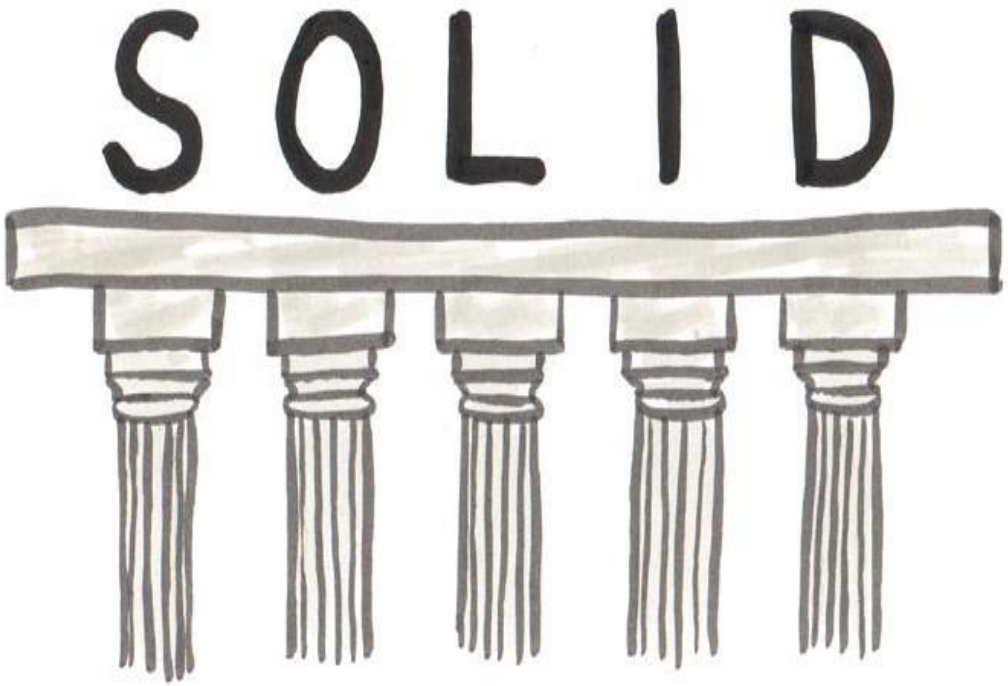
Code and Blog to showcase the understanding of the SOLID principles



INTRODUCTION-

- The SOLID principles are basically the Five Principles used in the Object –Oriented Class Design and these are also the set of rules and best practices to follow while designing an any class structure.

- SOLID Principles were first introduced by Computer Scientist Robert J. Martin in 2000, but the acronym was introduced later by Michael Feathers.

- These SOLID principles were developed in paper named "Design Principles and Design Patterns".

WHY SOLID PRINCIPLES-

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

- It also helps us in growing our application according to the size, and can reduce the complexity of the code.

SOLID PRINCIPLES-

- This principle is an acronym of the five principles of Object Oriented Designs. These are as –

  - Single Responsibility Principle (SRP)
  - Open/Closed Principle
  - Liskov's Substitution Principle (LSP)
  - Interface Segregation Principle (ISP)
  - Dependency Inversion Principle (DIP)

- These are helps us in reducing Tight Coupling. Tight Coupling simply means a group of classes that are highly dependent on one another and SOLID Principles helps you to avoid that in your code.

- Loosely Coupled Classes minimize changes in your code, helps in making code more reusable, maintainable, flexible and stable.

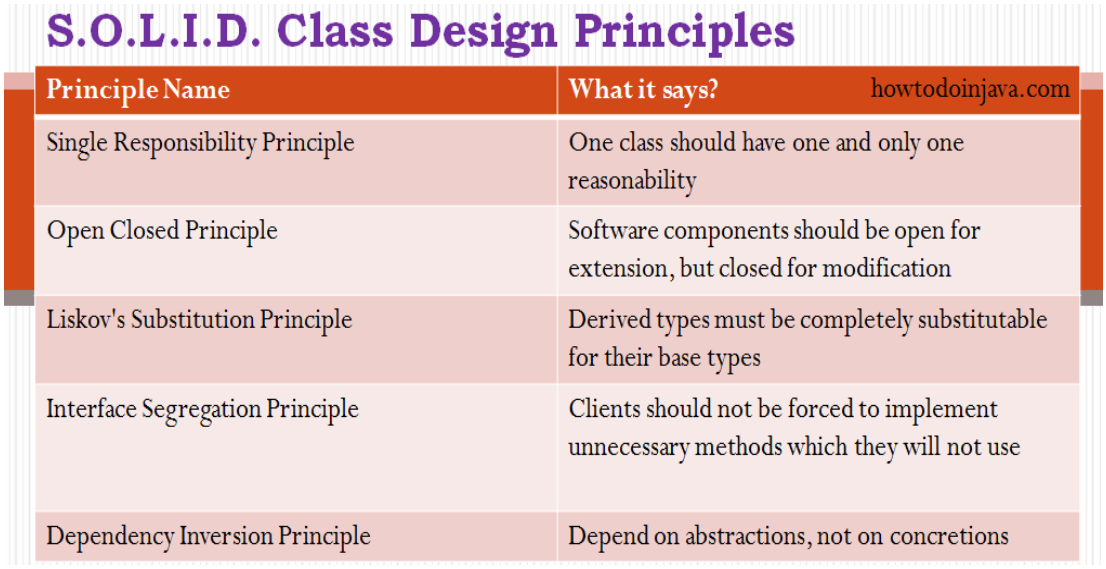
BENEFITS OF SOLID PRINCIPLES-

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

- Loose Coupling
- Code Maintainability
- Dependency Management

EXPLANATION-

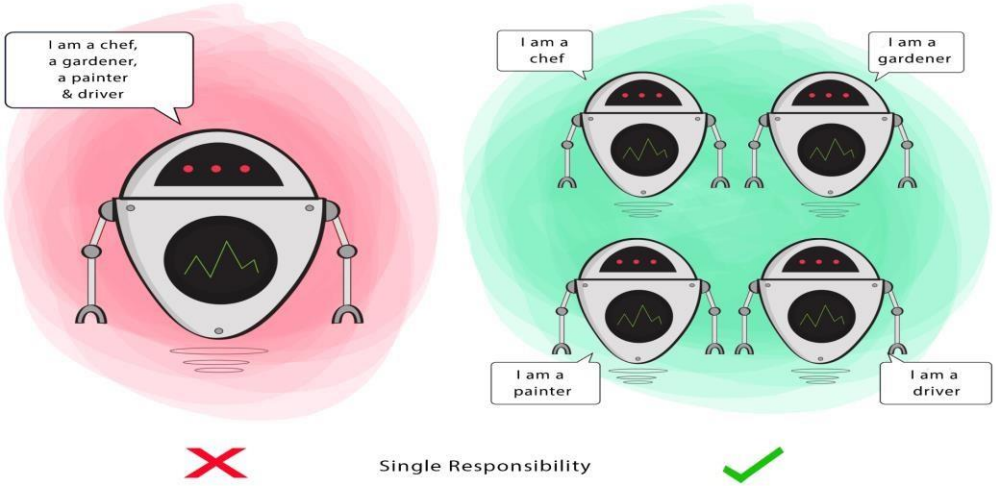
Let's see the SOLID Principles in brief:-



S.O.L.I.D. Class Design Principles

Principle Name	What it says? howtodoinjava.com
Single Responsibility Principle	One class should have one and only one reasonability
Open Closed Principle	Software components should be open for extension, but closed for modification
Liskov's Substitution Principle	Derived types must be completely substitutable for their base types
Interface Segregation Principle	Clients should not be forced to implement unnecessary methods which they will not use
Dependency Inversion Principle	Depend on abstractions, not on concretions

- **Single Responsibility Principle (SRP) -** This principle states that "a class should do one thing and therefore it should have only single reason to change". It means that every class should have a single responsibility or single job or a single purpose.



Benefits –

- The code quality of the application is much better.
- Testing and Writing test cases is much simpler.
- New members can be on boarded easily.

Example-

This example shows the 1st principle that is Single Responsibility Principle (SRP), here we created a main class (Book.java) and then created two more classes which inherit the main class and here it explains every class should have a single responsibility or single job or a single purpose.

- **Book.java**

```
package SingleResponsibility;

import java.util.*;
public class Book {
int id;
```

```
        String name;
        String author;

    public Book(){
        id=123;
        name="Bhagavad Gita";
        author="Vyasa";
    }
}
```

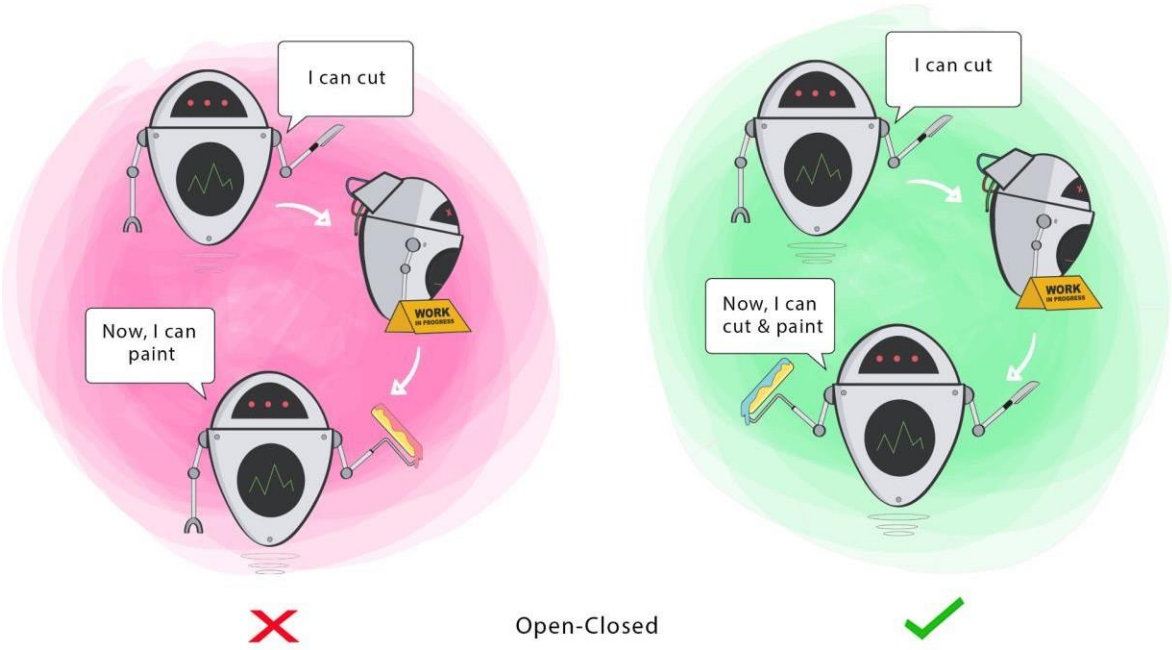
- **BookIssued.java**

```
package SingleResponsibility;
public class BookIssued {

    String date= "01/09/2021";
    void issuedBook(String name,String n){
System.out.println("The book "+name+" issued to "+n+" on
"+date);
    }
    public static void main(String[] args) {
        Book B1 =new Book();
        User u1 = new User();
        BookIssued i1 =new BookIssued();
        i1.issuedBook(B1.name,u1.name);

        }
    }
```

- **User.java**

```
package SingleResponsibility;
public class User {
    int userId;
    String name;
    String address;
    int contactNumber;
    User(){
        userId=1234;
        name="Deeksha Tripathi";
        address="Kanpur";
        contactNumber= 9823654321;
    }
}
```

- **Open/Closed Principle-** This principle states that the “software entities like (classes, modules, functions and many more) should be open for extension, but closed for modification” which means that one should be able to extend a class behavior, with modifying it.



Benefits-

- Helps to write flexible, scalable and reusable code.
- It also minimizes the changes happened in the code.

Example-

Here again in this example we created a main class and then two child classes and to prove the 2nd principle OpenClose we created a 3rd child class which is ready to change the behavior but it cannot be modified into another functionality

- **Book.java**

```
package OpenClose;
public class Book {
    int id;
    String name;
    String author;
    public Book(){
        id=1619;
        name="Shri Guru Granth Sahib";
        author="Guru Nanak";
    }
}
```

- **BookIssued.java**

```
package OpenClose;
public class BookIssued {
    String date = "10/10/2021";
    void issuedBook(String name, String userName, String c)
    {System.out.println("The book " + name + " issued to " +
        userName + " on " + date);
```

```
        System.out.println("The category of " + name + " is " + c);
    }

    public static void main(String[] args) {
        booksOpenClosed B1 = new booksOpenClosed();
        User u1 = new User();
        BookIssued i1 = new BookIssued();
        i1.issuedBook(B1.name, u1.name, B1.category);
    }
}
```

- **booksOpenClosed-**

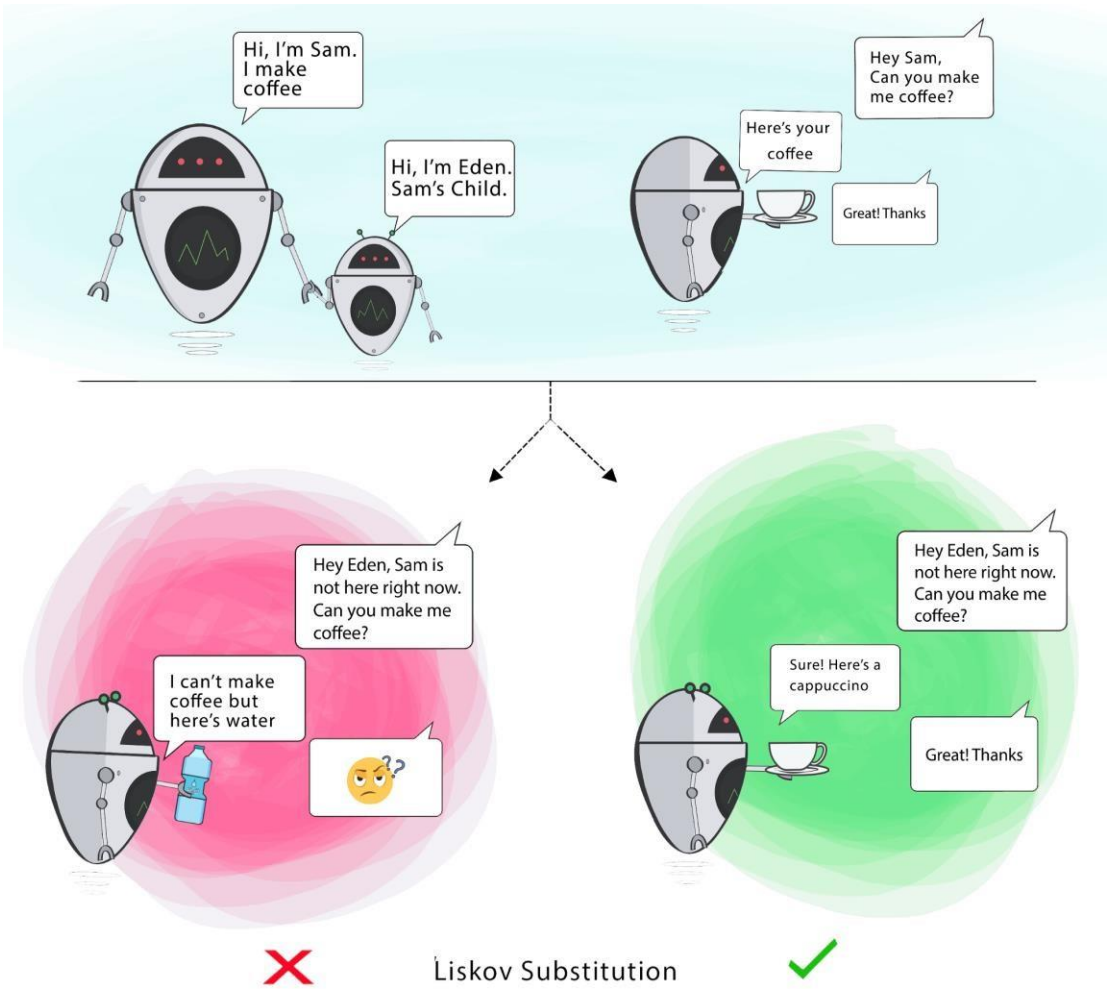
```
package OpenClose;
public class booksOpenClosed extends Book {
    String category;
 booksOpenClosed(){
        id=1619;
        name = "Bible";
        author="Jesus Christ";
        category="Religious";
   }
```

- **User.java**

```
package OpenClose;
public class User {
    int userId;
```

```
    String name;
    String address;
    int contactNumber;
    User(){
        userId=1619;
        name="Deeksha Tripathi";
        address="Kanpur";
        contactNumber= 982365432;
    }
}
```

- **Liskov's Substitution Principle (LSP)-** This principle states that "Derived or child classes must be substitutable for their base or parent classes". It was introduced by Barbara Liskov in 1987 and it also ensures that any class that is the child of a parent class should be usable in place of its parent without any unexpected behaviour.



Benefits-

- It makes code re-usable.
- Reduced Coupling.
- It also helps us in easier maintenance.

Example-

Here we created many child classes in order to show that any class that is the child of a parent class should be usable in place of its parent without any unexpected behavior.

- **Book.java**

```
package LiskovSubstitution; public
interface Book {
}
```

- **dramaBook.java**

```
package LiskovSubstitution;

public class dramaBook extends fictionalBook{
    int id=128;
    String name="Romeo and Juliet";
    String author="William Shakespeare";

    public static void main(String[] args) {
        dramaBook d = new dramaBook();
        System.out.println(d.name+" is a drama book and belong to
fictional book");
    }
}
```

- **fictionalBook.java**

```
package LiskovSubstitution;

public class fictionalBook implements Book{
    int id=125;
    String name="Avengers";
    String author="Stan lee";
    public static void main(String[] args) {
        nonFictionalBook d = new nonFictionalBook();
        System.out.println("Fictional is a category of book");
    }
```

- **fiananceBook.java**

```
package LiskovSubstitution;
public class financeBook extends nonFictionalBook {
    int id=120;
    String name="Rich Dad Poor Dad";
    String author="Robert T. Kiyosaki";

    public static void main(String[] args) {
```

```
        financeBook f = new financeBook();
        System.out.println(f.name+" is a finance book and belong to
nonfictional book");
    }
}
```

- **nonfictionamBook.java**

```
package LiskovSubstitution;

public class nonFictionalBook implements Book{
    int id=126;
    String name="The Beauty Myth";
    String author="Naomi Wolf";

    public static void main(String[] args) {
        nonFictionalBook d = new nonFictionalBook();
        System.out.println("nonFictional is a category of book");
    }
}
```

- **novel.java**

```
package LiskovSubstitution;

public class novelBook extends nonFictionalBook{
    int id=129;
    String name="Invisible Man";
    String author="Ralph Ellison";

    public static void main(String[] args) {
        novelBook n = new novelBook();
        System.out.println(n.name+" is a novel book and belong to
nonfictional book");
    }
}
```

- **superHeroesBook.java**

```
package LiskovSubstitution;

public class superHeroesBook extends fictionalBook{
    int id=127;
    String name="Captain America";
    String author="Stan lee";

    public static void main(String[] args) {
        superHeroesBook s = new superHeroesBook();
        System.out.println(s.name+" is a super heroes book and
belong to fictional book");
    }
}
```

SUMMARY-

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