The goal of this assignment is to gain experience working with concurrent programming via multiple threads and thread synchronization. You'll write an implementation which simulates producer and consumer processes. The simulated production procedure will take far less time to complete than the consummation procedure, therefore causing the machine to run out of memory if the threads aren’t synchronized such that no more than a specified number of units are produced prior to being consumed.

1. [20] Create a new Eclipse project namedAssignment7. In your main method or Main class constructor, create a java.util.concurrent.LinkedBlockingQueue which will hold 100,000 UUID entries (yes, UUIDs again…).

2. [30] Create a class which implements Runnable to simulate the producer. When executed, this process should add 2,000,000 random UUIDs to the above queue (i.e., using the LinkedBlockingQueue put() method), calling Thread.yield() between each addition (we’ll discuss this in class). Print your progress once every 1000 additions (e.g., “1000 UUIDs produced“). Note that creating and adding 2,000,000 UUIDs to a simple LinkedList (or ArrayList, etc) should cause Java to run out of memory (at least it does on my machine)!

3. [30] Create another class which implements Runnable to simulate the consumer. When executed, this process should find and print the max UUID String (i.e., using LinkedBlockingQueue take() method to get each UUID and the UUID toString().compareTo() method for the comparisons) of the 2,000,000 random UUIDs above, calling Thread.sleep(1) between each comparison (i.e., to simulate a more complicated process). Do not use the UUID compareTo() method directly because this method does not compare based on the String representation of the UUID, meaning that you wouldn’t get the max String value. Print your progress once every 1000 additions (e.g., “1000 UUIDs consumed“). This process is going to take some time to complete, so I recommend starting with a smaller queue size and number of UUIDs for initial testing.

4. [20] Use a java.util.concurrent.ExecutorService to execute your producer and consumer concurrently. Shut down the service after starting the two processes.

5. [+20] (Extra credit) Use a synchronized LinkedList in part 1 rather than a LinkedBlockingQueue (hint: see the Collections.synchronizedList method). You’ll then need to an the Object wait method in part 2 to halt the producer process and an Object notify or notifyAll method in part 3 to resume the producer.

Zip the Assignment7 folder in your Eclipse workspace directory and upload the .zip file to Blackboard (see Assignment 7 assignment in the Course Documents area).