420/620 Parallel Programming
Activity 3: Java Blocking Queue

Rosalyn Shin

1. When many producer threads are waiting on a full queue, what happens when a thread `takes` a String and calls `notifyAll()`? Why is this thread safe?

   `notifyAll()` wakes all currently waiting threads, then one of those threads (most of times the one with highest priority) will come in and block the read&wirte from other threads. This is thread safe because of the condition when `notifyAll()` is called. `notifyAll()` is called only when the queue is full or 0 (1 for our implementation), thus, there can no threads waiting to either `put` or `take` string.

2. Read the documentation for synchronized methods. Do synchronized methods synchronize on the object or the class? Why is this the right scope? (Consider that there may be multiple queues in an application.)

   The synchronized methods synchronize on the object. By doing so, we are preventing any read&write from other threads that share the same object while any changes to the object is visible by all threads. Also, making a class synchronized does not make sense because the specific class should be only visible to the thread.

3. One might want the `String` copy operations to proceed in parallel because they are expensive (see below). This is not thread safe. Why? Give an example.

   This is because the copy operations that are outside of synchronized block are not 'visible' to other threads. In other words, other threads can read&write on the queue before the operation is completed. For example, shown in Table 1, first the `Producer 1` reads `head` and `qlen` and update `qlen`. Then Consumer 1 is notified because `qlen==1`. `Consumer 1` comes in and read `head`. In Step 3, the consumer thread continues reading an item at `head`. However, the item has yet been added to the index. In our code, the code doesn't crash, but consumer takes null. However, in actual codes, the program may crash for attempting to reading non-existing data.

Table 1: Unsafe thread example

Step	var	Producer 1	Consumer 1
1	head=0 qlen=0	read head read qlen qlen++	* notified by notifyAll()
2			read head (because qlen became 1)
3			read an item at head
4		(add item at head)	