* 设计模式 - 观察者模式（observer）：

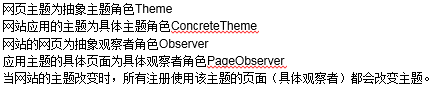
观察者设计模式解决的问题是：当一个对象发生指定的动作时，要通过另外一个对象做出相应的处理。



观察者模式完美的将观察者和被观察者的对象分离开。此模式中，一个目标物件管理所有相依于它的观察者物件，并且在它本身的状态改变时主动发出通知。

定义了一种一对多的依赖关系，让多个观察者对象同时监听某一个主题对象。这个主题对象在状态上发生变化时，会通知所有观察者对象，使他们能够自动更新自己。

示例：**模拟网站主题和页面的实例**



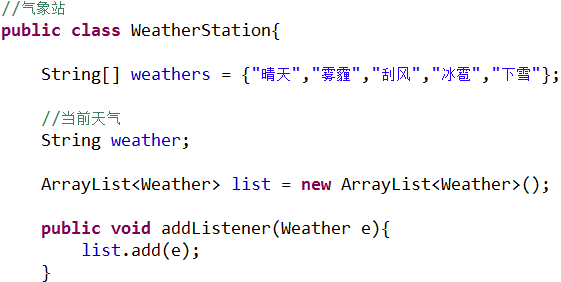
**观察者模式角色：**

**1）抽象主题(Subject)角色：**抽象主题角色把所有对观察者对象的引用保存在一个聚集(比如ArrayList对象)里，每个主题都可以有任何数量的观察者。抽象主题提供一个接口，可以增加和删除观察者对象，抽象主题角色又叫做抽象被观察者(Observable)角色。

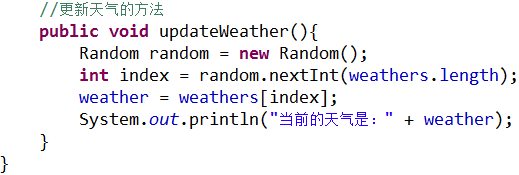
2）**具体主题(ConcreteSubject)角色：**将有关状态存入具体观察者对象；在具体主题的内部状态改变时，给所有登记过的观察者发出通知。具体主题角色又叫做具体被观察者(Concrete Observable)角色。

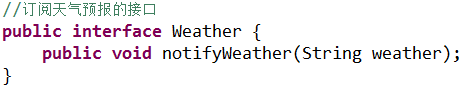
3）**抽象观察者(Observer)角色：**为所有的具体观察者定义一个接口，在得到主题的通知时更新自己，这个接口叫做更新接口。

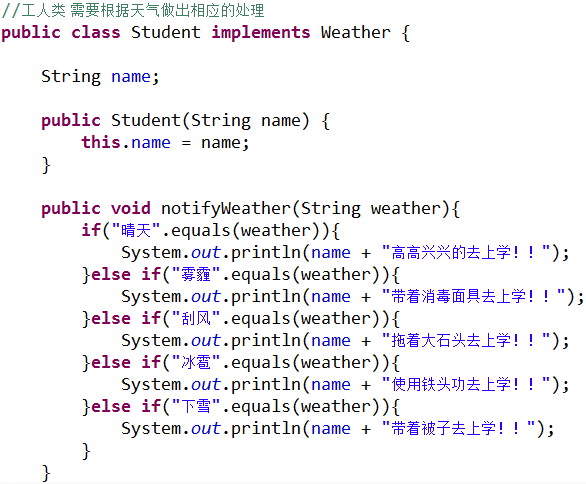
4）**具体观察者(ConcreteObserver)角色：**存储与主题的状态相应的状态。具体观察者角色实现抽象观察者角色所要求的更新接口，以便使本身的状态与主题的状态 协调。如果需要，具体观察者角色可以保持一个指向具体主题对象的引用。



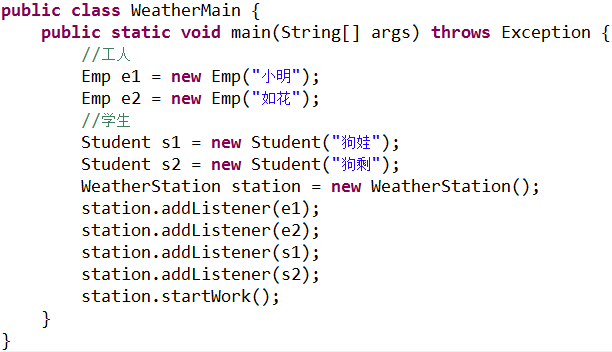


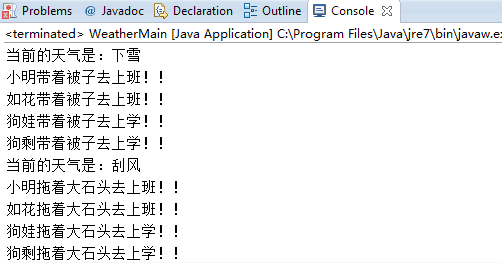














如下的实现方式很依赖于Emp工人类，假如有一天这个Emp类被删了，使用接口以后实现了解耦，就算多加一些群体(比如以公司为单位关注天气)，就可以直接实现接口实现，而不用去修改气象站WeatherStation类。实际上就是接口充当了中介的角色了。

