Programming Homework Assignment #2 (updated 04-25-2017)

Due Date: Sunday May 7, 11:55 PM

- Upload the source files (.java files) with output (copied and pasted to the end of the main file) (see Catalyst under Week 4 for where to submit)

Problem:

Write a Java program which completes and tests the LinkedQueue class as declared in the LinkedQueue.java file (in the Queue Code File folder on Catalyst under Week 2). CHANGE the LinkedQueue file so that it reads:

```
public class LinkedQueue<T extends DeepCloneable<T>>
    implements QueueInterface<T>, DeepCloneable<LinkedQueue<T>>
```

The methods that need to be completed in LinkedQueue.java are:

- default constructor (does nothing)
- deepClone method that returns another LinkedQueue<T> (MUST make a DEEP COPY*) (note: expect T to do a deep clone, too)
- enqueue
- dequeue
- isEmpty
- size
- clear() no return value (public method that empties out the LinkedQueue)
- toString() (public method that returns a String that has all the data in the Queue separated by a space, BUT NOT CHANGING ANYTHING in the Queue) THIS ISN'T IN THE LinkedQueue class yet, so you MUST ADD it!!!!

* A "deep copy" or deep clone copies the data elements (T) of the queue, NOT have the cloned queue refer to the same data elements, but to clones (copies) of the data elements

HINTS: The Java code for these methods will be VERY SIMILAR to the LinkedStack class, with a few changes, as mentioned in class, AND don't forget:

- enqueue adds to the back of the queue, BUT you must check if it was empty (if it was, also assign the frontNode, but it not, link the new node to the old back node)
- dequeue adds removes from the front of the queue, but it becomes empty after removing it, you need to update the backNode (you should figure out to what!)
- in the toString() method (which overrides the Object toString()), DON'T call dequeue nor enqueue and DON'T change anything in the Queue, but traverse similar to the LList.java code

Run a bank queue simulation, simplified from other simulations so it has no priority queue and only 1 teller, AND you are to use a

subclass of the Simulation abstract class given in the HW2_JavaCodeFile. Other simplifications include not displaying/writing anything until the teller is starting or finishing with a customer.

In main, declare Scanner and SingleQueueSimulator variables. Then do the following:

- instantiate a default SingleQueueSimulator and assign to the main variable
- open an input file (Scanner) using the same openInputFile from HW#1, and assign to the Scanner variable declared in main
- LOOP while the Scanner you just assigned isn't null
    - call the setInputFile mutator for the SingleQueueSimulator (passing the Scanner from openInputFile)
    - call runSimulation() for the SingleQueueSimulator you created
    - AGAIN, call the same openInputFile from HW#1, and assign to the Scanner variable declared in main
    - (end of loop)
- After the loop, call the cloneTester() method given in the HW2_JavaCodeFile

To stop the loop, enter an empty String for the filename. See test runs on Catalyst. Test your programs using the same test input files to see if you get the same results. Turn in with the output using both of the test input files (one at a time), and then the HW2 Input.txt file.

More hints will be given in class (SEE the Class Notes in Weeks 3 &4).

DON'T USE RECURSION.

Extra Credit Problems (due the last day of the quarter!): (Data Structures and Abstractions with Java by Carrano) 4th Ed. p.328 #2 . OR re-do Prog.HW#1 to use an ArrayQueue<String> instead of a ArrayList<String> for the tokenized postix expression.