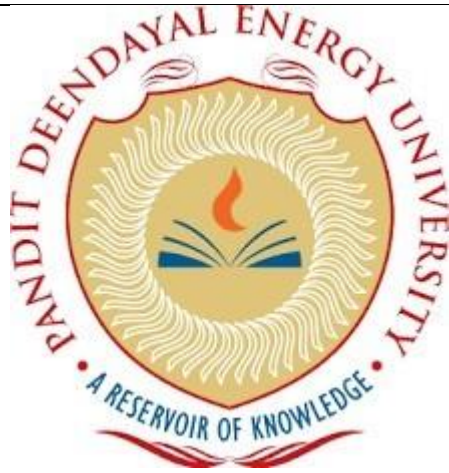


| | | | | |
|---|--|--|--------------------------|---------------------|
|  | Course Name: Design Patterns/Thinking LAB | | EXPERIMENT NO. 14 | |
| | Course Code: 20CP210P Faculty: Dr. Ketan Sabale | | Branch: CSE | Semester: IV |
| Submitted by: Rhythm Shah Roll no: 22BCP071 | | | | |

Objective: To familiarize students with standard Behavioral design patterns.
Experiment: Explain the Observer design pattern and write a program using any object-oriented programming language to demonstrate the working of Observer design pattern.

Theory:-

Observer is a behavioral design pattern that lets you define a subscription mechanism to notify multiple objects about any events that happen to the object they're observing.

A behavioral design pattern called the Observer Design Pattern establishes a one-to-many dependency between objects so that all of the dependents, or observers, of the subject object are automatically updated and notified when the subject object changes state.

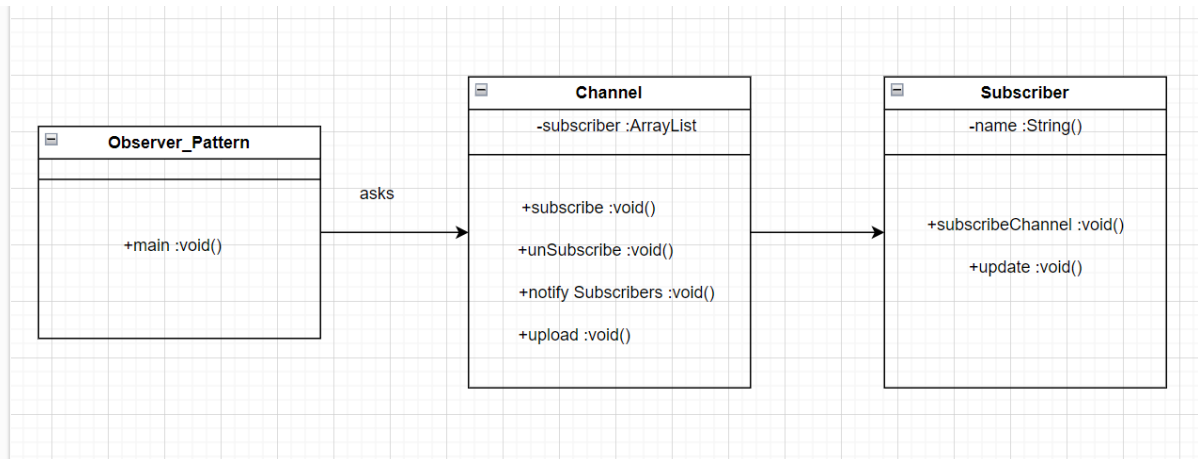
It is mostly concerned with how objects interact and communicate with one another, with a particular emphasis on how items respond to changes in the states of other objects.

Implementation:-

The code here is the implementation for YouTube channel. When a person who owns the channel uploads a new video. It notifies to all the subscribers who has subscribed the channel so that the owner do not need to tell each and everyone.

Two classes are made one for Channel and one for Subscriber and all the necessary methods are applied inside the classes like Subscribe , Unsubscribe , notifySubscribers and upload. One main method to create objects.

UML DIAGRAM:-



Code:-

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

class Channel {
    List <Subscriber> subs = new ArrayList<>();
    private String title;

    public void subscribe(Subscriber sub) {
        subs.add(sub);
    }

    public void unSubscribe(Subscriber sub) {
        subs.remove(sub);
    }

    public void notifySubscribers() {
        for(Subscriber sub:subs)
        {
            sub.update();
        }
    }

    public void upload(String title)
    {
```

```

        this.title = title;
        notifySubscribers();
    }
}

class Subscriber {
    private String name;
    private Channel channel = new Channel();

    public Subscriber(String name) {
        this.name = name;
    }

    public void update() {
        System.out.println("Hello " + name + " Your Video is Uploaded");
    }

    public void subscribeChannel(Channel ch) {
        channel = ch;
    }
}

public class Observer_pattern{

    public static void main(String[] args) {

        Channel design_pattern = new Channel();

        Subscriber s1 = new Subscriber("Rhythm");
        Subscriber s2 = new Subscriber("Anshul");
        Subscriber s3 = new Subscriber("Parth");
        Subscriber s4 = new Subscriber("Smit");
        Subscriber s5 = new Subscriber("Ishan");

        design_pattern.subscribe(s1);
        design_pattern.subscribe(s2);
        design_pattern.subscribe(s3);
        design_pattern.subscribe(s4);
        design_pattern.subscribe(s5);
        design_pattern.unsubscribe(s5);

        s1.subscribeChannel(design_pattern);
        s2.subscribeChannel(design_pattern);
        s3.subscribeChannel(design_pattern);
        s4.subscribeChannel(design_pattern);
        s5.subscribeChannel(design_pattern);

        design_pattern.upload("Learn Python");
    }
}

```

```
}  
  
}
```

Output:-

```
PS E:\Fourth sem\Design pattern lab> cd "e:\  
tern }  
Hello Rhythm Your Video is Uploaded  
Hello Anshul Your Video is Uploaded  
Hello Parth Your Video is Uploaded  
Hello Smit Your Video is Uploaded  
PS E:\Fourth sem\Design pattern lab>
```