JAVAChat

By Kamil Adylov

Table Of Content

Project Description and Background

JavaChat is a simple console chat application fully implemented in java programming language. The application is launched from the command line, and the server and clients can run on different computers through the local network.

First thing to get the application working properly, the server needs to be started and running. In the command line, a user of the app needs to enter ***Javac Server.java*** to compile the server file. Then the user needs to enter java **Server** to start the server. After that a message "Waiting for client connection..." will be displayed in the command line indicating that the server has started and waiting for client connection.

Once the server has been started, clients from other computers in the same network can start to connect to the server by entering **Javac ClientApp.java** file and then **java ClientApp** in their command line in order to run the client application. As long as the client app starts executing, the app will prompt user to enter username. Then the app will send the created username to the server to check whether the username is in use by other user in the chat or not. If the server determines that someone with this username is already connected to the chat, it will respond back to the client app that the someone is already using the username.

The client app will inform the user that the username is in use and prompt him to create another username. The client app will keep prompting the user to enter new username until it is approved in the server side.

Once the user's username is validated, the user will see a welcome message and an info of how to use the chat on the screen. In this stage, the server will notify him of newly connected users and new messages. User can begin chatting with other users by entering text in the console.

If the user wants to send a private message to another user or group of users in the chat, he simply types the user names of the users. Each username needs to have a symbol “@” in the beginning of the username to indicate the server that the message is private. The server will deliver the message to the recipients. The private messages will be displayed in green color in the recipient screen to distinct the public and private message. Note that the other users in the chat will not see the private messages in their screen, but only those users who is intended to receive the message.

If the users decides to see help info of how to use the app or what commands to use, he needs to enter command #help. The server will send a list of the available commands to the client app which will display it to the user’s screen.

System Design Description

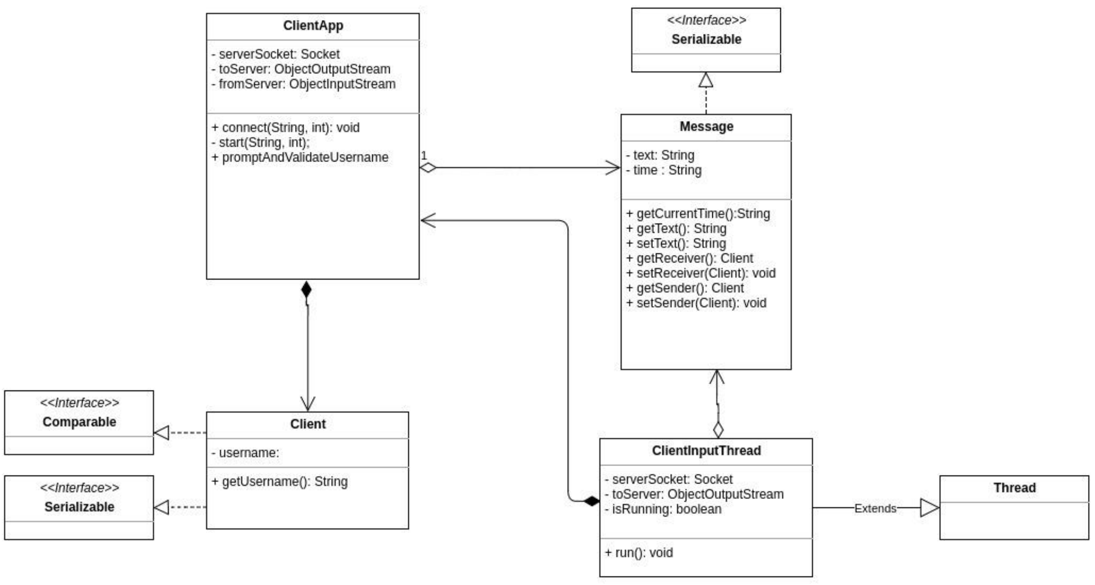

Figure 1: Client part

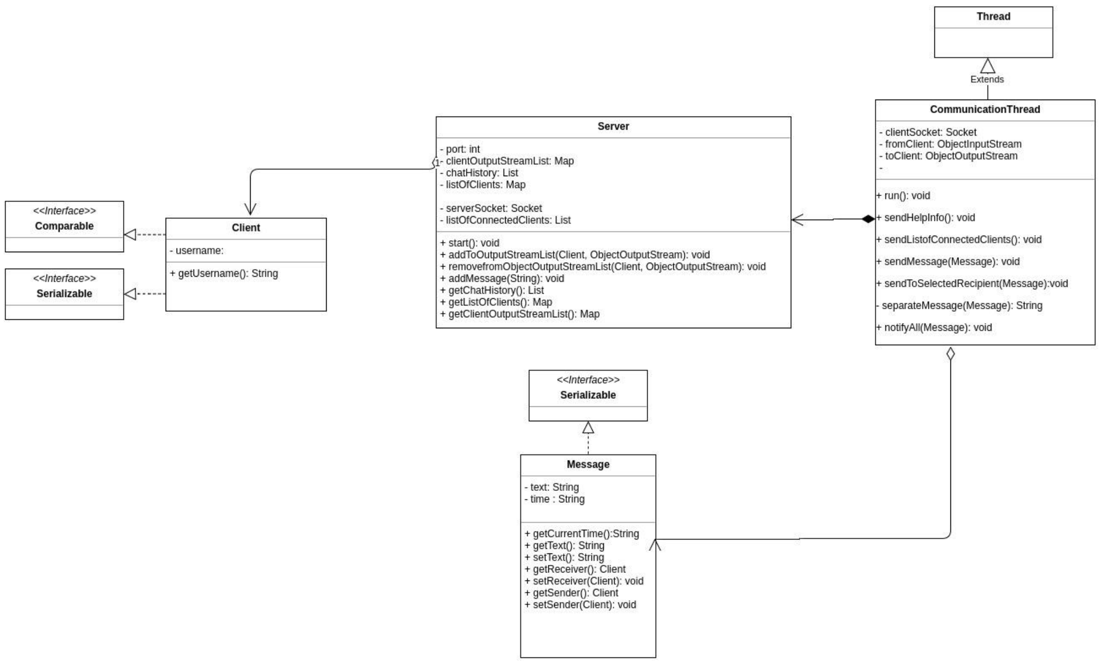
Thread
Extends
CommunicationThread
- clientSocket: Socket
- fromClient: ObjectInputStream
- toClient: ObjectOutputStream
-
+ run(): void
+ sendHelpInfo(): void
+ sendListofConnectedClients(): void
+ sendMessage(Message): void
+ sendToSelectedRecipient(Message):void
- separateMessage(Message): String
+ notifyAll(Message): void
Server
- port: int
- clientOutputStreamList: Map
- chatHistory: List
- listOfClients: Map
- serverSocket: Socket
- listOfConnectedClients: List
+ start(): void
+ addToOutputStreamList(Client, ObjectOutputStream): void
+ removefromObjectOutputStreamList(Client, ObjectOutputStream): void
+ addMessage(String): void
+ getChatHistory(): List
+ getListOfClients(): Map
+ getClientOutputStreamList(): Map
1
<<Interface>>
Comparable
<<Interface>>
Serializable
Client
- username:
+ getUsername(): String
<<Interface>>
Serializable
Message
- text: String
- time : String
+ getCurrentTime():String
+ getText(): String
+ setText(): String
+ getReceiver(): Client
+ setReceiver(Client): void
+ getSender(): Client
+ setSender(Client): void

Figure 2: Server Side

Code Analysis

```
public CommunicationThread(String username, Socket socket, Server server) {
     super(username);
     this.clientSocket = socket;

     this.mainServer = server;

     try {
         toClient = new ObjectOutputStream(socket.getOutputStream());
         fromClient = new ObjectInputStream(socket.getInputStream());

     } catch (IOException e) {
         System.out.println("Error in serverIO constructor");
         e.printStackTrace();
     }

  }

  @Override
  public void run() {
     Message newMessage = null;

     // validate username
     try {

         System.out.println("validating username");
         boolean created = false;
         while (!created) {


             // verify client first
             while ((client = (Client) fromClient.readObject()) != null) {

                 if (mainServer.addClient(client) == true) {
                     sendMessage(new Message("ok"));
                     System.out.println(client + " was added to the list");
                     created = true;
                     break;
                 } else {
                     sendMessage(new Message("no"));
```

```java
                    }

                }
            }

            if (client == null)
                System.out.println("Client is null");
            else if (toClient == null)
                System.out.println("toClient is null");

            mainServer.addToOutputStreamList(client, toClient);

            newMessage = new Message(new Client("server"), client + " has connected...");


//          sendMessage(newMessage);
            notifyAll(newMessage);

            sendMessage(new Message("\n\nWelcome to JavaChat"));
            sendHelpInfo();

            // for debugging purpose
            System.out.println("Reading user newMessage...");

            // wait for client response
            while ((newMessage = (Message) fromClient.readObject()) != null) {

                String txt = newMessage.getText();

                if (txt.equalsIgnoreCase("#q")) {
                    sendMessage(new Message(client, null, "#q"));

                    notifyAll(new Message(null, new Client("server"), client + " disconnected..."));
                    mainServer.removeClient(client);
                    mainServer.removefromObjectOutputStreamList(client, toClient);

                    break;
                } else if (txt.equalsIgnoreCase("#help")) {

                    sendHelpInfo();

                } else if (txt.equalsIgnoreCase("#list")) {
                    sendListofConnectedClients();

                } else if (txt.startsWith("@")) {

                    sendToSelectedRecipient(newMessage);


                } else {

                    System.out.println(newMessage);
                    notifyAll(newMessage);
                }

            }

            fromClient.close();
            toClient.close();
            clientSocket.close();
            System.out.println(client + "  disconnected...");


        } catch (IOException e) {
            System.out.println("Failed to read " + client + "'s message");
            System.out.println(e.getMessage());
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }


    }
```

Tests

Since the application is simple and small, there was not any frameworks used but just printing out the server logs or errors to the server console.

Results

The project is fully functional and all detected bugs have been fixed.

First thing, the server needs to be running. Run the Server.java either in console or in Idea. Once the server is running, it will start to listening for client connection; it will print the message “waiting for connection” out to the screen. Then you can run one or multiple ClientApp.java in the multiple console windows or in Idea. It will prompt you to create an username before joining to the chat.

After that, you will see a welcome screen and list of available command to use (e.g. #list, #help, #q). You can type right in the console and press enter to send a message. Other users will see your message, and they can respond to you.

You can also send a private message and the recipient of the message will see it on his screen in green color. To send a private message, begin your message with symbol “@” and username of the user and then text. For example, if I want to send a private message to user whose username is @bella, I would type @bella hello world.

Lessons Learned

Learned how java networking and multithreading work in Java. I believe they are the one of the main strength of java.

The most difficult aspect I found was input/out between server and client through the socket. Also, debugging my code and dealing with thrown exceptions when multiple threads running.

In this section give a high-level summary of what you learned. Including... what were the aspects you found more difficult? what are the technologies that you actually learn? how do you consider you can use this technologies into a company?, etc.