**Design Patterns in Java**

[**https://sourcemaking.com/design\_patterns**](https://sourcemaking.com/design_patterns)

A design patterns are **well-proved solution** for solving the specific problem/task.

### When should we use the design patterns?

We must use the design patterns **during the analysis and requirement phase of SDLC**(Software Development Life Cycle).

|  |  |  |
| --- | --- | --- |
| 1.Creational Design Pattern | 2. Structural Design Pattern | 3. Behavioral Design Pattern |
| 1. Factory Pattern 2. Abstract Factory Pattern 3. Singleton Pattern 4. Prototype Pattern 5. Builder Pattern | 1. Adapter Pattern 2. Bridge Pattern 3. Composite Pattern 4. Decorator Pattern 5. Facade Pattern 6. Flyweight Pattern 7. Proxy Pattern | 1. Chain Of Responsibility Pattern 2. Command Pattern 3. Interpreter Pattern 4. Iterator Pattern 5. Mediator Pattern 6. Memento Pattern 7. Observer Pattern 8. State Pattern 9. Strategy Pattern 10. Template Pattern 11. Visitor Pattern |

# How to prevent Singleton Pattern from Reflection, Serialization and Cloning?

<https://www.geeksforgeeks.org/prevent-singleton-pattern-reflection-serialization-cloning/>

**Singleton design pattern**: Singleton Pattern says that just**"define a class that has only one instance and provides a global point of access to it".**

In other words, a class must ensure that only single instance should be created and single object can be used by all other classes.

* **Early Instantiation:** creation of instance at load time.
* **Lazy Instantiation:** creation of instance when required.

**How to create Singleton design pattern?**

To create the singleton class, we need to have static member of class, private constructor and static factory method.

* **Static member**: It gets memory only once because of static, it contains the instance of the Singleton class.
* **Private constructor**: It will prevent to instantiate the Singleton class from outside the class.
* **Static factory method:** This provides the global point of access to the Singleton object and returns the instance to the caller.

class A{

private static A obj;

private A(){}

public static A getA(){

if (obj == null){

synchronized(Singleton.class){

if (obj == null){

obj = new Singleton();//instance will be created at request time

}

}

}

return obj;

}

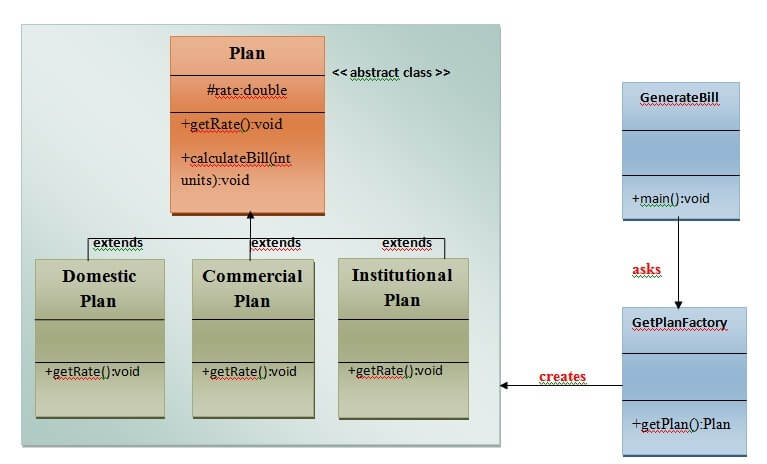
public void doSomething(){

//write your code } }

**Factory Pattern :**

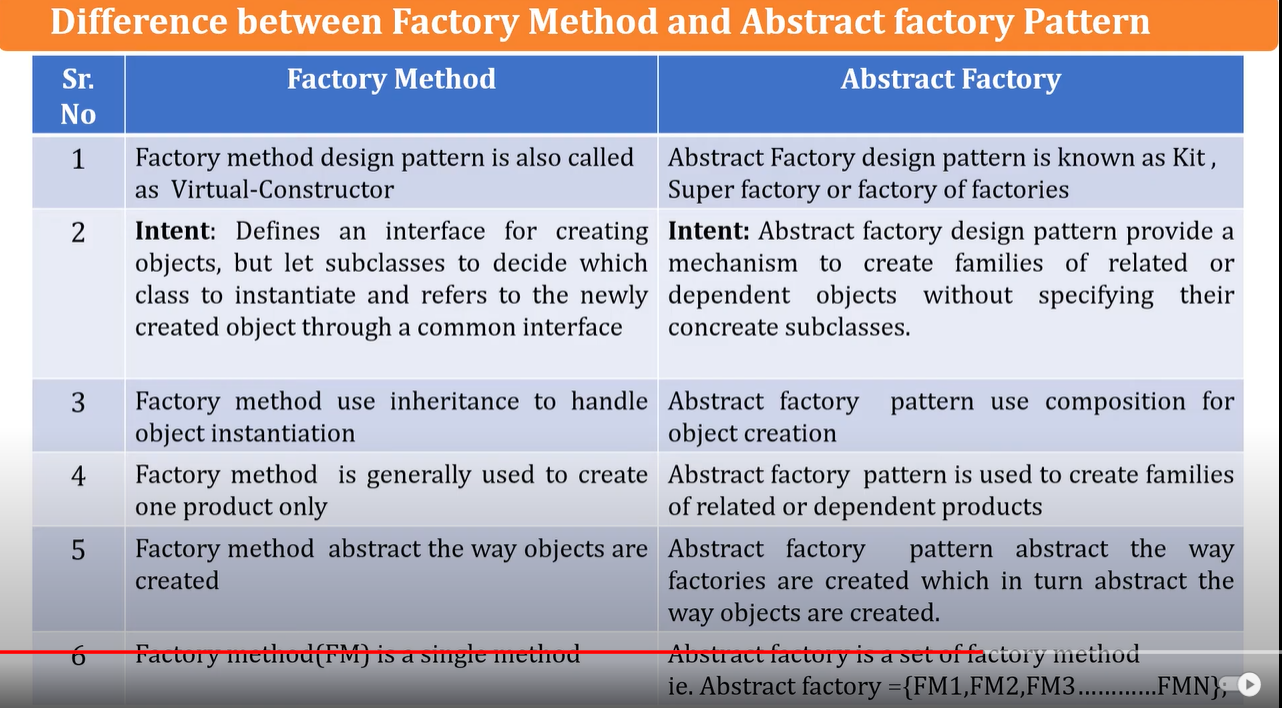
A Factory Pattern or Factory Method Pattern says that just **define an interface or abstract class for creating an object but let the subclasses decide which class to instantiate.** In other words, subclasses are responsible to create the instance of the class.

The Factory Method Pattern is also known as **Virtual Constructor.**



**Abstract Factory:**

**Abstract Factory** Pattern says that just **define an interface or abstract class for creating families of related (or dependent) objects but without specifying their concrete sub-classes.**That means Abstract Factory lets a class returns a factory of classes. So, this is the reason that Abstract Factory Pattern is one level higher than the Factory Pattern.



# Prototype Design Pattern

Prototype Pattern says that **cloning of an existing object instead of creating new one and can also be customized as per the requirement**.

This pattern should be followed, if the cost of creating a new object is expensive and resource intensive.

**Builder:** construct a complex object from simple objects using step-by-step approach

The main advantages of Builder Pattern are as follows:

* It provides clear separation between the construction and representation of an object.
* It provides better control over construction process.
* It supports to change the internal representation of objects.

# Object Pool Pattern

Mostly, performance is the key issue during the software development and the object creation, which may be a costly step.

Object Pool Pattern says that **" to reuse the object that are expensive to create".**

Basically, an Object pool is a container which contains a specified amount of objects. When an object is taken from the pool, it is not available in the pool until it is put back.**Objects in the pool have a lifecycle: creation, validation and destroy.**

A pool helps to manage available resources in a better way. There are many using examples: especially in application servers there are data source pools, thread pools etc.

**(Behavioural)Observer Design** <https://www.youtube.com/watch?v=wiQdrH2YpT4>

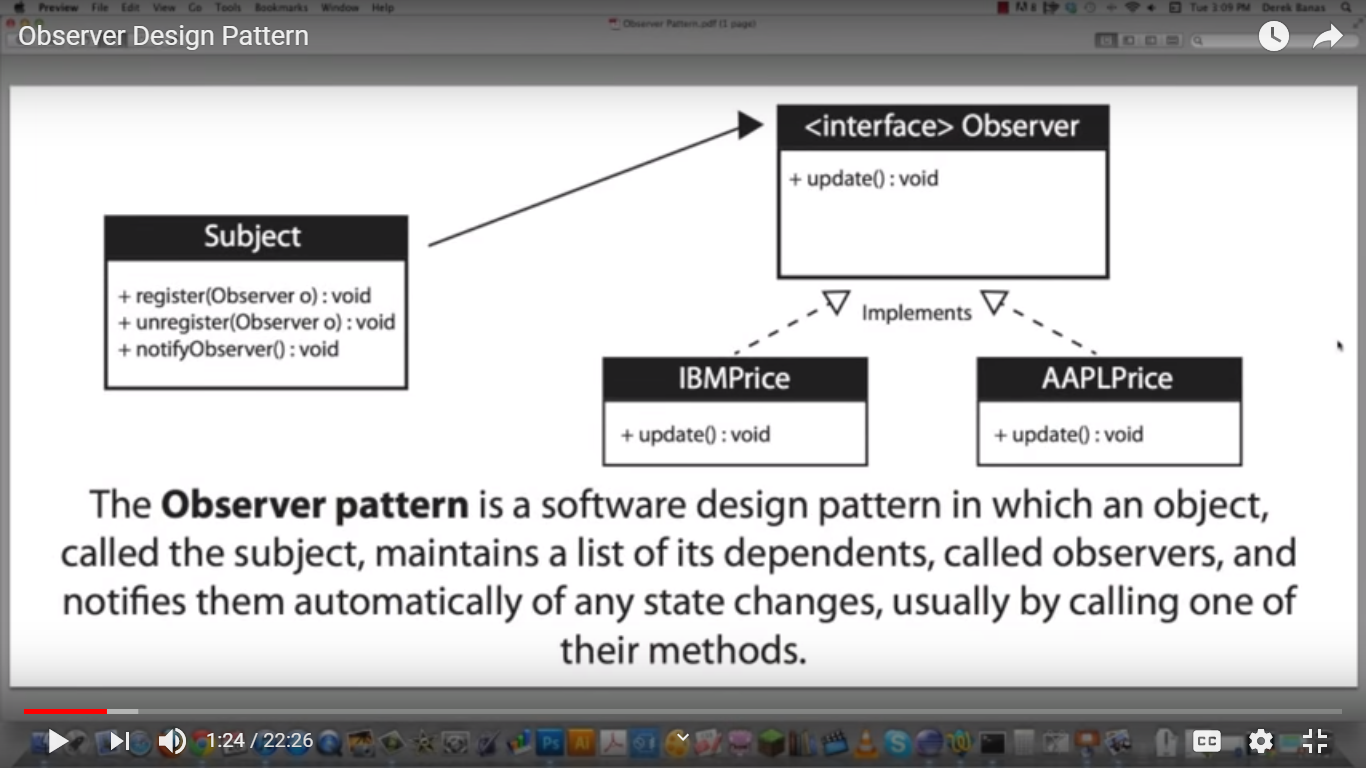
An Observer Pattern says that "just define a one-to-one dependency so that when one object changes state, all its dependents are notified and updated automatically".

The Memento pattern is also known as Dependents or Publish-Subscribe.

**Example Flipkart TV purchase -> Not avaible -> notification when available**

## Benefits:

* It describes the coupling between the objects and the observer.
* It provides the support for broadcast-type communication.



## Design patterns of Elevator Car(LIFT)

<https://tedweishiwang.github.io/journal/object-oriented-design-elevator.html>

One can also implement appropriate design patterns to enhance the functionality and code style. For this problem, I think these design pattern could be useful:

1. [Observer Design Pattern](https://en.wikipedia.org/wiki/Observer_pattern)
2. [Composite Design Pattern](https://en.wikipedia.org/wiki/Composite_pattern)
3. [State Design Pattern](https://en.wikipedia.org/wiki/State_pattern)
4. [Singleton Design Pattern](https://en.wikipedia.org/wiki/Singleton_pattern)
5. Maybe some others, imagination is what powers us!