

SOLID Principles

**SOLID** is a [mnemonic](https://en.wikipedia.org/wiki/Mnemonic) [acronym](https://en.wikipedia.org/wiki/Acronym) for five design principles intended to make software designs more understandable, flexible, and [maintainable](https://en.wikipedia.org/wiki/Software_maintenance).

SOLID principles are a set of recommendations to design and write better software systems.

They were proposed by Robert C. Martin.

Their purposes are:

* Write more maintainable code, which it's easy to change.
* Write code which can be extended with new functionalities easily.
* Write more readable code.

Adopting these practices can also contribute to avoiding code smells, refactoring code, and Agile or

Adaptive software development.

**SOLID is conformed by 5 principles:**

* Single Responsibility Principle (SRP)
* Open Closed Principle (OCP)
* Liskov Substitution Principle (LSP)
* Interface Segregation Principle (ISP)
* Dependency Inversion Principle (DIP)

**In this article, we will be discussing these 3 principles.**

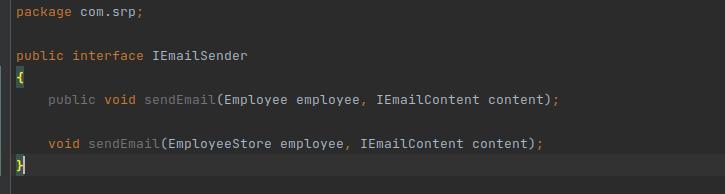
* Single Responsibility Principle (SRP)
* Open Closed Principle (OCP)
* Liskov Substitution Principle (LSP)

**=>Single Responsibility Principle (SRP) -**

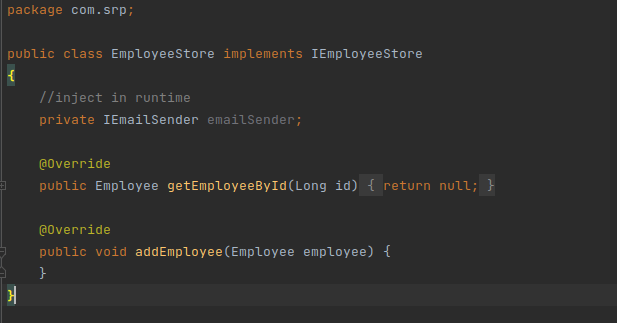
**Single-responsibility Principle (SRP) states:** A class should have one and only one reason to change, meaning that a class should have only one job. This principle stands that each class should have a unique responsibility. Each class manages a specific part of the system and performs very well on one concrete task.

As Robert said: "Gather together the things that change for the same reasons. Separate those things that change for different reasons."

Example:- let’s consider a MailSender class that sends the name of the argument which is being added to the class.



EmployeeStore class collect the data from interface IEmployeeStore as argument id

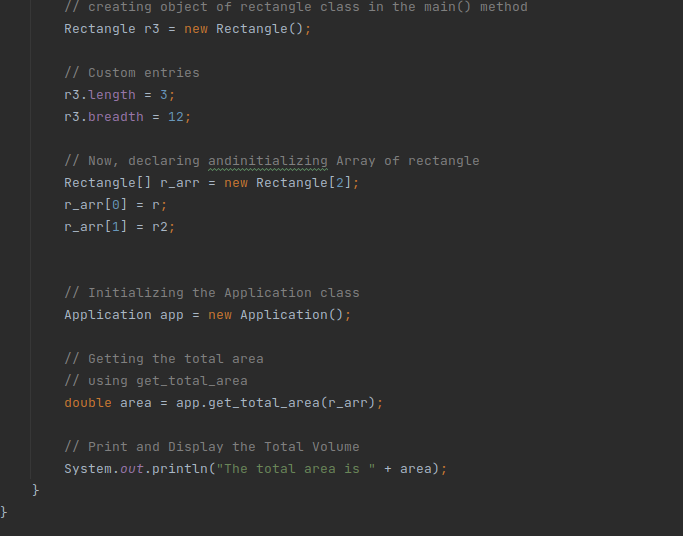


## =>Open-Closed Principle

It states that Objects or entities should be open for extension but closed for modification.

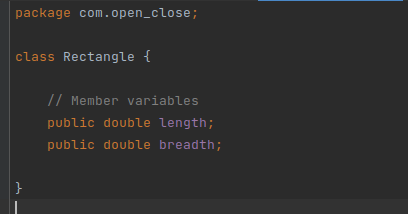
What this means is that if we want to add a new functionality, then we should write new code instead of modifying new one. Preventing bugs is much easier in this way.

Eg -:



In open Close package, we have defined method in which we initialize the rectangle

and declare the dimension of it.



And there is a class application which returns the total volume of the geometric objects.

# 

# 

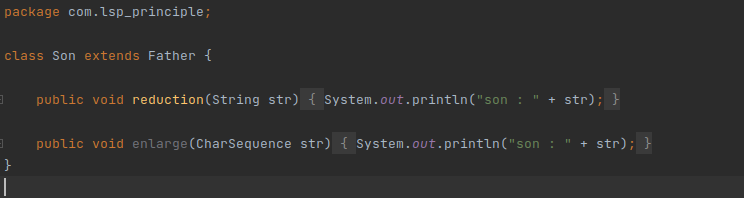
## 

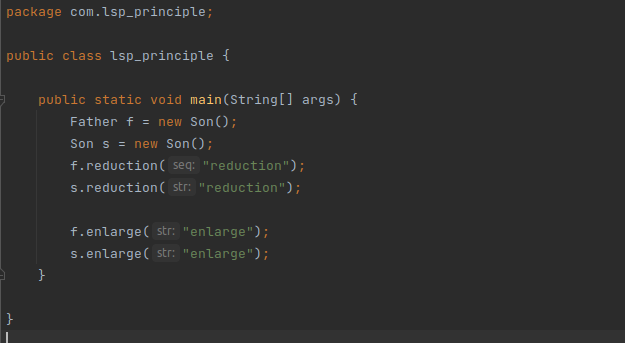
## =>Liskov Substitution Principle

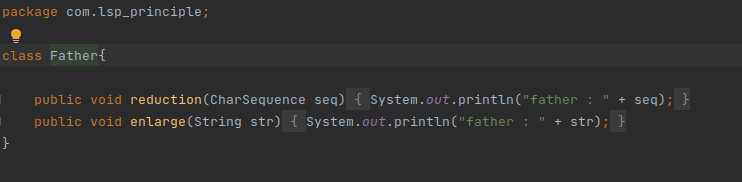
## Liskov Substitution Principle states: Let q(x) be a property provable about objects of x of type T. Then q(y) should be provable for objects y of type S where S is a subtype of T.

The Liskov Substitution Principle in practical software development. The principle defines that **objects of a superclass shall be replaceable with objects of its subclasses without breaking the application**.

Lets take an example of LSP:-







## 