list of obsevers:

Observer Design Pattern

Affan Rauf

September 20, 2024

Introduction to Observer Pattern

- The Observer pattern defines a one-to-many relationship between objects.
- When the state of the **Subject**[1] changes, all its **Observers** are notified automatically.
- Promotes loose coupling between the subject and its observers.

[1] Observable

Introduction to Observer Pattern

We make interface then make multiple classes for the flexibility of user behaviour

- The Observer pattern defines a one-to-many relationship between objects.
- When the state of the **Subject**[1] changes, all its **Observers** are notified automatically.
- Promotes loose coupling between the subject and its observers.

Example:

- Weather monitoring system where displays (observers) are updated when weather data changes (subject).

[1] Observable

Structure of Observer Pattern

- **Subject:** Maintains a list of observers and notifies them of state changes.
- **Observer:** Defines an update interface for objects that should be notified of changes.
- **Concrete Subject:** Stores state and notifies observers when it changes.
- **Concrete Observer:** Implements the update method to reflect state changes.

Structure of Observer Pattern

- **Subject:** Maintains a list of observers and notifies them of state changes.
- **Observer:** Defines an update interface for objects that should be notified of changes.
- **Concrete Subject:** Stores state and notifies observers when it changes.
- **Concrete Observer:** Implements the update method to reflect state changes.

Example of Observer Pattern in Java

- Let's consider an example of a Weather Station (subject) and various display units (observers) that react to changes in weather data.

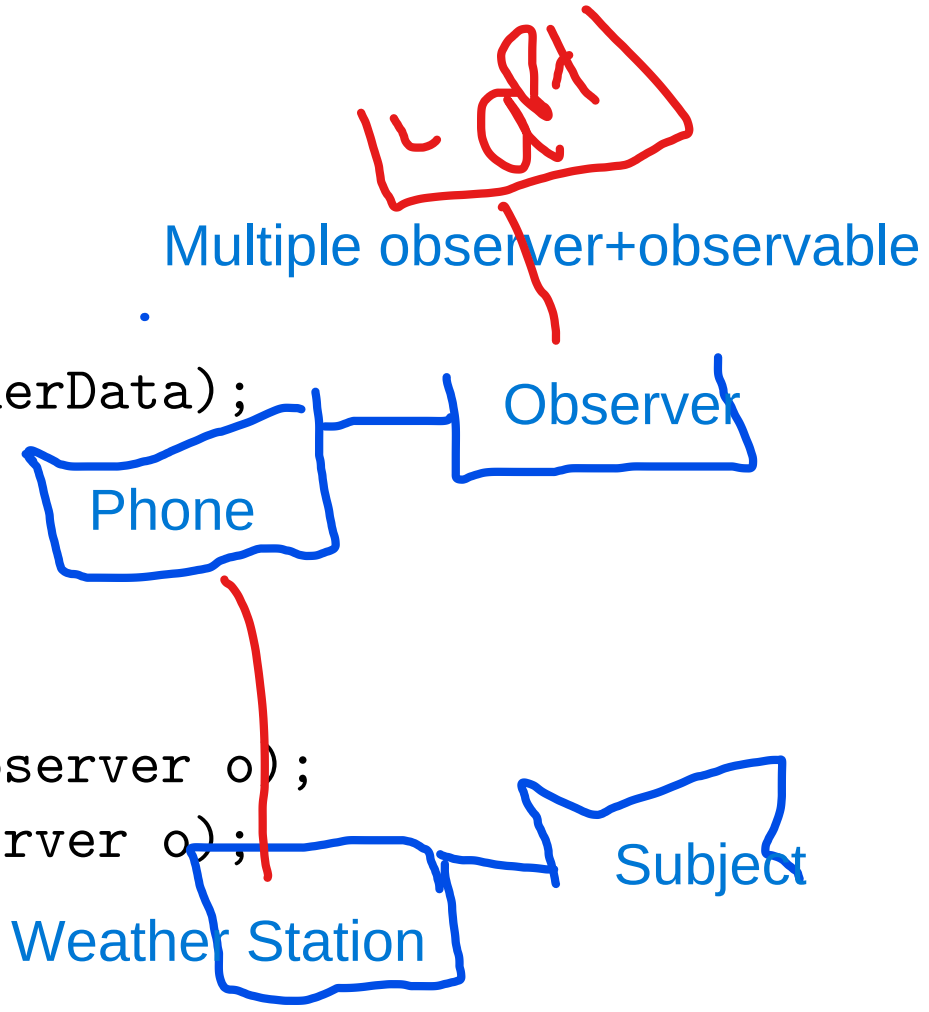
Example of Observer Pattern in Java

- Let's consider an example of a Weather Station (subject) and various display units (observers) that react to changes in weather data.

Code Example:

```
// Observer interface
interface Observer {
    void update(String weatherData);
}

// Subject interface
interface Subject {
    void registerObserver(Observer o);
    void removeObserver(Observer o);
    void notifyObservers();
}
```



Concrete Subject and Observer

Concrete Subject: Weather Station

```
class WeatherStation implements Subject {
    private List<Observer> observers = new ArrayList<>();
    private String weatherData;

    public void registerObserver(Observer o) { observers.add(o); }
    public void removeObserver(Observer o) { observers.remove(o); }
    public void notifyObservers() {
        for (Observer o : observers) { o.update(weatherData); }
    }

    public void setWeatherData(String data) {
        this.weatherData = data;
        notifyObservers();
    }
}
```

Concrete Subject and Observer

Concrete Subject: Weather Station

```
class WeatherStation implements Subject {
    private List<Observer> observers = new ArrayList<>();
    private String weatherData;

    public void registerObserver(Observer o) { observers.add(o); }
    public void removeObserver(Observer o) { observers.remove(o); }
    public void notifyObservers() {
        for (Observer o : observers) { o.update(weatherData); }
    }

    public void setWeatherData(String data) {
        this.weatherData = data;
        notifyObservers();
    }
}
```

Concrete Observer: Phone Display

```
class PhoneDisplay implements Observer {
    public void update(String weatherData) {
        System.out.println("Phone Display: " + weatherData);
    }
}
```

Uses of Observer Pattern

- **Event-driven systems:** GUI frameworks where user actions trigger updates.
- **Notification systems:** Email notifications or event subscriptions.
- **Real-time monitoring systems:** Weather monitoring, stock market tickers.
- **Distributed systems:** Where components need to be notified of state changes across different systems.

Uses of Observer Pattern

- **Event-driven systems:** GUI frameworks where user actions trigger updates.
- **Notification systems:** Email notifications or event subscriptions.
- **Real-time monitoring systems:** Weather monitoring, stock market tickers.
- **Distributed systems:** Where components need to be notified of state changes across different systems.

Common in:

- MVC (Model-View-Controller) architecture.
- Event listeners in user interfaces.

OB

MVC ke under observer use kr rahe to MVC hoga otherwise then it will be MVP.

Benefits of Observer Pattern

- **Loose Coupling:** The subject and observers are loosely coupled, allowing them to vary independently.
- **Flexibility:** You can easily add, remove, or change observers without modifying the subject.
- **Scalability:** Supports dynamic relationships, allowing multiple observers to respond to changes in real time.
- **Reusability:** Observers can be reused across different subjects.

Benefits of Observer Pattern

- **Loose Coupling:** The subject and observers are loosely coupled, allowing them to vary independently.
- **Flexibility:** You can easily add, remove, or change observers without modifying the subject.
- **Scalability:** Supports dynamic relationships, allowing multiple observers to respond to changes in real time.
- **Reusability:** Observers can be reused across different subjects.

Key Benefit: The pattern allows for reactive updates without hard-coding dependencies between components.