**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.

Code –

package Design\_Patterns\_And\_Principles.ObserverPatternExample;

import java.util.ArrayList;

import java.util.List;

interface Stock {

    void register(Observer observer);

    void deregister(Observer observer);

    void notifyObservers();

}

interface Observer {

    void update(String stockName, double stockPrice);

}

class StockMarket implements Stock {

    private List<Observer> observers;

    private String stockName;

    private double stockPrice;

    public StockMarket(String stockName, double stockPrice) {

        this.stockName = stockName;

        this.stockPrice = stockPrice;

        this.observers = new ArrayList<>();

    }

    @Override

    public void register(Observer observer) {

        observers.add(observer);

    }

    @Override

    public void deregister(Observer observer) {

        observers.remove(observer);

    }

    @Override

    public void notifyObservers() {

        for (Observer observer : observers) {

            observer.update(stockName, stockPrice);

        }

    }

    public void setStockPrice(double stockPrice) {

        this.stockPrice = stockPrice;

        notifyObservers();

    }

}

class MobileApp implements Observer {

    @Override

    public void update(String stockName, double stockPrice) {

        System.out.println("MobileApp: Stock " + stockName + " is now $" + stockPrice);

    }

}

class WebApp implements Observer {

    @Override

    public void update(String stockName, double stockPrice) {

        System.out.println("WebApp: Stock " + stockName + " is now $" + stockPrice);

    }

}

public class ObserverPatternDemo {

    public static void main(String[] args) {

        // Create a StockMarket object

        StockMarket stockMarket = new StockMarket("Apple", 150.00);

        // Create observer objects

        Observer mobileApp = new MobileApp();

        Observer webApp = new WebApp();

        // Register observers with the stock market

        stockMarket.register(mobileApp);

        stockMarket.register(webApp);

        // Change stock price and notify observers

        System.out.println("Updating stock price...");

        stockMarket.setStockPrice(155.00);

        System.out.println();

        // Deregister one observer and update stock price again

        stockMarket.deregister(mobileApp);

        System.out.println("Updating stock price again...");

        stockMarket.setStockPrice(160.00);

    }

}