CS 1316 – Homework 8 – Quack / Ring Buffer

Due: Friday October 30th, 2009 before 6pm.

Out of 100 points

This is an individual assignment. You may collaborate with other students in the class but your solutions must be your own. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. You should not exchange code or write code for others. Collaboration at a reasonable level will not result in substantially similar code.

Homework 8: Implementing a Quack using a RingBuffer

Part 1:

For this assignment, you will be completing several methods in the (currently incomplete) RingQuack.java class file. Specifically, you will fill in the implementation of:

- void enqueue(E element)
- void push(E element)
- E pop()
- E dequeue()

The specifications and behavior for each method in this assignment are located in the Javadoc comments inside the incomplete RingQuack.java class file. There are several completed methods in addition to the ones that you need to implement. It is your task to implement each of the missing methods.

While implementing the above methods, you may not use any methods from java.util or any other Java libraries.

NOTE: The Quack.java interface is using Java Generics, so that your Quack can contain objects of any type. The main() method in the RingQuack.java file instantiates an object of type RingQuack that can hold Strings, but your code should work with any objects stored within the list.

WARNING: If you are implementing your code correctly, you will have to cast Objects stored in the array to type E (the generic type). This will cause a compiler warning. Although you do NOT have to suppress the compiler warnings, you should understand

why they are appearing. The peek() method has an example of how to suppress the compiler warning if you wish.

There is a main method in the RingQuack.java file that you can use to test your code. As you work on each method, you can test it by commenting out sections of the main method that use the methods you have not yet written. Note that the provided RingQuack.java file uses an array size of 3 for testing. Your code must work for ANY non-zero array size, using the MAX_SIZE constant instead of hard coding the number 3.

Part 2:

Download the PerformanceTest.java class file. It instantiates an OurLinkedList and a RingQuack (using the code you have written for this and the last homework). It then pushes two strings onto each data structure, and then pops them off. It repeats this 80 MILLION times! It also prints out the date/time when it starts and stops. Examine the code so you understand that it is doing the same operations for each implementation of the stack (It is only using the stack part of the Quack).
Write a one to two paragraph report telling how much time each of your data structure implementations took, and explaining any differences. Submit this as a text or pdf file to T-Square, along with your RingQueue.java file.

Grading Criteria 100 points total

Correct operation:

- enqueue(item) ……..…….……. 25 pts
  - Checks for full array 5pts
  - Updates numStored 5pts
  - Updates tail index 5pts
  - ....corrects for wraparound 5pts
  - Places item in store 5 pts
- push(item) …..……... 25 pts
  - Checks for full array 5pts
  - Updates numStored 5pts
  - Updates head index 5pts
  - ....corrects for wraparound 5pts
  - Places item in store 5 pts
- pop( ) & dequeue() ……...……...… 20 pts
  - both metods return the correct data 5 pts
  - both methods update the head index correctly 5 pts
  - ...corrects for wraparound 5 pts
  - Both have the exact same behavior. 5 pts
- Performance Result Writeup..... 30 pts
  - Includes timings for both implementations 10 pts
  - Explains why they are the same or different 20 pts