John Brusaw

CS300

ChatApp project final deliverables

GitHub repository: <https://github.com/Jbrusaw/CS300_ChatApp>

For this project, I have 2 java source files: server.java and client.java.

Server.java is a Server class with several sub-classes: User, ServerWindow, and ClientThread. The Server class establishes a few variables that all other classes need access to. There’s a HashMap<String,String> (named *accounts*) that imports account usernames and passwords and stores them for easy access to logging in. The User subclass stores account usernames and DataOutputStreams that an ArrayList<User> uses to keep track of all online users, as well as their output streams for easy access. There’s an int chatID that used to give every group chat their own unique ID. This is useful for both clients and the server to store group chat information. The server also utilizes a HashMap<Integer,User[]> (named *chats*) to store group chats.

When server.java is started, *accounts* imports data from accounts.txt and builds a list of valid usernames and passwords. The server then launches a   
ServerWindow GUI that contains a list of active online users and a way to exit the program. Then the server opens up a serversocket and starts an infinite loop that simply listens for socket connections. When a socket is connect, it starts a new thread (ClientThread class) for communication and goes back to listening for socket connections.

ClientThread then gets user login info and adds it to the useronline list. After some basic information exchange, ClientThread then just starts an infinite loop listening to input from the user. The User class has an output stream for each user as well as a SendMsg() method so that the ClientThread can simply pass along the message to each User that needs it and let’s the User send the message.

Client.java is what the users use to send messages. It starts off by opening a socket, then launching a LoginWindow class that send login info to the server. When the user has successfully logged in, LoginWindow then closes itself and opens a OULWindow.

OULWindow is a class that displays a list of all online users (provided by the server) and displays notifications when someone logs on or off. It also starts an Input() thread that listens to the server and parses information. Once the user selects who they want to chat with, OULWindow then launches a ChatWindow for those select users. OULWindows keeps track of all the available ChatWindows through a HashMap<Integer,ChatWindow> function that maps the chatID to the ChatWindow being used.  
 Lastly, ChatWindow is simply a GUI that has a few text fields that allows the user to send messages and displays messages received by users.

By far the trickiest thing for me on this project was managing the input/output streams to get the correct data back and forth. Both the server and the client have threads dedicated to simply listening to the other, so all input/output has to be managed with that in mind. For example, a bug that took me a while to figure out was I had an “in.readUTF()” call in a function that was outside the Input() class, which also has an “in.readUTF()” call still active. When the client sent the info, it was going to the wrong “in.readUTF()” which caused incorrect behavior. To fix it I had to re-work the function so that it was only called within Input() so that they both couldn’t be active simulteaneously. Another problem I ran into was I wasn’t sure how the send messages from the server side to multiple users. Each ClientThread only had one output stream and could only directly communicate with the client it was connected to, and I really wasn’t sure how to pass information between threads. Eventually I came up with the idea of a static HashMap that stored each User’s info that was online. Now all each ClientThread has to do is lookup the User’s info via searching for their username, then send the message along the associated output stream