CLI NETWORKED DISTRIBUTED SYSTEM

Tyler Owen-Thomas and
Nathan J Hopson

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

Abstract - We have produced a program that allows users to communicate with one another via a local chatroom, this was done as specified in the brief. The chat can be used in the command-line interface and can handle an unlimited number of users, which can all talk in the same chat allowing them to communicate with one another.

Keywords: Keywords: Client, Server, IDE (Integrated Development Environment), socket, thread, classes, methods.

I. Introduction

In the world of today, online messaging is used around the world to enable people to communicate from anywhere at any time if there is internet access. Apps and services such as Discord, which now has over 130 million users according to businessofapps.com[2]are prime examples of how sophisticated this technology has become, and they are an inspiration to us all when it comes to creating chat programs/applications. Like our application, parts of Discord are also written in Java. There are countless online chat websites, apps, and services available to people around the world. Our program uses the core principles that are used by the leaders in this field and applies this to a local version we have created, an example of this is the client-server model used in a network application. This model is based on the user being a client and the server being hosted by the chat application which the person is using, the client sends requests to the server and communicates with it. Each user will do this, and the information will then be exchanged appropriately across the server, then feedback to the user will be given.

II. Design/implementation

For our project, we wanted to keep the design and implementation as easy to understand as possible.

To create our project, we used one of the most popular Java IDE named "IntelliJ IDEA". We used this because it is an IDE created purely for Java development, which means that its workflow processes are perfect for our project. It also supports many useful frameworks and add-ons, more so than most other IDE which supports Java. When looking for the perfect environment to design our project in we took inspiration directly from other Java developers which had views such as "*IntelliJ IDEA is undoubtedly the top-choice IDE for software developers. Efficiency and intelligence are built into the design, which enables a very smooth development workflow experience, from design, implementation, building, deploying, testing, and debugging, to refactoring! It is loaded with features and also offers a plethora of plugins that we can integrate into the editor. I switched to using IntelliJ IDEA 5 years ago and have never looked back. It has certainly made my life easier. I am producing more with less effort"[1]*. IntelliJ is leaving its mark on the programming world despite it not being the industry standard.

The server process starts on a computer system and then waits for a user to request a connection. After that, the UserManager process can start. This can occur in the same system or on another. This begins by sending a request to the server. Then, the server will receive this, then it will wait for the next.

This will repeat while the server is active. To communicate, a connection must be established between the Server and the User. A connection is created by five components: the protocol used, the source IP address and its port number and the destination IP address and its port number.

The ”java.io” package allows the input and output streams to write to and read from while communicating.

The Java.net class is used to implement socket objects and to perform various network operations such as sending/reading data and closing the connection. Each socket that has been created using this class is associated with exactly one remote host.

Java.util.scanner: This uses the terminal to capture data that will be inputted by a user. This can then be used to store as a variable.

This java.net.server.socket class is used to create a system independent implementation of the server-side of a client-server/socket connection. The server waits for the request to come in over the network, it then performs some operations based on that request and finally returns a result/message to the requester.

Classes used in our design

In our design, we have 3 classes which are server, user, and user manager.

Server: The Server class is used to open and create a server socket so that users can connect to the server. This is where the server will be started and closed. It has three methods, startServer, closeServerSocket and main. The Main method is responsible for taking the port information from the user. First The startServer method will run a while loop in a try block. The while loop will continue running as long as the server sockets aren’t closed, this is to make sure that more than one user can connect, and it also means it is always listening for new connections. Furthermore, the server will wait for a user to connect, once it does a new thread will be created for each UserManager instance. In this method, we use threads because for this program we will have more than one user and threads allow us to run more processes at one time.

User: This is the class that is responsible for connecting the user to the server via a socket. It does this by taking the user input, (this is their username and port number) in the Main method and then sending it to the server. It is also responsible for managing the messages sent from users to the server via two methods sendMessage and listenForMessage. These methods use the character stream from the readerBuff and the writerBuff. Lastly, contains a method that is responsible for preventing an infinite loop and closing the processes and connections.

User manager: This is the class that managing key lists, “userList” is responsible for storing each user within an array. This array then makes it easier for each user to be managed. Index 0 is the first person who joins, and this is the coordinator, in this class, we have made it so that when one person leaves or if the coordinator leaves the users are moved and reassigned accordingly. This makes it so that we can assign a new coordinator in case the original coordinator disconnects. User Manager is a component of the class User and is a key example of where we have used component-based development.

Design patterns

To implement our program, we used the classes above and design patterns together to create a functioning program. Listed below are some of the design patterns we incorporated into our program.

Adapter pattern
This is a structural design pattern that allows two things that cannot be directly connected. A prime example of where this is done is in UserManager where which brings together subclasses from different super classes and incorporates them to work together. An example of this is in the class User, which is where the information such as port number and username are imported from the user, these are then stored and used in the class UserManger.

Prototype pattern

This design pattern has been implemented in our code multiple times; the prototype pattern is where multiple objects are reused to enhance performance. An example of this is when we implement the socket connections for every new client. This is vital to our application because we want to connect multiple users with the same privileges.

Group formation, connection, and communication:

Groups are formed in the server by each client joining the server with the same port number. Threads are then created by the server which reads data passed in from the UserManager class creating the group on the server. Users are then able to communicate with one another on the server via the UserManager class which sends data to the server via a socket. The server is constantly listening for new communications from the client and when it receives a new communication it will then act accordingly.

For each User, a static array list called userManagers is used to store the User information. The main objective of this array list is to keep track of the users and when a new user is sending a message, a loop through the array will be performed to make sure all the users will be able to read the messages apart from the original sender.

Unique ID:
For representing each username, a String called “userName” is passed in from the User class which uses

the Scanner to capture user input. See source code User.java line 108 – 131. Before the new users' information is stored, the code will check to see if "userName" is unique, if it isn't then the server will randomly assign a number to the end of the username. For example, if Jim is already in the group and someone else tries to join with the name "Jim" it will change it to "Jim(Random Number) the new username will be stored in an Array List "userLists." See source code UserManager.java line 41 – 47.

Port address of the server:
Upon start-up of the server, the server user is prompted to enter a port in which they want to host the server, this can be found in line 52 to 64 in the server class. Additionally, the user after inputting their username will use the scanner to input what port they want to connect to. Refer to source code in User class lines 112 to 127.

Communication:
The bufferedwriter and buffer.reader are used to send and read messages. These are used to get the input and the output streams from the sockets. The run method will continually listen for messages while the socket is connected. A String called messageFromUser will be assigned by bufferedReader.readline.

To send a message, a for-each loop is implemented to iterate throughout the userManagers Array, to make sure everyone gets the message apart from the original sender. This uses buffered write, followed by a newline and a flush. To make sure that the original sender doesn't get their own message an if statement is used to ensure this.

A system.out.println is used to communicate actions to the server. Whereas the messageToSend function is used to broadcast actions to the users. For example, when a user leaves or the socket gets closed, the username and a message are sent to the server and users. This is done through the retrospective forms of communication.

Group state maintenance

The group state is maintained with unique usernames, for example when a user enters the same username as another user. In real-world chatrooms there may be an occasion where two people join with the same name, this means that there needs to be the ability to distinguish between the two same users on the same server. Another example of group maintenance is storing the users in two different arrays, one array is used for maintaining the group while the other is used for maintaining the coordinator. The server also creates a new thread for each user.

Coordinator selection:
The Array List userList is specifically used to control who the coordinator is. The way this is implemented is that each user gets added to the list when they join, keeping an ascending order of connections based on when the user establishes a connection. The coordinator is always the first person who joins, this means that the coordinator will always be at index 0 in the array. Using a Get function at index 0 will ensure the user at index 0 Is always assigned, coordinator.

As outlined in the brief, the coordinator needs to be allowed to access certain information. How our program has allowed this is by allowing the coordinator to input commands. These commands will then return information, this is not available to any other user other than the coordinator.

III. Analysis and Critical Discussion

Due to limitations of the server design and user class, it was not feasible to implement an IP address function. As a result of this, the user connects to the server via a port number exclusively. Additionally, this means that the coordinator cannot see the other users' IP addresses.

When the server is active, every time a new user connects a new thread is created, and the information of the users gets stored in an array list. This means every time the user leaves or the server restarts no information is stored. This allows an easy implementation of the coordinator; however, no history of the users is saved.

Port connection
Another limitation of our application is when a new user inputs a port number that the server is not hosting, a message to try again will be outputted, and the application will quit. If a loop is implemented and the user inputs a port number that the server is not hosting, this will loop until the user will put in the right port number. Unfortunately, the way the code is designed and implemented the user won't be able to connect to the server as the port number variable will not be able to be passed through, so the user will have to restart the application anyway.

Fault tolerance

In our application we have ensured that there is fault tolerance, this is so that the users experience of the program is seamless and the user understands what is going on when something goes wrong.

To make our application robust we used the 'try and catch' method in the code. These blocks will have implicit exception handling which will print suitable error messages that will allow the users to understand the errors encountered and to understand why the application is not running properly.

IO Exception
Regarding the network-related methods, they will throw an "IOException" in case any error occurs. When this

exception occurs the code will close the application, this is to ensure no errors can continuously run.

Input mismatch exception
This exception occurs when the user inputs a different variable from the one that is required. For example when a user inputs a string instead of an integer. This then implements a try-catch, along with a while loop to ensure the right data variable is inputted, it also stops the application from crashing. This is used on both ports for the server and the user.
Index out of bounds exception
Due to using array lists to store user information, the server and user manager always look for index 0 in the array. Once all the users have left there is no index 0 which then throws an error. A try-catch block is implemented to catch the exception and return it with a string, informing the server that there is no one left to assign the coordinator to.

IV. Conclusions

One of the strongest points of our code is the fact that if two users have the same user name, the second user will have a random number added to the end of their username. Additionally, the error and fault handling has been executed flawlessly. Due to time constraints and knowledge limitations, unfortunately, not all of the requirements have been met, however, a great effort has been made. All the clients can successfully communicate with one another and the server. The coordinator handling works as expected. Although Junit testing hasn't been completed successfully, if we were to do this project again, next time we would allow more time to implement this.

Acknowledgements
We would like to take this opportunity to thank our tutors Taimoor and Markus for always being swift in helping us with any queries we may have.

References

[1] Mary Grygleski. (). IntelliJ Idea: What Developers Say. Available: https://www.jetbrains.com/idea/. Last accessed 27 March 2022.
[2] DAVID CURRY. (2022). Discord Revenue and Usage Statistics (2022). Available: https://www.businessofapps.com/data/discord-statistics/. Last accessed 24 Mar 2022.
In code
Server Line 29: Nam Ha Minh. (2019). How to Create a Chat Console Application in Java using Socket. Available: https://www.codejava.net/java-se/networking/how-to-create-a-chat-console-application-in-java-using-socket. Last accessed 20 Mar 2022.

User line 90: Anon. (2006). *are there any ways to catch cntrl-c in a console java program??*. Available: https://www.linuxquestions.org/questions/programming-9/are-there-any-ways-to-catch-ctrl-c-in-a-console-java-program-464904/. Last accessed 22 Mar 2022.
Usermanager line 27:
anon. (2014). Generating a Random Number between 1 and 10 Java. Available: https://stackoverflow.com/questions/20389890/generating-a-random-number-between-1-and-10-java. Last accessed 12 Mar 2022.
Usermanager line 17: mkyong. (2021). How to get current timestamps in Java. Available: https://mkyong.com/java/how-to-get-current-timestamps-in-java/. Last accessed 15 Mar 2022.
Server line 50: anon. (2014). Catching an InputMismatchException until it is correct. Available: https://stackoverflow.com/questions/20075940/catching-an-inputmismatchexception-until-it-is-correct. Last accessed 14 Mar 2022.