고급객체지향 프로그래밍
강의노트 #04

Observer Pattern

조용주
ycho@smu.ac.kr

옵저버 패턴 (Observer Pattern)

- 목적
  - Define a one-to-many dependency between objects so t hat when one object changes state, all its dependent are notified and updated automatically.
  - 객체간 1: 다 의존 관계를 정의함. 한 개 객체 상태가 변화될 때, 그 객체와 의존 관계에 있는 모든 객체들이 자동으로 알림을 받고 상태를 갱신

옵저버 패턴 (Observer Pattern)

- 옵저버 패턴은 일종의 푸쉬 (push) 서비스를 구현
  - 뉴스
    - 뉴스 사이트를 방문해서 매번 새로운 뉴스가 있는지 확인하는 것은 번거로움
    - 오히려 구독 서비스를 신청하고, 새로운 뉴스가 있으면 알려주는 것이 편리함
    - 뉴닉 (Newneek) 같은 메일링 서비스를 생각하면 됨
  - 호텔의 모닝콜 서비스
    - 일어날 시간을 확인하기 위해 자다 깨다를 반복하는 것은 어려움
    - 아침에 일어나야 할 시간에 모닝콜을 받거나 알람을 맞추는 것이 바람직함

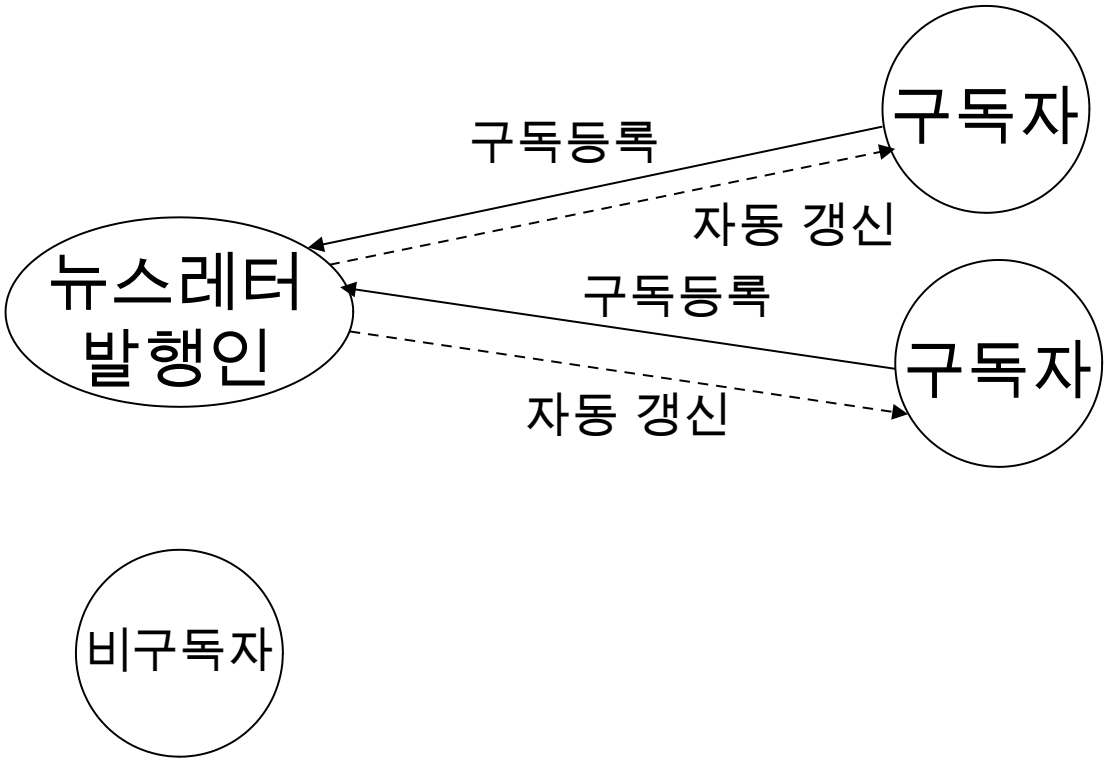
옵저버 패턴 (Observer Pattern)

- 전화번호 프로그램에서 정보를 확인하기 위해 마우스로 화면의 버튼을 누르는 경우
  - 버튼이 클릭되었는지 프로그램에서 계속 확인 (polling) 하는 것은 낭비에 가까움
  - 버튼이 클릭된 이벤트가 발생하는 경우, 프로그램에 이벤트 발생 정보를 전달하고, 처리하도록 하는 것 (push) 이 효율적
- 옵저버 패턴은 푸쉬를 사용하며 절차는 :
  - 푸쉬를 받고자 하는 사용자가 등록
  - 특정 상황이 발생하면, 등록된 사용자에게 모두 알리고 자동으로 데이터가 갱신 됨

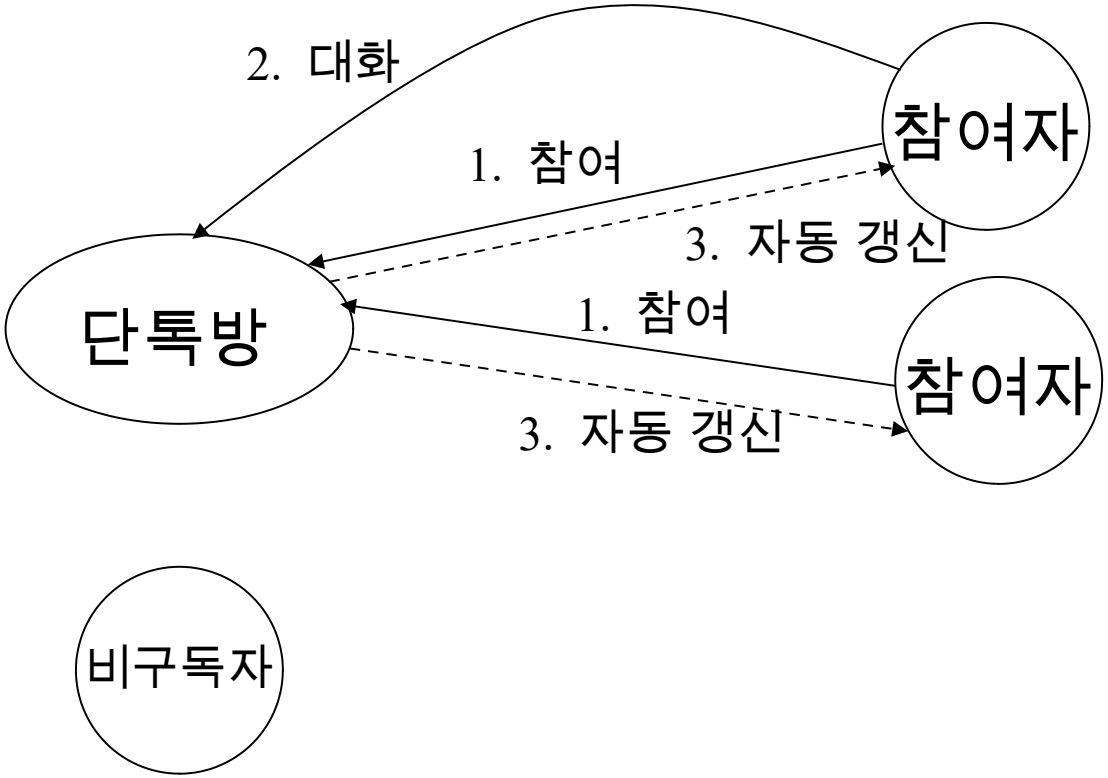
옵저버 패턴 (Observer Pattern)

- 뉴스레터 + 구독자 = 옵저버 패턴
  - 뉴스레터 발행자는 Subject (Publisher 라고 부르기도 함)
  - 구독자는 Observer (Subscriber 라고 부르기도 함)

옵저버 패턴 (Observer Pattern)



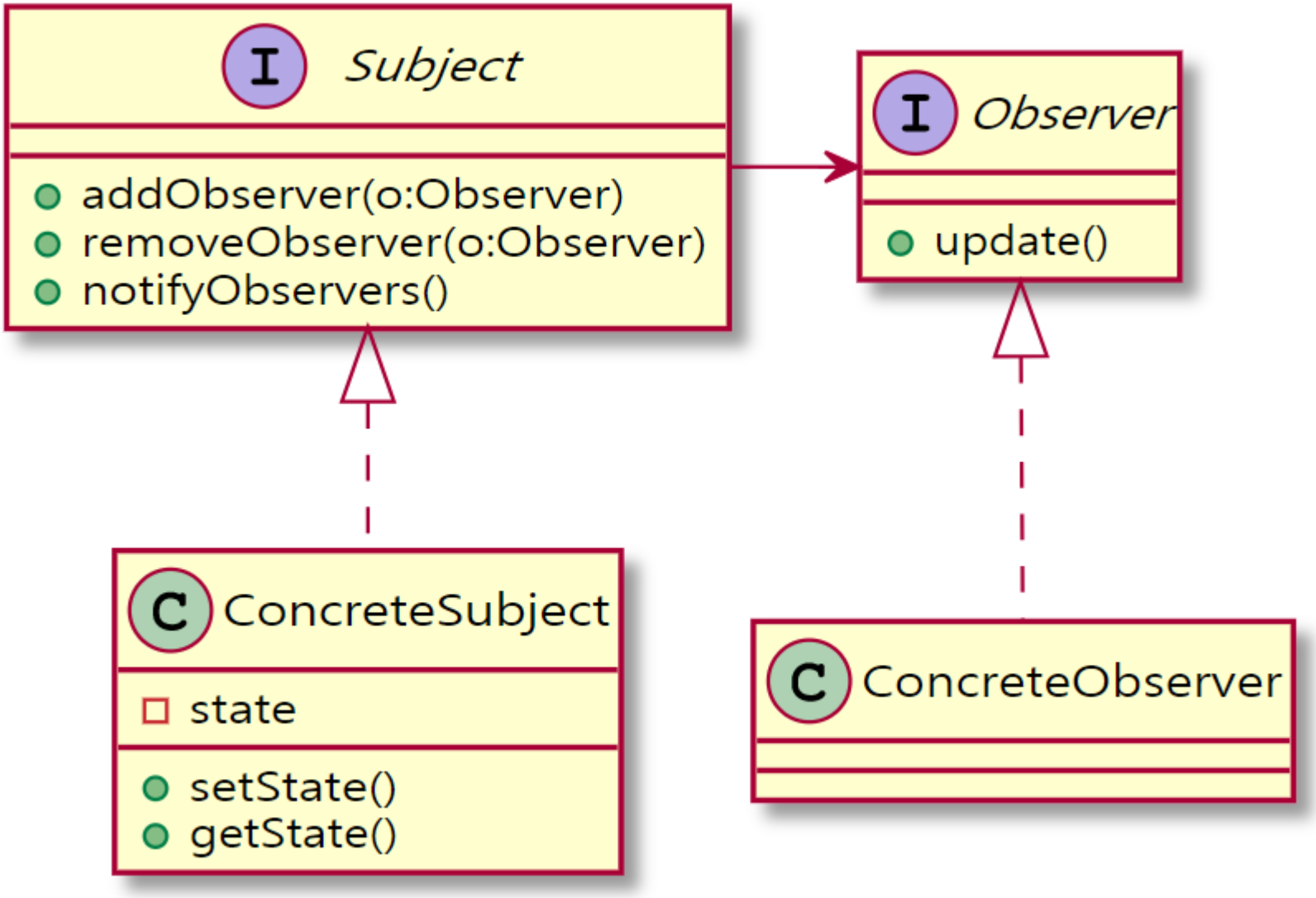
옵저버 패턴 (Observer Pattern)



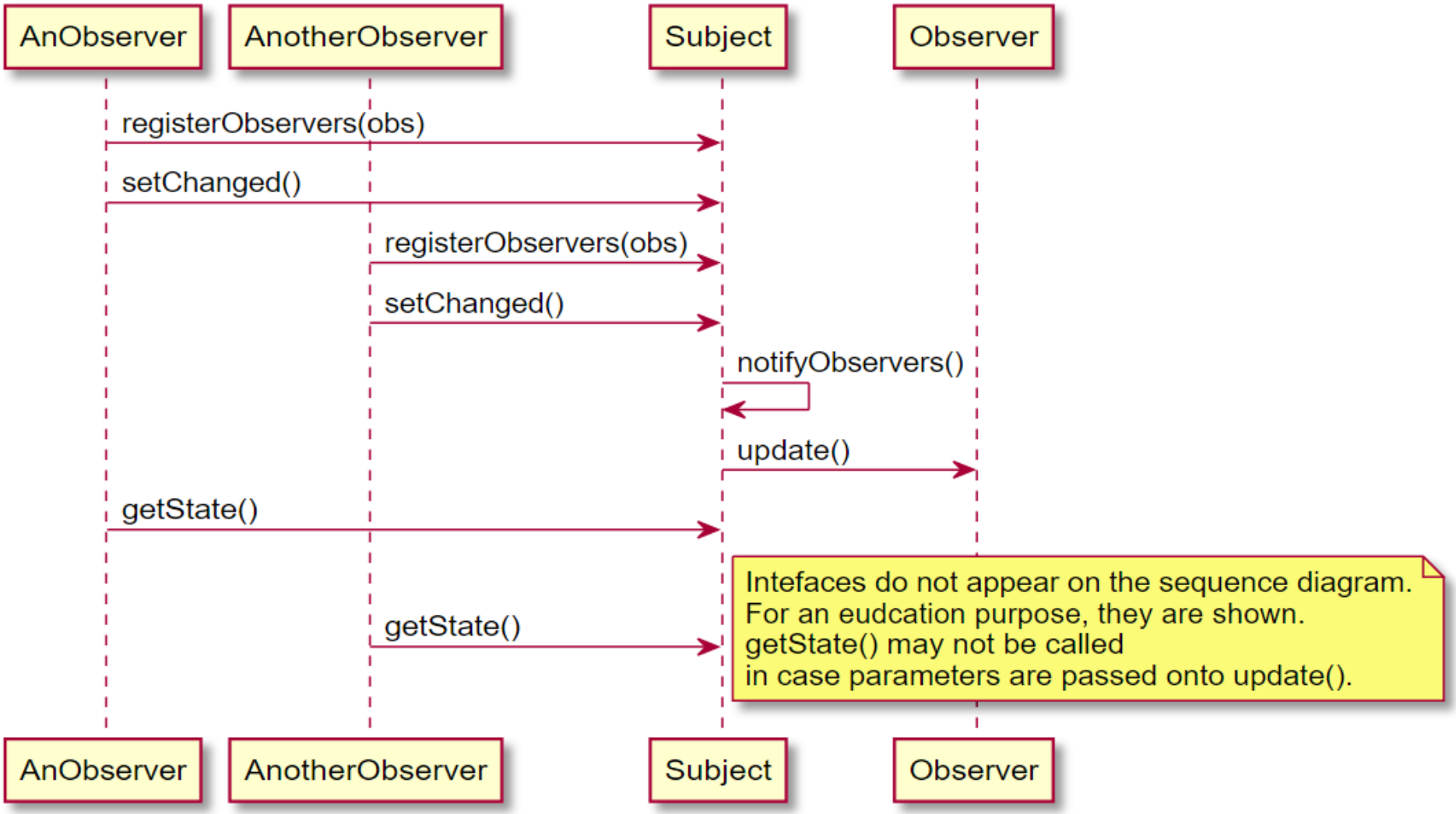
디자인 패턴 요소

요소	설명
이름	옵저버 (Observer)
문제	1:n 관계에서의 정보 갱신
해결방안	사용자를 등록하고, 정보가 변동하는 경우 알려주고 값을 자동으로 갱신
결과	느슨한 커플링 (loose coupling), 확장성

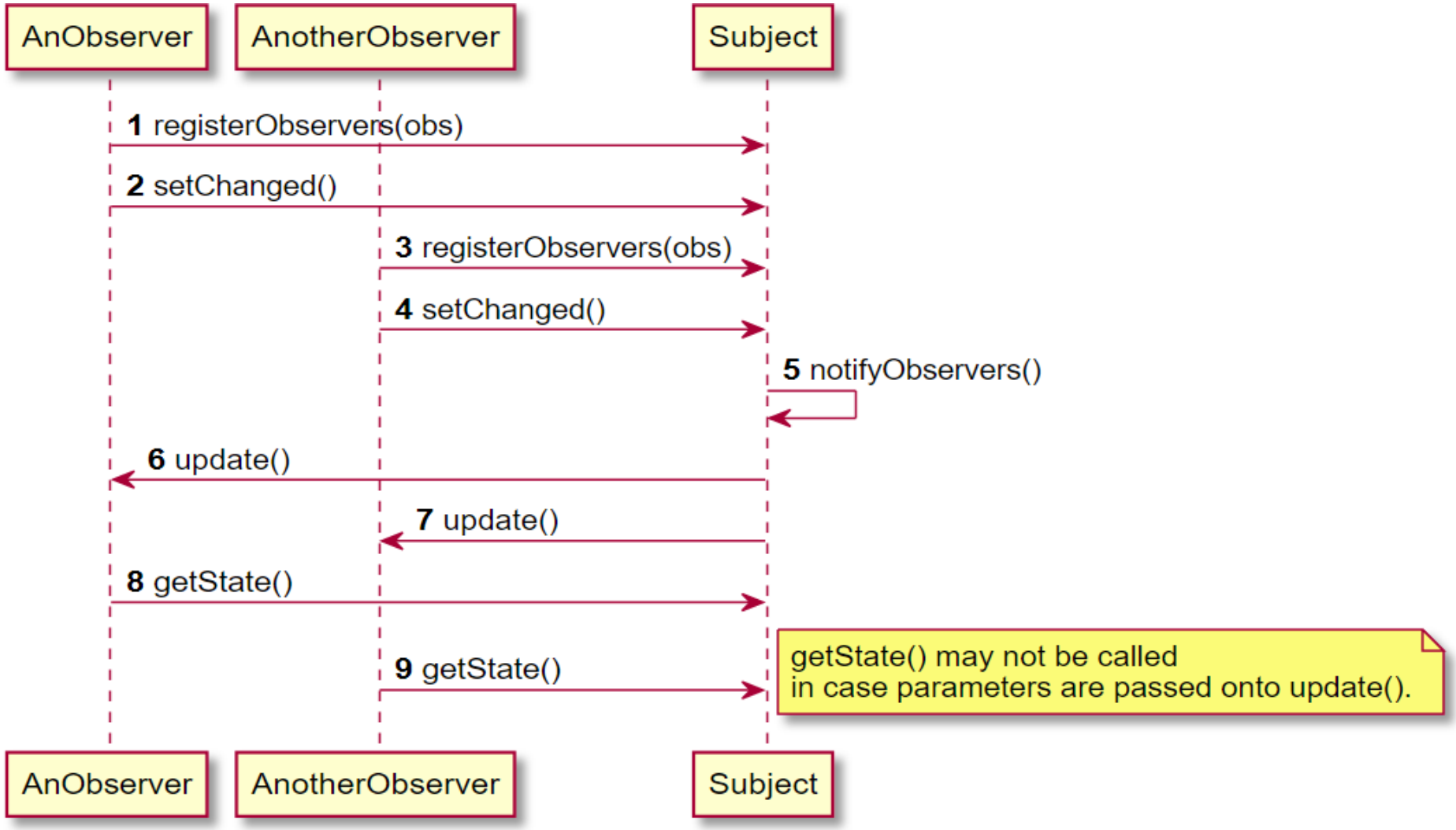
옵저버 패턴


I Subject
addObserver(o:Observer)
removeObserver(o:Observer)
notifyObservers()
I Observer
update()
C ConcreteSubject
state
setState()
getState()
C ConcreteObserver

옵저버 패턴


AnObserver
AnotherObserver
Subject
Observer
registerObservers(obs)
setChanged()
registerObservers(obs)
setChanged()
notifyObservers()
update()
getState()
Intefaces do not appear on the sequence diagram.
For an eudcation purpose, they are shown.
getState() may not be called
in case parameters are passed onto update().
getState()
AnObserver
AnotherObserver
Subject
Observer

옵저버 패턴

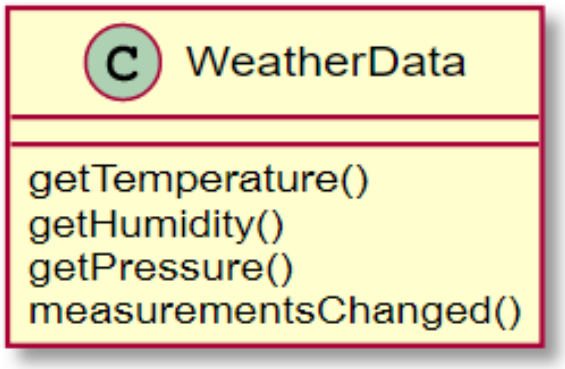

AnObserver
AnotherObserver
Subject
1 registerObservers(obs)
2 setChanged()
3 registerObservers(obs)
4 setChanged()
5 notifyObservers()
6 update()
7 update()
8 getState()
9 getState()
getState() may not be called
in case parameters are passed onto update().
AnObserver
AnotherObserver
Subject

사례 1 – Weather (HFDP Ch. 2)

- 날씨 (HFDP Ch. 2)
  - Weather-O-Rama 사의 차세대 인터넷 기반 기상 정보 스테이션 구축
    - 시스템 구성
      - 기상 스테이션 : 기상 정보를 수집하는 장비
      - WeatherData 객체 : 기상 스테이션으로부터 오는 데이터를 추적하는 객체
      - 디스플레이 : 사용자에게 현재 기상 조건을 보여주는 장치
        - 현재 조건 (온도 , 습도 , 압력), 기상 통계 , 간단한 기상 예보를 다른 화면에 표시 가능

사례 1 – Weather (HFDP Ch. 2)

- 정보 공급자는 정기적으로 계속 온도, 습도, 기압을 측정 수집함
- WeatherData 클래스



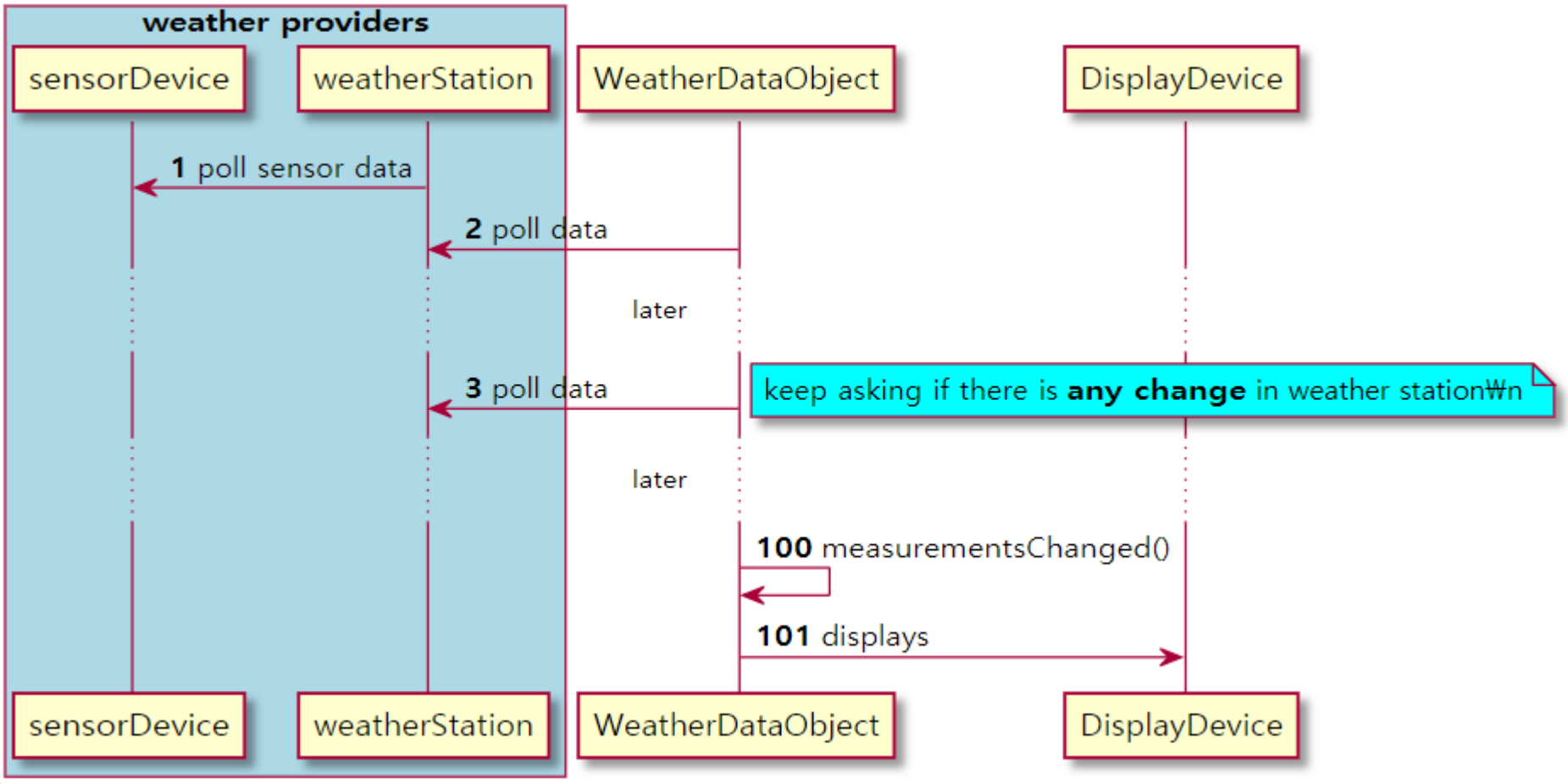
getTemperature(), getHumidity(), getPressure() 메소드는 가장 최근에 측정된 온도, 습도, 기압 값을 반환

  - 기상 측정 값이 바뀔 때마다 measurementsChanged() 함수가 호출된다고 함
    - 이 함수를 구현해서 디스플레이에 정보를 표시하도록 해야 함

사례 1 – Weather (HFDP Ch. 2)

```
public void measurementsChanged() {
    //  이미 구현된 함수를 통해 최신 측정값 가져옴
    float temp = getTemperature();
    float humidity = getHumidity();
    float pressure = getPressure();
    //  디스플레이 갱신
    currentConditionsDisplay.update(temp,
                                    humidity, pressure);
    statisticsDisplay.update(temp, humidity,
                                      pressure);
    forecastDisplay.update(temp, humidity,
                                    pressure);
}
```

사례 1 – Weather (HFDP Ch. 2)

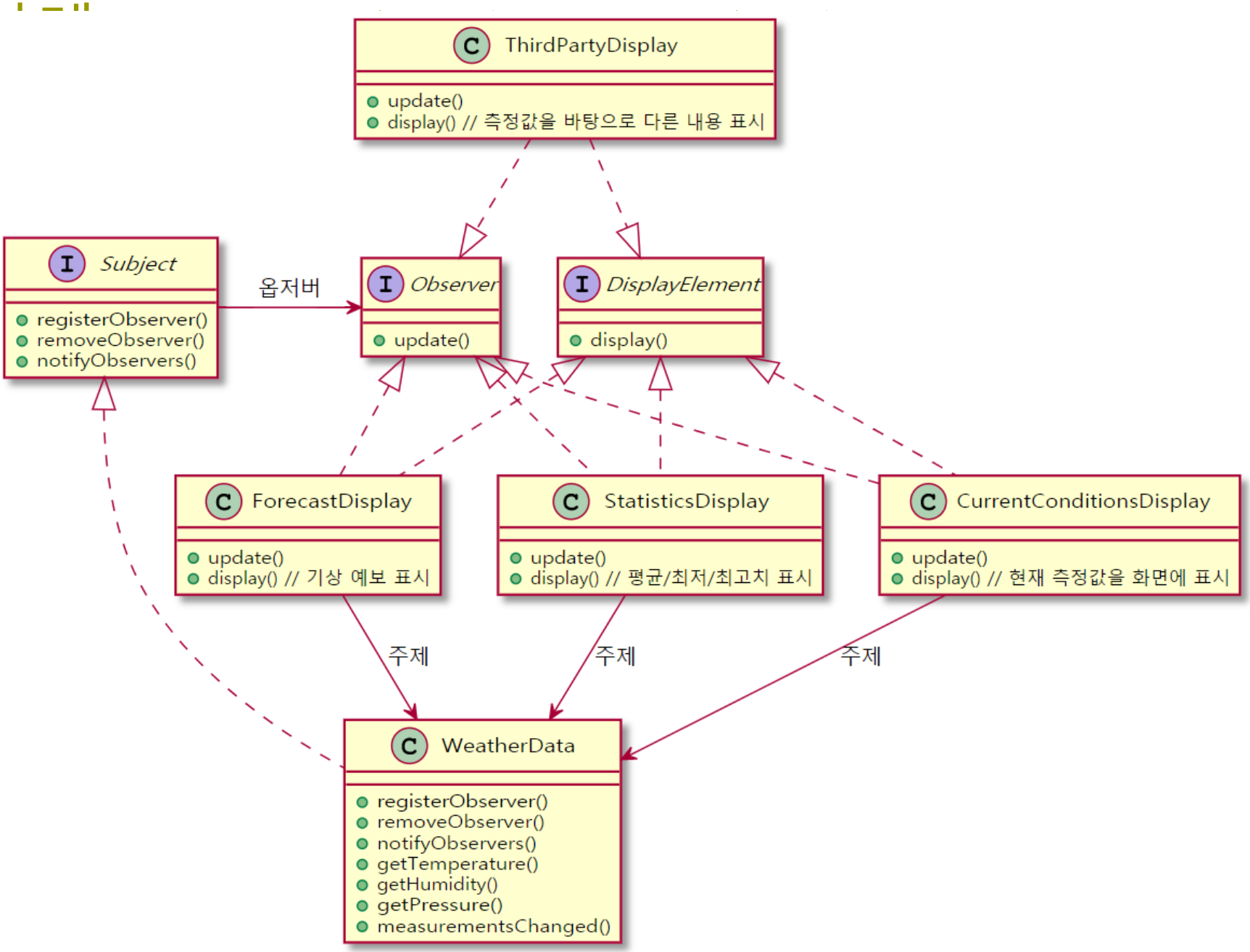


사례 1 – Weather (HFDP Ch. 2)

- 문제
  - measurementsChanged 함수 내부에서 `currentConditionsDisplay, statisticsDisplay, forecastDisplay` 같은 구체적인 객체를 사용하고 있는데 새로운 디스플레이 화면을 추가하거나 기존 화면을 제거하려면 이 함수를 수정해야만 함
  - 구체적인 객체를 사용하는 부분이 바뀔 수 있는 부분이므로 캡슐화가 필요함

느슨한 결합 (loose coupling)

- 두 객체가 느슨하게 결합되어 있다는 것은, 그 둘이 상호작용을 하긴 하지만 서로에 대해 잘 모른다는 것을 의미함
- 옵저버 패턴에서는 Subject 와 Observer 간 느슨한 결합이 만들어짐
  - Subject 가 Observer 에 대해서 아는 것은 특정 인터페이스를 구현한다는 것뿐 (Observer 를 구현하는 실제 클래스가 무엇인지, Observer 의 역할이 무엇인지 몰라도 됨)
  - 새로운 Observer 는 쉽게 추가하거나 제거 가능 (실행 중에도 가능)
  - Observer 가 새로 생겨도 Subject 가 바뀌지 않음
  - Subject 와 Observer 는 독립적으로 재사용 가능


ThirdPartyDisplay
update()
display() // 측정값을 바탕으로 다른 내용 표시
Subject
registerObserver()
removeObserver()
notifyObservers()
옵저버
Observer
update()
DisplayElement
display()
ForecastDisplay
update()
display() // 기상 예보 표시
StatisticsDisplay
update()
display() // 평균/최저/최고치 표시
CurrentConditionsDisplay
update()
display() // 현재 측정값을 화면에 표시
주제
주제
주제
WeatherData
registerObserver()
removeObserver()
notifyObservers()
getTemperature()
getHumidity()
getPressure()
measurementsChanged()

사례 1 – Weather (HFDP Ch. 2)

- 일반 ArrayList 버전

```
// Subject.java
package headfirst.observer.weather;

public interface Subject {
   public void registerObserver(Observer o);
   public void removeObserver(Observer o);
   public void notifyObservers();
}
```

```
// Observer.java
package headfirst.observer.weather;

public interface Observer {
   public void update(float temp,
                      float humidity, float pressure);
}
```

```
// WeatherData.java
package headfirst.observer.weather;

import java.util.*;

public class WeatherData implements Subject {
    private ArrayList observers;
    private float temperature;
    private float humidity;
    private float pressure;
    public WeatherData() {
        observers = new ArrayList();
    }
    public void registerObserver(Observer o) {
        observers.add(o);
    }
    public void removeObserver(Observer o) {
        int i = observers.indexOf(o);
        if (i >= 0) {
            observers.remove(i);
        }
    }
```

```
    public void notifyObservers() {
        for (int i = 0; i < observers.size(); i++) {
            Observer observer =
                    (Observer) observers.get(i);
      observer.update(temperature, humidity,
                                pressure);
        }
    }
    public void measurementsChanged() {
        notifyObservers();
    }
    public void setMeasurements(float temperature,
                        float humidity, float pressure) {
        this.temperature = temperature;
        this.humidity = humidity;
        this.pressure = pressure;
        measurementsChanged();
    }
```

사례 1 – Weather (HFDP Ch. 2)

```
    public float getTemperature() {
        return temperature;
    }
    public float getHumidity() {
        return humidity;
    }
    public float getPressure() {
        return pressure;
    }
}
```

사례 1 – Weather (HFDP Ch. 2)

```
// DisplayElement.java
package headfirst.observer.weather;

public interface DisplayElement {
   public void display();
}
```

```
// CurrentConditionsDisplay.java
package headfirst.observer.weather;

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

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

사례 1 – Weather (HFDP Ch. 2)

```
// WeatherStation.java
package headfirst.observer.weather;

import java.util.*;

public class WeatherStation {
    public static void main(String[] args) {
        WeatherData weatherData = new WeatherData();
     CurrentConditionsDisplay currentDisplay
            = new CurrentConditionsDisplay(weatherData);
/*   StatisticsDisplay statisticsDisplay
                    = new StatisticsDisplay(weatherData);
     ForecastDisplay forecastDisplay
                    = new ForecastDisplay(weatherData); */
        weatherData.setMeasurements(80, 65, 30.4f);
        weatherData.setMeasurements(82, 70, 29.2f);
        weatherData.setMeasurements(78, 90, 29.2f);
    }
}
```

사례 1 – Weather (HFDP Ch. 2)

- 제네릭스 버전

```
package headfirst.observer.weather;

import java.util.*;

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

    public WeatherData() {
        observers = new ArrayList<Observer>();
    }
    public void registerObserver(Observer o) {
        observers.add(o);
    }
```

```
    public void removeObserver(Observer o) {
        int i = observers.indexOf(o);
        if (i >= 0) {
            observers.remove(i);
        }
    }
    public void notifyObservers() {
        for (int i = 0; i < observers.size(); i++) {
            Observer observer = observers.get(i);
      observer.update(temperature, humidity,
                              pressure);
        }
/*        for (Observer observer : observers) {
            observer.update(temperature, humidity,
                              pressure);
        }*/
    }
```

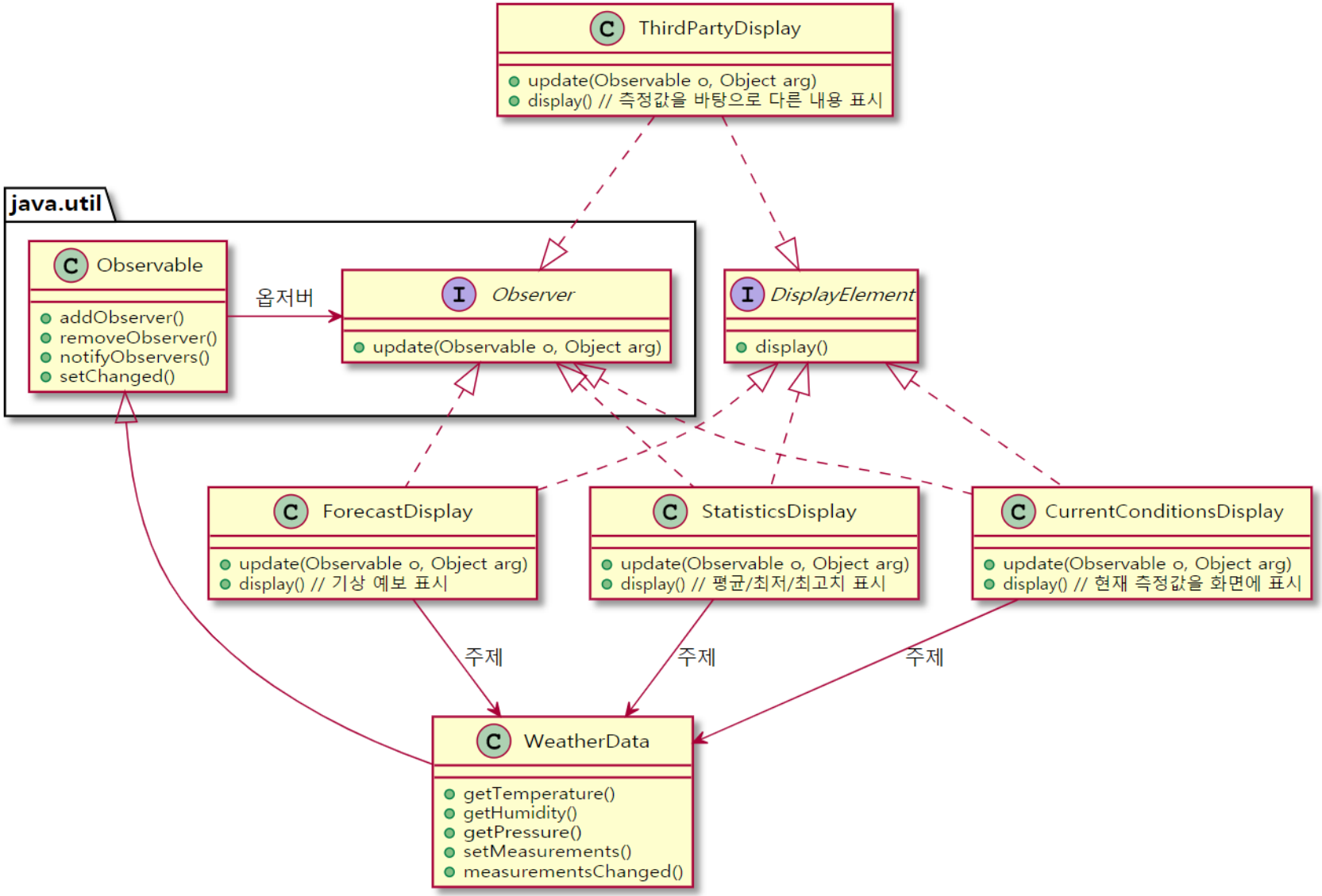
사례 1 – Weather (HFDP Ch. 2)

```
    public void measurementsChanged() {
        notifyObservers();
    }
    public void setMeasurements(float temperature,
                                float humidity, float pressure) {
        this.temperature = temperature;
        this.humidity = humidity;
        this.pressure = pressure;
        measurementsChanged();
    }
    public float getTemperature() {
        return temperature;
    }
    public float getHumidity() {
        return humidity;
    }
    public float getPressure() {
        return pressure;
    }
}
```

사례 1 – Weather (HFDP Ch. 2) – Java 버전

- Java java.util.Observable/java.util.Observer 사용
  - 주의 : Deprecated (since Java 9)
  - 객체가 Observer 가 되는 방법
    - Observer 인터페이스 구현 후 Observable 의 addObserver() 함수 호출
  - Observable 에서 알림을 주는 (push 하는) 방법
    - 첫 번째로 setChanged() 메소드를 호출해서 객체의 상태가 바뀌었음을 알림
    - 두 번째로 notifyObservers() 또는 notifyObservers(Object arg) 함수를 호출해서 알림
  - Observer 가 연락 받는 방법
    - update(Observable o, Object arg) 구현

사례 1 – Weather (HFDP Ch. 2) – Java 버전



```
package headfirst.observer.weather;

import java.util.*;

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

    public WeatherData() { }
    public void measurementsChanged() {
        setChanged(); //  상태가 바뀜을 알림
        notifyObservers();
    }
    public void setMeasurements(float temperature,
                        float humidity, float pressure) {
        this.temperature = temperature;
        this.humidity = humidity;
        this.pressure = pressure;
        measurementsChanged();
    }
```

사례 1 – Weather (HFDP Ch. 2) – Java 버전

```
    public float getTemperature() {
        return temperature;
    }
    public float getHumidity() {
        return humidity;
    }
    public float getPressure() {
        return pressure;
    }
}
```

사례 1 – Weather (HFDP Ch. 2)

```
// CurrentConditionsDisplay.java
package headfirst.observer.weather;

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

    public CurrentConditionsDisplay(
                                        Observable observable) {
        this.observable = observable;
        observable.addObserver(this);
    }
```

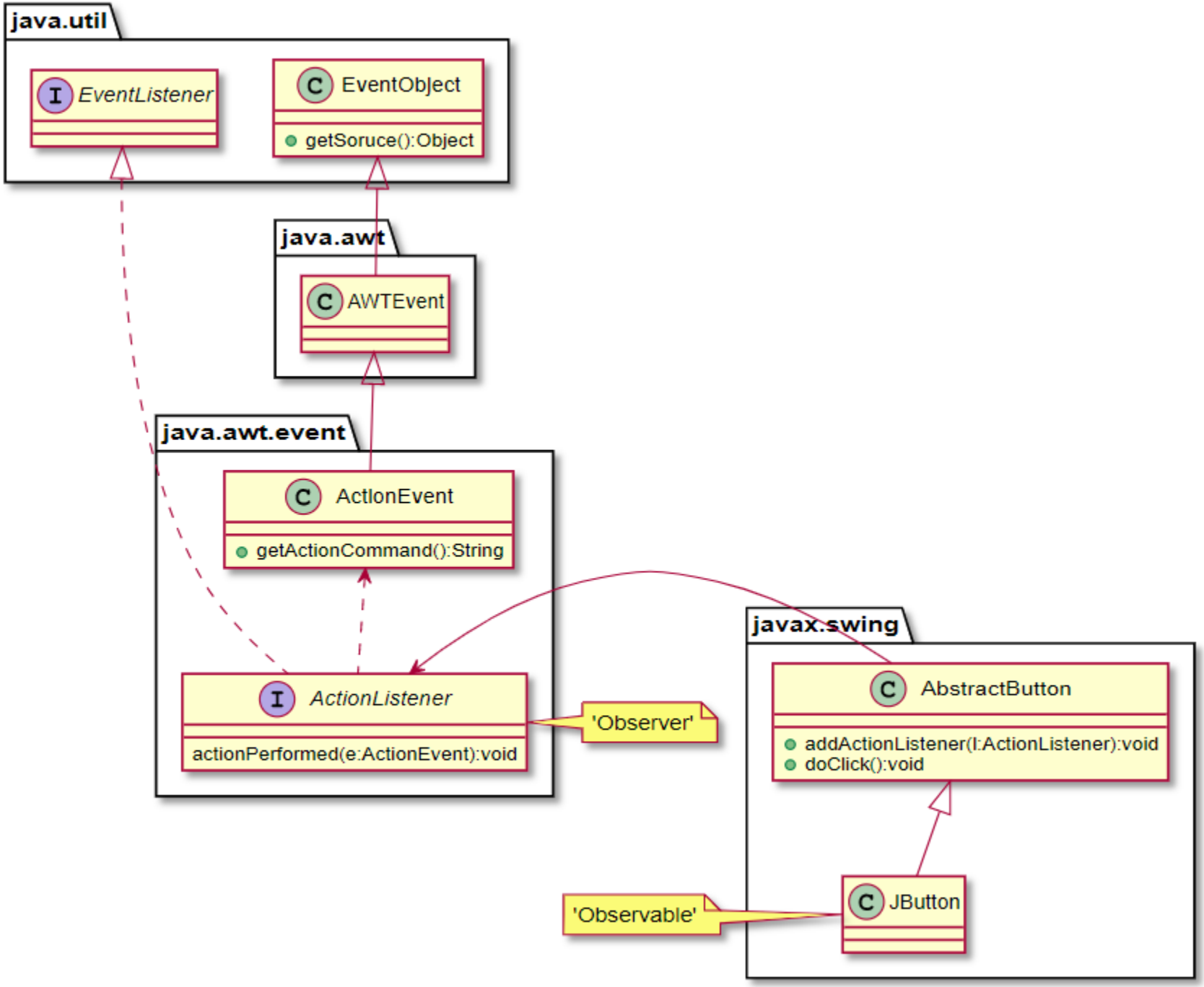
```
    public void update(Observable o, Object arg) {
        if (arg instanceof WeatherData) {
            WeatherData wd = (WeatherData) arg;
            this.temperature = wd.getTemperature();
            this.humidity = wd.getHumidity();
            display();
        }
    }

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

사례 1 – Weather (HFDP Ch. 2) – Java 버전

- java.util.Observable/java.util.Observer 문제점
  - java.util.Observable 이 인터페이스가 아니라 클래스로 되어 있음
  - 다른 클래스로부터 상속 받아야 하는 클래스는 Observable 에서부터 상속 받을 수 없음
  - setChanged() 함수가 protected 로 되어 있음. 어차피 Observable 에서 상속 받아야 쓸 수 있으므로 문제가 될 것은 아니지만, 상속보다는 구성을 사용한다는 디자인 원칙에 위배됨
  - 자바 9 부터 deprecated 됨 (사용 권장 않음)

사례 2 – Swing 의 ActionListener

- Swing 의 JButton 은 Observable (Subject)
- JButton 의 부모 클래스인 AbstractButton 에는 addActionListener() 함수가 존재함
  - Swing 의 이벤트 리스너 (event listener) 는 Observer 에 해당됨
- JButton 에 이벤트가 발생하면 , JButton 에 등록되어 있는 리스너의 actionPerformed() 함수를 호출


java.util
EventListener
EventObject
getSoruce():Object
java.awt
AWTEvent
java.awt.event
ActionEvent
getActionCommand():String
ActionListener
actionPerformed(e:ActionEvent):void
'Observer'
javax.swing
AbstractButton
addActionListener(l:ActionListener):void
doClick():void
'Observable'
JButton

사례 2 – Swing 의 ActionListener

```
public class SwingObserverExample {
  JFrame frame;
  public static void main(String[] args) {
    SwingObserverExample example
                          = new SwingObserverExample();
    example.go();
  }

  public void go() {
    frame = new JFrame();
    JButton button = new JButton(" 정말 해도 될까 ?");
    button.addActionListener(new AngelListener());
    button.addActionListener(new DevilListener());
    frame.getContentPane().add(
                          BorderLayout.CENTER, button);
    frame.setSize(200, 200);
    frame.setVisible(true);
  }
```

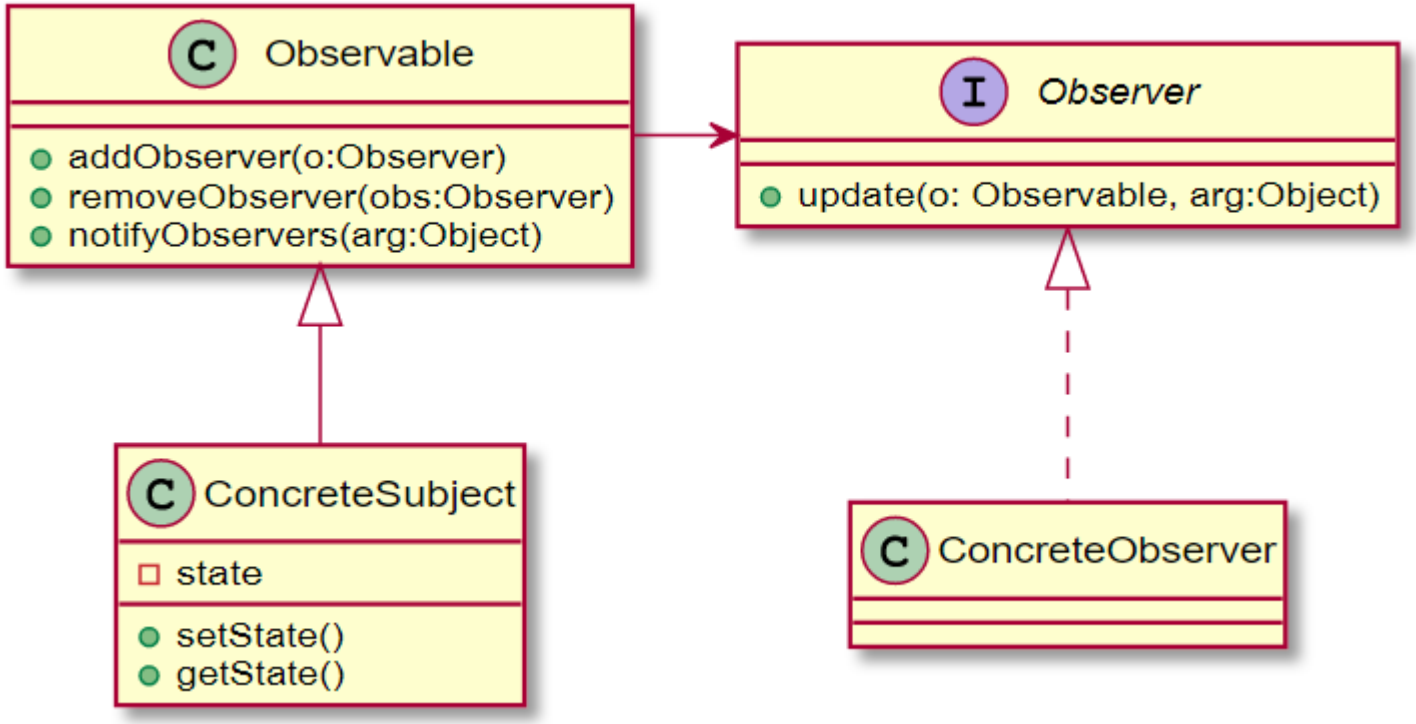
사례 2 – Swing 의 ActionListener

```
class AngelListener implements ActionListener {
  public void actionPerformed(ActionEvent event) {
    System.out.println(" 안돼 .  분명 나중에 후회할거야 ");
  }
}

class DevilListener implements ActionListener {
  public void actionPerformed(ActionEvent event) {
    System.out.println(" 당연하지 .  그냥 저질러 버려 !");
  }
}
```

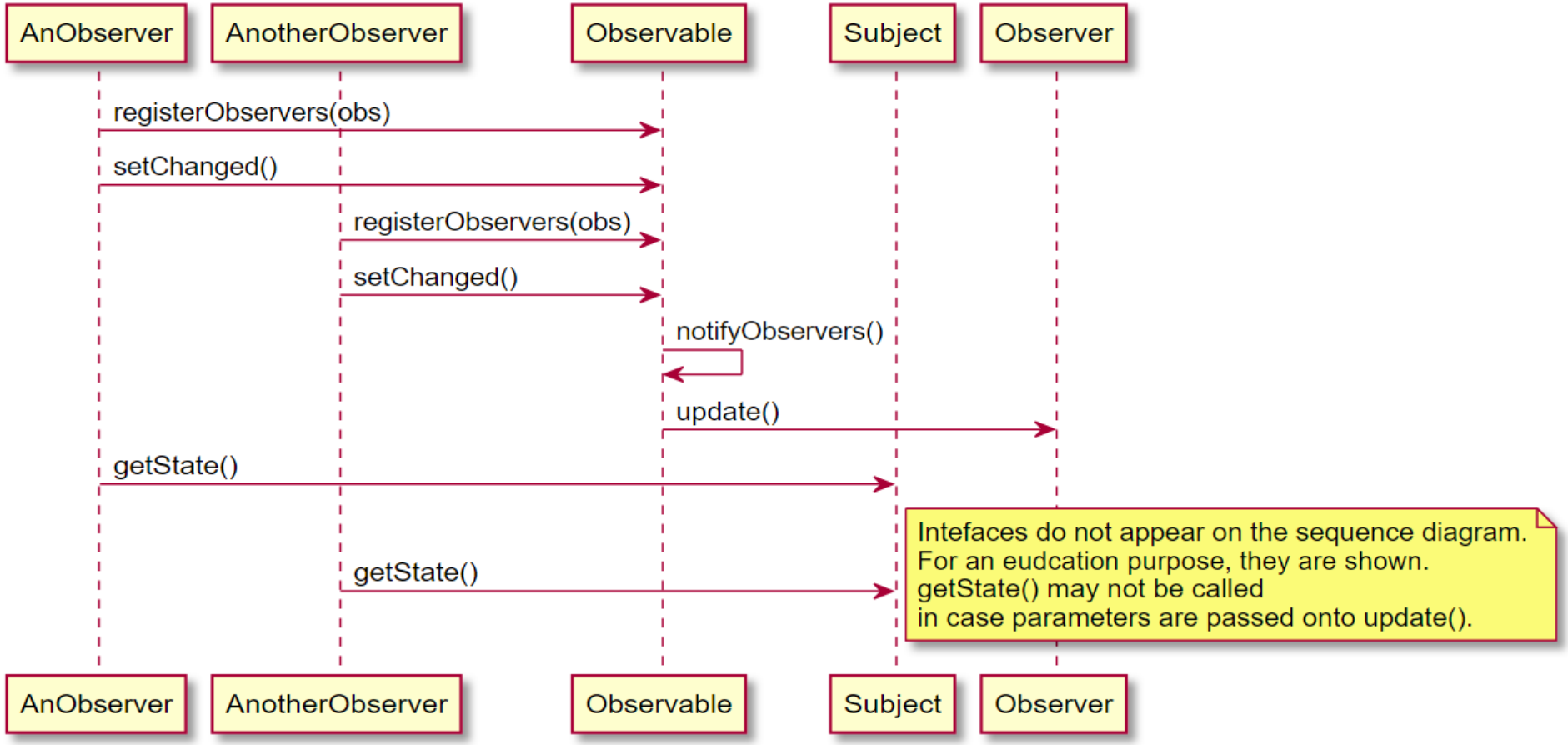
옵저버 패턴

- 설계
  - 인터페이스 분리 (ISP)
  - 구체적인 클래스 상속
  - 클라이언트는 구체 클래스보다는 추상 클래스 사용 후 상속
  - 정보 제공자 (Subject)
    - 상태가 변경되면 알림 기능 (notify)
    - 알림 대상이 되는 옵저버를 사전 등록 (register)
    - Observable 또는 Publisher 라고도 함
  - 정부 수신자 (Observer)
    - 정보 제공자의 상태가 변경되면 그 내용을 받아서 반영함 (update)
    - Subscriber 라고도 함

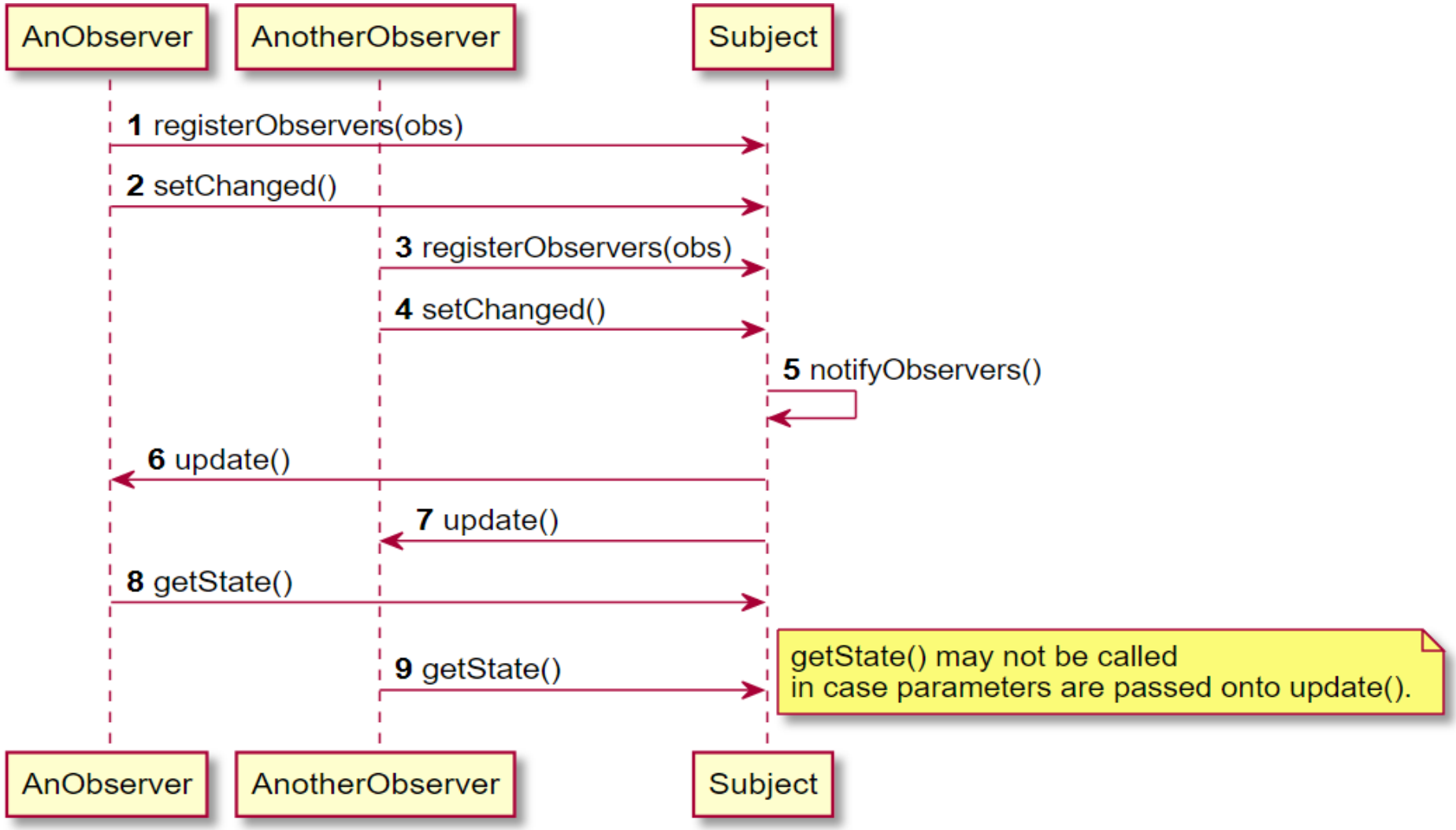
옵저버 패턴


Observable
addObserver(o:Observer)
removeObserver(obs:Observer)
notifyObservers(arg:Object)
Observer
update(o: Observable, arg:Object)
ConcreteSubject
state
setState()
getState()
ConcreteObserver

옵저버 패턴


AnObserver
AnotherObserver
Observable
Subject
Observer
registerObservers(obs)
setChanged()
registerObservers(obs)
setChanged()
notifyObservers()
update()
getState()
getState()
Intefaces do not appear on the sequence diagram.
For an eudcation purpose, they are shown.
getState() may not be called
in case parameters are passed onto update().
AnObserver
AnotherObserver
Observable
Subject
Observer

옵저버 패턴


AnObserver
AnotherObserver
Subject
1 registerObservers(obs)
2 setChanged()
3 registerObservers(obs)
4 setChanged()
5 notifyObservers()
6 update()
7 update()
8 getState()
9 getState()
getState() may not be called
in case parameters are passed onto update().
AnObserver
AnotherObserver
Subject