Java Lab 23: Sockets with Threads

This lab practices with sockets, clients, and servers, and threads.

1. Create a new Java project called Lab23. Download the file Lab23.zip. Unzip it and copy the two .java files into the src directory.

There are two main programs, one each in the two classes Client and Server. These are set up a little differently than the previous lab. The server code contains an infinite loop that doesn't read from the socket; instead, it delegates that job to the ServerWorker class, which you'll implement. The Client class is very similar to the previous lab, but uses DataInputStream and DataOutputStream for reading and writing from the socket. Client does not need any changes.

2. Create the ServerWorker class with the following:
- it should extend Thread
- it needs four data items: a DataInputStream, DataOutputStream, Socket, and Scanner

3. Add an overloaded constructor with one parameter, a Socket
- set this.socket to the parameter
- new up the DataInputStream and DataOutputStream by wrapping the socket's getInputStream( ) and getOutputStream( ), respectively; these require a try-catch block
- new up the Scanner to read from the keyboard
- call this.start( )

4. Create run( ), required by the Thread class.
- declare two Strings, message and reply
- display "Starting new connection"
- do a forever while loop. Inside it:
    - set message using DataInputStream.readUTF( ) – this returns a String
    - if the message is "QUIT", break the loop
    - print the message
    - prompt for a reply and get it using the keyboard scanner
    - write the reply using DataOutputStream.writeUTF( )

Catch two exceptions, EOFException and IOException
Close the socket; this also requires a try-catch block

5. Now run the server. Then run the client. Things should work the way they did in the last lab - ***except*** that the server should not quit when the client sends "QUIT" – it should only stop that thread. So start the client a second time and make sure it can still connect to the server. Then manually stop the server (red button).

6. Add a static int variable, clientCounter, set to zero, to ServerWorker's data. In its run( ) method, increment it as the first line. Change the first println to "Starting new connection for " + clientCounter. Change the message output and the prompt for a reply to print the clientCounter at the beginning of the line, like this:
1) Server received: <message>
1) Enter a reply: <reply>
Run the server again; run a client for a few messages then QUIT; run another client. Make sure the server output looks correct.

7. On a Mac, start a terminal window; if you're using Windows, start up a cmd window. Navigate to the folder containing this project, then down from there to /out/production/Lab23. Type ls (Mac) or dir (Windows), and you should see the .class files for this project. Then type "java Server". Nothing should display – yet. Then go

back to IntelliJ and run the Client. The client output should be inside IntelliJ, but the server output should be inside the terminal/cmd window. Type QUIT to quit the client, but type control-C inside the terminal/cmd window to quit the server.

8. This part is (slightly) lame, because a JavaFX program would be better for this. Open up two more terminal or cmd windows. Navigate to the .class folder again. In one window, type "java Server". In the second window, type "java Client", and in the third, type "java Client" again. Here's the lame part: the server output will now by interleaved. Test the clients, then quit. Again, kill the server with ctrl-C.