**Implementation of weather station problem using the Java’s built-in support**

**Group members**

Muhammad Hamad

Bukhtawar elahi

Sarmad Bashir

**Table of Contents**

1. Approaches used in observer pattern
2. Push description
3. Pull description
4. How we implemented

**APPROACHES USED IN OBSERVER PATTREN**

Whenever we are going to implement Observer pattern using java built-in support we need to use java.util.observable and java.util.observer .There are two main approaches that we can follow depending on our own requirements. The approaches used are

* Push
* Pull

Each of the above approaches have their own pros and cons, we will see each of them individually.

**PUSH**

In push mechanism observers doesn’t require to know anything about subject in order to get updated when there is some change in subject. The main advantage of this model is it provides low coupling between subject and observers.

However disadvantage is less flexibility in a way that it isn’t necessary that subject always know that what exact information the specific observer wants to know in our case e.g. it is possible that some specific observer let say “**Statisticaldisplay”** classis only interested in temperature change in “**Wheatherdata**” class but using pull approach the “**Wheatherdata**” class will provide same information to all of its observers , and specific required information access is something very difficult to achieve in pull approach because all the observers have implemented same “**Observer”** interface. So if we are interested in acquiring the required information only we will move towards pull mechanism.

**PULL**

In order to use push mechanism in java, the observers must contain some reference (pointer) to the “**Wheatherdata**” class inside them, through which they can access public interfaces of “**Wheatherdata**” class whenever there is some change in subject.

The main advantage is more flexibility in a sense that observer do not need to rely completely on subject to retrieve desired information. Each observer can decide itself when to query the subject for requesting information.

In our case we can say that whenever there is some change in “**Wheatherdata**” class it will notify all the observers by calling their “**update**” method now after getting notified it is the responsibility of observers to retrieve the information they needed by using the reference (pointer) to the “**Wheatherdata**” class they already contain.

**How we implemented**

The essential difference between the pull and push models is the flow of control.

In the **push** model, the flow goes like this:

1. Someone calls the **Wheatherdata** object's **setmeasurments ()** method.

2. **Setmeasurments ()** method (directly or indirectly) calls the **notifyObservers ()** method.

3. The **notifyObservers ()** method reaches out of the **Wheatherdata** object to call the **Forecastdisplay** object’s **update** method.

With the push model, most of the action takes place in the **Wheatherdata** class.

But in the **pull** model, the flow works differently:

1. Someone calls the **Wheatherdata** object's **setmeasurments ()** method.

2. **Setmeasurments ()** method (directly or indirectly) calls the **setChanged ()** and **notifyObservers ()** methods.

3. Behind the scenes, in the Java API, the call to **notifyObservers** triggers a call to the **Forecastdisplay** **update ()** method. Now the ball is in **Forecastdisplay’s** court!

4. The **Forecastdisplay** object takes the initiative to reach out and pull in the new weather information data. That is, the **Forecastdisplay** object calls the **Wheatherdata**’**s** getter methods.

Herein lies the difference between pushing and pulling. With the pull model, the **Forecastdisplay** observer pulls information from the **Wheatherdata** observable.

This interface **observer** defines only one method:

**Update (Observable, Object): void** which is called whenever the observed object is changed. The first parameter, an Observable, is the object that changed. The second parameter may be used as follows:

* If using **PUSH** notifications, the Object parameter contains the information needed by the observers about the change.
* If using **PULL** notifications, the Object parameter is null and you should use the Observable parameter in order to extract the information needed.

**In my code:**

* Open my project named as “**Observable\_Pattren\_using\_pull**” and then click source packages.
* Source packages contains 2 folders “**observer\_pattern\_pull\_mechanism**” and “**observer\_pattern\_push\_mechanism**”
* To check **pull** approach of observer pattern open folder “**observer\_pattern\_pull\_mechanism**” and right click on “**Observable\_Pattren\_Assignment**” class and click run.
* Similarly, to check **push** approach of observer pattern open folder “**observer\_pattern\_push\_mechanism**” and right click on “**Observable\_Pattren\_Assignment**” class and click run.