观察者模式

1.观察者模式（Observer Pattern），在对象之间定义一对多的依赖，这样一来，当一个对象改变状态，依赖它的对象都会收到通知，并自动更新。

在这里，一的一方被称为主题（Subject），多的一方被称为观察者（Observer）

2.现在假设一个场景，我们需要做一个气象观察站的系统，有一个气象观察站，拥有关于气象的数据，还有其他一些订阅者，需要在气象观察站数据更新的时候，把数据自动发送给他们。这个时候，气象观察站就是那个一，而订阅者则是多，订阅者关于气象的数据依赖与气象观察站的数据，所以当气象观察站的数据变化的时候，订阅者的数据也会变化。

3.现在开始实现这个系统，代码如下：

首先，定义主题的接口：

```
public interface Subject{
 public void registerObserver(Observer observer);
 public void removeObserver(Observer observer);
 public void notifyObserver();
}
```

我们的气象观察站，也就是主题的一方，实现这个接口：

```
import java.util.*;
public class WeatherData implements Subject{
 private ArrayList<Observer> observerList;

 private float humidity;
 private float wind;
 private float temperatrue;


 public WeatherData(){
 observerList = new ArrayList<Observer>();
 }

 public void setMeasurements(float temperatrue,float humidity,float wind){
 this.humidity = humidity;
 this.wind = wind;
 this.temperatrue = temperatrue;
 measuermentsChange();
 }


 public void registerObserver(Observer observer){
 observerList.add(observer);
 }

 public void removeObserver(Observer observer){
 int i = observerList.indexOf(observer);
 if(i>=0){
 observerList.remove(observer);
 }
 }

 public void notifyObserver(){
 for(int i=0;i<observerList.size();i++){
```

```
 observerList.get(i).update(temperatrue,humidity,wind);
 }
 }

 public void measuermentsChange(){
 notifyObserver();
 }


}
```

观察者有两个行为，一个是更新数据，一个展示数据，所以我们定义了如下两个接口：

观察者接口：

```
public interface Observer{
 public void update(float temperatrue,float humidity,float wind);
}
```

展示行为（display）接口：

```
public interface DisplayElement{
 public void display();
}
```

假设有一个今日天气预报订阅了这个气象观察站的推送数据，则实现如下：

```
public class TodayWeatherBoard implements Observer,DisplayElement{

 private float humidity;
 private float wind;
 private float temperatrue;

 private Subject weatherData;

 public TodayWeatherBoard(Subject weatherData){
 this.weatherData = weatherData;
 weatherData.registerObserver(this);
 }

 public void update(float temperatrue,float humidity,float wind){
 this.humidity = humidity;
 this.wind = wind;
 this.temperatrue = temperatrue;
 display();
 }

 public void display(){
 System.out.println("接下来播报今天的天气预报，今天的温度为"+temperatrue+"摄氏度，今天的湿度为"+humidity+"度，今天的
 };

}
```

做一个测试类：

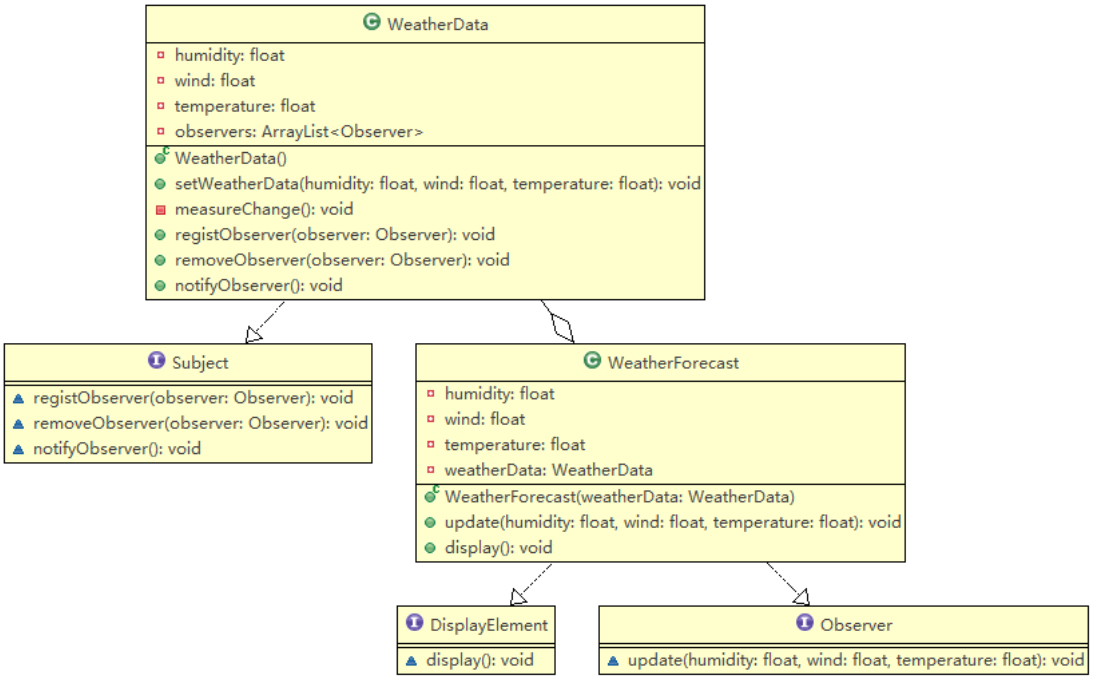
```
public class weatherExample{
 public static void main(String[] args){
 WeatherData weatherData = new WeatherData();
 TodayWeatherBoard todayWeatherBoard = new TodayWeatherBoard(weatherData);

 weatherData.setMeasurements(12,23f,45f);

 }
}
```

输出：接下来播报今天的天气预报，今天的温度为12.0摄氏度，今天的湿度为23.0度，今天的风力为45.0度。天气预报播报完毕，祝大家心情愉快。

4.观察者模式的特点：

- 观察者和主题之间实现了松耦合，主题对象维护了一个订阅者的列表，并使用一个接口来更新所有订阅者的数据。
- 主题维护了一个列表，这个列表放了订阅者对象的强引用，所以当订阅者太多的时候，很容易造成内存溢出，需要从强引用改变成弱引用。
- 数据可以是主题对象主动“推送”指定的数据给订阅者，也可以是订阅者主题从主题对对象那里“拉”数据，如果要实现拉数据，则主题对象需要实现getter接口。
- JDK内部的有很多地方使用了观察者模式，比如Swing，JavaBeans和RMI。
- JDK有自己专门的接口来支持观察者模式，使用的是Observable类和Obserber接口，一般情况下，使用这两个接口也可以实现观察者模式，但是由于Observable是一个类，所以如果你的类需要继承其他的类，则Observable不适用于这个场景，并且，Observer接口只能和Observable类配合使用，如果你的主题对象不是继承于Observable类，则无法配合Observer接口实现观察者模式。

5.观察者模式的UML图：



其中，DisplayElement接口不是必要的，主要是Subject和Observer两个接口

6.使用Observable和Observer接口实现的观察者模式的主题对象和订阅者的代码：

主题对象：

```
import java.util.Observable;
import java.util.Observer;
public class WeatherData extends Observable{

 private float humidity;
 private float wind;
 private float temperatrue;


 public WeatherData(){

 }

 public void setMeasurements(float temperatrue,float humidity,float wind){
 this.humidity = humidity;
 this.wind = wind;
 this.temperatrue = temperatrue;
 measuermentsChange();
 }

 public void measuermentsChange(){
 setChanged();
 notifyObservers("The truth that you leave");
 }

 public float getHumidity(){
 return humidity;
 }

 public float getTemperatrue(){
 return temperatrue;
 }

 public float getWind(){
 return wind;
 }

}
```

订阅者对象：

```
import java.util.Observable;
import java.util.Observer;
public class TodayWeatherBoard implements Observer,DisplayElement{

 private float humidity;
 private float wind;
 private float temperatrue;

 Observable observable;

 public TodayWeatherBoard(Observable observable){
 this.observable = observable;
 observable.addObserver(this);
 }

 public void update(Observable observable,Object arg){
 System.out.println(arg);
 if(observable instanceof WeatherData){
 WeatherData weatherData = (WeatherData)observable;
```

```
this.humidity = weatherData.getHumidity();
this.temperatrue = weatherData.getTemperatrue();
this.wind = weatherData.getWind();
display();
}
}

public void display(){
System.out.println("接下来播报今天的天气预报，今天的温度为"+temperatrue+"摄氏度，今天的湿度为"+humidity+"度，今天的
};

}
```