**Project 7 / Chat Room Service**

You will be designing a chat room service. It entails three components:

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

Review provided sample test runs.

**Specifications**

* Client – The Chat application for each user. The client will consist of an executive (ChatClientExec) and ChatClient classes.
  1. ChatClientExec will implement ChatClientExecInterface (provided).
  2. Start ChatClientExec from the GUI after the Server is running.
  3. ChatClientExec will create a new ChatClient and run it in its own thread, then ChatClientExec will exit in order not to block the GUI ChatClientExec
  4. ChatClient will implement ChatClientInterface (provided).
  5. ChatClient will create a new Stage, which will produce a separate GUI for each client.
  6. The user will be asked to enter a screen name
  7. 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.
  8. When the client receives messages from the server, it will be displayed in the client’s text area.
* Server – Manages the clients.

1. ChatServerExec will implement ChatServerExecInterface (provided).
2. Start this class before the first client.
3. ChatServerExec will instantiate and run ChatServer in a thread, then exit in order not to block the GUI.
4. ChatServer will:
   1. maintain a list of screen names in use in this session
   2. check new client names for duplication.
   3. maintain a list of PrintWriter objects for each client
   4. Create a server-side thread to listen for each client’s messages
      1. create an instance of an inner class to run in the thread
      2. listen for messages from each client
      3. iterate through the PrintWriter objects to echo-print clients’ messages.

* 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.

**GUI**

* Utilize the provided UI, or create a GUI Controller of your own
  + It will have instructions, and buttons to start the server, to start each client, and to exit.
  + Start Server Button:
    - This button will have a mnemonic and a tooltip.
    - When this button is selected, create a new ChatServerExec.
    - Then call the ChatServerExec.startServer method with the port number as an argument if the server has not yet been created.
    - 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.
  + Start Client Button:
    - This button will have a mnemonic and a tooltip.
    - Each time the “Start Client” button is selected in the GUI, create a new ChatClientExec.
    - Then call the ChatClientExec.startClient method with the port number as an argument
  + Exit Button:
    - This button will have a mnemonic and a tooltip.
    - When selected, the server and any clients, along with the controller GUI, will close.

**The JUnit Test Class**

* Ensure that you are able to successfully test all provided Junit test cases. Include screen captures of successful runs

**Concepts Utilized in Project**

* Socket programming

**Write-up Requirements**

Be sure to review the provided rubric to understand project expectations, including the documentation, CMSC204, and programming requirements.

A write-up is required for each project. If the project consists of several applications, include several paragraphs to discuss each application in details. At a minimum, the write-up should address:

* Project Design / Algorithm / Pseudocode
  + Include a list of detailed steps to capture the design of the project (or applications)
  + Include UML diagrams of your Java classes
  + Complete this step first, and then write your code
* Test Plan & Test Cases
  + Which test cases did you test your program against?
  + Include screen shots showing most of your test runs – test cases of good and bad data points. (Convince yourself that your code works!)
  + Include JUnit test cases for all of your Java classes
  + I want to see your “thinking,” as to how you are testing your program
  + In theory, each submitted project should be a “rock solid” working program with “near zero” bugs
* Document any assumptions that you are making about your project
* In 3 or more paragraphs, highlights of your learning experience and lessons learned
  + I am very interested to learn more as to what you have done, how you did it, etc.
  + **I make it a point to read your write-up first, before looking at your code**
* Anything else that you want to share with the instructor
* Only one write-up is needed for the entire project

**Submission Requirements**

Each student must submit one compressed (.zip) file to the Assignment (link,) and the submission must include the following project deliverables:

* Source code – one or more project folder(s) that you have created in Eclipse
  + The entire “working” project folder (so that your instructor can compile and test your project
  + One folder per program (or application,) if there are multiple programs being specified in the project
  + I MUST BE able to compile and test every submission
  + Ensure that correct source files are submitted, or I won’t be able to grade your project
* Project Write-up (in a Word document)
* Name the compressed file (zip format) as <lastname>\_poject\_x
  + where x is the project number and your last name (e.g. Thai\_Project\_1.zip)
* Review provided instructions on how to submit your project, if needed
* Double check your submission! I can only grade what you have submitted
* Not clear or need additional clarifications? That’s okay, but you are expected to ask questions or seek help from your instructor or other means
* “I did not know …” or “I did not understand …” is not good enough.

**Programming Principles and Good Programming Practices –**

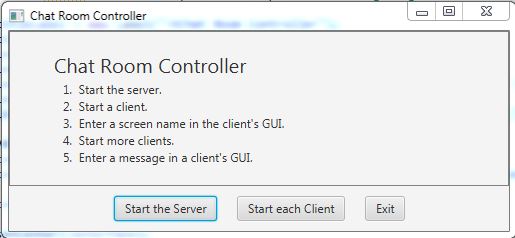
* The computer does exactly what it is told to do
* Think before you code
* Start early!
* You may discuss the project with others, but the actual deliverable must be created and implement by you!

**Academic Honesty Policy (Reminder)**

* It’s simple; don’t cheat!
* Each submitted project will be compared against other submissions from current and previous semesters
* Violation will result in failure of the course and further disciplinary action.

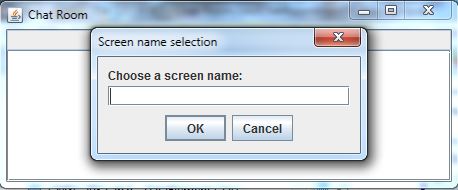
**Sample Test Runs & Outputs**

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.

