# [Open Closed Design Principle in Java](http://javarevisited.blogspot.in/2011/11/great-example-of-open-closed-design.html" \o "Great Example of Open Closed Design Principle in Java)

* **S** – Single-responsiblity principle
* **O** – Open-closed principle
* **L** – Liskov substitution principle
* **I** – Interface segregation principle
* **D** – Dependency Inversion Principle
* Single-responsibility Principl

S.R.P for short – this principle states that:

A class should have one and only one reason to change, meaning that a class should have only one job.

One word about **Open Closed principle** is a design principle which says that a class should be open for extension but closed for modification. Open Closed Principles is one of the principle from SOLID design principle where it represent "O". Read more about OOPS and SOLID design pattern in my post [10 OOPS and SOLID design principles Java programmer should know](http://javarevisited.blogspot.sg/2012/03/10-object-oriented-design-principles.html).    
  
  
In Simple language Open closed design principles says that new functionality should be added by introducing new classes, methods or fields instead of modifying already tried and tested code. One of the way to achieve this is Inheritance where class is extended to introduce new functionality on top of inherited basic features.

Benefit or Open Closed Design Principle:

1) Application will be more robust because we are not changing already tested class.

2) Flexible because we can easily accommodate new requirements.

3) Easy to test and less error prone.

**How to make code extensible**

Basic principle of making your [code](http://javarevisited.blogspot.sg/2011/09/code-review-checklist-best-practice.html) extensible and following open closed principle is providing object to [class](http://javarevisited.blogspot.sg/2011/10/class-in-java-programming-general.html) at run time and making use of polymorphism to invoke extended functionality.

If functionality is hard Coded than it wouldn’t be extensible but if you write interface and provide implementation of that interface at run time you make it extensible.

**Design Pattern Basics:**

Strategy has found its place in JDK, and you know what I mean if you have sorted ArrayList in Java. Yes, combination of Comparator, Comparable andCollections.sort() method are one of the best real world example of Strategy design pattern. To understand it more, let's first find out *what is Strategy pattern?*First clue is in the name itself.  The strategy pattern defines a family of related algorithms e.g. sorting algorithms like [bubble sort](http://javarevisited.blogspot.sg/2014/08/bubble-sort-algorithm-in-java-with.html),[quicksort](http://java67.blogspot.sg/2014/07/quicksort-algorithm-in-java-in-place-example.html), [insertion sort](http://java67.blogspot.sg/2014/09/insertion-sort-in-java-with-example.html)  and merge sort, or compression algorithm e.g. zip, gzip, tar, jar, encryption algorithm e.g. MD 5, AES etc and lets the algorithm vary independently from clients that use it. For example, you can use Strategy pattern to implement a method which sort numbers and allows client to choose any sorting algorithm at run time, without modifying client's code. So essentially Strategy pattern provides flexibility, extensible and choice. You should consider using this pattern when you need to select an algorithm at runtime. In Java, strategy is usually implemented by creating a hierarchy of classes that extend from a base interface known as Strategy.

This pattern has two main component, Strategy interface and Context class. Strategy interface declares the type of algorithm, it can be [abstract class or interface](http://java67.blogspot.sg/2012/09/what-is-difference-between-interface-abstract-class-java.html).

**Real World Examples of Strategy Design Pattern**

JDK has couple of examples of this pattern, first is Collection.sort(List, Comparator) method, where Comparator is Strategy and Collections.sort() is Context. Because of this pattern your sort method can sort any object, the object which doesn't exists when this method was written. As long as, Object will implement Comparator interface (Strategy interface), Collections.sort() method will sort it.

## Things to Remember about Strategy Pattern in Java

Now let's revise what you have learn in this tutorial about strategy design pattern  :  
  
1) This pattern defines a set of related algorithm and encapsulate them in separated classes, and allows client to choose any algorithm at run time.  
  
2) It allows to add new algorithm without modifying existing algorithms or context class, which uses algorithm or strategies.  
  
3) Strategy is a behavioral pattern in GOF list.  
  
4) Strategy pattern is based upon Open Closed design principle of SOLID.  
  
5) Collections.sort() and Comparator interface is real world example of Strategy pattern.

## **Observer design Pattern**

**Advantage of Observer Design Pattern in Java:**

Main advantage is **loose coupling** between objects called observer and observable. The subject only know the list of observers it don’t care about how they have their implementation.All the observers are notified by subject in a single event call as **Broadcast communication**

**Disadvantage of Observer Design Pattern in Java:**

          The disadvantage is that the sometime if any problem comes, [debugging](http://javarevisited.blogspot.com/2011/07/java-debugging-tutorial-example-tips.html) becomes very difficult because flow of control is implicitly between **observers** and **observable** we can predict that now observer is going to fire and if there is chain between observers then debugging become more complex.

          Another issue is Memory management because subject will hold all the reference of all the observers if we not unregister the object it can create the memory issue.

Read more: <http://javarevisited.blogspot.com/2011/12/observer-design-pattern-java-example.html#ixzz3ZfAC6rUO>