Chat Application

Andrei Ardelean, Cosmin Sarageaua and Mac Hayatla (Group 37)

COMP1549: Advanced Programming

University of Greenwich

Old Royal Naval College

United Kingdom

Abstract

In this report we will talk about how the team managed to create a fully working chat application that the users can use to communicate between each other. We will go through our design and implementation, analyse, and discuss about our work, how we tested the application, what are the limitations etc. Our application contains most of the required functions, although two are missing due to constraints we have further talked about in this report.

I. Introduction

The Chat Application has been designed in Java. The aim of this application is group conversation between multiple users. There are two ways in which we could develop this application, either Graphical User Interface or Command-Line Interface. We chose to go for Command-Line Interface, as we believe that, although it is not as appealing as a User Interface, it still gets the job done and looks decent.

The development of this application follows the requirements of the coursework for the most part. There is implementation on some of the input required such as an ID, however the port and IP address have been left out for the sake of simplicity, as we are less experienced and therefore tried our best to keep it simple to make sure we would manage to bring the task to an end.

II. Design and Implementation

The first technical requirement in the coursework states that when a new member connects, they must provide a few parameters as input, such as an ID, port it will listen to and their IP address. In our implementation, the user connects to the server using the CLI command ***java Client***. After hitting ***Enter***, the user is required to input a username for themselves. As for the port and IP address, they have been implemented as static values in the client java file, for simplicity, and the port has been included in the server java file in order for this work. We used a ***localhost*** for the IP address and the port is bound to ***1234*** so that all clients automatically connect to the same port.

```
73  public static void main(String[] args) throws IOException {
74
75      Scanner scanner = new Scanner(System.in);
76      System.out.println("Enter your username for the group chat: ");
77      String username = scanner.nextLine();
78      System.out.println("Welcome " + username + "!");
79      Socket socket = new Socket( host: "localhost", port: 1234);
80      Client client = new Client(socket, username);
81      client.listenForMessage();
82      client.sendMessage();
83
84  }
```

For the server file, we used socket programming and multithreading. We started off initialising a server socket, then created a function for the start-up of the server that is responsible for the chat application. With the help of a ***try catch***, we test if the server socket is closed, case in which the server will be turned on and await new clients to connect. Through a client handler, every time a new user joins, the server will be informed through a ***print***

statement. The ***main*** function of the server file is short and compact, specifying the port of the running server and calling the start-up function.

```
public void startServer() {

    try {

        while(!serverSocket.isClosed()) {

            Socket socket = serverSocket.accept();
            System.out.println("A new client has connected!");
            ClientHandler clientHandler = new ClientHandler(socket);

            Thread thread = new Thread(clientHandler);
            thread.start();

        }
    } catch (IOException e) {

    }
}
```

The main requirement of the coursework is that, in the application created, all members should be able to send messages to every other member through the server. The client handler java file is the file where this is implemented. In addition to this, the client connecting and disconnecting is handled here, just as the name of the file suggests. The client handler utilises the ***bufferedReader*** to get the username from the user that connects to announce that the user has entered the chat. To implement group chatting, we chose to create a function that would the next line of input from the user and save it in a variable. Later, in another function, we use the ***bufferedWriter.write()*** so that we can broadcast the message to all users connected. After the message is sent, the ***bufferedWriter*** is flushed so that it can be empty for future messages.

```
public void broadcastMessage(String messageToSend) {
    for(ClientHandler clientHandler : clientHandlers) {
        try {
            if(!clientHandler.clientUsername.equals(clientUsername)) {
                clientHandler.bufferedWriter.write(messageToSend);
                clientHandler.bufferedWriter.newLine();
                clientHandler.bufferedWriter.flush();
            }
        } catch (IOException e) {
            closeEverything(socket, bufferedReader, bufferedWriter);
        }
    }
}
```

The last implementation that is worth mentioning, from the client handler java file, is a small function that broadcasts a message on the chat whenever a user has disconnected. This is a useful feature, as we believe it is important to let the active members know when someone has left the chat. Therefore, there won't be any confusion as to why someone has stopped answering back or messaging in the chat due to leaving it.

The last java file that we created for the application is the client. This client file contains a pretty straight forward ***main*** function. We used a socket that connects to the ***localhost*** address and the same port as mentioned above, and a client variable that uses the said socket and the username that is previously declared and inputted by the future user that will try to access the server.

III. Analysis and Critical Discussion

Looking back at the first and second year of Computer Science, this coursework can be evaluated as one of the more challenging ones. It takes a lot of dedication, studying, research and teamwork for a group of students to achieve the desired result for this coursework. We believe that we managed to design a fairly decent chat application after a lot of hours of work. Although it does not satisfy all the requirements, it is a very much functional application that can definitely do the job asked.

If we take a look at the work done, the chat application consists of a few java files instead of just one. This is the way we achieved modularity, by breaking the project down into multiple pieces so that whenever we encounter a problem, it would not compromise the entire application. Therefore, if there is any problem within the client, this will not affect the server and vice versa. As for fault tolerance, we have encountered a few errors along the way. To move forward from these errors, we had to use some ***IOExceptions***, so that the program would not fail to run because of them. We also used ***try catch*** blocks in order to achieve fault tolerance.

The code that was developed for our application is clean, tidy, and short to an extent. We designed it so that the code would be easy to understand and follow.

As for the testing, we have made a separate word document in which we created a table to keep records of testing. Records we kept consist of Function Tested, Did it work, What needs to be improved/fixed if applicable, Date tested and lastly a Complete section where we ticked off any function that passed all testing. This aided us into making sure that each function of the program we created worked as it should and problems don't appear out of nowhere when using the program, giving the user a pleasant experience to use.

As for the limitations of the program, we have tried our best to implement everything we could, but due to our expertise in Java and also the time frame, we unfortunately couldn't do some of the functions. The things we weren't able to do are the coordinator role and also the private message function. By having these two missing it unfortunately limits the program a bit as the users cannot perform some of the actions therefore it creates a weakness in our implementation choice. We believe that having a bit more time would have probably allowed us to create all required functions for the chat application.

IV. Conclusions

In conclusion, we believe we have created a fairly acceptable and properly usable chat application that contains most of the important functions required. As for the future work, we can definitely improve this application by implementing the missing functions and maybe more stuff that will improve user satisfaction.

References

Barhoumi, J., 2021. *Medium.* https://medium.com/nerd-for-tech/create-a-chat-app-with-java-sockets-8449fdaa933

Kabra, A., 2014. *StackOverflow.* https://stackoverflow.com/questions/22719106/java-client-server-chatting-program

Malachi, 2009. *Stack Overflow.* https://stackoverflow.com/questions/471342/java-socket-programming

Marian, T., 2019. *StackOverflow.* https://stackoverflow.com/questions/56454187/multi-client-chat-application-in-java

Minh, N. H., 2019. *CodeJava.* https://www.codejava.net/java-se/networking/how-to-create-a-chat-console-application-in-java-using-socket