The Observer Pattern defines one-to-many dependency between objects so that when one subject changes state, all of its dependents are notified and updated automatically.

Java is having built in ***Observer*** and ***Observable*** classes.

For and Object to become Observer:

* You must implement the Observer interface
* Then call addObserver() on any Observable object(same applies to remove, call deleteObserver())

For the Observable to send notifications:

* Then must call setChanged() (because Observabe implementation check for changed Boolean value in notifyObserver which is getting set by this method) before notifyObserver() or notifyObservers(Object obj)

Disadvantages of Observable:

* Observable is a class not an interface, so any class which extends some other class cannot extend this one, therefore it is a drawback in point of code reusability.
* setChanged() is protected method, without extending Observable class we cannot use this method.

OtherPlaces you see ObserverPattern in Java are ***JavaBeans*** and ***Swing(JButton)***

Notes:

* The Observer Patterns defines a one-to-many relationships between objects.
* Subjects, or as we also know them, Observables, update Observers using a common interface.
* Observers are loosely coupled in that the Observable knows nothing about them, other than they implement the Observer interface.
* You can push or pull data from the Observable when using pattern(pull is considered more correct)
* Don’t depend on a specific order of notification for your Observers.
* Java has several implementations of the Observer Pattern, including the general purpose java.util.Observable
* Watch our for issues with the java.util.Observable implementation
* Don’t be afraid to create your own Observable implementation if needed.
* Swing makes heavy user of the Observer Pattern, as do many GUI frameworks
* You’ll also find the pattern in many other places including JavaBeans and RMI