AADS 573 Honors
Lab Queue

Name_____
date _____ / _____

queues

pronounced "**Q**"

A *queue* is similar to a list except that it has restrictions on the way you put items in and take items out. Specifically, a queue uses *first-in, first-out* (FIFO) processing. That is, the first item put in the list is the first item that comes out of the list. Figure 12.6 depicts the FIFO processing of a queue.

key concept: A queue is a linear data structure that manages data in a first-in, first-out manner.

Any waiting line is a queue. Think about a line of people waiting for a teller at a bank. A customer enters the queue at the back and moves forward as earlier customers are serviced. Eventually, each customer comes to the front of the queue to be processed.

Note that the processing of a queue is conceptual. We may speak in terms of people moving forward until they reach the front of the queue, but the reality might be that the front of the queue moves as elements come off. That is, we are not concerned at this point with whether the queue of customers moves toward the teller, or remains stationary as the teller moves when customers are serviced.

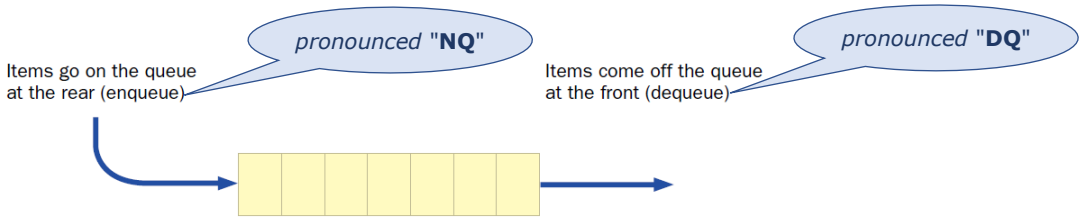

figure 12.6 A queue data structure

A queue data structure typically has the following operations:

- enqueue—adds an item to the rear of the queue
- dequeue—removes an item from the front of the queue
- empty—returns true if the queue is empty

We studied the data structure called **Stack** which has the policy that the most recent object added is also the first object removed (***LIFO***: last in, first out). We used this interface:	The data structure described above, named **Queue**, has the policy the last object added is the last object removed (***FIFO***: first in, first out). We will use this interface:
interface Stack<ItemType> **{** ***//No constructors allowed*** **ItemType peek();** **ItemType pop();** **void push(ItemType x);** **boolean isEmpty();** **}**	**interface Queue<ItemType>** **{** ***//No constructors allowed*** **ItemType peekFront();** **ItemType dequeue();** **void enqueue(ItemType x);** **boolean isEmpty();** **}**

Your first implementation of the **Queue** interface is named **GBSLNQ** (*GBS* **ListNode** queue). This class will store all the data added to the queue using **ListNode** objects. The **ListNode** class methods are shown at right for reference.

```
public class ListNode<ItemType> extends Object
{
  public ListNode(ItemType v, ListNode n)
  public ListNode(ItemType v)
  public ItemType getValue()
  public ListNode<ItemType> getNext()
  public void setValue(ItemType v)
  public void setNext(ListNode< ItemType> n)
}
```

The **GBSLNQ** class has the three instance variables shown at right. We need a reference to the front of the queue (**ref2FirstLN**) for the dequeue operation, and we need a reference to the back of the queue (**ref2LastLN**) for the enqueue operation. The last instance variable is maybe not entirely necessary, but might make some of your code easier(?).

GBSLNQ.java

```
1  public class GBSLNQ<ItemType> extends Object implements Queue
2  {
3    private ListNode<ItemType> ref2FirstLN, ref2LastLN;
4    private int numItemsInQueue;
5
6    public GBSLNQ()
7    {
8
9    }
10 }
```

1. Complete the **GBSLNQ** constructor- both references should be set to **null** (I know that is the default value for instance variable references, but it's helpful to communicate that is indeed what you want).

2. Implement the **isEmpty** method of the queue interface. Be sure to uncomment it in **Queue.java** and to write its implementation in **GBSLNQ.java**.

3. Implement the **enqueue** method. Make sure that you correctly handle the case when the 0^{th} item is enqueued.

4. Implement the **peekFront** method. This returns the value at the front of the queue, but does not remove it. If the queue is empty, **throw new Error("Empty queue")**

5. Implement the **dequeue** method. Make sure that you correctly handle the case when the queue becomes empty.

6. Write the **toString()** method. Its format should be **Front: value0, value1, value2, … value *n***.