CU
CHANDIGARH UNIVERSITY

DEPARTMENT OF
COMPUTER SCIENCE &
ENGINEERING

Discover. Learn. Empower.

Assignment : 1

Student Name: Sandeep kumar **UID: 23BCS11489**

Branch: BE-CSE **Group: KRG-3_B**

Semester: 6th **Date of Performance: 4/02/26**

Subject Name: System Design **Subject Code: 23CSH-314**

Question 1. Explain SRP and OCP in detail with proper examples.

1. Single Responsibility Principle (SRP)

Definition

The **Single Responsibility Principle** states that:

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

In simple words, **one class = one job**.

Why SRP Is Important

- Makes code easier to understand
- Reduces impact of changes
- Improves maintainability
- Simplifies testing

Example (Without SRP)

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

void saveEmployee() {
   // database save logic
}

void generateReport() {
   // report generation logic
}
}
```

Problem:

This class has **three responsibilities**:

1. Business logic
2. Database operations
3. Reporting

Any change in database or reporting forces modification of the same class.

Example (With SRP Applied)

```
class SalaryCalculator {

void calculateSalary() {
    // salary calculation logic
 }
}

class EmployeeRepository {
 void saveEmployee() {
    // database logic
 }
}

class ReportGenerator {
 void generateReport() {
    // report generation logic
 }
}
```

Result:

Each class has **one responsibility** and **one reason to change**.

2. Open/Closed Principle (OCP)

Definition

The **Open/Closed Principle** states that:

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

This means you should be able to **add new features without changing existing code**.

Why OCP Is Important

- Prevents breaking existing functionality
- Makes system extensible
- Encourages clean architecture
- Reduces regression bugs

Example (Without OCP)

```
class Shape {
 String type;
}

class AreaCalculator {
 double calculateArea(Shape shape) {
    if (shape.type.equals("Circle")) {
        // circle area logic
    } else if (shape.type.equals("Rectangle")) {
        // rectangle area logic
    }
```

Problem:
Adding a new shape (Triangle) requires modifying calculateArea().

◆ **Example (With OCP Applied)**

```
interface Shape {
 double calculateArea();
}

class Circle implements Shape {
 double radius;

 public double calculateArea() {
     return Math.PI * radius * radius;
   }
}

class Rectangle implements Shape {
 double length, breadth;

 public double calculateArea() {
     return length * breadth;
 }
}

class AreaCalculator {
 double calculateArea(Shape shape) {
     return shape.calculateArea();
 }
}
```

Result:

To add a new shape:

```
Class Triangle implements Shape {
 double base, height;

 public double calculateArea() {
     return 0.5 * base * height;
   }
}
```

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

1. Violations of Single Responsibility Principle (SRP)

SRP Reminder

The **Single Responsibility Principle** states that a class should have **only one responsibility** and **one**

SRP Violation

violation occurs when a **single class performs multiple unrelated tasks**.

Example (SRP Violation)

```
class User {
 void validateUser() {
    // validation logic
 }

 void saveUser() {
    // database operation
 }

 void sendEmail() {
    // email notification
 }
}
```

Why This Is a Violation

This class has **three responsibilities**:

1. Validation logic
2. Database persistence
3. Email communication

Changes in **any one responsibility** force changes in the same class, making it:

- Hard to maintain
- Difficult to test
- Error-prone

Fix for SRP Violation

Solution: Separate each responsibility into its own class.

Example (SRP Applied)

```
class UserValidator {
 void validateUser() {
    // validation logic
 }
}

class UserRepository {
 void saveUser() {
    // database logic
 }
}
```

```
class EmailService {
 void sendEmail() {
    // email logic
 }
}
```

Advantages After Fix

- Each class has **one clear purpose**
- Changes are isolated
- Code becomes more readable and reusable

2. Violations of Open/Closed Principle (OCP)

OCP Reminder

The **Open/Closed Principle** states that software components should be **open for extension** but **closed for modification**.

OCP Violation

violation occurs when **new functionality requires modifying existing code**, usually through if-else

Example (OCP Violation)

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

Why This Is a Violation

- Adding a new customer type requires **modifying the method**
- Existing tested code is altered
- Increases chances of regression bugs

Fix for OCP Violation

Solution: Use **abstraction and polymorphism**.

Example (OCP Applied)

```
 interface Discount {
 double getDiscount();
}

class RegularCustomer implements Discount {
 public double getDiscount() {
    return 10;
 }
}

class PremiumCustomer implements Discount {
 public double getDiscount() {
    return 20;
  }
}

class DiscountCalculator {
 double calculateDiscount(Discount discount) {
    return discount.getDiscount();
 }
}
```

Adding New Discount Type (No Modification)

```
class VIPCustomer implements Discount {
 public double getDiscount() {
    return 30;
 }
}
```

No changes required in existing classes.