Project 1

A CLIENT-SERVER CHAT PROGRAM

GROUP 17

01/03/23

Contents

[Report: Client-Server chat program 2](#_Toc1937397905)

[Report: Client-Server chat program 2](#_Toc1001215130)

[Introduction 3](#_Toc754856820)

[Features 3](#_Toc477048727)

[Description of files 3](#_Toc1051880346)

[Program description 3](#_Toc1416077686)

[Experiments 5](#_Toc1529694759)

[Experiment 1: 21 clients connect to the server at the same time 5](#_Toc1911857090)

[Experiment 2: multiple clients connect and disconnect simultaneously 5](#_Toc928636732)

[Experiment 2: 5](#_Toc2099135230)

[Issues Encountered 6](#_Toc74236773)

[Significant data structures 6](#_Toc1701123180)

[Array Lists: 6](#_Toc830697040)

[Design 6](#_Toc1815777838)

[Compilation 6](#_Toc2058691117)

[Roles 6](#_Toc357663400)

[Group Leader: 6](#_Toc403206571)

[Client: 6](#_Toc768779741)

[Server: 6](#_Toc1916544812)

[GUI: 6](#_Toc1575096234)

[Report: 6](#_Toc1967300298)

# Report: Client-Server chat program

# Introduction

The scope of this project entailed the creation of a chat application utilizing the single server-multiple client model. The purpose of this report is to present an analysis of the project including all features implemented, comprehensive explanations of source code files, program flow, as well as a detailed discussion of experiments conducted with the application along with the conclusions drawn from them. Additionally, this report will examine the data structures used within the implementation of this application as well as issues encountered along the way.

# Features

All features were implemented as well as an additional implementation of a graphical user interface for the server was executed.

# Description of files

Server.java: listens up for incoming connections coming up on port 8005 and makes a new thread for each connection and calls the ClientHandler which handles the connection.

ClientHandler.java: handles connection with each client and each instance of the clientHandler runs on its own separate thread. It listens to messages from the client it is associated with and broadcast a message or whisper the message.

Client.java: it initiates communication with the server. Listens to incoming messages from the server and send outgoing messages to the server as well

# Program description

Client class: it has three instance variables which are the Socket socket, BufferedReader bufferedReader and BufferedWritter bufferedWriter which are all initialised inside the Client constructor.

* ***StartUpWindow()***: Creates the startup Gui for the Client application’s startup window. The Gui requires the client to type in the Server IP, Port, and the client username for the client to connect these three things needs to be valid. There is a connect button that a new client clicks on to connect. When a client clicks on this button the input they typed like the Server IP, Port and the username will be checked if it is valid and if it is not then some errors will be printed notifying the client that they are not valid and if they are, the client will be successfully connected There is startPanel object of type Jpanel which makes use the GridBagConstraints for it to organise the componets. The constraints which is a type of GridBagConstraints object specifies the location of the components in the Gui. Once the client is connected it can start listening to messages by calling the method listenForMessages().
* ***chatWindow():*** Creates the Gui for the Client Application’s chat box. The chat box has a side panel which displays a list of connected clients, a chat panel where the client can send and receive messages and a bottom panel that has the buttons for sending messages and commands. A Jframe called chatframe is created and its title is set to the application name and the name of the client’s username. There is the button sendMessageBtn that is clicked to send the message a user writes but the message must be longer than zero or else nothing will be done, if the message is a #clear then the screen will be cleaned or else it will just write the message the client sends.
* ***writer (String message):*** it writes the message the client wants to send to a bufferedWritet associated with the socket provided the socket is still coneected to the server and if any IOException occurs then the the killAll(socket, bufferedReader, bufferdWritter) will be called and it will close the socket, bufferedReader, and bufferedWritter.
* ***listenFormessage():*** this method runs on separate thread and it will be listening to incoming messages by reading them using a bufferedReader. If the message read contains “#USERNAMEVALID” then it will know that the username is unique and the chat window will be displayed. If the message read contains “#USERNAMEINVALID” it will know that the username is not unique. If the message read contains “#SERVER” and “joined” this indicates that a new client has joined the chat. If the message contains “#SERVER” and “left the chat” it means a client has left the chat and their username will be removed from the list of connected clients. if the message read contains “YOU LEFT” it means that you as the client has left the chat and your chat window will be disabled. Otherwise, it will just display the message read on the chat window.
* ***killAll(Socket socket, BufferedReader bufferedReader, Buffered bufferedWritter):*** It checks if the socket, bufferedReader and bufferedWritter if they are null, if indeed they are null, it will close all of them and if any of them is null then it will close the ones which a not null and leave null one as it is. If an IOExpetion occurs, we print the stack.
* ***main():*** mkes a thread, overide the run method in the thread so that it can show the start up window by calling the startUpWindow() method and then starts the thread.

Server class: It has one instance variable called severSocket.

* ***Server (ServerSocket severSocket):*** it is the Server Class’s constructor that initialises the instance variable severSocket to the parameter variable that is severScocket.
* ***ServerSocket():***  it listens to incoming connections and makes a new thread for each connection. And if an IOException occurs then it will just close the socket by calling the CloseSSocket() method.
* ***CloseSSocket():*** it closes the socket if it is not null and if an IOException occurs it will just print the stack.
* ***Gui():*** makes the Gui for the Server using swing. It makes a Jframe called serverframe and set its title to "SERVER:8005" and displays the text "SERVER running!
* ***Main ():*** creates a new ServerSocket objects called serverSocket and pass it as a parameter variable to the Server object created. It makes the GUI to run on a separate thread and start the thread. Then it starts the Server by calling the method startServer(). If an IOException occurs, it prints out “PORT already in use!"

ClientHandler class: It has five instant variables the Socket socket, BufferedReader bufferedReader, Buffered bufferedWritter and a JTextArea serverTA.

* **ClientHandler(Socket socket, JTextArea textAreaServer):** it initialise the instance variables. And if any IOException occurs it will close the socket, bufferedReader and bufferedWriter. It also check is the username the client typed is unique or not print “”
* **Writter():** it writes the message using a bufferedWritter only when the socket is connected and if an IOException occurs it will print the stack.
* **isUserNameUnique(String name):** loops through the list of clientHandler and checks whether the username the client wants to join is unique or not, if it is it will return true or else it will return false.
* **run():** runs as long as the socket is connected. It reads a message from the command line and checks if it has special commands which is “#@” for whispering to one client or more multiple clients, “#EXIT” for when a client wants to exit the chat. Else it will just broadcast message to the clients that are connected
* **broadcastMessage(String messageToSend):** it broadcasts the message the client wants to send to all the clients in the connections except for the client that sent the message.
* **secretMessage(String messageToSend, String whoToSend):** it broadcasts the message the client wants to send to the specified client in the connections except for the client that sent the message.
* **kickClient():** it kicks off a client if they disconnected unexpectedly. It then removes the client and notifies the other clients that the client has left the chat.
* **getUserList():** it returns the list of all the connected clients
* **UpdateOnlineStatus():** it makes sure that all the clients in the connection have the updated list of connected clients
* **killAll(Socket socket, BufferedReader bufferedReader, BufferedWriter bufferedWriter):** It checks if the socket, bufferedReader and bufferedWritter if they are null, if indeed they are null, it will close all of them and if any of them is null then it will close the ones which a not null and leave null one as it is. If an IOExpetion occurs, we print the stack.

# Experiments

## Experiment 1: 21 clients connect to the server at the same time

Hypothesis: Expect that server will be able to support multiple clients.

Testing: connected 21 clients to the server to test the functionality of the multithreading implementation.

Conclusion: The program was able to run successfully while having 21 clients connected to a single server. Thus, proving that the functionality is working.

## Experiment 2: multiple clients connect and disconnect simultaneously

Hypothesis: Server should stay functional while clients connect and disconnect at different times.

Testing: Tested by connecting 10 clients to the server, and then disconnecting five, then reconnecting three.

Conclusion: Server can stay operational while clients are connecting and disconnecting.

# Issues Encountered

No issues encountered.

# Significant data structures

## Array Lists:

Array lists are dynamic data structures that allow us to store a collection of elements that can be accessed using an index-based system. Unlike arrays, array lists do not have a fixed size, which made the storing of clients more flexible.

When a new client connects, they can be added to the ArrayList and removed if they disconnect from the server.

# Design

The program made use of Object-Oriented Programming. Multithreading was used to allow clients to see messages sent by other clients as well as send their own messages and to allow the server to remain operational as connections and disconnection requests occurred. Socket programming was implemented to allow communication between the client and server. Exception handling was implemented to ensure the server remained operational even if a client’s connection failed. Used Java Swing to implement the GUI for both the client and the server.

# Compilation

It is advisable to ensure that the Server is operational and running before initiating any client programs.

To compile and run the Server:

* “Javac Server.java”
* “Java Server”

To compile and run the Client:

* “Javac Client.java”
* “Java Client”

# Roles

## Group Leader:

Clinton Elves - [24007676@sun.ac.za](mailto:24007676@sun.ac.za)

## Client:

Jayden Abrahams – [23559322@sun.ac.za](mailto:23559322@sun.ac.za)

Langton Chikukwa – [23607769@sun.ac.za](mailto:23607769@sun.ac.za)

## Server:

Clinton Elves - [24007676@sun.ac.za](mailto:24007676@sun.ac.za)

Jayden Abrahams – [23559322@sun.ac.za](mailto:23559322@sun.ac.za)

## GUI:

Langton Chikukwa – [23607769@sun.ac.za](mailto:23607769@sun.ac.za)

Cindi-Ann Jeffery – [24695823@sun.ac.za](mailto:24695823@sun.ac.za)

Jayden Abrahams – [23559322@sun.ac.za](mailto:23559322@sun.ac.za)

## Report:

Jayden Abrahams – [23559322@sun.ac.za](mailto:23559322@sun.ac.za)

Cindi-Ann Jeffery – [24695823@sun.ac.za](mailto:24695823@sun.ac.za)

Konanani Mathoho – [24774898@sun.ac.za](mailto:24774898@sun.ac.za)