Tutorial 1 Report

M Shepherd, 19059019; M von Fintel, 20058837

8 August 2018

ChitChat - It's where it's at!

1 Project Description

This project was to implement a single server, multiple client chat program. Many clients can connect to the server and can chat, that being send messages to all the other clients connected to the server. Clients can also send messages to other individual clients in a function known as whispering. What follows is a brief, concise description of the different parts of the project.

1.1 The Server

The server handles all the incoming messages sent from clients and sends them on to all the other clients. It creates a `ServerSocket` on port 8000. It continually listens for new clients to connect and creates a new socket each time someone does. It is therefore a multi-threaded implementation. More will be said on this when the client side is discussed.

There is currently a limit to the amount of clients that can join to make sure clients have a seamless, exclusive experience. This is currently set at ten clients. If the chatroom is full, i.e. there are already ten clients, the new client will be warned with an appropriate message and will be shutdown.

1.2 The Client

The client runs behind a GUI. The GUI has text areas for the chat and message composing. A list of online users is also displayed. When the user starts the client, the user has to input a username and the host (IP address of the server) they wish to connect to. The input is done via `JOption.showInputDialog()`. The inputted host is verified and the user is notified if it is unknown. If it is known, the client connects to the Server at that host. The username is also checked against the list of online users to make sure it will be unique.

The client then listens for messages that come in the `DataInputStream` from the Server. All messages, including whispers, and the list of online users are sent to clients on this `DataInputStream`. Messages types are identified by the prefix of the string. For example, if the message starts with "`*userNames*`", then it is a string representation of the list of online users.

Sending messages, whispering etcetera are all done by buttons that have `ActionPerformed` listeners.

2 Features

2.1 Included Features

The list of features included, with a description of each:

- Client GUI
  - The GUI has text area for message input and messages sent from other users.
  - A list of online users is sent to all Clients connected and is displayed. If a user is selected from the list, the Client will whisper to that user.
  - Then number of online users is also displayed.
- The Client can send and receive messages from the Server.
  - This is done concurrently.
  - Many Clients can connect to the same Server.
  - When a Client sends a message to the Server, it is then sent to all the other Clients connected to the Server.
- Different Client instances can also message each other privately, known as whispering. The message is sent via the Server, but is only displayed in the GUI of the Client it is meant for.
- All users connected are notified when other users are connecting and disconnecting.
- When connecting to the Server, all users have the ability to chose a username.
  - This username is checked for duplicates against the list of other users connected to the Server.
- Stable connection between Client and Server.

- If either are interrupted, the other will remain stable.
- Clients connecting and disconnecting do not affect the functioning of the Server.

3 Algorithms and Data Structures Used

A `clientInstance` class was made to store the information of each Client instance. The Server stores an array of type `clientInstance`, with each instance representing a Client that is connected to the Server. `clientInstance` stores the `DataInputStream` for the Client, the `PrintStream` for messages coming from the Server, the socket and other information. In the `clientInstance`'s `run()` method, it listens for messages and performs certain tasks based on the prefix, as described earlier. The `run()` method runs concurrently as the `clientInstance` class extends the Java `Thread` class.

The `ClientPane` class, which runs the Client behind the GUI, is also concurrent. Each user has their own client, each with their own GUI. Messages can be sent and received concurrently.

Messages are sent from a Client to the Server on a `PrintStream` and messages are sent from the Server to all the Clients by a `DataInputStream`. All the messages get sent to all the Client instances connected to the server. If, say, the message is a whisper between two specific Clients, each Client will receive the message, but will only display it to the user if the message is meant for that Client.

4 Experiments and Testing

The project was tested throughly. We started out by making a basic working implementation of a single client message passer and then worked from there, adding features and improvements in small steps. The first small steps making separate threads for sending and receiving data and then allowing the server to spawn multiple clientInstance threads to allow for multiple client connections. Every time a feature was added, it was experimented and tested to make sure it worked. At each step we also tested the general functionality of the project to ensure the addition did not stop anything else from working

properly.

When we had what we thought was our final implementation, we tested it at the NARGA labs on many different computers, all connecting at the same time. We tried our best to model a real world situation, by having many users typing, sending, receiving, whispering, connecting and disconnecting, all at the same time. We also tested for Server disruptions and made sure it was stable.

The GUI was also subject to much experimenting and testing to make sure that it was usable and did not break. We implemented many logic rules for buttons and other input areas to make sure the user could not do something that they should not do.

Most of the issues that we came into contact with involved learning how threading and sockets worked together in Java and making sure that the client and server were communicating as we expected. Specifically, we struggled with keeping the list of connected users consistent between all of the clients and servers and it took many different tests before we settled on the method that we used.

Many other small bugs and other problems were found. These were all fixed, resulting in a more functional and more stable project.

5 Conclusion

This was a fun project to work on, and both of us learnt a lot. The whole cycle, from building a basic implementation to testing the final product and all the steps in between, was a big learning curve. It is safe to say that we are both more confident in doing development in a team and know how to optimise the process.