Work on project. Stage 3/5: Useful connections

Project: Music Advisor

Useful connections

Hard 2 hours **409** users solved this stage. Latest completion was **14 days ago**.

§1. Description

As you can see from the previous stage, we need to find a way to get a response code from the URL in the user's browser. By default, Java doesn't have browser capabilities but has opportunities to create your own simple HTTP server.

§2. What is HTTP

HTTP means Hypertext Transfer Protocol, which is a stateless data transfer protocol based on client-server technology, where messaging occurs according to the request-response scheme. The main manipulation object is the resource indexed by the URI. Every time you need to make an HTTP request, follow this structure:

A request line:

```
method_name URI HTTP/version
```

Example:

```
GET /some/uri HTTP/1.1
```

There are a lot of HTTP methods, but in this project, we will consider two of them:

`GET` requests a representation of the specified resource. They only retrieve data and have no other effect.

The `POST` method requests that the server accepts the entity enclosed in the request as a new subordinate of the web resource identified by the URI.

The response looks like this:
A response line:

```
HTTP/version status_code message
```

Example:

```
HTTP/1.1 200 OK
```

§3. Creating a server in java

`com.sun.net.httpserver` package included in Java SE contains classes. The main class is `HttpServer`. This class implements a simple HTTP server. To create an instance of this class, you need to use static factory method "create" and bind it to IP and port.

```
HttpServer server = HttpServer.create();
server.bind(new InetSocketAddress(8080), 0);
```

These lines will create an `http` server that will listen for incoming TCP connections from clients on 8080 port. Another main concept is context. When an HTTP request is received, the appropriate `HttpContext` (and handler) is located by finding the context whose path is the longest matching prefix of the request URI's path. To create the context, you should use the method `createContext` and pass a string of URI path and handler that implements the `HttpHandler` interface.

Code snippets

```
server.createContext("/",
    new HttpHandler() {
        public void handle(HttpExchange exchange) throws IOException {
            String hello = "hello, world";
            exchange.sendResponseHeaders(200, hello.length());
            exchange.getResponseBody().write(hello.getBytes());
            exchange.getResponseBody().close();
        }
    }
);
```

The lines above will create a context to which all requests will be redirected by the server, and the context handler will always return "hello world".

To start the server, add the line `server.start()`; use the command `server.stop(1)` to shut down the server. `1` here is the maximum delay in seconds to wait until all handlers have finished.

If you try to run it, you can open your browser at `localhost:8080`, and you will see this message.

In this stage, you will receive a query parameter with the authorization code from the Spotify page. It looks like `http:localhost:8080?code=123`. To get the query inside the `HttpExchange` handler, you can use the following line:

```
String query = exchange.getRequestURI().getQuery();
```

§4. Making HTTP requests in Java

JDK 11 provides a few classes in the `java.net.http` package to make HTTP requests: read more about them at openjdk.java.net. First, you should create an `HttpClient` instance:

```
HttpClient client = HttpClient.newBuilder().build();
```

Then you should setup the http request by creating an `HttpRequest` instance. It supports the Builder pattern, so you should just call `HttpRequest.newBuilder()`, then add some methods to setup your request and then call a `build()` method to create it. Here is an example how to create a simple GET request:

```
HttpRequest request = HttpRequest.newBuilder()
        .uri(URI.create("http://localhost:8080"))
        .GET()
        .build();
```

To send the request, use the `client` instance:

```
HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());

System.out.println(response.body());
```

This code will print "hello, world" if it connects to the server from the section above.

In this stage, you will create a POST request to get the Spotify access token. The main difference between GET and POST requests is that POST may contain a body with some data. It may be a file, json, xml, or other format. You should set the type of data with "Content-type" header. This example shows how to send a POST request with data in the format `application/x-www-form-urlencoded`:

```
HttpRequest request = HttpRequest.newBuilder()
                .header("Content-Type", "application/x-www-form-urlencoded")
                .uri(URI.create("http://localhost:8080"))
                .POST(HttpRequest.BodyPublishers.ofString("login=admin&password=admin"))
                .build();
```

You should put the body data inside the `POST()` method using the `HttpRequest.BodyPublishers.ofString` method. `x-www-form-urlencoded` data consists of `key=value` pairs, separated by the `&` symbol.

Similarly, you can send json data by setting "Content-type" as `application/json` and passing json inside the `ofString()` method.

Useful hint: you can use reqbin to test POST and GET requests to API without Java. ReqBin is an online API testing tool where you can send API requests directly from the browser and check the response. You can specify the URL of an API endpoint, select an appropriate HTTP method and enter the data you want to send in the Content tab to send the request. Then you can check the returned status code, its response time and content to see if the API functions as expected.

§5. Objectives

Using the Spotify authorization guide and the information given here (you need the section *Authorization Code Flow*), improve your program by adding real authorization on Spotify.

1. Choose any free port on your machine (for example, 8080), and add the `http://localhost:your_port` to the whitelist of `redirect_uri` in your application settings on the Spotify site (Dashboard -> your app -> edit settings -> redirect URIs).
   Note that you should use the `http` protocol for localhost, not `https`, like in the Spotify example.)
2. On the `auth` command, before printing the auth link (from the previous stage), you should start an HTTP server that will listen for the incoming requests. When the user confirms or rejects the authorization, the server should return the following text to the browser:
   - *"Got the code. Return back to your program."* if the query contains the authorization code.
   - *"Authorization code not found. Try again."* otherwise.

   This code is bound to each user who has a Spotify account and uses your app. Actually, you should ask this code once for each new user and save it somewhere.
3. After the code is received, the server must shut down and you should get `access_token` by making a POST request on `https://accounts.spotify.com/api/token` with parameters described in the guide, and then print the response body.

Also, in this stage, you should read the Spotify access server point from the command line argument. Server path should be passed to the program using `-access` argument. If this argument is not set, you should use a default argument, `https://accounts.spotify.com`. Make sure you replace constants to this argument value everywhere!

§6. Example

Below is an output example of the described program. Try to output all cases like in the example.

The greater-than symbol followed by a space (`> `) represents the user input. Note that it's not part of the input.

```
> new
Please, provide access for application.
> auth
use this link to request the access code:
https://accounts.spotify.com/authorize?client_id=a19ee7dbfda443b2a8150c9101bfd645&redirect_uri=http:
//localhost:8080&response_type=code
waiting for code...
code received
making http request for access_token...
response:
{"access_token":"BQBSZ0CA3KR0cf0LxmiNK_E87ZqnkJKDD89VOWAZ9f0QXJcsCiHtl5Om-
EVhkIfwt1AZs5WeXgfEF69e4JxL3YX6IIW9zl9WegTmgLkb4xLXWwhryty488CLoL2SM9VIY6HaHgxYxdmRFGWSzrgH3dEqcvPoLpd26
D8Y","token_type":"Bearer","expires_in":3600,"refresh_token":"AQCSmdQsvsvpneadsdq1brfKlbEWleTE3nprDwPbZg
NSge5dVe_svYBG-
RG-_PxIGxVvA7gSnehFJjDRAczLDbbdWPjW1yUq2gtKbbNrCQVAH5ZBtY8wAYskmOIW7zn3IEiBzg","scope":""}
---SUCCESS---
> new
---NEW RELEASES---
Mountains [Sia, Diplo, Labrinth]
Runaway [Lil Peep]
The Greatest Show [Panic! At The Disco]
All Out Life [Slipknot]
```

```
> exit
---GOODBYE!---
```

Report a typo

HINT by **Konstantin Proskurnya**

In server.createContext() you store code in static variable and then use while cycle with Thread.sleep. And after that server.stop().

```
while (CODE.equals("")) {
        Thread.sleep(10);
}
server.stop(10);
```

Was this hint helpful? Yes No Report

Write a program

Code Editor IDE

CONNECTION STATUS

- IDE is responding IntelliJ IDEA 2022.2.1
- EduTools plugin is responding 2022.7-2022.2-291

Solve in IDE Synchronizing IDE may take a while

Look up solution (100)

Comments (123) Hints (24) Useful links (5) Solutions (66) Show discussion