Chat Application implementation in Java: Report 1

Azhar Rohiman (RHMMUH005), Naeem Levy (LVYNAE001),
Shaylin Padayachee (PDYSHA009)

March 23, 2018

Abstract

A Java implementation of a simple GUI chat application that allows users to send broadcast, group or private text messages to each other. Modeled on a client - server model , the application is built around a central server which forwards client messages to each other as requested. The features of the application are: One - to - one messaging (private messaging), broadcast chat (A broadcast chat group available to all users on the server similar to those used in website comment sections and online games) and group chat (a user is able to send the same message to a specified group of users - multicast).

---

1 Design:

1.1 User Interface:

When designing the GUI we opted to only give the client a user interface while the server is interfaced with via terminal. A GUI is easier to use and allows for more control of a users interaction and input with the interface, resulting in a lower chance of a user committing an error and easier to handle user input for error checking, and reduces the learning curve of the application.
The GUI interface was designed using a minimalist style philosophy, this aids in the application being more user friendly and making it easier for a user to intuitively understand how the application works.
The GUI is divided into five sections; in the top northern section the user has a input text field - **The message field** - where the message is typed and sent upon pressing the return key. In the east section of the frame a panel which lists - **Friend List** - all the friends a user wants to privately or multicast message. When a friends name is double clicked it automatically appears in **The message field** appended with a #, this is explained in the second report which discusses protocol. In the south section of the panel the GUI has a **Broadcast Field** which displays broadcast messages, to send a broadcast message the command `all#` must be typed before typing the message. The west section contains the **Add friend** when clicked will prompt the user with a JOptionpane to enter the friend name. The center section - **Chat Screen** - displays the private and multicast messages received and sent. The **Broadcast Field** displays the sent and received broadcast messages.

1.2 Server:

The server doesn't have a GUI and is interfaced with via terminal. It is multithreaded so for each new client that connects to the server a new client thread, a socket thread, is made to handle the input and output streams of that particular client. The the server only handles text data on the client input streams. When a client connects to the server a user name must be provided, this is enforced on the client application, before a client can continue using the server to forward and receive messages. To summarize: The server hosts a number of clients and forwards messages between them to and from each other.

1.3 Client:

The client uses the GUI described in subsection 1.1. A user[1] is notified by the server, in the **Broadcast Field**, every time a new user is added to the server this allows the user to add a new friend in the **Friend List** and ensures that the user can enter the friends name correctly.

2 Functionality:

One of the main challenges when implementing any type of functionality in an application that uses multiple threads to handle I/O streams is maintaining correct state - this means that the clients are forwarded the correct data, however, apart ensuring that state is correct and creating thread safe data. The added factor of bandwidth speed and having to work with extremely sensitive I/O streams were the main constraints , with the goal building safe and robust functionalities into the application, that informed our decisions on functionality.

2.1 Private Messages:

Each message has a header added to it which acts as a flag [2] that determines the type of message sent. The server will check the flag if it's a private message it will check for the recipient name and forward it, this is implemented using a synchronized loop block where each user name is a key–value pair hashmap if the the name of the recipient exist in the hashmap the message is forwarded.

2.2 Broadcast Chat:

Broadcast chat is when the message is sent to all users and displayed on the **Broadcast Section**. This functionality is very similar to private messaging in program logic, as in the same error checking and thread handling, except iterates over the hashmap the of clients and sends it to each user, of course it is on the client side where the message will be displayed is controlled.

[1] User and client have the same meaning in this context

[2] This is expanded on in the second report.

2.3 Multicast Chat:

Multicast chat allows the user to use the select the desired recipients in the **Friend List** a user would like to send the same message. As with the above functionalities it is just as simple in implementation simply using a synchronized block to iterate over users and send the message to the specified clients, this is handled by the protocol discussed in Report 2. This message is also displayed on the **Center Screen** which it shares with the private messages as it is essentially the same message type except with a different recipient cardinality.

3 Screenshots:

Screenshots of the Group members chatting on different computers:

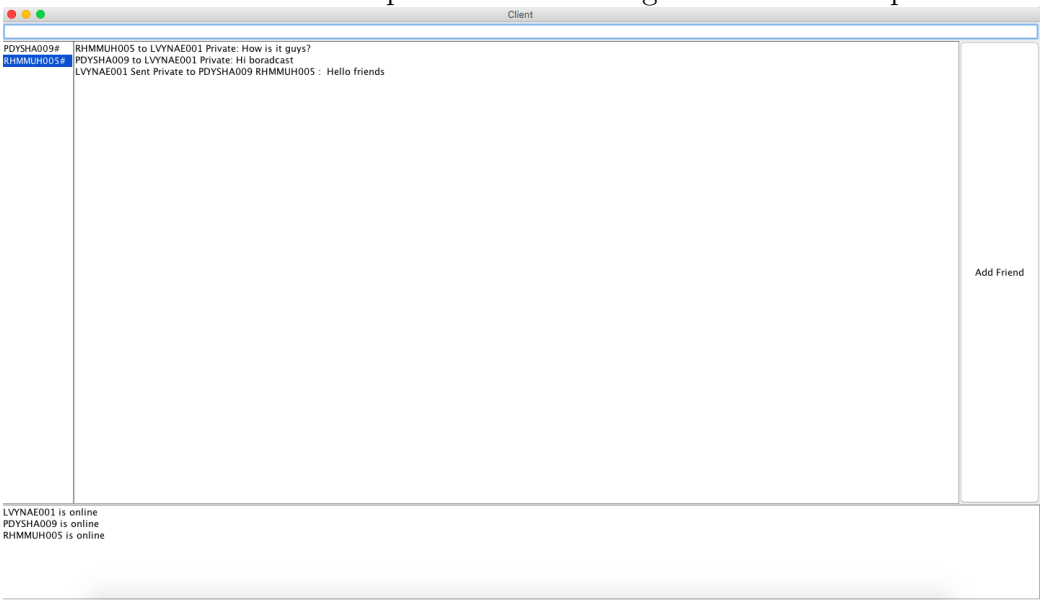

Figure 1: LVYNAE001 - screen shot on MacOSX.

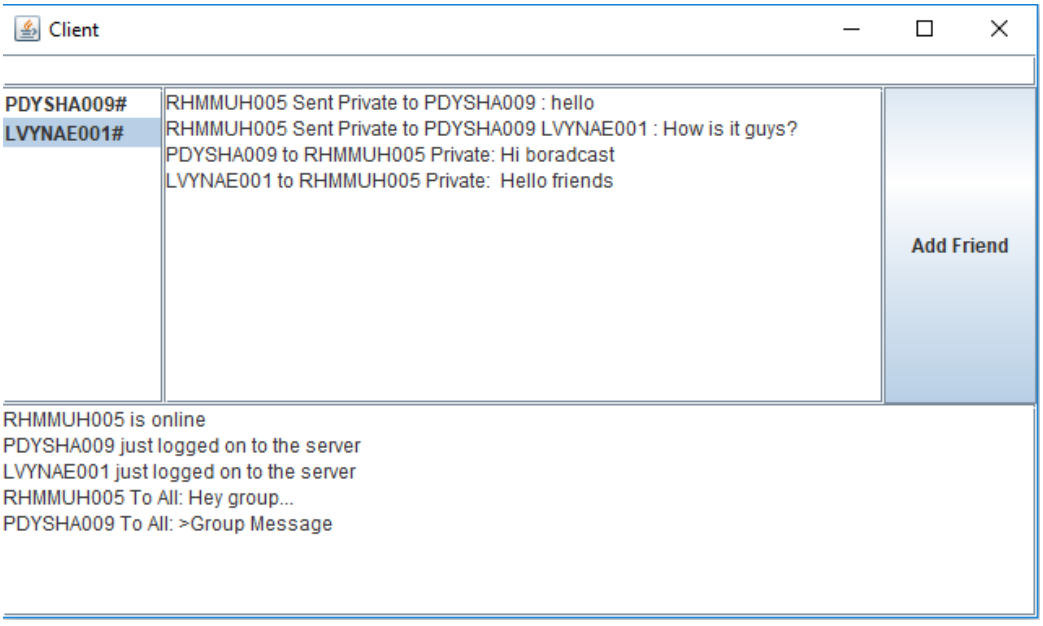

Figure 2: RHMMUH005 on Windows 10.

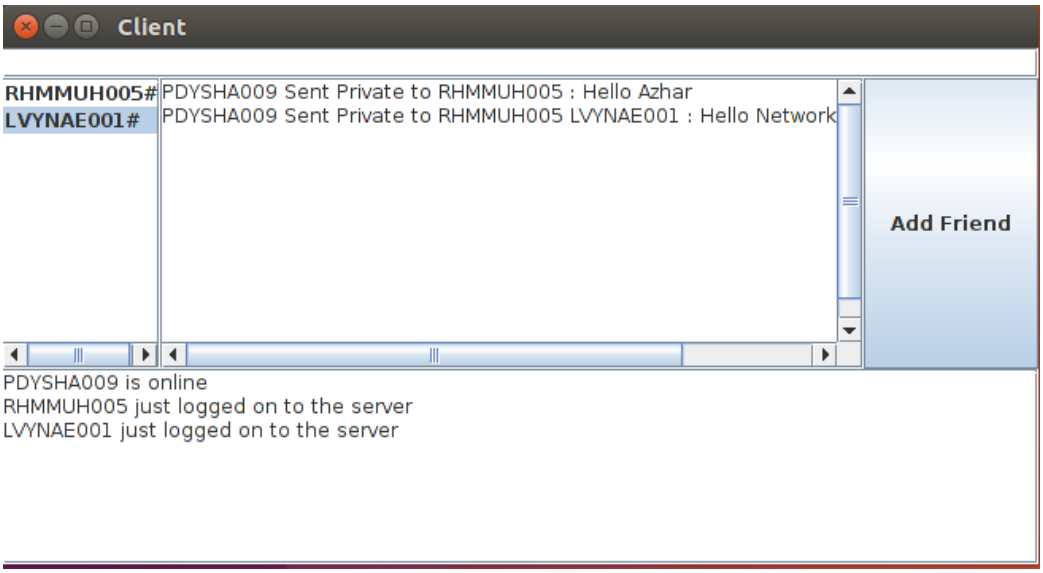

Figure 3: PDYSHA009 using Ubuntu

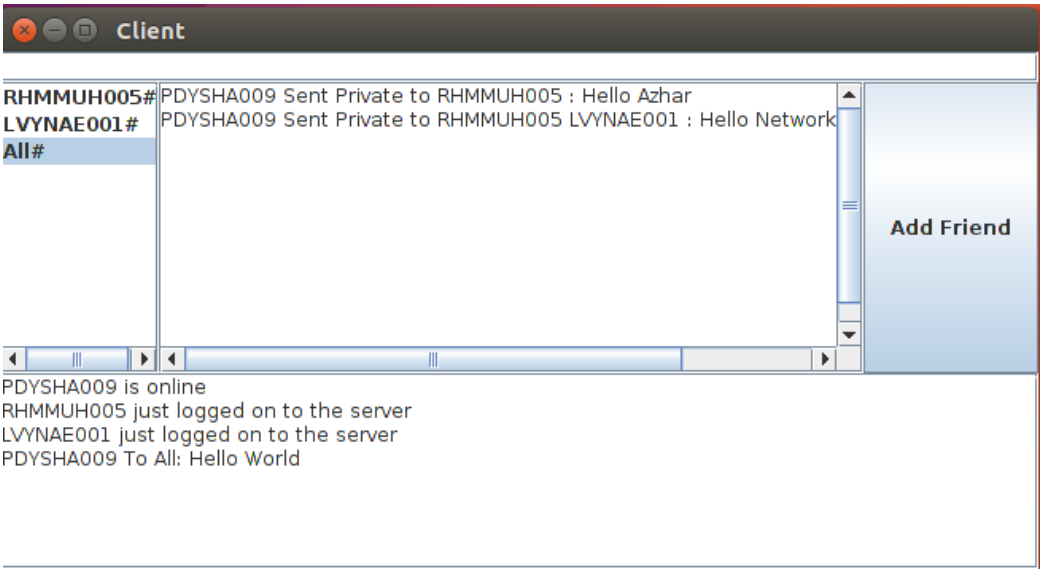

Figure 4: Example of saving the all command for broadcasting.