COMP3021 Java Programming
Programming Assignment #2
Implementation of an Online Chatting Room
Due Date: 11:59pm on May 7th

1 Prerequisites, Goals, and Outcomes

1.1 Prerequisites

For the second programming assignment, you need to know:

GUI programming knowledge

- To create user interfaces using frames, panels, and simple GUI components.
- To understand the role of layout managers and use the FlowLayout, GridLayout, and BorderLayout, etc. managers to lay out components in a container.
- To use JPanel to group components in a subcontainer.
- To create image icons using the ImageIcon class.
- To create and use buttons using the JButton class.
- To create and use check boxes using the JCheckBox class.
- To create and use labels using the JLabel class.
- To create and use text fields using the JTextField class.

Multithreading and Parallel Programming knowledge

- To create threads to run tasks using the Thread class.
- To control threads using the methods in the Thread class.

Networking in Java knowledge

- To create servers using server sockets and clients using client sockets.

- To implement Java networking programs using sockets.
- To develop an example of a client/server application.
- To develop servers for multiple clients.

1.2 Goals

Reinforce your ability to implement a Java Networking Application using the above knowledge.

1.3 Outcomes

You will have demonstrated the following "know how to" abilities:

- Implement a Java Networking Application which consists of two parts: chatting server and chatting client by using GUI programming.
- Using multi-threads technology to handle multiple clients connecting to the chatting server.
- To apply what you have learned to practice.

2 Description

Nowadays, we have various types of online chatting services to choose from, such as Tecent QQ, Whatsapp, WeChat, Line, etc (Fig. 1). Programming Assignment #2 asks you to implement an Online Chatting room system based on C/S mode. This system consists of two parts: Server part and Client part. Sever-Side is mainly responsible for listening messages sent from Client-Slides. And Client-Side allows you to logon first and then carry out the chatting process.

3 Requirements and Qualifications

The functions of Sever-Side should at least contain:

- Listen on a specific port, and wait for connections required from clients.
- Users can configure the listening port of Server-Side, while the default port is 8888.
- Send a system level message to online users who have connected to the Server-Side.
- Count the number of the current online users.

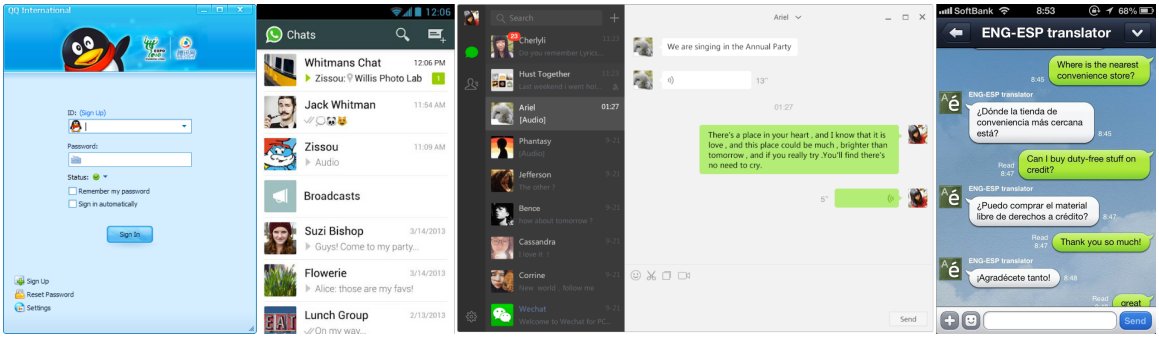

Figure 1: Various types of online chatting services.

- When stopping the chatting service, disconnect all users' connections.

The functions of Client-Side should at least contain:

- Connect to the Server-Side which has provided the chatting service sucessfully.
- Users can configure the IP address and port number of Server-Side to be connected.
- Users can configure the user name for displaying after connection.
- When the Server-Side is enabled, users can log on and log off at any time.
- Users can send messages to all users or a single person in a whisper.

4 System Overview

Here are some sample implementation screenshots of both Server-Side and Client-Side (Fig. 2). You can get a main idea but you should have your own implementation, making it personalization.

The main function framework is as follows (Fig. 3):

Server-Side:

- Port Configuration: Set the listening port. The default port is 8888.
- Start the Service: Enable the listening port and allow clients to connect.
- System Level Message: When starting the service, Server-Side can send notification in the chatting room.
- Stop the Service: Disable the listening port, and disconnect all the connections with clients.

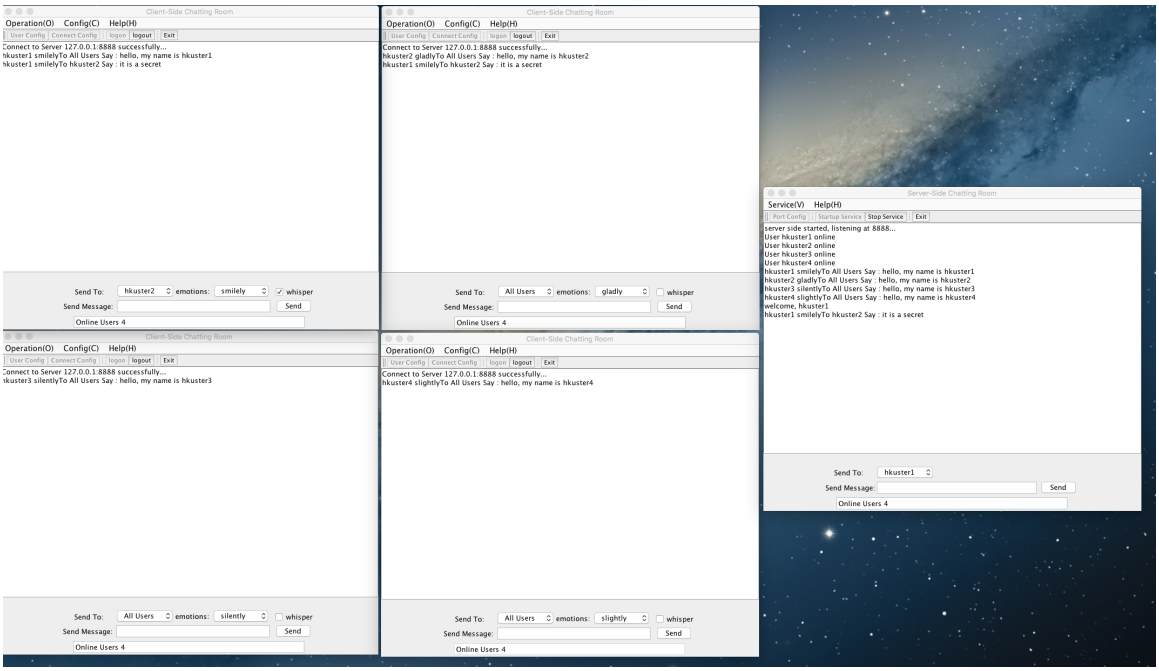

Figure 2: A sample implementation screenshot: In this screenshot, we have four online clients connecting to the server. They can chat in the chatting room or chat with an individual in a whisper way, which is invisible to other clients.

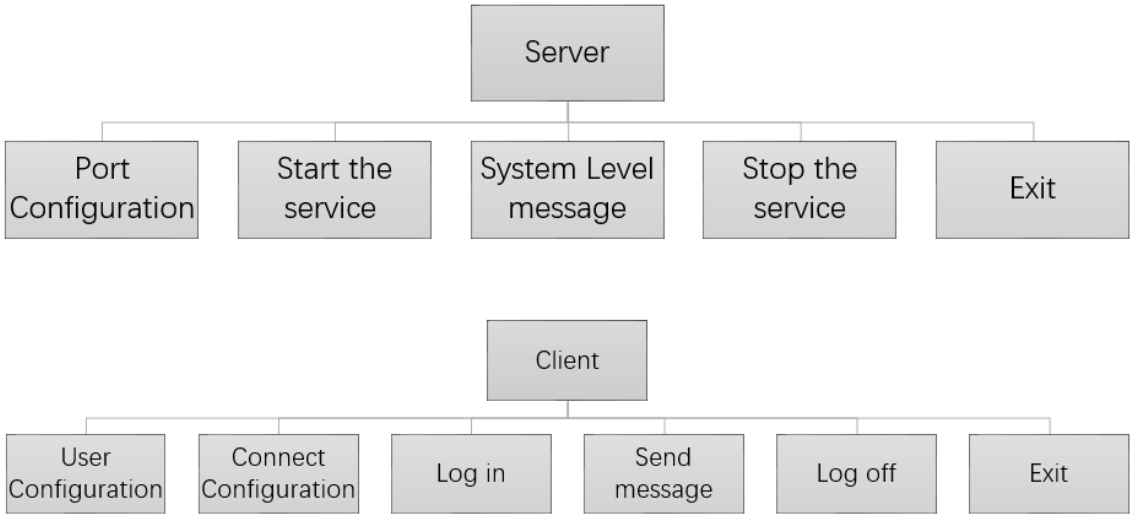

Figure 3: Framework of Server-Side and Client-Side.

- Exit: Shut down Server-Side.

Client-Side

- User Configuration: Set the user name used for chatting. The default name is "hkuster".
- Connection Configuration: Set the IP address and port of the Server-Side to connect.
- Log In: Connect to Server-Side.
- Send Message: Chatting or can send messages to any individual in the chatting room.
- Log Off: Disconnect with the Server-Side.
- Exit: Shut down Client-Side.

5 Implementation Guidance

In this section, we provide some detailed sample screenshots of both Server (Fig. 4) and Client (Fig. 5). These are the basic functions and you can do anything you like to expand, making it personalization.

5.1 Server Screenshots

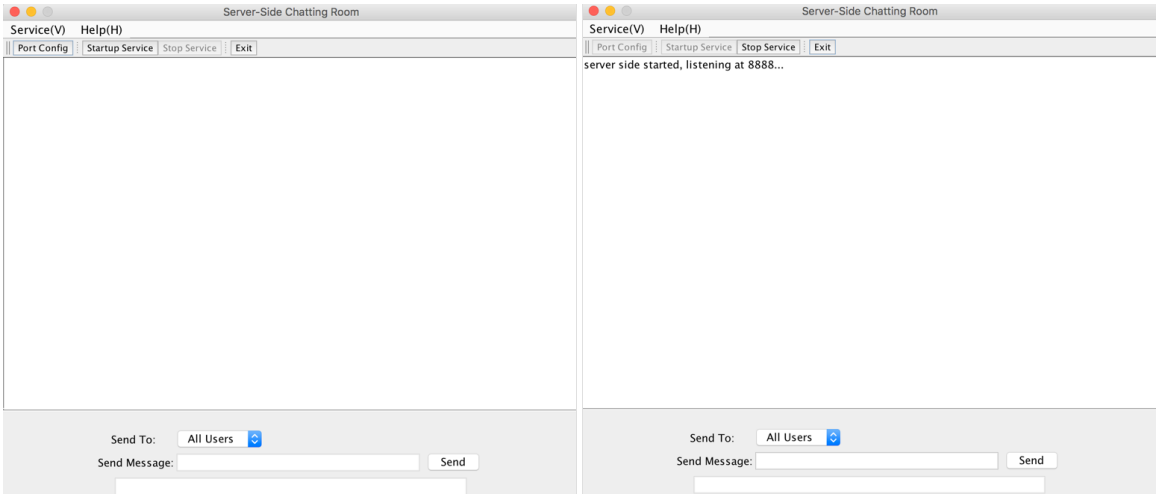

Figure 4: Server starts up successfully.

To implement the Server-Side, you should first construct a *ChatServer* class that extends *JFrame* and implements *ActionListener* interface. To support users to config the

server port and demonstrate how to use Server-Side, you should also provide two classes, named Class *Help* and Class *PortConfig* to accomplish these. Apart from this, Server should be able to listen to the actions of clients, such as online and offline, etc by extending from *Thread*. Server can also send and receive messages from all clients by extending from *Thread*. To manage and maintain all the clients, Server should implement a data structure (e.g. *LinkedList*, etc) to organize all clients.

5.2 Client Screenshots

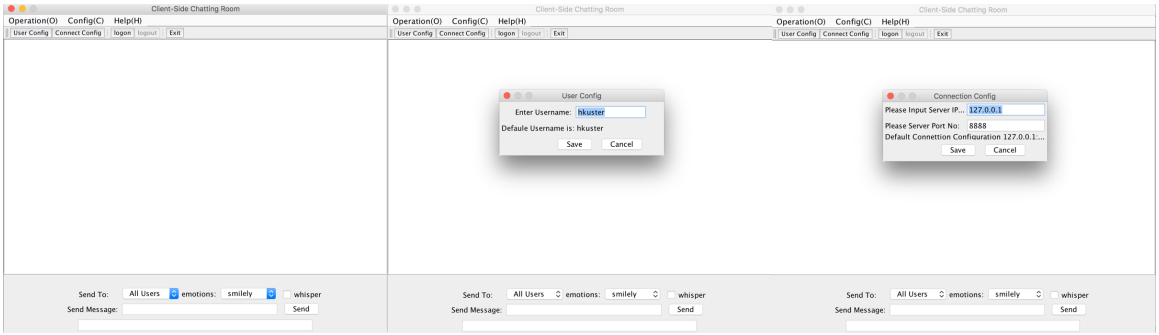

Figure 5: A client starts up and connects to Server successfully.

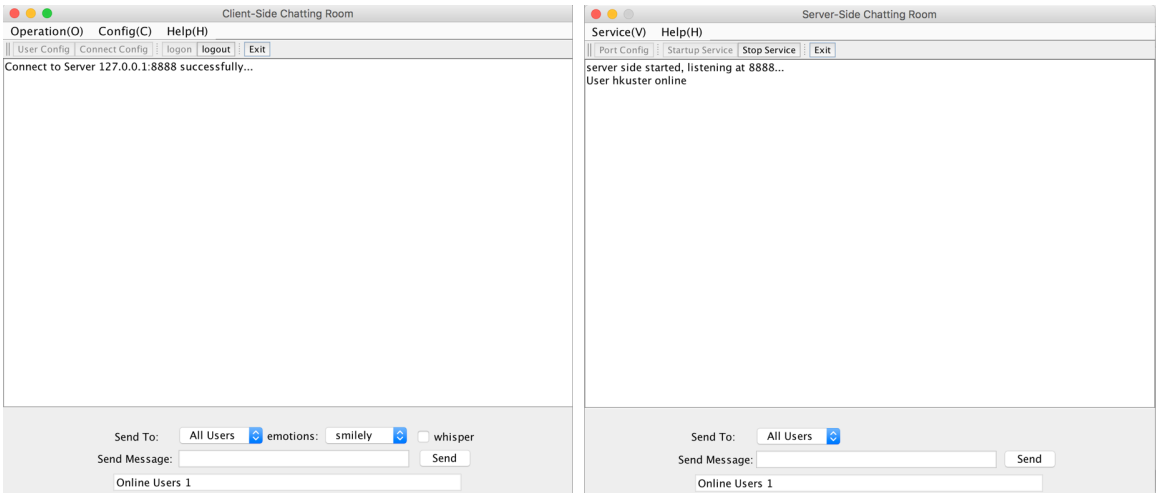

Figure 6: When a client connects to Server successfully, Server can detect and send system information showing that this client is online now.

To implement the Client-Side, you should first construct a *ChatClient* class that extends *JFrame* and implements *ActionListener* interface. To support users to config the server

port and IP address, as well as demonstrate how to use Client-Side, you should also provide two classes, named Class *Help* and Class *ConnectConfig* to accomplish these. The two classes are all extended from *JDialog*. Apart from this, Client supports users to modify their displaying user name, thus, you can implement a *UserConfig* Class which extends *JDialog* to support this. Client can send and receive messages. You should extend *Thread* to do this.

Again, these above are only some tips for you. You can do anything you like to accomplish these tasks and the grading will be based on your implementation.

6 Tasks

Implement both the Server-Side and Client-Side with a friendly GUI. This time you should also provide a document explaining your implementation process and functions of each module. Document your code using Javadoc and follow Sun's code conventions. Our advice: Test Often. Save Often.

7 Submission

Upon completion, submit the entire project files and pack it to .zip or other compressed format with your student ID number followed by "A2". For example, if your student ID number is 12345678, then you should name your zip file as "12345678A2.zip". We will test your work in Eclipse environment.

8 How to submit

We will use TurnItIn to collect and mark your assignment. We have created an account for each of you for you to submit your work at http://turnitin.com/en_us/home.

9 Due Time

11:59pm on May 7th. Late submission will be penalized by 20% per day. Good Luck!