CLI-based networked distributed system for a group-based client-server communication

Radoslav Gadzhovski, Petya Petrov, Ivaylo Yordanov, Antony Clark

COMP1549: Advanced Programming
University of Greenwich
Old Royal Naval College
United Kingdom

***Abstract*—This report introduces a CLI-based networked distributed system for group-based client-server communication. The system is designed to provide an intuitive platform for users to share information and collaborate efficiently across different locations. Key features include a client-server model, a CLI-based interface, and support for group-based communication. The system is ideal for businesses with remote workers and other collaborative work settings.**

Keywords: *client, server, user, sockets, threads, methods, package, class*

I. Introduction

This project aims to implement a chatroom program which allows multiple users to communicate with each other in real-time. The program is built utilising a collection of imported classes and interfaces.

The system uses a client-server model to establish communication between users, with the server acting as a central point for managing and distributing messages. The CLI-based interface provides users with an intuitive and user-friendly platform for sending and receiving messages.

Some of the key features of this system are its ability to support both group-based and private communication achieved via concurrency, making it ideal for collaborative work settings. Additionally, the system's networked architecture allows it to be used across different locations, making it an ideal solution for users in separate spaces. Aimed features of the project include modularity, JUnit based testing, fault tolerance and component-based development.

II. Design/Implementation

A fundamental goal for the project was to keep the design as user-friendly as possible by providing easy-to-understand instructions to operate the program.

During the production, our team used a prevalent Java IDE's named "Intellij IDEA". The primary reason for choosing Intellij is due to the authors of the software, "JetBrains" who are also the creators of "PyCharm IDE", an IDE the team is familiar with. As both IDE's were developed by the same company, they are built with a similar structure, supporting a variety of useful plugins and frameworks which helps with the workflow.

Components

Our program consists of 3 packages: ***client***, ***server*** and ***shared***.

The ***client*** package has 3 different files: *ChatClient.java*, *NetworkClient.java* and *UserInput.java*.

ChatClient.java is the entry point for the client program where the main method is.

NetworkClient.java handles the network communication with the server.

UserInput.java handles user input.

ChatClient.java: The ChatClient class is responsible for starting the program and passing user input to the NetworkClient class. The ChatClient class takes user input arguments for the server IP address, port number, and username. The IP address and port number are validated to ensure

they are in the correct format and range. The username is validated to ensure it is not empty, not longer than 20 characters, and does not contain the word 'admin'. The ChatClient class then creates a new instance of the NetworkClient class and passes the validated arguments to it.

NetworkClient.java: The NetworkClient class is responsible for handling network communication with the server. It takes the validated IP address, port number, and username from the ChatClient class and creates a new socket connection to the server. It then creates ObjectOutputStream and ObjectInputStream instances for sending and receiving messages from the server. The username is sent to the server to identify the user. The NetworkClient class also includes a ListenFromServer class that runs on a separate thread and continuously listens for incoming messages from the server. When a message is received, it is printed to the console.

UserInput.java: The UserInput class is responsible for handling user input. It takes an instance of the NetworkClient class as input and prompts the user to enter a message. It then sends the message to the server through the NetworkClient class. The UserInput class also includes instructions for using the program and provides a list of available commands.

Our ***server*** package has 4 files in total: *ClientHandler.java*, *Server.java*, *BadWords.txt* and *ChatHistory.txt*.

Server.java: Server class is the entry point for the server program where its main method is.

ClientHandler.java: Handles incoming client connections and manages communication between clients and server.

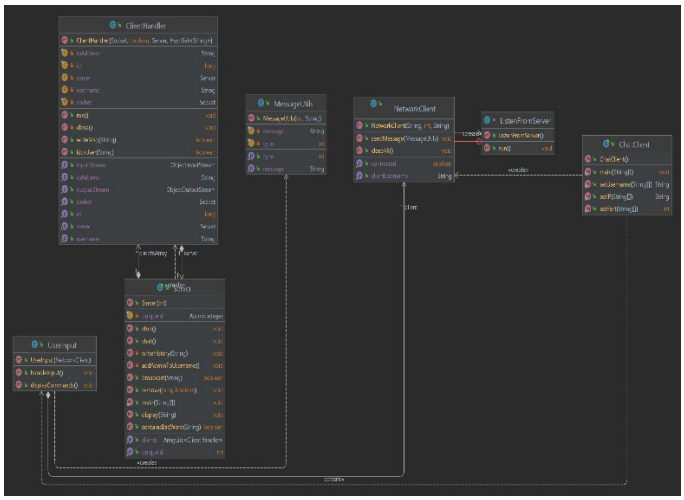
Badwords.txt: This is a text file filled with words that are considered inappropriate or offensive. When a user sends a message in the chat room, the server checks if the message contains any of the words listed.

ChatHistory.txt: This is a text file used to store a record of the public messages sent in the chat room.

The last package in our program is the ***shared*** package which has only one file named *MessageUtils.java*.

MessageUtils.java: This class is used to structure and standardize messages, the server and clients can easily recognize and handle different types of messages.

UML diagram of the project:



Key Methods

In the **Server** class, the **start()** method is responsible for setting up the server socket and listening for incoming connections. It creates a new **ClientHandler thread** for each connected client and adds it to the clientsArray. If the client array has only one element, it adds an admin prefix to the username. The method also handles the closing of the server and clients when the **keepGoing** boolean flag is set to false.

The **broadcast()** method in the **Server** class is used to send messages to all clients in the **clientsArray**. It checks if the message contains a bad word and sends a warning message to the client if it does. If the message is private, it sends it only to the specified recipient. Otherwise, it sends the message to all clients except the sender.

The **containsBadWord()** method in the **Server** class checks if a given message contains any bad words by splitting it into individual words and checking them against a **HashSet** of bad words.

The **run()** method in the **ClientHandler** class handles the communication between the client and server. It reads incoming messages from the client and processes them based on their type. If the message is a broadcast message or a private message, it calls the **broadcast()** method in the Server class to send the message to all or specific clients. If the message is a **logout** message, it sets the **keepGoing** flag to false to exit the loop and calls the **remove()** method in the Server class to remove the client from the **clientsArray**.

The **kickUser()** method in the **ClientHandler** class is called when the client sends a kick message to the server. It checks if the client is authorized to kick other clients (only the admin can do this), and then sends a message to the kicked client and removes them from the **clientsArray**.

The **getUsername()** and **setUsername()** methods in the **ClientHandler** class are used to get and set the username of the client. The HashSet usernames in the **ClientHandler** class is used to store all unique **usernames**, ensuring that no two clients can have the same username.

Design Patterns

We designed our program with the use of a few design patterns. Below are listed some of the design patterns we integrated into our program.

Observer Pattern:

The Observer Pattern defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. This pattern is used in the ClientHandler and Server classes to broadcast messages to all connected clients. When a message is sent by a client, the Server class notifies all clients by calling the writeMsg method of each ClientHandler object in the clientsArray list.

Factory Method Pattern:

Factory Method pattern, which is used to create new instances of the ClientHandler class. This is done through the use of a constructor with parameters, where the necessary parameters are passed in to create a new instance of the ClientHandler class.

Fault tolerance

Fault tolerance is an important aspect of any distributed system, including chat applications. In our chat application, fault tolerance is implemented in several ways. First, the server is designed to handle disconnections gracefully. If a client disconnects unexpectedly, the server removes the client from the list of active clients and broadcasts a message notifying the other clients. The system can continue to operate even if one or more clients disconnect, ensuring that the remaining clients can continue to communicate with each other.

The program makes use of several exceptions to handle unexpected situations that may arise during execution. Here are some of the exceptions used in the code:

- **IOException**: This exception is thrown when there is an error while performing input or output operations. It is used in the program when reading from or writing to files, sockets, or streams.
- **SocketException**: This exception is thrown when there is an error while using sockets. It is used in the program when there is a problem with the socket connection between the client and server.
- **ClassNotFoundException**: This exception is thrown when the program tries to load a class that is not found in the classpath. It is used in the program when reading objects from streams.
- **NumberFormatException**: This exception is thrown when the program tries to convert a string into a number, but the string is not in the correct format. It is used in the program when converting the command line argument for the port number into an integer.
- **NullPointerException**: This exception is thrown when the program tries to use a null object reference. It is used in the program when checking if the input or output streams are null before performing operations on them.

By handling these exceptions, the program is able to gracefully recover from errors and continue execution without crashing.

III. Analysis and Critical Discussion

Analysis:

The program is a multi-client chat server and client application. The server is designed to handle multiple clients at the same time, allowing users to communicate with each other in real time. The application is designed using Java programming language and uses socket programming to establish connections between the server and clients.

The server application has the ability to accept multiple client connections simultaneously, and once a connection is established, it creates a new thread to handle each client's communication with the server. The client application allows users to enter a username, which is checked for availability on the server, and then allows them to send messages to other clients in the chat room.

The server is also designed to handle different types of messages, such as regular text messages, requesting a list of connected users, and requesting to disconnect from the server using special commands. Additionally, it allows an admin to remove a user from the server if necessary.

The program also includes features such as logging chat history to a file and checking for the use of inappropriate language by checking the message against a list of bad words.

Critical Discussion:

The program demonstrates a good understanding of socket programming concepts and multi-threading in Java. The server is able to handle multiple clients concurrently, and the client application allows users to communicate in real time. The program also includes several useful features such as chat history logging and checking for inappropriate language.

However, there are some areas where the program could be improved. Firstly, the program could be made more secure by implementing user authentication and encryption of messages sent between the server and clients. Secondly, the program could also be improved by implementing additional features such as the ability to share files between clients and admin ranking. And Finally building a nice GUI(Graphical User Interface) for the program to be more user-friendly.

Overall, the program demonstrates a good understanding of Java socket programming concepts and multi-threading, but there is room for improvement in terms of error handling, security, and additional features.

IV. Conclusions

In conclusion, the chat room program that we have developed is a fully functional client-server application that allows multiple clients to connect to a central server and chat with each other. The program is written in Java and uses sockets to establish communication between the clients and the server. In addition to the basic chat room functionality, the program also includes some additional features such as support for private messaging and a filter for bad words and others.

Overall, the chat room program is a great example of a real-world client-server application that utilizes sockets for communication. The program is flexible, easily customizable, and can be used for a variety of different purposes. With some modifications, the program could be adapted for use in online classrooms, team collaboration, or any other scenario where real-time communication is necessary.

Acknowledgements

We would like to express our gratitude to the tutors: Dr.M.Taimoor Khan and Dr.Markus Wolf who taught us Java programming and provided us with the knowledge and skills necessary to develop this program. Their lectures and practical sessions were invaluable in guiding us through the process of understanding Java concepts and applying them to real-world problems.

References

Donahoo, K. L. C. a. M. J., 2008. *TCP/IP Sockets in Java: Practical Guide for Programmers.* Second ed. s.l.:Morgan Kaufmann.

geeksforgeeks, 2023. *geeksforgeeks.* [Online] Available at: https://www.geeksforgeeks.org/socket-programming-in-java/ [Accessed 20 3 2023].

Graba, J., 2007. *Network Programming with Java.* s.l.:s.n.

Priyanjalee, M., 2020. *medium.* [Online] Available at: https://medium.com/analytics-vidhya/socket-programming-in-java-75918f7e99bf [Accessed 23 3 2023].