Design Manual

We have created a messenger app that allows several people to connect to one common server and communicate with one another instantaneously. As long as each user connects to the same server, they should all be able to interact with one another. Each user also is able to create a distinguishable username so that one can tell who is sending each message as they type. The chat client itself distinguishes between the user's messages and the incoming messages by making the current user blue on their screen and aligned to the right side, while incoming messages are orange and aligned on the left. The program works by first opening a Server object. This Server object runs on a specified port, in our case on port 1500, and begins actively searching for new user connections. When a new server socket object connects to the server on the same port, a new User object is created, who's name is determined by the current user. After it accepts a user, it resumes actively searching for new connections. The User object is created on a thread and contains an input stream and an output stream. The output stream is created through a PrintWriter object and is used to send messages through the socket to other users connected to the server. The input stream is created using a BufferedReader object and reads in messages received by the server and registers them as an incoming message. That is the backend of our application.

On the front end, we have a Client class, a MainClient class, and a ChatController class. The MainClient class is where the user is able to run the application from. When it is run, it first displays the pop-up window where the user can select their username. Once a valid name is entered, ChatController is initialized and a new Client object is created using the entered

username and a set IP address. The Client object creates a new socket that the server can accept, as long as the IP and port number are correct. The Client object is what the ChatController uses to send and receive messages and display them on the screen. When the Client object is created, it itself creates a BufferedReader and PrintWriter object to use as its input and output streams respectively. The streams work similarly to the User object's streams, but the BufferedReader reads in from the ChatController's text field. The ChatController was created to manage the gui created by Scene Builder. After creating a new Client object, it takes the messages received from the BufferedReader and displays them on the left side of the screen in an orange text, while displaying the sent messages entered in the text field on the right side in a blue text. The app uses encryption to ensure a secure sending and receiving of messages. The encryption works through two functions, sendMessage and receiveMessage. When the message is sent, it encrypts it using a Caesar cipher, and when the message is received, it decrypts it. This ensures that all messages sent over the server are secure and encrypted.

We had several user stories whose requirements we attempted to fulfill, ranging from an ordinary college student all the way to an illegal narcotics dealer. Starting with the college student, Aaron Dennisgrew up as an introverted, intelligent person. Now, as a college student, he doesn't feel comfortable going out to parties or meeting people in real life. So, Aaron wants a way to be engaged in the school community without leaving the comfort of his own dorm room. We believe that this was partially accomplished. We provided Aaron with a messenger app that allows multiple people to connect at once so he can be in a group with multiple people and communicate with them all at once. The one thing that we didn't provide for Aaron was a way to meet strangers in our application as we pivoted from a social media-esque app to a secure messenger app. The second user story we had was a social media influencer named Mathilde

Christensen. Mathilde wanted an application that allows her to communicate with several fans at once so that she may feel more connected to her fanbase. We provided this for her by giving her an application that allows for group messaging where each user has a distinct username and one can distinguish between who is texting them which is especially important in a large group chat. Our application also helps Mathilde and her fans because one does not need to pay for a data plan to access the messages, all one would need is an internet connection. A third user story that we had came from a spy named Krystian Haukedal. Krystian works with confidential material and needed a way to securely communicate with his colleagues. Our app incorporates encryption using a Caesar cipher and ensures security when sending and receiving messages through the server. This works for Krystian's needs as any messages he sends using our application will be encrypted and safe from prying eyes. Our final user story comes from an illegal narcotics dealer named Kaya Çapanoğlu. Kaya, like Krystian, wants a secure and encrypted messaging app, but for his own reasons. Like before, our encryption ensures that Kaya's messages cannot be intercepted by other people and thus his requirements are met.

CRC Cards

Server	
• Server object listens for incoming connections and hosts all clients in one place to communicate with one another through server sockets	• User

User	
• Thread class that connects to server and sends/receives messages	• Server

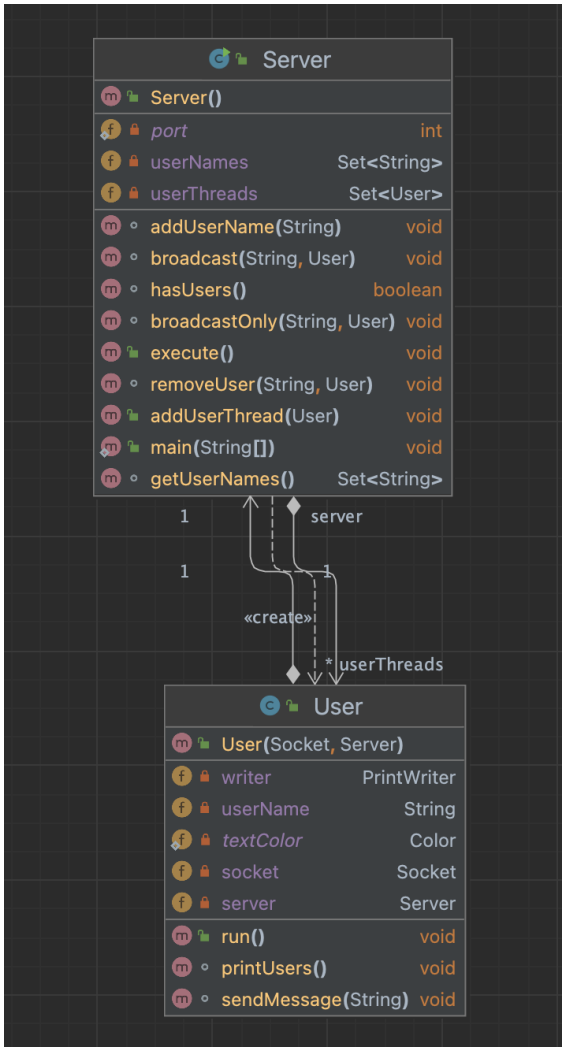
MainClient	
• The class that runs the application, initializes a popup window to create a username and then launches the ChatController class in order to run the applicationn	• ChatController • Client

Client	
• Sets up a server socket on a predetermined IP and port and sends/receives messages using I/O streams through sockets.	• MainClient • ChatController

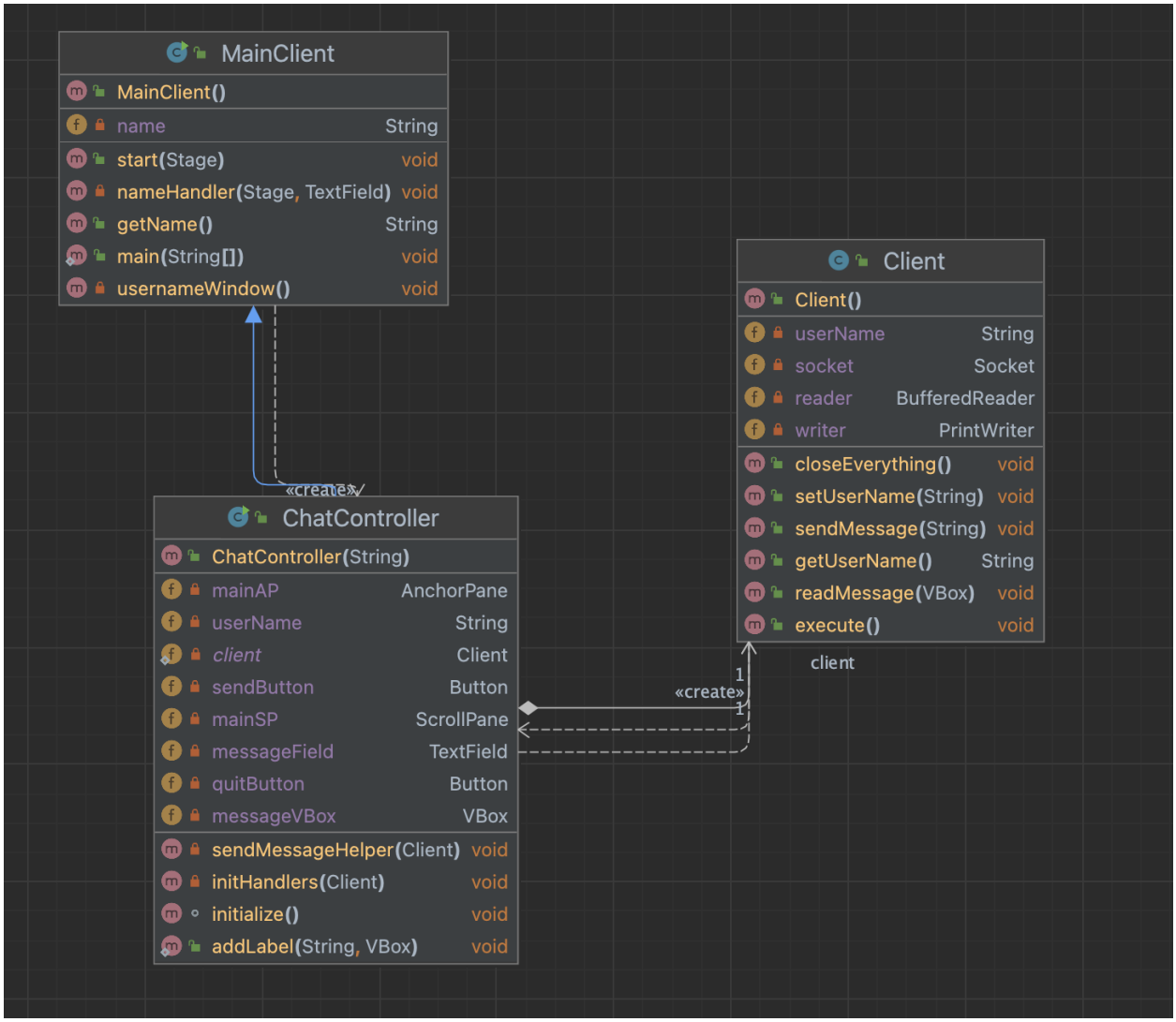
ChatController	
• Runs the chat application • Reads in messages to send from the text field and displays them on the right side of the screen while also waiting for incoming messages to display on the left side of the screen. • Creates client object when initialized	• Client • MainClient

https://echeung.me/crcmaker/

IntelliJ UML Diagram



Above you can see the UML diagram for the backend of our product. It includes the Server class and the User class. The User class creates an instance of the server class when initialized so that it has its own server socket to transmit information with. As noted above, the User class is a thread that can send messages using sendMessage() and can display the current active users using printUsers(). The server class collects many user threads and can broadcast messages from the user through the socket.

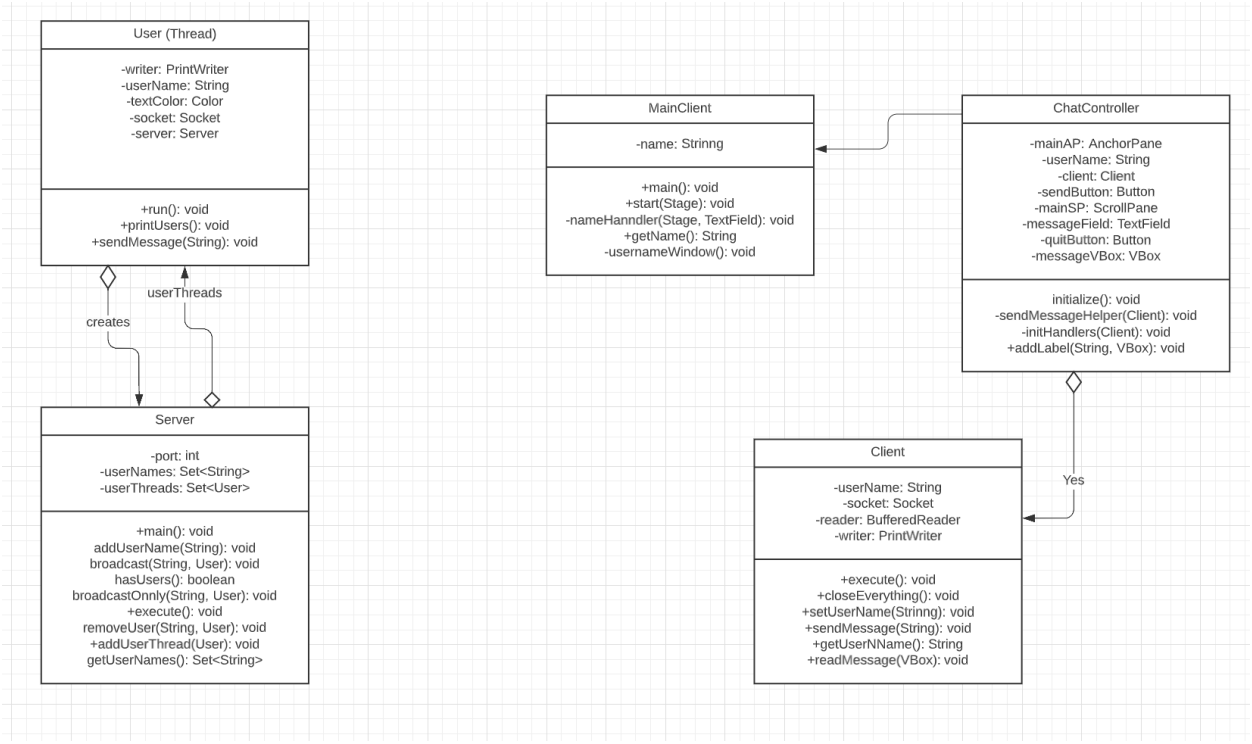


Here is the UML diagram for the frontend of our program. The MainClient class runs the entire program and launches the ChatController class. The ChatController class creates a

Client object which sends and reads messages received through the server socket. These messages are then displayed on the interface using the ChatController.

Let it be noted that the ChatController, MainClient, and Client class share no dependency with the User and Server classes and are only connected through the server socket that allows for data to flow between them all.

UML Diagram (Lucid Charts)



www.LucidCharts.com