Introduction:

This Weather Application micro project is developed using Java and integrates GUI components through AWT and Swing. The primary objective of the project is to provide users with real-time weather information of any city by accessing data from an external weather API, specifically OpenWeatherMap. The application enables users to input a city name, and upon submission, it retrieves and displays current weather conditions including temperature, weather description, and humidity.

By combining Java's GUI capabilities with API integration, this project serves as a practical implementation of core programming concepts like event handling, networking. It also highlights how desktop applications can be made dynamic and interactive by connecting them with live web services. This project is ideal for beginners who want to explore real-world applications of Java and understand the fundamentals of working with third-party APIs.

Program:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;


public class WeatherApp1 extends JFrame {
   private static final String API_KEY = "70c80a2fbbda76031e9f084ae41c0e6d"; // Replace with your OpenWeatherMap API key
   private JTextField cityField;
   private JTextArea resultArea;


   public WeatherApp1() {
      setTitle("Weather Application");
      setSize(500,400);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setLayout(new BorderLayout());


      // Input panel
      JPanel inputPanel = new JPanel();
      cityField = new JTextField(20);
      JButton getWeatherButton = new JButton("Get Weather");


      inputPanel.add(new JLabel("Enter City:"));
      inputPanel.add(cityField);
```

```
        inputPanel.add(getWeatherButton);

        // Result area
        resultArea = new JTextArea();
        resultArea.setEditable(false);
        JScrollPane scrollPane = new JScrollPane(resultArea);

        add(inputPanel, BorderLayout.NORTH);
        add(scrollPane, BorderLayout.CENTER);

        // Button action
        getWeatherButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String city = cityField.getText();
                getWeather(city);
            }
        });
    }

    private void getWeather(String city) {
        String urlString = "http://api.openweathermap.org/data/2.5/weather?q=" + city +
"&appid=" + API_KEY + "&units=metric"; // Metric for Celsius

        try {
            URL url = new URL(urlString);
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");

            int responseCode = conn.getResponseCode();
```

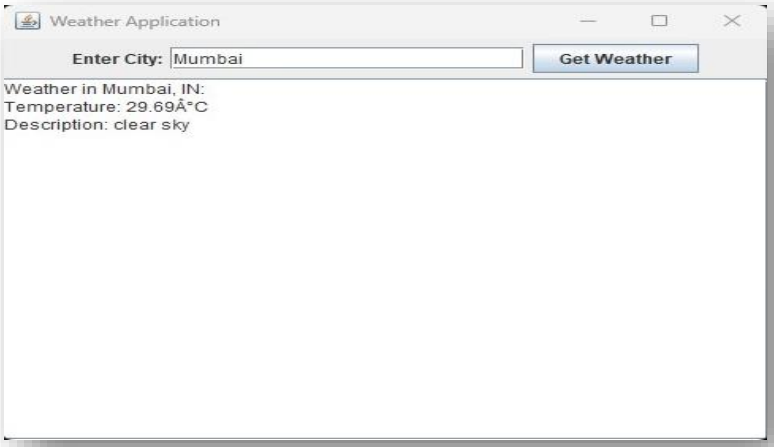
```
        if (responseCode == HttpURLConnection.HTTP_OK) {
            BufferedReader in = new BufferedReader(new
InputStreamReader(conn.getInputStream()));
            String inputLine;
            StringBuilder response = new StringBuilder();

            while ((inputLine = in.readLine()) != null) {
                response.append(inputLine);
            }
            in.close();
            // Parse the JSON response
            parseWeatherData(response.toString());
        } else {
            resultArea.setText("City not found. Please check the name and try again.");
        }
    } catch (IOException e) {
        resultArea.setText("Error fetching weather data: " + e.getMessage());
    }
  }
private void parseWeatherData(String jsonResponse) {
    try {
        String cityName = jsonResponse.split("\"name\":\"")[1].split("\"")[0];
        String country = jsonResponse.split("\"country\":\"")[1].split("\"")[0];
        String temperature = jsonResponse.split("\"temp\":")[1].split(",")[0];
        String weatherDescription = jsonResponse.split("\"description\":\"")[1].split("\"")[0];
        String result = "Weather in " + cityName + ", " + country + ":\n" +
                "Temperature: " + temperature + "°C\n" +
                "Description: " + weatherDescription;
        resultArea.setText(result);
    } catch (ArrayIndexOutOfBoundsException e) {
```

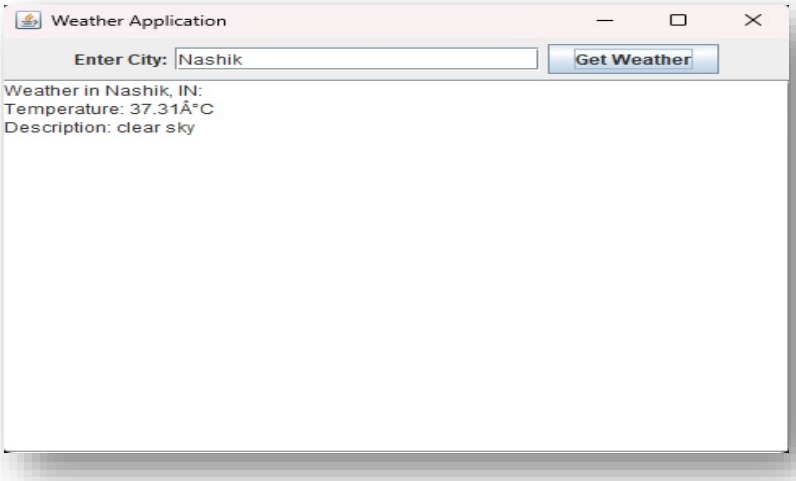
```
            resultArea.setText("Error parsing weather data.");
        }
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            WeatherApp1 app = new WeatherApp1();
            app.setVisible(true);
        });
    }
}
```

Output:

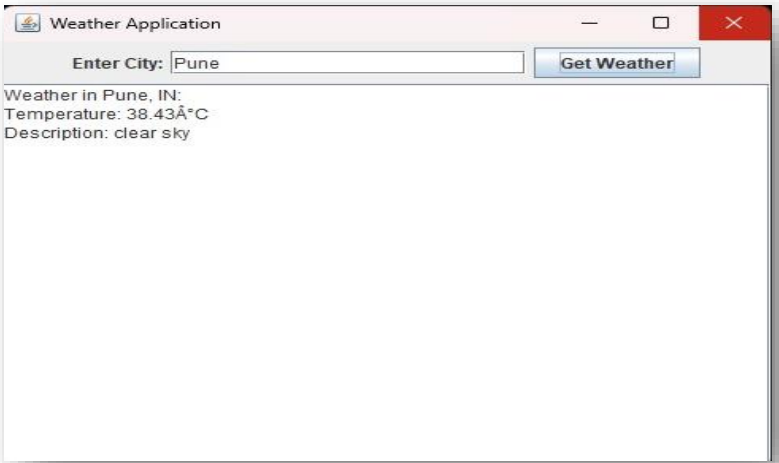
1.


Weather Application
Enter City: Mumbai
Get Weather
Weather in Mumbai, IN:
Temperature: 29.69Â°C
Description: clear sky

2.


Weather Application
Enter City: Nashik
Get Weather
Weather in Nashik, IN:
Temperature: 37.31Â°C
Description: clear sky

3.


Weather Application
Enter City: Pune
Get Weather
Weather in Pune, IN:
Temperature: 38.43Â°C
Description: clear sky

Working:

The Weather Application micro project is developed using Java with AWT and Swing for creating the graphical user interface, and it integrates a public weather API to fetch real-time weather data. When the application is launched, the user is presented with a simple GUI where they can enter the name of any city. Upon clicking the "Get Weather" button, the application sends an HTTP request to the OpenWeatherMap API using the entered city name. The API responds with a JSON object containing current weather details such as temperature, weather condition, and humidity. This response is then parsed using a JSON parsing library (like org.json), and the extracted data is displayed in the GUI using text components from Swing. AWT is used for basic layout management, while Swing provides components like JFrame, JTextField, JButton, and JTextArea to enhance the user interface. Overall, the project demonstrates how Java can be used to interact with web APIs and build user-friendly desktop applications using AWT and Swing.

Advantages:

1. **User-Friendly Interface:**
   The use of AWT and Swing provides a simple and interactive GUI that is easy to navigate for users of all ages.

2. **Real-Time Weather Data:**
   The application fetches live weather data from the OpenWeatherMap API, ensuring up-to-date and accurate information.

3. **Learning API Integration:**
   This project helps students understand how to connect Java applications with external web services using HTTP requests.

4. **Practical Use Case:**
   Weather apps are relevant and widely used; this project demonstrates how to build a real-world application in Java.

5. **Platform Independent:**
   As a Java-based application, it can run on any operating system that supports Java (Windows, macOS, Linux).

6. **JSON Data Handling:**
   Teaches the parsing and handling of JSON data, which is a commonly used data format in web and mobile applications.

7. **Lightweight Application:**
   The project is lightweight and does not require high system resources, making it suitable for any basic machine.

8. **Expandable Design:**
   The structure of the project allows future enhancements, such as adding a forecast feature, weather icons, or storing search history.

Hardware requirement:

1. Modern CPU (dual-core or better).
2. 2GB or more of RAM.
3. Adequate storage space for audio files.
4. Basic integrated graphics card.
5. Sound card or integrated audio chipset.
6. Compatible operating system with Java Runtime Environment (JRE) installed.

Software requirement:

1. Java Development Kit (JDK)
2. JavaFX or Swing for GUI development
3. Audio libraries or APIs (e.g., Java Sound API)
4. Integrated Development Environment (IDE)
5. Supported audio file formats
6. Dependencies management tool (e.g., Maven, Gradle)
7. Java Runtime Environment (JRE) for end-users

Conclusion:

The Weather Application micro project successfully demonstrates how Java can be used to build a functional and interactive desktop application by integrating AWT and Swing for the graphical user interface and utilizing an external weather API for real-time data. It also highlighted the practical use of APIs in modern application development. The project not only enhances understanding of core Java concepts but also provides a foundation for building more complex applications in the future. Overall, this application serves as a simple yet effective tool to learn and showcase skills in Java programming and API integration.

References:

1. https://openweathermap.org/api
2. https://docs.oracle.com/javase/8/docs/
3. https://mvnrepository.com/artifact/org.json/json
4. https://www.geeksforgeeks.org/java-swing/
5. https://stackoverflow.com/