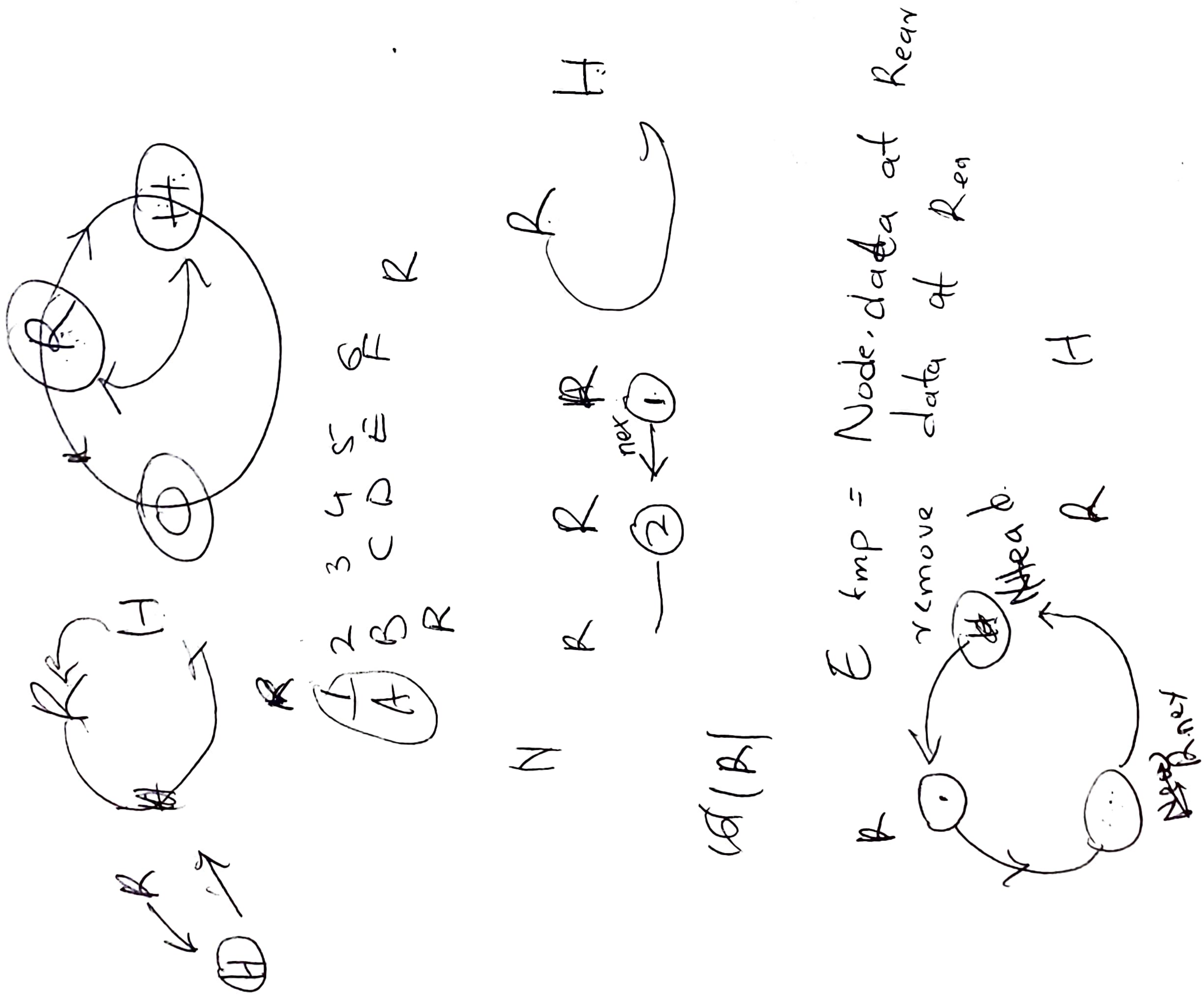
CSC 1351-3, Spring 2020, Lab 7   Name this project CircularQueue

Generic Circular Queue with Singly-Linked Nodes and One External Pointer

A queue is a FIFO (first in first out) abstract data type (ADT) in which nodes are added at the rear of the queue and removed at the head of the queue. There are several ways of implementing a queue. In lecture class we presented a generic straight-line queue with singly-linked nodes (each node has one pointer, named next, that points to the next node that was added to the queue) and two external pointers, named head and rear, that point to the head node and rear node of the queue.

In today's lab you will implement a circular variant of the generic straight-line queue presented in class. In this circular variant, each node has one pointer, named next, that points to the next node that was added to the queue (as in the one we did in class), but the rear node's pointer, instead of being null, points to the head node, and the queue has only one external pointer, named rear, that points to the rear node (there is no external pointer to the head node). Thus in this variant, the head node can only be accessed using the rear node's pointer. This circular queue is illustrated on the following page:



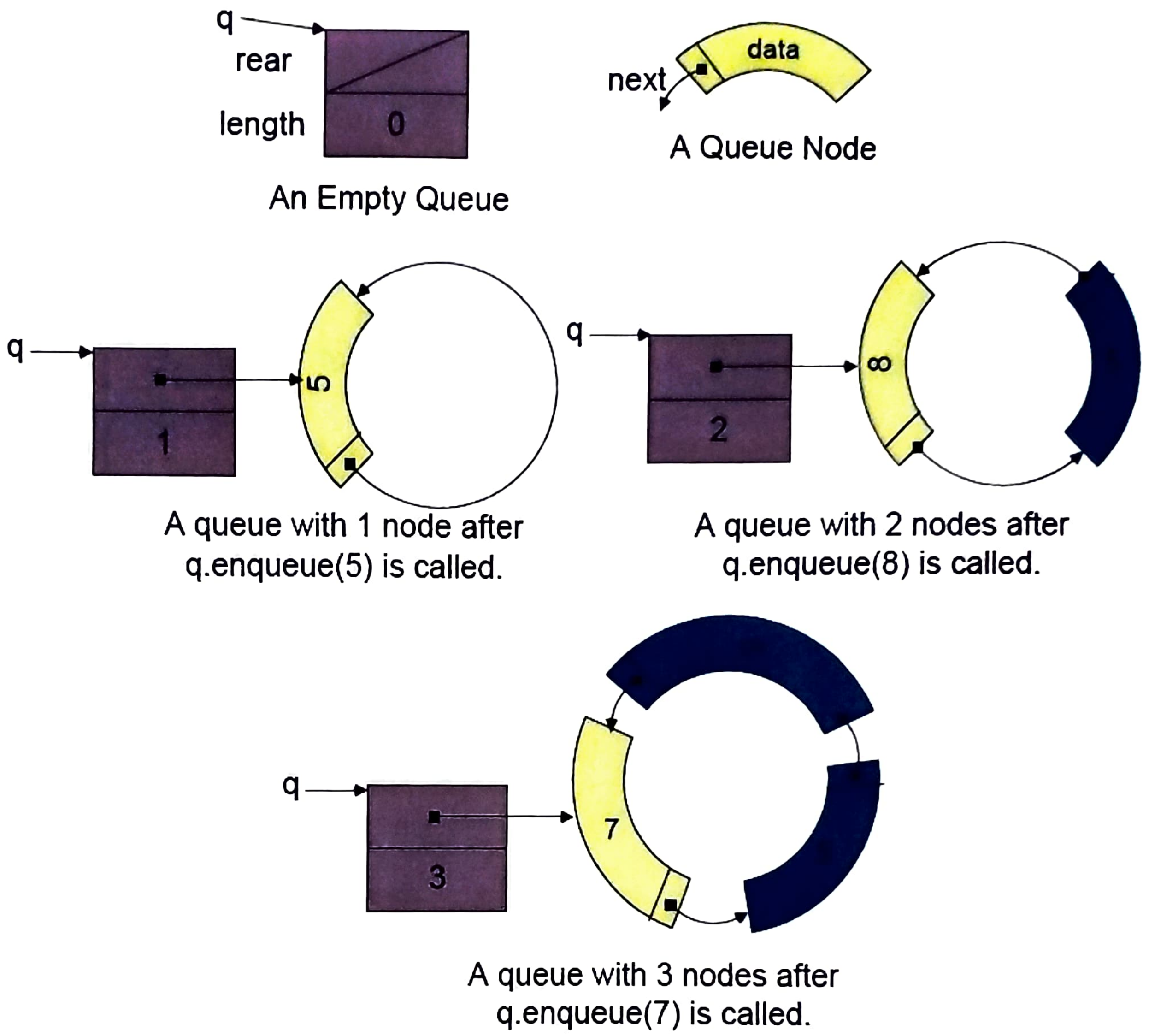

q
rear
length
0
An Empty Queue
data
next
A Queue Node
q
5
1
A queue with 1 node after q.enqueue(5) is called.
q
8
2
A queue with 2 nodes after q.enqueue(8) is called.
q
7
3
A queue with 3 nodes after q.enqueue(7) is called.

Figure 1: Illustration of a Queue in various states

This project will have the following interface and classes:

public interface CircularQueueInterface<E>

This interface is provided in the starter code.

public class CircularQueue<E> implements CircularQueueInterface<E>

Complete the implementation of the constructor and methods in the starter code.

public class CircularQueueDemo

This class is provided in the starter code.

Output

```
S inserted in the queue.
T inserted in the queue.
R inserted in the queue.
E inserted in the queue.
S inserted in the queue.
S inserted in the queue.
E inserted in the queue.
D inserted in the queue.

Queue = [D, E, S, S, E, R, T, S]

The queue is not empty.

After rotating counterclockwise, queue = [E, S, S, E, R, T, S, D]
After rotating clockwise, queue = [D, E, S, S, E, R, T, S]

S is at the front of the queue.
S has been removed from the queue.
Queue = [D, E, S, S, E, R, T]

T is at the front of the queue.
T has been removed from the queue.
Queue = [D, E, S, S, E, R]

R is at the front of the queue.
R has been removed from the queue.
Queue = [D, E, S, S, E]

E is at the front of the queue.
E has been removed from the queue.
Queue = [D, E, S, S]

S is at the front of the queue.
S has been removed from the queue.
Queue = [D, E, S]

S is at the front of the queue.
S has been removed from the queue.
Queue = [D, E]

E is at the front of the queue.
E has been removed from the queue.
Queue = [D]

D is at the front of the queue.
D has been removed from the queue.
Queue = []

The queue is empty.

Calling nums.front()
java.lang.Exception: Non-empty queue expected
```