DESIGN PATTERNS - CPIT252

LAB_8

Shehab

CONTENTS

Classes 2

Subject 2

Observer 2

EmailObserver 2

FacebookObserver 2

TwitterObserver 3

WhatsaooObserver 3

MessageSubject 3

Main 4

Qustions 4

Explain how the observer design pattern simlified the process of adding a new notification service in this implementation? 4

Output 4

CLASSES

SUBJECT

```
public interface Subject {
    public void subscribe(Observer o);
    public void unsubscribe(Observer o);
    public void notifyUpdate(String m);
}
```

OBSERVER

```
public abstract class Observer {

    private String recipient;

    public String getRecipient() {
        return recipient;
    }

    public void setRecipient(String recipient) {
        this.recipient = recipient;
    }

    public abstract void update(String m);
}
```

Here I just made a normal setter and getter for recipient.

EMAILOBSERVER

```
public class EmailObserver extends Observer {

    public EmailObserver(String recipient){
        super.setRecipient(recipient);
    }

    @Override
    public void update(String m) {
        System.out.println("Email Observer :: @" + getRecipient() + " " + m);

    }
}
```

FACEBOOKOBSERVER

```
public class FacebookObserver extends Observer{
    public FacebookObserver(String recipient) {
        super.setRecipient(recipient);
    }

    @Override
    public void update(String message) {
        System.out.println("Facebook Observer :: @" + getRecipient() + " " + message);
    }
}
```

TWITTEROBSERVER

```
public class TwitterObserver extends Observer{
    public TwitterObserver(String recipient){
        super.setRecipient(recipient);
    }
    @Override
    public void update(String message){
        System.out.println("Twitter Observer :: @" + getRecipient() + " " + message);
    }
}
```

WHATSAOOOBSERVER

```
public class WhatsappObserver extends Observer{
    public WhatsappObserver(String recipient) {
        super.setRecipient(recipient);
    }

    @Override
    public void update(String message) {
        System.out.println("Whatsapp Observer :: @" + getRecipient() + " " + message);
    }
}
```

MESSAGESUBJECT

```
java.util.ArrayList;

public class MessageSubject implements Subject{

    private ArrayList<Observer> list = new ArrayList<>();

    @Override
    public void subscribe(Observer o) {
        list.add(o);
    }

    @Override
    public void unsubscribe(Observer o) {
        list.remove(o);
    }

    @Override
    public void notifyUpdate(String m) {
        for (int i = 0; i < list.size(); i++) {
            list.get(i).update(m);
        }
    }
}
```

Here I've created an array list of Observer type for subscribe method it adds the object and for unsubscribe it do remove it. For notify it loops through the array list and update all the messages.

MAIN

```
public class Main {

    public static void main(String[] args) {
        Observer twObserver = new TwitterObserver("CPIT_252_price_watcher");
        Observer fbObserver = new FacebookObserver("CPIT_252_price_watcher");
        Observer emailObserver = new EmailObserver("PUT YOUR EMAIL HERE");
        Observer waObserver = new WhatsappObserver("9660000000000");

        Subject s = new MessageSubject();

        s.subscribe(twObserver);
        s.subscribe(fbObserver);
        s.subscribe(emailObserver);

        s.notifyUpdate("The price has dropped from SAR 299.99 to SAR 249.99.");

        System.out.println("\nChange observers: unsubscribed email and subscribed
Whatsapp\n");
        s.unsubscribe(emailObserver);
        s.subscribe(waObserver);

        s.notifyUpdate("The price has dropped from SAR 249.99 to SAR 199.99.");

    }
}
```

QUSTIONS

EXPLAIN HOW THE OBSERVER DESIGN PATTERN SIMLIFIED THE PROCESS OF ADDING A NEW NOTIFICATION SERVICE IN THIS IMPLEMENTATION?

First, we are creating classes the do implement the observer abstract class. So, all these classes do have a pattern to follow, after that with the help of an object from class MessageSubject we are keeping track of all objects that has been created by adding them to the array list that we've created earlier in the MessageSubject class. Therefore, we can manipulate these objects the way we want.

OUTPUT

```
Twitter Observer :: @CPIT_252_price_watcher The price has dropped from SAR 299.99 to SAR 249.99.
Facebook Observer :: @CPIT_252_price_watcher The price has dropped from SAR 299.99 to SAR 249.99.
Email Observer :: @PUT YOUR EMAIL HERE The price has dropped from SAR 299.99 to SAR 249.99.

Change observers: unsubscribed email and subscribed Whatsapp

Twitter Observer :: @CPIT_252_price_watcher The price has dropped from SAR 249.99 to SAR 199.99.
Facebook Observer :: @CPIT_252_price_watcher The price has dropped from SAR 249.99 to SAR 199.99.
Whatsapp Observer :: @9660000000000 The price has dropped from SAR 249.99 to SAR 199.99.
```