**Implementing Queues and Stacks with ListNodes**

## Queue

A real-world example of a **queue** is a lunch line. People always enter the line at the back and leave the line at the front. Cuts are not allowed. In other words, elements in a queue are processed in a First-In-First-Out (FIFO) order. As another example, the playlist on your electronic device probably uses a queue to hold the songs. The first song in the queue gets played first.

The Queue class is often implemented using ListNodes. We need references to two ListNodes, first and last. Between those two, we store as many ListNode objects with data as we need, always adding at the last and removing from the first, which is normal queue behavior.

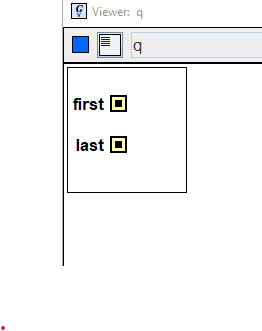
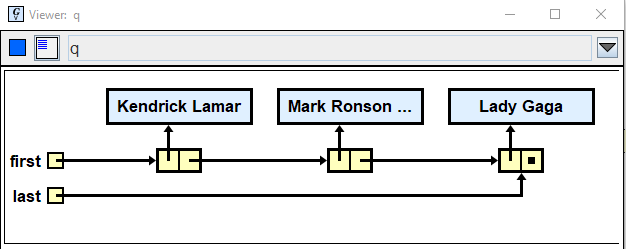
public class Queue  
{  
 private ListNode first;  
 private ListNode last;

Write the Queue class from scratch, implementing these methods:

* **void** add(Object obj) *// enqueues* obj *at the end of the queue*
* Object remove() *// dequeues and returns the element at the front; returns* null *if the queue is empty*
* Object peek() *// returns the element at the front of the queue; returns* null *if the queue is empty*
* **boolean** isEmpty()
* **String** toString() *// returns a string of the elements enclosed in square brackets, separated by commas*

What does the constructor need to do?

In add, what happens in these two cases?

In remove, what happens in the two cases?

In peek, what happens in the two cases?

In isEmpty, what happens in the two cases?

In toString, what happens in the two cases?

The Queue\_driver is given to you. A sample run is pasted at the bottom of the driver. Turn in Queue.java.

## Stack

| D |
| --- |
| C |
| B |
| A |

A real-world example of a **stack** is the dispenser of the trays in the lunch room. You always take the top tray from the top, never from the middle or the bottom. Similarly, the lunch workers always put trays on the top, never at the middle or bottom. In other words, you put the elements on the top of the stack, then take it out from the top, always in a Last-In, First-Out (LIFO) order.

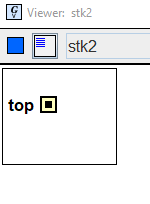
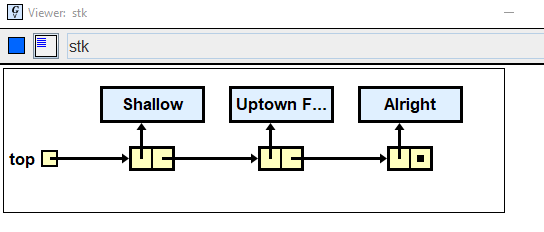
A stack is usually implemented by an array. However, we will implement it using ListNodes. We only need one private field, pointing to the top of the stack.

public class Stack  
 {  
 private ListNode top;

Write the Stack class from scratch, implementing these methods. The names of the methods are the traditional ones.

* void push(Object item) *// pushes* item *onto the top*
* Object pop() *// removes and returns the element at the top; throws an exception if the stack is empty*
* Object peek() *// returns the element at the top; throws an exception if the stack is empty*
* **boolean** isEmpty()
* **String** toString() *// returns a string of the items,* ***starting from the bottom and ending with the top****,   
   // enclosed in square brackets, separated by commas*

As with queues, you have to deal with two different *states* of the stack:

Notice that the toString comes out in reverse: [Alright, Uptown Funk, Shallow]

Notice that pop and peek both throw Exceptions. Use this syntax:

try  
{  
 // the statement which might crash   
}  
catch (NullPointerException e)  
{  
 e.printStackTrace();  
 System.exit(0);   
}

The Stack\_driver is given to you. A sample run is pasted at the bottom of the driver. Turn in Stack.java.

Extension: can you build the string in toString recursively?