README

The project consists of two separate packages. One client package and one server package.

- The client package contains the ClientSide class.
- The server package contains the Server class, and the ServerThreader class.

Please see source code for additional comments on design and implementation

Instructions to run application (from terminal)

Please see demo videos included to fully clarify the points made below!

In the command prompt, for the flow to run smoothly please note the following:

- Ensure correct terminal is clicked with the cursor properly before typing message. If you don't click properly, it can cause serious errors with flow of dialogue.
- Only write one message per client at a time. (you can write >1, but can cause issues)
- Flow works most smoothly if e.g Client 1 -> Server : Server -> Client 1.

  Client 2 -> Server : Server -> Client 2

  Client 3 -> Server : Server -> Client 3

  Client 2 -> Server : Server -> Client 2 etc..

  i.e. flow -> 1,2,3,2,1,2,3 etc..

In the following scenario: (see video demo)

- If client 1, 2, & 3 each send a message to server, without server first responding, you must then input 3 messages to server.
- Messages are then all sent back to clients together (but can be out of order). ** In addition, the client who receives the most recent message from server must be the first to respond back for normal chat flow to function correctly.

Please note: because of this, the program works best with just one, or two clients. Multiple clients caused some console bugs that I could not resolve.

Instructions (See set-up demo video)

1) Classes have been compiled already in their respective folders
2) From src folder, type java server.Server --> runs Server program from server package
   Follow terminal instructions to assign port number
3) From src folder, type java client.ClientSide --> runs ClientSide program from client package
   Follow terminal instructions to assign port number.
4) You can now chat between client(s) and server
5) type \q to quit chat.

Additional credits

How does the client know what address to find the server on?

The program allows for a default port number to be selected (port 9999), or can be chosen by the user via the console. (see set-up demo).

What happens if the client can't reach the server when it starts up?

socket.connect() is surrounded by a try catch block. If no connection is established an error is thrown and the following is printed to console:

System.out.println("Failed to connect to server..try again");

System.out.println("Have you set up server port first?");

What happens if the client and server connect initially but the connection is lost during the chat session?

This is handled by a try catch block, where the readLine() method of the BufferedReader class checks for a null value.

```
while (true) {
   try {
        if ((sendMessage = readUserInput.readLine()) != null) <--- will break loop if null returned
        //.....remaining code..
           }
}
```

If null is returned and the loop broken, the following message is printed to the console:

System.out.println("Client: " + this.clientNumber + " has disconnected unexpectedly");

*Note: When multiple clients are connected, one client losing connection will cause the server program to terminate.

Additional advanced functionality

Multi-client server functionality implemented/attempted as described above

References

www.youtube.com. (n.d.). Multi-Client-Server Chatting || MultiUser Chat || Java Socket Programming - YouTube.

[online] Available at: https://www.youtube.com/watch?v=8lXI4YIIR9k

Network Technologies - Network Programming Week 11, Week 12, GMIT, Available at: login.microsoftonline.com. (n.d.). Sign in to your account. [online]

Available at: https://learnonline.gmit.ie/course/view.php?id=557

Rao, S.N. and Trainer, C. (2011). Chat Program two way communication Java. [online] Way2Java.

Available at: https://way2java.com/networking/chat-program-two-way-communication/ [Accessed 6 Jan. 2021].