



KONGU ENGINEERING COLLEGE

(Autonomous)

Perundurai, Erode – 638 060

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING



WEATHER FORCASTING SYSTEM

AN MICROPROJECT REPORT

for

JAVA PROGRAMMING (22ITC31)

Submitted by

DEVISRI P(23EIR019)

DHAKSHINYA Y(23EIR020)

DHNRAJ T(23EIR021)



KONGU ENGINEERING COLLEGE

(Autonomous)

Perundurai, Erode – 638 060

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING



Transform Yourself

BONAFIDE CERTIFICATE

Name & Roll No.

DEVISRI P(23EIR019)

DHAKSHINYA Y(23EIR020)

DHNRAJ T(23EIR021)

Course Code : **22ITC31**

Course Name : **JAVA PROGRAMMING**

Semester : **III**

Certified that this is a bonafide record of work for application project done by the above students for **22ITC31-JAVA PROGRAMMING** during the academic year **2024-2025**.

Submitted for the Viva Voce Examination held on _____

Faculty In-Charge

Year In-charge

TABLE OF CONTENTS

CHAPTER No.	TITLE	PAGE No.
1.	Abstract	4
2	Problem Statement	4
3.	Methodology	5
4.	Implementation	6
5.	Results and Discussion	7
6.	Conclusion	7
7.	Sample Coding	8
8.	Output	13

Weather Forecasting System: A Menu-Driven Application for City Temperature Management

Abstract

The **Weather Forecasting System** is a Java-based application designed to manage and display temperature records for multiple cities. The system provides functionalities to add, update, delete, and view city temperature data, with temperature conversion between Celsius and Fahrenheit. Utilizing a menu-driven interface, the program demonstrates key object-oriented programming principles such as encapsulation and modularity. This report outlines the problem statement, methodology, implementation, and results achieved using the application.

Problem Statement

In modern weather applications, managing temperature records for various cities dynamically and efficiently is critical. There is a need for a lightweight system that enables the user to interactively add, update, or delete temperature records and display them in different formats. This project addresses this need by creating a simple console-based weather forecasting system to manage city-wise temperatures.

Methodology

1. **Requirements Analysis:**

- Identify operations required for city temperature management: Add, update, delete, and display.
- Include functionality for temperature conversion from Celsius to Fahrenheit.

2. **Design:**

- Create a modular system with separate classes for functionality (`WeatherForecastingSystem`) and user interaction (`Main` class).
- Use `ArrayList` for dynamic storage of city and temperature data.

3. **Implementation Steps:**

- Design methods for adding, updating, deleting, and displaying city temperatures.
- Implement a utility function for Celsius-to-Fahrenheit conversion.
- Develop a menu-driven console-based interface.

4. **Testing:**

- Test each functionality independently and validate the results for different input cases.

Implementation

Class Design

- **Weather Forecasting System:** Manages temperature data and provides operations like add, update, delete, and display.
- **Main Class:** Provides a menu-driven interface to interact with the user.

Key Features

1. Add a city and its temperature in Celsius.
2. Update the temperature of an existing city.
3. Delete a city from the records.
4. Insert a city and temperature at a specific position in the list.

5. Convert temperatures from Celsius to Fahrenheit.
6. Display records in both Celsius and Fahrenheit formats.

Results and Discussion

1. **Functionality Verification:**

- Successfully added, updated, and deleted city-temperature pairs dynamically.
- Displayed city temperatures accurately in both Celsius and Fahrenheit.

2. **Efficiency:**

- The use of `arraylist` allowed dynamic handling of records without predetermined limits.

3. **Limitations:**

- No data persistence: All records are lost when the program exits.
- Limited input validation: Users can input invalid data, causing errors.

4. **Future Enhancements:**

- Implement database or file-based storage for persistence.
- Improve error handling and validation for robust input processing.

Conclusion

The **Weather Forecasting System** successfully demonstrates the implementation of a dynamic temperature management application using Java. It showcases the application of object-oriented programming and a menu-driven interface for user interaction. While functional, the system can be further enhanced with persistent storage and additional features for real-world applications.

Sample coding

```
import java.util.ArrayList;
import java.util.Scanner;

class WeatherForecastingSystem { // Removed 'public' keyword
    private ArrayList<String> cities;
    private ArrayList<Double> temperatures;

    public WeatherForecastingSystem() {
        cities = new ArrayList<>();
        temperatures = new ArrayList<>();
    }

    // Method to convert Celsius to Fahrenheit
    public double convertToFahrenheit(double celsius) {
        return (celsius * 9/5) + 32;
    }

    // Method to add a new city and temperature
    public void addCity(String city, double temperature) {
        cities.add(city);
        temperatures.add(temperature);
    }

    // Method to update temperature for an existing city
    public void updateTemperature(String city, double newTemperature) {
        int index = cities.indexOf(city);
        if (index != -1) {
```

```
        temperatures.set(index, newTemperature);
        System.out.println("Temperature updated successfully.");
    } else {
        System.out.println("City not found.");
    }
}
```

// Method to delete a city

```
public void deleteCity(String city) {
    int index = cities.indexOf(city);
    if (index != -1) {
        cities.remove(index);
        temperatures.remove(index);
        System.out.println("City deleted successfully.");
    } else {
        System.out.println("City not found.");
    }
}
```

// Method to insert a city at a specific position

```
public void insertCityAtPosition(String city, double temperature, int position) {
    if (position >= 0 && position <= cities.size()) {
        cities.add(position, city);
        temperatures.add(position, temperature);
        System.out.println("City inserted successfully at position " + position);
    } else {
        System.out.println("Invalid position.");
    }
}
```



```

// Method to display the temperature for a given city
public void displayCityTemperature(String city) {
    int index = cities.indexOf(city);
    if (index != -1) {
        double temperatureCelsius = temperatures.get(index);
        double temperatureFahrenheit = convertToFahrenheit(temperatureCelsius);
        System.out.println("Temperature in " + city + " is " + temperatureCelsius + "°C
/ " + temperatureFahrenheit + "°F");
    } else {
        System.out.println("City not found.");
    }
}

// Method to display all cities and temperatures
public void displayAllRecords() {
    if (cities.isEmpty()) {
        System.out.println("No records available.");
        return;
    }
    System.out.println("City | Temperature (°C) | Temperature (°F)");
    for (int i = 0; i < cities.size(); i++) {
        double tempCelsius = temperatures.get(i);
        double tempFahrenheit = convertToFahrenheit(tempCelsius);
        System.out.println(cities.get(i) + " | " + tempCelsius + "°C | " + tempFahrenheit
+ "°F");
    }
}

```

```
public class Main {  
    public static void main(String[] args) {  
        WeatherForecastingSystem weatherSystem = new  
WeatherForecastingSystem();  
        Scanner scanner = new Scanner(System.in);  
  
        while (true) {  
            System.out.println("\nWeather Forecasting System");  
            System.out.println("1. Add City");  
            System.out.println("2. Update Temperature");  
            System.out.println("3. Delete City");  
            System.out.println("4. Insert City at Position");  
            System.out.println("5. Display Temperature of City");  
            System.out.println("6. Display All Records");  
            System.out.println("0. Exit");  
            System.out.print("Enter your choice: ");  
  
            int choice = scanner.nextInt();  
            scanner.nextLine(); // consume newline  
  
            switch (choice) {  
                case 1:  
                    System.out.print("Enter city name: ");  
                    String city = scanner.nextLine();  
                    System.out.print("Enter temperature (°C): ");  
                    double temperature = scanner.nextDouble();  
                    weatherSystem.addCity(city, temperature);  
                    break;
```

case 2:

```
System.out.print("Enter city name to update: ");  
city = scanner.nextLine();  
System.out.print("Enter new temperature (°C): ");  
double newTemperature = scanner.nextDouble();  
weatherSystem.updateTemperature(city, newTemperature);  
break;
```

case 3:

```
System.out.print("Enter city name to delete: ");  
city = scanner.nextLine();  
weatherSystem.deleteCity(city);  
break;
```

case 4:

```
System.out.print("Enter city name: ");  
city = scanner.nextLine();  
System.out.print("Enter temperature (°C): ");  
temperature = scanner.nextDouble();  
System.out.print("Enter position to insert: ");  
int position = scanner.nextInt();  
weatherSystem.insertCityAtPosition(city, temperature, position);  
break;
```

case 5:

```
System.out.print("Enter city name to display: ");  
city = scanner.nextLine();  
weatherSystem.displayCityTemperature(city);  
break;
```

case 6:

```
weatherSystem.displayAllRecords();  
break;
```

case 0:

```
System.out.println("Exiting...");
```

```
scanner.close();
```

```
return;
```

default:

```
System.out.println("Invalid choice. Please try again.");
```

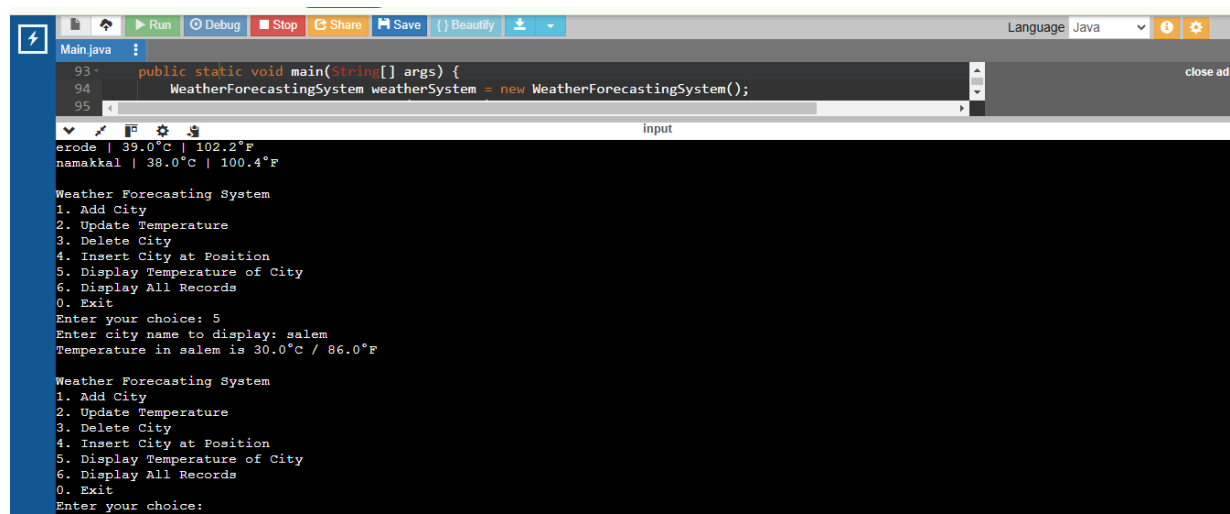
```
}
```

```
}
```

```
}
```

```
}
```

Output



The screenshot shows a Java IDE with a code editor and an output console. The code editor displays the following code:

```
93 public static void main(String[] args) {
94     WeatherForecastingSystem weatherSystem = new WeatherForecastingSystem();
95 }
```

The output console shows the following text:

```
erode | 39.0°C | 102.2°F
namakkal | 38.0°C | 100.4°F

Weather Forecasting System
1. Add City
2. Update Temperature
3. Delete City
4. Insert City at Position
5. Display Temperature of City
6. Display All Records
0. Exit
Enter your choice: 5
Enter city name to display: salem
Temperature in salem is 30.0°C / 86.0°F

Weather Forecasting System
1. Add City
2. Update Temperature
3. Delete City
4. Insert City at Position
5. Display Temperature of City
6. Display All Records
0. Exit
Enter your choice:
```

```
93 public static void main(String[] args) {
94     WeatherForecastingSystem weatherSystem = new WeatherForecastingSystem();
95 }
```

Weather Forecasting System

1. Add City
2. Update Temperature
3. Delete City
4. Insert City at Position
5. Display Temperature of City
6. Display All Records
0. Exit

Enter your choice: 6

City	Temperature (°C)	Temperature (°F)
saalem	30.0°C	86.0°F
erode	39.0°C	102.2°F
omalur	29.0°C	84.2°F

Weather Forecasting System

1. Add City
2. Update Temperature
3. Delete City
4. Insert City at Position
5. Display Temperature of City
6. Display All Records
0. Exit

Enter your choice:

Faculty Incharge
HOD

Academic Coordinator