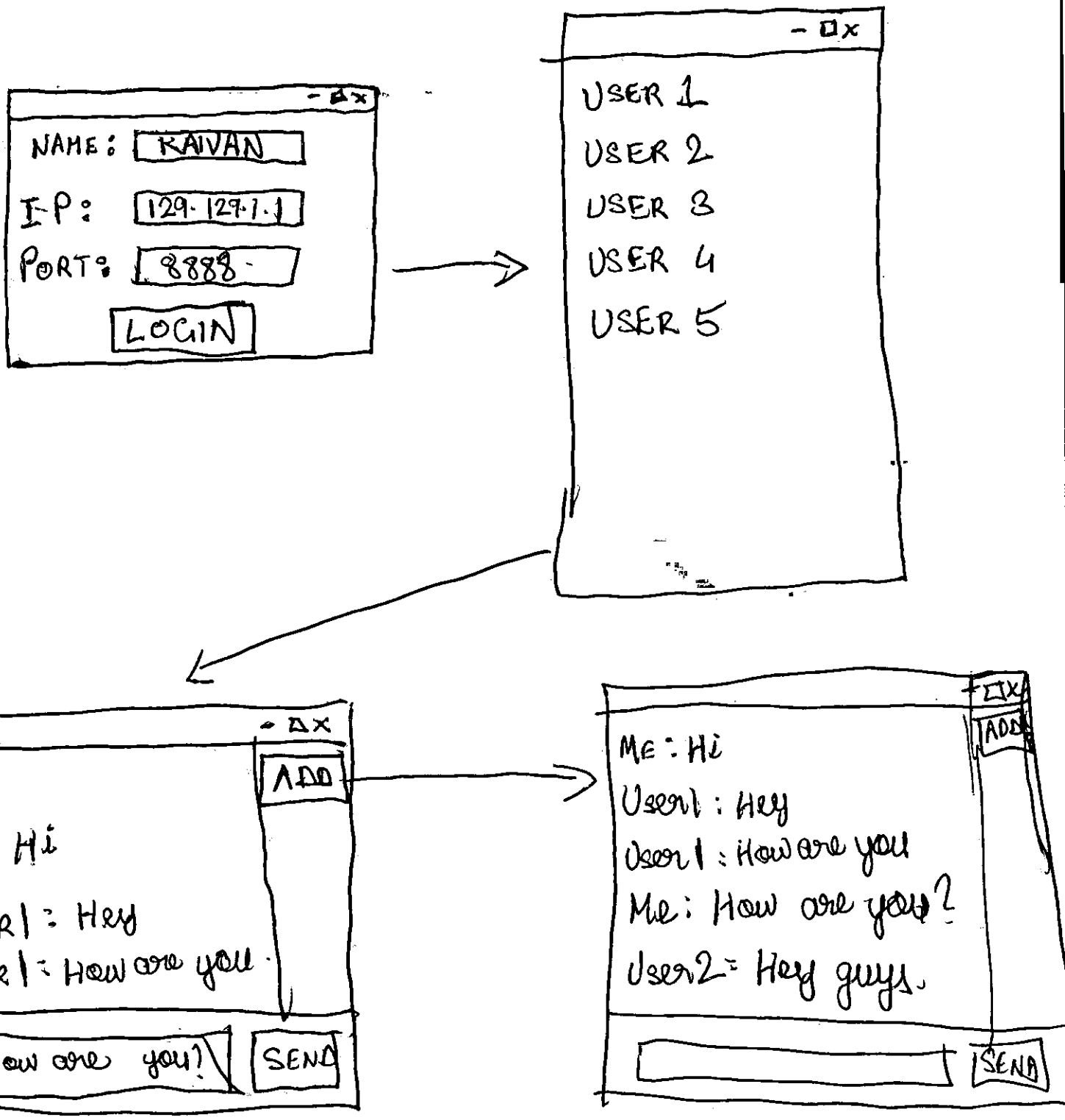
CONCURRENCY STRATEGY:

Our concurrency strategy is as follows:

1. The server is run on a single thread called the Server thread. All the clients first send a request to this main server thread.
2. Upon receiving a new client request the Server thread spawns a separate ClientHandler (Runnable) thread for each new client. Therefore, on the server side there is one ClientHandler thread running for each client.
3. When the client sends a new chat request a new ChatHandler (Runable) thread is spawned on the server side to handle the Chat between the two clients.
4. The shared objects between the Server and ClientHandler are the list of all online clients and the list of all existing chats on the server. Upon receiving a message from the client's input stream the appropriate ClientHandler thread interprets the message while synchronizing the global list of online clients and the list of all existing chats so that only one user can be modifying/reading from those lists at once. This guarantees that we don't run into any race conditions with different clients reading different data from the server.
5. Each ChatHandler (Runnable) thread has a list of all the messages, a list of connected users, and a Chat ID associated with it. All the instances of these variables are confined to the ChatHandler itself except the Chat ID. We ensure that the Chat ID is immutable and that all threads that use it simply read from it and never modify it.
6. A ChatHandler also has various synchronized methods with the lock on the instance variables to ensure that when a user leaves or enters the chat or a new message is received that all the internal data is modified by only one client at a time.
7. We avoid deadlock using a specific lock ordering for the instance variables.
8. In terms of the GUI, our controller will use SwingUtilities and spawn a new Thread to invoke later whenever it wants to make a change to the GUI so that there is no lag from the user's perspective.

TESTING STRATEGY:

1. The design of our serve is going to be as functional as possible so as to be able to test the server using JUnit tests.
2. We shall write automated JUnit tests which shall send a message to the server and read a response to check whether it's the correct response or not.
3. On the client side we shall test to see the correct message is sent to the server by invoking methods like which do so.
4. The testing of the GUI will be done manually and extensively to check all possible scenario's for race conditions and deadlocks and also check for lags.


NAME: KAIVAN
IP: 129.129.1.1
PORT: 8888
LOGIN
USER 1
USER 2
USER 3
USER 4
USER 5
ADD
ME: Hi
USER1: Hey
USER1: How are you
How are you?
SEND
ME: Hi
User1: Hey
User1: How are you
Me: How are you?
User2: Hey guys.
SEND