/\*

\* Juan Tena

\* NMSU-Grants CS187-G01 Java Programming

\* Final Project Log: junkie-soup

\* May 9-May 12, 2016

\*/

**May 9, 2016**

Today I am contemplating my final project for the company. I am constantly stumbling upon interesting subject matter regarding Socket programming. I first began my research online @ [*www.tutorialspoint.com/java/java\_networking.htm*](http://www.tutorialspoint.com/java/java_networking.htm)this page consisted of tabs that went into vivid detail about the java.net package and how it consisted of two common network protocols. TCP and UDP. Obvious TCP is the transmission protocol we want considering it is more reliable. Without hesitation we can make this conclusion.

The next thing to consider would be whether or not the final project should be a Socket scanning program of some sort that makes contact with a server. There was even an option of URL Processing. I decided to continue my research on Socket programming and not get too caught up in the vast number of options at our company disposal. Since Socket programming is the communication mechanism between two computers via TCP, I felt I had made the right choice.

I found a really interesting option in my search of programs. I found a program that establishes a connection at the client’s end, it sends almost a ‘ping’ type call to the server. If the server is present and listening on that port, the server will respond to the client by capitalizing whatever input message was typed in. For example, if we needed to find a server, we could establish a socket communication with the intended server, type a message like ‘hi’ into the text box and if the server is found it will respond with HI. I know I’ve been tasked with making something useful but, I always intended on it being fun too. Undoubtedly, I believe our interests should be in Socket programming.

One of the most useful sites I found today concluded a lot about the Socket class and what kind of java programs will be necessary for establishing a reliable TCP connection between a client and a sever. In short reference it was Prof. Drake’s networking and Sockets page or online @ [*https://www.cs.uic.edu/~troy/spring05/cs450/sockets/socket.html*](https://www.cs.uic.edu/~troy/spring05/cs450/sockets/socket.html)this page consisted of what Prof. Drake defined as ‘simple TCP programs’. I was intrigued by this page and the simplicity in the breakdown of its code. The commentary was easy to comprehend and so today, I decided to sample some of Prof. Drake’s work. The program wouldn’t be successful without the use of a program running on the client and one running on the server. The client established connection was found @ [*https://www.cs.uic.edu/~troy/spring05/cs450/sockets/EchoClient.java*](https://www.cs.uic.edu/~troy/spring05/cs450/sockets/EchoClient.java) and the server established connection was found @ [*https://www.cs.uic.edu/~troy/spring05/cs450/sockets/EchoServer.java*](https://www.cs.uic.edu/~troy/spring05/cs450/sockets/EchoServer.java) I believe I need them both to make a secure line of communication.

I opened up my Eclipse IDE and decided to administer some testing. I wrote the EchoClient.java program first and followed up with the EchoServer.java program in the same *com.tenaj.junkiesoup;* package. This will be the package that consists of code I want to test. The first test was a great success. It is key to run the server program first. It might not respond right away but, it is important to keep running it. The server program will eventually respond with “waiting for connection…” When this happens you return to the EchoClient.java and run that program. When the program runs the client console will inform you that the connection has been established. After the established connection the client console will ask you for an input. As the user, you can type anything you want and if the connection truly was successful the server should echo back what you typed in.

I feel I should note a small flaw in socket programming with the server. Although my client code is capable of speaking with my server code, it is import to remember that server’s only establish a line of communication for a short while. Basically, on a server there is a TTL (time to live) with every established form of communication taking place. This does make sense though. We don’t want our server’s to be overwhelmed by occupying programs. The solution to the problem is to simply shut Eclipse down and reopen it. This kills the server code that establishes the connection, and upon reopening it revives the connection when the program is ran again. This program might be a good addition to my final project. Along the lines of Prof. Drake’s more advanced work, there are alternative options that still deal with client server interaction. Some of Prof. Drake’s code looks like it modifies my previous ClientEcho.java code and the ServerEcho.java code.

Before going forward on modifying my Echo program, I felt it necessary to make myself aware of the many other options available via Socket programming. I went back to my original source of research to see if I could find a runnable program there. I found two programs that were supposed to run compatibly much like the Echo program. I did create another Java Project in Eclipse’s IDE and called it junkie-soup2. In this project I created two separate classes. My client class will be referenced as GreetingClient.java and my server class will be referenced as GreetingServer.java. I found this runnable program @ [*www.tutorialspoint.com/cgi-bin/printpage.cgi*](http://www.tutorialspoint.com/cgi-bin/printpage.cgi)*.*

After running both GreetingClient.java and GreetingServer.java, I had no success. I keep receiving an ArrayIndexOutOfBoundsException. I reviewed the lines where the exception error is said to be taking place but, can’t seem to spot the issue. I’ve concluded that the error resides within the arg[] parameters. It seems the value within the square brackets might not be sufficive to run the program. Writing both these Java classes wasn’t a complete waste though. I was able to spot a line of code in the GreetingServer.java that I feel I could use in my ServerEcho.java file. The line goes just below the serverSocket declaration and it spells out: *serverSocket.setSoTimeout(10000);.* What this will allow me to do is set a timer when the server times out the session. As previously discussed, we had issues keeping the server socket session open for long enough to establish a connection through the client. Now with this little piece of code modification we will be able to customize how long we want our socket to remain open and actively listening. Since GreetingClient.java and Greeting.Server.java weren’t successful, I decided to retort back to Prof. Drake’s simplified Socket programs but, hopefully I can continue to take lines from GreetingServer.java to use in my ServerEcho.java. The plan is to merge as much Socket code as possible and hopefully in the end, our company will have a useful port scanning tool with multiple functionalities.

**May 10, 2016**

Today I decided to review Prof. Drake’s networking and Sockets page again to inquire potential code to add to my ClientEcho.java/ServerEcho.java file. As of yesterday, I applied a serverSocket timer to my ServerEcho.java file. This tiny modification should allow us to keep our TCP thread open longer so the connection between the server and the client can happen with more ease. We established yesterday that the server is only listening on that port for a short period of time unless you specify when to terminate. For the time being I did modify the connection to stay open for (10000) milliseconds, which I believe is the equivalent of 10 seconds. I can always make this number larger if I need to but, for the time being this will work.

It was established yesterday that we were going to focus mostly on reliable connections (TCP) but, that’s not a good enough reason not to look into (UDP). I feel our company can still benefit from utilizing information revolving around the unreliable data packet ports. For security reasons we don’t want to pass a whole lot of information across this protocol but, I figured why not scan the occupied UDP ports? We may not communicate on these channels but, we could certainly add some code to our program that does this. I was luckily enough to have found this simplified UDP port scanner program @ [*https://www.cs.uic.edu/~troy/spring05/cs450/sockets/UDPPortScanner.java*](https://www.cs.uic.edu/~troy/spring05/cs450/sockets/UDPPortScanner.java). This program establishes the starting point and ending point values for the ports to be scanned. The program seems to begin scanning at 1024 and works its way to 65,535. This program is capable of doing this in a matter of seconds. Previously I had visited a site similar to Prof. Drake’s. The website consisted of an interesting combination of programs that involved Socket programming. All of the programs seemed worth running and finding use out of. Similar to my previous program by Prof. Drake, we have a program that allows for user input within a ten second period, the server will echo back what was written. On this particular website I found a program that relays back the echoed message in all caps. I will be trying this program now as well as my other options to see if they are functional. I will continue to keep everything in the com.tenaj.junkiesoup; package but, I will create a few new Java projects containing two separate classes. The separate classes are necessary. One class for the client and one for the server. I found the programs I’m about to run @ [*http://cs.lmu.edu/~ray/notes/javanetexamples/#tictactoe*](http://cs.lmu.edu/~ray/notes/javanetexamples/#tictactoe). On this page there is also a short passage briefly summarizing how communication takes place between a client and a server. It explains what happens when the program compiles the server blocks. It gets the server listening in the event a client might want to communicate.

The first program I decided to run was a capitalize program. It didn’t seem to be working at first but, after running it a few times it seemed to respond as expected. I wrote the CapitalizeServer.java program first then a CapitalizeClient.java program. It is important to run the server program first. We depend on the server listening for a connection through a specified port for a reliable connection. In the console the server will tell you it’s currently listening. When this occurs run the CapitalizeClient.java program and you should be prompted with an applet window asking for an IP address. I got the results I wanted after typing the host address: 127.0.0.1. When the connection is established the applet prompt disappears and an applet window opens up just behind it. The window will greet you and assign you a client ID number. It will also ask the user to enter a line to capitalize specifying that typing a single (.) period will terminate the session. My favorite thing about this program is that it leaves the line of communication open until the user decides to terminate the session. This was the single most obvious flaw in my last couple programs. Another perk is the console keeps a log of the clients who contacted the server via the capitalize applet. It includes the ID, the time and date of termination, and the port of access.

**May 11, 2016**

I am positive our company can benefit from this CapitalizeServer.java/CapitalizeClient.java program. Now the challenging aspect before our deadline at midnight tomorrow is to pair this program with another and format the program to where it’s specific to our company needs. At the moment I am set on merging my TCP capitalize program with my UDP port scanning program. I’m still not sure ‘how’ just yet but, I’m confident there is a way to make the functionality more convenient. Before I decide to commit to this as my final production, I continue searching for code to test on Prof. Drake’s website. I become curious of this program with the description: webserver. The code for it is one sided, meaning it doesn’t seem to contain a client class for it to work. I decide to create another Java Project; this time containing a single class called WebServer.java. Looking at the code, I’m not certain what this will do but, I’m willing to run it and see. I found this program @ <https://www.cs.uic.edu/~troy/spring05/cs450/sockets/WebServer.java>.

After running the program, I didn’t seem to get any errors but, the program also doesn’t seem to do much but wait for responses on a specified port. The program only let you know it was listening on but, nothing more. After running this program and not learning much, I decided to move forward on merging my TCP program with my UDP program. I’m determined to spending my time now to understanding how the programs work and in what areas I can make alterations. I know my final product can be of great benefit to the company. It will allow the company to make quick server presence validations, as well as get accurate occupancy on UDP ports. This program is intended to be a quick, fun, network scanning tool. Although it is not fully detailed to map a network, it should work just fine in acknowledging its presence and how its ports are being utilized.

I have had a few hours to ponder on how to further go my company project. I was very much curious about JFrames in particular. I wish there was a way for me to merge the UDP console aspect to a JFrame output. This would spare me anguish, considering the TCP portion of the code logs client presence on the CapitalizeServer.java. I don’t want to lose this part of the code. I find it valuable but, I also need to find a way to work around it. I must look at my CapitalizeClient.java class in order to determine a good area to place my UDP output. I need to consider the dimensions when making this adjustment, so a little background knowledge might be helpful. I generally find what I’m looking for. I realize I have very little time to thoroughly read through all the JFrame methods but, I am at least able to familiarize myself @ [*http://www.tutorialspoint.com/swing/swing\_jframe.htm*](http://www.tutorialspoint.com/swing/swing_jframe.htm)*.* I think at this point in the process, it’s just best to copy the functional code I have on a notepad document, save it, and put it somewhere easy to find. I will then return to Eclipse where I will begin the experimental process of making this code more fitting to our company needs.

I will need to step away from my log for a few hours now. Not everything I do at this point will be logged. I intend on logging what proves to be successful. Anything I may forget to mention will be in my final code contrasted with the sourced code I referenced throughout this log. Undoubtedly, I will drop a great portion of the stock commentary in the code as it should be referenced in a manner that validates my understanding of the code. Meaning the commentary in the code will be my own. Any changes thereafter still have yet to be discovered and resolved. It is certain, some of the immediate changes will include alternating the print lines, paraphrasing the Javadoc commentary, adding log methods, and changing the conditions in which a window closes.

**May 12, 2016**

As of last night, I had mentioned stepping away from the log for a few hours to test some changes I had made to my code. After reviewing my code intensely, I found a good starting point to begin making my changes. I realize I can start by altering my CapitalizeServer.java class. I realized if there was any hope of merging UDP into my TCP program, it would require me to merge the UDPPortScanner.java with the CapitalizeClient.java class. With the CapitalizeServer.java I was able to make the necessary modifications to make this project my own while still maintaining the functionality. My next task is to make modifications to my CapitalizeClient.java. Since this is the class that contains the JFrames, I plan on tinkering with the dimensions, trying different headers, and most importantly trying to nest my UDP program within the CapitalizeClient.java class. I realize if this can’t be accomplished, I won’t beat myself up. I know our company can make good use of Junkie (servername), even if’s just the simplicity of knowing Junkie is present on the network. Nonetheless, I do want to alter CapitalizeClient.java to the best of my ability. Once again I will be stepping away from my log for a few hours to make some changes. Any changes not mentioned in this log, will be included in the log commentary, any changes made thereafter can be contrasted with the original source code.

Many hours have passed since I last logged my modifications. Undoubtedly we are near the end of our company project but, it gives me great pleasure to announce, that we were successful in nesting the UDPPortScanner.java class within the parameters of our functioning Junkie-Server. With a little help from my instructor, I was able to write UDPPortScanner.java within the same Java Project as my other two classes. I was unsure about how to go about adding the UDP program. First, we needed an instance of the UDP class which we created on its own tab. We than had to decide whether the instance is getting called from within the JFrame or the console? With much thought, we decided to slide the UDP instance just below the main method in our CapitalizeServer.java class. This allows us to run the unreliable data packets within the console first before Junkie starts requesting the IP address and various input. Luckily, the console based UDP program doesn’t seem to have a negative effect on the Capitalize portion of the program. They seem to be working in perfect harmony. With this last frontier of success, the project will come to a close.

All (3) of my classes can be accessed via GitHub. Run code in the following order:

* Package com.tenaj.junkiesoup;
  + CapitalizeServer.java (also exe. UDPPortScanner.java)
  + CapitalizeClient.java