**Observer Design Pattern**

**Observer Pattern** is one of the **behavioral**[**design pattern**](https://www.journaldev.com/1827/java-design-patterns-example-tutorial). Observer design pattern is useful when you are interested in the state of an object and want to get notified whenever there is any change. In observer pattern, the object that watch on the state of another object are called **Observer** and the object that is being watched is called **Subject**.

According to GoF, observer design pattern intent is;

Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

**Subject** contains a list of observers to notify of any change in it’s state, so it should provide methods using which observers can register and unregister themselves. Subject also contain a method to notify all the observers of any change and either it can send the update while notifying the observer or it can provide another method to get the update.

Observer should have a method to set the object to watch and another method that will be used by Subject to notify them of any updates.

Java provides inbuilt platform for implementing Observer pattern through java.util.Observable class and java.util.Observer interface. However it’s not widely used because the implementation is really simple and most of the times we don’t want to end up extending a class just for implementing Observer pattern as java doesn’t provide multiple inheritance in classes.

Java Message Service (JMS) uses **Observer design pattern** along with [**Mediator pattern**](https://www.journaldev.com/1730/mediator-design-pattern-java)to allow applications to subscribe and publish data to other applications.

Model-View-Controller (MVC) frameworks also use Observer pattern where Model is the Subject and Views are observers that can register to get notified of any change to the model.

### Observer Pattern Java Example

For our observer pattern java program example, we would implement a simple topic and observers can register to this topic. Whenever any new message will be posted to the topic, all the registered observers will be notified and they can consume the message.

Based on the requirements of Subject, here is the base Subject interface that defines the contract methods to be implemented by any concrete subject.

Next we will create contract for Observer, there will be a method to attach the Subject to the observer and another method to be used by Subject to notify of any change.

The method implementation to register and unregister an observer is very simple, the extra method is postMessage() that will be used by client application to post String message to the topic. Notice the boolean variable to keep track of the change in the state of topic and used in notifying observers. This variable is required so that if there is no update and somebody calls notifyObservers() method, it doesn’t send false notifications to the observers.

Also notice the use of [synchronization](https://www.journaldev.com/1061/thread-safety-in-java) in notifyObservers() method to make sure the notification is sent only to the observers registered before the message is published to the topic.

Notice the implementation of update() method where it’s calling Subject getUpdate() method to get the message to consume. We could have avoided this call by passing message as argument to update() method.

Here is a simple test program to consume our topic implementation

ObserverPatternTest

When we run above program, we get following output.

Obj1:: No new message

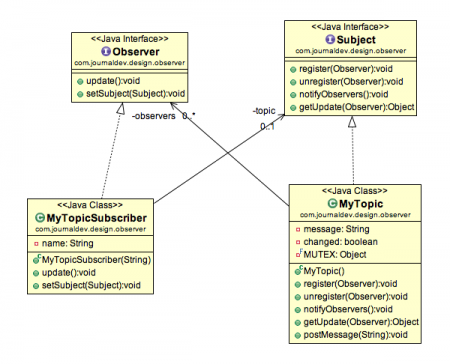
Message Posted to Topic:New Message

Obj1:: Consuming message::New Message

Obj2:: Consuming message::New Message

Obj3:: Consuming message::New Message

### Java Observer Pattern Class Diagram

[](https://cdn.journaldev.com/wp-content/uploads/2013/07/observer-pattern.png)

Observer design pattern is also called as publish-subscribe pattern. Some of it’s implementations are;

* java.util.EventListener in Swing
* javax.servlet.http.HttpSessionBindingListener
* javax.servlet.http.HttpSessionAttributeListener

That’s all for Observer design pattern in java, I hope you liked it. Share your love with comments and by sharing it with others.