Object Oriented Analysis & Design
面向对象分析与设计

Lecture_09 GOF 设计模式（一）

1）单实例　　2）适配器　　3）外观　　4）观察者

主讲: 姜宁康 博士

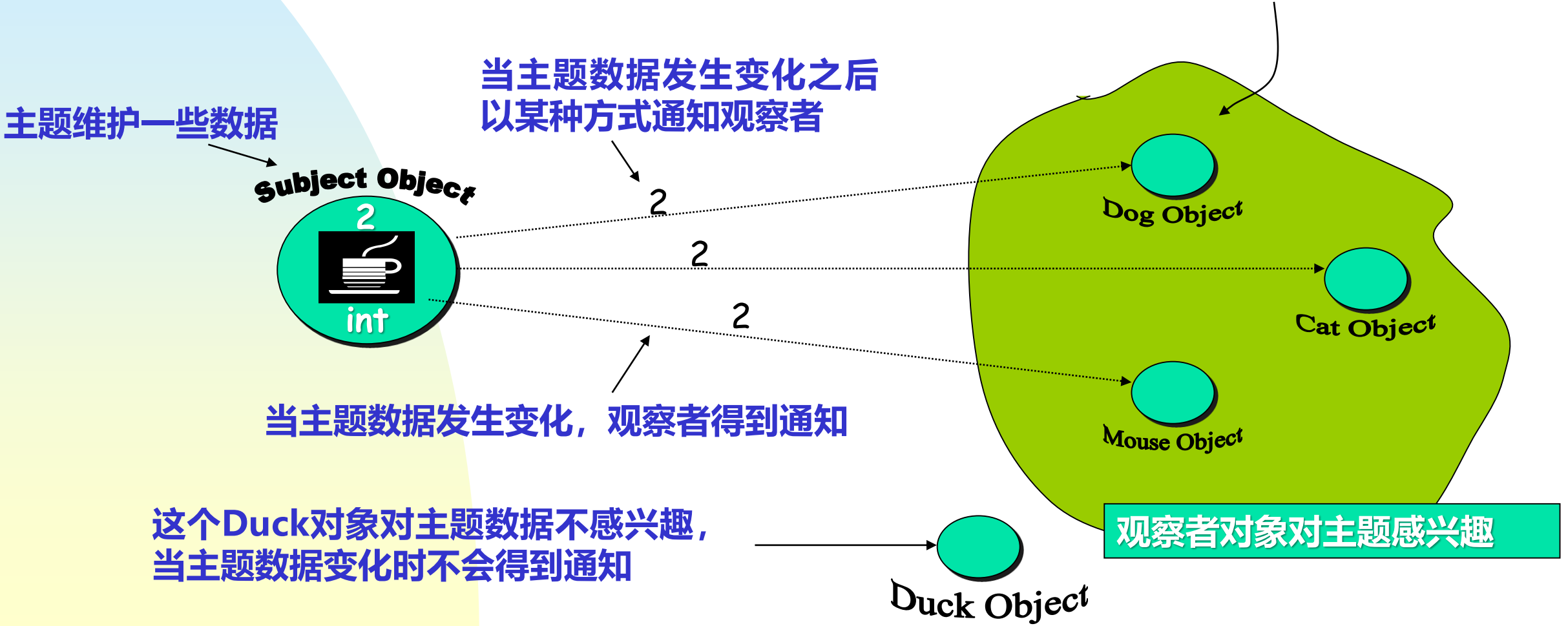
6、GOF设计模式四: 观察者模式 Observer（二）

- 观察者模式 Observer

6.1 Publishers + Subscribers = Observer Pattern

- Publisher == Subject 主题
- Subscribers == Observers 观察者

观察者已经订阅了主题数据，当数据发生变化之后将会收到更新数据或者更新通知

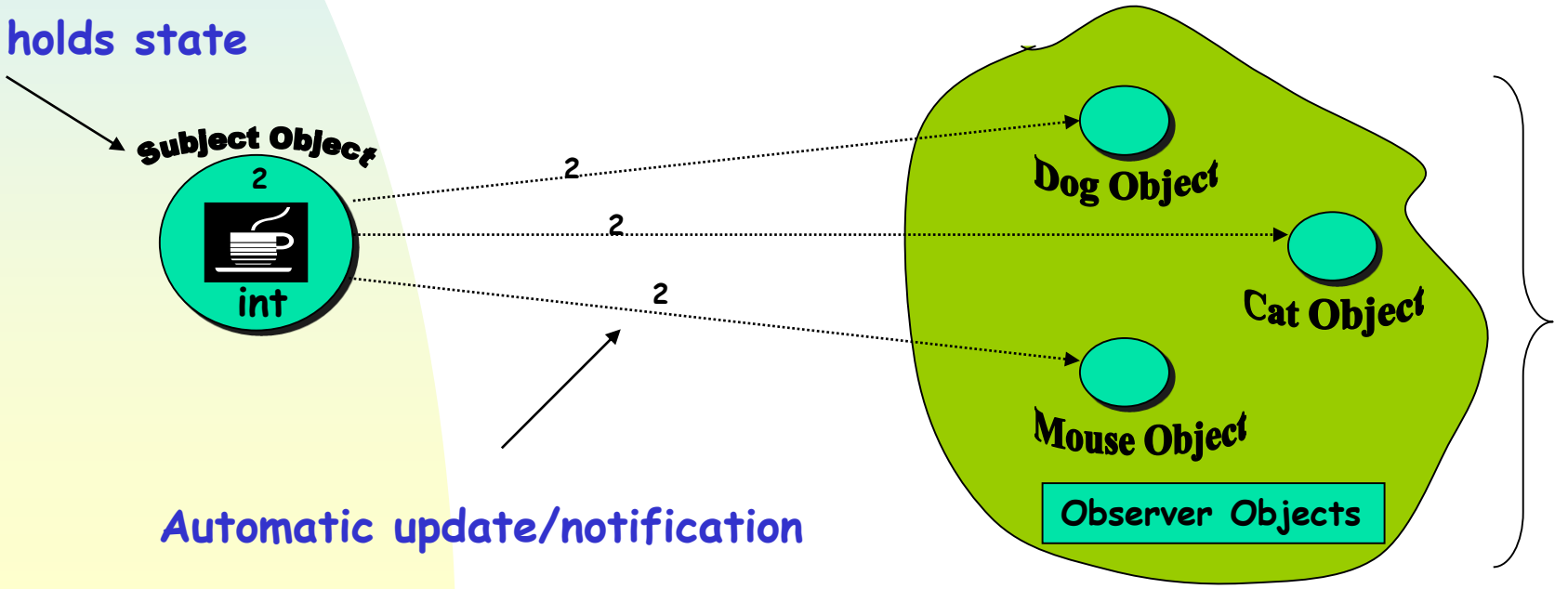


6.2 观察者模式定义 Observer Pattern Defined

The Observer Pattern defines a one-to-many dependency between objects so that when one object changes state, all of its dependents are notified and updated automatically 定义对象之间的一对多依赖关系，当一个对象改变状态时，所有依赖于它的对象都会自动获得通知

One to many relationship (Subject can have many observers)

Object that holds state



Automatic update/notification

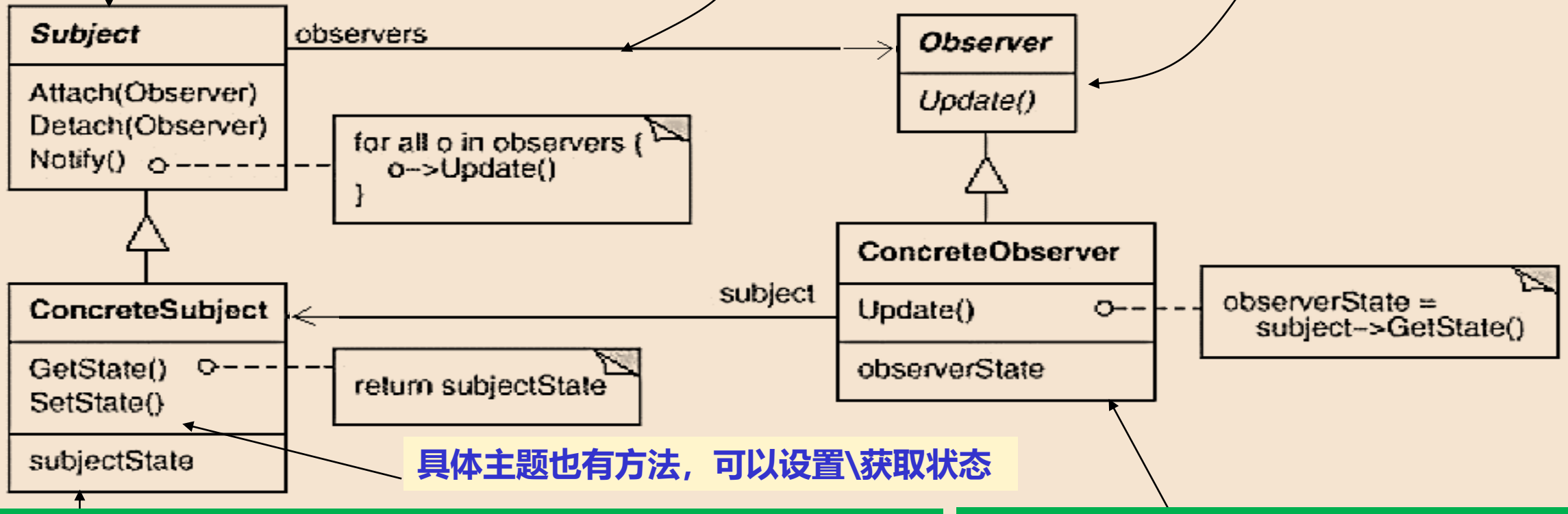
Dependent objects (observers are dependent on the subject to update them when data changes)

6.3 观察者模式架构 Observer Class Diagram

这是主题接口。观察者利用这个接口注册到主题中，或者取消关注不再是观察者

每个主题能有多个观察者

所有潜在的观察者都要实现 Observer接口，它只有一个方法 update（ ），当主题的状态发生变化之后被调用



具体主题也有方法，可以设置\获取状态

一个具体主题需要实现抽象主题的接口。除了register (attach) 和 remove（detach）方法，具体主题还要实现 notify() 方法，在主题状态发生变化时通知观察者

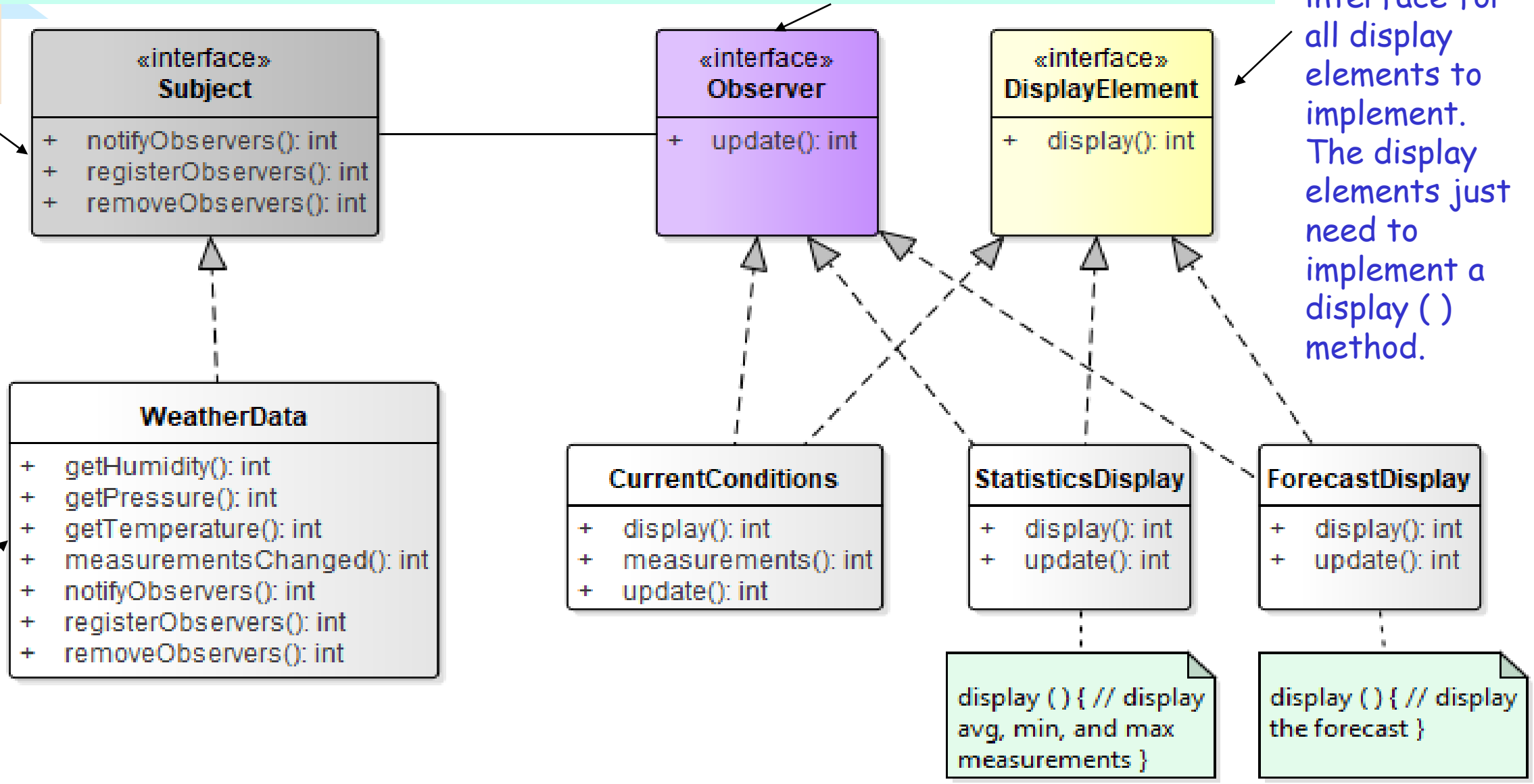
具体观察者实现 Observer 接口. 每个观察者都要注册到一个具体的主题以获得更新通知

6.4 Weather Station 新的设计方案

所有的观察者都要实现的接口。这样一来，subject 就可以用同一的方式与所有观察者进行交流

Subject interface

Create an interface for all display elements to implement. The display elements just need to implement a display ( ) method.



Weather Data类实现了 Subject 接口

6.4 Weather Station 新的设计方案: 实现

```
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 ( );
}
```

Both of these methods take an Observer as an argument, that is the Observer to be registered or removed.

This method is called to notify all observers when the Subject's state has changed.

The Observer interface is implemented by all observers, so they all have to implement the update ( ) method.

These are the state values the Observers get from the Subject when a weather measurement changes.

The DisplayElement interface just includes one method, display ( ), that we will call when the display element needs to be displayed.

6.4 Weather Station 新的设计方案: 实现

```
public class WeatherData implements Subject {
    private ArrayList observers;
    private float temperature;
    private float humidity;
    private float pressure;
    public WeatherData ( ){
        observers = new ArrayList ( );
    }
    public void registerObservers (Observer o) {
        observers.add(o);
    }
    public void removeObservers (Observer o) {
        int j = observer.indexOf(o);
        if (j >= 0) {
            observers.remove(j);
        } }
    public void notifyObservers ( ) {
        for (int j = 0; j < observers.size(); j++) {
            Observer observer = (Observer)observers.get(j);
            observer.update(temperature, humidity, pressure);
        }}
    public void measurementsChanged ( ) {
        notifyObservers ( ); }
    // add a set method for testing + other methods.}
```

Added an ArrayList to hold the Observers, and we create it in the constructor.

Here we implement the Subject Interface

Notify the observers when measurements change.

6.4 Weather Station 实现: The Display Elements

```
public class CurrentConditionsDisplay implements Observer, DisplayElement {
    private float temperatue;
    private float humidity;
    private Subject weatherData;

    public CurrentConditionsDisplay (Subject weatherDataS) {
        this.weatherData = weatherDataS;
        weatherData.registerObserver (this);
    }
    public void update (float temperature, float humidity, float pressure) {
        this.temperature = temperature;
        this.humidity = humidity;
         display ( );
    }
    public void display ( ){
        System.out.println(" Current conditions : " + temperature + " F degrees  and " + humidity + " %
    humidity" );
    }
}
```

Implements the Observer and DisplayElement interfaces

The constructors passed the weatherData object (the subject) and we use it to register the display as an observer.

When update ( ) is called, we save the temp and humidity and call display ( )

The display ( ) method just prints out the most recent temp and humidity.

6.5 推模式

- 推模式是当通知消息来之时，把所有相关信息都通过参数的形式“推给”观察者

优点：

1. 所有信息通过参数传递过来，直接、简单，观察者可以马上进行处理
2. 观察者与被观察者没有一点联系，两者几乎没有耦合

缺点：

1. 所有信息都强迫推给观察者，不管有用与否
2. 如果想添加一个参数，那就需要修改所有观察者的接口函数

6.6 拉模式

- 当通知消息来之时，通知的函数不带任何相关的信息，而是要观察者主动去被主题对象那里去“拉”信息

优点：

1. 可以主动去取自己感星期的信息
2. 如要添加一个参数，无需修改观察者

缺点：

1. 观察者与被观察者有一定的联系

6.7 How to apply 应用 Observer DP

- **Check list**
  - 主题对象只与观察者基类有耦合 The Subject is coupled only to the Observer base class
  - 客户配置观察者的数量与类型 The client configures the number and type of Observers
  - Observers 首先要知道 Subject，然后把自己注册到 Subject 中
  - Subject 保存所有注册过的 Observer，当状态发生变化时，广播给所有注册过的观察者
  - Subject 可以采用 “push”或者“pull”的方式，与 Observer 交流信息

- 本讲结束