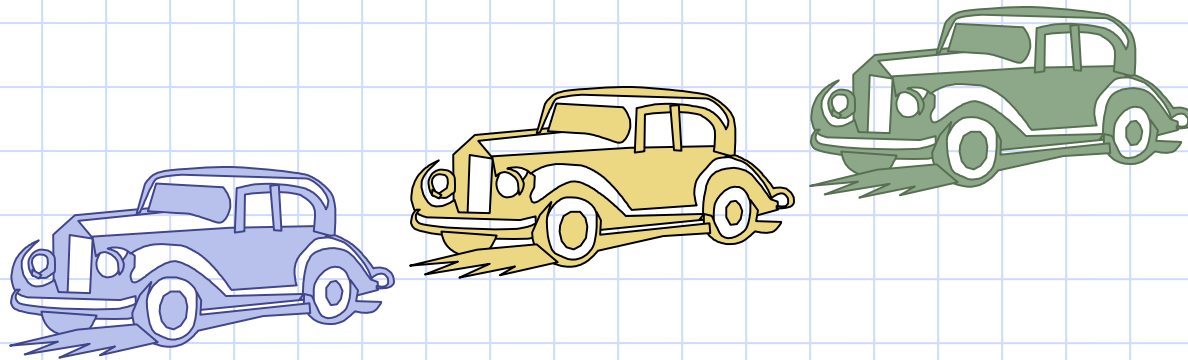


Queues



The Queue ADT

- The **Queue** ADT stores arbitrary objects
- Insertions and deletions follow the first-in first-out scheme
- Insertions are at the rear of the queue and removals are at the front of the queue
- Main queue operations:
 - **enqueue**(object): inserts an element at the end of the queue
 - object **dequeue**(): removes and returns the element at the front of the queue
- Auxiliary queue operations:
 - object **front**(): returns the element at the front without removing it
 - integer **size**(): returns the number of elements stored
 - boolean **isEmpty**(): indicates whether no elements are stored
- Exceptions
 - Attempting the execution of dequeue or front on an empty queue throws an **EmptyQueueException**

Example

<i>Operation</i>		<i>Output</i>	<i>Q</i>
enqueue(5)	–	(5)	
enqueue(3)	–	(5, 3)	
dequeue()	5	(3)	
enqueue(7)	–	(3, 7)	
dequeue()	3	(7)	
front()	7	(7)	
dequeue()	7	()	
dequeue()	“error”	()	
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)	

Applications of Queues

- ❑ Direct applications
 - Waiting lists, bureaucracy
 - Access to shared resources (e.g., printer)
 - Multiprogramming
- ❑ Indirect applications
 - Auxiliary data structure for algorithms
 - Component of other data structures

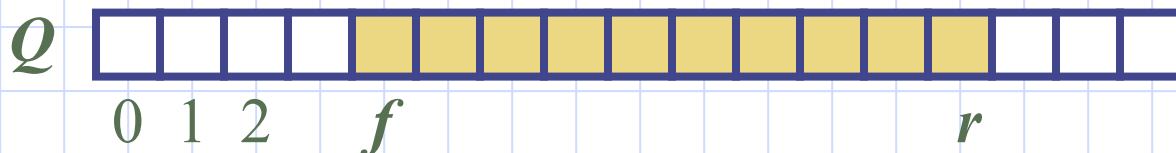
Array-based Queue

- Use an array of size N in a circular fashion
- Two variables keep track of the front and rear

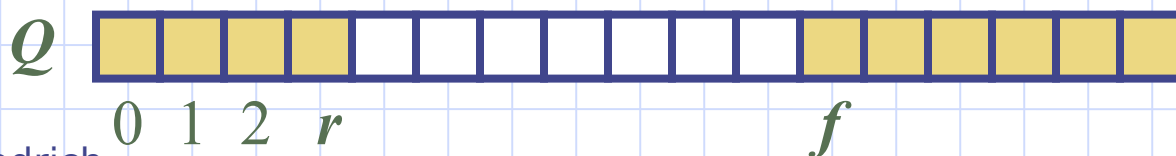
f index of the front element

r index immediately past the rear element

- Array location r is kept empty
normal configuration



wrapped-around configuration



Queue Operations

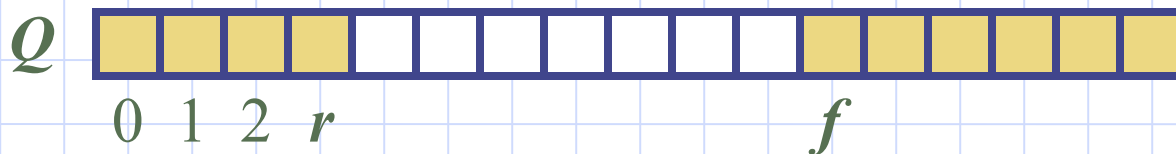
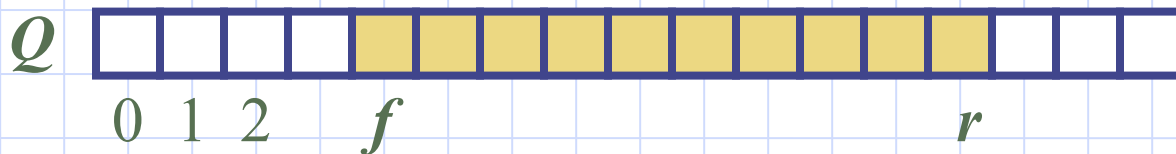
- We use the modulo operator (remainder of division)

Algorithm *size()*

```
if ( $r \geq 0$  &&  $r < f$ )  
    return  $(N - f) + (r + 1)$ ;  
return  $(r + 1) - f$ ;
```

Algorithm *isEmpty()*

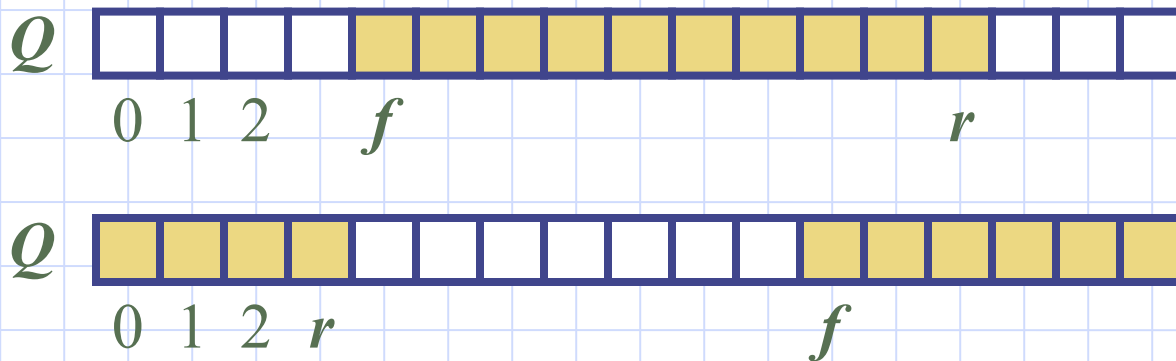
```
return  $size() == 0$ 
```



Queue Operations (cont.)

- ❑ Operation enqueue throws an exception if the array is full
- ❑ This exception is implementation-dependent

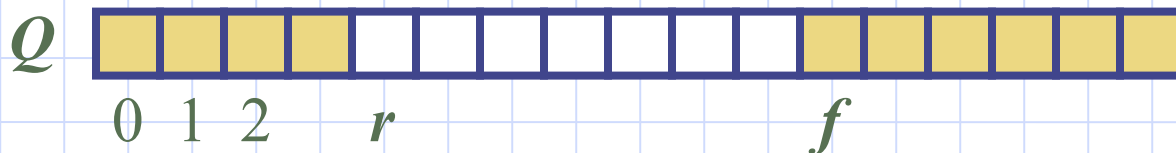
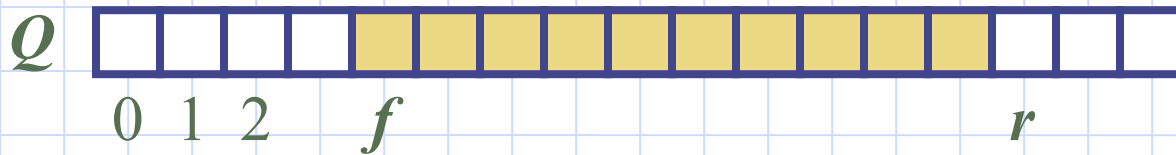
```
Algorithm enqueue(o)  
  if size() =  $N$   
    throw FullQueueException  
  else  
     $r \leftarrow (r + 1) \bmod N$   
     $Q[r] \leftarrow o$ 
```



Queue Operations (cont.)

- ❑ Operation `dequeue` throws an exception if the queue is empty
- ❑ This exception is specified in the queue ADT

```
Algorithm dequeue()  
  if isEmpty() then  
    throw EmptyQueueException  
  else  
     $o \leftarrow Q[f]$   
     $f \leftarrow (f + 1) \bmod N$   
    return  $o$ 
```



Queue Interface in Java

- ❑ Java interface corresponding to our Queue ADT
- ❑ Requires the
- ❑ definition of class `EmptyQueueException`
- ❑ No corresponding built-in Java class

```
public interface Queue<E> {  
    public int size();  
    public boolean isEmpty();  
    public E front()  
        throws EmptyQueueException;  
    public void enqueue(E element);  
    public E dequeue()  
        throws EmptyQueueException;  
}
```

Application: Round Robin Schedulers

- We can implement a round robin scheduler using a queue Q by repeatedly performing the following steps:
 1. `e = Q.dequeue()`
 2. Service element `e`
 3. `Q.enqueue(e)`

