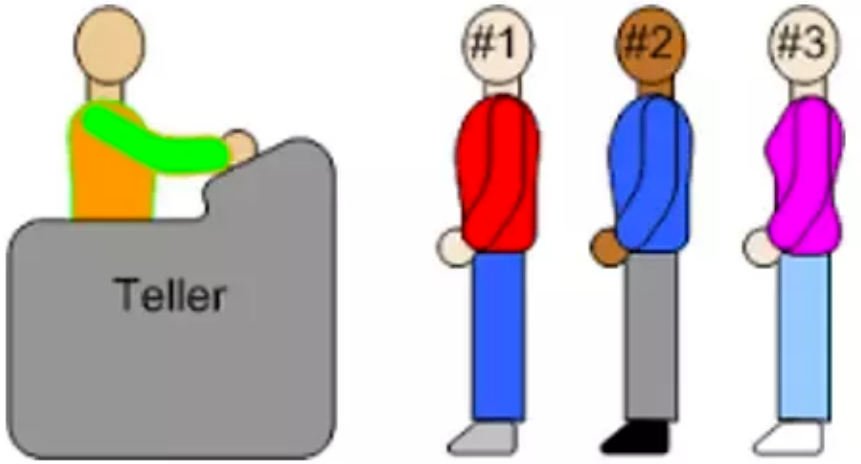
CS206

Queues

Queues


© Mike Baldwin / Cornered
PLEASE WAIT TO BE SEATED
CARTOONSTOCK
Search ID: mba0605


Teller
#1
#2
#3

Queueing Theory

Agner Krarup Erlang

Queues

- Insertions and deletions are First In First Out – FIFO
  - Insert at the back
  - Delete from the front

`Queue` Interface

- `null` is returned from `peek()` and `poll()` when queue is empty
- `add()`, `remove()`, `element()` are identical to `offer()`, `poll()`, `peak()` but for throw.

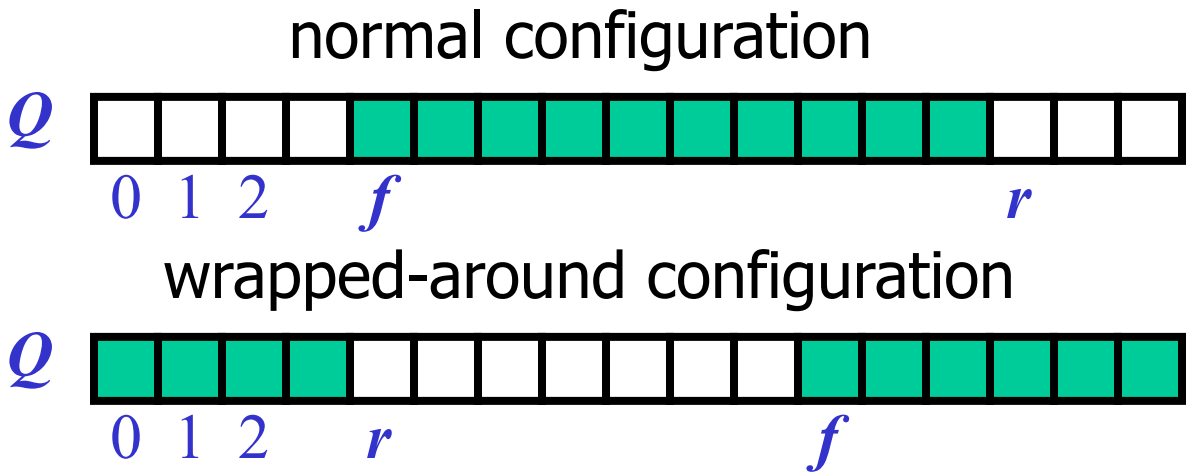
```
public interface QueueIntf<Q> {
    boolean isEmpty();
    int size();
    boolean add(Q q)
       throws IllegalStateException;
    Q remove()
       throws NoSuchElementException;
    Q element()
       throws NoSuchElementException;
    boolean offer(Q q);
    Q poll();
    Q peek();
}
```

Example

Operation	output	Queue Contents
offer(5)	TRUE	{5}
offer(3)	TRUE	{5,3}
poll()	5	{3}
offer(7)	TRUE	{3, 7}
poll()	3	{3,7}
peek()	7	{7}
poll()	7	{}
poll()	null	{}

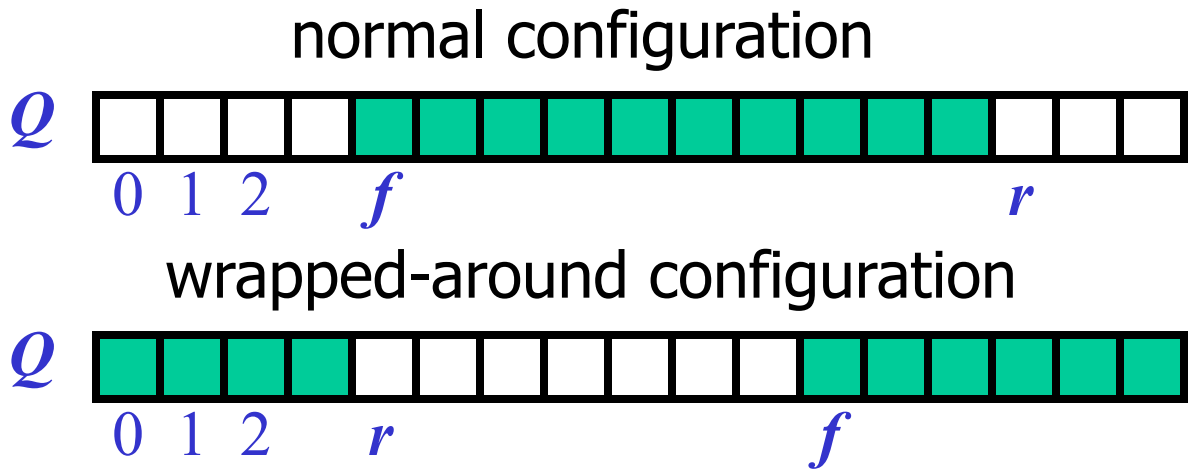
Array-based Queue

- An array of size `n` in a circular fashion
- Two `int`s to track front and size
  - `f`: index of the front element
  - `co`: number of stored elements



Circular Array and Queue

- When the queue has fewer than `n` elements, location `r =(f+co)%n` is the first empty slot past the rear of the queue



Start of Queue Implementation

```
public class ArrayQueue<Q> implements QueueIntf<Q> {
    /** the default capacity for the backing array */
    private static final int CAPACITY = 40;
    /** The array in which the queue data is stored */
    private Q[] backingArray;
    /** The array location of the head of the queue */
    private int count;
    /** The array location of the end of the queue */
    private int frontLoc;
    /**
     * Create an array backed queue with the default capacity. */
    public ArrayQueue() {
        this(CAPACITY);
    }
    /**
     * Create an array backed queue with the given capacity
     * @param qSize the capacity for the queue */
    public ArrayQueue(int qSize) {
        count = 0;
        frontLoc = 0;
        backingArray = (Q[]) new Object[qSize];
    }
```

offer(), add()

- must handle case if the array becomes full
  - Limitation of the array-based implementation
    - offer returns false
    - add throws exception

Performance and Limitations for array-based Queue

- Performance
  - let n be the number of objects in the queue
  - The space used is $O(n)$
  - Each operation runs in time $O(1)$
- Limitations
  - Max size is limited and can not be changed
  - Pushing onto a full stack queue in an exception

Simulating a Bank