PROJECT REPORT

OBJECT ORIENTED PROGRAMMING

CSE – 2005

SLOT : C1 + TC1

SUBMITTED BY :

CHARUGUNDLA VISHAL – 23BCE7106

TO

Dr. D. SANTHADEVI

VIT-AP UNIVERSITY

VIT-AP UNIVERSITY
AMARAVATI
ANDHRA PRADESH, INDIA

INDEX

WEATHER FORECAST APPLICATION

A JAVA BASED APPLICATION WHICH DISPLAYS REAL - TIME WEATHER DETAILS

1) INTRODUCTION

The Weather Forecast Application is a Java-based application that provides users with real-time weather information for a specified location. It fetches weather data from an external API named “Weather Forecast API” and displays it in a graphical user interface (GUI). Users can enter a location, and the app retrieves and presents weather details, including temperature, weather condition, humidity, and wind speed. This documentation outlines the project's architecture, technologies used, and the functionality

2) TECHNOLOGIES USED

The Technologies used to build the Weather Forecast Application are :

- Java 21
- JSON Simple - Used to parse and read through JSON data
- HTTPURLConnection: Java's built-in library for making HTTP requests to fetch data from external APIs.
- Java Swing

The API used in retrieving data for the weather conditions is :

- Weather Application API
- Geolocation API

3) SYSTEM REQUIREMENTS

- Java Development Kit
- JAVA IDE
- Weather App Images

https://www.dropbox.com/scl/fi/wl5mfpt0fwyol14ku0jxu/weatherapp_images.rar?rlkey=oknkxq85slvx59yyw7iyogvm9&e=1&dl=0

4) PROJECT STRUCTURE

The Structure of the Weather Forecast App consists of Built – in classes like

- Javax.swing.*
- Javax.awt.*
- Java.util.Scanner
- Javax.net.HttpURLConnection
- Java.time.*

Java awt (Abstract Windowing Toolkit):

Java AWT (Abstract Window Toolkit) is an API to develop Graphical User Interface (GUI) to display content according to the code written in java .

Java swing:

Java Swing is also used to create window-based applications. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java

And also User Defined classes like

- WeatherAppGUI
- WeatherApp
- AppLauncher

1) WeatherAppGUI

The WeatherAppGui class represents the graphical user interface (GUI) of the Weather App. It is responsible for displaying weather information for a specified location which is taken as a user input . This class handles the layout and display of GUI components, including text fields, labels, buttons, and images that will be displayed . It also implements the user interface for entering a location and updating the weather information based on user input.

2) WeatherApp

The WeatherApp class contains the backend logic for fetching weather data from an external API. It retrieves geographic coordinates for a location like Latitude and Longitude, fetches weather data for that location, and provides methods to convert weather codes . This class encapsulates the core functionality of the Weather App. It includes methods to fetch weather data and location coordinates, convert weather codes into readable weather conditions, and manage API requests. This class acts as the bridge between the GUI and the external weather data source, ensuring that weather information is retrieved and displayed accurately. This is similar to the Backend part of a website .

3) AppLauncher

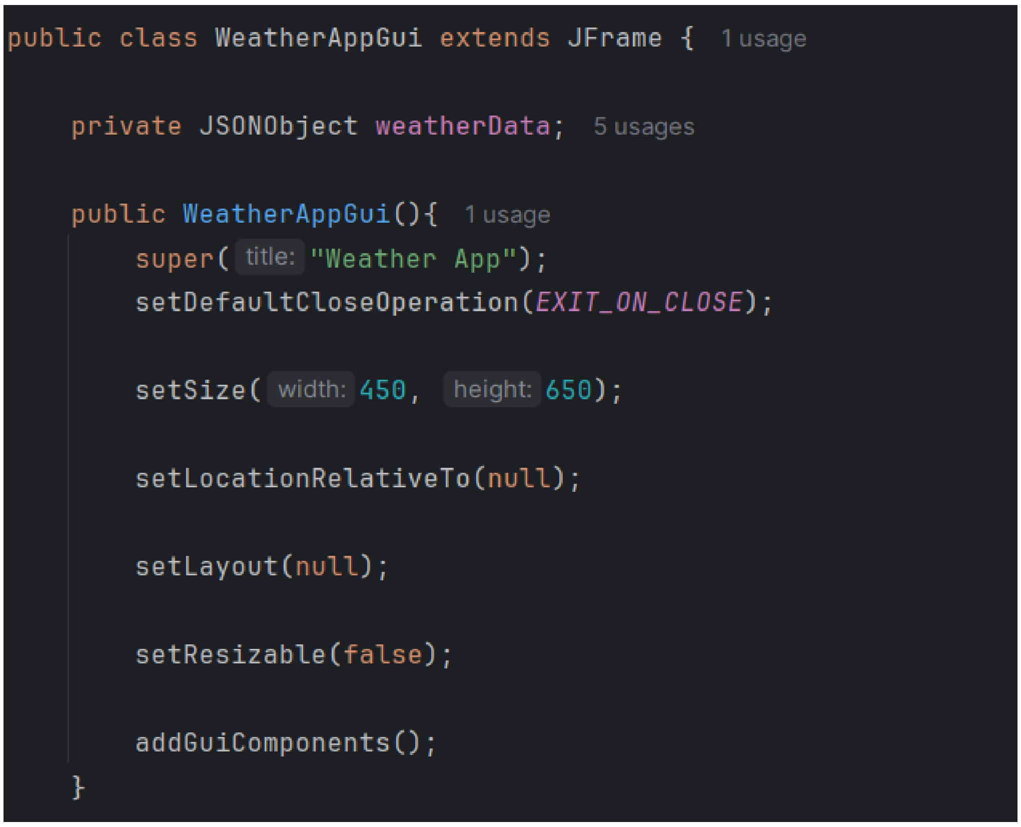
The AppLauncher class serves as the entry point for the Weather App. It initializes the GUI and displays the main application window where the user inputs the location name , and the App displays the corresponding weather information that has been retrieved from the API . It also displays the Weather Condition , Temperature , Humidity and also the Windspeed . This is similar to the frontend part of a Website .

5) SOURCE CODE

- CLASSES

1) WeatherAppGui

Here we are building the Display unit of our application .

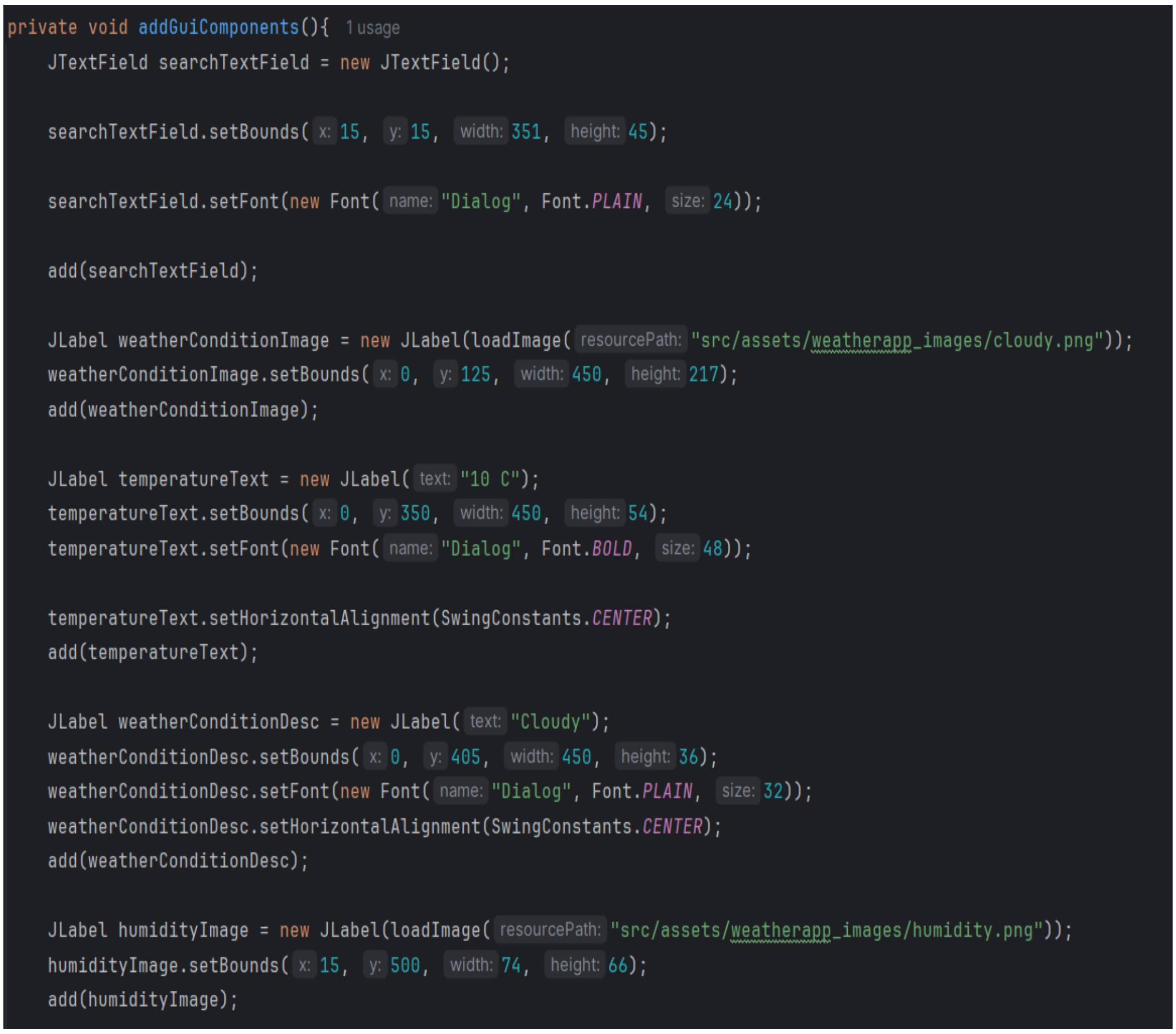


```
public class WeatherAppGui extends JFrame {  1 usage

    private JSONObject weatherData;  5 usages

    public WeatherAppGui(){  1 usage
        super( title: "Weather App");
        setDefaultCloseOperation(EXIT_ON_CLOSE);

        setSize( width: 450,  height: 650);

        setLocationRelativeTo(null);

        setLayout(null);

        setResizable(false);

        addGuiComponents();
    }
```

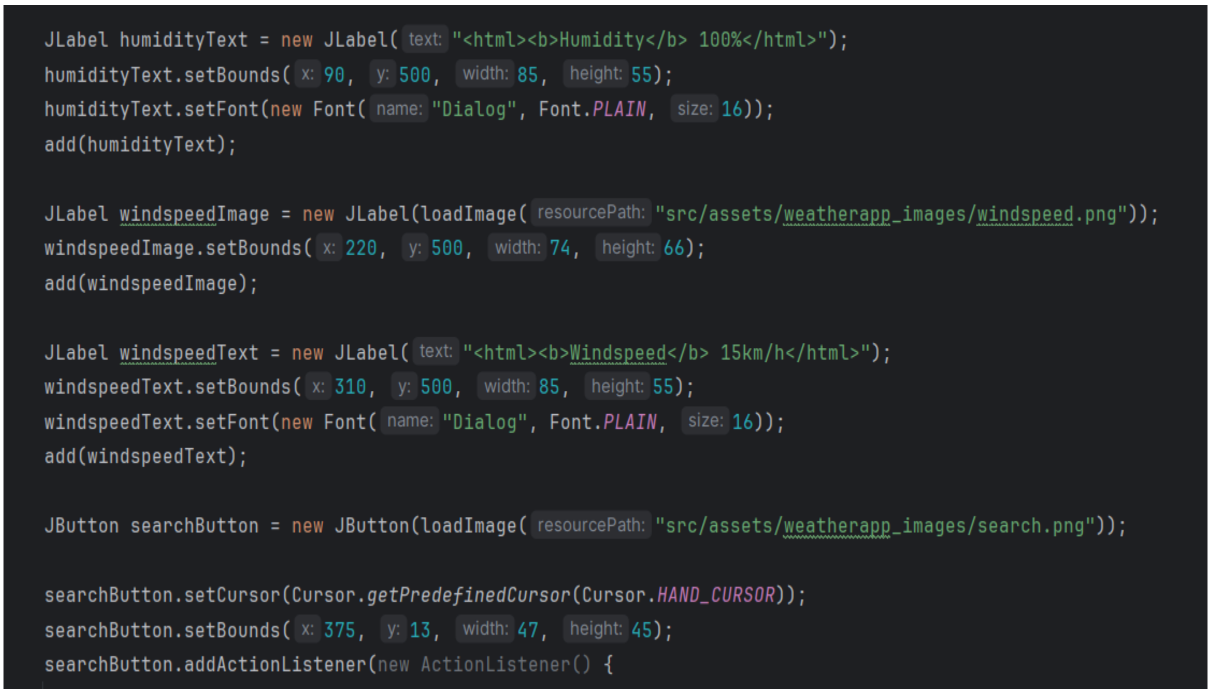
Methods

- addGuiComponents

Now we are Adding the components to the display unit like search bar , search button , Weather condition image , temperature , humidity , windspeed text and images .



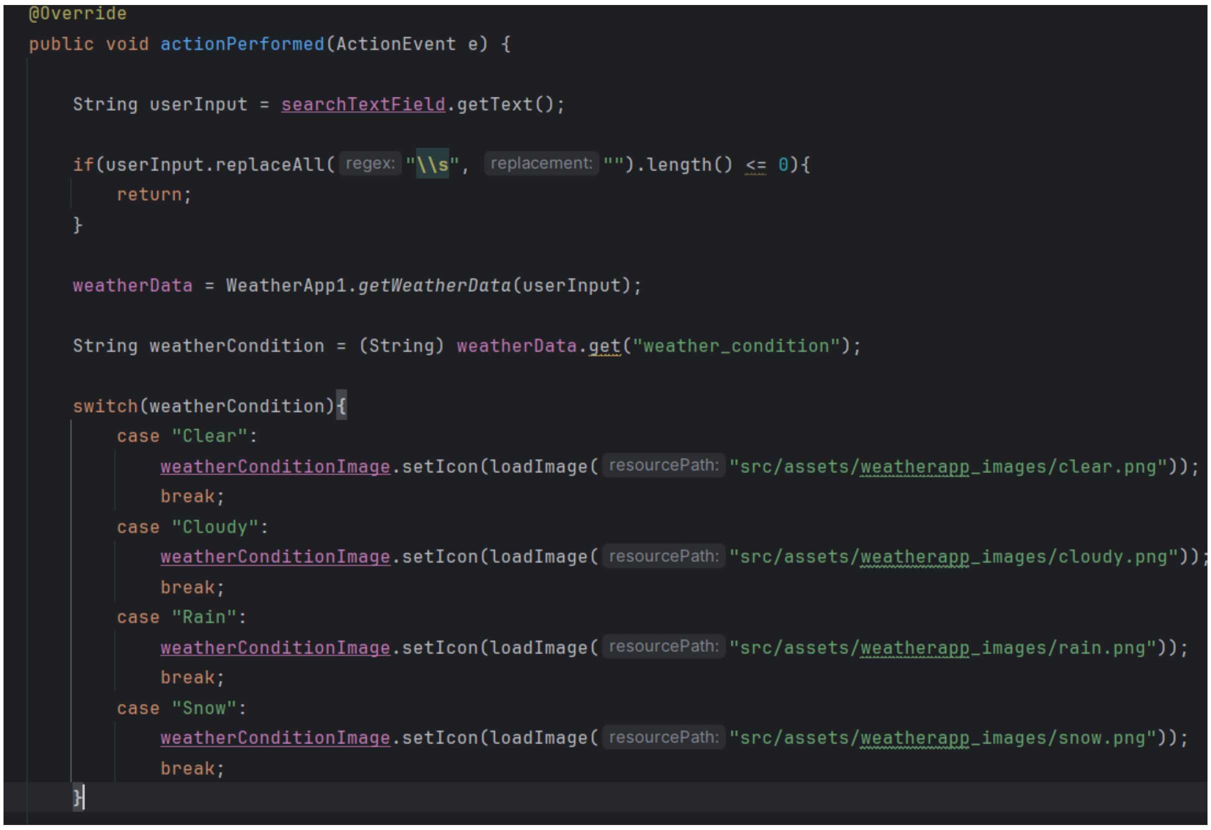
```
private void addGuiComponents(){  1 usage
    JTextField searchTextField = new JTextField();

    searchTextField.setBounds( x: 15,  y: 15,  width: 351,  height: 45);

    searchTextField.setFont(new Font( name: "Dialog", Font.PLAIN,  size: 24));

    add(searchTextField);

    JLabel weatherConditionImage = new JLabel(loadImage( resourcePath: "src/assets/weatherapp_images/cloudy.png"));
    weatherConditionImage.setBounds( x: 0,  y: 125,  width: 450,  height: 217);
    add(weatherConditionImage);

    JLabel temperatureText = new JLabel( text: "10 C");
    temperatureText.setBounds( x: 0,  y: 350,  width: 450,  height: 54);
    temperatureText.setFont(new Font( name: "Dialog", Font.BOLD,  size: 48));

    temperatureText.setHorizontalAlignment(SwingConstants.CENTER);
    add(temperatureText);

    JLabel weatherConditionDesc = new JLabel( text: "Cloudy");
    weatherConditionDesc.setBounds( x: 0,  y: 405,  width: 450,  height: 36);
    weatherConditionDesc.setFont(new Font( name: "Dialog", Font.PLAIN,  size: 32));
    weatherConditionDesc.setHorizontalAlignment(SwingConstants.CENTER);
    add(weatherConditionDesc);

    JLabel humidityImage = new JLabel(loadImage( resourcePath: "src/assets/weatherapp_images/humidity.png"));
    humidityImage.setBounds( x: 15,  y: 500,  width: 74,  height: 66);
    add(humidityImage);
```



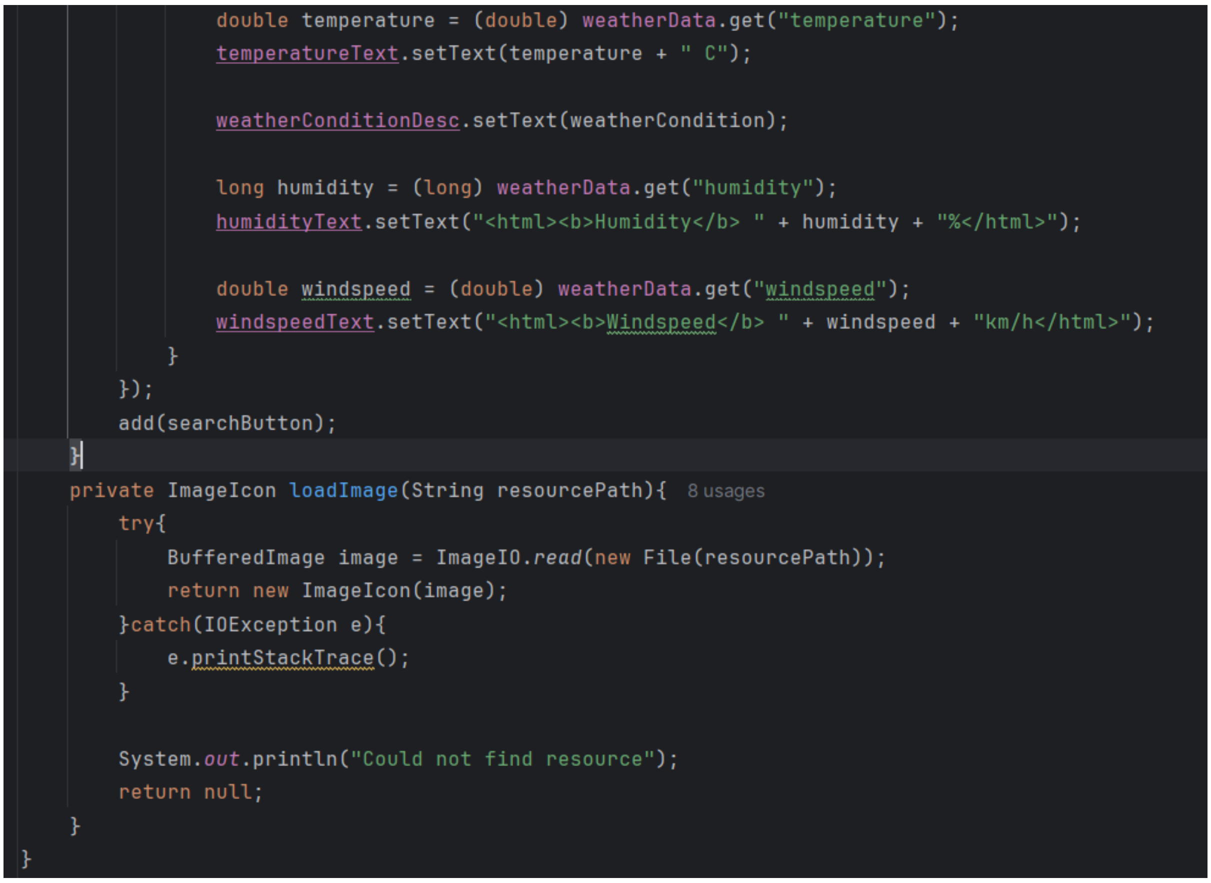
```
    JLabel humidityText = new JLabel( text: "<html><b>Humidity</b> 100%</html>");
    humidityText.setBounds( x: 90,  y: 500,  width: 85,  height: 55);
    humidityText.setFont(new Font( name: "Dialog", Font.PLAIN,  size: 16));
    add(humidityText);

    JLabel windspeedImage = new JLabel(loadImage( resourcePath: "src/assets/weatherapp_images/windspeed.png"));
    windspeedImage.setBounds( x: 220,  y: 500,  width: 74,  height: 66);
    add(windspeedImage);

    JLabel windspeedText = new JLabel( text: "<html><b>Windspeed</b> 15km/h</html>");
    windspeedText.setBounds( x: 310,  y: 500,  width: 85,  height: 55);
    windspeedText.setFont(new Font( name: "Dialog", Font.PLAIN,  size: 16));
    add(windspeedText);

    JButton searchButton = new JButton(loadImage( resourcePath: "src/assets/weatherapp_images/search.png"));

    searchButton.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
    searchButton.setBounds( x: 375,  y: 13,  width: 47,  height: 45);
    searchButton.addActionListener(new ActionListener() {
```

- actionPerformed

Here we get the location from the user input , then we retrieve weather data from the API , and depending on the condition matched we update the image of the weather condition , similarly we also update the temperature text , humidity and windspeed texts .



```java
@Override
public void actionPerformed(ActionEvent e) {

    String userInput = searchTextField.getText();

    if(userInput.replaceAll( regex: "\\s",  replacement: "").length() <= 0){
        return;
    }

    weatherData = WeatherApp1.getWeatherData(userInput);

    String weatherCondition = (String) weatherData.get("weather_condition");

    switch(weatherCondition){
        case "Clear":
            weatherConditionImage.setIcon(loadImage( resourcePath: "src/assets/weatherapp_images/clear.png"));
            break;
        case "Cloudy":
            weatherConditionImage.setIcon(loadImage( resourcePath: "src/assets/weatherapp_images/cloudy.png"));
            break;
        case "Rain":
            weatherConditionImage.setIcon(loadImage( resourcePath: "src/assets/weatherapp_images/rain.png"));
            break;
        case "Snow":
            weatherConditionImage.setIcon(loadImage( resourcePath: "src/assets/weatherapp_images/snow.png"));
            break;
    }
```



```
                double temperature = (double) weatherData.get("temperature");
                temperatureText.setText(temperature + " C");

                weatherConditionDesc.setText(weatherCondition);

                long humidity = (long) weatherData.get("humidity");
                humidityText.setText("<html><b>Humidity</b> " + humidity + "%</html>");

                double windspeed = (double) weatherData.get("windspeed");
                windspeedText.setText("<html><b>Windspeed</b> " + windspeed + "km/h</html>");
            }
        });
        add(searchButton);
    }
    private ImageIcon loadImage(String resourcePath){  8 usages
        try{
            BufferedImage image = ImageIO.read(new File(resourcePath));
            return new ImageIcon(image);
        }catch(IOException e){
            e.printStackTrace();
        }

        System.out.println("Could not find resource");
        return null;
    }
}
```

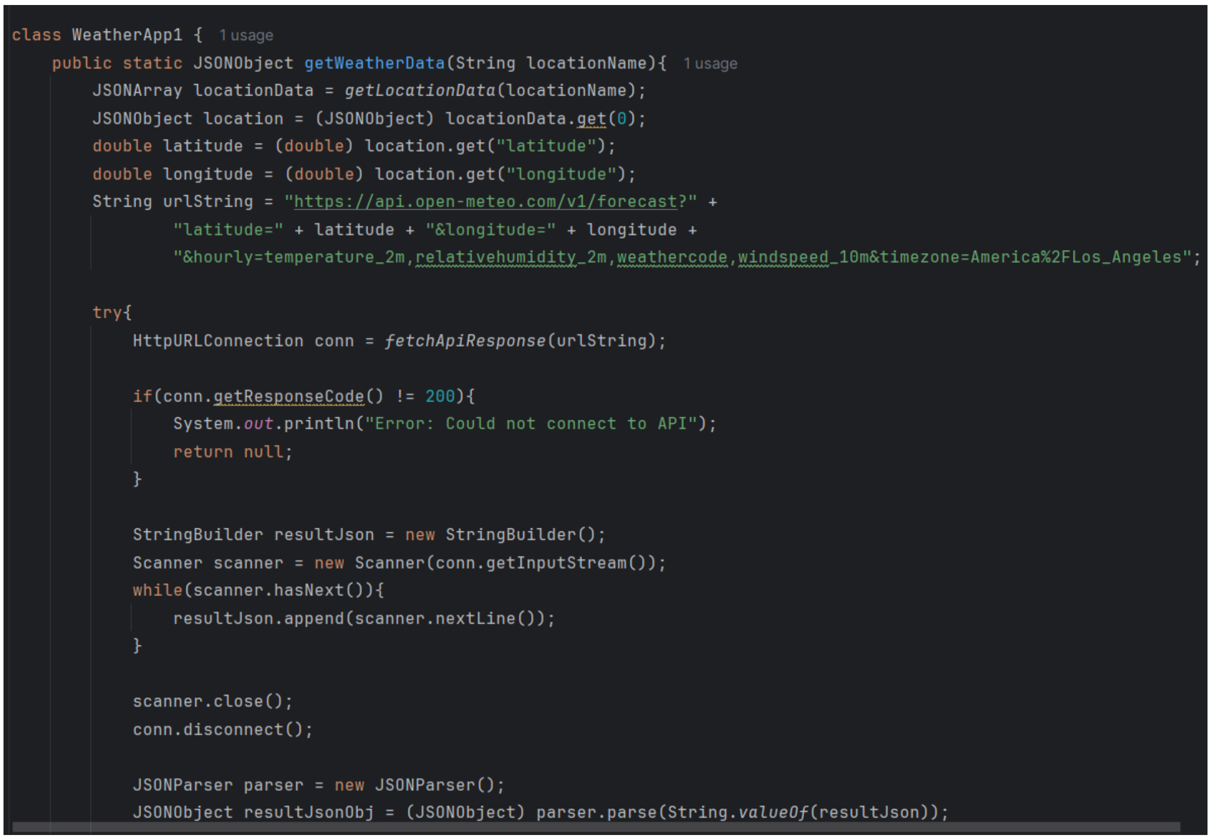
2) WeatherApp

In this class we retreive weather data from API - this backend logic will fetch the latest weather data from the external API and return it. The GUI which is built will display this data to the user for the input that they have given.

Methods

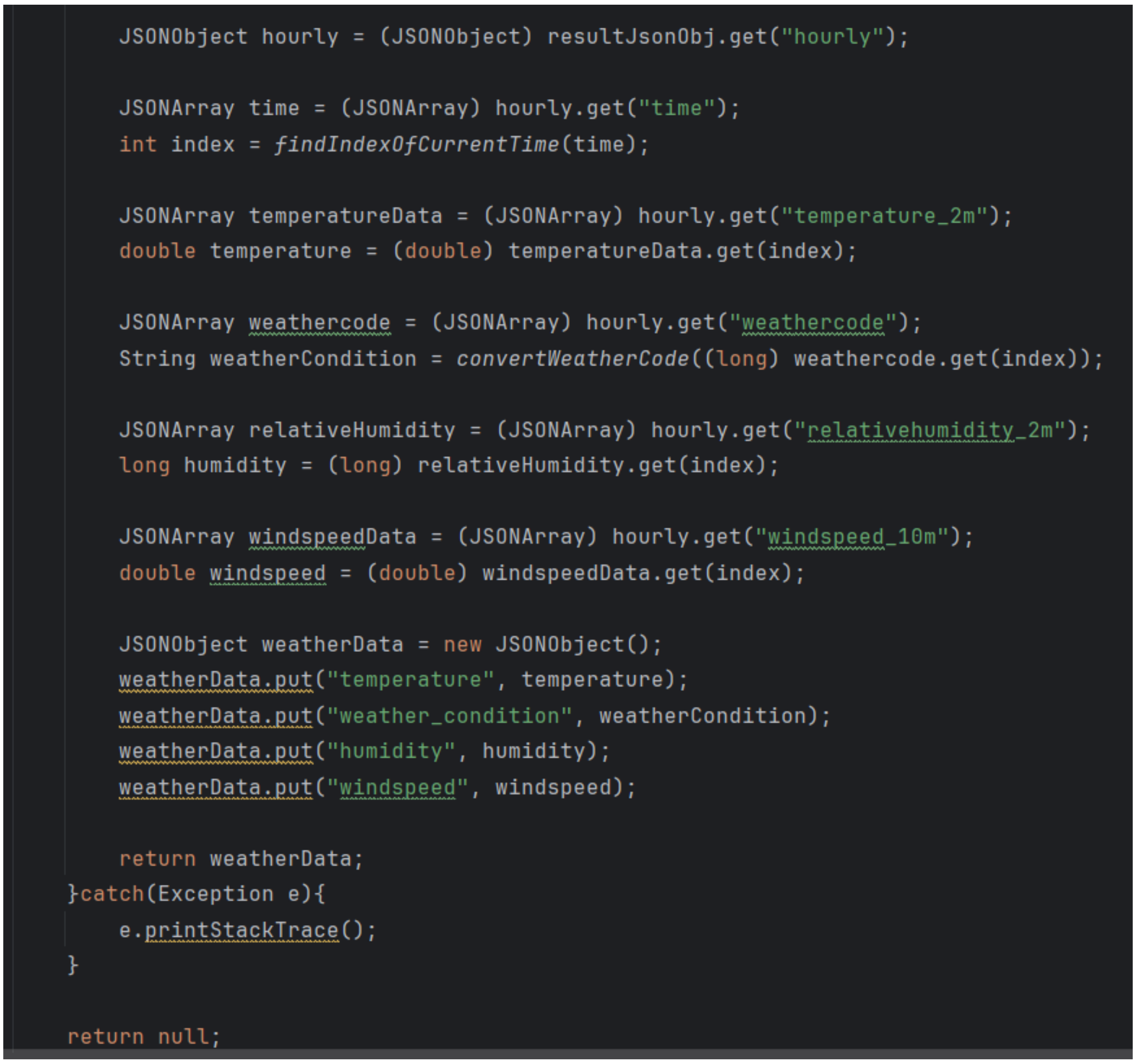
- getWeatherData

From this method we fetch data for the given location and get location from the External API , then we extract latitude and longitude data and then we build API request URL with the coordinates . In the try block we call api using HttpURLConnection and get response . If the response is 200 then it is a success . Later we store the result json data and parse through our data.



```
class WeatherApp1 {  1 usage
    public static JSONObject getWeatherData(String locationName){  1 usage
        JSONArray locationData = getLocationData(locationName);
        JSONObject location = (JSONObject) locationData.get(0);
        double latitude = (double) location.get("latitude");
        double longitude = (double) location.get("longitude");
        String urlString = "https://api.open-meteo.com/v1/forecast?" +
                "latitude=" + latitude + "&longitude=" + longitude +
                "&hourly=temperature_2m,relativehumidity_2m,weathercode,windspeed_10m&timezone=America%2FLos_Angeles";

        try{
            HttpURLConnection conn = fetchApiResponse(urlString);

            if(conn.getResponseCode() != 200){
                System.out.println("Error: Could not connect to API");
                return null;
            }

            StringBuilder resultJson = new StringBuilder();
            Scanner scanner = new Scanner(conn.getInputStream());
            while(scanner.hasNext()){
                resultJson.append(scanner.nextLine());
            }

            scanner.close();
            conn.disconnect();

            JSONParser parser = new JSONParser();
            JSONObject resultJsonObj = (JSONObject) parser.parse(String.valueOf(resultJson));
```

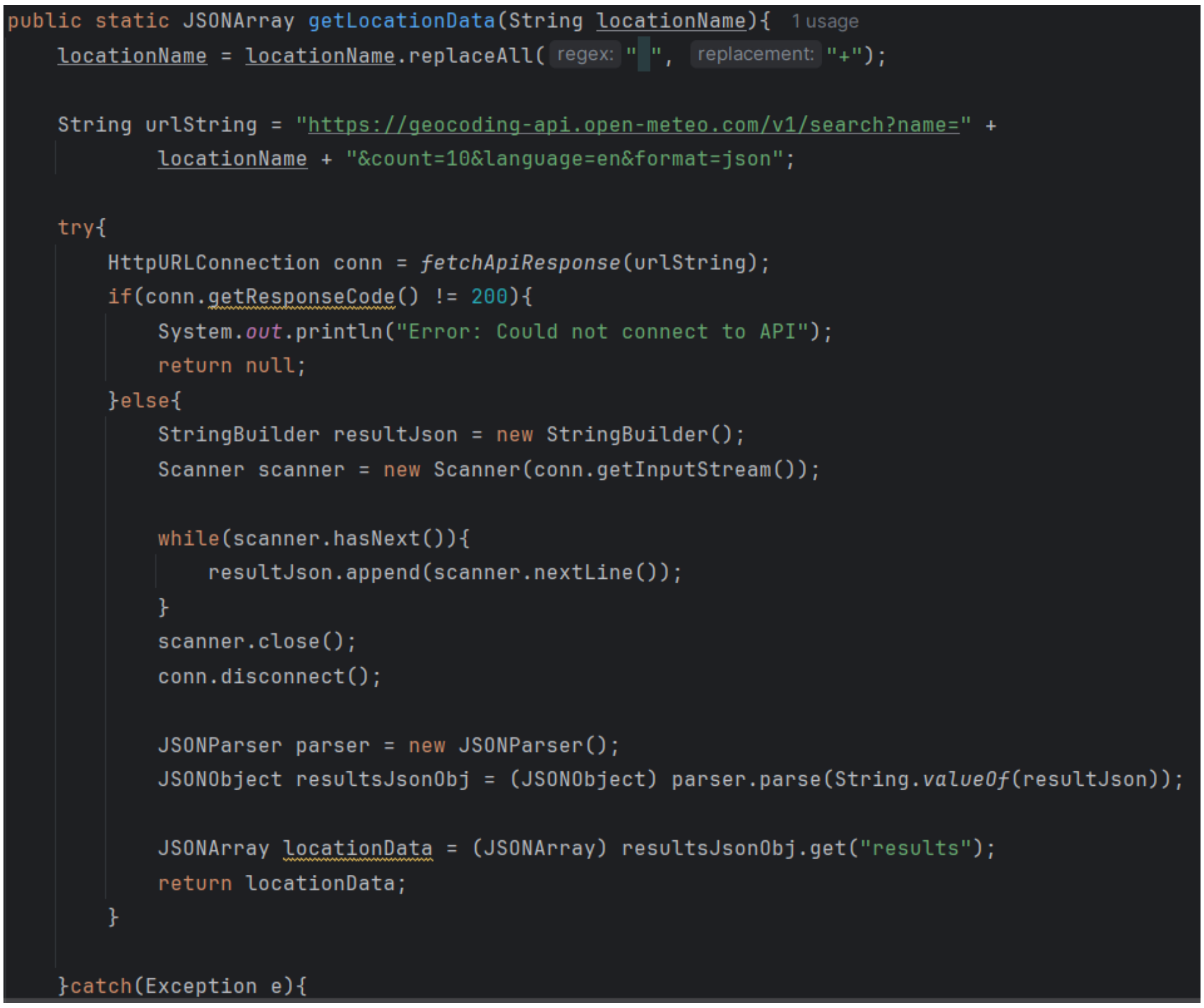
Now, the hourly data from the api is retrieved so that we can get the weather data of the current hour , so index of the current hour is required . For the retrieved hourly data , the temperature , windspeed , humidity and weather conditions are retrieved . To display this data in the WeatherAppGui. The json data object is created to access it in the WeatherAppGUI.



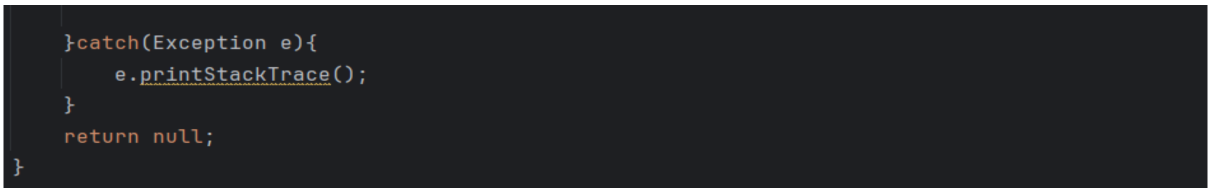
```
        JSONObject hourly = (JSONObject) resultJsonObj.get("hourly");

        JSONArray time = (JSONArray) hourly.get("time");
        int index = findIndexOfCurrentTime(time);

        JSONArray temperatureData = (JSONArray) hourly.get("temperature_2m");
        double temperature = (double) temperatureData.get(index);

        JSONArray weathercode = (JSONArray) hourly.get("weathercode");
        String weatherCondition = convertWeatherCode((long) weathercode.get(index));

        JSONArray relativeHumidity = (JSONArray) hourly.get("relativehumidity_2m");
        long humidity = (long) relativeHumidity.get(index);

        JSONArray windspeedData = (JSONArray) hourly.get("windspeed_10m");
        double windspeed = (double) windspeedData.get(index);

        JSONObject weatherData = new JSONObject();
        weatherData.put("temperature", temperature);
        weatherData.put("weather_condition", weatherCondition);
        weatherData.put("humidity", humidity);
        weatherData.put("windspeed", windspeed);

        return weatherData;
    }catch(Exception e){
        e.printStackTrace();
    }

    return null;
```

- getLocationData

In this method the geographic location of the user input passed is retrieved , the data is strored in resultJson variable , the resulting json data is stored in string builder , Later the Json string is parsed into json object



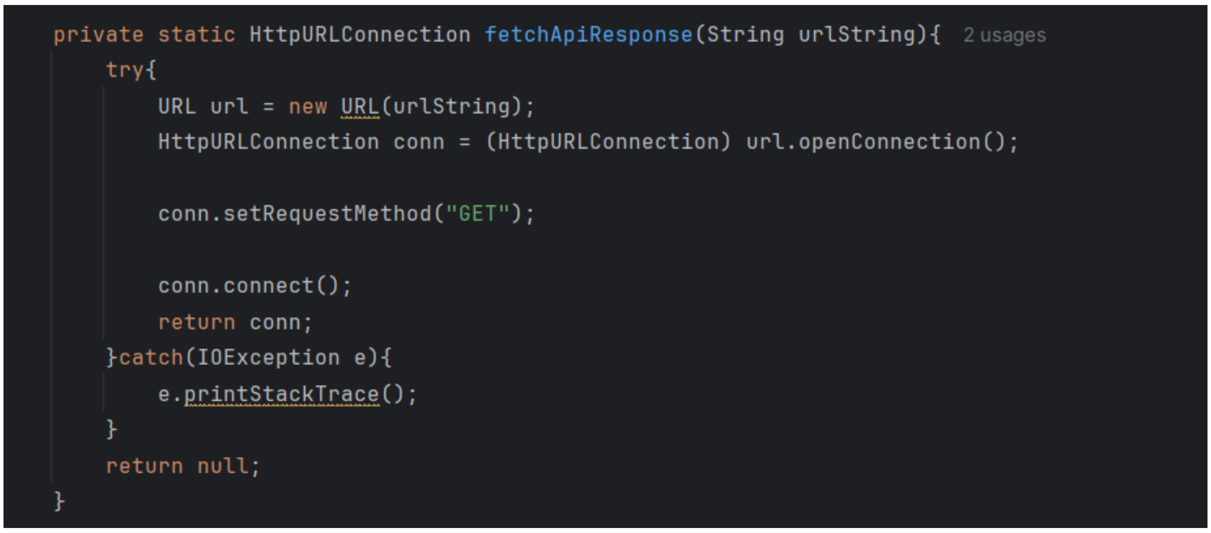
```java
public static JSONArray getLocationData(String locationName){  1 usage
    locationName = locationName.replaceAll( regex: " ",  replacement: "+");

    String urlString = "https://geocoding-api.open-meteo.com/v1/search?name=" +
            locationName + "&count=10&language=en&format=json";

    try{
        HttpURLConnection conn = fetchApiResponse(urlString);
        if(conn.getResponseCode() != 200){
            System.out.println("Error: Could not connect to API");
            return null;
        }else{
            StringBuilder resultJson = new StringBuilder();
            Scanner scanner = new Scanner(conn.getInputStream());

            while(scanner.hasNext()){
                resultJson.append(scanner.nextLine());
            }
            scanner.close();
            conn.disconnect();

            JSONParser parser = new JSONParser();
            JSONObject resultsJsonObj = (JSONObject) parser.parse(String.valueOf(resultJson));

            JSONArray locationData = (JSONArray) resultsJsonObj.get("results");
            return locationData;
        }

    }catch(Exception e){
```



```java
    }catch(Exception e){
        e.printStackTrace();
    }
    return null;
}
```

- fetchApiResponse

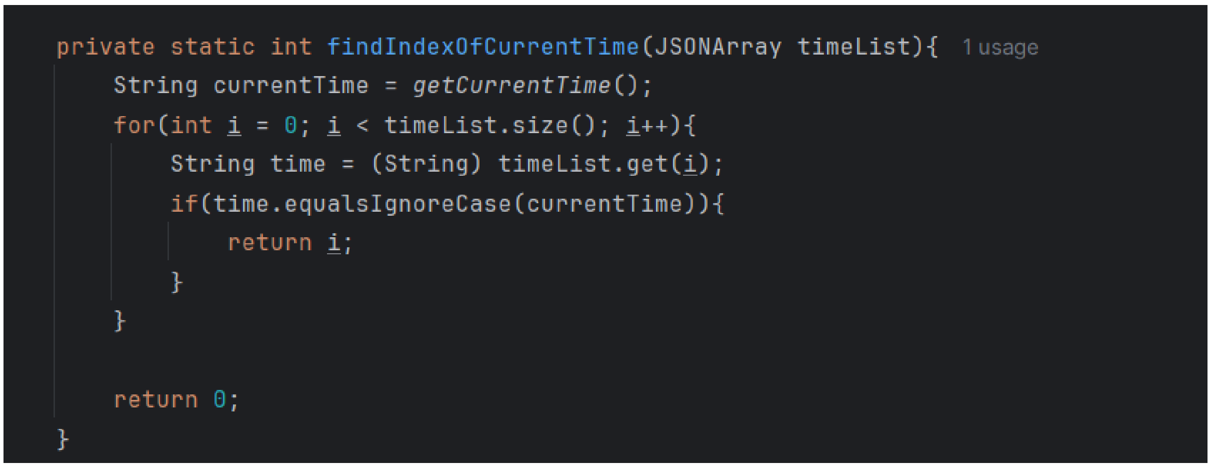
In this method the connection is being set by the URL class . Usin the HttpURLConnection class the connection to api is being built.



```java
private static HttpURLConnection fetchApiResponse(String urlString){  2 usages
    try{
        URL url = new URL(urlString);
        HttpURLConnection conn = (HttpURLConnection) url.openConnection();

        conn.setRequestMethod("GET");

        conn.connect();
        return conn;
    }catch(IOException e){
        e.printStackTrace();
    }
    return null;
}
```

- findIndexOfCurrentTime

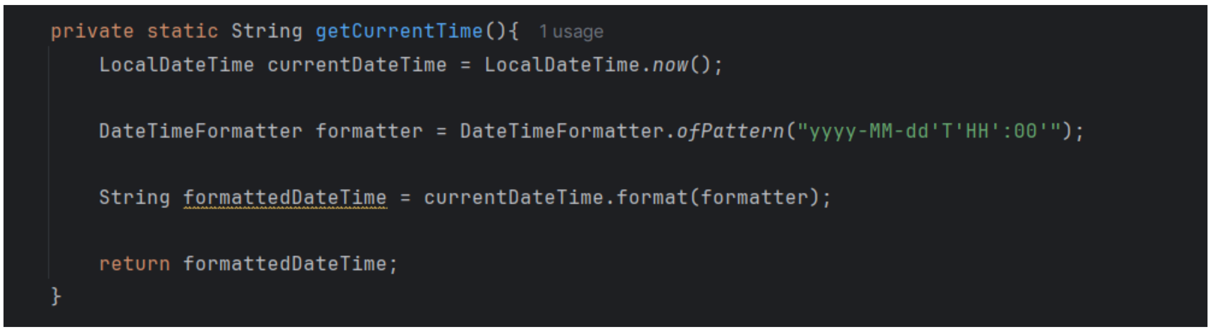
In this method we iterate through the time list and see which one matches our current time and get it in the next method



```java
private static int findIndexOfCurrentTime(JSONArray timeList){  1 usage
    String currentTime = getCurrentTime();
    for(int i = 0; i < timeList.size(); i++){
        String time = (String) timeList.get(i);
        if(time.equalsIgnoreCase(currentTime)){
            return i;
        }
    }

    return 0;
}
```

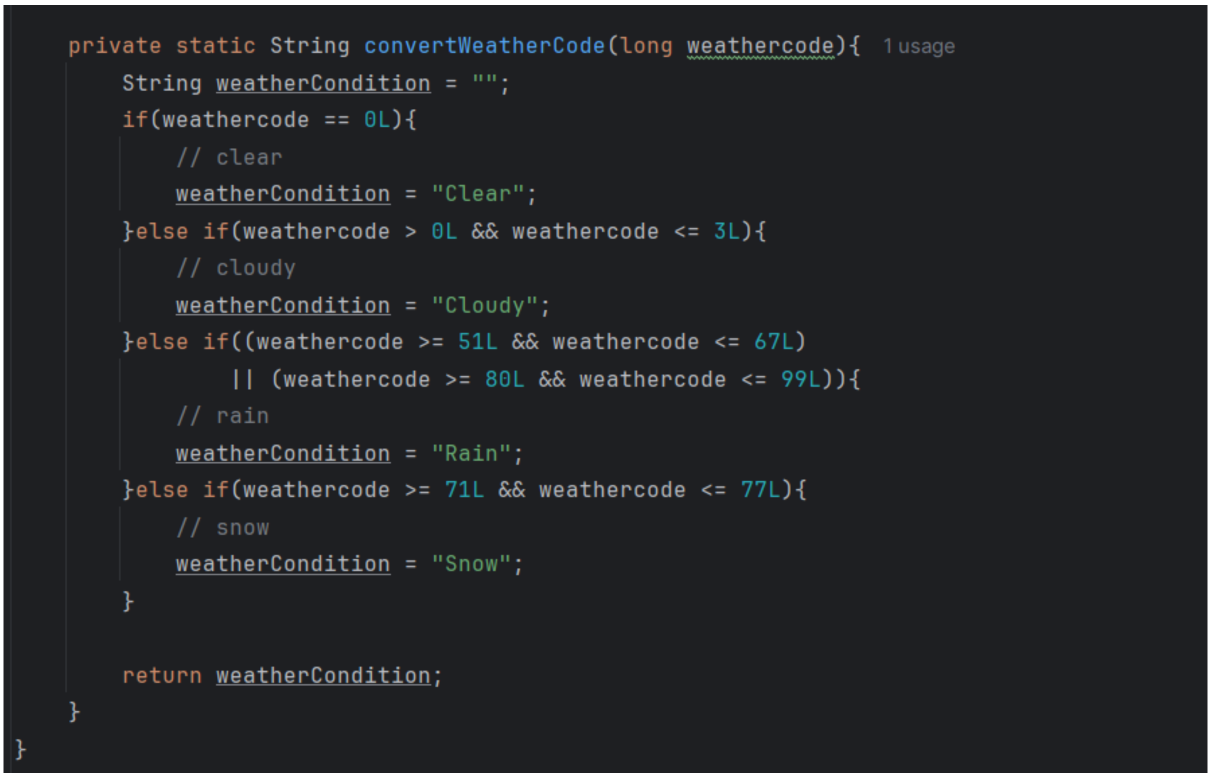
- getCurrentTime

This method is used to get current date and time and then it is formatted to 2023-09-02T00:00 because this is how it is read by the API. Later, we format and return the current date and time


```java
    private static String getCurrentTime(){  1 usage
        LocalDateTime currentDateTime = LocalDateTime.now();

        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd'T'HH':00'");

        String formattedDateTime = currentDateTime.format(formatter);

        return formattedDateTime;
    }
```

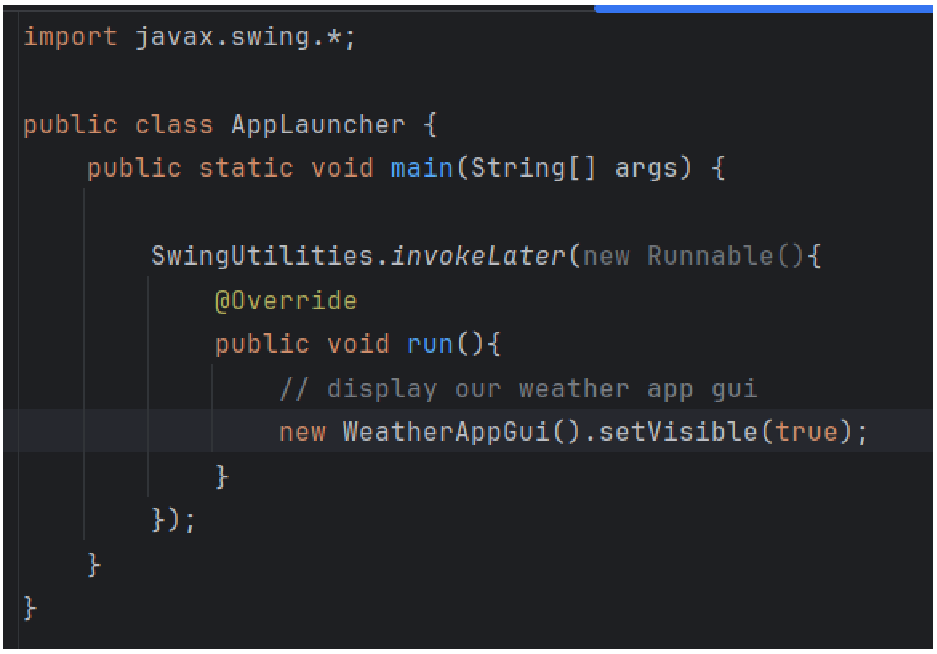
- currentWeatherCode

According to the weather interpretation codes there are a certain number codes assigned to the weather condition , these weather condition are set with a few modifications for better usability of the codes using if-else if-ladder.


```java
    private static String convertWeatherCode(long weathercode){  1 usage
        String weatherCondition = "";
        if(weathercode == 0L){
            // clear
            weatherCondition = "Clear";
        }else if(weathercode > 0L && weathercode <= 3L){
            // cloudy
            weatherCondition = "Cloudy";
        }else if((weathercode >= 51L && weathercode <= 67L)
                    || (weathercode >= 80L && weathercode <= 99L)){
            // rain
            weatherCondition = "Rain";
        }else if(weathercode >= 71L && weathercode <= 77L){
            // snow
            weatherCondition = "Snow";
        }

        return weatherCondition;
    }
}
```

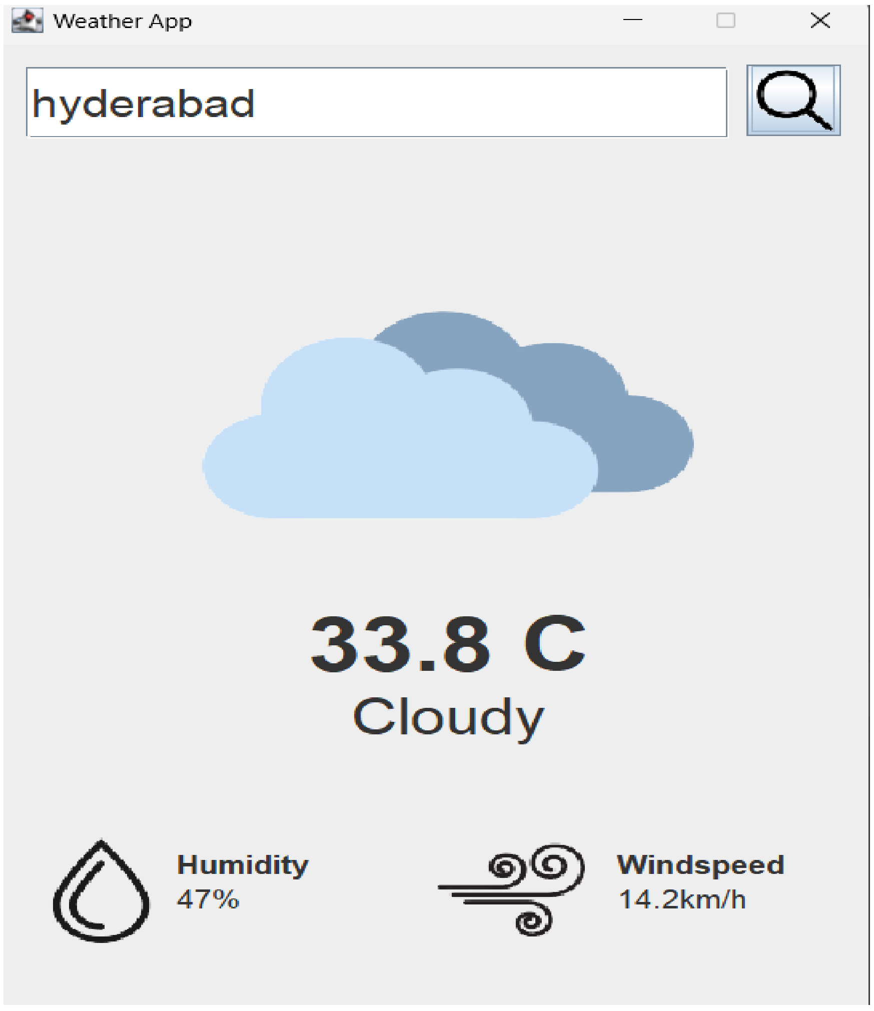
3) AppLauncher

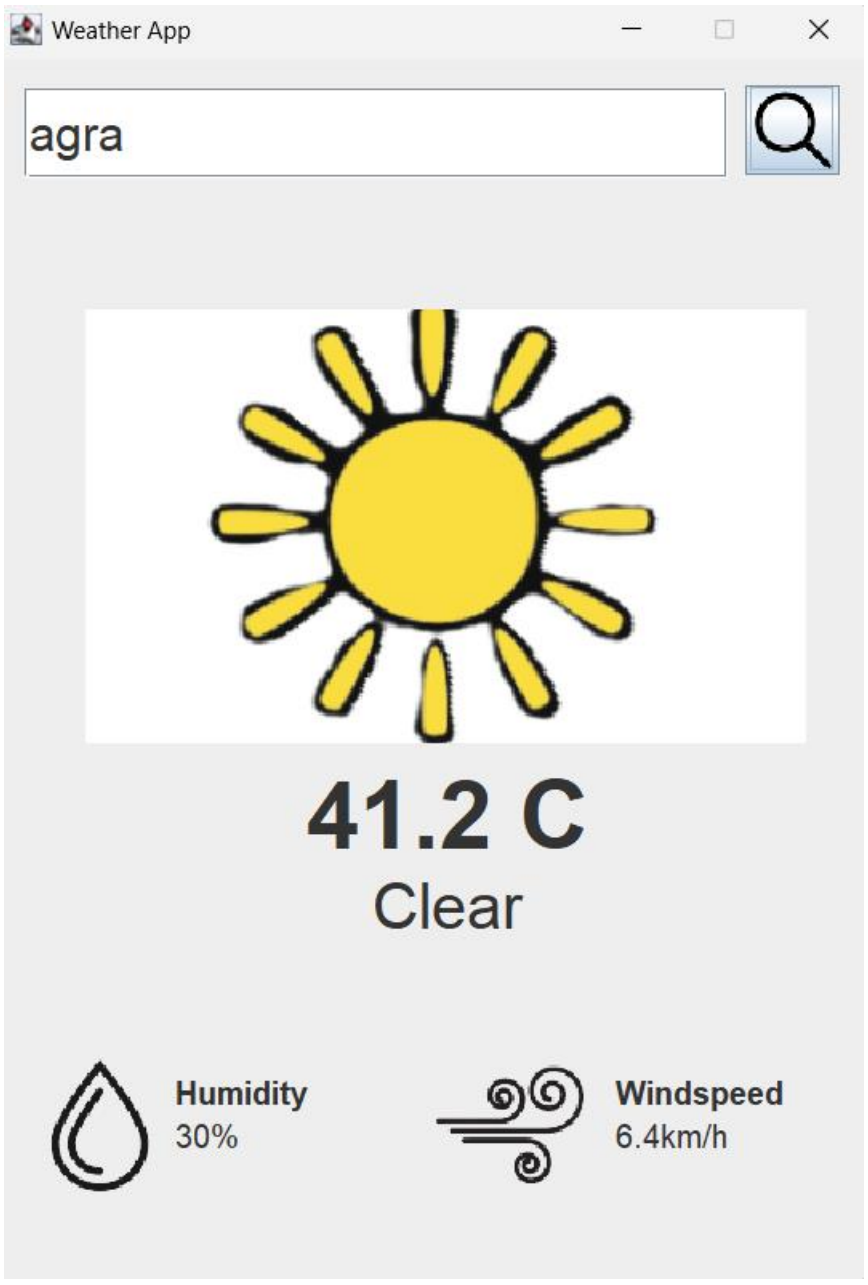
This class is the main through which we run the entire code and get the display of Graphical User Interface which is created .



```java
import javax.swing.*;

public class AppLauncher {
    public static void main(String[] args) {

        SwingUtilities.invokeLater(new Runnable(){
            @Override
            public void run(){
                // display our weather app gui
                new WeatherAppGui().setVisible(true);
            }
        });
    }
}
```

6)OUTPUT

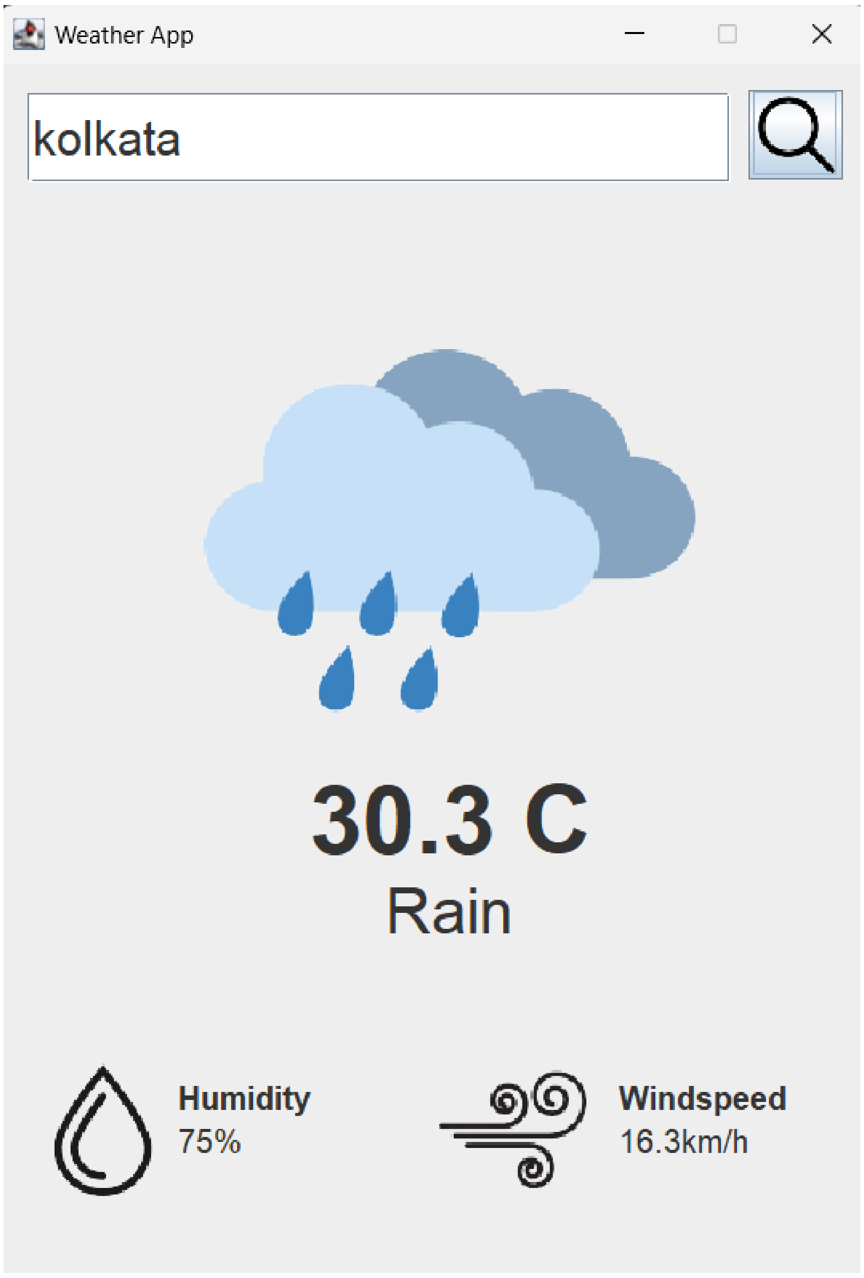




41.2 C
Clear

Humidity
30%

Windspeed
6.4km/h



30.3 C
Rain

Humidity
75%

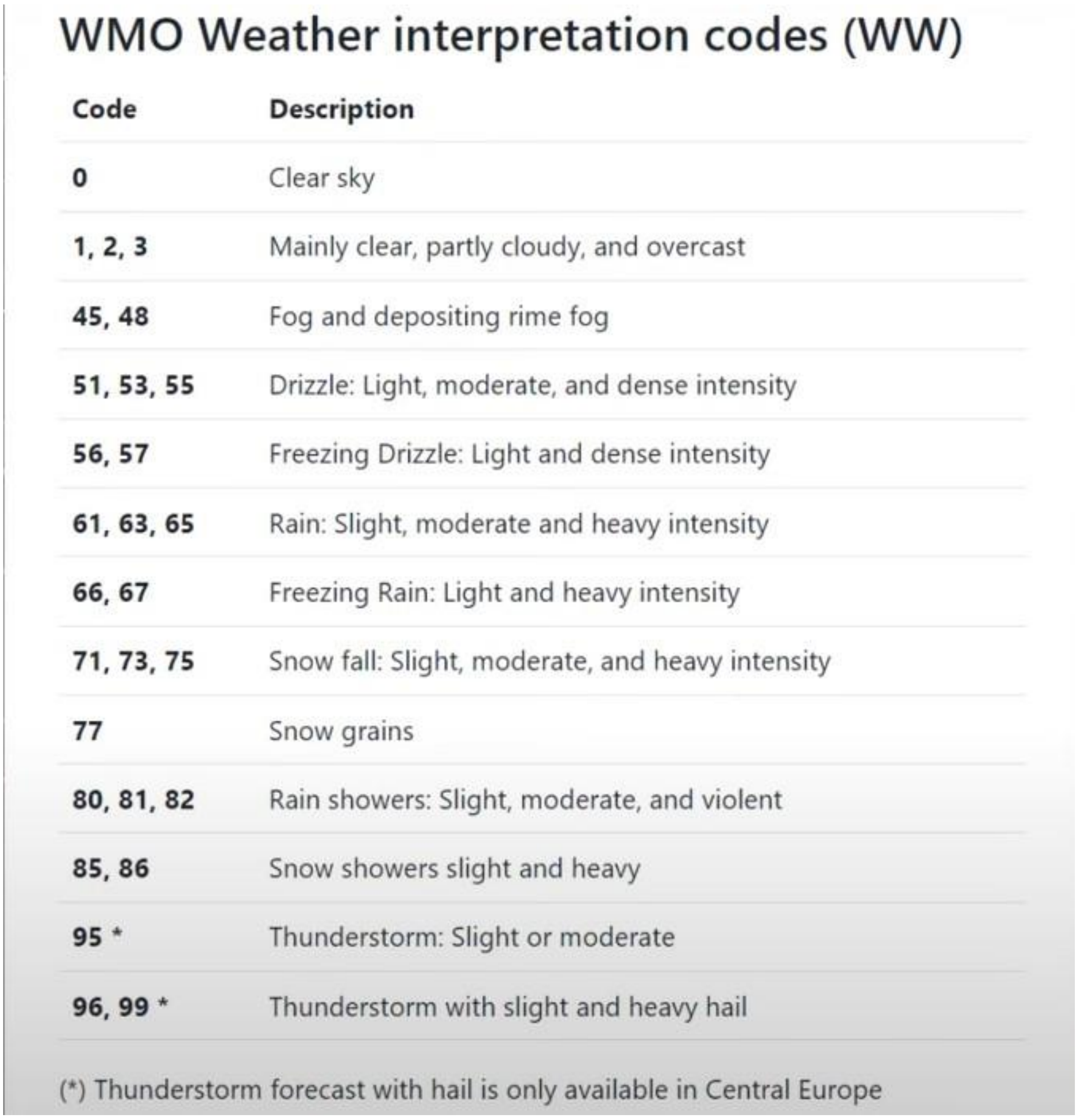
Windspeed
16.3km/h

7)REFERENCES

- Javat Point
- GeekforGeeks.org
- Geolocation API – for retrieving co-ordinates
  https://open-meteo.com/en/docs/geocoding-api

- Weather Forecast API - for weather data
  https://open-meteo.com/en/docs

- Weather Interpretation Codes



WMO Weather interpretation codes (WW)

Code	Description
0	Clear sky
1, 2, 3	Mainly clear, partly cloudy, and overcast
45, 48	Fog and depositing rime fog
51, 53, 55	Drizzle: Light, moderate, and dense intensity
56, 57	Freezing Drizzle: Light and dense intensity
61, 63, 65	Rain: Slight, moderate and heavy intensity
66, 67	Freezing Rain: Light and heavy intensity
71, 73, 75	Snow fall: Slight, moderate, and heavy intensity
77	Snow grains
80, 81, 82	Rain showers: Slight, moderate, and violent
85, 86	Snow showers slight and heavy
95 *	Thunderstorm: Slight or moderate
96, 99 *	Thunderstorm with slight and heavy hail

(*) Thunderstorm forecast with hail is only available in Central Europe

8) CONCLUSION

This Weather Forecast Application provides us the real time weather data of the particular City / area that is in input by the user in the graphical user interface and retrieves data from the Weather Forecast API for that particular city / area and returns the

Components in the GUI displaying the temperature , humidity , windspeed and weather conditions in that region.