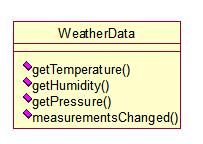
**观察者模式：**在对象之间定义一对多的依赖，这样一来，当一个对象改变状态，依赖它的对象都会收到通知，并自动更新。

　　举个例子：气象监测应用

　　　　此系统的三个部分是气象站（获取实际气象数据的物理装置）、WeatherData对象（追踪来自气象站的数据，并更新布告板）和布告板（有三个布告板，一个用来显示目前天气状况，一个显示气象统计，最后一个用来显示天气预报）

　　看一下WeatherData源文件：



　　上面的三个get各自返回最近的气象测量数据（分别为：温度、湿度、气压），我们不在乎这些变量如何通过气象站这个物理装置得到以及它们如何设置，WeatherData对象自己知道如何从气象站获取更新信息。

　　measurementsChanged() {/\* 一旦气象测量更新，此方法会调用，这里面写自己的代码 \*/}

**先看一个错误示范：**

[复制代码](javascript:void(0);)

public class WeatherData{

//实例变量声明

public void measurementsChanged()

{

//调用get函数，获取最近的测量值

float temp = getTemperature();

float humidity = getHumidity();

float pressure = getPressure();

//更新现在布告板、气象统计布告板、天气预报布告板

currentConditionsDisplay.update(temp,humidity,pressure);

statisticsDisplay.update(temp,humidity,pressure);

forecastDisplay.update(temp,humidity,pressure);

}

//这里是其他的WeatherData方法

}

[复制代码](javascript:void(0);)

　　问题出在哪里？为什么这样是错误的？

[复制代码](javascript:void(0);)

currentConditionsDisplay.update(temp,humidity,pressure);

statisticsDisplay.update(temp,humidity,pressure);

forecastDisplay.update(temp,humidity,pressure);

/\*

以上属于具体实现编程，会导致我们以后在增加或删除布告板时必须修改程序\*/

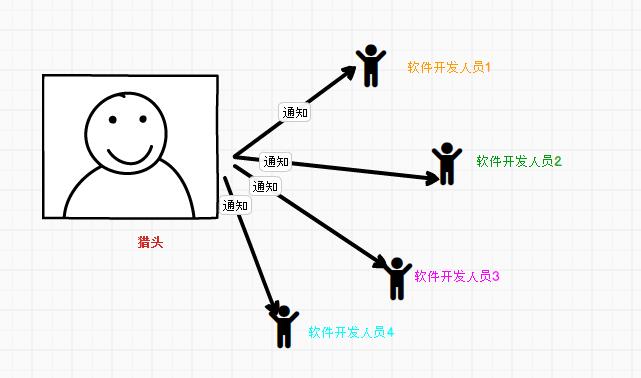
/\*

另外上面的代码属于改变的部分，因此需要封装独立出来

\*/

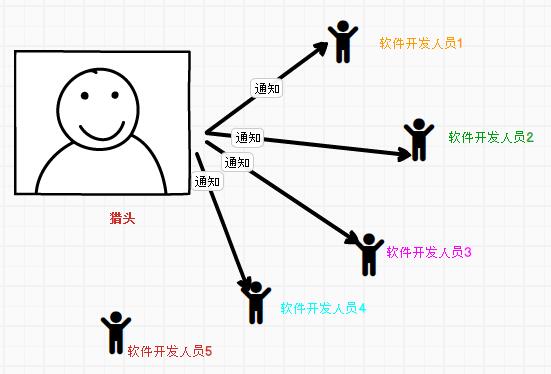
[复制代码](javascript:void(0);)

 　　在继续向下走之前，先看一个观察者模式的实例

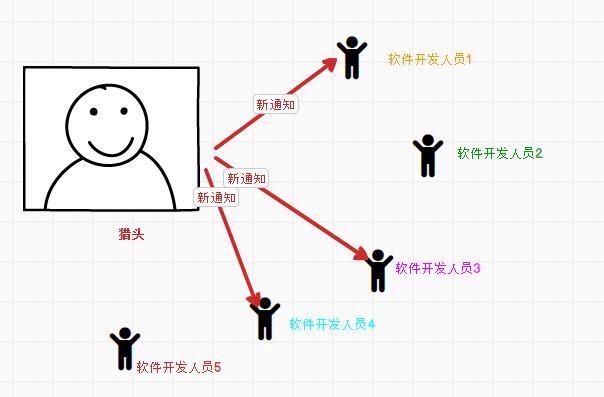


　　寻找工作的软件开发人员将自己的联系方式留给猎头，然后当猎头获取工作应聘消息时需要通知他手里名单上的所有人员。

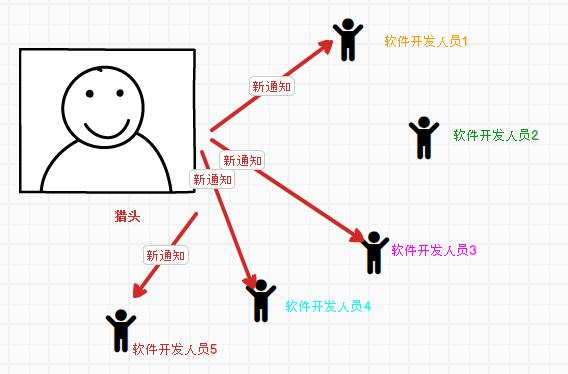
　　但是也有这样的情况，当软件开发人员没有向猎头提交信息时，猎头在收到消息后不同通知他。



　　同样接受猎头通知的求职人员同样可以取消通过猎头来寻求工作，比如**软件开发人员2取消了这种求职方式**，那么他就不会再收到猎头的通知。

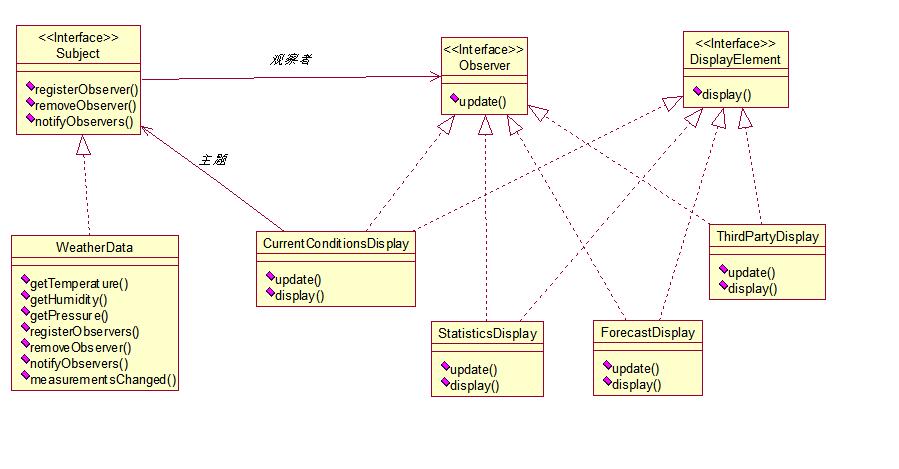


　　但是**软件开发人员5希望通过这种方式来找工作**，那么就需要去和猎头联系并留下自己的联系方式，那么从此软件开发人员5也可以收到通知了



　　在明白上面的例子之后，重新回到我们的气象监测应用上。

　　尝试着画出实现气象站所需要的类，其中包括WeatherData类以及布告栏组件。确定你的图像能够显示出各个部分如何结合起来，以及别的开发人员如何能够实现自己的布告栏组件。**（每个布告栏都应该有一个被命名为“subject（主题）”的指针指向WeatherData对象，为了避免混乱下面没有画出）**



**实现气象站：**

**从建立接口开始**

[复制代码](javascript:void(0);)

public interface Subject{

public void registerObserver(Observer o);

public void removeObserver(Observer o);

public void notifyObservers();

}

public interface Observer{

public void update(float temp,float humidity,float pressure);

}

public interface DisplayElement{

public void display();

}

[复制代码](javascript:void(0);)

**在WeatherData中实现主题接口**

[复制代码](javascript:void(0);)

public class WeatherData implements Subject{//WeatherData实现了Subject接口

private ArrayList observers;//添加一个ArrayList来记录观察者，此ArrayList是在构造器中建立的

private float temperature;

private float humidity;

private float pressure;

public WeatherData(){

observers = new ArrayList();

}

public void registerObserver(Observer o){//当注册观察者时，我们只要把它加到ArrayList后面即可

observers.add(o);

}

public void removeObserver(Observer o){//同样删除时，得到索引然后从ArrayList删除即可

int i = observers.indexOf(o);

if(i >= 0){

observers.remove(i);

}

}

public void notifyObservers(){//这里我们把状态都告诉每一个观察者，因为观察者都实现了update(),

for(int i = 0 ;i < observers.size(); i++){

Observer observer = (Observer)observers.get(i);

observer.update(temperature,humidity,pressure);

}

}

public void measurementsChanged(){//当气象站更新数据时，我们通知观察者

notifyObservers();

}

public void setMeasurements(float temperature,float humidity,float pressure){

this.temperature = temperature;

this.humidity = humidity;

this.pressure = pressure;

measurementsChanged();

}

}

[复制代码](javascript:void(0);)

**下面只给出了目前状况的布告板：**

[复制代码](javascript:void(0);)

//此布告板实现了Observer接口，所以可以从WeatherData对象中获取改变，同样也实现了DisplayElement的接口

public class CurrentConditionsDisplay implements Observer,DisplayElement{

private float temperature;

private float humidity;

private Subject weatherData;

public CurrentConditionsDisplay(Subject weatherData){//构造器需要weatherData对象 作为注册使用

this.weatherData = weatherData;

weatherData.registerObserver(this);

}

public void update(float temperature,float humidity,float pressure){

this.temperature = temperature;

this.humidity = humidity;//当update()调用时，我们把温度和湿度保存起来，然后调用display()

display();//这里之前说的可能不清楚，目前状况只需要显示温度和湿度就可以了，

//气象检测系统不要求在这个布告板显示气压

}

public void display(){

System.out.println("Current conditions:"+temperature

+"F degrees and "+ humidity +" % humidity");

}

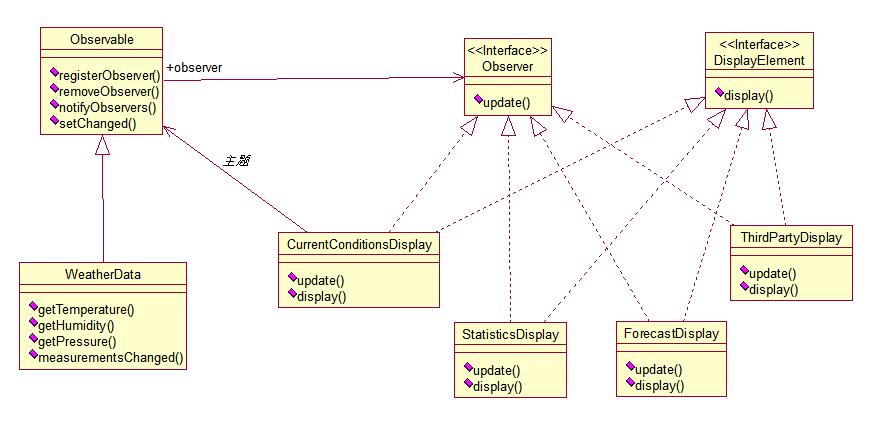
}

[复制代码](javascript:void(0);)

**下面讨论使用Java内置的观察者模式来完成气象监测应用**

在java.util包中包含最基本的Observer接口与Observable类。甚至可以实现推(push)和拉(pull)方式来传送数据。**什么叫做push，什么叫做pull呢？**

　 还用之前的例子，**猎头**主动将信息通知给每一位软件开发人员叫做push，那么至于拉（pull）就很直观了，就是**软件开发人员**主动向猎头联系索取信息。**（每个布告栏都应该有一个被命名为“subject（主题）”的指针指向WeatherData对象，为了避免混乱下面没有画出）**



　　在这里需要注意的地方：

　　在Observable里面有一个setChanged()方法，它的作用是在调用notifyObserver()之前先调用setChanged()，如果changed为true那么就进行通知，否则就不通知。（**setChanged()在每次消息有所改变时都要进行修改**）

　　看一下里面的代码：

[复制代码](javascript:void(0);)

setChanged(){

changed= true;//将changed标志设为true

}

notifyObservers(Object arg){

if(changed){//只有在是true是才通知

for every observer on the list{

call update(this.arg);

}

changed=false;//通知完观察者之后，把changed标志设回false

}

}

notifyObservers(){

notifyObservers(null);

}

[复制代码](javascript:void(0);)

　　1.**Observable是一个类**，首先既然它是个类，那么我们需要设计一个类来继承它。但是Java不支持多重继承，这就限制了Observable的服用能力。在再者因为没有Observable接口，所以自己无法创建自己的实现来和Java内置的Observer API搭配使用，也无法将java.util的实现换成另一套做法的实现。

　　2.Observable将关键的方法保护起来，如果查看Observable API则会发现，setChanged()方法被保护了起来（被定义为protected）。这就意味着：除非你继承自Observable，否则你无法创建Observable实例并组合到你自己的对象中来。**这违背了设计原则“多用组合，少用继承”**

　　到此我们现在的设计模式已经学习了三个——简单工厂模式、策略模式以及现在的观察者模式。

　　未来的路还很远，加油呐！