QUEUE ADT

Metodi

createQueue
// effect: Create an empty queue

isEmpty( )
// effect: Determines if a queue is empty

getFront( )
// effect: Returns, but does not remove, the head of the queue.
// effect: Throws QueueException if the queue is empty.

enqueue(newElem)
// effect: Inserts newElem at the back of the queue, if there is no violation of
// effect: capacity. Throws QueueException if the queue is full.

dequeue( )
// effect: Retrieves and removes the head of the queue. Throws QueueException
// effect: if the queue is empty.

Assiomi

1. (aQueue.createQueue( )).isEmpty = true
2. (aQueue.enqueue(Elem)).isEmpty() = false
3. (aQueue.createQueue( )).getFront( ) = error
4. pre: aQueue.isEmpty() = true:
   (aQueue.enqueue(Elem)).getFront( ) = Elem
5. (aQueue.createQueue( )).dequeue( ) = error
6. pre: aQueue.isEmpty( )=false:
   (aQueue.enqueue(Elem)).dequeue( )=(aQueue. dequeue( )). enqueue(Elem)

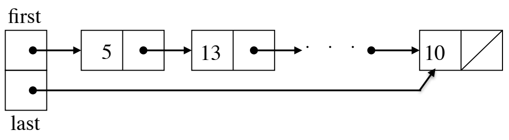
ADT QUEUE

```
public interface Queue<T>{
    public boolean isEmpty();
    //Requires: none
    //Effect: Returns true if the queue is empty, otherwise returns false.
    public void enqueue(T elem) throws QueueException;
    //Require: none
    //Effect: If insertion is successful, item is at the end of the queue.
    //Effect: Throws QueueException if the item cannot be added to the queue.
    public T getFront( ) throws QueueException;
    //Require: none
    //Effect: If queue is not empty, the item at the front of a queue is returned, and the queue
    //Effect: is left unchanged. Throws QueueException if the queue is empty.
    public T dequeue( ) throws QueueException;
    //Require: none
    //Effect: If queue is not empty, the item at the front of the queue is retrieved and removed
    //Effect: from the queue. Throws QueueException if the queue is empty.
}//end of interface
```

QUEUE Exception

```
public class QueueException extends java.lang.RuntimeException {
        public QueueException(String s){
                super(s);
        }// end constructor
}// end QueueException
```

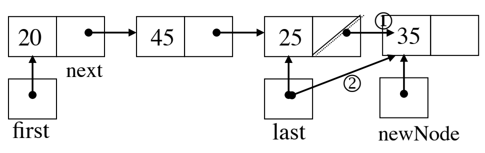
Implementazione Dinamica



```
class Node<T>{
  private T element;
  private Node<T> next;
  …………
}
```

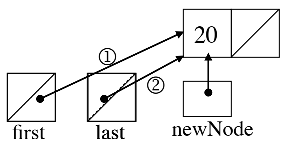
```
class LinkedBasedQueue<T>{
    private Node<T> first;
    private Node<T> last;
      …………
}
```

Inserimento di un nuovo nodo



1. last.setNext(newNode);
2. last = newNode;

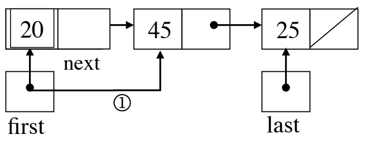
non-empty



1. first = newNode;
2. last = newNode;

Empty

eliminazione



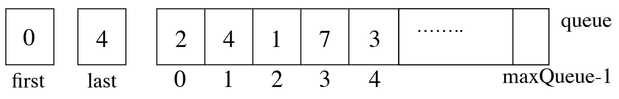
1. first = first.getNext( );

non-empty

Il codice Java

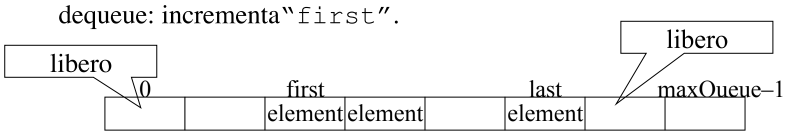
```
public class LinkedBasedQueue<T> implements Queue<T> {

    private Node<T> first;
    private Node<T> last;

    public void enqueue(T elem) {
       Node<T> newNode = new Node(elem);
       if (isEmpty( )) {                   // insertion into empty queue
           first = newNode;                // new node is referenced by first
           last = newNode;
       }
       else {  last.setNext(newNode);      // insertion into non-empty queue
               last = newNode; }
    }

    public T dequeue( )throws QueueException{
       if (!isEmpty( )) {
           T result = first.getElem();
           (first == last) {last = null;}
           first = first.getNext( );
           return result; }
       else{ throw new QueueException("QueueException on dequeue: empty");}
    }
    .........
}
```

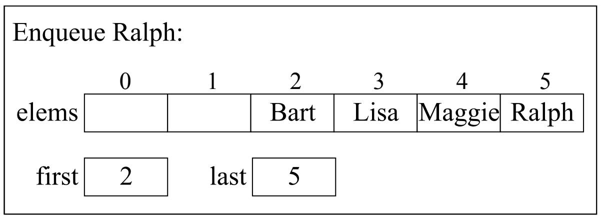
Implementazione con array



```
public class QueueArrayBased<T> implements Queue<T>{

      private final int maxQueue = 50;
      private T[ ] queue;
      private int first, last;
      .........
 }
```

enqueue: incrementa“`last`” e inserisce l’elemento in `queue[last]`.
dequeue: incrementa“`first`”.



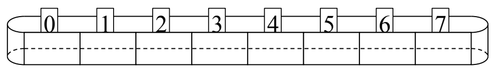


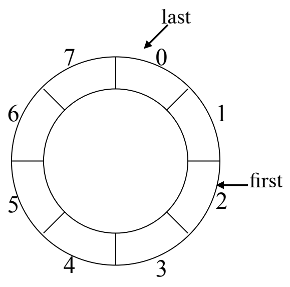
Quale e’ il problema??.

- Usare Array circolari !!!

Array Circolare

- Il successore di $a[n-1] = a[0]$
- Il predecessore di $a[0] = a[n-1]$.





Enqueue:
```
last=(last+1)%maxQueue;
queue[last] = newElem;
  ++count;
```

Dequeue:
```
first = (first+1)%maxQueue;
--count;
```

Invariante di rappresentazione

- Typical elements:
  queue[*first*…*last*] **oppure**
  queue[*first*…*maxQueue*–1] and *queue*[0…*last*].

Invariant:

0		first			last		maxQueue–1
		element	element		element		

or:

0		last			first		maxQueue–1
element		element			element		element

Eliminare l’elemento fron:

	0	1	2	3	4	5
queue	Nelson	Martin		Lisa	Maggie	Ralph

first	3	last	1	count	5

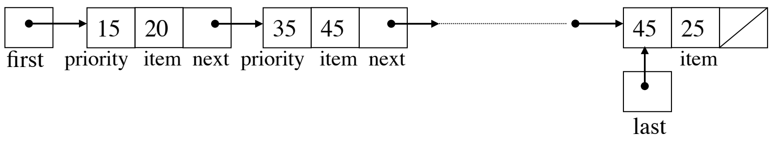
A Implementazione

```
public class QueueArrayBased<T> implements Queue<T>{

   private final int maxQueue = 50;
   private T[ ] queue;
   private int first, last, count;

   public QueueArrayBased( ){
       queue = (T[ ]) new Object[maxQueue];
       first = 0; count = 0; last = maxQueue-1;
   }

   public void enqueue(T newElem) throws QueueException {
       if (!isFull( )){
          last = (last+1) % maxQueue;
          queue[last] = newElem;
          count++;
       }
       else {throw new QueueException("Queue is full");}
   }
```

```
   public T dequeue( ) throws QueueException {
       if (!isEmpty( ))
          { T queuefront = queue[first];
             first =(first+1)% maxQueue;
             count--;
             return queuefront;
           }
       else {throw new QueueException("Queue is empty");}
   }

   private boolean isFull( )
   {
         return count = = maxQueue};
   }
```

ADT Priority Queue

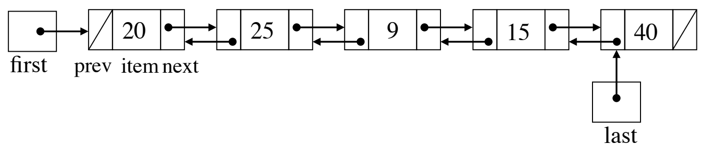
- **ADT Priority Queue** e’ una sequenza di elementi che sono ordinati
- Le procedure di accesso inseriscono e eliminao in base alla priorita’.
- Typical element:



```
public interface PriorityQueue<T>{

    public boolean isEmpty();
    //Require: none
    //Effect: Returns true if the queue is empty, otherwise returns false.

    public void add(int priority, T elem) throws QueueException;
    //Effect: If insertion is successful, elem is added to the queue in priority order.
    //Effect: Throws QueueException if elem cannot be added to the queue.

    public T peek( ) throws QueueException;
    //Effect: If queue is not empty, the element with highest priority value is returned,
    //Effect: and the queue is left unchanged. Throws QueueException if the queue is empty.

    public T remove( ) throws QueueException;
    //Effect: If queue is not empty, the element with highest priority is retrieved and
    //Effect: removed from the queue. Throws QueueException if the queue is empty.
}
```

ADT Double-Ended Queue

- **ADT Double-ended Queue** e’ una coda con meccanismo di accesso a entrambi I lati



```
public interface DEQueue<T>{
    public boolean isEmpty();
    //Effect: Returns true if the queue is empty, otherwise returns false.
    public void addToBack(T elem) throws QueueException;
    //Effect: Item is added at the end of the queue.  Throws QueueException if the item cannot be added.
    public void addToFront(T elem) throws QueueException;
    //Post: Item is added at the front of the queue. Throws QueueException if the item cannot be added.
    public T removeFront( ) throws QueueException;
    //Effect: If queue is not empty, the item at the front of a queue is retrieved and removed
    //        from the queue. Throws QueueException if the queue is empty.
    public T removeBack( ) throws QueueException;
    //Effect: If queue is not empty, the item at the back of a queue is retrieved and removed
    //        from the queue. Throws QueueException if the queue is empty.
    public T getFront( ) throws QueueException;
    //Effect: If queue is not empty, the item at the front of the queue is retrieved without changing the
    //        queue. Throws QueueException if the queue is empty.
    public T getBack( ) throws QueueException;
    //Effect: If queue is not empty, the item at the back of the queue is retrieved and removed
    //        from the queue. Throws QueueException if the queue is empty.  }
```