1. S - Single Responsibility Principle (SRP)

Resources

- Video → 1. SOLID Principles with Easy Examples (Hindi) | OOPs SOLID Principles - Low Level Design

What is the Single Responsibility Principle (SRP)?

> *Single Responsibility Principle(SRP) states that* ***"A class should have only ONE reason to change"****, meaning it should have ONE and ONLY ONE job or responsibility.*

Meaning, if the class has multiple jobs or responsibilities, changes to one responsibility might affect or break the other critical responsibilities, making the **code bloated, fragile, and harder to maintain.** Hence, the Single Responsibility Principle(SRP) focuses on restricting the concern to only a single responsibility.

Code Example: Violating SRP

```java
public class Marker {
    String name;
    String color;
    int price;
    int year;

    public Marker(String name, String color, int price, int year) {
        this.name = name;
        this.color = color;
        this.price = price;
        this.year = year;
    }
}

// BAD: This class violates SRP by having multiple responsibilities
public class Invoice {
    private Marker marker;
    private int quantity;
    private int total;

    public Invoice(Marker marker, int quantity) {
        this.marker = marker;
        this.quantity = quantity;
    }

    // Responsibility 1: Calculate the total(business logic)
    public void calculateTotal() {
        System.out.println("Calculating total...");
        this.total = this.marker.price * this.quantity;
    }

    // Responsibility 2: Database Operations
    public void saveToDB() {
        // Save the data into DB
        System.out.println("Saving to DB...");
    }

    // Responsibility 3: Print the Invoice
    public void printInvoice() {
        // print the Invoice
        System.out.println("Printing Invoice...");
    }
}

// Usage example
public class Demo {
    public static void main(String[] args) {
        Invoice invoice = new Invoice(new Marker("name", "color", 10, 2020), 10);
        invoice.calculateTotal();
        invoice.saveToDB();
        invoice.printInvoice();
    }
}
```

Problems with the Above Code

The `Invoice` class has 3 different responsibilities

1. `calculateTotal()` → Responsibility 1: Calculate the total(business logic)
2. saveToDB() → Responsibility 2: Database Operations
3. printInvoice() → Responsibility 3: Print the Invoice

This violates SRP because:

- If the tax calculation rules change, we need to modify the `Invoice` class.
- If the database structure changes, we need to modify the `Invoice` class.
- If the printing requirement changes, we need to modify the `Invoice` class.

Code Example: Follows SRP

Here's the refactored code that follows the Single Responsibility Principle:

```java

public class Marker {
    String name;
    String color;
    int price;
    int year;

    public Marker(String name, String color, int price, int year) {
        this.name = name;
        this.color = color;
        this.price = price;
        this.year = year;
    }
}

// GOOD: Following SRP - Each class has a single responsibility

// Responsibility: Managing Invoice data only
public class Invoice {

    private Marker marker;
    private int quantity;
    private int total;

    public Invoice(Marker marker, int quantity) {
        this.marker = marker;
        this.quantity = quantity;
    }

    // Responsibility 1: Calculate the total(business logic)
    public void calculateTotal() {
        System.out.println("Calculating total...");
        this.total = this.marker.price * this.quantity;
    }
}

// Responsibility 2: Managing Database Operations only
public class InvoiceDao {

    Invoice invoice;

    public InvoiceDao(Invoice invoice) {
        this.invoice = invoice;
    }

    public void saveToDB() {
        // Save into the DB the invoice
        System.out.println("Saving to DB...");
    }
}

// Responsibility 3: Printing the Invoice only
public class InvoicePrinter {

    private Invoice invoice;

    public InvoicePrinter(Invoice invoice) {
        this.invoice = invoice;
    }

    public void print() {
        // print the invoice
        System.out.println("Printing Invoice...");
    }
}

// Usage example showing how all classes work together
public class Demo {
    public static void main(String[] args) {

        // create the service objects
        Invoice invoice = new Invoice(new Marker("name", "color", 10, 
2020), 10);
        InvoiceDao invoiceDao = new InvoiceDao(invoice);
        InvoicePrinter invoicePrinter = new InvoicePrinter(invoice);

        // Use the services
        invoice.calculateTotal();
        invoiceDao.saveToDB();
        invoicePrinter.print();
    }
}
```

Key Benefits of the Refactored Code

- Single Responsibility: Each class now has only one reason to change
    - `Invoice` : Only changes if the `Invoice` tax calculation rules change
    - `InvoiceDao` : Only changes if `InvoiceDao` database operations change
    - `InvoicePrinter` : Only changes if `InvoicePrinter` requirements change
- Better Maintainability: Changes to one functionality don't affect others.

- Improved Testability: Each class can be tested in isolation.
- Enhanced Reusability: Classes can be reused in different contexts. For example, different types of invoices(domestic and international) can use the same `InvoicePrinter` class.

Summary

The Single Responsibility Principle (SRP) is key to maintainable, testable, and flexible code. It ensures each class has one reason to change, fostering a modular system. SRP isn't about limiting classes to a single method, but rather about ensuring they have one responsibility, even with multiple methods that align with that purpose.