2025

Object Oriented Programming and Design

PROBLEM SHEET 01

1. **Student Attendance Management System**

A teacher needs a Java program to keep track of student attendance in her class.

The class currently has the following students:

Student ID	Name	Days Attended
101	Alice Smith	12
102	Bob Jones	15
103	Carol Lee	10

1. Design a Student class with the following private fields:
    - studentId (int)
    - name (String)
    - daysAttended (int)
2. Provide a constructor to initialize all fields.
3. Implement getter and setter methods for all fields using appropriate access modifiers.
4. Design a Classroom class that:
    - Holds an array of Student objects (maximum capacity 10).
    - Keeps track of the number of students added.
5. Implement methods in the Classroom class to:
    - Add a new student to the array.
    - Update the number of days attended for a student, identified by their student ID. If the student ID does not exist, display an appropriate message.
    - Display the details of all students in a readable format.
6. In the main method:
    - Create an instance of Classroom.
    - Add the three students from the table above.
    - Update Bob Jones's attendance to 16 days.
    - Attempt to update attendance for a student with ID 104 (who does not exist).
    - Display all student details.

2. Parking Lot Vehicle Tracker

A small parking lot wants a program to track vehicles parked inside.

- Each vehicle has:
    - licensePlate (String)
    - ownerName (String)
    - hoursParked (int)
- The parking lot can hold up to 5 vehicles.

1. Define a Vehicle class with the above private fields, a constructor, and getter/setter methods.
2. Define a ParkingLot class that stores vehicles in an array (max 5) and tracks how many vehicles are currently parked.
3. Implement methods in ParkingLot to:
    - Park a new vehicle (add it to the array).
    - Remove a vehicle by license plate (freeing the slot, shifting remaining vehicles to avoid gaps).
    - Display details of all parked vehicles.
4. In your main method:
    - Create a ParkingLot instance.
    - Park these vehicles:
        - License: "ABC123", Owner: "John Doe", Hours: 2
        - License: "XYZ789", Owner: "Jane Smith", Hours: 4
        - License: "LMN456", Owner: "Bob Brown", Hours: 1
    - Remove the vehicle with license "XYZ789".
    - Display all parked vehicles.

3. **Simple Bank Account Manager with Error Handling**

A bank wants a simple Java program to manage customer bank accounts.

Each bank account has:

- accountNumber (int)
- accountHolder (String)
- balance (double)

1. Define a BankAccount class with the above private fields, constructor, and getter/setter methods.
2. Implement a method withdraw(double amount) in BankAccount that subtracts the amount from the balance only if there is enough balance; otherwise, it throws an IllegalArgumentException with the message "Insufficient balance".
3. Create a Bank class that stores up to 5 BankAccount objects in an array and keeps track of the number of accounts.
4. Implement methods in Bank to:
    - Add a new bank account.
    - Withdraw money from an account given its account number and amount. Use try-catch in this method to catch and handle IllegalArgumentException, printing the exception message.
    - Display all accounts' details.
5. In the main method:
    - Create a Bank instance.
    - Add these accounts:
        - Account 1001, Holder: "Alice", Balance: 5000.0
        - Account 1002, Holder: "Bob", Balance: 3000.0
    - Withdraw 6000.0 from account 1001 (should cause an exception).
    - Withdraw 1000.0 from account 1002 (successful).
    - Display all accounts.

4. **Movie Ticket Booking System**

A small cinema wants a program to handle ticket bookings for a single movie.

Scenario:

- The cinema shows one movie.
- The movie theater has a maximum of 10 seats.
- Each ticket booking includes:
    - ticketNumber (int)
    - customerName (String)
    - seatNumber (int)

Requirements:

- Design appropriate classes to represent tickets and the booking system.
- Implement methods to:
    - Book a ticket for a customer, assigning a seat number. If all seats are booked, the booking should fail gracefully.
    - Cancel a ticket by ticket number, freeing the seat.
    - Display all booked tickets with their details.

Constraints:

- Use arrays only to store tickets (no collections).
- Implement input validation where necessary (e.g., no double booking of seats).
- Use proper access modifiers and naming conventions.

In your main method:

- Book tickets for three customers with ticket numbers 1, 2, and 3, assigned to seats 1, 2, and 3 respectively.
- Cancel the ticket with ticket number 2.
- Attempt to book a new ticket for a customer at seat 2 (which is now free).
- Display all current bookings.

5. **Student Grade Calculator with Exception Handling**

Write a Java program to manage a student's exam scores and calculate the average, including input validation with exception handling.

Instructions:

1. Create a class named Student with these private fields:
    - name (String)
    - exam1 (int)
    - exam2 (int)
    - exam3 (int)
2. Create a constructor that accepts the student's name and three exam scores, initializing the fields.
3. Implement input validation in the constructor or setters to ensure that exam scores are between 0 and 100. If an invalid score is passed, throw an IllegalArgumentException with the message "Exam scores must be between 0 and 100."
4. Create getter methods for each field.
5. Create a method named calculateAverage() that returns the average of the three exam scores as a double.
6. In the main method:
    - Use try-catch to create a Student object for a student named "John" with exam scores 75, 110, and 90. (Notice 110 is invalid.)
    - If an exception is caught, print the exception message.
    - If creation succeeds, print the student's name and average exam score (formatted to two decimal places).