Queues
Queues
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

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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Output</th>
<th>$Q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>enqueue(5)</td>
<td>–</td>
<td>(5)</td>
</tr>
<tr>
<td>enqueue(3)</td>
<td>–</td>
<td>(5, 3)</td>
</tr>
<tr>
<td>dequeue()</td>
<td>5</td>
<td>(3)</td>
</tr>
<tr>
<td>enqueue(7)</td>
<td>–</td>
<td>(3, 7)</td>
</tr>
<tr>
<td>dequeue()</td>
<td>3</td>
<td>(7)</td>
</tr>
<tr>
<td>first()</td>
<td>7</td>
<td>(7)</td>
</tr>
<tr>
<td>dequeue()</td>
<td>7</td>
<td>()</td>
</tr>
<tr>
<td>dequeue()</td>
<td>null</td>
<td>()</td>
</tr>
<tr>
<td>isEmpty()</td>
<td>true</td>
<td>()</td>
</tr>
<tr>
<td>enqueue(9)</td>
<td>–</td>
<td>(9)</td>
</tr>
<tr>
<td>enqueue(7)</td>
<td>–</td>
<td>(9, 7)</td>
</tr>
<tr>
<td>size()</td>
<td>2</td>
<td>(9, 7)</td>
</tr>
<tr>
<td>enqueue(3)</td>
<td>–</td>
<td>(9, 7, 3)</td>
</tr>
<tr>
<td>enqueue(5)</td>
<td>–</td>
<td>(9, 7, 3, 5)</td>
</tr>
<tr>
<td>dequeue()</td>
<td>9</td>
<td>(7, 3, 5)</td>
</tr>
</tbody>
</table>
Array-based Queue

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

$Q_{0\ 1\ 2\ f\ r}$

- 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.

```
0 1 2   f     r
```

normal configuration

```
Q 0 1 2   f     r
```

wrapped-around configuration

```
Q r 0 1 2   f
```

- CS206
- Lec10
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
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++;
    }
}
```java
@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];
}
```