Assignment: Single Responsibility Principle (SRP) and Open/Closed Principle (OCP)

Introduction

Q1. Single Responsibility Principle (SRP)

Definition

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

A class should have only one responsibility, and therefore, only one reason to change.

A responsibility refers to a **specific functionality or concern** handled by a class. If a class is responsible for more than one concern, changes in one area may affect the other, leading to increased coupling and reduced maintainability.

Importance of SRP

- Improves **code readability**
- Enhances **maintainability**
- Simplifies **unit testing**
- Reduces **coupling**
- Promotes **high cohesion**

Example of SRP Violation

```
class Employee {
   private String name;
   private double salary;

   public double calculateTax() {
      return salary * 0.2;
   }

   public void saveToDatabase() {
      // database logic
   }

   public String generatePayslip() {
      return "Payslip for " + name;
   }
}
```

Explanation:
The Employee class performs multiple responsibilities such as tax calculation, database

operations, and payslip generation. Any change in tax rules, database structure, or report format will require modifying this class, thereby violating SRP.

SRP-Compliant Design

```
class Employee {
   String name;
   double salary;
}
class TaxCalculator {
   public double calculate(Employee employee) {
      return employee.salary * 0.2;
   }
}
class EmployeeRepository {
   public void save(Employee employee) {
      // database logic
   }
}
class PayslipGenerator {
   public String generate(Employee employee) {
      return "Payslip for " + employee.name;
   }
}
```

Explanation:
Each class now has a **single responsibility**, and changes are isolated, making the system easier to maintain and extend.

Q2. Open/Closed Principle (OCP)

Definition

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

Software entities such as classes, modules, and functions should be open for extension but closed for modification.

This means that new functionality should be added through **extension**, not by modifying existing code.

Importance of OCP

- Prevents regression errors
- Improves system extensibility
- Encourages use of abstraction
- Supports scalable design

Example of OCP Violation

```
class PaymentService {
   public void process(String paymentType) {
      if (paymentType.equals("CARD")) {
         // card payment logic
      } else if (paymentType.equals("UPI")) {
         // upi payment logic
      }
   }
}
```

Explanation:
Whenever a new payment method is introduced, the PaymentService class must be modified, which violates OCP.

OCP-Compliant Design

```
interface PaymentMethod {
   void pay();
}
class CardPayment implements PaymentMethod {
   public void pay() {
      System.out.println("Card payment processed");
   }
}
class UPIPayment implements PaymentMethod {
   public void pay() {
      System.out.println("UPI payment processed");
   }
}
class PaymentService {
   public void process(PaymentMethod payment) {
      payment.pay();
   }
}
```

Explanation:
New payment methods can be added by creating new classes that implement PaymentMethod, without modifying existing code.

Q3. Violations of SRP and Their Fixes

1. God Class

A **God Class** is a class that handles multiple unrelated responsibilities.

```
class OrderManager {
   void createOrder() {}
   void validateOrder() {}
   void saveOrder() {}
```

```
    void sendEmail() {}
    void generateInvoice() {}
}
```

Fix:
Responsibilities should be separated into individual classes such as OrderService, OrderValidator, OrderRepository, NotificationService, and InvoiceGenerator.

2. Mixing Business Logic with Presentation Logic

```
class Report {
    public void generate() {
        System.out.println("Generating report");
    }
}
```

Fix:

```
class ReportService {
    public String generate() {
        return "Report Data";
    }
}
class ReportPrinter {
    public void print(String data) {
        System.out.println(data);
    }
}
```

Q4. Violations of OCP and Their Fixes

1. Use of Conditional Statements

```
double calculateDiscount(String customerType) {
    if (customerType.equals("REGULAR")) return 0.1;
    if (customerType.equals("PREMIUM")) return 0.2;
    return 0;
}
```

Fix Using Polymorphism:

```
interface DiscountPolicy {
    double discount();
}
class RegularCustomer implements DiscountPolicy {
    public double discount() { return 0.1; }
}
class PremiumCustomer implements DiscountPolicy {
    public double discount() { return 0.2; }
}
```

2. Hardcoded Dependencies

```
class NotificationService {
   EmailService email = new EmailService();
}
```

Fix:

```
interface Notifier {
   void notifyUser();
}
class EmailNotifier implements Notifier {
   public void notifyUser() {}
}
class SMSNotifier implements Notifier {
   public void notifyUser() {}
}
class NotificationService {
   private Notifier notifier;
   public NotificationService(Notifier notifier) {
      this.notifier = notifier;
   }
}
```

Conclusion

The **Single Responsibility Principle (SRP)** ensures that a class remains focused on a single task, making the codebase easier to maintain and understand. The **Open/Closed Principle (OCP)** promotes extensibility by encouraging the use of abstraction and polymorphism instead of modification.

Adhering to these principles leads to **clean architecture**, **reduced coupling**, and **highly scalable software systems**, which are essential characteristics of modern enterprise applications.