Project Documentation: tcpChat

Package: hosts

Class: Client

Attributes:

- `socket`: the socket through which the client communicates with the server.
- `name`: the name of the client on the server.
- `inStream`: the stream through which the client can receive data from the server.
- `outStream`: the stream through which the client can send data to the server.
- `rt`: the thread that receives any data coming from the server.

Methods:

- `void join(String hostname, int port, ClientController gui):` sends a join request to the server and promotes the client to enter a nickname that is checked for validity at the server side. Once the nickname is valid, the method initiates the receiving thread that will receive data coming from the server.
- `void chat(String destination, String message):` calls the following method and sets the TTL to 2. This values is to be used later when updating the application to have 4 servers.
- `void chat(String destination, String message):` forwards the message to the server through the output stream with the name of the receiver.
- `void quit(TreeMap<String, JTextArea> x):` sends a request to the server to terminate the connection and infroms the clients who were with this client that he has logged off.
- Some getters for attributes.

Class: DNSServer

Attributes:

- `hostname`: the name of the machine running the DNS server.
- `port`: the port number of the DNS server.
- DNS: the socket that accepts connection requests from clients (lower-level servers).
- toDNS: the socket through which the server communicates (as a client) to the DNS server.
- `servers`: a mapping of each server ID to its dedicated server thread.
- `clients`: a mapping of all online clients to the ID of the server directly connected to each of them.
- `cap`: the number of the servers connected to the DNS server.

Methods:

- `void run()`: accepts any incoming connection requests and initiates a separate thread to serve this connection.
- `int joinResponse(ServerThread st)`: adds a new server to the network and assigns and ID for it.
- `void logOff(String name)`: removes the client with the passed username from the list of connected clients.
- `void route(ChatMessage message)`: forwards the message to the server that is connected to the receiver.
- `void updateMemberLists()`:updates the member list for all connected users.
- `boolean memberLogin(String name, int ct)`: adds the client with the requested username to the list of connected clients if the username is valid.
- `boolean isUsedName(String name)`: checks whether the requested username is already used or not.
- some getters for the attributes.

Class: Server

Attributes:

- `hostname`: the name of the machine running the server.
- `port`: the port number of the server.

- `ServerSocket`: the socket that accepts connection requests from clients.
- `toDNS`: the socket through which the server communicates (as a client) to the DNS server.
- `outStream`: the output stream through which the server can send data to the DNS server.
- `inStream`: the input stream through which the server can receive data from the DNS server.
- `clients`: a mapping of each client name to its dedicated thread.
- `ctID`: a mapping of the client thread ID to the corresponding thread object.
- `nxtCT`: a counter for the ID of the next created client thread.
- `allClients`: all online users on the network.

Methods:

- `void connectToDNS()`: initiates a connection between the server and the DNS server.
- `void run()`: accepts any incoming connection requests and initiates a separate thread to serve this connection.
- `void joinRequest(int ct, String username)`: sends a request to the DNS server to check if this username is already used or not.
- `String joinResponse(String ct, boolean accepted)`: informs the client thread whether the requested username is accpeted as a valid one or not to take furthur actions.
- `void logOff(String name)`: closes the client socket, removes its name from the map and informs the DNS server about that.
- `void route(ChatMessage message)`: sends the message to the receiver if the receiver is connected to the network or informs the sender the receiver is not found otherwise.
- `String memberListResponse(TreeSet<String> allClients)`: updates the member list for each client connected to the server.
- `void addClient(String name, ClientThread ct)`: maps a new client to its dedicated thread.

- some getters for the attributes.

Package: messages

Class: ChatMessage

Attributes:

- `type`: the type of the message sent to the server.
- `sender`: the client sending the message.
- `receiver`: the client (if any) to which the message will be sent.
- `TTL`: a checker that is used so as the message does not loop infinitely between the servers.
- message: the message to be send through the chat.

Methods:

- some getters to access the attributes

Class: MessageType

- an enumartion to determine the type of the message
  - ***`WHO_IS_IN`*** : a request for connected users list.
  - ***`MESSAGE`*** : a message to another client.
  - ***`LOGOUT`*** : a request to terminate the connection.
  - ***`LOGIN`*** : a request to initiate a connection.
  - ***`SERVER_RESPONSE`*** : a direct message from the server to a client (for client-server requests or error feedback).

Package: threads

Class: ClientThread (Server side)

Attributes:

- ctID: the ID of the client thread relative to the creating server.
- `server`: the server that created this thread to serve a client.
- `clientSocket`: the socket through which the server communicates with that client.
- `username`: the name of the client on the server.
- `sInput`: the input stream that receives any data from the client.
- `sOutput`: the output stream that sends any data to the client.

Methods:

- **`void`** `setUsername()`: promotes the user to enter a username and sends the proposed username of the client to the server.

- `void joinResponse()`: receives the server response on the proposed username and starts the service if it is valid or call the previous method otherwise.
- `void run()`: waits for a request from the client and responds to the request according to the type of the request.

Class: DNSReceivingThread (Server side)

Attributes:

- `serverSocket`: the socket through which the server communicates with the DNS server.
- `server`: the server which created this thread.
- `inStream`: the input stream that receives any data from the DNS server.

Methods:

- `void run()`: waits for any message from the DNS server and does the appropriate action accordingly.

Class: ReceivingThread (Client side)

Attributes:

- `clientSocket`: the socket through which the client communicates with the server.
- `inStream`: the input stream that receives any data from the server.
- `controller`: the controller of the client application.

Methods:

- `void run()`: waits for any message from the server and does the appropriate action accordingly.

Class: ServerThread (DNS Server side)

Attributes:

- `dns`: the DNS server that created that thread.
- serverID: the ID of the server served by this thread.
- `serverSocket`: the socket through which the DNS server communicates with the server served by this thread.
- `sInput`: the input stream that receives any data from the server.
- `sOutput`: the output stream that sends any data to the server.

Methods:

- `void run()`: waits for any request from the server and does the appropriate action accordingly.
- some getters to access the attributes.

Packages: controller, gui, listeners

These three packages contain classes that build up the user interface including the windows, listeners to different button actions and the link between the model and the view. So, it is not important to mention them here, however, the methods names imply what each method does.

Package: main

This package contain three classes that are used to run

1. The DNS server.
2. The servers.
3. The Client applications.

- **Note that:**
  - The servers have no gui, thus they must be initiated through the console.
  - Any client will be connected by default to the server running on the client machine with port number 6000 unless the server information is changed prior to the join request. This can be done throught the Join window that appears when the client application is run.
  - Servers are connected through a centralized server (the DNS server) to keep track of all connected users so that every user will have a unique name. Each client is served by a single server. The server may forward the message to the DNS server if the message receiver is served by another server. Thus, the DNS server routes the messages between the servers only when necessary.
  - Servers act as a client with respect to the DNS server and as servers with respect to normal clients.
  - There is no command for the member list because the list of online users is always available for each client and is updated automatically.
  - There is no bye/quit message. However, any client can close any open chat and can log off the application. In such a case, the client will be removed from other connected clients' members list and the clients chatting with this client will be informed that this client has logged off.
- **Scenario**
  - Suppose that we have to clients "A" and "B" connected to the network through the same server.
  - Each client will appear in the member list at the other client side.
  - "A" can chat with "B" by selecting "B" from the member list and press "start chat" button. The text area is now dedicated for chatting with client "B". "A" can write a message and press "send" or enter.
  - The controller will call the chat method in the client class which will in turn send a request to the server to forward this message to client "B". The server will look for "B" in the clients list and will find it. The message will be forwarded to "B" directly.
  - The receiving thread of "B" will get a message from the server that "A" has sent a message to "B". The controller at "B" will notify the client that there is an incoming message from "A". Once the client presses the button carrying the name "A", the message will be displayed in the text area.
  - The only difference when "A" and "B" are on two different servers is that the server to which "A" is connected will not find "B" in the clients list. Therefore, it

will forward the message to the DNS server. The DNS server knows exactly which server client “B” is connected to.