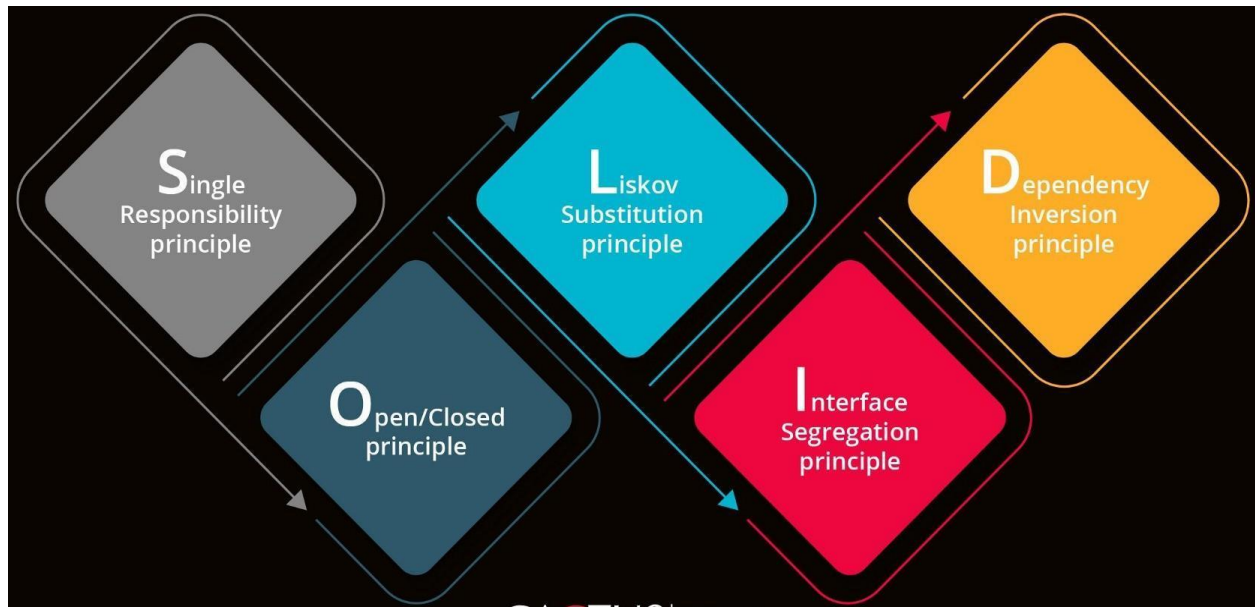


The Understanding Of The SOLID Principles



Introduction

SOLID principles come into picture because these are the design principles that help us in encouraging creating more maintainable, understandable, and flexible software.

As our applications grow in size, we can reduce their complexity by using **SOLID** principles.

SOLID Principles:-

The following five concepts make up our SOLID principles:

1. **S**ingle Responsibility
2. **O**pen/Closed
3. **L**iskov Substitution
4. **I**nterface Segregation
5. **D**ependency Inversion

These five software development principles are guidelines to follow when building software so that it is easier to scale and maintain. They were made popular by a software engineer, Robert C. Martin.

BENEFITS OF SOLID PRINCIPLES-

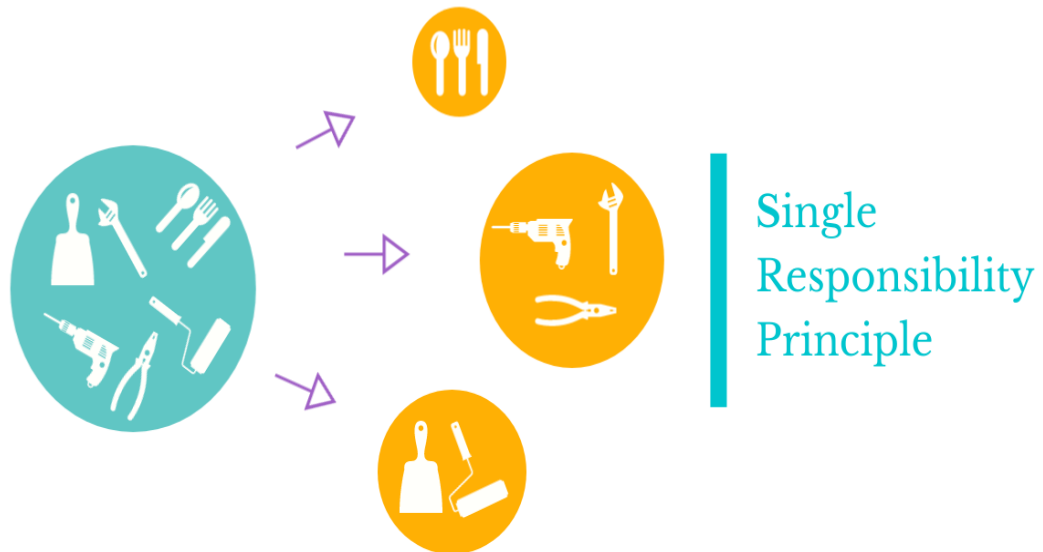
Some of the benefits SOLID Principle holds are as follows:-

- ❖ Loose Coupling
- ❖ Code Maintainability
- ❖ Dependency Management

The SOLID Principles:

1. S — Single Responsibility:

Define:- A class should have a single responsibility.



- If a Class has many responsibilities, it increases the possibility of bugs because making changes to one of its responsibilities could affect the other ones without you knowing.

Code Snippet:-

```
1 package SingleResponsibility;  
2  
3 import java.util.*;  
4 public class Book {  
5     int id;  
6     String name;  
7     String author;  
8     public Book(){  
9         id=123;  
10        name="Harry Potter";  
11        author="J.K. Rowling";  
12    }  
13 }  
14  
15
```

```

1  package SingleResponsibility;
2  ▶ public class Book_issued {
3      String date= "09/08/2021";
4      void issuedBook(String name,String n){
5
6          System.out.println("The book "+name+" issued to "+n+" on "+date);
7      }
8  ▶ public static void main(String[] args) {
9      Book B1 =new Book();
10     User u1 = new User();
11     Book_issued i1 =new Book_issued();
12     i1.issuedBook(B1.name,u1.name);
13 }
14 }
15
16
17

```

Ⓢ User.java ×

```

1  package SingleResponsibility;
2      ⚡
3  public class User {
4      int userId;
5      String name;
6      String address;
7      int contactNumber;
8      User(){
9          userId=1234;
10         name="Rakhi Pareek";
11         address="Sikar";
12         contactNumber= 123456789;
13     }
14 }
15

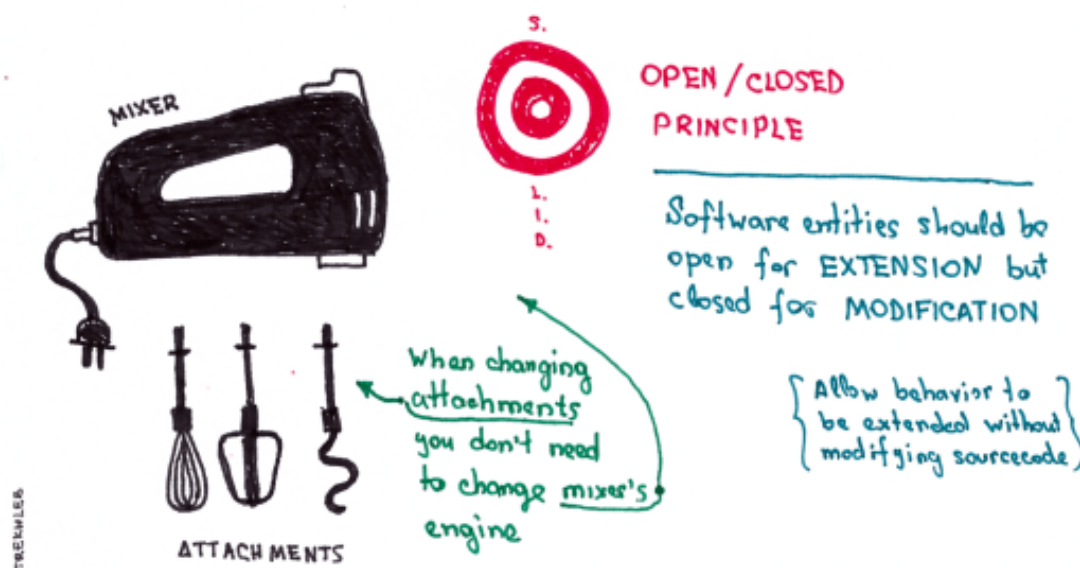
```

Goal:

This principle aims to separate behaviours so that if bugs arise as a result of your change, it won't affect other unrelated behaviours.

2. O — Open-Closed:

Define:- Classes should be open for extension, but closed for modification



- Changing the current behaviour of a Class will affect all the systems using that Class.
- If you want the Class to perform more functions, the ideal approach is to add to the functions that already exist NOT change them.

Code Snippet:-

```
1 package OpenClose;
2
3 public class Book {
4     int id;
5     String name;
6     String author;
7     public Book(){
8         id=123;
9         name="Harry Potter";
10        author="J.K. Rowling";
11    }
12
13 }
14
15
```

```
1 package OpenClose;
2
3 public class Book_issued {
4     String date= "09/08/2021";
5     void issuedBook(String name,String userName,String c){
6
7         System.out.println("The book "+name+" issued to "+userName+" on "+date);
8         System.out.println("The category of "+name+" is "+c);
9     }
10
11     public static void main(String[] args) {
12         booksOpenClosed B1 =new booksOpenClosed();
13         User u1 = new User();
14         Book_issued i1 =new Book_issued();
15         i1.issuedBook(B1.name,u1.name,B1.category);
16     }
17 }
18
```

```
1 package OpenClose;
2
3 public class booksOpenClosed extends Book {
4     String category;
5
6     booksOpenClosed(){
7         id=124;
8         name = "Harry Potter";
9         author="J.K. Rowling";
10        category="Fiction";
11
12    }
13 }
14
```

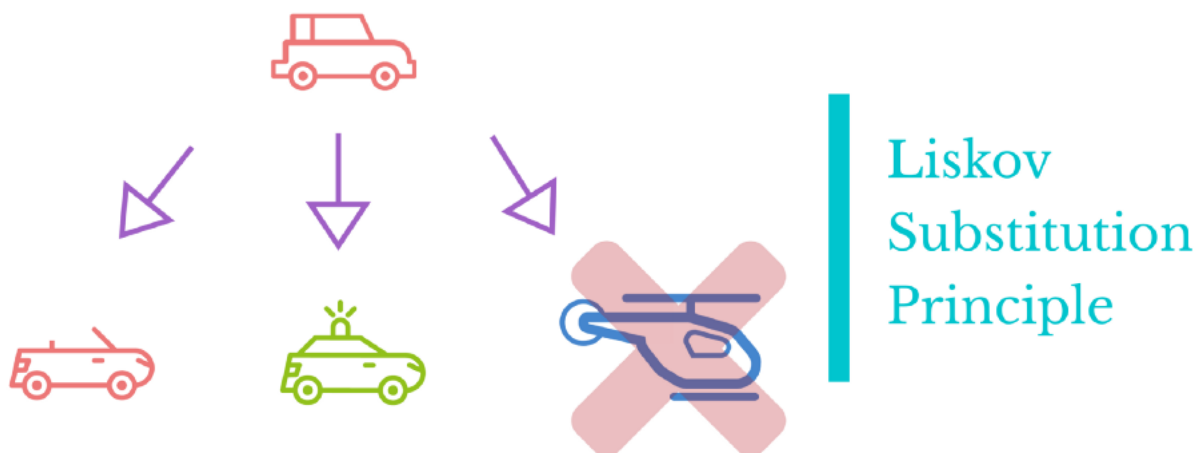
```
User.java x
1 package OpenClose;
2
3 public class User {
4     int userId;
5     String name;
6     String address;
7     int contactNumber;
8     User(){
9         userId=1234;
10        name="Rakhi Pareek";
11        address="Sikar";
12        contactNumber= 123456789;
13    }
14 }
15
```

Goal:

This principle aims to extend a Class's behaviour without changing the existing behaviour of that Class. This is to avoid causing bugs wherever the Class is being used.

3. L — Liskov Substitution:

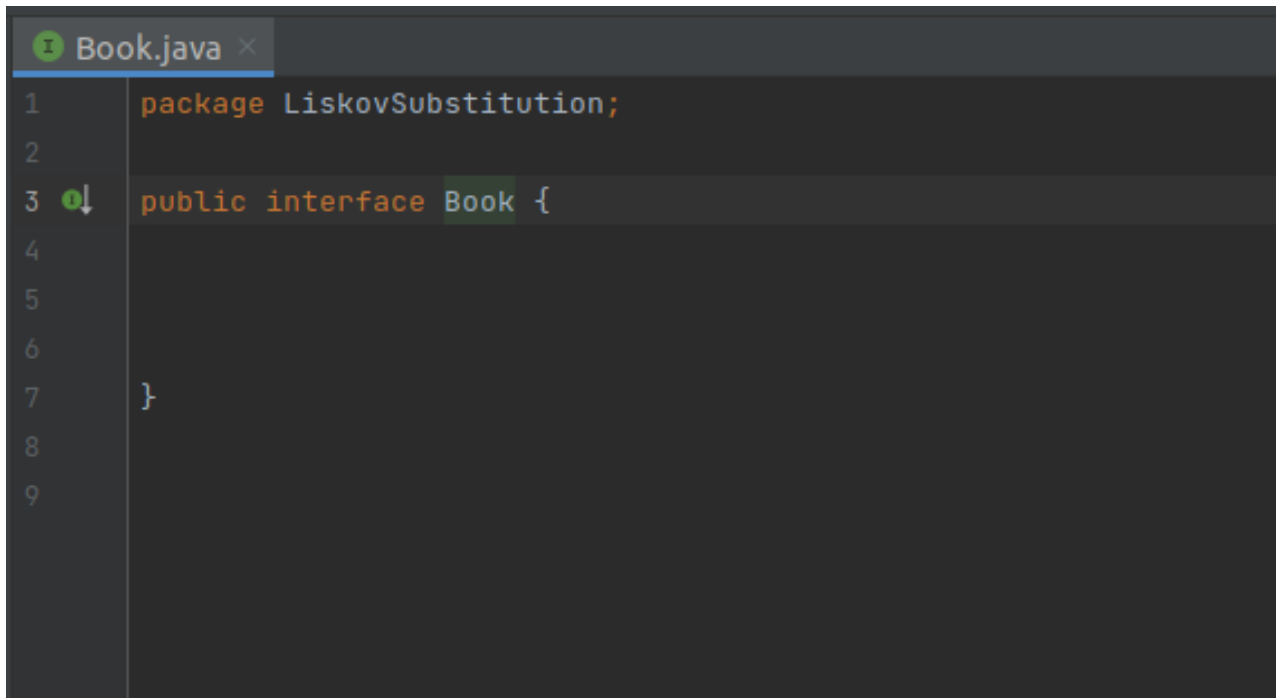
Define:- If S is a subtype of T, then objects of type T in a program may be replaced with objects of type S without altering any of the desirable properties of that program.



- When a child Class cannot perform the same actions as its parent Class, this can cause bugs.
- If you have a Class and create another Class from it, it becomes a parent and the new Class becomes a child. The child Class should be able to do everything the parent Class can do. This process is called Inheritance.
- The child Class should be able to process the same requests and deliver the same result as the parent Class or it could deliver a result that is of the same type.

- The picture shows that the parent Class delivers Coffee(it could be any type of coffee). It is acceptable for the child Class to deliver Cappuccino because it is a specific type of Coffee, but it is NOT acceptable to deliver Water.
- If the child Class doesn't meet these requirements, it means the child Class is changed completely and violates this principle.

Code Snippet:-



```
Book.java x
1 package LiskovSubstitution;
2
3 public interface Book {
4
5
6
7 }
8
9
```

```
1 package LiskovSubstitution;
2
3 public class LSP {
4     public static void main(String[] args) {
5         nonFictionalBook f1 = new novelBook();
6         f1.func1();
7     }
8 }
9
10 |
```

```
1 package LiskovSubstitution;
2
3 public class nonFictionalBook implements Book{
4
5     void func1(){
6         nonFictionalBook d = new nonFictionalBook();
7         System.out.println("The Beauty Myth is a nonFictional book");
8     }
9 }
10
```

```
1 package LiskovSubstitution;
2
3 public class novelBook extends nonFictionalBook{
4     void func1() {
5         novelBook n = new novelBook();
6         System.out.println(" Invisible Man is a novel book and belong to nonfictional book");
7     }
8 }
9
```

Goal:-

This principle aims to enforce consistency so that the parent Class or its child Class can be used in the same way without any errors.

Summary:-

So far, we have discussed these three principles and highlighted their goals. They are to help you make your code easy to adjust, extend and test with little to no problems.