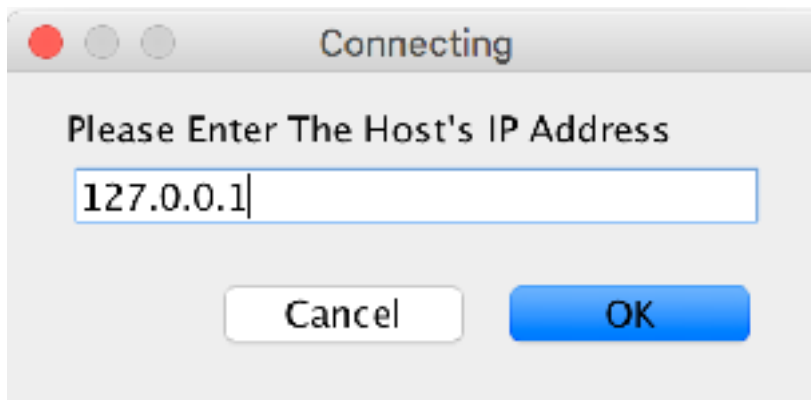


Networks Project: Chat

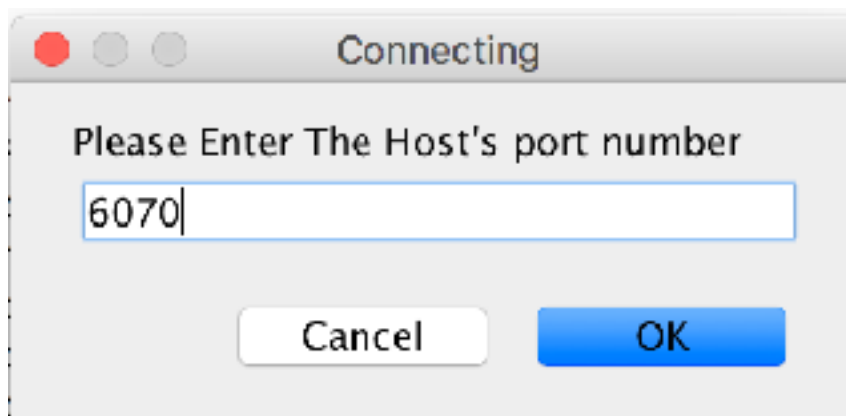
Process Steps:

1. The DNSserver starts and is ready to receive connections from servers.
2. The server starts in the console and requests the IP address of the DNS and the Port to run the Server on, it connects to the DNS and is ready to receive connections from the Clients.
3. The Client runs the Program and is requested to enter the IP address of the Server and the port number of the server, if the connection is accepted, He is requested to Enter a Unique username to be Identified in the Servers.
4. If The Username is Accepted The Chat GUI Appears, else, The User Is requested to enter another name.
5. In the Chat GUI, The User can see all users on all servers on a panel on the right.
6. If the User A clicks a button of user B on The members list panel, all messages will be sent to that user until the user A clicks another member or the the member list or the User B Logsoff.
7. When The User receives a message from any other user, the Message appears on the Chat History box along with the name of the sender.

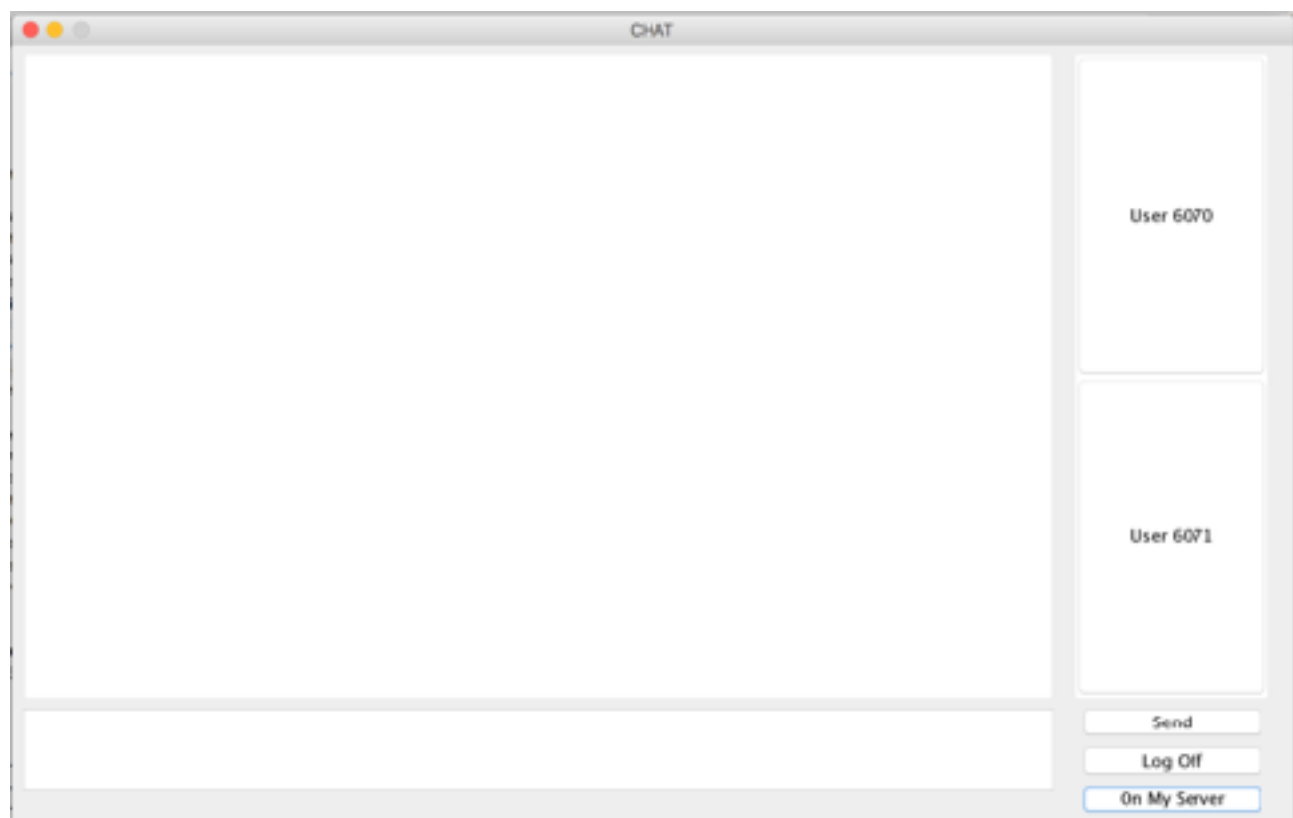
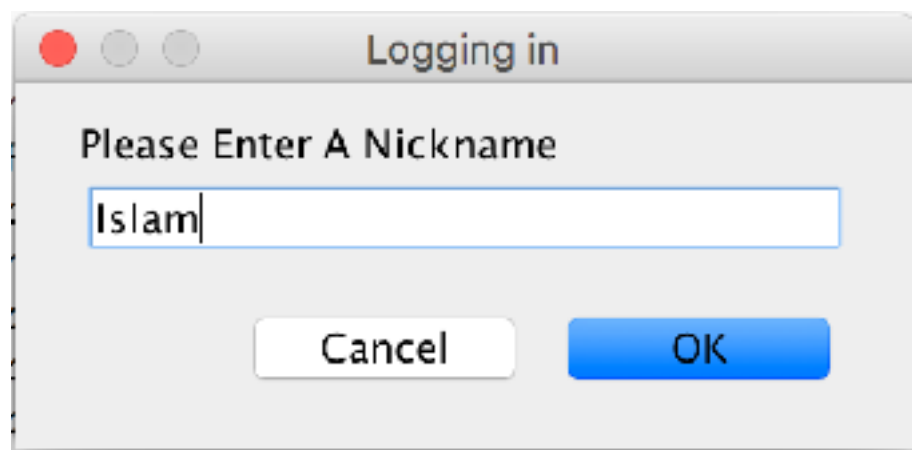
```
RunServer [Java Application] /Library/Java/Java'  
Enter DNS IP Address:  
127.0.0.1  
Enter Server Port:  
6070
```

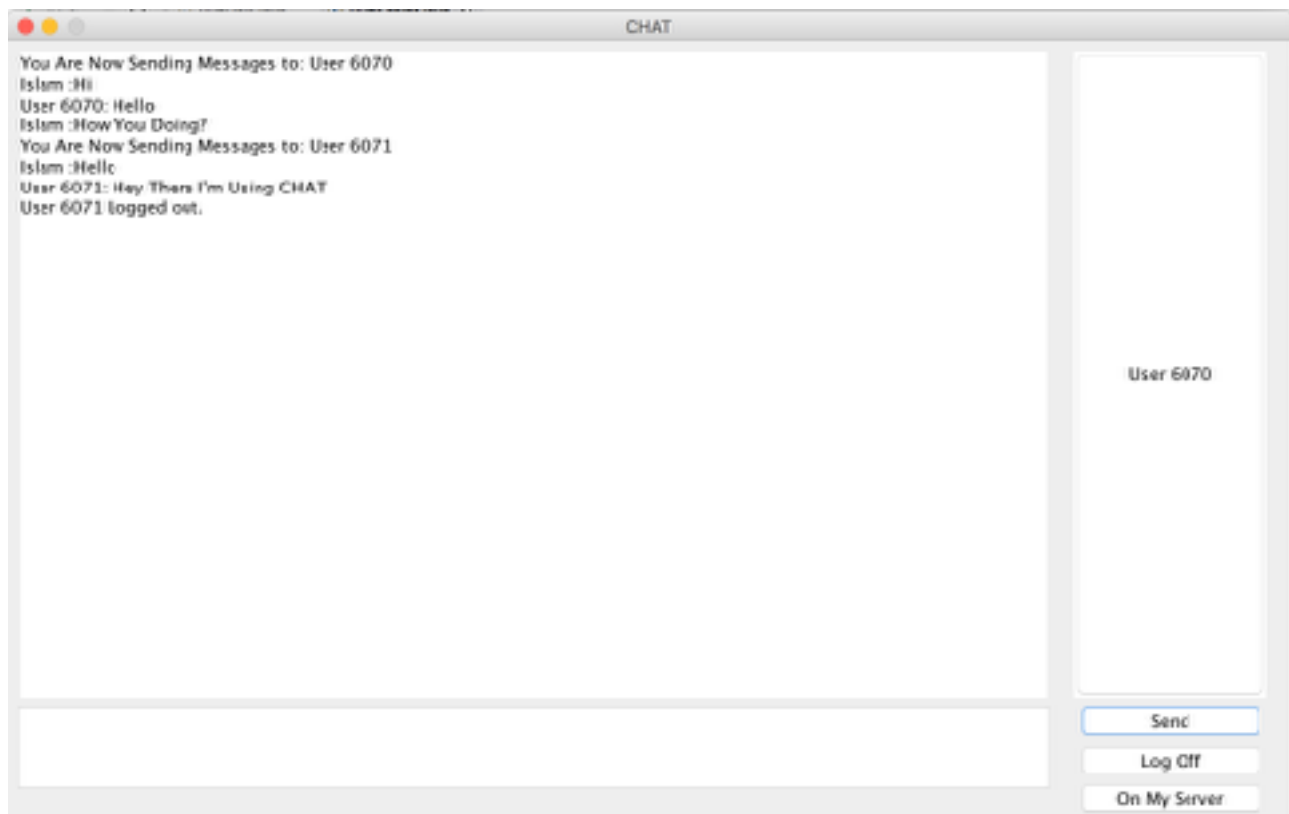
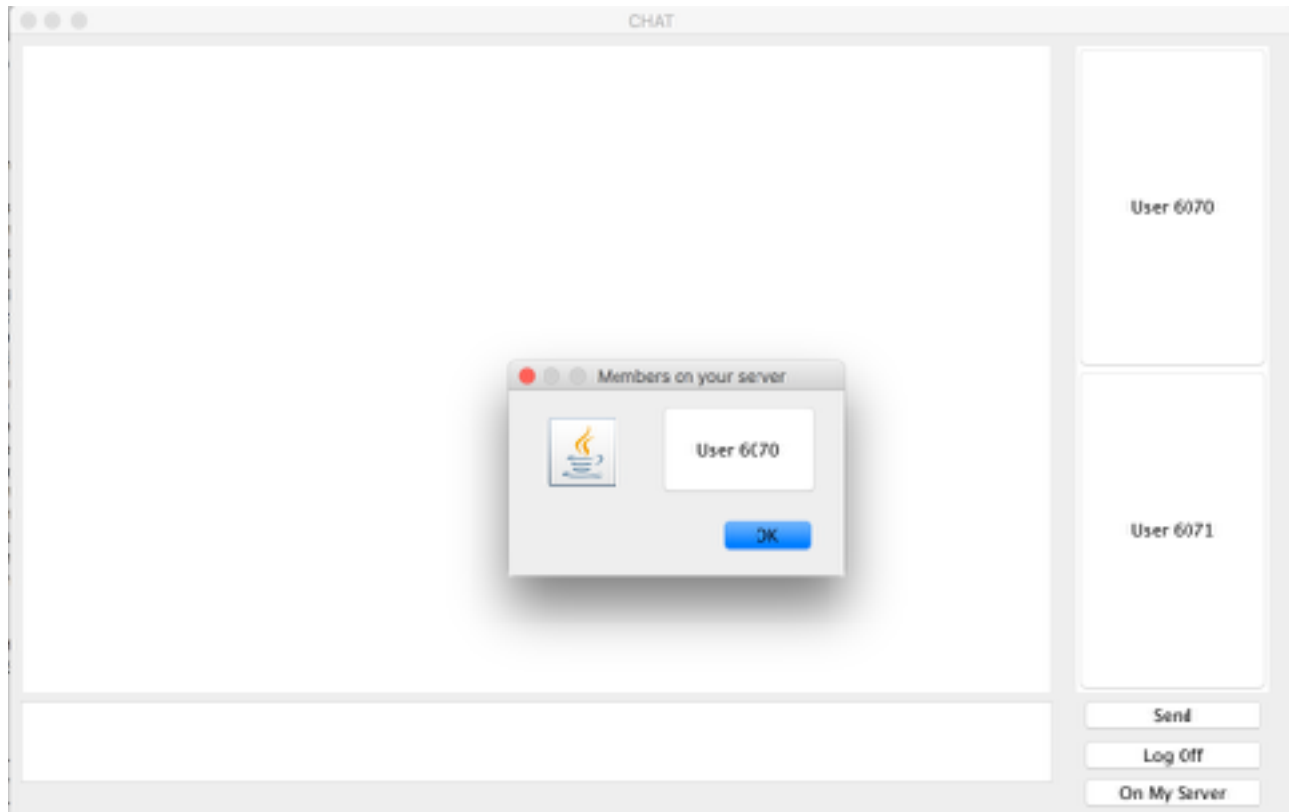


A screenshot of a macOS-style dialog box titled "Connecting". The dialog has a light gray background and a title bar with three window control buttons (red, yellow, green). The main text inside the dialog reads "Please Enter The Host's IP Address". Below this text is a text input field containing the IP address "127.0.0.1". At the bottom of the dialog are two buttons: "Cancel" and "OK". The "OK" button is highlighted in blue.



A screenshot of a macOS-style dialog box titled "Connecting". The dialog has a light gray background and a title bar with three window control buttons (red, yellow, green). The main text inside the dialog reads "Please Enter The Host's port number". Below this text is a text input field containing the port number "6070". At the bottom of the dialog are two buttons: "Cancel" and "OK". The "OK" button is highlighted in blue.





Package: controller

Class: Controller

Attributes:

Client : Instance of the Client
gui: Instance of ChatGUI
destination: String of the User I am currently Chatting with.

Methods:

Controller(): Constructor of the Controller Initializes a Client and gets the IP and port number of the Server The Client wants to connect to, Then Connects the Client to that server.

InitiateNameRequest(): Request a unique username from the client and Calls Join method in Client with the entered name.

Join(Boolean a): Takes a boolean specifying Whether The username was accepted or not, If yes, The Chat GUI is shown, else, Another name is requested from the user.

actionPerformed(ActionEvent e): Responds to Actions from the GUI.

ShowMessage(String s): Shows String s -which is the received message- on the GUI.

UpdateMembers(TreeSet<String> treeSet): Takes a TreeSet of members on all Server and updates the Member list on the GUI.

ShowMembersOfServer(TreeSet<String> content): Takes a TreeSet of Members on the Same server and shows them to the user.

Package: gui

Class: ChatGUI

Attributes:

chatMessage: The Message to be sent to other users.
chatHistory: The Chat log for the client.
members: All online members on all servers.
LogOff: Button to Log off the server.
Send: Button to send the message in the chatMessage.
membersOfServer: Button to get Members on the same server only.

Methods:

ChatGUI(Controller c): Constructor takes a Controller, Creates the Whole View and adds the Controller as listener to all buttons.

Package: Host

Class: Client

Attributes:

socket: Socket with the server.

in: `ObjectInputStream` from the server.
out: `ObjectOutputStream` to the server.
name: name of the client.
control: Instance of the Controller.
rThread: Thread to receive messages from the server.

Methods:

`Connect(String host, int port)`: Connects the Client to the sever using the host and port and creates the Receiving Thread rThread.

`join(String name1)`: Sends to the Server name1 as the Chosen username.

`getMembersOfServer()`: Requests all members on the same server from the server.

`LoginSuccess(String s)`: Changes the clients name to the String s.

`Chat (String Source, String Destination, int TTL, String Message)`: Sends a message to The Server to be sent to another user (Destination) with TTL = TTL and the message is the String "Message".

`LogOff()`: Log off the Server.

Class: DNSserver

Attributes:

DNS : Socket that other servers connect to.
port : Port to run the DNS on.
servers: Mapping the server id to the `ServerThread`.
clients: Mapping Clients' usernames to their servers.
count: number of connected servers.

Methods:

`DNSserver()`: constructor creates the server socket DNS and waits for connections to it.

`LogOff(String name)`: respond to the Users' requests to logoff the server and removes the User from the list of online clients and updates the list on other servers.

`login(SystemMessage msg, ServerThread server)`: Respond to users' requests to login, Checks if the name in msg is valid or not and updates the list of members accordingly.

`route(Message msg)`: Routes the message from on server to the Destination Client's Server.

`members()`: returns a set containing all online users.

Class: Server

Attributes:

connection: Socket that Clients connect to.
dns: socket to connect to `DNSserver`.
in: input from dns.
out: output to dns.
Clients: Mapping usernames to `ClientThreads`.
ClientsId :Mapping Clients' Ids to `ClientThreads`.
count: number of Clients on the server
rThread: `ServerThread` to receive from `DNSserver`.

Methods:

Server(String dns, int port): Constructor. connects to dns, creates rThread and accept connections from clients.

join(int id, String username): forwards client's username request to DNS.

joinResponse(int id,String username): receives DNS's response to the join request and react to the client accordingly. If Accepted, user is added to list of users on the server.

route(Message msg): Sends message to receiver if receiver is on the same server, else, The message is forwarded to the DNS.

MembersListResponse(TreeSet<String> t): Takes a set of members and Update the memberList for all clients.

logout(String username): Removes username from the list of users and notifies the dns that the user has logged off.

getMembersList(): returns set of members on that server.

Package: Main

Class: RunClient

Runs the Controller of the Client.

Class: RunServer + RunDNS

Run The server and the DNS without GUI.

Package: Messages

Class: Message

Chat Message from Client to another.

Attributes:

msg: The message to be sent.

sender: sender's username.

receiver: receiver's username.

TTL: Message's Time to Live

Class: SystemMessage

Send Objects between servers, DNS, and Clients.

Attributes:

content: Object sent between end systems.

type: type of the message.

sender: String used for either specifying the sender's username or add a string that the receiver needs to process the message.

Enum: SystemMessageType

Type of the message to be processed by its receivers.

Package: Threads

Class: ClientReceive

Created by Client to receive from Server.

Attributes:

- client: Instance of the client who created the Thread.
- in: Receive Input from Server.
- control: instance of the Controller of that client.

Methods:

- run(): receives objects from the Server and calls methods in the controller or the Client to react to those messages according to the type of the message.

Class: ClientThread

Created by Server to Respond to Messages and requests from the Client.

Attributes:

- server: Instance of the server that created the Thread.
- client: Socket connecting the server to the client.
- out: Output stream to the client.
- in: Input stream from the client.
- ClientId: The ID of the Client on the Server.

Methods:

- run(): receives messages from the Client and calls methods in the Server to react to them accordingly.

Class: ServerReceive

Created by Server to Receive Messages from DNS.

Attributes:

- server: instance of the server that created the Thread.
- in: Input Stream from the DNS.

Methods:

- run(): receives messages from the DNS and calls methods in the Server to react to them accordingly.

Class: ServerThread

Created by DNS to Respond to Server's Requests.

Attributes:

- DNSserver: Instance of the DNSserver that Created the thread.
- serverSocket: Socket between DNSserver and Server.
- out: Output Stream to the Server.
- in: Input Stream from the Server.
- ServerId: ID of the Server at the DNSserver.

Methods:

- run(): Receives messages from the Server and calls Methods in the DNSserver to react to them accordingly.

