Exercise 7: Implementing the Observer Pattern

Objective

To build a stock market monitoring app using the Observer Pattern where multiple clients (observers) receive updates when stock prices change.

Java Code – ObserverPatternExample.java import java.util.ArrayList;

import java.util.List;

// Step 1 & 2: Subject Interface interface Stock {

void registerObserver(Observer o); void removeObserver(Observer o); void notifyObservers();

}

// Step 3: Concrete Subject

class StockMarket implements Stock {

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

public void setPrice(double price) { this.price = price;

notifyObservers(); // Notify when price is updated

}

@Override

public void registerObserver(Observer o) { observers.add(o);

System.out.println(o.getName() + " registered.");

}

@Override

public void removeObserver(Observer o) { observers.remove(o); System.out.println(o.getName() + " removed.");

}

@Override

public void notifyObservers() {

for (Observer o : observers) { o.update(price);

}

}

}

// Step 4: Observer Interface interface Observer {

void update(double price); String getName();

}

// Step 5: Concrete Observers

class MobileApp implements Observer { private String name;

public MobileApp(String name) {

this.name = name;

}

@Override

public void update(double price) {

System.out.println("MobileApp " + name + " received price update:

$" + price);

}

@Override

public String getName() {

return "MobileApp " + name;

}

}

class WebApp implements Observer { private String name;

public WebApp(String name) { this.name = name;

}

@Override

public void update(double price) {

System.out.println("WebApp " + name + " received price update: $"

+ price);

}

@Override

public String getName() { return "WebApp " + name;

}

}

// Step 6: Test Class

public class ObserverPatternExample {

public static void main(String[] args) { StockMarket stockMarket = new StockMarket();

Observer mobile1 = new MobileApp("Alpha"); Observer web1 = new WebApp("Beta");

stockMarket.registerObserver(mobile1); stockMarket.registerObserver(web1);

System.out.println("\n--- Price Update 1 ---"); stockMarket.setPrice(120.50);

stockMarket.removeObserver(mobile1);

System.out.println("\n--- Price Update 2 ---"); stockMarket.setPrice(130.75);

}

}

Sample Output

MobileApp Alpha registered. WebApp Beta registered.

--- Price Update 1 ---

MobileApp Alpha received price update: $120.5 WebApp Beta received price update: $120.5 MobileApp Alpha removed.

--- Price Update 2 ---

WebApp Beta received price update: $130.75

Key Benefits of Observer Pattern

Feature Description

Decouples Subject & Observers Subject doesn’t know specific details of the observers

Dynamic Subscriptions Observers can register/unregister at runtime Automatic Notification All observers are updated when subject state changes