Observer Pattern

2015004239 정성운

원본 :
https://github.com/OdysseyJ/Software_Engineering/wiki/Observer_Pattern

Observer_Pattern

SeongWoon Jeong edited this page 2 minutes ago · 1 revision

Software Engineering - CSE4006

Observer Pattern

professor. 정소희

---

1. 개요

- Observer Pattern에 대해서 이해한다.
  - Observer Pattern : 한 객체의 변화가 여러객체에게 전달되어야 하고, 전달되는 객체의 수가 많을때 사용하는 패턴.
  - Subject, ConcreteSubject, Observer, ConcreteObserver로 구성된다.

2. 과제 설명

- 배터리 양을 관리하는 프로그램을 작성하려고 한다. 소스코드를 보고 설계 취약점을 OCP 측면에서 분석하여 설명하시오.
  - OCP(Open Closed Principle) : 소프트웨어는 확장에는 열려있고, 수정에는 닫혀있어야 한다는 특성.
- 설계의 취약점을 Observer Pattern을 활용해 개선할 수 있도록 설명하고 소스코드를 작성하시오

3. 사용언어 / 환경

- Java / Mac OS

[주어진 소스코드]

```java
package exercise04.observer;

public class Battery {
    private int level = 100 ;
    private BatteryLevelDisplay display ;
    private LowBatteryWarning warning ;
                구체적인 녀석들을 알고있는 설계
    public void setDisplay(BatteryLevelDisplay display) {
        this.display = display ;
    }
    public void setWarning(LowBatteryWarning warning) {
        this.warning = warning ;
    }
    public void consume(int amount) {
        level -= amount ;

                        바뀌는 부분
        display.update() ;
        warning.update() ;
    }
    public int getLevel() { return level ; }
}
```

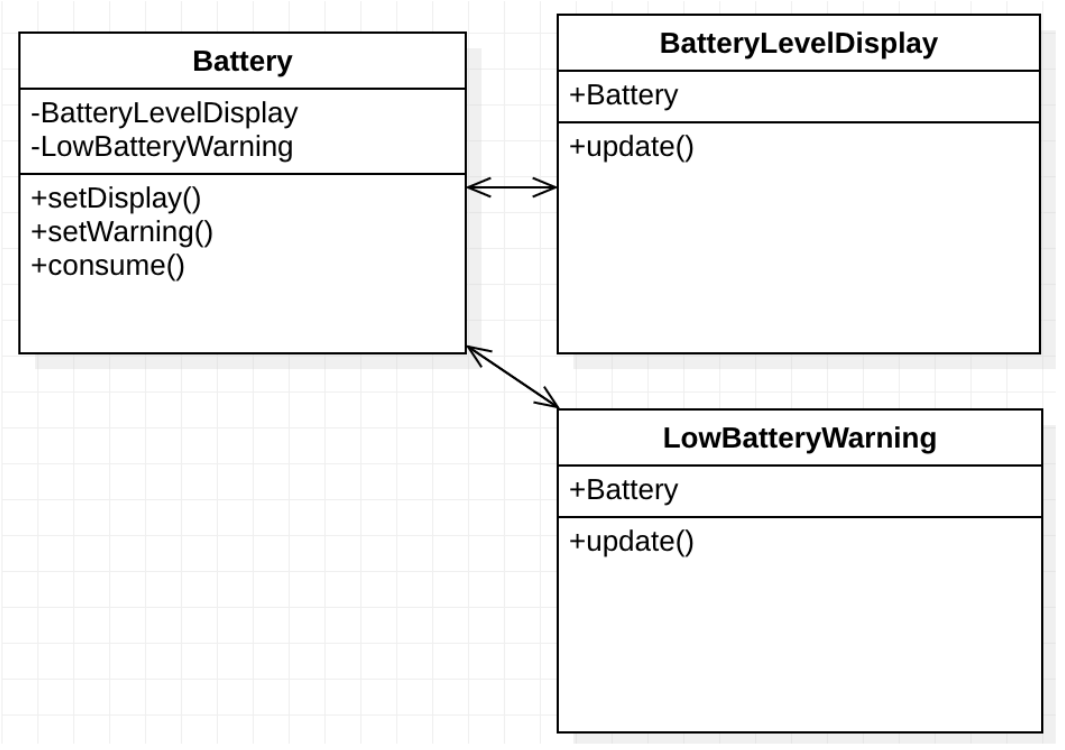
```java
package exercise04.observer;

public class BatteryLevelDisplay  {
    private Battery battery ;
    public BatteryLevelDisplay(Battery battery) {
        this.battery = battery ;
    }
    public void update() {
        int level = battery.getLevel() ;
        System.out.println("Level: " + level) ;
    }
}
```



```java
package exercise04.observer;

public class LowBatteryWarning {
    private static final int LOW_BATTERY = 30 ;
    private Battery battery ;
    public LowBatteryWarning(Battery battery) {
        this.battery = battery ;
    }
    public void update() {
        int level = battery.getLevel() ;
        if ( level < LOW_BATTERY )
            System.out.println("<Warning> Low Battery : "
                + level + " Compared with " + LOW_BATTERY) ;
    }
}
```

```java
package exercise04.observer;

public class Client {
    public static void main(String[] args) {
        Battery battery = new Battery() ;

        BatteryLevelDisplay batteryDisplay
            = new BatteryLevelDisplay(battery) ;
        LowBatteryWarning lowBatteryWarning
            = new LowBatteryWarning(battery) ;

        battery.setDisplay(batteryDisplay) ;
        battery.setWarning(lowBatteryWarning) ;

        battery.consume(20) ;
        battery.consume(50) ;
        battery.consume(10) ;
    }
}
```

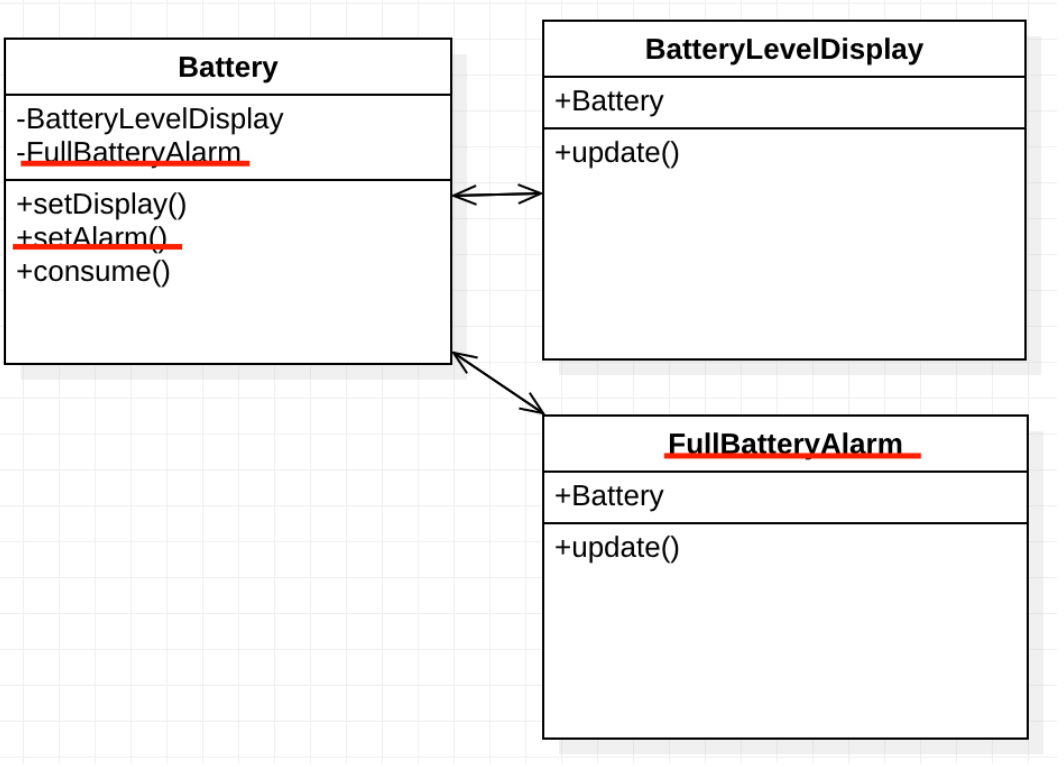
```
Level: 80
Level: 30
Level: 20
<Warning> Low Battery : 20 Compared with 30
```



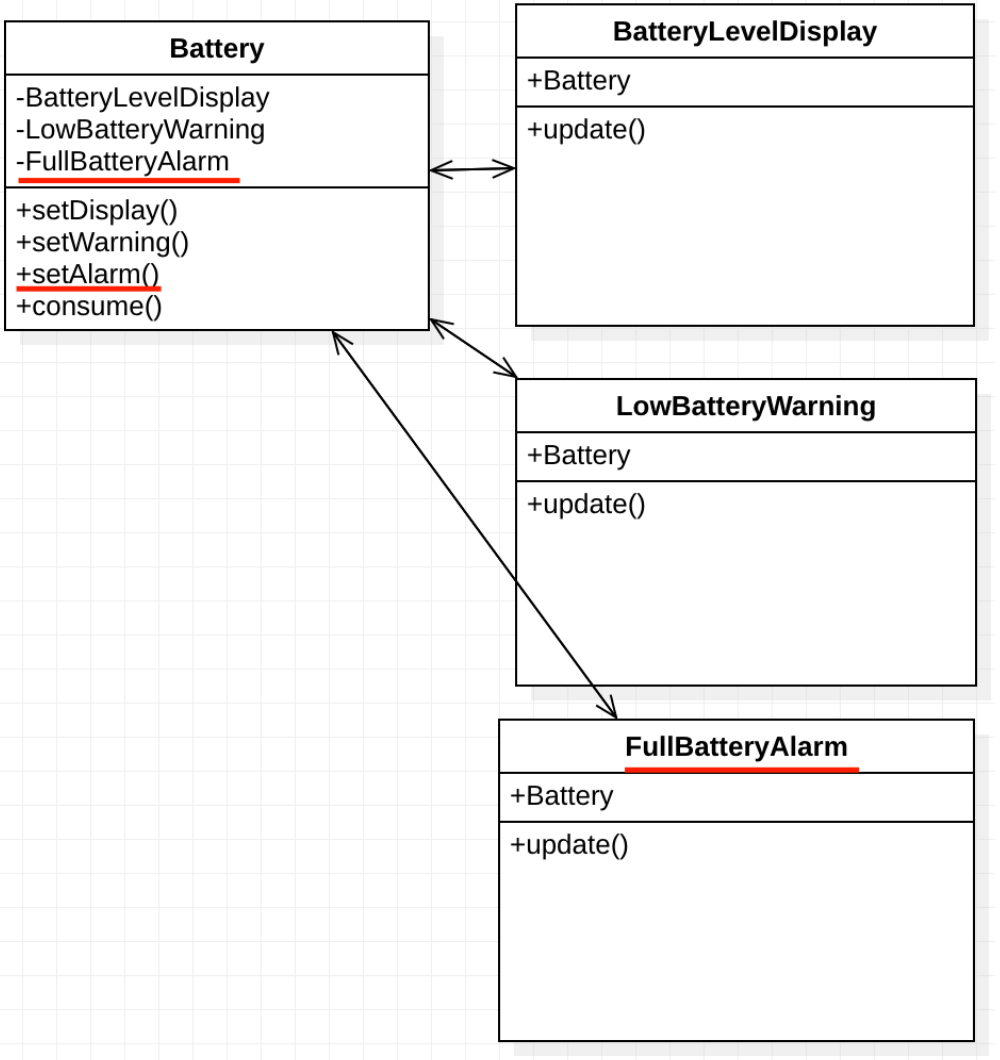
Battery가 자신의 변화를 직접 LowBatteryWarning과 BatteryLevelDisplay에 알려주는 구조이다.

[주어진 소스코드의 취약점 분석]

- OCP의 측면에서 소프트웨어는 확장에 열려있고 변경에 닫혀있어야 하지만, 해당 설계는 확장에 닫혀있고 변경에 열려있다.
  - 만약, 기존의 LowBatteryWarning의 동작방식이 변경되는 경우? Battery와 LowBatteryWarning의 코드를 모두 수정해야 한다.
  - 따라서 해당 설계는 수정에 열려있다



- 만약, 새로운 observer가 추가되는 경우? 새로운 옵저버 클래스와 Battery 클래스 모두 변경해주어야 한다.

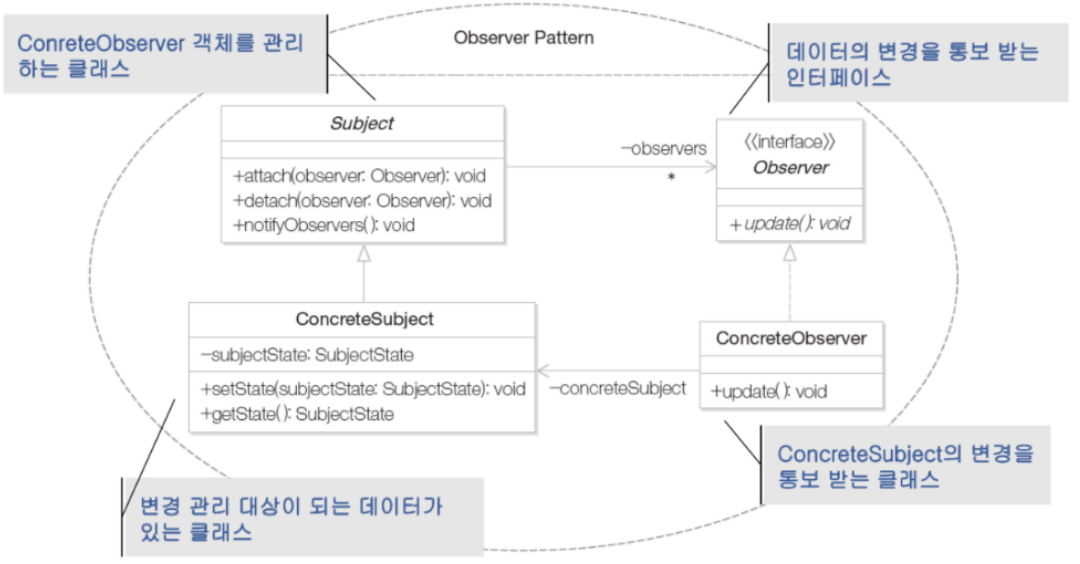


- 따라서 해당 코드는 OCP를 지키지 않고 확장에 닫혀있고 변경에 열려있는 취약한 코드이다.

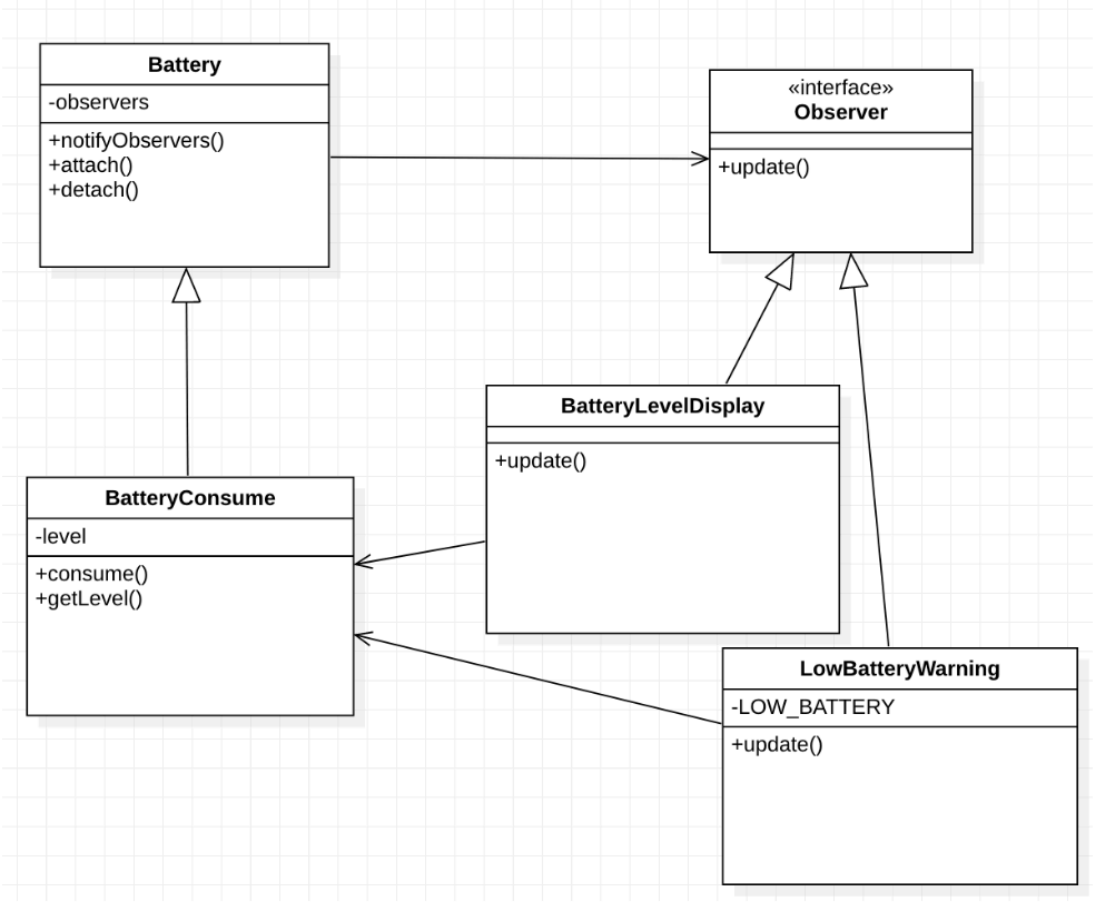
[취약점 개선 - Observer Pattern]

- Obserer Pattern은 아래의 4가지 요소로 구성된다.
  - Subject
  - Concrete Subject
  - Observer
  - Concrete Observer

Observer Pattern 클래스 다이어그램

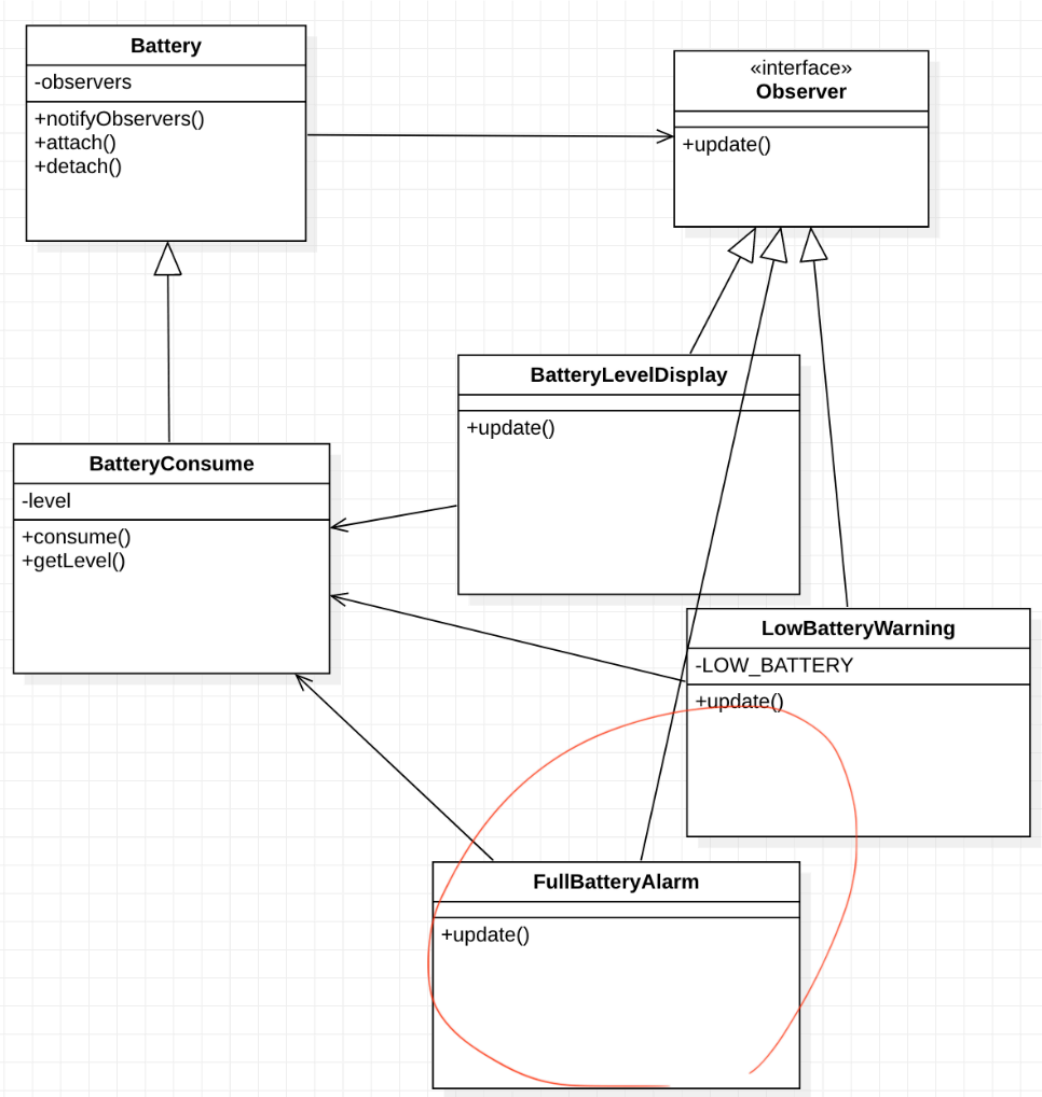


- 과제에서 주어진 소스코드를 Observer Pattern을 적용한 결과이다. (인터페이스 구현 부분에서 정확하게는 실선이 아닌 점선이 맞음.)
    - Subject : Battery
    - Concrete Subject : BatteryConsume
    - Observer : Observer
    - Concrete Observer : BatteryLevelDisplay, LowBatteryWarning

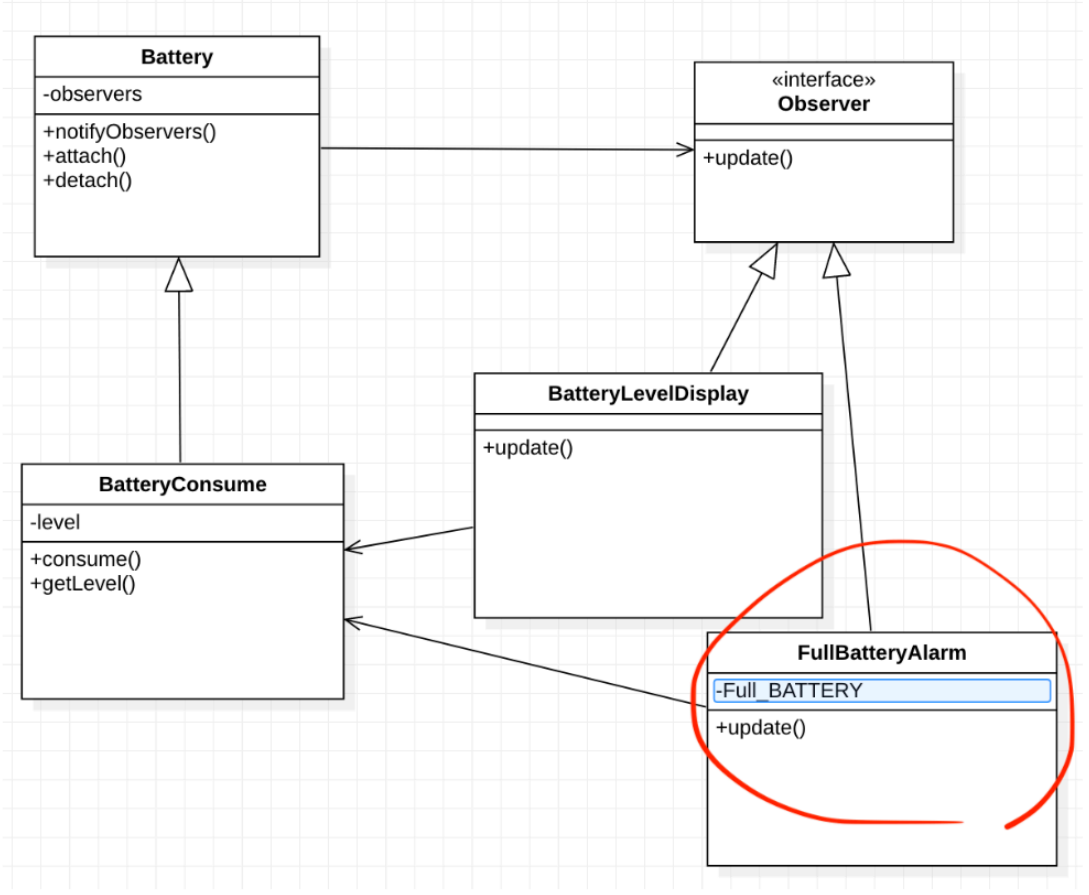


[개선된 패턴에 대한 분석]

- 해당 패턴을 적용한뒤에 얻을 수 있는 효과를 OCP의 측면에서 분석해보면,
    - 만약, 새로운 Observer가 추가될 경우, BatteryConsume, Battery, Observer에 대한 수정 없이 새로운 Observer의 추가만 있으면 된다.



- 만약, 기존의 Observer의 동작방식을 변경하려면, 원래는 Battery까지 함께 바뀌어야 했으나, 이제는 해당 Concrete Observer만 변경해주면 된다.



- 따라서 해당 설계는 OCP를 만족한다고 할 수 있다.

[전체 소스코드]

[Battery.java] :

```
import java.util.ArrayList;
import java.util.List;

// Subject Class (추상 클래스)
public abstract class Battery {

        //Observer저장
        private List<Observer> observers = new ArrayList<O

        //Observer추가
        public void attach(Observer observer) {
                observers.add(observer);
        }

        //Observer삭제
        public void detach(Observer observer) {
                observers.remove(observer);
        }

        //Observer에게 변경알림.
        public void notifyObservers() {
                for (Observer o : observers) o.update();
        }
}
```

옵저버들을 관리하고 옵저버들에게 알림을 주는 Subject 클래스이다.

[BatteryConsume.java] :

```
// Concrete Subject
public class BatteryConsume extends Battery {
        // 배터리 남은양
        private int level = 100;

        // 변경되는 부분 (알림이 필요한 부분)
        public void consume(int amount) {
                level -= amount;
                notifyObservers();
        }

        // concrete observer에서 부르기위함.
        public int getLevel() {return level;}
}
```

Subject Class를 상속받는 Concrete Subject 클래스이다. 실제로 배터리 양을 저장하고 있다.

[LowBatteryWarning.java] :

```
// Concrete Observer
public class LowBatteryWarning implements Observer {
        private static final int LOW_BATTERY = 30;
        private BatteryConsume battery;

        public LowBatteryWarning(BatteryConsume battery) {
                this.battery = battery;
        }

        // Observer를 실현한 관계, 필수적인 메서드
        // LOW_BATTERY 보다 Battery양이 낮아지면 경고메세지를 출력한
        @Override
        public void update() {
                // TODO Auto-generated method stub
                int level = battery.getLevel();
                if (level < LOW_BATTERY) {
                        System.out.println("<Warining Low
                }
        }

}
```

Concrete Observer클래스로 BatteryConsumer의 배터리가 LOW_BATTERY 이하가 되면 경고를 출력한다.

[Observer.java] :

```
public interface Observer {
        public void update();
}
```

Observer들의 기능을 담당하는 인터페이스다.

[Client.java] :

```
public class Client {
        public static void main(String[] args) {
                BatteryConsume battery = new BatteryConsum
                Observer batteryLevelDisplay = new Battery
                Observer lowBatteryWarning = new LowBatter

                // observer등록
                battery.attach(batteryLevelDisplay);
                battery.attach(lowBatteryWarning);

                // 배터리를 100->1퍼센트까지 연속적으로 감소시키기
                for (int i = 0; i < 99; i++) {
                battery.consume(1);
                }
        }
}
```

클라이언트 코드이다. 배터리를 100에서 1퍼센트까지 계속해서 감소시켜보았다.

[클라이언트 실행 결과 예시]

```
Level: 46
Level: 45
Level: 44
Level: 43
Level: 42
Level: 41
Level: 40
Level: 39
Level: 38
Level: 37
Level: 36
Level: 35
Level: 34
Level: 33
Level: 32
Level: 31
Level: 30
Level: 29
<Warining Low Battery : 29 Compared with 30
Level: 28
<Warining Low Battery : 28 Compared with 30
Level: 27
<Warining Low Battery : 27 Compared with 30
Level: 26
<Warining Low Battery : 26 Compared with 30
Level: 25
<Warining Low Battery : 25 Compared with 30
Level: 24
<Warining Low Battery : 24 Compared with 30
Level: 23
<Warining Low Battery : 23 Compared with 30
Level: 22
<Warining Low Battery : 22 Compared with 30
Level: 21
<Warining Low Battery : 21 Compared with 30
Level: 20
<Warining Low Battery : 20 Compared with 30
Level: 19
<Warining Low Battery : 19 Compared with 30
```

- 배터리가 순차적으로 감소하다가 30 이하가되면 배터리양과 함께 경고 메세지가 출력되는 것을 볼 수 있다.