CS 0445 Spring 2026
Recitation Exercise 2

[Note: Be sure you have viewed the Lecture 4 Powerpoint presentation prior to doing this exercise. Some important definitions and ideas are discussed in that lecture]

Introduction:

In Lecture 4 we briefly discussed the ideas of a queue and a deque (double ended queue) and their operations. Be sure to review Lecture 4 before beginning this exercise.

Recall the specifications for a queue from the textbook author. These were discussed in Lecture 4 and demonstrated in handout QueueDequeTest.java.

```
public interface QueueInterface<T>
{
  /** Adds a new entry to the back of this queue.
      @param newEntry  An object to be added. */
  public void enqueue(T newEntry);

  /** Removes and returns the entry at the front of this queue.
      @return  the object at the front of the queue or throw
      EmptyQueueException if the queue is empty. */
  public T dequeue();

  /**  Retrieves the entry at the front of this queue.
      @return  The object at the front of the queue or throw
      EmptyQueueException if the queue is empty */
  public T getFront();

  /** Detects whether this queue is empty.
      @return  True if the queue is empty, or false otherwise. */
  public boolean isEmpty();

  /** Removes all entries from this queue. */
  public void clear();
} // end QueueInterface
```

Because these operations are defined in an interface, the actual queue can be implemented in various ways. In this exercise you will implement two primitive versions of a queue and you will empirically determine whether or not these are efficient implementations. [In Assignment 1 you will implement the queue more efficiently – but not in this exercise].

The two approaches you will implement both use a simple array to store the queue data. The difference in the implementations is where the items will be placed and removed.

1) Add new items (enqueue) in the queue at the beginning of the array (**always index 0**) and remove (dequeue) from the logical end of the array. **Note that in this case the logical back of the queue will always be at index 0.**
2) Add new items (enqueue) in the queue at the end of the array and remove (dequeue) from the beginning of the array (**always index 0**). **Note that in this case the logical front of the queue will always be at index 0.**

Clearly in both of these implementations, shifting is required for one operation or the other. In this exercise you will test both of these implementations to see if one has any advantage over the other.

Note: For more information on Queues see Chapter 7 of the Carrano text.

Details:

To allow your queue classes to be flexible you will make them generic, utilizing a type parameter. Specifically, your classes will both implement the QueueInterface<T> interface. This interface is specified above and is also provided for you in file QueueInterface.java (see Canvas Files/Handouts/).

In order to enable the bookkeeping to determine the efficiency of your queue operations, your queue classes will also implement a second interface, Moves. This interface is provided for you in file Moves.java and contains the following two methods:

```
public int getMoves();     // return the value of the moves variable
public void setMoves(int val);  // initialize the moves variable to val
```

This interface will allow you to set and obtain the information in a moves variable within your queue classes. The moves variable will be updated during your enqueue() and dequeue() methods such that **each assignment to a location in the array causes an increment of the variable**. For example, given the first queue implementation above, an enqueue() to an empty queue will require 1 move (put item in location 0). However, the next enqueue() will require a shift of the first item down (1 move) plus the assignment of the new item (1 move) for a total of 2 moves.

Call your classes PrimQ1 (add at index 0, remove from end) and PrimQ2 (add at end, remove from index 0). Below are the class headers that are required for these:

```
public class PrimQ1<T> implements QueueInterface<T>, Moves

public class PrimQ2<T> implements QueueInterface<T>, Moves
```

To get you started and enable this to be completed within a single recitation, I have written a shell of both of these classes for you. See files PrimQ1.java and PrimQ2.java. Look at the code that is provided and complete the remaining methods as specified in the interfaces and in the description above.

Once you have completed your PrimQ1<T> and PrimQ2<T> classes, you will test them with the program CS445Rec2.java, which is also provided for you. Read over this program carefully – your classes should work with the program as it is written – if CS445Rec2.java will not compile or run correctly then you must fix your classes – do not change CS445Rec2.java. [Note: Besides the files mentioned already, you will also need file EmptyQueueException.java for the main program to compile and run]. These files can be found on Canvas at Files/Recitation/Recitation2/.

After everything is working, look at the output to your program and compare the results for PrimQ1<T> and PrimQ2<T>. Later I will be posting my results and you can compare yours to mine (they should be the same). Prior to the end of recitation, your TA will be soliciting volunteers to give a brief explanation and demonstration of their classes and results. These

demonstrations are not required but I encourage everyone to consider doing this at least once during the term.

Note: In the constructors provided in files PrimQ1.java and PrimQ2.java (and generally when creating the array in a constructor for a generic class) an array of type T is not explicitly created. Rather, an array of Object is created, then cast it to (T []). For more details on this issue see Section 2.7 of the Carrano text. This was also discussed in Lecture 3 and demonstrated in handout MyArray.java.