateStudent receives a 20% discount on their tuition if their GPA is

greater than or equal to 3.5, and a GraduateStudent receives a 10% discount on their tuition if their research fee is greater than $5000.

IV. Calculate and print the total tuition of the school.

Note: You may assume that the tuition rate for an undergraduate student is $500 per credit hour and the tuition rate for a graduate student is $1000 per credit hour.

8. Suppose you are developing a system for a hospital that has different types of patients, including in-patients and out-patients. You have a superclass called **Patient** that contains common attributes and methods for all types of patients. You also have two subclasses called **InPatient** and **OutPatient**, respectively, that inherit from the Patient class.

The **Patient** class has the following attributes:

I. name (String)
II. id (int)
III. age (int)

The Patient class also has the following methods:

I. calculateBill() - a method that calculates the bill of a patient based on their treatment charges.

The **InPatient** class has the following additional attributes:

I. numberOfDays (int)
II. roomCharges (double)
III. medicineCharges (double)

The InPatient class also has the following methods:

I. calculateBill() - a method that calculates the bill of an in-patient based on their treatment charges, room charges, and medicine charges.

The **OutPatient** class has the following additional attributes:

I. doctorFees (double)
II. testCharges (double)

The OutPatient class also has the following methods:

I. calculateBill() - a method that calculates the bill of an out-patient based on their treatment charges, doctor fees, and test charges.

Write a Java program that demonstrates the following:

I. Create an array of Patient objects that contains both InPatient and OutPatient objects.
II. Use dynamic method dispatch to call the calculateBill and printBill methods on all the Patient objects in the array.
III. Use the instanceof operator to identify the InPatient and OutPatient objects in the array and adjust their bill accordingly. An in-patient has an additional charge of $100 per day for their room, and a 10% discount on their medicine charges if they are over 60 years old. An out-patient has a discount of 5% on their test charges if they are over 60 years old.
IV. Calculate and print the total bill of the hospital.

Note: You may assume that the treatment charges for all patients are $500.

9. Suppose you are building a system that manages a fleet of vehicles for a rental company. You have a superclass called Vehicle that contains common attributes and methods for all types of vehicles. You also have three subclasses called **Car**, **Motorcycle**, and **Truck**, respectively, that inherit from the Vehicle class.

   The **Vehicle** class has the following attributes:

   I. make (String)
   II. model (String)
   III. year (int)
   IV. dailyRentalRate (double)

   The Vehicle class also has the following methods:

   I. calculateRentalFee(int days) - a method that calculates the rental fee for a vehicle based on the number of days rented and the daily rental rate.

   The **Car** class has the following additional attributes:

   I. numDoors (int)

   The Car class also has the following methods:

   I. calculateRentalFee(int days) - a method that calculates the rental fee for a car based on the number of days rented and the daily rental rate. If the car has more than two doors, there is a surcharge of $10 per day.

   The **Motorcycle** class has the following additional attributes:

   I. engineSize (int)

   The Motorcycle class also has the following methods:

   I. calculateRentalFee(int days) - a method that calculates the rental fee for a motorcycle based on the number of days rented and the daily rental rate. If the engine size is greater than 1000cc, there is a surcharge of $20 per day.

   The **Truck** class has the following additional attributes:

   I. cargoCapacity (double)

   The Truck class also has the following methods:

   I. calculateRentalFee(int days) - a method that calculates the rental fee for a truck based on the number of days rented and the daily rental rate. If the cargo capacity is greater than 5000 cubic feet, there is a surcharge of $30 per day.

   Write a Java program that demonstrates the following:

I. Create an array of Vehicle objects that contains both Car, Motorcycle, and Truck objects.
II. Use dynamic method dispatch to call the calculateRentalFee and printDetails methods on all the Vehicle objects in the array.
III. Use the instanceof operator to identify the Car, Motorcycle, and Truck objects in the array and adjust their rental fee accordingly. A car with more than two doors receives a surcharge of $10 per day, a motorcycle with an engine size greater than 1000cc receives a surcharge of $20 per day, and a truck with a cargo capacity greater than 5000 cubic feet receives a surcharge of $30 per day.
IV. Calculate and print the total rental fee for all the vehicles in the array.

10. Suppose you are designing a simulation of a battle between two armies, the Red Army and the Blue Army. Each army is made up of different types of units, including infantry, cavalry, and artillery. You have a superclass called **Unit** that contains common attributes and methods for all types of units. You also have three subclasses called **Infantry**, **Cavalry**, and **Artillery**, respectively, that inherit from the Unit class.

The **Unit** class has the following attributes:
I. name (String)
II. health (int)
III. attack (int)
IV. defense (int)

The Unit class also has the following methods:
I. move() - a method that prints a message to the console indicating that the unit is moving.
II. attack(Unit enemy) - a method that calculates the damage inflicted on the enemy unit based on the unit's attack and the enemy unit's defense, and reduces the enemy unit's health accordingly.

The **Infantry** class has the following additional attributes:
I. morale (int)

The Infantry class also has the following methods:
I. charge() - a method that increases the infantry's attack by 50% and decreases its defense by 50%.

The Cavalry class has the following additional attributes:

I. speed (int)

The Cavalry class also has the following methods:

I. charge() - a method that increases the cavalry's attack by 100% and decreases its defense by 50%.

The Artillery class has the following additional attributes:

I. range (int)

The Artillery class also has the following methods:

I. bombard() - a method that inflicts a fixed amount of damage on all enemy units within the artillery's range.

Write a Java program that simulates a battle between the Red Army and the Blue Army, and demonstrates the following:

I. Create an array of Unit objects that contains both Infantry, Cavalry, and Artillery objects for both armies.
II. Use dynamic method dispatch to call the move and attack methods on all the Unit objects in the array.
III. Use the instanceof operator to identify the Infantry, Cavalry, and Artillery objects in the array and call their respective methods (charge, bombard).
IV. Implement a logical calculation to determine which army wins the battle based on the remaining health points of each army's units. The army with the most remaining health points is declared the winner.
V. Print out the name, health, attack, and defense attributes of all the Unit objects in the array.

Note: You may also add any additional attributes or methods as necessary.