**Observer Design Pattern**

Observer design pattern is used to notify all the objects to update themselves when an object state gets changed. This is also known as Publish-Subscribe.

**Intent**

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

The intent is to have the notification and update to all the objects when one changes its state.

**Problem**

* Number of objects are dependent on another object and need to be notified for any changes to update themselves.

There may be dependency between different objects and they should be notified to update accordingly when one object changes its state. There may be requirement of adding and deleting new objects too and for different events of state changes.

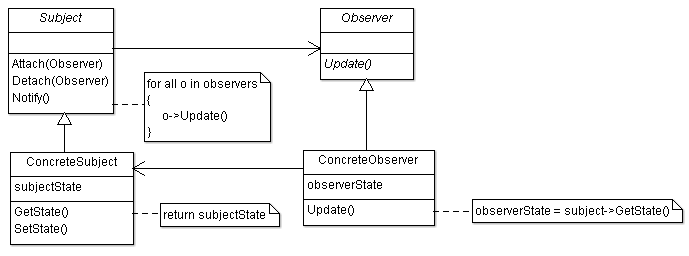
**Solution**

* Give the responsibility of monitoring the state change of object to separate object which should notify to all dependent objects.

**Where it is applicable?**

* There is a one to many dependency between objects.
* State change of one object requires changes in multiple objects.
* Scenarios of Publish-Subscriber, Broadcasting.

**Structure**



**Participant classes**

* **Subject** class provides interface Attach() and Detach() for attaching and detaching observer. The observer objects observe the subject.
* **ConcreteSubject** class implements the method Attach() and Detach(). It keeps state information. It notifies to observers for change of state and also provides method for getting the state information.
* **Observer** class provides Update() interface for updating the Observers.
* **ConcreteObserver** class implements the method Update() for updating the state. It has instance of class ConcreteSubject which is used to get the state information of ConcreteSubject.

**How they work together?**

* Observers are attached with ConcreteSubject. Whenever ConcreteSubject state gets changed, it uses Notify() method to notify all the observers. It calls method Update() of all observers while notification. ConcreteObserver gets the state information from ConcreteSubject using its instance and updates the applicable changes.
* So you can see monitoring of state change of objects is given to Subject and it notifies to all observers for state change. Observers get the subject state information and update themselves. It is easy to add and delete the observer and also to make changes in subject as they do not affect each other. There are two ways to get the update information, one is the push model where subject sends detail information to observer; and another one is pull model where subject sends minimal information and observer gets the information from subject. Push model makes it less reusable and pull model is not much efficient. As the requirement of observer, state changes and need of multiple subject grows, there can be separate mechanism for update, and that responsibility can be given to separate object. The change of state in subject has to be handled properly, as it proceeds to update of lot of observer objects.