**Design patterns & Solid Principles.**

**Solid Principles:-**

**S**ingle Responsibility Principle (SRP)

**O**pen/Closed Principle (OCP)

**L**iskov Substitution Principle (LSP)

**I**nterface Segregation Principle (ISP)

**D**ependency Inversion Principle (DIP)

**SRP-**

states that a class should have only **one reason to change**, meaning it should focus on a single responsibility or functionality. This principle ensures that the class is easy to maintain, test, and understand.

Imagine we are creating a program to manage employees. We have a class that handles employee details, saving employee data to a database, and generating reports.

Issues with This Design:

**Multiple Responsibilities**: The Employee class handles:

* Employee data management.
* Database operations.
* Report generation.

This violates SRP because:

* Changes in database logic will affect the Employee class.
* Changes in reporting logic will also affect the Employee class.

We fix this by refactor the code by separating responsibilities into different classes.

**(OCP)-**

is a design principle in object-oriented programming that states:  
**"Software entities (classes, modules, functions) should be open for extension but closed for modification."**

This means:

* You can add new functionality to an existing class without altering its existing code.
* This minimizes the risk of introducing bugs in the existing functionality and makes the system more maintainable.

we should wright the code in such a way that if we need to add a new functionality we shouldn't modify the code instead we should wright it in a separate class and just extend it.

**LSP-**

**"Objects of a superclass should be replaceable with objects of its subclasses without affecting the correctness of the program."**

In simpler terms, any subclass should be able to stand in for its superclass without altering the desired behaviour of the program.

Eg: <https://github.com/MohammadAasimB/dp-sp/blob/main/Liskov%20Substitution%20Principle%20(LSP)>