2. 옵저버 패턴

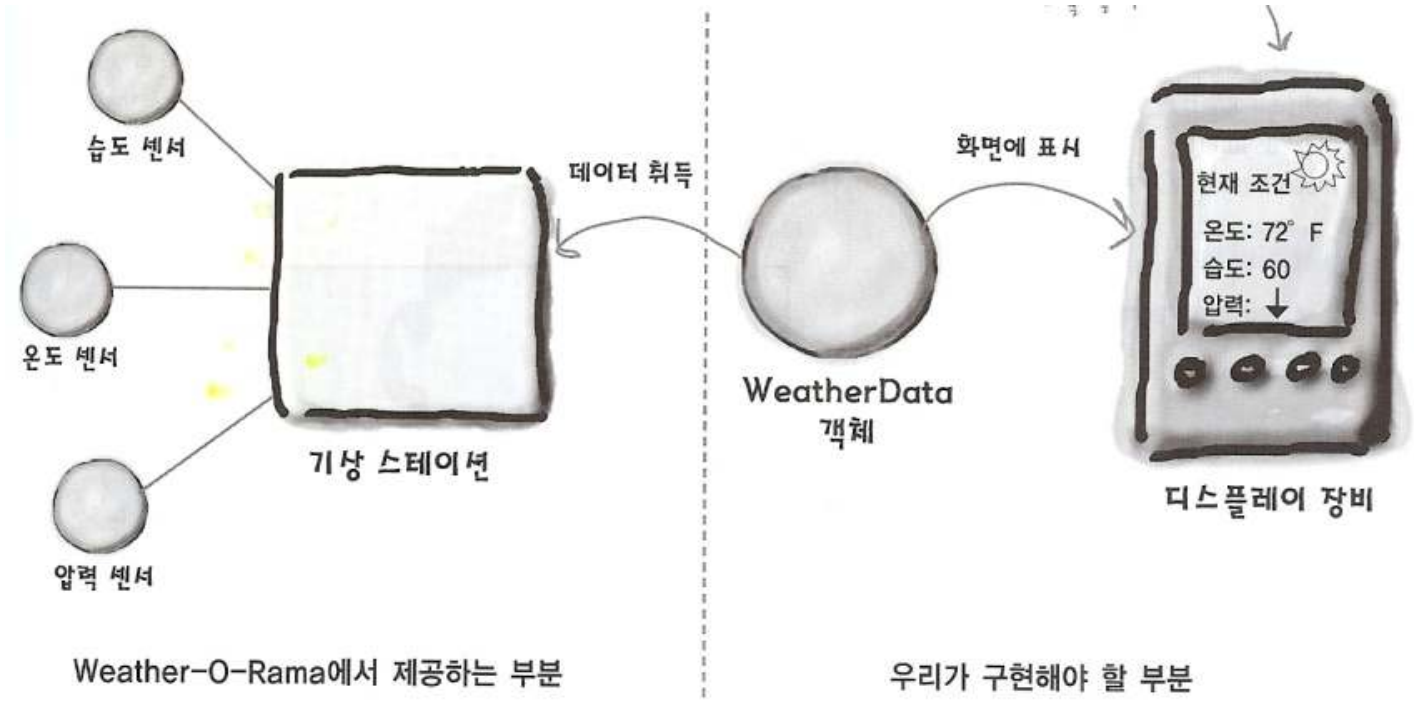
기상 모니터링 애플리케이션

(Observer Pattern)

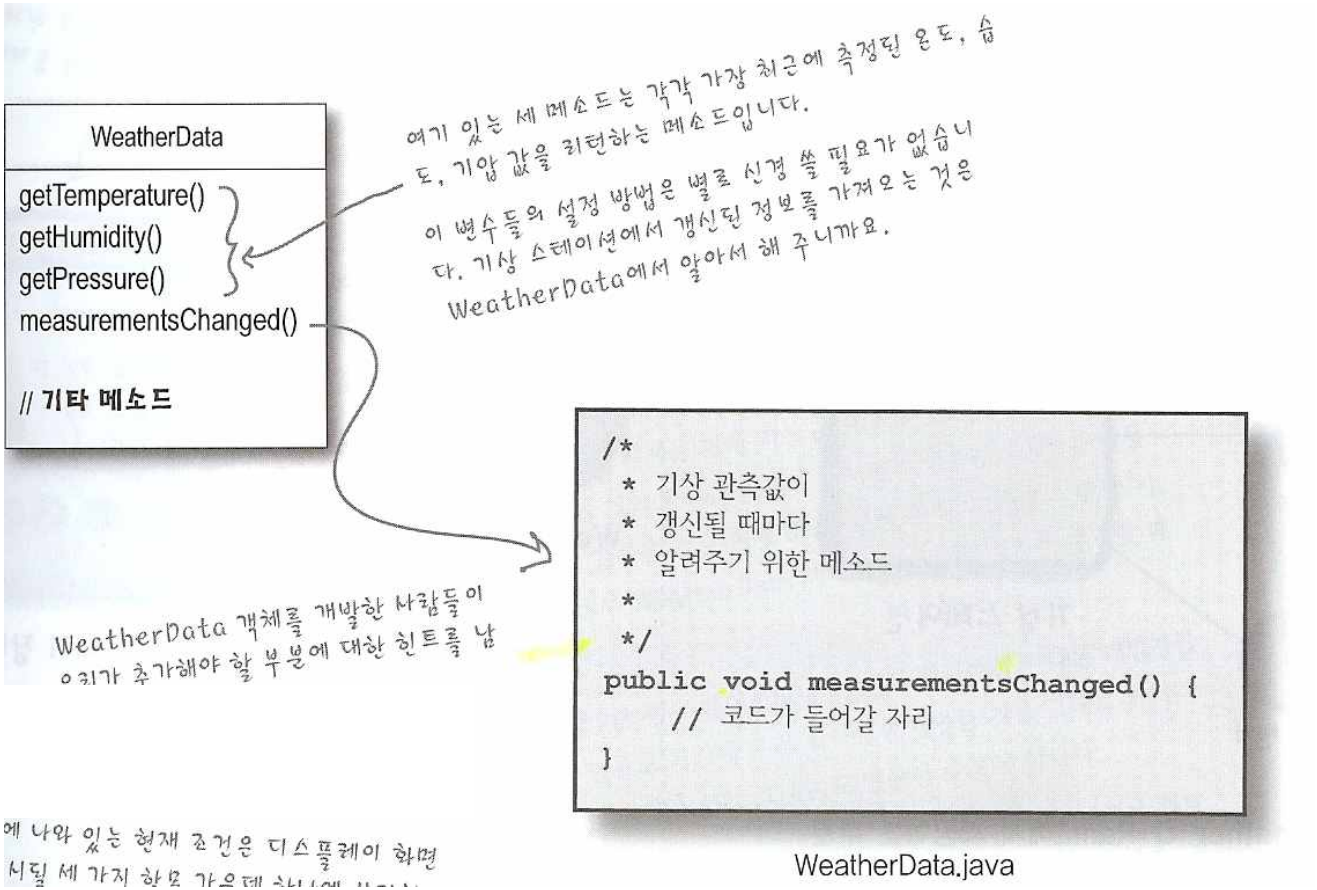
기상 모니터링 애플리케이션

- 개요
  - WeatherData 객체를 바탕으로 만들 예정
    - 현재 기상 조건(기온, 습도, 기압)을 추적
  - 화면에 현재 기상 조건, 기상 통계, 기상 예보를 표시
    - WeatherData 객체에서 최신 측정치를 수집할 때마다 실시간으로 갱신해야 함.
  - 시스템이 확장 가능해야 함.
    - 다른 개발자들이 별도의 디스플레이 항목을 만들 수 있도록 해야 함.
    - 사용자들이 애플리케이션에 마음대로 디스플레이 항목을 추가/제거할 수 있어야 함.

1차 모델링


습도 센서
온도 센서
압력 센서
기상 스테이션
데이터 취득
WeatherData
객체
화면에 표시
현재 조건
온도: 72° F
습도: 60
압력:
디스플레이 장비
Weather-O-Rama에서 제공하는 부분
우리가 구현해야 할 부분

WeatherData 클래스 개요



WeatherData.java

WeatherData 구현 (model1)

- measurementsChanged() 메소드 구현 예

```
public class WeatherData {

    // 인스턴스 변수 선언

    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);
    }

    // 기타 메소드
```

(이미 구현되어 있는) WeatherData의 게터 메소드를 써서 최신 측정값을 가져옵니다.

디스플레이 갱신

각 디스플레이 항목을 불러서 디스플레이를 갱신하도록 합니다. 이때 최신 측정값을 전달해 주죠.

구현한 코드의 문제점

```
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);
}
```

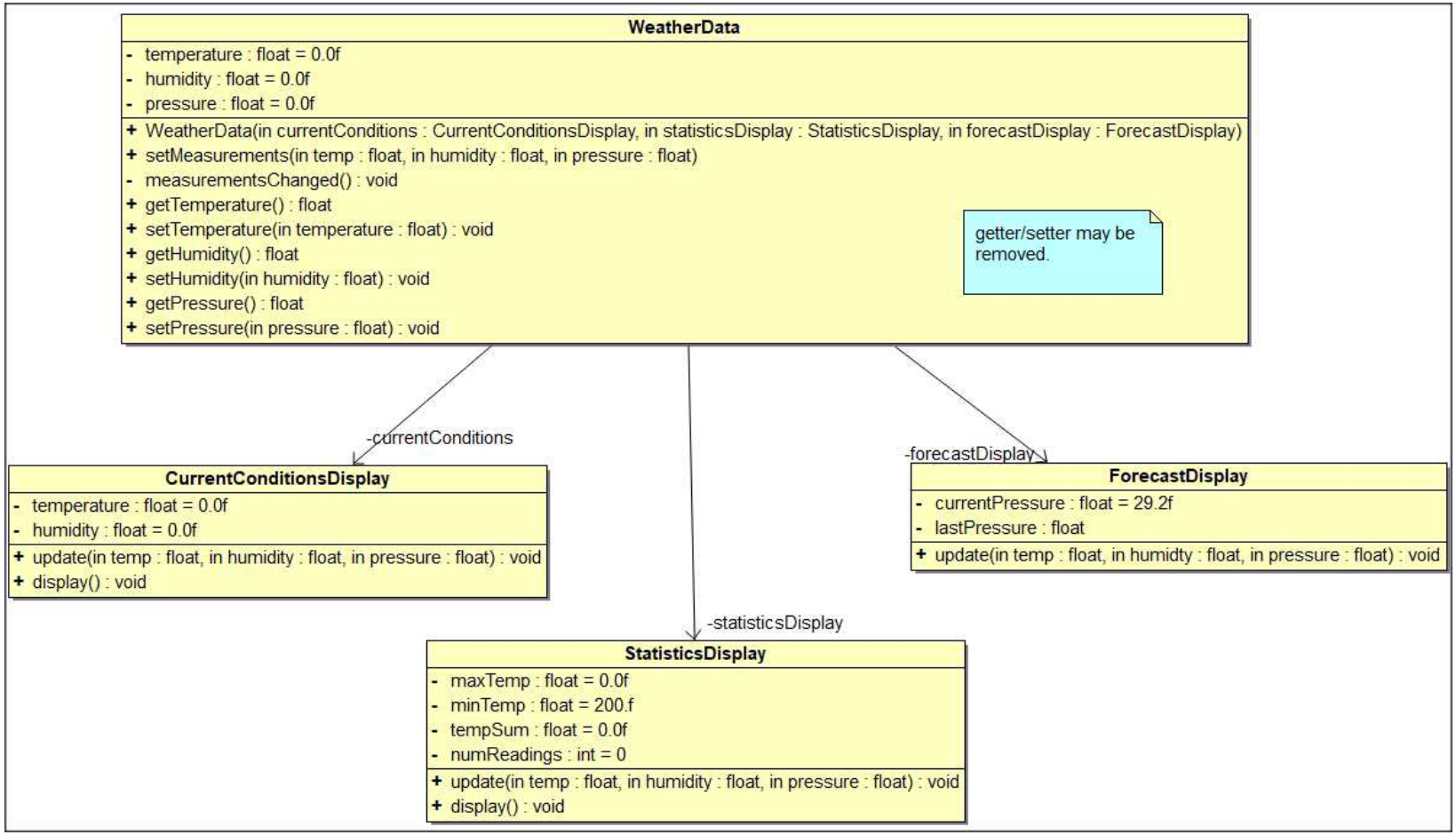
바뀔 수 있는 부분. 캡슐화해야 합니다.

디스플레이 항목과 데이터를 주고받는 데 있어서 적어도 공통된 인터페이스를 사용하고 있는 것 같긴 하군요. 모두 기온, 습도, 기압 값을 받아들이는 update() 메소드를 가지고 있으니까요.

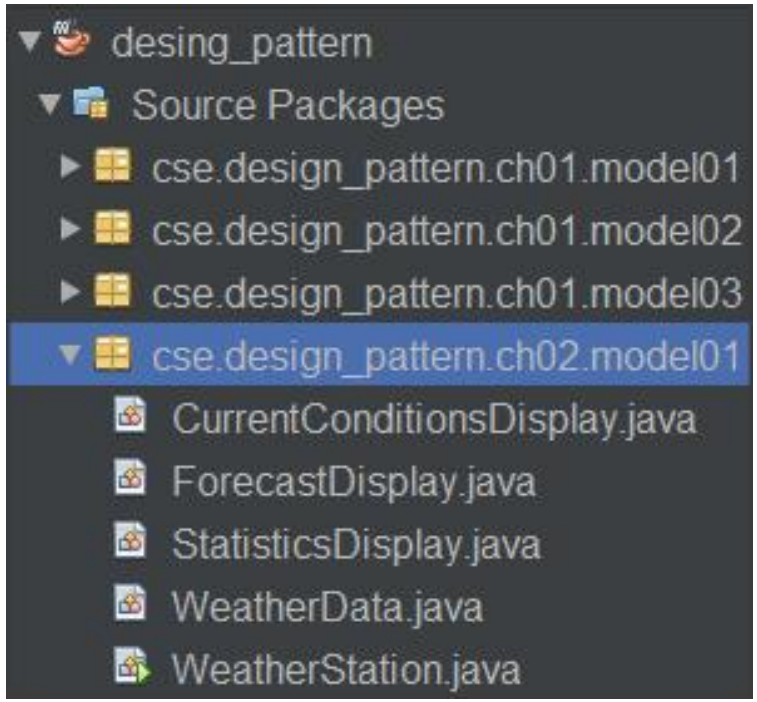
구체적인 구현에 맞춰서 코딩했기 때문에 프로그램을 고치지 않고는 다른 디스플레이 항목을 추가/제거할 수 없습니다.

클래스 다이어그램 (model1)

단점: 새로운 디스플레이를 추가하려면
WeatherData 클래스를 수정해야 함.



프로젝트 파일


desing_pattern
Source Packages
cse.design_pattern.ch01.model01
cse.design_pattern.ch01.model02
cse.design_pattern.ch01.model03
cse.design_pattern.ch02.model01
CurrentConditionsDisplay.java
ForecastDisplay.java
StatisticsDisplay.java
WeatherData.java
WeatherStation.java

CurrentConditionsDisplay.java

```
6   package cse.design_pattern.ch02.model01;
7
8   /**...4 lines */
12  public class CurrentConditionsDisplay {
13      private float temperature  = 0.0f;
14      private float humidity  = 0.0f;
15
16      public void update(float temp, float humidity, float pressure) {
17          this.temperature = temp;
18          this.humidity = humidity;
19          display();
20      }
21
22      public void display() {
23          System.out.println("Current conditions: " + temperature
24                  + "F degress and " + humidity + "% humidity");
25      }
26  }
```

StatisticsDisplay.java

```
6   package cse.design_pattern.ch02.model01;
7
8   /**...4 lines */
12  public class StatisticsDisplay {
13
14      private float maxTemp = 0.0f;
15      private float minTemp = 200;
16      private float tempSum = 0.0f;
17      private int numReadings;
18
19      public void update(float temp, float humidity, float pressure) {
20          tempSum += temp;
21          numReadings++;
22
23          if (temp > maxTemp) {
24              maxTemp = temp;
25          }
26
27          if (temp < minTemp) {
28              minTemp = temp;
29          }
30          display();
31      }
32
33      public void display() {
34          System.out.println("Avg/Max/Min temperature = " + (tempSum / numReadings)
35                  + "/" + maxTemp + "/" + minTemp);
36      }
37  }
```

ForecastDisplay.java

```
6   package cse.design_pattern.ch02.model01;
7
8   /**...4 lines */
12  public class ForecastDisplay {
13
14      private float currentPressure = 29.92f;
15      private float lastPressure;
16
17      public void update(float temp, float humidity, float pressure) {
18          lastPressure = currentPressure;
19          currentPressure = pressure;
20          display();
21      }
22
23      public void display() {
24          System.out.print("Forecast: ");
25          if (currentPressure > lastPressure) {
26              System.out.println("Improving weather on the way!");
27          } else if (currentPressure == lastPressure) {
28              System.out.println("More of the same");
29          } else if (currentPressure < lastPressure) {
30              System.out.println("Watch out for cooler, rainy weather");
31          }
32      }
33  }
```

WeatherData.java

```
6   package cse.design_pattern.ch02.model01;
7
8   /**...4 lines */
    public class WeatherData {
13
14      private CurrentConditionsDisplay currentConditions;
15      private StatisticsDisplay statisticsDisplay;
16      private ForecastDisplay forecastDisplay;
17
18      private float temperature = 0.0f;
19      private float humidity = 0.0f;
20      private float pressure = 0.0f;
21
22      public WeatherData(CurrentConditionsDisplay currentConditions,
23              StatisticsDisplay statisticsDisplay, ForecastDisplay forecastDisplay) {
24          this.currentConditions = currentConditions;
25          this.statisticsDisplay = statisticsDisplay;
26          this.forecastDisplay = forecastDisplay;
27      }
28
29      public void setMeasurements(float temp, float humidity, float pressure) {
30          setTemperature(temp);
31          setHumidity(humidity);
32          setPressure(pressure);
33
34          measurementsChanged();
35      }
```

구현에 대해 프로그래밍!
→ 수정하기 어려움.

```
37  /**
38   * setMeasurements()에서 호출하므로  public --> private으로 변경
39   */
40  private void measurementsChanged() {
41      float temp = getTemperature();
        float humidity = getHumidity();
        float pressure = getPressure();
44
45      currentConditions.update(temp, humidity, pressure);
46      statisticsDisplay.update(temp, humidity, pressure);
47      forecastDisplay.update(temp, humidity, pressure);
48  }
49
50  public float getTemperature() {
51      return temperature;
52  }
53
54  public void setTemperature(float temperature) {
55      this.temperature = temperature;
56  }
```

```
58      public float getHumidity() {
59          return humidity;
60      }
61
62      public void setHumidity(float humidity) {
63          this.humidity = humidity;
64      }
65
66      public float getPressure() {
67          return pressure;
68      }
69
70      public void setPressure(float pressure) {
71          this.pressure = pressure;
72      }
73
74  }
```

WeatherStation.java

```
5   package cse.design_pattern.ch02.model01;
6
7   /**...4 lines */
    public class WeatherStation {
12
13      /**
14       * @param args the command line arguments
15       */
16      public static void main(String[] args) {
17          // TODO code application logic here
18
19          CurrentConditionsDisplay currentDisplay = new CurrentConditionsDisplay();
20          StatisticsDisplay statisticsDisplay = new StatisticsDisplay();
21          ForecastDisplay forecastDisplay = new ForecastDisplay();
22
23          WeatherData weatherData = new WeatherData(currentDisplay, statisticsDisplay,  forecastDisplay);
24
25          weatherData.setMeasurements(80, 65, 30.4f);
26          System.out.println("------------------------------");
27          weatherData.setMeasurements(82, 70, 29.2f);
28          System.out.println("------------------------------");
29          weatherData.setMeasurements(78, 90, 29.2f);
30      }
31
32  }
```

실행 결과

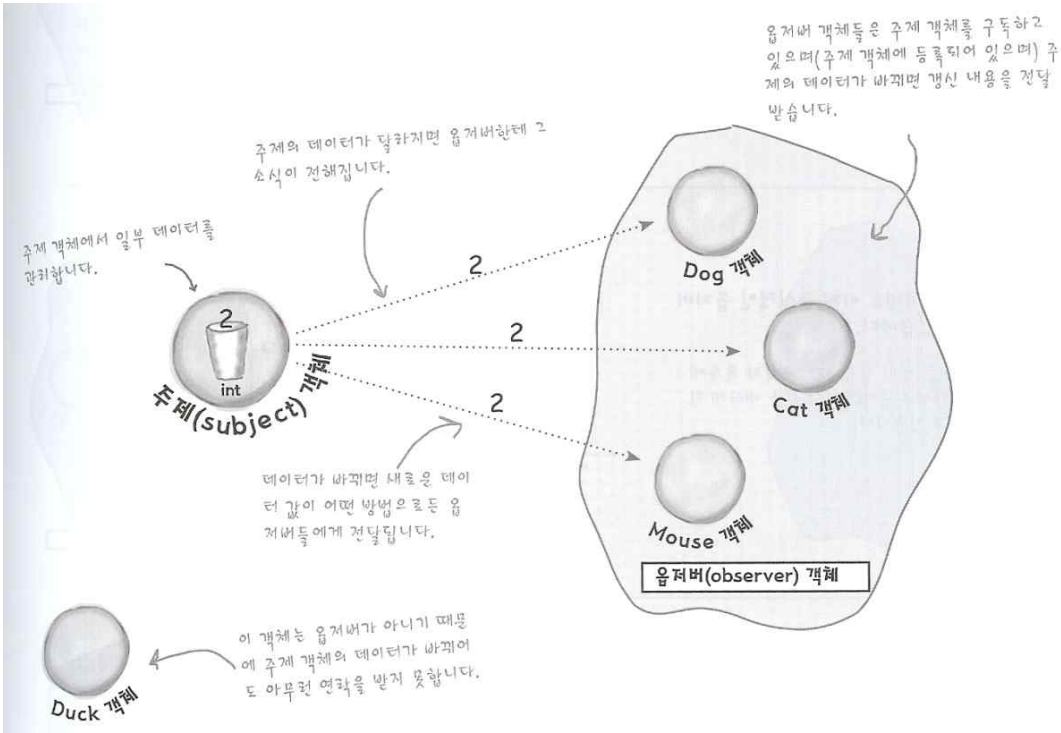
```
Building desing_pattern 1.0-SNAPSHOT
--------------------------------------------------------------------------------


--- exec-maven-plugin:1.2.1:exec (default-cli) @ desing_pattern ---
Current conditions: 80.0F degress and 65.0% humidity
Avg/Max/Min temperature = 80.0/80.0/80.0
Forecast: Improving weather on the way!
---------------------------------
Current conditions: 82.0F degress and 70.0% humidity
Avg/Max/Min temperature = 81.0/82.0/80.0
Forecast: Watch out for cooler, rainy weather
---------------------------------
Current conditions: 78.0F degress and 90.0% humidity
Avg/Max/Min temperature = 80.0/82.0/78.0
Forecast: More of the same
--------------------------------------------------------------------------------
```

옵저버 패턴 적용 (MODEL2)

Observer Pattern

- 출판사 + 구독자 = 옵저버 패턴
- 출판사 → 주제 (subject)
- 구독자 → 옵저버 (observer)

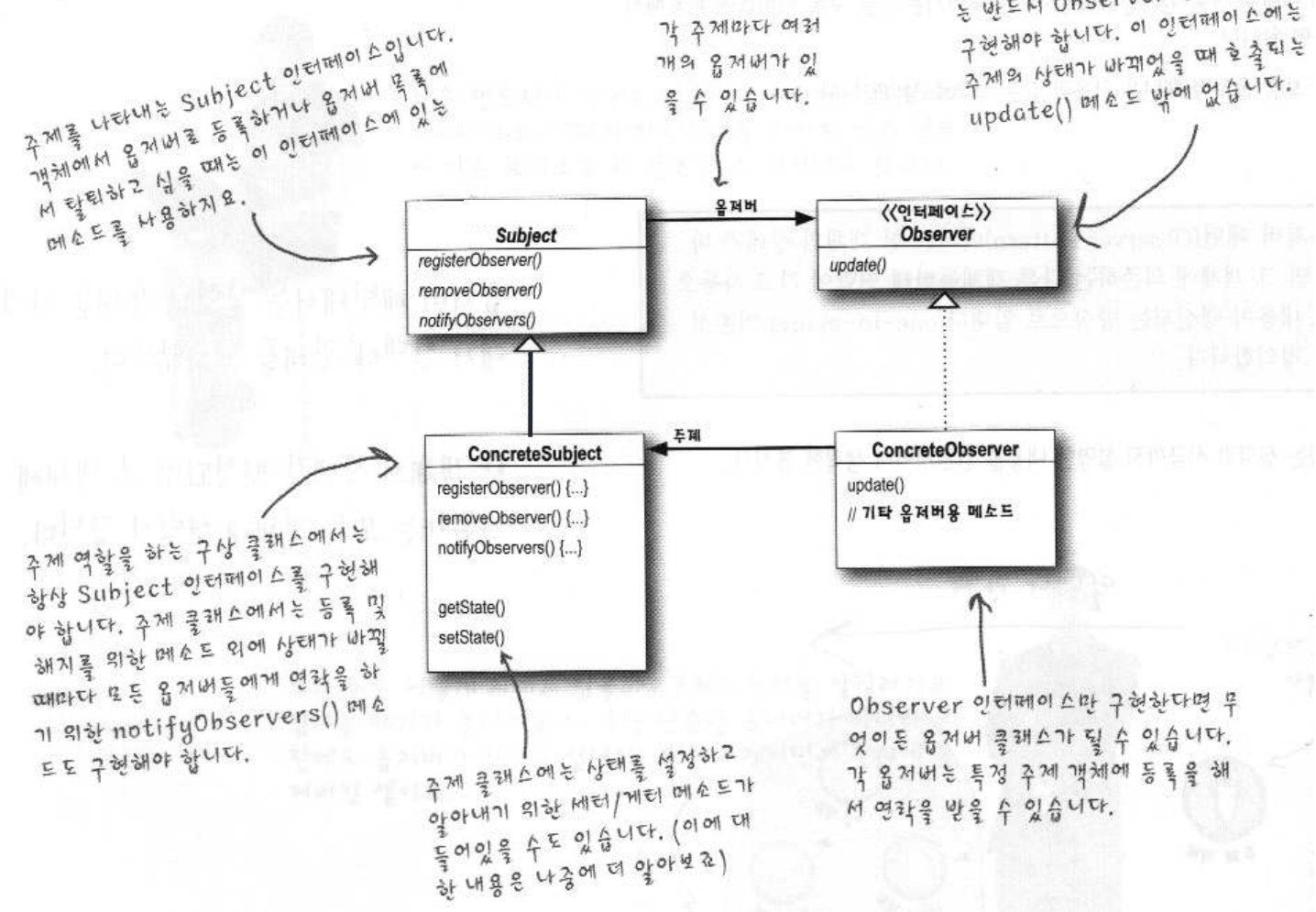


p.87

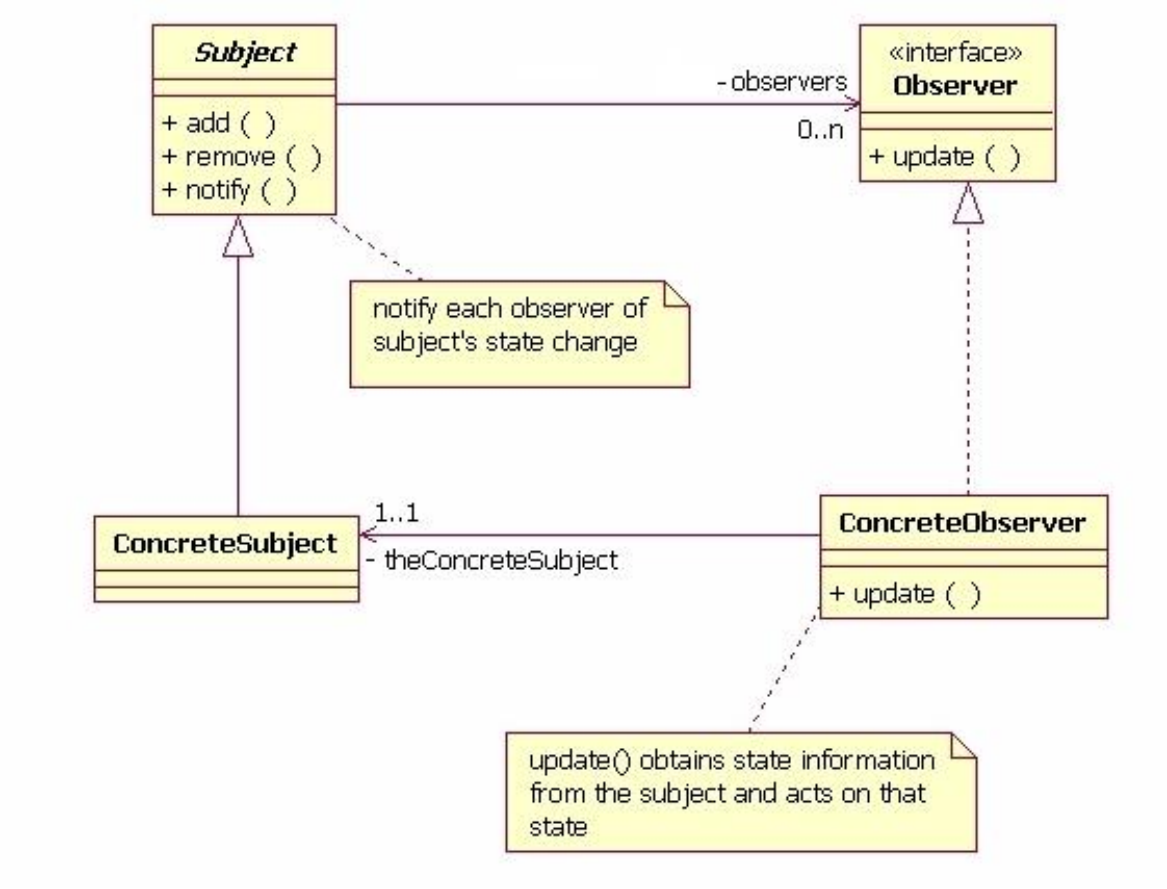
- 옵저버 패턴의 정의
  - 한 객체의 상태가 바뀌면 그 객체에 의존하는 다른 객체들한테 연락이 가고 자동으로 내용이 갱신되는 방식
  - 일대다 의존성 (one-to-many dependency)
- 주제와 옵저버가 느슨하게 결합되어 있는 (loosely coupled) 객체 디자인
  - 변경 사항 발생시 쉽게 처리할 수 있는 유연한 객체지향 시스템 구축 가능

이름	**Observer(Publish-Subscribe)**
문제	서로 다른 종류의 구독자(subscriber) 객체는 발행자(publisher) 객체의 상태 변화나 이벤트에 관심을 가지며, 발행자가 이벤트를 생성시키면 이벤트에 대해서 자신의 고유한 방식으로 반응하기를 원한다. 더구나, 발행자는 구독자와 낮은 결합도를 유지하고자 한다. 무엇을 해야 하는가?
해결책 (조언)	"구독자" 또는 "청취자(listener)" 인터페이스를 정의하고 구독자 객체는 이 인터페이스를 구현한다. 발행자 객체는 이벤트에 관심이 있는 구독자를 동적으로 등록할 수 있고 이벤트가 발생하면 구독자에게 통보할 수 있다.

옵저버 패턴: 클래스 다이어그램



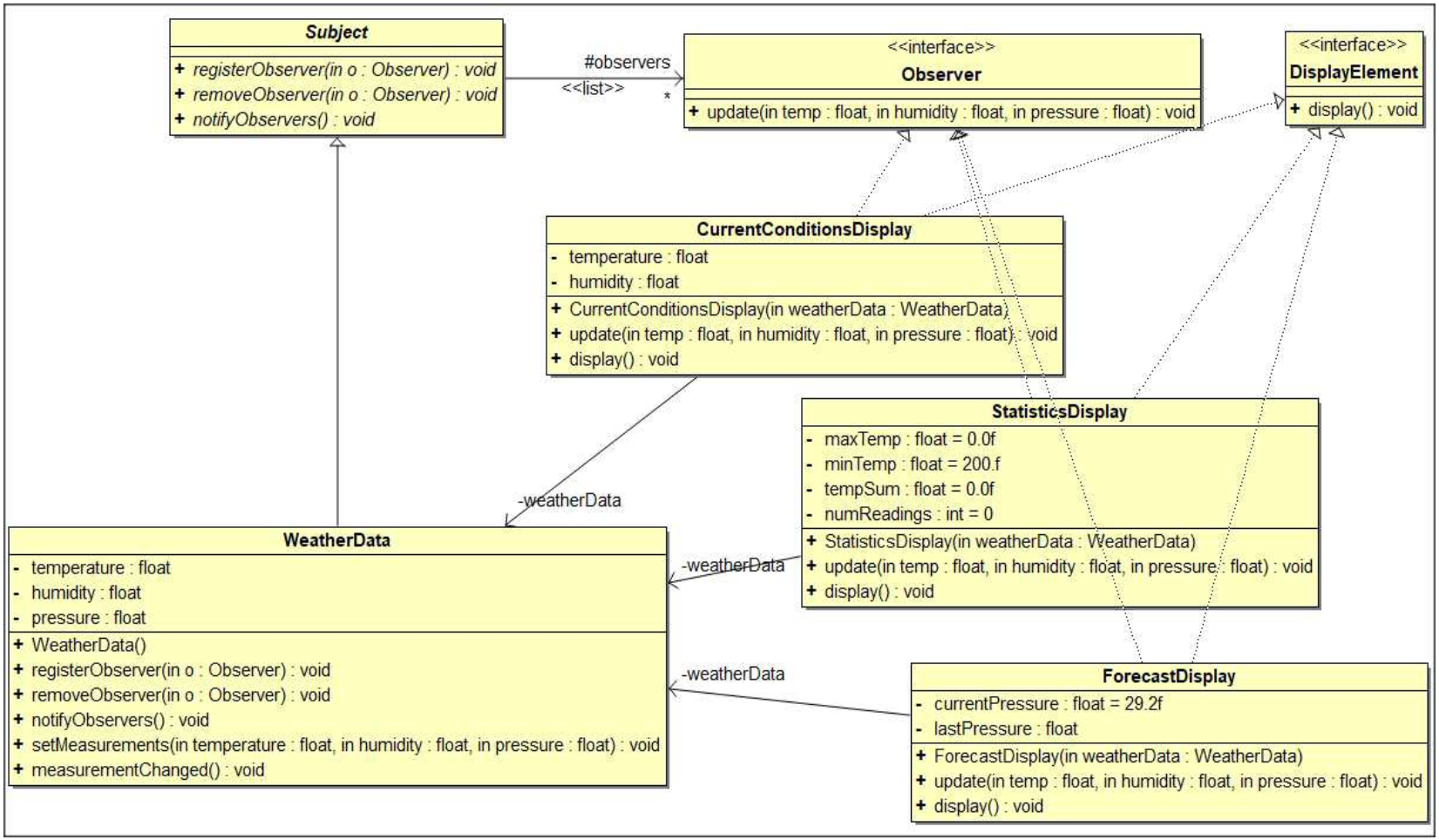
p.88


Subject
+ add ( )
+ remove ( )
+ notify ( )
-observers
0..n
«interface»
Observer
+ update ( )
notify each observer of subject's state change
ConcreteSubject
1..1
- theConcreteSubject
ConcreteObserver
+ update ( )
update() obtains state information from the subject and acts on that state

디자인 원칙

- (Loose Coupling) 서로 상호작용을 하는 객체 사이에서는 가능하면 느슨하게 결합하는 디자인을 사용해야 한다.
  - 주제가 옵저버에 대해서 아는 것은 옵저버가 특정 인터페이스(Observer interface)를 구현한다는 것뿐이다.
  - 옵저버는 언제든지 새로 추가할 수 있다.
  - 새로운 형식의 옵저버를 추가하려고 할 때도 주제를 전혀 변경할 필요가 없다.
  - 주제와 옵저버는 서로 독립적으로 재사용할 수 있다.
  - 주제나 옵저버가 바뀌더라도 서로한테 영향을 미치지 않는다.

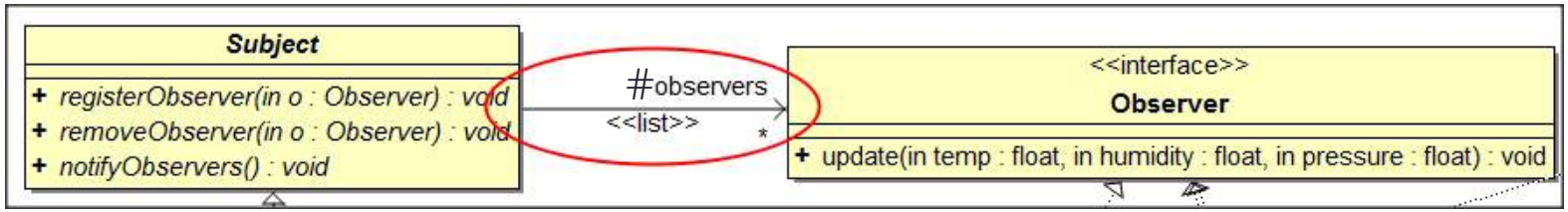
기상 스테이션 모델링 (2차)

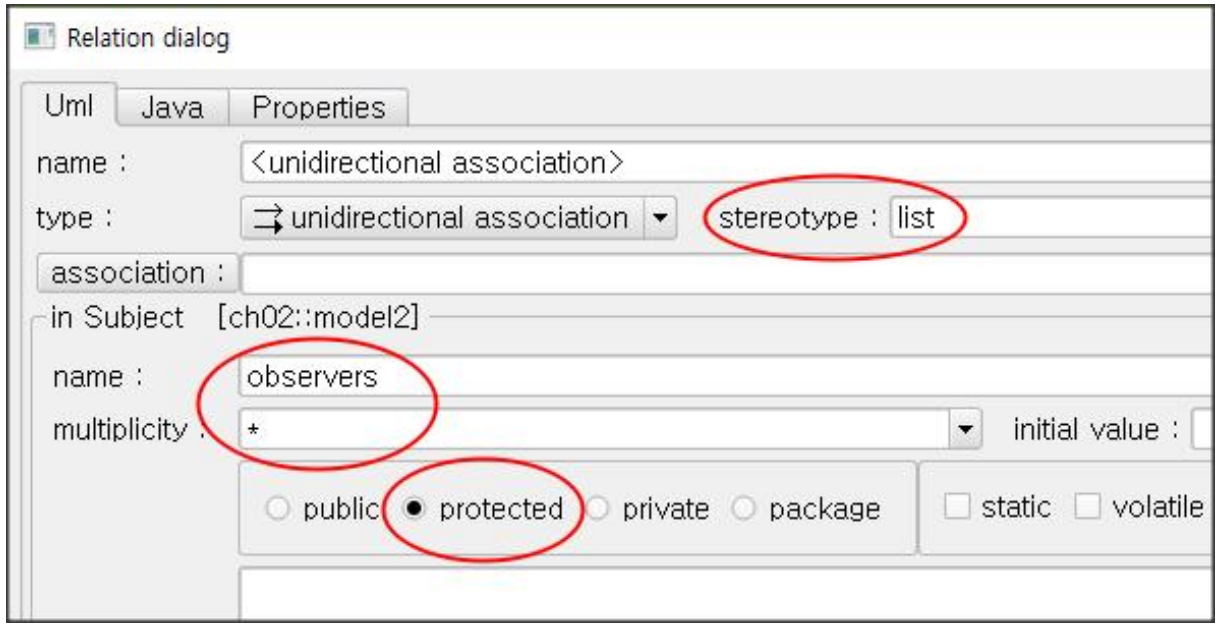


p.93

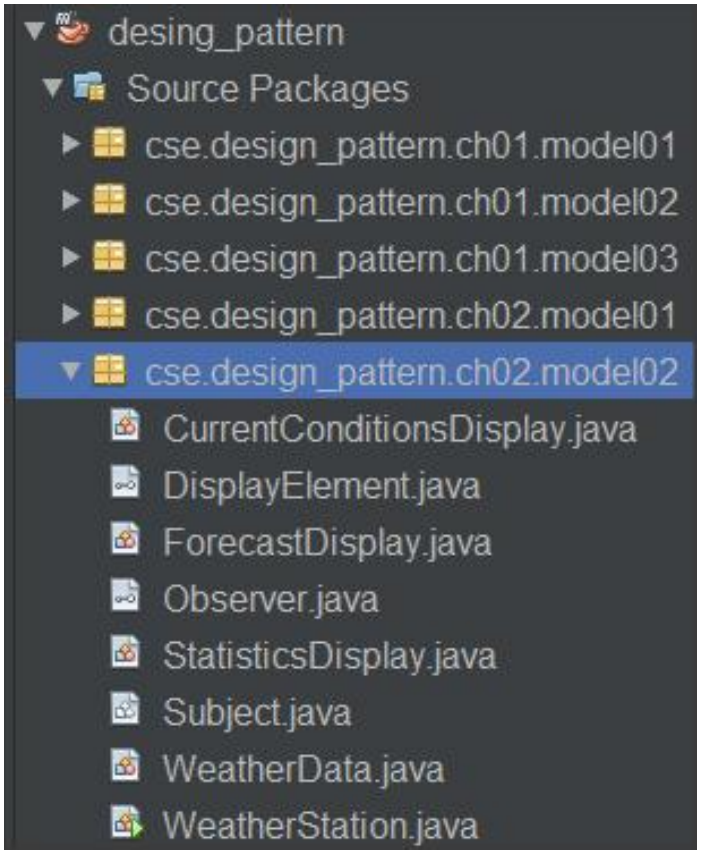
실습 (MODEL2)

기상 스테이션 모델링 (실습용)





프로젝트 파일


desing_pattern
Source Packages
cse.design_pattern.ch01.model01
cse.design_pattern.ch01.model02
cse.design_pattern.ch01.model03
cse.design_pattern.ch02.model01
cse.design_pattern.ch02.model02
CurrentConditionsDisplay.java
DisplayElement.java
ForecastDisplay.java
Observer.java
StatisticsDisplay.java
Subject.java
WeatherData.java
WeatherStation.java

Subject.java

```
1   package cse.design_pattern.ch02.model02;
2
3   import java.util.List;
4
    public abstract class Subject {
6     //protected Vector<Observer> observers;
      protected List<Observer> observers;
8
      public abstract void registerObserver(Observer o);
      public abstract void removeObserver(Observer o);
      public abstract void notifyObservers();
12  }
```

교재에서는 인터페이스로 구현하고 있으나, 클래스 다이어그램처럼 observers 연관을 Subject와 Obeserver 간에 두기 위해서는 추상 클래스로 정의해야 함.

p.94

WeatherData.java

```
1   package cse.design_pattern.ch02.model02;
2
3   import java.util.LinkedList;
4
5   public class WeatherData extends Subject {
6
7       private float temperature;
8       private float humidity;
9       private float pressure;
10
11      public WeatherData() {
12          observers = new LinkedList<>();
13
14      }
15
16      public void registerObserver(Observer o) {
17          observers.add(o);
18      }
19
20      public void removeObserver(Observer o) {
21          int i = observers.indexOf(o);
22          if (i >= 0) {
23              observers.remove(i);
24          }
25      }
```

p.95

- (중요!) for문 변형

```
28  public void notifyObservers() {
29      /*
30      for (int i = 0; i < observers.size(); i++) {
31          Observer o = (Observer) observers.get(i);
32          o.update(temperature, humidity, pressure);
33      }
34       */
35      /* enhanced for
36      for (Observer o : observers) {
37          o.update(temperature, humidity, pressure);
38      }
39      */
40      observers.forEach((observer) -> {
41          observer.update(temperature, humidity, pressure);
42      });
    }
```

enhanced for

enhanced for로 구현한 것과 비교해 볼 것!
java.lang.Iterable.forEach()

참고) http://www.baeldung.com/foreach-java

p.95

참고. For-each Loop

For-each loop	Equivalent for loop

`for (type var : arr) {`
`  body-of-loop`
`}`
	`for (int i = 0; i < arr.length; i++) {`
	`  type var = arr[i];`
	`  body-of-loop`
	`}`

`for (type var : coll) {`
`  body-of-loop`
`}`
	`for (Iterator<type> iter = coll.iterator(); iter.hasNext(); ) {`
	`  type var = iter.next();`
	`  body-of-loop`
	`}`

```
44      public void measurementChanged() {
45          notifyObservers();
46      }
47
48      // Test code: 실제 데이터 가져오는 코드 대신
49      // 프로그램을 테스트할 수 있도록 추가한 부분임.
50      public void setMeasurements(float temperature, float humidity, float pressure) {
51          this.temperature = temperature;
52          this.humidity = humidity;
53          this.pressure = pressure;
54
55          measurementChanged();
56      }
57  }
```

Observer.java

```
1  package cse.design_pattern.ch02.model02;
2
3  interface Observer {
4
5    void update(float temp, float humidity, float pressure);
6  }
```

DisplayElement.java

```
1  package cse.design_pattern.ch02.model02;
2
3  interface DisplayElement {
4
5      void display();
6  }
```

CurrentConditionsDisplay.java

```
1   package cse.design_pattern.ch02.model02;
2
3   public class CurrentConditionsDisplay implements Observer, DisplayElement {
4
5       private float temperature;
6       private float humidity;
7       private Subject weatherData;
8
9       public CurrentConditionsDisplay(Subject weatherData) {
10          this.weatherData = weatherData;
            this.weatherData.registerObserver(this);
12      }
13
        public void update(float temp, float humidity, float pressure) {
15          this.temperature = temp;
16          this.humidity = humidity;
17          display();
18      }
19
        public void display() {
21          System.out.println("Current conditions: " + temperature
22                  + "F degress and " + humidity + "% humidity");
23      }
24  }
```

p.96

StatisticsDisplay.java

```
1   package cse.design_pattern.ch02.model02;
2
3   public class StatisticsDisplay implements Observer, DisplayElement {
4
5       private WeatherData weatherData;
6
7       private float maxTemp = 0.0f;
8       private float minTemp = 200;
9       private float tempSum = 0.0f;
10      private int numReadings;
11
12      public StatisticsDisplay(WeatherData weatherData) {
13          this.weatherData = weatherData;
            weatherData.registerObserver(this);
15      }
```

```
17      @Override
        public void update(float temp, float humidity, float pressure) {
19          tempSum += temp;
20          numReadings++;
21
22          if (temp > maxTemp) {
23              maxTemp = temp;
24          }
25          if (temp < minTemp) {
26              minTemp = temp;
27          }
28          display();
29      }
30
31      @Override
        public void display() {
33          System.out.println("Avg/Max/Min temperature = " + (tempSum / numReadings)
34                  + "/" + maxTemp + "/" + minTemp);
35      }
36
37  }
```

ForecastDisplay.java

```
1   package cse.design_pattern.ch02.model02;
2
4   public class ForecastDisplay implements Observer, DisplayElement {
5
6       private float currentPressure = 29.92f;
7       private float lastPressure;
8       private WeatherData weatherData;
9
10      public ForecastDisplay(WeatherData weatherData) {
12          this.weatherData = weatherData;
13          weatherData.registerObserver(this);
15      }
16
18      public void update(float temp, float humidity, float pressure) {
19          lastPressure = currentPressure;
20          currentPressure = pressure;
21
22          display();
23      }
```

```
    public void display() {
22      System.out.print("Forecast: ");
23      if (currentPressure > lastPressure) {
24          System.out.println("Improving weather on the way!");
25      } else if (currentPressure == lastPressure) {
26          System.out.println("More of the same");
27      } else if (currentPressure < lastPressure) {
28          System.out.println("Watch out for cooler, rainy weather");
29      }
30  }
31
32  }
```

WeatherStation.java

```
5   package cse.design_pattern.ch02.model02;
6
7   /**...4 lines */
11  public class WeatherStation {
12
13      /**
14       * @param args the command line arguments
15       */
16      public static void main(String[] args) {
17          WeatherData weatherData = new WeatherData();
18
19          CurrentConditionsDisplay currentDisplay = new CurrentConditionsDisplay(weatherData);
20          StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);
21          ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);
22
23          weatherData.setMeasurements(80, 65, 30.4f);
24          System.out.println("------------------------------");
25          weatherData.setMeasurements(82, 70, 29.2f);
26          System.out.println("------------------------------");
27          weatherData.setMeasurements(78, 90, 29.2f);
28      }
29
30  }
```

p.97

실행 결과

```
--- exec-maven-plugin:1.2.1:exec (default-cli) @ desing_pattern ---
Current conditions: 80.0F degress and 65.0% humidity
Avg/Max/Min temperature = 80.0/80.0/80.0
Forecast: Improving weather on the way!
----------------------------------
Current conditions: 82.0F degress and 70.0% humidity
Avg/Max/Min temperature = 81.0/82.0/80.0
Forecast: Watch out for cooler, rainy weather
----------------------------------
Current conditions: 78.0F degress and 90.0% humidity
Avg/Max/Min temperature = 80.0/82.0/78.0
Forecast: More of the same
```