1 Introduction

You now have some experience with the implementations of Collections. In this assignment, you will develop your own generic collection type. While lists, stacks, queues, sets, etc. are the most common interfaces for data access with a variety of backing implementations, sometimes we’d like to define some more restrictive behavior of them to support other algorithms.

One important case is limited-size collections. In this assignment, we’ll explore an implementation of a queue that only allows a fixed number of elements. If this capacity has been reached when a new element is enqueued, the first item in the queue is necessarily dequeued to maintain the size limit. For instance, if we have a fixed length of 3, we can add up to 3 elements before items are forcibly evicted from the queue. Consider the following example using this length with a queue of characters. The front of the queue is at the right, while the rear is toward the left.

<table>
<thead>
<tr>
<th>Added</th>
<th>Queue</th>
<th>Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>R F</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>K R F</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>S K R F</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>L S K R</td>
<td></td>
</tr>
<tr>
<td>W</td>
<td>W L S K</td>
<td></td>
</tr>
</tbody>
</table>

2 Assignment

Your task is to implement an array-backed version of such a fixed-length queue. This means your class should use an array of a constant size and use wrap-around indexing as items are added to the queue.

A generic interface for `FixedLengthQueue` for you to implement is provided in

`~weinman/courses/CSC207/code/f1queue/FixedLengthQueue.java`

Indices

Your implementation should explicitly track only the size and the index of the rear item in the queue.

Calculating the Front

Add a utility/helper object method `getFront` that calculates and returns the index of the front item in the queue.
Implementing iterator

To return an iterator for the object, you must implement an inner class. The iterator should return items from the front of the queue to the rear. You do not need to support remove; it should throw an UnsupportedOperationException. You should make sure that the iterator’s queue does not change while it is being iterated. If it does, hasNext and next should throw a ConcurrentModificationException.

Implementing equals

Your implementation should override Object’s equals method. Two fixed-length queues are considered equal if they have the same length, the same size, and the objects in the queues at all corresponding logical locations (e.g., front to rear) are considered equal.

Implementing toString

Your implementation should override Object’s toString method. Since repeatedly appending String objects is inefficient in Java, you should use a StringBuilder object. The order should be from (L-R) front to rear. Like Java’s collections, the values should be comma-separated and enclosed in square brackets. For example, the output for the queue in the last line of the example above should be

\[S, L, W]\n
since S is the (least-recently added) front of the queue and W is at the rear.