**Solid Principles in Java**

1. **Single Responsibility Principle** - A class should have only one reason to change

* We should be able to tell what the application does in a single line
* If the class contains many methods, then it is not suitable for Single Responsibility
* If the methods are more, then the reuse will be more

1. **Open- Closed Principle**

Software entites should be open for extension and closed for modification

Given a method where it changes based upon the condition

String input;

Double Area;

Public int Area(){

If (input==Rectangle){

}

Else{

}

Instead of this ,

We can write the methods in abstract class and interface and extend them in the other class and change the behavior in that class.

1. **Liskov Substitution Principle**

Subclasses should be substitutable for their base types.

Best example is Abstract class

* Create Abstract class
* Extend the class and implement the methods
* Now extend the extended class

Eg:

// Parent Class

abstract class Bird {

public abstract void move();

}

// Flying Birds

class FlyingBird extends Bird {

@Override

public void move() {

System.out.println("I can fly!");

}

}

// Non-Flying Birds

class NonFlyingBird extends Bird {

@Override

public void move() {

System.out.println("I walk or run!");

}

}

// Subclasses

class Sparrow extends FlyingBird {

}

class Ostrich extends NonFlyingBird {

}

public class Main {

public static void main(String[] args) {

Bird sparrow = new Sparrow();

sparrow.move(); // Outputs: I can fly!

Bird ostrich = new Ostrich();

ostrich.move(); // Outputs: I walk or run!

}

}

1. **Interface Segregation Principle**

The dependency of one class to another one should depend on the smallest possible interface

The interface should be as small as possible.

Steps

Create interface

Implement it in other class

Later extends that class in other class

In this way, methods can be minimized and code can be reused

1. **Dependency inversion principle**

Instead of depending on the concrete classes, depending on interfaces and abstract classes

Because as the type increases, you should go on writing the if cases in the methods