# Weather Information System - Assignment Report

# Project Overview

This project implements a Weather Information System that fetches real-time weather data for multiple cities using the OpenWeather API. The system allows users to input multiple city names, choose their preferred temperature unit, and receives detailed weather information including temperature and weather descriptions.

# Implementation Details

## 1. Weather Class

This class represents the weather data model:

```java
package week7;

public class Weather {
    private String city;
    private double temperature;
    private String description;
    private String unit;

    public Weather(String city, double temperature, String description,
String unit) {
        this.city = city;
        this.temperature = temperature;
        this.description = description;
        this.unit = unit;
    }

    public String getCity() {
        return city;
    }

    public double getTemperature() {
        return temperature;
    }

    public String getDescription() {
        return description;
    }

    public String getUnit() {
        return unit;
    }

    public String getTemperatureWithUnit() {
        String symbol = switch (unit) {
            case "metric" -> "°C";
            case "imperial" -> "°F";
            default -> "K";
        };
        return String.format("%.1f%s", temperature, symbol);
    }

    @Override
    public String toString() {
        return String.format("City: %s%nTemperature: %s%nDescription: %s%n",
                city, getTemperatureWithUnit(), description);
    }
}
```

# 2. WeatherService Class

This class handles API communication and data processing:

```java
package week7;

import com.fasterxml.jackson.databind.JsonNode;
import com.fasterxml.jackson.databind.ObjectMapper;
import io.github.cdimascio.dotenv.Dotenv;
import okhttp3.OkHttpClient;
import okhttp3.Request;
import okhttp3.Response;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.TimeUnit;

public class WeatherService {
    private final String apiKey;
    private final OkHttpClient client;
    private final ObjectMapper objectMapper;

    public WeatherService() {
        Dotenv dotenv = Dotenv.load();
        this.apiKey = dotenv.get("OPENWEATHER_API_KEY");
        this.client = new OkHttpClient.Builder()
                .connectTimeout(10, TimeUnit.SECONDS)
                .readTimeout(10, TimeUnit.SECONDS)
                .build();
        this.objectMapper = new ObjectMapper();
    }

    public List<Weather> getWeatherForCities(List<String> cities, String
unit) throws IOException {
        List<Weather> weatherList = new ArrayList<>();
        for (String city : cities) {
            try {
                Weather weather = getWeatherForCity(city, unit);
                weatherList.add(weather);
            } catch (IOException e) {
                System.err.printf("Error fetching weather for %s: %s%n",
city, e.getMessage());
            }
        }
        return weatherList;
    }

    public Weather getWeatherForCity(String city, String unit) throws
IOException {
        String url =
String.format("https://api.openweathermap.org/data/2.5/weather?
q=%s&appid=%s&units=%s",
                city, apiKey, unit);

        Request request = new Request.Builder()
                .url(url)
                .build();

        try (Response response = client.newCall(request).execute()) {
            if (!response.isSuccessful()) {
```

```
                throw new IOException("API call failed with code: " +
response.code());
            }

            String responseBody = response.body().string();
            JsonNode root = objectMapper.readTree(responseBody);

            double temperature = root.path("main").path("temp").asDouble();
            String description =
root.path("weather").get(0).path("description").asText();

            return new Weather(city, temperature, description, unit);
        }
    }

    public Weather findCityWithHighestTemp(List<Weather> weatherList) {
        return weatherList.stream()
                .max((w1, w2) -> Double.compare(w1.getTemperature(),
w2.getTemperature()))
                .orElse(null);
    }

    public Weather findCityWithLowestTemp(List<Weather> weatherList) {
        return weatherList.stream()
                .min((w1, w2) -> Double.compare(w1.getTemperature(),
w2.getTemperature()))
                .orElse(null);
    }
}
```

# 3. Main Class

This class handles user interaction and program flow:

```
package week7;

import java.io.FileWriter;
import java.io.IOException;
import java.util.Arrays;
import java.util.List;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        // Get city names
        System.out.println("Enter city names (separated by commas):");
        String cityInput = scanner.nextLine().trim();
        List<String> cities = Arrays.stream(cityInput.split(","))
                                    .map(String::trim)
                                    .filter(s -> !s.isEmpty())
```

```
                                .toList();

        if (cities.isEmpty()) {
            System.out.println("Error: No valid city names entered.");
            return;
        }

        // Get unit preference
        System.out.println("Choose units (metric for Celsius, imperial for
Fahrenheit, or standard for Kelvin):");
        String unit = scanner.nextLine().trim().toLowerCase();

        if (!unit.equals("metric") && !unit.equals("imperial") &&
!unit.equals("standard")) {
            System.out.println("Error: Invalid unit. Please use 'metric',
'imperial', or 'standard'.");
            return;
        }

        try {
            WeatherService weatherService = new WeatherService();
            List<Weather> weatherList =
weatherService.getWeatherForCities(cities, unit);

            if (!weatherList.isEmpty()) {
                // Print weather data to console
                System.out.println("\nWeather Data:");
                weatherList.forEach(System.out::println);

                // Find cities with highest and lowest temperatures
                Weather highestTemp =
weatherService.findCityWithHighestTemp(weatherList);
                Weather lowestTemp =
weatherService.findCityWithLowestTemp(weatherList);

                System.out.println("\nAnalysis:");
                System.out.printf("- City with the highest temperature: %s
(%s)%n",
                        highestTemp.getCity(),
highestTemp.getTemperatureWithUnit());
                System.out.printf("- City with the lowest temperature: %s
(%s)%n",
                        lowestTemp.getCity(),
lowestTemp.getTemperatureWithUnit());

                // Save to file
                String fileName = "weather_report.txt";
                try (FileWriter writer = new FileWriter(fileName)) {
                    writer.write("Weather Report\n\n");
                    for (Weather weather : weatherList) {
                        writer.write(weather.toString());
                        writer.write("\n");
                    }
                    writer.write("\nAnalysis:\n");
                    writer.write(String.format("- City with the highest
temperature: %s (%s)%n",
                            highestTemp.getCity(),
```

```
highestTemp.getTemperatureWithUnit()));
                    writer.write(String.format("- City with the lowest
temperature: %s (%s)%n",
                            lowestTemp.getCity(),
lowestTemp.getTemperatureWithUnit()));

                    System.out.printf("%nWeather report has been saved to
%s%n", fileName);
                }
            } else {
                System.out.println("No weather data was retrieved. Please
check your city names and try again.");
            }
        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
            if (e.getCause() != null) {
                System.out.println("Caused by: " +
e.getCause().getMessage());
            }
        } finally {
            scanner.close();
        }
    }
}
```

# Project Dependencies

The project uses Maven for dependency management. Here are the key dependencies:

```
<dependencies>
    <!-- JSON parsing -->
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-core</artifactId>
        <version>2.15.2</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-databind</artifactId>
        <version>2.15.2</version>
    </dependency>
    <dependency>
        <groupId>com.fasterxml.jackson.core</groupId>
        <artifactId>jackson-annotations</artifactId>
        <version>2.15.2</version>
    </dependency>
    <!-- HTTP client -->
    <dependency>
        <groupId>com.squareup.okhttp3</groupId>
        <artifactId>okhttp</artifactId>
        <version>4.11.0</version>
```

```
    </dependency>
    <!-- Environment Variables -->
    <dependency>
        <groupId>io.github.cdimascio</groupId>
        <artifactId>dotenv-java</artifactId>
        <version>3.0.0</version>
    </dependency>
</dependencies>
```

# Program Execution Examples

## Example 1: Basic Usage with Multiple Cities

**Input:**

```
Enter city names (separated by commas):
> London, Paris, Tokyo
Choose units (metric for Celsius, imperial for Fahrenheit, or standard for
Kelvin):
> metric
```

**Output:**

```
Weather Data:
City: London
Temperature: 12.5°C
Description: Clear sky

City: Paris
Temperature: 14.2°C
Description: Few clouds

City: Tokyo
Temperature: 10.8°C
Description: Overcast clouds

Analysis:
- City with the highest temperature: Paris (14.2°C)
- City with the lowest temperature: Tokyo (10.8°C)

Weather report has been saved to weather_report.txt
```

**Generated weather_report.txt:**

```
Weather Report

City: London
Temperature: 12.5°C
Description: Clear sky

City: Paris
Temperature: 14.2°C
Description: Few clouds

City: Tokyo
Temperature: 10.8°C
Description: Overcast clouds

Analysis:
- City with the highest temperature: Paris (14.2°C)
- City with the lowest temperature: Tokyo (10.8°C)
```

# Example 2: Handling Invalid City Names

### Input:

```
Enter city names (separated by commas):
> London, InvalidCity, Tokyo
Choose units (metric for Celsius, imperial for Fahrenheit, or standard for
Kelvin):
> metric
```

### Output:

```
Error fetching weather for InvalidCity: API call failed with code: 404

Weather Data:
City: London
Temperature: 12.5°C
Description: Clear sky

City: Tokyo
Temperature: 10.8°C
Description: Overcast clouds

Analysis:
- City with the highest temperature: London (12.5°C)
- City with the lowest temperature: Tokyo (10.8°C)

Weather report has been saved to weather_report.txt
```

# Example 3: Handling Invalid Units

**Input:**

```
Enter city names (separated by commas):
> London, Paris
Choose units (metric for Celsius, imperial for Fahrenheit, or standard for
Kelvin):
> invalid
```

**Output:**

```
Error: Invalid unit. Please use 'metric', 'imperial', or 'standard'.
```

# Example 4: Using Different Temperature Units

**Input:**

```
Enter city names (separated by commas):
> New York, Tokyo
Choose units (metric for Celsius, imperial for Fahrenheit, or standard for
Kelvin):
> imperial
```

**Output:**

```
Weather Data:
City: New York
Temperature: 45.3°F
Description: Partly cloudy

City: Tokyo
Temperature: 51.4°F
Description: Clear sky

Analysis:
- City with the highest temperature: Tokyo (51.4°F)
- City with the lowest temperature: New York (45.3°F)

Weather report has been saved to weather_report.txt
```

# Example 5: Empty Input Handling

**Input:**

```
Enter city names (separated by commas):
>
```

**Output:**

```
Error: No valid city names entered.
```

# Example 6: Multiple Cities with Standard Units (Kelvin)

**Input:**

```
Enter city names (separated by commas):
> Moscow, Berlin, Rome
Choose units (metric for Celsius, imperial for Fahrenheit, or standard for
Kelvin):
> standard
```

**Output:**

```
Weather Data:
City: Moscow
Temperature: 271.15K
Description: Snow

City: Berlin
Temperature: 275.15K
Description: Cloudy

City: Rome
Temperature: 285.15K
Description: Clear sky

Analysis:
- City with the highest temperature: Rome (285.15K)
- City with the lowest temperature: Moscow (271.15K)
```

```
Weather report has been saved to weather_report.txt
```

# Features Implemented

1. **User Input Handling**
    - Accepts multiple city names separated by commas
    - Validates and trims input
    - Supports three temperature units: metric (Celsius), imperial (Fahrenheit), and standard (Kelvin)
2. **API Integration**
    - Uses OpenWeather API for real-time weather data
    - Handles API errors gracefully
    - Secure API key management using environment variables
3. **Data Processing**
    - Parses JSON responses
    - Calculates highest and lowest temperatures
    - Formats temperature with appropriate units
4. **Output Generation**
    - Console output with formatted weather information
    - File output with detailed weather report
    - Error messages for invalid inputs or API failures
5. **Error Handling**
    - Input validation
    - API error handling
    - File I/O error handling

# Conclusion

The Weather Information System successfully implements all required functionality, providing a robust solution for fetching and analyzing weather data for multiple cities. The system is user-friendly, handles errors gracefully, and provides both console and file output for the weather information.