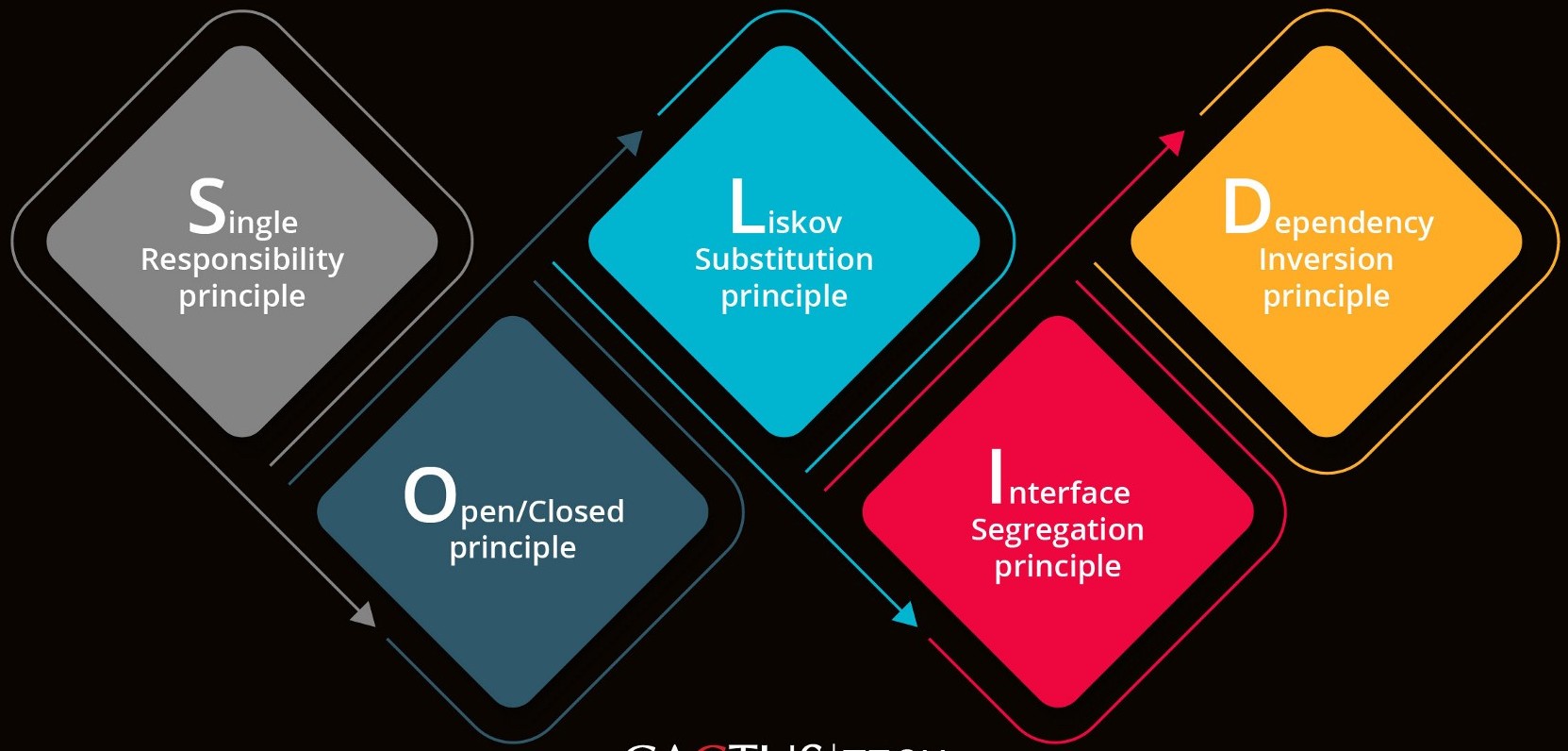
**The Understanding Of The SOLID Principles**



**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](https://en.wikipedia.org/wiki/Robert_C._Martin).

**BENEFITS OF SOLID PRINCIPLES-**

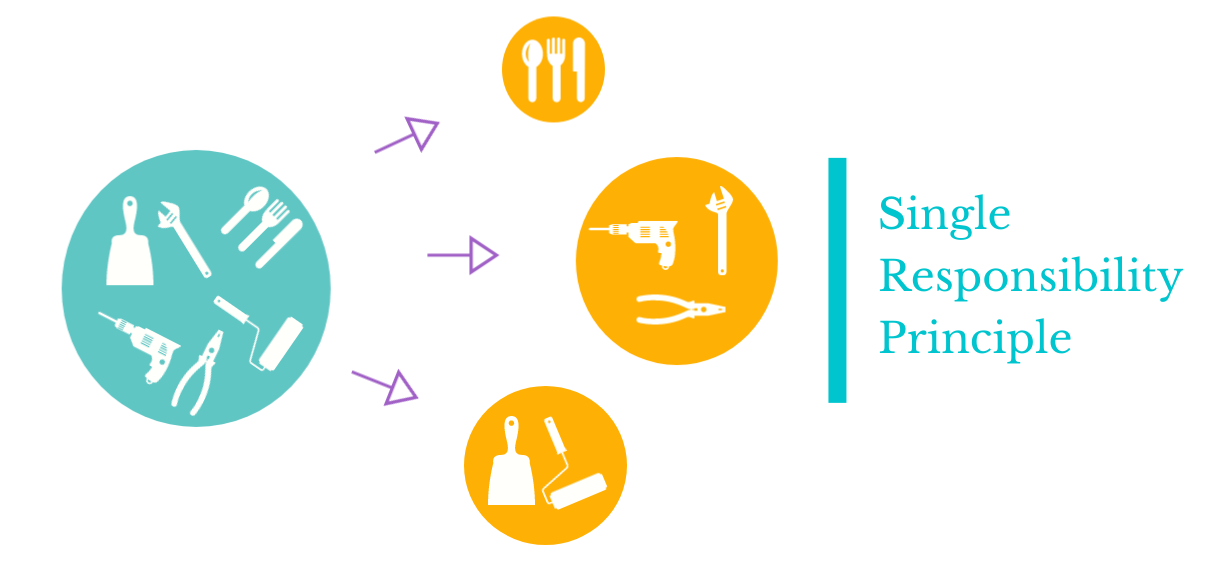
Some of the benefits SOLID Principle holds are as follows:-

* Loose Coupling
* Code Maintainability
* Dependency Management

# **The SOLID Principles:-**

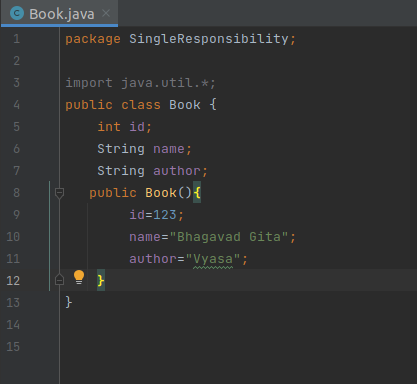
## **S — Single Responsibility:**

Define:- A class should have a single responsibility.

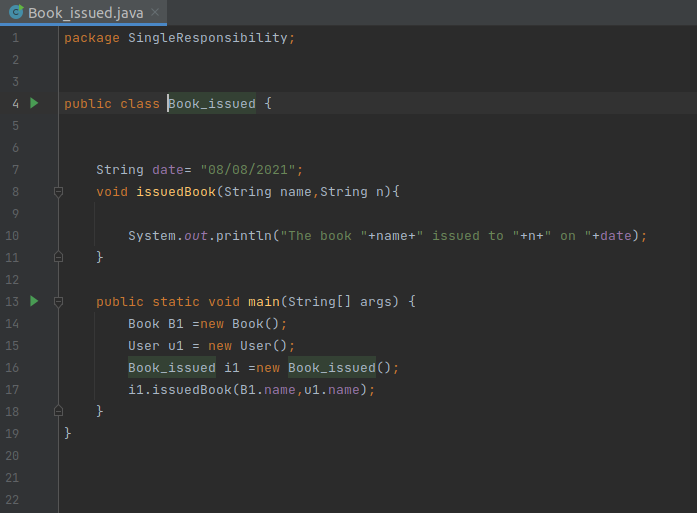


* If a Class has many responsibilities, it increases the possibility of bugs because making changes to one of its responsibilities could affect the other ones without you knowing.

Code Snippet:-



****

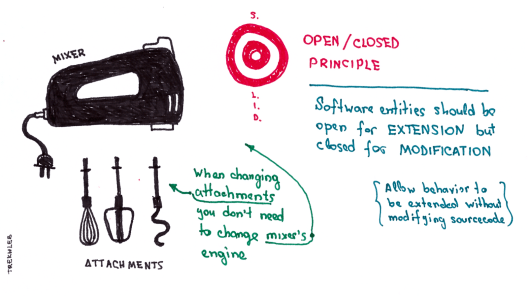
****

**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.**

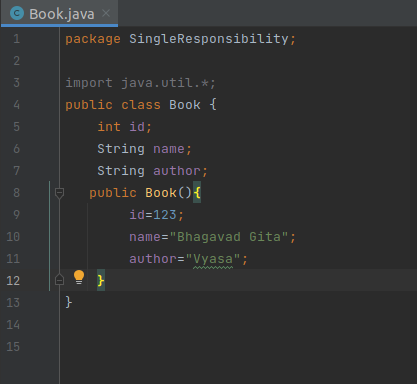
**2. O — Open-Closed:**

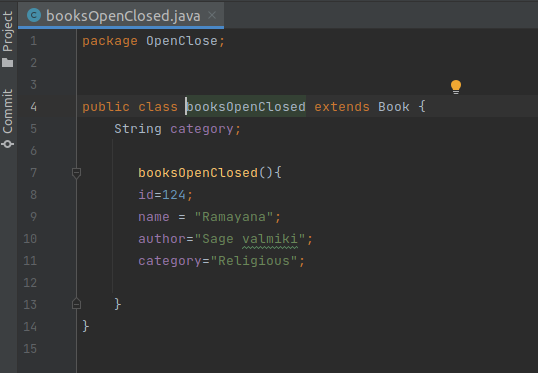
**Define:- Classes should be open for extension, but closed for modification**



* 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:-



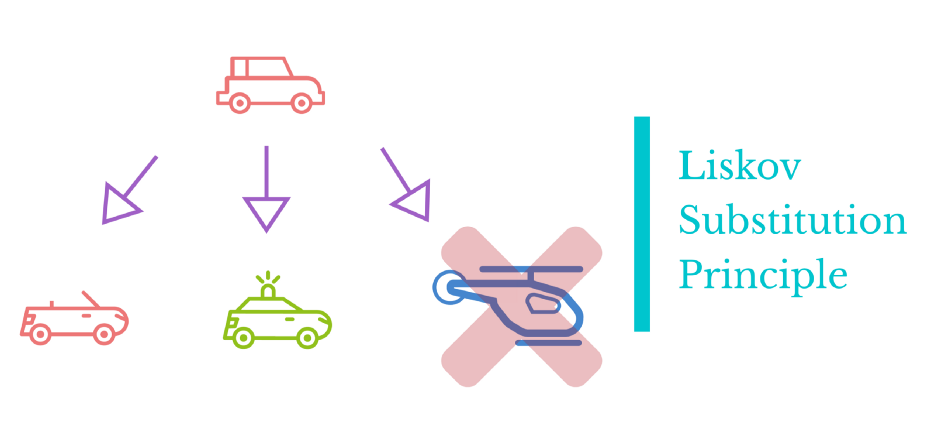


**Goal:-**

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.

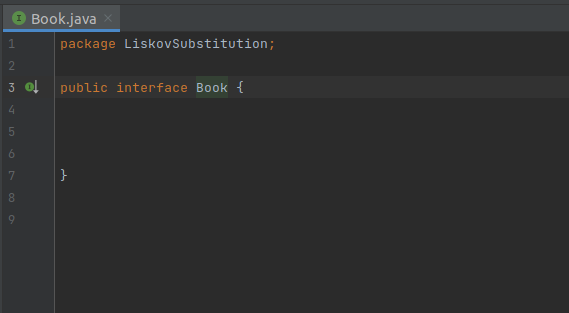
**3. L** — **Liskov Substitution:**

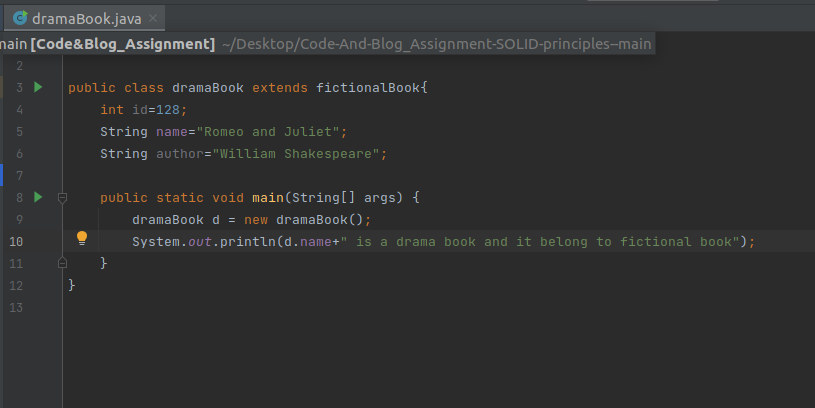
**Define:- If S is a subtype of T, then objects of type T in a program may be replaced with objects of type S without altering any of the desirable properties of that program.**

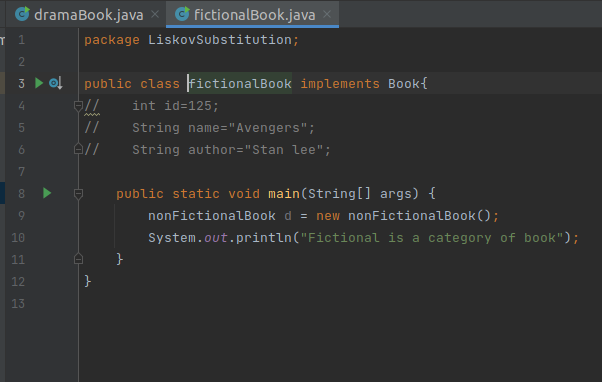
****

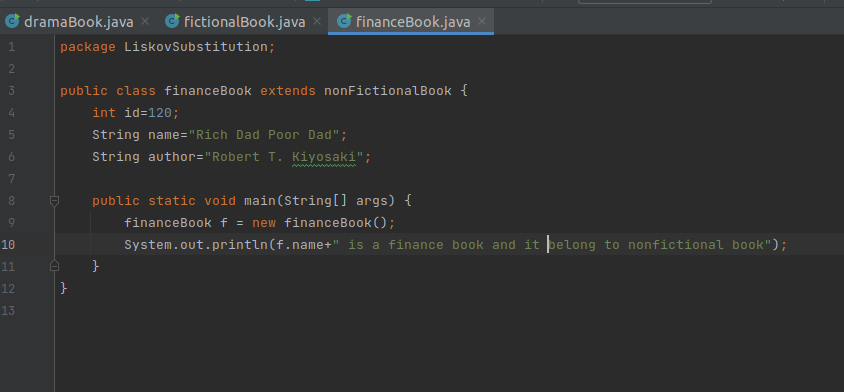
* When a child Class cannot perform the same actions as its parent Class, this can cause bugs.
* If you have a Class and create another Class from it, it becomes a parent and the new Class becomes a child. The child Class should be able to do everything the parent Class can do. This process is called 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.

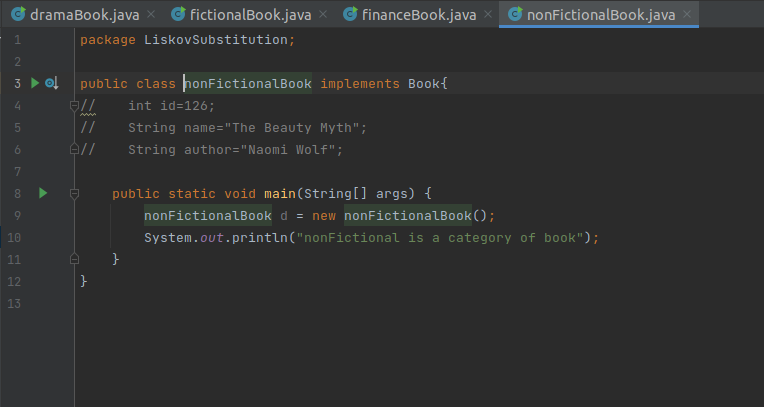
Code Snippet:-

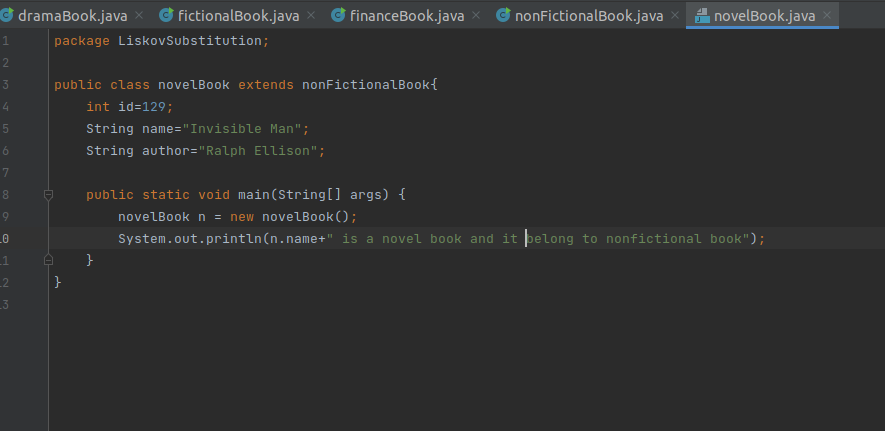


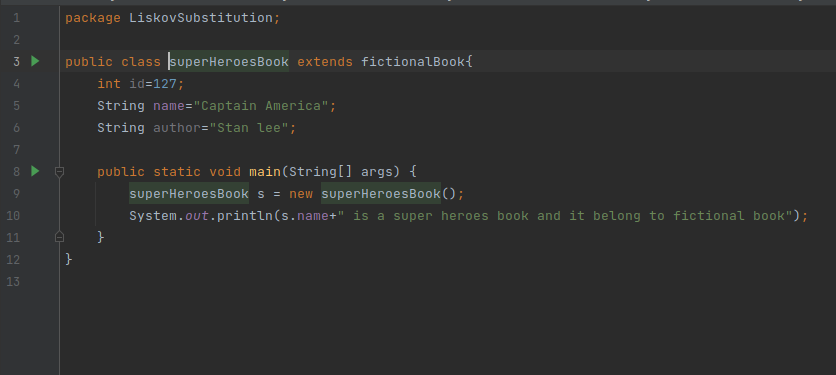


￼￼￼￼









**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.