**DESIGN PATTERNS AND PRINCIPLES**

**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.

**DESCRIPTION:**

This program illustrates the Observer Pattern in a stock market application.

* Stock is the subject interface that defines methods to manage observers.
* StockMarket is the concrete subject that maintains stock prices and notifies all registered observers when prices change.
* Observer is the interface for receiving updates.
* MobileApp and WebApp are concrete observer classes that respond to stock price updates.

The client (ObserverPatternExample) adds and removes observers, demonstrating how updates are sent in real-time to all active subscribers.  
This pattern ensures efficient and automatic communication between the stock market and multiple clients.

**PROGRAM:**

import java.util.\*;

public class ObserverPatternExample {

public static void main(String[] args) {

StockMarket market = new StockMarket();

Observer mobileApp = new MobileApp("Investor A");

Observer webApp = new WebApp("Dashboard B");

market.registerObserver(mobileApp);

market.registerObserver(webApp);

market.setStockPrice("AAPL", 180.25);

System.out.println("---");

market.setStockPrice("GOOGL", 2850.10);

System.out.println("---");

market.removeObserver(webApp);

market.setStockPrice("MSFT", 330.00);

}

}

interface Stock {

void registerObserver(Observer o);

void removeObserver(Observer o);

void notifyObservers(String stockName, double price);

}

class StockMarket implements Stock {

private List<Observer> observers = new ArrayList<>();

private Map<String, Double> stockPrices = new HashMap<>();

public void registerObserver(Observer o) {

observers.add(o);

}

public void removeObserver(Observer o) {

observers.remove(o);

}

public void notifyObservers(String stockName, double price) {

for (Observer o : observers) {

o.update(stockName, price);

}

}

public void setStockPrice(String stockName, double price) {

stockPrices.put(stockName, price);

notifyObservers(stockName, price);

}

}

interface Observer {

void update(String stockName, double price);

}

class MobileApp implements Observer {

private String user;

public MobileApp(String user) {

this.user = user;

}

public void update(String stockName, double price) {

System.out.println("MobileApp [" + user + "] - " + stockName + " updated to ₹" + price);

}

}

class WebApp implements Observer {

private String dashboard;

public WebApp(String dashboard) {

this.dashboard = dashboard;

}

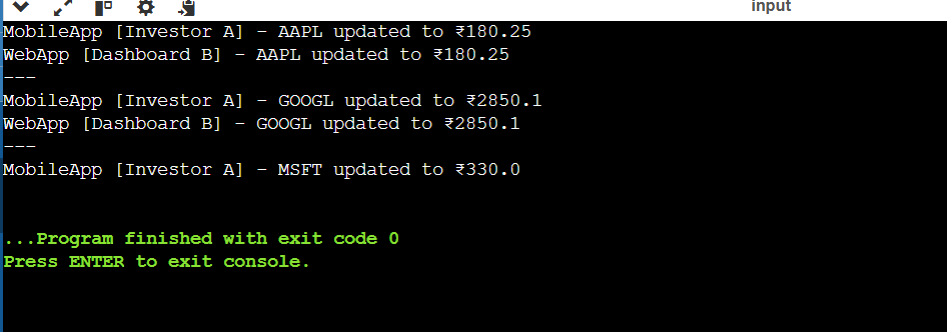
public void update(String stockName, double price) {

System.out.println("WebApp [" + dashboard + "] - " + stockName + " updated to ₹" + price);

}

}

**OUTPUT:**

****