SOLID Design Principles

## SOLID Introduction

1. SOLID principles are the design principles that enable us manage most of the software design problems.
2. The term SOLID is an acronym for five design principles intended to make software designs more understandable, flexible and maintainable
3. The principles are a subset of many principles promoted by Robert C. Martin
4. The SOLID acronym was first introduced by Michael Feathers

## SOLID Acronym

* S: Single Responsibility Principle (SRP)
* O: Open Closed Principle (OSP)
* L: Liskov Substitution Principle (LSP)
* I: Interface Segregation Principle (ISP)
* D: Dependency Inversion Principle (DIP)

## Single Responsibility Principle:

**Robert C. Martin** express the principle as “A class should have one and only one reason to change, meaning that a class should have only one job.”

* Every Module or class should have responsibility over a single part of the functionality provided by the software, and that responsibility should be entirely encapsulated by the class.

## Open Close Principle

“Objects or entities should be open for extension but closed for modification.”

* This means that a class should be extendable without modifying the class itself.
* The design and writing to the code should be done in a way that new functionality should be added with minimum changes in the existing code.

## Liskov Substitution Principle

Introduced by Barbara Liskov state that “**objects in a program should be replaceable with instances of their subtypes without altering the correctness of that program**”

If S is a subtype of T, then objects of type T may be replaced with objects of type S.

* If a program module is using a Base class, then the reference to the Base class can be replaced with a Derived class without affecting the functionality of the program module.
* We can also state that Derived types must be substitutable for their base types.

## Interface Segregation Principle

“Many Client-specific interfaces are better than one general purpose interface”

* We should not enforce clients to implement interfaces that they don’t use. Instead of creating one big interface we can break down it to smaller interfaces.

## Dependency Inversion Principle:

“Entities must depend on abstractions, not on concretions. It states that the high-level module must not depend on the low-level module, but they should depend on abstractions.”

## Solution to develop a successful application depends on

1. **Architecture**: Choosing an architecture is the first step in designing application based on the requirements. Eg. MVC, WEBAPI, eta
2. **Design Principles**: Application development process need to follow the design principles
3. **Design Patterns**: We need to choose current design patterns to build the software.