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 first(): 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
- Boundary cases:
  - Attempting the execution of dequeue or first on an empty queue returns null

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)	

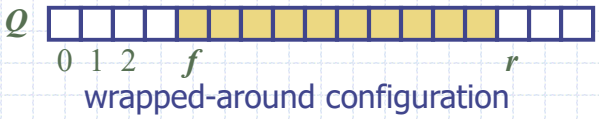
Applications of Queues

- 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

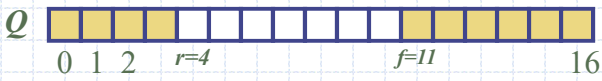
Array-based Queue

- Use an array of size N in a circular fashion
- Two variables keep track of the front and size
  - f index of the front element
  - sz number of stored elements
- When the queue has fewer than N elements, array location $r = (f + sz) \bmod N$ is the first empty slot past the rear of the queue

normal configuration



wrapped-around configuration



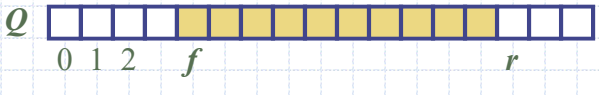
5

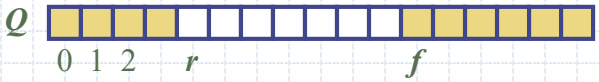
Queue Operations

- We use the modulo operator (remainder of division)

```
Algorithm size()
  return sz

Algorithm isEmpty()
  return (sz == 0)
```



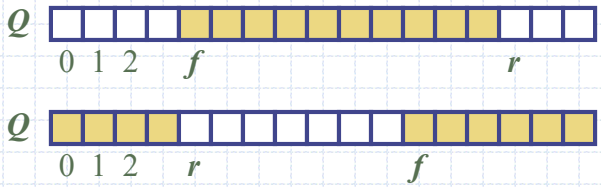


6

Queue Operations (cont.)

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

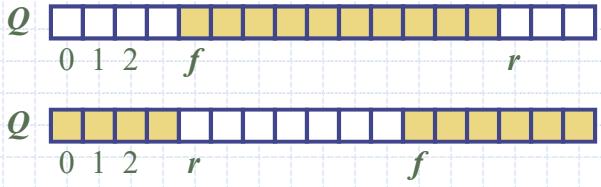
```
Algorithm enqueue(o)
  if size() = N then
    throw IllegalStateException
  else
    r ← (f + sz) mod N
    Q[r] ← o
    sz ← (sz + 1)
```



Queue Operations (cont.)

- Note that operation dequeue returns null if the queue is empty

```
Algorithm dequeue()
  if isEmpty() then
    return null
  else
    o ← Q[f]
    f ← (f + 1) mod N
    sz ← (sz − 1)
    return o
```



Queue Interface in Java

- Java interface corresponding to our Queue ADT
- Assumes that first() and dequeue() return null if queue is empty

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

9

Array-based Implementation

```
1  /** Implementation of the queue ADT using a fixed-length array. */
2  public class ArrayQueue<E> implements Queue<E> {
3    // instance variables
4    private E[ ] data;                    // generic array used for storage
5    private int f = 0;                    // index of the front element
6    private int sz = 0;                   // current number of elements
7
8    // constructors
9    public ArrayQueue( ) {this(CAPACITY);} // constructs queue with default capacity
10   public ArrayQueue(int capacity) {     // constructs queue with given capacity
11     data = (E[ ]) new Object[capacity]; // safe cast; compiler may give warning
12   }
13
14   // methods
15   /** Returns the number of elements in the queue. */
16   public int size( ) { return sz; }
17
18   /** Tests whether the queue is empty. */
19   public boolean isEmpty( ) { return (sz == 0); }
20
```

10

Array-based Implementation (2)

```
21   /** Inserts an element at the rear of the queue. */
22   public void enqueue(E e) throws IllegalStateException {
23     if (sz == data.length) throw new IllegalStateException("Queue is full");
24     int avail = (f + sz) % data.length;      // use modular arithmetic
25     data[avail] = e;
26     sz++;
27   }
28
29   /** Returns, but does not remove, the first element of the queue (null if empty). */
30   public E first( ) {
31     if (isEmpty( )) return null;
32     return data[f];
33   }
34
35   /** Removes and returns the first element of the queue (null if empty). */
36   public E dequeue( ) {
37     if (isEmpty( )) return null;
38     E answer = data[f];
39     data[f] = null;                          // dereference to help garbage collection
40     f = (f + 1) % data.length;
41     sz--;
42     return answer;
43   }
```

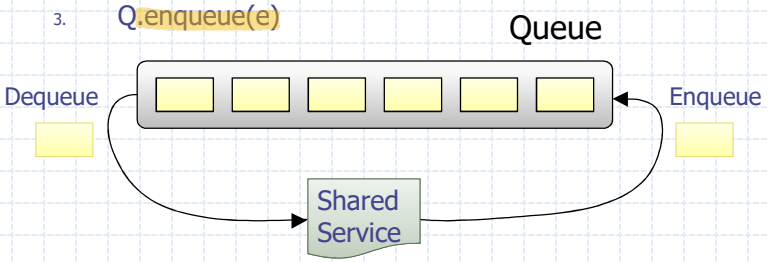
↳ Se siamo in fondo deve tornare all'inizio

Linked_List-based Queue in Java

```
public class LinkedQueue<E> implements Queue<E> {
  private SinglyLinkedList<E> list = new SinglyLinkedList<>();
  /** Constructs an initially empty queue. */
  public LinkedQueue() { }
  public int size() { return list.size(); }
  public boolean isEmpty() { return list.isEmpty(); }
  public void enqueue(E element) { list.addLast(element); }
  public E first() { return list.first(); }
  public E dequeue() { return list.removeFirst(); }
}
```

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)



13

DoubleQueue Interface in Java

```
public interface Deque<E> {

  /**
   * Returns the number of elements in the deque.
   * @return number of elements in the deque
   */
  int size();

  /**
   * Tests whether the deque is empty.
   * @return true if the deque is empty, false otherwise
   */
  boolean isEmpty();

  /**
   * Returns (but does not remove) the first element of the deque.
   * @return first element of the deque (or null if empty)
   */
  E first();

  /**
   * Returns (but does not remove) the last element of the deque.
   * @return last element of the deque (or null if empty)
   */
  E last();

  /**
   * Inserts an element at the front of the deque.
   * @param e   the new element
   */
  void addFirst(E e);

  /**
   * Inserts an element at the back of the deque.
   * @param e   the new element
   */
  void addLast(E e);

  /**
   * Removes and returns the first element of the deque.
   * @return element removed (or null if empty)
   */
  E removeFirst();

  /**
   * Removes and returns the last element of the deque.
   * @return element removed (or null if empty)
   */
  E removeLast();
}
```

DoubleQueue implementation

```
public class DoublyLinkedList<E> implements Deque<E> {

                    Nothing to add!!!!


}
```

15