### ****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.

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

Create a new Java project named ObserverPatternExample.

### ****Step 2: Define Subject Interface****

public interface Stock {

void registerObserver(Observer o);

void removeObserver(Observer o);

void notifyObservers();

}

### ****Step 3: Implement Concrete Subject****

import java.util.ArrayList;import java.util.List;

public class StockMarket implements Stock {

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

private double stockPrice;

@Override

public void registerObserver(Observer o) {

observers.add(o);

}

@Override

public void removeObserver(Observer o) {

observers.remove(o);

}

@Override

public void notifyObservers() {

for (Observer o : observers) {

o.update(stockPrice);

}

}

public void setStockPrice(double price) {

this.stockPrice = price;

notifyObservers();

}

}

### ****Step 4: Define Observer Interface****

public interface Observer {

void update(double stockPrice);

}

### ****Step 5: Implement Concrete Observers****

public class MobileApp implements Observer {

@Override

public void update(double stockPrice) {

System.out.println("MobileApp - Stock Price Updated: " + stockPrice);

}

}

public class WebApp implements Observer {

@Override

public void update(double stockPrice) {

System.out.println("WebApp - Stock Price Updated: " + stockPrice);

}

}

### ****Step 6: Test the Observer Implementation****

public class TestObserverPattern {

public static void main(String[] args) {

StockMarket market = new StockMarket();

Observer mobile = new MobileApp();

Observer web = new WebApp();

market.registerObserver(mobile);

market.registerObserver(web);

market.setStockPrice(150.75);

market.setStockPrice(170.25);

}

}

OUTPUT:  
