CIS 212 Assignment 7: 100 (regular) points + 100 extra points

Note: The regular 100 points can be partial, as usual; the extra 100 credits are atomic (i.e., either "0" or "100"), and cannot be partial. In order to have your work on the extra points taken into account, you need to work on the regular points first; only working on the extra part will lead to the 0 grade for this entire assignment.

The goal of this assignment is to gain experience working with concurrent programming via multiple threads and thread synchronization. You will write an implementation that simulates producer and consumer threads. The consumer will take more time to complete than the producer, causing the machine to run out of memory if threads are not synchronized in such a way that no more than a specified number of units are produced prior to being consumed.

1. [10] Create a new class with a public static void main method that creates a java.util.concurrent.LinkedBlockingQueue with enough capacity for 100 String entries.

2. [20] Create a new class which implements Runnable to simulate the producer. When executed, a total of 1000 random Strings (e.g., using Math.random() and Double.toString()) are added to the above queue. You need to wait until there is space in the queue if necessary (i.e., using the LinkedBlockingQueue<T> put() method to add Strings). Print the progress (i.e., the total number of Strings added so far) once every 100 Strings added.

3. [40] Create another new class which implements Runnable to simulate the consumer. When executed, this thread consumes Strings from the queue. Consumption continues as long as there are Strings in the queue or the producer hasn't finished (i.e., the consumer doesn't quit if the queue happens to be empty before the producer finishes). Use the Thread sleep() method to wait up to 10 milliseconds (i.e., a random number of milliseconds on the range [0, 10)) before each String consumption to ensure that the consumer takes longer time to execute than the producer. Print the progress once every 100 Strings consumed.

You need to implement your program in such a way that it supports one producer and an arbitrary number of consumers. The consumers are expected to consume at different rates, so consumers should not simply consume a predetermined number of Strings and then quit; one consumer may complete significantly earlier than the others and would be idle waiting for the others in this case. As such, you will want a way for your producer to signal its completion to all the consumers (e.g., creating a Boolean variable shared by the producer and all the consumers); you will also need to ensure that you don't have a consumer potentially waiting indefinitely for a String to be produced (e.g., using LinkedBlockingQueue<T> poll() method with a timeout).

4. [20] Execute your program with 1 producer and at least 2 consumers concurrently. Your program needs to automatically quit after all 1000 Strings are produced and consumed. Before

quitting, print the summarized information about how many Strings the producer produces and how many Strings each consumer consumes exactly.

5. [10] Write code that is clear and efficient. Specifically, your code should be indented with respect to code blocks, avoid unnecessarily repetitive code, avoid code that is commented out or otherwise unused, use descriptive and consistent class/method/variable names, etc.

The following is an example output for 1 producer and 4 consumers, using the LinkedBlockingQueue<T> implementation:

```
"Producer 1": 100 events produced

"Producer 1": 200 events produced

"Producer 1": 300 events produced

"Producer 1": 400 events produced

"Consumer 1": 100 events consumed

"Consumer 2": 100 events consumed

"Producer 1": 500 events produced

"Consumer 4": 100 events consumed

"Consumer 3": 100 events consumed

"Producer 1": 600 events produced

"Producer 1": 700 events produced

"Producer 1": 800 events produced

"Consumer 1": 200 events consumed

"Consumer 3": 200 events consumed

"Producer 1": 900 events produced

"Consumer 4": 200 events consumed

"Consumer 2": 200 events consumed

"Producer 1": 1000 events produced

Summary:

"Producer 1" produces 1000 events.

"Consumer 1" consumes 252 events.

"Consumer 2" consumes 246 events.
```

```
"Consumer 3" consumes 255 events.

"Consumer 4" consumes 247 events.
```

6. [+100] (Extra credits) Implement all of the above for an arbitrary number of producers and consumers using Java's default LinkedList<T>, rather than LinkedBlockingQueue<T>. Note the following in your implementation:

- LinkedList<T> is not thread-safe, so you need to make sure every time there is only one thread, either a producer or a consumer, exclusively adds or removes a String to or from the list. The total number of Strings added by all producers is exactly 1000.
- LinkedList<T> has no maximum capacity, so you need to find a way to enforce the capacity of 100, so that if this capacity is reached then the producers need to wait (i.e., using the Object wait() method) before adding more Strings.
- If the LinkedList<T> is empty (i.e., nothing to consume for the consumer) but the total added Strings has not reached 1000, then you need to wake up (i.e., using the Object notify()) the producers to let them add more Strings.
- All other requirements are the same as in the LinkedBlockingQueue<T> implementation. Sleep before every String consumption. Print the progress once every 100 Strings are added or consumed, and print a summary information about the exact number of Strings added and consumed by each producer or consumer. Execute your program with at least 2 producers and at least 2 consumers concurrently. Program automatically quits when all production and consumption are done.

The following is an example output for 4 producers and 6 consumers, using the LinkedList<T> implementation:

```
"Producer 3": 100 events produced

"Producer 4": 100 events produced

"Producer 2": 100 events produced

"Producer 1": 100 events produced

"Consumer 2": 100 events consumed

"Consumer 5": 100 events consumed

"Consumer 6": 100 events consumed

"Consumer 3": 100 events consumed

"Consumer 4": 100 events consumed

"Consumer 1": 100 events consumed
```

```
"Producer 2": 200 events produced

"Producer 4": 200 events produced

"Producer 3": 200 events produced

"Producer 1": 200 events produced

Summary:

"Producer 1" produces 248 events.

"Producer 2" produces 253 events.

"Producer 3" produces 250 events.

"Producer 4" produces 249 events.

"Consumer 1" consumes 146 events.

"Consumer 2" consumes 178 events.

"Consumer 3" consumes 175 events.

"Consumer 4" consumes 167 events.

"Consumer 5" consumes 162 events.

"Consumer 6" consumes 172 events.
```

Note: If you choose not to work on the extra points, please use one folder, put your Java source file(s), i.e., .java file(s), into that folder, rename that folder as [YourName]Assignment7Regular, e.g., BillGatesAssignment7Regular, and then zip the folder into a single zipped file. If you choose to work on the extra points as well, in addition to the folder mentioned above, please have another folder, put your Java source file(s), i.e., .java file(s), on the extra points into that folder, rename that folder as [YourName]Assignment7Extra, e.g., BillGatesAssignment7Extra, and then select both folders and zip both of them altogether into a single zipped file. Finally, you rename your zipped file as [YourName]Assignment7.zip, e.g., BillGatesAssignment7.zip, and then upload it to Canvas. Please do not use the “package” keyword in any of your source file.