Theory: Observer

19 minutes 0 / 5 problems solved

§1. Design problem

When you release new videos on YouTube, update your profile info on Facebook or share a photo on Instagram, all of this information is automatically sent to your subscribers and friends. It means that an observable state is changed and the observers are notified. If you subscribe to a mailing list, you will get a message about new goods and stocks from the observable domain.

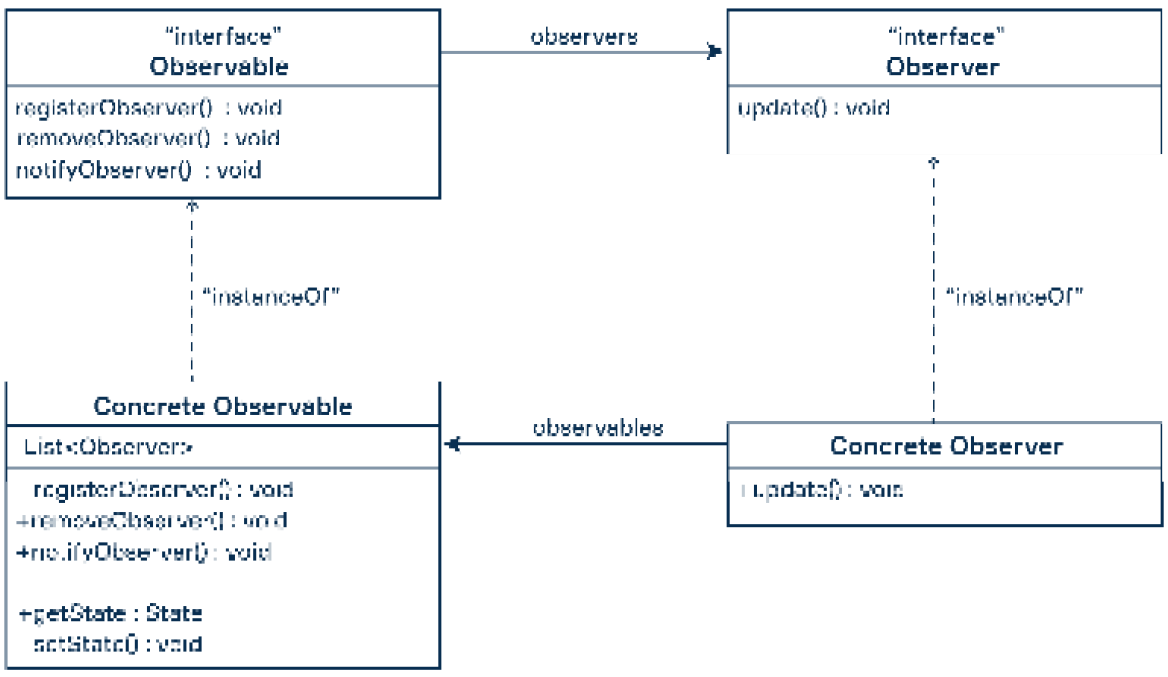
§2. Observer

The **observer pattern** is related to behavioral patterns. It is used to define the publisher-subscriber dependencies and the subscription mechanism so that when one object changes the state, all its dependents are notified and updated automatically.

The observer notifies all the interested parties about an event that has occurred or a change in its state. In many cases, the observer pattern is used to build low coupling – reducing the coherence of a class with its dependencies by destroying the connection of the initiator of some event with its handlers.

The observer pattern has the following components :

- Observable
- Concrete Observable
- Observer
- Concrete Observer



These four components carry out different functions:

1. Observable subscribes observers, removes them, and notifies them about the changes;
2. Concrete Observable implements Observable operations and describes some states;
3. Observer subscribes to Observable and listens to its notification;
4. Concrete Observer implements Observer interface and reacts on updating (Observable notification).

The observer pattern in JDK is available in `java.util.Observer` and `java.util.EventListener`.

§3. Practice example

Current topic:

Observer

Topic depends on:

✓ The concept of patterns

✗ Interface

Table of contents:

YouTube is a good demonstration of the Observer pattern. We have a YouTube channel (Observable) and its subscribers (Observers). Every subscriber will be notified when a new video is released.

The Observable interface describes the operation to add, remove and notify the observers:

```
public interface Observable {
    public void addObserver(Observer observer);
    public void removeObserver(Observer observer);
    public void notifyObserver();
}
```

YouTubeChannel is the concrete implementation of the Observable interface with a list of observers. Generic <Observer> provides low coupling between objects:

```
public class YoutubeChannel implements Observable {
    private ArrayList<Observer> observers = new ArrayList<>();

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

    @Override

    public void removeObserver(Observer observer) {

        observers.remove(observer);

    }



    public void releaseNewVideo(String video) {

        System.out.println("Release new video : " + video);

        notifyObserver();

    }



    @Override

    public void notifyObserver() {

        for(Observer observer: observers) {

            observer.update();

        }

    }

}
```

Observer interface updates action when Observable makes notifications to its observers:

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

Youtube Subscriber (Concrete Observer) will get a notification about a new video on the channel:

```
public class YoutubeSubscriber implements Observer {

    private Observable observable;

    public YoutubeSubscriber(Observable observable) {
        this.observable = observable;
    }

    @Override

    public void update() {

        System.out.println("New video on channel!");

    }

}
```

Here is a demo of the Observer pattern:

```
public class Main {
    public static void main(String[] args) {
        YoutubeChannel youtubeChannel = new YoutubeChannel();

YoutubeSubscriber subscriberA = new YoutubeSubscriber(youtubeChannel);

YoutubeSubscriber subscriberB = new YoutubeSubscriber(youtubeChannel);

YoutubeSubscriber subscriberC = new YoutubeSubscriber(youtubeChannel);
        youtubeChannel.addObserver(subscriberA);
        youtubeChannel.addObserver(subscriberB);
        youtubeChannel.addObserver(subscriberC);


youtubeChannel.releaseNewVideo("Design Patterns : Factory Method");

        youtubeChannel.releaseNewVideo("Design Patterns : Proxy");

        youtubeChannel.releaseNewVideo("Design Patterns : Visitor");

    }

}
```

§4. Conclusion

The observer pattern is applicable in the following cases:

- When changing one component influences other objects;
- When subscriber-publisher dependencies are present;
- When you need to have low coupling between the components.

Report a typo

64 users liked this theory. **1** didn't like it. **What about you?**

Start practicing

Comments (7) Hints (0) Useful links (1) Show discussion