ECE361 Lab Instruction 1:

Socket programming, Echo server/client, Chat server/client

Section 1:

In this section, you will write an *Echo client* and connect to the ECE361 Echo server.

Details:

1. Use `java.net.Socket` to connect to the server. (`Socket socket=new Socket (host, port);`).
   Ask for the server's `host address` and port from the lab instructor.

2. Upon successful connection, the server will send you a welcome message. Read the welcome message using `java.io.BufferedReader`. In order to do that you should pass the socket's `InputStream` to the buffered reader: `BufferedReader reader=new BufferedReader(new InputStreamReader (socket.getInputStream()))`.

3. You can read lines from the socket's input stream using `readLine()` method of the `bufferedReader`.

4. Read an input message from the standard input (`System.in`) and send it to the server. The echo server will send it back to you. Display the response from the server on the standard output. (`System.out`)

   In order to write to the output stream of the socket, you can use `java.io.DataOutputStream`. Pass the socket 's output stream to the DataOutputStream:
   `DataOutputStream writer=new DataOutputStream (socket.getOutputStream())`

   The method `writeBytes(String str)` can bye used to send strings on the output stream.

5. Start communicating to the Server by sending any message and reading it back via the writer and reader you defined.

6. You can quit the connection to the server by sending it the message "quit".

7. Close the socket at the end of the program (use `Socket.close()`).

Note that the server is expecting each message to end with a linefeed and a newline indicator. Therefore, you should send a linefeed+newline ("\r\n") at the end of each message.

Section 2:

Use the above echo client to measure the round trip time from your computer to the server. To do this, you can use the command “`System.currentTimeMillis()`” to measure the time with milliseconds accuracy. Such system is called a ping client. Measuring the round trip time is useful in some network protocols, as you will see throughout the course later.

Question: How long is the measured round trip time?

Section 3:

Implement the Echo server and connect your Echo client, to your own echo client.
The echo server is very similar to the echo client. The only differences are:

1. The server should wait listening on a network port until a client connects to it. For this purpose you can use (`java.net.ServerSocket`). A server socket can listen on a port via the `accept()` method. The `accept()` method will return a connected socket, that can be used for communication to the client.
2. The server, after sending a welcome message, should wait to receive a message and then send it back to the client (the order of sending and receiving is the reverse of what you did in the client).

Question: How long is the round trip time from your own server?

Section 4:

Write a chat server/client using the previously implemented client and server. A chat client/server is similar to an echo client/server, except that both the server and client can type in messages and send it to each other. In other words, in a chat application, the server will not echo the client’s messages back to him. Instead, the server will send its own messages to the client.

Section 5 (optional)

The above chat program that you implemented, forces the client and server to send messages every other time. In other words, the client and the server should take turns for sending and receiving. Such communication is called *Half duplex*. You can make sending and receiving *Full duplex* by implementing the sending part and receiving part of the application on separate threads. A thread (roughly speaking,) is a part of the application that can run independently and parallel to the other parts of the program. An application can have (practically) an arbitrary number of threads. (The previous application that you implemented runs on a single thread only.)

You can make your application to run on multiple threads using the `java.lang.Thread`. To run a new thread:

1. Implement a new class that `implements` the `Runnable` interface.
2. Pass the runnable class to the new thread. For example, if your runnable class is called `RunnableClass`, write :
   ```
   Thread thread=new Thread (new RunnableClass(socket))
   ```
3. To run the thread, simply use the command: `thread.start()`.
4. The class that implements Runnable should have a method with the name:
   `void run()`.
   Once the command `thread.start()` is called, `the run() method from the runnable` class is called to run on a new thread.

Note: Don't for to close the sockets at the end of the application.

Goodluck. ☺