Online Chat Java Application

Author Izmir Uruci & Mohammed Ali
COMP1549: Advanced Programming
University of Greenwich
Old Royal Naval College
United Kingdom

***Abstract*- Creating a Java application has many different components which are needed to be included as well as providing a very easy to follow flow while entering the codes. In this report, we are going to describe how we were able to create a peer-to-peer chat room, one specific coordinator initiating the connection, adding time stamps on each activity as well as notifying everyone when a user joins or disconnects.**

When creating this application, we used many different java elements such as threads, polymorphism, inheritance, classes, objects components, etc.

Key words: *(Java Application, Peer-to-Peer, Threads, sockets, server, client)*

I. Introduction

The application we have produced is a peer-to-peer chat application where all the users are responsibly to communicate separately as well as broadcast a message to a specific user. We use direct messaging over the internet in a daily basis which emphasises the importance of such application and its implementations in our lives. Our application is run in one machine for testing purposes and simplicity; however, the application will run as well in a real-life scenario. The following is a brief explanation of how this application works. When the program is run, the user will enter port number to and server address to initiate a listening port. This user will then become the co-ordinator of this chat.

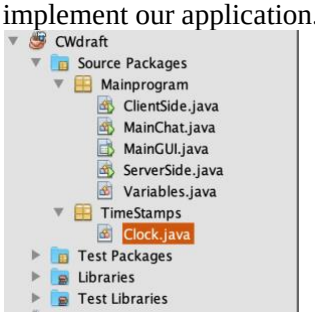
Once the co-ordinator initiates his listening port, the next person, will run an instance of the GUI which will also require the user to enter his listening port and server address as well as the port number (Co-ordinators listening port) in order to establish connection. When the third user joins, he/she will also need to enter the same details and to connect, he/she needs to enter one of the active user's port number in order to be part of the connection. Each of the user will have to enter a username in order to identify themselves in the chat.

Each user will also be notified in case of an event happening such as a user connecting or disconnecting. Everything that is happening in this group chat is followed by a time stamp. Another feature of this application is to let each of the connected user to see who is actively connected to the chat at a given time. It is a simple as clicking a button to find out the list of active users in the chat.

An important feature implemented in this application is the fact that the users are connected to one another independently meaning if the co-ordinator, who initiated the chat, leaves the group chat, the other user's communication shouldn't be affected. In addition, a next co-ordinator is chosen to take over as well as everyone is notified when the co-ordinator leaves and a new one takes over.

II. Design/Implementation

Creating this application, we have implemented different design different implementation methods and components which are going to be discussed in this section of the report. To start off with, we have used six (6) different classes to implement our application.



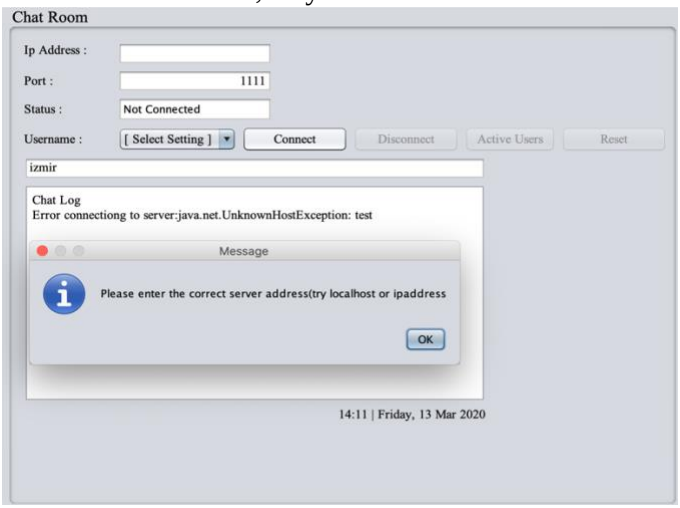
The "ServerSide" class is responsible of maintaining the active port running and waiting for connection. The method in this class takes two parameters, being a port number (int) and also a server address (String). This is then passed to the next method where the ServerSocket is initialised. This way, the server is then ready to accept connection at the given port. Keeping in mind, each of the application in a computer has its own port where it accepts connection from the server, it is important that the port isn't already taken. To make sure this isn't the case, we were able to implement an error checking condition which checks for live ports of the same number. Here is the result:

Having this in place, would prevent any system crashing when running two or more instances of the application in the same port number at the same device, which like mentioned above, it is not allowed. In this application we were able to make use of threads which allowed us to connect users separately from one to another. Even though we have set the server address at localhost to be default, after testing it from different machines using live ip addresses, three users were able to communicate through the internet without any errors.

One of the requirements of this application was to be able to update the online users each time a user joins or leaves. The way we were able to achieve this was by creating an ArrayList of type ClientThread created in the "ClientSide" class, which would add each of the threads initiated from each instance of the application to their own ArrayList. This way, each of the users have their own list of online users, so each of the other users will have everyone connected to the chat expect themselves because everyone has a separate list which they store their separate users connected to them. Here is an example of what it looks like in the actual application:

While crating the application, we though carefully about fault tolerance and where we thought of every possible fault which could occur in runtime therefor, we are going to now take you through some of the main ones which are essential to be mentioned. To start off with, when a user tries to initiate a connection, all the required fields must have specific data in order for the application to comment the connection. Such fields are port number, server address, and Id or Name. If the port number is left empty and the "Connect" button is clicked, the user will receive this error:

If the server address field is left empty or an invalid one is entered, they will receive this error:

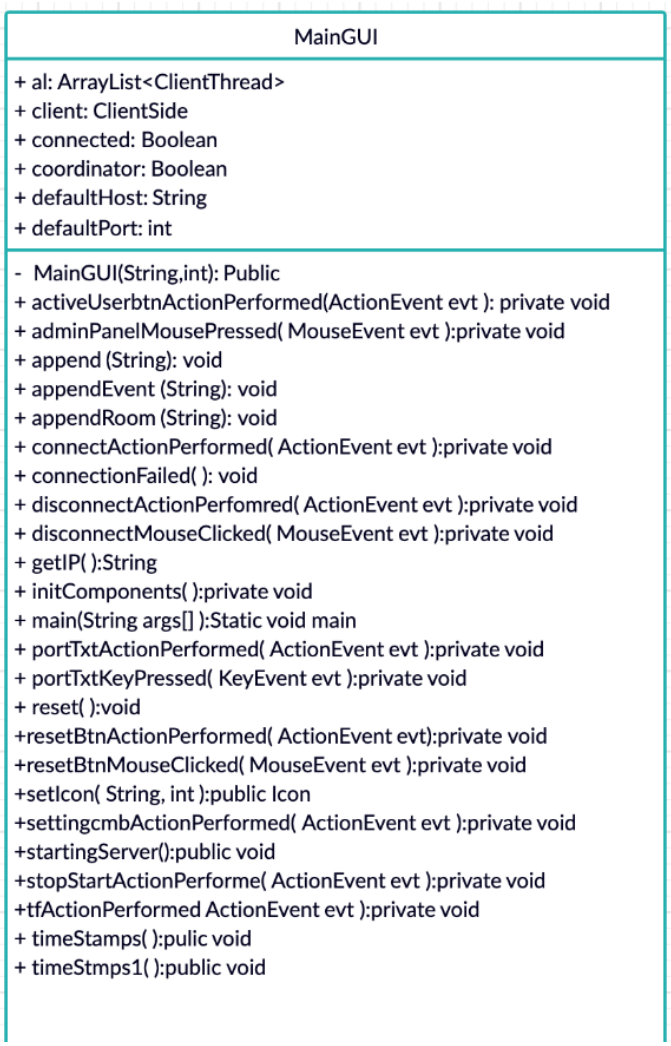


Another requirement of this chat was to implement a log feature which would record the behaviours of the users in the chat such as new connection, co-ordinator leaving the chat, or a user leaving the chat. We were able to implement this feature into a separate text area which is automatically updated. In addition to this, everything in this chat has a time stamp, which also fulfils the requirement which was to have a clean time of everything happening in the chat. The way how we achieved this, we created a separate component, in this case a class which is able to achieve this. Throughout the source code we have provided, you will come across GRASP design pattern where we made sure touching the inner components within the pattern. Coupling is one of the first one used in this application due to the need to inherit data from one class to another, or even information between methods. For example, we needed to have knowledge of the ServerSide class in order to initiate a user, so this information was needed to be accessible before running the client thread. Cohesion is another element which we tried to keep as low as possible. By having high cohesion, one function would have a lot of responsibility, therefor it would be a lot harder to produce

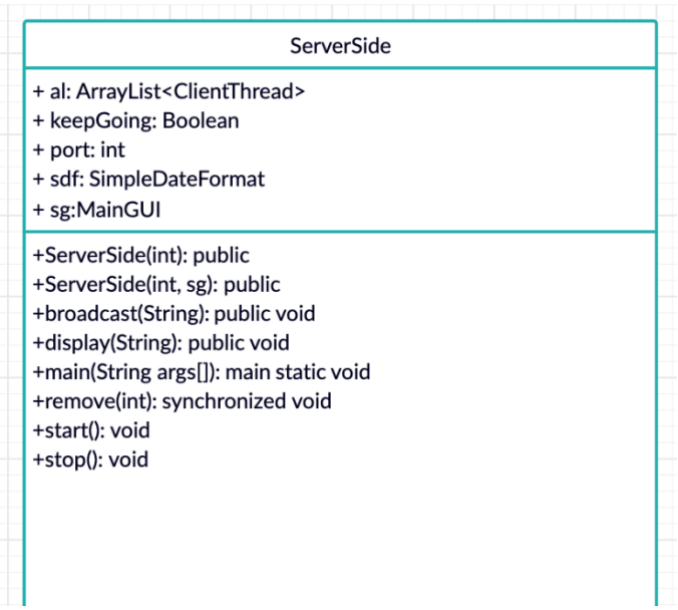
an efficient code. An example of this is the facts that we have used six different classes to do separate tasks without staffing everything in one class. Another design pattern component which played an important role, was the ability to have the right creator of objects. An object in created from a specific class only if a set of rules are met. For example, in our application, we have created a objects "client" of type ClientSide from "MainGUI" class with its parameters and the reason why we were able to use "MainGUI" as the creator is because class ClientSide has all the initialising data for class "MainGUI" to create that specific object. Another elemnt of the design pattern which we have implemented is the protected variation. The reason for using this, is to protect the behaviour or the application by keeping the code authentic. There are different methods this how this can be achieved, but we have used encapsulation as one of the ways which would prevent the changes of any method my making them private so the elements within that method are only visible to the method only. An example of this is the use of private method "writeMsg" which takes a String as its parameter and stores it into a local variable which is then used to be displayed. Notice that this string isn't stored into a variable in order to keep the String content authentic. We have also implemented into the application the class "Variables" where we use objects to pass the data from one client to another instead of passing bytes. This is a much easier way to achieve this task. We have also used the aspect of method overloading where the same method is used twice with different parameters. Another implementation in this application is the unique id system. This was achieved in the background of the application by making sure each of the threads(users) have a unique id which is then stored in the arraylist which is then used when deleting a specific user from the active users when they disconnect.

We were able to also give the users the ability to send a message to a specific user or broadcast the message to the entire group. The way we achieved this, was to allow the user to enter a specific port number which represents the user that way, they can communicate to the user separately. As specified in the coursework specification, the port n umber is a verbal communication, meaning each user knows the ports of other users.

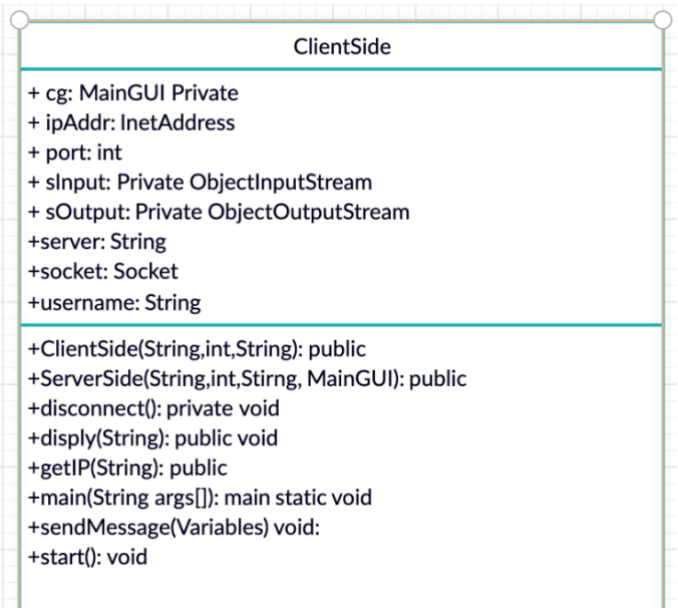
Here is a breakdown of the UML diagrams of the classes in this application:



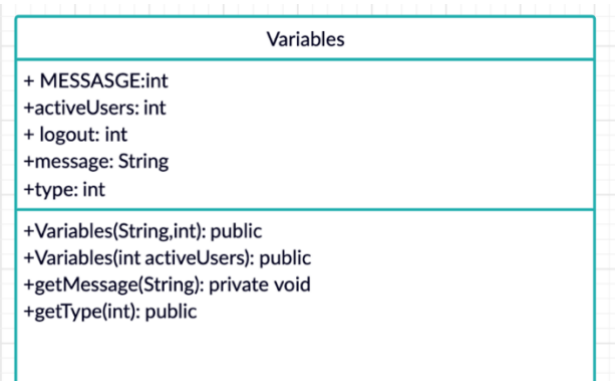
ServerSide Class



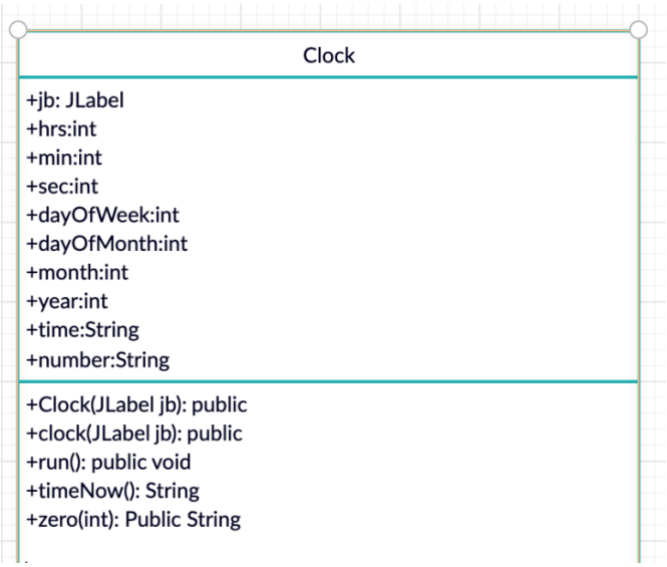
ClientSide



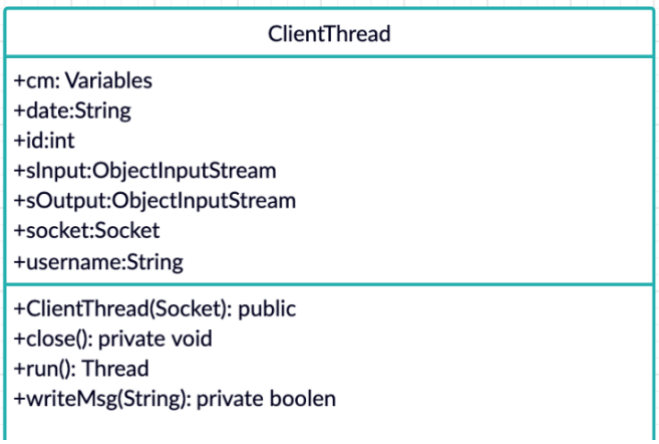
Variables



Clock



ClientThread



III. Analysis and Critical Discussion

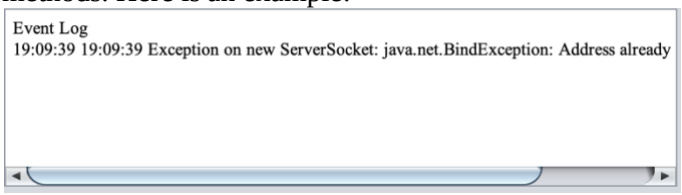
This application we have created, matches most of the required features in the specifications. However, we are aware that there are still improvements which can be done to the application. One of the improvements which we are aware of, it's that the application is able to maintain the communication between users even if the co-ordinator leaves the chat room, however we were not able to notify the users of who is the next co-ordinator taking over. The service remains working without any error.

Another improvement which we should mention is that we should have carried our Junit testing throughout the application classes in order to maintain the flexibility and control over our code. While trying to do this, we were facing a lot of difficulties with the void methods which we

weren't able to test. The examples given to us in the lecture weren't really similar to the actual task we were asked to do, therefore our ability to carry out these tests was not at the right standard.

In addition, even though we have used many different designing tools, there are also a few other tools which we were unable to implement. Having as many tools as possible in our code, would have helped us a lot with organising and producing a much better application, however we were able to implement a few of the methods which are motioned in section two of this report.

When getting Java generated exceptions, we were not able to remove them from the application text areas. An example of this is when trying to run the same port number in the same machine. We were able to implement our own fault measures; however, Java also implements its own exceptions which are displayed when using append methods. Here is an example:



IV. Conclusion

To summarize, we believe that our application fulfils most of the requirements specified. We have used many different methods and techniques in order to achieve what we have produced. We of course faced a few difficulties during the process, like we have mentioned above with the bits that need improving, but an overall application where users are able to communicate with each other without any problem even of the application is run in different computers.

We have used different methods, we were also introduced to the aspect of sockets, port, threads, etc which was all new to us and we believe that we have produced a good overall application considering our experience and knowledge with such aspects.

Using inheritance, conditions, polymorphism, design patterns, different encapsulation variable status is what helped us getting this program put together to what we have submitted. We now understand the importance of testing, using the current approach to a specific task, we also understand what components are and how they help programmers to reuse code.

References

Class

InetAddress @ https://docs.oracle.com/javase/7/docs/api/java/net/InetAddress.html. Copyright © 1993, 2018, Oracle and/or its affiliates. All rights reserved. Nat. Comput. Security Center, USA

Advanced Programming Lectures "*Dr. M. Taimoor Khan & Dr. Markus A. Wolf"*. University of Greenwich, "Computer Security and forensics":

Docs.oracle.com. 2020. *Writing The Server Side Of A Socket (The Java™ Tutorials > Custom Networking > All About Sockets)*.

www.javatpoint.com. 2020. *Java Socket Programming (Java Networking Tutorial) - Javatpoint*. [online] Available at: <https://www.javatpoint.com/socket-programming> [Accessed 01 March 2020].