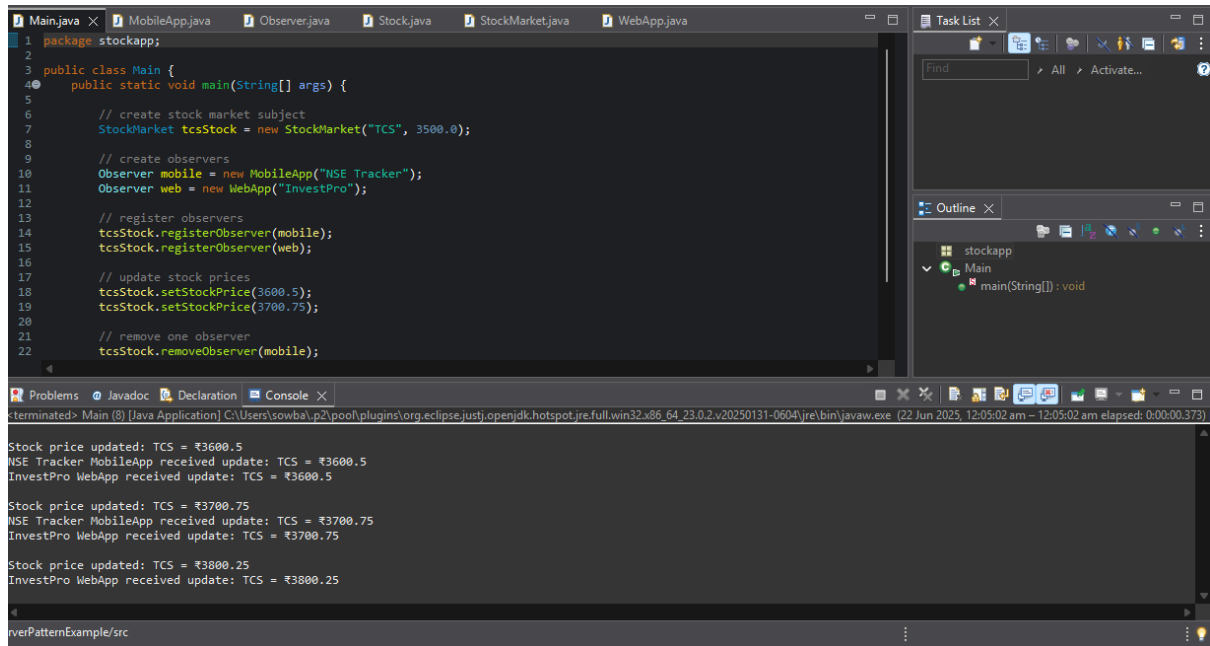


Exercise 7: Implementing the Observer Pattern

OUTPUT:



```
1 package stockapp;
2
3 public class Main {
4     public static void main(String[] args) {
5
6         // create stock market subject
7         StockMarket tcsStock = new StockMarket("TCS", 3500.0);
8
9         // create observers
10        Observer mobile = new MobileApp("NSE Tracker");
11        Observer web = new WebApp("InvestPro");
12
13        // register observers
14        tcsStock.registerObserver(mobile);
15        tcsStock.registerObserver(web);
16
17        // update stock prices
18        tcsStock.setStockPrice(3600.5);
19        tcsStock.setStockPrice(3700.75);
20
21        // remove one observer
22        tcsStock.removeObserver(mobile);
23    }
24 }
```

terminated> Main (0) [Java Application] C:\Users\sowba\p2\pool\plugins\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64_23.0.2.v20250131-0604\jre\bin\javaw.exe (22 Jun 2025, 12:05:02 am — 12:05:02 am elapsed: 0:00:00.373)

Stock price updated: TCS = ₹3600.5
NSE Tracker MobileApp received update: TCS = ₹3600.5
InvestPro WebApp received update: TCS = ₹3600.5

Stock price updated: TCS = ₹3700.75
NSE Tracker MobileApp received update: TCS = ₹3700.75
InvestPro WebApp received update: TCS = ₹3700.75

Stock price updated: TCS = ₹3800.25
InvestPro WebApp received update: TCS = ₹3800.25

verPatternExample/src

The Observer Pattern allows multiple clients (observers) to receive automatic updates when stock prices change. It supports real-time notifications, promotes loose coupling, and is ideal for event-driven systems.