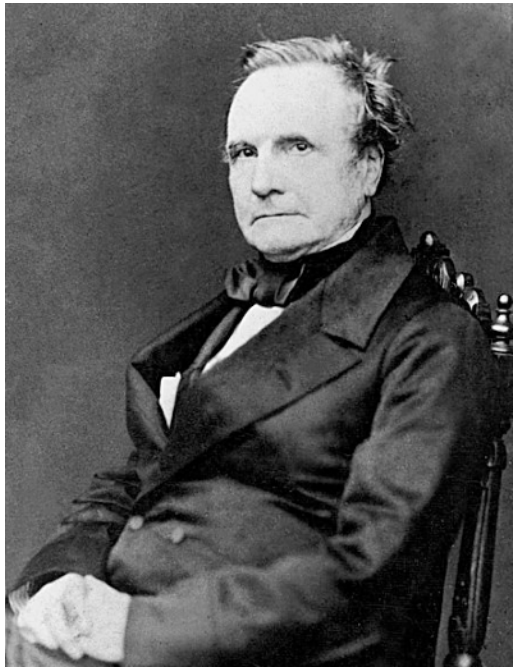
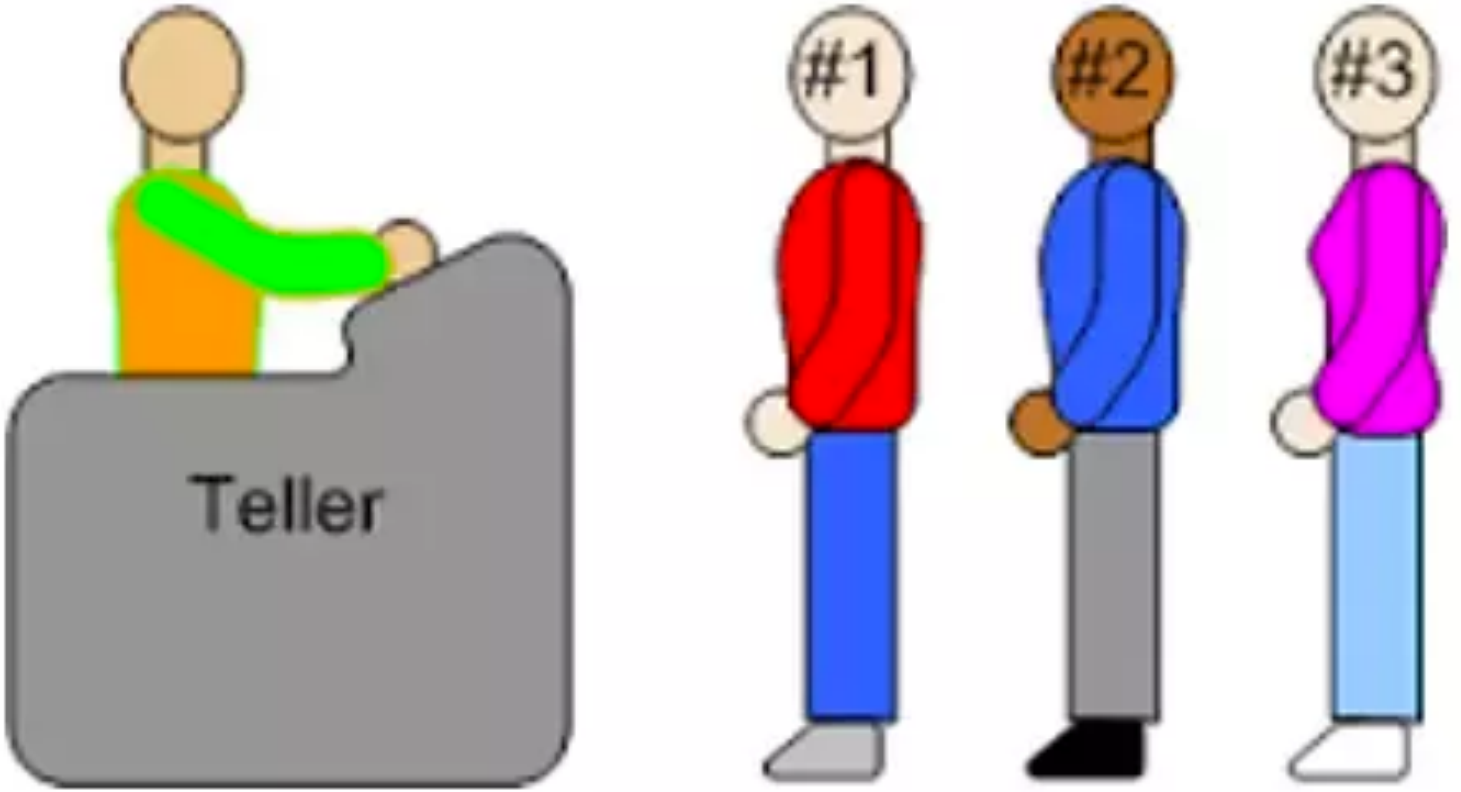
# CS206

## Queues



# Queues



# Queueing Theory

Agner Krarup Erlang

# `Queue` Interface

- `null` is returned from `peek()` and `poll()` when queue is empty
- return false from offer when cannot add to queue.

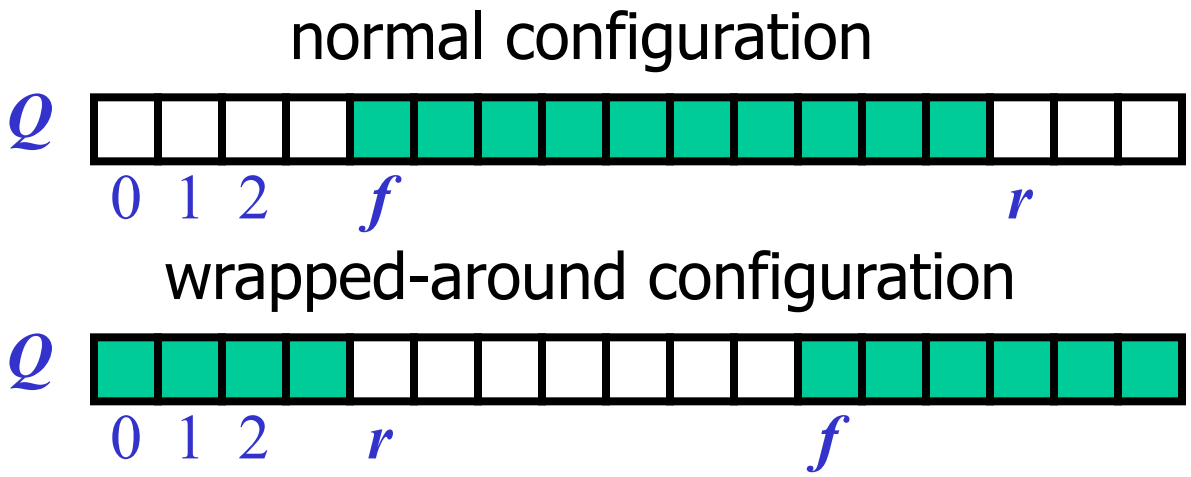
```
public interface QueueIntf<Q> {
    boolean isEmpty();
    int size();
    boolean add(Q q)
        throws IllegalStateException;
    Q remove()
        throws NoSuchElementException;
    Q element()
        throws NoSuchElementException;
    boolean offer(Q q);
    Q poll();
    Q peek();
}
```

# Example

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

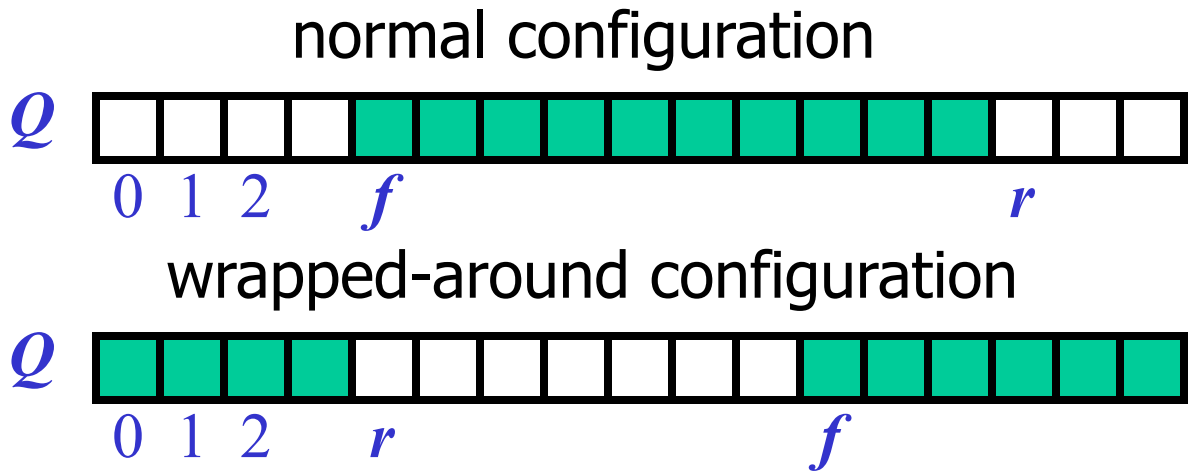
# Array-based Queue

- An array of size `n` in a circular fashion
  - `frontLoc`: index of the front element
    - where objects are read
  - `count`: number of stored elements
  - `rearLoc`: index of rear element
    - where objects are added



# Circular Array and Queue

- When the queue has fewer than `n` elements, location
  - `rearLoc =(frontLoc+count)%n`



# Start of Queue Implementation

```
public class ArrayQueue<Q> implements QueueInterface<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 number of items in the queue */
    private int count;
    /** The array location of the end of the queue  (ie the
     * location of the item shown by the peek command) */
    private int frontLoc;
    public ArrayQueue(int qSize) {
        count = 0;
        frontLoc = 0;
        backingArray = (Q[]) new Object[qSize];
    }
```

write add, remove

# 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
  - Adding to a full queue returns false (offer method)

# Queue Offer Method

`boolean offer(E e)`

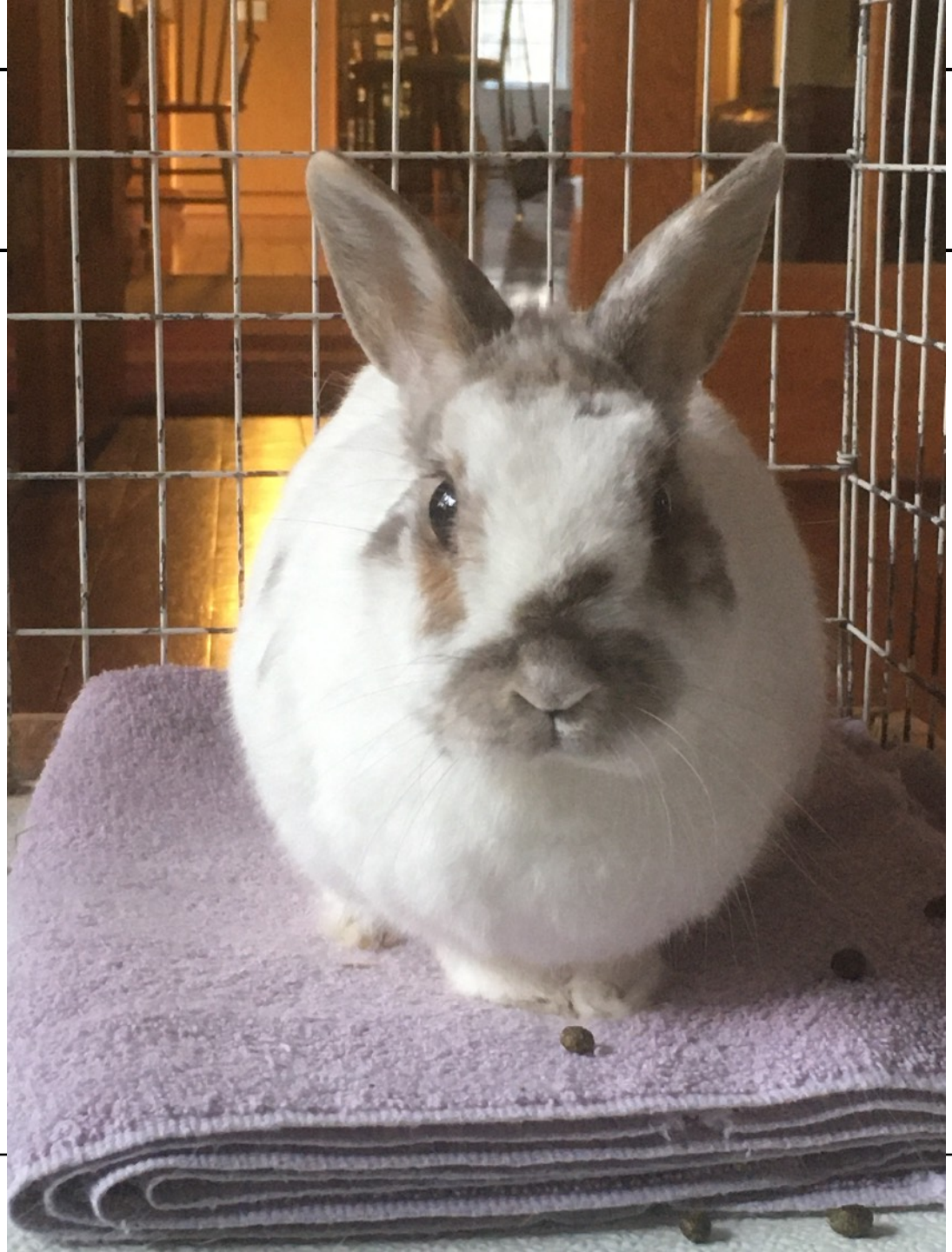
Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions. When using a capacity-restricted queue, this method is generally preferable to `add(E)`, which can fail to insert an element only by throwing an exception.

**Parameters:**

`e` - the element to add

**Returns:**

`true` if the element was added to this queue, else `false`



# Comparable Rabbit

```
public class Rabbit implements Comparable<Rabbit> {
    private final int iD;
    private final String nickname;
    public Rabbit(int id, String nn) {
        this.iD = id;
        this.nickname = nn==null ? makeName() : nn;
    }

// implement Comparable interface so that rabbits
// are sorted based on their nickname.  If the nickname
// is the same, then use id
```

# Putting this together

```
public class CompRabbits {
    public static void main(String[] args) {
        SAL<Rabbit> rsal = new SAL<>();
        rsal.add(new Rabbit(Rabbit.BreedEnum.Angora, 45, "Flopsy"));
        rsal.add(new Rabbit(Rabbit.BreedEnum.DwarfDutch, 46, "Mopsy"));
        rsal.add(new Rabbit(Rabbit.BreedEnum.FrenchLop, 47, "Cottontail"))
        rsal.add(new Rabbit(Rabbit.BreedEnum.Angora, 44, "Peter"));
        rsal.add(new Rabbit(Rabbit.BreedEnum.DwarfDutch, 10, "Josephine"))
        rsal.add(new Rabbit(Rabbit.BreedEnum.FrenchLop, 17, "Benjamin"));
        System.out.println(rsal);
    }
}
```