Chapter 3

What is Observer Pattern?

Think of it like **subscribing to a YouTube channel**:

- When your favorite YouTuber uploads a new video, **all subscribers get notified**
- You don't have to keep checking their channel manually
- If you unsubscribe, you stop getting notifications
- The YouTuber doesn't need to know who you are personally - they just notify everyone who subscribed

How It Works

The Observer pattern has two main parts:

Subject (Observable) - The thing being watched (like the YouTube channel) **Observer** - The things that want to know when something changes (like the subscribers)

Simple Example - Weather Station

Let's say you have a weather station that measures temperature, and different devices want to know when the temperature changes.

java

```java
// Observer interface - what all observers must have
interface Observer {
  void update(float temperature);
}

// Subject interface - what all subjects must have
interface Subject {
  void addObserver(Observer observer);
  void removeObserver(Observer observer);
  void notifyObservers();
}

// The weather station (Subject)
class WeatherStation implements Subject {
  private List<Observer> observers = new ArrayList<>();
  private float temperature;
```

```java
    // Add a new observer (like subscribing)
    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    // Remove an observer (like unsubscribing)
    public void removeObserver(Observer observer) {
        observers.remove(observer);
    }

    // Tell all observers about the change
    public void notifyObservers() {
        for (Observer observer : observers) {
            observer.update(temperature);
        }
    }

    // When temperature changes, notify everyone
    public void setTemperature(float temperature) {
        this.temperature = temperature;
        notifyObservers(); // Automatically notify all observers
    }
}

// Different observers (subscribers)
class PhoneApp implements Observer {
    public void update(float temperature) {
        System.out.println("Phone App: Temperature is now " + temperature + "°C");
    }
}

class WebsiteDisplay implements Observer {
    public void update(float temperature) {
        System.out.println("Website: Current temperature: " + temperature + "°C");
    }
}

class EmailAlert implements Observer {
    public void update(float temperature) {
        if (temperature > 35) {
            System.out.println("Email Alert: It's hot! Temperature is " + temperature + "°C");
        }
    }
}

// How to use it
public class ObserverExample {
    public static void main(String[] args) {
        WeatherStation station = new WeatherStation();

        // Create observers
        PhoneApp phone = new PhoneApp();
        WebsiteDisplay website = new WebsiteDisplay();
        EmailAlert email = new EmailAlert();

        // Subscribe to weather updates
        station.addObserver(phone);
        station.addObserver(website);
        station.addObserver(email);

        // Change temperature - everyone gets notified automatically!
        station.setTemperature(25.0f);
        // Output:
        // Phone App: Temperature is now 25.0°C
        // Website: Current temperature: 25.0°C

        station.setTemperature(40.0f);
        // Output:
        // Phone App: Temperature is now 40.0°C
        // Website: Current temperature: 40.0°C
        // Email Alert: It's hot! Temperature is 40.0°C
    }
}
```

Adhi ek interface for observable (ObservableInterface)→ hyamdhe add, remove and notify

Ata ek interface for observer (ObserverInterface)→ update method

Ata ek class which implements observable interface (ObservableImpl) →create a variable data, create ArrayList of ObserverInterface and implement add,remove and notify methods(which will internally call the update method present in the ObserverInterface) on these objects. along with a method to setdata which when called will modify the data and call notify method.

Ata ek class (ObserverImpl) which implements ObserverInterface→ override the function update with its implementation here.

Now create a main class→ create object of OnservableImpl class. Create 2-3 objects of ObserverImpl class. Add objects of ObserverImpl classes into object of ObservableImpl class.
Then call setdata method via the object of ObservableImpl class.