

1. S - Single Responsibility Principle (SRP)

What is the Single Responsibility Principle (SRP)?

Code Example: Violating SRP

Problems with the Above Code

Code Example: Follows SRP

Key Benefits of the Refactored Code

Summary

▼ 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

```
1 public class Marker {
2     String name;
3     String color;
4     int price;
5     int year;
6
7     public Marker(String name, String color, int price, int year) {
8         this.name = name;
9         this.color = color;
10        this.price = price;
11        this.year = year;
12    }
13 }
14
15 // BAD: This class violates SRP by having multiple responsibilities
16 public class Invoice {
17     private Marker marker;
18     private int quantity;
19     private int total;
20
21     public Invoice(Marker marker, int quantity) {
22         this.marker = marker;
23         this.quantity = quantity;
24     }
25
26     // Responsibility 1: Calculate the total(business logic)
27     public void calculateTotal() {
28         System.out.println("Calculating total...");
29         this.total = this.marker.price * this.quantity;
30     }
31
32     // Responsibility 2: Database Operations
33     public void saveToDB() {
34         // Save the data into DB
```

```

35     System.out.println("Saving to DB...");
36 }
37
38 // Responsibility 3: Print the Invoice
39 public void printInvoice() {
40     // print the Invoice
41     System.out.println("Printing Invoice...");
42 }
43 }
44
45 // Usage example
46 public class Demo {
47     public static void main(String[] args) {
48         Invoice invoice = new Invoice(new Marker("name", "color", 10,
49         2020), 10);
50         invoice.calculateTotal();
51         invoice.saveToDB();
52         invoice.printInvoice();
53     }
54 }

```

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:

```

1
2 public class Marker {
3     String name;
4     String color;
5     int price;
6     int year;
7
8     public Marker(String name, String color, int price, int year) {
9         this.name = name;
10        this.color = color;
11        this.price = price;
12        this.year = year;
13    }
14 }
15
16 // GOOD: Following SRP - Each class has a single responsibility
17 // Responsibility: Managing Invoice data only
18 public class Invoice {
19
20
21     private Marker marker;
22     private int quantity;
23     private int total;
24
25     public Invoice(Marker marker, int quantity) {

```

```

26         this.marker = marker;
27         this.quantity = quantity;
28     }
29
30     // Responsibility 1: Calculate the total(business logic)
31     public void calculateTotal() {
32         System.out.println("Calculating total...");
33         this.total = this.marker.price * this.quantity;
34     }
35 }
36
37 // Responsibility 2: Managing Database Operations only
38 public class InvoiceDao {
39
40     Invoice invoice;
41
42     public InvoiceDao(Invoice invoice) {
43         this.invoice = invoice;
44     }
45
46     public void saveToDB() {
47         // Save into the DB the invoice
48         System.out.println("Saving to DB...");
49     }
50 }
51
52 // Responsibility 3: Printing the Invoice only
53 public class InvoicePrinter {
54
55     private Invoice invoice;
56
57     public InvoicePrinter(Invoice invoice) {
58         this.invoice = invoice;
59     }
60
61     public void print() {
62         // print the invoice
63         System.out.println("Printing Invoice...");
64     }
65 }
66
67 // Usage example showing how all classes work together
68 public class Demo {
69     public static void main(String[] args) {
70
71         // create the service objects
72         Invoice invoice = new Invoice(new Marker("name", "color", 10,
2020), 10);
73         InvoiceDao invoiceDao = new InvoiceDao(invoice);
74         InvoicePrinter invoicePrinter = new InvoicePrinter(invoice);
75
76         // Use the services
77         invoice.calculateTotal();
78         invoiceDao.saveToDB();
79         invoicePrinter.print();
80     }
81 }

```

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.