Observer Pattern

Weather Monitoring Application

Weather Station

Device that receives the weather data

Weather Data Object

Guy that tracks the data coming from weather station and updates the display

Display

Shows the current weather conditions.

Job description

- Create an application that uses the weather data object to update the three displays for weather condition
- The three displays are current conditions, weather stats and forecasts.

Weather Data Class

- getTemprature()
- getPressure()
- getHumidity()
- measurementChanged()

Weather
Data Class

Our Task

Implement measurementchanged() function so that it updates the displays accordingly

Simple Solution

- Float temp = getTemperature();
- Float Pr= getPressure();
- Float Hum = getHumidity();
- currentConditionsDisplay.update(temp,Pr,Hum);
- statisticsDisplay.update(temp,Pr,Hum);
- forecastDisplay.update(temp,Pr,Hum);

Problems with the previous approach

We are coding to concrete implementation rather then to an interface.

We have not encapsulated the part that changes from the part that remains constant. (Display Changes)

We have no way to add or remove the weather displays at run time.

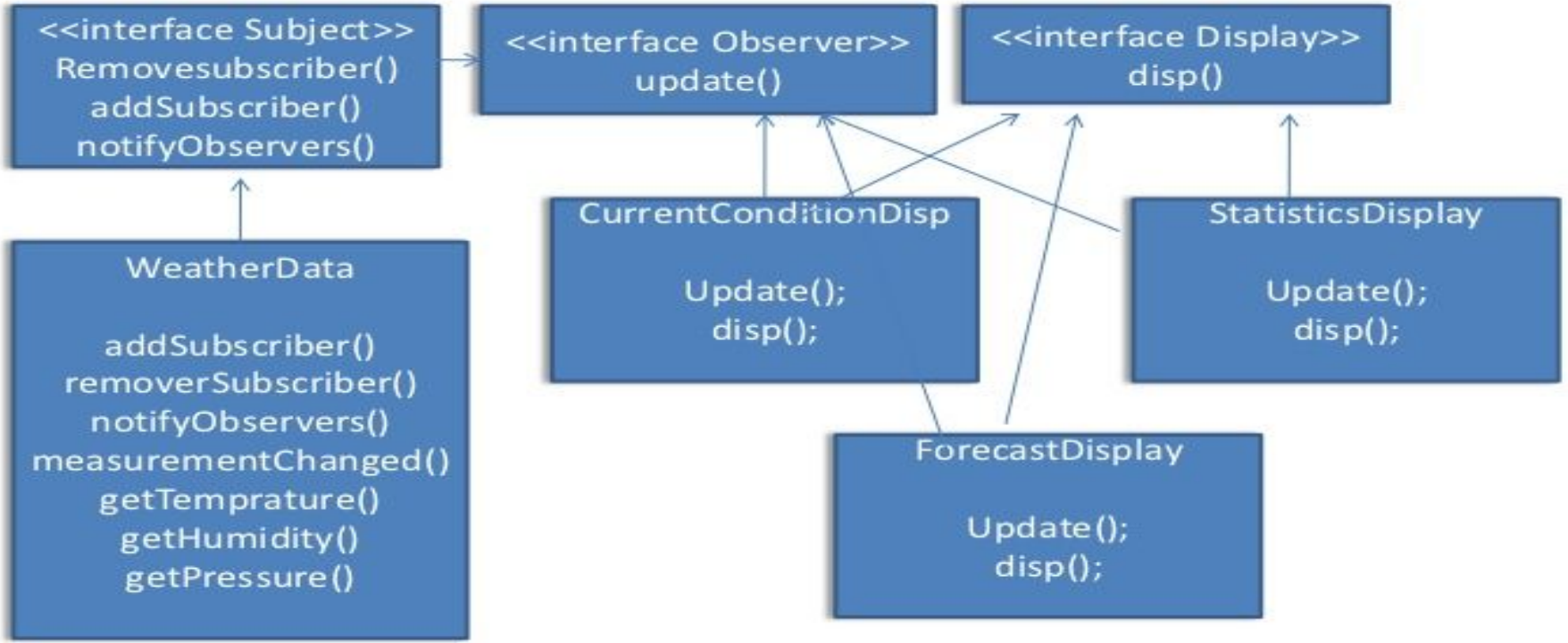
If we want to add new display we have to alter code.

The Observer Pattern

- Similar to Newspaper subscription.
- Publishers publish and subscribers read.
- Once subscribers unsubscribe publishers wont get anything to read.
- Once subscribers subscribe again they get to read.
- Publishers are Subject and subscribers are Observer.

> "The Observer pattern defines a one to many relationship between a set of objects. When the state of one changes it is notified to all the others."

Design: Class Diagram



Implementation (Observer)

```
public class CurrentConditionsDisplay implements Observer,Display {
private float temp,hum,press;
private Subject weatherData;

public CurrentConditionsDisplay (Subject s){
      this.weatherData = s;
   weatherData.addObserver(this);
}
Public void update(float temp, float press, float  hum)
{
      this.temp= temp; this. Press = press;
      display();
}
public void display()
{
      System.out.println(" The current conditions of temperature are good : "+temp);
}
```

Java has built in support for Observer

- You don't have to implement the addObserver, removeObserver etc since these are already implemented in Observable class (not an interface).
- Both are present in the java.util package.
- You can also implement either pull or push style to your observers.
- For an object to become Observer:
  - Implement the Observer Interface and call addObserver() on any Observable Object.

Cont...

- For the Observable to send notifications:
  - You must first call the protected setChanged() method to signify that the state has changed.
  - Then call one of the two notifyObservers() methos
    - notifyObservers()
    - notifyObservers(Object arg)
      - The argument passed is the data Obect.
  - For an Observer
    - Update(Observable o, Object arg)
      - Implements the update method. If you want the push model you can put the data in the dataObject else in case of pull method the Observer can get the data when ever it wants.

Dark Side of Observer API

- The Observable is a class not an interface. Any already defined class which has already extended a class cannot subclass it.
- You can’t create your own implementation that plays well with Java built in Observer API
- Observable class protects setChanged(), so any has to subclass it if they want to use Observable. This hampers the design principle “favor composite over inheritance”.

Advantages

- Minimal coupling between the Subject and Observer Objects
- Many Observers can be added to a Subject without having to modify the Subject.
- Reuse of Subjects without needing to also reuse any of their Observers. The opposite also holds true.
- The only thing a Subject needs to keep track of is its list of Observers.
- The Subject does not need to know the concrete classes of its Observers, only that each one implements the Observer interface

Reference

- Head First design pattern book
- Slideshare slides