Assignment 1

Student Name: Aariz Amaan **UID:** 23BCS10422
Branch: CSE **Section:** KRG-1A
Semester: 6th **DOP: 04/02/26**
Subject: System Design **Subject Code:** 23CSH-314

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

1. Single Responsibility Principle (SRP)

Definition: The Single Responsibility Principle (SRP) states that:

A class should have only one reason to change.

This means that a class should focus on a single functionality or responsibility. If a class performs multiple unrelated tasks, it becomes difficult to maintain and test.

Problem with Violating SRP

When a class handles more than one responsibility:

- Changes in one functionality may affect others
- The class becomes difficult to understand
- Debugging and testing become complex

Example of SRP Violation

```
class Invoice {
    void calculateTotal() {
        // Calculates total amount
    }

    void printInvoice() {
        // Prints invoice
    }

    void saveToDatabase() {
        // Saves invoice to database
    }
```

```
}
```

Explanation:
The above class performs three different tasks:

1. Calculation of invoice amount
2. Printing the invoice
3. Saving data to the database

Hence, the class has multiple reasons to change, which violates SRP.

Example Following SRP

```
class Invoice {
    double calculateTotal() {
        return 1000;
    }
}

class InvoicePrinter {
    void print(Invoice invoice) {
        System.out.println("Invoice printed");
    }
}

class InvoiceRepository {
    void save(Invoice invoice) {
        System.out.println("Invoice saved to database");
    }
}
```

Explanation:
Each class has a single responsibility:

- Invoice handles business logic
- InvoicePrinter handles printing
- InvoiceRepository handles database operations

This design improves maintainability and readability.

Advantages of SRP

- Easier to understand and modify code
- Improved testability
- Reduced coupling between classes
- Better scalability

2. Open/Closed Principle (OCP)

Defination : The **Open/Closed Principle (OCP)** states that:

Software entities should be open for extension but closed for modification.

This means that new functionality should be added without changing existing code.

Problem with Violating OCP

If existing code is frequently modified:

- There is a high risk of introducing bugs
- Already tested functionality may break
- Code becomes difficult to manage

Example of OCP Violation

```
class DiscountCalculator {
    double calculateDiscount(String customerType) {
        if (customerType.equals("Regular")) {
            return 10;
        } else if (customerType.equals("Premium")) {
            return 20;
        }
        return 0;
    }
}
```

Explanation:

Whenever a new customer type is added, the class must be modified, which violates OCP.

Example Following OCP

```
interface Discount {
    double applyDiscount();
}
class RegularDiscount implements Discount {
    public double applyDiscount() {
        return 10;
    }
}

class PremiumDiscount implements Discount {
    public double applyDiscount() {
        return 20;
    }
}
class DiscountCalculator {
    double calculate(Discount discount) {
        return discount.applyDiscount();
    }
}
```

Explanation:
New discount types can be added by creating new classes without modifying existing ones, thus following OCP.

Advantages of OCP

- Reduces risk of breaking existing code
- Improves system extensibility
- Encourages use of abstraction and polymorphism
- Enhances maintainability

Q2. Discuss in detail about the violations in SRP and OCP along with their fixes.

The SOLID principles provide guidelines for writing clean, maintainable, and scalable object-oriented software. Among them, violations of the Single Responsibility Principle (SRP) and the Open/Closed Principle (OCP) are very common in real-world applications. Understanding these violations and their fixes helps in designing better software systems and avoiding tightly coupled code.

1. **Violations of Single Responsibility Principle (SRP)**
   A violation of SRP occurs when a class has more than one responsibility, meaning it has multiple reasons to change. Such classes are often referred to as *"God Classes"* because they handle multiple unrelated tasks.
   Example of SRP Violation

```
class Employee {
   void calculateSalary() {
      // Salary calculation logic
   }

   void generatePayslip() {
      // Payslip generation logic
   }

   void saveEmployeeData() {
      // Database storage logic
   }
    }
```

Explanation

The Employee class performs:

- Business logic (salary calculation)
- Presentation logic (payslip generation)
- Persistence logic (database operations)

This creates multiple reasons for change and violates SRP.

Problems Caused by SRP Violation

- Difficult to maintain and modify
- Increased coupling
- Harder unit testing
- Higher chance of bugs

Fix for SRP Violation

```
class Employee {
    double calculateSalary() {
        return 50000;
    }
}

class PayslipGenerator {
    void generate(Employee employee) {
        System.out.println("Payslip generated");
    }
}

class EmployeeRepository {
    void save(Employee employee) {
        System.out.println("Employee data saved");
    }
     }
```

2.Violations of Open/Closed Principle (OCP)

An OCP violation occurs when existing code needs to be modified every time new functionality is added. This often happens due to excessive use of conditional statements such as if-else or switch-case.

Example of OCP Violation

```
    class PaymentProcessor {
        double processPayment(String paymentType) {
            if (paymentType.equals("CreditCard")) {
                return 100;
            } else if (paymentType.equals("DebitCard")) {
                return 200;
            }
            return 0;
        }
}
```

Explanation

Whenever a new payment method is introduced, the existing class must be modified,

violating the Open/Closed Principle.

Problems Caused by OCP Violation

- Existing code must be rewritten frequently
- Higher risk of breaking working functionality
- Poor scalability
- Difficult to test and maintain

Fix for OCP Violation

```
interface Payment {
    double pay();
}

class CreditCardPayment implements Payment {
    public double pay() {
        return 100;
    }
}

class DebitCardPayment implements Payment {
    public double pay() {
        return 200;
    }
}
```

```
class PaymentProcessor {
    double process(Payment payment) {
        return payment.pay();
    }
}
```

Explanation

New payment methods can be added by implementing the Payment interface without changing existing code. This makes the system extensible and follows OCP.