Overview

In this part, you will implement a web server. It is a web server that does not do very much, but you will probably find it useful for the rest of this assignment. The server accepts an incoming TCP connection, reads data from it until the client closes the connection, and returns ("echoes") an HTTP response back with the data in it.

This may seem a bit odd. But the idea is that if you use your web browser to navigate to this server, what you will see in your browser window is the complete HTTP request that your browser sent.

So, let's say that you are running this server on port 8888 on your computer. In the navigation field of your browser, type in the URL "http://localhost:8888". What you then should see in your browser is something like this:

```
GET / HTTP/1.1

Host: localhost:8888

Upgrade-Insecure-Requests: 1

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_2) AppleWebKit/604
.4.7 (KHTML, like Gecko) Version/11.0.2 Safari/604.4.7

Accept-Language: en-us

Accept-Encoding: gzip, deflate

Connection: keep-alive
```

So this is what the browser sent to the server.

Task

Your job is to implement a class called HTTPEcho. It's "main" method implements the server. It takes one argument: the port number. So if you want your server to run on port 8888, you would start it in the following way:

```
$ java HTTPEcho 8888
```

You will be provided with a file with a template for the HTTPEcho class. There is not much content in it though, but your task is to fill in the missing Java code *in this file*.

Since HTTPEcho is a TCP server, you should implement it in the same way as other servers: it should run in an infinite loop, and when one client has been served, the server should be prepared to take care of the next. Your server does not need to serve more than one client at a time, though. It does not need to be multithreaded, in other words (that's for later).

Instructions and Tips

- Use the ServerSocket (Länkar till en externa sida.) class to create the socket for the server.
- The format for marking end-of-line varies between different systems. Sometimes it is marked with the ASCII symbol "carriage return", sometimes the symbol "line feed", and sometimes both. HTTP marks end of line with carriage returned, followed by line feed. This is probably not what your operating system uses to mark end of line. Instead, you must use the HTTP end-of-line sequence specify in the HTTP strings that you generate from your program. In Java, the syntax for carriage return and line feed in a string is "\r\n". So make sure to write this sequence marks the end of each line.
- A web browser understands and can present a plethora of different data formats (that's what the browser is telling the server through the "Accept" header line that you can see in the example above). So what format should you use? Luckily, the default format for HTTP is plain text, "text/plain". As you can tell by the name, this format is just plain text, without any formatting. So if you don't specify anything else, the web client will assume that your response is plain text, and present it as such in the browser window. (If you want, you can format it more nicely with HTML code, but that is not required. But if you do that, you should specify that the content format is HTML – with the header line "Content-Type: text/html".)
- An HTTP response should also specify the character encoding that is used. The default is UTF-8, which is a superset of ASCII, so here you can just use the default as well. In other words, you don't need to specify this either.
- It is tricky to get everything right at once. Follow a sound software engineering approach and develop your code step by step. After each step, design and run one or more tests to verify that your code works.
- A suitably first step can be to generate a static HTTP response. In other words, ignore the data that the client is sending. Instead, return an HTTP response that always has the same data (like "hello there"). In this way, you can verify that you can generate a correct HTTP response that the browser recognises.

Simplification

It is convenient to use a browser for testing. However, a browser normally does not display an object until it has received the entire object. Since you are most likely generating a simplistic HTTP response, the browser will not be able to determine where the object ends. The result is that you will not see anything in your browser window (except for a spinning symbol perhaps, or a status line saying that the browser is waiting, etc.). The browser has to close to connection before the browser will display anything.

Therefore we introduce the following extension to the definition of the HTTPEcho server, which is supposed to make it easier for you to use a browser for testing:

When the server detects an empty line in the input from the client (the browser), it means that the server has received all data.

Then the server can, after having returned the data as an HTTP response to the client, close the connection. In other words, in your code you can check for an empty line and use that as a condition to terminate the loop. And fortunately, since HTTP already marks the end of the header with an empty line, this means that the server will immediately return the HTTP header for you!

Testing

Your server needs to reply with a valid HTTP response, otherwise the web browser will not recognise it. To debug your HTTP response, you may want to use some other tool instead of a web browser to connect to the server. Use netcat ("nc") or telnet, for instance.

Resources

For this task, you will be provided with the following:

- HTTPEcho.java – Skeleton declaration for the HTTPEcho class.

The file are available in a zip archive called "task2.zip". In this zip archive, there is a directory called "task2" with the file in it. So, when you unzip the archive, the "task2" directory will be created, and in this directory you will find the Java file. *This is important, because you are expected to submit your solutions in a zip archive with exactly the same structure!*

You can find the zip archive here ladda ner.