Lab assignment

22 September 2021 08:05

Problem:

In this assignment, you need to code a **group chat room**. In this chat room, clients connect to a remote server with a designated IP and port number. The server accepts connections from an arbitrary number of clients. Any message sent from one client is broadcast to all other clients. You MUST leverage the concepts of socket programming and *multi-threading* to achieve this task.

NOTE: If you like (not mandatory), you can create a GUI for your chat room.

Solution:

Server:

1. In main(), Sever class is creating a ***server socket***
2. On initiateServer() it is accepting client requests until the serverSocket is closed (this is running on **the current thread**)
3. Once a request is accepted, ***a socket*** object is returned to connect with the client
4. Post that, it is creating a ClientHandler object which contains client's socket
5. Then it creates **a new Thread** using Runnable Interface implemented by ClientHandler and starts that thread (triggers the run())

ClientHandler:

1. On instatiation, clients socket, its outputStream and inputStream, and Username is getting assigned. On entering username, it is broadcasted to all the other clients
2. When run() is triggered by the server (threadObj.start()), **the socket starts to listen for new messages** from the client as long as it is connected. **On receiving input, it broadcasts that message**.
3. When the terminal closes and the socket is by default closed, catch block in run() closes input and output streams and the socket as well. It braodcasts the message to all the other users that the current client has left the chat. (closeEverything())

Client:

1. Post initiation of the server, the main() in client creates ***a new socket*** that connects to the server using ipAddress and port no.
2. A client object is created using client socket and username. It assigns socket, username, and input and output streams.
3. Via listenForMessage(), **a new thread** is created for client that **listens for incoming messages** from the Server (thru the connection made when the server accepted clients' request) as long as the socket is connected.
4. Via sendMessage(), **the current thread** of client waits for the input from the client console to **send the message** to server as long as the client is connected.
5. In case of any errors (termination of console), closeEverything() gets called and it closes current socket and outputStream and inputStream

Overall:

There are three types of sockets (when one client is connected)

1. A serverSocket for listening to client requests -> *server.java*
2. A client socket created from Server to communicate with the client (via accept()) -> *server.java*
3. A client socket to initiate a connection with the server -> *client.java*

There are three new threads (when one client is connected)

1. Main thread of server -> *server.java*
2. Main thread of Client -> *client.java*
3. New thread created by *server.java* for each client
4. New thread created by *Client.java* for listening to messages from Server

Common Pitfall: Calling run() Instead of start()

When creating and starting a thread a common mistake is to call the `run()` method of the `Thread` instead of `start()`, like this:

```
Thread newThread = new Thread(MyRunnable());
newThread.run();  //should be start();
```

At first you may not notice anything because the `Runnable`'s `run()` method is executed like you expected. However, it is NOT executed by the new thread you just created. Instead the `run()` method is executed by the thread that created the thread. In other words, the thread that executed the above two lines of code. To have the `run()` method of the `MyRunnable` instance called by the new created thread, `newThread`, you MUST call the `newThread.start()` method.