Activity - Weather App using APIs

In this activity, we will call a RESTful API using Java in order to construct a rudimentary Weather program. This lesson will also teach you how to load and use an *external* library (i.e. a collection of custom Classes that someone wrote that doesn't come with the base Java language).

Setup. Download the `WeatherProgram` folder on the class Google Drive. Right click on the zip folder in your Downloads folder and select "Extract All...". Extract the folder into your Documents folder (e.g. c:\Users\[buid]\Documents). You should see the folder now in Documents and it should contain three files:

- `WeatherApp.java` - the partial java file you will complete over the course of this activity
- `api-key.txt` - an API key, we'll explain what this below
- `json.jar` - an external library for parsing JSON in Java

An *API key* is a secret unique identifier used to authenticate a user before replying to a API call. In order to prevent a server from being overloaded, a *rate limit* may be placed on users (e.g. can only make 1000 a minute). The API may even be placed behind a paywall. The API key is how the server identifies who is making the request so that it can apply a limit or paywall. `api-key.txt` is my private API key for the API we will be using today. It is placed in a separate file because we never want to *hard-code* our secret key, lest we accidentally leak it when sharing the code.

Problem 1. Complete the `loadAPIKey()` method in `WeatherApp.java` by reading from `api-key.txt` file. **Observe:** Part of the method signature says `throws IOException`. This means that we do not need to handle the IOException ourselves in the method body, even though it is a checked Exception. Any sucker that calls `loadAPIKey()` will have to handle it however...

Problem 2. In the `main` method, assign the API Key to `apiKey` by calling `loadAPIKey()`. You will need to handle the exception: in the case of an error, tell the user that something has gone wrong loading the api-key.txt file and halt the program.

We will be using the `HTTPClient` and `HTTPRequest` classes in Java.net package in order to communicate with the API and receive the server's response.
Today we are using Weather Api to get the current weather. Requests made to this API look like:

```
GET http://api.weatherapi.com/v1/current.json?key=APIKEY&q=QUERY
```

Where we replace APIKEY with our API key, and QUERY with the location we want the weather of. We can give the City name, coordinates, or Zipcode. The Weather API will also accept our API key has a header under the name "key". We will implement it this way so you can try adding a header to the request.

Problem 3. Finish the `makeRequest` method, which constructs our HTTPRequest object by adding the apiKey to the header under the name "key". Constructing `HTTPClient` and `HTTPRequest` objects requires using a special type of class called a *builder*. The purpose of a Builder class is to make constructing complicated types easier. Builder classes have specific methods to set specific fields, and then a method `build()` that constructs the object. The code for constructing the HTTPClient object has already been written, and you can use it as a reference for how a Builder works.

Hint: You only need to use the `.header()` method. If you uncomment it and hover over it you can read the documentation.

Do not end the .header() call with a semicolon. We are actually currying all the Builder methods together, placing each new call on it's own line for readability. This is one advantage to semicolons, we can have one "line of code" split up over multiple lines without semicolons and it'll be the same for the compiler as one really long line.

If you've done everything correctly running the program should print the response to the terminal. Unfortunately, the response is a JSON file and we will need to parse it. Rather than using Scanner, we will use an *external library* org.json. In order to use it we must do some set up so VS Code knows the library exists.

Setup.

1. Uncomment `import org.json.JSONObject;` at the top of your WeatherApp.java file. VS Code should complain that the import cannot be resolved.
2. Press Ctrl + Shift + P (i.e press Ctrl, Shift, and P all at once) to bring the `Command Palette` in VS Code. A search bar should open at the top of your screen. You can also alternatively select the search bar at the top of your screen and select "Show and Run commands"
3. Search for and select `Java: Open Project Settings`. This will open a new tab in VS Code.
4. Navigate to the Libraries tab and then click Add Library. Select json.jar.
5. json.jar should appear under Libraries. Click `Apply Settings` and return to your WeatherApp.java file. You may now see a folder called .vscode containing `settings.json`. You can ignore this.

VS Code should no longer complain about your import statement.

Problem 4. Use the `JSONObject` class to parse the JSON response (`response.body()`). Pass the response to the JSONObject constructor and then use JSONObject methods to get the `double` stored under `temp_f`. **Hint:** The response JSON has two fields: `location` and `current`, both of which are themselves JSON objects. You will want to use `getJSONObject()` and `getDouble()`.

If you have done everything correctly, your program when run will print the current Boston tempature to the terminal. To run your program from the Windows command prompt you will need to the following commands:

```
javac -cp ".;json.jar" WeatherApp.java
java -cp ".;json.jar" WeatherApp
```

This tells the compiler that you need to use the `json.jar` file stored in the same folder. Otherwise it will not be imported and you will get an error.

Problem 5. Challenge Modify WeatherApp to change the city from Boston to another city of your choice.

For an extra challenge, have the program take in a command line argument for the City and print out "The temperature in [CITY] is currently: "

You can also try printing other weather data besides temperature or to get forecasted weather data rather than current data. You can find the Weather API Documentation online at weatherapi.com