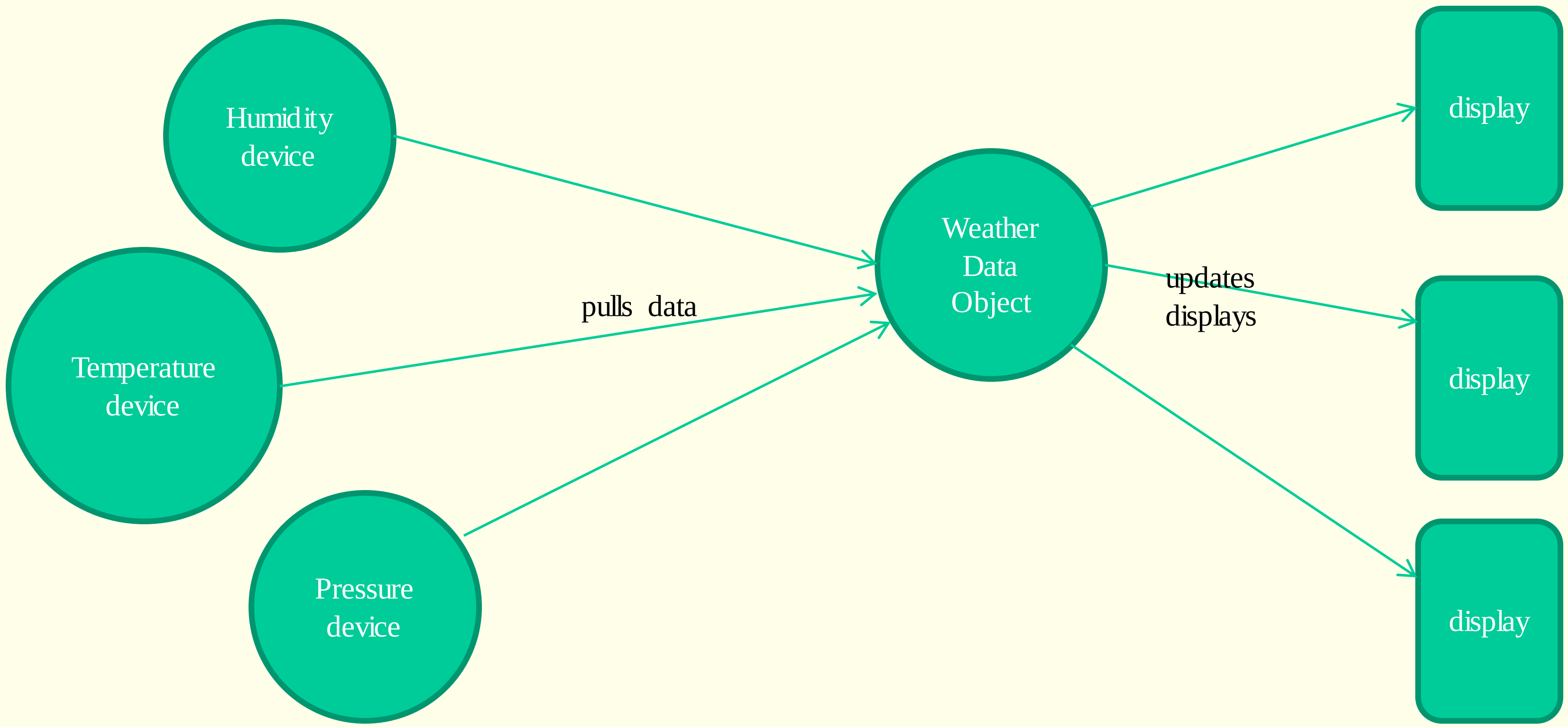
Weather Station Page 39+

In this application, weather station devices supply data to a weather data object. As the data changes various displays are updated.



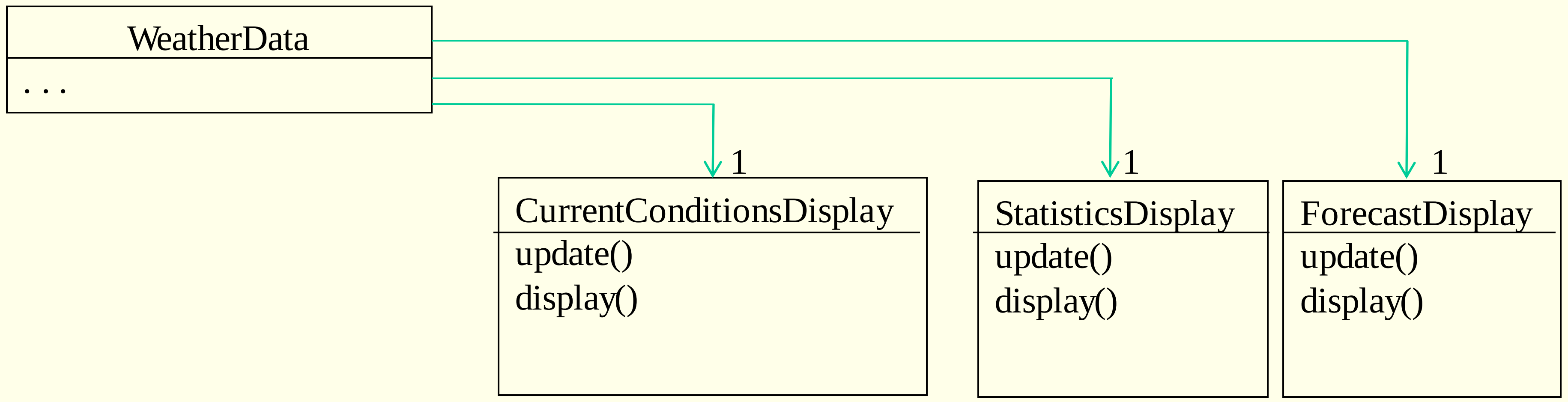
Weather Station Page 42

The getter methods just return current values for temperature, etc.

measurementsChanged() is called whenever temperature etc have changed …. So when this is called, the system must update the displays, so a first approach is:

```
Public void measurementsChanged() {
        float temp = getTemperature();
        float humidity = getHumidity();
        float pressure = getPressure();
        currentConditionsDisplay.update(temp, humidity, pressure);
        statisticsDisplay.update(temp, humidity, pressure);
        forecastDisplay.update(temp, humidity, pressure);
```

Weather Station Page 42



WeatherData is tightly coupled to the displays

To add or remove displays it is necessary to modify the code of WeatherData … can this be avoided?

Observer Pattern

Also known as **publish/subscribe**

The essence of this pattern is that one or more objects (observers/listeners) are registered to observe an event which may be raised by the observed object (the subject).

The subject maintains a collection of the observers.

Observer Pattern

Problem:

There are many objects (observers / subscribers) needing to know of the state changes, or events, of another object (subject / publisher), and we want to keep the coupling low.

Solution:

The object that is responsible for the event is given the responsibility of monitoring for the event – this object is the subject.

Objects that are interested in the event must register with the subject as interested parties – as observers.

The subject will notify its observers when the event occurs.

Observer Pattern

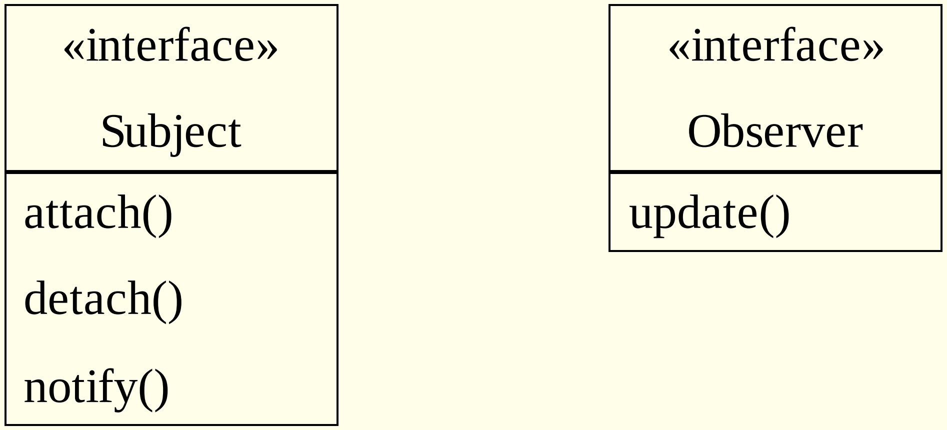
The Observer Pattern defines a one to many dependency between objects so that when one object changes state, all its dependents are notified automatically

Observer: objects that want to be notified of a certain event. An observer must have an ***update*** method whereby it is notified of an event.

Subject: the object that triggers the event. It must implement:

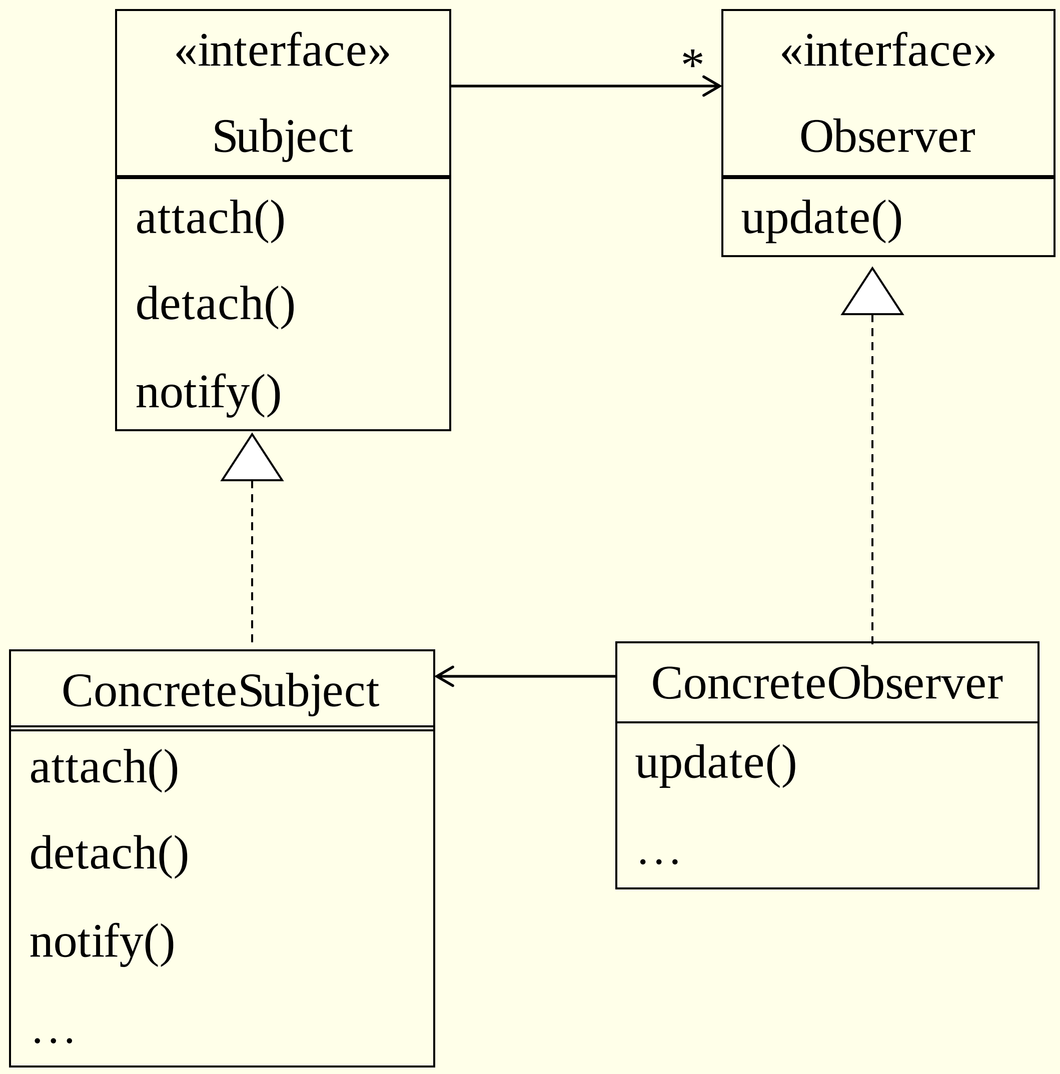
- ***attach*** (observer) - add an observer to its list of observers
- ***detach*** (observer) - remove an observer from …
- ***notify*** () - goes through its list of observers calling each observer's update method
- As needed - additional methods to allow an observer to get additional information

Interfaces



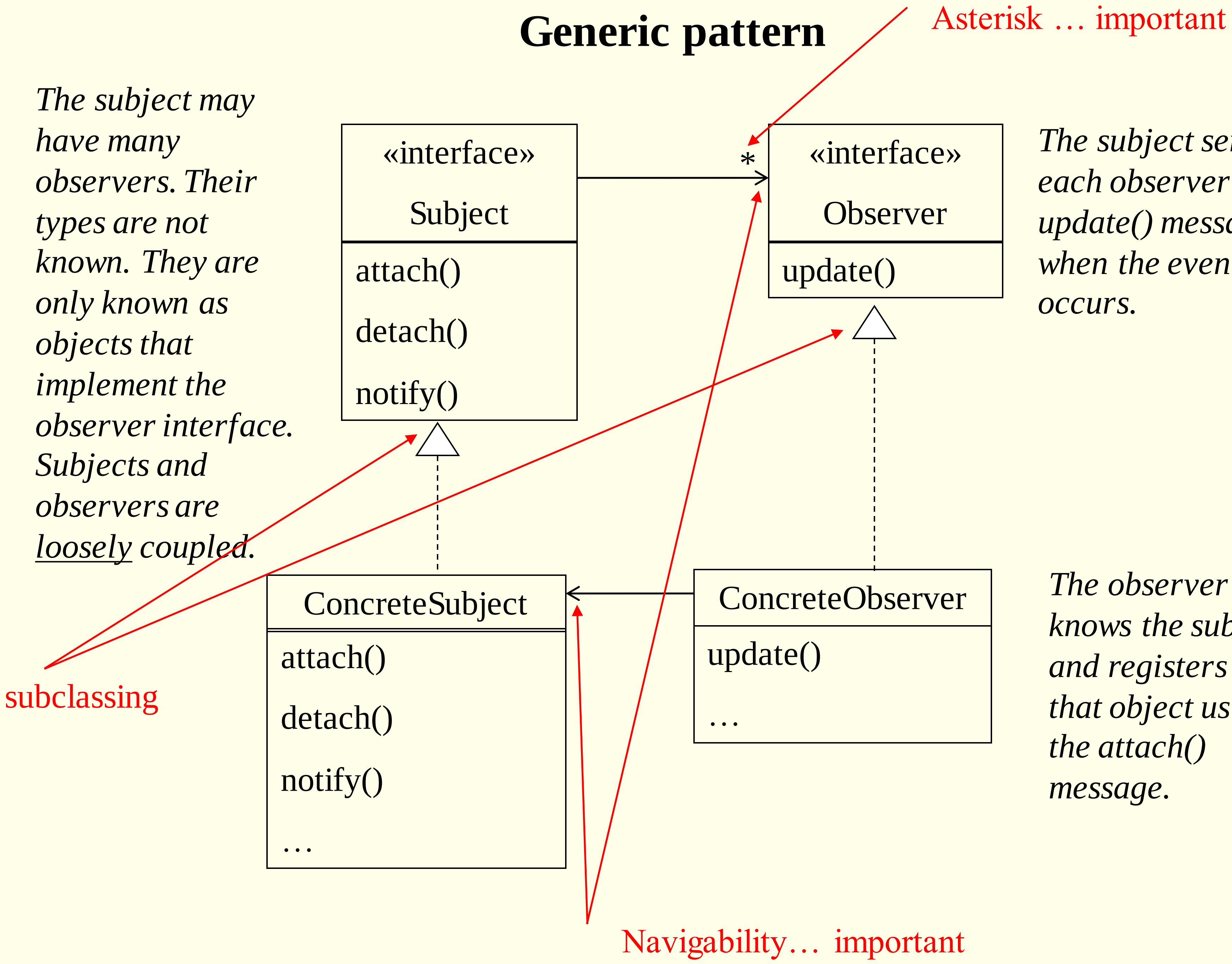
Different implementations may give different (but usually similar) names to the above methods

Generic pattern



Generic pattern

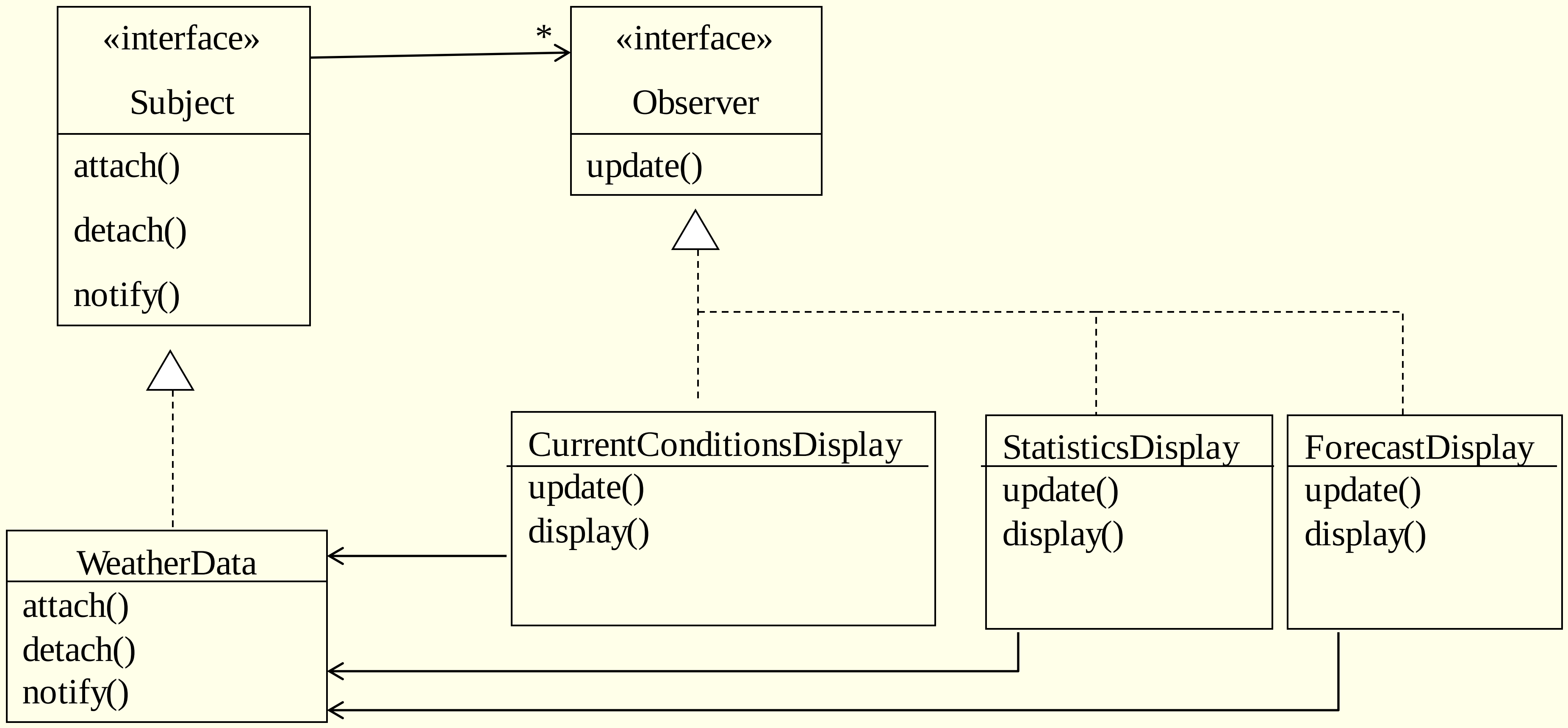
The subject may have many observers. Their types are not known. They are only known as objects that implement the observer interface. Subjects and observers are loosely coupled.



The subject sends each observer the update() message when the event occurs.

The observer knows the subject and registers with that object using the attach() message.

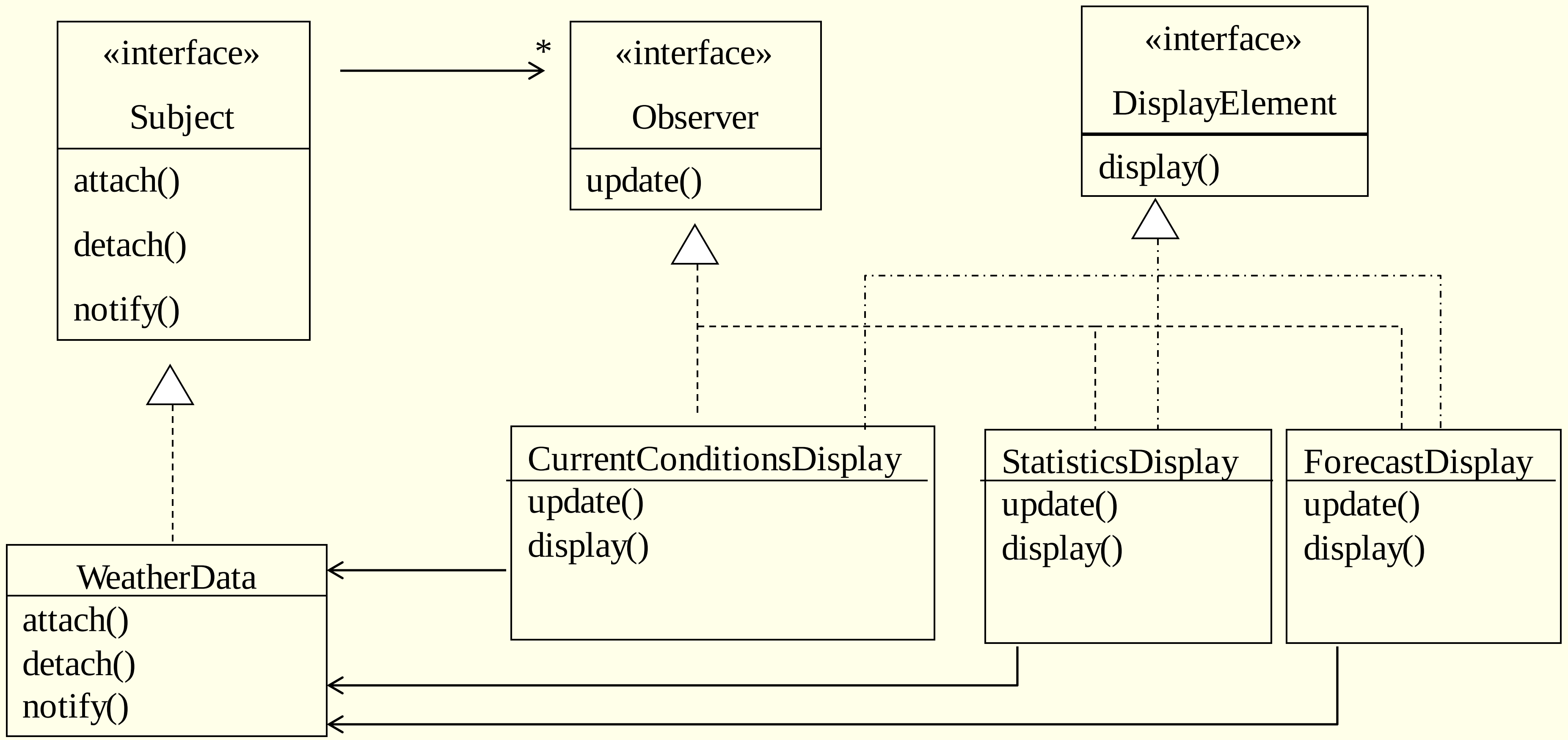
Weather Station Example



The class diagram

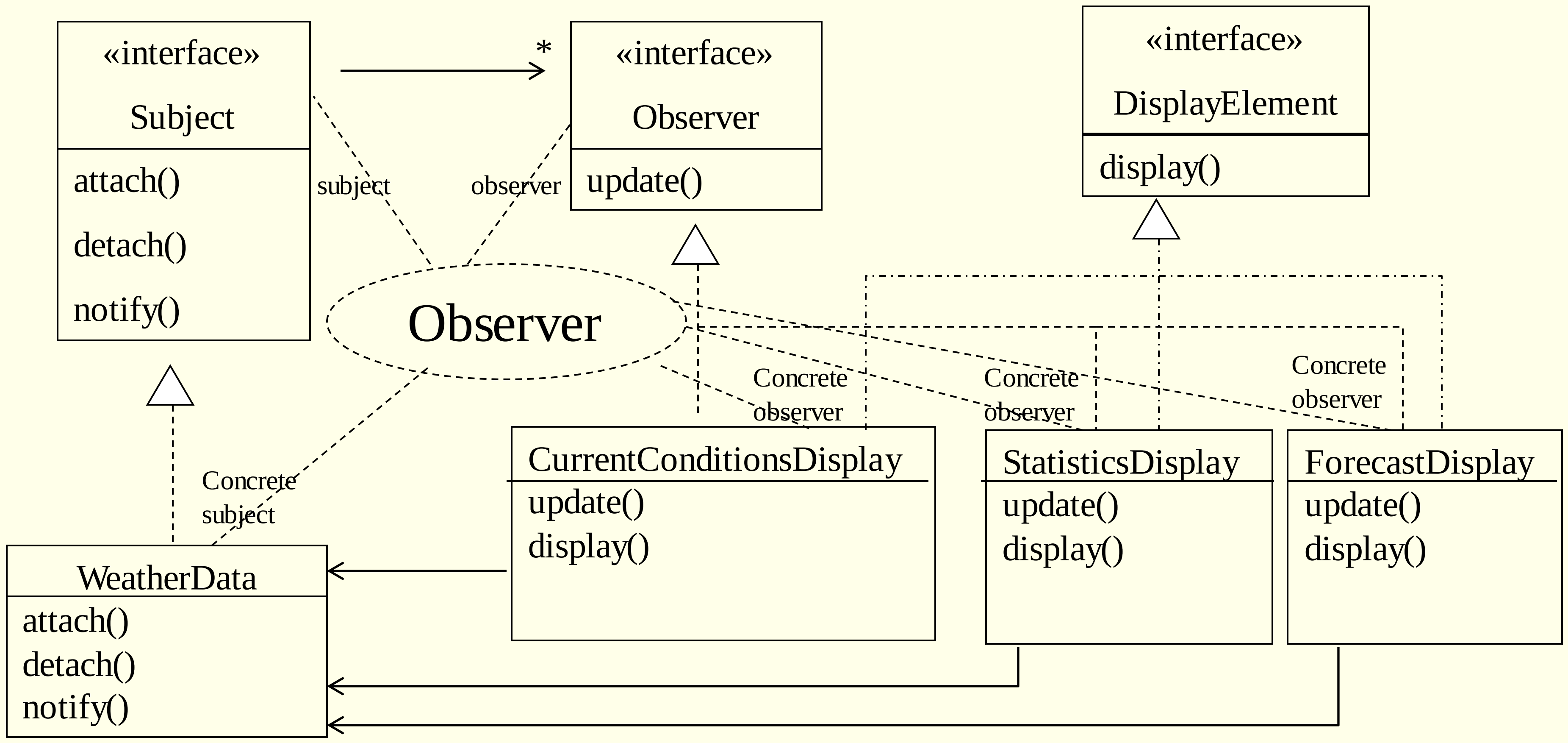
Weather Station Example

Another feature of this application – another use for an interface



The class diagram

Weather Station Example



This class diagram is adorned with a collaboration showing the pattern's participants

Weather Station Example

Consider the text example.

- Examine the code to ensure you understand how the observer pattern is implemented.
- Is it pushing or pulling data from the subject?

Suppose you are a developer and there is a requirement for a new type of observer (e.g. heat index display)

- What must you change in the existing code?
- What classes must you write? What methods do they have?

Draw sequence diagrams to illustrate behaviour. What messages are sent

- When an object registers?
- When an object unregisters?
- When the event of interest occurs?

Pages 64-71

Aside:

Java - consider the Observable class and the Observer interface.

- •Observable: http://java.sun.com/j2se/1.5.0/docs/api/java/util/Observable.html
- •Observer: http://java.sun.com/j2se/1.5.0/docs/api/java/util/Observer.html
- •Aside: How do we rewrite our weather station example to use these features of Java?

Other sources

Wikipedia

- Software design patterns
  - Observer

www.dofactory.com

- Reference Guides >> .NET Design Patterns
  - 23 Gang of Four (GoF) patterns
  - Definitions, generic UML diagram plus ‘real-world’ examples (in C#)