Extra Credit: Socket Programming
Implement an HTTP server!

Due: November 25th, 11:59 pm

Note: this is an individual assignment

Your task for this problem is to implement a basic HTTP server in Java. You can optionally add the features described in order to earn more points. Essentially, earn as many as you wish. The features add onto each other, so you should completely them sequentially.

Many Java HTTP server implementations already exist. You are essentially going to create one of these, so you may not use these (e.g. Spring, Jersey, Tomcat),

Here are the Java APIs you are allowed to use, don't use others without permission:
(if you have questions about what is allowed, ask!)
https://docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.html
https://docs.oracle.com/javase/8/docs/api/java/net/Socket.html
https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html
https://docs.oracle.com/javase/8/docs/api/java/io/File.html
https://docs.oracle.com/javase/8/docs/api/java/util/Date.html
https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html

Obviously, the standard API is fine, like String, ArrayList, HashMap, etc. The idea is just to NOT use the built-in HTTP implementation.

Another source of necessary reference material would be the RFC 2616, defining the HTTP 1.1 protocol. You do not need to read this entire document, only use it as a reference for the needed information.
https://www.ietf.org/rfc/rfc2616.txt

Basic server (1 point)

Start by creating a basic server using the ServerSocket API linked above. Your server will first create a new ServerSocket based on a port supplied via a constant value (you can use 8080). Accept a new connection, which will return a Socket. The Socket is able to provide you with an InputStream and an OutputStream so that you can both receive and send data.

After you are done responding to the client, you can simply close the streams and close the socket. Then, accept a new connection for the next request, and repeat the process (use a loop).

Recall the general format of HTTP request messages: ('\r\n' is the literal characters carriage return followed by new line)

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.utexas.edu\r\n
User-Agent: Firefox/3.6.10\r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Connection: keep-alive\r\n
\r\n
```

Also recall the general format of HTTP response messages:

```
HTTP/1.1 200 OK\r\n
Date: Sun, 26 Sep 2010 20:09:20 GMT\r\n
Server: Apache/2.0.52 (CentOS)\r\n
Last-Modified: Tue, 30 Oct 2007 17:00:02 GMT\r\n
ETag: "17dc6-a5c-bf716880"\r\n
Accept-Ranges: bytes\r\n
Content-Length: 2652\r\n
Keep-Alive: timeout=10, max=100\r\n
Connection: Keep-Alive\r\n
Content-Type: text/html;
charset=ISO-8859-1\r\n
\r\n
data data data data data ...
```

Your server will need to support 3 different response codes:

- `200 OK`
- `400 Bad Request`

- `404 Not Found`

You will need to parse the HTTP request message from the client, ensuring validity along the way. Return a code of 400 if the request is incorrectly formatted/not parseable. Ensure clients are using HTTP/1.1.

Your basic server only needs to implement the GET method. When you receive a request, you will return the data of the file requested by the given path, assuming that the current working directory is the "root" of the server ('/). If the file exists, you would use code 200. Otherwise, you would use code 404 and return no data.

For example, if the user requests /meme.txt, you should open the file meme.txt and return the bytes from the file, as a part of an HTTP response message.

At this point, you do not need to interpret any of the HTTP request headers, so just ignore them for this part, but assume the client will send some.

The HTTP response headers you should send back are:
- `Date` (current time of the server when request was made)
- `Server` (choose your own server name!)
- `Content-Length` (the number of bytes of the jdata you are returning, ignoring the HTTP headers / response line)

Virtual hosts / Host header (1 point)

Recall the HTTP Host header:

```
Host: www.cs.utexas.edu
```

Assume that our server wants to potentially host multiple sites. Also assume that all client requests for this feature need to supply this header.

If a request is made with `Host: meme.com,` you will only look for files to return from the directory `./meme.com/`

If a request is made with `Host: cs356.org,` you will only look for files to return from the directory `./cs356.org/`

You should create a file called `hosts.txt` that your server uses. On startup, the server will create a directory for each of the given hosts, if it doesn't already exist.

As the same as before, if a given file doesn't exist, simply return status code 404 with no data.

HTTP POST (1 point)

With HTTP GET, the client makes a simple request without providing the server with any additional data. With HTTP POST, data is provided to the server.

Recall the general format of HTTP POST request messages:

```
POST /newFile.txt HTTP/1.1\r\n
Host: www.cs.utexas.edu\r\n
User-Agent: Firefox/3.6.10\r\n
\r\n
data data data data ...
```

With our HTTP implementation, when a client makes an HTTP POST at a given URL, create a new file filled with the data provided by the client. If the targeted file already exists, overwrite the existing data with the provided data.

To know how much data is being uploaded by the client, the client will now send the `Content-Length` header to know the number of bytes to expect.

So, if a client does a POST to `/newFile.txt` with some data, the client should then be able to make a GET at that same URL and get back the data from the new file.

Caching (1 point)

Now, assume that HTTP clients (only for GET requests) will be supplying an additional header:

```
If-Modified-Since: Tue, 27 Aug 2019 20:49:28 -0700
```

If the file has NOT changed since the supplied timestamp, then simply return status code `304 Not Modified` without any of the file data. Otherwise, return the current version of the file with a normal status code (200, assuming it exists).

How do you know the modification timestamp of a File? Java's File object has a lastModified() method.

So, with this new feature, we could get the same file twice. The second request would return status code 304, as the client already has the data cached. Then, after performing a POST to overwrite the file, we would finally request the file again, and be able to get new data back, as the file's modification time has changed.

This might be helpful for this step:

```
public static final SimpleDateFormat DATE_FORMAT = new
SimpleDateFormat("EEE, dd MMM yyyy HH:mm:ss zzz");
```

Multi-threaded server (1 point)

Your current server implementation is only able to handle one client at a time. This is not very scalable to large application servers, handling thousands of requests a second.

Now, after accepting a connection from a client, giving us a Socket, we want to handle this client in a new Thread. The main thread's job will be to accept connections from clients, and create a new thread for each Socket. Then, each individual Thread will handle a client. The API documentation provides example code on how to create and start Threads.

*** Some optional parts to this feature:*

However, feel free to implement some sort of timeout, if feasible (this is what real servers do). For example, if there has been no activity on a thread/socket for 15 seconds, close it and stop the thread. This may be too complicated, feel free to ignore this part.

With this feature, many more clients could theoretically all request the same file simultaneously. You might want to make a global in-memory cache (just a HashMap) so you do not have to continually read the same file from disk (which is slow).

Testing script

I will be providing a testing script for your implementations, so that you can easily figure out what is / isn't working. It should test each of these features (sequentially), and give you output on what is working.

Submission / final notes

This assignment is purposefully open-ended, it is extra credit and allows you the freedom to construct your own creative implementation. This assignment will absolutely be challenging, as any extra credit should be, and we encourage you to ask questions and get help when needed.

Obviously, you can search for various web resources, as this is part of the learning process, but please do not just copy-and-paste extended code snippets from StackOverflow or elsewhere. Most important, while working on the assignment, if hosting on a Git server, please use a private repository.

As a part of your submission, please include a script `start.sh` that both compiles and runs your Java server. Plan to turn in your code as an archive file (.zip or .tar.gz).