CS200: Queues

- Prichard Ch. 8

First In First Out (FIFO) structure
- Imagine a checkout line
- So removing and adding are done from opposite ends of structure.
- Used in operating systems (e.g. print queue).

Operations

- Create an empty queue
- Determine whether a queue is empty
- Add a new item to the queue
- Remove item from the queue (that was added the earliest)
- Remove all items from the queue
- Retrieve item from queue that was added earliest

Queue Operations

- enqueue(in newItem: QueueItemType)
  - Add new item at the back of a queue
- dequeue():QueueItemType
  - Retrieves and removes the item at the front of a queue
- peek(): queueItemType {query}
  - Retrieve item from the front of the queue. Retrieve the item that was added earliest.
- isEmpty():boolean{query}
- createQueue()
- dequeueAll()
Reference-Based Implementation 1

A linked list with two external references
- A reference to the front
- A reference to the back

At which end do we dequeue?

Reference of the First Node
Reference of the Last Node

Reference-Based Implementation 2

A circular linked list with one external reference
- lastNode references the back of the queue
- lastNode.getNext() references the front

Inserting an item into a nonempty queue
1. newNode.next = lastNode.next;
2. lastNode.next = newNode;
3. lastNode = newNode;

Inserting a New Item
- Insert a new item into the empty queue
Insert new item into the queue

```java
public void enqueue(Object newItem) {
    Node newNode = new Node(newItem);
    if (isEmpty()) {
        newNode.next = newNode;
    } else {
        newNode.next = lastNode.next;
        lastNode.next = newNode;
    }
    lastNode = newNode;
}
```

Removing an item from queue

```java
public Object dequeue() throws QueueException {
    if (!isEmpty()) {
        Node firstNode = lastNode.next;
        if (firstNode == lastNode) {
            lastNode = null;
        } else {
            lastNode.next = firstNode.next;
        }
        return firstNode.item;
    } else {
        // exception handling...
    }
}
```

Naïve Array-Based Implementation

Drift can cause the queue to appear full

How do we initialize front and back?
(Hint: what does a queue with a single element look like?)
Solving Drift: Circular implementation of a queue

- Delete

When either front or back advances past MAX_QUEUE-1, it wraps around 0

Solving Drift

- Insert u
Queue with Single Item

- back and front are pointing at the same slot.

Insert the last item

back catches up to front when the queue becomes full.

When the queue is FULL, front is one slot ahead of back as well.

Wrap the values for front and back

- Initializing
  front = 0
  back = MAX_QUEUE - 1
  count = 0

- Adding
  back = (back + 1) % MAX_QUEUE;
  items[back] = newItem;
  ++count;

- Deleting
  deleteItem = items[front];
  front = (front + 1) % MAX_QUEUE;
  --count;
**enqueue with Array**

```java
public void enqueue(Object newItem) throws QueueException{
    if (!isFull()){
        back = (back+1) % (MAX_QUEUE);
        items[back] = newItem;
        ++count;
    }else {
        throw QueueException(your_message);
    }
}
```

**dequeue()**

```java
public Object dequeue() throws QueueException{
    if (!isEmpty()){
        Object queueFront = items[front];
        front = (front+1) % (MAX_QUEUE);
        --count;
        return queueFront;
    }else{
        throw new QueueException (your_message);
    }
}
```

**Implementation with List**

- You can implement operation dequeue() as the list operation remove(0).
- peek() as get(0)
- enqueue() as add (size()-1, newItem)

**Queue implementations**

- What are the advantages/disadvantages of the circular array / linked list implementations?