# Testing:

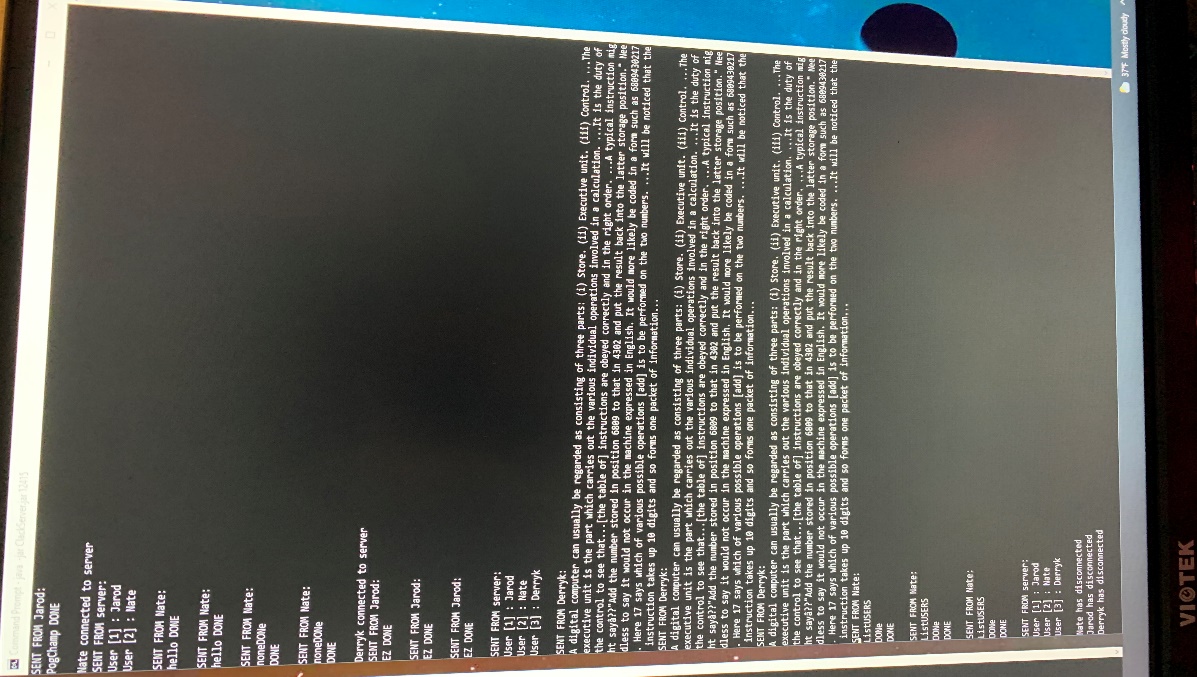
To test multiple clients, the server was run with a port 12415. Then three clients, with the names Nate, Jarod, Derryk, connected to with the following command in terminal

“java –jar ClackClient.jar [NAME]@128.153.218.196:12415”

We tested sending files, listing users(which gives the user list to only the client that requests it), communicating through messages, and disconnecting with the command “DONE”. All these tests were successful. Below are screenshots of all the command prompts across the three computers.

**\*code was changed after test cases so that “DONE” is not sent with the message**

**Server:**



**Client #1 (“Jarod”):**

A picture containing text, monitor, electronics, indoor

Description automatically generated

**Client #2 (“Nate”):**

A picture containing text

Description automatically generated

**Client #3 (“Derryk”):**

A computer screen with text

Description automatically generated with low confidence

# Questions:

**Explain why you need a separate thread for each client, and why you cannot handle all clients in the main server thread. Conceptually, why is the listener class ‘ClientSideServerListener’ different from the class ‘ServerSideClientIO’?**

The reason why you need a separate thread for each client is because the reading and printing data from the server should not have to wait on reading client data and sending that data to the server. For example, if a client was sent a message, it would have to wait for the readClientData() method before the receiveData() and printData() methods are called. For this same reasoning, we want a separate server thread for each client. This is because a client should not have to wait for in a line when communicating with the server. With a separate thread, the server is only called when it is necessary so no build up in client lines occur.

The reason the ClientSideServerListener is different from ServerSideClientIO is because ClientSideServerListener is only going to be created once for each client, whereas ServerSideClientIO is created for each client connection. When a ClientSideServerListener thread is closed, the client main thread is also closed. In the case of ServerSideClientIO however, when the ServerSideClientIO thread stops, the Server main thread continues to run.

**Explain why the broadcast() and remove() methods are synchronized. You may find it easier to answer this question after completing all programming.**

The reason the remove() and broadcast() methods are synchronized is because you do not want to broadcast to a client that has disconnected from the server. This will happen is the broadcast method is called by one client and at the exact same time another client disconnects from the server. When you synchronize the two methods, broadcast will not occur at the same time as the remove method preventing communication to disconnected clients.

**Discuss all new methods and new code in existing methods that you wrote to handle LISTUSERS**.

To handle the LISTUSERS command, I created three new methods, two in ClackServer and one in ServerSideClientIO. Inside ClackServer I created getUsers(ServerSideClientIO ssc) method that takes in the ServerSideClientIO requesting the user list and printUserList which returns a string containing all the users connected to the server. Inside the run method in ServerSideClientIO, an if statement will detect the ClackData type 0 (CONSTANT\_LISTUSERS) and call the server method getUsers with this as the parameter. getUsers(ServerSideClientIO ssc) then iterates through the serverSideClientIOList until the requesting ServerSideClientIO (given parameter) is found. It then gets a string containing all the users with the printUserList function and calls sendData(ClackData data), an overloaded method of sendData that takes in the data being send to the client, which sends the user list back to the client which then is printed when ClackClient detects type 0 (CONSTANT\_LISTUSERS) in the printData() method.