Computer Science 112: Lab4 Queue Interface Implementations

Work with a Partner ("Mandatory" – I really think you will get more out of this lab if you work with a partner, but I also realize that with timezone issues, internet issues and issues related to the pacing of the course, this may be impractical for some of us. Please work with a partner if you are starting this in lab.) Due Jan 11

In parts 1+2 please turn in only one program per partnership/group

Part 1.

Download the java files from Moodle.

You will finish the implementation of the queue in CS112QueueB. One of you should control the code. Exchange contact information so that you can finish later if you need to. **There will not be a new lab on Monday, so you can work together then as well.**

This is a version of the second implementation of a queue we discussed in class, but it is missing the contents of two methods:

a. **shiftBack()** which should move the elements of the queue so that the front is in location 0. It is used when the queue is not full, but **ONLY when it** has bumped into the end of the array. It is not used with every remove.

b. **writeQueue** should print the elements of the queue, from front to back. (Note that there is already a method called writeArray, but it writes the whole array, not just the queue. **writeQueue** should print only as many elements as there are on the array, in order from front to back.

Note that this version has instance fields count, front and back. Be sure you understand how they work before you try to change the methods.

Modify UseQueue to better test the implementation.

Submit CS112QueueB only to Lab4A Be sure all names are in CS112QueueB.java

Part 2.

Modify your queue so that it implements a wraparound queue – when you get to the end of the array, you wrap around to the beginning (unless the queue is full, in which case you expand).

For example, in a four element wraparound queue:

add 1	1 x x x
add 2	1 2 x x
add 3	1 2 3 x
add4	1 2 3 4

remove (1)	x 2 3 4
add 5	5 2 3 4
add 6	2 3 4 5 6 x x x or 5 6 x x x 2 3 4
add 7	2 3 4 5 6 7 x x or 5 6 7 x x 2 3 4

You will need to change many of methods. Some may no longer be needed. Remember % is the mod (remainder) operator. You will need to check for wraparound EVERY time an index is incremented.

Submit this to **Lab4B** Again, be sure all names are in CS112QueueB.java

At this point, you will need to copy the Queue class from one partner to the other (or e-mail it, etc.)

Part 3: The Augustine Problem (Again)

Run all partners' versions of the Augustine Problem with your new Queue implementation. If any of them does not still work the same way it did before, you (as a group) probably have a problem with your queue implementation.

You should do this to create and use the new Queue implementation:

CS112QueueInterface myQ = **new** CS112QueueB();

(in place of

CS112QueueInterface myQ = **new** CS112Queue();
)

Because myQ is an interface, you cannot use the non-interface methods (e.g. writeQueue). But you didn't use them in your Augustine problem.

If you find a problem, fix it and resubmit CS112QueueB.java to CS112Lab4B

EACH PERSON

submit your CS112QueueB.java and your Augustine program to **Lab4C**