Name: Ishaan Jhamb **Date: 4th feb 2026**
UID :23BCS11668
Subject: System Design

Assignment -1

Q1. Explain SRP and OCP in detail with proper examples.

1. Single Responsibility Principle (SRP)

Definition: "A module should be responsible to one, and only one, actor."

In simpler terms, a class should have only **one reason to change**. If a class handles logic for both the User Interface and the Database, it violates SRP because a change in the UI logic *or* a change in the Database schema would both require this single class to be modified.

Example: Imagine a Report class.

- **Violating SRP:** The class calculates statistics *and* generates HTML for the report.
- **Applying SRP:**
    - ReportCalculator: Responsible for the logic/math (Changed only if business logic changes).
    - ReportPrinter: Responsible for formatting/printing (Changed only if UI/Format changes).

2. Open/Closed Principle (OCP)

Definition: "Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification." You should be able to add new functionality without touching existing, tested code. This is usually achieved using **Interfaces** or **Abstract Classes**.

Example: Imagine a PaymentProcessor.

- **Violating OCP:** You have a big if-else block checking for CreditCard or PayPal. To add Bitcoin, you must modify the class.
- **Applying OCP:** You create an interface IPaymentMethod with a method pay(). CreditCard and PayPal implement this. The PaymentProcessor just calls paymentMethod.pay(). To add Bitcoin, you create a new class BitcoinPayment implementing the interface. The PaymentProcessor code remains untouched.

Q2. Violations in SRP and OCP along with their fixes.

SRP Violation:

A common violation in an Examination System is a Student class that handles everything.

The Violation:

Java

```
public class Student {
   public void registerStudent() { } Auth Logic
   public void takeExam() {}     Exam Session Logic
   public void calculateGrade() { }  Grading Logic
   public void sendEmail() { }        Notification Logic
}
```

- **Why it's bad:** If the email provider changes, you risk breaking the exam logic. The class is coupled to too many dependencies.

The Fix: Break it down by responsibility.

1. **AuthService**: Handles registration and login.
2. **ExamSessionService**: Handles the state of the active exam.
3. **GradingService**: Handles math and scoring.
4. **NotificationService**: Handles emails/SMS.

OCP Violation: The "Switch Statement" Trap

A common violation is hardcoding Question Types in the Grading logic.

The Violation:

Java

```
public class Grader {
   public int calculateScore(Question q) {
      if (q.type == "MCQ") {
```

```
            // logic for MCQ
        } else if (q.type == "Descriptive") {
            // logic for Descriptive
        }
        // To add "FillInTheBlank", we must modify this tested class
    }
}
```

The Fix: Use Polymorphism (Strategy Pattern).

1. Create an interface IQuestionEvaluator.
2. Implement MCQEvaluator, DescriptiveEvaluator.
3. The Grader class simply asks the evaluator to calculate the score.

Java

```
// Logic can be extended by adding new classes, not touching existing ones
public class Grader {
    public int calculateScore(IQuestionEvaluator evaluator) {
        return evaluator.evaluate();
    }
}
```