Java Dog Facts API Tutorial

Contents

Java Dog Facts API Tutorial 1
What is an API? 1
Why Use a Web API? 3
Public API's 3
Dog Facts API 3
What is JSON? 3
Tutorial 1: Setup the Java Project 4
Tutorial 2: JSON to POJO 6
Tutorial 3: Java App 7
Time for a Dog Fact 11
Assignment Submission 12

This tutorial was inspired by a student screencast submission.

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.
- Please read all the directions before beginning the assignment.

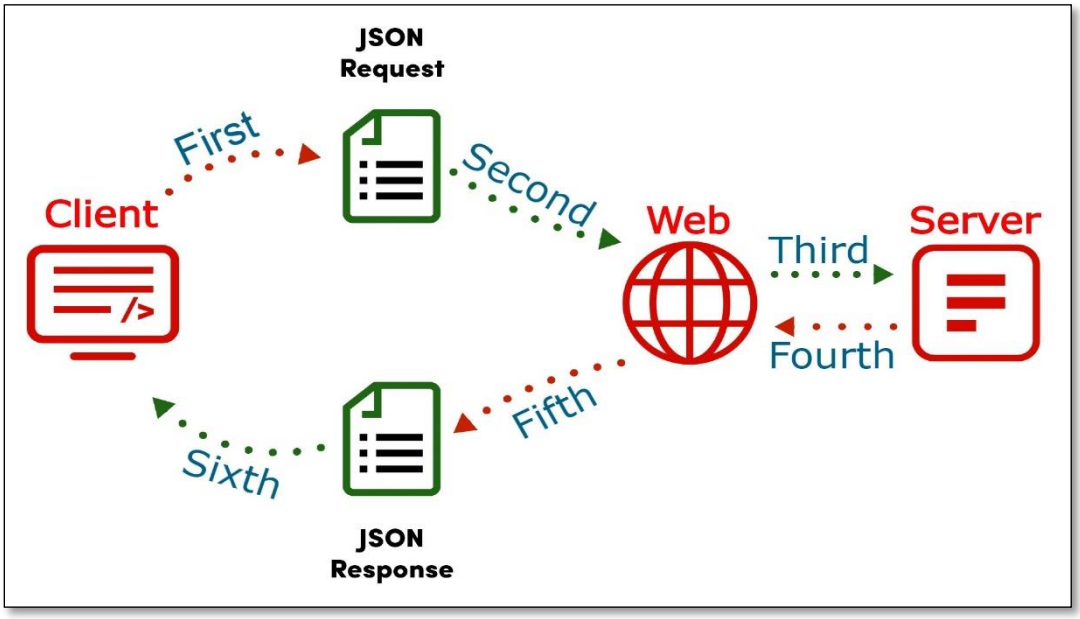
What is an API?

What is an API? This is what Wikipedia says.

"An **application programming interface** ('API') is a computing interface that defines interactions between multiple software intermediaries. It defines the kinds of calls or requests that can be made, how to make them, the data formats that should be used, the conventions to follow, etc. It can also provide extension mechanisms so that users can extend existing functionality in various ways and to varying degrees. An API can be entirely custom, specific to a component, or designed based on an industry-standard to ensure

interoperability. Through information hiding, APIs enable modular programming, allowing users to use the interface independently of the implementation."

An API is a software intermediary that allows two applications to talk to each other. In other words, an API is the messenger that delivers your request to the provider that you're requesting it from and then delivers the response back to you.



Web API's are hosted on webservers. Instead of a web browser asking for a webpage (like how most interactions with the internet is done today) your program asks for data. This data is usually returned to your program in a JSON format. JSON stands for JavaScript Object Notation and is a lightweight text-based data-interchange format.

To get data to our program we make a request to a web server. The webserver then replies with our desired data and a status code. Sometimes an API key is needed to be sent along with the request to successfully return data.

There are many ways to call information from an API. One of the most common are **get** requests. You use a get request each time you browse to a web site.

For example, to request data in a Python script using the extra functionality gained from the request library the statement below would be used.

```
response = requests.get('http://api_example.com/example.json')
```

This data object can be named anything, response was used as the identifier for this example. This API doesn't exist. This is the format they tend to take.

Data from the web can tell us the exact weather situation of any place on the globe, the location of faces on an image we provide or the number of astronauts currently floating far, far out in space. Any information freely accessible on the World Wide Web is literally within our grasp from within our programs.

Why Use a Web API?

You could use static data which you have downloaded or created yourself. Data used this way with Java would work perfectly fine. Here are some examples of when APIs are more useful than static data sets.

- When the data is changing quickly, and you need the data to be accurate and up to date. Planes are currently in the air or the location of the international space station, the data is constantly changing.
- When only a small piece of a much larger data set is desired.
- When you are taking advantage of already completed, repeated computations. Sources such Facebook, Spotify and Google already have a ton of data, which have undergone serious calculations, which you can access easily with an API request.

In cases like the ones above, a Web API is the right solution. Using an API will save time and effort over doing all the computation ourselves or constantly refreshing our desired data.

Public API's

There are hundreds of public API's available. This website contains a list of some of them.

https://github.com/public-apis/public-apis

Dog Facts API

Who doesn't want to know random facts about dogs? We do! Let's create a Java program to help us with our dog education. Before we do that, we have a few new concepts to go through.

What is JSON?

The Dog Facts API is a free https-based Web API which provides a JSON response.

Raw API data is typically in a text file in JSON or XML format. JSON is the most popular format for web API's.

JavaScript Object Notation (JSON) is an open standard text-based file and data interchange format. It uses human-readable text to store and transmit data objects consisting of attribute–value pairs and array data types.

Go to https://dog-api.kinduff.com/api/facts to see some raw JSON data. This view is using Firefox.


```
{
  "facts": [
    "Highly trainable dog breeds like Golden Retrievers, Labrador Retrievers, German Shepherds and Collies are more kid-friendly than some other breeds."
  ],
  "success": true
}
```


Tutorial 1: Setup the Java Project

This will be a Visual Studio Code project in Java.

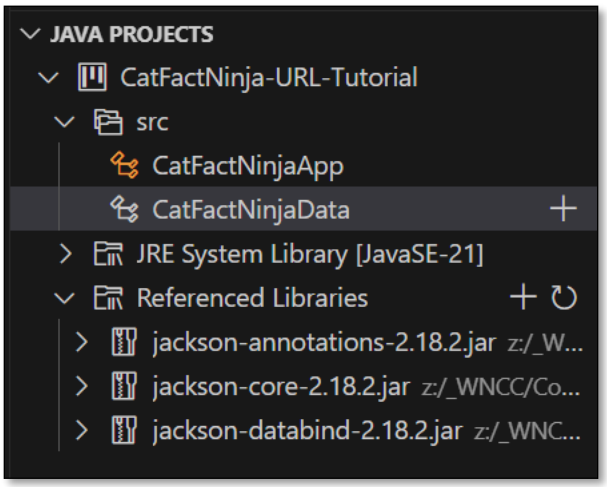
We will be retrieving data from the web in JSON format. Java doesn't have any native JSON parsing or handy dandy requests library like Python. We will use the Jackson JSON parser.

1. Create a folder to hold your project: **DogFacts**
2. Open the **DogFacts** folder with Visual Studio Code.
3. Create two folders in your **DogFacts** program folder.
    a. **lib**
    b. **src**

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

4. Download these three files.
    a. jackson-annotations-2.18.2.jar
    b. jackson-core-2.18.2.jar
    c. jackson-databind-2.18.2.jar
5. Copy and paste the three jackson jar files into the **lib** folder.

6. In the **src** folder, create two files.
    a. **DogFactsApp.java**
    b. **DogFactsData.java**
7. Open **DogFactsApp.java**
8. You should see the following at the bottom left if you open **Java Projects → Referenced Libraries**



Tutorial 2: JSON to POJO

JSON stands for JavaScript Object Notation, a lightweight data-interchange format. To convert JSON (JavaScript Object Notation) to POJO (Plain Old Java Object) in Java, follow these steps:

1. Create a Java class: Define a Java class that mirrors the structure of the JSON data. Include fields that match the JSON object's properties.
2. Use a library: Libraries like Jackson, Gson, or org.json can parse JSON to Java objects automatically by mapping JSON keys to Java object fields.

Go to https://dog-api.kinduff.com/api/facts to see some raw JSON data. This view is using Firefox.

```
{
  "facts": [
    "Highly trainable dog breeds like Golden Retrievers, Labrador
Retrievers, German Shepherds and Collies are more kid-friendly than
some other breeds."
  ],
  "success": true
}
```

There are many JSON to POJO converters on the web. We are going to use https://jsonformatter.org/json-to-java to convert the JSON to a Plain Old Java Object.

This is the result after pasting the sample JSON into the above web site.

```
package io.codebeautify;

public class Welcome5 {
    private String[] facts;
    private boolean success;

    public String[] getFacts() { return facts; }
    public void setFacts(String[] value) { this.facts = value; }

    public boolean getSuccess() { return success; }
    public void setSuccess(boolean value) { this.success = value; }
}
```

To convert the JSON data from the API into a Plain Old Java Object, we create a class modeled after this conversion. We make a couple of changes.

1. Remove package io.codebeautify;
2. Change the class name.

Tutorial 3: Java App

Enter the following code into **DogFactsData.java**

```java
/**
 * Filename: DogFactsData.java
 * Written by:
 * Written on:
 * Revised:
 * Description:
 * Convert data fetched from API endpoint
 */

public class DogFactsData {

    // Create a String array to hold the facts
    private String[] facts;

    // Even though we don't use this field,
    // it is required to be present to handle the JSON response
    private boolean success;

    public String[] getFacts() {
        return facts;
    }

    public void setFacts(String[] value) {
        this.facts = value;
    }

    public boolean getSuccess() {
        return success;
    }

    public void setSuccess(boolean value) {
        this.success = value;
    }

    // Override system object to String method with
    // User-friendly String representation of object
    @Override
    public String toString() {
        String output = String.format("\nDog Fact: %s \n", this.facts[0]);
        return output;
    }
}
```

We added a toString() @Override to more easily print the object. Though we don't use the length attribute, we do have to handle it in the class.

The main program uses the **DogFactsData** class to convert the incoming JSON to an object with attributes, getters and setters. We can access API data attributes like any other object.

Add the following code to the **DogFactsApp.java** file.

```java
/**
* Filename: DogFactsApp.java
* Written by:
* Written on:
* Revised:
* Description: Get Dog Facts from API endpoint
*/

// Handle any Input Output exceptions
import java.io.IOException;
import java.net.URI;
// Create a URL object
import java.net.URL;
import java.util.Scanner;

// Import Jackson JSON libraries
import com.fasterxml.jackson.core.JsonGenerationException;
import com.fasterxml.jackson.databind.JsonMappingException;
import com.fasterxml.jackson.databind.ObjectMapper;

public class DogFactsApp {
    static Scanner keyboard = new Scanner(System.in);
    // API endpoint URL
    private final static String API = "https://dog-api.kinduff.com/api/facts";

    // Create DogData array object
    private static DogFactsData fact;
    // Create a menu character
    private static char menu = 'y';

    Run | Debug
    public static void main(String[] args) {
        while (menu != 'n') {
            fetchData();

            // Display a Fact at the console
            System.out.println(fact.toString());

            System.out.print("Another Dog Fact (y, n): ");

            // Get the next character from the keyboard
            menu = keyboard.next().charAt(0);
        }
        System.out.println("Thanks for using Dog Facts.");
        fetchData();
        // Display a Dog Fact at the console
        System.out.println(fact.toString());
    }
}
```

```
49      // Get data from the API endpoint
50      public static void fetchData() {
51          try {
52              // Create URI object
53              URI uri = new URI(API);
54
55              // Create URL object from URI
56              URL url = uri.toURL();
57
58              // Create Jackson JSON object to read the raw JSON data
59              // into POJO (Plain Old Java Objects)
60              ObjectMapper objectMapper = new ObjectMapper();
61
62              // Read JSON source into DogData Object
63              // The incoming json is an array []
64              fact = objectMapper.readValue(url, DogFactsData.class);
65
66              // e.printStackTrace() will print the exception
67              // JSON Exception handling
68          } catch (JsonGenerationException | JsonMappingException e) {
69              e.printStackTrace();
70              // Input Output Exception handling
71          } catch (IOException e) {
72              e.printStackTrace();
73              // Handle any other exceptions
74          } catch (Exception e) {
75              e.printStackTrace();
76          }
77      }
78  }
```

The above code does 3 things.

1. Creates a Jackson object to read JSON into POJO (Plain Old Java Objects).
2. Reads the JSON into a **DogFactsData** object.
3. Displays a Dog Fact.

Time for a Dog Fact

Time to run your new application.

Example run:

```
Dog Fact: Dogs curl up to keep themselves warm and protect vital organs.

Another Dog Fact (y, n):
```

Assignment Submission

1. Insert a screenshot of a successful run of the program.
2. Zip up the DogFactsNinja folder.
3. Attach the zip file to the assignment in Blackboard.