Network Programming Project Autumn 2021

Student #: G00398242

Description

This is a network-based chat application built with the Java Socket API.

There are four classes in this application:

1. Server
2. ClientThread
3. Client
4. Addressing

The Server application creates a serverSocket to listen for incoming TCP connections and then creates a Socket object to communicate with a client over that serverSocket connection using the Sockets getInputStream() and getOutputStream methods. If more than one client attempts to connect to the Server while it is waiting for input from another client, the new client would have to wait.

Therefore, the Server class creates a Socket in a thread for each new client connection using the Runnable class ClientThread. This allows the server to create multiple threads to receive/send messages from multiple clients simultaneously while still listening for incoming connection attempts over the ServerSocket with accept().

The Client application creates a Socket with an IP address and port number for the listening serverSocket that was created by the Server class's connect() method. The serverSocket listens for these incoming connections and accepts the connection so that the server and client can now begin communication via each other's Sockets. The Client application can send messages using the sendMessage method and listen for incoming communication from the Server's outputStream using the listen() method.

The Client application also creates a thread for the listen() method so that it can accept incoming messages at the same time while waiting for console input from the user.

The Addressing class contains some shared methods between the client and server application for assign IP address/port number by the user.

How to Run Program

Client Application:

Once you start the client application you will be:

1. given IP address options (use default 'localhost' or manually enter an address)
2. asked to input a port number (must be within a valid range)
3. asked to input a chat username.

Then the client will try to connect to that IP address/port number and return a message if it was successful or not.

Server Application:

Once you start the server application you will be:

1. asked to input a port number (must be within a valid range)

Then the server will return a console message and start listening on that port number.

Features

Client unable to connect or reconnecting after a lost connection

- Client application will try to reconnect with the server at set intervals for a set number of attempts.
- If the client fails to reconnect, then it will then prompt the user to close the application or enter another port number.

Client-side commands

There is three chat room command that the user can use.

1.	\q	Exits the chat room and closes the chat client.
2.	#userlist	Returns a list of all active user in the chat room
3.	#*	Sends a private message to the username by replacing the * (e.g. #ann sends DM to ann)

Chat room features

1. The name of the sender prefixes the message.
2. If a user enters a chat name that another user is currently using then the chat client will automatically assign a numerical suffix to their input.
3. When a user enters or leaves the chat room, the other user will receive a chat message to let them know.

Server features

1. The server logs the clients port number to the console when they enter the chat room.
2. When the client connection ends between the server socket, the server confirms that the client's IO resources have been closed in the console log.

References

Elliotte Rusty Harold (2014). Java network programming. Beijing: O’reilly.

Video Tutorial Resources:

https://fullstackmastery.com/page/5/how-build-multiuser-chat-application-in-java

https://www.youtube.com/watch?v=gLfuZrrfKes&t=253s

Java Documentation:

https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html

https://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html