**Lab 3 Out**

**ChatClient Implementation/Echo Server Implementation**

**Topic: Frameworks – Chapter 3**

**(50 Points)**

Lab3Out implements a system called an **Echo Server**. The concept of an Echo Server is the following:

* One Client Connects to the Server
* The Client Sends any message to the Server (in this case it’s in the form of a String)
* The Server will send the **exact** same message it received back to the Client

Lab 3out will utilize 3 classes implemented in the previous labs – *ChatServer*, *ServerGUI* and *ClientGUI*. You will complete the implementation of *ChatClient* that was started in lab3in. *ChatClient* must extend *AbstractClient*.

You will also need to perform some modifications to the *ChatServer*, *ServerGUI* and *ClientGUI* classes.

.

**ChatClient**

Implement a Java Class named *ChatClient* (*ChatClient* must extend *AbstractClient*) . Make sure that “Implement abstract methods” is selected when creating the *ChatClient* class. This class should implement the following private data fields:

**private** JLabel status;

**private** JTextArea serverMsg;

private JTextField clientID;

Where status is the topmost status JLabel of the GUI displayed by the ClientGUI class, serverMsg is the JTextArea corresponding to messages received from the ChatServer class, and clientID is the topmost, non-editable JTextField containing the clientID that eventually is assigned by the server.

The method description for *ChatClient* is given below:

public ChatClient()

Default Constructor for the *ChatClient*. Should invoke the Base class Constructor that accepts 2 parameters – *host* and *port*. You can just set the *host* and *port* to some default values in this method.

public void setStatus(JLabel status)

Sets the private data field status to the input parameter status. The input parameter corresponds to the status JLabel created in the *ClientGUI*.

public void setServerMsg(JTextArea serverMsg)

Sets the private data field serverMsg to the input parameter serverMsg. The input parameter corresponds to the JTextArea containing the server responses created in the *ClientGUI*.

public void setClientID(JTextField clientID)

Sets the private data field clientID to the input parameter clientID. The input parameter corresponds to the JTextField containing the client ID created in the *ClientGUI*.

public void connectionEstablished()

Hook method for client. Invoked whenever the client has successfully connected to the server. Set the status JLabel of the GUI to the word “Connected” and colors it Green within this method.

public void handleMessageFromServer(Object arg0)

Slot method for client. Invoked whenever the server has sent a message to the Client. Display the contents of parameter arg0 (**for this lab you can assume** arg0 **is always a String**) to the JTextArea of the GUI that contains the messages received from the Server. The serverMsg data field corresponds to where this message should be displayed.

The displayed message should be formatted as follows:

Server: *arg0*

Where *arg0* represents the String sent from the server. For instance, suppose the server sent the words “Hello Everyone” to the client. Then the client should display the following in the Server Response JTextArea of the ClientGUI class:

Server: Hello Everyone

**Note**: The server will be sending two types of messages back to the Client:

1. The server will send back exactly what the client has sent it (via client’s submit button)
2. The server will send back a key/value pair response expecting the Client to use the response in some way.

For this system, the second type of response involves the server sending the client a key/value pair that the client must parse and use in some way (similar to a command). The responses are in the form of key/value pairs formatted as follows:

*key*:*value*

Where the “:” separates the *key* from the value. There should be no white space in either the key or the value. Only one such response of this type will be utilized in this lab – it will have the following form:

username:*client-id#*

Where username is the key, while *client-id#* is the client id as assigned by the server (i.e., the client cannot choose their own id).

**The server should send this response back to the client as soon as the client has successfully connected to the server (performed within the** clientConnected **method of the server) .**

An example response of this form is shown below:

username: client-18

Where client-18 is the username for the client as assigned by the server.

Whenever the handleMessageFromServer method encounters this type of response, the method should parse the username from the value and then display the newly assigned client’s id in the 1st JTextField labeled as Client ID.

**This method should set the Client ID JTextField created within the ClientGUI class to the client-id sent from the server**.

**public** **void** connectionClosed()

Hook method. Displays the String “Not Connected” in the status JLabel colored Red.

**Modifications to ChatServer**

Modify the following methods:

public void handleMessageFromClient(Object arg0, ConnectionToClient arg1)

Display the message in the Server Log JTextArea using the following format:

“Client” *client-id* *message\_from\_client*

Where *client-id* is the id of the client retrieved by invoking method getId on parameter arg1 (e.g., arg1.getId()) and *message\_from\_client* is the String contained in parameter arg0.

Next, send the message contained in arg0 (assume arg0 is String for this lab) **exactly** back to the Client identified by arg1.

public void clientConnected(ConnectionToClient client)

Sends a response back to the client in the following String formatted as:

username:*client-id*

Where the username is considered a key, while the value associated with this key, *client-id,* is a String which is constructed as follows:

“Client-“ + client.getId()

Also display in the Server Log JTextArea “Client *client-id Connected”*

**Modification to ClientGUI**

You will need to add the following private data field to the *ClientGUI* class:

**private** ChatClient client;

This private data field should be instantiated with the Constructor of the *ClientGUI* class

The *ClientGUI* Constructor should set the host, *port#*, *status*, *serverMsg*, and *clientID* data fields of the client object.

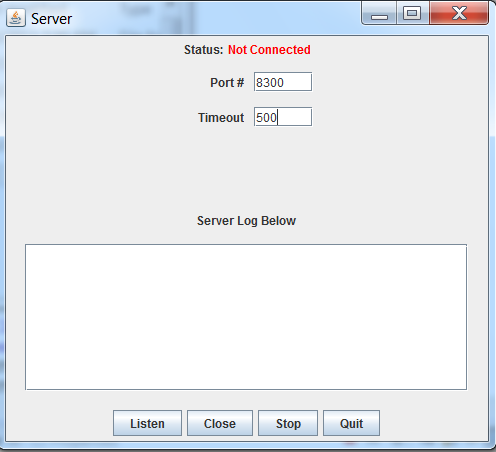
Pressing the “Connect” JButton will invoke the openConnection method of the client object.

Pressing the “Submit” JButton will send the message in the Client JTextArea to the Server

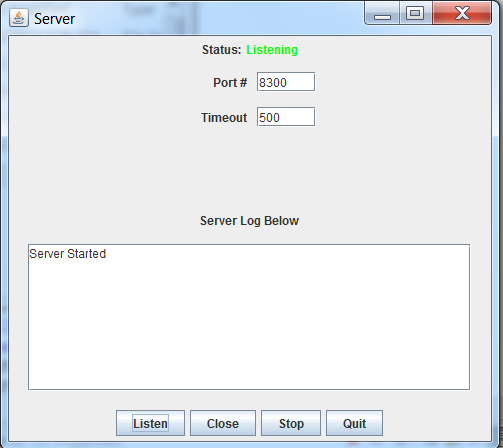
Pressing the “Stop” JButton will invoke the closeConnection method on the client object.

**Echo Server Process Sequence**

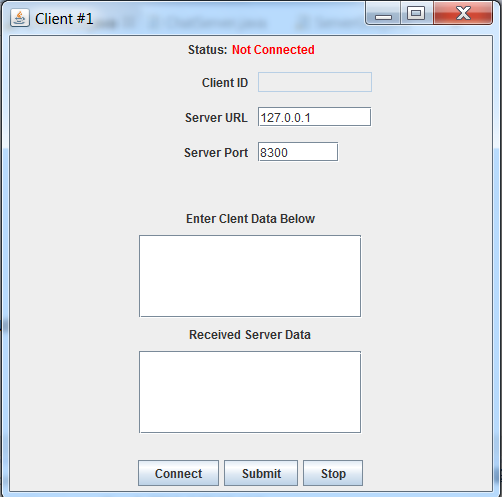
Start the ServerGUI class and fill in the GUI as follows:



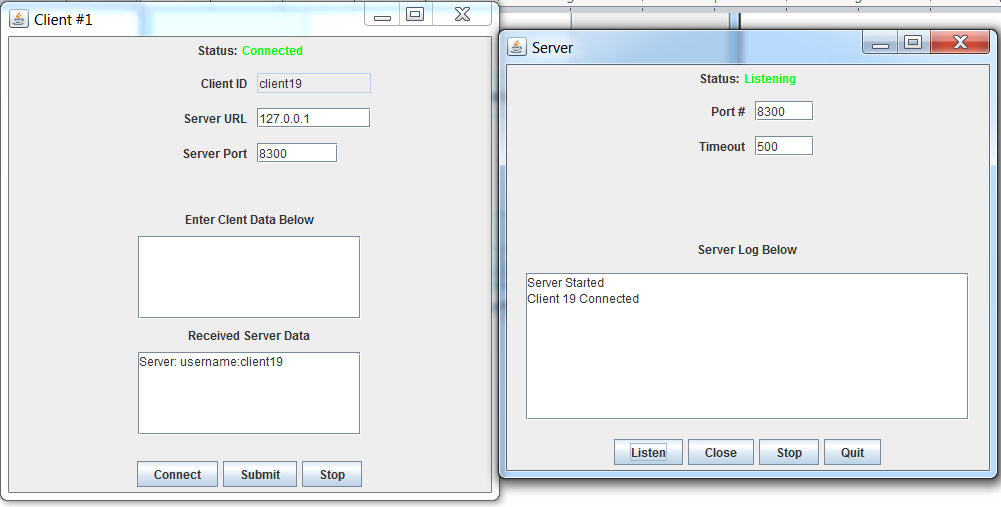
Press the “Listen” Button and the *ServerGUI* should be displayed as follows:



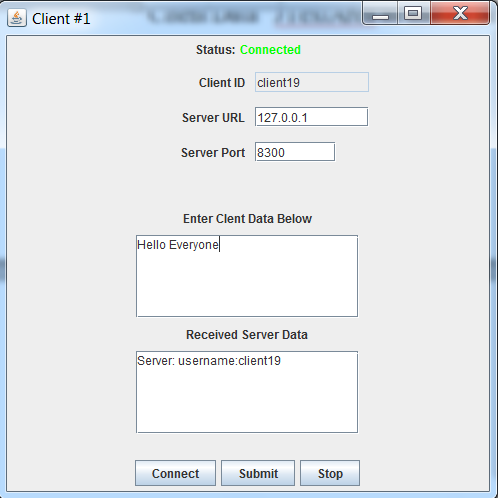
Next, the *ClientGUI* should be started and the JTextFields should be filled in as shown below:



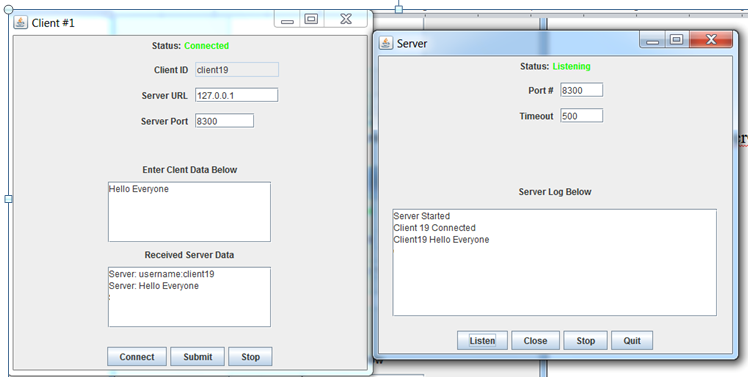
Pressing the “Connect” button results in the following modifications to the *ServerGUI* and *ClientGUI* **simultaneously**:



Note how the clientID is set automatically by the server. Next enter “Hello Everyone” in the “Client Data” JTextArea as shown below:



Pressing “Submit” results in the following modifications to both the ClientGUI and ServerGUI **simultaneously**:



Pressing the “Stop” JButton on the ClientGUI invokes the closeConnection method on the *ChatClient* object and displays “Not Connected” in the status JLabel colored Red. The user will have to press “Connect” JButton again before the client can reconnect to the server.

Pressing the “Close”, “Stop” and “Quit” JButtons on the ServerGUI results in the same functionality as in lab2out.

The ClientGUI should still receive the 1 command line argument as in lab1out – the title of application.

The ServerGUI shoud still receive the 1 command line argument as in lab1out and lab2out – the title of the application

**Required Package**

Both classes must be stored in lab3out package.

**Running from the Command Line Argument**

Run this lab using lab3outClient.bat in one command prompt and lab3outServer.bat in another command prompt before you submit. Be sure to place these bat files one directory above the lab3out package (usually this is in your Eclipse project directory).

**What you Submit**

**Z**ip up your entire package folder (lab3out) using a standard Windows Compression program. Do this by right-clicking on the lab3out folder and selecting “Send to Compressed (zipped) folder”