**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.

1. Create a new Java Project:

* Create a new Java Project named ObserverPatternExample.
* A Java project named ObserverPatternExample is created in eclipse IDE.

2. Define Subject Interface:

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

import java.util.List;

public interface Stock {

void registerObserver(Observer observer);

void deregisterObserver(Observer observer);

void notifyObservers();

}

* The Stock interface defines methods to register, deregister, and notify observers.

3. Implement Concrete Subject:

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

import java.util.ArrayList;

import java.util.List;

public class StockMarket implements Stock {

private List<Observer> observers;

private double price;

public StockMarket() {

this.observers = new ArrayList<>();

}

@Override

public void registerObserver(Observer observer) {

observers.add(observer);

}

@Override

public void deregisterObserver(Observer observer) {

observers.remove(observer);

}

@Override

public void notifyObservers() {

for (Observer observer : observers) {

observer.update(price);

}

}

public void setPrice(double price) {

this.price = price;

notifyObservers();

}

}

* The StockMarket class implements the Stock interface and maintains a list of observers.
* It also includes a method to set the stock price and notify observers of the change.

4. Define Observer Interface:

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

public interface Observer {

void update(double price);

}

* The Observer interface defines a single method update() to be called when the subject's state changes.

5. Implement Concrete Observer:

* Create classes **MobileApp**, **WebApp** that implement Observer.

public class MobileApp implements Observer {

private String name;

public MobileApp(String name) {

this.name = name;

}

@Override

public void update(double price) {

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

}

}

public class WebApp implements Observer {

private String name;

public WebApp(String name) {

this.name = name;

}

@Override

public void update(double price) {

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

}

}

* The MobileApp and WebApp classes implement the Observer interface and provide specific implementations of the update() method.

6. Test the Observer Implementation:

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

import java.util.Scanner;

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);

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.print("Enter new stock price (or 'exit' to quit): ");

String input = scanner.nextLine();

if (input.equalsIgnoreCase("exit")) {

break;

}

try {

double price = Double.parseDouble(input);

stockMarket.setPrice(price);

} catch (NumberFormatException e) {

System.out.println("Invalid input. Please enter a valid number.");

}

}

scanner.close();

}

}

* Enter new stock prices when prompted.
* Observe that the MobileApp and WebApp receive updates whenever the stock price changes.
* Output

