Xideral

Java Academy

Week 3-Day 1

Observer pattern

Presented by:

Edgar Itzak Sánchez Rogers

Monterrey Nuevo León México at 27 august 2024

Introduction:

Observer is a design pattern of behavior that allows notify to different objects (observers) if the subject observed has changed or has executed an action. This design allows the subject and observers to be decoupled. The subject does not need to know anything specific about the observers.

Observer design pattern demonstration:

Subject and Observer interface:

```java
package com.edgaritzak.observerpattern;

public interface Subject {

    void notifyObservers();
    void addObserver(Observer o);
    Observer removeObserver(Observer o);

}
```

```java
package com.edgaritzak.observerpattern;

public interface Observer {

    void update(Channel c);

}
```

Channel class (Subject):

```java
package com.edgaritzak.observerpattern;
import java.util.ArrayList;

public class Channel implements Subject{

    static int counter =0;
    int id;
    String channelName;
    ArrayList<Observer> viewerList = new ArrayList<>();


    public Channel(String channelName) {
        super();
        id = counter++;
        this.channelName = channelName;
    }

    public void postVideo() {
        System.out.println("Video posted successfully!");
        notifyObservers();
    }

    @Override
    public void notifyObservers() {
        for(Observer o:viewerList) {
            o.update(this);
        }
    }

}
```

```
30      @Override
31      public void addObserver(Observer o) {
32          viewerList.add(o);
33      }
34
35      @Override
36      public Observer removeObserver(Observer o) {
37          viewerList.remove(o);
38          return o;
39      }
40
41      public String getChannelName() {
42          return channelName;
43      }
44
45      public void setChannelName(String channelName) {
46          this.channelName = channelName;
47      }
```

Viewer class (observer):

```
1  package com.edgaritzak.observerpattern;
2
3  import java.util.ArrayList;
4
5  public class Viewer implements Observer{
6
7      static int counter =0;
8      int id;
9      String Username;
10     ArrayList<String> notificationList = new ArrayList<>();
11
12     public Viewer(String username) {
13         super();
14         id = counter++;
15         Username = username;
16     }
17
18     @Override
19     public void update(Channel c) {
20         notificationList.add(c.getChannelName()+" uploaded a new video.");
21     }
22 }
```

Main class

```java
package com.edgaritzak.observerpattern;

public class Main {
    public static void main(String[] args) {

        //Create accounts
        Viewer viewer1 = new Viewer("User1");
        Viewer viewer2 = new Viewer("User2");
        Viewer viewer3 = new Viewer("User3");
        Viewer viewer4 = new Viewer("User4");

        //Create channels
        Channel channel1 = new Channel("Veritasium");
        Channel channel2 = new Channel("Vsauce");

        //Users subscribe to channel 1
        channel1.addObserver(viewer1);
        channel1.addObserver(viewer2);
        channel1.addObserver(viewer3);

        //Users subscribe to channel 2
        channel2.addObserver(viewer2);
        channel2.addObserver(viewer3);
        channel2.addObserver(viewer4);

        //Channel1 posts a video
        channel1.postVideo();

        //Check viewers notifications viewer 1
        System.out.println("\nViewer 1 notification list:");
        for (String s : viewer1.notificationList) {
            System.out.println(s);
        }

        System.out.println("");
        //Channel2 posts a video
                channel2.postVideo();

        //Check viewers notifications viewer 2
        System.out.println("\nViewer 2 notification list:");
        for (String s : viewer2.notificationList) {
            System.out.println(s);
        }
    }
```

Output:

```
Video posted successfully!

Viewer 1 notification list:
Veritasium uploaded a new video.

Video posted successfully!

Viewer 2 notification list:
Veritasium uploaded a new video.
Vsauce uploaded a new video.
```

Explanation:

When a channel executes the method postVideo() it prints “Video posted successfully” then calls the method notifyObservers() to notify each viewer subscribed to the channel, notifyObservers() send the channel that triggered the update, with this object, the update() method adds a notification to the viewer.

Conclusion:

Observer design pattern is an essential tool in the development of complex and dynamic software systems. Its ability to establish a dependency relationship between objects, where changes in a subject trigger automatic updates on their observers, not only facilitates event management and data synchronization.

References:

[1] Observer (refactoring.guru)

[2] Observer pattern - Wikipedia