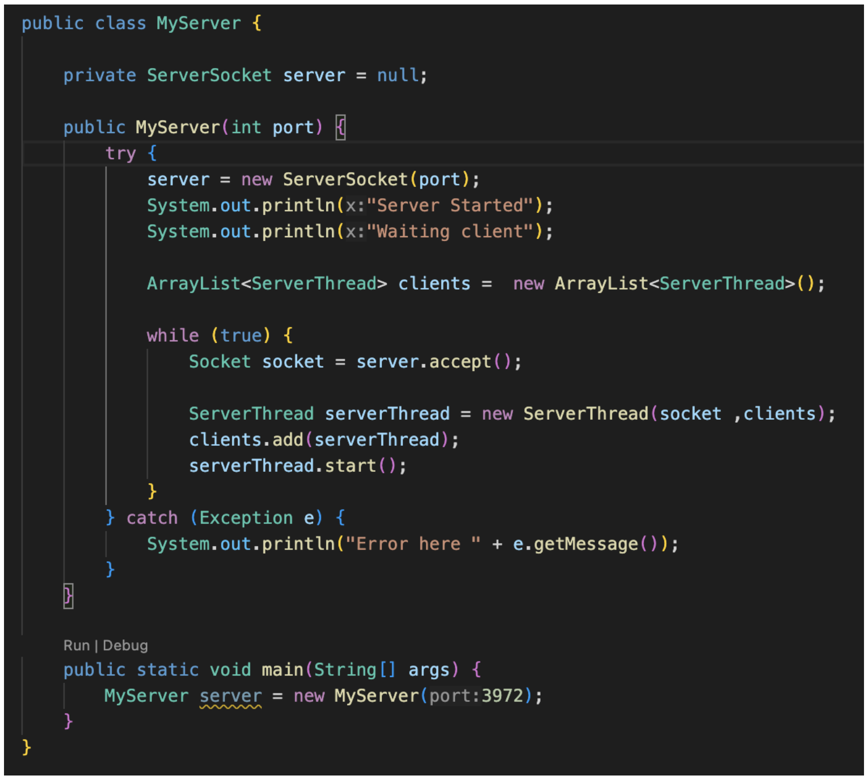
Jumana Suleiman
CSCI 4311

Programming Assignment 1 Report

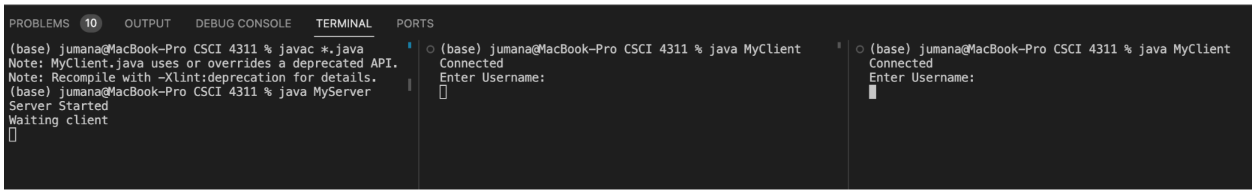
MyServer.java:

This file is the server file. This file includes the socket programming technique. The server class establishes a connection between the clients. It prints out 2 messages indicating that the server has started and that it is now waiting for the clients to join. This class also includes the ArrayList I created to keep track of all the clients connecting. Then I created an infinite loop that accepts client connections that way more clients can join continuously. I use the server port number 3972 for the connection.



```java
public class MyServer {

    private ServerSocket server = null;

    public MyServer(int port) {
        try {
            server = new ServerSocket(port);
            System.out.println(x:"Server Started");
            System.out.println(x:"Waiting client");

            ArrayList<ServerThread> clients =  new ArrayList<ServerThread>();

            while (true) {
                Socket socket = server.accept();

                ServerThread serverThread = new ServerThread(socket ,clients);
                clients.add(serverThread);
                serverThread.start();
            }
        } catch (Exception e) {
            System.out.println("Error here " + e.getMessage());
        }
    }

    Run | Debug
    public static void main(String[] args) {
        MyServer server = new MyServer(port:3972);
    }
}
```

Here is what happens when the files are compiled. The leftmost terminal is the MyServer class. As you can see, it printed out the 2 messages and is now waiting for a connection.



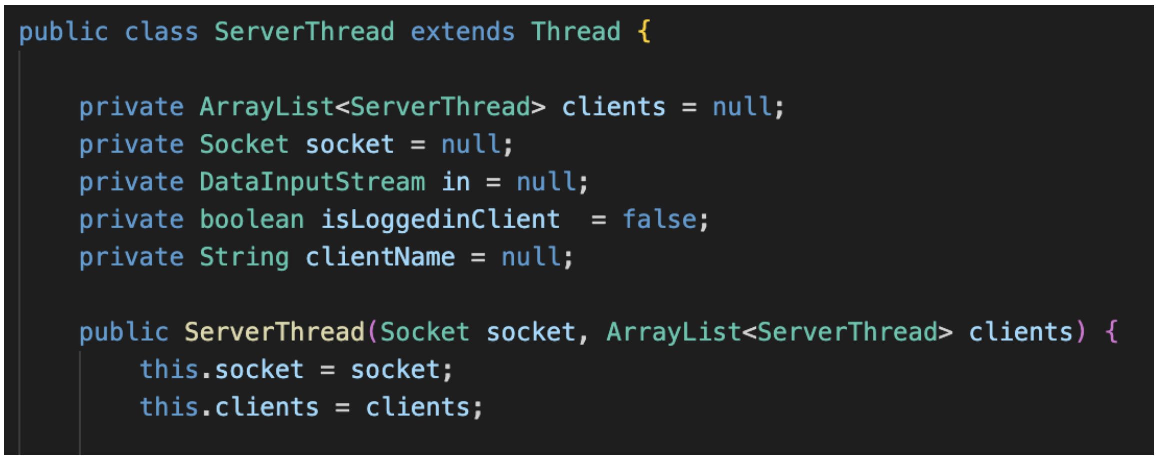
```
(base) jumana@MacBook-Pro CSCI 4311 % javac *.java
Note: MyClient.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.
(base) jumana@MacBook-Pro CSCI 4311 % java MyServer
Server Started
Waiting client
```

```
(base) jumana@MacBook-Pro CSCI 4311 % java MyClient
Connected
Enter Username:
```

```
(base) jumana@MacBook-Pro CSCI 4311 % java MyClient
Connected
Enter Username:
```

ServerThread.java:

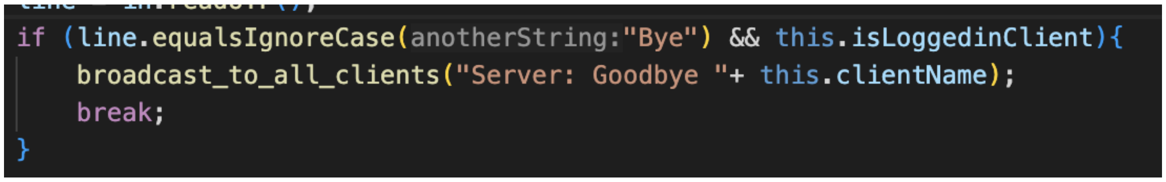
This is like an extension of the server class. I first had it in the same file but it started to get unorganized so I separated them. This class extends ‘Thread”, which allows the objects of the file to use the Thread properties. In the constructor, once a user ‘logs in’ a new socket is created as well as they are added to the ArrayList.



```java
public class ServerThread extends Thread {

    private ArrayList<ServerThread> clients = null;
    private Socket socket = null;
    private DataInputStream in = null;
    private boolean isLoggedinClient  = false;
    private String clientName = null;

    public ServerThread(Socket socket, ArrayList<ServerThread> clients) {
        this.socket = socket;
        this.clients = clients;
```

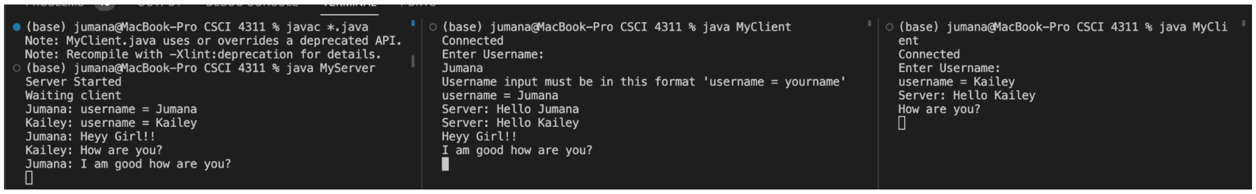
Then I created a run method that sets up the input and output stream. It prompts the user to enter a username. Then another infinite loop that continuously reads the clients' messages. Then there are several different actions that are taken based on the users' messages.

If the user says any form of “bye” where the case does not matter, the server gives a goodbye message to them and other clients and closes their socket.



```java
if (line.equalsIgnoreCase(anotherString:"Bye") && this.isLoggedinClient){
    broadcast_to_all_clients("Server: Goodbye "+ this.clientName);
    break;
}
```

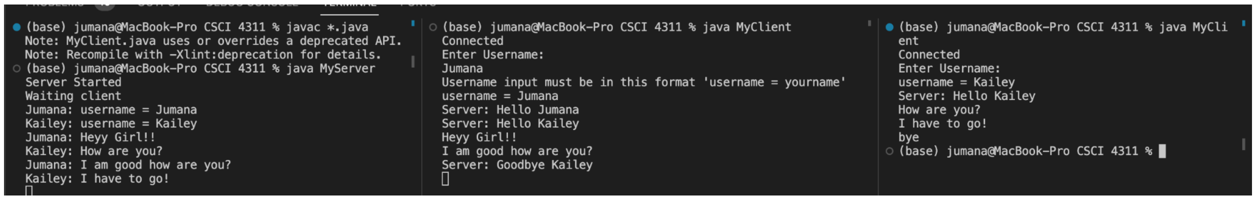
Terminal Example:



```
● (base) jumana@MacBook-Pro CSCI 4311 % javac *.java
  Note: MyClient.java uses or overrides a deprecated API.
  Note: Recompile with -Xlint:deprecation for details.
○ (base) jumana@MacBook-Pro CSCI 4311 % java MyServer
  Server Started
  Waiting client
  Jumana: username = Jumana
  Kailey: username = Kailey
  Jumana: Heyy Girl!!
  Kailey: How are you?
  Jumana: I am good how are you?
```

```
○ (base) jumana@MacBook-Pro CSCI 4311 % java MyClient
  Connected
  Enter Username:
  Jumana
  Username input must be in this format 'username = yourname'
  username = Jumana
  Server: Hello Jumana
  Server: Hello Kailey
  Heyy Girl!!
  I am good how are you?
```

```
○ (base) jumana@MacBook-Pro CSCI 4311 % java MyCli
  ent
  Connected
  Enter Username:
  username = Kailey
  Server: Hello Kailey
  How are you?
```

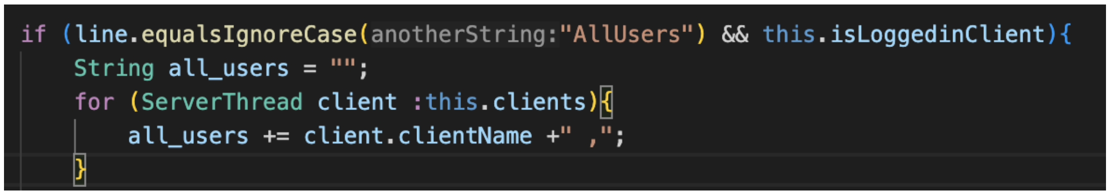


```
● (base) jumana@MacBook-Pro CSCI 4311 % javac *.java
  Note: MyClient.java uses or overrides a deprecated API.
  Note: Recompile with -Xlint:deprecation for details.
○ (base) jumana@MacBook-Pro CSCI 4311 % java MyServer
  Server Started
  Waiting client
  Jumana: username = Jumana
  Kailey: username = Kailey
  Jumana: Heyy Girl!!
  Kailey: How are you?
  Jumana: I am good how are you?
  Kailey: I have to go!
```

```
○ (base) jumana@MacBook-Pro CSCI 4311 % java MyClient
  Connected
  Enter Username:
  Jumana
  Username input must be in this format 'username = yourname'
  username = Jumana
  Server: Hello Jumana
  Server: Hello Kailey
  Heyy Girl!!
  I am good how are you?
  Server: Goodbye Kailey
```

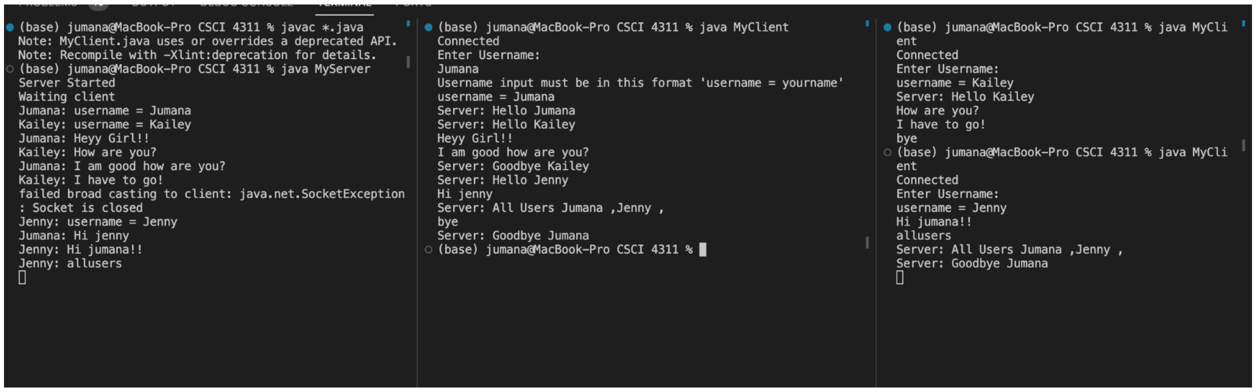
```
● (base) jumana@MacBook-Pro CSCI 4311 % java MyCli
  ent
  Connected
  Enter Username:
  username = Kailey
  Server: Hello Kailey
  How are you?
  I have to go!
  bye
○ (base) jumana@MacBook-Pro CSCI 4311 %
```

If the user says “allusers”, again case does not matter, then it will give the client and all the other clients a list of all the users on the server.



```
if (line.equalsIgnoreCase(anotherString:"AllUsers") && this.isLoggedinClient){
    String all_users = "";
    for (ServerThread client :this.clients){
        all_users += client.clientName +"  ,";
    }
```

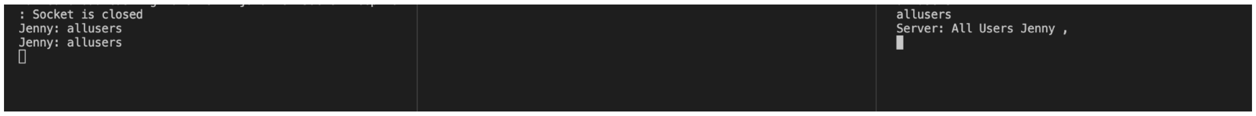
Terminal Example:



```
● (base) jumana@MacBook-Pro CSCI 4311 % javac *.java
  Note: MyClient.java uses or overrides a deprecated API.
  Note: Recompile with -Xlint:deprecation for details.
○ (base) jumana@MacBook-Pro CSCI 4311 % java MyServer
  Server Started
  Waiting client
  Jumana: username = Jumana
  Kailey: username = Kailey
  Jumana: Heyy Girl!!
  Kailey: How are you?
  Jumana: I am good how are you?
  Kailey: I have to go!
  failed broad casting to client: java.net.SocketException
  : Socket is closed
  Jenny: username = Jenny
  Jumana: Hi jenny
  Jenny: Hi jumana!!
  Jenny: allusers
```

```
● (base) jumana@MacBook-Pro CSCI 4311 % java MyClient
  Connected
  Enter Username:
  Jumana
  Username input must be in this format 'username = yourname'
  username = Jumana
  Server: Hello Jumana
  Server: Hello Kailey
  Heyy Girl!!
  I am good how are you?
  Server: Goodbye Kailey
  Server: Hello Jenny
  Hi jenny
  Server: All Users Jumana ,Jenny ,
  bye
  Server: Goodbye Jumana
○ (base) jumana@MacBook-Pro CSCI 4311 % 
```

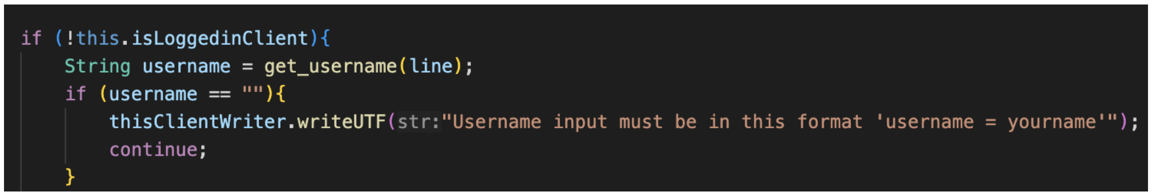
```
● (base) jumana@MacBook-Pro CSCI 4311 % java MyCli
  ent
  Connected
  Enter Username:
  username = Kailey
  Server: Hello Kailey
  How are you?
  I have to go!
  bye
○ (base) jumana@MacBook-Pro CSCI 4311 % java MyCli
  ent
  Connected
  Enter Username:
  username = Jenny
  Hi jumana!!
  allusers
  Server: All Users Jumana ,Jenny ,
  Server: Goodbye Jumana
```



```
  : Socket is closed
  Jenny: allusers
  Jenny: allusers
```

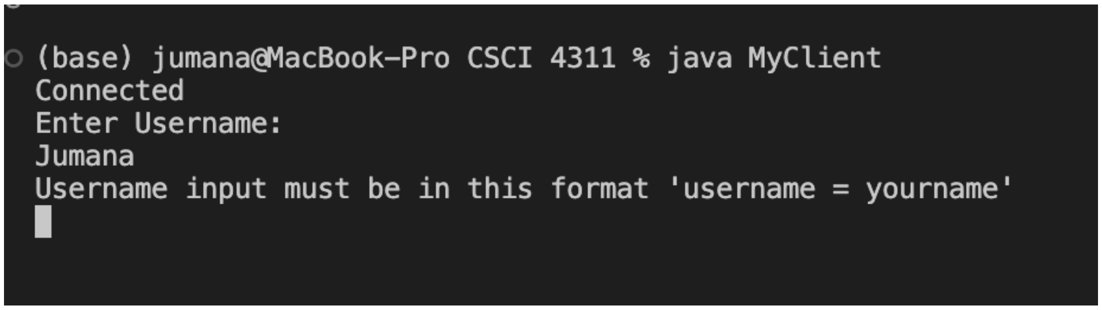
```
  allusers
  Server: All Users Jenny ,
```

The next lines check for the username's correctness. The user needs to enter a username in a specific format like the one given in the document. They have to enter "username = 'their user' " and if they do not then it will give them an error message in which after they can try again.



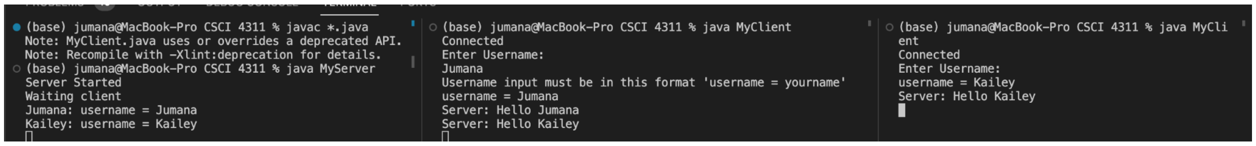
```
if (!this.isLoggedinClient){
    String username = get_username(line);
    if (username == ""){
        thisClientWriter.writeUTF(str:"Username input must be in this format 'username = yourname'");
        continue;
    }
```

Terminal example:



```
○ (base) jumana@MacBook-Pro CSCI 4311 % java MyClient
  Connected
  Enter Username:
  Jumana
  Username input must be in this format 'username = yourname'
```

Once the user is entered correctly, it displays a Hello message to the user and the other active clients:

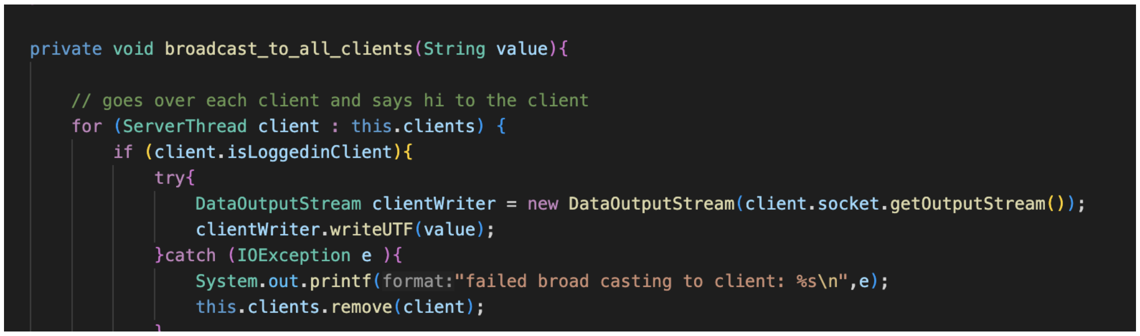


```
● (base) jumana@MacBook-Pro CSCI 4311 % javac *.java
  Note: MyClient.java uses or overrides a deprecated API.
  Note: Recompile with -Xlint:deprecation for details.
○ (base) jumana@MacBook-Pro CSCI 4311 % java MyServer
  Server Started
  Waiting client
  Jumana: username = Jumana
  Kailey: username = Kailey
```

```
○ (base) jumana@MacBook-Pro CSCI 4311 % java MyClient
  Connected
  Enter Username:
  Jumana
  Username input must be in this format 'username = yourname'
  username = Jumana
  Server: Hello Jumana
  Server: Hello Kailey
```

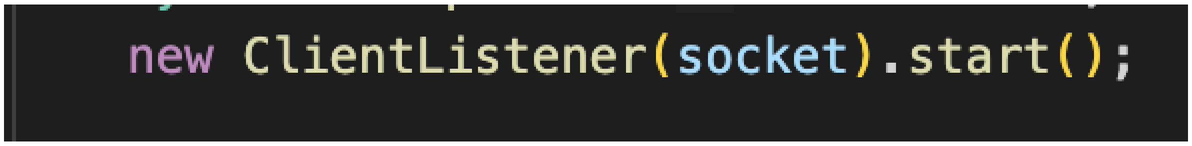
```
○ (base) jumana@MacBook-Pro CSCI 4311 % java MyCli
  ent
  Connected
  Enter Username:
  username = Kailey
  Server: Hello Kailey
```

Lastly, I created a helper method to broadcast the messages to the server.



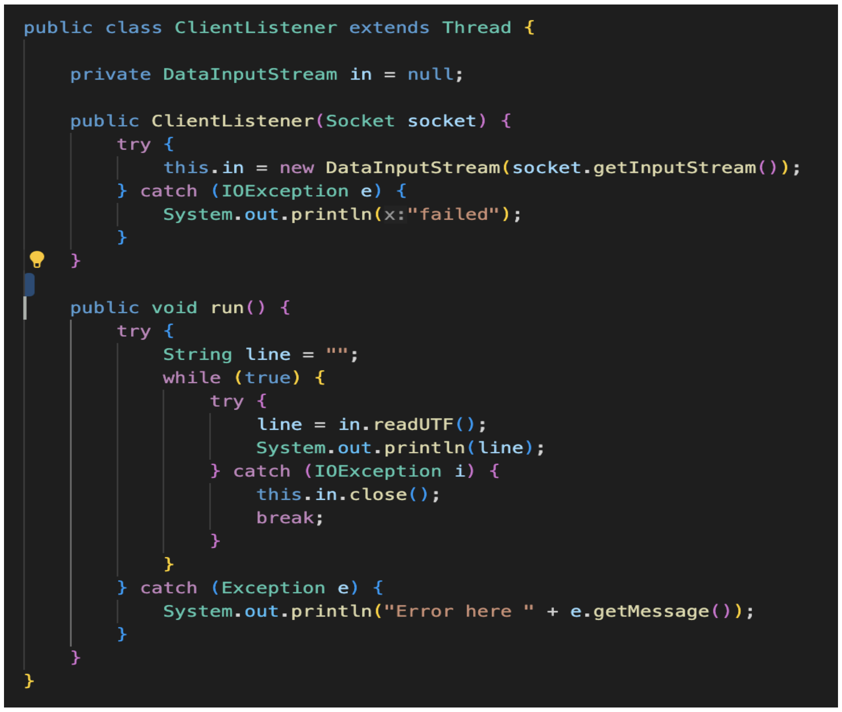
```
private void broadcast_to_all_clients(String value){

    // goes over each client and says hi to the client
    for (ServerThread client : this.clients) {
        if (client.isLoggedinClient){
            try{
                DataOutputStream clientWriter = new DataOutputStream(client.socket.getOutputStream());
                clientWriter.writeUTF(value);
            }catch (IOException e ){
                System.out.printf(format:"failed broad casting to client: %s\n",e);
                this.clients.remove(client);
```

MyClient.java:
This is pretty much the exact copy of the code that was provided in class. The only thing I changed was the variable names to make them more documentational and I started the ClientListener file inside of it. As well as the goodbye message was changed to "Bye"



```
    new ClientListener(socket).start();
```

ClientListener.java:
This file just represents a thread that listens to any messages that are sent from the server from the socket connections and prints them to the console. It just receives updates and responses from the server and that is its only purpose.



```
public class ClientListener extends Thread {

    private DataInputStream in = null;

    public ClientListener(Socket socket) {
        try {
            this.in = new DataInputStream(socket.getInputStream());
        } catch (IOException e) {
            System.out.println(x:"failed");
        }
    }

    public void run() {
        try {
            String line = "";
            while (true) {
                try {
                    line = in.readUTF();
                    System.out.println(line);
                } catch (IOException i) {
                    this.in.close();
                    break;
                }
            }
        } catch (Exception e) {
            System.out.println("Error here " + e.getMessage());
        }
    }
}
```