WEEK – 1

Implementing the observer Pattern

Nipuna Amanapu

[namanapu@gitam.in](mailto:namanapu@gitam.in)

superset id : 6432842

**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.

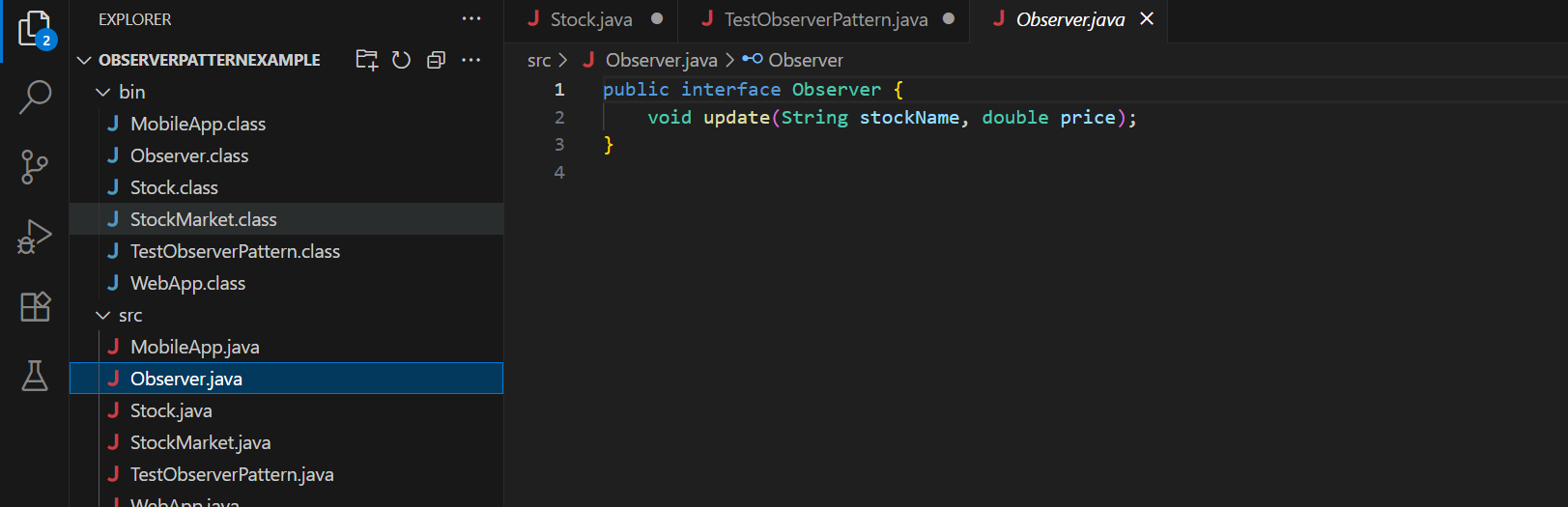
Solution:

1)Observer.java

public interface Observer {

void update(String stockName, double price);

}



2) StockMarket.java

import java.util.ArrayList;

import java.util.List;

public class StockMarket implements Stock {

private List<Observer> observers;

private String stockName;

private double price;

public StockMarket(String stockName) {

this.stockName = stockName;

this.observers = new ArrayList<>();

}

@Override

public void registerObserver(Observer observer) {

observers.add(observer);

}

@Override

public void removeObserver(Observer observer) {

observers.remove(observer);

}

@Override

public void notifyObservers() {

for (Observer observer : observers) {

observer.update(stockName, price);

}

}

public void setPrice(double newPrice) {

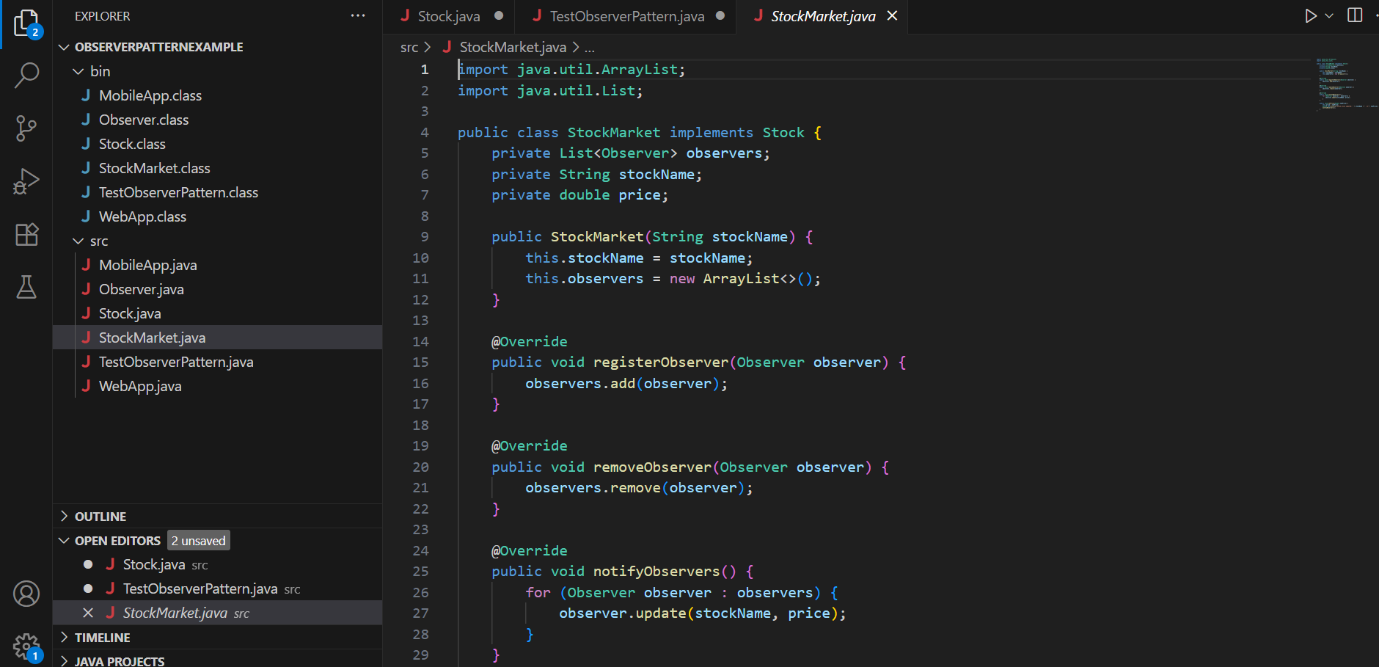
this.price = newPrice;

System.out.println("Stock price updated: " + stockName + " = $" + newPrice);

notifyObservers();

}

}



3) MobileApp.java

public class MobileApp implements Observer {

private String appName;

public MobileApp(String appName) {

this.appName = appName;

}

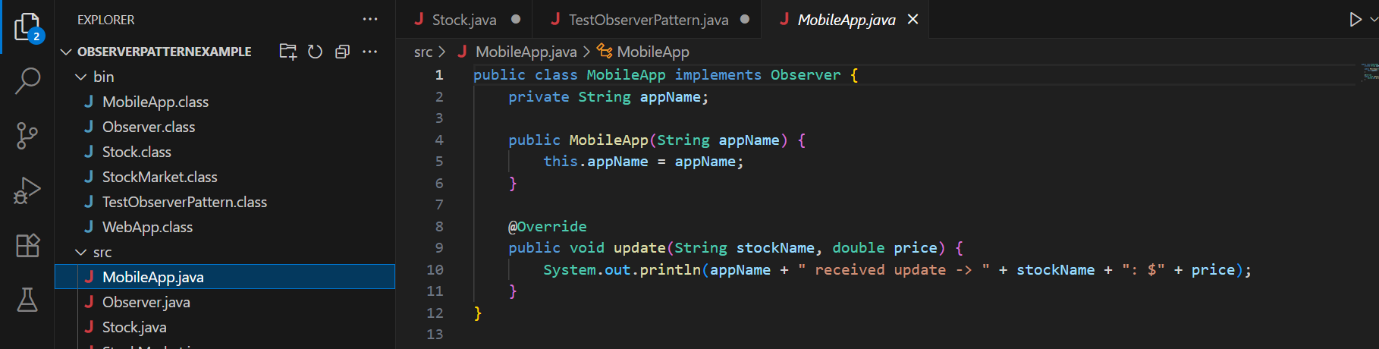
@Override

public void update(String stockName, double price) {

System.out.println(appName + " received update -> " + stockName + ": $" + price);

}

}



4) WebApp.java

public class WebApp implements Observer {

private String appName;

public WebApp(String appName) {

this.appName = appName;

}

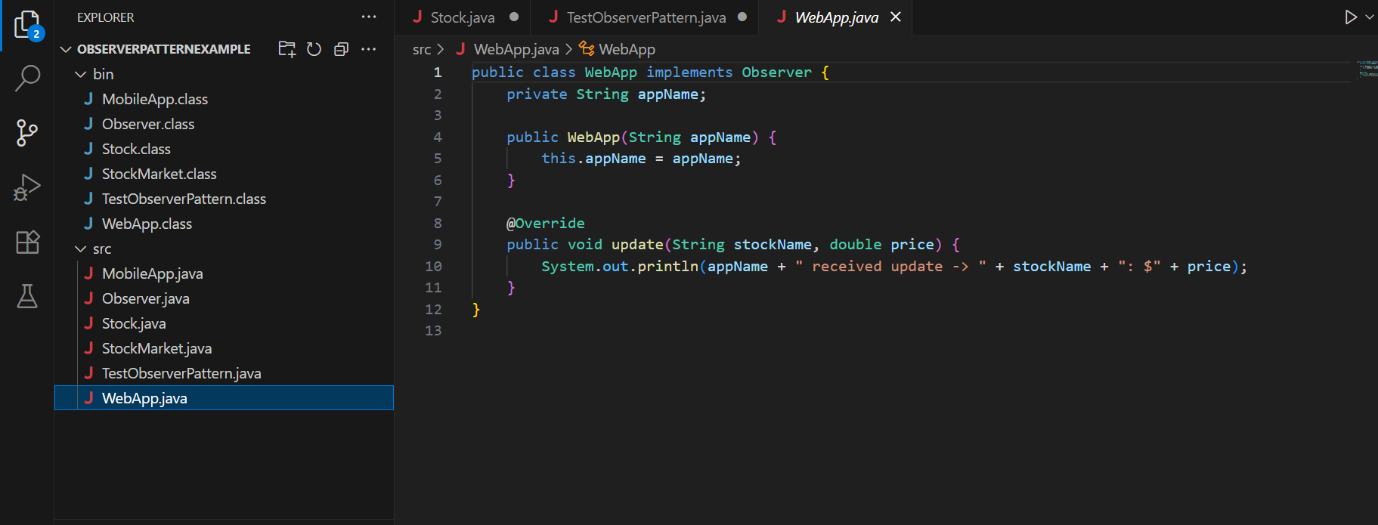
@Override

public void update(String stockName, double price) {

System.out.println(appName + " received update -> " + stockName + ": $" + price);

}

}



5)TestObserverPattern.java

public class TestObserverPattern {

public static void main(String[] args) {

StockMarket appleStock = new StockMarket("AAPL");

Observer mobileApp = new MobileApp("iOS App");

Observer webApp = new WebApp("Chrome Web");

appleStock.registerObserver(mobileApp);

appleStock.registerObserver(webApp);

appleStock.setPrice(150.75);

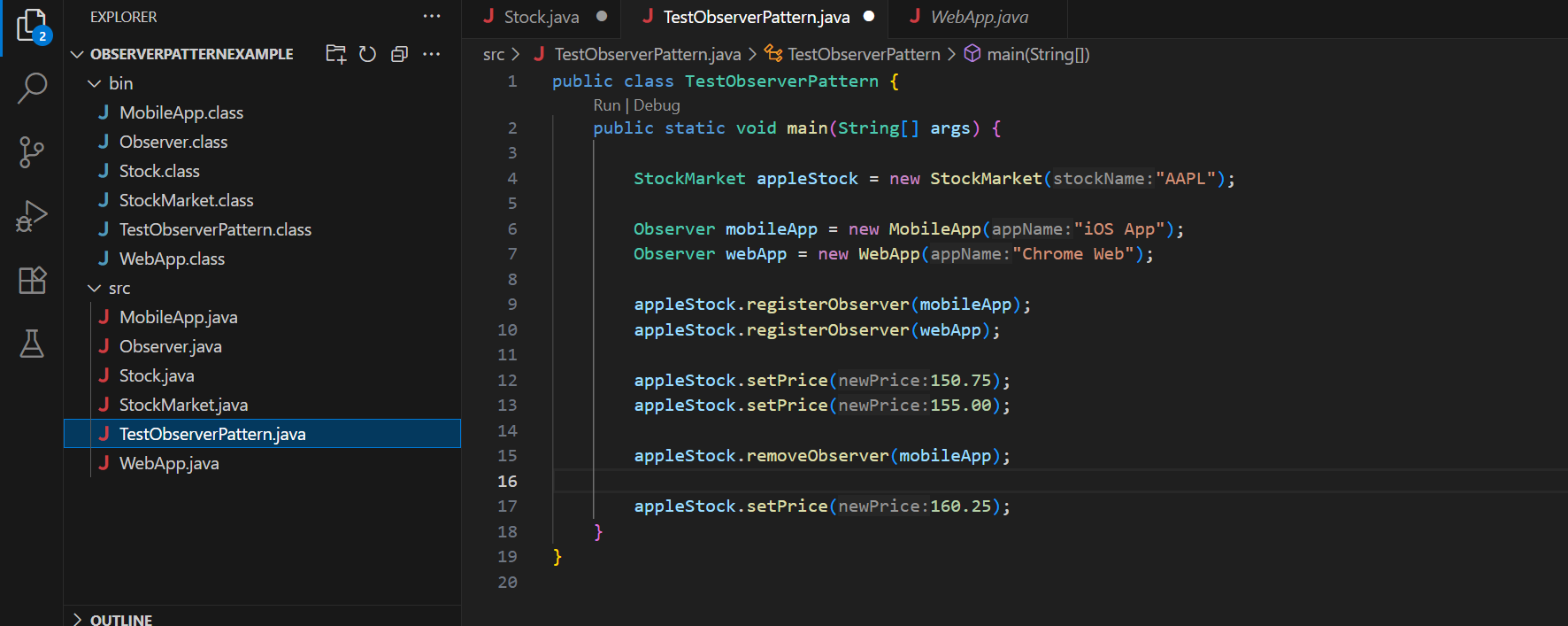
appleStock.setPrice(155.00);

appleStock.removeObserver(mobileApp);

appleStock.setPrice(160.25);

}

}



Output:

