Java OpenWeatherMap with Eclipse CLI

Contents

Time required: 60 minutes

- Comment each line of code as shown in the tutorials and other code examples.
- Follow all directions carefully and accurately.
- Think of the directions as minimum requirements.

www.OpenWeatherMap.org provides access to current weather data for any location on Earth including over 200,000 cities! They collect and process weather data from different sources such as global and local weather models, satellites, radars and vast network of weather stations. Data is available in JSON, XML, or HTML format.

Our application is going to call current weather data for one location.

Openweathermap.org API ID

The first step in using openweathermap.org is to get an API key.

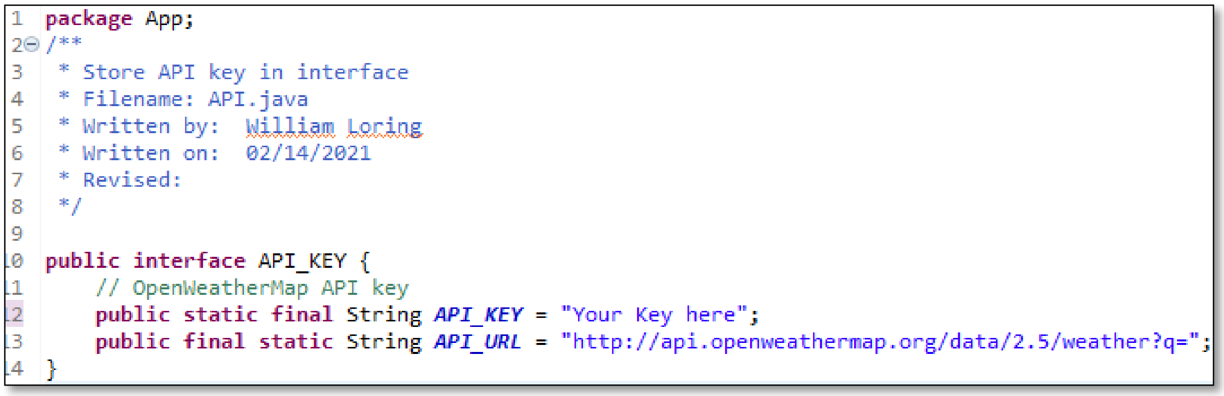
NOTE: You can skip this step if you already have an API key.

1. Go to www.openweathermap.org.
2. Create a free account.
3. You will be automatically setup with a free plan.

4. OpenWeatherMap will send you your API key in the confirmation email. You can also access your key from your account.

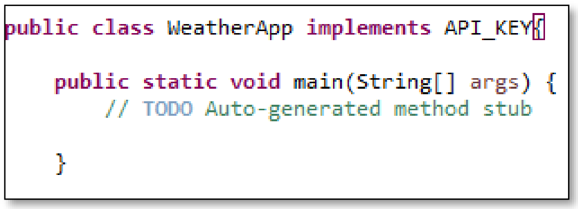
Create Application

1. In **Eclipse** → close the Welcome window.
2. Click **File** → **New** → **Project**. Go to **Java** → **Java Project**. Click **Next**.
3. Project Name: **OpenWeatherMapConsole**. Click **Next**.
4. Java Settings: Uncheck **Create module-info.java file** → Click **Finish.**
5. Open the **OpenWeatherMapConsole** project:
6. Right click the **src** folder, **New** → **Package**.
7. Name: **App**
8. Create another package under **src** named: **Models**
9. Right click the **App** package, **New** → **Class**.
10. Name: **API_KEY** Add the following code.



```java
package App;
/**
 * Store API key in interface
 * Filename: API.java
 * Written by:  William Loring
 * Written on:  02/14/2021
 * Revised:
 */

public interface API_KEY {
    // OpenWeatherMap API key
    public static final String API_KEY = "Your Key here";
    public final static String API_URL = "http://api.openweathermap.org/data/2.5/weather?q=";
}
```

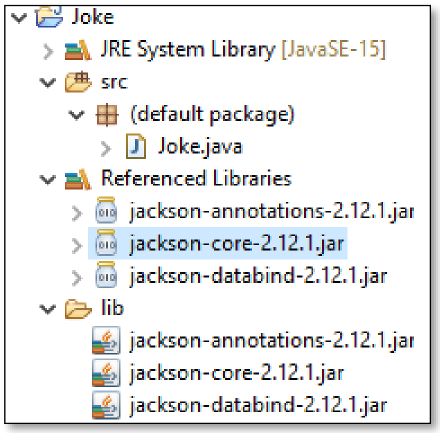
11. Right click the **App** package, **New** → **Class**.
12. Name: **WeatherApp**
    a. Which method stubs would you like to create? **public static void main(String[] args)**
13. Click **Finish**. The file opens automatically in **Source** View.
14. Add **implements API_KEY** to the Public Class WeatherApp.


```
public class WeatherApp implements API_KEY{

    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }
```

Jackson JSON Libraries

We need to add three libraries to parse out the JSON data into Java classes.

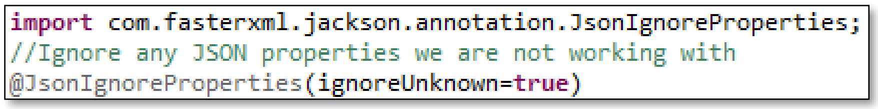
1. Go to https://jar-download.com/artifacts/com.fasterxml.jackson.core
2. Download **jackson-databind.jar**. This will be a zip file.
3. Extract the files from the zip file.
4. You should have the following 3 files. The numbers might change if there has been an update.
    a. jackson-annotations-2.12.2.jar
    b. jackson-core-2.12.2.jar
    c. jackson-databind-2.12.2.jar
5. Right Click **OpenWeatherMapConsole → New → Folder**
6. Name the new folder: **lib**
7. Copy and paste the three jar files into this folder in Eclipse.
8. Right Click each jar file **→ Build Path → Add to Build Path**
9. Your project folder should look like this.


```
Joke
  JRE System Library [JavaSE-15]
  src
    (default package)
      Joke.java
  Referenced Libraries
    jackson-annotations-2.12.1.jar
    jackson-core-2.12.1.jar
    jackson-databind-2.12.1.jar
  lib
    jackson-annotations-2.12.1.jar
    jackson-core-2.12.1.jar
    jackson-databind-2.12.1.jar
```

JSON to POJO

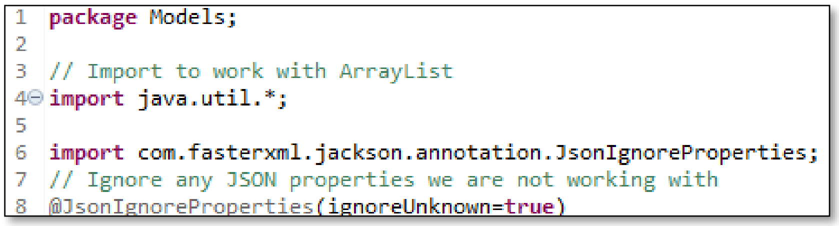
We need to create some classes to store the data retrieved from the OpenWeatherMap JSON.

1. Got to https://openweathermap.org/current
2. Go down the page until you find → **JSON Example of an API response**.
3. Copy the example JSON text.
4. Go to https://json2csharp.com
    a. Choose JSON to Java.
    b. Under Settings, Click Use Properties.
    c. Click Convert.
5. The **Result** will be several classes with fields, getters, and setters.
6. Click the **Copy** button. Paste the code into a text file for ease of use. We will create several classes from this code.
7. In the **OpenWeatherMapAPI** project in the **Models** package:
    a. Create a new Java public class for each of the classes.
    b. Paste the corresponding class code into each class.
    c. Remove the **@JsonProperty("")** from each class file. There will be one entry for each field. This will leave just the getters, setters, and fields.
    d. Add the following code to the top of each class.



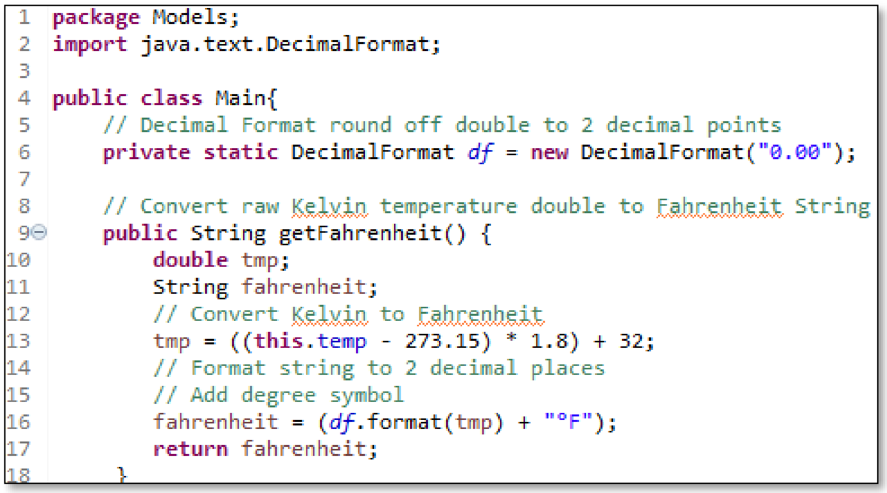
```java
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
//Ignore any JSON properties we are not working with
@JsonIgnoreProperties(ignoreUnknown=true)
```

8. **Root:** Add the following code.



```java
package Models;

// Import to work with ArrayList
import java.util.*;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
// Ignore any JSON properties we are not working with
@JsonIgnoreProperties(ignoreUnknown=true)
```

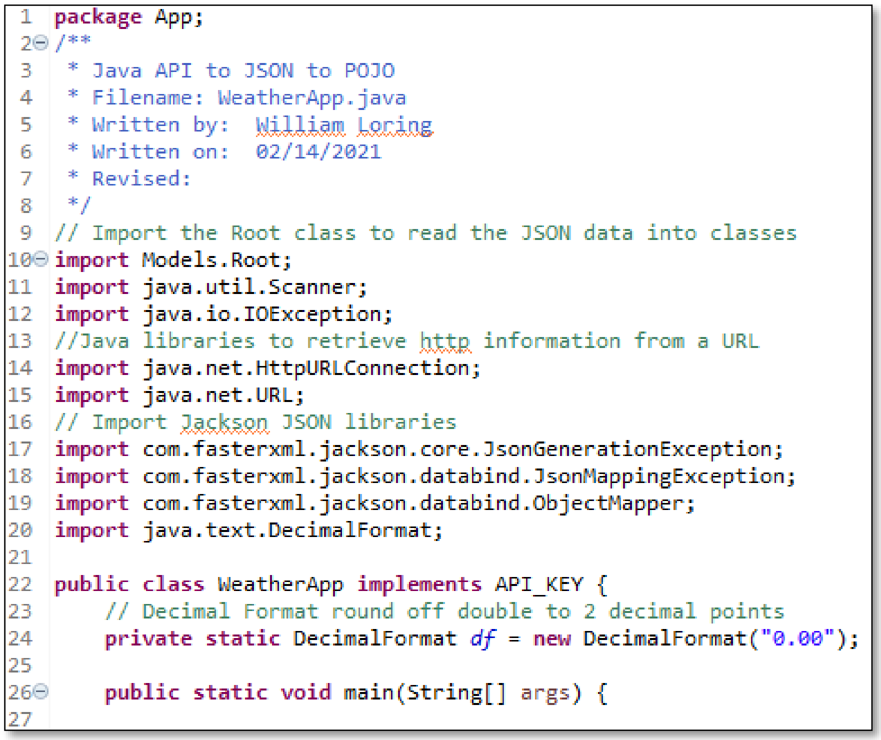
9. **Main**: Add the following code to convert the incoming Kelvin temperature to Fahrenheit temperature.



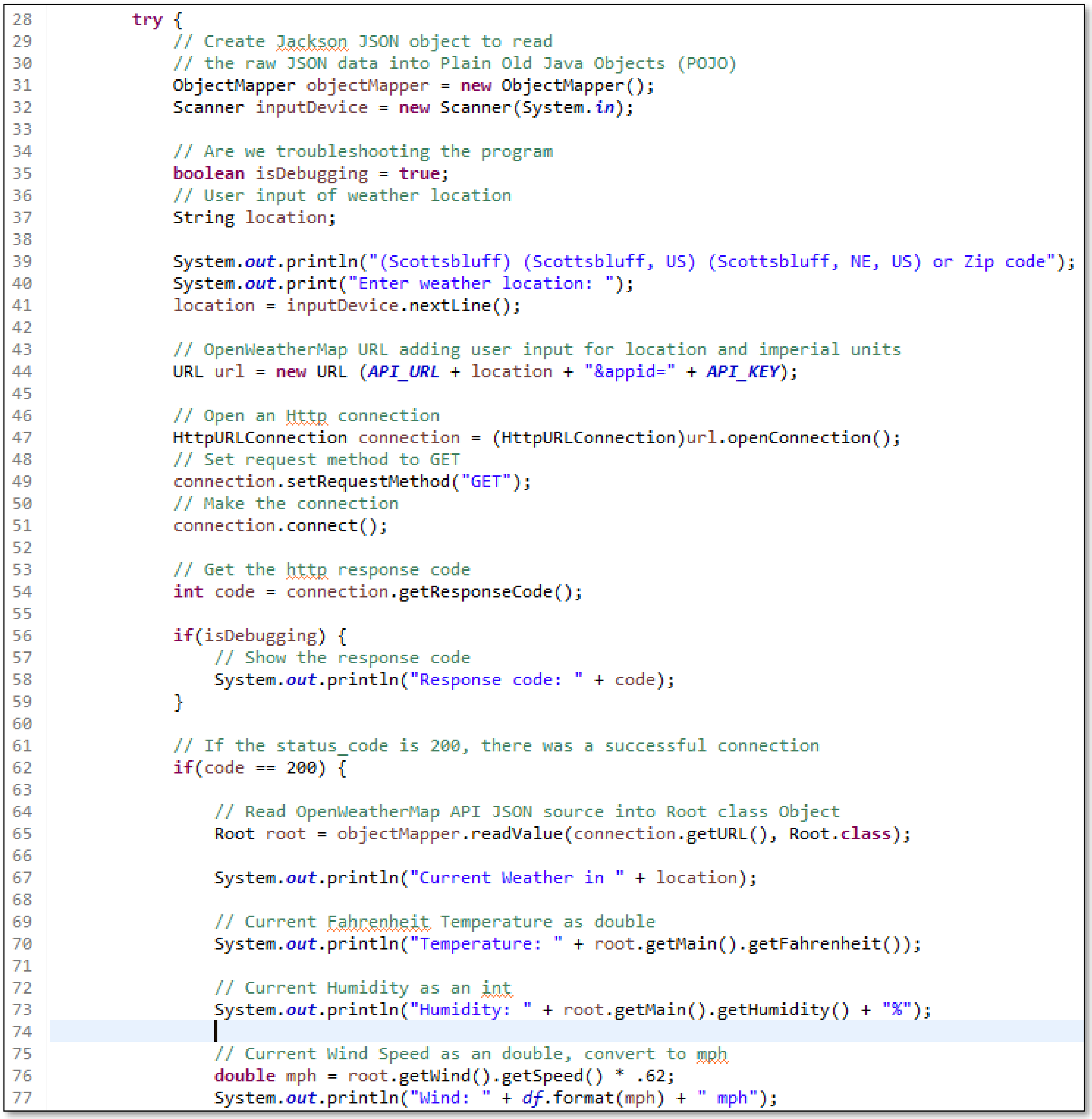
```java
package Models;
import java.text.DecimalFormat;

public class Main{
    // Decimal Format round off double to 2 decimal points
    private static DecimalFormat df = new DecimalFormat("0.00");

    // Convert raw Kelvin temperature double to Fahrenheit String
    public String getFahrenheit() {
        double tmp;
        String fahrenheit;
        // Convert Kelvin to Fahrenheit
        tmp = ((this.temp - 273.15) * 1.8) + 32;
        // Format string to 2 decimal places
        // Add degree symbol
        fahrenheit = (df.format(tmp) + "°F");
        return fahrenheit;
     }
```

WeatherApp Class

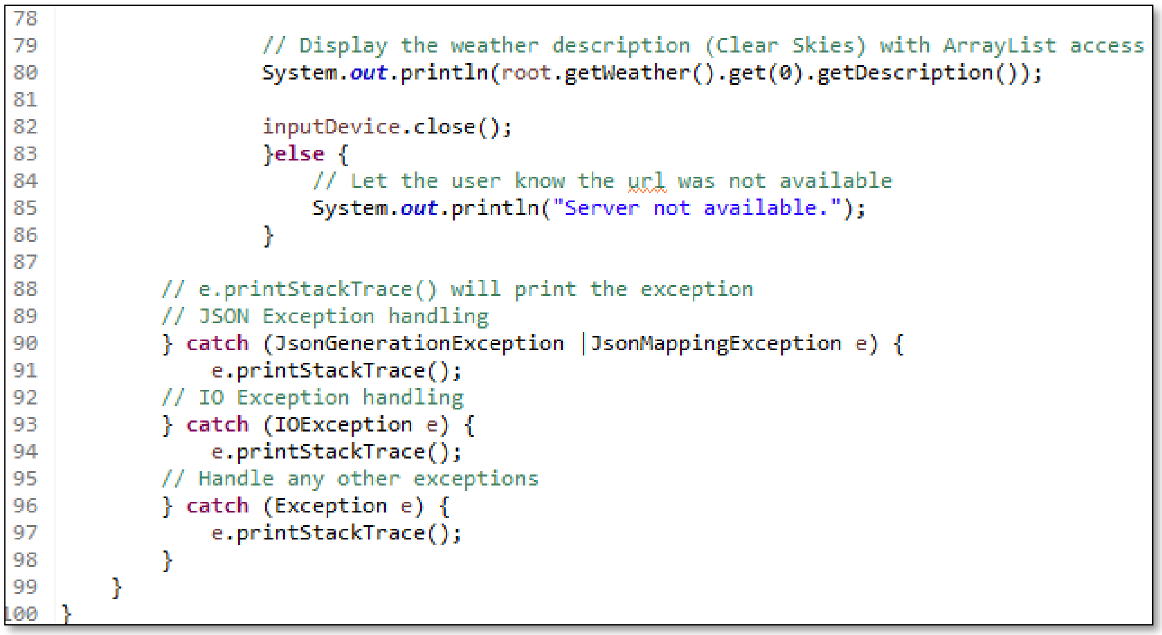
Add the following code to the WeatherApp class.



```java
package App;
/**
 * Java API to JSON to POJO
 * Filename: WeatherApp.java
 * Written by:  William Loring
 * Written on:  02/14/2021
 * Revised:
 */
// Import the Root class to read the JSON data into classes
import Models.Root;
import java.util.Scanner;
import java.io.IOException;
//Java libraries to retrieve http information from a URL
import java.net.HttpURLConnection;
import java.net.URL;
// Import Jackson JSON libraries
import com.fasterxml.jackson.core.JsonGenerationException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import java.text.DecimalFormat;

public class WeatherApp implements API_KEY {
    // Decimal Format round off double to 2 decimal points
    private static DecimalFormat df = new DecimalFormat("0.00");

    public static void main(String[] args) {

```



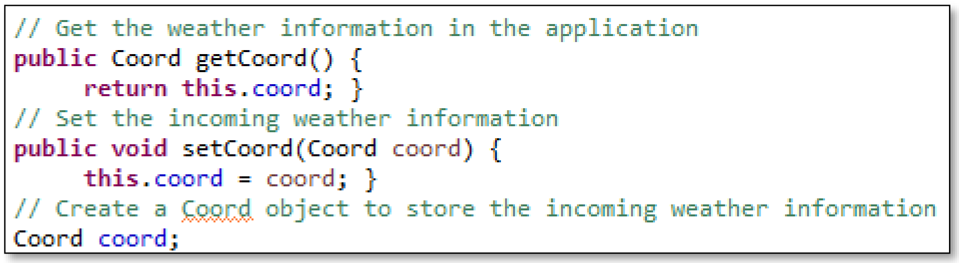
```java
        try {
            // Create Jackson JSON object to read
            // the raw JSON data into Plain Old Java Objects (POJO)
            ObjectMapper objectMapper = new ObjectMapper();
            Scanner inputDevice = new Scanner(System.in);

            // Are we troubleshooting the program
            boolean isDebugging = true;
            // User input of weather location
            String location;

            System.out.println("(Scottsbluff) (Scottsbluff, US) (Scottsbluff, NE, US) or Zip code");
            System.out.print("Enter weather location: ");
            location = inputDevice.nextLine();

            // OpenWeatherMap URL adding user input for location and imperial units
            URL url = new URL (API_URL + location + "&appid=" + API_KEY);

            // Open an Http connection
            HttpURLConnection connection = (HttpURLConnection)url.openConnection();
            // Set request method to GET
            connection.setRequestMethod("GET");
            // Make the connection
            connection.connect();

            // Get the http response code
            int code = connection.getResponseCode();

            if(isDebugging) {
                // Show the response code
                System.out.println("Response code: " + code);
            }

            // If the status_code is 200, there was a successful connection
            if(code == 200) {

                // Read OpenWeatherMap API JSON source into Root class Object
                Root root = objectMapper.readValue(connection.getURL(), Root.class);

                System.out.println("Current Weather in " + location);

                // Current Fahrenheit Temperature as double
                System.out.println("Temperature: " + root.getMain().getFahrenheit());

                // Current Humidity as an int
                System.out.println("Humidity: " + root.getMain().getHumidity() + "%");

                // Current Wind Speed as an double, convert to mph
                double mph = root.getWind().getSpeed() * .62;
                System.out.println("Wind: " + df.format(mph) + " mph");
```



```java

                // Display the weather description (Clear Skies) with ArrayList access
                System.out.println(root.getWeather().get(0).getDescription());

                inputDevice.close();
                }else {
                    // Let the user know the url was not available
                    System.out.println("Server not available.");
                }

        // e.printStackTrace() will print the exception
        // JSON Exception handling
        } catch (JsonGenerationException |JsonMappingException e) {
            e.printStackTrace();
        // IO Exception handling
        } catch (IOException e) {
            e.printStackTrace();
        // Handle any other exceptions
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

The above code does 3 things.

1. Creates a Jackson object to read JSON into POJO (Plain Old Java Objects).
2. Reads the converted JSON data into the Root class.
    a. The Root class creates objects from the other classes.
    b. The Root object sets the converted JSON weather information into those objects.
3. Displays the current weather at the location you type in.
    a. The application retrieves the weather by getting the properties of the objects.



```java
// Get the weather information in the application
public Coord getCoord() {
     return this.coord; }
// Set the incoming weather information
public void setCoord(Coord coord) {
     this.coord = coord; }
// Create a Coord object to store the incoming weather information
Coord coord;
```

This is an example of how the Root class creates the weather objects.

1) A Root object is created in the application.

2) The JSON data is read into the Root object.

```
// Read OpenWeatherMap API JSON source into Root class Object
Root root = objectMapper.readValue(connection.getURL(), Root.class);
```

3) The Root object (class) creates the other weather objects (classes) that we created earlier, including a Coord object.

4) The Root object sets incoming weather information into the Coord object using the setCoord method.

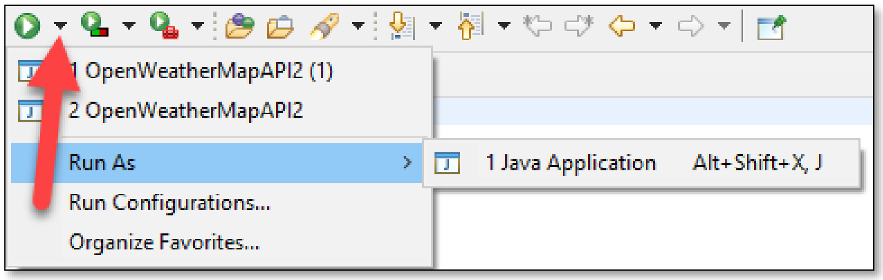
5) The application uses the objects getCoord to retrieve the weather information.

6) The process is the same for the other classes in the Models package.

Current Weather

Time to run your new application.

1. Click the project folder.
2. Click downward pointing arrow next to the green run button.
3. Click **Run As → Java Application**.



Example run as shown in the console at the bottom of Eclipse:

```
(Scottsbluff) (Scottsbluff, US) (Scottsbluff, NE, US) or Zip code
Enter weather location: 69361
Response code: 200
Current Weather in 69361
Temperature: 37.76°F
Humidity: 87%
Wind: 4.79 mph
Overcast Clouds
```

Final Steps

Add a minimum of 3 more weather items to your program.

Look up the units in the OpenWeatherMap.org API documentation. You may have to do some converting.

Assignment Submission

1. Zip up the OpenWeatherMapAPI folder under your Eclipse workspace.
2. Attach the zip file to the assignment in Blackboard.