Programming Laboratory 9
CSCI 1913: Introduction to Algorithms,
Data Structures, and Program Development
November 5–6, 2019

0. Introduction.

Two queue classes were discussed in the lectures. One implemented a queue using a linear linked list of nodes. The other implemented a queue using a circular array. For this lab assignment, you must implement an iterator for the queue class that uses a circular array.

1. Theory.

Suppose that we want to visit the elements stored in a sequence, like a stack or a queue. Also suppose that we don't want to modify the sequence as we visit its elements. Then we can visit them by using an *iterator*. An iterator is class whose instances can visit the elements of a sequence. Each iterator has a method called `hasNext` that tests if there are more elements to be visited. It has a method called `next` that returns the next element to be visited, and advances to the following element. It also has a method called `remove` that is supposed to remove the element that is being visited, but we won't use `remove` here.

We can simplify an iterator's design by assuming that the sequence won't change while we visit its elements. For example, if we use an iterator to visit the elements of a stack, then we assume that the stack will not be `push`'ed or `pop`'ped. Similarly, if we use an iterator to visit the elements of a queue, then we assume that the queue will not be `enqueue`'d or `dequeue`'d. If a sequence changes while an iterator visits its elements, then the actions of the iterator become *undefined*—which means they don't have to work correctly.

2. Implementation.

First, you must place the following line at the top of your file.

```
import java.util.Iterator;
```

This will define the interface `Iterator` as discussed in lectures. You must use Java's `Iterator` interface in your code. You must not write your own `Iterator` interface. **IF YOU DO NOT MEET THESE REQUIREMENTS, THEN YOU WILL RECEIVE ZERO POINTS FOR THIS ASSIGNMENT.**

You must then add the following members to the class **`ArrayQueue`**, whose Java source code is available on Canvas. These members implement an iterator for `ArrayQueue`. You are not allowed to modify `ArrayQueue` except to add these additional members.

```
private class ArrayQueueIterator implements Iterator<Base>
```

> This class must be nested inside `ArrayQueue`. An instance of this class may be used to visit the current elements of an instance of `ArrayQueue`. It must have one or more private variables that let it "know" which elements of `ArrayQueue` are to be visited next. You must decide what those private variables are.

```
private ArrayQueueIterator(...)
```

> This is `ArrayQueueIterator`'s constructor. Of course it must be inside `ArrayQueueIterator`. It must set the `ArrayQueueIterator`'s private variables to the values of its parameters. You must decide what these parameters are.

```
public boolean hasNext()
```

> This method must be inside `ArrayQueueIterator`. It must return `true` if there are more elements of `ArrayQueue` that remain to be visited. It must return `false` otherwise. This method must use `ArrayQueueIterator`'s private variables only. Hint: use ideas from `ArrayQueue`'s method `isEmpty`.

```
public Iterator<Base> iterator()
```

> This method must be inside `ArrayQueue`. It must make a new instance of `ArrayQueueIterator` and return it.

```
public Base next()
```

> This method must be inside `ArrayQueueIterator`. It must return the next `Base` element to be visited from `ArrayQueue`. If no more elements remain to be visited, then it must throw an `IllegalStateException`. This method must use `ArrayQueueIterator`'s private variables only. Hint: use ideas from `ArrayQueue`'s method `dequeue`.

```
public void remove()
```

> This method must be inside `ArrayQueueIterator`. However, it must do *nothing*—it's there only because the `Iterator` interface makes us define it. The method `remove` is supposed to remove the element currently being visited by the `ArrayQueueIterator`, but that doesn't make sense for a queue, where elements can be deleted only from the front.

Be careful to put these members in the right places. For example, the class `ArrayQueueIterator` must be nested inside the class `ArrayQueue`, and the method `next` must5 be inside the class `ArrayQueueIterator`, metc. If anything is not in its right place, then the iterator will not work.

All of `ArrayQueueIterator`'s methods must work in $O(1)$ time. That means you are not allowed to copy `bases` from `ArrayQueue` into a similar array inside `ArrayQueueIterator`. **IF YOU DO NOT MEET THIS REQUIREMENT, THEN YOU WILL RECEIVE ZERO POINTS**

FOR THIS ASSIGNMENT.

3. Deliverables.

The file **`tests9.java`** contains Java code that performs a series of tests. The tests call methods from the `ArrayQueue` and `ArrayQueueIterator` classes. Each test is followed by a comment that tells how many points it is worth, and what must be printed if it works correctly.

Run the tests, then turn in the Java source code for the modified `ArrayQueue` class, with your `ArrayQueueIterator` class in it. If you don't know how to turn in your work, then ask your lab TA's. Your work must be submitted by **11:55 PM** on **Wednesday, November 13, 2019**.