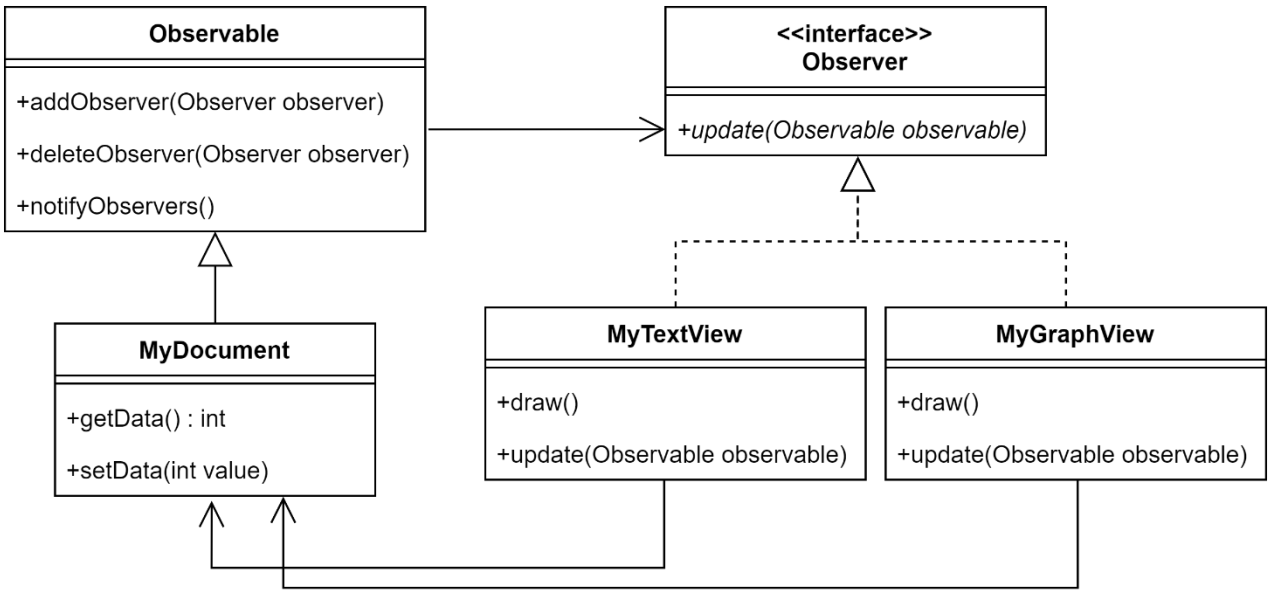
1) 개요



2) Observer.java

```java
package observer.e2;

interface Observer {
    void update(Observable observable);
}
```

3) Observable.java

```java
package observer.e2;

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

public class Observable {
    List<Observer> observers = new ArrayList<Observer>();

    void addObserver(Observer observer) {
        observers.add(observer);
    }

    void deleteObserver(Observer observer) {
        observers.remove(observer);
    }

    void notifyObservers() {
        for (int i = 0; i < observers.size(); ++i)
            observers.get(i).update(this);
    }
}
```

보통 Observer, Observable는 라이브러리에 이미 구현되어 있는 유틸러티 클래스이다.

라이브러리에 들어있는 유틸러티 클래스를 참조하는 것 때문에
유지보수성이 나빠지지 않는다.
(수정될 일이 거의 없는 클래스들이므로)

4) MyIntDocument.java

```java
package observer.e2;

public class MyIntDocument extends Observable {
    private int data;

    public int getData() {
        return data;
    }

    public void setData(int i) {
        data = i;
        notifyObservers();
    }
}
```

view 클래스를 참조하지 않는다.

5) MyTextView.java

```java
package observer.e2;

public class MyTextView implements Observer {
    public MyIntDocument document;

    public MyTextView(MyIntDocument doc) {
        document = doc;
        document.addObserver(this);
    }

    public void draw() {
        int data = document.getData();
        System.out.printf("View1: %dWn", data);
    }

    @Override
    public void update(Observable observable) {
        draw();
    }
};
```

6) MyGraphView.java

```java
package observer.e2;

public class MyGraphView implements Observer {
    MyIntDocument document;

    public MyGraphView(MyIntDocument doc) {
        document = doc;
        document.addObserver(this);
    }

    public void draw() {
        int data = document.getData();
        System.out.printf("View2: ");
        for (int i = 0; i < data; ++i)
            System.out.print('*');
        System.out.print('Wn');
    }

    @Override
    public void update(Observable observable) {
        draw();
    }
}
```

7) Example2.java

```
1  package observer.e2;
2
3  import java.util.Scanner;
4
5  public class Example2 {
6
7      public static void main(String[] args) {
8          try (Scanner scanner = new Scanner(System.in)) {
9              MyIntDocument doc = new MyIntDocument();
10             new MyTextView(doc);
11             new MyGraphView(doc);
12
13             for (;;) {
14                 int n;
15                 System.out.print("Wn set document data (0 = quit) : ");
16                 n = scanner.nextInt();
17                 if (n <= 0)
18                     break;
19                 doc.setData(n);
20             }
21         }
22     }
23
24 }
```

실행 사례

```
 set document data (0 = quit) : 10
View1: 10
View2: **********

 set document data (0 = quit) : 15
View1: 15
View2: ***************

 set document data (0 = quit) : 20
View1: 20
View2: ********************

 set document data (0 = quit) : 25
View1: 25
View2: *************************

 set document data (0 = quit) : 0
```

실행 결과는 동일하다.

document 소스코드에는 view에 대한 참조가 없고,
대신 observable에 대한 참조가 추가되었다.

view 소스코드는 여전히 document를 참조하고 있고,
추가로 observer에 대한 참조가 추가되었다.

유지보수성이 개선된 것이 맞는가?