**Summary**

The summary section should discuss your project experience. You should discuss how you approached the project, including anything difficult or interesting, what was learned, and the end result. This should be at least 1 page in length.

This project is implemented by Socket and thread. The project programmed in JAVA and Eclipse neon.2 IDE. The java JDK used is 1.8. I created three classes in this project. Client class works as a client-side program and presents a user interface. ClientWorker and ThrdServer class work together to support multithreaded server.

**client class**

This class works as a user interface and communicate with server to simulate a client in a chat room. In its main method, users are requests to provide two arguments. One is the host name and another is portnumber. Then main method call a listenSocket to establish connection to server with the host name and port number. This method first creates a Socket object with the host name (for example csgrads1) and port number (a four-digit integer) where the server program is listening for client connection requests. Next, it creates a PrintWriter object to send data over the socket connection to the server program. It also creates a BufferedReader object to read the text sent by the server back to the client.

After connection is established, method communication will be called to interact with server. In the communication method, first ask user to provide a user name and client send user name to server. After checking the user is good to log in (if an unknown user, the amount of the known user cannot reach the limit), a menu with 7 choice will be showed on the screen. Whenever a valid choice( valid input from 1 to 7) is made method execChoice will be called to execute corresponding choice and waits feedback from server (which is implemented by swictch-case). While the thread is active (no interrupt or exception occurred), users are continued been asking to make a choice.

**ClientWorker class**

The ClientWorker class implements the Runnable interface, which has one method, run. The run method executes independently in each thread. If multiple clients request connections, same amount of ClientWorker instances are created, a thread is started for each ClientWorker instance, and the run method executes for each thread. The run method helps server to communicate with the client with each other, First it creates a PrintWriter object to send data over the socket connection to the client program and a BufferedReader object to read the text sent by the server back to the Server. Then while the thread is not exited, the thread continues to read text from client, at each time, if an input is a string (in that case that would be a name), a user want to log in. then check the status of the user’s name (is that name already unknown? if not known, is the amount of known user already reach the limit?) and do the correct response. If an input is a number (in that case client make a choice), execute that choice by exec method which implements a switch case that help server communicates with client according to choice made by client.

Also, in ClientWoker class, a static semaphore sem (1, true) was created to enforce mutual exclusion which could ensure that at each time, only 1 client can edit or retrieving user information (including known users, online users, and message of all users) that stores in the ThrdServer class. Following scenario showing the pseudo code where applies semaphore:

// scenario that execute a choice

sem.acquire();

exec(choice);

sem.release;

// scenario that a client want to log on

sem.acquire();

get name of this client;

//if user unknown, add this name to known or known users reach the limit

// if user known add user’s name to online hashset

sem.release;

**ThrdServer Class**

The main method of ThrdServer class requires passing a port number. After a new instance of ThrdServer is created, call the listenSocket method with port number to wait for a client to log in. The listen Socket loops on the server.accept call waiting for client connections and creates an instance of the ClientWorker class for each client connection it accepts. The ThrdServer also hold the information of all users. In my design a hashset name stores all the unknown users’ name, a hashset online stores names of all client that connect to the server, and a hashmap msgTable stores the messages of all known users. In the hasmap, client name work as a key to the hashmap. Each key value can retrieve an arraylist that contains up to 10 messages(string).

In this project, I learned how to build connection between client socket and server socket, how to implement multithreaded server, and how to implement communication between client and server with BufferedReader and PrintWriter. In this project, I also learned very useful pattern and matcher class of java, which is very helpful in split a string according to some pattern, and

The most difficult part of this project is to decide which message need to transfer between server and client and the order of the message been passed. This is especially difficult to arrange this logic in choice 3. I want to consider the length of the message, is the user unknown, is the user amount reach the limit, and is the user already have 10 message. After times of attempt, I finally find an appropriate order to communicate information under choice 3.

Another difficulty is to decide a data structure to store user information in the server. After few rounds of attempt, I decide to save each client’s message in an arraylist, and save each arraylist in an hashmap with the that client’s name as the key. Then every time a client read their message, the arraylist with the key of that client’ name will be clear. Also, I store known clients’ name and online users’ name in two separate hashset. I think the benefit of structure, when you compare it with arraylist, is that in hashset, duplicate name is not allowed.