EECS 3214
Assignment 2 Report
Daniel Diep – 213684667
April 2017

Contents

1 Introduction

This report contains the documentation describing a very simple peer-to-peer distributed message system. This system has been built off of the previous system from assignment 1 (Please refer to http://www.eecs.yorku.ca/course_archive/2016-17/W/3214/ASSGMTS/3214W17PA1.html) which was a basic directory server application. The system still has the same functionality as the previous assignment (JOIN, LEAVE, and LIST) but slightly adjusted as well as the new function that allows clients to start a chat session with another client that is in the list. Both the client and server code were implemented in Java following the specifications given from the assignment page (http://www.eecs.yorku.ca/course_archive/2016-17/W/3214/ASSGMTS/3214W17PA2.html).

2 Small changes to Directory functions from Assignment 1

For assignment 2, I made some adjustments to some of the functionality of the directory to accommodate for the new messaging feature of assignment 2. The first change was in how a client's information was stored. Before I had used just an Arraylist of strings to hold the unique user name that each client chose when they joined the directory. However for this assignment I had to keep track of more information than just the unique name such as what that client's listening port was and also a Boolean variable called inChat. This inChat variable kept track of whether the client is currently in a chat session or not. We will describe how these two changes will be used in our application below.

3 Program Design – Client Code (EchoClient.java)

The client code of this application connects to a server and the user can input commands to communicate with the server. This client can also connect to other clients that are connected to the main server and begin a chat session. NOTE* the client code is a modified version of the barebones client code given on the assignment page.

3.1 Starting up Client code

When the client first starts up the program they must provide the name of the host computer that the server is running on as well as the port number that the server is on. Then you must also include a listening port for the client so that it can start listening for connections. The port number that was instructed to use so that the chances of the port already being used is 30000 (or 40000 or 50000) + x, where x is the last 4 digits of our student number.
The host name and the port number must be correct or else the program will print out an error and close itself. One thing to keep in mind is that the server must already be up and running and is opened to accepting new clients before the client can connect to a server.

3.2 What the Client does

After you have successfully connected the client to the main server, you can start entering commands to communicate with the server. To input a command the client just needs to type it into the standard input (through the console). After the input has been entered through the standard input, it will then call a PrintWriter and send the string from the standard input to the server so that it can be processed on the server side. After the server has processed the input, it will send back a string that will be read by a BufferReader and output the message to the client. This processed is encapsulated in a while loop so the client application will continue to ask for inputs from the client. To begin a chat session the user will have to type in the command 'CHAT' and it will send this to the server notifying it that they will like to start a chat session. After this there are three responses from the server that can happen; the server says that there is no one in the list, the server says that the current user isn't in the list or that the user should enter the name of the user that they want to start a chat with. The user will keep entering users until the server finds a suitable client to start a chat with and then it starts a connection using the send() method (see details below).

3.3 send() method

This method is used to initiate a connection with the other client. This creates a new Thread and gives it the class MessageReader which outputs all the messages being received by the other client. This also lets the client type input messages in the standard input and it sends it to the other client (which also has a MessageReader class) and it reads and displays out those messages on their screen. If either clients enters the keyword 'CLOSE', the two clients will close their connecting sockets and returns them back to the main loop to start entering more commands.

3.4 get() method

The get method has the same functions to the send method but the only difference is that it is only invoked when the listening port of the client accepts a connection.

4 Program Design – Server Code (EchoServer.java)

The server code of this application is used to set up the server on a specific socket to listen to clients connecting to the server. The server will read inputs from the clients and send back a response base on the request. The server can also handle multiple clients at once. This is possible due to having multiple threads, one for each client that is connected to the server. The server also has a list that will keep track of clients who wish to join it. Clients can individually join the list or leave the list. The server is also the mediator of all clients, keeps all information involving each clients and lets clients connect to each other. NOTE* the server code is a modified version of the barebones server code given on the assignment page.

4.1 Starting up Server code

When the server program starts, it must get a port number that the clients will connect to. If no port number is given, then an error message is displayed and the program will exit. The port number that was instructed to use so that the chances of the port already being used is 20000 + x, where x is the last 4 digits of our student number.

4.2 Chat Mediation

The server mediates chat sessions like how the client does. It checks if there is anyone in the chat, whether the client is in the directory and if the client they are requesting to chat with is already in a chat or not. The server sets the status (inChat) of the clients so that it knows which clients are ready to chat and which are not.

5 Errors that may cause the application to not work correctly

One error that may happen in this application is during the transition period where the clients are either creating or closing a chat session. This is because in order to chat, the clients must have the status of inChat to be false. So when a client is starting a chat, there is a little delay where the inChat has not switched yet for both of the peers involved. If a third client tries to connect to one of them, they maybe be able to and it will interrupt the connection. However this is very unlikely as Threads to connect and switch the inChat is almost instant.

6 How to compile and run the server and client application

When testing this application, I compile and ran it using both Linux Terminal (in the Lassonde computer labs) and using Java Eclipse.

6.1 Server and Client on Linux

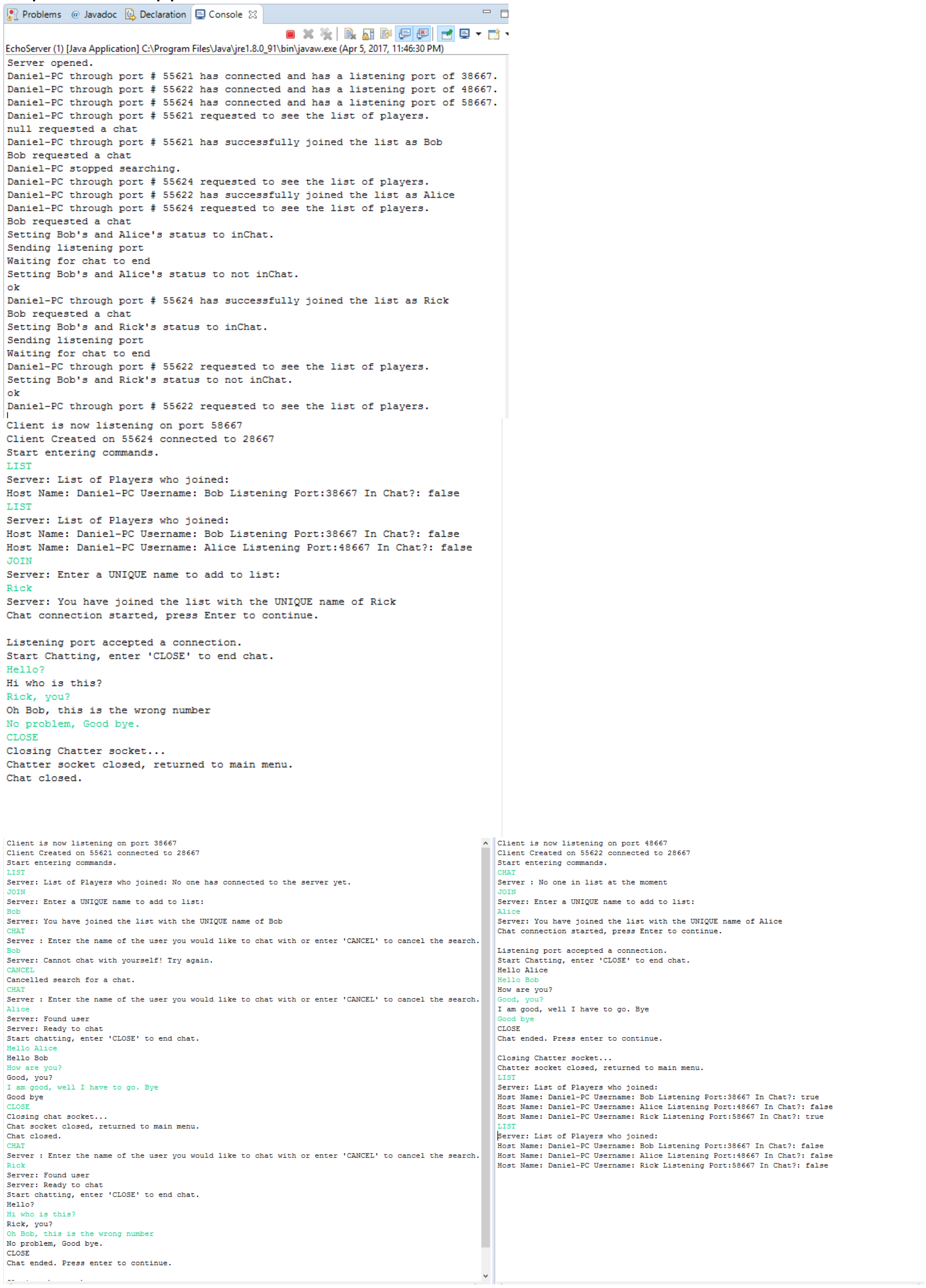
First off, we must compile both the server and client code before we can run them. Download both EchoServer.java and EchoClient.java and put them in the same folder. Then using the Linux terminal change the current directory (‘cd’) to the folder with both programs and compile them using javac command. After you have compiled both, open up a new terminal and change the directory to that folder as well, this will be the client’s console. On the original terminal we opened and run the server using java EchoServer 28667, where the number is the port which the server is listening from. Afterwards using the other terminal, we run the client using java EchoClient Daniel-PC 28667 38667 (listening port), where Daniel-PC can be the name of the computer where the server is hosted and 28667 is the server’s port number. From there the client and server connection is complete and the client may enter commands to the server. New clients may connect to the server using the same method described above.

6.2 Server and Client on Java Eclipse

Similarly to Linux, the server and client must first be compiled before it can be ran. However in Eclipse the program can just be ran and it will compile itself first before executing the code. To be able to have two (or more) separate console so that you can have one for the server and another for the client, you must first go to the bottom and click on the drop down menu beside open console and create a new console field. Next you must run the programs, however since you can’t type anything into the console to run the program and add the arguments, you must set up the individual run configurations. On the menu bar at the top, click on run then run configurations, then choose the main class to be your EchoServer first. Afterwards click on arguments and enter in the port number 28667 and then click run to get the server started. Now go back to run configurations again and select EchoClient for the main program this time and change the arguments to include the host name and the listening port and click run. Now you have the two consoles, one for your server and one for your client. If you can only see one when both programs are running, click the drop down menu beside Display Selected Console and choose the correct one.

7 Conclusion

In conclusion, the modifications of the first basic directory application created a peer-to-peer messaging app. This assignment has helped me to understand how peer-to-peer TCP/IP connections work and communicate with each other. Below is a picture of a test run I did in Java Eclipse of the application.



```
EchoServer (1) [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (Apr 5, 2017, 11:46:30 PM)
Server opened.
Daniel-PC through port # 55621 has connected and has a listening port of 38667.
Daniel-PC through port # 55622 has connected and has a listening port of 48667.
Daniel-PC through port # 55624 has connected and has a listening port of 58667.
Daniel-PC through port # 55621 requested to see the list of players.
null requested a chat
Daniel-PC through port # 55621 has successfully joined the list as Bob
Bob requested a chat
Daniel-PC stopped searching.
Daniel-PC through port # 55624 requested to see the list of players.
Daniel-PC through port # 55622 has successfully joined the list as Alice
Daniel-PC through port # 55624 requested to see the list of players.
Bob requested a chat
Setting Bob's and Alice's status to inChat.
Sending listening port
Waiting for chat to end
Setting Bob's and Alice's status to not inChat.
ok
Daniel-PC through port # 55624 has successfully joined the list as Rick
Bob requested a chat
Setting Bob's and Rick's status to inChat.
Sending listening port
Waiting for chat to end
Daniel-PC through port # 55622 requested to see the list of players.
Setting Bob's and Rick's status to not inChat.
ok
Daniel-PC through port # 55622 requested to see the list of players.
```

```
Client is now listening on port 58667
Client Created on 55624 connected to 28667
Start entering commands.
LIST
Server: List of Players who joined:
Host Name: Daniel-PC Username: Bob Listening Port:38667 In Chat?: false
LIST
Server: List of Players who joined:
Host Name: Daniel-PC Username: Bob Listening Port:38667 In Chat?: false
Host Name: Daniel-PC Username: Alice Listening Port:48667 In Chat?: false
JOIN
Server: Enter a UNIQUE name to add to list:
Rick
Server: You have joined the list with the UNIQUE name of Rick
Chat connection started, press Enter to continue.

Listening port accepted a connection.
Start Chatting, enter 'CLOSE' to end chat.
Hello?
Hi who is this?
Rick, you?
Oh Bob, this is the wrong number
No problem, Good bye.
CLOSE
Closing Chatter socket...
Chatter socket closed, returned to main menu.
Chat closed.
```

```
Client is now listening on port 38667
Client Created on 55621 connected to 28667
Start entering commands.
LIST
Server: List of Players who joined: No one has connected to the server yet.
JOIN
Server: Enter a UNIQUE name to add to list:
Bob
Server: You have joined the list with the UNIQUE name of Bob
CHAT
Server : Enter the name of the user you would like to chat with or enter 'CANCEL' to cancel the search.
Bob
Server: Cannot chat with yourself! Try again.
CANCEL
Cancelled search for a chat.
CHAT
Server : Enter the name of the user you would like to chat with or enter 'CANCEL' to cancel the search.
Alice
Server: Found user
Server: Ready to chat
Start chatting, enter 'CLOSE' to end chat.
Hello Alice
Hello Bob
How are you?
Good, you?
I am good, well I have to go. Bye
Good bye
CLOSE
Closing chat socket...
Chat socket closed, returned to main menu.
Chat closed.
CHAT
Server : Enter the name of the user you would like to chat with or enter 'CANCEL' to cancel the search.
Rick
Server: Found user
Server: Ready to chat
Start chatting, enter 'CLOSE' to end chat.
Hello?
Hi who is this?
Rick, you?
Oh Bob, this is the wrong number
No problem, Good bye.
CLOSE
Chat ended. Press enter to continue.
```

```
Client is now listening on port 48667
Client Created on 55622 connected to 28667
Start entering commands.
CHAT
Server : No one in list at the moment
JOIN
Server: Enter a UNIQUE name to add to list:
Alice
Server: You have joined the list with the UNIQUE name of Alice
Chat connection started, press Enter to continue.

Listening port accepted a connection.
Start Chatting, enter 'CLOSE' to end chat.
Hello Alice
Hello Bob
How are you?
Good, you?
I am good, well I have to go. Bye
Good bye
CLOSE
Chat ended. Press enter to continue.

Closing Chatter socket...
Chatter socket closed, returned to main menu.
LIST
Server: List of Players who joined:
Host Name: Daniel-PC Username: Bob Listening Port:38667 In Chat?: true
Host Name: Daniel-PC Username: Alice Listening Port:48667 In Chat?: false
Host Name: Daniel-PC Username: Rick Listening Port:58667 In Chat?: true
LIST
Server: List of Players who joined:
Host Name: Daniel-PC Username: Bob Listening Port:38667 In Chat?: false
Host Name: Daniel-PC Username: Alice Listening Port:48667 In Chat?: false
Host Name: Daniel-PC Username: Rick Listening Port:58667 In Chat?: false
```