**Implementing Queues**

**Linked Lists**

Assume you have your List API available. Sketch the Java pseudocode for implementing the following methods. Begin by first thinking about the necessary instance data to implement a queue.

**Instance Data**

**Methods**

// adds item to the rear of the queue
**add(T item)**

// remove and return the item at the front of the queue
**T remove()**

**Arrays**

A second strategy for implementing queues is to use an array. (For now don't worry about expanding the capacity of the array if it is full.)

| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |

What is the necessary instance data?

**Instance Data**

Assume you perform four additions, followed by two removes, followed by two additions. (These can be any arbitrary values being added/removed.)

What is the contents of your array?

| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |

Is the array full?

Consider how you might add one more element. Where will it be placed?

How many elements do you have to add before the array is full?

Perform three more removes, followed by two additions.

What is the contents of the array?

| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 |

What index refers to the front of the array?

What index refers to the rear of the array?

**Circular Queues**

A common way of representing an array-based queue is to use a circular queue.

Instance Data

**Methods**

// returns true if the queue is empty, false otherwise
**boolean isEmpty()**

// returns the number of elements in the queue
**int size()**

// adds item to the rear of the queue
**add(T item)**

// remove and return the item  at the front of the queue
**T remove(**)