객체지향개발론및실습

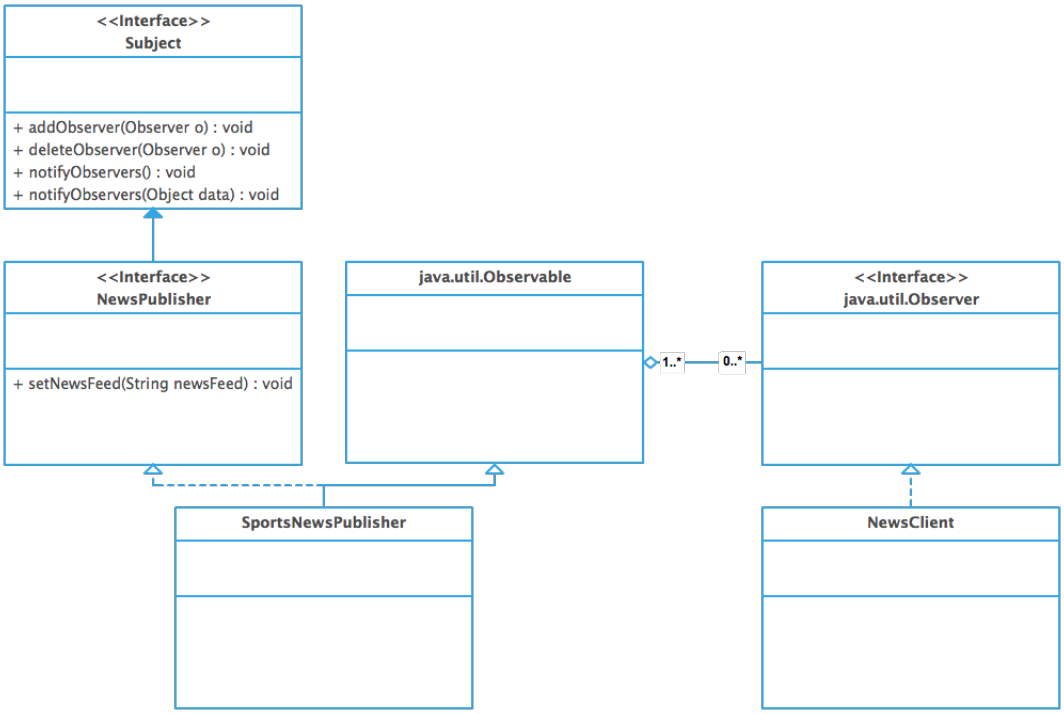
Laboratory 2. Observer 패턴: 실시간 뉴스 서버

1. 목적

- 1대다 관계를 형성해주며, 한 객체의 상태에 의존하는 모든 객체들에게 객체의 상태가 변화면 항상 자동으로 통보를 해주는 Observer 패턴을 실습해본다.

2. 개요

- 뉴스를 제공하는 사이트에 클라이언트들이 등록하면 새 뉴스가 있을 때마다 자동 통보해주는 응용을 고려하여 보자. 클라이언트의 경우 새 뉴스가 있을 때마다 문자메시지로 통보해 줄 수도 있고, 전자우편을 통해 통보해 줄 수도 있으며, 그 밖에 다양한 종류의 클라이언트가 존재할 수 있음
- 이 실습에서는 통보방식은 고려하지 않고, 새 뉴스를 통보받아 화면에 출력하여 주는 클라이언트만 생성한다.
- 자바 라이브러리에서 제공하는 java.util.Observeable과 java.util.Observer를 이용하여 구현함
- 구현하고자 하는 소프트웨어의 클래스 관계도는 다음과 같다.



<그림 2.1> 실시간 뉴스 제공 응용

3. Realtime News Agency Application

3.1 Subject 인터페이스

- 연산

  - `public void addObserver(Observer o)`
  - `public void deleteObserver(Observer o)`
  - `public void notifyObservers(Subject subject)`: pull을 고려
  - `public void notifyObservers(Object data)`: push를 고려
- Subject 인터페이스는 왜 정의할까?

3.2 NewsPublisher 인터페이스

- Subject 인터페이스를 상속받아 정의함
- 연산
  - `public void setNewFeed(String newFeed)`

3.3 SportsNewsPublisher 클래스

- java.util.Observable 클래스를 상속받고 NewsPublisher 인터페이스를 구현하여 정의함
- 새 뉴스가 발생하면 이를 클라이언트에게 통보하여 주는 클래스임
- 구성요소
  - 새 뉴스(latestFeed): **String** 타입
- 연산
  - `public void setNewFeed(String newFeed)`
    - 사후조건: 주어진 뉴스를 부모 클래스의 `setChanged` 메소드와 `notifyObservers`를 차례로 호출하여 등록된 클라이언트에게 통보하여 줌
    - 자바 라이브러리의 `Observerable` 클래스의 `notifyObservers` 메소드의 경우에는 push와 pull 두 가지 형태로 데이터를 전달할 수 있도록 해줌. 이 실습에서는 push 방식으로 구현함
    - `notifyObservers(Subject)`는 Observable에서는 인자가 없는 메소드로 정의되어 있음. 따라서 이 메소드는 재정의가 필요함

3.4 NewsClient 클래스

- java.util.Observer 인터페이스를 구현하여 정의함
- 새로 발생된 뉴스를 받아보는 클래스임
- 구성요소
  - id: **String** 타입. 클라이언트의 id를 나타냄.
- 연산
  - `public void update(Observable subject, Object arg)`
    - 사후조건: push방식으로 통보된 뉴스를 화면에 출력함

4. 실습 1

- 2개의 클래스를 구현하고 다음 TestProgram 프로그램을 실행해본다.
- 수업 시간에 deleteObserver가 잘 동작하기 위해서는 Observer에 equals 메소드의 재정의를 해주어야 한다고 하였음. 하지만 equals 메소드를 재정의하지 않아도 아래 테스트 프로그램은 잘 동작함. 그 이유는? 참고로 deleteObserver 소스 코드를 분석한 결과 equals 메소드를 사용하고 있음.
- NewsClient에 equals 메소드를 재정의하시오.

```
public class TestProgram {
    public static void main(String[] args) {
        NewsPublisher newsServer = new SportsNewsPublisher();
        NewsClient client1 = new NewsClient("sangjin");
        NewsClient client2 = new NewsClient("jinhee");
        newsServer.addObserver(client1);
        newsServer.addObserver(client2);
        newsServer.setNewFeed("리버풀 1: 맨유 0");
        newsServer.deleteObserver(client2);
        newsServer.setNewFeed("리버풀 2: 맨유 0");
    }
}
```

5. 실습 2

- 자바의 Observable 클래스를 상속받아 Subject 클래스를 만들면 자바는 다중상속을 제공하지 않으므로 다른 클래스를 상속받아야 하는 클래스는 Subject로 만들 수 없다. 이를 보완하기 위해 교안에 제시한 해결책을 사용할 수 있다. 이를 위해 다음 클래스의 정의가 필요함

```
public class ImprovedObservable extends Observable{
    public void setChanged(){
        super.setChanged();
    }
}
```

5.1 ITNewsPublisher 클래스

- ITNewsPublisher는 SportsNewsPublisher와 마찬가지로 NewsPublisher 인터페이스를 구현하여 정의하는데, 이 때 멤버로 ImprovedObservable 타입을 추가함
- TestProgram에서 SportNewsPublisher 객체를 ITNewsPublisher 객체로 바꿈
- 시간이 남으면 NewsClient를 push 방법을 사용하도록 기존 소스를 수정하시오. SportsNewsPublisher에 getter 추가와 setNewFeed의 수정이 필요하고, NewsClient의 update 메소드의 수정이 필요함

6. 숙제

- 클라이언트마다 뉴스를 받는 주기를 설정할 수 있도록 구현하고 싶음 즉, 클라이언트가 받는 주기를 1로 설정하면 새 뉴스가 발생할 때마다 통보받고, 이 값이 n이면 n개의 뉴스가 발생하면 통보받는 방식임. 주기의 시작은 등록한 시점으로 설정해야 함.
- 이 실습처럼 자바 라이브러리의 관찰자 패턴을 이용하면 관찰자 목록을 직접 제어하지 못하기 때문에 이와 같은 응용을 개발하기 힘듦. 따라서 Subject 인터페이스와 관련된 addObserver, deleteObserver, notifyObservers를 직접 구현하여야 함. 이 때 이 숙제에서는 push 방법을 이용하여 구현함
- 실습에서 정의한 SportsNewsPublisher나 ITNewsPublisher와 같은 구체화된 Subject 타입의 클래스와 NewsClient 클래스가 위와 같이 동작하도록 구현하여야 함
- 클라이언트는 뉴스를 받는 주기를 설정할 수 있어야 하며, 정보제공주체는 클라이언트의 주기, 이전 통보 시점 등을 알 수 있어야 함
- 제출: 전체 프로젝트를 압축하여 제출하고, notifyObservers와 update 메소드를 어떻게 모델링하였는지 설명하는 내용을 포함하시오.
- 테스트 프로그램 예시

```
public class TestProgram {
    public static void main(String[] args) {
        NewsPublisher newsServer = new SportsNewsPublisher();
        NewsClient client1 = new NewsClient("sangjin");
```

```
        NewsClient client2 = new NewsClient("jinhee");
        client1.setInterval(1);
        client2.setInterval(3);
        newsServer.setNewFeed("리버풀 1: 맨유 0");
        newsServer.setNewFeed("리버풀 2: 맨유 0");
        newsServer.setNewFeed("리버풀 3: 맨유 0");
        newsServer.setNewFeed("리버풀 3: 맨유 1");
        newsServer.deleteObserver(client1);
        newsServer.setNewFeed("리버풀 3: 맨유 2");
        newsServer.setNewFeed("리버풀 4: 맨유 2");
        newsServer.setNewFeed("루니 퇴장");
    }
}
```

- 출력 결과 예시

```
sangjin:
리버풀 1: 맨유 0

===
sangjin:
리버풀 2: 맨유 0

===
sangjin:
리버풀 3: 맨유 0

===
jinhee:
리버풀 1: 맨유 0
리버풀 2: 맨유 0
리버풀 3: 맨유 0

===
sangjin:
리버풀 3: 맨유 1

===
jinhee:
리버풀 3: 맨유 1
리버풀 3: 맨유 2
리버풀 4: 맨유 2

===
```