Programming Project Report: A Client-Server Chat Program

Thi Quynh Ha Nguyen, IMGD WPI’23

Worcester Polytechnic Institute

Author Note

Date: 7/6/20

Abstract

This report outlines the design and development of a multiple client – server chat program which follow these requirements:

- Multiple clients
- No GUI is needed for the server
- A simple GUI can be implemented for the server
- Clients must be able to choose a nickname
- Clients must be able to "whisper" to each other without having messages displayed to other users

The program was written in Java and Windows socket commands. The design and program are modular in nature and make maximum use of abstract data types and of software re-use. Particular attention is paid to Client Interface and experience. The report includes the test strategies used to verify the correct operation of the program, as well as the entire code. This project was developed in IntelliJ IDEA Community Edition 2019.3.4 x64.

Content Page

This report contains these sections:

Project Description

This project is a **multiple client server chat application** based on the client-server model using Java and Windows socket commands. A chat application has two basic components: server and client. The project itself has six classes. The chat application is a chat room, where multiple clients in the same port and ip address can send texts to each other via the chat program. Clients can send messages to the whole chat channel or privately send message to a pacific person in the chat program. This project was developed using IntelliJ IDEA Community Edition 2019.3.4 x64.

Detail Design

The project contains of 6 different classes with different function to run the application:

- **ClientThread:**

  ClientThread consists of the socket the clients communicate through and an array containing all other threads in the chatroom. This java class deals with client problem like checking duplicate name, enter clients name, managing the list of online clients, welcoming new clients, broadcast message, category message….

- **ChatView:**

  This class sets up the Chat Window GUI for client. There are chat windows for each client when entering the channel.

- **ChatServer:**

  ChatServer is in charged of setting up the server after the host starting it and running the server.

- **ChatProgram: run the Chat Program.**

  ChatProgram opens the socket and let host to start the server and clients to enter their desired sever.

- **ChatModel:**

ChatModel starts Client Interface by letting users choosing their desired IP address and port to enter the chat. Then it stores customers info in the thread and starts the Chat Window.

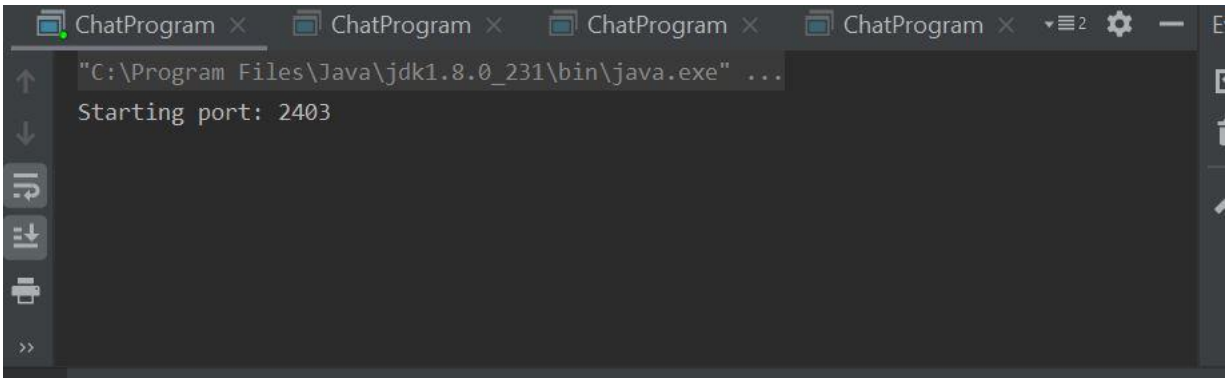
- **ChatClient:**

ChatClient handles creating socket, and receiving thread, sending message and closing the socket.

Testing

1. Test Set up server: This test is to check if the port number is smaller than 1024 or greater than 65535, the server will notify the host that the listed port is invalid. (Succeed)

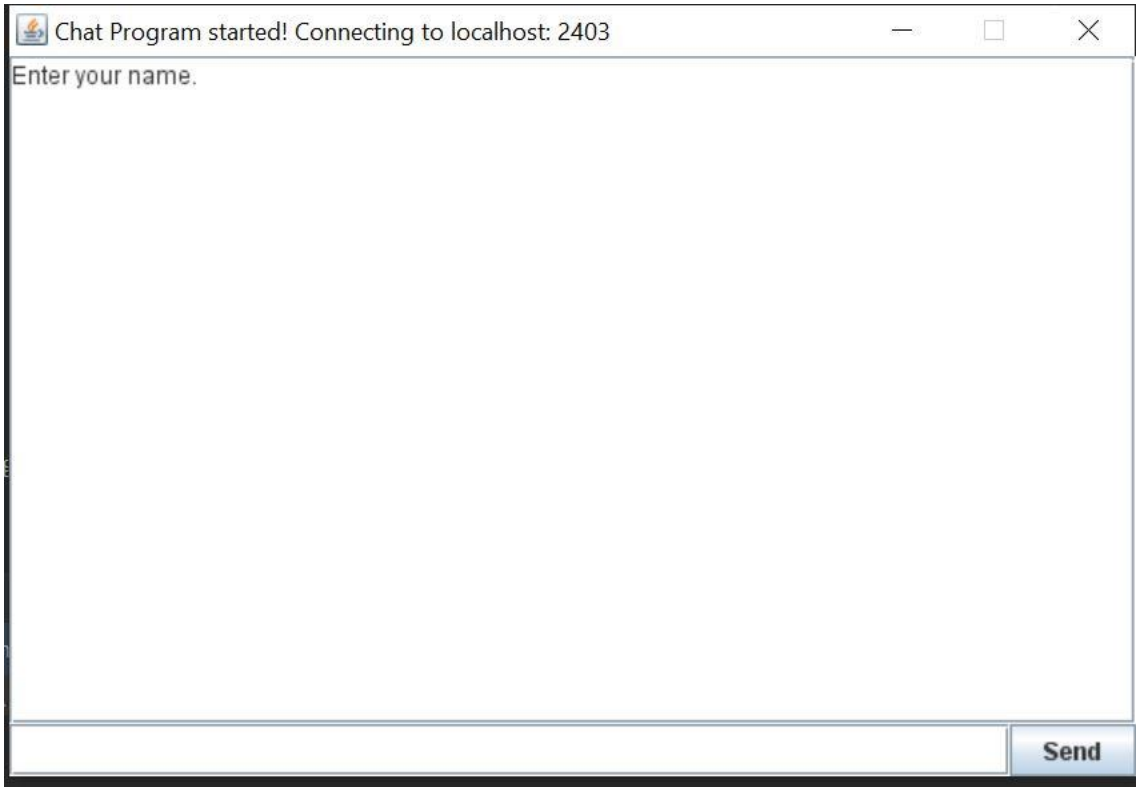
<If wrong port is entered>



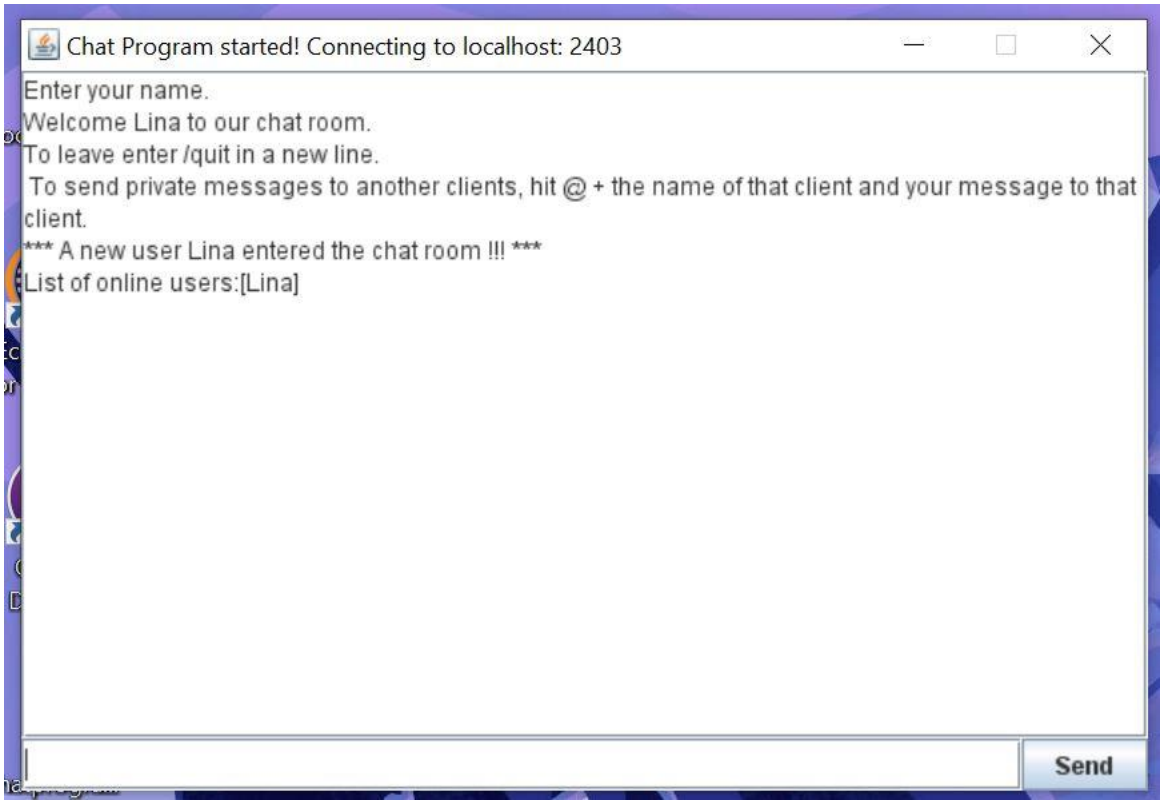
If the right port is entered, System will announce that the port is starting.

2. Client GUI (Succeed)

   This test checks if client enter the right ip address and port, client GUI starts! If the wrong ip and/or port are entered, the server will not start GUI and report an error!

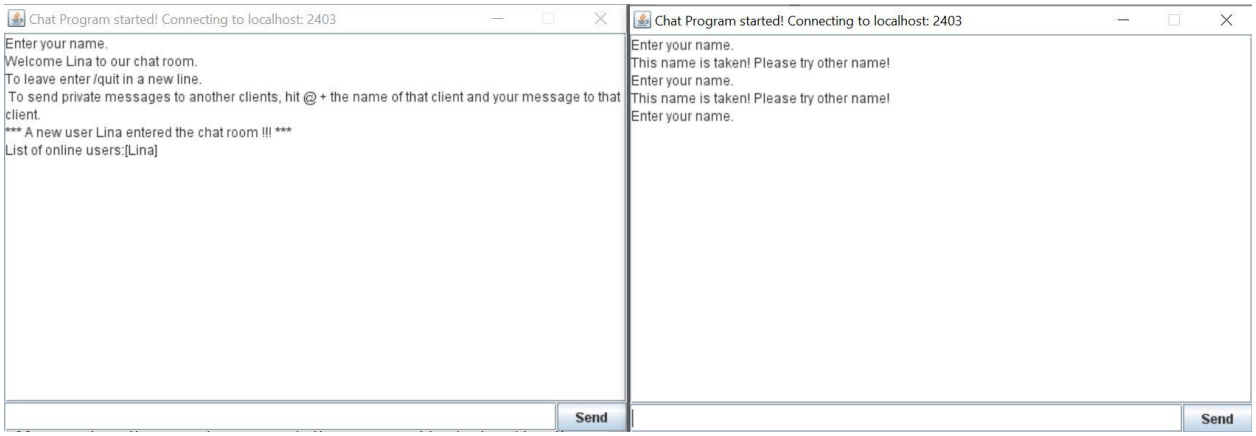



```
Cannot connect to localhost:2222
java.net.ConnectException: Connection refused: connect
    at java.net.DualStackPlainSocketImpl.connect0(Native Method)
    at java.net.DualStackPlainSocketImpl.socketConnect(DualStackPlainSocketImpl.java:79)
    at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:350)
    at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:206)
    at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:188)
    at java.net.PlainSocketImpl.connect(PlainSocketImpl.java:172)
    at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:392)
    at java.net.Socket.connect(Socket.java:606)
    at java.net.Socket.connect(Socket.java:555)
    at java.net.Socket.<init>(Socket.java:451)
    at java.net.Socket.<init>(Socket.java:228)
    at Client.ChatClient.InitSocket(ChatClient.java:22)
    at Client.ChatModel.setupClient(ChatModel.java:29)
    at ChatProgram.startAsClient(ChatProgram.java:50)
    at ChatProgram.main(ChatProgram.java:23)

Process finished with exit code 0
```



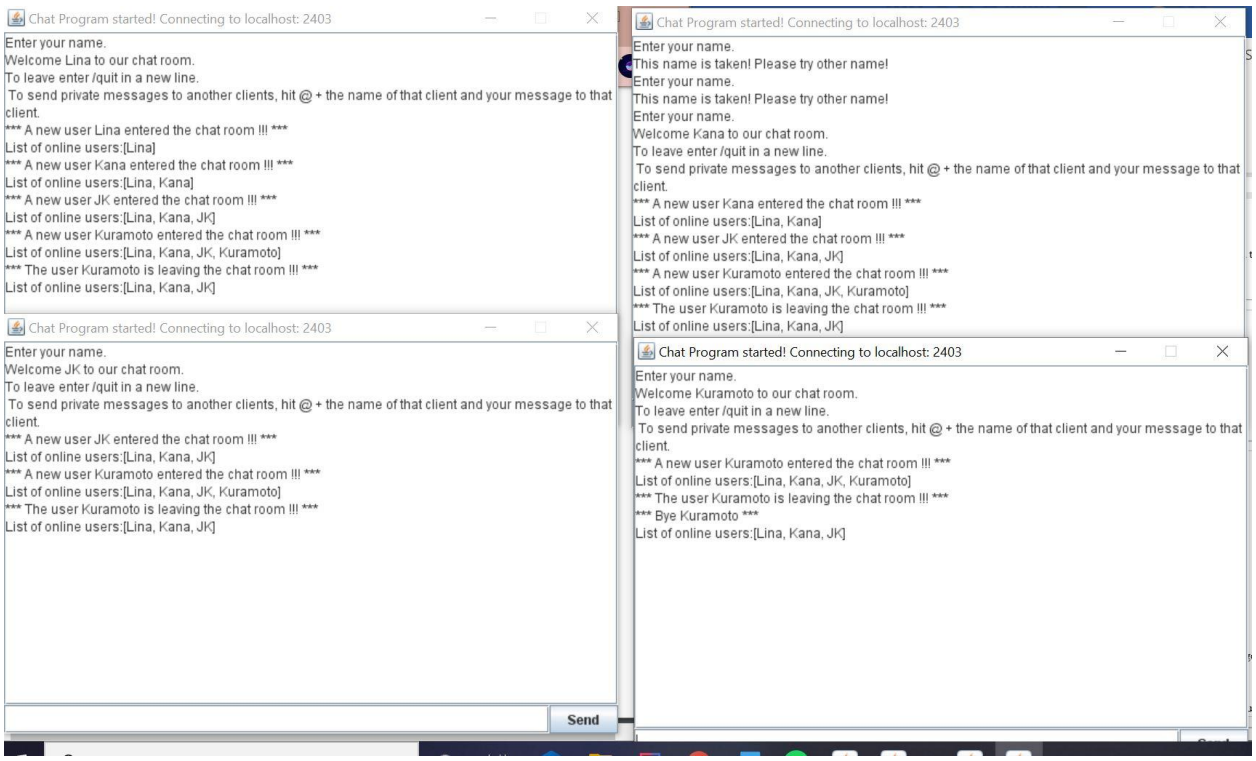
3. Check unique name, duplication. (Succeed)

   In the picture, if enter the name Lina, which is the name of one online user in the server, it notifies back that the name is taken and telling users to choose another name.



4. Check online users and update the list if there is new connection or disconnection.

   (Succeed)

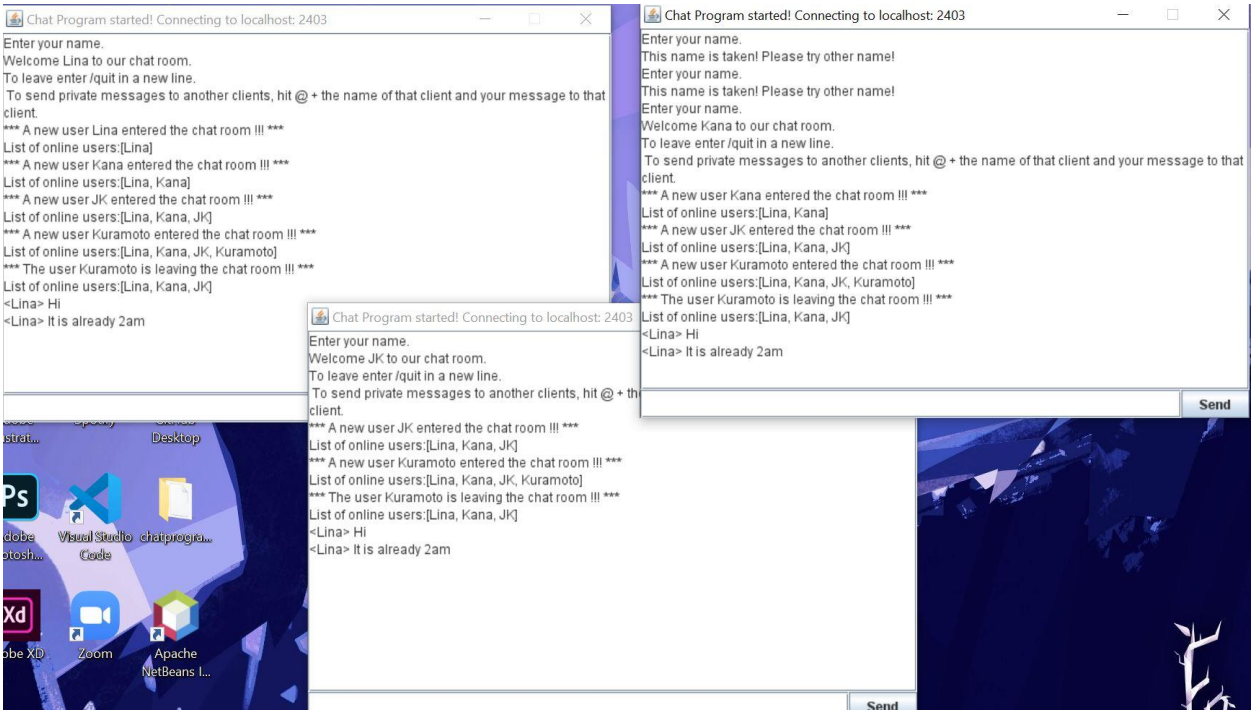


   List of online users is updated whenever there is a new client enter the port and update when clients leave the channel.

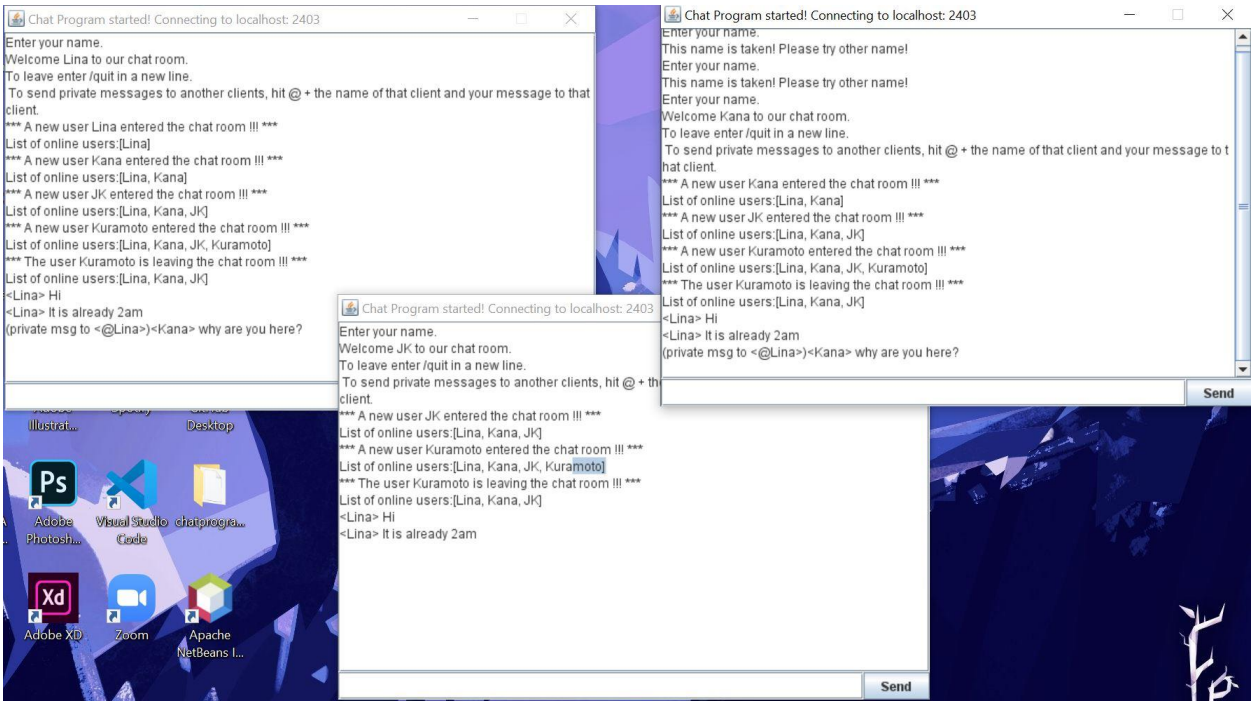
5. Check sending message publicly and privately: (Succeed)

   This test check if the message doesn't clarify a receiver then it will broadcast to the whole server. If the receiver is declared, the message will only be sent to the receiver and the sender.

- Sending message to the whole channel



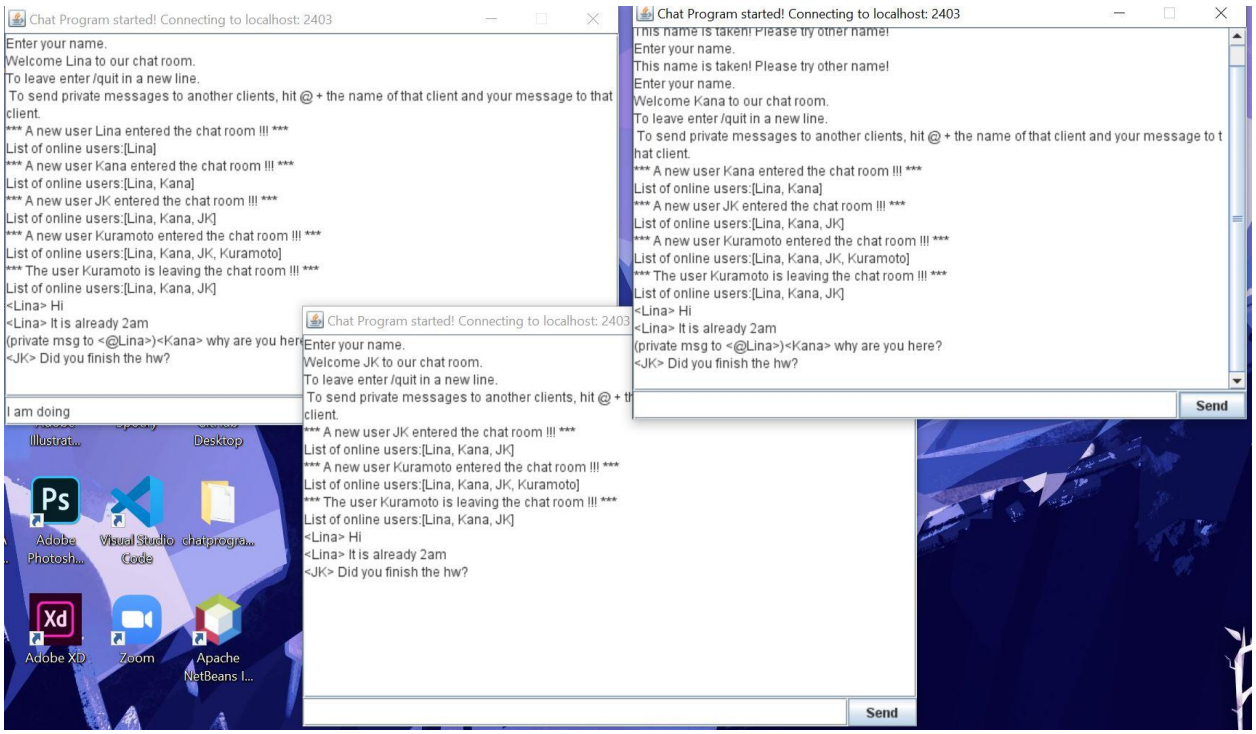
- Send message privately:



-

6. Check if able to receive messages / actions while typing a message. (Succeed)

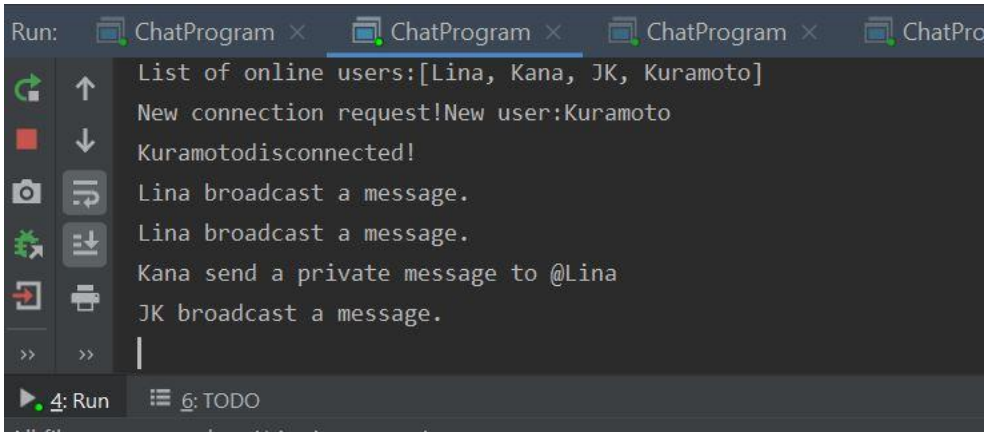
This test tests if a client is in proceed of typing her/his message, that person can still receive message from other clients.



7. Server Notification System + Server Stability after Clients Terminate

(Partly Succeed. Actions are displaced in the system. However, some unexpected null exception occurred if the client terminated by hit close or Alt + F4 instead of typing "/quit. But this exception didn't affect the stability of the server.)

This test is to see if actions of clients are printed out by the server. Also, to check if a client disconnect from the server, will the server function fine.

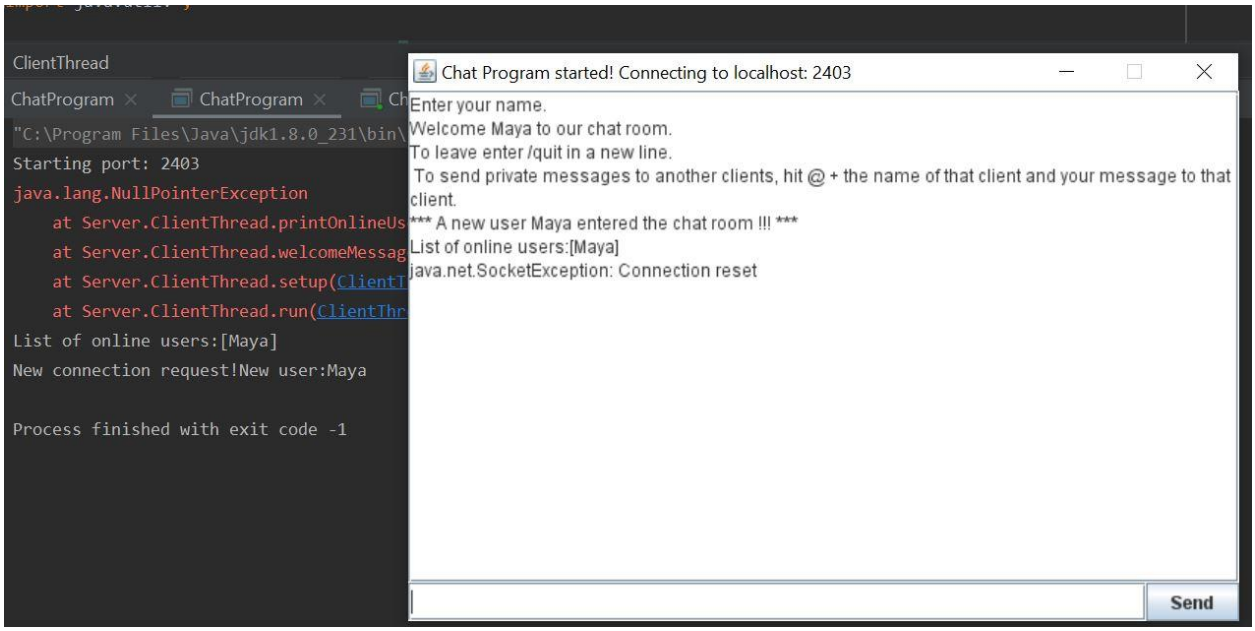



```
New connection request!New user:Kana
java.lang.NullPointerException
    at Server.ClientThread.printOnlineUsers(ClientThread.java:120)
    at Server.ClientThread.welcomeMessage(ClientThread.java:143)
    at Server.ClientThread.setup(ClientThread.java:77)
    at Server.ClientThread.run(ClientThread.java:54)
List of online users:[Lina, Kana, JK]
New connection request!New user:JK
java.lang.NullPointerException
    at Server.ClientThread.printOnlineUsers(ClientThread.java:120)
    at Server.ClientThread.welcomeMessage(ClientThread.java:143)
    at Server.ClientThread.setup(ClientThread.java:77)
    at Server.ClientThread.run(ClientThread.java:54)
List of online users:[Lina, Kana, JK, Kuramoto]
New connection request!New user:Kuramoto
Kuramotodisconnected!
Lina broadcast a message.
Lina broadcast a message.
Kana send a private message to @Lina
JK broadcast a message.
java.lang.NullPointerException
    at Server.ClientThread.runConversation(ClientThread.java:158)
```

```
Starting port: 2403
java.lang.NullPointerException
    at Server.ClientThread.printOnlineUsers(ClientThread.java:120)
    at Server.ClientThread.welcomeMessage(ClientThread.java:143)
    at Server.ClientThread.setup(ClientThread.java:77)
    at Server.ClientThread.run(ClientThread.java:54)
List of online users:[Lina]
New connection request!New user:Lina
java.lang.NullPointerException
    at Server.ClientThread.printOnlineUsers(ClientThread.java:120)
    at Server.ClientThread.welcomeMessage(ClientThread.java:143)
    at Server.ClientThread.setup(ClientThread.java:77)
    at Server.ClientThread.run(ClientThread.java:54)
List of online users:[Lina, Kana]
New connection request!New user:Kana
java.lang.NullPointerException
    at Server.ClientThread.printOnlineUsers(ClientThread.java:120)
    at Server.ClientThread.welcomeMessage(ClientThread.java:143)
    at Server.ClientThread.setup(ClientThread.java:77)
    at Server.ClientThread.run(ClientThread.java:54)
List of online users:[Lina, Kana, JK]
```

8. Client Stable after Server Termination. (Succeed)

   This test is to check if the server is terminated, will the client chap program break.



Future Development

A few possible future developments like ability to attach and send pictures, documents, … In current design, null exception pointer occurred when client instead of typing /quit, they hit close or Alt + F4 or a client cancel connection before enter the ip and port. An improved version of this project can maybe have a database of users containing username and password can be coupled with the design to main the account. A separate area to display lists of users.

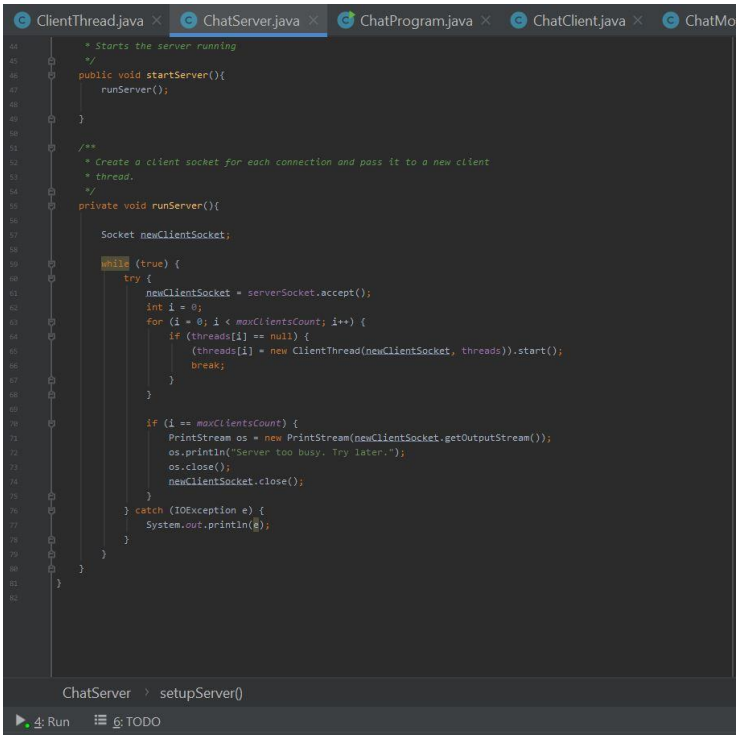
Conclusion – Solution Summary

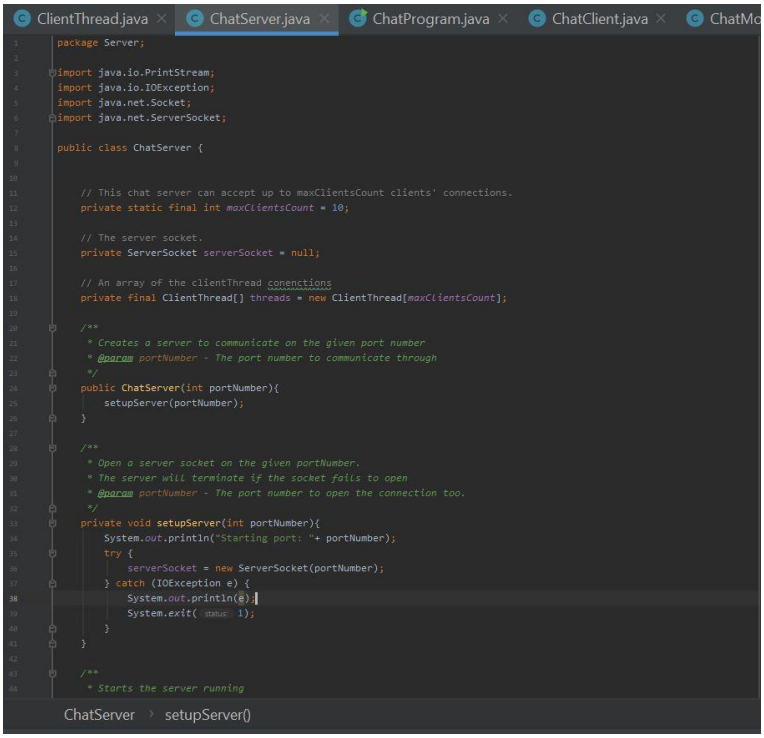
Chat application is very come used application among users. This report has described the successful design and development of a multiple client server chat application. With professional development, this application can come useful for communicating, exchange information, ….

Appendices

- Codes: https://github.com/quynhha/Multuple-Client-Server-Chat-Application

• ChatServer:


ClientThread.java
ChatServer.java
ChatProgram.java
ChatClient.java
ChatMo
* Starts the server running
*/
public void startServer(){
runServer();
}
/**
* Create a client socket for each connection and pass it to a new client
* thread.
*/
private void runServer(){
Socket newClientSocket;
while (true) {
try {
newClientSocket = serverSocket.accept();
int i = 0;
for (i = 0; i < maxClientsCount; i++) {
if (threads[i] == null) {
(threads[i] = new ClientThread(newClientSocket, threads)).start();
break;
}
}
if (i == maxClientsCount) {
PrintStream os = new PrintStream(newClientSocket.getOutputStream());
os.println("Server too busy. Try later.");
os.close();
newClientSocket.close();
}
} catch (IOException e) {
System.out.println(e);
}
}
}
}
ChatServer › setupServer()
4: Run
6: TODO


ClientThread.java
ChatServer.java
ChatProgram.java
ChatClient.java
ChatMo
package Server;
import java.io.PrintStream;
import java.io.IOException;
import java.net.Socket;
import java.net.ServerSocket;
public class ChatServer {
// This chat server can accept up to maxClientsCount clients' connections.
private static final int maxClientsCount = 10;
// The server socket.
private ServerSocket serverSocket = null;
// An array of the clientThread conenctions
private final ClientThread[] threads = new ClientThread[maxClientsCount];
/**
* Creates a server to communicate on the given port number
* @param portNumber - The port number to communicate through
*/
public ChatServer(int portNumber){
setupServer(portNumber);
}
/**
* Open a server socket on the given portNumber.
* The server will terminate if the socket fails to open
* @param portNumber - The port number to open the connection too.
*/
private void setupServer(int portNumber){
System.out.println("Starting port: "+ portNumber);
try {
serverSocket = new ServerSocket(portNumber);
} catch (IOException e) {
System.out.println(e);
System.exit(status: 1);
}
}
/**
* Starts the server running
ChatServer › setupServer()

- ClientThread:

```
    /**
     * Gets the name of the client
     * @return This users name with an '@' character prepended. If a name has not been set, null is returned
     */
    private String getClientName(){
        if(name != null){
            return '@' + name;
        }
        return null;
    }
}
```

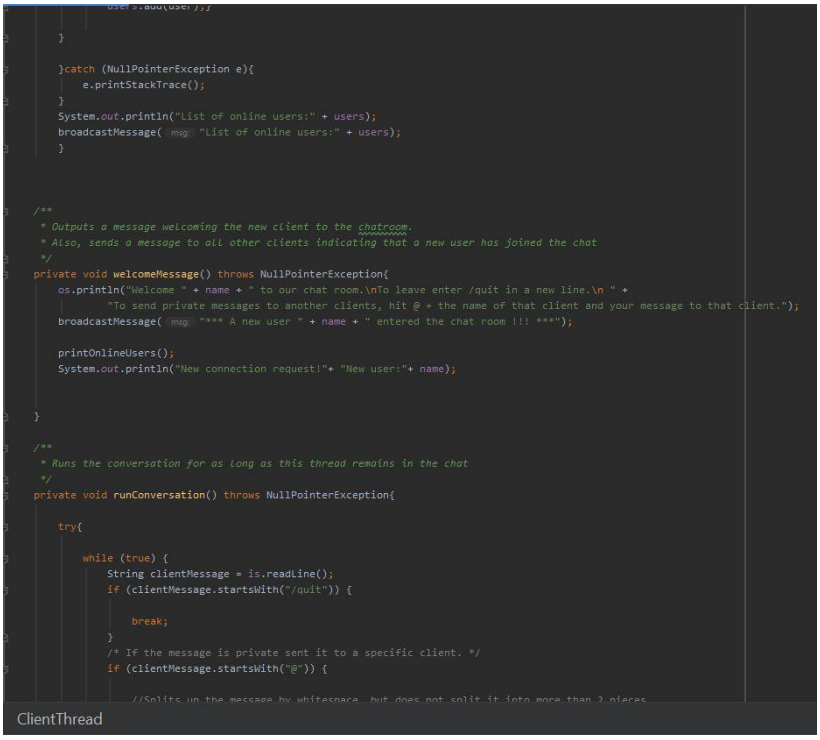
```
    /**
     * Sends a message to all clients, including this one
     * @param msg - The msg to be sent
     */
    private synchronized void broadcastMessage(String msg){
        for(int i = 0; i < threads.length; i++){
            if(threads[i] != null
                    && threads[i].getClientName() != null){

                threads[i].os.println(msg);
            }
        }
    }

    /**
     * Send a message to a specific client, as well as this client.
     * If the specified client is not found, the sending client will be notified
     * @param clientName - The name of the client to send a message to
     * @param msg - The message to send
     */
    private synchronized void sendMessageTo(String clientName, String msg){
        int i;
        for(i = 0; i < threads.length; i++){
            if  (threads[i] != null && threads[i].getClientName() != null
                    && threads[i].getClientName().equals(clientName)){

                threads[i].os.println(msg);
                this.os.println(msg);
                break;
            }
        }
        if(i == threads.length){
            os.println("There are no clients with the name <" + clientName + "> currently in this chat group.");
            os.println("Your message <" + msg + "> could not be sent");
        }
    }
```

```java
private synchronized void cleanup(){
    try{

        goodbyeMessage();
        for (int i = 0; i < threads.length; i++) {
            if (threads[i] == this) {
                threads[i] = null;
            }
        }

        /*
         * Close the output stream, close the input stream, close the socket.
         */
        is.close();
        os.close();
        clientSocket.close();

    } catch(IOException e){
        e.printStackTrace();
        System.err.println("Error closing clientThread");
    }
}

/**
 * Sends a message to all other clients indicating that this user is leaving the chatroom.
 * Also, outputs a goodbye message to this client
 */
private void goodbyeMessage(){

    broadcastMessage( msg: "*** The user " + name + " is leaving the chat room !!! ***");
    os.println("*** Bye " + name + " ***");
    users.remove(name);
    broadcastMessage( msg: "List of online users:" + users);
    System.out.println(name + " disconnected!" );
    System.out.println("List of online users:" + users);

}
```

ClientThread.java | ChatServer.java | ChatProgram.java | ChatClient.java | ChatMod

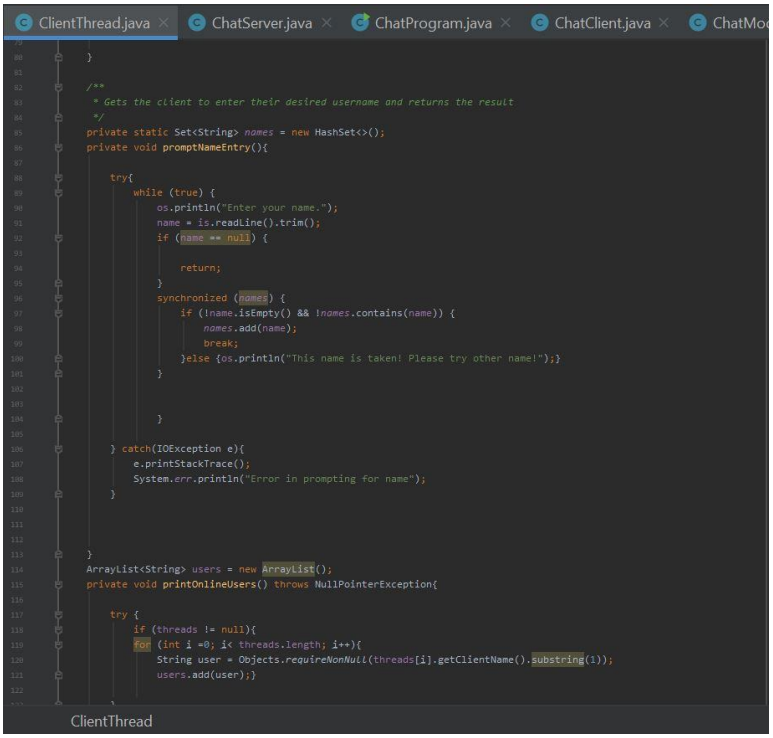
```java
     */
    private void runConversation() throws NullPointerException{

        try{

            while (true) {
                String clientMessage = is.readLine();
                if (clientMessage.startsWith("/quit")) {

                    break;
                }
                /* If the message is private sent it to a specific client. */
                if (clientMessage.startsWith("@")) {

                    //Splits up the message by whitespace, but does not split it into more than 2 pieces.
                    String[] words = clientMessage.split( regex: "\\s", limit: 2);

                    // Check to make sure the message is not empty
                    if (words.length > 1 && words[1] != null && !words[1].isEmpty()) {
                        String recipient = words[0];
                        String msg = "(private msg to <" + recipient + ">)<" + name + "> " + words[1].trim();
                        System.out.println(name+ " send a private message to " + recipient);
                        sendMessageTo(recipient, msg);
                    }
                } else {
                    /* The message is public, broadcast it to all clients. */
                    broadcastMessage( msg: "<" + name + "> " + clientMessage);
                    System.out.println(name+ " broadcast a message.");
                }
            }//End of while

        } catch(IOException e){
            e.printStackTrace();
            System.err.println("Error running clientThread conversation");
        } catch (NullPointerException e){
            e.printStackTrace();
            System.err.println("");
        }
    }

    /**
     * Clean up. Set the current thread variable to null so that a new client
     * could be accepted by the server.
     */
```

ClientThread



```
        }

        }catch (NullPointerException e){
            e.printStackTrace();
        }
        System.out.println("List of online users:" + users);
        broadcastMessage( msg: "List of online users:" + users);
        }



    /**
     * Outputs a message welcoming the new client to the chatroom.
     * Also, sends a message to all other clients indicating that a new user has joined the chat
     */
    private void welcomeMessage() throws NullPointerException{
        os.println("Welcome " + name + " to our chat room.\nTo leave enter /quit in a new line.\n " +
                "To send private messages to another clients, hit @ + the name of that client and your message to that client.");
        broadcastMessage( msg: "*** A new user " + name + " entered the chat room !!! ***");

        printOnlineUsers();
        System.out.println("New connection request!"+ "New user:"+ name);


    }

    /**
     * Runs the conversation for as long as this thread remains in the chat
     */
    private void runConversation() throws NullPointerException{

        try{

            while (true) {
                String clientMessage = is.readLine();
                if (clientMessage.startsWith("/quit")) {

                    break;
                }
                /* If the message is private sent it to a specific client. */
                if (clientMessage.startsWith("@")) {

                    //Splits up the message by whitespace, but does not split it into more than 2 pieces
```

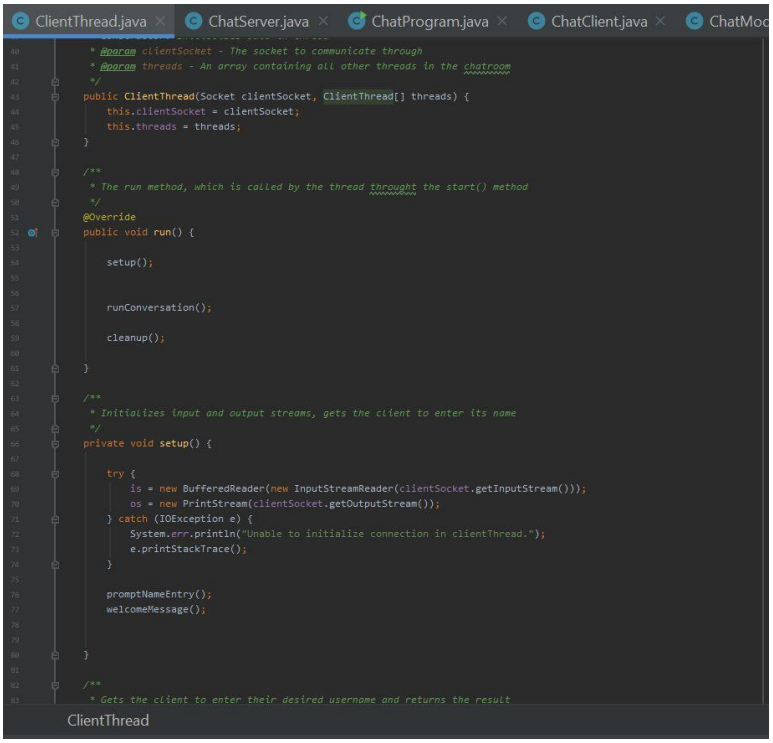
ClientThread

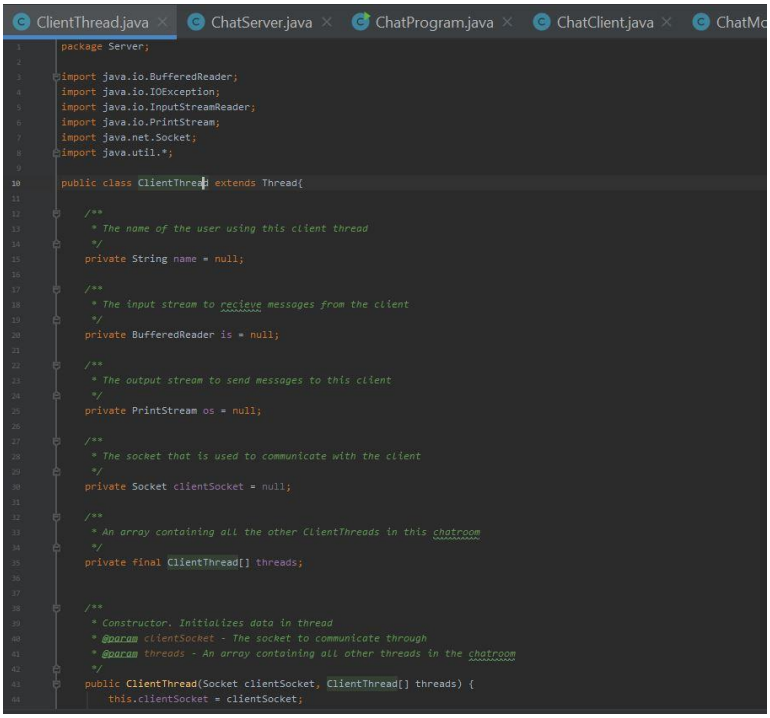


ClientThread.java | ChatServer.java | ChatProgram.java | ChatClient.java | ChatMod

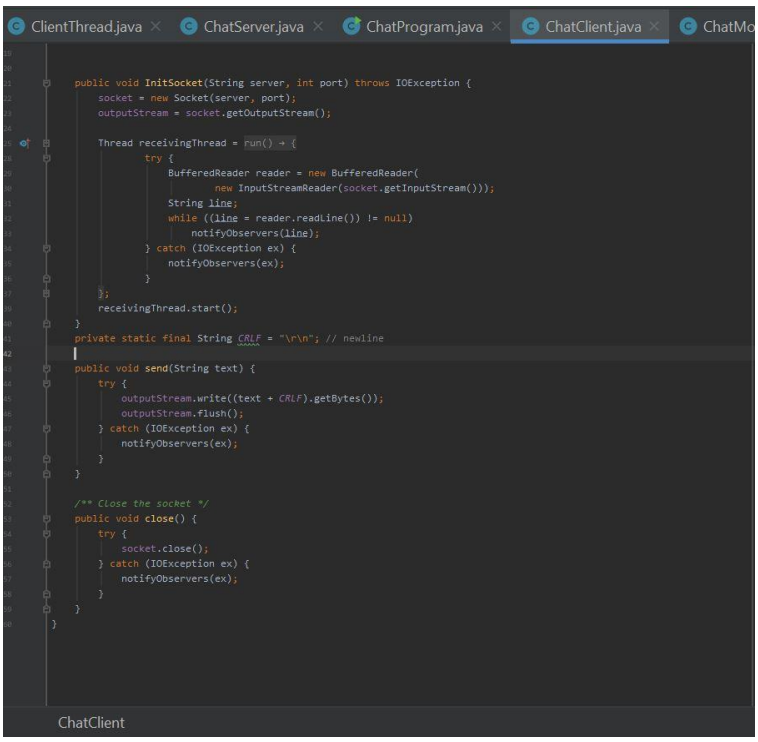
```
    }

    /**
     * Gets the client to enter their desired username and returns the result
     */
    private static Set<String> names = new HashSet<>();
    private void promptNameEntry(){

        try{
            while (true) {
                os.println("Enter your name.");
                name = is.readLine().trim();
                if (name == null) {

                    return;
                }
                synchronized (names) {
                    if (!name.isEmpty() && !names.contains(name)) {
                        names.add(name);
                        break;
                    }else {os.println("This name is taken! Please try other name!");}
                }


                }
        } catch(IOException e){
            e.printStackTrace();
            System.err.println("Error in prompting for name");
        }



    }
    ArrayList<String> users = new ArrayList();
    private void printOnlineUsers() throws NullPointerException{

        try {
            if (threads != null){
            for (int i =0; i< threads.length; i++){
                String user = Objects.requireNonNull(threads[i].getClientName().substring(1));
                users.add(user);}

            }
```

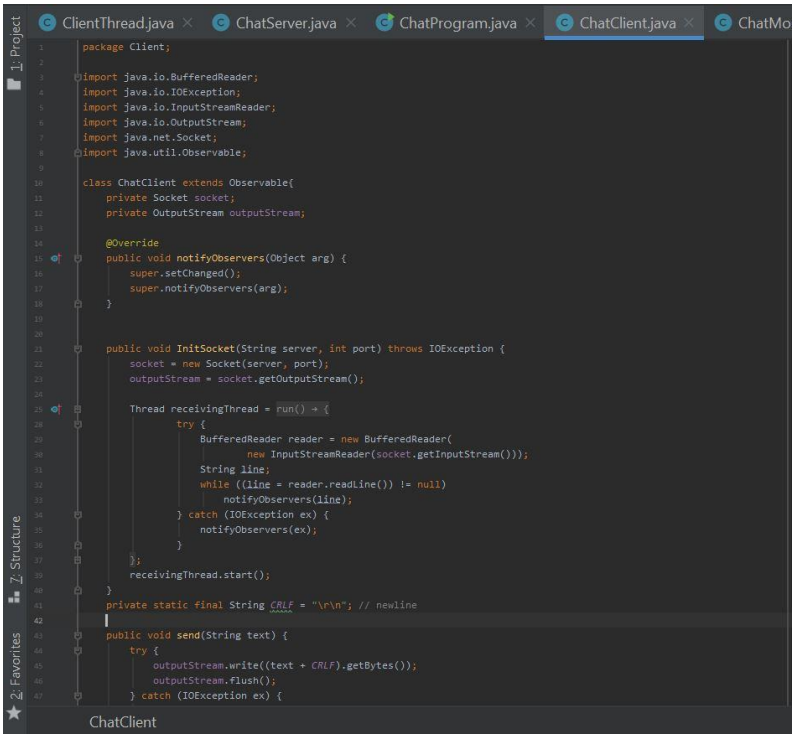
ClientThread


ClientThread.java
ChatServer.java
ChatProgram.java
ChatClient.java
ChatMoc
* @param clientSocket - The socket to communicate through
* @param threads - An array containing all other threads in the chatroom
*/
public ClientThread(Socket clientSocket, ClientThread[] threads) {
this.clientSocket = clientSocket;
this.threads = threads;
}
/**
* The run method, which is called by the thread throught the start() method
*/
@Override
public void run() {
setup();
runConversation();
cleanup();
}
/**
* Initializes input and output streams, gets the client to enter its name
*/
private void setup() {
try {
is = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
os = new PrintStream(clientSocket.getOutputStream());
} catch (IOException e) {
System.err.println("Unable to initialize connection in clientThread.");
e.printStackTrace();
}
promptNameEntry();
welcomeMessage();
}
/**
* Gets the client to enter their desired username and returns the result
ClientThread

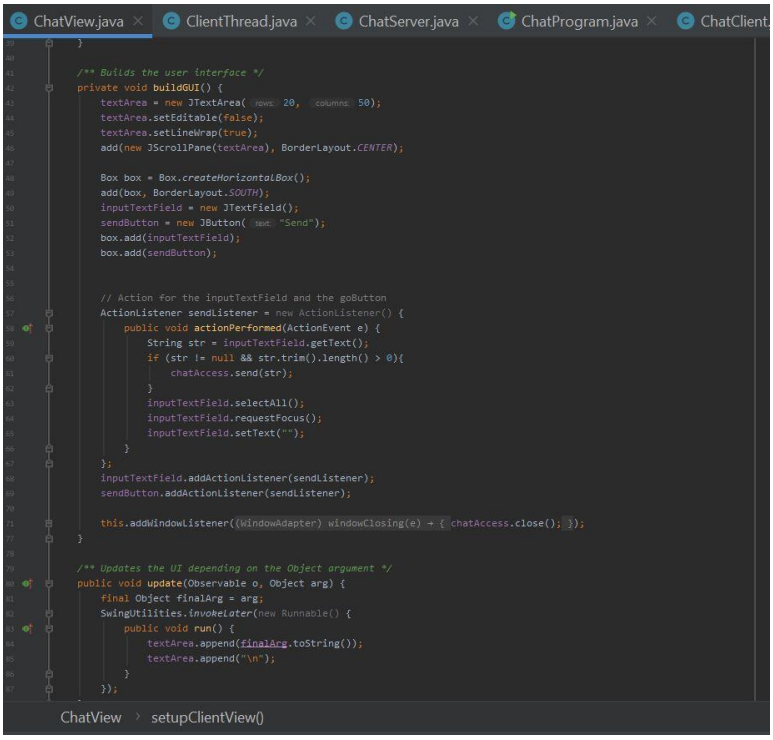

ClientThread.java
ChatServer.java
ChatProgram.java
ChatClient.java
ChatMc
package Server;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintStream;
import java.net.Socket;
import java.util.*;
public class ClientThread extends Thread{
/**
* The name of the user using this client thread
*/
private String name = null;
/**
* The input stream to recieve messages from the client
*/
private BufferedReader is = null;
/**
* The output stream to send messages to this client
*/
private PrintStream os = null;
/**
* The socket that is used to communicate with the client
*/
private Socket clientSocket = null;
/**
* An array containing all the other ClientThreads in this chatroom
*/
private final ClientThread[] threads;
/**
* Constructor. Initializes data in thread
* @param clientSocket - The socket to communicate through
* @param threads - An array containing all other threads in the chatroom
*/
public ClientThread(Socket clientSocket, ClientThread[] threads) {
this.clientSocket = clientSocket;

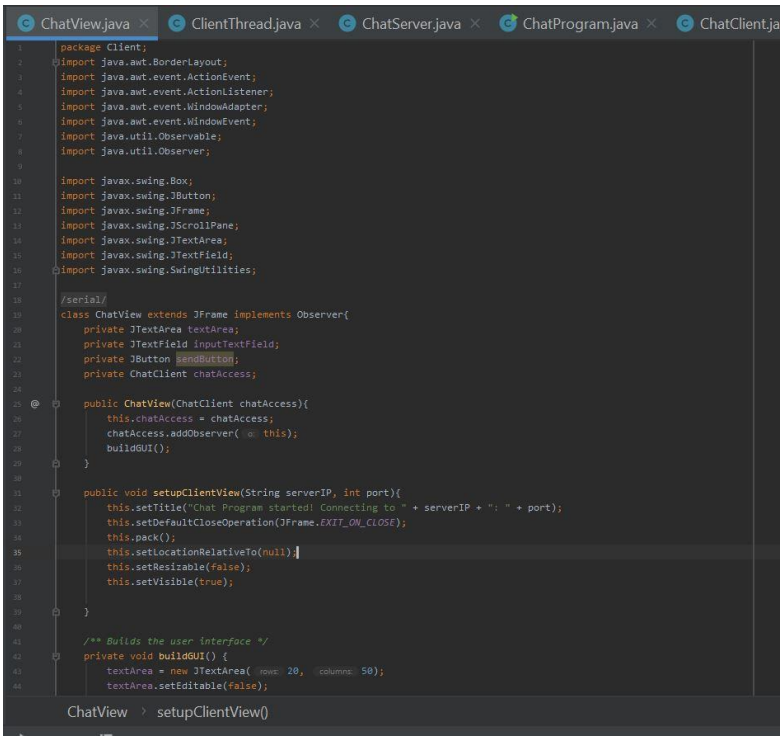
- ChatClient:


ClientThread.java
ChatServer.java
ChatProgram.java
ChatClient.java
ChatMo
public void InitSocket(String server, int port) throws IOException {
socket = new Socket(server, port);
outputStream = socket.getOutputStream();
Thread receivingThread = run() -> {
try {
BufferedReader reader = new BufferedReader(
new InputStreamReader(socket.getInputStream()));
String line;
while ((line = reader.readLine()) != null)
notifyObservers(line);
} catch (IOException ex) {
notifyObservers(ex);
}
};
receivingThread.start();
}
private static final String CRLF = "\r\n"; // newline
public void send(String text) {
try {
outputStream.write((text + CRLF).getBytes());
outputStream.flush();
} catch (IOException ex) {
notifyObservers(ex);
}
}
/** Close the socket */
public void close() {
try {
socket.close();
} catch (IOException ex) {
notifyObservers(ex);
}
}
}
ChatClient

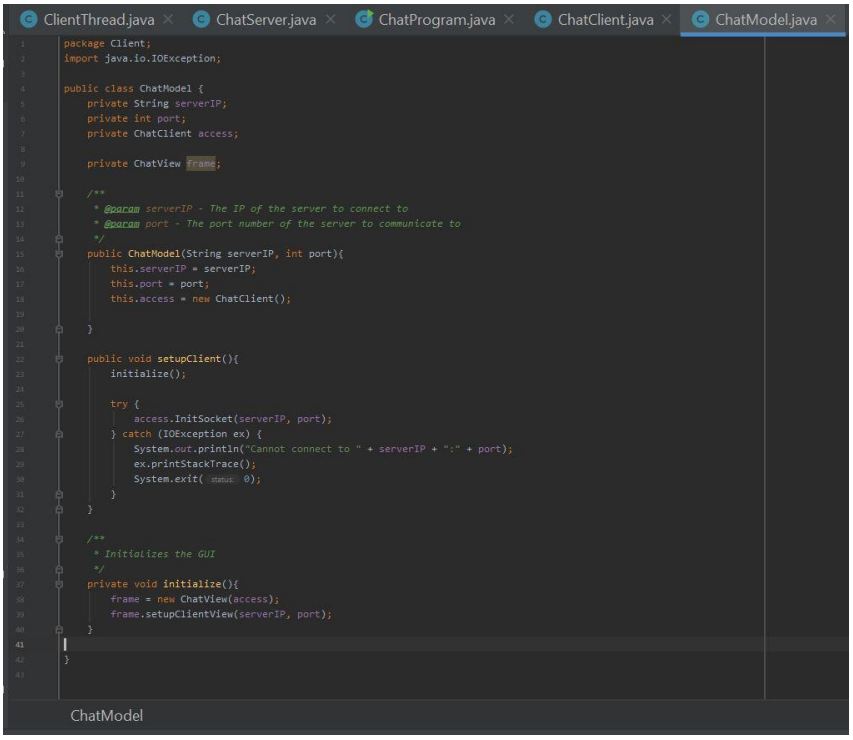

1: Project
ClientThread.java
ChatServer.java
ChatProgram.java
ChatClient.java
ChatMo
package Client;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.Socket;
import java.util.Observable;
class ChatClient extends Observable{
private Socket socket;
private OutputStream outputStream;
@Override
public void notifyObservers(Object arg) {
super.setChanged();
super.notifyObservers(arg);
}
public void InitSocket(String server, int port) throws IOException {
socket = new Socket(server, port);
outputStream = socket.getOutputStream();
Thread receivingThread = run() -> {
try {
BufferedReader reader = new BufferedReader(
new InputStreamReader(socket.getInputStream()));
String line;
while ((line = reader.readLine()) != null)
notifyObservers(line);
} catch (IOException ex) {
notifyObservers(ex);
}
};
receivingThread.start();
}
private static final String CRLF = "\r\n"; // newline
public void send(String text) {
try {
outputStream.write((text + CRLF).getBytes());
outputStream.flush();
} catch (IOException ex) {
Z: Structure
2: Favorites
ChatClient

- ChatView:


ChatView.java
ClientThread.java
ChatServer.java
ChatProgram.java
ChatClient
}
/** Builds the user interface */
private void buildGUI() {
textArea = new JTextArea(rows: 20, columns: 50);
textArea.setEditable(false);
textArea.setLineWrap(true);
add(new JScrollPane(textArea), BorderLayout.CENTER);
Box box = Box.createHorizontalBox();
add(box, BorderLayout.SOUTH);
inputTextField = new JTextField();
sendButton = new JButton(text: "Send");
box.add(inputTextField);
box.add(sendButton);
// Action for the inputTextField and the goButton
ActionListener sendListener = new ActionListener() {
public void actionPerformed(ActionEvent e) {
String str = inputTextField.getText();
if (str != null && str.trim().length() > 0){
chatAccess.send(str);
}
inputTextField.selectAll();
inputTextField.requestFocus();
inputTextField.setText("");
}
};
inputTextField.addActionListener(sendListener);
sendButton.addActionListener(sendListener);
this.addWindowListener((WindowAdapter) windowClosing(e) → { chatAccess.close(); });
}
/** Updates the UI depending on the Object argument */
public void update(Observable o, Object arg) {
final Object finalArg = arg;
SwingUtilities.invokeLater(new Runnable() {
public void run() {
textArea.append(finalArg.toString());
textArea.append("\n");
}
});
ChatView › setupClientView()

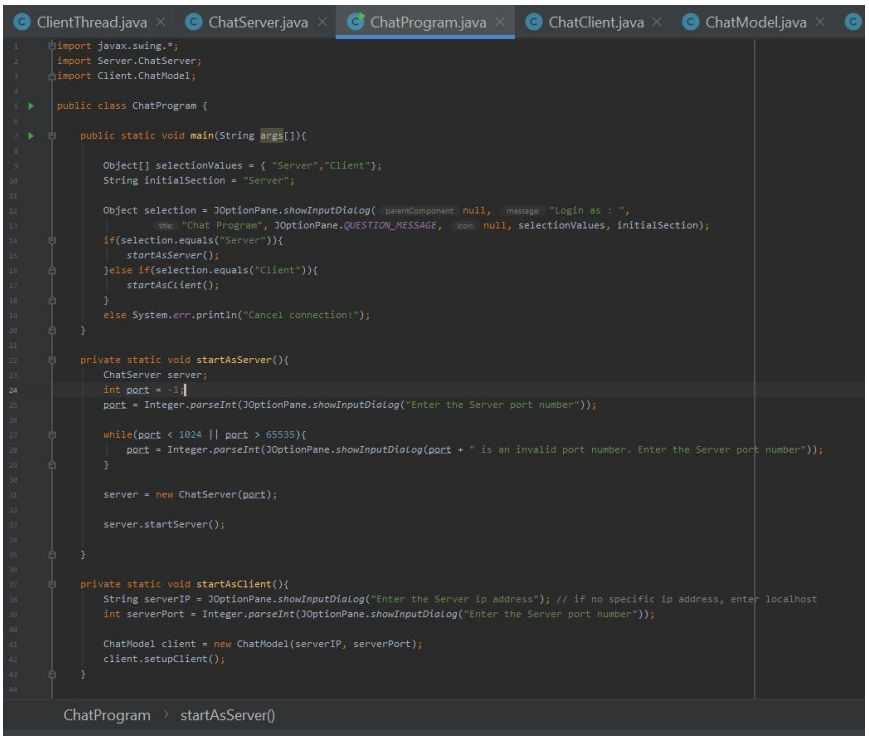

ChatView.java
ClientThread.java
ChatServer.java
ChatProgram.java
ChatClient.ja
package Client;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.util.Observable;
import java.util.Observer;
import javax.swing.Box;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.JTextField;
import javax.swing.SwingUtilities;
/serial/
class ChatView extends JFrame implements Observer{
private JTextArea textArea;
private JTextField inputTextField;
private JButton sendButton;
private ChatClient chatAccess;
public ChatView(ChatClient chatAccess){
this.chatAccess = chatAccess;
chatAccess.addObserver(o: this);
buildGUI();
}
public void setupClientView(String serverIP, int port){
this.setTitle("Chat Program started! Connecting to " + serverIP + ": " + port);
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
this.pack();
this.setLocationRelativeTo(null);
this.setResizable(false);
this.setVisible(true);
}
/** Builds the user interface */
private void buildGUI() {
textArea = new JTextArea(rows: 20, columns: 50);
textArea.setEditable(false);
ChatView › setupClientView()

- ChatModel:




```java
package Client;
import java.io.IOException;

public class ChatModel {
    private String serverIP;
    private int port;
    private ChatClient access;

    private ChatView frame;

    /**
     * @param serverIP - The IP of the server to connect to
     * @param port - The port number of the server to communicate to
     */
    public ChatModel(String serverIP, int port){
        this.serverIP = serverIP;
        this.port = port;
        this.access = new ChatClient();

    }

    public void setupClient(){
        initialize();

        try {
            access.InitSocket(serverIP, port);
        } catch (IOException ex) {
            System.out.println("Cannot connect to " + serverIP + ":" + port);
            ex.printStackTrace();
            System.exit( status: 0);
        }
    }

    /**
     * Initializes the GUI
     */
    private void initialize(){
        frame = new ChatView(access);
        frame.setupClientView(serverIP, port);
    }
}
```

- ChatProgram:




```java
import javax.swing.*;
import Server.ChatServer;
import Client.ChatModel;

public class ChatProgram {

    public static void main(String args[]){

        Object[] selectionValues = { "Server","Client"};
        String initialSection = "Server";

        Object selection = JOptionPane.showInputDialog( parentComponent: null, message: "Login as : ",
                title: "Chat Program", JOptionPane.QUESTION_MESSAGE, icon: null, selectionValues, initialSection);
        if(selection.equals("Server")){
            startAsServer();
        }else if(selection.equals("Client")){
            startAsClient();
        }
        else System.err.println("Cancel connection!");
    }

    private static void startAsServer(){
        ChatServer server;
        int port = -1;
        port = Integer.parseInt(JOptionPane.showInputDialog("Enter the Server port number"));

        while(port < 1024 || port > 65535){
            port = Integer.parseInt(JOptionPane.showInputDialog(port + " is an invalid port number. Enter the Server port number"));
        }

        server = new ChatServer(port);

        server.startServer();

    }

    private static void startAsClient(){
        String serverIP = JOptionPane.showInputDialog("Enter the Server ip address"); // if no specific ip address, enter localhost
        int serverPort = Integer.parseInt(JOptionPane.showInputDialog("Enter the Server port number"));

        ChatModel client = new ChatModel(serverIP, serverPort);
        client.setupClient();
    }
```