

A8: Build a Chatroom

CS 5010

Northeastern University—Seattle

Fall 2019

Adrienne Slaughter

ahslaughter@northeastern.edu

This assignment is due 2 December 2019, 6pm. Submissions should be made using your group repo.

In this assignment, you will build a simple chat room. It will consist of a client and a server. The server will be responsible for managing the client connections (up to 10 clients can be connected and in a chat room at one time), accepting messages from one client and sending the messages to all attached clients. Clients will be able to either send a message that is public in the chat room, or that goes directly to a single, specified client.

To add some interest to the chatroom, participants can also send a randomly generated insult to another client in the chatroom. You'll add this feature by incorporating your random sentence generator from Assignment 7.

Objectives

- Build competence with socket programming
- Apply your multi-threading skills
- Re-use code you've built before as a library
- Programming to an interface (that is, the protocol)

Details

Server

The server is responsible for allowing clients to connect, get a list of all other clients connected, and disconnect. It also handles sending messages to all clients or just one.

When the server starts up, it should start listening on any available port. Once it is up and listening, it should print a reasonable message on the console that includes what port it is listening on (so clients can connect).

The server will need to continually listen for more connections (handling them appropriately), as well as handle each connected client. The server should handle up to 10 clients connected at a single time. The server will be responsible for listening for commands from the client as well as sending messages to the client.

Client

The client will open a socket to communicate with the server. It will maintain the socket to listen for incoming messages from the server (public or private messages), as well as listen to the UI (terminal) for messages from the user to send to the server.

When starting the client, you will need to pass in the IP address (or `localhost`) and port for the server. Not providing these details should result in a graceful failure.

Client interface

In addition to your client allowing a user to send messages and all chat room messages to be displayed, it will require additional commands.

Your client must list all commands when the user types a `?` (alone on a line—you can ignore question marks as part of a message).

Other commands your client must provide for:

- **logoff**: sends a `DISCONNECT_MESSAGE` to the server
- **who**: sends a `QUERY_CONNECTED_USERS` to the server
- **@user**: sends a `DIRECT_MESSAGE` to the specified user to the server
- **@all**: sends a `BROADCAST_MESSAGE` to the server, to be sent to all users connected
- **!user**: sends a `SEND_INSULT` message to the server, to be sent to the specified user

While the above commands must be implemented, you do not need to call them these words specifically (you may choose to use different words, and possibly include shortcut commands), but this functionality needs to exist. You **ARE**, however, required to implement the `'?` to print the command menu. You may have an additional menu command if you would like.

Chatroom Protocol

The application data for the chatroom consists of a sequence of data. The format depends on what kind of message is being sent. All frames of application data begin with a message identifier. Strings are sent as a byte array; before a string is sent, an `int` indicating the length of the array/string is sent. Each field of a frame will be separated by a single space character.

Example:

The following frame sends a message that a new client is connecting to the server with the username of `"aha"`:

19		3		aha
----	--	---	--	-----

Or as a string: `"19 3 aha"`.

All of the supported messages/frames are described below.

Connect message:

int Message Identifier: CONNECT_MESSAGE
int size of username: integer denoting size of the username being sent
byte[]: username

Connect response:

int Message Identifier: CONNECT_RESPONSE
boolean success: true if connection was successful
int msgSize: size of message sent in response
byte[] message: String in byte[]. If the connect was successful, should respond with a message such as "There are X other connected clients". If the connect failed, a message explaining.

Disconnect message:

int Message Identifier: DISCONNECT_MESSAGE
int size of username: integer denoting size of the username being sent
byte[]: username

Disconnect response:

int Message Identifier: DISCONNECT_RESPONSE
int msgSize: size of message sent in response
byte[] message: String in byte[]. If the disconnect was successful, the message should be "You are no longer connected.". If the disconnect failed, a message explaining.

Query users:

int Message Identifier: QUERY_CONNECTED_USERS
int size of username: integer noting the size of the username
byte[] username: username (who's requesting)

Query response:

int Message Identifier: QUERY_USER_RESPONSE
int numberOfUsers: if the request fails this will be 0. If there are no other users connected, this will be 0.
int usernameSize1: length of the first username
byte[] username: username1
...
int usernameSize2: length of the last username
byte[] username: usernameX

When responding to a query, the server must make sure the request is coming from an already connected user. A list of all the OTHER connected users will be sent.

Note: The server will need to send a block of data for *every* connected user.

Broadcast Message:

int Message Identifier: BROADCAST_MESSAGE
int sender username size: length of sender's username

byte[]: sender username
int message size: length of message
byte[]: Message

Server will broadcast this message to all connected users, specifying that it came from sender. If the sender username is invalid, the server will respond with a FAILED_MESSAGE.

Direct Message:

int Message Identifier: DIRECT_MESSAGE
int sender username size: length of sender's username
byte[]: sender username
int recipient username size: length of recipient's username
byte[]: recipient username
int message size: length of message
byte[]: Message

Sending a direct message will fail if the sender or recipient ID is invalid.

Failed Message:

int Message Identifier: FAILED_MESSAGE
int message size: length of message
byte[]: Message describing the failure.

Send Insult:

int Message Identifier: SEND_INSULT
int sender username size: length of sender's username
byte[]: sender username
int recipient username size: length of recipient's username
byte[]: recipient username

The server will randomly generate an insult (from A7's Random Sentence Generator) and broadcast it to the chat room. Something like the following should be posted:

```
aha -> sparky: You mutilated goat.
```

When the client receives either a BROADCAST_MESSAGE or a DIRECT_MESSAGE, it should print the message out to the console. When printing out a message, please indicate which username sent the message in some way.

Message Identifiers

Message Type	Value
CONNECT_MESSAGE	19
CONNECT_RESPONSE	20
DISCONNECT_MESSAGE	21

QUERY_CONNECTED_USERS	22
QUERY_USER_RESPONSE	23
BROADCAST_MESSAGE	24
DIRECT_MESSAGE	25
FAILED_MESSAGE	26
SEND_INSULT	27

Tips

- Do not simply copy and paste your code from A7 into A8. Package it up into a jar and include it as a library.
- Be sure to program to the protocol, and ask questions if you have any.
- When you program to the protocol, your server can communicate with someone else's client, and vice versa.
- If you would like to enhance the protocol by ADDING message types, you may. If you do so:
 - Be sure to document the new message types and include the documentation (as an .md or .pdf file) in your project repo
 - Ensure that your client and server work with the base protocol, and is NOT DEPENDENT on the enhanced protocol to work properly
- This is an example of a program that will use multiple threads, but not for the purpose of attempting to improve throughput.
 - It is not possible to implement this project properly without multiple threads.
- It would be lovely to send compliments rather than insults. If you would like to implement a compliment grammar rather than the insult grammar, you are welcome to replace the insults with compliments. Feel free to share it with the class if you like.

Logistics

Grading

The correctness grade for this assignment will be based on:

- Following and implementing the protocol correctly
- Properly handling multiple clients on the server side
- Properly handling the UI, sending messages to the server and receiving messages from the server on the client side.
- Utilizing A7 as a library

Gradle and Github

The Gradle requirements are the same as for the previous assignments.

As with your previous team assignments, each individual group member should create their own branch while working on this assignment. Only merge to the master branch when you're confident that your code is working well. You may submit pull requests and merge to master as often as you like but you should not tag your TA until you are ready to submit. Only one person will need to tag the TA when you're ready to submit your finished assignment.

Deliverables

- All of your source and test code, no class files or other binaries
- Your a7.jar (this is separate from the "no other binaries" bullet point above)
- You may include a grammar file
 - This could be included in the a7.jar—you decide if that's appropriate or not.
- A README.md file, where you describe how to run your program
 - Include the entry point to your program.
 - Include how to run your server and client
 - Give a high level description of key classes/methods.
 - Include any assumptions you made about the nature of the problem.
 - Include steps you took to ensure correctness.
 - Make sure it's clear how to run your program from the command line.
 - If you'd like, you can include an example of the client or server side output
 - See <https://guides.github.com/features/mastering-markdown/> for info on formatting code/terminal snippets properly
- A class diagram representing your code