SOLID Principles - SRP, OCP, LSP (Java mein Hinglish)

1. Single Responsibility Principle (SRP)

Matlab: Har class ka sirf ek hi kaam hona chahiye. Ek class sirf ek reason se change honi chahiye.

❌ Galat Example:

java

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

    public Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    // Salary calculation - ek responsibility
    public double calculateAnnualSalary() {
        return salary * 12;
    }

    // Database operations - dusri responsibility
    public void saveToDatabase() {
        System.out.println("Saving " + name + " to database");
        // Database logic yahan
    }

    // Email operations - teesri responsibility
    public void sendEmail() {
        System.out.println("Sending email to " + name);
        // Email logic yahan
    }
}
```

Problem: Ye class teen different kaam kar rahi hai!

✅ Sahi Example:

java

```java
// Sirf employee data ke liye
class Employee {
    private String name;
    private double salary;

    public Employee(String name, double salary) {
        this.name = name;
        this.salary = salary;
    }

    // Getters
    public String getName() { return name; }
    public double getSalary() { return salary; }
}

// Sirf salary calculation ke liye
class SalaryCalculator {
    public double calculateAnnualSalary(Employee employee) {
        return employee.getSalary() * 12;
    }
}

// Sirf database operations ke liye
class EmployeeRepository {
    public void save(Employee employee) {
        System.out.println("Saving " + employee.getName() + " to database");
    }
}

// Sirf email operations ke liye
class EmailService {
    public void sendEmail(Employee employee) {
        System.out.println("Sending email to " + employee.getName());
    }
}
```

Diagram:

❌ Galat Way:

```
[Employee Class]
├── calculateSalary()
├── saveToDatabase()
└── sendEmail()
```

✅ Sahi Way:

```
[Employee] ← [SalaryCalculator]
        ← [EmployeeRepository]
        ← [EmailService]
```

---

2. Open-Closed Principle (OCP)

Matlab: Classes extension ke liye open, modification ke liye closed. Naya feature add karne ke liye existing code change nahi karna chahiye.

❌ Galat Example:

java

```java
class AreaCalculator {
    public double calculateArea(Object shape) {
        if (shape instanceof Rectangle) {
            Rectangle rect = (Rectangle) shape;
            return rect.getWidth() * rect.getHeight();
        } else if (shape instanceof Circle) {
            Circle circle = (Circle) shape;
            return Math.PI * circle.getRadius() * circle.getRadius();
        }
        // Har naye shape ke liye yahan modification karna padega!
        return 0;
    }
}

class Rectangle {
    private double width, height;
    // constructor aur getters
}

class Circle {
    private double radius;
    // constructor aur getters
}
```

✅ Sahi Example:

java

```java
// Base interface
interface Shape {
    double calculateArea();
}

class Rectangle implements Shape {
    private double width;
    private double height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }

    @Override
    public double calculateArea() {
        return width * height;
    }
}

class Circle implements Shape {
    private double radius;

    public Circle(double radius) {
        this.radius = radius;
    }

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

// Naya shape add karna easy hai!
class Triangle implements Shape {
    private double base;
    private double height;

    public Triangle(double base, double height) {
        this.base = base;
        this.height = height;
    }

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

```
    }
}

class AreaCalculator {
    public double calculateTotalArea(Shape[] shapes) {
        double total = 0;
        for (Shape shape : shapes) {
            total += shape.calculateArea(); // Polymorphism!
        }
        return total;
    }
}
```

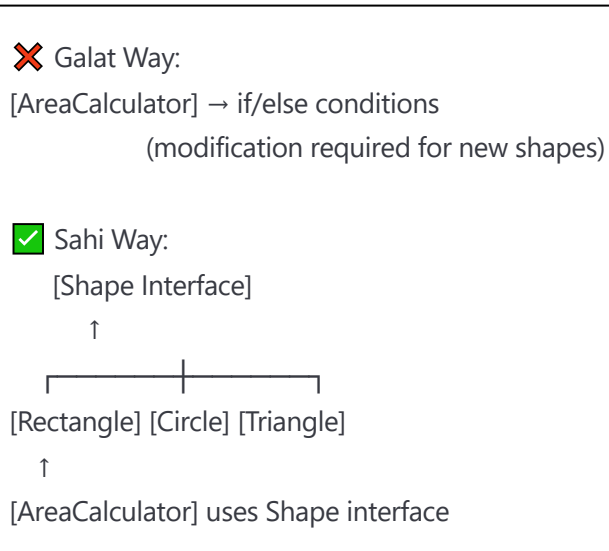
Usage Example:

```
java

public class Main {
    public static void main(String[] args) {
        Shape[] shapes = {
            new Rectangle(5, 10),
            new Circle(3),
            new Triangle(4, 6)
        };

        AreaCalculator calculator = new AreaCalculator();
        System.out.println("Total Area: " + calculator.calculateTotalArea(shapes));
    }
}
```

Diagram:



---

3. Liskov Substitution Principle (LSP)

Matlab: Child class ko parent class ki jagah use kar sakte hain bina problem ke. Child class parent ke behavior ko break nahi karna chahiye.

❌ Galat Example:

java

```java
class Bird {
    public void fly() {
        System.out.println("Bird is flying");
    }
}

class Penguin extends Bird {
    @Override
    public void fly() {
        throw new UnsupportedOperationException("Penguins can't fly!");
    }
}

// Client code
class BirdHandler {
    public void makeBirdFly(Bird bird) {
        bird.fly(); // Penguin ke liye exception aayegi!
    }
}
```

Problem: Penguin ko Bird ki jagah use nahi kar sakte safely.

✅ Sahi Example:

java

```java
// Base class sirf common behavior ke liye
abstract class Bird {
    public abstract void eat();
    public abstract void makeSound();
}

// Flying birds ke liye separate interface
interface Flyable {
    void fly();
}

// Swimming birds ke liye separate interface
interface Swimmable {
    void swim();
}

class Sparrow extends Bird implements Flyable {
    @Override
    public void eat() {
        System.out.println("Sparrow is eating seeds");
    }

    @Override
    public void makeSound() {
        System.out.println("Chirp chirp!");
    }

    @Override
    public void fly() {
        System.out.println("Sparrow is flying");
    }
}

class Penguin extends Bird implements Swimmable {
    @Override
    public void eat() {
        System.out.println("Penguin is eating fish");
    }

    @Override
    public void makeSound() {
        System.out.println("Squawk!");
    }

    @Override
    public void swim() {
```

```
        System.out.println("Penguin is swimming");
    }
}

// Client code
class BirdHandler {
    public void handleBird(Bird bird) {
        bird.eat(); // Safe for all birds
        bird.makeSound(); // Safe for all birds
    }

    public void makeFlyableFly(Flyable flyable) {
        flyable.fly(); // Safe sirf flying birds ke liye
    }

    public void makeSwimmableSwim(Swimmable swimmable) {
        swimmable.swim(); // Safe sirf swimming birds ke liye
    }
}
```

Usage Example:

java

```
public class Main {
    public static void main(String[] args) {
        Bird sparrow = new Sparrow();
        Bird penguin = new Penguin();

        BirdHandler handler = new BirdHandler();

        // Dono birds ko safely handle kar sakte hain
        handler.handleBird(sparrow);
        handler.handleBird(penguin);

        // Type-specific operations
        if (sparrow instanceof Flyable) {
            handler.makeFlyableFly((Flyable) sparrow);
        }

        if (penguin instanceof Swimmable) {
            handler.makeSwimmableSwim((Swimmable) penguin);
        }
    }
}
```

Diagram:

Key Points:

1. **SRP**: Ek class = Ek responsibility
2. **OCP**: Extension ke liye open, modification ke liye closed
3. **LSP**: Child class parent ki jagah safely use hona chahiye

Ye principles follow karke aap clean, maintainable aur scalable code likh sakte hain!