University of Information Technology and Sciences (UITS)

Department of Information Technology

Lab Report : 3

IT-324 : Design Pattern Lab

Observer Pattern

Submitted To:

Sifat Nawrin Nova
Lecturer,
Department of IT, UITS
Email:sifat.nawrin@uits.edu.bd

Submitted By:

Name:Nazmul Zaman
Student ID:2014755055
Department of IT, UITS

June 12, 2022 Department of IT, UITS © All rights reserved.

Contents

1 Abstraction

In this lab report we learn about Observer Pattern method and how it's work and why we used this method. In Java, Observer pattern is used when there is one-to-many relationship between objects such as if one object is modified, its depenedent objects are to be notified automatically. Observer pattern falls under behavioral pattern category.
One to many dependency is between Subject(One) and Observer(Many).
There is dependency as Observers themselves don't have access to data. They are dependent on Subject to provide them data.

2 Theory

Observer pattern Also known as Event-subscriber, Listener. Observer is a behavioral design pattern that allows you to specify a subscription mechanism fornumerous objects to be notified of any events that occur on the object they're following.
According to the Observer pattern, the publisher class should provide a subscription mechanism so that individual objects can subscribe to or unsubscribe from a stream of events originating from that publisher.

3 Objective

In this section we Discuss the remarks of using this pattern.Discuss the remarks of using this pattern.

Here Observer and Subject are interfaces(can be any abstract super type not necessarily java interface).
1.All observers who need the data need to implement observer interface.
2.notify() method in observer interface defines the action to be taken when the subject provides it data
3.The subject maintains an observerCollection which is simply the list of currently registered(subscribed) observers.
4.registerObserver(observer) and unregisterObserver(observer) are methods to add and remove observers respectively.
5.notifyObservers() is called when the data is changed and the observers need to be supplied with new data.

4 When to use this pattern

You should consider using this pattern in your application when multiple objects are dependent on the state of one object as it provides a neat and well tested design for the same.

5 Source Code

5.1 (Lab3 class/Main Class)

```java

package lab3;

public class Lab3 {


    public static void main(String[] args) {
        // TODO code application logic here

        Page Nazmul =  new Page();

        Page Zaman =  new Page ();


        Follower obj1 = new Follower("Nazmul");
        Follower obj2 = new Follower("Zaman");
        Follower obj3 = new Follower("Shael");
        Follower obj4 = new  Follower ("Haider");
        Follower  obj5 = new  Follower ("Afif");


        Nazmul.follow(obj2);

        Zaman.follow(obj1);
        Zaman.follow(obj2);
        Zaman.follow(obj3);
        Zaman.follow(obj4);
        Zaman.follow(obj5);

        Nazmul.unfollow(obj3);


        Nazmul.upload("New documentary on whales!");
        Zaman.upload("How to learn design pattern");







    }

}
```

?

5.2 (Follower Class)

```

package lab3;

public class Follower{
    private String name;

    Page page = new Page();

    public Follower (String name)
    {
        this.name = name;

    }

        public void update()
    {
        System.out.println("Hello! "+ name +",new video
    uploaded : "+ Page.videotitle);


    }

         public void  followpage(Page ch)
    {
      page = ch;


    }





}
```

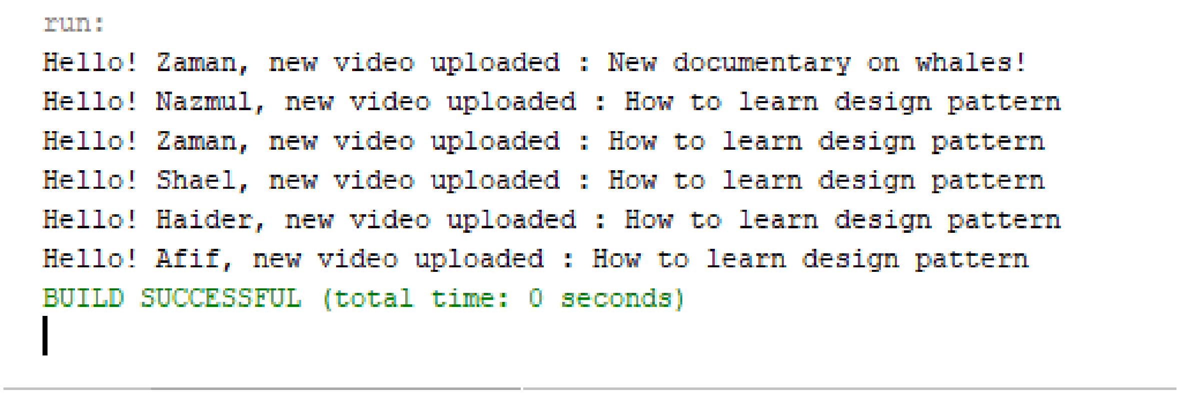
?

5.3 (Page Class)

```

package lab3;

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


public class Page {



    private List<Follower> subs  = new ArrayList<>();

    static String videotitle;

    public void follow (Follower sub)
    {

        subs.add(sub);
    }

     public void unfollow(Follower sub)
    {

        subs.remove(sub);
    }

        public void notifysub()
    {


        for(Follower sub : subs)
        {
            sub.update();
        }


    }


           public void upload (String titleman)
    {

        this.videotitle = titleman;

        notifysub();
    }


}
```

?

5.4 Output



```
run:
Hello! Zaman, new video uploaded : New documentary on whales!
Hello! Nazmul, new video uploaded : How to learn design pattern
Hello! Zaman, new video uploaded : How to learn design pattern
Hello! Shael, new video uploaded : How to learn design pattern
Hello! Haider, new video uploaded : How to learn design pattern
Hello! Afif, new video uploaded : How to learn design pattern
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figure 1: Output

6 Conclusion

In this lab report we learn about Observer pattern in practically. How we used observed design pattern with real life example like locial media, RSS feeds, email subscription in which you have the option to follow or subscribe and you receive latest notification.
Another way to say,The observer pattern provides the kind of power we've come to expect from patterns. It allows for multiple observing clients to be updated as and when data changes occur.

7 References

1.https://www.tutorialspoint.com/design$_pattern/singleton_pattern.htm$
2.$https://www.geeksforgeeks.org/singleton-design-pattern-introduction$
3.$https://www.javatpoint.com/singleton-design-pattern-in-java$