LIÈGE université
Sciences Appliquées

# INFO0010: Introduction to Computer Networking
# JavaSocketProject - Report

December $04^{th}$, 2020

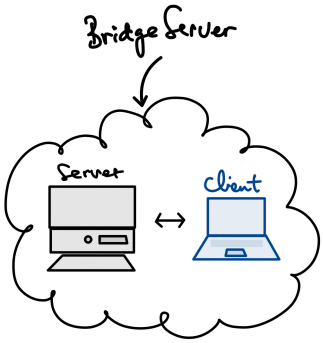

| | | |
|---|---|---|
| Alain Eros Prestige | Musoni | s202208 |
| Ayawo Désiré | Dandji | s197206 |


## Introduction

This assignment is about implementing an application using Java Sockets that acts as a server and client at the same time, this one may also be called a bridge or bridge server. This report is organised following different the parts that were implemented during the project, how we implemented them and then the last part will discuss the difficulties and how we overcame them.

## 1 Bridge Server

The bridge server plays the main role in this project, it runs the server and the client as mentioned above. We can visualize the abstraction of it as shown in the figure bellow.



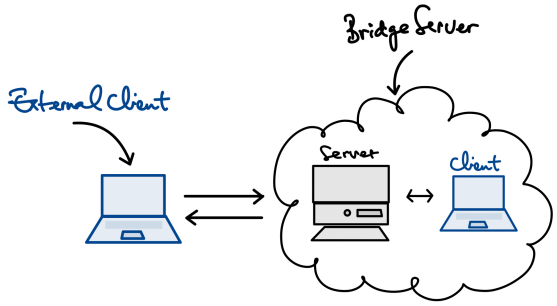

This is implemented in the java class `BridgeServer.java` which instantiate the Server object, in order to start listening to whoever is going to ask for connection. In addition, this class accepts threads. You have to mention the **maximum number of threads** it should accepts which corresponds to how many clients that will be handled at the same time.

# 2 Server in the bridge

The server its self is implemented in its own class called `Server.java` then instantiated in the bridge. Its role is to handle incoming clients from the external world as shown in the figure bellow.

## 2.1 Incoming from external clients

There is socket communication between the external client and server of the bridge that is happening here. To achieve this in the implementation, we created a new socket object called `_serverSocket` which uses `InputStream` object that we named `_in` to receive connection.




The external client above is trying to connect to the server, and the two lines in the middle above describes packets exchanged during the process.

## 2.2 Client handle

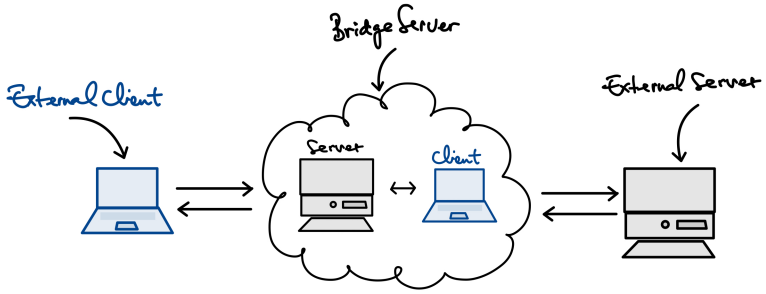
When an external client's connection is accepted, it is passed a thread in order to be executed. The implementation of this is in the java class called `ClientHandler.java`, this class implements Runnable and it is the one in charge of executing a client's commands out of a number of clients that are being processed thus enabling **multithreading** as requested. In practice the external client has to provide a valid **url string** that will later be passed to the client side of the bridge to be processed.

# 3 Client in the bridge

After the client receiving the url string, it checks if the url is valid according to the requirements, it has to use http protocol for example. Then, we have used the URL java class to make a valid url object that we will use to build Http Request to the external server(s) as shown in the figure bellow.

## 3.1 Outgoing request to external servers

In this project we kept in mind that a client might want to make multiple requests at the same time.



This is allowed as long as url strings are separated by `CRLF` to delimite each one of them. The management of url links given is implemented in a function called `getURLs` which takes in the link and returns delimited links as an array list which are then turned into actual URL objects to make it easy to work with.

### 3.2 Server handler

For the sake of performance, we made the external servers run into their own threads each. We implemented this in the class we named `ServerHandle.java` which receives a URL, builds a request from it, sends it to the external server and wait for a response in order to calculate statistics. We created a bunch of helper functions like `buildRequest` which returns a valid request as a string, `getResponseCode` which retrieves response code from the response got from external server and so on. The method that does a lot of work is the `sendGetRequest` function, it is in charge of sending request and receiving the response, call redirect function if necessary then send back statistics to the client who sent the request.

### 3.3 Statistics

The statistics calculation was based on counting the number of occurrences of words in the response content we received. This is implemented int the `statistics` function which uses the `tokenizer` function the returns tokens from the data received as a string. After getting tokens as an array list, we then make use of a **TreeMap**$< Integer, String >$ java class to store our values in a sorted manner. The last thing to do was to retrieve top 10 starting from the last element of the sorted TreeMap and send it as a string to the external client.

## 4 Difficulties

A large amount the of difficulties that we faced with were not huge, it required us to read documentations and dig deeper. However we have faced a threading issue that was tricky to diagnose. This was due to a shared OutputStream Object that was being accessed by a number of threads at the same time. Finally we solved this by making our function synchronous so that a threads wouldn't write on the stream at the same time, if a thread starts to use it others are locked until this one finishes to write then others can follow.

## 5 Conclusion

This project re-enforced our understanding of how socket works in a visual way which cannot be visualized clearly in theoretical sessions. We managed to work on the project by dividing the tasks and recombining after in order to understand each and every part of the project. Ayawo Desire worked on everything related to the client and Alain Eros worked on the other part related to the server, we had time approximately every two days to discuss the progress and gather ideas that will be implemented the next day.