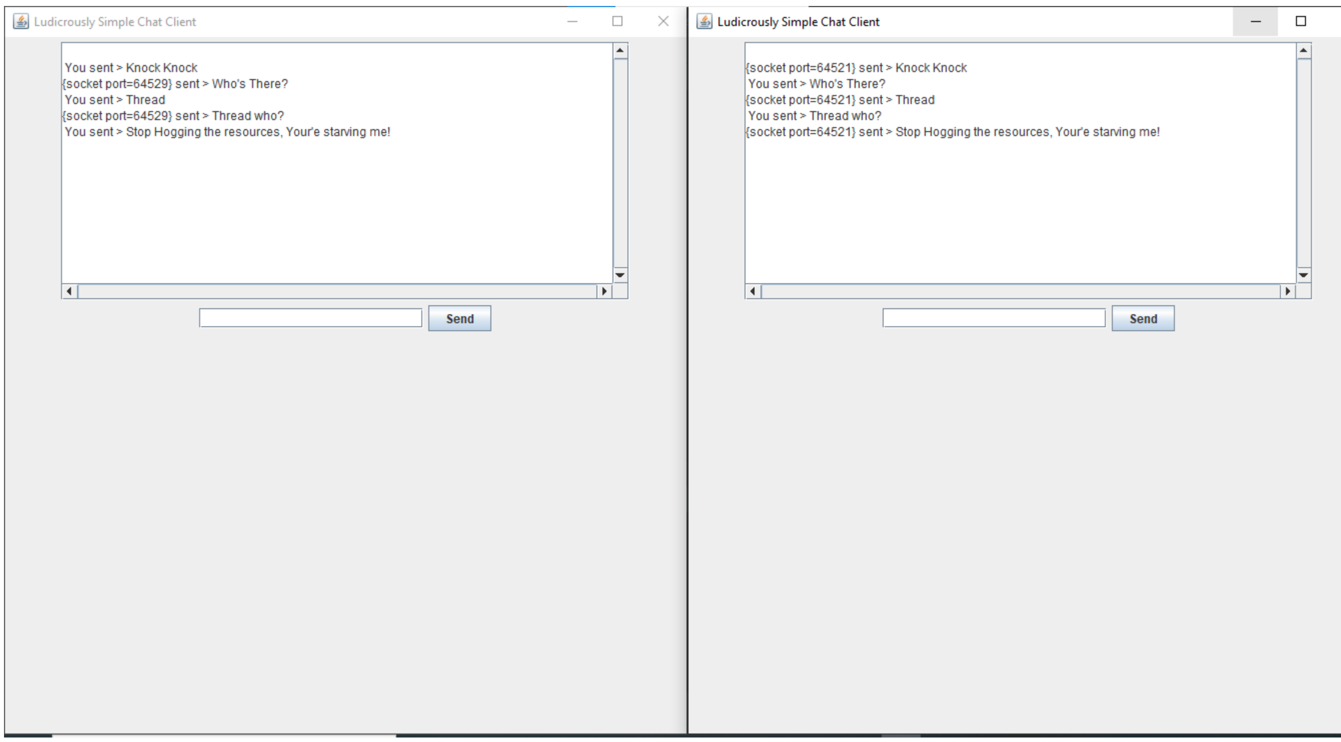
Homework Assignment 1: Chat Client

Michael Edelman

September 13, 2021

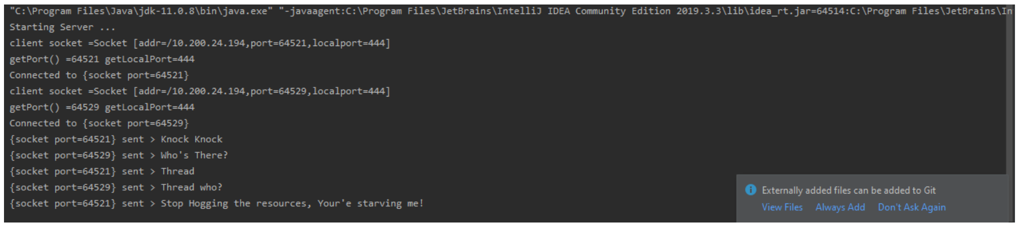
1 Chat Clients


Ludicrously Simple Chat Client
Send
Ludicrously Simple Chat Client
Send

```
You sent > Knock Knock
{socket port=64529} sent > Who's There?
You sent > Thread
{socket port=64529} sent > Thread who?
You sent > Stop Hogging the resources, Your'e starving me!
```

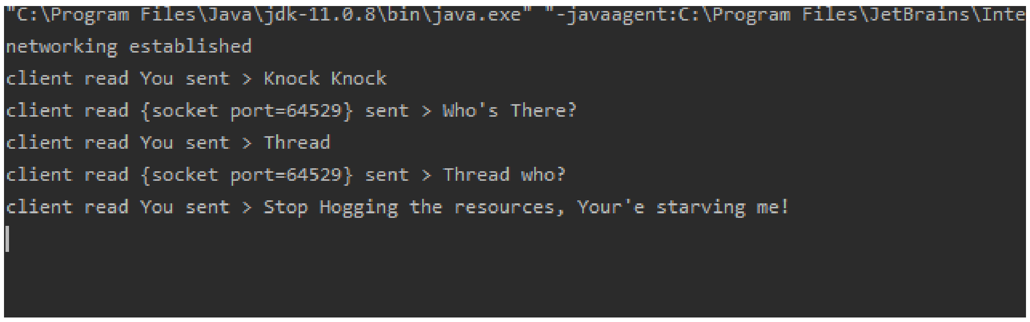
```
{socket port=64521} sent > Knock Knock
You sent > Who's There?
{socket port=64521} sent > Thread
You sent > Thread who?
{socket port=64521} sent > Stop Hogging the resources, Your'e starving me!
```

2 Text Emitted By Server and Clients



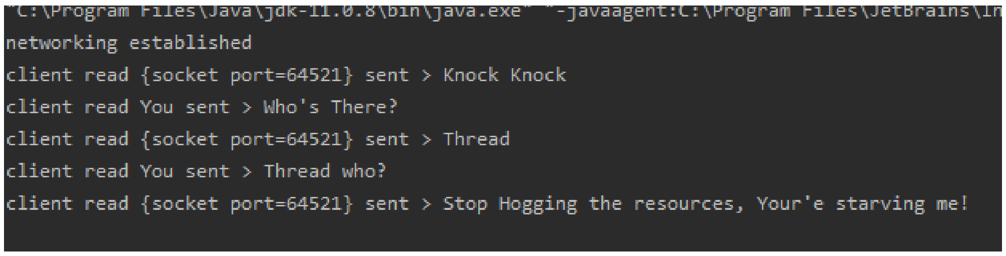
```
"C:\Program Files\Java\jdk-11.0.8\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2019.3.3\lib\idea_rt.jar=64514:C:\Program Files\JetBrains\In
Starting Server ...
client socket =Socket [addr=/10.200.24.194,port=64521,localport=444]
getPort() =64521 getLocalPort=444
Connected to {socket port=64521}
client socket =Socket [addr=/10.200.24.194,port=64529,localport=444]
getPort() =64529 getLocalPort=444
Connected to {socket port=64529}
{socket port=64521} sent > Knock Knock
{socket port=64529} sent > Who's There?
{socket port=64521} sent > Thread
{socket port=64529} sent > Thread who?
{socket port=64521} sent > Stop Hogging the resources, Your'e starving me!
```

Figure 1: Server Window



```
"C:\Program Files\Java\jdk-11.0.8\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\Inte
networking established
client read You sent > Knock Knock
client read {socket port=64529} sent > Who's There?
client read You sent > Thread
client read {socket port=64529} sent > Thread who?
client read You sent > Stop Hogging the resources, Your'e starving me!
```

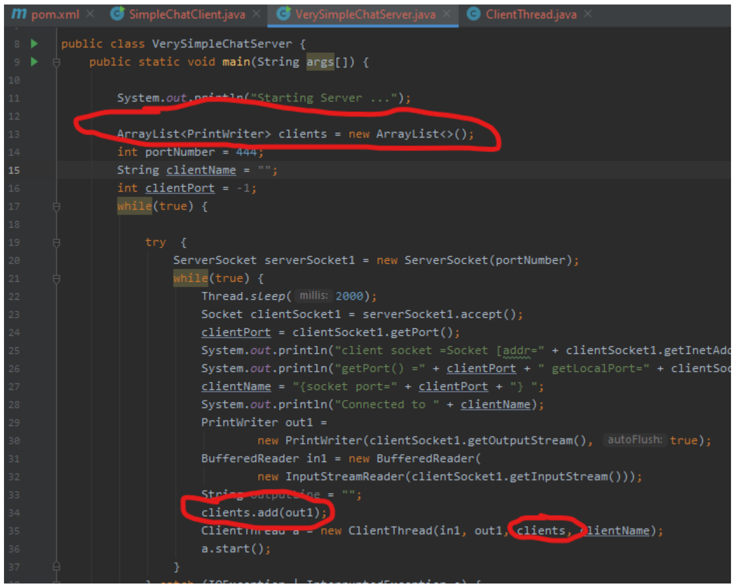
Figure 2: Client 1



```
"C:\Program Files\Java\jdk-11.0.8\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\In
networking established
client read {socket port=64521} sent > Knock Knock
client read You sent > Who's There?
client read {socket port=64521} sent > Thread
client read You sent > Thread who?
client read {socket port=64521} sent > Stop Hogging the resources, Your'e starving me!
```

Figure 3: Client 2

3 Mechanism to Distinguish between Clients



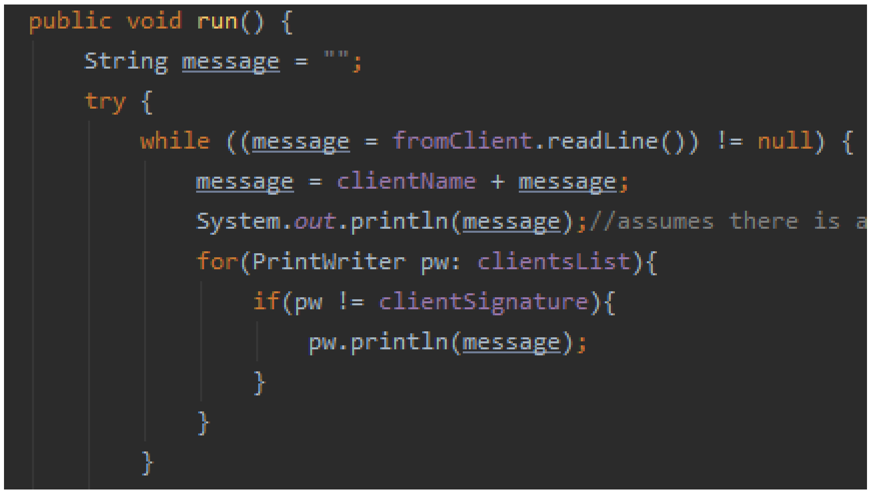
```java
public class VerySimpleChatServer {
    public static void main(String args[]) {

        System.out.println("Starting Server ...");

        ArrayList<PrintWriter> clients = new ArrayList<>();
        int portNumber = 444;
        String clientName = "";
        int clientPort = -1;
        while(true) {

            try  {
                ServerSocket serverSocket1 = new ServerSocket(portNumber);
                while(true) {
                    Thread.sleep( millis: 2000);
                    Socket clientSocket1 = serverSocket1.accept();
                    clientPort = clientSocket1.getPort();
                    System.out.println("client socket =Socket [addr=" + clientSocket1.getInetAdd
                    System.out.println("getPort() =" + clientPort + " getLocalPort=" + clientSoc
                    clientName = "{socket port=" + clientPort + "} ";
                    System.out.println("Connected to " + clientName);
                    PrintWriter out1 =
                            new PrintWriter(clientSocket1.getOutputStream(),  autoFlush: true);
                    BufferedReader in1 = new BufferedReader(
                            new InputStreamReader(clientSocket1.getInputStream()));
                    String outputLine = "";
                    clients.add(out1);
                    ClientThread a = new ClientThread(in1, out1, clients, clientName);
                    a.start();
                }
```

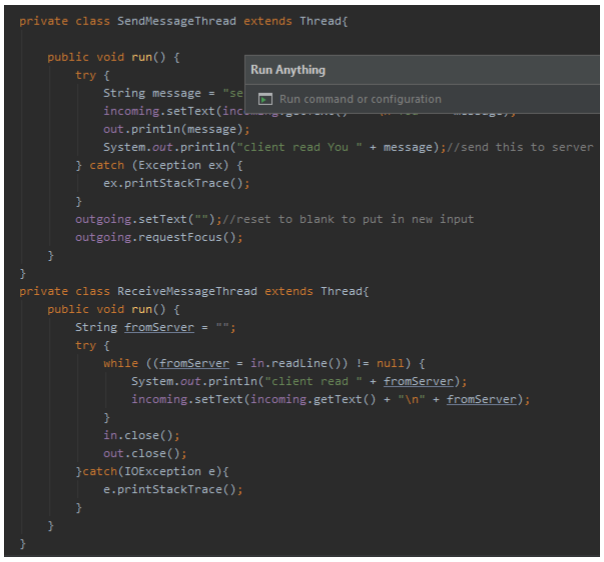
Figure 4: The server maintains a list of Clients that are connected to the server by storing their output streams. This way a client message can be sent to all the Clients connected with the server. The next figure shows how we ensure a client doesn't send a message to himself



```java
public void run() {
    String message = "";
    try {
        while ((message = fromClient.readLine()) != null) {
            message = clientName + message;
            System.out.println(message);//assumes there is a
            for(PrintWriter pw: clientsList){
                if(pw != clientSignature){
                    pw.println(message);
                }
            }
        }
```

Figure 5: This code is the task of a typical thread created by the server that deals with indivdual clients, sending their messages when they send one. As you can see we send the message to every pw, that is th PrintWriter/output stream, in the list of clients. However, before a thread will send the message it checks that the outputstream is not the same one of the client it is representing. Therefore, the thread will send the message to all the clients BUT its own

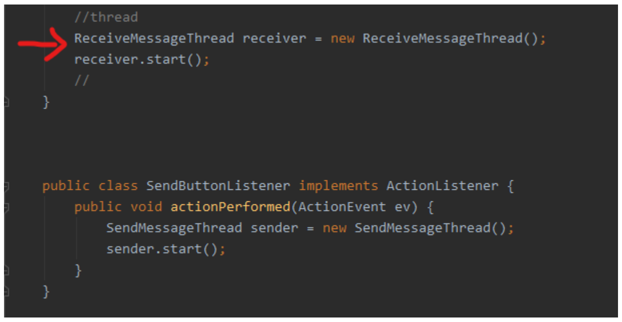
4 Client Transmit and receive messages simultaneously



```java
private class SendMessageThread extends Thread{

    public void run() {
        try {
            String message = "se
            incoming.setText(inc
            out.println(message);
            System.out.println("client read You " + message);//send this to server
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        outgoing.setText("");//reset to blank to put in new input
        outgoing.requestFocus();
    }
}
private class ReceiveMessageThread extends Thread{
    public void run() {
        String fromServer = "";
        try {
            while ((fromServer = in.readLine()) != null) {
                System.out.println("client read " + fromServer);
                incoming.setText(incoming.getText() + "\n" + fromServer);
            }
            in.close();
            out.close();
        }catch(IOException e){
            e.printStackTrace();
        }
    }
}
```

Figure 6: These two private classes of SimpleChatClient model a Thread that will deal with sending/transmitting messages and a Thread that will deal with receiving messages



```java
        //thread
        ReceiveMessageThread receiver = new ReceiveMessageThread();
        receiver.start();
        //
    }


    public class SendButtonListener implements ActionListener {
        public void actionPerformed(ActionEvent ev) {
            SendMessageThread sender = new SendMessageThread();
            sender.start();
        }
    }
```

Figure 7: This code shows the Client instantiating those Threads. The actionPerformed() method creates and executes a Sender Thread whenever the Send button is pressed. The Receiver Thread, which is pointed to with a red arrow, is at the end of the go() function and updates the client window with any incoming messages. Therefore the SimpleChatClient class can concurrently transmit and receive messages