

CS 513 Class Project Report

A Client-Server Chat Program

Junying Li

(Student ID Number: 160880411)

Jul. 3, 2021

Statement: This project is all my work.

Sign: Junying Li

Abstract

This report outlines the design and development of a chat program which based on the client - server model. Using Java Socket programming, the server can handle all requests from multiple clients. Multiple users are able to connect or disconnect to the server, and chat with each other in this chat room. To protect users' privacy, clients are able to choose a nickname before joining the chat room. They can also whisper to some other client without having messages displayed to others, or they can make it public. There is a user list board on the right side of client GUI which displays all online users. This list will refresh every time when someone join or leave the chat room. The design and program are modular in nature and make maximum use of abstract data types and of software re-use. The report includes abstract, content pages, project description, detailed design, future development, conclusion, and appendices which contains test cases used to verify the correct operation of the program, annotated screenshots to demonstrate that my program has successfully implemented all required specifications and functions, and the entire code too.

Contents Page

Abstract	2
Contents Page	3
Project Description	4
Detailed Design	5
Testing and Evaluation	8
Future Development.....	10
Conclusion – Solution Summary	11
Appendices	12

Project Description

This project is an individual class project of WPI's CS 513 Computer Network. It aims to construct a multi-person online chat program. I named it as "ChaToGo" (chat to go) which means it is a convenient and lightweight online chat room. This program is designed and developed based on the client-server model using Java Socket programming. All codes are developed and run in JetBrains IntelliJ IDEA 2020.2.3 x64. For safety concern, the server is designed stable after client termination, and so does the clients after server termination. By making use of Java Exceptions, all opened sockets and streams will be closed on client after termination. To make sure every feature works well, I also designed test cases and tested the program for three times. Since I tested this program in my local environment, I set the default host and port of server as 127.0.0.1:6209. If you need to change them, you can open `src/server/GroupChatServer.java` and change the address.

With this program I have developed, I want to demonstrate that a multi-threaded TCP/IP Socket server is able to handle multiple requests from clients concurrently without disruption of other services.

Detailed Design

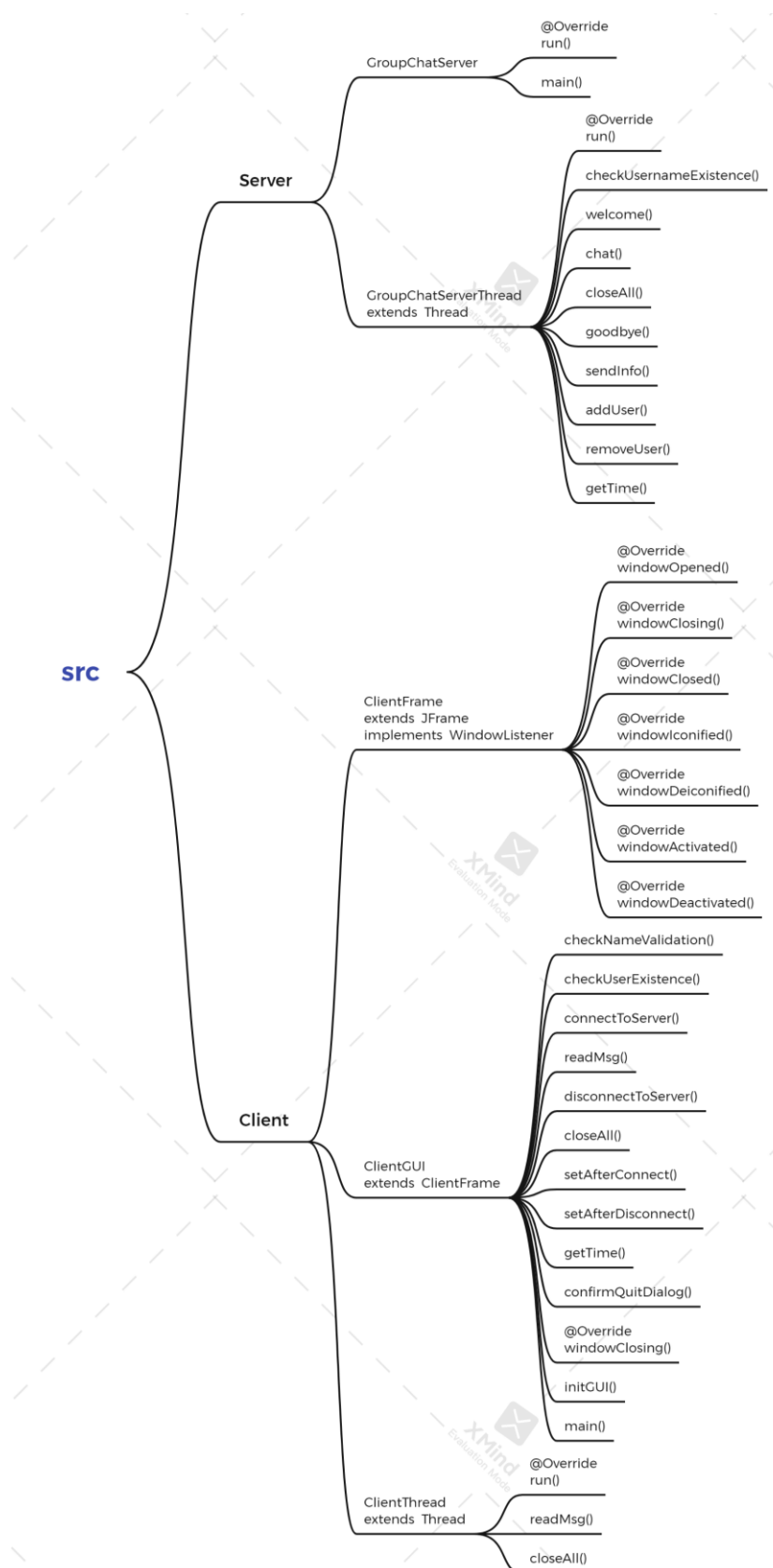


Figure-1 Program Design Mind Map

For the first step of designing this program, I drew a mind map and listed what packages, classes, and functions I need. I also noted the inheritance of classes in the mind map, as you can see in *Figure-1*. There are two major parts in the design of this project: server, and client. The details of the two parts will be described as follows.

For the server, I designed a multi-threaded TCP/IP server to handle with all the requests from clients. The server is supposed to: 1) process connect and disconnect requests from clients, and also print these operations out. 2) handle connections and disconnections without disruption of other services. 3) make sure every client has a unique nickname. 4) send broadcast to every online client of changes in the list of connected users.

To implement the functions above, I designed 2 classes for server packet: GroupChatServer and GroupChatServerThread. The GroupChatServer class is used to run the socket server by its private function runServer(). The GroupChatServerThread class extends Thread, and it is used to handle with threads. I made use of BufferedReader to read input stream from socket, and made use of BufferedWriter to write output stream from socket. There are 10 functions in GroupChatServerThread class: 1) run(). 2) checkUsernameExistence(), being used to check if a username is duplicated or not. 3) welcome(), being used to send welcome message to new client, and send broadcast to all other online clients to inform that a new client has joined in the chat room. 4) chat(), being used to process public and private (whisper) messages sent by clients. 5) closeAll(), being used to close BufferedReader, BufferedWriter, input stream, output stream, and socket. Then remove current thread from thread list. 6) goodbye(), being used to send broadcast to all online clients to inform that someone has left the chat room. 7) sendInfo(), being used to process sending messages by making use of BufferedWriter. 8) addUser(), being used to add a new user. 9) removeUser(), being used to remove a user. 10) getTime(), being used to get the current time.

For the client, I designed a GUI for user operations by making use of Java Swing JFrame. The client is supposed to: 1) show the list of online users. 2) be able to let user choose a nickname as well as connect/ disconnect to the server. 3) be able to receive public and private (whisper) messages from other clients. 4) be able to send public and private (whisper) messages to other clients. 5) be able to display system messages.

To implement the functions above, I designed 3 classes for client packet: ClientFrame, ClientGUI, and ClientThread. The ClientFrame class extends JFrame and implements WindowListener, it is the super class of ClientGUI. ClientGUI class is being used to set up a GUI for client. The GUI consists 4 parts. 1) Settings Part - the upper area of GUI. It is being used to let user input host address, port, and nickname. And there are connect and disconnect button in this area too. 2) Group Chat Part - the left middle area of GUI. It is the

message board, displaying server messages and messages sent by clients. 3) Online Users Part - the right middle area of GUI. It displays all online users' nickname. 4) Send Message Part - the bottom of GUI. In this area, there is a JComboBox for users to choose their message receivers (everyone or some specific client), a JTextField to input messages, and a Send button. The ClientThread class extends Thread, it is being used to handle with client threads. The ClientThread class has 3 functions: 1) run(), override the run() function of Thread. 2) readMsg(), is being used to read messages sent by server by making use of BufferedReader. 3) closeAll(), is being used to close all closeable io.

Testing and Evaluation

My testing strategy is to design test cases that cover all main scenarios and as many corner cases as I can come up with, then go through all test cases for three times. The second test and the third test are regression tests. For the first test, I will run all test cases and annotate all problem. Then I will fix those problems and start the second test. After that, I'll fix all problems I've found in the second test, then start the third test. If all test cases pass, my program could be considered working well. If there is still problem been found in the third test, I will repeat the process until bug free.

The main scenarios and corner cases I have come up with so far are listed below.

Main Scenarios:

1. Start server.
2. Stop server.
3. Client connects to server.
4. Client disconnects to server.
5. Client chooses nickname.
6. Client sends public messages (concurrency).
7. Client whispers to another client (concurrency).
8. Client receives messages (concurrency).

Corner Cases:

1. Client tries to connect to server with wrong address.
2. Client chooses invalid nickname.
3. Client chooses duplicated nickname.
4. Client closes GUI without disconnecting to server first.
5. Server closes while clients are online.

I made a JComboBox in the bottom of client GUI for users to choose message receiver, and the combo box only has current online clients in it. Thus, it won't happen that any client tries to send messages to a non-existing client, that's why this corner case hasn't been covered in my test cases.

I designed 17 test cases for my project and run them for three times. You can find all test cases and their results in the appendix. I also took some annotated screenshots to demonstrate that my program has successfully implemented all of the required general project specifications, the specifications for the client(s) and server, and also the list of

functions contained in the Project Grade Allocation summary. You can find those screenshots in appendix too.

In conclusion, all test cases pass. Hence, I suppose my program works well and meet professor's requirements.

Future Development

For future development, I'm considering to add a new feature, group. When new clients get in the chat room, they will join the default group "Public Group". Then they can leave current group and join another group that they are more interested in, for instance, "Sports Group", "Music Group", etc. In addition, clients are also able to create new groups or delete groups. This may lead to some permission issues, so I will design a permission system in future work as well.

Another thing I would like to do in future development is to set the server on AWS and invite enough volunteers to chat online through my client GUI. Then I could do server stress tests and evaluate my program's performance at high concurrency. After that, I would consider how to improve the performance based on the stress test results. The optimization strategy will probably involve load balancing.

Conclusion – Solution Summary

This report has described the successful design and development of an online chat program based on client-server model. In conclusion, a multi-threaded TCP/IP socket server is able to accomplish the following tasks: 1) handle connections / disconnections from clients without disruption of other services, 2) process sending / receiving public and private messages (concurrency), 3) store and refresh the list of currently connected users. In addition, `BufferedReader` and `BufferedWriter` do a good job in reading and writing messages between server and clients.

Appendices

1. Test Cases and Test Results:

Test Case#	Action	Pre-requisite	Input	Expected Output	Actual Output	Test Result
1	Start server.	Null.	host: 127.0.0.1 port: 6209	1. Server starts successfully. 2. Server prints out "Server is running! Waiting for connection..."	Same as expected.	Pass
2	Open client GUI.	Null.	No input.	1. Client GUI is being opened successfully. 2. Elements are being displayed correctly.	Same as expected.	Pass
3	Client connects to server with wrong host address.	1. Server is running.	host: 125.0.0.1 port: 6209	1. Pop up a warning window "Connection failed. Please check host and port address."	Same as expected.	Pass
4	Client connects to server with wrong port address.	1. Server is running.	host: 127.0.0.1 port: 1234	1. Pop up a warning window "Connection failed. Please check host and port address."	Same as expected.	Pass
5	Client choose "Everyone" as nickname.	1. Server is running.	nickname: Everyone	1. Pop up a warning window "You can't use 'Everyone' as nickname."	Same as expected.	Pass
6	Client choose a nickname contains space.	1. Server is running.	nickname: Jess ica	1. Pop up a warning window "Nickname can't contain space."	Same as expected.	Pass
7	Client choose a blank nickname.	1. Server is running.	No input.	1. Pop up a warning window "Please entry a nickname."	Same as expected.	Pass

8	Client choose a duplicated nickname.	1. Server is running. 2. There is already a client named Jack.	host: 127.0.0.1 port: 6209 nickname: Jack.	1. Pop up a warning window "The nickname has already been taken. Please re-entry!"	Same as expected.	Pass
9	Client connects to server with correct address and a unique nickname.	1. Server is running. 2. There is not a client named Kate.	host: 127.0.0.1 port: 6209 nickname: Kate	1. Connect to the server successfully. 2. Client displays welcome message. 3. Server prints out Kate's connect request.	Same as expected.	Pass
10	Another client connects to server.	1. Server is running. 2. Jack and Kate are online.	host: 127.0.0.1 port: 6209 nickname: Amy	1. Server prints out Amy's connect request. 2. Jack and Kate receive broadcast in their clients to inform that Amy joined the chat room. 3. User list refreshed and add Amy.	Same as expected.	Pass
11	Client Jack sends public messages.	1. Server is running. 2. Jack, Kate, and Amy are online.	"Hi, everyone!"	1. Server prints out messages: "Jack: /Everyone Hi, everyone!" 2. Kate and Amy's clients receive Jack's public message in the same time: "Jack: Hi, everyone!" 3. Jack's client displays his message in the same time as Kate and Amy receive message: "You: Hi, everyone!"	Same as expected.	Pass
12	Client Kate whispers to Client Amy.	1. Server is running. 2. Jack, Kate, and Amy are online.	"Hi Amy, I'm Kate."	1. Server prints out messages: "Kate: /Amy Hi Amy, I'm Kate." 2. Amy's clients receive Kate's private message: "Kate whispers to you: Hi	Same as expected.	Pass

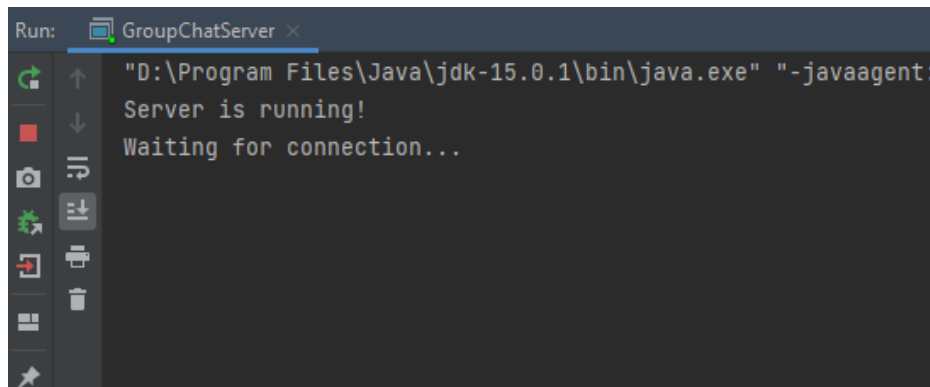
				<p>Amy, I'm Kate."</p> <p>3. Kate's client displays her message: "You whisper to Amy: Hi Amy, I'm Kate."</p> <p>4. Jack's client doesn't receive this message.</p>		
13	Client Jack disconnects to server.	<p>1. Server is running.</p> <p>2. Jack, Kate, and Amy are online.</p>	No input.	<p>1. Jack's client pop up a warning window: "You are disconnected."</p> <p>2. Jack cannot send messages after disconnected.</p> <p>3. Server prints out Jack's disconnect request.</p> <p>4. Kate and Amy receive broadcast in their clients to inform that Jack left the chat room.</p> <p>5. User list refreshed and remove Jack.</p>	Same as expected.	Pass
14	Client Kate closes client GUI.	<p>1. Server is running.</p> <p>2. Kate and Amy are online.</p>	No input.	<p>1. Kate's client pop up a warning window: "Do you really want to quit?"</p>	Same as expected.	Pass
15	Client Kate closes client GUI and choose "Yes".	<p>1. Server is running.</p> <p>2. Kate and Amy are online.</p>	No input.	<p>1. Amy receives a broadcast in her client to inform that Kate left the chat room.</p> <p>2. User list refreshed and remove Kate.</p> <p>3. Kate's client closed.</p> <p>4. Server prints out Kate's disconnect request.</p>	Same as expected.	Pass
16	Close server while client is online.	<p>1. Server is running.</p> <p>2. Amy is online.</p>	No input.	<p>1. Server stopped.</p> <p>2. Amy's client pops up a warning window: " You are disconnected."</p>	Same as expected.	Pass

17	Close server while no client is online.	1. Server is running. 2. No client is online.	No input.	1. Server stopped.	Same as expected.	Pass
----	---	--	-----------	--------------------	-------------------	------

2. Annotated Screenshots

1. Server starts.

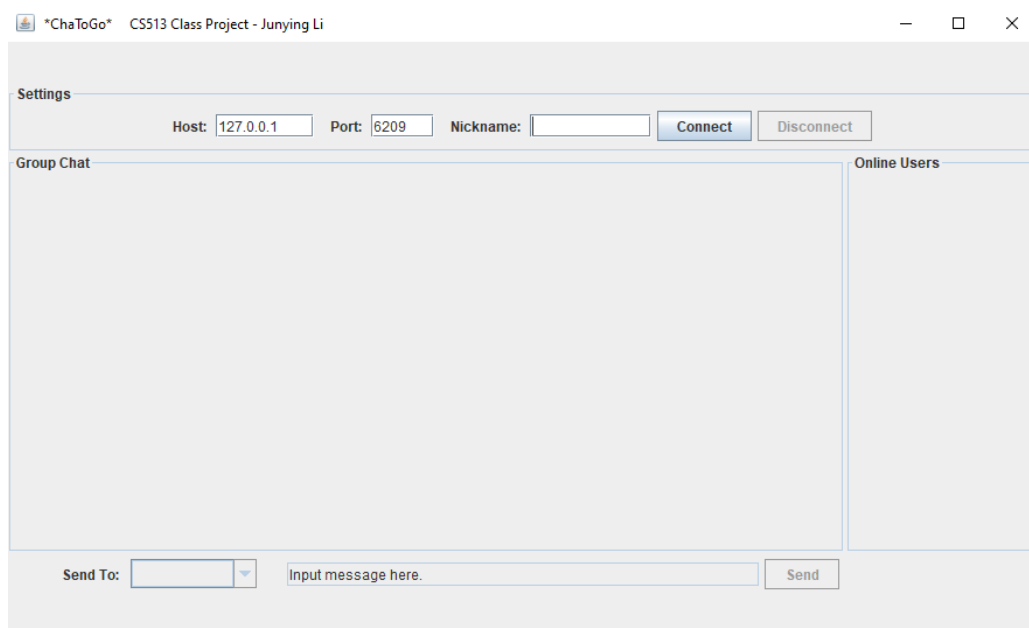
Server prints out "Server is running! Waiting for connection..."



```
Run: GroupChatServer x
"D:\Program Files\Java\jdk-15.0.1\bin\java.exe" "-javaagent:
Server is running!
Waiting for connection...
```

2. Open Client GUI.

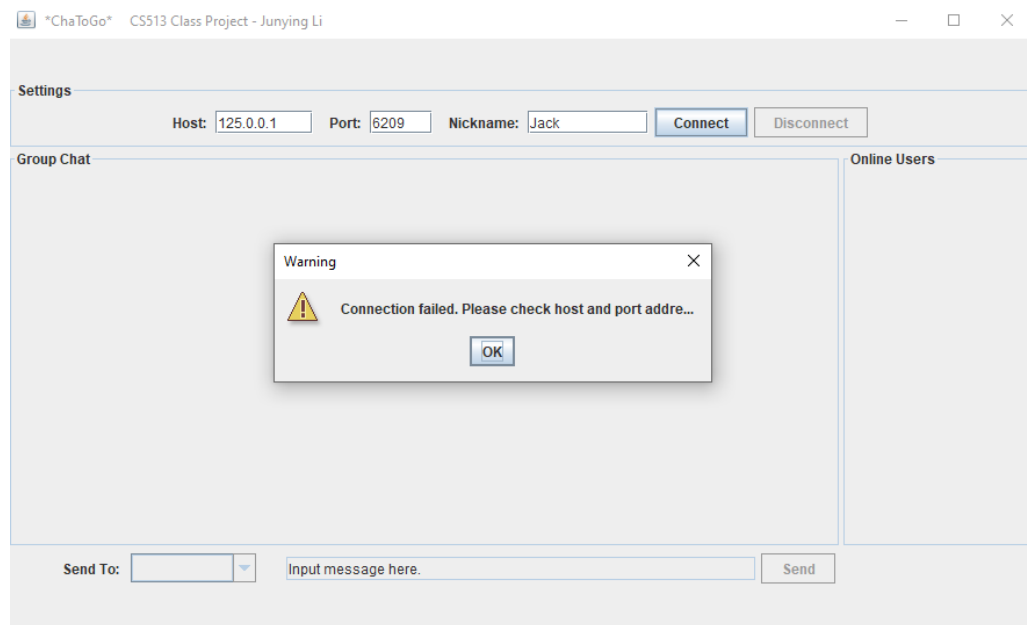
Client GUI's elements display correctly. Waiting for user to input nickname and connect to server.



3. Connect to Server.

(1) Connect with wrong address (trying to connect to non-existing server).

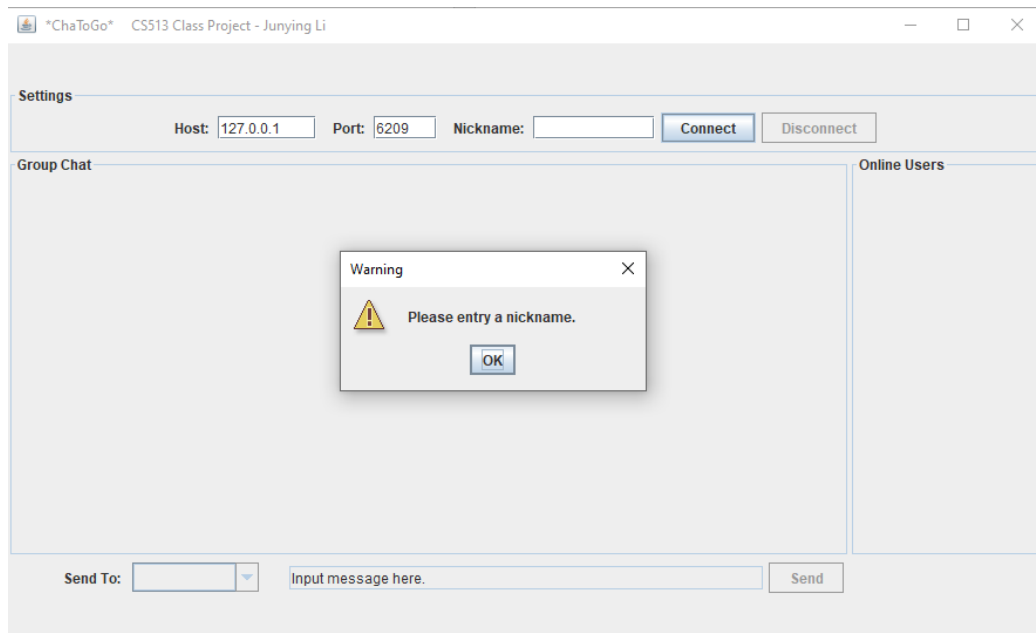
Client pops up a warning window "Connection failed. Please check host and port address."



(2) Connect with correct address and invalid nickname.

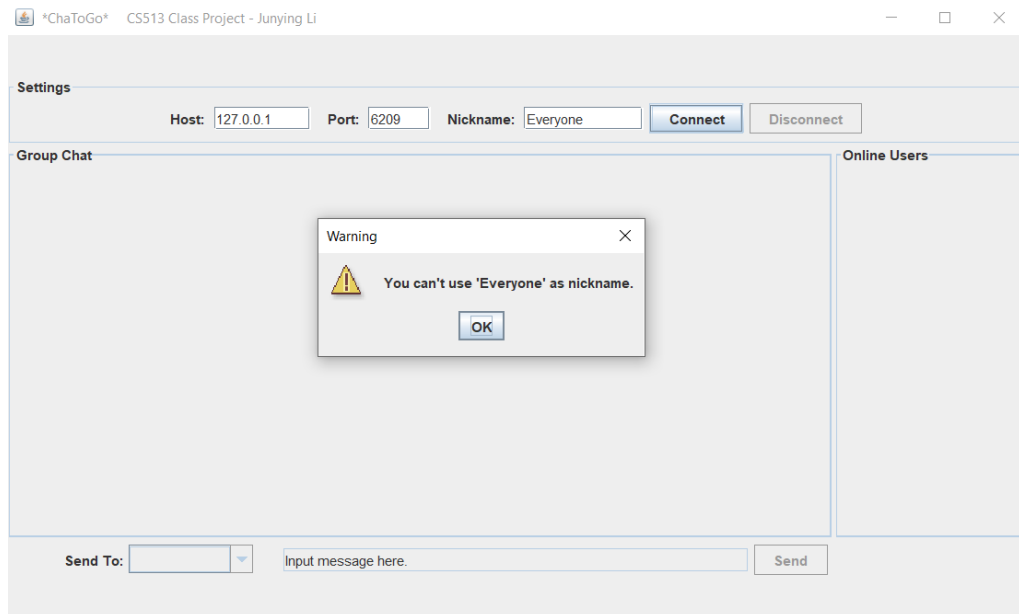
- Blank nickname.

Client pops up a warning window "Please entry a nickname."



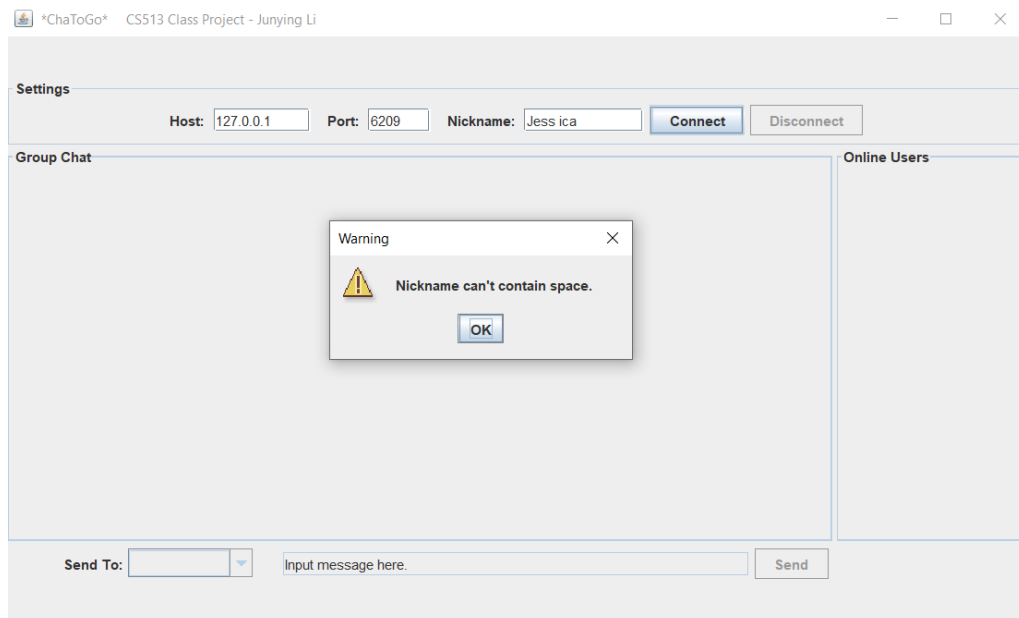
- Use "Everyone" as nickname.

Client pops up a warning window "You can't use 'Everyone' as nickname."



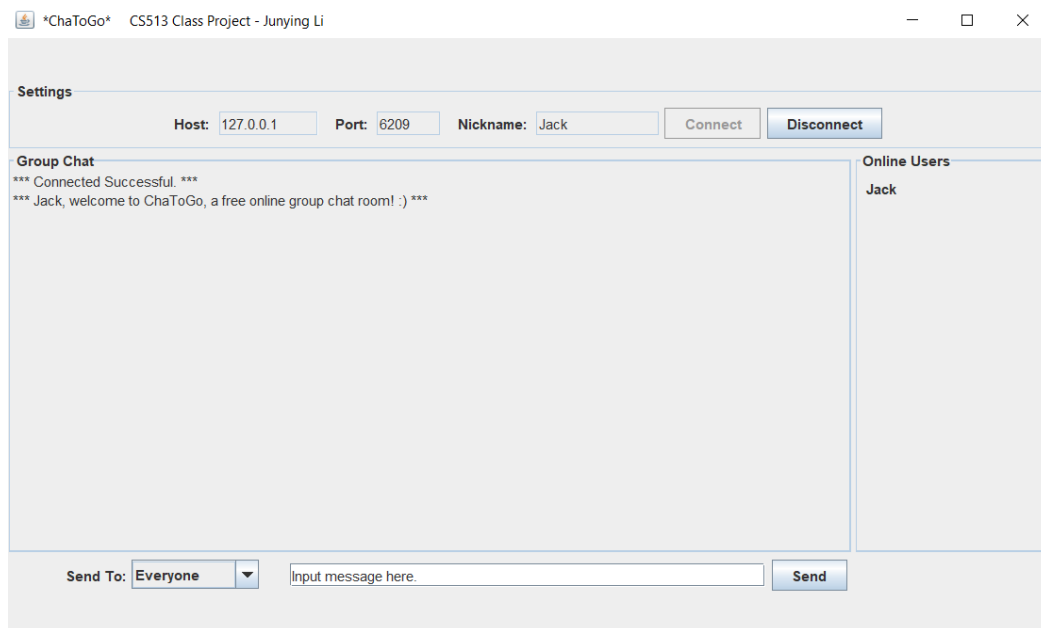
- Nickname contains space.

Client pops up a warning window “Nickname can’t contain space.”



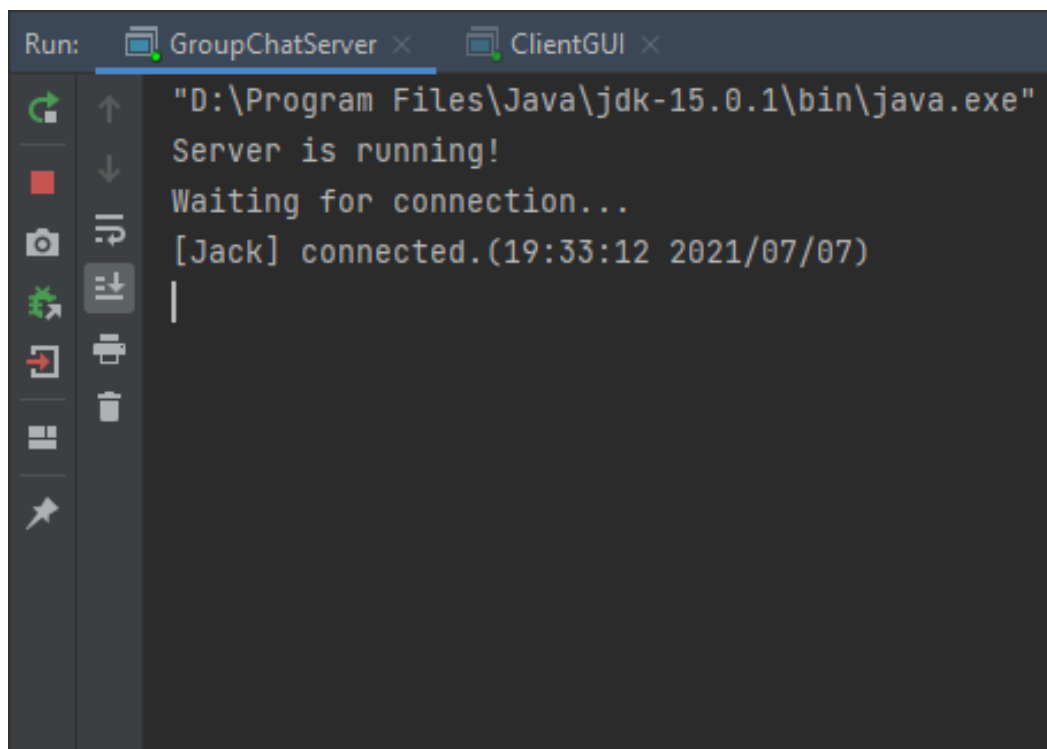
(3) Connect with correct address and unique nickname (Jack).

- Client
Client displays welcome message. User list refreshed and add “Jack”.



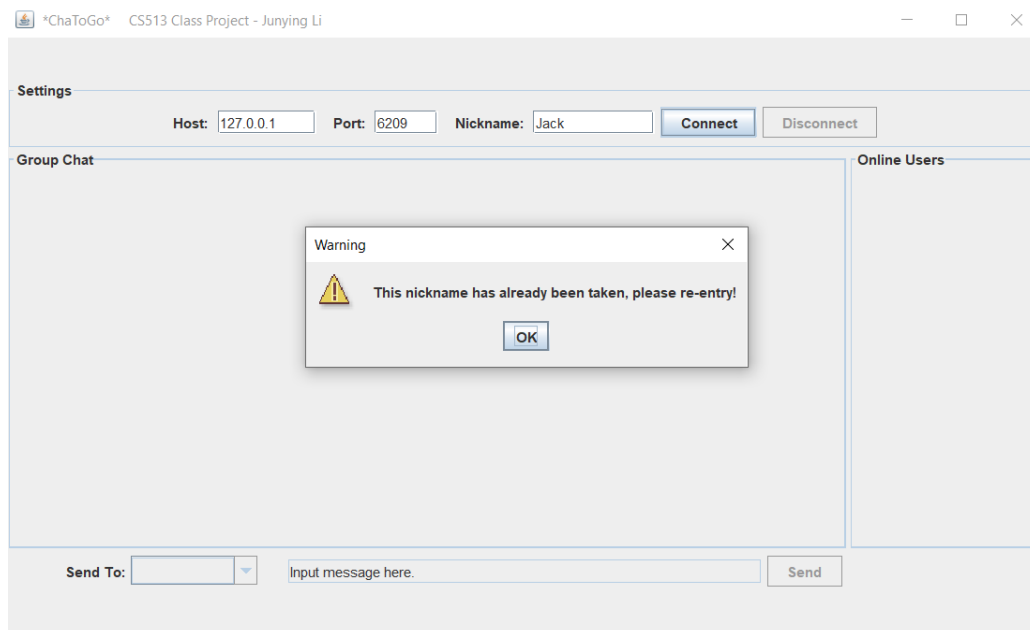
- Server

Server prints out Jack's connect request.



- (4) Connect with correct address and duplicated nickname.

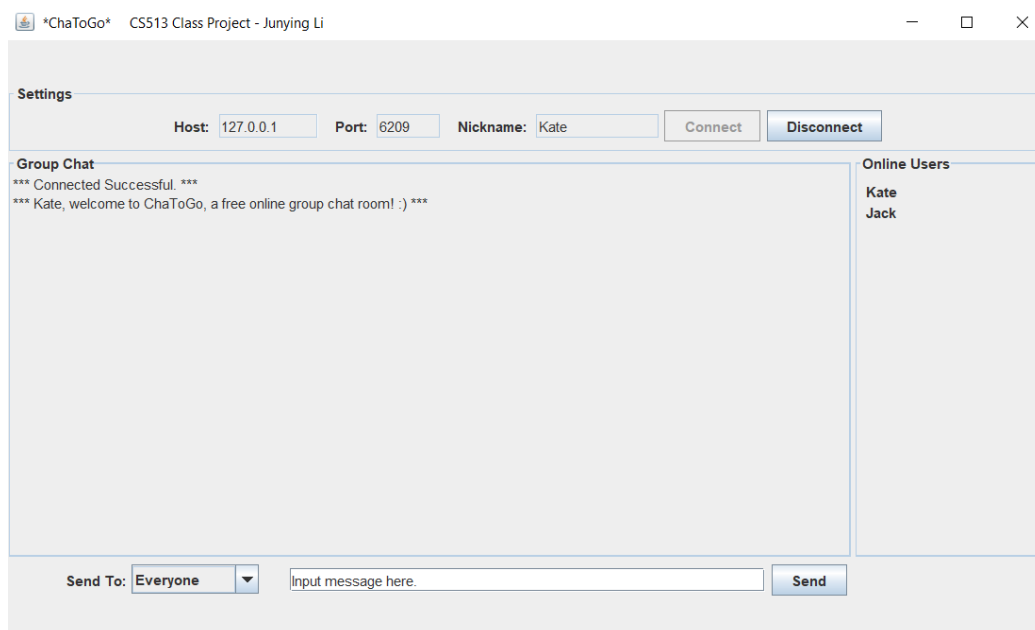
Client pops up a warning window "This nickname has already been taken. Please re-entry!"



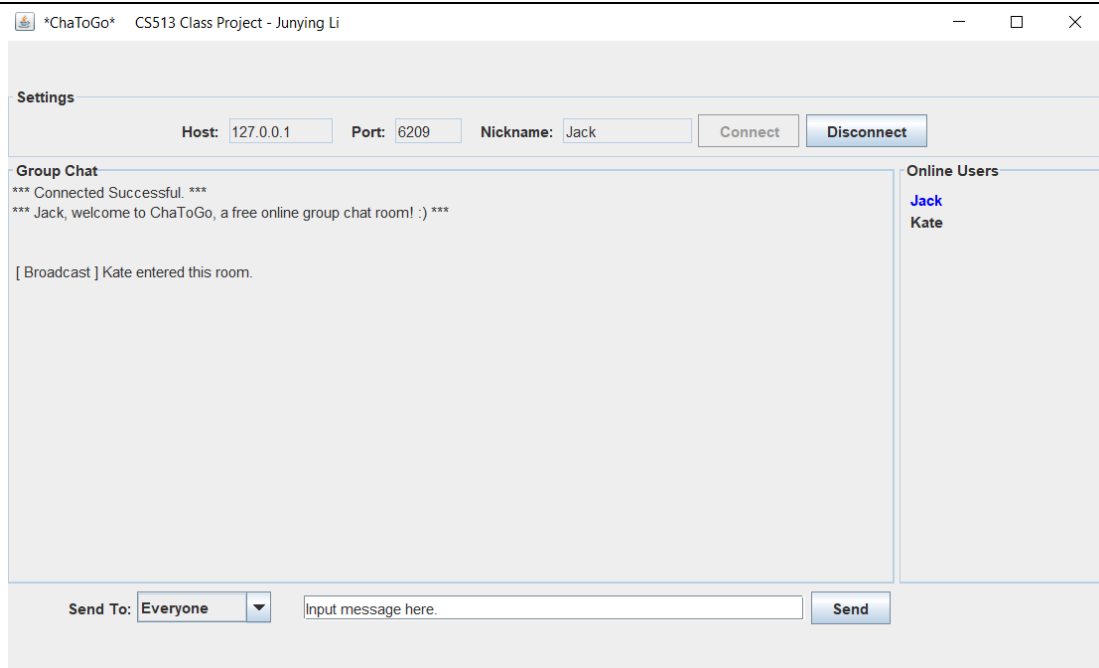
4. New Client Connects to Server.

Kate connects to server. Jack receives broadcast to inform that Kate joined. User list refreshed and add "Kate".

- Kate's Client GUI.

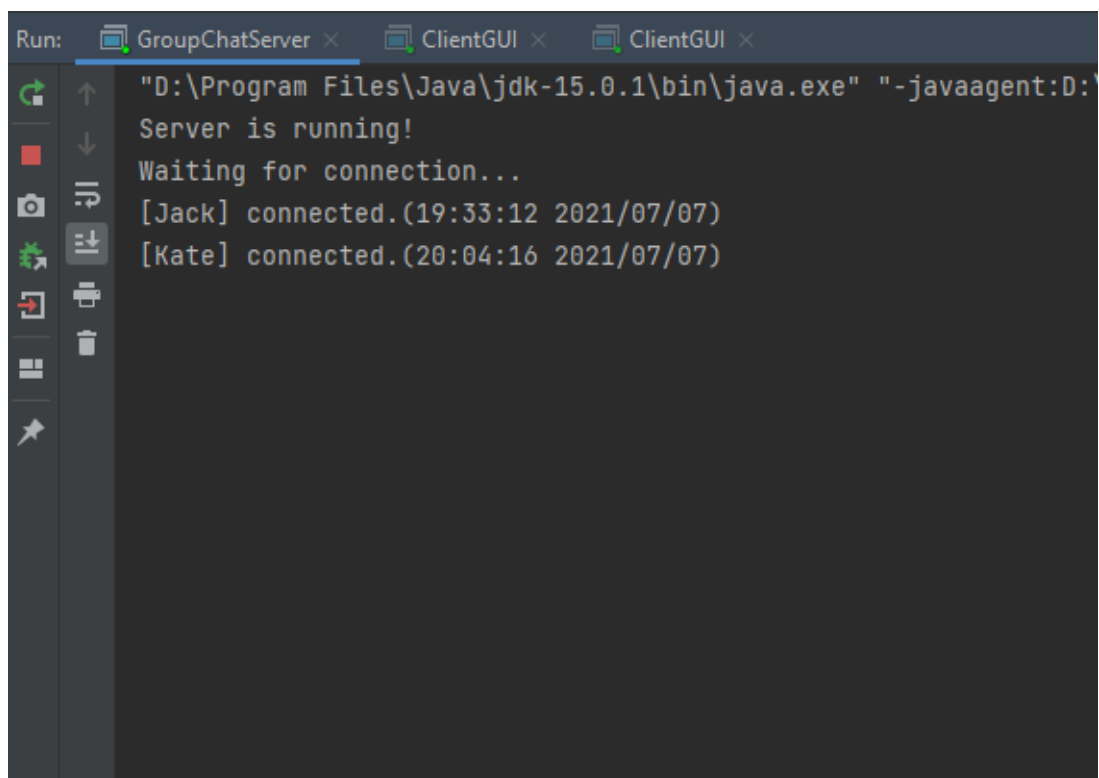


- Jack's Client GUI.



- Server.

Server is stable. It prints out Kate's connect request.

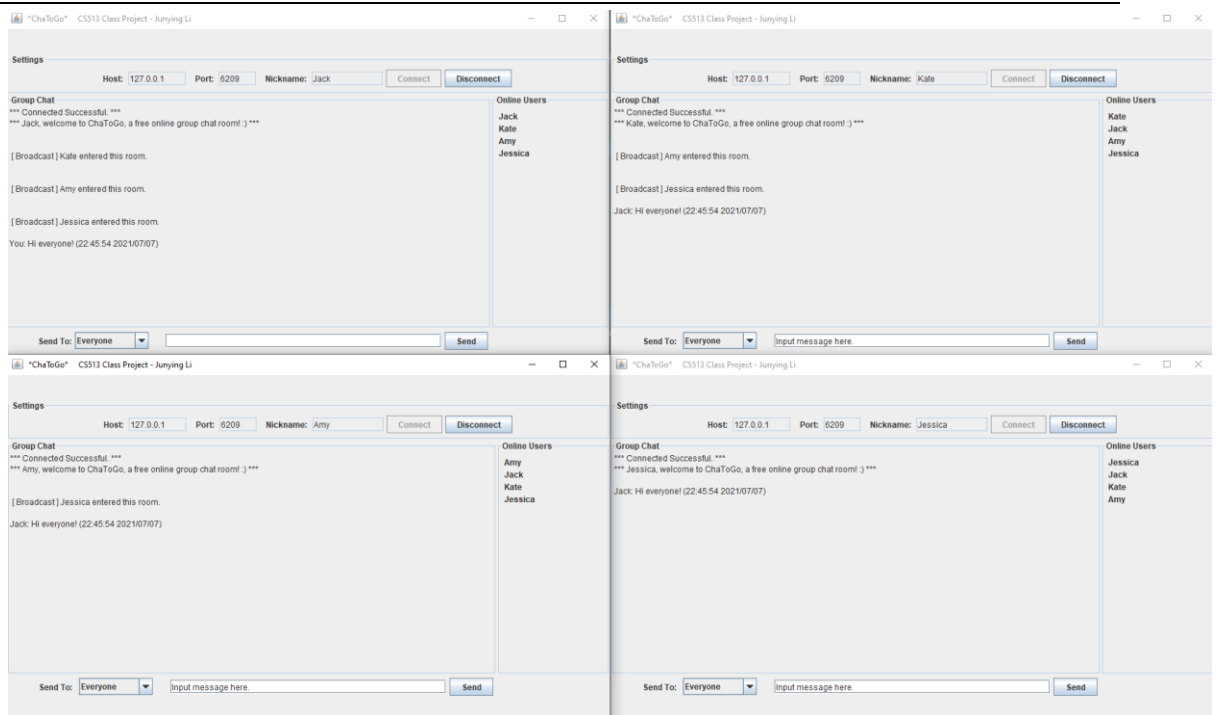


5. Clients send / receive public messages.

- Clients.

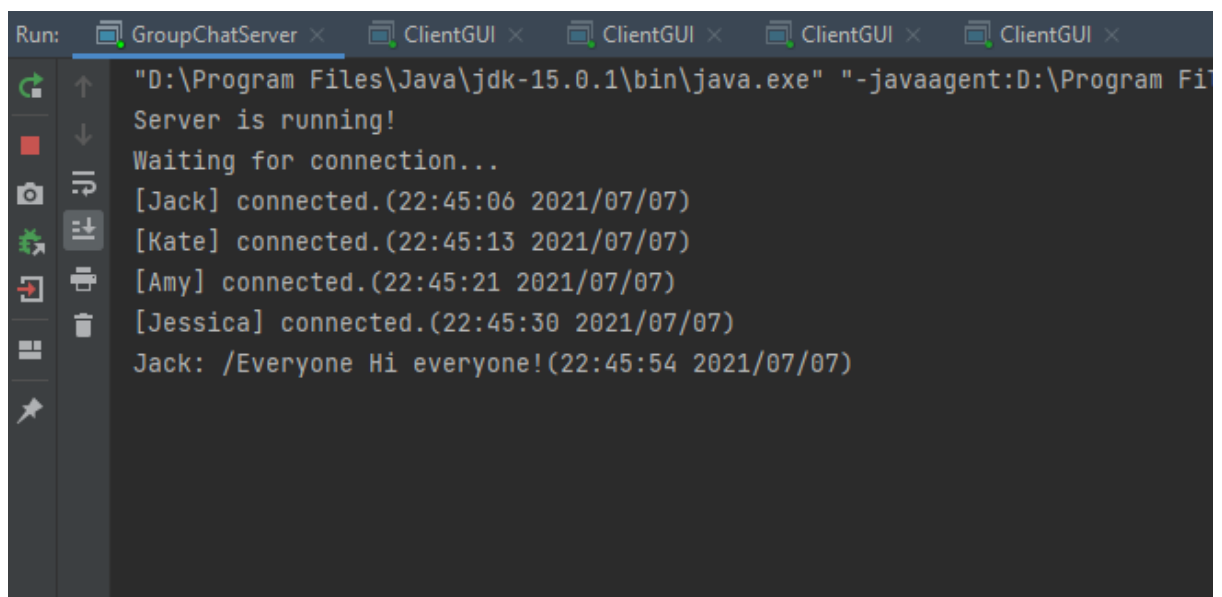
Clients are stable.

All online clients can send and receive public messages.



- Server.

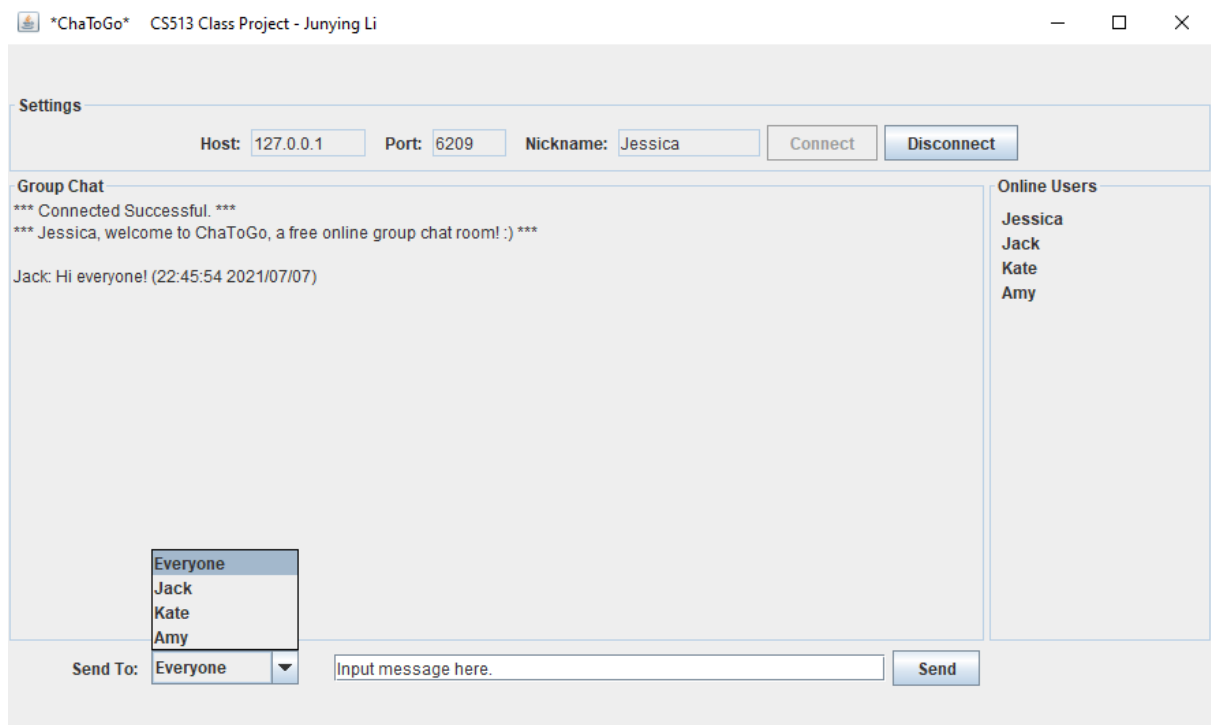
Server is stable and it prints out message: "Jack: /Everyone Hi everyone!".



6. Clients send / receive private messages (whisper).

- Choose message receiver.

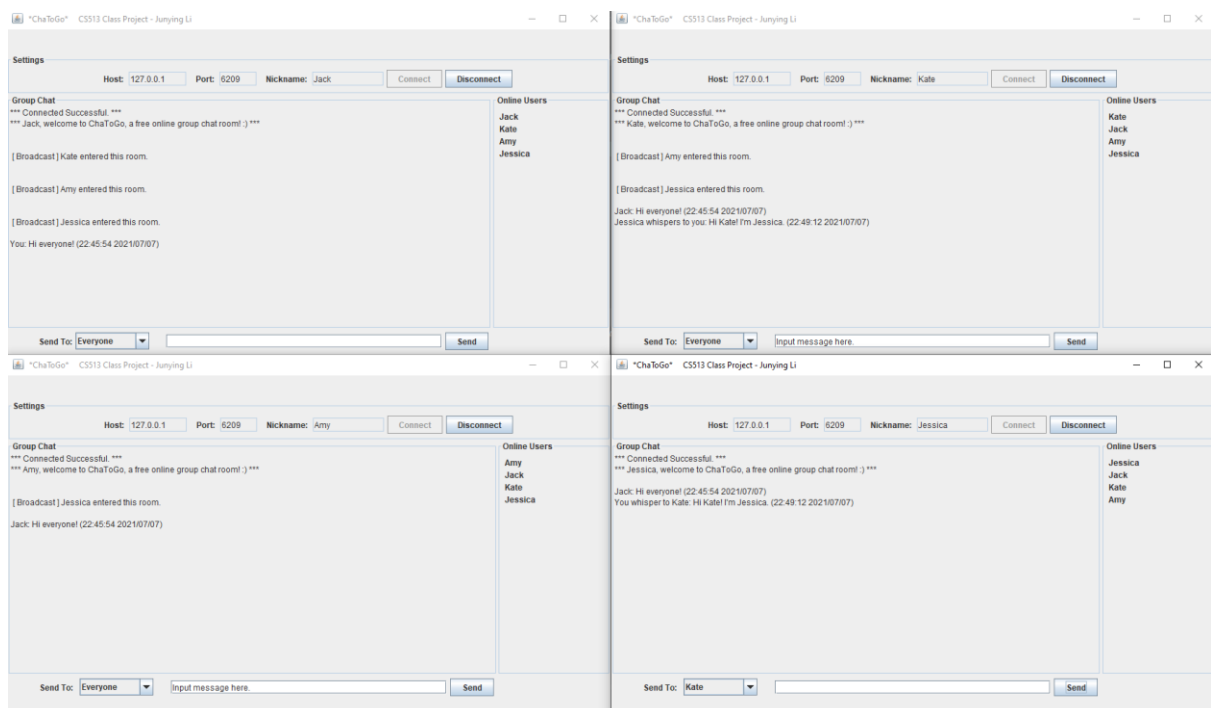
The JComboBox only contains currently online clients. Thus, users are not able to send message to non-existence client.



- Whisper - Client

Jessica whispers to Kate: Hi Kate! I'm Jessica.

Clients can send and receive whispers without having messages displayed to other users.



- Whisper – Server

Server is stable and it prints out message: "Jessica: /Kate Hi Kate! I'm Jessica."

```

Run: GroupChatServer x ClientGUI x ClientGUI x ClientGUI x ClientGUI x
"D:\Program Files\Java\jdk-15.0.1\bin\java.exe" "-javaagent:D:\Program
Server is running!
Waiting for connection...
[Jack] connected.(22:45:06 2021/07/07)
[Kate] connected.(22:45:13 2021/07/07)
[Amy] connected.(22:45:21 2021/07/07)
[Jessica] connected.(22:45:30 2021/07/07)
Jack: /Everyone Hi everyone!(22:45:54 2021/07/07)
Jessica: /Kate Hi Kate! I'm Jessica.(22:49:12 2021/07/07)
|

```

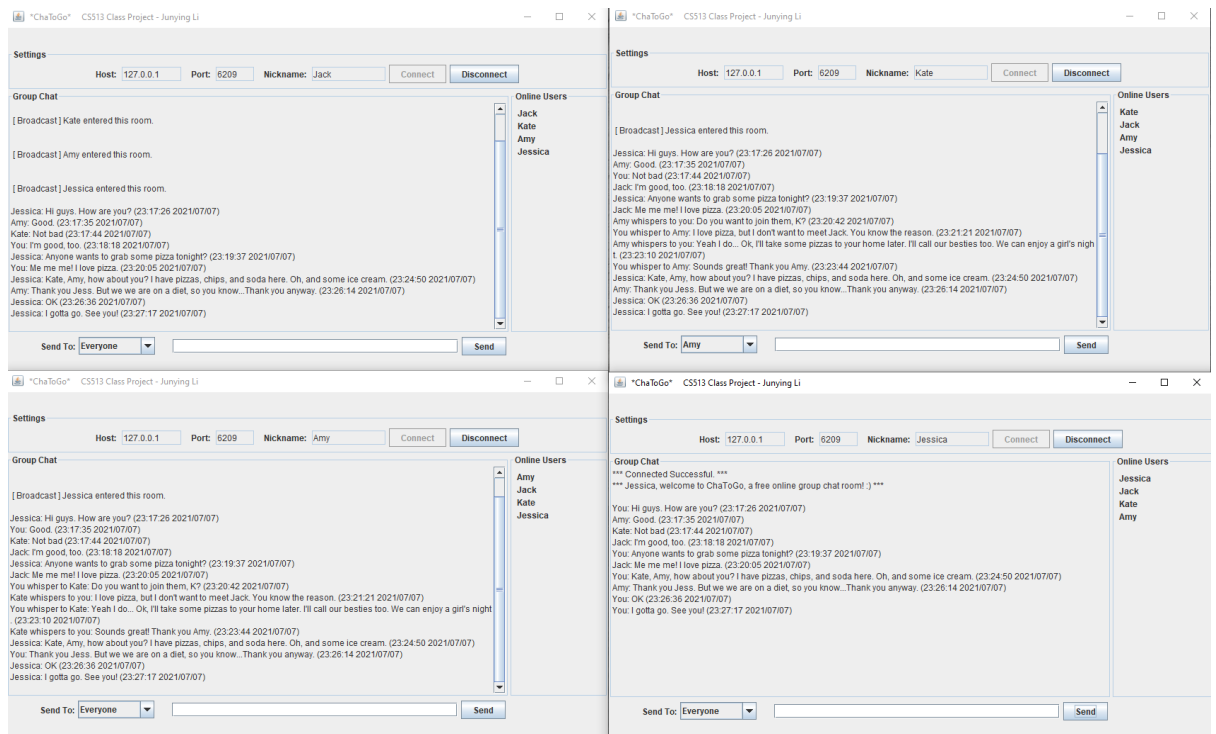
7. Send / receive messages concurrently.

- Client.

Clients are stable at concurrency.

All online clients can receive public messages.

Clients can send and receive whispers without having messages displayed to other users.



- Server.

Server is stable at concurrency.

It prints out clients' messages and their requests.

```

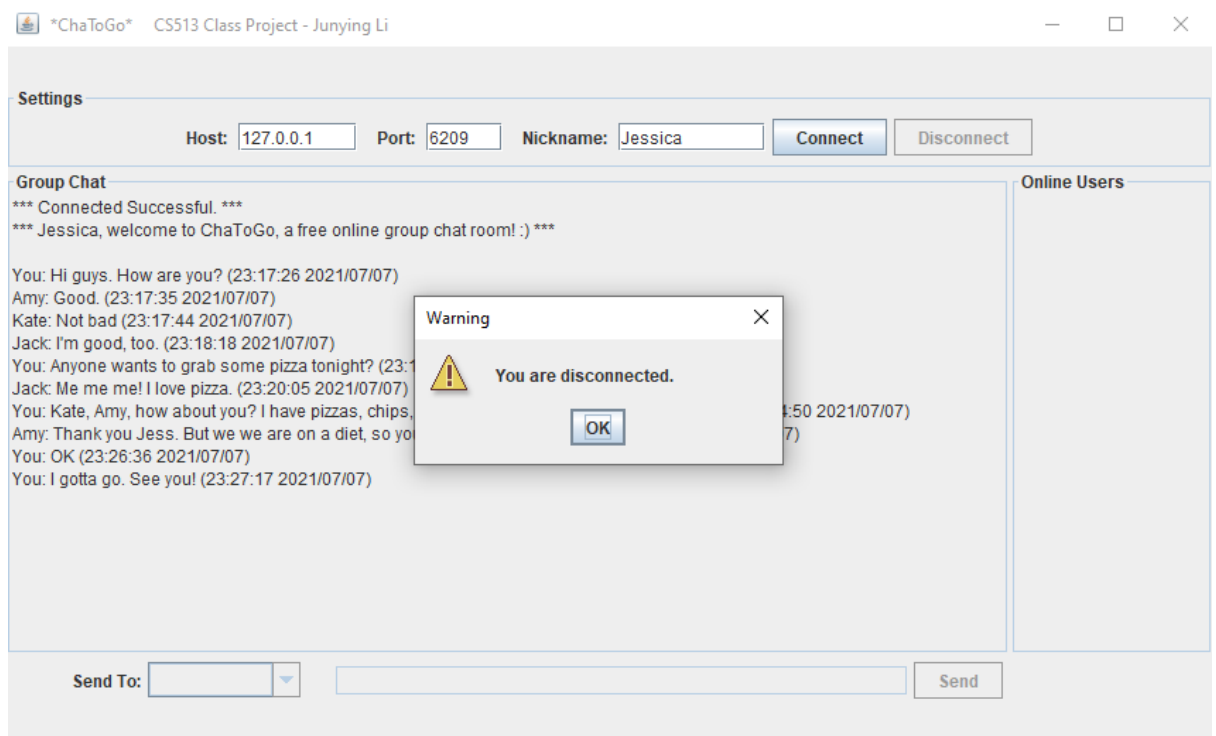
Run: GroupChatServer x ClientGUI x ClientGUI x ClientGUI x ClientGUI x
"D:\Program Files\Java\jdk-15.0.1\bin\java.exe" "-javaagent:D:\Program Files\JetBrains\IntelliJ IDEA 2020.2.3\lib\idea_rt.jar=56823:D:\Program Files
Server is running!
Waiting for connection...
[Jack] connected.(23:16:09 2021/07/07)
[Kate] connected.(23:16:14 2021/07/07)
[Amy] connected.(23:16:19 2021/07/07)
[Jessica] connected.(23:16:24 2021/07/07)
Jessica: /Everyone Hi guys. How are you?(23:17:26 2021/07/07)
Amy: /Everyone Good.(23:17:35 2021/07/07)
Kate: /Everyone Not bad(23:17:44 2021/07/07)
Jack: /Everyone I'm good, too.(23:18:18 2021/07/07)
Jessica: /Everyone Anyone wants to grab some pizza tonight?(23:19:37 2021/07/07)
Jack: /Everyone Me me me! I love pizza.(23:20:05 2021/07/07)
Amy: /Kate Do you want to join them, K?(23:20:42 2021/07/07)
Kate: /Amy I love pizza, but I don't want to meet Jack. You know the reason.(23:21:21 2021/07/07)
Amy: /Kate Yeah I do... OK, I'll take some pizzas to your home later. I'll call our besties too. We can enjoy a girl's night.(23:23:10 2021/07/07)
Kate: /Amy Sounds great! Thank you Amy.(23:23:44 2021/07/07)
Jessica: /Everyone Kate, Amy, how about you? I have pizzas, chips, and soda here. Oh, and some ice cream.(23:24:50 2021/07/07)
Amy: /Everyone Thank you Jess. But we we are on a diet, so you know...Thank you anyway.(23:26:14 2021/07/07)
Jessica: /Everyone OK(23:26:36 2021/07/07)
Jessica: /Everyone I gotta go. See you!(23:27:17 2021/07/07)

```

8. Disconnect to Server.

Client Jessica disconnected by clicking the disconnect button.

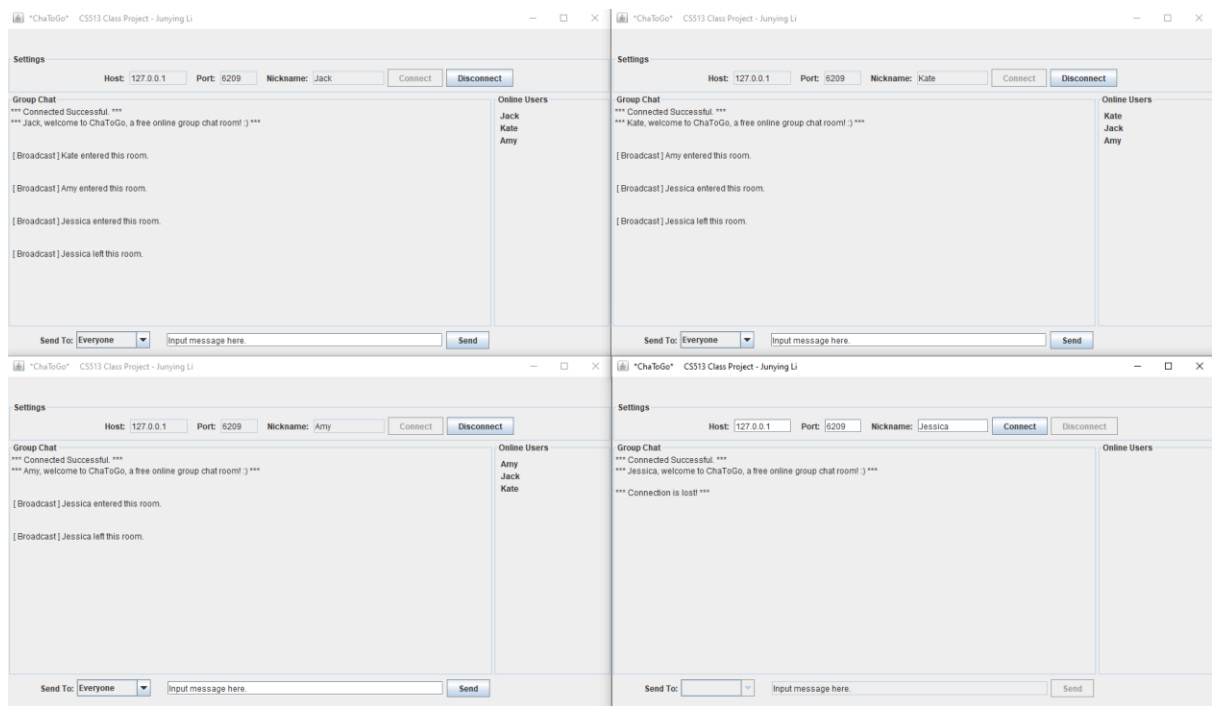
- Disconnected Client



- Other online clients

Online clients receive broadcast to inform that Jessica left the room.

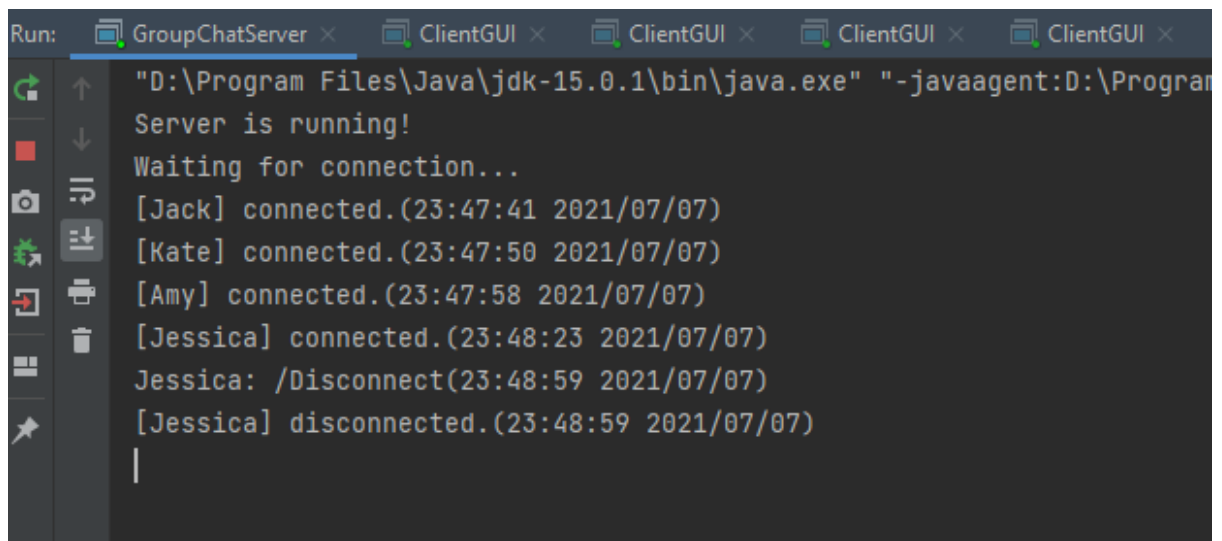
Online user list refreshed. "Jessica" has been removed from user list.



- Server

Server prints out disconnect request.

Server is stable after client termination.

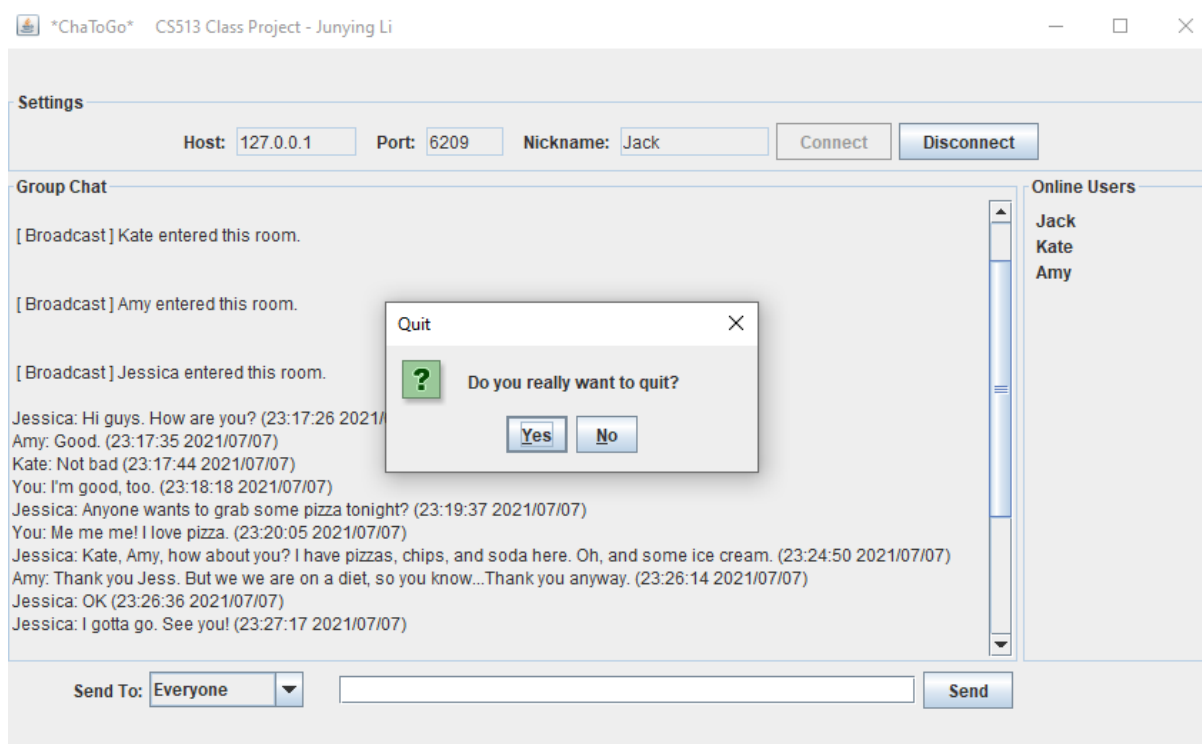


Disconnect by closing Client GUI's window.

Client Jack disconnected by closing window.

- Disconnected Client

When user closes the window, it popped up a confirm dialog. If user chooses “Yes”, the window will close and the client will be terminated.

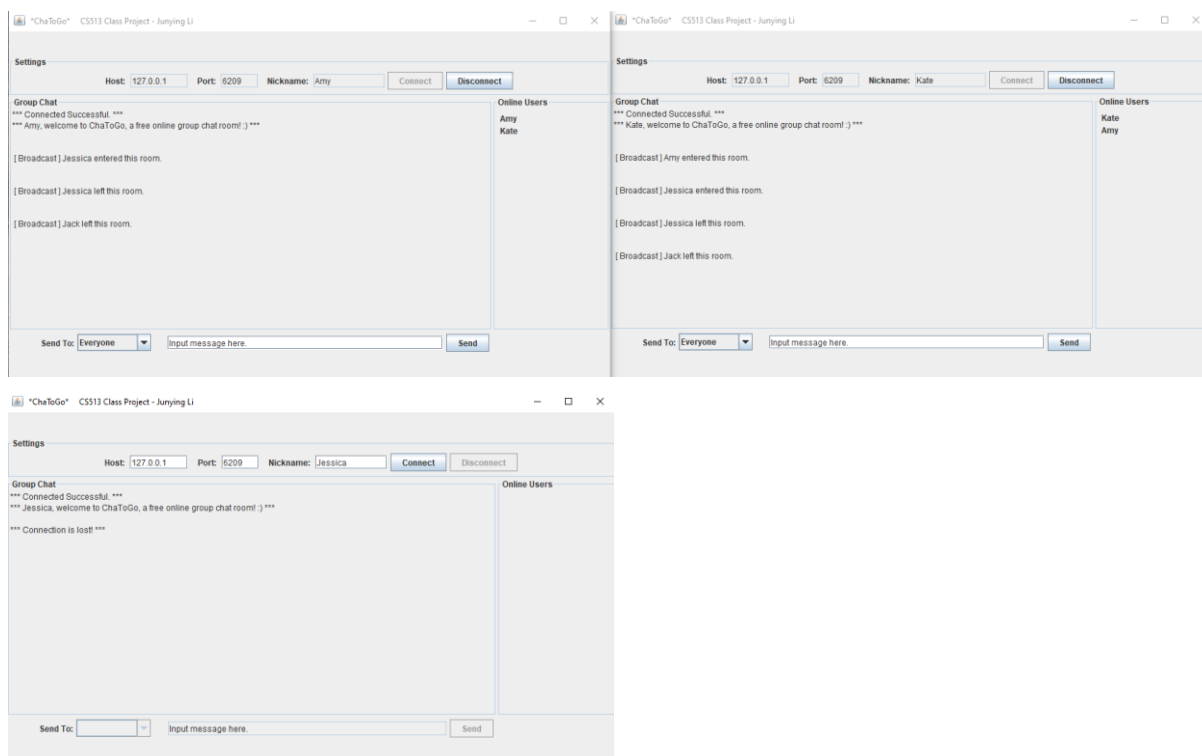


- Other online clients

Online clients receive broadcast to inform that Jack left this room.

Disconnected client doesn't receive anything. Because all opened sockets and streams closed on client after termination.

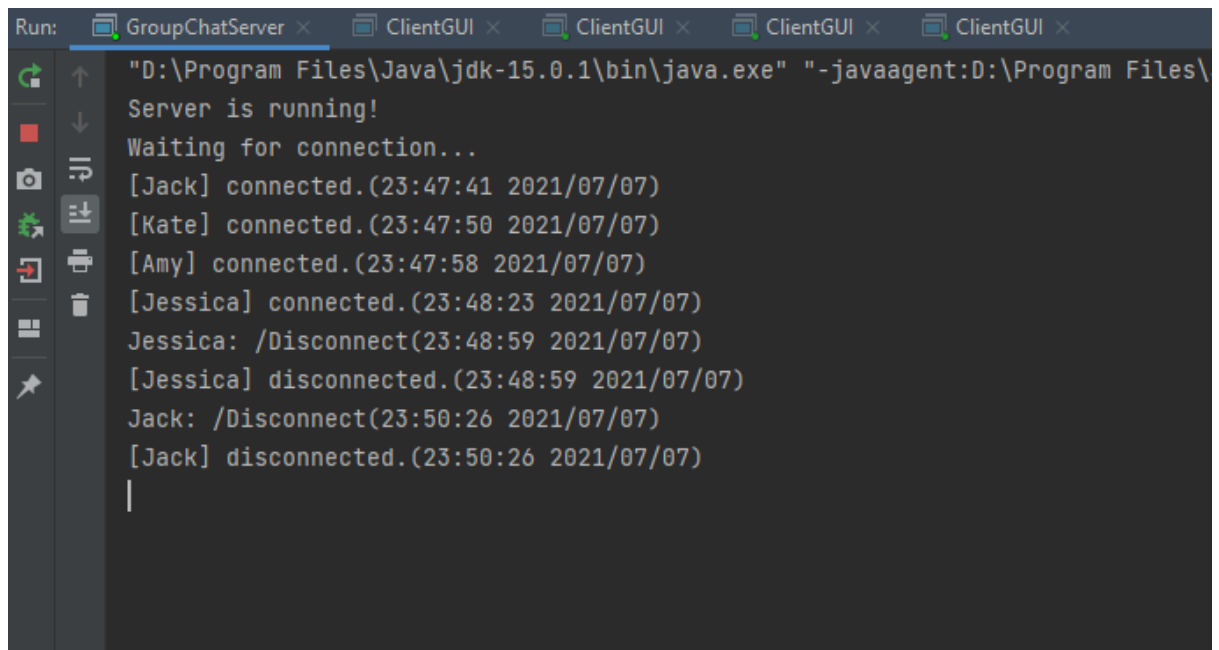
Online user list refreshed. "Jack" has been removed from online user list.



- Server

Server is stable after client termination.

Server prints out Jack's disconnect request.



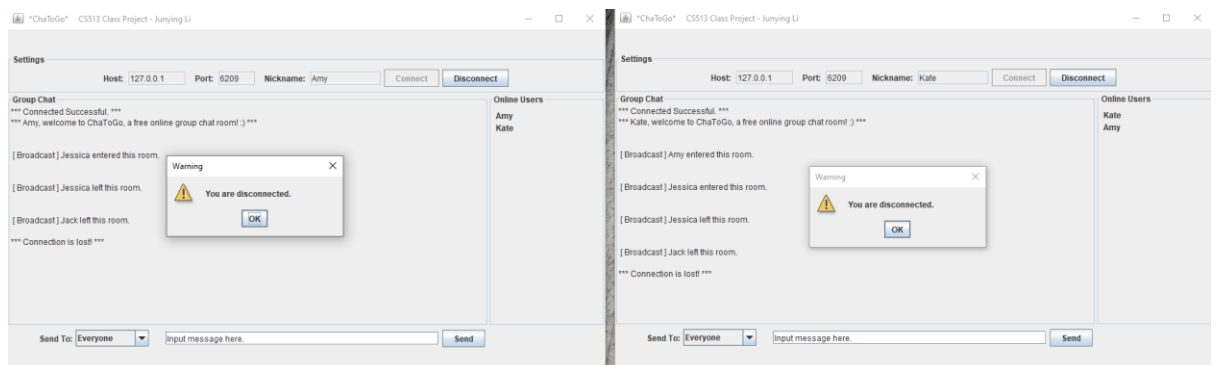
```
Run: GroupChatServer x ClientGUI x ClientGUI x ClientGUI x ClientGUI x
"D:\Program Files\Java\jdk-15.0.1\bin\java.exe" "-javaagent:D:\Program Files\
Server is running!
Waiting for connection...
[Jack] connected.(23:47:41 2021/07/07)
[Kate] connected.(23:47:50 2021/07/07)
[Amy] connected.(23:47:58 2021/07/07)
[Jessica] connected.(23:48:23 2021/07/07)
Jessica: /Disconnect(23:48:59 2021/07/07)
[Jessica] disconnected.(23:48:59 2021/07/07)
Jack: /Disconnect(23:50:26 2021/07/07)
[Jack] disconnected.(23:50:26 2021/07/07)
```

9. Server termination

- Online clients

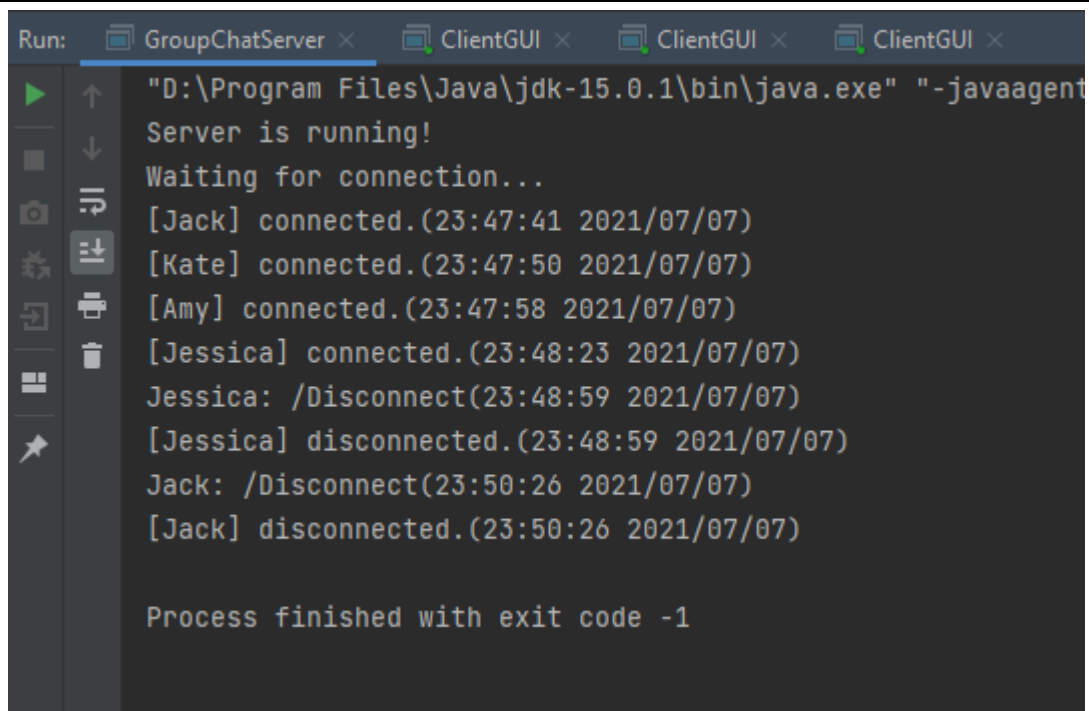
Clients are stable after server termination.

Clients pop up warning windows to inform that this client is disconnected.



- Server

Server terminated.



```
Run: GroupChatServer x ClientGUI x ClientGUI x ClientGUI x
"D:\Program Files\Java\jdk-15.0.1\bin\java.exe" "-javaagent
Server is running!
Waiting for connection...
[Jack] connected.(23:47:41 2021/07/07)
[Kate] connected.(23:47:50 2021/07/07)
[Amy] connected.(23:47:58 2021/07/07)
[Jessica] connected.(23:48:23 2021/07/07)
Jessica: /Disconnect(23:48:59 2021/07/07)
[Jessica] disconnected.(23:48:59 2021/07/07)
Jack: /Disconnect(23:50:26 2021/07/07)
[Jack] disconnected.(23:50:26 2021/07/07)

Process finished with exit code -1
```

3. Codes

1. server/GroupChatServer.java

```
package server;

import java.io.*;
import java.net.*;
import java.util.*;

public class GroupChatServer {

    private static ServerSocket serverSocket;

    private static Socket socket;

    public static List<GroupChatServerThread> threadList = new
    ArrayList<GroupChatServerThread>();

    private static void runServer() {

        try {

            serverSocket = new ServerSocket(6209);

            System.out.println("Server is running!");

            System.out.println("Waiting for connection...");

            while (true) {

                socket = serverSocket.accept();

                GroupChatServerThread thread = new GroupChatServerThread(socket);

                thread.start();
            }
        }
    }
}
```

```
        threadList.add(thread);  
    }  
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}  
  
public static void main(String[] args) {  
    runServer();  
}  
}
```

2. server/GroupChatServerThread.java

```
package server;  
  
import java.io.*;  
import java.net.*;  
import java.text.SimpleDateFormat;  
import java.util.*;  
  
@SuppressWarnings("Duplicates")  
public class GroupChatServerThread extends Thread {  
    private final Socket socket;  
    private InputStream input;  
    private OutputStream output;  
    private BufferedReader bufferedReader;  
    private BufferedWriter bufferedWriter;  
    private String username;  
    private boolean isConnected = true;  
  
    public GroupChatServerThread(Socket socket) {  
        this.socket = socket;  
    }  
  
    @Override  
    public void run() {  
        try {  
            input = socket.getInputStream();  
            output = socket.getOutputStream();  

```

```
bufferedReader = new BufferedReader(new InputStreamReader(input));
bufferedWriter = new BufferedWriter(new OutputStreamWriter(output));
username = bufferedReader.readLine();

if (checkUsernameExistence(username)) {
    sendInfo(bufferedWriter, "/isExistedUsername");
    try {
        if (bufferedWriter != null) {
            bufferedWriter.close();
        }
        if (bufferedReader != null) {
            bufferedReader.close();
        }
        if (input != null) {
            input.close();
        }
        if (output != null) {
            output.close();
        }
        if (socket != null) {
            socket.close();
        }
        GroupChatServer.threadList.remove(this);
    } catch (IOException e) {
        e.printStackTrace();
    }
} else {
    welcome();
    chat();
    closeAll();
    goodBye();
};
} catch (IOException e) {
    e.printStackTrace();
}
}

private boolean checkUsernameExistence (String username) {
    boolean isExistedUsername = false;
```

```
for (GroupChatServerThread thread : GroupChatServer.threadList) {
    if (username.equals(thread.username) && thread != this) {
        isExistedUsername = true;
        break;
    }
}

return isExistedUsername;
}

private void welcome () {
    sendInfo(bufferedWriter, "\n*** Connected Successful. *** \n*** " + username +
    ", welcome to ChaToGo, a free online group chat room! :) *** \n");
    System.out.println "[" + username + "] connected." + getTime();
    for (GroupChatServerThread thread : GroupChatServer.threadList) {
        if (thread != this) {
            // Broadcast who's coming in
            sendInfo(thread.bufferedWriter, "\n [ Broadcast ] " + username + "
entered this room.\n");
            addUser(thread.bufferedWriter, username);
            addUser(this.bufferedWriter, thread.username);
        }
    }
}

private void chat () {
    String msg = null;
    try {
        msg = bufferedReader.readLine();
        System.out.println(username + ": " + msg + getTime());
    } catch (IOException e) {
        isConnected = false;
    }
    while (!msg.equals("/Disconnect") && isConnected) {
        for (GroupChatServerThread thread : GroupChatServer.threadList) {
            String[] m = msg.split(" ");
            String receiver = m[0].substring(1);
            String message = msg.substring(m[0].length()+1);
            if (receiver.equals("Everyone")) {
                if (!thread.username.equals(username)) {
```

```
        sendInfo(thread.bufferedWriter, username + ": " + message + " " +
getTime());

    }

    } else {

        if (thread.username.equals(receiver)) {

            sendInfo(thread.bufferedWriter, username + " whispers to you: " +
message + " " + getTime());

        }

    }

}

try {

    msg = bufferedReader.readLine();

    System.out.println(username + ": " + msg + getTime());

} catch (IOException e) {

    isConnected = false;

}

}

sendInfo(bufferedWriter, "/Disconnect");

}

private void goodBye() {

    // Broadcast who left the room

    for (GroupChatServerThread thread : GroupChatServer.threadList) {

        if (thread != this) {

            sendInfo(thread.bufferedWriter, "\n [ Broadcast ] " + username + " left
this room.\n");

            removeUser(thread.bufferedWriter, username);

        }

    }

}

private void sendInfo(BufferedWriter bWriter, String msg) {

    try {

        bWriter.write(msg);

        bWriter.newLine();

        bWriter.flush();

    } catch (IOException e) {

        // e.printStackTrace();

        isConnected = false;

    }

}
```



```
    }  
}  
  
private void addUser (BufferedWriter bWriter, String username) {  
    sendInfo(bWriter, "/add " + username);  
}  
  
private void removeUser (BufferedWriter bWriter, String username) {  
    sendInfo(bWriter, "/remove " + username);  
}  
  
private String getTime () {  
    @SuppressWarnings("SimpleDateFormat") SimpleDateFormat time = new  
SimpleDateFormat();  
    time.applyPattern("HH:mm:ss yyyy/MM/dd");  
    Date date = new Date();  
    return "(" + time.format(date) + ")";  
}  
  
// Close all opened sockets and streams on client after its termination.  
private void closeAll () {  
    try {  
        if (bufferedWriter != null) {  
            bufferedWriter.close();  
        }  
        if (bufferedReader != null) {  
            bufferedReader.close();  
        }  
        if (input != null) {  
            input.close();  
        }  
        if (output != null) {  
            output.close();  
        }  
        if (socket != null) {  
            socket.close();  
            System.out.println "[" + username + "] disconnected." + getTime());  
        }  
        GroupChatServer.threadList.remove(this);  
    }  
}
```

```
    } catch (IOException e) {  
        e.printStackTrace();  
    }  
}  
}
```

3. client/ClientFrame.java

```
package client;  
  
import javax.swing.*;  
import java.awt.event.WindowEvent;  
import java.awt.event.WindowListener;  
  
public class ClientFrame extends JFrame implements WindowListener {  
    @Override  
    public void windowOpened(WindowEvent e) {  
  
    }  
  
    @Override  
    public void windowClosing(WindowEvent e) {  
  
    }  
  
    @Override  
    public void windowClosed(WindowEvent e) {  
  
    }  
  
    @Override  
    public void windowIconified(WindowEvent e) {  
  
    }  
  
    @Override  
    public void windowDeiconified(WindowEvent e) {  
  
    }  
}
```

```
@Override

public void windowActivated(WindowEvent e) {

}

@Override

public void windowDeactivated(WindowEvent e) {

}

}
```

4. client/ClientGUI.java

```
package client;

import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.Socket;
import java.net.UnknownHostException;
import java.text.SimpleDateFormat;
import java.util.Date;
import javax.swing.*;

@SuppressWarnings("Duplicates")
public class ClientGUI extends JFrame {

    private static Socket client = null;
    private BufferedWriter bufferedWriter;
    private BufferedReader bufferedReader;
    private ClientThread clientThread;
    private static boolean isConnected = false;
    private final DefaultListModel<String> userList;
    private final JList<String> userListBoard;

    private final JComboBox<String> usernameBox = new JComboBox<>();
    private final JButton sendButton = new JButton("Send");
    private final JButton connectButton = new JButton("Connect");
    private final JButton disconnectButton = new JButton("Disconnect");
    private final JTextField sendInputTF = new JTextField("Input message here.", 40);
    private final JTextArea messageBoard = new JTextArea(20, 40);
```

```
private final JLabel hostLabel = new JLabel("Host: ");
private final JLabel portLabel = new JLabel("  Port: ");
private final JLabel usernameLabel = new JLabel("  Nickname: ");
private final JLabel sendTo = new JLabel("          Send To:");
private final JLabel blank = new JLabel("      ");
private final JTextField hostTF = new JTextField("127.0.0.1", 8);
private final JTextField portTF = new JTextField("6209", 5);
private final JTextField usernameTF = new JTextField(10);
private final JPanel headPanel = new JPanel();
private final JPanel sendPanel = new JPanel();
private final JScrollPane messagePanel = new JScrollPane();
private final JPanel userPanel = new JPanel();

public ClientGUI () throws UnknownHostException {
    messageBoard.setEditable(false);
    userList = new DefaultListModel<>();
    userListBoard = new JList<>(userList);
    usernameBox.setPrototypeDisplayValue("XXXXXXXXXX");
    usernameBox.setEnabled(false);
    usernameBox.setEditable(false);
    disconnectButton.setEnabled(false);
    sendButton.setEnabled(false);
    sendInputTF.setEditable(false);

    sendButton.addActionListener(e -> {
        if (!isConnected) {
            JOptionPane.showMessageDialog(null, "Please connect to the server.",
"Warning", JOptionPane.WARNING_MESSAGE);
        } else {
            String msg = sendInputTF.getText().trim();
            if (msg.isEmpty()) {
                JOptionPane.showMessageDialog(null, "You can't send a blank message.",
"Warning", JOptionPane.WARNING_MESSAGE);
            } else {
                try {
                    bufferedWriter.write("/" + usernameBox.getSelectedItem() + " " +
msg);

                    bufferedWriter.newLine();
                    bufferedWriter.flush();
```

```
        } catch (IOException e1) {
            e1.printStackTrace();
        }

        if (usernameBox.getSelectedItem().equals("Everyone")) {
            messageBoard.append("You: " + msg + " " + getTime() + "\n");
        } else {
            messageBoard.append("You whisper to " +
usernameBox.getSelectedItem() + ": " + msg + " " + getTime() + "\n");
        }

        sendInputTF.setText(null);
    }
}

connectButton.addActionListener(e -> {
    if (!isConnected) {
        String username = usernameTF.getText();
        // Check username validation
        if (checkNameValidation(username)) {
            connectToServer();
            if (!isConnected) {
                return;
            }
            try {
                bufferedWriter.write(username);
                bufferedWriter.newLine();
                bufferedWriter.flush();
            } catch (IOException e1) {
                e1.printStackTrace();
            }

            // Check username existed or not
            String serverMsg = readMsg();
            //
            messageBoard.append(serverMsg + "\n");
            if (checkUserExistence(serverMsg)) {
                JOptionPane.showMessageDialog(null, "This nickname has already
been taken, please re-entry!", "Warning", JOptionPane.WARNING_MESSAGE);
                closeAll(false);
                setAfterDisconnect();
            }
        }
    }
});
```

```
        } else {  
            // Start thread  
            clientThread = new ClientThread(client, bufferedReader,  
bufferedWriter, messageBoard, userList, usernameBox);  
            clientThread.start();  
            userList.addElement(usernameTF.getText());  
            setAfterConnect();  
        }  
    }  
}  
});  
  
disconnectButton.addActionListener(e -> {  
    if (isConnected) {  
        disconnectToServer();  
        setAfterDisconnect();  
    } else {  
        JOptionPane.showMessageDialog(null, "Please connect to the server first.",  
"Warning", JOptionPane.WARNING_MESSAGE);  
    }  
});  
}  
  
private boolean checkNameValidation(String username) {  
    boolean validation = false;  
    if (username.isEmpty()) {  
        JOptionPane.showMessageDialog(null, "Please entry a nickname.", "Warning",  
JOptionPane.WARNING_MESSAGE);  
    }  
    else if (username.contains(" ")) {  
        JOptionPane.showMessageDialog(null, "Nickname can't contain space.",  
"Warning", JOptionPane.WARNING_MESSAGE);  
    }  
    else if (username.equals("Everyone")) {  
        JOptionPane.showMessageDialog(null, "You can't use 'Everyone' as nickname.",  
"Warning", JOptionPane.WARNING_MESSAGE);  
    } else {  
        validation = true;  
    }  
}
```

```
        return validation;
    }

    private boolean checkUserExistence(String serverMsg) {
        return serverMsg.equals("/isExistedUsername");
    }

    private void connectToServer() {
        try {
            String host = hostTF.getText().trim();
            String stringPort = portTF.getText();

            if (host.isEmpty() || stringPort.isEmpty()) {
                JOptionPane.showMessageDialog(null, "Please make sure to input correct
host and port address.", "Warning", JOptionPane.WARNING_MESSAGE);
            } else {
                int port = Integer.parseInt(stringPort);
                client = new Socket(host, port);

                try {
                    if (client != null) {
                        bufferedReader = new BufferedReader(new
InputStreamReader(client.getInputStream()));
                        bufferedWriter = new BufferedWriter(new
OutputStreamWriter(client.getOutputStream()));
                        isConnected = true;
                    }
                } catch (IOException e) {
                    JOptionPane.showMessageDialog(null, "Connection failed. Please check
host and port address.", "Warning", JOptionPane.WARNING_MESSAGE);
                    isConnected = false;
                    closeAll(false);
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(null, "Connection failed. Please check host
and port address.", "Warning", JOptionPane.WARNING_MESSAGE);
            isConnected = false;
        }
    }
}
```

```
private String readMsg() {
    String msg = "\n*** Connection is lost! ***\n";
    // 0703
    if (!isConnected) {
        closeAll(true);
        setAfterDisconnect();
        return msg;
    }
    //
    try {
        msg = this.bufferedReader.readLine();
    } catch (IOException e) {
        //      e.printStackTrace();
        isConnected = false;
        closeAll(true);
        setAfterDisconnect();
    }
    return msg;
}

private void disconnectToServer() {
    try {
        bufferedWriter.write("/Disconnect");
        bufferedWriter.newLine();
        bufferedWriter.flush();
        //      clientThread.stop();
        closeAll(false);
        setAfterDisconnect();
        isConnected = false;
    } catch (IOException e) {
        e.printStackTrace();
    }
}

// Close all opened Sockets, buffered writer, and buffered reader.
private void closeAll(boolean warning) {
    try {
        if (bufferedWriter != null) {
```



```
        bufferedWriter.close();
    }

    if (bufferedReader != null) {
        bufferedReader.close();
    }

    if (client != null) {
        client.close();
    }

    assert client != null;

    if (client.isClosed() && warning) {
        JOptionPane.showMessageDialog(null, "Connection is closed.", "Warning",
JOptionPane.WARNING_MESSAGE);

        isConnected = false;
    }

    } catch (IOException e) {
//        e.printStackTrace();
    }
}

private void setAfterConnect() {
    usernameTF.setEditable(false);
    hostTF.setEditable(false);
    portTF.setEditable(false);
    connectButton.setEnabled(false);
    disconnectButton.setEnabled(true);
    sendInputTF.setEditable(true);
    sendButton.setEnabled(true);
    usernameBox.setEnabled(true);
    usernameBox.setEditable(false);
    setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
}

private void setAfterDisconnect() {
    usernameTF.setEditable(true);
    hostTF.setEditable(true);
    portTF.setEditable(true);
    connectButton.setEnabled(true);
    disconnectButton.setEnabled(false);
    sendInputTF.setEditable(false);
}
```

```
        sendButton.setEnabled(false);

        usernameBox.setEnabled(false);

        usernameBox.removeAllItems();

        userList.removeAllElements();

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    private String getTime() {

        @SuppressWarnings("SimpleDateFormat") SimpleDateFormat time = new
SimpleDateFormat();

        time.applyPattern("HH:mm:ss yyyy/MM/dd");

        Date date = new Date();

        return "(" + time.format(date) + ")";
    }

    private void confirmQuitDialog() {

        int result = JOptionPane.showConfirmDialog(null, "Do you really want to quit?",
"Quit", JOptionPane.YES_NO_OPTION);

        if (result==JOptionPane.OK_OPTION) {

            if (isConnected) {

                disconnectToServer();

                setAfterDisconnect();

            }

            System.exit(0);

        }

        else {

            this.setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);

        }

    }

    @Override
    public void windowClosing(WindowEvent e) {

        confirmQuitDialog();

    }

    private void initGUI() {

        userListBoard.setPreferredSize(new Dimension(100, 310));

        userListBoard.setBackground(new Color(238, 238, 238));

        messageBoard.setBackground(new Color(238, 238, 238));
```

```
messagePanel.setBackground(new Color(238, 238, 238));
messageBoard.setLineWrap(true);
userListBoard.setSelectionForeground(Color.BLUE);
userListBoard.setSelectionBackground(new Color(238, 238, 238));
this.setTitle(" *ChaToGo*      CS513 Class Project - Junying Li");

GridBagLayout layout = new GridBagLayout();
setLayout(layout);
this.add(headPanel);
this.add(messagePanel);
this.add(userPanel);
this.add(sendPanel);

GridBagConstraints constraint= new GridBagConstraints();
constraint.fill = GridBagConstraints.HORIZONTAL;
constraint.gridx = 0;
constraint.gridy = 0;
constraint.gridwidth = 3;
constraint.weightx = 0;
constraint.weighty = 0;
layout.setConstraints(headPanel, constraint);
constraint.gridx = 0;
constraint.gridy = 1;
constraint.gridwidth = 1;
constraint.weightx = 0;
constraint.weighty = 0;
layout.setConstraints(messagePanel, constraint);
constraint.gridx = 1;
constraint.gridy = 1;
constraint.gridwidth = 1;
constraint.weightx = 1;
constraint.weighty = 0;
layout.setConstraints(userPanel, constraint);
constraint.gridx = 0;
constraint.gridy = 2;
constraint.weightx = 0;
constraint.weighty = 0;
layout.setConstraints(sendPanel, constraint);
```

```
headPanel.setLayout(new FlowLayout(FlowLayout.CENTER));
headPanel.add(hostLabel);
headPanel.add(hostTF);
headPanel.add(portLabel);
headPanel.add(portTF);
headPanel.add(usernameLabel);
headPanel.add(usernameTF);
headPanel.add(connectButton);
headPanel.add(disconnectButton);

sendPanel.setLayout(new FlowLayout(FlowLayout.CENTER));
sendPanel.add(sendTo);
sendPanel.add(usernameBox);
sendPanel.add(blank);
sendPanel.add(sendInputTF);
sendPanel.add(sendButton);

userPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
userPanel.add(userListBoard);

headPanel.setBorder(BorderFactory.createTitledBorder("Settings"));
messagePanel.setViewportView(messageBoard);
messagePanel.setBorder(BorderFactory.createTitledBorder("Group Chat"));
userPanel.setBorder(BorderFactory.createTitledBorder("Online Users"));
}

public static void main(String[] args) throws IOException {
    int width = 900;
    int height = 550;
    ClientGUI client = new ClientGUI();
    client.setSize(width, height);
    int screenWidth = (int)client.getToolkit().getScreenSize().getWidth();
    int screenHeight = (int)client.getToolkit().getScreenSize().getHeight();
    client.setLocation((screenWidth-width)/2, (screenHeight-height)/2);
    client.initGUI();
    client.addWindowListener(client);
    client.setVisible(true);
}
}
```

5. client/ClientThread.java

```
package client;

import javax.swing.*;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.Closeable;
import java.io.IOException;
import java.net.Socket;

public class ClientThread extends Thread {
    private final Socket client;
    private final BufferedReader bufferedReader;
    private final BufferedWriter bufferedWriter;
    private final JTextArea messageBoard;
    private final DefaultListModel<String> userList;
    private final JComboBox<String> usernameBox;
    boolean isRunning = true;

    public ClientThread (Socket client,
                        BufferedReader bufferedReader,
                        BufferedWriter bufferedWriter,
                        JTextArea messageBoard,
                        DefaultListModel<String> userList,
                        JComboBox<String> usernameBox) {
        this.client = client;
        this.bufferedReader = bufferedReader;
        this.bufferedWriter = bufferedWriter;
        this.messageBoard = messageBoard;
        this.userList = userList;
        this.usernameBox = usernameBox;
    }

    @Override
    public void run() {
        usernameBox.addItem("Everyone");
        while (isRunning) {
            String receiveMsg = readMsg();
```

```
        if (receiveMsg.startsWith("/add")) {
            String newUser = receiveMsg.substring(5);
            usernameBox.addItem(newUser);
            userList.addElement(newUser);
            continue;
        }

        if (receiveMsg.startsWith("/remove")) {
            String rmUser = receiveMsg.substring(8);
            usernameBox.removeItem(rmUser);
            userList.removeElement(rmUser);
            continue;
        }

        if (receiveMsg.startsWith("/userList")) {
            String[] list = receiveMsg.substring(10).split(",");
            for (String user : list) {
                userList.addElement(user);
            }
            continue;
        }

        if (receiveMsg.startsWith("/Disconnect")) {
            break;
        }

        if (receiveMsg.startsWith("/isExistedUsername")) {
            JOptionPane.showMessageDialog(null, "This username is already taken,
please re-entry!", "Warning", JOptionPane.WARNING_MESSAGE);
            break;
        }

        messageBoard.append(receiveMsg + "\n");
    }

    // Close all opened Sockets, buffered writer, and buffered reader.
    try {
        if (bufferedWriter != null) {
            bufferedWriter.close();
        }

        if (bufferedReader != null) {
            bufferedReader.close();
        }

        if (client != null) {

```

```
        client.close();
    }

    assert client != null;

    if (client.isClosed()) {
        isRunning = false;

        JOptionPane.showMessageDialog(null, "You are disconnected.", "Warning",
JOptionPane.WARNING_MESSAGE);
    }

    } catch (IOException e) {
        e.printStackTrace();
    }
}

private String readMsg () {
    String msg = "*** Connection is lost! ***\n";

    if (!isRunning) {
        closeAll(bufferedReader);

        return msg;
    }

    try {
        msg = this.bufferedReader.readLine();
    } catch (IOException e) {
        //        e.printStackTrace();

        isRunning = false;

        closeAll(bufferedReader);
    }

    return msg;
}

// Close all closeable IOs.
private static void closeAll(Closeable...io) {
    for (Closeable temp : io) {
        try {
            if (null!=temp) {
                temp.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
        }  
    }  
}
```