**SOLID Principles**

**What are SOLID Principles:**

They are five design guidelines intended to make software design more understandable, flexible, and maintainable. These principles were introduced by Robert C. Martin and are foundational in Object-Oriented design (OOD), they help in avoiding software design issues that can lead to rigid, fragile, and hard-to-maintain code.

**The SOLID Stands for:**

1. **S** – Single Responsibility Principle (SRP).
2. **O** – Open/Closed Principle (OCP).
3. **L** – Liskov Substitution Principle (LSP).
4. **I** – Interface Segregation Principle (ISP).
5. **D** – Dependency Inversion Principle (DIP).

**Single Responsibility Principle**

A class should have only one reason to change, meaning it should have only one job or responsibility.

This principle helps to ensure that each class or module in you system does one thing and does it well. If class has more than one responsibility, changes to one responsibility may affect the others, making the class more difficult to maintain.

**Open Closed Principle (OCP)**

Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification.

This means that you should be able to extend the behavior of a class without modifying its source code. It’s a way of ensuring that changes don’t break existing code, allowing for greater flexibility and less risk when introducing new functionality.

In software OCP encourages you to extent the functionality of your class without altering their original structure, much like extending your house without knocking it down.

And ensures that the core class (the app) doesn’t need to be modified for every new feature (plugin).

**Liskov Substitution Principle (LSP)**

Objects of a superclass should be replaceable with objects of a subclass without affecting the correctness of the program.

In simpler terms, if a class B is a subclass of class A, then B should be able to replace A without breaking the behavior of the system.

**Any subclass should only extend the behavior of the parent class, not narrow it down.**

Which means, derived classes should be substitutable for their base classes without altering the behavior of the system. This principle is used to ensure that inheritance is used properly.

* LSP ensures that subclass behave consistently when replacing the base class.
* It’s about ensuring that subclasses can replace their base class without breaking the socket. If a class is derived from another, it should still work as expected when substituted.

**Interface Segregation (ISP)**

Clients should not be forced to depend on interfaces they don’t use.

Instead of one large interface, you should break down the interface into smaller, more specific ones. This allows clients to depend only on the methods they need, rather than being force to implement unused methods.

ISP works by ensuring that the clients only interact with the specific functionality they need, without being forced to deal with unnecessary details.

It follows this logic by creating smaller, more specific interfaces so that the clients use only the methods they need.

It is like having those interfaces each interface should offer only what the client needs, not a bloated of unrelated methods. It’s like breaking big-bulky interfaces into smaller, focused ones so that client can use just what they need.