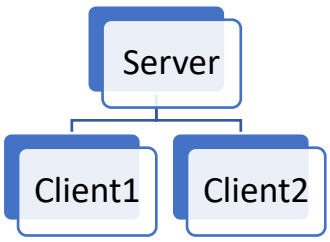

Functionality:

When client machines connect with the Server, they must specify their username. After that the client(client1) requests to connect with any other client (client2). Client1 will request the server for setting up a connection with client2. Then

Case1: Client2 is not connected with the server

The Server will inform client1 that client2 is not online

Case2: Client2 is connected with some other client

The server will inform client1 that client2 is busy with some other client

Case3: Client2 is available for connection

In Case3, Client2 will receive a message specifying “Client1 wants to connect with you” and ask for response (Y/N)

Case31: Client2 responds “N”

Then server will inform Client1 that Client2 has refused the connection

Case32: Client2 responds “Y”

Then the server will establish a connection between Client1 and Client2. And after that Client1 and Client2 will be able to communicate with each other

Case4: if in between the communication of Client1 and Client2 one of the systems crash/disconnect with the server

Then the other device will be informed, and the connection will break. After that the other client will be able to connect with remaining client devices

Case5: if client1 wants to leave the conversation then it will send “exit” to the server

Then the server will break the connection between client1 and client2. And client2 will be informed that client1 has left the chat

Implementation Detail:

Client side:

1. As the program starts, we will take input of the username
2. Then we will execute the client-side program in two thread
3. Thread1: for receiving messages from the client side
4. Thread2 (main thread): for taking input from the client

Server side:

1. As the program starts the Server starts listening for incoming client requests (after setting up the socket and binding the ip address, port address)
2. Afterwards when the server accepts any client request it assigns a new thread that handles requests from that client
3. The first message from the client will be the username
4. If the username is not unique the program will continue asking for unique usernames
5. After receiving a unique username, the server will maintain a dictionary of the client details

```
Clients = {
        Username: {
        socket:
        engaged:
        connected_with:
        }
}
```

6. Then inside a while loop the thread will start managing the client requests
7. Each client requests are handled as a command
   7.1. Command== ”connect”
   (client-> current client who sent the request)
   Then the client will be prompted for the username the client wants to connect with
   other_user = prompt (“”)
   7.1.1. other_user == client username
   client will be messaged “You can’t connect with yourself”
   7.1.2. client already connected with someone
   Client will be messaged to exit that connection first
   7.1.3. other_user is in a connection
   Client will be messaged that “other_user is busy in a conversation”
   7.1.4. other_user is available for connection
   other_user will be prompted “username is trying to connect with you (y/n)”
   other_user responds
   other_user is response is mapped with the client connection request using another class variable.
   Connection_requests
   {
   other_user: username
   }

7.1.4.1. N

User will be informed “other_user declined connection request”

7.1.4.2. Y

User will be informed “You are now connected with other_user”

Other_user will be informed “You are now connected with username”

7.1 command == “exit”

If the current_user is in a connection with some other user, that connection will be broken by modifying the clients(dictionary)