**SOLID Principal:** it is the best practices of software development to deliver good quality of app because it’s help us to write clean & elegant/nice of code with SOC(separation of concept).

1. **S => SRP = Single Responsibility principal :** https://www.geeksforgeeks.org/single-responsibility-principle-in-java-with-examples**/**
   1. A class should have only one reason to change. This means when we design our classes, we need to ensure that our class is responsible only for 1 task or functionality and when there is a change in that task/functionality, only then, that class should change.
   2. Achieve through create separate class for each responsibility.
   3. **Benefits of Single Responsibility Principle**
      1. When an application has multiple classes, each of them following this principle, then the applicable becomes more maintainable, easier to understand.
      2. The code quality of the application is better, thereby having fewer defects.
      3. Onboarding new members are easy, and they can start contributing much faster.
      4. Testing and writing test cases is much simpler

**Example:** We have activity and having task, 1. getting some input from user, 2. preparing request data, 3. preparing the request call, 4. Handling response, 5.parsing the response data.

**Note: 1.** So once case we create each & everything in same activity, which is not follow SRP**. 2.** Achieve SRP we have create separate class for each task

1. **O => OCP = Open-Close Principal: https://www.geeksforgeeks.org/open-closed-principle-in-java-with-examples/**
   1. Software entities such as classes, functions, modules should be open for extension but closed for modification.
   2. Create an abstract class that serves as a base class for all types of objects.

**Example:** Program to calculate the volume in which let us consider the task of building an application that calculates the volumes of all the geometric objects.

Note: **1.** Don’t create method or inner class for achieve this.

**2.** Create an abstract class that serves as a base class for all types of objects(rectangle, circular).

1. **L => LSP = Liskov substitution of principal**
   1. The derived class must be usable through the base class interface, without the need for the user to know the difference.Simply put, if class *A* is a subtype of class *B*, then we should be able to replace objects of *B*with objects of *A*(i.e., objects of type A may substitute objects of type B)without changing the behavior (correctness, functionality, etc.) of our program.
   2. “Derived types must be completely substitutable for their base types”
2. **I => ISP = Interface Segregation Principal**
   1. No client should be forced to depend on methods it does not use.
   2. In Interface Segregation Principle interface should not have methods that implementing classes don’t require. Implementing classes are for no reason forced to provide implementations for those methods it does not require to.
3. **D => DIP = Dependancy Inversion principal** 
   1. High-level modules should not depend on low-level modules. Both should depend on abstractions.
   2. Abstractions should not depend on details. Details should depend on abstractions.
   3. gh Any higher classes should always depend upon the abstraction of the class rather than the detail