Observer Design Pattern

Through Maya Mnaizel

It is a behavioral design pattern, that notifies multiple objects, or subscribers, about any events that happen to the object they’re observing, or the publisher

It defines a one-to-many dependency between objects so that when one object changes its state, all dependents are notified and updated automatically.

This pattern is widely used to implement distributed event-handling systems

Example:

A notification service inside a store

In the Observer pattern, there are three main components:

- Subject: this is the object that maintains a list of dependents, known as observers, and notifies them of any changes in its state.
- Observer: This is the interface or abstract class that defines the update method. The concrete observers implement this interface or extend the abstract class to receive notification from the subject
- Concrete Subject: This is the concrete implementation of the Subject interface. It maintains the state of interest and notifies observers when the state changes

- Concrete Observer: This is the concrete implementation of the observer registers itself with a concrete subject to receive updates

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


interface Observer{
  void update(String message)
}

interface Subject{
  void addObserver(Observer observer)
  void removeObserver(Observer observer)
  void notifyObservers(String message)
}
//Object being observed
class ConcreteSubject implements Subject{
  private List<Observer> observers = new ArrayList<>();
  private String state;

  public void setState(String state){
    this.state = state;
    notifyObservers("State updated to: " + state);
  }

  @Override
  public void addObserver(Observer observer){
    observers.add(observer);
  }

  @Override
  public void removeObserver(Observer observer){
    observers.remove(observer);
  }

  @Override
  public void notifyObservers(String message){
    for(Observer observer : observers) {
      observer.update(message);
    }
  }
}
//the object that receives notification when the state of the subject changes
class ConcreteObserver implements Observer{
  private String name;

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

  @Override
```

```
  public void update(String message){
    System.out.println(name + " received message: " + message):
  }
}

public class ObserverPatternExample {
    public static void main(String[] args) {
        ConcreteSubject subject = new ConcreteSubject();

        ConcreteObserver observer1 = new ConcreteObserver("Observer 1");
        ConcreteObserver observer2 = new ConcreteObserver("Observer 2");

        subject.addObserver(observer1);
        subject.addObserver(observer2);

        subject.setState("New State 1");
        subject.setState("New State 2");

        subject.removeObserver(observer1);

        subject.setState("New State 3");
    }
}
```