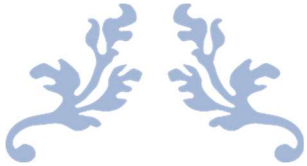

A CLIENT-SERVER CHAT PROGRAM

Project 1

AUGUST 5, 2022

Markus Sass (22548890@sun.ac.za)
Chris Langeveldt (23632135@sun.ac.za)

Table of Contents

Overview 2
Required Features Implemented 2
Functionality: 2
Graphical User Interface: 2
Unimplemented Features 2
Additional Features Implemented 3
Description of files 3
Server.java 3
ClientHandler.java 3
ClientGUI.java 3
ClientListenerThread.java 3
Message.java 3
Timeout.java 3
Program Description 4
Server 4
Client 4
Experiments 4
Conclusion from experiments 6
Issues Encountered 6
Significant Data Structures 6
Compilation 6
Server 6
Client 6
Execution 7
Server 7
Client 7
Libraries and Packages 7

Overview

The aim of this project was to develop a client-server chat program. This program functions as a “group chat” that allows to multiple clients to communicate using a simple graphical user interface. We chose Java as our stack and made use of Java socket programming as well as multithreading.

In this document we discuss all required features implemented, unimplemented features, additional features implemented, description of files, program description, experiments, issues encountered, significant data structures, compilation, execution and libraries and packages.

Required Features Implemented

Functionality:

1. Sending and receiving messages
2. Sending and receiving works concurrently
3. Whispering to a single person at a time, and receiving whispers (“@<name> <message>)
4. List of online users (by typing “\list”)
5. List is updated automatically after a user connection/disconnection
6. Client is stable after Server termination
7. Server is stable after Client termination
8. Cannot whisper to non-existing Client
9. Cannot connect to non-existing server
10. Duplicate Usernames not possible
11. All opened sockets and streams closed on Client termination
12. Concurrent Server service model
13. Server notification system

Graphical User Interface:

14. Output area
15. Input area
16. List of current users

Unimplemented Features

- It is not possible to whisper to multiple users at the same time. (It seems that this is the only thing we lost marks for.)

Additional Features Implemented

- Error checking for port number between 1 and 65535 (Server side)
- Name of current user displayed on top of the graphical user interface
- Error checking for invalid username (only alpha-numerical characters allowed, name cannot be blank, name cannot be “SERVER”)
- Error checking for invalid port number and IP address
- Included a timeout functionality when socket takes more than 5 seconds to connect
- User can disconnect by typing “\exit”

Description of files

Server.java

This file is run to start the server with specified port number. This file creates the server socket and then creates a new thread that handles that client.

ClientHandler.java

This class is for each thread that handles each client from the server side. This file mainly receives messages form clients and then send it the correct clients. This file checks uniqueness of usernames and contains the list of connected users. The file also adds and removes clients from the list as the connect/disconnect.

ClientGUI.java

This file is run to start the client side and the graphical user interface. This file checks validity of the IP address and port number entered by the client. The socket is created here. The thread that waits to receive messages is created and called here. This file is also responsible for sending messages to the server.

ClientListenerThread.java

This class is the thread of each client that waits for messages.

Message.java

This file contains the class for the main object that is being sent from client to server and vice versa.

Timeout.java

This class is for the thread created by ClientGUI.java that checks if a socket takes too long to connect.

Program Description

Server

Server.java is run with “java Server <port number>”. The server socket is created which is then used to initialize the Server object. The server then starts. The server continuously waits for a client to connect. When a client connects a new thread is created which then handles all interactions with that client. This thread continuously waits for messages from its client. Once a message is received, the message is processed and sent to the necessary clients.

Client

CleintGUI.java is run with “java ClientGUI”. The user is prompt for the IP address, port number and username. The socket is then created and connects to the server. A thread is created that continuously waits for messages to be received and then handles these and prints them to the graphical user interface. Messages are also sent when the client enters them into the input box.

Experiments

These experiments relate to the features mentioned in the Required Features Implemented section.

1. We connected two clients to the server and sent a message from the one. The other client did indeed receive the message. We concluded that sending and receiving messages works.
2. We started typing a message on one client. While typing we sent a message from another client. The client busy typing did indeed receive the message. We concluded that sending and receiving messages concurrently works.
3. We connected three clients to the server and sent a whisper from client 1 to client 2. Client 3 did not see this message. Only the sender and the receiver saw this message. We concluded whispering to a single person at a time works.
4. We connected three clients to the server. By typing “\list”, the program returned the correct list of currently connected users. We concluded that the program does indeed return a list of users.
5. We connected one client to the server. The list displayed the only this user. We then connected another client to the server. Both clients’ lists now displayed both these clients’ usernames. We connected a 3rd client and all three displayed the correct updated list. We then disconnected one client. The remaining clients’ lists now displayed the updated list of only these 2. We then connected another client, and all lists displayed the updated correct list. We concluded that the list does indeed update automatically after a user connection/disconnection.

6. We connected two clients to the server and sent a couple of messages. We then closed the server (forcefully). Both clients received the message "SERVER: Shut down" and then the program for both clients exited. We concluded that a client is stable after server termination.
7. We connected three clients to the server and sent some messages from all of them. We then closed one client's program (forcefully). The server experienced no errors, and the other two clients could continue sending messages as normal. We concluded that the server is stable after client termination.
8. We connected two clients to the server. From the client 1 we tried to send a whisper to a client that does not exist. Only client 1 received the message "SERVER: User, <name entered>, does not exit!".
9. We ran a client program and then entered a random word for the IP address and then a random port number. The program outputted the message "Incorrect IP address or incorrect port number" and the prompts started again. We then entered random numbers. In the terminal we received the message "Waiting to connect…". A few seconds later the message "Failed to connect" appeared and the program exited. We concluded that you cannot connect to a non-existing server and that our error checking was done properly.
10. We connected two clients to the server. We then connected a 3rd trying to use client 1's name. The program then prompted client 3 to choose another name as this one already exists. We then tried to enter client 2's name and we got the same error. We then entered a unique name and we connected to the server. We concluded that duplicate usernames are not possible.
11. We put print statements at the end of our "closeEverything" function in the ClientHandler.java file. We then ran the server and connected a client to the server. We then terminated the client. On the server side, the message was printed out showing that this closing function had been executed. We concluded that the opened socket and streams closed on client termination.
12. We connected two clients to the server. We typed a message from both clients and sent them at the same time. Both messages displayed in the output box as it should. We concluded that the server does indeed work concurrently as each client has its own client handler thread on the server side.
13. We connected client 1 to the server and client saw the message from the server that this client has connected. We then connected another client and now both these clients received the connection message from the server. We connected a 3rd client, and everyone received the same message. We then disconnected a client and the other two received the disconnection message from the server. We disconnected another and again, the remaining client received the disconnection message. We concluded that the server notification system works.
14. We ran all these previous tests with the graphical user interface and the output area worked as intended with no issues. We conclude that the output area works.

15. We ran all these previous tests with the graphical user interface and the input area worked as intended with no issues. We concluded that the input area works.
16. The list of current users also worked as expected from the experiments done in points 4 and 5. We concluded that the graphical list works.

Conclusion from experiments

As mentioned in every point above, we concluded that all required functionality works as we thought intended. However, there is one part of functionality that we did not realise should be implemented: The ability to whisper to multiple users at the same time. Trying to do this by adding more "@"s results in only the first mentioned user to receive the message that then still includes the other "@"s in the message itself. Other than that, all these experiments above (testing every single required functionality) shows that our program works as intended.

Issues Encountered

One of the main issues we encountered was integrated the functional code with the graphical user interface. After an initial struggle, we got everything working. Another issue we only encountered in our project demo. Our program is not able to whisper to multiple clients at the same time.

Significant Data Structures

The only notable data structure we have used is an Array list of the client handlers on the server side. We used this to keep track of current clients connected to the server.

Compilation

Server

javac Server.java

Client

javac ClientGUI.java

Execution

Server
java Server <port number>

Client
java ClientGUI

Libraries and Packages

- java.io
- java.net
- java.util
- java.awt
- javax.swing