In this task, you will take the HTTPAsk server you did in Task 3, and turn it into a *concurrent* server. The server you did for Task 3 deals with one client at a time (most likely). A concurrent server can handle multiple clients in parallel.

What you will learn here is to:

- Use Java threading to implement a concurrent server that can handle many clients at the same time.

Task

The description here is almost the same as for Task 3: You should implement a class called ConcHTTPAsk ("Conc" because it is concurrent). 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 ConcHTTPAsk 8888
```

The difference between ConcHTTPAsk and HTTPAsk from Task 3 is that as soon as a client contacts your ConcHTTPAsk server, the client will be served immediately. It does not have to wait for ConcHTTPAsk to finish serving the current client.

Instructions and Tips

There are many different ways of implementing servers that can handle multiple clients in parallel. The approach we will use here is to use Java's support for *multithreading*. In Java, a thread represents a line of execution. A Java program can consist of many threads executing at the same time. What you should do here is to let the ConcHTTPAsk server *create one thread for each client*. In other words, when a new client contacts the server (the server returns from calling the *accept* method on its welcoming socket), the server creates a new thread, and this thread will serve the client.

So how do we create threads in Java? There are basically two ways: See Defining and Starting a Thread in the Java Tutorials. We recommend that you use the first method (Provide a Runnable object), by creating a class that implements the Runnable interface. Creating a thread involves the following steps:

1. Define a class that implements the Runnable interface. (Let's call this class MyRunnable.)
2. Create a MyRunnable object
3. Create a Thread object, using the MyRunnable object from step 2 as parameter
4. Take the Thread object from step 3, and call its *start* method
5. This will, in turn, call the *run* method in the class that you defined in step 1.
    - The run method is where there thread will do its work.
    - This is where you should implement the code that corresponds to the HTTPAsk server from Task 3

This may seem complicated at first, but this simple example on StackOverflow is easy to follow. The example also shows how you can pass a parameter to a thread. That is something that you need to do: each thread should serve a specific client, on a separate connection socket, so the connection socket needs to be an argument to the thread.

Resources

For this task, you will not be provided with any additional resources. You should be able to take your code from Task 3, and continue working on this.

Submission

Your submission should be a zip file named "task4.zip".

We use automatic tools for the grading. If your submission does not follow this format, the tools will not recognise your submission, and it cannot be graded. Before you upload your submission, make sure that it follows the format requirements. You can do that by repeating what the submissions tools do, by running the following commands *at the command line*:

```
$ unzip task4.zip
$ cd task4
$ javac ConcHTTPAsk.java
$ java ConcHTTPAsk <port number>
```

where <port number> represents a port number (and not the string "<port number>"!).

Submit by uploading your "task4.zip" file below. You can submit as many times as you like before the due date.

Evaluation

The evaluation consists of making sure that your new ConcHTTPAsk can deal with multiple clients at the same time. Test ConcHTTPAsk by connecting to a server that takes long time to complete. While you are doing that, use another client to connect to a server that responds quickly. This should give an immediate response.

Netcat ("nc") is a great tool for this. By running netcat in server mode (or listen mode, "-l" flag), you can control exactly when and how this server responds.

The first part of the evaluation is to verify that your submission has the correct format. If it does not have the correct format, no further evaluations are made, and the result will be "fail".

As part of the evaluation of your submission, a number of tests are conducted to check the functionality of your submission. The following tests are included in the evaluation:

- Send a request to connect to a daytime server, and verify that the output is correct.
- Send requests to connect to two daytime servers, one fast and one slow, and verify that the output is correct for both requests.