SOLID PRINCIPLES

I. Single Responsibility Principle

- Each software module should have one & only one reason to change.

- Kind of like loose coupling
  - offers a modular way to chose which details are involved in a particular operation.

- Will, responsibilities is ~~directly~~ inversely proportional to Testability

Logging | Persistence | Validation

- Keep classes, small, focused, & testable.

II. Open / Closed Principle

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

- Typical approaches to extension
  - Parameter-Based extension ✓
  - Inheritance-based extension ✓
  - Composition/Injection extension ✓

- Eg:- switch case statements

↓

Factory

↓

Return new classes.

III. Liskov Substitution Principle

- Subtypes must be substitutable for their base types.
- Tell , Don't ask
  - This can be done by avoiding if checks & assigning those responsibilities to a different class.
- Look for
  - Type checking
  - Null checking
  - Not Implemented Exception.

IV. Interface Segregation Principle

- Clients should not be forced to depend on methods they do not use.
- Avoid large interfaces, b/c more dependencies means i) more coupling
  ii) More difficult testing.

eg.

INotificationService
SendEmail
SendText
×

IEmailNotificationService
ITextNotificationService
✓

public interface INotificationService : IEmail, IText
✓

↑
It also can be extended