

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Assignment: 1

Student name: Mohammad Kaif
Branch: B.E./C.S.E.
Semester: 6th
Subject Name: System Design
Subject Code: 23CSH-314

UID: 23BCS12884
Section/Group: KRG-2A
Date of Submission: 04/02/2026

1. Aim:

- Q1. Explain SRP and OCP in detail with proper examples.
- Q2. Discuss in detail about the violations in SRP and OCP along with their fixes.
- Q3. Design an HLD for an Online Examination System applying these principles.

2. Objectives:

- To understand SOLID principle to explain what is SRP and OCP
- To make an .io file representing the Online Examination System.

Single Responsibility Principle (SRP) - The S in SOLID principle

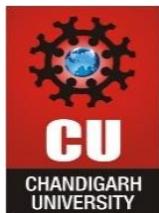
In object-oriented software engineering, maintaining **clean, manageable, and scalable code** is a major challenge. To address this, **Robert C. Martin (Uncle Bob)** introduced the **SOLID principles**, which are a set of five design principles that help developers build robust software systems.

The Single Responsibility Principle (SRP) is the first and most fundamental principle of SOLID. It states that :

- A class should have only one responsibility and therefore only one reason to change
- Every class must perform a single functionality.
- That job should be completely focused
- Any change in the system should affect only one class

Without SRP:

- Classes become large and complex
- Small changes cause unexpected bugs
- Testing becomes difficult
- Maintenance cost increases



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

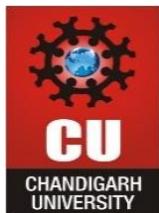
Discover. Learn. Empower.

With SRP:

- Code becomes modular
- Classes are easy to modify
- System becomes scalable and flexible

Example we have :

```
class BankService {  
  
    public void deposit(double amount, int accountNo) {  
        // deposit logic  
    }  
  
    public void withdraw(double amount, int accountNo) {  
        // withdrawal logic  
    }  
  
    public void printPassbook(int accountNo) {  
        // passbook printing logic  
    }  
  
    public void getLoanInfo(String loanType) {  
        if (loanType.equals("HOME")) {  
            // home loan logic  
        } else if (loanType.equals("PERSONAL")) {  
            // personal loan logic  
        }  
    }  
  
    public void sendOTP(String method) {  
        if (method.equals("EMAIL")) {  
            // send OTP via email  
        }  
    }  
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

}

}

Violations :

If we observe the given class, it clearly violates the Single Responsibility Principle because it has multiple reasons to change.

For example:

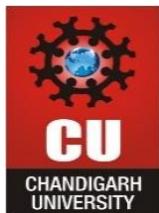
- If a new loan type (like Gold Loan) is added, the loan logic must be modified.
- If the OTP sending method changes from Email to Mobile/SMS, the notification logic must be changed.

How to fix it ?

Instead of using one large class:

- Separate responsibilities into different classes/services
- Each class should handle only one functionality

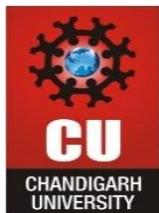
```
class PrinterService {  
  
    public void printPassbook(int accountNo) {  
  
        // logic to update and print passbook information  
  
    }  
  
}  
  
class LoanService {  
  
    public void getLoanInfo(String loanType) {  
  
        if (loanType.equals("HOMELOAN")) {  
  
            // home loan logic  
  
        }  
  
        else if (loanType.equals("PERSONAL")) {  
  
            // personal loan logic  
  
        }  
  
    }  
  
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
else if (loanType.equals("CAR")) {  
    // car loan logic  
}  
  
else if (loanType.equals("GOLD")) {  
    // gold loan logic  
}  
  
}  
  
}  
  
class NotificationService {  
  
    public void sendOTP(String method) {  
        if (method.equals("EMAIL")) {  
            // email API logic to send OTP  
        }  
  
        else if (method.equals("MOBILE")) {  
            // SMS API logic to send OTP (Twilio or similar)  
        }  
  
    }  
  
}
```



Open Closed Principle (OCP) – O of SOLID

- Open Closed Principle states that a software module should be open for extension but closed for modification
- This means new functionality should be added by extending the existing code, not by changing it.
- In other words, we should be able to change the behavior of a class without modifying its source code.

Example we have :

```
CLASS NOTIFICATION_SERVICE
{
    PUBLIC SEND_OTP(string method)
    {
        IF (method == "EMAIL")
        {
            // send otp using email api
        }

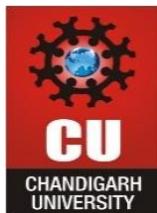
        IF (method == "MOBILE")
        {
            // send otp using mobile api
        }
    }
}
```

Violations:

- A new requirement comes to send otp through whatsapp, we must add one more IF condition inside the same class.
- This means the existing class is being modified again and again, which violates the Open Closed Principle.

Every new notification type requires:

- modifying existing code
- adding more IF conditions
- increasing risk of bugs
- breaking already working logic



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

How to fix it ?

According to OCP, instead of modifying the existing class, we should extend its behavior. This is usually achieved using interfaces.

```
public interface NotificationService
{
    public void sendOtp(string method); // abstract method
}

public class EmailNotification implements NotificationService
{
    public void sendOtp(string method)
    {
        // call email notification api here
    }

    // other email-related notifications can be added here
}

public class MobileNotification implements NotificationService
{
    public void sendOtp(string method)
    {
        // call mobile notification api here
    }
}

public class WhatsappNotification implements NotificationService
{
    public void sendOtp(string method)
    {
        // call whatsapp notification api here
    }
}
```