**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:**

* **Project Name**: ObserverPatternExample
* Use your preferred IDE (like IntelliJ IDEA, Eclipse, or NetBeans) to create a new Java project.

**2. Define Subject Interface:**

**Stock Interface**:

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

import java.util.ArrayList;

public interface Stock {

void registerObserver(Observer o);

void deregisterObserver(Observer o);

void notifyObservers();

}

**3. Implement Concrete Subject:**

**StockMarket Class**:

* Create a class StockMarket that implements Stock and maintains a list of observers.
* This class will also have methods to set and get the stock price.

import java.util.ArrayList;

import java.util.List;

public class StockMarket implements Stock {

private List<Observer> observers;

private double stockPrice;

public StockMarket() {

this.observers = new ArrayList<>();

}

@Override

public void registerObserver(Observer o) {

observers.add(o);

}

@Override

public void deregisterObserver(Observer o) {

observers.remove(o);

}

@Override

public void notifyObservers() {

for (Observer observer : observers) {

observer.update(stockPrice);

}

}

public void setStockPrice(double stockPrice) {

this.stockPrice = stockPrice;

notifyObservers();

}

public double getStockPrice() {

return stockPrice;

}

}

**4. Define Observer Interface:**

**Observer Interface**:

* Create an interface Observer with a method update().

public interface Observer {

void update(double stockPrice);

}

**5. Implement Concrete Observers:**

**MobileApp Class**:

* Create a class MobileApp that implements Observer.

public class MobileApp implements Observer {

private String name;

public MobileApp(String name) {

this.name = name;

}

@Override

public void update(double stockPrice) {

System.out.println("MobileApp " + name + " received stock price update: " + stockPrice);

}

}

**WebApp Class**:

* Create a class WebApp that implements Observer.

public class WebApp implements Observer {

private String name;

public WebApp(String name) {

this.name = name;

}

@Override

public void update(double stockPrice) {

System.out.println("WebApp " + name + " received stock price update: " + stockPrice);

}

}

**6. Test the Observer Implementation:**

**TestObserverPattern Class**:

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

public class TestObserverPattern {

public static void main(String[] args) {

// Create StockMarket (Subject)

StockMarket stockMarket = new StockMarket();

// Create Observers

Observer mobileApp1 = new MobileApp("Mobile App 1");

Observer webApp1 = new WebApp("Web App 1");

// Register Observers

stockMarket.registerObserver(mobileApp1);

stockMarket.registerObserver(webApp1);

// Change Stock Price

stockMarket.setStockPrice(100.00);

// Deregister an Observer

stockMarket.deregisterObserver(mobileApp1);

// Change Stock Price Again

stockMarket.setStockPrice(105.50);

}

}