Assignment 7

- Abstract classes
- Abstract methods
- Overrding methods to use Java Collections framework effectively.

1. Create an abstract class called Shape that serves as a blueprint for a drawing application, containing a String color attribute, a constructor to initialize it, and a concrete method showColor() that prints the shape's color. You must define an abstract method calculateArea() within this class, as the mathematical formula for area depends entirely on the specific type of shape being drawn. Implement two subclasses, Circle (using a radius) and Rectangle (using length and width), which provide their own unique versions of the area calculation. Finally, in a Main class, demonstrate polymorphism by creating a Shape array containing both a circle and a rectangle, then use a loop to display the color and calculated area for each.

2. Imagine you are developing a payment processing system for an e-commerce platform. The system must support different ways to pay, such as credit card, google pay, BHIM UPI. Assume that the payment gateway gets different transactions (can be stored in an array). These transactions are one of the above ways which means instances of credit card, or google pay or BHIM UPI.. Payment gateway calls process payment method associated with each transaction. Take appropriate design decisions.

3. Consider a scenario where you have a base class called Employee, and two derived classes called Manager and Developer. The Manager class should have an additional attribute called departmentName, and the Developer class should have an additional attribute called programmingLanguage.
    - Define the Employee class with attributes name and salary, add a method abstract method displayDetails() that prints the details of the employee, add set and get methods for the attributes, add a method computeIncomeTax() which computes 10% of the salary as income tax.
    - Define the Manager class that extends the Employee class, and include the departmentName attribute. Override the displayDetails() method to include the department name.
    - Define the Developer class that extends the Employee class, and include the programmingLanguage attribute. Override the displayDetails() method to include the programming language.
    - Create instances of Manager and Developer, and demonstrate the use of the displayDetails() method.

4. You are working on an e-commerce application where you need to calculate the total price of an order. The system should handle different types of orders:

1. Single Item: The total price is calculated based on the price of a single item and its quantity.

2. Multiple Items: The total price is calculated based on multiple item prices and quantities.

3. Discounted Order: The total price includes a discount, so you need to apply a percentage discount on the total price.

Implement a method calculateTotalPrice() using method overloading to handle the following scenarios:

● Calculate the total price for a single item (quantity and price).

● Calculate the total price for multiple items (arrays of prices and quantities).

● Calculate the total price for an order that includes a discount (total price and

discount percentage).

5. You are working on a travel booking application where users can book different types of transportation for their journey. The application should handle:

   1. Booking a flight: The user provides the flight number, departure city, and arrival city.

   2. Booking a train: The user provides the train number and the class of service (e.g., economy, business).

   3. Booking a cab: The user provides the pickup location and destination, with an option to specify the number of passengers.

   Implement a method bookTransport() using method overloading to handle the following scenarios:

   ● Book a flight by providing the flight number, departure city, and arrival city.

   ● Book a train by providing the train number and class of service.

   ● Book a cab by providing the pickup location and destination, with an optional parameter for the number of passengers.

6. Create a patient management system for a hospital emergency room using a PriorityQueue. Design a Patient class with attributes: name (String), age (int), ailment (String), and priority (int) where 1=Critical, 2=Serious, 3=Moderate, 4=Minor, and 5=Non-urgent. Implement a HospitalEmergencyRoom class that uses a PriorityQueue<Patient> with a custom Comparator to ensure patients with lower priority numbers are treated first. If two patients have the same priority level, they should be

treated in the order they arrived (FIFO). The class should include methods to add patients, treat the next patient (remove and return), display all waiting patients, and get the count of waiting patients. In the Main class, demonstrate the system by creating 6-8 patients with different priority levels, adding them in random order, displaying the waiting list, treating several patients, and showing the updated queue after each treatment to prove that patients are being treated by priority rather than arrival order.

Hint: Use Comparator and override the compare() method

7. At NITC, there are three popular student clubs: Debate Club, Coding Club, and Cycling Club. Students can join multiple clubs based on their interests. The Student Affairs Office needs a system to manage student memberships across these clubs efficiently. Your task is to design a club management system that ensures each student has only one Student object instance in memory, regardless of how many clubs they join. This prevents data redundancy and ensures consistency across all club memberships.
   Use a HashSet to manage the students and do the needful overriding in the Student class.