Design Document

Jason Yang, Jenny Lin, Sayeed Tasnim

Server

Class: Server

- Attributes:
  - UserListHandler allUsers: UserListHandler object containing all users connected to the server
  - ChatHandler allChats: ChatHandler object containing all chatrooms in the server
- Methods:
  - serve(): Runs the server while listening to connections and creates ClientHandlers for those connections

Class UserListHandler

- Attributes:
  - ArrayList<ClientHandler> users: All clients in this user list
- Methods:
  - removeUser(ClientHandler user): Removes user from users [and informs all remaining ClientHandlers in users of the change by adding a message to their outputQueue]
  - addUser(ClientHandler user): Adds user to users [and informs all ClientHandlers in users of the change by adding a message to their outputQueue]
  - getUsers(): Gets the list of usernames as a space separated String

Class ChatHandler

- Attributes:
  - ArrayList<Conversation> chats: All chatrooms in this user list
- Methods:
  - removeRoom(Conversation chatroom): Removes chatroom from chats and informs all remaining ClientHandlers in the server's users of the change by adding a message to their outputQueue
  - addRoom(Conversation chatroom): Adds chatroom to chats and informs all ClientHandlers in users of the change by adding a message to their outputQueue

Class: ClientHandler

- Attributes:
  - ChatHandler myChats: ChatHandler containing all chatrooms client is a member of
  - Queue outputQueue: Queue containing commands to be sent to the client from an appropriate Conversation
- Methods:
  - handleConnection(Socket socket): Makes bufferedReader around the

socket. Parses input from the bufferedReader and sends commands through the socket back to the client

        - parseCommand(String command): parses a command to figure out what action to perform
        - joinChat(Conversation Chatroom): calls addRoom(Chatroom) in myChats, and calls Chatroom.users.addUser(this)
        - leaveChat(Conversation Chatroom): calls removeRoom(Chatroom) in myChats, and calls Chatroom.users.removeUser(this)
        - makeChat(String chatName): creates a new Conversation object with the given chatName and adds it to both the server's ChatHandler and this ClientHandler's ChatHandler; this ClientHandler is the first ClientHandler in the Conversation's UserListHandler
        - sayMessage(Conversation Chatroom, String Message): consumes a String from the inputQueue and sends the Message to the Chatroom's queue
        - hearCommand(String Command): consumes a Command from the outputQueue and sends it back to the Client

Class: ChatRoom(mutable & threaded)

- Attributes:
    - final String Name: the name of the chatroom, set by client
    - UserListHandler users: the UserListHandler of all the clients in that particular chatroom
    - Queue Messages: Queue containing messages that were said in the chatroom
- Methods:
    - relayMessage(String message): consume a message from Messages and send that message to the outputQueue of each user in the users

<u>ClientHandler</u>

This object will handle in general the IO to and from the respective client. Upon construction, the client will require a username from the client and will be checked against the list of users in the userlist object. if the username is unique, the clienthandler will add the instance to the userlist object. if the username is not unique, the socket will be disconnected.

The client handler will have an blocking 'output' queue to handle sending messages back to the client. It will also have a list of all of the chatroom objects that the client is currently associated with.

This class will also be threaded such that each instance will be its own thing. in this thread will be a loop that will check if the output queue is empty and if not, relay the queue back to the client. The client will also check if the buffered reader is empty and if not, start parsing the strings in the reader. These strings will be of the form [header] [message]. The header will contain information of what the clienthandler should do (make a message to room x, disconnect, etc).

If the command is to connect to a chatroom, the client handler will use the Conversation object's UserListHandler to add this instance to the UserListHandler object. It will also append the Conversation object to this instance's ChatHandler, representing the rooms it is connected to.

If the command is to send a message, the clienthandler will parse and append the message to the chatroom object's output queue.

If the command is to disconnect the user, the handler will iterate through all of the rooms that the client is connected to and disconnect the client before disconnecting the client from the server.

UserListHandler
This object will hold a list of all of the connected users on the server. It will have provisions to add a user or remove a user from the list where this object will inform all of the users on this list of the change by appending a command to each users output queue.

ChatHandler
Much like the UserListHandler object, this object will hold a list of all of the chatrooms currently created. if a room is created or destroyed, this class will access the main UserListHandler object and inform all members of the change.

Client

Class Client

- Attributes:
  - String userName
  - GUI listOfUsers
  - GUI listOfChatrooms
  - ArrayList<GUI> chats
- Methods:
  - addNewConversation(String conversationName)
  - updateListOfUsers()
  - updateListOfUsersInConversation(String conversationName)
  - addMessage(String chatroom, String user, String message)
  - checkName(String name)

Clients (people) will connect first to a server at a specified IP address and port, and be required to provide a username - Once a username is given, the gui checks to make sure that the username has no spaces. Once that check passes, a connection is made to the server, and it's checked to see if the username is taken. This will be done on the server-side with a synchronized method that goes through the list of all current users. If the username is taken, a message is returned to the client telling them to try something else, and then they are disconnected from the server. The client then keeps submitting possible usernames until they choose one that is not taken.

If the client receives a message for a chatroom it is no longer a member of, just silently do nothing with that message.

Client-Server Protocol

From client to server:

- connect [username]
    - Sent when a client attempts to connect with a username
- disconnect [username]
    - Sent when a client with the specified username attempts to disconnect

- make [chatroom name]
    - Sent when a client attempts to make the specified chatroom
- join [chatroom name]
    - Sent when a client attempts to join the specified chatroom
- exit [chatroom name]
    - Sent when a client attempts to exist the specified chatroom

- message [chatroom name] [message]
    - Sent when a user attempts to send the particular message to the specified chatroom.

From server to client:

- connectedServer
    - Sent when the server successfully connects the user
- invalidUsername
    - Sent when the user submits a username that is already taken
- disconnectedServer
    - Sent when the server is about to disconnect the user
- connectedRoom [chatroom name]
    - Sent when the server successfully connects the user to the specified chatroom
- disconnectedRoom [chatroom name]
    - Sent when the server successfully disconnects the user to the specified chatroom
- serverUserList [username list]
    - Sent when the server updates the list of users on the entire server
- chatUserList [chatroom name] [username list]
    - Sent when the chatroom updates the list of users in the chat
- serverRoomList [chatroom list]
    - Sent when the server updates the list of chatrooms on the entire server
- message [chatroom name] [username] [message]

Sent when the chatroom sends a message to the users

Client to Server Grammar

COMMAND :== (CONNECT | DISCONNECT | MAKECHAT | JOINCHAT | EXITCHAT | MESSAGE) NEWLINE
CONNECT :== "connect " USERNAME
DISCONNECT :== "disconnect “ USERNAME
MAKECHAT :== "make " CHATROOMNAME
JOINCHAT :== "join " CHATROOMNAME
EXITCHAT :== "exit " CHATROOMNAME
MESSAGE :== "message " CHATROOMNAME " " MESSAGETEXT
USERNAME :== [\p{Graph}]+
CHATROOMNAME :== [\p{Graph}]+
MESSAGETEXT :== [\p{Print}]+
NEWLINE = "\r?\n"

Server to Client Grammar

COMMAND:== (CONNECTSERVER | INVALIDUSERNAME | DISCONNECTSERVER | CONNECTROOM | DISCONNECTROOM | SERVERUSERLIST | CHATUSERLIST | SERVERROOMLIST | MESSAGE) NEWLINE
CONNECTEDSERVER :== “connectedServerSent”
INVALIDUSERNAME :== “invalidUsernameSent”
DISCONNECTEDSERVER :== “disconnectedServerSent”
CONNECTROOM :== “connectedRoom “ CHATROOMNAME
DISCONNECTROOM :== “disconnectedRoom ” CHATROOMNAME
SERVERUSERLIST :== “serverUserList ” USERNAMELIST
CHATUSERLIST :== “chatUserList” CHATROOMNAME “ “ USERNAMELIST
SERVERROOMLIST :== “serverRoomList “ CHATROOMLIST
MESSAGE :== “message “ CHATROOMNAME “ “ USERNAME “ “ MESSAGETEXT
USERNAMELIST :== (USERNAME “ “)* USERNAME
CHATROOMLIST :== (CHATROOMNAME “ “)* CHATROOMNAME
USERNAME :== [\p{Graph}]+
CHATROOMNAME :== [\p{Graph}]+
MESSAGETEXT :== [\p{Print}]+
NEWLINE = "\r?\n"

Notes: \p{Graph} is any alphanumeric character or punctuation and \p{Print} is any printable character (alphanumeric, punctuation, or space).

Before a client attempts to connect, the client will need to provide a username first. This will establish the connection and send the string “connect [username]”. After the client side checks to make sure that the username contains valid characters only (no spaces), the server will check the available list of usernames and if the username has not been taken yet, the server will construct a ClientHandler for the particular client and add the client to the UserListHandler. The server will send a “connectedServer” command to let the client know that the client is connected and then display the UserListHandler and the ChatHandler to the client as a list of current users and chats by sending a “serverUserList” command. Otherwise, if the username is taken, the server will send a ‘serverUserList’ message with a list of users in the chat displayed in a separate window and an ‘invalid’ message to the user. This will state to the user that the username is already taken and that the user should select a different chat name.

Whenever the list of all the server users is updated, the server sends a chatUserList command to each of the clients.

When a user requests to make or join a chatroom, the client will send the respective command “make” or “join” to the server. The GUI will create a new window for the specific chat and the user will be able to send messages through a text box. The server will send a “connectedRoom” command to let the client know that the client is connected to the chatroom. When making a chatroom, the chatroom’s name cannot already be taken, otherwise an ‘invalid’ message will sent to the user, prompting them to select a different name for the chatroom.

When a user sends messages, the client sends a “message” command to the server. When a server receives the “message” command, it will process it and place the message into the proper conversation queue. The conversation queue will consume the messages and the server will send “message” commands to each of the clients in the particular chat room.

When the list of users in the particular chat room changes, the server will send a “chatUserList” command to each of the clients in the particular chat room. This will update each client and GUI with a new list of users in the chatroom.

When a user disconnects from a chatroom, the client will send an “exit” command to the server. The server will process the disconnection and relay a “disconnectedRoom” command back to the client if the client is still connected to the server.

When the user disconnects from the server, a “disconnect” command is sent to the server. The server will then remove the user from all chatrooms and from the list of all users on the server and return a “disconnectedServer” command. All open GUI chat windows will be closed except for the original username prompt window. The user will need to login again in order to reconnect to the chatroom.

Conversation

The Conversation will be a class that serves as one ‘room’ of people chatting. It will contain a list of all the connected clients in that room (subscriber) and upon receiving a message from the server with text, update all of the connected clients of the change (publisher). People are free to come and go from this object (subscribe/unsubscribe). As long as one person remains in the room, the Conversation will still be valid and upon becoming invalid, the Conversation will deregister itself from the server and all connected clients will be notified of this change.

Conversation classes will contain an instance of UserList that is a subset of the UserList that the server has. The Conversation will need at least one client in the list of ClientHandlers for the Conversation to hold. Otherwise, the server will delete the Conversation by calling the respective command in the ChatHandler. The Conversation will have a blocking queue of the messages to be passed to the members of this conversation. This class will be threaded such that the thread checks if anything is in the queue and if so, alert all parties. This class will aslo have provisions to add or remove a client from this object, updating the respective lists and informing all members of this change.

Concurrency

On the server side, the ConnectionHandler and the ChatRoom objects both receive individual threads. This is so that multiple ChatRooms can process messages at the same time, and multiple ConnectionHandlers can handle connections at the same time. This works because the connection handler will have an input and output buffers. any message that has to be sent to the client will be queued in a blocking queue which is thread safe. One thread will consume from this queue in its own time and push the messages to the network. The other thread will similarly check the buffered reader for data and parse it. This makes the connection handler modular such that multiple people can queue writes to it. Similarly, multiple people can write to a chat room using this queue strategy.

In order to prevent race conditions from occurring, we will make sure that the different list objects are all threadsafe, since the list objects are the only modifiable objects shared between the different threads. In RoomList, getRooms iterates over the list of Rooms, but it is a private method that is only called by add and remove, so we do not need to worry about RoomList's hashmap getting modified while we are iterating over its values. This means we can make RoomList threadsafe just by synchronizing the add and remove methods. In UserList, however, informAll is a public method, so we are not allowed to directly iterate over the HashMap's values, as the add or remove method could be called during iteration, modifying the collection of Connection Handlers and producing incorrect output. Therefore every time a call to informAll is made, a copy of the HashMap's values is made for informAll to iterate over. Therefore if someone is added to the userList after informAll has been called, they simply will not receive the message. If someone is removed from the ServerUserList after informAll has been called, that ConnectionHandler's queue will still receive a message, but because the client for that ConnectionHandler has left the entire server, there will no longer be a thread to read from the queue, so the message will just sit there unread. If someone is removed from the ChatUserList after informAll had been called, the client will just receive a message for a chat they are no longer a member of, and the message will be discarded clientside. The copying of the Hashmap's values, the add method and the remove method are all synchronized on UserList's HashMap to prevent concurrency issues.

When the client tries to connect to the server, a thread is made that attempts to create a connection Handler. This is to allow multiple clients to connect to the server at the same time. The only potential concurrency issue comes when the thread adds the new connectionHandler to the server's userList, and that issue was handled when we made the Lists theadsafe. Once the connectionHandler is successfully made, the thread terminates.

Aside from the main thread that handles all the GUI issues, the client will also have a background thread (from a swingworker) that receives messages from the server and modifies the appropriate ChatRoomModel, which is a model for a particular chatroom, or the MainRoomModel, which is a model for the chat as a whole. There will not be a separate thread for sending messages to the server, as we will just use action listeners to send messages when the appropriate action event occurs and then immediately go back to dealing with the Gui. The Gui will then update itself based on changes in the model. As we have currently designed the client, there should be only one thread that modifies the model, so we shouldn't have to worry about any concurrency issues in terms of multiple threads trying to modify the same ChatRoomModel at the same time. However, if we decide to implement multiple threads, we will make sure to synchronize the models so that only one thread can modify it at a time.

Testing Strategy

Serverside Testing

- Test Connection Handler
    - Make sure that a connection handler can be instantiated properly
        - Should be getting the right output
        - Should be getting added to the userlist *after* username is chosen
        - Should close socket if username is already taken
    - Check that parseInputs does what is expected
        - try disconnecting
        - make a chatroom (both existing and non existing)
        - join a chatroom (both existing and non existing)
        - leave a chatroom (one connectionhandler is a part of, one it isn't a part of, and one that doesn't exist)
        - send a message to a chatroom (one connectionhandler is a part of, one it isn't a part of, and one that doesn't exist)
    - Check that removeAllConnections does what is expected
        - try with both empty and full list of connected chatrooms
    - Make sure updateQueue works properly
        - spam a bunch of threads and update the queue all at once to make sure no messages are lost
- Test the List Objects independently
    - Make a bunch of connection handlers to add and remove from a single server user list
    - Make a bunch of connection handlers to add and remove from a single chat user list
    - Make two user lists. Add/remove the same user from both

            - In all these cases, we check the connection handler's queue afterwards to make sure the user is informed properly.
            - Do something to stop connection handler from reading it's queue right away?
        - Make a single chat user list, but make two threads that add/remove from it at the same time.
        - Make a chatroom list and add/remove a bunch of chatrooms from it
        - Have multiple threads add/remove chatrooms from a chatroom list
        - Have a chatroom list, add/remove someone from it, then try to modify the userlist while that chatroom list is informing everyone
    - Test the Chatroom Object independently
        - Create a chatroom, add a bunch of users to it and then remove them all
            - Chatroom should add self to list when starting, then terminate thread and remove self from chatroom list once empty
        - Make sure updateQueue is working
            - make a bunch of threads to send messages to the chatroom. then make sure all the users in it got all the messages
    - Test the Server as a whole

Clientside Testing

- Test individual chatting system
    - Create a valid username and attempt to login
    - Create an invalid username and check to see an indication if the username is taken or invalid
    - Create a chatroom with a valid chatroom name
    - Create a chatroom with an invalid chatroom name (bad characters/already taken)
    - Join chatrooms
    - Exit chatrooms
    - Message all the users in the chatroom
- Test multiple tabbed chatting system
    - Create and join multiple chatrooms and messages to different chatrooms
    - Check for concurrency bugs for a particular client
- Test multiple users in a conversation
    - Multiple users will message in the conversation
    - Users will join and exit the conversation and the output of the messages will be checked against expected chatroom behavior
- Combine multiple chatrooms and multiple users
    - Consider all cases of performing actions on the room, e.g. The last person exits the room and someone attempts to join the room at the same time.
- Deal with sudden disconnects
    -

Things to Add Someday

- Letting the original creator of a chatroom kick someone out?