Build a Chatroom

Write a program to 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 either:

- Creating a simple component that returns a random insult/sentence. There should be at least 5 different randomly chosen statements.
- Incorporating your random sentence generator from Assignment 4.

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 an 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 **?** (a question mark 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

With the above commands, no messages are sent unless the @user, @all, !user commands are used. You may optionally send any strings the user enters that does not match any of the above commands as if it were @all. This is entirely up to you. If you choose not to, then you may want to give an error to the user stating it is an unknown command. Keep in mind that the optional implementation means that a user *must* use "@all who" to send "who" to all connected clients (not a likely message, but something to consider).

Example

Sample command	Result
@bob hello bob, how are you?	"hello bob, how are you?" is sent to user "bob"
@all Hello Everyone!	"Hello Everyone!" sent to all connected clients
Hello World	You can do either: 1. Error message to user 2. "Hello World" sent to all connected clients

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

You may want to look into the DataInputStream and DataOutputStream classes for writing your data as bytes.

Strings should be UTF-8
As an example: the following code creates a string, converts to byte[] encoded as UTF-8 and then back to another string:

```
String test = "HelloEncoding";
byte[] b = test.getBytes(StandardCharsets.UTF_8);
String test2 = new String(b, StandardCharsets.UTF_8);
```

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:
Send back a CONNECT_RESPONSE

The success field will return “true” if the disconnect is successful (valid user)
The message field is set as follows:

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 (either from a simple insult generator you create for this assignment or optionally from using your Assignment 4'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