18__10-09-2022

Saturday, 10 September 2022 8:01 PM

Observer design pattern

↳ Asyn commⁿ

↳ non-blocking commⁿ

Implementation

3 Actor classes – { Subject ✓, Observer ✓, Client }

Subject

↳ Subject is an object having methods to attach and detach observers to a client object.

Eg- subscribe and unsubscribe

Observer

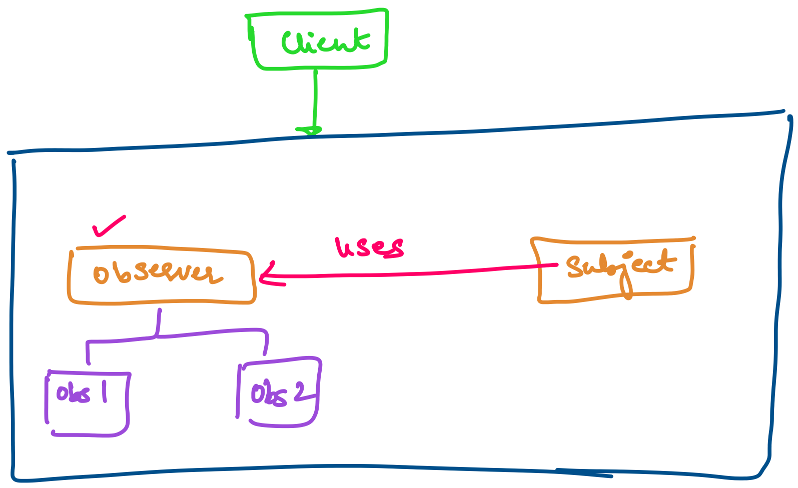
↳ Abstract class. (or interface)

↳ subject class will be extending Observer class.

Client

↳ main method

↳ will use subject and concrete class object



Advantages -

1. It provides the support for broad-cast type communication.

2. It describes coupling between the objects and the observers.

Implementation guidelines –

1. When the change of state in one object must be reflected in another object without keeping the objects tightly coupled.

2. When the framework we write and need to be enhanced in future with new observers with minimal changes.

for a phone, different states

could be—

→ only 5 left in stock

→ " 4 " " "

→ " 3 " " "

→ " 2 " " "

→ " 1 " " "

→ out of stock

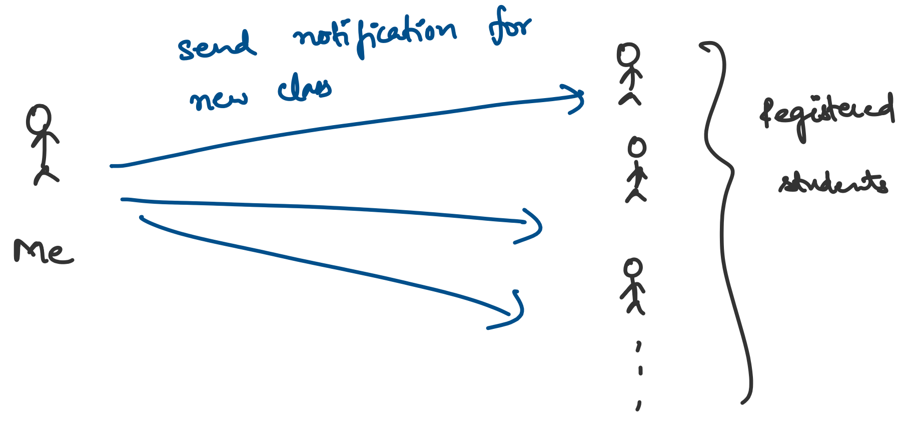
→ Back in stock

Different observers for the phone

can be—

- Availability → If phone is available or not
- Price
  ↳ for price change

Problem statement



Steps -

1. Create an interface for the session.

I Session —

- Register ()
  - ↳ method to allow student registration
- Unregister ()
  - ↳ method to unregister from the class
- Notify Students ()
  - ↳ Notify all the students about the class.

GetUpdate ()

↳ students can call and ask about the class.

2. Create interface for the observer.

IObserver student

→ update ()

↳ This method can be used to update the student.

→ set Class ()

↳ Make students aware that they have to take the class.

→ getName ()

↳ This will return the class topic to be discussed

3. create class – Batch Subscriber
$\uparrow$
implements IObserver Student

4. Create class – Batch
$\hookrightarrow$ implement ISession

In this create a list of observer students.

```
public interface ISession {
   //Method to allow student registration
   public void register(IObserverStudent student);

   //Method to unregister the class
   public void unregister(IObserverStudent student);

   //Notify all the students about the class
   public void notifyStudents();

   //Student can call and ask about the class
   public String getUpdate(IObserverStudent student);
}
```

```
public interface IObserverStudent {
   //This method can be used to update the students
   public void update();
   //Inform students about the class
   public void setClass(ISession session);

   //this returns topic name to be discussed
   public String getName();
}
```

```
public class BatchSubscriber implements IObserverStudent {
   private String name;
   private ISession session;

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

   @Override
   public void update() {
      String sessionPlan = session.getUpdate(this);
      System.out.println("Fetched the session plan of the class");
   }

   @Override
   public void setClass(ISession session) {
      this.session = session;
   }

   @Override
   public String getName() {
      return this.name;
   }
}
```

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

public class Batch implements ISession {
   List<IObserverStudent> registeredStudents;
   private String studyTopic;
```

```
public Batch()
{
   //in the beginning there will be no students
   this.registeredStudents = new ArrayList<>();
}

//Method to register for the new class
@Override
public void register(IObserverStudent student) {
   System.out.println("Registering student : " + student.getName());
   this.registeredStudents.add(student);
}

//Method to unregister from the class
@Override
public void unregister(IObserverStudent student) {
   System.out.println("Removing student : " + student.getName());
   this.registeredStudents.remove(student);
}

//Method to notify all the registered students about the new class
@Override
public void notifyStudents() {
   for (IObserverStudent observerStudent : registeredStudents)
   {
      observerStudent.update();
   }
}

//Student will be calling this method to know the session details
@Override
public String getUpdate(IObserverStudent student) {
   //check if the student is registered
   if(registeredStudents.contains(student))
   {
      return this.studyTopic;
   }
   return null;
}

//Method to update the topics of discussion for every session
public void addStudyTopic(String studyTopic)
{
   System.out.println("Added the study topic : " + studyTopic);
   this.studyTopic = studyTopic;

   //Notify all the registered students
   notifyStudents();
}
```

```
}
```

```
public class Program {
   public static void main(String[] args) {
      //Create a batch
      Batch batch = new Batch();

      //Create students
      IObserverStudent student1 = new BatchSubscriber("StudentName1");
      IObserverStudent student2 = new BatchSubscriber("StudentName2");
      IObserverStudent student3 = new BatchSubscriber("StudentName3");
      IObserverStudent student4 = new BatchSubscriber("StudentName4");
      IObserverStudent student5 = new BatchSubscriber("StudentName5");

      //Registering students to the course
      batch.register(student1);
      batch.register(student2);
      batch.register(student3);
      batch.register(student4);

      //Attaching the teacher to each student
      student1.setClass(batch);
      student2.setClass(batch);
      student3.setClass(batch);
      student4.setClass(batch);

      //Add study topic for the class
      batch.addStudyTopic("Observer pattern");
   }
}
```

Output:

```
Registering student : StudentName1
Registering student : StudentName2
Registering student : StudentName3
Registering student : StudentName4
Added the study topic : Observer pattern
Fetched the session plan of the class
Fetched the session plan of the class
Fetched the session plan of the class
```

Fetched the session plan of the class

Another eg —

1. Create subject class.

I Subject —
- Register ()
- unregister ()
- Notify ()

Subject → implements I Subject
- → private variable
  - Product Name
  - Availability
- → GetAvailability ()
- → Set Availability ()

```
public interface ISubject {
   public void register(IObserver observer);
   public void unregister(IObserver observer);
   public void notifyObserver();
}
```

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

public class Subject implements ISubject {
   private final List<IObserver> observerList = new ArrayList<>();

   private String productName;
   private String availability;

   public Subject(String productName, String availability) {
      this.availability = availability;
      this.productName = productName;
   }

   public String getAvailability()
   {
      return this.availability;
   }

   public void setAvailability(String availability)
   {
      this.availability = availability;
      System.out.println("Availability changed from out of stock to available");
      notifyObserver();
   }

   @Override
   public void register(IObserver observer) {
      System.out.println("Observer added : " + observer.getName());
      observerList.add(observer);
   }

   @Override
   public void unregister(IObserver observer) {
      System.out.println("Observer removed");
      observerList.remove(observer);
   }
```

```
    @Override
    public void notifyObserver() {
        System.out.println("Product name : " + productName + " is now available, notifying all re

        for(IObserver observer : observerList)
        {
            observer.update(availability);
        }
    }
}
```

```
public interface IObserver {
    public void update(String availability);
    public String getName();
}
```

```
public class Observer implements IObserver {
    private String userName;

    public Observer(String userName, ISubject subject) {
        this.userName = userName;
        subject.register(this);
    }

    @Override
    public void update(String availability) {
        System.out.println("Hello " + userName + ". Product is now " + availability);
    }

    @Override
    public String getName() {
        return this.userName;
    }
}
```

```
public class Program {
    public static void main(String[] args) {
        Subject phone = new Subject("Mobile", "Out of stock");
        Subject tv = new Subject("Television", "Out of stock");

        Observer user1 = new Observer("User 1", phone);
        Observer user2 = new Observer("User 2", phone);
        Observer user3 = new Observer("User 3", phone);
        Observer user4 = new Observer("User 4", phone);
        Observer user5 = new Observer("User 5", phone);
        Observer user6 = new Observer("User 1", tv);
        Observer user7 = new Observer("User 5", tv);

        System.out.println("Phone's current state : " + phone.getAvailability());

        phone.setAvailability("Available");

        System.out.println();

        System.out.println("Tv's current state : " + phone.getAvailability());

        tv.setAvailability("Available");
    }
}
```

Output:

```
Observer added : User 1
Observer added : User 2
Observer added : User 3
Observer added : User 4
Observer added : User 5
Observer added : User 1
Observer added : User 5
Phone's current state : Out of stock
Availability changed from out of stock to available
Product name : Mobile is now available, notifying all registered users
Hello User 1. Product is now Available
Hello User 2. Product is now Available
Hello User 3. Product is now Available
Hello User 4. Product is now Available
Hello User 5. Product is now Available

Tv's current state : Available
Availability changed from out of stock to available
Product name : Television is now available, notifying all registered users
Hello User 1. Product is now Available
Hello User 5. Product is now Available
```