Multi-Client Chat Program

By Kevin Humphreys

Goal

To create a program that would allow multiple clients to talk to each other. The client will need to be handle as many clients as there are resources available and facilitate communications between them. It will also need to execute any commands that the clients send. The client must be able to send messages to the server as well as commands (messages prefaced by a ‘/’)

**Design**

Classes:

* ThreadedSocket
* ThreadedSocketEvent
* ThreadedSocketHandler
* Client
* Server
* ClientApp
* ServerApp

A ThreadedSocket is a container for a Socket able to concurrently send and receive from the enclosed Socket. Allows buffering multiple messages to be sent. Sending and receiving is event based. This handles all communications being sent over the socket. It further abstracts the underlying Socket allowing the implementing ThreadedSocketHandler’s to simply handle onMessageReceived() and sendMessage() without worrying about waiting for things to happen.

A ThreadedSocketEvent is created whenever a message is sent or received by a ThreadedSocket. It contains a reference to the ThreadedSocket that sent the message and the message that it sent.

A ThreadedSocketHander is an interface that is implemented by the Client and Server or any other class that might contain a ThreadedSocket. This interface contains the methods onMessageSent(), onMessageReceived(), and onSocketClose(). This allows the class that contains a ThreadedSocket to individually control what happens when a ThreadedSocket receives or sends a message depending on what the container wants to do.

The Client class contains all the barebones logic that is required for the client to function. It handles what should happen when it receives a message from the server which is to check if it is a query. A query message is what the server or client sends to each other if it wants to inform the program not the user of something. On the client side, I have query messages it can receive be *%%kick* (if the client has been kicked) or *%%setname* (returned by the server if it successfully changed the clients name) If the message is not a query then it just prints the message.

The Server has many things it is doing Its first asynchronous task is to accept new clients. Upon accepting one, it creates a new ThreadedSocket for the client. Since all ThreadedSockets can be sent and receive messages on their own threads, no other threads need to be created. Upon receiving a message from a client, the server checks if the message is a command, query or just plain text. If it is just a message, it relays it to all other clients with the name of the sender prepended. If It is a query message (the server supports %%disconnect from the client which just informs the server that a client has disconnected) it handles that, if it is a command it then does anything necessary for the command. I currently added several commands which include setname (changes the clients name), m(send a private message to another user), online(returns a count of the number of users online), online -list (same as online but also prints all the users online), and help (prints a list of all these commands with description and syntax)

The ClientApp is the wrapper GUI for the Client. It uses JavaFX to create a TextField for showing the chat and another TextField for the user to type in. Upon hitting enter, assuming the user has entered text, it will send it to the server. To do this, the ClientApp contains a Client object that handles everything related to the actual communication. This ClientApp overrides the methods implemented from ThreadedSocketHandler. onMessageSent() is overridden to print the message sent in the TextField in addition to its super function which just sent the message and printed it to the console. onMessageReceived() is overridden to print the message to the TextField as well as super functions. onSocketClosed() closes all streams and stops all threads that the client may have started.

The ServerApp is much like the Client app in that it is very barebones as most of its functionality is encapsulated in a Server object. It has a TextField to display console output.

**To Run**

1. Have JRE installed
2. Launch ServerApp.jar
3. Launch as many clients (ClientApp.jar) as you wish (CPU load may be an issue)
4. Have JDK installed
5. Import Eclipse project
6. Run ServerApp.java
7. Run ClientApp.java for as many clients as you wish (CPU load may be an issue)

