

## (2) Solid Design Principles

What are SOLID principles?

SOLID principles of OOPS (Object Oriented Programming System).

The five concepts make up our SOLID principles :-

S → Single Responsibility Principle

O → Open/Closed Principle

L → Liskov Substitution Principle

I → Interface Segmented Principle

D → Dependency Inversion Principle

Why we need SOLID principle/  
Advantages of SOLID Principles

Help us to write better code by :-

- Avoiding duplicate values
- Easy to maintain
- Easy to Understand
- Flexible software
- Reduce Complexity

## (11) S → Single Responsibility Principle (One reason to change).

- A class should have only 1 responsibility principle / 1 reason to change.

### Advantages:-

- Simplifies Debugging
- Testing and
- Enhances maintainability

(Q) Have you ever seen the worst single responsibility principle violation?

Yes When I merge my <sup>UI Layer /</sup> ~~entity layer~~  
.. Entity layer with my service layer.  
(As all the layers are segregated accordingly)  
As:-

**UI Layer** → Represents / accepts the requests from the server and is very first layer of Client-server content.

**Service Layer** → Service layer / Business layer where we write business logics.



## Why Single Responsibility Principle Important?

(a) In Real world, Requirement changes and so does our code implementation to maintain the changing requirements.

(b) Testing is Easier → With a single responsibility, the class will have

(c) Easier to understand

(d) Prevents frequent changes to the same class.

## What will happen if there will be frequent changes to same class?

The more we <sup>makes</sup> changes to the same class the existing implemented functionality of that class may shift here and there because of which some code may make and can face problem during production.

(2) → 0 → Open for Extension but closed for modification

## Benefits

Promotes flexibility and minimizes the risk of breaking existing code.

Open for Extension → you should be able to add new functionality to a class.  
closed for modification → you shouldn't modify existing code in the class to add new functionality.

The term "open for extension" means that we can extend and include extra functionalities/features in our code without affecting/altering our existing implementation.

The term "closed for modification"

means that after we add the extra functionality, we should not modify the existing implementation.

## How to implement Open/Closed principles

By simply extends the class and override some functions.



## QVI (3) / L → Liskov Substitution Principles

If class B is subtype of class A, then we should be able to replace object of A with B without breaking the behaviour of the program.

Subclass should extend the capability of parent class not narrow it down.

### Advantages

Ensures Compatibility and promotes reusable and reliable code.

Wrongly implementing it can prove real world objects wrong like:

✓✓✓  
Interview

Square is a rectangle, in real it's not true but can be easily implemented through code. To prevent this better use Liskov Substitution Principles.

Is Square a subclass of Rectangle?

No making Square a Subclass of Rectangle in object-oriented programming is considered a bad idea in practice because it violates the Liskov Substitution Principle (LSP), which is a key principle of object-oriented design.

(4) I → Interface

Principle

Segregation

- Interfaces should be such that client should not implement unnecessary functions they do not need.
- ISP states that we should split our interfaces into smaller and more specific ones.

Advantages

- Reduces code bloat and enhances modularity.
- To prevent Client from unnecessarily getting stuck in implementing unwanted methods.



### 5) Dependency Inversion Principle

- (a) It states that "Depend on abstractions, not on concretions".
- (b) we should design our software in such a way that various modules can be separated from each other using an abstract layer to bind them together.