Lab #6
CSCI 201

Title
Message Queue

Lecture Topics Emphasized
Threads

Introduction
Multi-threading is a very important task that is involved in nearly every program you run. The ability to have multiple sections of code appear to be executing simultaneously has enabled applications such as auto-save, gaming, message notification, and AJAX. You will get some experience using multiple threads to implement a message queue.

Description
You will create a message queue class that allows messages to be stored from multiple threads simultaneously. Messages queues are often used for notification-based applications. For example, one thread can put a message into the queue while another thread can "subscribe" to receive notifications. Each thread will operate independently of the other, sharing the message queue.

Create three classes for this program – `MessageQueue`, `Subscriber`, `Messenger`. The Subscriber and Messenger classes should be separate threads that share an instance of `MessageQueue`. The `MessageQueue` should contain a data structure that allows inserting and removing (`ArrayList`, `Vector`, `Stack`, `Queue`, etc. are good ones).

The `Messenger` should be inside a loop that iterates 20 times and inserts a different message into the `MessageQueue`. The message can be whatever you want, but make sure you include a unique identifier with each message (such as the message number being inserted). After inserting each message, the `Messenger` should sleep for a random amount of time between 0-1 seconds. After each message is inserted into the `MessageQueue`, output the message and a date/time stamp to the console. Make sure to distinguish this output from the `Subscriber` output.

The `Subscriber` thread should query the `MessageQueue` by calling a method in the `MessageQueue`. It should continue to query until it has read 20 messages. The `Subscriber` thread should sleep for a random amount of time between 0-1 seconds after attempting to read a message. If there is no message, do not increment the number of read messages. That will ensure that 20 messages will eventually be read before terminating. Output each message to the console after it has been read, along with a date/time stamp. If there is no message to be read, output that as well.

Your program should not have any exceptions thrown. Since multi-threading does not always generate the same output, you will need to run your program multiple times. To make this easier, create another class named `MessageTest` that contains a main method. This class will run the above program 10 times, and it will also be responsible for starting the `Subscriber` and `Messenger` within each iteration of the loop. Make sure these threads are started properly so that they run concurrently.

Below is one possible output of your program, though there are many variations that would still be correct.

```
2017-07-17 7:13:23.03 Messenger – insert “message #1”
2017-07-17 7:13:23.09 Messenger – insert “message #2”
2017-07-17 7:13:23.40 Subscriber – read “message #1”
2017-07-17 7:13:24.00 Subscriber – read “message #2”
2017-07-17 7:13:24.30 Subscriber – tried to read but no message...
2017-07-17 7:13:24.51 Messenger – insert “message #3”
2017-07-17 7:13:24.58 Subscriber – read “message #3”
2017-07-17 7:13:25.03 Messenger – insert “message #4”
2017-07-17 7:13:25.10 Messenger – insert “message #5”
2017-07-17 7:13:25.15 Subscriber – read “message #4”
2017-07-17 7:13:25.29 Messenger – insert “message #6”
2017-07-17 7:13:25.50 Subscriber – read “message #5”
2017-07-17 7:13:26.11 Subscriber – read “message #6”
2017-07-17 7:13:26.30 Subscriber – tried to read but no message...

<program continues>
```

Grading Criteria

Labs are not graded based on any given criteria but are instead graded on effort and attendance. If you arrived to lab within the first 10 minutes and worked on it the for the entire duration of the lab, you will receive full credit regardless of whether you completed it. TAs will not grade labs until after at least half the lab period has elapsed. Use the lab time as an opportunity to more fully understand the course material and ask your TA questions.