**Exercise 7: Implementing the Observer Pattern**

**Scenario:**

You are developing a stock market monitoring application where multiple clients need to be notified whenever stock prices change. Use the Observer Pattern to achieve this.

**Steps:**

1. **Create a New Java Project:**
   * Create a new Java project named **ObserverPatternExample**.
2. **Define Subject Interface:**
   * Create an interface **Stock** with methods to **register**, **deregister**, and **notify** observers.
3. **Implement Concrete Subject:**
   * Create a class **StockMarket** that implements **Stock** and maintains a list of observers.
4. **Define Observer Interface:**
   * Create an interface Observer with a method **update().**
5. **Implement Concrete Observers:**
   * Create classes **MobileApp**, **WebApp** that implement Observer.
6. **Test the Observer Implementation:**
   * Create a test class to demonstrate the registration and notification of observers.

#### Step 1: Create a New Java Project

Create a new Java project named ObserverPatternExample.

#### Step 2: Define Subject Interface

Create an interface Stock with methods to register, deregister, and notify observers.

Stock.java

public interface Stock {

void registerObserver(Observer o);

void deregisterObserver(Observer o);

void notifyObservers();

}

**Step 3: Implement Concrete Subject**

Create a class StockMarket that implements Stock and maintains a list of observers.

StockMarket.java

import java.util.ArrayList;

import java.util.List;

public class StockMarket implements Stock {

private List<Observer> observers;

private String stockName;

private double stockPrice;

public StockMarket() {

observers = new ArrayList<>();

}

public void setStock(String stockName, double stockPrice) {

this.stockName = stockName;

this.stockPrice = stockPrice;

notifyObservers();

}

@Override

public void registerObserver(Observer o) {

observers.add(o);

}

@Override

public void deregisterObserver(Observer o) {

observers.remove(o);

}

@Override

public void notifyObservers() {

for (Observer o : observers) {

o.update(stockName, stockPrice);

}

}

}

**Step 4: Define Observer Interface**

Create an interface Observer with a method update().

Observer.java

public interface Observer {

void update(String stockName, double stockPrice);

}

**Step 5: Implement Concrete Observers**

Create classes MobileApp and WebApp that implement Observer.

MobileApp.java

public class MobileApp implements Observer {

private String appName;

public MobileApp(String appName) {

this.appName = appName;

}

@Override

public void update(String stockName, double stockPrice) {

System.out.println(appName + " received update: " + stockName + " is now $" + stockPrice);

}

}

WebApp.java

public class WebApp implements Observer {

private String appName;

public WebApp(String appName) {

this.appName = appName;

}

@Override

public void update(String stockName, double stockPrice) {

System.out.println(appName + " received update: " + stockName + " is now $" + stockPrice);

}

}

**Step 6: Test the Observer Implementation**

Create a test class to demonstrate the registration and notification of observers.

ObserverPatternTest.java

public class ObserverPatternTest {

public static void main(String[] args) {

StockMarket stockMarket = new StockMarket();

Observer mobileApp = new MobileApp("MobileApp");

Observer webApp = new WebApp("WebApp");

stockMarket.registerObserver(mobileApp);

stockMarket.registerObserver(webApp);

stockMarket.setStock("Apple", 150.50);

stockMarket.setStock("Google", 2800.75);

stockMarket.deregisterObserver(mobileApp);

stockMarket.setStock("Amazon", 3400.10);

}

}