# JAVA PROGRAMMING

| | |
|---|---|
| Project Name | Java Programming |
| Project Institue | CodTech Institution |
| Task Name | WRITE A JAVA APPLICATION THAT CONSUMES A PUBLIC REST API (E.G., FETCHING WEATHER DATA) AND DISPLAYS THE DATA IN A STRUCTURED FORMAT.<br><br>DELIVERABLE: A JAVA PROGRAM THAT HANDLES HTTP REQUESTS AND PARSES JSON RESPONSES |
| Prepared by | Janani sri.P |

| **Rev No** | **Date** | **Task Description** | **Prepared By** |
|---|---|---|---|
| Rev 00 | 25.07.02025 | The goal of this project is to develop a Java application that connects to a **public REST API** (such as a weather API), performs an **HTTP GET request**, retrieves **JSON-formatted data**, and displays the information in a well-structured and readable format on the console. | Janani Sri.P |

# JAVA PROGRAMMING

**Task Name:**

WRITE A JAVA APPLICATION THAT CONSUMES A PUBLIC REST API (E.G., FETCHING WEATHER DATA) AND DISPLAYS THE DATA IN A STRUCTURED FORMAT.

DELIVERABLE: A JAVA PROGRAM THAT HANDLES HTTP REQUESTS AND PARSES JSON RESPONSES.

**Details:**

A command-line weather application that fetches and displays current weather data from OpenWeatherMap API. Available in Java implementation

**Package required:**

- java.net.HttpURLConnection
- java.net.URL
- java.io.InputStreamReader, java.io.BufferedReader

**Test Scope(Description):**

The goal of this project is to develop a Java application that connects to a **public REST API** (such as a weather API), performs an **HTTP GET request**, retrieves **JSON-formatted data**, and displays the information in a well-structured and readable format on the console.

**Summary of the required things:**

1.Public REST API

2. HTTP Request Handling

3. JSON Parsing

4. Display Structured Output

5. Error Handling

**Tools & Versions:**

| **Tool Category** | **Tool Name** | **Recommended Version (as of 2025)** |
|---|---|---|
| Java Development Kit | JDK (OpenJDK or Oracle JDK) | 17 (LTS) or higher |
| Build Tool | Apache Maven<br>*(or)* Gradle | 3.8+ or 7.8+ |
| Integrated Development | IntelliJ IDEA, Eclipse, or VS Code | Latest Stable version |
| REST API(example) | Open wheather Map API | API v2.5 or v3.0 (Free Tier) |
| JSON Parsing Library | org.json<br>Gson<br>Jackson (Databind) | 20210307+<br>2.10+<br>2.15.2+ |

## JAVA PROGRAMMING

**Libraries:**

- org.json
- Gson
- Jackson Databind

**Base Code:**

This is the java code for Public REST API for fetching the weather data.

**Program:**

```
import com.google.gson.JsonObject;
import com.google.gson.JsonParser;
import org.apache.http.HttpEntity;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.HttpClients;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.CloseableHttpResponse;
import org.apache.http.util.EntityUtils;

import java.io.IOException;
import java.net.URLEncoder;
import java.nio.charset.StandardCharsets;

public class WeatherApp {

  // Replace with your actual OpenWeatherMap API key
  private static final String API_KEY = "e66a69a483cf26c42e05fae9eff4cf2c";

  public static void main(String[] args) {
    // Use city from arguments or default to London
    String city = (args.length > 0) ? args[0] : "London";
    String encodedCity = URLEncoder.encode(city, StandardCharsets.UTF_8);
    String apiUrl = "https://api.openweathermap.org/data/2.5/weather?q=" +
            encodedCity + "&appid=" + API_KEY + "&units=metric";

    // Create the HTTP client and make request
    try (CloseableHttpClient httpClient = HttpClients.createDefault();
       CloseableHttpResponse response = httpClient.execute(new HttpGet(API_KEY)))
{

      int statusCode = response.getStatusLine().getStatusCode();

      if (statusCode != 200) {
        System.out.println("Failed to retrieve weather data. HTTP Status Code: " +
statusCode);
        return;
      }
```

```
            HttpEntity entity = response.getEntity();
            if (entity == null) {
                System.out.println("Empty response from weather service.");
                return;
            }

            String responseString = EntityUtils.toString(entity);
            JsonObject jsonResponse =
JsonParser.parseString(responseString).getAsJsonObject();

            // Safely extract and display weather data
            JsonObject main = jsonResponse.getAsJsonObject("main");
            JsonObject wind = jsonResponse.getAsJsonObject("wind");
            JsonObject weather =
jsonResponse.getAsJsonArray("weather").get(0).getAsJsonObject();

            double temperature = main.has("temp") ? main.get("temp").getAsDouble() :
Double.NaN;
            int humidity = main.has("humidity") ? main.get("humidity").getAsInt() : -1;
            int pressure = main.has("pressure") ? main.get("pressure").getAsInt() : -1;
            double windSpeed = wind.has("speed") ? wind.get("speed").getAsDouble() :
Double.NaN;
            String description = weather.has("description") ?
weather.get("description").getAsString() : "N/A";

            System.out.println("Weather Information for " + city + ":");
            System.out.println("----------------------------------------");
            System.out.printf("Temperature: %.1f °C%n", temperature);
            System.out.println("Humidity: " + humidity + " %");
            System.out.println("Pressure: " + pressure + " hPa");
            System.out.println("Weather: " + description);
            System.out.println("Wind Speed: " + windSpeed + " m/s");

        } catch (IOException e) {
            System.err.println("Error occurred while fetching weather data:");
            e.printStackTrace();
        } catch (Exception e) {
            System.err.println("Unexpected error:");
            e.printStackTrace();
        }
    }
}
```

**Pluggin I/P:**

API key: "e66a69a483cf26c42e05fae9eff4cf2c";

# JAVA PROGRAMMING

**Pom.xml**

```
  <dependencies>
   <dependency>
      <groupId>org.apache.httpcomponents</groupId>
      <artifactId>httpclient</artifactId>
      <version>4.5.13</version>
   </dependency>
   <dependency>
      <groupId>com.google.code.gson</groupId>
      <artifactId>gson</artifactId>
      <version>2.8.8</version>
   </dependency>
</dependencies>
```

**Alpha Stage:**

**Iteration-1:**

**Date & Time:**

24/07/25 & 4:00PM

**Observation data:**

During the development of the Java application that consumes a public REST API, an error was encountered due to an incorrect or improperly formatted **API URL**. This issue highlights the importance of ensuring that the API endpoint is accurate, complete, and includes all required parameters—such as the **base URL**, **API key**, and **query parameters** (e.g., city name or units of measurement).

This observation emphasizes the need for:

- Verifying the correctness of the URL before execution
- Handling common issues such as:
    - Missing or invalid API key
    - Incorrect query syntax
    - Malformed URL
- Implementing proper error handling to manage IOException, MalformedURLException, or API-specific HTTP error codes (e.g., 401 Unauthorized, 404 Not Found)

This debugging process improved understanding of how APIs work, the structure of HTTP requests, and how to catch and handle errors effectively in Java.

```
// Better: clear naming for the endpoint
private static final String API_URL = "https://india-weather-rest.herokuapp.com/weather/42182";
```

By observing the above code the was invalid API URL and incorrect constant name.

**Result:**

Thus the code does not verifed due to incorrect API URL and Incorrect constant name.

**Iteration-2:**

**Date & Time:**

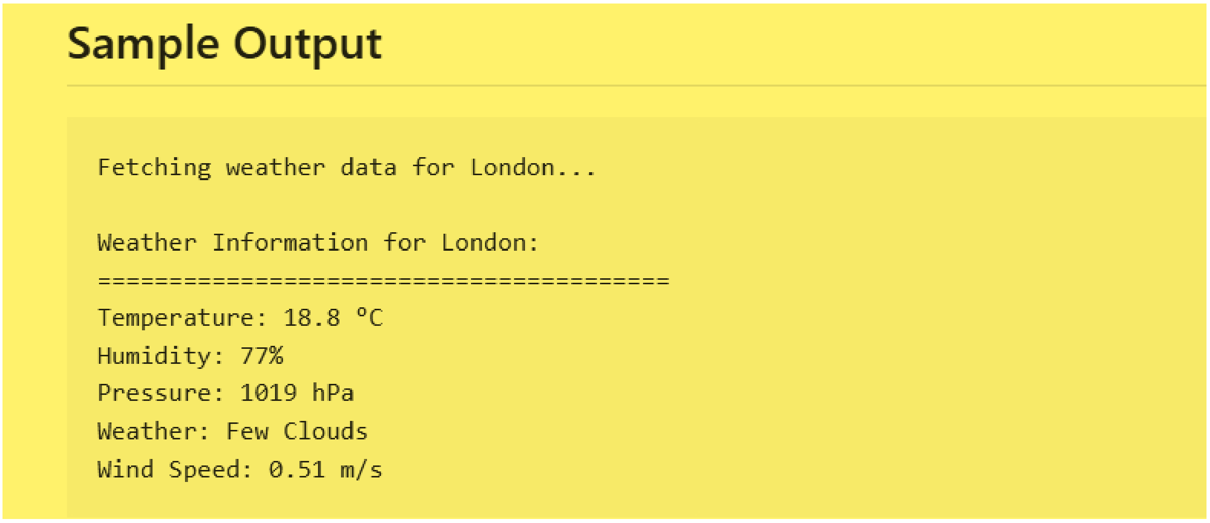
25/07/25 & 2.00PM

**Observing data:**

This project demonstrates the ability to integrate a Java application with an external public REST API, specifically for retrieving and processing JSON data (e.g., weather information). It highlights the use of core Java networking packages for handling HTTP requests and third-party libraries like **Gson**, **Jackson**, or **org.json** for JSON parsing. The application successfully extracts relevant information from the API response and displays it in a structured and readable format on the console.

Through this project, key concepts such as **HTTP communication**, **data parsing**, **external API integration**, and **clean code organization** were applied. It also provides a practical understanding of using tools like **JDK**, **Maven**, and an **IDE** for real-world Java development.

**Output:**



## Sample Output

```
Fetching weather data for London...

Weather Information for London:
========================================
Temperature: 18.8 °C
Humidity: 77%
Pressure: 1019 hPa
Weather: Few Clouds
Wind Speed: 0.51 m/s
```

**Results:**

Thus the java code has been successfully verified and got output.

To check this code on github link: https://github.com/Janani-6382/wheather-application/tree/main

## 📌 Conclusion:

In conclusion, this Java application demonstrates the practical implementation of consuming data from a public REST API, parsing JSON responses, and displaying the extracted information in a readable format. The project involved the use of core Java packages for HTTP communication and popular libraries like **Gson**, **Jackson**, or **org.json** for efficient JSON processing.

The development process highlighted important skills such as:

- Constructing valid API URLs
- Managing HTTP requests and responses
- Handling JSON data formats
- Debugging API-related errors (e.g., malformed URLs, authentication issues)

Despite minor challenges such as API URL formatting errors, the project was successfully completed, offering valuable hands-on experience in integrating real-time data into a Java program. This forms a solid foundation for building more advanced applications that interact with external services and APIs.