Homework-2

SDA

Zunaira Abdul Aziz
BSSE23058
Section A

Overview

This project implements the Observer Pattern to create a weather monitoring system. It consists of the following components:

Subject (WeatherData): Maintains weather data and notifies observers of changes.

Observers (CurrentConditionsDisplay, StatisticsDisplay, ForecastDisplay): Displays different types of weather-related data when updated.

DisplayElement Interface: Defines a method for displaying data.

Execution Steps

1. Setup and Compilation

Compiled it using Termminal. Compile the Java files using:

javac *.java

2. Run the Application

Execute the main class that initializes the WeatherData and display components:

java WeatherStation

3. Expected Output

Once the program runs, it will display weather updates for Current Conditions, Statistics, and Forecast based on changing data.

4. Modify Weather Data

To simulate weather changes, modify setMeasurements(float temperature, float humidity, float pressure) in WeatherData and observe how displays update.

Files Included

Subject.java: Interface defining the Subject.

Observer.java: Interface defining the Observer.

WeatherData.java: Implements the Subject.

CurrentConditionsDisplay.java: Displays current weather conditions.

StatisticsDisplay.java: Displays min, max, and average temperature.

ForecastDisplay.java: Displays weather forecast.

WeatherStation.java: Initializes and runs the application.

Code:

```
public interface Subject {
    public void registerObserver(Observer o); //these methods take an Observer as an
argument that is the Observer to be registerd

    public void removeObserver(Observer o); //or removed

    public void notifyObservers();//this will notify all observer of any changes that are
made to SUbject state
}


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

public class WeatherData implements Subject {

    private List<Observer> observers;
    private float temperature;
    private float humidity;
    private float pressure;

    public WeatherData() {
        observers = new ArrayList<Observer>(); //holds the new added observers
    }

    @Override
    public void registerObserver(Observer o) {
        observers.add(o); // adds a new observer at the end of the list
    }

    @Override
    public void removeObserver(Observer o) {
        observers.remove(o); // removes a new observer at the end of the list
    }

    @Override
    public void notifyObservers() {
        for (Observer observer : observers) {  //this will access the observers one by
one and implement the update() method and notify them
            observer.update(temperature, humidity, pressure);
        }
    }

    //setter for setting the new measurements
    public void setMeasurements(float temperature, float humidity, float pressure) {
        this.temperature = temperature;
        this.humidity = humidity;
        this.pressure = pressure;
        measurementsChanged();
    }

    //getters of the private variables
    public float getTemperature() {
        return temperature;
    }

    public float getHumidity() {
        return humidity;
    }

    public float getPressure() {
        return pressure;
    }

    public void measurementsChanged() { //calls the notify method, if the values are
changed in the setter than this is called
        notifyObservers();
    }
}


public interface Observer {//to implemnt all the chnages in the observers list

    public void update(float temperature, float humidity, float pressure); //shows the
updated info
}


public interface DisplayElement {

    public void display(); //interface for implementing display elements
}


public class StatisticsDisplay implements Observer, DisplayElement { //this one
keeps the min/avg/max measurement and displays them

    private float maxTemp = Float.MIN_VALUE;//took the least smallest float to set to
max
    private float minTemp = Float.MAX_VALUE;//took the max greayet float to set to
min
    private float tempSum = 0;
    private int numReadings;
    private WeatherData weatherData;

    public StatisticsDisplay(WeatherData weatherData) {
        this.weatherData = weatherData;
```

```java
        weatherData.registerObserver(this);
    }

    @Override
    public void update(float temperature, float humidity, float pressure) {
        tempSum += temperature;
        numReadings++;
        if (temperature > maxTemp) {
            maxTemp = temperature;
        }
        if (temperature < minTemp) {
            minTemp = temperature;
        }
        display();
    }

    @Override
    public void display() {
        System.out.println("Average temperature = " + (tempSum / numReadings));
        System.out.println("Max Temperature = " + maxTemp);
        System.out.println("Min Temperature = " + minTemp);
    }
}


public class ForecastDisplay implements Observer, DisplayElement {

    private float currentPressure = 29.92f;
    private float lastPressure;
    private WeatherData weatherData;

    public ForecastDisplay(WeatherData weatherData) {
        this.weatherData = weatherData;
        weatherData.registerObserver(this);
    }

    @Override
    public void update(float temperature, float humidity, float pressure) {
        lastPressure = currentPressure;
        currentPressure = pressure;
        display();
    }

    @Override
    public void display() {
        System.out.print("Forecast: ");
        if (currentPressure > lastPressure) {
            System.out.println("Improving weather on the way!");
        } else if (currentPressure < lastPressure) {
            System.out.println("Watch out for cooler, rainy weather.");
        } else {
            System.out.println("More of the same.");
        }
    }
}


public class WeatherStation {

    public static void main(String[] args) {
        WeatherData weatherData = new WeatherData(); //create the WeatherData object.
        CurrentConditionsDisplay currentDisplay = new CurrentConditionsDisplay(weatherData); // Create the three displays and pass them the WeatherData object
        StatisticsDisplay statisticsDisplay = new StatisticsDisplay(weatherData);
        ForecastDisplay forecastDisplay = new ForecastDisplay(weatherData);
        //Simulate new weathers measurements.
        weatherData.setMeasurements(80, 65, 30.4f);
        weatherData.setMeasurements(82, 70, 29.2f);
        weatherData.setMeasurements(78, 90, 29.2f);
    }
}
```