SOLID

- Single Responsibility Principle
- Open Closed Principle
- Liskov's Substitution Principle
- Interface segregation Principle
- Dependency Inversion

  (Not be confused with dependency injection)

Single Responsibility Principle

This principle states that every unit of code (method / class / package) should only have a single responsibility.

Suppose, if we want to design a code where we need to save something to the DB. Then, if we write a code like:

```
saveToDB() {

    // Generating a query
    String q = "Update table_name";
    q += "---";
    q += "where ...";

    // Connect to db
    ConnectionDB dbc = new ConnectionDB(username, password);
    dbc.connect()
    ...

    // Execute query & saveToDB
    dbc.save(q);
}
```

following code has multiple responsibilities so it should be avoided by doing such

generateQuery()

CreateDBconnection()

SaveToDB()

} → Delegate responsibilities to others which are not related

* SRP is built on seperation of concerns & stating the boundaries of encapsulation

Violations of SRP

① if-else methods (multiple)

② Monster method

③ Utils/Commons package

↳ Biggest garbage ⇒ java.util package

Open Closed Principle

This principle states that a class should be open for extension but closed for modifications

For example, if we want add any new feature to the class. The class should be able to extend without modifying the running codebase.

Bob Martin says,

You should be able to extend the behaviour of a system without having to modify that system.

Approaches to OCP

1. Passing parameters
2. Using inheritance → might lead to class explosion & if changes are implemented on a parent class blast radius is very high
3. Using abstract classes & interfaces

Attempts should be made to not create unnecessary abstractions as well as not basic concrete classes. The implementation must lie in between.

* Also remember the keyword 'new' is glue

Liskov's substitution principle

This principle states that subtypes must be substitutable for their base types.

> Inheritance → IS-A
> Composition → HAS-A

Violations of LSP

1. Type checking with is or as in polymorphic code
2. Null checks
3. NotImplementedException

* LSP is a subset of polymorphism

Interface Segregation Principle

Clients should not be forced to depend on methods they do not use

↳ Calling code (main)

* ISP is basically SRP for interfaces.

Dependency Inversion Principle

The principle states that high-level modules should not depend on low-level modules.

Both should depend on abstractions.

* Also,

Abstractions should not depend on details.

Details should depend on abstractions.

Abstractions shouldn't be coupled to details

- Abstractions describe what
- Details specify how

* Abstractions introduces loose coupling