Web Services GET tutorial (Java)

In this tutorial, you'll work on a command-line application that displays Tech Elevator locations. The command-line application is partially complete. You'll write the remaining functionality.

Once the application is running, you'll need to call a web API to both get a list of locations and the details for a single location.

Step one: Start the server

Before you start, you need to ensure that the web API is up and running. You need to change directories into the `./server/` folder.

Next, from the command line, run the command `npm install` to install any dependencies. You won't need to do this on any subsequent run.

While still in the command line, run the command `npm start` to start the server. If there aren't any errors, you'll see the following, which means that you've successfully set up your web API:

```
Resources
http://localhost:3000/locations
```

```
 \{^_^}/ hi!

  Loading ./locations.json
  Done

  Resources
  http://localhost:3000/locations

  Home
  http://localhost:3000

  Type s + enter at any time to create a snapshot of the database
  Watching...
```

> You can stop the server, or any other process that you've started from the console, by using the keyboard shortcut `ctrl + c`.

Step Two: Explore the API

Before moving on to the next step, explore the web API using Postman. You can access the following endpoints:

- GET: http://localhost:3000/locations
- GET: http://localhost:3000/locations/{id}

Step Three: Review the starting code

Data model

There's a class in `/src/main/java/com/techelevator/model/Location.java` that represents the data model for a location object.

Provided code

You'll find the `main()` method in `/src/main/java/com/techelevator/locations/App.java`.

Also in the `App.java` file, you'll find two classes:

```
ConsoleService consoleService = new ConsoleService();
LocationService locationService = new
LocationService("http://localhost:3000/locations/");
```

The `ConsoleService` class handles printing to the console and retrieving user input. The `LocationService` class handles interacting with the web API to retrieve data.

Your code

You'll place most of your code in `App.java` in the `main()` method or in `src/main/java/com/techelevator/services/LocationService`.

Step Four: Run the console application

If you run the application, you'll see the following output in the console:

```
Welcome to Tech Elevator Locations. Please make a selection:
1: List Tech Elevator Locations
2: Show Tech Elevator Location Details
0: Exit

Please choose an option:
```

The `printMainMenu()` method in the `ConsoleService` prints the menu and retrieves the user's input. When the user makes a selection, it's stored in the `menuSelection` variable. You'll need to handle the menu selections for listing all locations and getting the details of a single location:

```
while (menuSelection != 0) {
  menuSelection = consoleService.printMainMenu();
  if (menuSelection == 1) {
    // TODO: list all locations

  } else if (menuSelection == 2) {
    // TODO: get one location
```

```
    } else if (menuSelection == 0) {
      // exit
      consoleService.exit();
    } else {
      // defensive programming: anything else is not valid
      System.out.println("Invalid Selection");
    }
  }
```

Step Five: List all locations

If the user selects `1`, you need to list all locations returned from the web API. Open `LocationService` and find the method `getAll()`:

```
public Location[] getAll() {
    return null;
}
```

Before you make a call to the web API to get a list of locations, you need to know the URL of the service.

Back in `App.java`, you passed in the base URL for your server:

```
private static final String API_URL = "http://localhost:3000/locations";
```

This is stored in your `LocationService` using the variable `BASE_URL`:

```
private String BASE_URL;
```

Next, you'll create a new instance of the RestTemplate. This is the class that you'll use to perform a `GET` request to the web API.

You could create this in the `getAll()` method, but you'll need this elsewhere, so it's better to create an instance variable at the class level:

```
private String BASE_URL;
private RestTemplate restTemplate = new RestTemplate();
```

You can use the `RestTemplate`'s `getForObject()` method to perform a `GET` request to the web API. This method takes the URL of the API and the response type:

```

```

```
public Location[] getAll() {
    return restTemplate.getForObject(BASE_URL, Location[].class);
}
```

Next, return to `App.java` and locate the part of program that executes when the user selects `1`. You can call the method you completed in the `LocationService` class to get an array of locations.

The `ConsoleService` class has a method that handles printing an array of locations to the console:

```
if (menuSelection == 1) {
    // list all locations
    Location[] locations = locationService.getAll();
    consoleService.printLocations(locations);
}
```

If you run the application, you'll see this:

```
Welcome to Tech Elevator Locations. Please make a selection:
1: List Tech Elevator Locations
2: Show Tech Elevator Location Details
0: Exit

Please choose an option: 1
-----------------------------------------
Locations
-----------------------------------------
1: Tech Elevator Cleveland
2: Tech Elevator Columbus
3: Tech Elevator Cincinnati
4: Tech Elevator Pittsburgh
5: Tech Elevator Detroit
6: Tech Elevator Philadelphia
```

Step Six: Get Location data

Back in `App.java`, when the user selects menu option `2`, you need to perform these tasks:

1. Display a list of locations to the user and ask them to select one.
2. Read in the ID of the location.
3. Pass the ID to the `LocationService`'s `getOne()` method.

You can use the `promptForLocation()` method in the `ConsoleService` class which takes an array of locations and returns the user's selection. Once you have the location ID, you can pass that to the `getOne()` method in the `LocationService` class to get the details for a single location:

```
} else if (menuSelection == 2) {
    // get one location
    Location[] locations = locationService.getAll();
    int locationId = consoleService.promptForLocation(locations, "View");
    consoleService.printLocation(locationService.getOne(locationId));
}
```

Next, open the `LocationService` class and locate the `getOne()` method which currently returns null. You can use the `getForObject()` method that you previously used, but with two differences.

First, you need to call your web API with the `BASE_URL` and the ID appended. Next, the response type is a `Location.class`:

```
public Location getOne(int id) {
    return restTemplate.getForObject(BASE_URL + id, Location.class);
}
```

> If you had a chance to test the API in Postman, you know that calling `/locations/1` returns the location data for Tech Elevator Cleveland.

If you run the application, you'll see this:

```
Welcome to Tech Elevator Locations. Please make a selection:
1: List Tech Elevator Locations
2: Show Tech Elevator Location Details
0: Exit

Please choose an option: 2
-------------------------------------------
Locations
-------------------------------------------
0. Exit
1: Tech Elevator Cleveland
2: Tech Elevator Columbus
3: Tech Elevator Cincinnati
4: Tech Elevator Pittsburgh
5: Tech Elevator Detroit
6: Tech Elevator Philadelphia

Please enter a location id to View: 1

-------------------------------------------
Location Details
-------------------------------------------
Id: 1
Name: Tech Elevator Cleveland
Address: 7100 Euclid Ave #140
City: Cleveland
```

```
State: OH
Zip: 44103
```

Summary

In this tutorial, you learned:

- How to make an HTTP GET request using Postman and inspect the result
- How to make an HTTP GET request to a RESTful web service using Java process the response
- How to convert a single JSON object into a Java Object
- How to convert an array of JSON objects into an array of Java Objects