**Description**

My program is a simple proxy that takes a pc’s network traffic and runs it through a local proxy port for both in and out bound traffic. The program makes use of a few key components in order to function; these include Streams and buffers, both for input and output, as well as sockets, both standard and the ServerSocket type. The general flow of a successful connection is as follows:

Browser sends HTTP request -> Proxy creates a socket, intercepts and reads the get message with an input stream -> program parses request, and if the proper message type is found, the program moves forward -> program searches its cache for an instance of the requested content with in it -> **Branch**

**Branch A**: Program does not have a cache instance of webpage:

the program creates a socket with the desired webserver -> program writes request to webserver with a socket output stream -> program then waits and reads the server reply with an accompanying socket input stream -> program then does any response processing as needed -> program writes response objects to browser with the help of that client’s socket’s output stream -> Program creates cache for later use.

**Branch B**: Program has cache of webpage

The program finds the accompanying information by getting the Map tuple -> program writes response objects to browser with the help of that client’s socket’s output stream.

The final step is that the browser then displays the written information, and the program writes the proper logs and information to their corresponding places for later review and/or troubleshooting.

**Hurdles, Setbacks, and Lessons**

When looking back on the obstacles in place on my path to completing this project, I can honestly say they are few in number. But the presented issues still may not have a definitive and issue-free solution, at least in the scope of the classes and libraries we were allowed to use.

The biggest hurdle I ran into was an issues with and infinite loop when reading the InputStream attached to the web browser, until the browser was closed or one hit the ‘stop’ button in the browser, after which the program would continue, though obviously without properly getting, processing, and sending on the request. My remedy ended up being to use a BufferedReader instance to ensure the loop ends and the program responds in a timely fashion. This issue is of note particularly because of the difficulty I had consistently recreating the issue, as some sort of outside environment factors that were out of my control seemed to have been at play during my exhaustive testing. Interestingly enough, I did not require a buffered reader for the server replies. Given this, I had to conclude that there was some sort of quirk with my specific browser configuration that made a raw InputStream read() unusable.

One other issue was dealing with CONNECT requests from both the browsers underlying processes and the numerous online-connected applications running on my pc. And when I say ‘deal’ with them, it was more pertaining to properly managing and joining the threads that these intermittent requests would create. Before dealing with it, I would take huge performance hits in trying to test the intended functions of my program, if not just causing a myriad of exceptions to occur, upsetting the control flow of my proxy and causing a lot of unwanted behavior. Eventually the solution came about and is rather simple. I just made sure to verify the request type as early in the processing stage as possible, and quickly join the RequestHandlers that were erroneously created for CONNECT messages.

Finally, and more generally, I certainly found it difficult to be sure I was on the right track while setting up the initial program. This was simply due to the fact that a lot of the steps in the process are very invisible to me, so I was unsure where certain problems in the communication chain lied, as well as whether or not any given single step was properly constructed. Though I will note this obfuscation and frustration with error diagnosis culminated in an immense sense of accomplishment once I had successfully and entirely loaded a test webpage, so I believe that initial strife was more than worth it for the payoff.

**Lacking Features, Incompatibilities and Improvements**

The first and most glaring omission in regards to my proxy’s feature set is the ability to properly work for https websites. This was intentional from within the assignment itself, but it still merits mentioning that certain sites, likely the less secure ones, do still work through my proxy.

After that largest glaring absence, there are a few other features that I believe would increase the programs utility greatly. The first is obviously the implementation of features one finds standard on a commercial / full proxy. Those would be things like virus-checking, firewall/anonymity features, bandwidth / page size limitations, and the like. Many of these would require a much more sophisticated set of algorithms and classes, though using higher-level classes in Java rather than my custom-made proxy would probably alleviate those requirements.

Related to this is the fact that our implementations needed to use such a brute-force methodology when interpreting messages. Java has plenty of classes for getting, for instance, whether or not a message is a GET or POST. This led to more time than I would’ve preferred just parsing strings in ways that are properly error-resistant when tools that solved these problems are available. But handling and processing that sort of thing is the point of the assignment so it’s a minor complaint at most.