Web Services GET exercise (Java)

In this exercise, you'll work on a command-line application that displays online auction info. Most of the command-line application is provided. You'll write the code that calls the API.

You'll add web API calls using `RestTemplate` to retrieve a list of auctions, details for a single auction, and filter the list of auctions by title and current bid.

Step One: Start the server

Before starting, make sure the web API is up and running. Open the command line and navigate to the `./server/` folder in this exercise.

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

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

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

Loading data-generation.js
Done

Resources
http://localhost:3000/auctions

Home
http://localhost:3000

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

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/auctions
- GET: http://localhost:3000/auctions/{id} (try a number between 1 and 7 in place of `{id}`)
- GET: http://localhost:3000/auctions?title_like=<*value*> (try a string like *watch* in place of `<value>`)
- GET: http://localhost:3000/auctions?currentBid_lte=<*value*> (try a number like 150 in place of `<value>`)

Step Three: Review the starting code

Data model

There's a class provided in `model/Auction.java` that represents the data model for an auction object. If you've looked at the JSON results from the API, the properties for the class should look familiar.

Provided code

In `App.java` and `services/ConsoleService.java`, you'll find methods that print information to the console and retrieve input from the user. Notice that each of the four methods that you'll implement are called there as well.

Your code

`services/AuctionService.java` contains four methods where you'll add code to call the API methods:

- `getAllAuctions`
- `getAuction`
- `getAuctionsMatchingTitle`
- `getAuctionsAtOrBelowPrice`

Step Four: Complete `AuctionService`

There are two variables declared for you at the top of the class that you can use throughout your exercise:

```
public static final String API_BASE_URL = "http://localhost:3000/auctions/";
private RestTemplate restTemplate = new RestTemplate();
```

List all auctions

In the `getAllAuctions` method, find the comment `// call api here`. Add code here to:

- Use the `RestTemplate` to request all auctions and save them into an array of `Auction`s.
- Replace the current return statement to return the array of `Auction`s.

Once you've done this, run the unit tests. After the test `listAllAuctions` passes, you can run the application. If you select option 1 on the menu, you'll see the ID, title, and current bid for each auction.

List details for a specific auction

In the `getAuction` method, find the `// call api here` comment. Add code here to:

- Use `RestTemplate` to request a specific auction by ID.
- Return the single `Auction`.

Once you've done this, run the unit tests. After the test `listDetailsForAuction` passes, you can run the application. If you select option 2 on the menu, and enter an ID of one of the auctions, you'll see the full details for that auction.

Find auctions with a specified term in the title

In the `getAuctionsMatchingTitle` method, find the `// call api here` comment. Add code here to:

- Use `RestTemplate` to request all auctions which match the `title` parameter and save them into an array of `Auction`s.
- The API URL requires a query string with the key named `title_like` and the value of the `title` parameter. This requests the server to search for auctions that have a title containing the value in the `title` parameter.
- Replace the current return statement to return the array of `Auction`s.

Once you've done this, run the unit tests. After the test `findAuctionsSearchTitle` passes, run the application. If you select option 3 on the menu, and enter a string, like `watch`, you'll see the ID, title, and current bid for each auction that matches.

Find auctions below a specified price

The final method, `getAuctionsAtOrBelowPrice`, also uses a query string. The parameter key is `currentBid_lte` and has the value of the `price` parameter. This requests the server to search for auctions that have a `currentBid` that's **L**ess **T**han or **E**qual to the value you supply.

Find the `// call api here` comment and add code here to:

- Use `RestTemplate` to request all auctions which are less than or equal to the `price` parameter and save them into an array of `Auction`s.
- The API URL requires a query string with the key named `currentBid_lte` and the value of the `price` parameter. This requests the server to search for auctions that have a price less than or equal to the value in the `price` parameter.
- Replace the current return statement to return the array of `Auction`s.

Once you've done this, run the unit tests. After the test `findAuctionsSearchPrice()` passes, run the application. If you select option 4 on the menu and enter a number, like `150`, you'll see the ID, title, and current bid for each auction that matches.

Since the value is a `double`, you can enter a decimal value, too. Try entering `125.25`, and then `125.20`, and observe the differences between the two result sets. The "Mad-dog Sneakers" don't appear in the second list because the current bid for them is `125.23`.