Padrões de Projeto de Software Orientados a Objetos

Tecnologia em Análise e Desenvolvimento de Sistemas

Paulo Mauricio Gonçalves Júnior

Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco

16 de março de 2018

Parte II

Observer

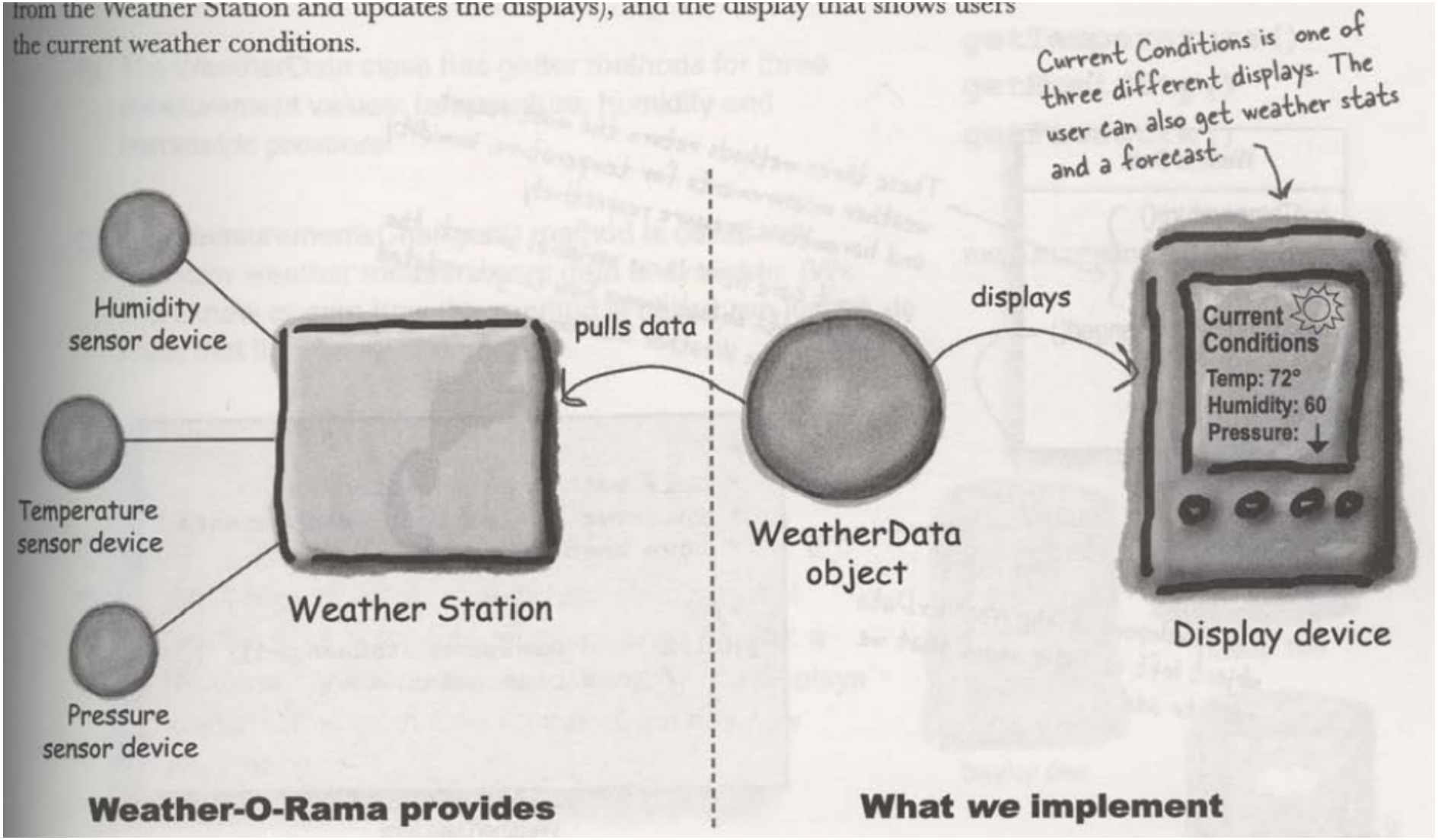
Introdução

- Notifique objetos toda vez que algo importante acontece.
- Objetos podem decidir em tempo de execução se eles querem continuar sendo informados.

Problema I

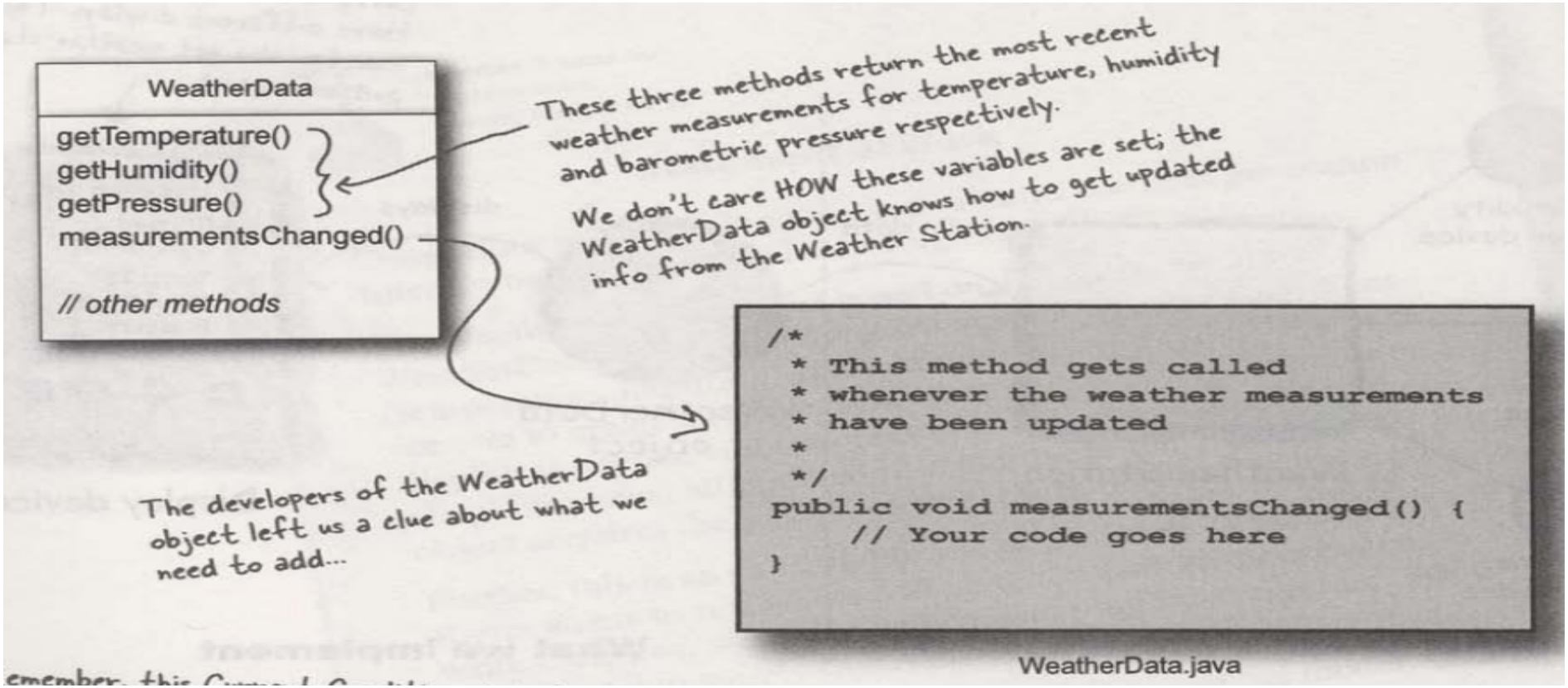
- Imagine que seja contratado para desenvolver um sistema que receberá dados de uma estação meteorológica, que informa temperatura, umidade e pressão atmosférica, e criará três telas: condições atuais, estatísticas do tempo, e previsão do tempo, tudo em tempo real.
- Deverá ainda ser possível que os clientes criem outras telas a partir dos dados da estação meteorológica e plugar à aplicação.

Problema II


from the Weather Station and updates the displays), and the display that shows users the current weather conditions.
Current Conditions is one of three different displays. The user can also get weather stats and a forecast.
Humidity sensor device
Temperature sensor device
Pressure sensor device
Weather Station
pulls data
WeatherData object
displays
Current Conditions
Temp: 72°
Humidity: 60
Pressure:
Display device
Weather-O-Rama provides
What we implement

Problema III

- Você recebe o arquivo WeatherData.java e terá que implementar um método dele.



Implementação inicial I

```
public class WeatherData {
  // Atributos

  public void measurementsChanged() {
    // Obtendo as medições mais atuais
    float temp = getTemperature();
    float humidity = getHumidity();
    float pressure = getPressure();
    // Atualizando as telas
    currentConditionsDisplay.update(temp, humidity, pressure);
    statisticsDisplay.update(temp, humidity, pressure);
    forecastDisplay.update(temp, humidity, pressure);
  }

  // Outros métodos
}
```

Problemas da implementação

- Codificação para implementações concretas: não podemos adicionar ou remover outras telas sem alterar o código.
- Todas as atualizações utilizam uma mesma interface:

```
update(temp, humidity, pressure).
```

Padrão Observer I

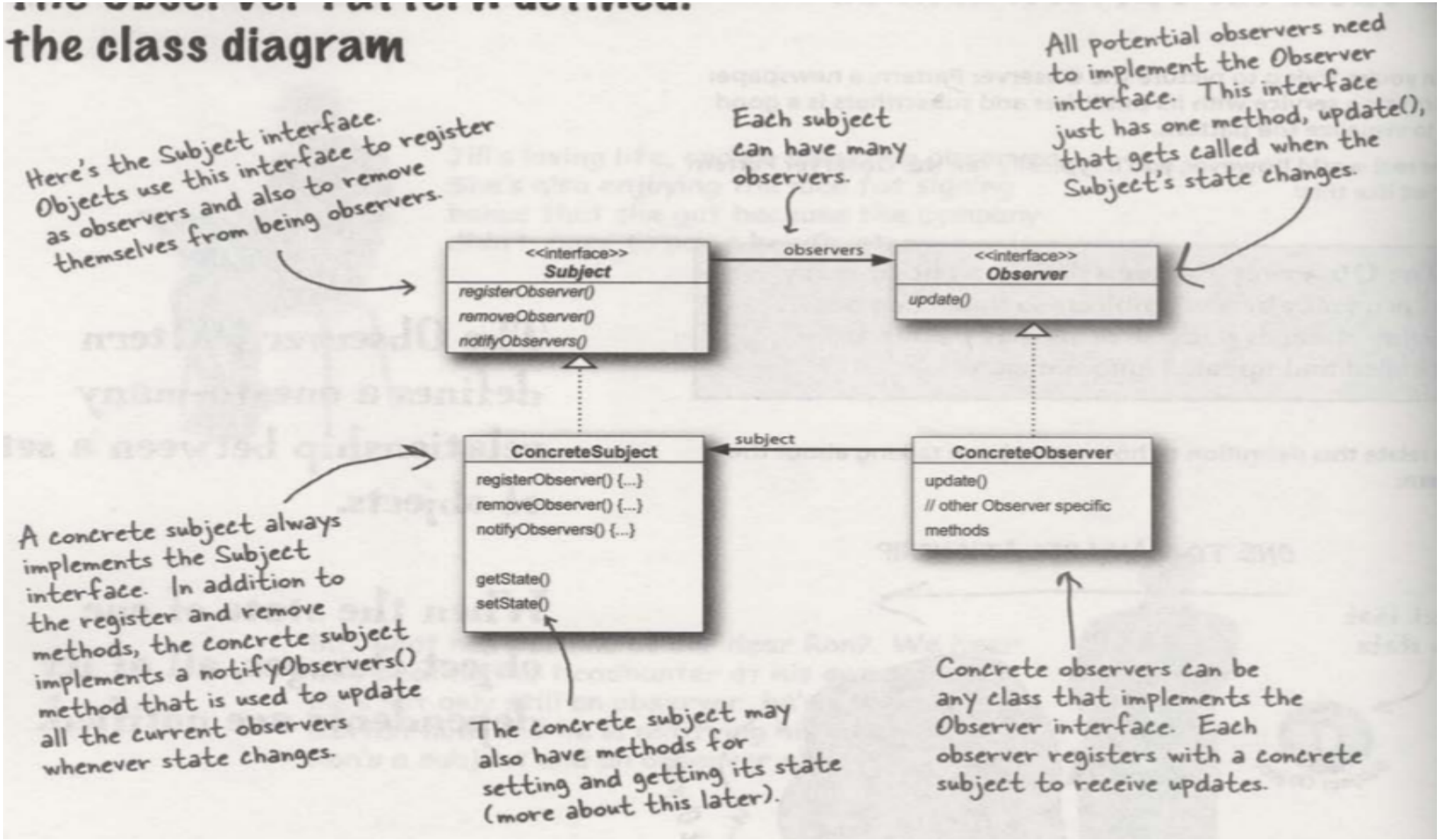
- Você conhece como uma assinatura de jornal ou revista funciona:
  1. Uma editora começa a publicar jornais.
  2. Você assina o jornal e toda nova edição é enviada para você. Enquanto você é assinante você recebe novos jornais.
  3. Você cancela sua assinatura quando não quer mais jornais e eles param de serem enviados para você.
  4. Enquanto a editora continua em funcionamento, pessoas, hotéis, empresas aéreas, ficam constantemente assinando e cancelando assinaturas do jornal.

Padrão Observer II

Definição

O padrão Observer define uma dependência de um-para-muitos entre objetos de forma que quando o estado de um objeto muda, todos seus dependentes são notificados e atualizados automaticamente.

Padrão Observer III



O poder do baixo acoplamento

- A única coisa que o subject sabe dos observers é que eles implementam uma determinada interface.
- Podemos adicionar novos observers a qualquer momento.
- Nunca precisamos mudar o subject para adicionar novos tipos de observers.
- Podemos reusar subjects e observers de forma independente.
- Modificações no subject ou observers não afetam um ao outro.

Princípio de Projeto

Esforce-se em criar projetos com baixo acoplamento entre objetos que interagem.

Implementando a estação Weather I

```
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();
}

public class WeatherData implements Subject {
  private ArrayList observers;
  private float temperature;
  private float humidity;
  private float pressure;

  public WeatherData() {
```

Implementando a estação Weather II

```
    observers = new ArrayList();
}

public void registerObserver(Observer o) {
    observers.add(o);
}

public void removeObserver(Observer o) {
    observers.remove(o);
}

public void notifyObservers() {
    for(Observer o : observers) {
        observer.update(temperature, humidity, pressure);
    }
}

public void measurementsChanged() {
    notifyObservers();
}
```

Implementando a estação Weather III

```
  public void setMeasurements(float temperature, float humidity,
      float pressure) {
    this.temperature = temperature;
    this.humidity = humidity;
    this.pressure = pressure;
    measurementsChanged();
  }
}

public class CurrentConditionsDisplay implements Observer,
    DisplayElement {
  private float temperature;
  private float humidity;
  private float pressure;
  private Subject weatherData;

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

Implementando a estação Weather IV

```
  public void update(float temperature, float humidity, float
      pressure) {
    this.temperature = temperature;
    this.humidity = humidity;
    this.pressure = pressure;
    display();
  }

  public void display() {
    System.out.println("Current conditions: " + temperature + "F
        degrees and " + humidity + "% humidity");
  }
}

public class WeatherStation {
  public static void main(String[] args) {
    CurrentConditionsDisplay currentDisplay = new
        CurrentConditionsDisplay(weatherData);
    StatisticsDisplay statisticsDisplay = new StatisticsDisplay(
        weatherData);
```

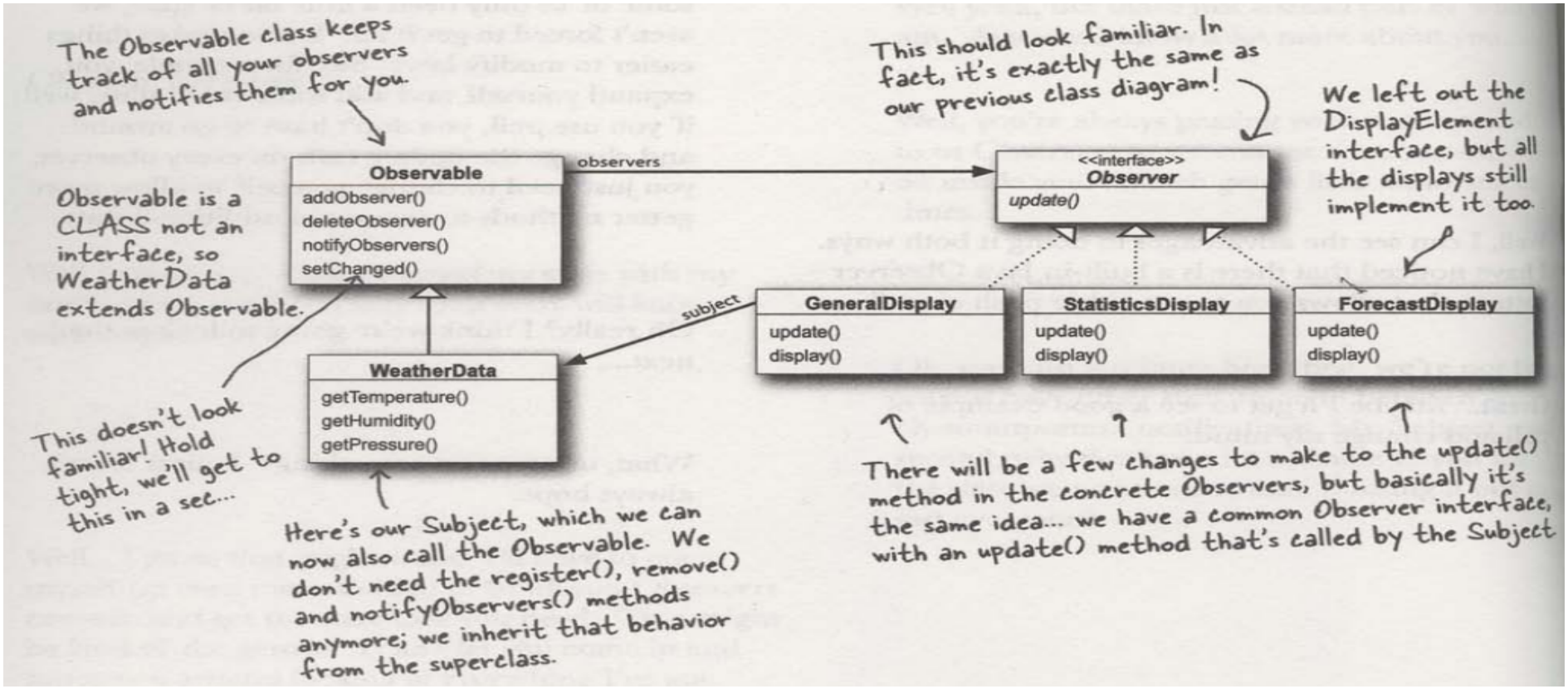
Implementando a estação Weather V

```
        ForecastDisplay forecastDisplay = new ForecastDisplay(
            weatherData);

        weatherData.setMeasurements(80, 65, 30.4f);
        weatherData.setMeasurements(82, 70, 29.2f);
        weatherData.setMeasurements(78, 90, 29.2f);
    }
}
```

Usando a API Java I

- A API Java oferece a interface Observer e a classe Observable.
- Podemos usar os estilos de atualização push ou pull para os observers.



Usando a API Java II

- Para um objeto se tornar um observer, implementamos a interface Observer e chamamos os métodos `addObserver` ou `deleteObserver` em um objeto Observable.
- Para a classe Observable enviar notificações, primeiramente extenderá a superclasse. Depois, chamar o método `setChanged()` para informar que o estado mudou. Então chamar um de dois métodos: `notifyObservers()` ou `notifyObservers(Object arg)`.
- O observer recebe as notificações através do método `update` da seguinte forma: `update(Observable o, Object arg)`.

Usando a API Java III

- O método `setChanged` serve para informar que o estado mudou e que o método `notifyObservers`, quando chamado, deve atualizar os observers. Se o método `notifyObservers` é chamado sem antes chamar `setChanged`, os observers não serão atualizados. Isso acontece para dar mais flexibilidade. Exemplo, se sua estação é muito sensível e a temperatura muda em décimos de grau, os observadores serão notificados a toda hora. Podemos querer gerar notificações apenas se a temperatura mudar mais de meio grau, chamando o método `setChanged`.

Usando a API Java IV

```
import    java.util.Observable;
import    java.util.Observer;

public class WeatherData extends Observable {
  private float temperature;
  private float humidity;
  private float pressure;

  public WeatherData() { }

  public void measurementsChanged() {
    setChanged();
    notifyObservers();
  }

  public void setMeasurements(float temperature, float humidity,
      float pressure) {
    this.temperature = temperature;
    this.humidity = humidity;
    this.pressure = pressure;
    measurementsChanged();
```

Usando a API Java V

```
  }

  public float getTemperature() {
    return temperature;
  }

  public float getHumidity() {
    return humidity;
  }

  public float getPressure() {
    return pressure;
  }
}

import    java.util.Observable;
import    java.util.Observer;
public class CurrentConditionsDisplay implements Observer,
    DisplayElement {
  Observable observable;
  private float temperature;
```

Usando a API Java VI

```
  private float humidity;

  public CurrentConditionsDisplay(Observable observable) {
    this.observable = observable;
    observable.addObserver(this);
  }
  public void update(Observable obs, Object arg) {
    if(obs instanceof WeatherData) {
      WeatherData weatherData = (WeatherData) obs;
      this.temperature = weatherData.getTemperature();
      this.humidity = weatherData.getHumidity();
      display();
    }
  }

  public void display() {
    System.out.println("Current conditions: " + temperature + "F
        degrees and " + humidity + "% humidity");
  }
}
```