

# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Assignment 1

**Student Name:** Sujal Srivastava

**UID:** 23BCS11842

**Branch:** BE-CSE

**Section/Group:** KRG-3B

**Semester:** 6<sup>th</sup>

**Date of Submission:** 04/02/26

**Subject Name:** SYSTEM DESIGN

**Subject Code:** 23CSH-314

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

ANS:

Single Responsibility Principle : The Single Responsibility Principle states that a class should have only one reason to change, meaning it should do only one specific job.

If a class handles multiple responsibilities, any change in one responsibility may affect the others, making the system harder to maintain.

Example :

```
class Student {  
    void calculateResult() {  
        // logic to calculate result  
    }  
  
    void saveToDatabase() {  
        // logic to save student data  
    }  
  
    void printReport() {  
        // logic to print report  
    }  
}
```

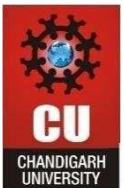
Here, the Student class:

- Calculates results
- Saves data
- Prints reports

This violates SRP because multiple responsibilities are mixed.

Example :

```
class Student {  
    String name;  
    int marks;  
}
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
class ResultCalculator {  
    void calculate(Student s) { }  
}  
  
class StudentRepository {  
    void save(Student s) { }  
}  
  
class ReportPrinter {  
    void print(Student s) { }  
}
```

Now each class has one responsibility, making the design clean and maintainable.

**Open/Closed Principle :** The Open/Closed Principle states that software entities should be open for extension but closed for modification.  
This means we should be able to add new functionality without changing existing code.

Example :

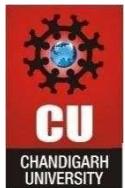
```
class Payment {  
    void pay(String type) {  
        if(type.equals("Card")) {  
            // card payment logic  
        } else if(type.equals("UPI")) {  
            // upi payment logic  
        }  
    }  
}
```

If a new payment method is added, we must modify this class, which violates OCP.

Example :

```
interface Payment {  
    void pay();  
}  
  
class CardPayment implements Payment {  
    public void pay() { }  
}  
  
class UPIPayment implements Payment {  
    public void pay() { }  
}
```

Now new payment methods can be added by creating new classes without modifying existing code.



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

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

ANS:

SRP Violation :

Problem:

When a single class performs multiple tasks, changes in one task may break others.  
It leads to:

- Difficult debugging
- Poor readability
- High maintenance cost

Fix:

Split responsibilities into separate classes, each handling only one task.

OCP Violation :

Problem:

When new requirements force us to modify existing code repeatedly, it increases:

- Risk of bugs
- Dependency issues

Use:

- Interfaces
- Abstract classes
- Polymorphism

This allows extending behavior without touching existing code.

3. Design an HLD for an Online Examination System applying these principles.

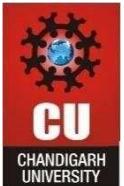
ANS:

High Level Design (HLD) – Online Examination System

Main Components :

1. User Management Module
  - Student
  - Admin
  - Authentication Service
2. Exam Management Module
  - Exam Creator
  - Question Bank
  - Exam Scheduler
3. Evaluation Module
  - Answer Submission
  - Auto Evaluation
  - Result Calculation
4. Notification Module
  - Email Service
  - SMS Service

Applying SRP :



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Each class has only one responsibility:

- Student → Stores student data
- AuthService → Handles login/authentication
- ExamService → Manages exams
- ResultService → Calculates results
- NotificationService → Sends notifications

This ensures that changes in one feature do not affect others.

Applying OCP :

Interfaces are used to allow extension:

```
interface Evaluation {  
    void evaluate();  
}  
  
class ObjectiveEvaluation implements Evaluation {  
    public void evaluate() { }  
}  
  
class DescriptiveEvaluation implements Evaluation {  
    public void evaluate() { }  
}
```

If a new evaluation type is introduced (e.g., AI-based evaluation), we simply add a new class without modifying existing ones.

---

Advantages of This Design

- Easy to maintain
- Scalable for future features
- Less error-prone
- Follows clean architecture principles