

CS 0445 Spring 2024

Recitation Exercise 1

[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. Thus, in this case, the logical rear of the queue will be at index 0.
- 2) Add new items (enqueue) in the queue at the logical end of the array and remove (dequeue) from the beginning of the array (**always index 0**). Thus, in this case, the logical front of the queue will 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](#). To keep this implementation simple, for the `enqueue()` operation, if the underlying array is full, simply do nothing (do not add the item but don't generate an error either). In Assignment 1 you will improve this operation by resizing the structure to accommodate the new item.

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) at index 0 for a total of 2 moves. For `dequeue()` you should also count the assignment of the return value as a move. Thus, for the first queue implementation above, each `dequeue()` would take exactly 1 move (since you are removing from the end of the array and no shifting is needed). However, in the second implementation (where `dequeue` is done at index 0) you will need 1 move for the return value plus a move for each remaining item to shift down one location.

Call your classes `PrimQ1` (enqueue at index 0, dequeue from end) and `PrimQ2` (enqueue at end, dequeue 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 `CS445Rec1.java`, which is also provided for you. Read over this program carefully – your classes should work with the program as it is written – if `CS445Rec1.java` will not compile or run correctly then you must fix your classes – do not change [CS445Rec1.java](#). [Note: Besides the files mentioned already, you will also need file [EmptyQueueException.java](#) for the main program to compile and run].

After everything is working, look at the output to your program and compare the results for $\text{PrimQ1}\langle T \rangle$ and $\text{PrimQ2}\langle T \rangle$. 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. As mentioned in the Course Policies, these demonstrations are not required but I encourage everyone to consider doing this at least once during the term.