**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:-**

import java.util.\*;

// Subject Interface

interface Stock {

    void registerObserver(Observer o);

    void removeObserver(Observer o);

    void notifyObservers();

}

// Observer Interface

interface Observer {

    void update(double stockPrice);

}

// Concrete Subject

class StockMarket implements Stock {

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

    private double stockPrice;

    public void setStockPrice(double price) {

        this.stockPrice = price;

        notifyObservers();

    }

    public double getStockPrice() {

        return 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);

        }

    }

}

// Concrete Observer - Mobile App

class MobileApp implements Observer {

    private String name;

    public MobileApp(String name) {

        this.name = name;

    }

    @Override

    public void update(double stockPrice) {

        System.out.println("MobileApp " + name + ": Stock price updated to $" + stockPrice);

    }

}

// Concrete Observer - Web App

class WebApp implements Observer {

    private String name;

    public WebApp(String name) {

        this.name = name;

    }

    @Override

    public void update(double stockPrice) {

        System.out.println("WebApp " + name + ": Stock price updated to $" + stockPrice);

    }

}

// Main class to test

public class ObserverPatternExample {

    public static void main(String[] args) {

        StockMarket stockMarket = new StockMarket();

        Observer mobile = new MobileApp("Alice");

        Observer web = new WebApp("Bob");

        stockMarket.registerObserver(mobile);

        stockMarket.registerObserver(web);

        System.out.println("Setting stock price to 200.5");

        stockMarket.setStockPrice(200.5);

        System.out.println("\nRemoving WebApp observer...");

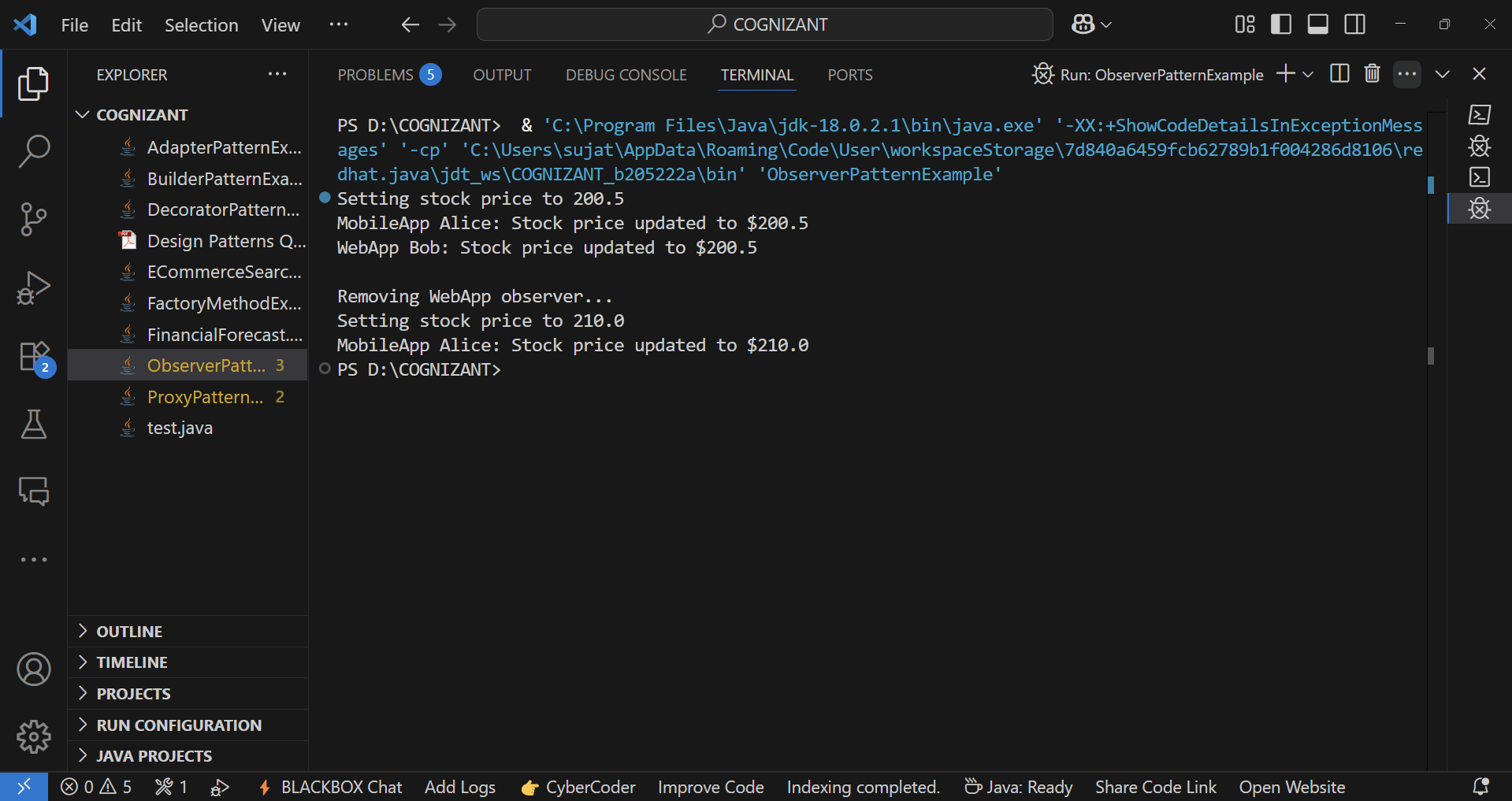
        stockMarket.removeObserver(web);

        System.out.println("Setting stock price to 210.0");

        stockMarket.setStockPrice(210.0);

    }

}

**OUTPUT:-**