Code Assignment 04 - Programming a Multi-threaded Web Server

In this TD, we are going to build our own web server that can respond to the requests generated from a HTTP client. We will begin with a single-thread server, and then extend it to a multithread server based on the concurrency model that we saw in the previous weeks.

Exercise 1 : single thread web server

Let's first write a single thread web server in class **`Xserver`**.
The server will be launched via command :

```
java Xserver serverPort
```

which tells the server to listen to the specified *serverPort*. If you find it useful to add other settings, document this clearly and have anyway a default mode.

then you can query this server from a web browser running on the same machine, querying an URL of the form

```
http://localhost:<serverPort>/...
```

you should also use your own Xurl program and then try to connect to the server of one of your classmates. **Interoperability** is the objective.

A couple of hints might not go amiss, to help with programming:

- To start a TCP server in Java, you must create a `ServerSocket` object.

Preferably use the constructor which allows to specify the size of the connection queue, as we want to bind to a specific port number and as using a very short connection queue is more instructive. Any exception catched at this point will be reported on `System.err` and must cause the program to terminate.

- Once the server socket is operating, the program enters a big endless loop consisting in listening on the port, until a connection is accepted, and then passing the connection socket to a method

```
static void handleConnection(Socket socket)
```

that you have to write in Xserver too.

Note that a server must survive on any error, so the loop body and method `handleConnection` itself should not throw any exception. Particularly, any `IOException` must be caught and reported.

- Then you have to implement in `handleConnection` some part of the HTTP protocol for the server side, as specified below (for a short description of the http protocol, go back to Assignment 01

The server-side protocol is restricted to answer to a single GET request and then close the connection.

Your server must first check for a well formed GET line specifying 1.1 HTTP protocol.

If the queried URL is

```
http://localhost:<serverPort>/
```

then reply with a simple welcome message (don't waste time in something beautiful, we're not in a graphic arts course) -- of course, you need to construct the correct header as you already learned from the first class. A detailed format is **here**.

Else, if the path part of the URL is not a / only then it is supposed to be the relative path to a file in the running directory of the server. Don't check now if the file exists.
Your server must then check for a well formed *Host* line, but don't take care to match on the hostname, so it is made more flexible to test from various clients.

If one of these two lines is wrong, in above sense, answer with *bad request* and close immediately the connection. Else, scan and display (on `System.out`) the whole header, but without parsing the content.

Then the server answers by sending the requested file or the *not found* message. You can suppose that the queried file is pure text. If needed, you can download the test pages that you used in TD02 **here**.
You can suppose that the received URL does not contain a fragment identifier (a trailing part of the form `#something`) which should not be sent to the server, nor any query parameter (a trailing part starting with a question mark). You can also suppose that the path has a single component (no / inside) else you can simply send back a *not found* message.
Handling multi-components path, that is querying a file in a subdirectory, is left as an extension.

When a content is send, the header of the answer must provide both the `Content-Length:` and the`Connection: close` specifications, so it is made compatible with most of clients.

Exercise 2 : multi thread web server

Using the thread-pool pattern, implements a multi threaded web server. You must ensure enough responsiveness so that accepted connections will be served correctly.
The server will be launched via command :

```
java Xserver serverPort nThreads
```

Again, document clearly any additional parameters you want.