**Assignment 1**

**Client-Server Multi Chat Rooms using Socket and Multi-threading using Java.**

The client server multi-chat room is called ChatApp. The server side and the client side are built-in JAVA eclipse. The Socket and Server Socket class is obtained from the Java.Net package, and JAVA swing (JFrame) is used to create the interface for the **chat server**, **login of the client,** and **chat room** interface.

Here the serverSocket has been instantiated with a designated port **8818**. The server socket will be initiated and clientAccept thread will be defined. Here, the socket accepts the serverSocket connection request by calling the accept () method on the serverSocket object.

In the server side DataInputStream, DataoutputStream class objects are initiated to retrieve input and send output to the clients, via method getInputStream and getOutputStream respectively. The readUTF() and writeUTF() methods are used inorder to read the encoding of UTF 8 into string and vice versa. Here a single Map is created inorder to store the total list of users i.e., **10** to the respective socket. (<**UserName**>, <**ClientSocket>**). The server side check the number of existing user count and notifies the new user if the server is full or not. The number of total clients connected till now and the message being sent to different users are being reported to the sever Message Board. A list of Total User and Total active user is maintained in the server side. All of these are done using a Thread called MessageRead () Thread initiated in the Server_chatApp.The server side differentiates between whether the message being sent is **Multicast** or a **Broadcast** message. In the ChatApp user have access to retrieve old history if they have any and send message to N number of users (Active users).

Each connected client is given a separate thread. Here the serve is locally hosted, so the IP used for the socket is of localhost. So first the client establishes the remote endpoint for the socket and then creates the TCP/IP socket. Then only it invokes receive method. The dataInputStream and dataOutputStream is used to read/send raw stream in the socket. The readUTF method of dataInput stream class reads from the stream in a representation of a UniCode character string encoded in modified UTF-8 format.

The User can provide the username and create a room between the active users in the chatApp and if the user is a old user it can fetch previous-stored chat history by clicking the history radio button in the JFrame. A text file will be created using the same username to save the new message and the same file will be used to retrieve the old history messages. When the send button will be pressed, then the identifier along with the message is send to the server using Unicode Transformation format to write a byte to the underlying stream as a 1byte value. The client can terminate and create a new chat room with different users.

Here the server should be run first i.e**. Server_chatApp.java** that will initiate the server and then Clien_login_chatApp.java should be executed. After the execution is complete and the username is added, the Client_chatApp.java is called, here thread for the client is called i.e., Client_chatApp.java. The constructor in the Client_chatApp.java. gets initialized.

**Steps to Run the chatApp.**

1. The project is build in eclipse as an IDE and here both the client and sever is build using Java.
2. The chatApp contain three files
    a. **Server_chatApp.java**
    b. **Client_Login_ChatApp.java**
    c. **Client_chatApp.java**
3. Before you run the server_chatApp.java file. One must update the code where the history file will be saved. The Location variable is called `databasePath` and it is present in **Client_chatApp.java** file. The location of the file will be used in three different place and should be leading to a directory. File with the name of Username will be create further when code in executed. It should be in line 56.

```
//Please update the path of your system where you want to save you history.
//Its recommended to be save in the same directory where you keep all files .

String databasePath ="C:\\Computer Science department\\CMPT-842\\Assignment1\\";
```

4. Start executing the code starting with the server (**Server_chatApp.java**). A serverView Jframe should popup as shown below. The Serveview contains the Active User list, All User List and a TextArea to display log and number of client counts.

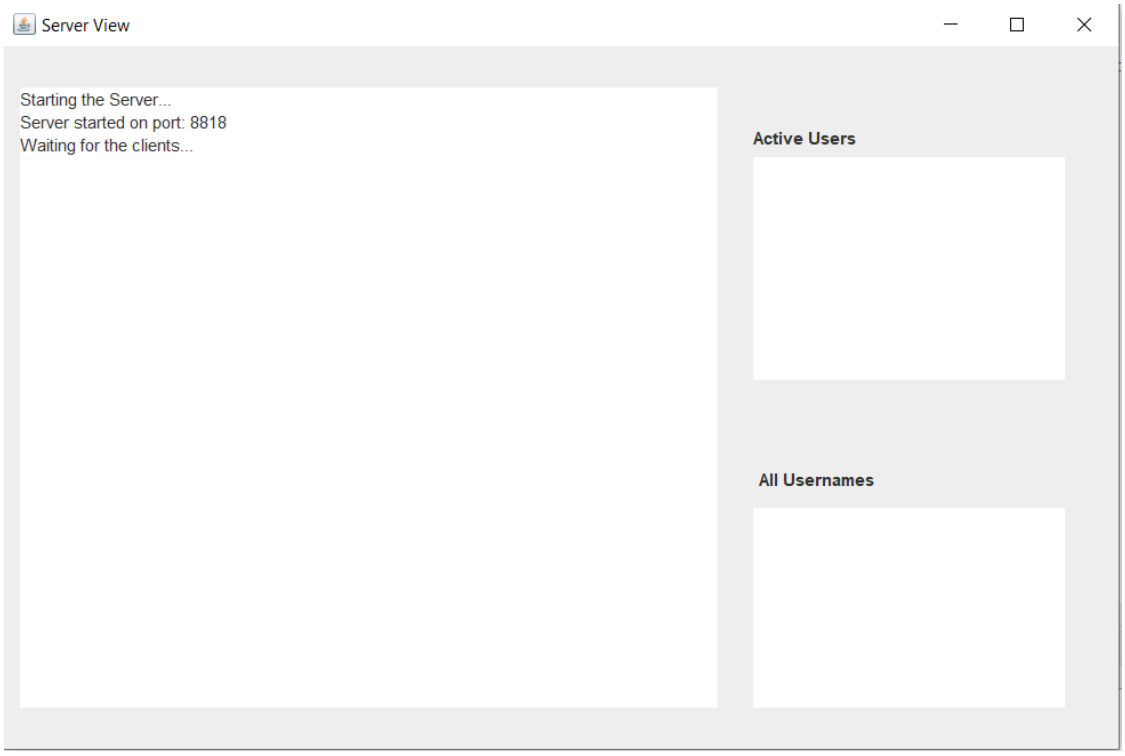
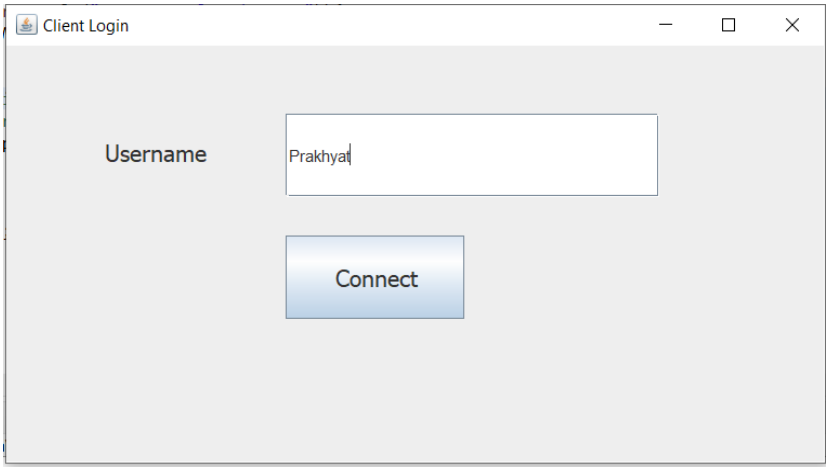



Fig :- Server View showing Active User and All Usernames list

The above shown list gets updated if a new user SignIn and is the user gets disconnected , All username list will still have their names.

5. Now Execute the Client_login_ChatApp.java inorder to signin as a client in the ChatApp system.




The above Jframe is used to enter the username and now to move forward into the chatApp connect will initiate the Client chat UI.

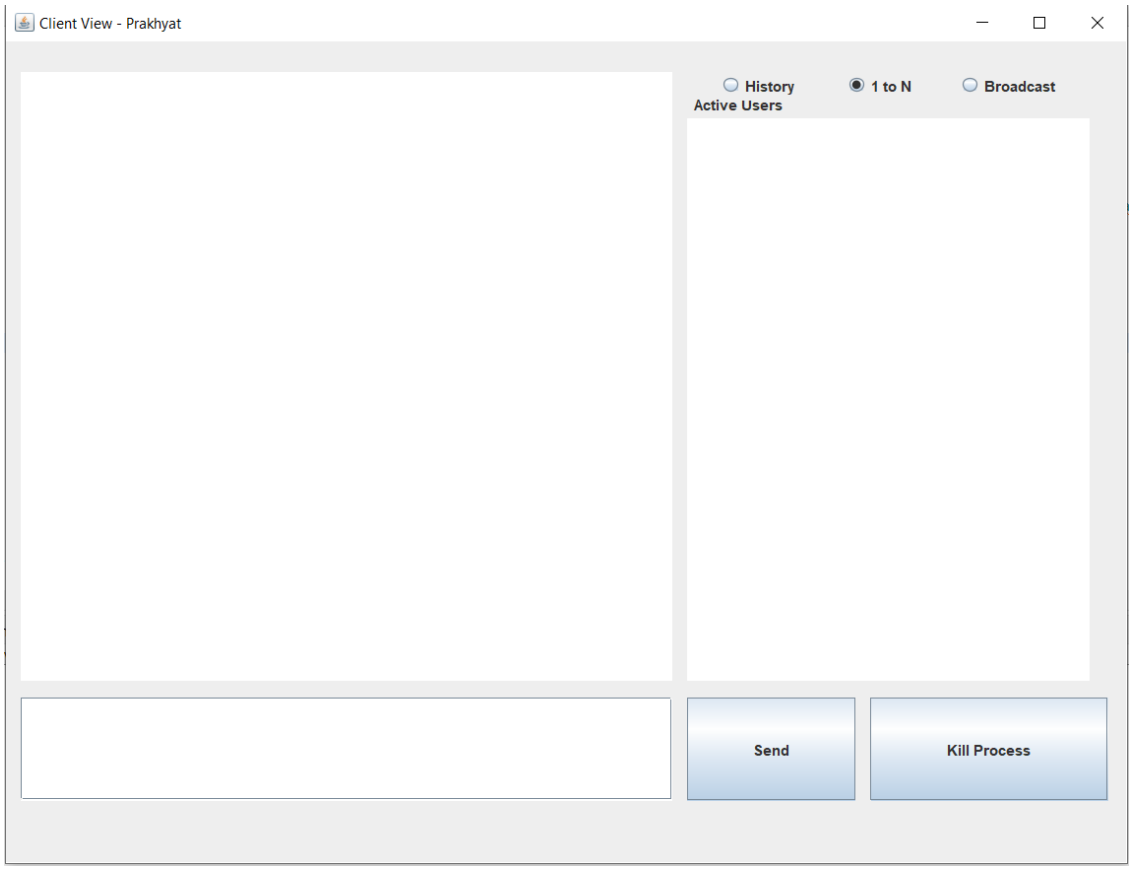
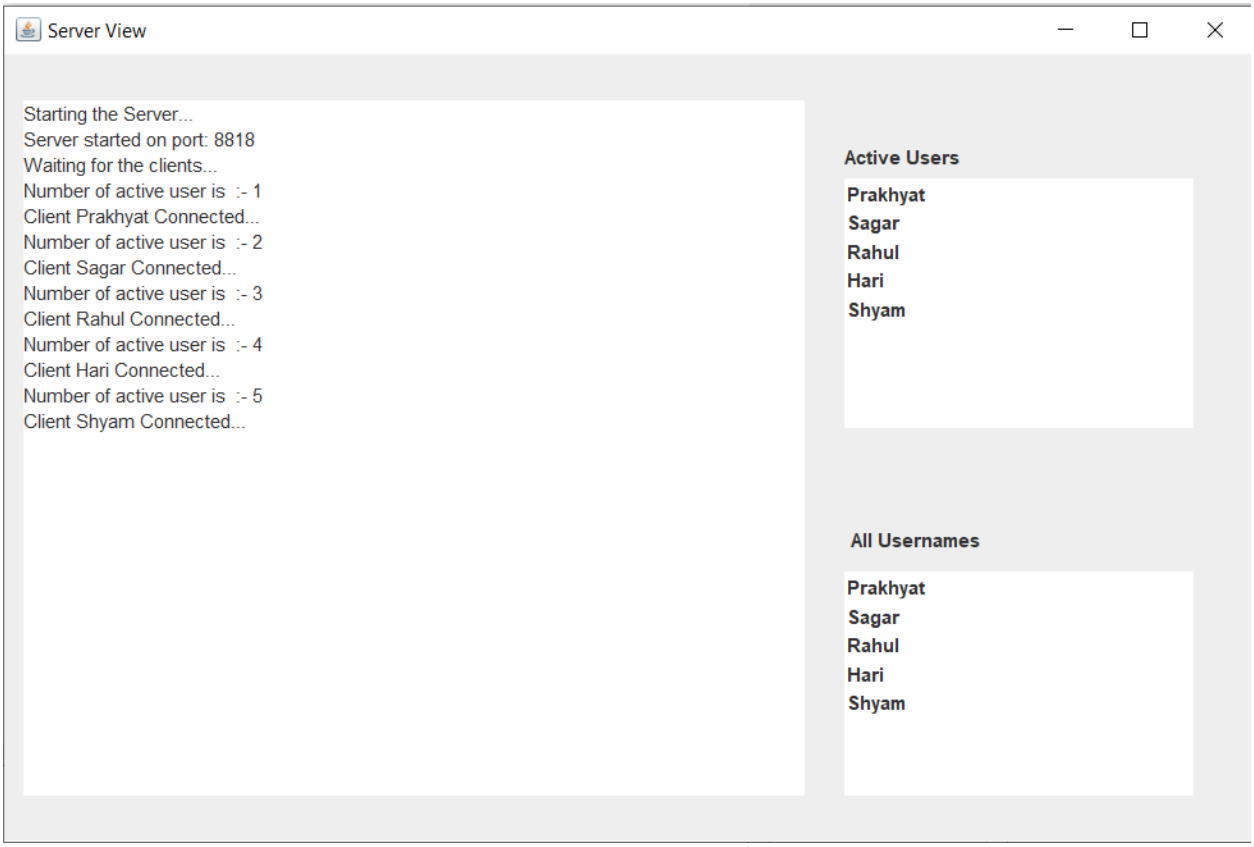


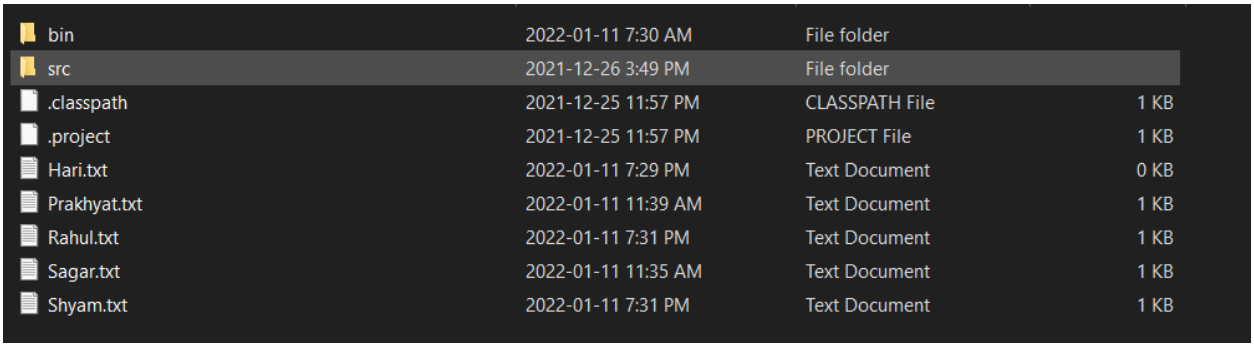

Fig:- Client View containing History, Multicast and Broadcast option.

Here the Client can Kill the Process and create a room with the Active users connected in the ChatApp. **10** number of User can only be connected. Here the Chat View displays the username.

To Test the system lets create other users called **Sagar**, **Rahul**, **Hari**, **Shyam** ,. So, in the server View it get updated.




Here for the demo part, 5 of the users are connected and All of them are active. Text files to save the history of the chat between the users will be create respectively.




Let's take an example.

Here **Shyam** is trying to send message to few selected users, and He can send message by select 1 to N option and select the Users.

All the users will receive the message and so does the server. The server will get the log of all the message being sent by all the users in their respective room.

If the user disconnects and reconnect, then the user can retrieve older chat history by clicking of the radio button Called **History**. The user will receive all the User old history and all the public messages send to the user.
The public message is the message send by selecting the broadcast radio button. It will disable the Active User list as Broadcast will send the message to all the connected users.

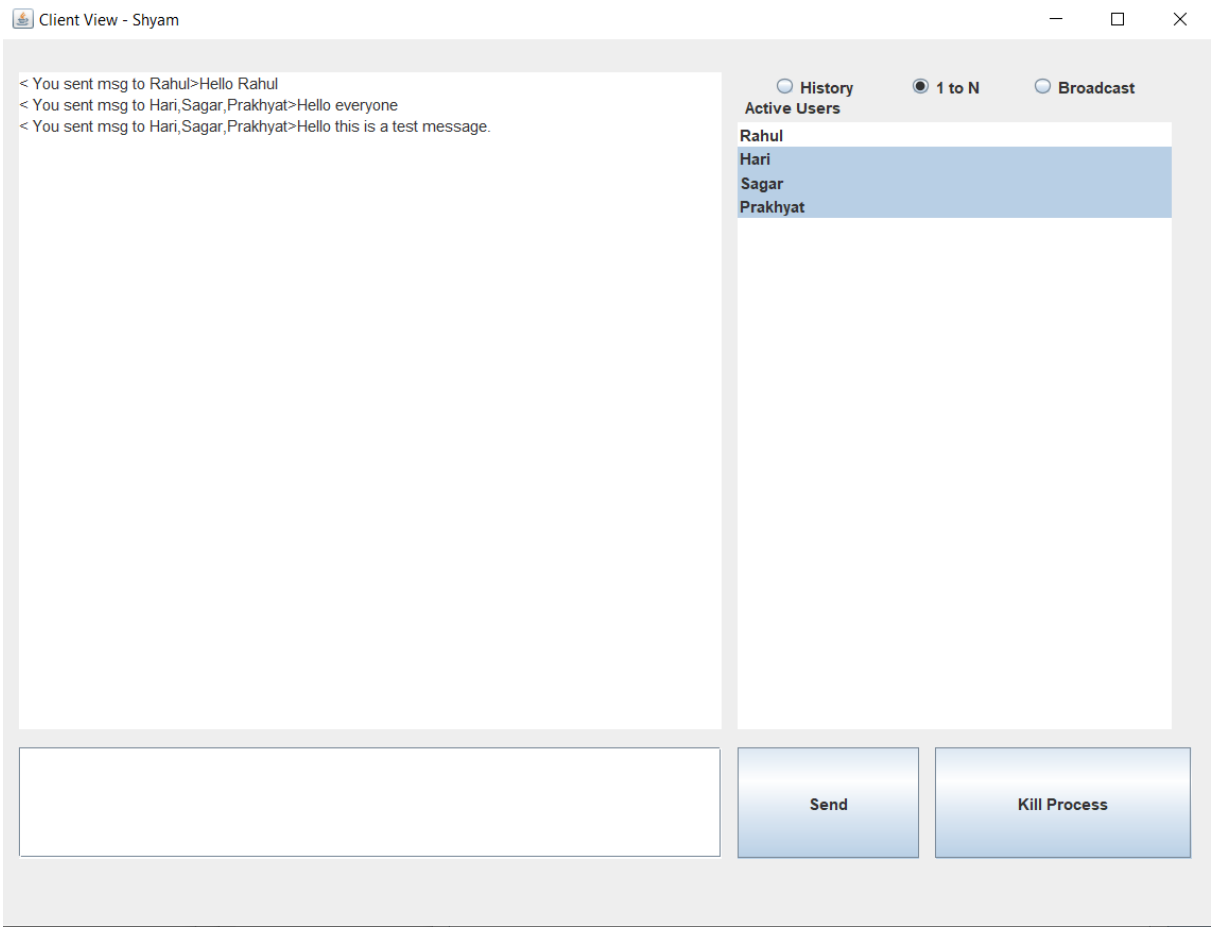



Fig: Sending message to selected Users.

To terminate or disconnect Kill process should be click and the server and other users gets update stating the user has been disconnected.

Here if the new user enters the same username and it matched with the active user list than User gets notified that username preexist in the chatApp.

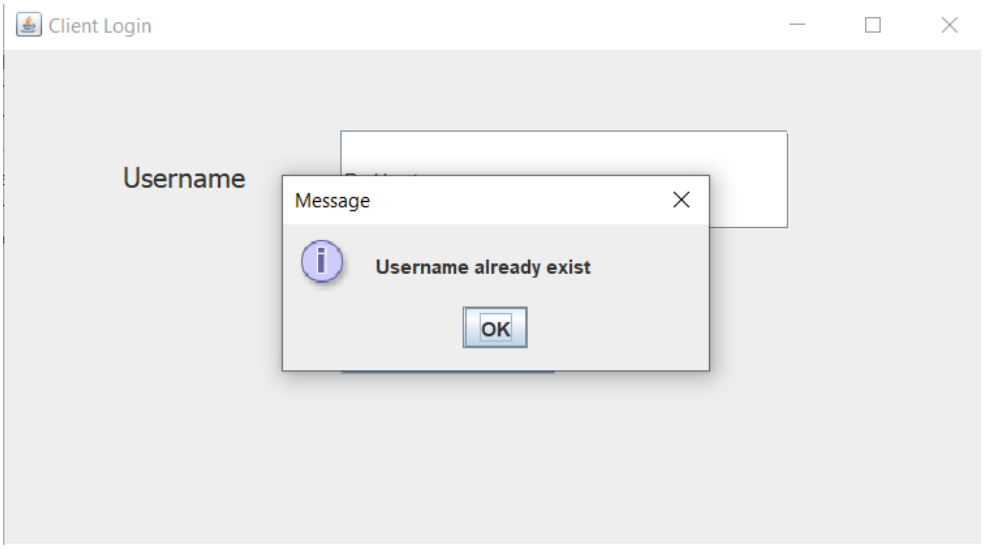
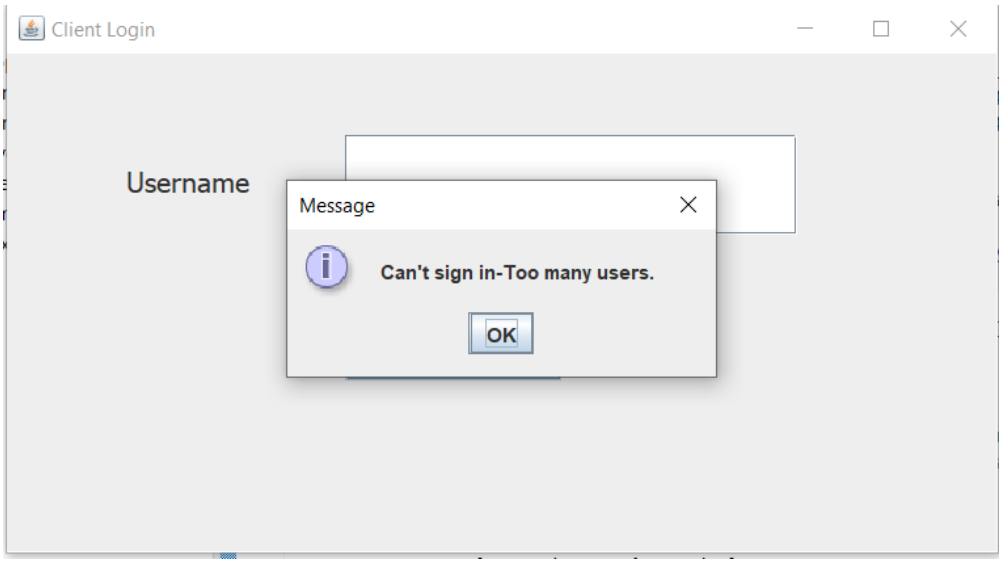



Fig: Username preexist.

If the number of users reaches the limit the user is notified in the similar manner.




Here the limit is of **10** users, and it can be changed using the variable `total_clients` in **server_chatApp.java**

To terminate the who chatApp. Client should be Killed so that the server and all other client will get notified. And then the server should be terminated.