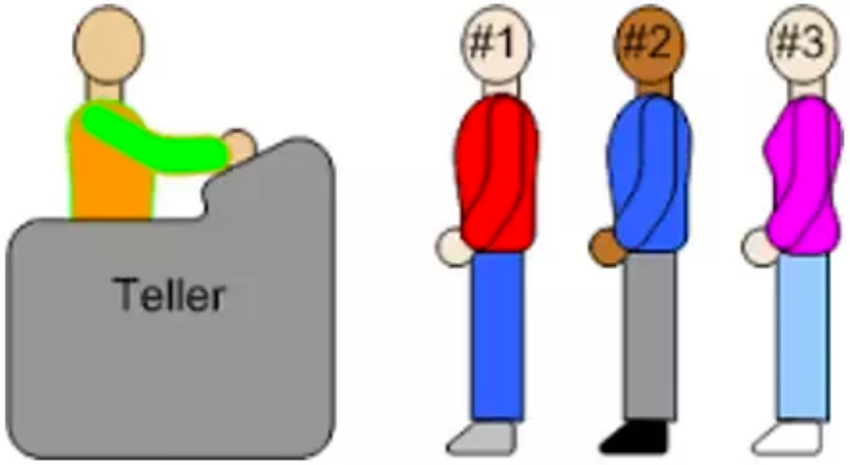
CS206

Queues

Queues

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


Teller
#1
#2
#3

The Queue ADT

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

`Queue` Interface

- Java interface describing the Queue ADT
- `null` is returned from `dequeue()` and `first()` when queue is empty

```
public interface
Queue<E> {
  int size();
  boolean isEmpty();
  E first();
  void enqueue(E e);
  E dequeue();
}
```

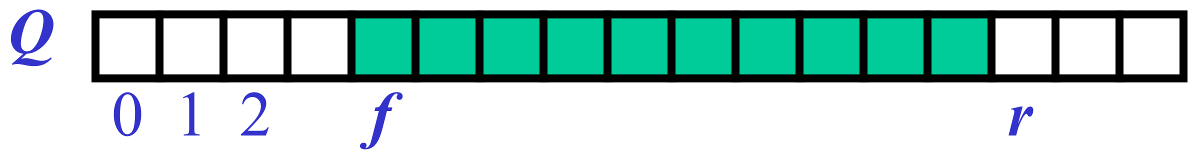
Example

Operation	***Output***	***Q***
enqueue(5)	–	(5)
enqueue(3)	–	(5, 3)
dequeue()	*5*	(3)
enqueue(7)	–	(3, 7)
dequeue()	*3*	(7)
first()	*7*	(7)
dequeue()	*7*	()
dequeue()	*null*	()
isEmpty()	*true*	()
enqueue(9)	–	(9)
enqueue(7)	–	(9, 7)
size()	*2*	(9, 7)
enqueue(3)	–	(9, 7, 3)
enqueue(5)	–	(9, 7, 3, 5)
dequeue()	*9*	(7, 3, 5)

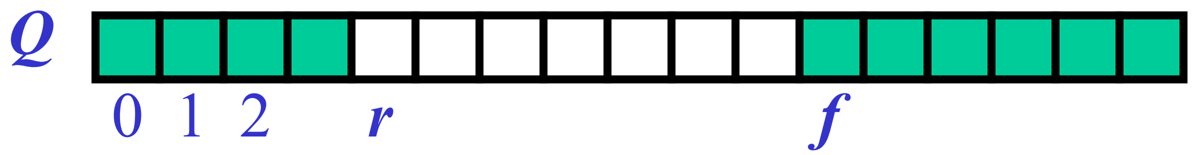
Array-based Queue

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

normal configuration



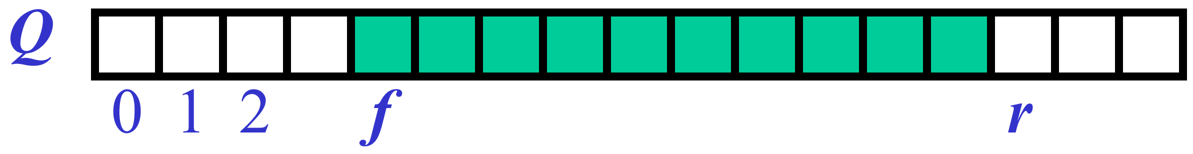
wrapped-around configuration



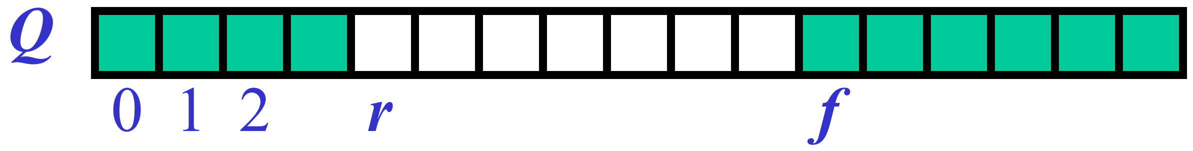
Circular Array and Queue

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

normal configuration



wrapped-around configuration



enqueue

- A `enqueue` will throw an exception if the array becomes full
    - Limitation of the array-based implementation

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

Array-Based Queue Code

```
public class ArrayQueue<E> implements QueueInterface<E> {
    private static final int CAPACITY = 1000;
    E[] queueArray;
    int front =0;
    int size=0;
    @Override
    public int size() {
     return size;
    }
    @Override
    public boolean isEmpty(){
     return size==0;
    }
    @Override
    public E first() {
     if (isEmpty()) return null;
     return queueArray[front];
    }
    @Override
    public void enqueue(E e) throws IllegalStateException {
     if (size==queueArray.length) throw new IllegalStateException("Queue full")
     queueArray[(front+size)%queueArray.length] = e;
     size++;
    }
```

Code

```
 @Override
 public E dequeue() {
if (isEmpty()) return null;
E e = queueArray[front];
queueArray[front] = null;
front = (front+1)%queueArray.length;
size--;
return e;
 }
 public ArrayQueue() {
this(CAPACITY);
 }
 @SuppressWarnings("unchecked")
 public ArrayQueue(int capacity) {
queueArray = (E[])new Object[capacity];
 }
```