CPSC 441 Assignment 2 Design Document

Fall 2022, Brett Gattinger, 30009390

How I read and parsed the Server Response

I began by initializing an empty string called 'Http_responseHead', an empty string called 'detector' and a Boolean called 'resposnseHeadRead' which was initialized to false

After I had sent my Http request to the server (and initialized the above variables) I began reading from the input stream of the server socket one byte at a time until either a -1 was returned by the read function or 'responseHeadRead' was set to true. Each byte that was read in was appended to 'Http_responseHead' and, if the Boolean 'responseHeadRead' was not yet set to true, was also checked to see if it equaled the line feed or carriage return characters. The goal was to detect the character sequence '\r\n\r\n' as I was reading in the initial bytes from the server's response as that would indicate when I had read the entirety of the response head and could start reading the response body using a larger buffer. I did this by performing the following:

- If a '\r' or '\n' was detected (read in) it would be appended to the string 'detector'
- If any other character was read in then 'detector' would be reset
- Each time a byte was read in 'detector' would be evaluated (after either being reset or having '\r' or '\n' appended to it) to see if it equaled '\r\n\r\n'

I knew that once the end of the response header was reached, the string 'detector' would have '\r' then '\n' then '\r' and finally '\n' consecutively appended to it, at which point the Boolean 'responseHeadRead' would be set to true (because the response header had indeed been read) and the while loop that was reading in 1 byte at a time from the server sockets input stream would then terminate. By then the string 'Http_responseHead', which was having every read-in character appended to it the whole time, would be equal to the entire response header returned by the server. This was all done using the basic InputStream object.

After that, I split 'Http_responseHead' on the regular expression '\r\n' to break it up into its status and header lines (stored in a string array called 'Http_responseHeadLines'). I retrieved the status code from 'Http_responseHeadLines[0]' by splitting on the space character and I retrieved the Content-Length value by iterating through the string array 'Http_responseHeadLines' and splitting each string in it on ":" and checking for "Content-Length".

Finally, if the extracted Status code was 200, I created a FileOutputStream object and began writing bytes into it from those bytes being read in again from the server sockets InputStream (This time using a 32*1024 sized byte buffer) thus downloading the data in the body of the server's Http response into a local file of the same name as that in the GET request.