National University of Singapore
School of Computing

CS2105 **Assignment 1** Semester 1 AY17/18

Learning Objective

In this assignment, you will implement a simple Web Server that is able to serve static web pages to a client. After solving this assignment, you should be able to (1) do simple network programming in Java, and (2) have a good understanding of HTTP.

Group Work

This assignment should be solved individually. However, if you struggle, you are free to form a team with another student (max. team size is 2). Group submissions are subject to **1 mark penalty** for each students.

Under no circumstances should you solve it in a group and then submit it as an individual solution. This is considered plagiarism. Please also refer to the detailed explanations below and the skeleton code for more information about plagiarism. If you have doubts, submit as group. Group work need special care during submission (see below).

We allow group work because some students might struggle to solve this assignment alone. Especially, if you are new to Java and programming. Do not give up, do not copy. Instead, form groups and solve the assignment together.

Submission Deadline

25th September 2017 (Monday), 6:00pm sharp. Please start early and submit early. You can always upload a newer version before the deadline. We strongly advice against submitting only a few minutes before the deadline. 1 mark penalty will be imposed on late submission (Late submission refers to submission or re-submission after the deadline, even if it is only 5 minutes late).

This programming assignment is worth 7 marks.

General Program Submission

Please submit your **`WebServer.java`** program to **`CodeCrunch`**: https://codecrunch.comp.nus.edu.sg. Only submit this file. Do not submit a folder or a zip file. If you want to use more classes, do not create extra Java files, but implement them all in **`WebServer.java`** instead. **Make sure the file contains your student number and name at the top**.

Special Instructions for Group Submissions

If you solve this assignment in a group, please ensure you declare it in the header of the submitted program. You have to (1) declare that it is a group submission, and (2) list both members of the team (check the skeleton code for instructions). Do not solve the assignment in a group and submit it as individual work. This is considered plagiarism. All submission will be subject to a thorough plagiarism check. Plagiarism cases will receive 0 mark. There will be no allowance for people who forgot to declare that they solved the assignment in a group.

For group submissions **one and only one** designated student should submit solutions to CodeCrunch. Other team members should never submit at all. It is up to you to decide who the designated submitter is.

Please do not change the designated submitter. For example, student A and student B form a group. After solving the assignment together, student A uploads the solution to CodeCrunch. Later, the group realizes that there is a bug. After fixing the bug, the same student A should re-upload the solution to CodeCrunch. Under no circumstances should student B upload any code.

You are not allowed to share your solution programs with anyone else.

Plagiarism Warning

You are free to discuss this assignment with your friends. However, you should **design and write** your own code. We employ a zero-tolerance policy against plagiarism. Confirmed breach will result in zero mark for the assignment and further disciplinary action from the school.

If you have questions about what constitutes plagiarism, please read the skeleton. It contains a fairly good description of acceptable behaviour. If you still have questions, please approach the teaching staff.

For example, you should refrain from having detailed discussions with other students. Additionally, you should not share notes. You should not allow anyone to look at your code. If you want to solve this assignment together, please do so **but declare it as group work**.

Do not copy code from the Internet. This assignment is relatively straightforward. If you are stuck, please approach the teaching staff. If you need to refer to the Internet, chances are high that you are thinking too much. Keep it simple and straight instead.

The same plagiarism rules will be applied to subsequent assignments.

Grading Rubric

1. **[2 points]** You followed the submission format, e.g. you submit one file only, the name of the file is **`WebServer.java`**, and the class has the correct name (**`WebServer`**). You use the given skeleton and your program compiles on sunfire without errors.

2. **[2 points]** Your Web Server can handle a single Web object like an HTML file or an image file. It is sufficient to support "GET" request only. You only have to support a very limited subset of HTTP. Please refer to the detailed explanation at the end of this paper.

3. **[1 point]** Your Web Server can handle a Web page that contains multiple objects. Again, supporting "GET" request is sufficient.

4. **[1 point]** Your Web Server sends a “404” response when appropriate. Please refer to the lecture notes or the Internet for an explanation of 404 error.

5. **[1 point]** Your Web Server understands both HTTP/1.0 and HTTP/1.1. For HTTP/1.1 (and HTTP/1.1 only), your Web Server supports persistent connections with pipelining. If you have to use a timer, set the time out value to 2 seconds.

Other Remarks

1. Submissions that **fail the plagiarism check** will be marked with **0**.
2. During grading, we will only transfer files that are sufficiently small (< 5 MB). Thus, it is safe to load the entire file into memory. Please refer to assignment 0 exercise 2 if you are unsure how to do it.

The Web Server

You are given a folder containing multiple files. `WebServer.java` is a skeleton program for a Web Server implementation. This is the only file you need to edit. The remaining files are for testing purpose. Use the given skeleton program and **do not start from scratch**.

To run the Web Server program on sunfire, use the following command:

```
java WebServer <port>
```

For example:

```
java WebServer 8000
```

You should choose a port number greater than 1023 because operating systems restrict the access to ports lower than 1024. If you get a **`BindException: Address already in use`** (or similar), try a different port number.

To solve this assignment, you have to fill in the blanks in the provided skeleton. All the blanks are marked. A Web Server is just a TCP server that understands HTTP. It will (1) create a TCP socket, (2) listen to the socket and accept a new TCP connection, (3) read HTTP request(s) from TCP connection and handle it/them, and finally, (4) send HTTP response(s) to the client through TCP.

Following good OOP practise, we will separate various functionalities into different classes/methods.

1. The **`start`** method of the **`WebServer`** class should create a **`ServerSocket`** and listen to new connection requests. Once a TCP connection is established, this method should pass the connection socket to **`handleClientSocket`** method (please refer to lecture 3 notes for a demonstration of TCP ServerSocket and Socket).
2. **`handleClientSocket`** will then take all necessary steps to handle a HTTP request from a client. It should make use of the **`HttpRequest`** class. First, you have to read a HTTP request from the socket. Next, you should call **`parseRequest`** of the **`HttpRequest`** class to parse the request.
3. An instantiated object of the **`HttpRequest`** class represents a single incoming HTTP request. Thus, it is a good idea to create a new **`HttpRequest`** object for each request.
4. Once the request is parsed, Web Server should call **`formHttpResponse.`** This method inspects the **`HttpRequest`** and decide whether to send a 200 OK or a 404 response. **`formHttpResponse`** returns a byte array which can be sent to the client using **`sendHttpResponse`**.

Testing Your Solution

We will grade your submission on **`sunfire`**. Thus, it is a good idea to test your submission on **`sunfire`** before submission. In the past we had submissions that do not work on **`sunfire`**, e.g. because **`sunfire`** might have a different Java versions installed.

You can test your Web Server in many different ways. For example, using browsers, using the command line, or by manually sending requests to the Web Server. We also provide a test case.

Test case: We will use carefully crafted scripts to test the functionality of your submission. For your convenience, we provide a simplified and condensed version of our test cases for your own testing. We strongly recommend you to actually execute this test case before submission. Passing this test case does not guarantee full marks but failing it definitely means something is wrong. To execute the test case, you need to (1) open two SSH windows to sunfire; (2) run your Web Server in one window, and (3) execute the following command in another window:

```
bash test.sh <port>
```

For example,

```
bash test.sh 8000
```

HTTP Specification

In this assignment, your Web Server has to support both HTTP/1.0 and HTTP/1.1. Do not despair. You only have to implement a very small subset of the two protocols as shown below. They are extracted from RFC 2616 (https://tools.ietf.org/html/rfc2616).

Your Web Server only has to support the following syntax.

HTTP 1.0

SP = " " (Exactly one single whitespace)

CRLF = "\r\n"

HTTP 1.0 Request

Request = Request-Line *(Header-Line) CRLF

Request-Line = "GET" SP Request-URI SP "HTTP/1.0" CRLF

Request-URI = absolute path

E.g. /demo.html or /some/other/file or similar. (Note that this path is absolute relative to the current working directory of the Web Server. Thus, /demo.html does not refer to /demo.html on the file system. Instead, it refers to demo.html in the current working directory)

Header-Line = Field-Name ":" SP Field-Value CRLF

(Field-Name is a string with no whitespaces and no ":". Field-Value is a string)

HTTP 1.0 Response

Response = Status-Line *(Entity-Header) CRLF

Status-Line = “HTTP/1.0” SP Status-Code CRLF

Status-Code = “404 Not Found” | “200 OK”

Entity-Header = “Content-Length:” SP Number CRLF

HTTP 1.1

For HTTP/1.1, your Web Server needs to support persistent connections with pipelining.

HTTP 1.1 Request

Request = Request-Line *(Header-Line) CRLF

Request-Line = “GET” SP Request-URI SP “HTTP/1.1” CRLF

Request-URI and Header-Line stay the same.

HTTP 1.1 Response

Response = Status-Line Entity-Header CRLF

Status-Line = “HTTP/1.1” SP Status-Code CRLF

Status-Code and Entity-Header stay the same.

In this assignment, we assume an HTTP request is valid. Therefore, you can ignore all the Header-Lines after reading them.