Chapter 20 Queues and Priority Queues

CS165
Colorado State University

Original slides by Daniel Liang
Modified slides by
Wim Bohm, Sudipto Ghosh

Queues and Priority Queues

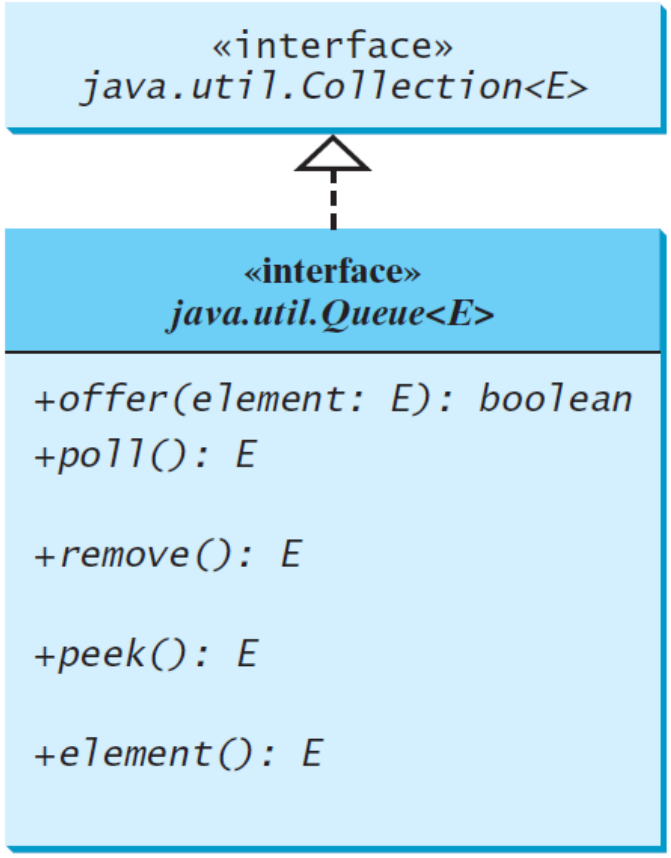
Queue is a first-in/first-out data structure.

- Elements are added to the end of the queue.
- Elements are removed from the beginning of the queue.

Priority queues assign priorities to elements.

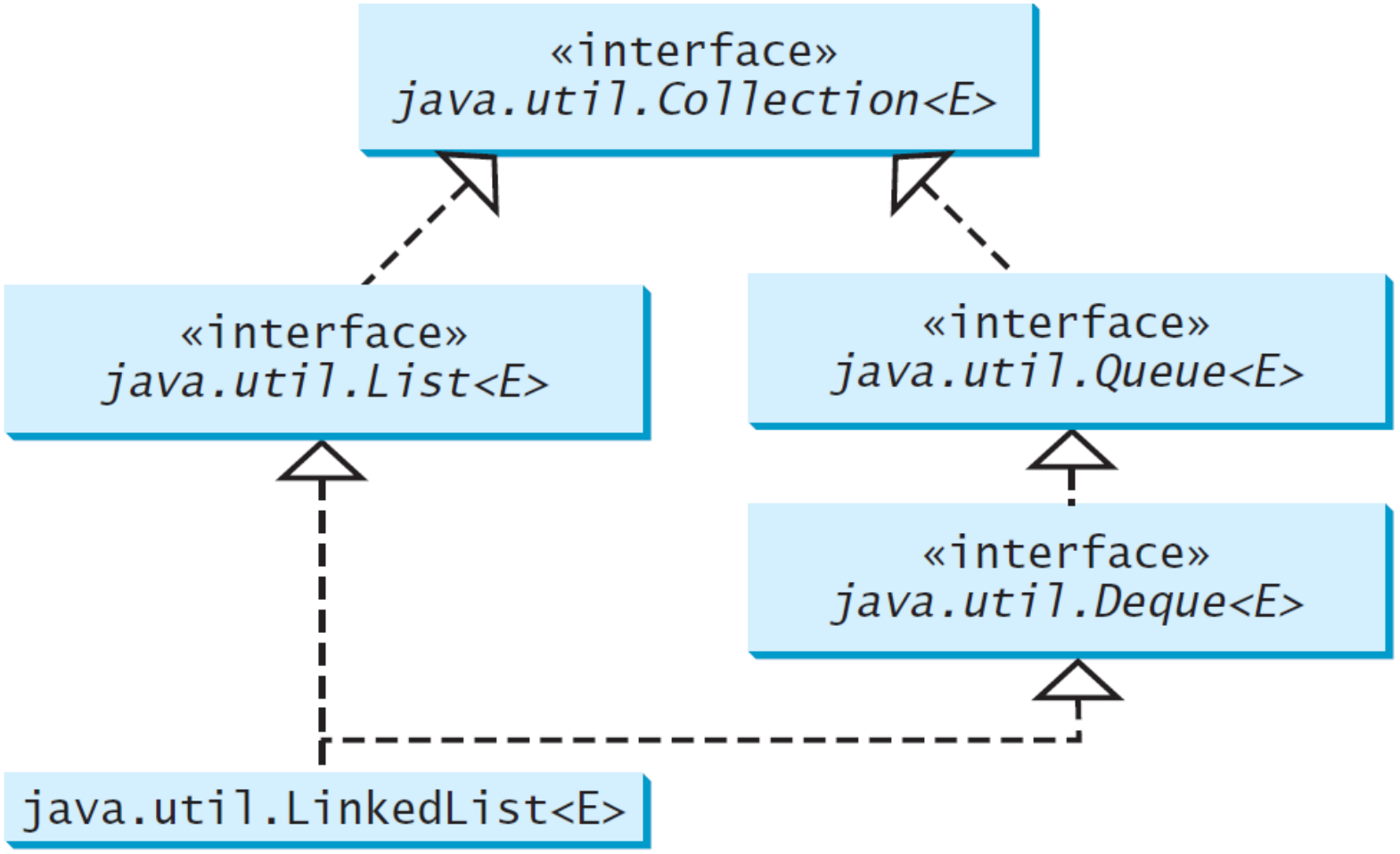
- The element with the highest priority is removed first.

The Queue Interface



Inserts an element into the queue.
Retrieves and removes the head of this queue, or `null` if this queue is empty.
Retrieves and removes the head of this queue and throws an exception if this queue is empty.
Retrieves, but does not remove, the head of this queue, returning `null` if this queue is empty.
Retrieves, but does not remove, the head of this queue, throws an exception if this queue is empty.

Using LinkedList for Queue

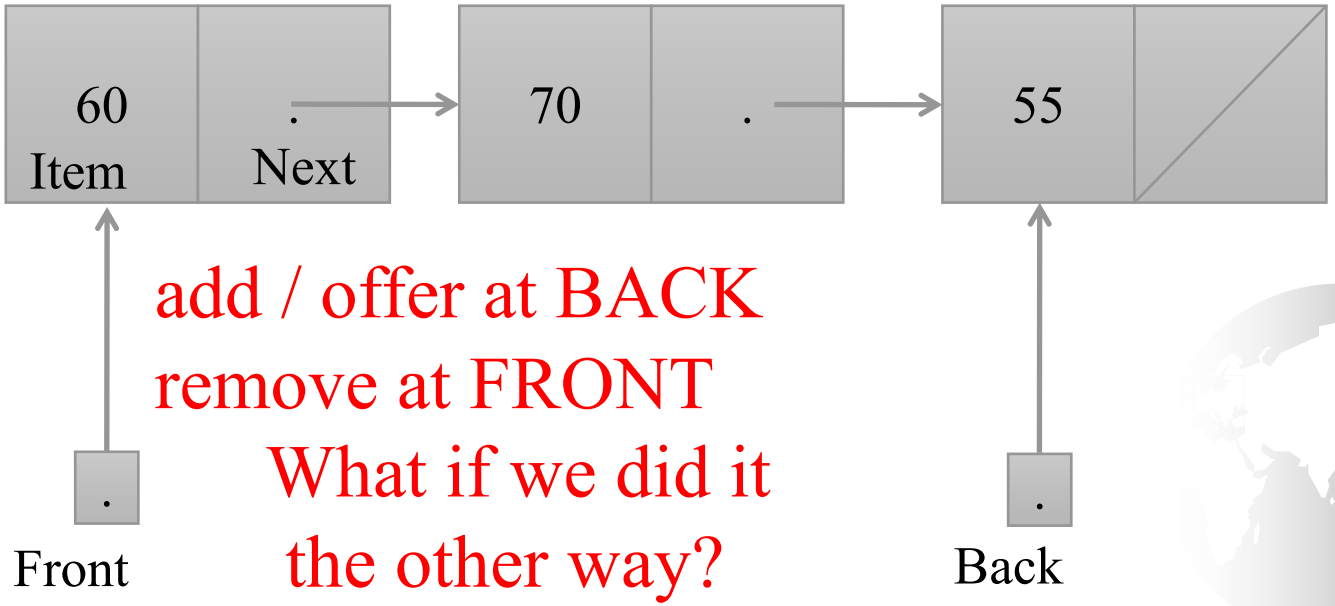


Reference-Based Implementation 1

A linked list with two external references

- A reference to the front
- A reference to the back

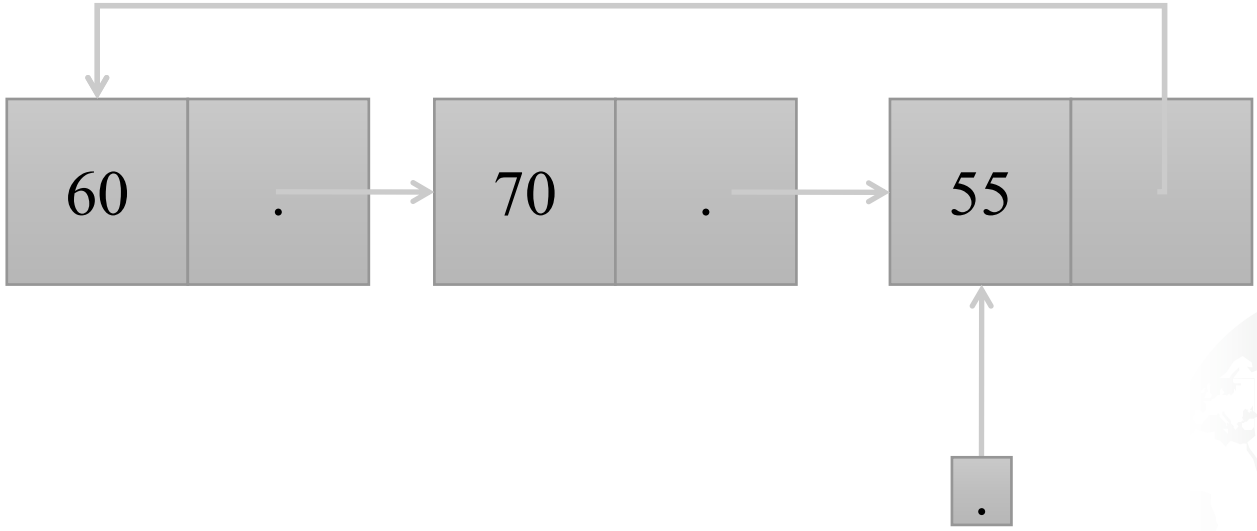
At which end do we enqueue / dequeue?



Reference-Based Implementation 2

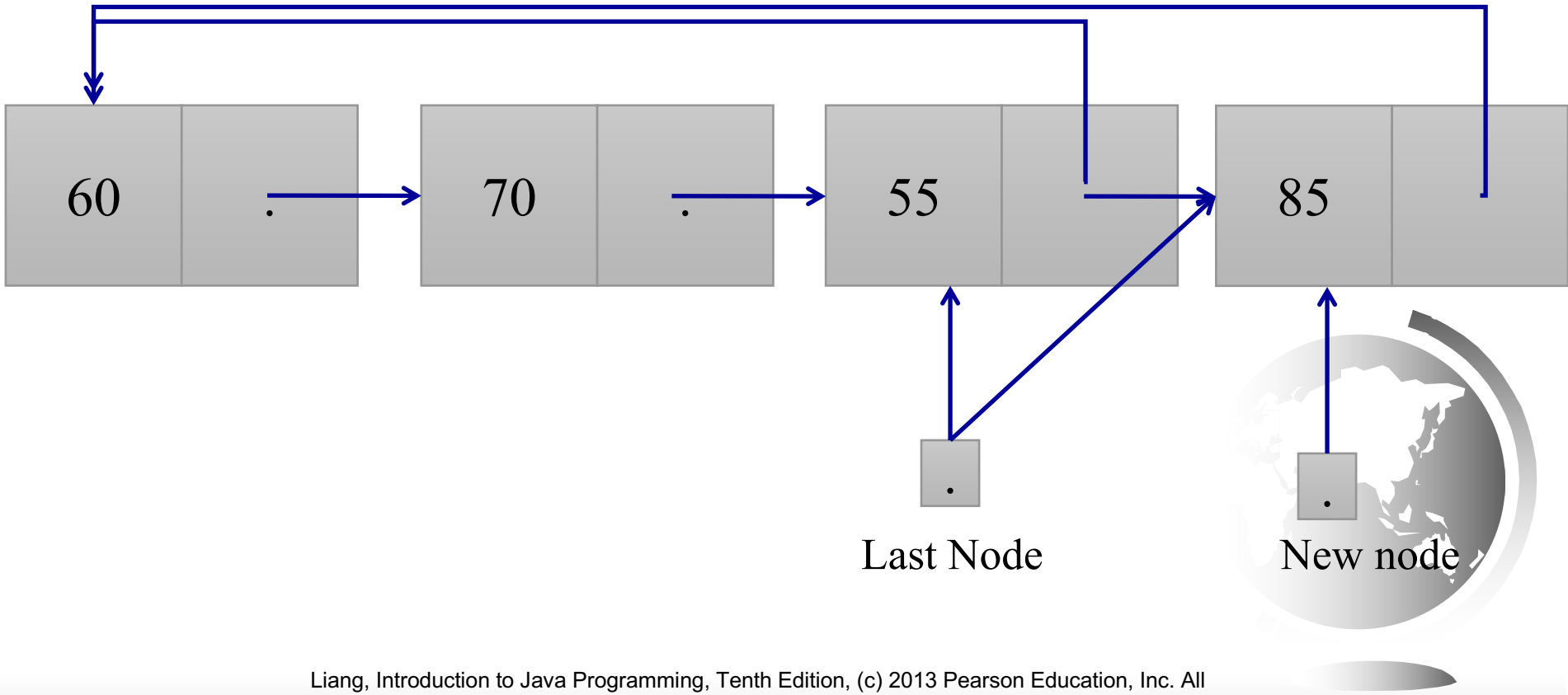
A circular linked list with one external reference

- lastNode references the back of the queue
- lastNode.getNext() references the front



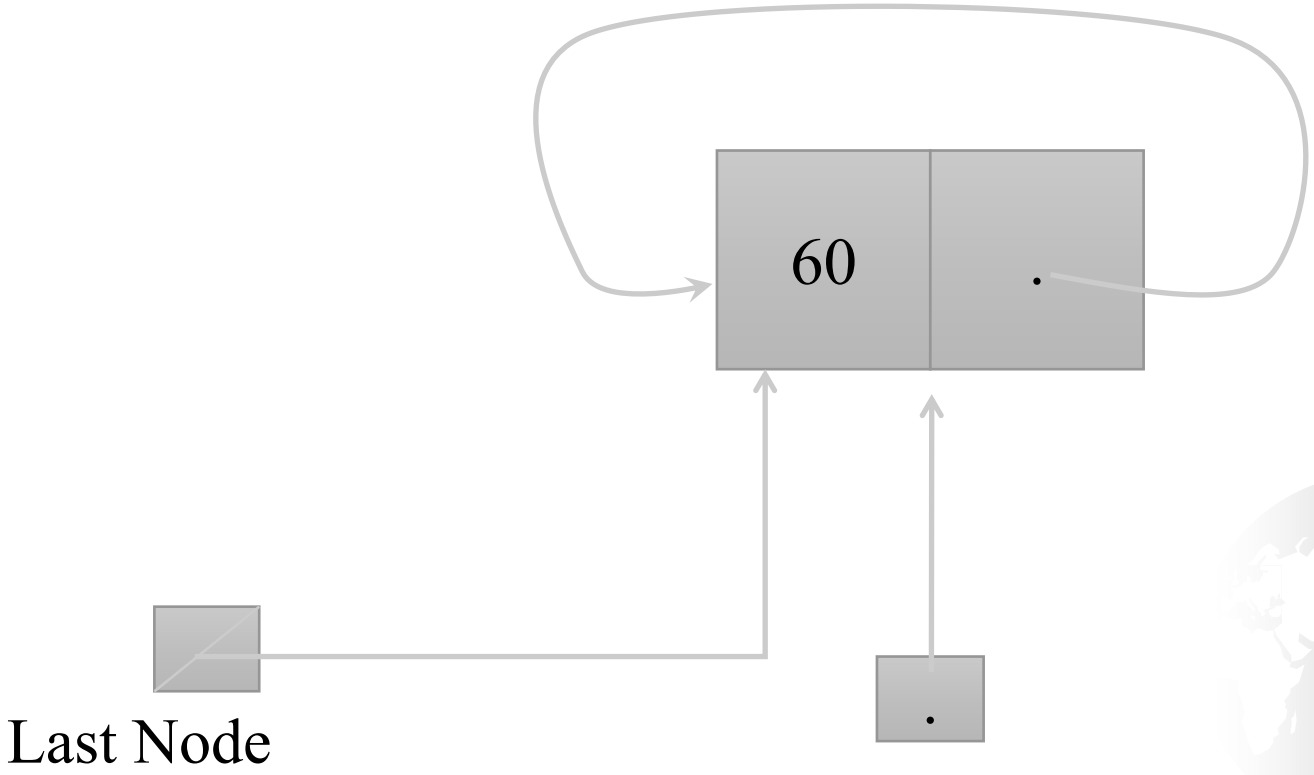
Adding an item into a nonempty queue

1. newNode.next = lastNode.next;
2. lastNode.next = newNode;
3. lastNode = newNode;



Adding an item to an empty queue

- Insert a ***new item*** into the **empty queue**



Add item to queue

```
public void add (Object newItem){
    Node newNode = new Node(newItem);

    if (isEmpty()){
      newNode.next = newNode;
    } else {

      newNode.next = lastNode.next;
      lastNode.next = newNode;
    }

    lastNode = newNode;
}
```

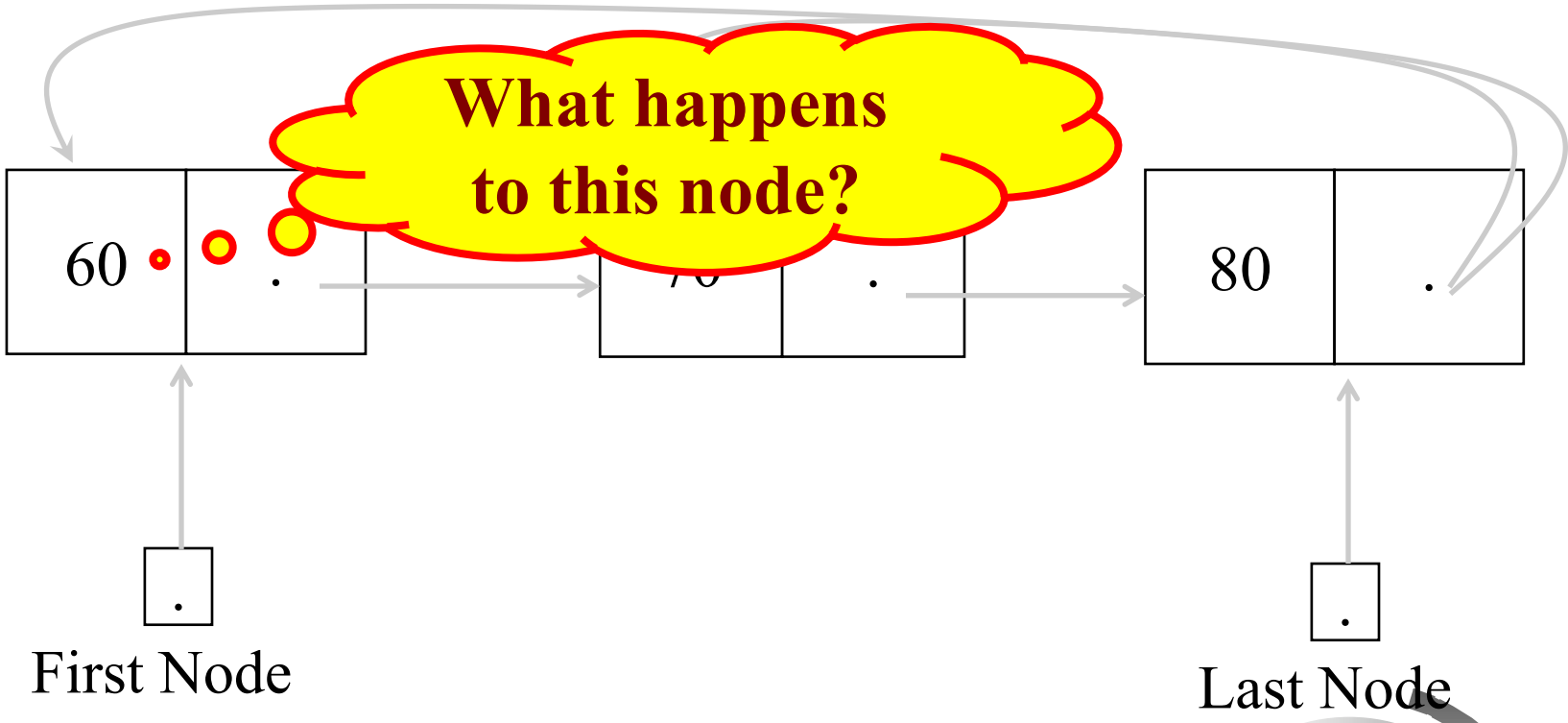
A. Empty queue

B. items in queue

Removing an item from queue

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

Removing an Item



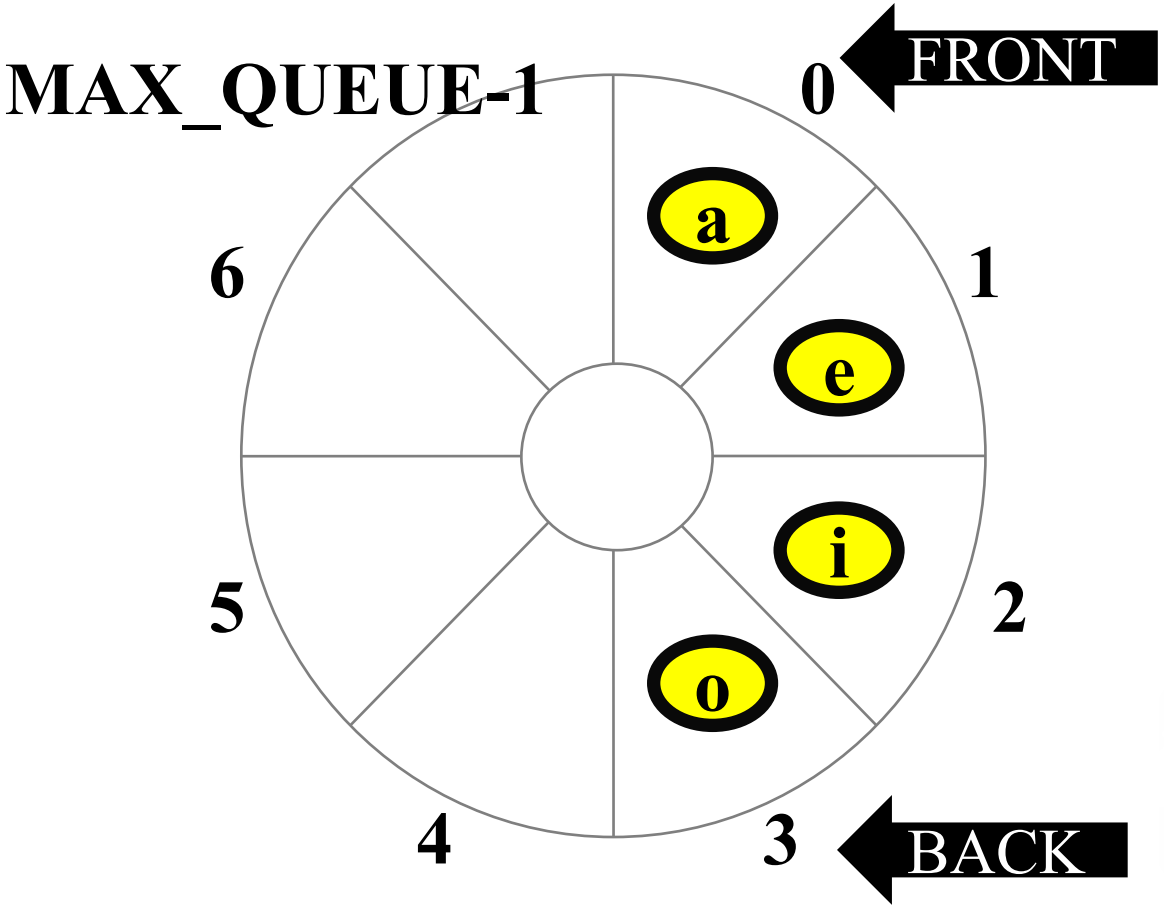

```
Node firstNode = lastNode.next;
if (firstNode == lastnode) {
            lastNode = null;}
   else{lastNode.next = firstNode.next;}
return firstNode.item;
```

Naïve Array-Based Implementation

Drift wastes space

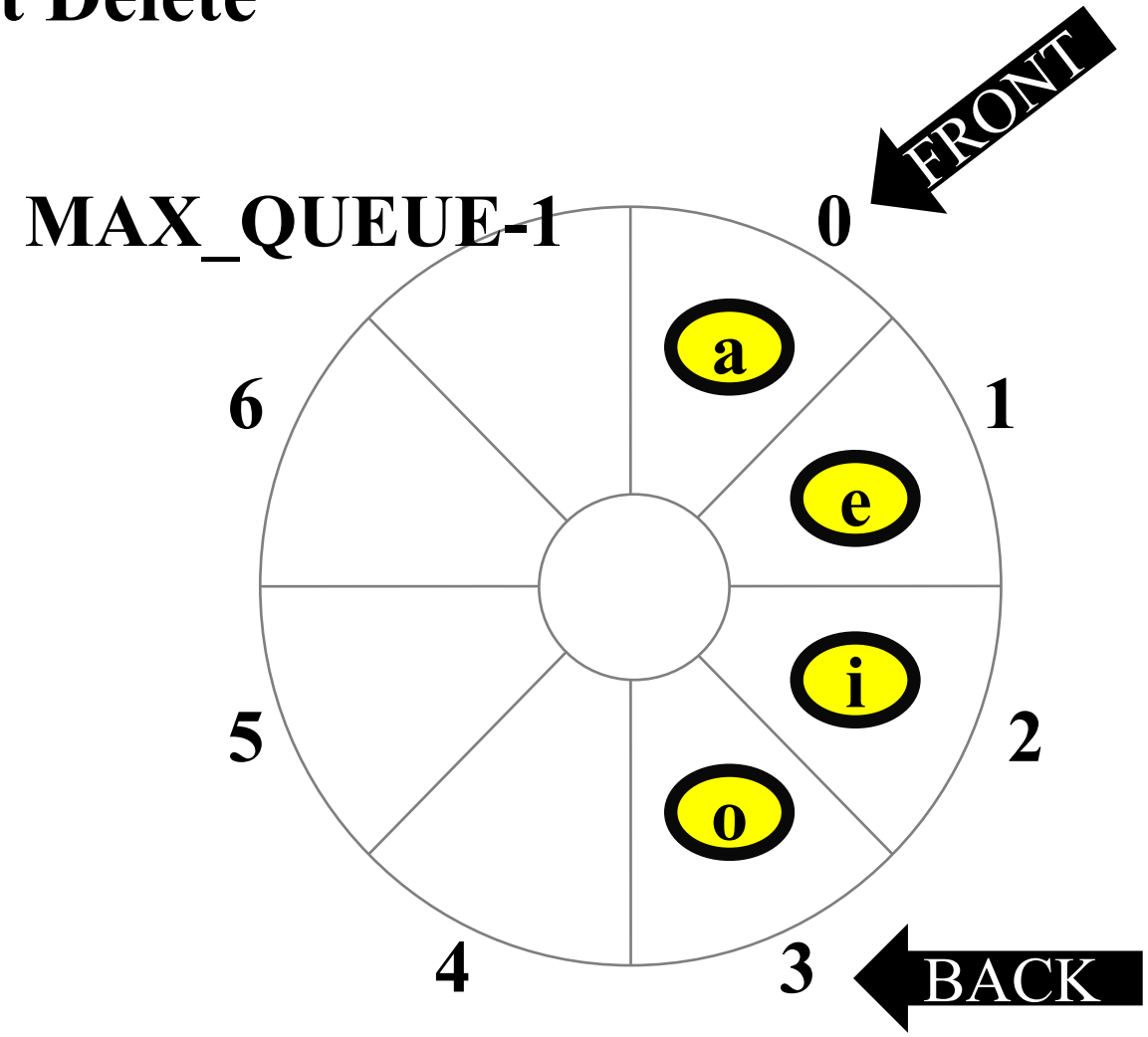
How do we initialize front and back?
(Hint: what does a queue with a single element look like?
what does an empty queue look like?
)
Problem: Drift

Circular implementation of a queue solves drift



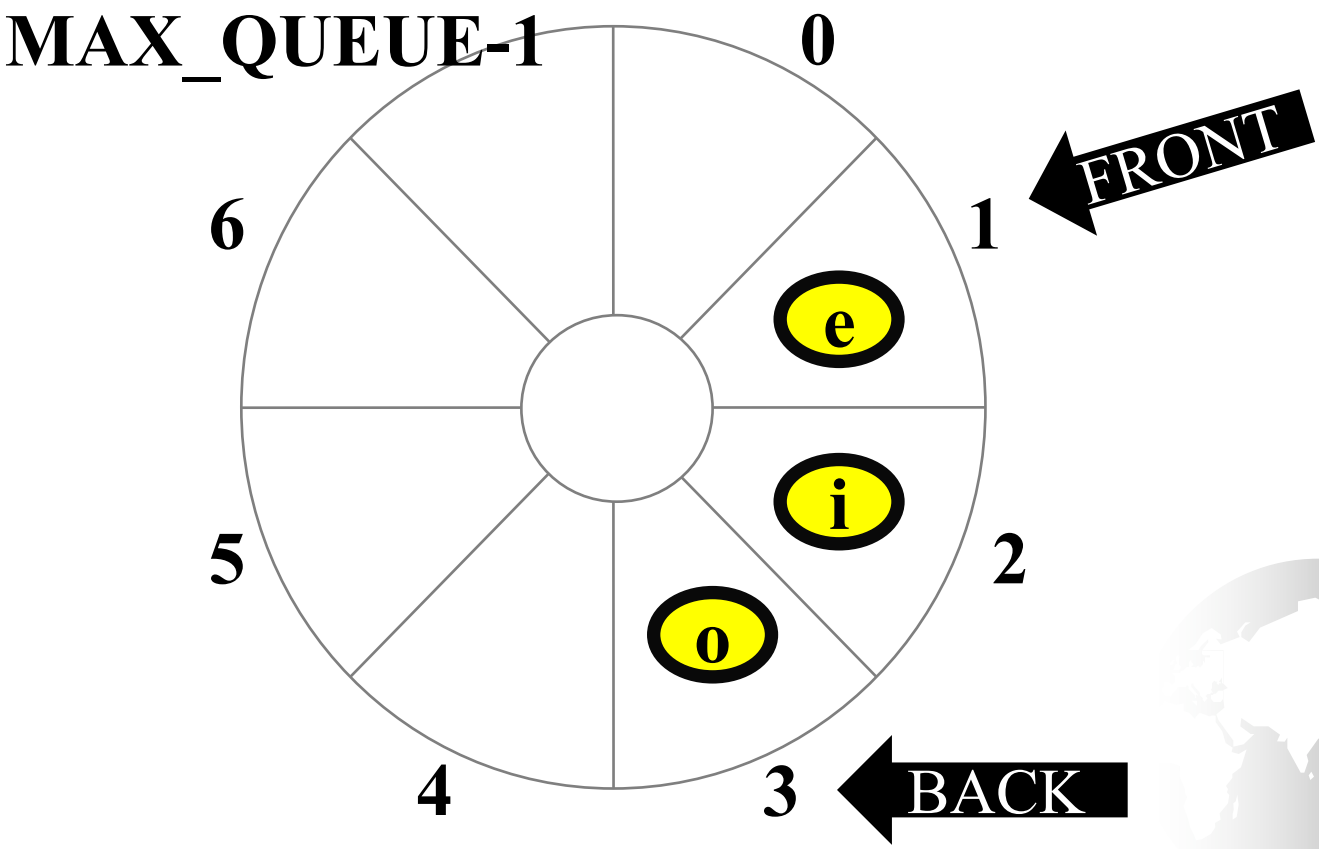
Solving Drift:

- **First Delete**



Solving Drift:

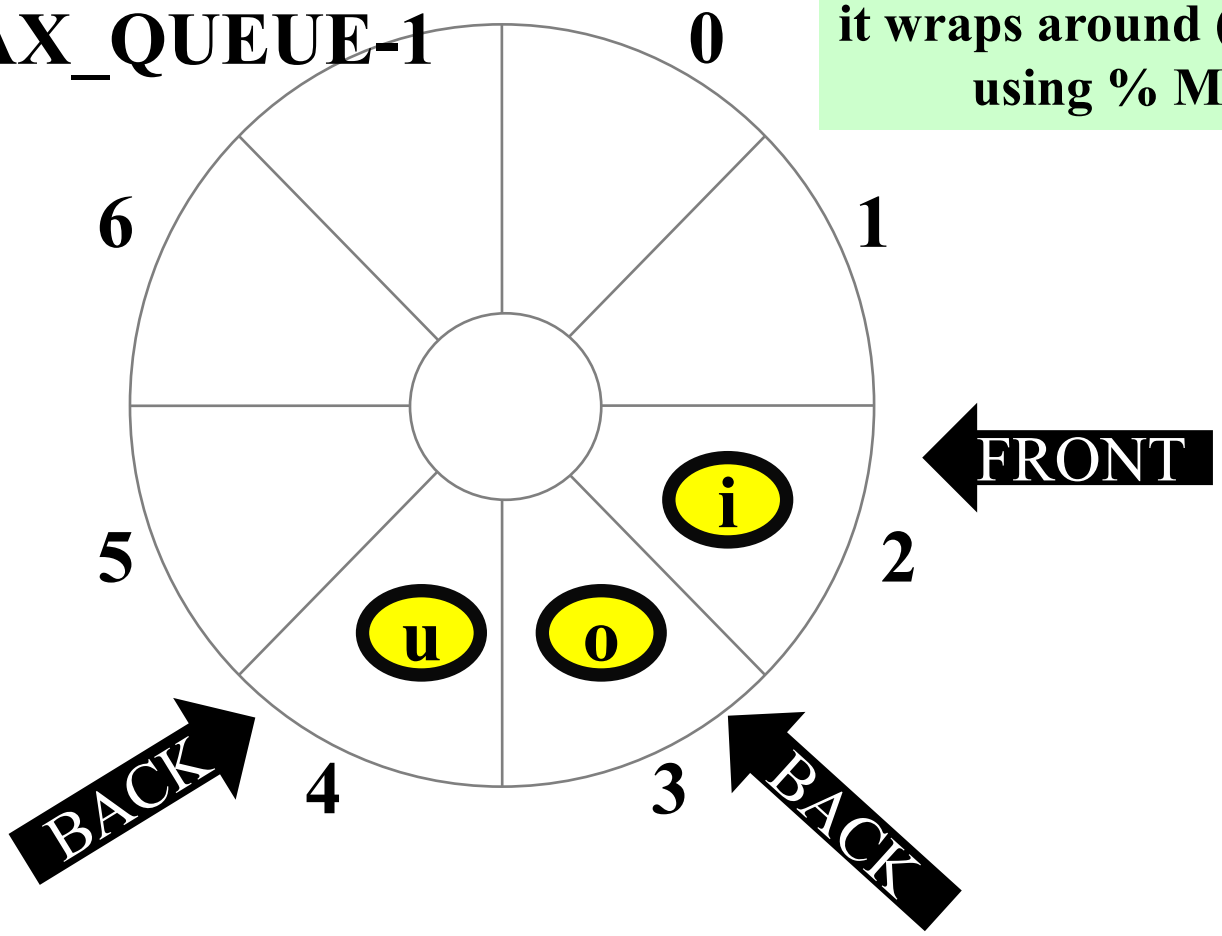
- **Second Delete**



Solving Drift

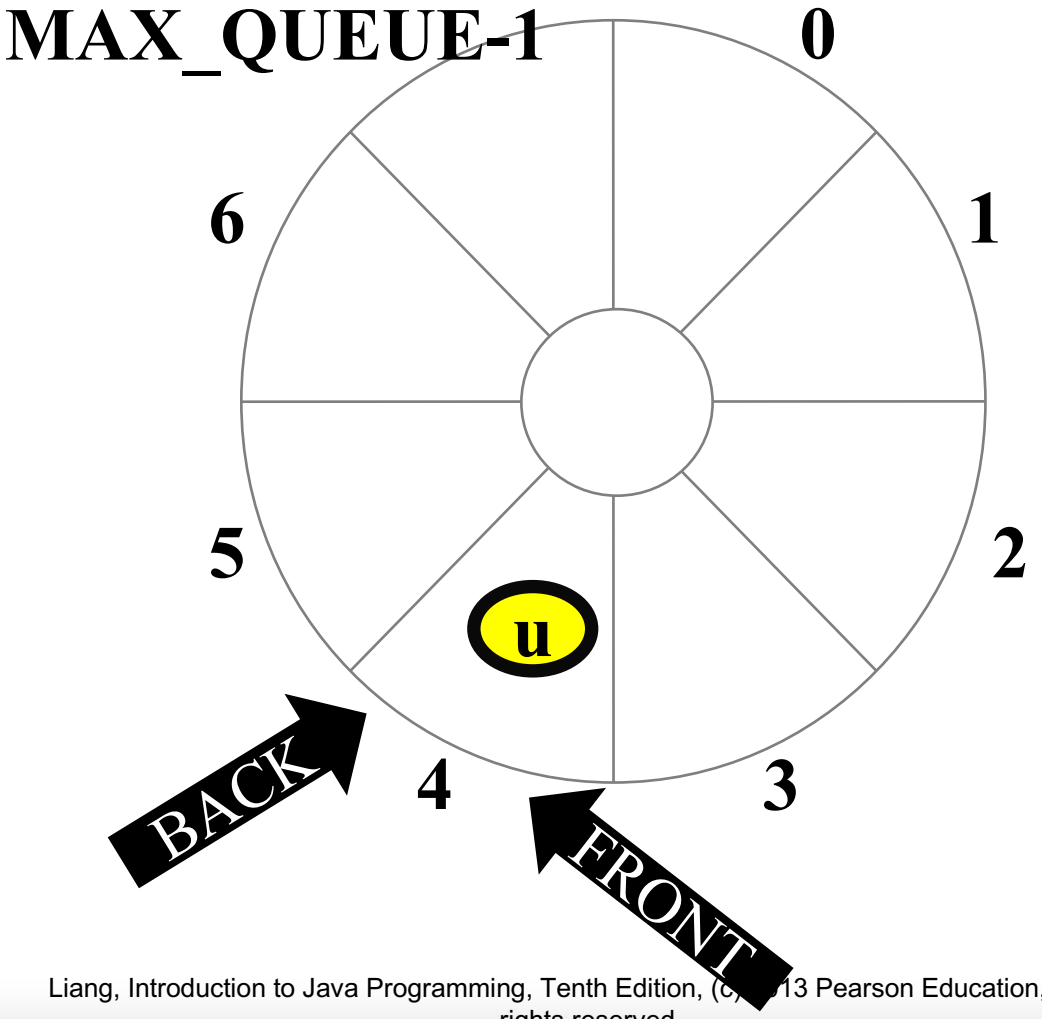
- **add u**

When either front or back advances past MAX_QUEUE-1, it wraps around (to 0: using % MAX_QUEUE)



Queue with Single Item

- *back* and *front* are pointing at the same slot.

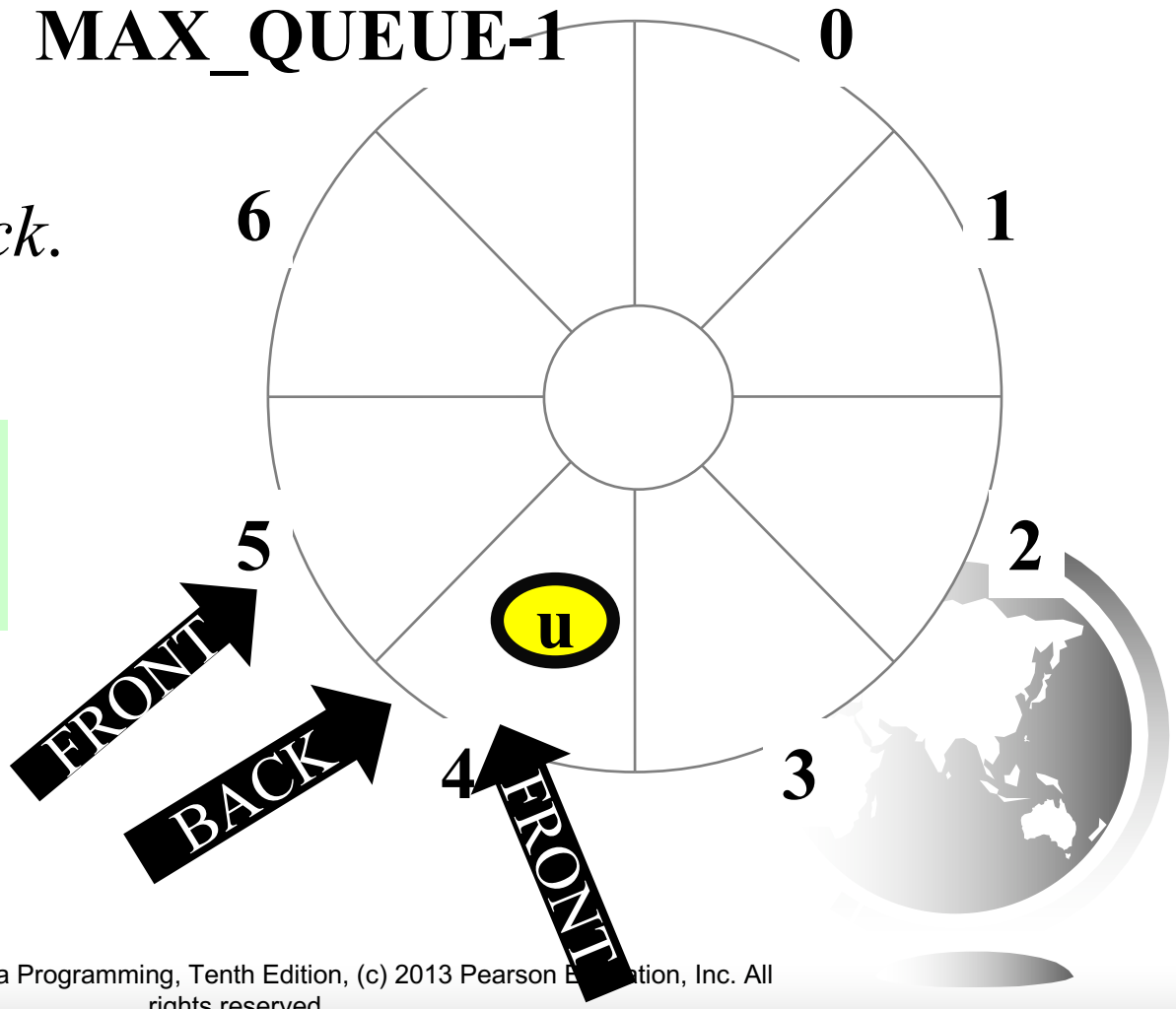


Empty Queue: remove Single Item

Remove last item.

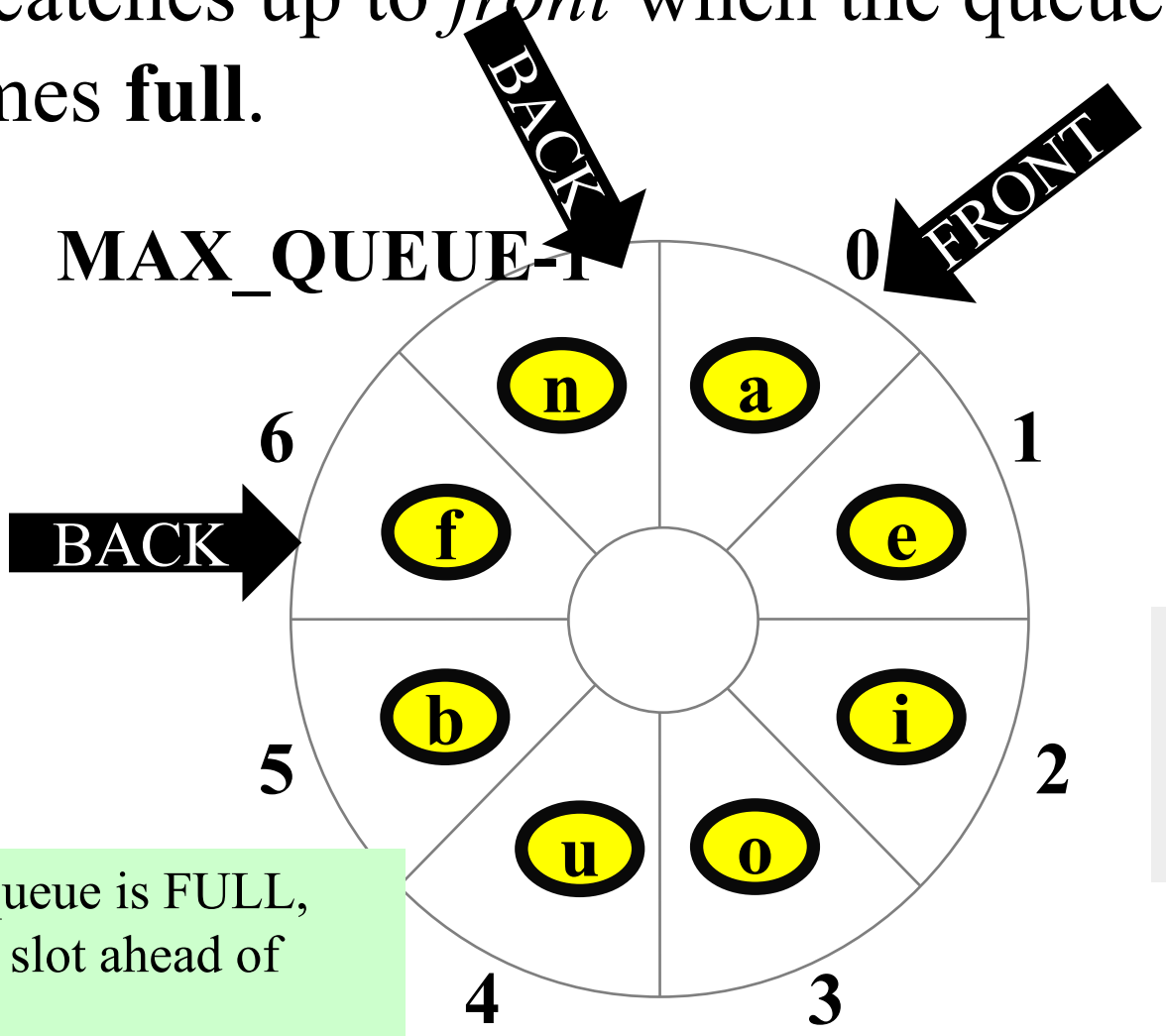
– *front* passed *back*.

When the queue is EMPTY, *front* is one slot ahead of *back*.



Insert the last item

back catches up to *front* when the queue becomes **full**.



Problem?

Solution?

Maintain size:

0:empty
max_queue: full

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

Wrapping the values for front and back

- Initializing

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

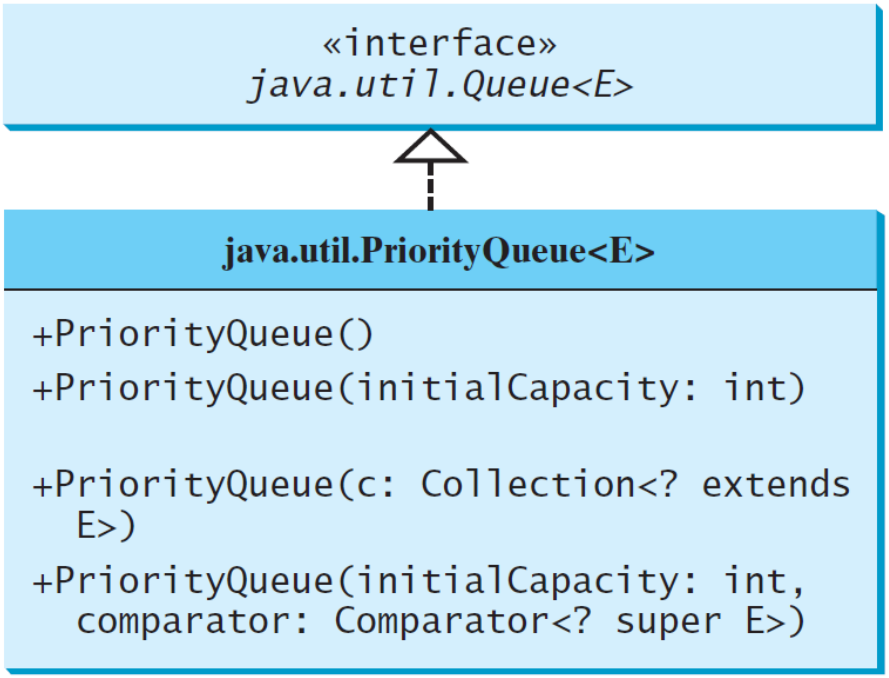
- Adding

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

- Deleting / dequeueing

```
removeItem = items[front];
front = (front +1) % MAX_QUEUE;
--count;
```

The PriorityQueue Class



Creates a default priority queue with initial capacity 11.

Creates a default priority queue with the specified initial capacity.

Creates a priority queue with the specified collection.

Creates a priority queue with the specified initial capacity and the comparator.

PriorityQueueDemo

Run

Implementation of Priority Queue

- Naïve: ArrayList or Linked List
  - Keeping the elements ordered
  - This will make add costly
- Better implementation: Heap (later)