LAB ASSIGNMENT 16

OBSERVER DESIGN PATTERN

Observer design pattern is useful when you are interested in the state of an object and want to get notified whenever there is any change.

CODE:

- InstrumentOrder.java

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

public class InstrumentOrder {

    private List<Observer> observers = new ArrayList<Observer>();
    private int order;

    public int getOrder() {
        return order;
    }

    public void setOrder(int order) {
        this.order = order;
        notifyAllObservers();
    }

    public void attach(Observer observer){
        observers.add(observer);
    }

    public void notifyAllObservers(){
        for (Observer observer : observers) {
            observer.update();
        }
    }
}
```

- Observer.java

```java
public abstract class Observer {
    protected InstrumentOrder instrumentOrder;
    public abstract void update();
}
```

- OrderConfirmed.java

```java
public class OrderConfirmed extends Observer{
    public OrderConfirmed(InstrumentOrder instrumentOrder) {
        this.instrumentOrder = instrumentOrder;
        this.instrumentOrder.attach(this);
    }

    @Override
    public void update() {
        System.out.println("Instrument Order confirm");
    }
}
```

- OrderDelivered.java

```java
public class OrderDelivered extends Observer{
    public OrderDelivered(InstrumentOrder instrumentOrder) {
        this.instrumentOrder = instrumentOrder;
        this.instrumentOrder.attach(this);
    }

    @Override
    public void update() {
        System.out.println("Instrument Order deliver");
    }
}
```

- OrderCancelled.java

```java
public class OrderCancelled extends Observer{
    public OrderCancelled(InstrumentOrder instrumentOrder) {
        this.instrumentOrder = instrumentOrder;
        this.instrumentOrder.attach(this);
    }

    @Override
    public void update() {
        System.out.println("Instrument Order Cancel");
    }
}
```

- Client.java

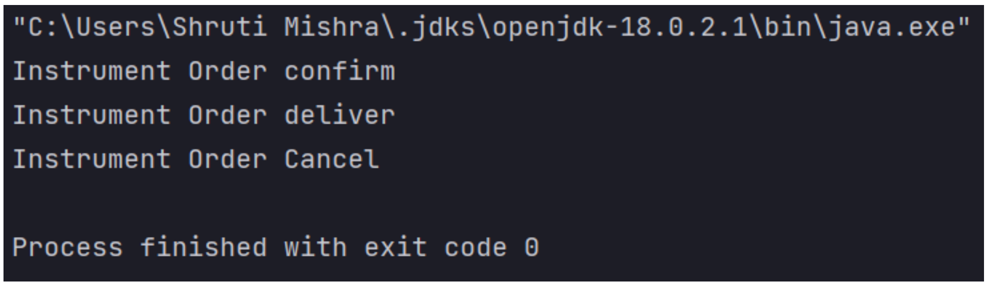
```java
public class Client {
    public static void main(String[] args) {
        InstrumentOrder instrumentOrder = new InstrumentOrder();

        new OrderConfirmed(instrumentOrder);
        new OrderDelivered(instrumentOrder);
        new OrderCancelled(instrumentOrder);


        instrumentOrder.setOrder(1);

    }
}
```

OUTPUT:



```
"C:\Users\Shruti Mishra\.jdks\openjdk-18.0.2.1\bin\java.exe"
Instrument Order confirm
Instrument Order deliver
Instrument Order Cancel

Process finished with exit code 0
```