CMSC204

Assignment #7

You will be designing a chat room service.

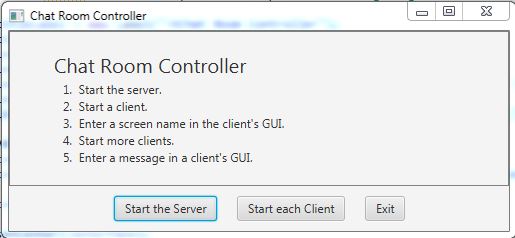
Your program will consist of three main parts:

1. A GUI Controller – this will have instructions, and buttons to start the server, to start each client, and to exit.
2. Client – The Chat application for each user. Each client will run in its own thread. Start this class after the Server is running.
3. Server – Manages the clients, checking their screen names and echo-printing their messages to the other clients. Start this class before the client.

Step1

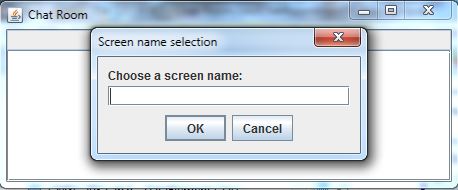
**Operation** (what the user sees):

When the application starts, the user is presented with a window that has a list of instructions and three buttons: “Start Server”, “Start Client”, and “Exit”.

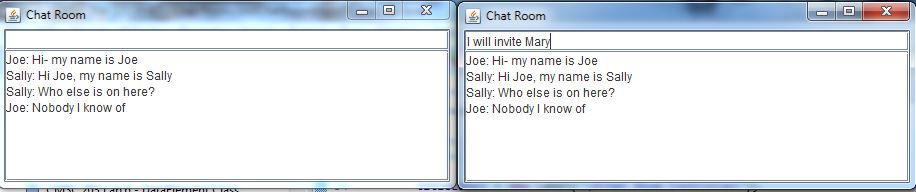


The user starts the server first. If they attempt to start a client before a server, or to start more than one server, an error message is given.

Then the user starts a client. The user is asked for a screen name. If the name is already in use in this session an error message is given the user is re-asked for a screen name, or if the user selects “Cancel”, the user is re-asked for a screen name. The user can start multiple clients.



When a user successfully enters a screen name, a window is shown where the user can type in a message. When the user enters a message, it is rebroadcast to all users.



**Specifications** (what the programmer does):

1. Create a GUI Controller: this will have instructions, and buttons to start the server, to start each client, and to exit.
   1. Instructions will be several labels
   2. Start Server Button:
      1. This button will have a mnemonic and a tooltip.
      2. When this button is selected, create a new ChatServerExec.
      3. Then call the ChatServerExec.startServer method with the port number as an argument if the server has not yet been created.
      4. The GUI controller will maintain a sentinel that specifies whether or not the server has been started yet, and display a JOptionPane message if it has.
   3. Start Client Button:
      1. This button will have a mnemonic and a tooltip.
      2. Each time the “Start Client” button is selected in the GUI, create a new ChatClientExec.
      3. Then call the ChatClientExec.startClient method with the port number as an argument
   4. Exit Button:
      1. This button will have a mnemonic and a tooltip.
      2. When selected, the server and any clients, along with the controller GUI, will close.
2. Client – The Chat application for each user. The client will consist of an executive (ChatClientExec) and ChatClient classes.
   1. Start ChatClientExec from the GUI after the Server is running.
   2. ChatClientExec will create a new ChatClient and run it in its own thread.
   3. The ChatClient process will create a new Stage, which will produce a separate GUI for each client.
   4. The user will be asked to enter a screen name
   5. When the screen name is accepted, the client’s chat textbox will be enabled. When the user types into the textbox, the client will transmit the message to the server.
   6. When the client receives messages from the server, it will be displayed in the client’s textarea.
3. Server – Manages the clients.
   1. Maintain a list of screen names in use in this session and check new names for duplication.
   2. Maintain a list of PrintWriter objects, and iterate through them to echo-print clients’ messages.
   3. Start this class before the client.
4. Assumptions:
   1. It is assumed that the GUI, the server, and clients will be run on the same computer.
   2. It is assumed that synchronization of objects with locks or conditions will not be needed.
5. Application Protocol. The Chat Protocol is as follows. (**Bold font** indicates strings sent between server and clients)

|  |  |  |
| --- | --- | --- |
| *Condition* | *Server Action* | *Client Action* |
| Client starts, server accepts client | Sends “**SUBMITNAME**” | Receives “**SUBMITNAME**” and asks user for screen name |
| Client has received “SUBMITNAME” | Server blocks waiting for input | Client sends **screen name** |
| Client has sent screen name | Receives client **screen name**, checks for duplicates, sends “**NAMEACCEPTED**" | Client blocks waiting for input |
| Server has sent “NAMEACCEPTED" | Server blocks waiting for input | Receives “**NAMEACCEPTED**", sets message text field to “editable” |
| Client has received “NAMEACCEPTED" | Server blocks waiting for input | Client types a message in text field, sends **message** |
| Client has sent message | Receives client **message**, sends “**MESSAGE *name: message***" to all clients | Client blocks waiting for input |
| Server has sent message to all clients | Server blocks waiting for input | Client receives “**MESSAGE *name: message***" and prints to text area. |

Deliverables:

Java files - The src folder with your client and server and test (.java) files

Javadoc files - The doc folder with your javadoc for student generated files

UML Class Diagram (an image, not the proprietary format, must be a .jpg or .pdf)

Deliverable format: The above deliverables will be packaged as follows. Two compressed files in the following formats:

LastNameFirstName\_AssignmentX\_**Complete**.zip [a compressed file containing the

following]

                        UML.jpg

Assignment 7 Checklist (filled in with YES or NO or ?)

                        doc [a directory] *please include the entire doc folder*

*generated files*

                                    file1.html (example)

                                    file2.html (example)

                        src [a directory] *contains your client and server (.java) files.  Include all*

*files that are needed to run your assignment, including those given to you*

                                    File1.java (example)

                                    File2.java (example)

                                    File\_Test.java (example)

LastNameFirstName\_AssignmentX\_**Moss**.zip [a compressed file containing only the

following]

*contains .java file which includes student generated client and server (.java) files – NO FOLDERS!!*

                        File1.java (example)

                        File2.java (example)

**Program Grade Sheet** Assignment #7 CMSC204

**Name \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Blackboard Date/Time: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

## **DOCUMENTATION** (20 pts)

## CheckList for Assignment 7 is included and completed 1 pt \_\_\_\_\_

## Javadoc generated for all student created classes: 4 pts \_\_\_\_\_

JUnit Test Class 6 pts \_\_\_\_\_

Create ChatServerExecTestSTUDENT

UML Diagram 4 pts \_\_\_\_\_

Lessons Learned 5 pt \_\_\_\_\_

In 3+ paragraphs, highlight your lessons learned and learning experience from working on this project.

How did you do? What have you learned? What did you struggle with? How will you approach your

next project differently?

## PROGRAMMING (80 pts)

Internal class documentation (within source code) 4 pts \_\_\_\_\_

Compiles and Runs without runtime errors 10 pts \_\_\_\_\_

Program user interface

Clear to user how data is to be entered 3 pts \_\_\_\_\_

Output is easy to understand 3 pts \_\_\_\_\_

Accuracy

Received correct output

Public tests – tests I gave you 6 pts \_\_\_\_\_

Your test – ChatServerExecTestSTUDENT 4 pts \_\_\_\_\_

Private tests 10 pts \_\_\_\_\_

Program Details

ChatClientExec Class 8 pts \_\_\_\_\_

1. Implements ChatClientExecInterface
2. Is called from the GUI when a client is created
3. Creates a new thread for each client

ChatClient Class 8 pts \_\_\_\_\_

1. Implements ChatClientInterface
2. Creates a window for user chat
3. Accepts/creates a socket for each client
4. Reads and writes messages to and from server according to the stated protocol

ChatServerExec Class 8 pts \_\_\_\_\_

1. Implements ChatServerExecInterface
2. Is called from the GUI once only when the server is created
3. Calls startServer, which runs ChatServer in a thread, in order not to block the GUI.
4. Asks for the client's Screen Name, checks it for duplication
5. Starts a new thread by which the server will interact with this client

ChatServer Class 8 pts \_\_\_\_\_

1. Runs in a separate thread for each client
2. Follows the stated protocol for receiving and sending messages to and from clients
3. Holds client screen names in a field HashSet<String> names
4. Holds client output streams in a field HashSet<PrintWriter> writers

Controller GUI 8 pts \_\_\_\_\_

1. Displays buttons which start the server and each client
2. Displays instructions to the user
3. Creates an instance of ChatServerExec and uses its methods
4. Creates an instance of ChatClientExec and uses its methods

Total 100 pts \_\_\_\_\_