HTTP Server Theory of Operation

A web server serves HTML documents and other resources to browser requests.

The browser (client) and web server uses TCP connection for communication. The default port that the client connects to is port 80.

Request from browser to server

Assume you type the following on the browser's address bar

```
http://localhost:3000/index.html
```

The browser will open a TCP connection to the web server's port at 3000 on localhost

The browser then issues the following request message to the server

```
GET<space>/index.html<space>HTTP/1.1
```

The first term `GET` is the method's name. In this case the browser is requesting a resource.

The second term `/index.html` is the name of the resource that the browser is requesting. The resource's name can change depending on what resource the browser is requesting.

The third term (`HTTP/1.1`) is the HTTP protocol's version number; this can be ignored.

The `<space>` is to show that there is a single space between all the terms.

There are other lines following this first request line; for this assessment, we will ignore all those and focus only on the first line (request line).

Response from server to browser

The server examines the request line and responds to the client's request.

Assume that the resource (`/index.html` in this case) that the browser is requesting exist, the HTTP server will respond with the following

```
HTTP/1.1<space>200<space>OK\r\n
\r\n
<contents of index.html in bytes>
```

Server closes connection after writing all the bytes

The server will first send a response line. The line consists of 3 terms separated by a single space (see above).

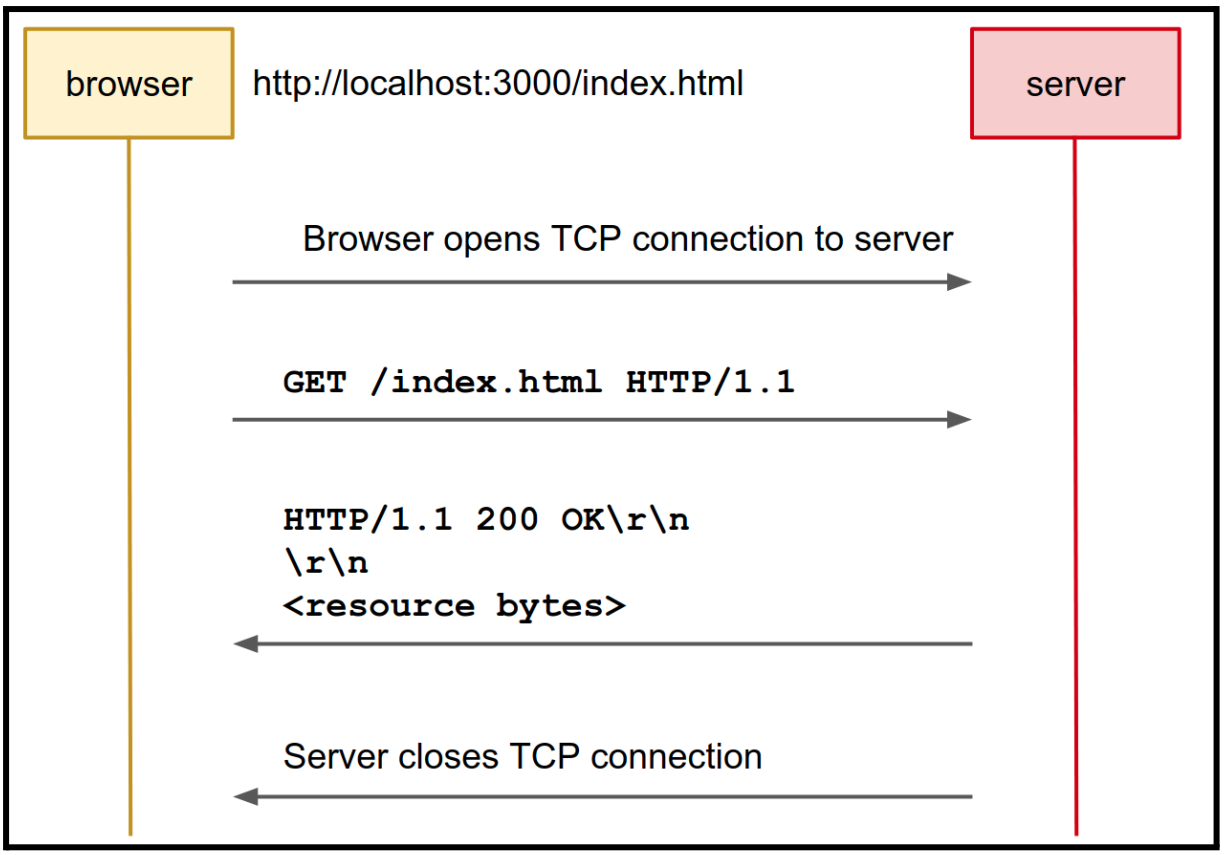
The first term is always `HTTP/1.1`.

If the resource exists, then the response line will be `200 OK` as shown above. The line ends with a `\r\n` string (carriage return, new line).

Immediately after the response line is a blank line (the `\r\n` string). The contents of the request resource (eg `/index.html`) will follow after the blank line. The contents will be returned as bytes.

Once all the bytes of the contents are sent, the server will close the TCP connection to the client.

The following figure summarizes the interaction between the browser and the server



For the assessment, your HTTP server will serve text files (eg. HTML, CSS, JavaScript) and PNG images only.

Assessment

Task 1 (2 marks)
Generate a Java Maven project. Create a remote git repository. Add/link the Maven project to a remote git repository.

You should now perform an initial commit and push of your project to the remote repository.

Task 2 (3 marks)
Your HTTP server must have the following 3 classes

- `Main` - this is the web server's main class viz. the entrypoint of the HTTP server
- `HttpServer` - this is the main web server class
- `HttpClientConnection` - this class handles the the request and response communication between the server and a client (browser)

You should implement all following tasks in one or more of the above classes according to their functionality.

You may add additional classes if you think you need them.

Add `Main` to your Maven `pom.xml` as the starting class (`mainClass`) so you can execute the HTTP server with the '`java -jar`' option.

Task 3 (5 marks)
The web server should accept the following command line options when starting

- `--port <port number>` - the port that the server will listen to. If this is not specified, then default to port 3000
- `--docRoot <colon delimited list of directories>` - one or more directories where the HTML, CSS and JavaScript files and images are stored. If not specified, default to `static` directory in the current path.

Command line examples; assume that your HTTP server is in `myserver.jar`

`java -cp ./myserver.jar`
will start the server on port 3000; the `docRoot` directory is `./target`

`java -cp ./myserver.jar --port 8080`
will start the server on port 8080; the `docRoot` directory is `./target`

`java -cp ./myserver.jar --docRoot ./target:/opt/tmp/www`
will start the server on port 3000; the `docRoot` directories will be `./target` and `/opt/tmp/www`

`java -cp ./myserver.jar --port 8080 --docRoot ./target:/opt/tmp/www`
will start the server on port 8080; the `docRoot` directories will be `./target` and `/opt/tmp/www`

Task 4 (marks 5)
When the HTTP server starts, perform the following

- open a TCP connection and listen on the port from `port` option
- Check each of the `docRoot` path; for each path verify that
  - the path exists, and
  - the path is a directory
  - the path readable by the server

If any of the above conditions fails you should print the failure reason on the console, stop the server and exit the program with `System.exit(1)`.

Task 5 (marks 5)
Create a thread pool with 3 threads.

The server will listen on the specified port, accept incoming connections from the browser. When a new connection is established, this connection

should be handled by a thread from the threadpool. The main control thread (server) should go back to waiting for new incoming connections.

Task 6 (25 marks)
The client thread (handling the client connection) should perform the following task

Read the first line of from the incoming request and perform one of the following actions

Action 1 - Not a GET method
If the request method is not a `GET` method, send the following response back to the client.

```
HTTP/1.1 405 Method Not Allowed\r\n
\r\n
<method name> not supported\r\n
```

Close the connection and exit the thread.

Action 2 - Resource does not exists
If the requested resource is not found send the following back to the client.

```
HTTP/1.1 404 Not Found\r\n
\r\n
<resource name> not found\r\n
```

Close the connection and exit the thread.

If the resource name is `/`, replace it with `/index.html` before performing the file search

Action 3 - Resource exist
If the resource is found in any of the `docRoot` directories, send the resource contents as bytes back to the client in the following response

```
HTTP/1.1 200 OK\r\n
\r\n
<resource contents as bytes>
```

Close the connection and exit the thread.

Action 4 - Resource exist and is a PNG image
If the requested resource exists and the name ends with the `.png` suffix, then the resource is a PNG image. Send the following response back to the client

```
HTTP/1.1 200 OK\r\n
Content-Type: image/png\r\n
\r\n
<resource contents as bytes>
```

Close the connection and exit the thread.

Example
If the HTTP server receives the following request

```
GET /about.html HTTP/1.1
```

Action 1 should check if the method is a `GET` method. In this case, this condition is true, so proceed to Action 2

Action 2 should check if the resource `/about.html` exists. The resource should be checked in the directories specified by the `docRoot` option. For example if the following is specified at startup

```
--docRoot /opt/tmp:/var/lib/www,
```

then you should check if a file `about.html` exists in `/opt/tmp` and `/var/lib/www` directories. If the file does not exist in any of the `docRoot` directories, perform Action 2. If the file exists, proceed to Action 3.

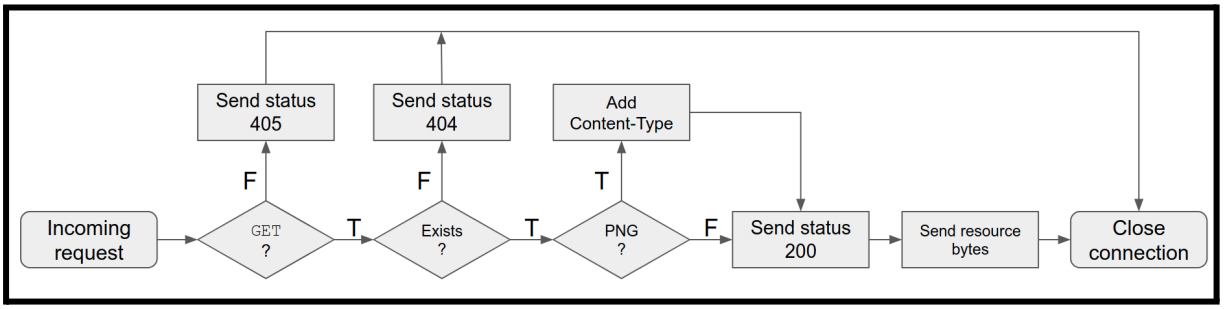
Action 3 is when the requested resource is found in one of the `docRoot` directories. When you have found the resource, you should stop

searching the remaining `docRoot` directories (if any); send the contents of the file to the client as specified in Action 3 above.

Action 4 is when the requested resource is a PNG image (files ending in `.png)`; in this case respond according to Action 4 above.

Once you have performed one of the above actions, close the connection with the client and exit the thread.

The following is a flow diagram of the process

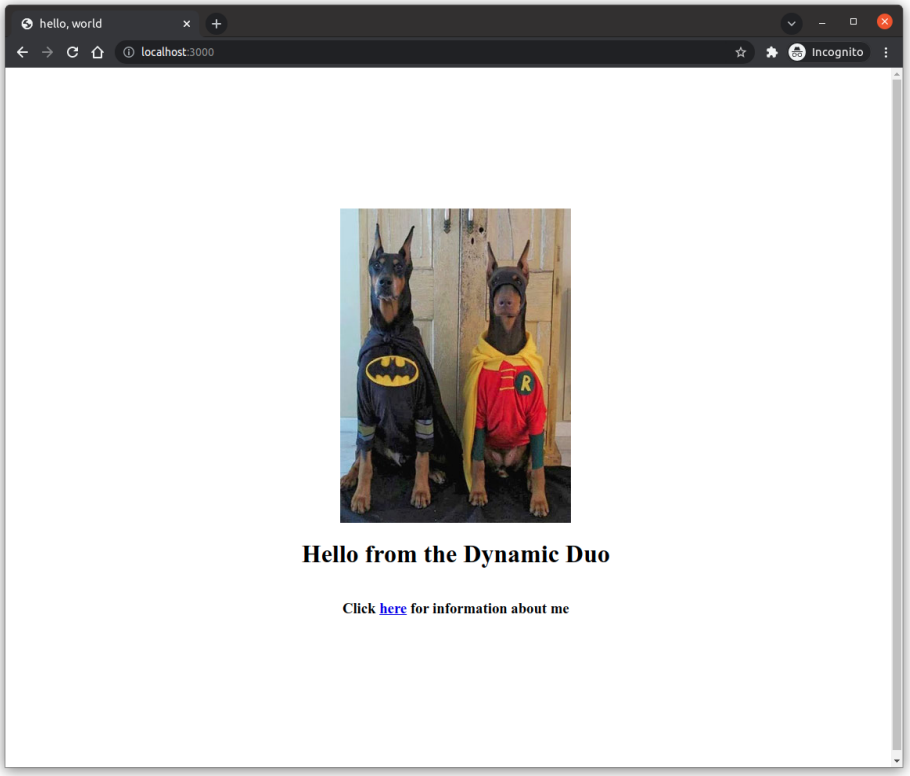


Task 7 (5 marks)

Create a directory called `static` at the root of your project folder. Write a HTML document called `index.html` with the following content

- any PNG image. This PNG image must be in the `static` folder. You should reference this image as `<img href="/mypic.png">` where `mypic.png` is your image
- a header line `<h1>` with any text
- a link to another HTML document in `static` directory
- any text resources (eg CSS, JavaScript) referenced by `index.html` should also be placed in the `static` directory

An example is shown in the following image



The image, the text and the link must be positioned at the center (both horizontally and vertically) of the browser's viewport (window); it should remain at the center when the browser's window is resized.

Run your HTTP server and test if you can access your HTML document from the browser. Try the following URLs

```
http://localhost:<port>
http://localhost:<port>/index.html
```

`HttpWriter` Utility Class

A helper class is provided for your convenience. Download the `HttpWriter.java` file provided by the invigilator to the appropriate directory in your Maven project.

Study the class and decide if you choose to use it. No marks will be deducted if you choose not to use this utility class.

Remember to add `package` to the `HttpWriter` class if you are using this class.