Title
Sequential Chatting

Lecture Topics Emphasized
Networking
Monitors
Locks and Conditions

Introduction
Chatting is a very common application. We wrote a chat server and chat client in class, but we are going to add some features to the chat application to force specific rules to be followed. We will make sure that clients communicate in a specific order. Multiple messages can be sent from a user, but the messages must be delivered in a given user order.

Description
For this lab, you will modify the ChatServer and ChatClient programs that are posted for lecture #19 (or feel free to write them from scratch if you would like). Currently, a client can send a message to the server at any time, and the server broadcasts the message back out to all the clients who are currently connected. This allows one client to send multiple messages before receiving a response from any of the other clients.

We want clients to broadcast messages based on a specific order. The order will be determined based on the order the clients have connected to the server. Let’s take an example here. Assume four clients have connected to the server: client 1, client 2, client 3, and client 4. During client 1’s turn, he will have a chance to send as many messages as he wants until he sends an “END_OF_MESSAGE” to indicate he no longer wishes to send messages. The “END_OF_MESSAGE” should not be broadcasted to other clients. During client 1’s turn, only he can send messages. But, as soon as he client 1 finished sending messages, it will be client 2’s turn and only client 2 can send messages during that time. This continues in a round-robin fashion until all the clients had a chance to send messages. It will then become client 1’s turn again.

Please note, although the problem can be solved in multiple ways, you are required to use locks and conditions in this lab.

Start with creating a standalone Java project. Download the three java files (ChatRoom.java, ChatClient.java, ServerThread.java) from lecture #19 and copy them into your project.

To make sure only one client is sending messages, we will need to have a lock and condition for each connected client. In your chatroom class, add two new vectors as instance variables, one
for locks and the other for conditions. Make sure you instantiate the vectors in your constructor. Now, when the server accepts a new socket connection, create a new lock and a new condition for that lock. Add them to their vectors respectively. Note that a server thread’s lock and condition will have the same index in their respective vector.

The server thread will need to acquire the lock and wait on the condition. So, add two more parameters in server thread’s constructor and pass the lock and condition. In the run method, instead of infinitely looping and reading messages from the client, we only do so if it is the client’s turn to send messages. We can achieve that by awaiting on the condition. But in order to be able to wait on the condition, the thread will need to acquire the lock first. So, in the while loop, try to acquire the lock first and then wait on the condition. Then, when the condition is signaled, proceed with reading all the messages sent from the client until you receive an “END_OF_MESSAGE”. A while loop can help you to achieve the above. Make sure you send a message to the client to let them know that it is their turn to send messages. Similarly, when an “END_OF_MESSAGE” is received, inform the client that they will need to wait until their turn to send again.

When the server thread gets the “END_OF_MESSAGE”, it means it is the next client’s turn. The server thread will need to inform the server to signal the next client. To do that, create a method in the chat room server, which will be used to call the signal method on the next client’s condition. Remember, before you can signal, the thread will need to acquire the condition’s lock first. Be sure to release the lock after signaling the next client. In your server thread, you can then call the server’s signal method to alert the next client.

Now, the implementation above almost works but has a small problem – who will signal the first client? To deal with this, you can add an additional parameter in the server thread’s constructor and let the thread know if it is the first thread. If it is the first thread, then it’s not necessary to ask the thread to wait on the condition, since the client will be the first to send messages.

The client code should not be affected. If it is not the client’s turn to broadcast and the client sends a message to the server, the server will simply hold onto the message (there is nothing you will need to do, as all the messages will just be stored in the buffered reader).
Grading Criteria

Labs are graded based on your understanding of the course material. To receive full credit, you will need to 1) complete the lab following the instructions above AND 2) show your understanding of the lab material by answering questions upon check-off.

If there is a discrepancy between your understanding of the material and your implementation (i.e. if your code is someone else’s work), you will receive a grade of 0 for the lab. Please note, it is the professor’s discretion to report the incident to SJACS.

Instructors, to ensure consistency across all lab sections, please strictly stick to the following criteria:

1) *ChatRoom Constructor*
   a) 0.10% - the constructor is complete and working, following the instructions above
   b) 0.05% - the constructor is complete (following the instructions above), but has bugs
   c) 0% - the constructor is incomplete (no partial credits)

2) *ServerThread Constructor*
   a) 0.05% - the constructor takes the required parameters
   b) 0% - the constructor is missing parameters (no partial credits)

3) *ServerThread* Run Method: **no partial credit will be given**
   a) 0.30% - the run method is complete and does the followings: acquires lock, awaits on condition, skips awaiting on condition for the first client, alerts client for their turn and when their turn ends, detects ”END_OF_MESSAGE”, and informs server when client is done
   b) 0.25% - the run method properly acquires lock, awaits on condition, and skips awaiting on condition for the first client, alerts client for their turn and when their turn ends, and detects ”END_OF_MESSAGE”
   c) 0.20% - the run method properly acquires lock, awaits on condition, and skips awaiting on condition for the first client, and alerts client for their turn and when their turn ends
   d) 0.15% - the run method properly acquires lock, awaits on condition, and skips *awaiting on condition for the first client*
   e) 0.10% - the run method properly *acquires lock and awaits on condition*
   f) 0% – the run method does not acquire lock or await on its condition

4) *ChatRoom* Signal Method
   a) 0.15% - the Signal method correctly identifies the next client, successfully acquires lock, and signals the appropriate thread
   b) 0.10% - the Signal method correctly identifies the next client, and successfully acquires lock
   c) 0.05% - the Signal method correctly identifies the next client
   c) 0% – the Signal method is unable to identify the next client (no partial credits)

5) Check-off Questions: Please randomly select three questions from the list on the next page. Each question is worth 0.07%, and the section is worth 0.20% total.
a) Why do we need to create a new lock and condition for each of the server threads?

b) What are the parameters that get passed into the server thread? Why are they necessary?

c) Explain when and where do we signal the next client? How does this ensure that only one client is broadcasting?

d) What do we need to do first before we can signal a condition?

e) Why don't we want the first client to wait on its condition? What happens if we do make the first client wait on the condition?

f) What happens if a client types a message into their console when it's not their turn?

g) How do we make sure we are only broadcasting one client's messages at a time?

h) Is it necessary to have a lock for each server thread? Can we only have one central lock and instantiate a new condition for each thread on that lock?