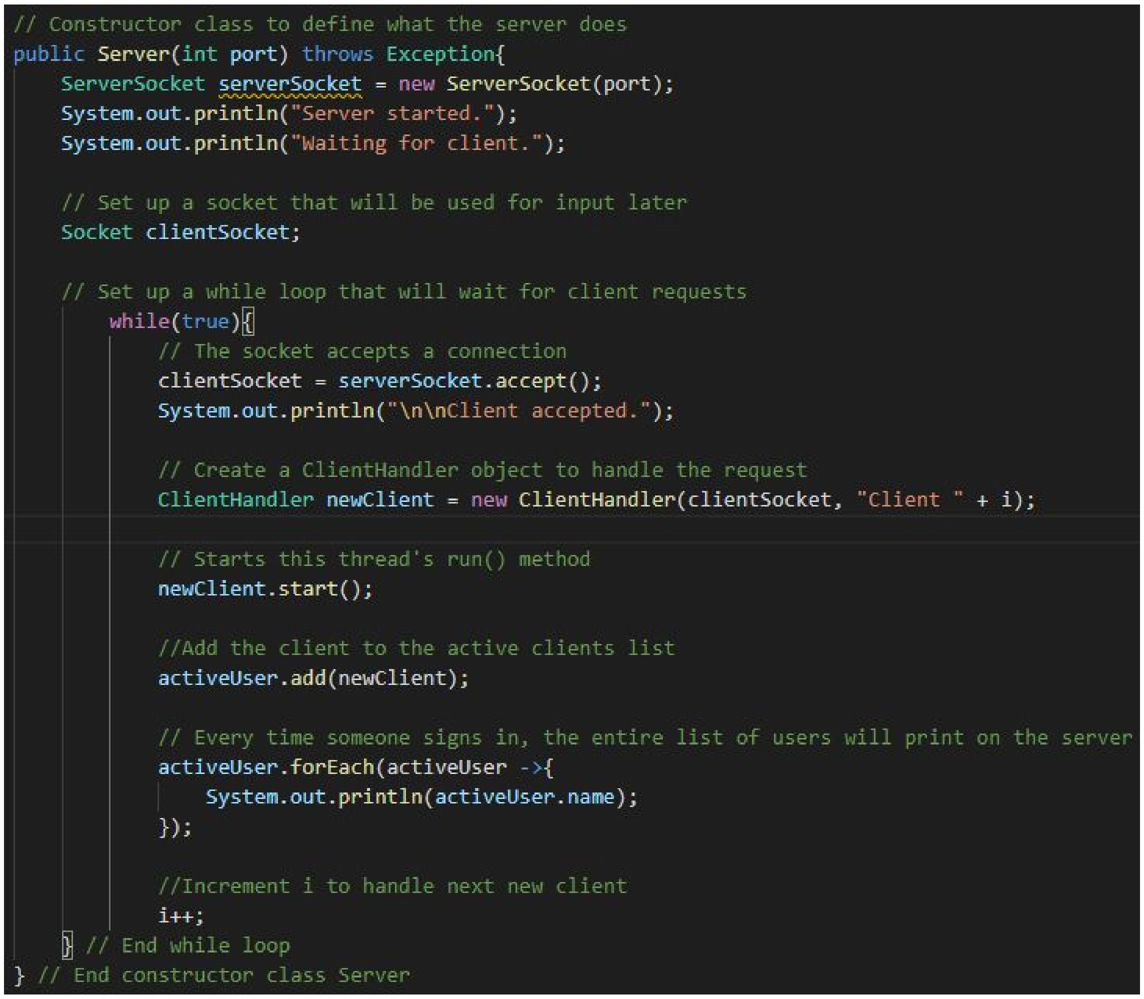
Gabriela Swanson
2542788
Due Date: 28 Feb 2020

Programming Assignment 1: REPORT

Server.java
The Server file has two classes: *Server* and *ClientHandler*. The main goal was to take advantage of a) socket programming because it allows programs to execute across multiple computers and b) threading to handle multiple clients so they could all be online and talk with each other instead of having the server dedicated to just one client. I used localhost and port 1775 for the server connection. The server is always listening for client connections, and once a connection is established, it listens for client input.

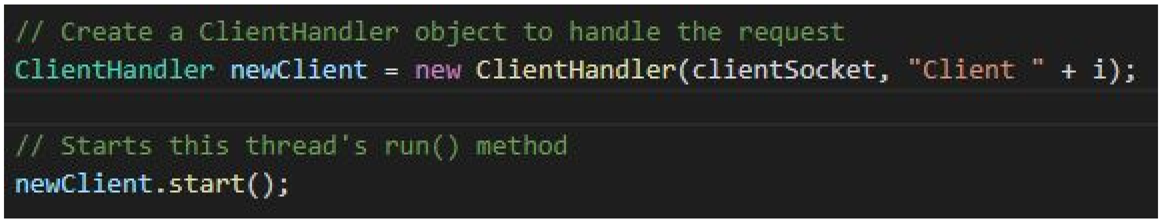
The Server Class
The server class establishes connection, obtains streams, creates the ClientHandler object, and invokes the start() method. I used an ArrayList to keep track of the clients as they logged in and as they left the chat service. The constructor uses sockets to accept the connection since they are the endpoints of a two-way communication link between two programs, enable the communication between the server and the client (by invoking the ClientHandler class and the start() method), and add new clients to the ArrayList.



```
// Constructor class to define what the server does
public Server(int port) throws Exception{
    ServerSocket serverSocket = new ServerSocket(port);
    System.out.println("Server started.");
    System.out.println("Waiting for client.");

    // Set up a socket that will be used for input later
    Socket clientSocket;

    // Set up a while loop that will wait for client requests
        while(true){
            // The socket accepts a connection
            clientSocket = serverSocket.accept();
            System.out.println("\n\nClient accepted.");

            // Create a ClientHandler object to handle the request
            ClientHandler newClient = new ClientHandler(clientSocket, "Client " + i);

            // Starts this thread's run() method
            newClient.start();

            //Add the client to the active clients list
            activeUser.add(newClient);

            // Every time someone signs in, the entire list of users will print on the server
            activeUser.forEach(activeUser ->{
                System.out.println(activeUser.name);
            });

            //Increment i to handle next new client
            i++;
    } // End while loop
} // End constructor class Server
```

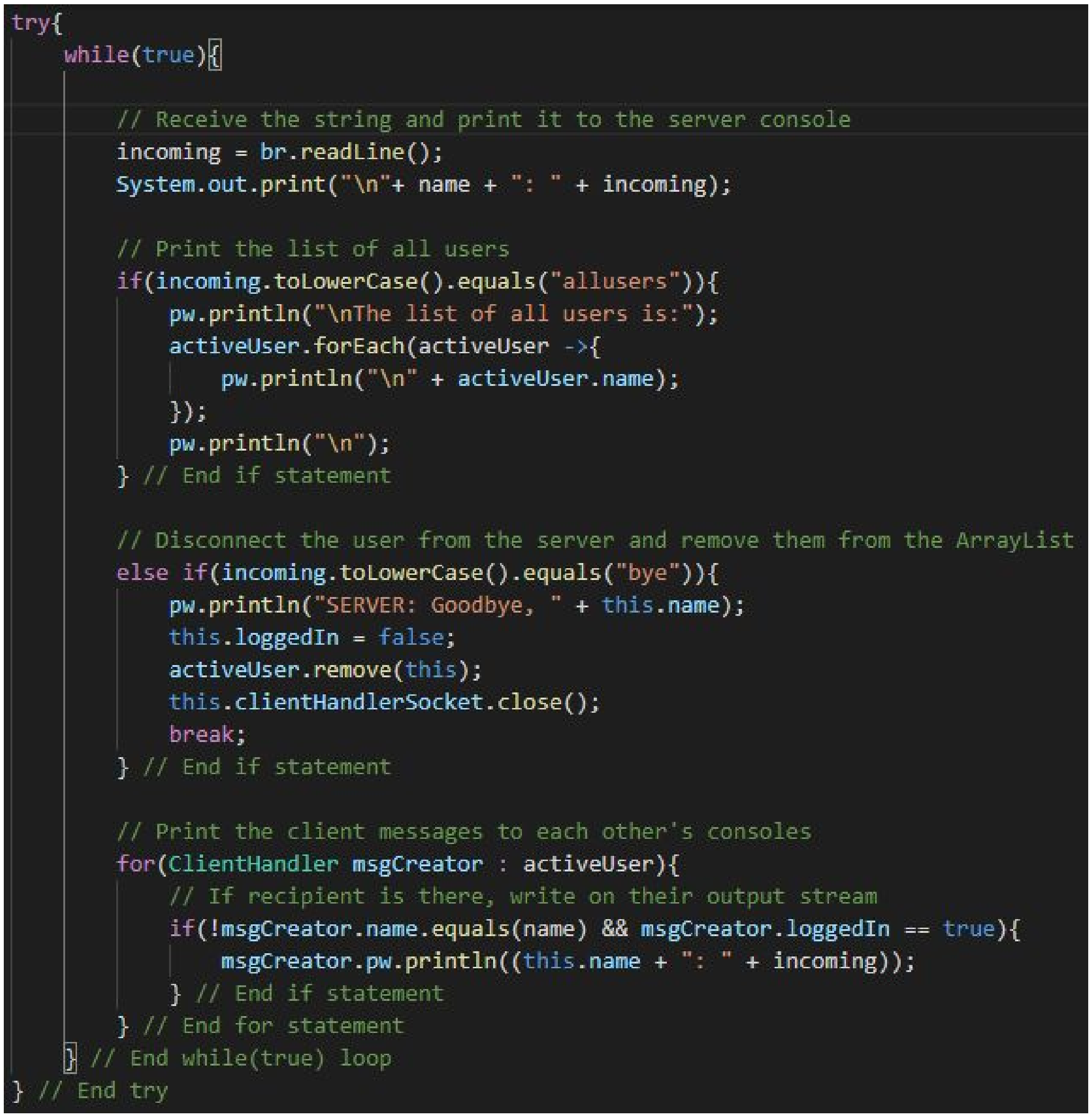
The ClientHandler Class

Since this class extends Thread, its objects are able to assume all Thread properties. Every time a new client signs into the server, a new socket is created for them, and a dedicated thread handles the client's requests. This code, though it's in the Server class, exemplifies this.



```
// Create a ClientHandler object to handle the request
ClientHandler newClient = new ClientHandler(clientSocket, "Client " + i);

// Starts this thread's run() method
newClient.start();
```

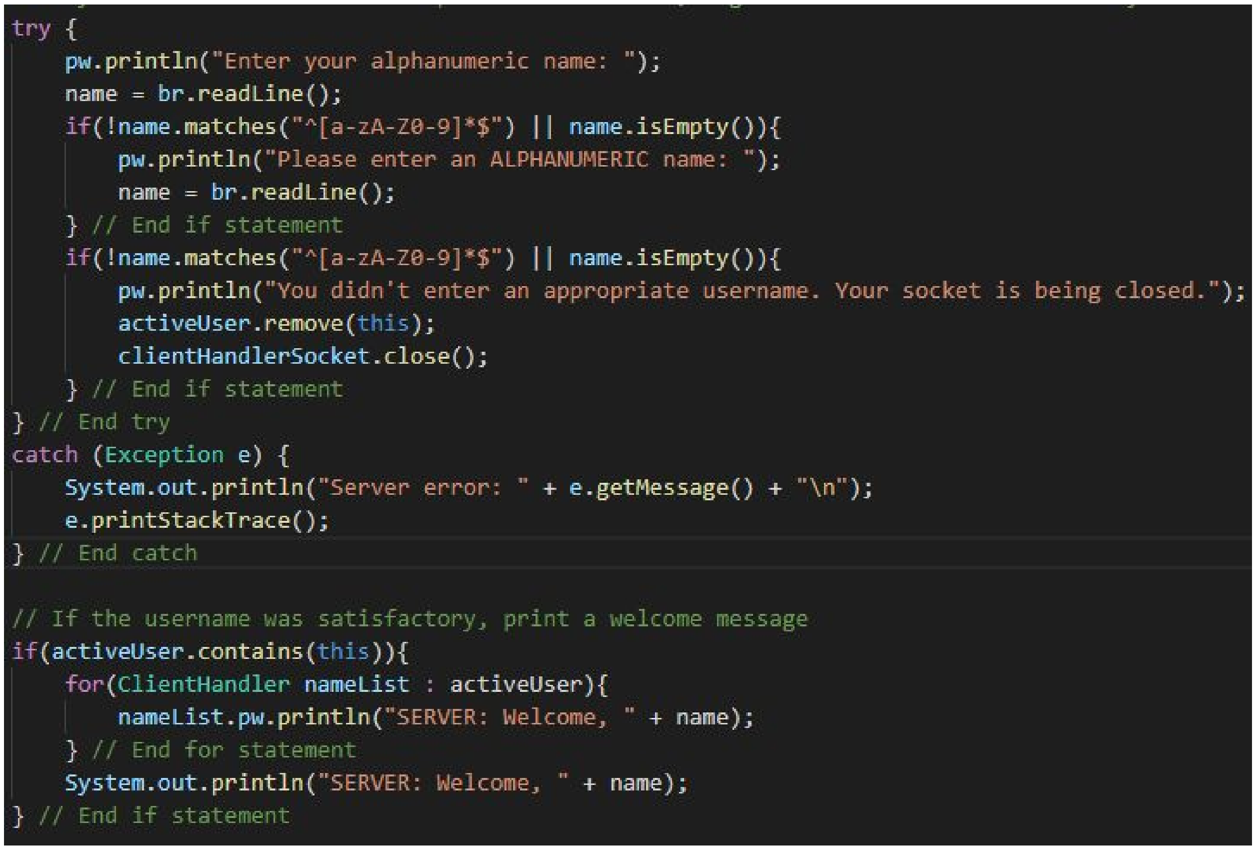
The main portion of the ClientHandler class is the run() block. The main thread runs in a while loop as it listens for connections. Strings are printed to all active consoles using a PrintWriter to handle output to the consoles and a BufferedReader to handle reading input streams.



```
try{
    while(true){

        // Receive the string and print it to the server console
        incoming = br.readLine();
        System.out.print("\n"+ name + ": " + incoming);

        // Print the list of all users
        if(incoming.toLowerCase().equals("allusers")){
            pw.println("\nThe list of all users is:");
            activeUser.forEach(activeUser ->{
                pw.println("\n" + activeUser.name);
            });
            pw.println("\n");
        } // End if statement

        // Disconnect the user from the server and remove them from the ArrayList
        else if(incoming.toLowerCase().equals("bye")){
            pw.println("SERVER: Goodbye, " + this.name);
            this.loggedIn = false;
            activeUser.remove(this);
            this.clientHandlerSocket.close();
            break;
        } // End if statement

        // Print the client messages to each other's consoles
        for(ClientHandler msgCreator : activeUser){
            // If recipient is there, write on their output stream
            if(!msgCreator.name.equals(name) && msgCreator.loggedIn == true){
                msgCreator.pw.println((this.name + ": " + incoming));
            } // End if statement
        } // End for statement
    } // End while(true) loop
} // End try
```

A series of try-catch blocks with embedded if statements ensure incoming strings meet requirements and action is taken on specific commands:

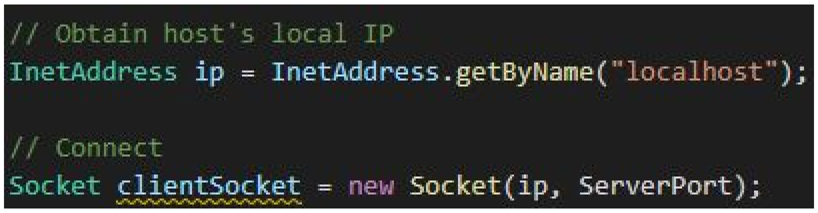
- Client username *must* be alphanumeric,
- Client is welcomed if their username is alphanumeric
- Clients' requested information (e.g., allusers) is printed to all the consoles,
- The client is disconnected when they're ready (e.g., bye)



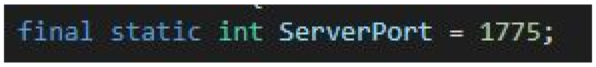
```
try {
    pw.println("Enter your alphanumeric name: ");
    name = br.readLine();
    if(!name.matches("^[a-zA-Z0-9]*$") || name.isEmpty()){
        pw.println("Please enter an ALPHANUMERIC name: ");
        name = br.readLine();
    } // End if statement
    if(!name.matches("^[a-zA-Z0-9]*$") || name.isEmpty()){
        pw.println("You didn't enter an appropriate username. Your socket is being closed.");
        activeUser.remove(this);
        clientHandlerSocket.close();
    } // End if statement
} // End try
catch (Exception e) {
    System.out.println("Server error: " + e.getMessage() + "\n");
    e.printStackTrace();
} // End catch

// If the username was satisfactory, print a welcome message
if(activeUser.contains(this)){
    for(ClientHandler nameList : activeUser){
        nameList.pw.println("SERVER: Welcome, " + name);
    } // End for statement
    System.out.println("SERVER: Welcome, " + name);
} // End if statement
```

Client.java

The client side program is responsible for establishing a socket connection and communicating. The clients connected using localhost and port 1775.

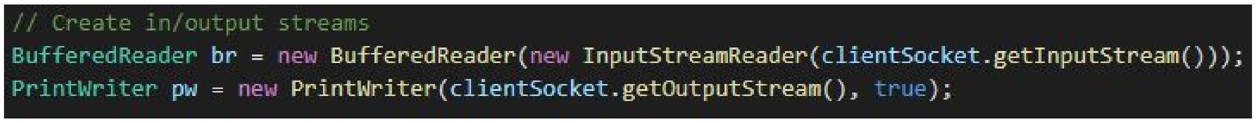


```
// Obtain host's local IP
InetAddress ip = InetAddress.getByName("localhost");

// Connect
Socket clientSocket = new Socket(ip, ServerPort);
```



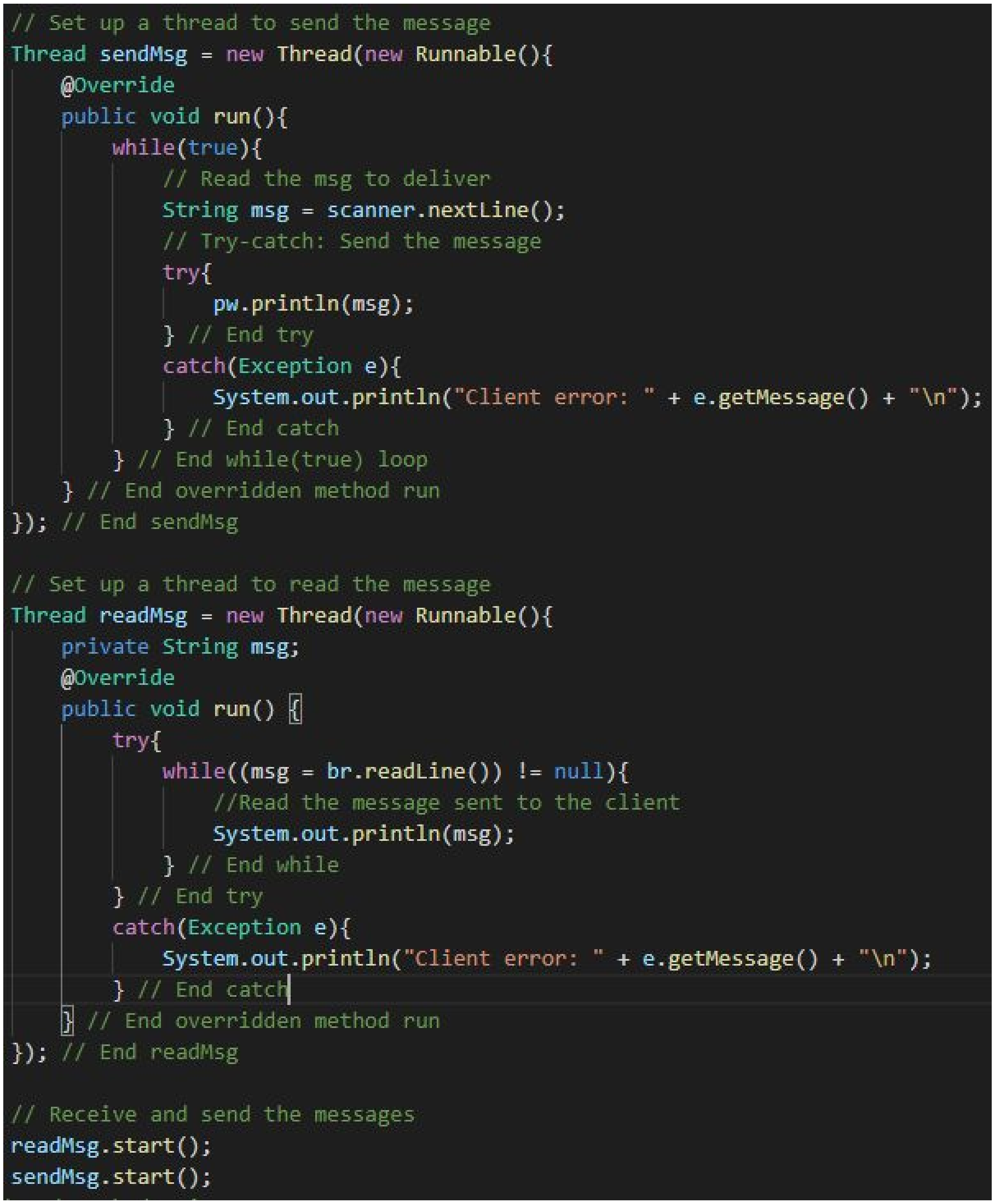
```
final static int ServerPort = 1775;
```

A BufferedReader and PrintWriter handled input and output.



```
// Create in/output streams
BufferedReader br = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
PrintWriter pw = new PrintWriter(clientSocket.getOutputStream(), true);
```

I took advantage of threads' ability for several simultaneous actions again in my sendMsg and readMsg methods. For both methods, I overrode the run() method send/receive messages. Once the threads were set-up, I started both methods so the client was always listening for other client input.

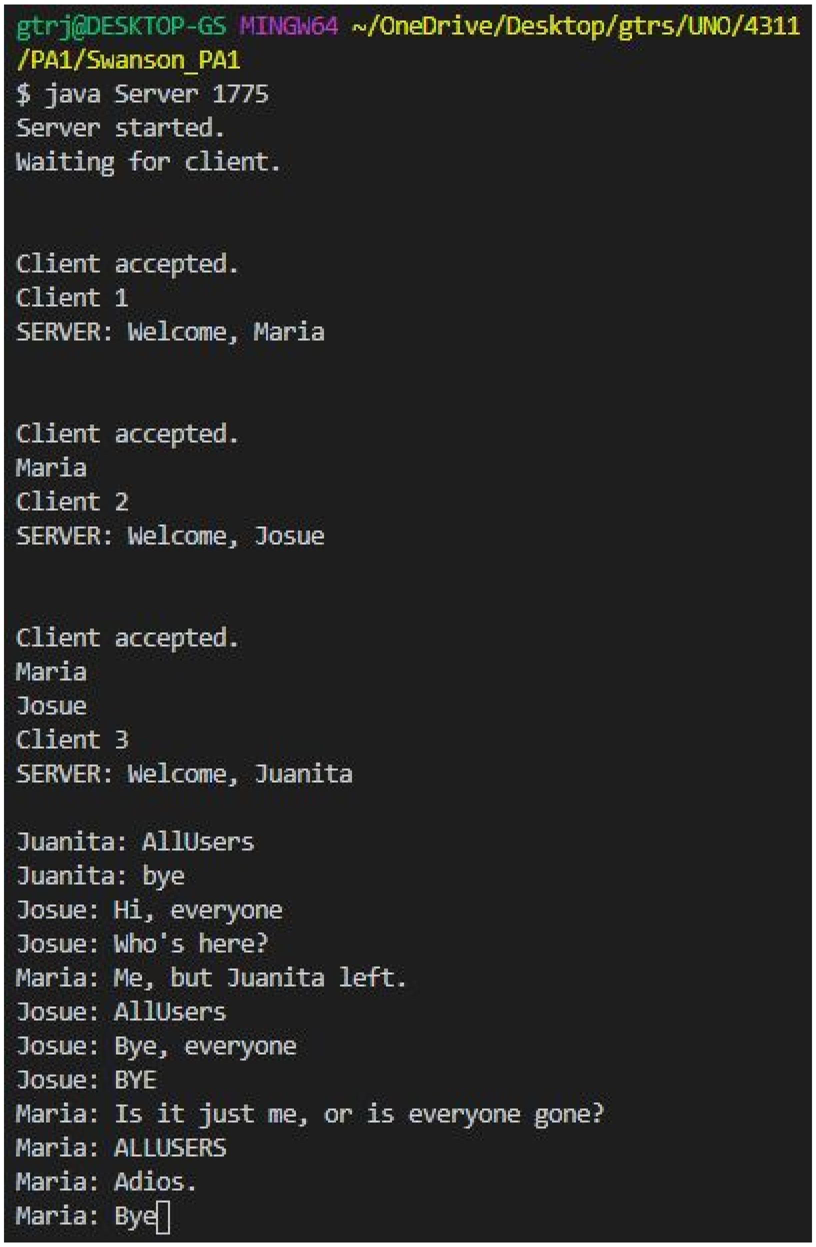


```java
// Set up a thread to send the message
Thread sendMsg = new Thread(new Runnable(){
    @Override
    public void run(){
        while(true){
            // Read the msg to deliver
            String msg = scanner.nextLine();
            // Try-catch: Send the message
            try{
                pw.println(msg);
            } // End try
            catch(Exception e){
                System.out.println("Client error: " + e.getMessage() + "\n");
            } // End catch
        } // End while(true) loop
    } // End overridden method run
}); // End sendMsg

// Set up a thread to read the message
Thread readMsg = new Thread(new Runnable(){
    private String msg;
    @Override
    public void run() {
        try{
            while((msg = br.readLine()) != null){
                //Read the message sent to the client
                System.out.println(msg);
            } // End while
        } // End try
        catch(Exception e){
            System.out.println("Client error: " + e.getMessage() + "\n");
        } // End catch
    } // End overridden method run
}); // End readMsg

// Receive and send the messages
readMsg.start();
sendMsg.start();
```

REFERENCES

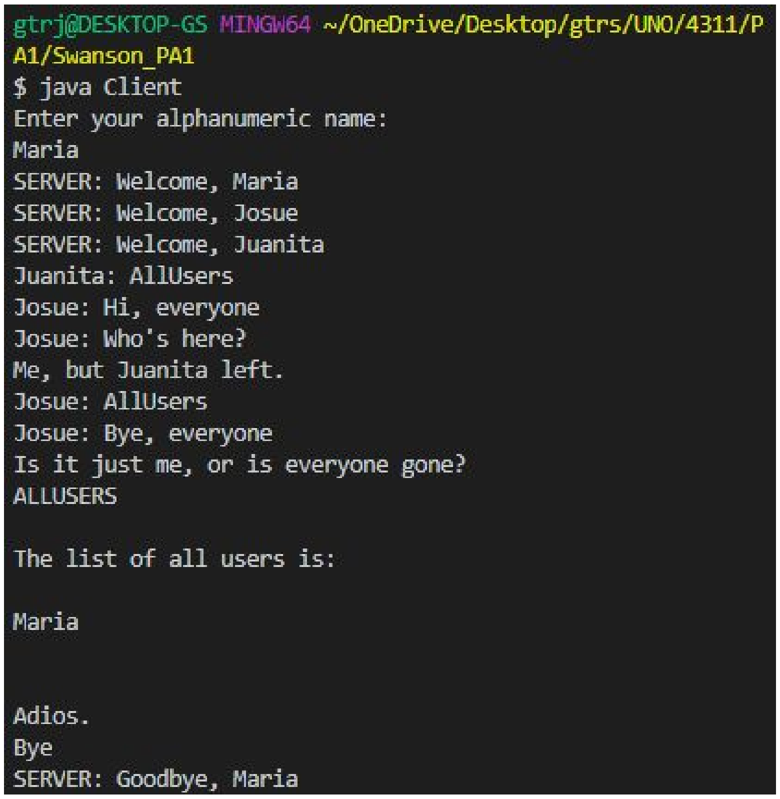
```
*Baeldung's "A Guide to Java Sockets"
*Geeks for Geeks' "Introducing Threads in Socket Programming"
*Geeks for Geeks' "Multi-Threaded Chat Application in Java"
*Java8 API (various searches)
*Sheldon Guillory, classmate
*Stack Overflow (various searches)
```

SERVER



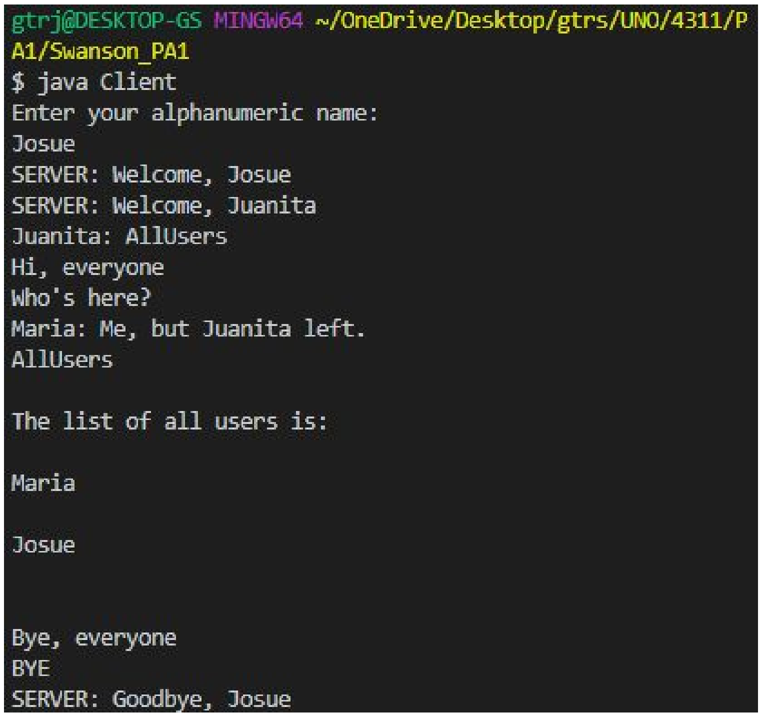
```
gtrj@DESKTOP-GS MINGW64 ~/OneDrive/Desktop/gtrs/UNO/4311
/PA1/Swanson_PA1
$ java Server 1775
Server started.
Waiting for client.


Client accepted.
Client 1
SERVER: Welcome, Maria


Client accepted.
Maria
Client 2
SERVER: Welcome, Josue


Client accepted.
Maria
Josue
Client 3
SERVER: Welcome, Juanita

Juanita: AllUsers
Juanita: bye
Josue: Hi, everyone
Josue: Who's here?
Maria: Me, but Juanita left.
Josue: AllUsers
Josue: Bye, everyone
Josue: BYE
Maria: Is it just me, or is everyone gone?
Maria: ALLUSERS
Maria: Adios.
Maria: Bye
```

CLIENT 1



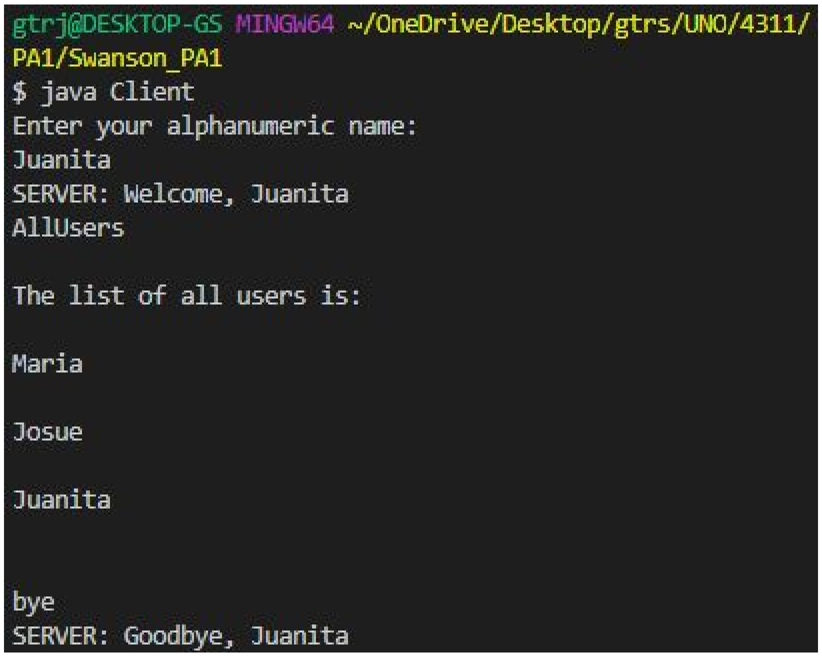
```
gtrj@DESKTOP-GS MINGW64 ~/OneDrive/Desktop/gtrs/UNO/4311/P
A1/Swanson_PA1
$ java Client
Enter your alphanumeric name:
Maria
SERVER: Welcome, Maria
SERVER: Welcome, Josue
SERVER: Welcome, Juanita
Juanita: AllUsers
Josue: Hi, everyone
Josue: Who's here?
Me, but Juanita left.
Josue: AllUsers
Josue: Bye, everyone
Is it just me, or is everyone gone?
ALLUSERS

The list of all users is:

Maria


Adios.
Bye
SERVER: Goodbye, Maria
```

CLIENT 2



```
gtrj@DESKTOP-GS MINGW64 ~/OneDrive/Desktop/gtrs/UNO/4311/P
A1/Swanson_PA1
$ java Client
Enter your alphanumeric name:
Josue
SERVER: Welcome, Josue
SERVER: Welcome, Juanita
Juanita: AllUsers
Hi, everyone
Who's here?
Maria: Me, but Juanita left.
AllUsers

The list of all users is:

Maria

Josue


Bye, everyone
BYE
SERVER: Goodbye, Josue
```

CLIENT 3



```
gtrj@DESKTOP-GS MINGW64 ~/OneDrive/Desktop/gtrs/UNO/4311/
PA1/Swanson_PA1
$ java Client
Enter your alphanumeric name:
Juanita
SERVER: Welcome, Juanita
AllUsers

The list of all users is:

Maria

Josue

Juanita


bye
SERVER: Goodbye, Juanita
```