Project #1 - Webserver

이름: Vitoria Baldan / 비토리아 학번: 2020049061
학과: 컴퓨터 소프트웨어 컴퓨터 네트워크

For this project, it was required to implement a Web Server program that would do port forwarding between two addresses (student and lab), and send files such as an HTML file and a JGP image. The challenges of this project were first, to make a TCP socket connection between the student server and the lab host. After that, we needed to implement an HTTP request class that would build a header, set responses and methods, and manage the socket connection.

Source Files

1. Web Server (main) Class

```
import java.io.* ;
import java.net.* ;

public final class WebServer {
    public static void main(String argv[]) throws Exception {

        try {
            // Could get the port number from the command line.
//          int port = (new Integer(argv[0])).intValue();

            // Establish the listen socket.
            ServerSocket serversocket = new ServerSocket( port: 7065);

            // Process HTTP service requests in an infinite loop.
            while (true) {
                // Listen for a TCP connection request.
                Socket socket = serversocket.accept();

                // Construct an object to process the HTTP request message.
                HttpRequest request = new HttpRequest(socket);

                // Create a new thread to process the request.
                Thread thread = new Thread(request);
                // Start the thread.
                thread.start();
            }
        }catch(IOException e) {
            System.out.print(e.getMessage());
        }catch(Exception e) {
            System.out.print(e.getMessage());
        }
```

2. HTTP Request Class (Some of the essential methods)

a) sendHeaderLines method

```
public void sendHeaderLines(DataOutputStream os, java.lang.String requestLine ) throws Exception{

    StringTokenizer tokens = new StringTokenizer(requestLine);

    String method = tokens.nextToken().toUpperCase();
    String fileName = tokens.nextToken();
    fileName = "C:\\Users\\balda\\IdeaProjects\\Project1\\src" + fileName;
    File file = new File(fileName);
    String fname = fileName.toLowerCase();
    int lastdot = fname.lastIndexOf( str: ".");

    StringBuffer headerLines = new StringBuffer();
    String contentTypeLine = "Content-Type: ";
    String contentLength = "Content-Length: ";
    System.out.println("code "+ code);
    //System.out.println(fname.substring(lastdot+1));
    switch (code) {
        case OK:
            if(fname.substring(lastdot+1) == "jgp") {
                contentTypeLine += "image/jpg" + CRLF;
            }
            else if(fname.substring(lastdot+1) == "html") {
                contentTypeLine += "text/html" + CRLF;
            }
        default:
            contentTypeLine += "text/html" + CRLF;
    }
    headerLines.append(contentTypeLine);
    os.writeBytes(headerLines.toString());
```

b) Constructor and run() method

```
// Constructor
public HttpRequest(Socket socket) throws Exception {
    this.socket = socket;
    this.code = null;
    this.requestedFile = null;
}

// Implement the run() method of the Runnable interface.
public void run() {
    try {
        processRequest();
    } catch (Exception e) {
        System.out.println(e);
    }
}
```

c) parseRequestLine

```
public String parseRequestLine(java.lang.String requestLine) {
    // TODO Auto-generated method stub
    System.out.println();
    System.out.println("Received HTTP request: ");
    System.out.println(requestLine);
    StringTokenizer tokens = new StringTokenizer(requestLine);
    if(tokens.countTokens() != 3) {
        code = StatusCode.NOT_FOUND;
        return "Request line malformed. Returning BAD NOT FOUND.";
    }

    String method = tokens.nextToken().toUpperCase();
    String fileName = tokens.nextToken();

    System.out.println("file name: "+fileName);

    //Modify to your path
    fileName = fileName.replace( target: "/",  replacement: "\\");
   // fileName = "." + fileName;
    fileName = "C:\\Users\\balda\\IdeaProjects\\Project1\\src" + fileName;
    File file = new File(fileName);
    System.out.println("file name: "+fileName);
   if(!file.exists()) {
       code = StatusCode.NOT_FOUND;
       return "File not FOUND";
   }
   if(!file.canRead()) {
       code = StatusCode.FORBIDDEN;
       return "File is not Readdable";
   }
   if(file.isDirectory()) {
       File[] list = file.listFiles(new FilenameFilter() {
           public boolean accept(File dir, String f) {
               if(f.equalsIgnoreCase( anotherString: "index.html"))
                   return true;
               return false;
           }
       });
       if(list == null || list.length == 0) {
           code = StatusCode.NOT_FOUND;
           return "File not FOUND";
       }
       else if(list.length != 1){
           code = StatusCode.INTERNAL_SERVER_ERROR;
           return "found more than one file";
       }
       file = list[0];
    requestedFile = file;
    String version = tokens.nextToken().toUpperCase();
    if(version.equals("HTTP/1.0")) {
        code = StatusCode.BAD_REQUEST;
        return "HTTP Version String is malformed.";
    }
    if(!version.matches( regex: "HTTP/([1-9][0-9.]*)")) {
        code = StatusCode.BAD_REQUEST;
        return "HTTP Version String is malformed.";
    }
    if(!version.equals("HTTP/1.0") && !version.equals("HTTP/1.1") ) {
        code = StatusCode.HTTP_VERSION_NOT_SUPPORTED;
        return "HTTP Version is not supported.";
    }

    code = StatusCode.OK;
    return null;
```

Instructions

First, you need to set a port number to start your socket connection with the other user. Also, the IP address is essential to make the connection work.

To test the connection by yourself, you can call localhost:(port)/filename. In this case, I created a simple index.html file to test the server and also got a jpg image to test. In this case, for example, you should first run your program by the Web Server class, then go to your browser and type http://localhost:port/index.html or /image.jpg.

If testing by port forwarding with another user in a different network you must first add the receiver's IP to your router's port forwarding allowed address. Also, you might need to define a common port bonded to the receiver's IP to set in your router port forwarding system. After setting all the permissions necessary, you may run the Web Server class, and when the receiver opens your file in the browser, you might receive the responses in your IDE or terminal.

How the program works

The Web Server class is the main runnable class. Its main function is to establish the socket connection and run the HTTP request. Also, initializing the thread, allowing multiple requests.

The HTTP request class is the actual heart of the Web Server application itself. First, in the HTTP class, we have the declarations of the response codes, sizes, types of files allowed, and the HTT version set as 1.1. Then we have the methods:

1. HTTP Request - Constructor - `HttpRequest(Socket socket)`
   This method works as a constructor for the HTTP class, setting the received socket, the code response is null, and the request is null as well.
2. Run - `run()`
   The run method will basically call the processRequest() method.
3. Process Request - `processRequest()`
   The process request method will receive the socket data, open the file, parse the request message and send the response message. All this by calling the designed methods for each function and at the end, it closes the socket.
4. Send Response - `sendResponseMessage(FileInputStream fis, DataOutputStream os, java.lang.String requestLine)`
   The send response method will build the status response line checking which code is received and the entity-body. Also, this method will call the sendHeaderlines method at the end passing the request as a parameter
5. Send Header - `sendHeaderLines(DataOutputStream os, java.lang.String requestLine )`
   This method will check which type of file is being received in the request and write in the header accordingly.
6. Parse Request Line - `parseRequestLine(java.lang.String requestLine)`
   This method will build the server response message, checking the file location, whether the file exists or not, if the file is protected, if the request line has any structural issue, and will associate these checks in a switch-case, defining each Status code for each particular issue.

Also, will check the HTTP version and will set the requested file as the received one, if there are no problems with the file.

Results

Automatic testing tool

When testing in the Automatic testing lab, we had 8 missions and it was as the image below:

Student INFO

Student Number	Student Name	WebServer IP	WebServer PORT	ACCESS TIME	SCORE
2020049061	VITORIA ONGARATTO BAL	166.104.28.166	7065	2021-15-25 06:15:37	90

List of Test Items

WEB SERVER SOCKET :	TRUE	
Multi Thread :	TRUE	
STATUS CODE : 200 OK	TRUE	
STATUS CODE : 404 NOT FOUND(EXCEPTION HANDLING)	TRUE	
STATUS CODE : 400 BAD REQUEST(HTTP PROTOCOL VERSION)	TRUE	
CONTENT LENGTH	TRUE	
CONTENT TYPE TEXT/HTML	TRUE	
CONTENT TYPE IMAGE/JPEG	False	Second(image) Page content type should be image/jpg foramt

Some responses at the terminal were:

```
Received HTTP request:
GET /image.jpg HTTP/1.1
file name: /image.jpg
file name: C:\Users\balda\IdeaProjects\Project1\src\image.jpg
Host: 166.104.28.166
Connection: keep-alive
Content-Length: 0
StatusLine : HTTP/1.1 200 OK
entityBody :
<HTML>
 <HEAD><TITLE>?</TITLE></HEAD>
 <BODY>?</BODY>

code OK
sending request file to Client...
```

```
Received HTTP request:
GET /index.html HTTP/1.1
file name: /index.html
file name: C:\Users\balda\IdeaProjects\Project1\src\index.html
Host: 166.104.28.166
Connection: keep-alive
Content-Length: 0
StatusLine : HTTP/1.1 200 OK
entityBody :
<HTML>
 <HEAD><TITLE>?</TITLE></HEAD>
 <BODY>?</BODY>

code OK
sending request file to Client...
```

```
Received HTTP request:
GET /mir.html HTTP/1.1
file name: /mir.html
file name: C:\Users\balda\IdeaProjects\Project1\src\mir.html
Host: 166.104.28.166
Connection: keep-alive
Content-Length: 0

File not FOUND
StatusLine : HTTP/1.1 404 NOT_FOUND
entityBody :
<HTML>
 <HEAD><TITLE>NOT_FOUND - sent by Students Webserver</TITLE></HEAD>
 <BODY>NOT_FOUND - sent by Students Webserver</BODY>

code NOT_FOUND
```

Conclusion

In general, the code works well making the socket connection and establishing an HTTP process. Most of the Missions given came out as 'True' what means that the code works properly. However, even after several attempts to fix the jpg file type message in my 'parseHeader' method, I still couldn't make it work. I believe that I might have needed to check in other methods if there is something wrong, but I honestly believe that the problem is still in the file name I access in the 'parseHeader' method.

Out of this, my personal feelings are very positive, I had fun while testing and coding, it was my first time coding something like that and I'm really interested in learning more about server-client coding.

Fun fact: I was doing some final tests on the code Monday, October 25th, in the morning. The exact moment when KT systems went down and collapsed. When my internet connection stopped working I got a bit worried (and, somehow proud) that I might have burned my router while running the socket program. It was fun at the end when I came to know it was KT's problem and it wasn't me that had ruined my router.