OOP Model Paper (Student Made)

1.

a. Create a Java class named **Student.** The following variables should only be accessible by the class itself. Use suitable data types for these attributes.
   - studentID
   - studentName
   - enrollmentYear (Should be same for all the students)
   - gpa
   - isProbation
   - department

b. Implement default constructor and two parameterized constructors for Student class.
   >> Student()
   >>Student(studentID, studentName, department)
   >>Student(studentID, studentName, gpa, department)

Default Constructor
studentID = NaN
studentName = Unnamed
enrollmentYear = 2025
gpa = 0.0
department = computing

c. Implement getters for all the values.
d. Implement setters for all the values except for **isProbation.** When setting a new GPA (Except through constructors) require a password (*admin123*). If the password is incorrect throw an assertion error and terminate the program.
e. Also, if the GPA is lower than 2.0, automatically set the isProbation value to false. But, if GPA changed above 2.0, it changed back to **true**. (Both through the constructor and the set GPA method.)
f. Store three students objects made from the Student class, using the three distinct constructors.
g. Create a method called ***displayDetails()*** that display single student objects details.

h. Create a method called ***displayDetails(Student[])*** that displays all the student details in the input array.

i. Now, create a overload method for ***displayDetails(Student[], String)*** that takes two arguments;
   - Student Array
   - Department

   Then only display the student details in the given department.

displayDetails(Student[]) – Sample Output

Student ID: 10020
Student Name: Dinindu
Enrollment Year: 2025
Current GPA: 2.9
Under Probation? : No
Department: Computing

Student ID: 10030
Student Name: Kamal
Enrollment Year: 2025
Current GPA: 1.2
Under Probation? : Yes
Department: Business

j. Create a main method that takes input values for student details. (Tip: Use Scanner)

   Then create three objects using three constructors. Include them in an array.
   Display the details of students.
   >> First: Using ***displayDetails(Student[])*** method
   >>Secondly: Using ***displayDetails(Student[], Department)*** method

   Then change the GPA of a student (use correct password), which should change their isProbation status.

   Again display the details.

   Try to change the GPA of a student again, but this time enter a wrong password (To demonstrate the assertion error.)

2. Consider the following scenario.

An online retail platform facilitates the buying and selling of various products. The system provides customers with a searchable catalog of items, ranging from electronics to clothing. Each product has a detailed description, specifications, price, seller information, and current stock status. Sellers can register on the platform to list their products for sale. The platform manages the entire product inventory, allowing sellers to add new items, update product details, and remove listings that are out of stock.

Customers can create a personal account by providing their name, shipping address, and payment information. Once registered, a customer can browse the product catalog, search for specific items using filters like brand or price range, and view product reviews from other buyers. Customers can add desired products to a virtual shopping cart before proceeding to checkout. The system calculates the total cost, including taxes and shipping fees, and processes the payment through various integrated payment gateways.

After a customer places an order, the system sends an order confirmation to the customer and notifies the seller. The seller then prepares the product for shipment. The warehouse staff packages the item and updates the order status within the system, which generates a tracking number for the customer. The customer can monitor the delivery progress through their account.

The platform also handles post-sale activities. Customers can initiate a return request for a damaged or incorrect item. The customer service department reviews the request and, upon approval, provides instructions for the return. Once the warehouse receives the returned item, the system processes a refund to the customer's original payment method. The system maintains a complete order history for each customer and generates sales reports for sellers to track their performance.

a. Run a noun analysis on this and write down the potential class.
b. Eliminate the potential classes using five rules. Clearly state the rule Infront of the potential class.
c. List the final classes.
d. Perform a verb analysis and identify the responsibilities for three classes.
e. Fill out three CRC cards for the identified classes *with responsibilities*.

<--------------->	

<--------------->	

<--------------->	

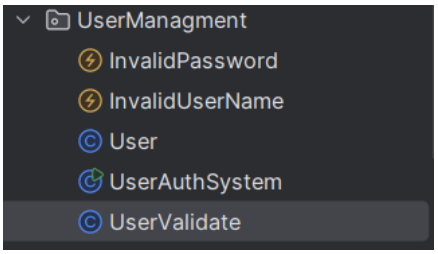
3. You are assigned to create a prototype for a user login system.
    a. Create two exceptions
       >> InvalidUserName
       >>InvalidPassword
    b. Create another class called userValidate. In this class implement two methods that ***could potentially*** throw exception.
        i. validateUserName: Throws an InvalidUserName exception if the username length is less than 6.
        ii. validateUserPassword: Throws an InvalidPassword exception if any of the following conditions becomes true.
            1. Password length is less than 8
            2. Password is “12345678”
        iii. Show appropriate messages for the above exceptions.
    c. Create another class called **User**, which has attributes:
        i. String username
        ii. String password
    d. Create a Scanner object in the **User** class. You should use this Scanner object to get the values for setters of userName and userPassword.
    e. Create setters for these two attributes. Handle the InvalidUserName exception and handle the InvalidPassword exception with the help of ***validateUserName*** and ***validatePassword*** methods.
       If the user enters a wrong password or username, instead of terminating the program, it should call the setter method recursively until user enters an appropriate password.

The main method

```
package UserManagment;

import java.util.Scanner;

public class UserAuthSystem {
    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        User user1 = new User();

        user1.setUserName();
        user1.setPwd();

    }
}
```

supportByDV

File Structure



Expected Output

```
Enter a username: Din
Error: Username should be longer than 6 characters!
Enter a username: Gh
Error: Username should be longer than 6 characters!
Enter a username: Dinnidu
Enter a password: 12
Error: Password must be longer than 8 characters!
Enter a password: 12345678
Error: Password pattern is common! Create a strong password!
Enter a password: dinindu@1234

Process finished with exit code 0
```

4. You are tasked with developing a vehicle rental management system. The system needs to handle different types of vehicles, manage rentals, and calculate costs. This requires modeling the relationships between vehicles, customers, and the rental agency itself.
    a. Create an abstract class names **vehicle**. It should contain the following;
        - vehicleID
        - brand
        - isRented (false)

        i. A parameterized constructor to initialize the attributes.
        ii. Getters for all attributes.
        iii. A setter for the **isRented** status.
        iv. An abstract method **displayDetails()** that will be implemented by subclass.
        v. An **abstract method calculateRentalCost(int days)** that takes the number of rental days and returns the total cost (double).
    b. Create two concrete classes, **Car** and **Motorcycle** that inherits from the vehicle class.
        i. Car Class:
            - Add a private attribute **numOfSeats**
            - Implement a constructor that initializes all attributes, including those from the Vehicle superclass.
            - Override the **displayDetails()** method to print all details of the car, including the number of seats.
            - Override the **calculateRentalCost(int days)** method. The rental cost for a car is **$50** per day.

        ii. Motorcycle Class
            - Add a private attribute: hasSidecar (boolean).
            - Implement a constructor that initializes all attributes.
            - Override the **displayDetails()** method to print all details of the motorcycle.
            - Override the **calculateRentalCost(int days)** method. The rental cost is **$30 per day**. If it has a sidecar, add a **one-time fee of $20** to the total cost.
    c. Create a class named **RentalAgency**. This class will manage a collection of vehicles.
        - Implement a **composition** relationship where the RentalAgency **has a** list of Vehicle objects. Use an ArrayList<Vehicle> to store the vehicles.
        - Create a method addVehicle(Vehicle vehicle) to add new vehicles to the agency's fleet.
        - Create a method rentVehicle(String vehicleId, int days) that:
            - Finds the vehicle by its ID in the ArrayList.
            - If the vehicle exists and is not already rented (isRented is false):
                - Calculate the rental cost using the vehicle's calculateRentalCost method.
                - Set the vehicle's status to rented.

        - Print a confirmation message with the total cost (e.g., "Vehicle [ID] rented for [days] days. Total cost: $[cost]").
    - If the vehicle is already rented or not found, print an appropriate message (e.g., "Vehicle is currently unavailable or does not exist.").
d. Create a method **displayAvailableVehicles()** that iterates through the list and prints the details of only the vehicles that are **not** currently rented.
e. Create a main application class RentalSystemApp to demonstrate the system's functionality.
    i. Create an instance of RentalAgency.
    ii. Create at least two Car objects (one with 4 seats, one with 7 seats) and one Motorcycle object (with a sidecar) in the RentalAgency instance.
    iii. Display all available vehicles before any rentals.
    iv. Simulate renting one car for 5 days and the motorcycle for 3 days.
    v. Display the available vehicles again to show that their rental status has been updated.
    vi. Attempt to rent a vehicle that is already rented to show the "unavailable" message.

Sample Output

Vehicle ID: 100
Vehicle Brand: Suzuki
Number of seats: 4
Rental Status: Available

Vehicle ID: 101
Vehicle Brand: Maruti
Number of seats: 7
Rental Status: Available

Vehicle ID: 102
Vehicle Brand: Bajaj
Includes side car? :Yes
Rental Status: Available

Vehicle available!
Vehicle 100 is rented for 5 days. Total cost is 250.000000$.

Vehicle available!
Vehicle 102 is rented for 3 days. Total cost is 110.000000$.

Vehicle ID: 100
Vehicle Brand: Suzuki
Number of seats: 4

Rental Status: Rented

Vehicle ID: 101
Vehicle Brand: Maruti
Number of seats: 7
Rental Status: Available

Vehicle ID: 102
Vehicle Brand: Bajaj
Includes side car? :Yes
Rental Status: Rented

Sorry. The vehicle is currently unavailable!

Sorry. This vehicle does not exists.

Process finished with exit code 0

5.

a. Consider the following code snippet.

```
interface Mammal {
    void walk();
    void breath();
}

class Whale implements Mammal{
    public void walk(){
        throw new UnsupportedOperationException();
    }

    public void breath(){

    }

    public void drinkMilk(){

    }
}
```

i. Which SOLID principle the above code violates? _____

ii. Which methods should be removed from the interface ***Mammal*** to achieve the desired functionality?

_____
_____

iii. What are the methods that can be added to ***Mammal*** interface, considering the given two classes.

_____
_____
_____

iv. In a software evaluation, a senior engineer identified that one class handles a lot of responsibilities at the same time. What is the most possible SOLID principle that violated in this scenario?

_____

b. In a certain E-Learning platform there are two types of accounts. **Student** and **Teacher**. A student has student ID, student name, assigned courses (array), and membership status (standard, gold, platinum (Use char data type)). They all share the same variable, which is available course list. This is a common and static value for all the users.

Standard members can study courses but can't get a certificate. Gold members can get a certificate. Platinum members can get both certificates and real-time help from the teachers.

The teachers have both teacher privileges and student privileges. Teachers do not explicitly have a teacher ID or teacher name. It's the same as their studentId and student name. They do have a array called assigned courses. If they are assigned to a course they can blacklist a student from enrolling into that particular course.

The course has a coursID, courseName, assignedTeacher and the blacklistStudentID array.

Use array list whenever it's needed to initialize a list.

i. Implement the student interface with common shareable values. Also initialize the available course list with a static initializer with some values.
ii. Implement the teacher interface with appropriate values.
iii. Implement the Student class.
iv. Implement the teacher class.
v. Implement the course array.
vi. Implement the enrollToCourse with validation in the student interface, that the particular student is not blacklisted.
vii. Implement the assignCourse. Validate thet no other teacher has been assigned to the course yet.

Note: Did not has time to complete the whole question (05 – Part B). Used ChatGPT to generate it. Above one is the uncompleted version (made by me). Consider the below one as the real question.

Question: Object-Oriented Design – E-Learning Platform (Interface-Based Design)

In a certain E-Learning platform, there are two types of accounts: Student and Teacher.

Entities and Relationships

Student

A student has:

- **studentId (String)**
- **studentName (String)**
- **assignedCourses (ArrayList)**
- **membershipStatus (char):**
    - **'S' – Standard**
    - **'G' – Gold**
    - **'P' – Platinum**

All students share a static list of available courses:
availableCourseList (ArrayList) — this is common to all users and initialized using a static block.

Membership Privileges

- **'S' – Study only**
- **'G' – Study + Certificate**
- **'P' – Study + Certificate + Real-time help**

Teacher

- **Teachers are also students in the sense that they have a studentId, studentName, etc.**
- **They implement both StudentInterface and TeacherInterface to reflect multiple inheritance via interfaces.**
- **Additional responsibilities:**

- **assignedCourses (ArrayList) they teach**
- **Can blacklist a student from a course if they are the assigned teacher**

Course

A course has:

- **courseId (String)**
- **courseName (String)**
- **assignedTeacher (object of type implementing TeacherInterface)**
- **blacklistStudentIDs (ArrayList)**

Interfaces to Implement

StudentInterface

- **Static availableCourseList with 3+ sample courses (static initializer block)**
- **void enrollToCourse(Course c) – validates:**
  - **Course exists in available list**
  - **Student is not blacklisted**
  - **Adds to student's assignedCourses if valid**
- **void displayDetails() – shows student ID, name, membership status, assigned courses**

TeacherInterface

- **void assignCourse(Course c) – assigns teacher to course if unassigned**
- **void blacklistStudent(Course c, String studentId) – adds to course's blacklist if teacher is assigned**
- **void displayDetails() – overrides student version to include teaching info**

Implementation Tasks

i. Implement StudentInterface

- **Define static initializer for availableCourseList**
- **Implement methods as described**

ii. Implement TeacherInterface

- **Define method contracts for assigning and blacklisting**
- **Override displayDetails() with teacher-specific info**

iii. Create the Student class that:

- **Implements StudentInterface**
- **Maintains private attributes and necessary methods**
- **Properly validates and handles enrollment logic**

iv. Create the Teacher class that:

- **Implements both StudentInterface and TeacherInterface**
- **Reuses student attributes and behavior**
- **Adds blacklist and assign-course functionalities**
- **Overrides displayDetails() for polymorphism**

v. Create the Course class:

- **Fields: courseId, courseName, assignedTeacher, blacklistStudentIDs**
- **Include constructor, getters, setters**
- **Override toString() for easy printing**

Object-Oriented Concepts to Demonstrate

- **Encapsulation – via private fields and public getters/setters**
- **Interfaces and Multiple Inheritance – Teacher implements both interfaces**
- **Static Initialization – shared course list in StudentInterface**
- **Polymorphism – calling displayDetails() using interface references**
- **Method Overriding – teacher overrides displayDetails() from student**