Colas

Carlos Delgado Kloos

Dep. Ingeniería Telemática

Univ. Carlos III de Madrid

Ejemplo

- La cola del autobús
- La cola de la impresora

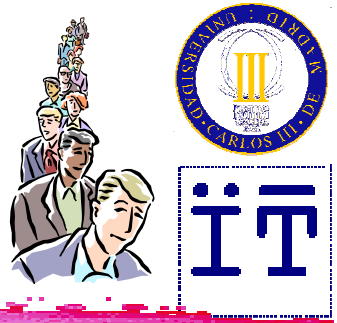
Características

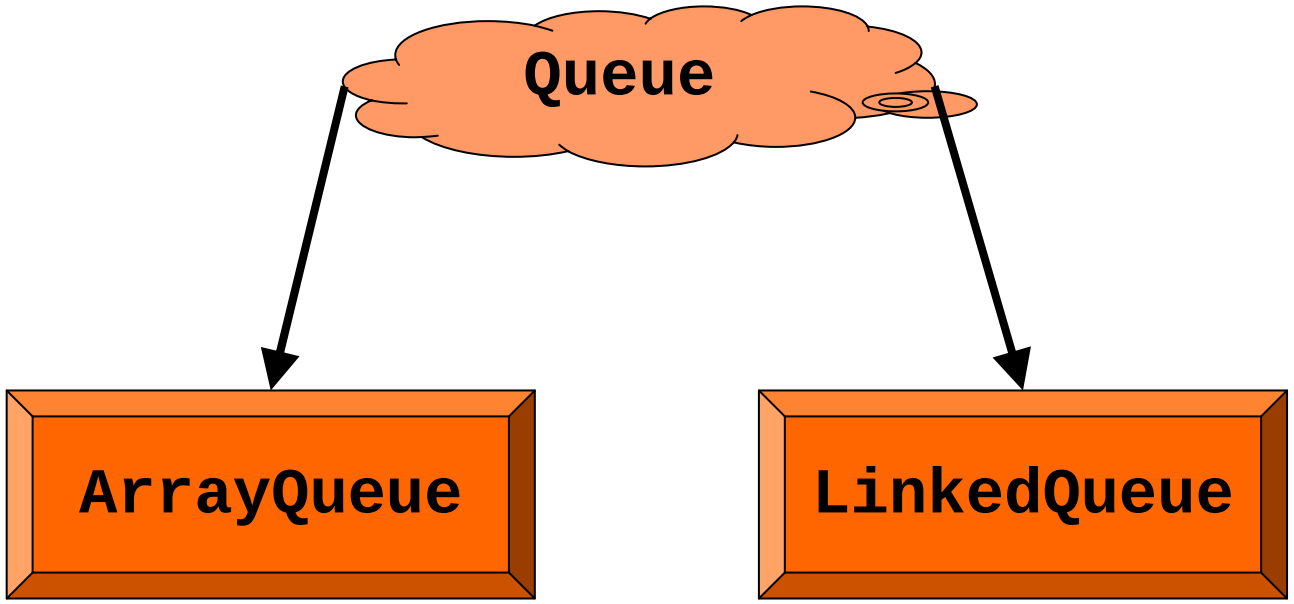
- Estructura lineal
- Acceso de inserción por un extremo y de eliminación por el otro extremo

Interfaz para colas

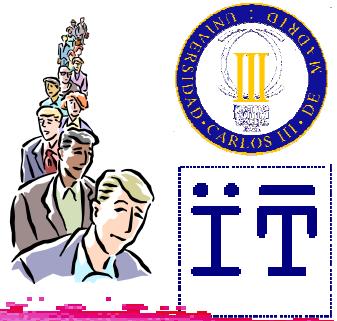
```
public interface Queue {
   public int size();
   public boolean isEmpty();
   public void enqueue(Object o)
      throws QueueOverflowException;
   public Object dequeue()
      throws EmptyQueueException;
   public Object front()
      throws EmptyQueueException;
}
```

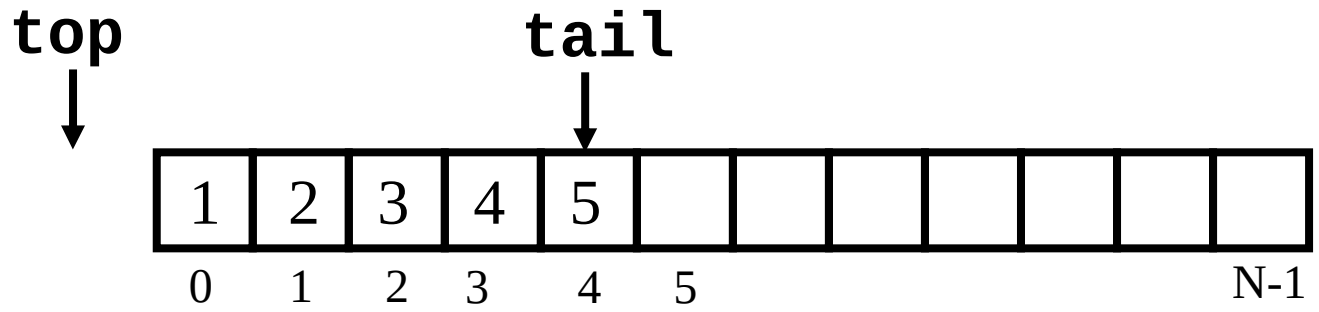
Un interfaz y varias implementaciones

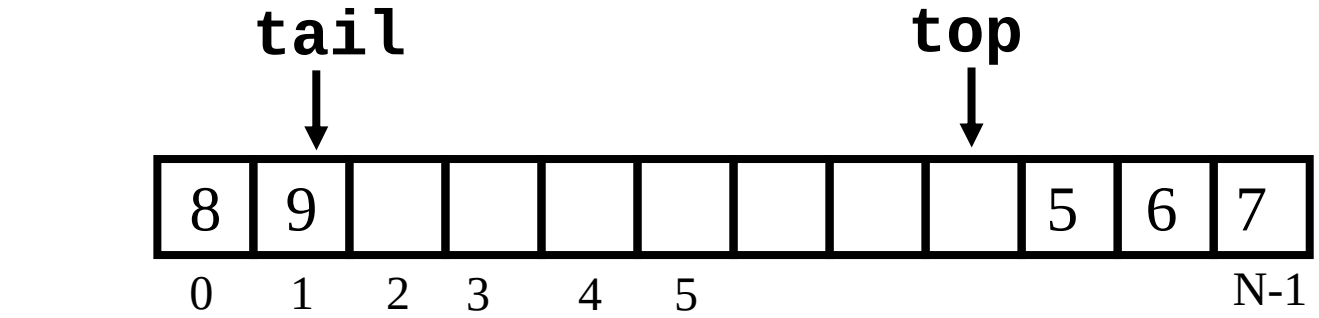




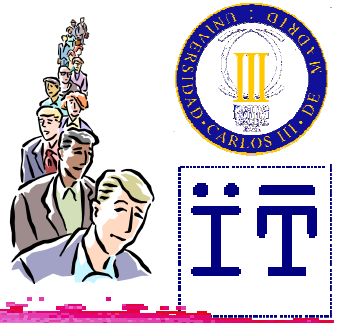
Implementación basada en arrays

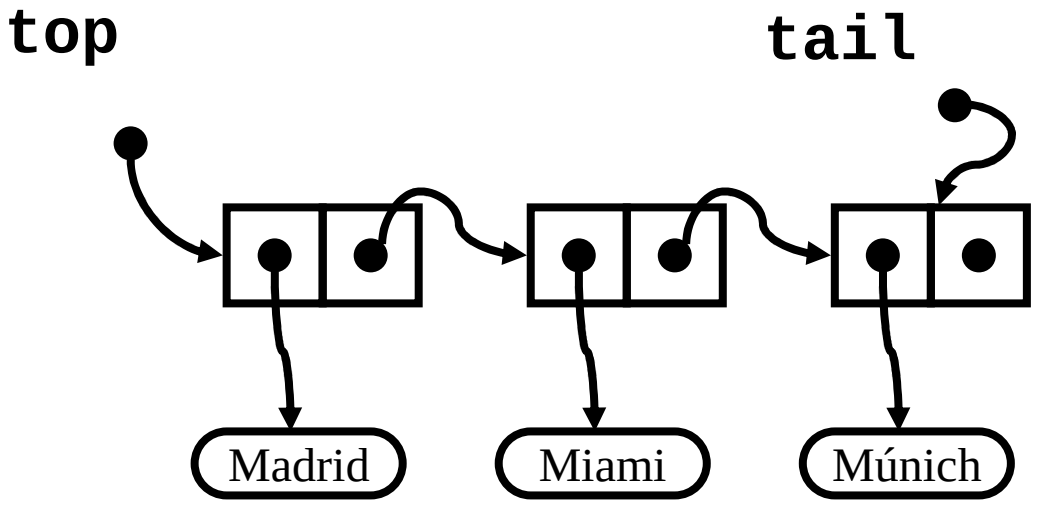




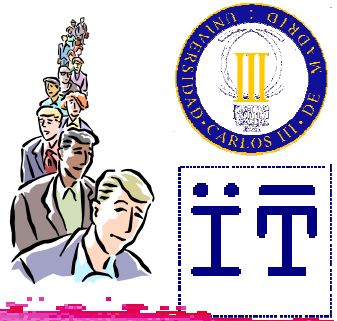


Implementación basada en listas encadenadas



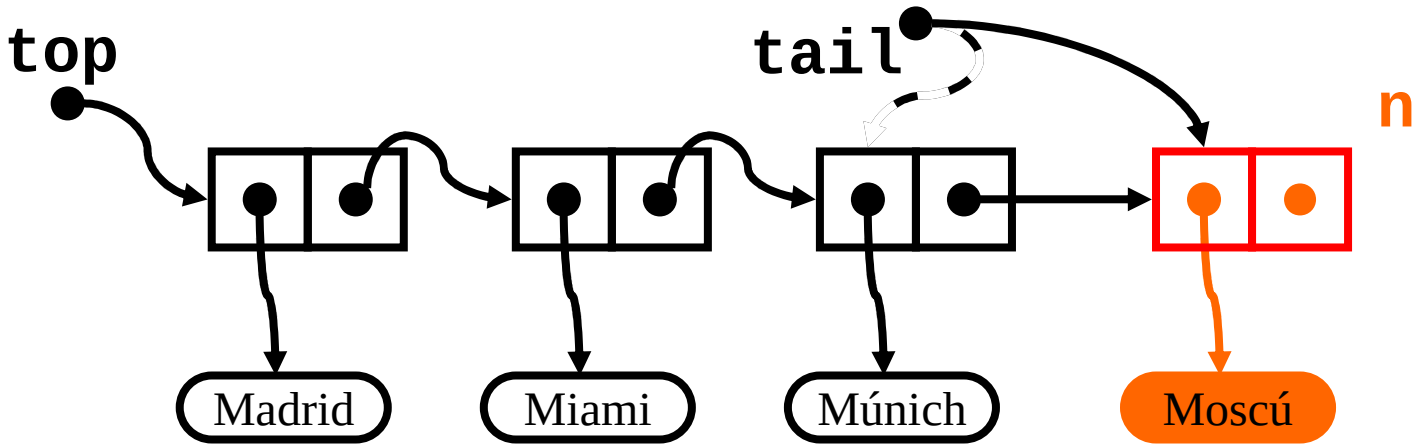


Implementación basada en listas encadenadas



```
public class LinkedQueue implements Queue {
  private Node top = null;
  private Node tail = null;
  private int size = 0;

  (…)
}
```

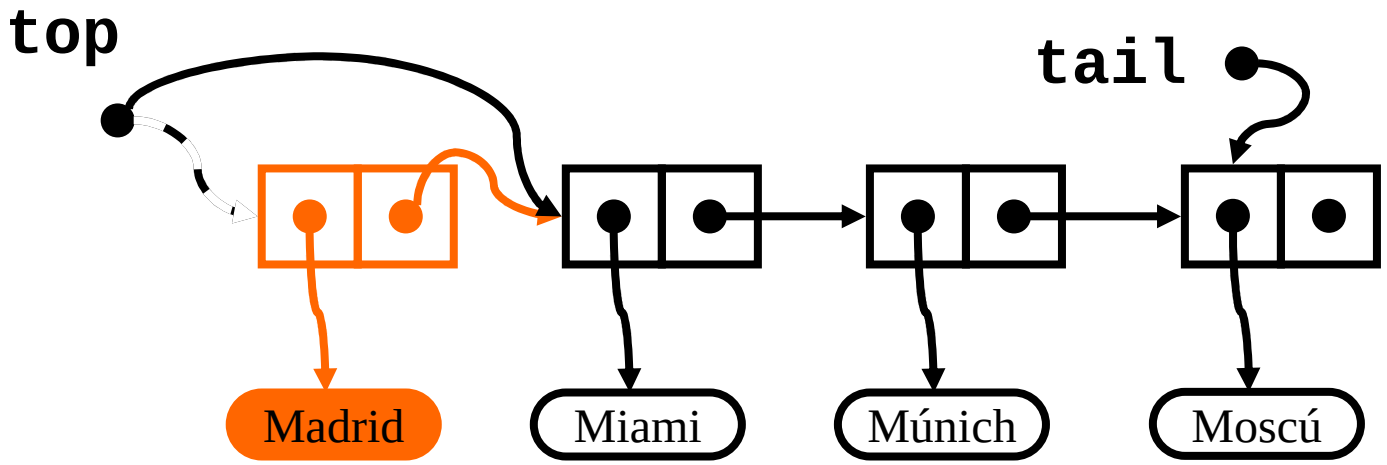
Inserción (enqueue)



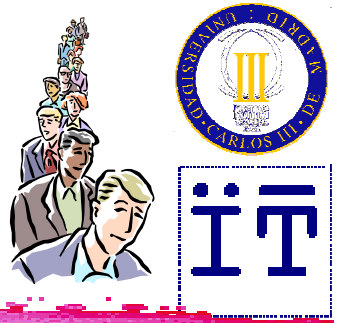
Implementación basada en listas encadenadas

```
public void enqueue(Object info) {
  Node n = new Node(info, null);
  if (top == null)
    top = n;
  else
    tail.setNext(n);
  tail = n;
  size++;
}
```

Borrado (dequeue)



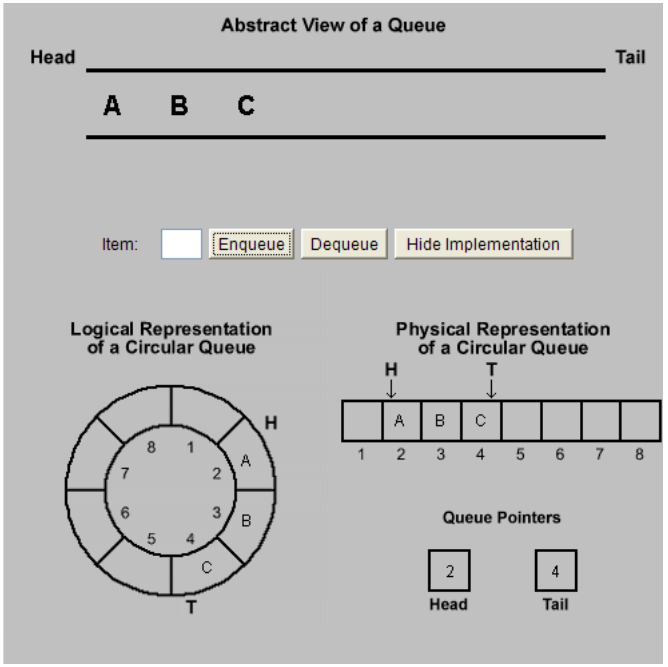
Implementación basada en listas encadenadas



```
public Object dequeue()
  throws EmptyQueueException {
  Object info;
  if (top == null)
    throw new EmptyQueueException();
  info = top.getInfo();
  top = top.getNext();
  if (top == null)
    tail = null;
  size--;
  return info;
}
```

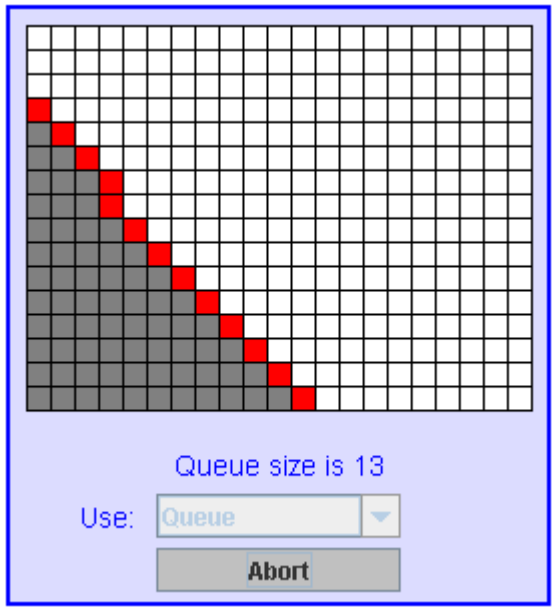
Actividad

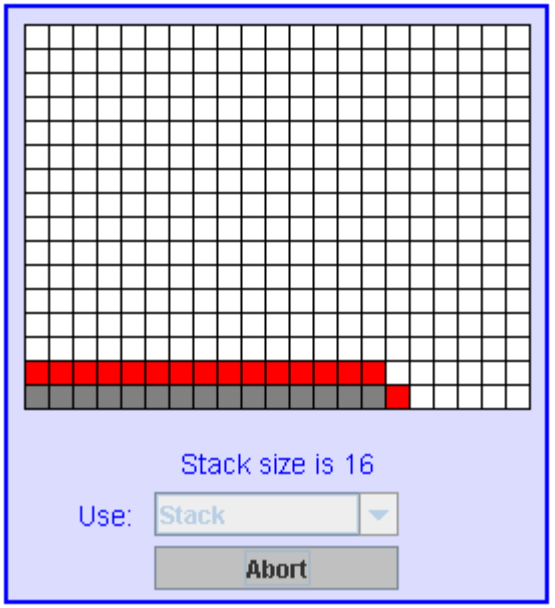
- Ver animaciones de colas:
- **http://courses.cs.vt.edu/csonline/DataStructures/Lessons/QueuesImplementationView/applet.html**



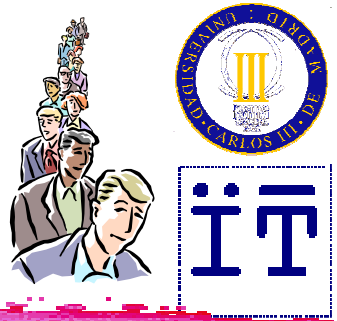
Actividad

- Probar el applet **`DepthBreadth.java`** que se encuentra en
  `http://www.faqs.org/docs/javap/c11/s3.html`



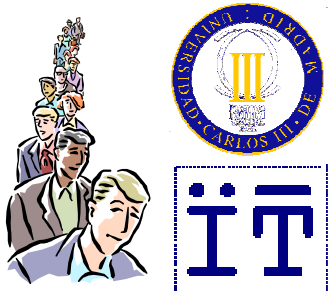


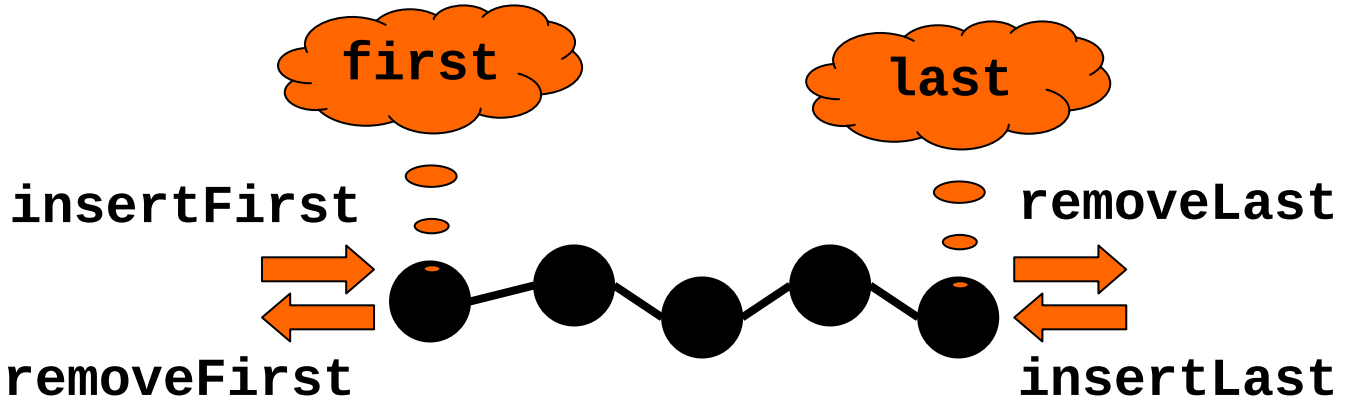
Otros tipos de colas (que ya no son colas)



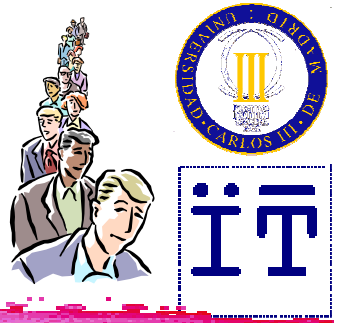
- Colas dobles
- Colas con prioridad

Deques (colas dobles)





Interfaz para colas dobles



```
public interface Deque {

   public int size();
   public boolean isEmpty();

   public void insertFirst(Object info);
   public void insertLast(Object info);
```

Interfaz para colas dobles

```
public Object removeFirst()
  throws EmptyDequeException;
public Object removeLast()
  throws EmptyDequeException;

public Object first()
  throws EmptyDequeException;
public Object last()
  throws EmptyDequeException;
```

Pilas y colas como deques

Stack	Deque
`size()`	`size()`
`isEmpty()`	`isEmpty()`
`top()`	`last()`
`push(e)`	`insertLast(e)`
`pop()`	`removeLast()`

Queue	Deque
`size()`	`size()`
`isEmpty()`	`isEmpty()`
`front()`	`first()`
`enqueue(e)`	`insertLast(e)`
`dequeue()`	`removeFirst()`

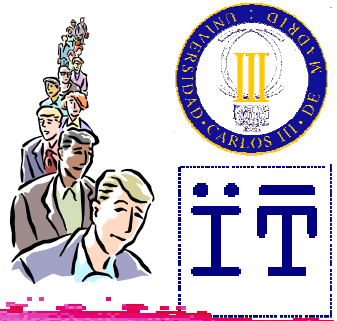
Definición de pilas a partir de deques

```
public class DequeStack implements Stack {
  private Deque deque;
  public DequeStack() {
    deque = new Deque();
  }
  public int size() {
    return deque.size();
  }
  public boolean isEmpty() {
    return deque.isEmpty();
  }
```

Definición de pilas a partir de deques

```
public void push(Object info) {
  deque.insertLast(info);
}
public Object pop()
  throws EmptyStackException {
  try {
    return deque.removeLast();
  } catch (EmptyDequeException ede) {
    throw new EmptyStackException();
  }
}
```

Definición de pilas a partir de deques



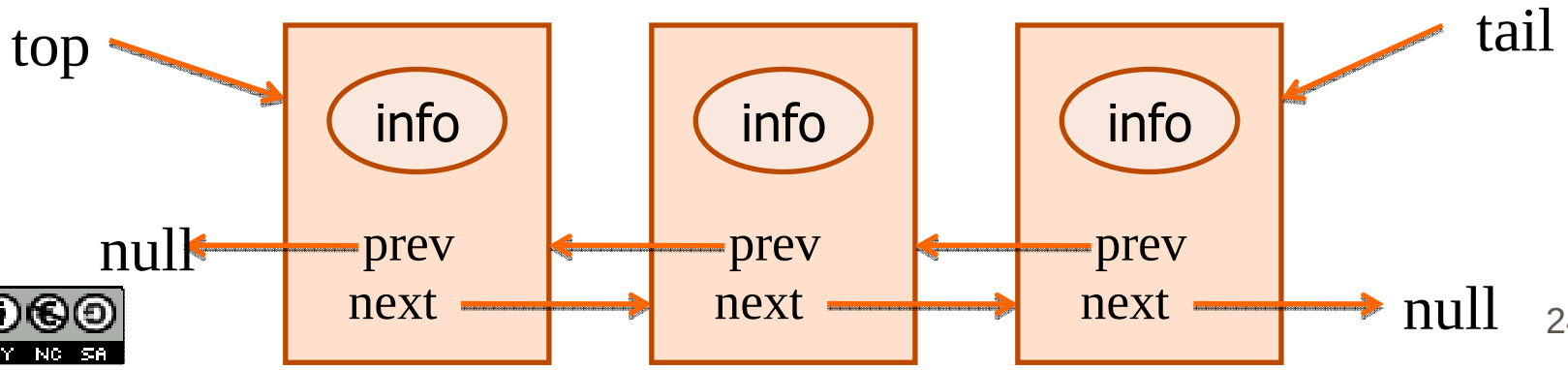
```
public Object top()
   throws EmptyStackException {
   try {
      return deque.last();
   } catch (EmptyDequeException ede) {
      throw new EmptyStackException();
   }
}
```

Implementación de deques basada en listas

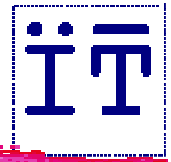
- Las listas enlazadas no son apropiadas porque `removeLast` necesita recorrer la lista completa para obtener la referencia del penúltimo
- Solución: listas doblemente enlazadas

Listas doblemente enlazadas

- Listas enlazadas en que cada nodo, además de almacenar el dato y la referencia del siguiente nodo, almacena también la referencia del nodo anterior
  - Permiten recorrer la lista en ambos sentidos
  - Reducen el coste de extracción del último nodo

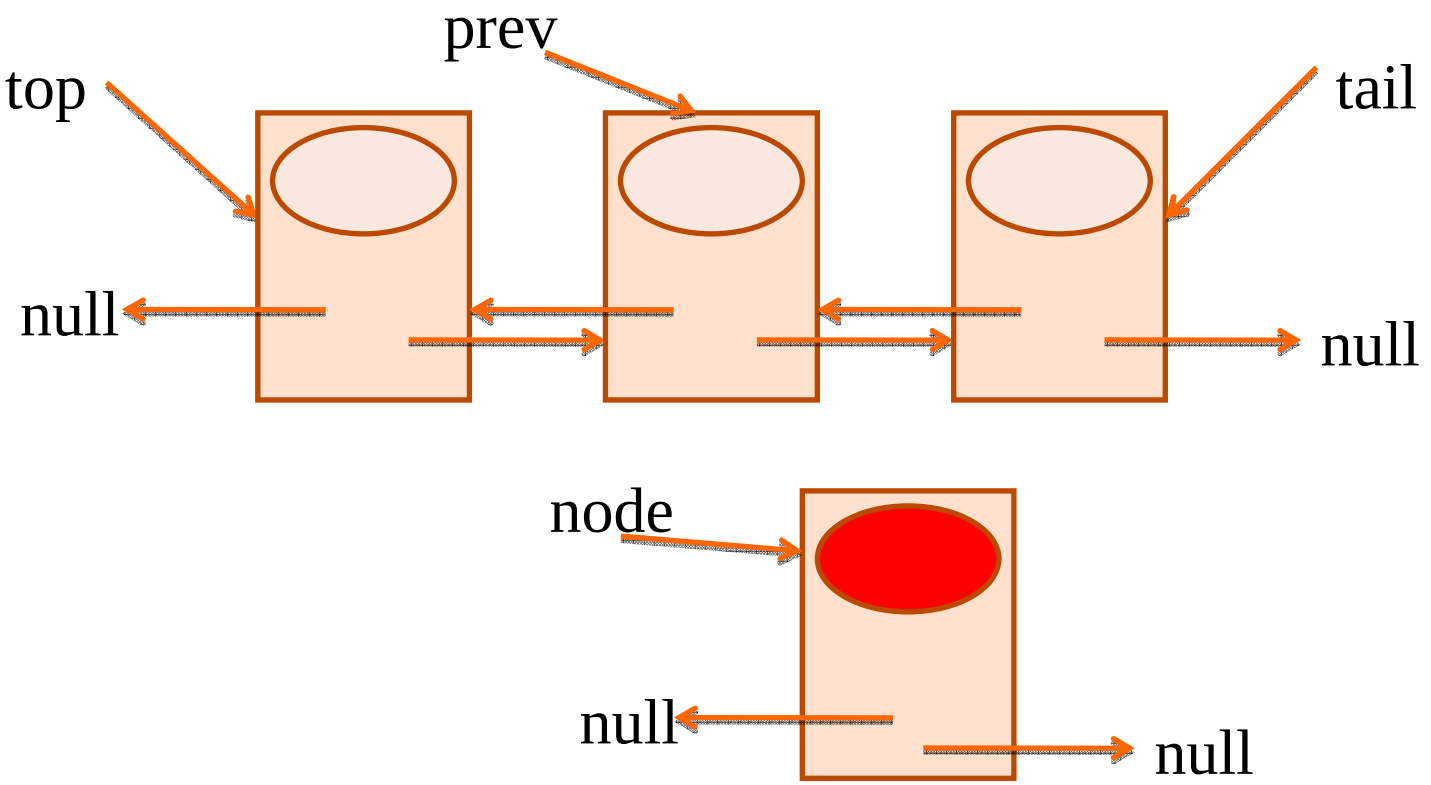


La clase DLNode



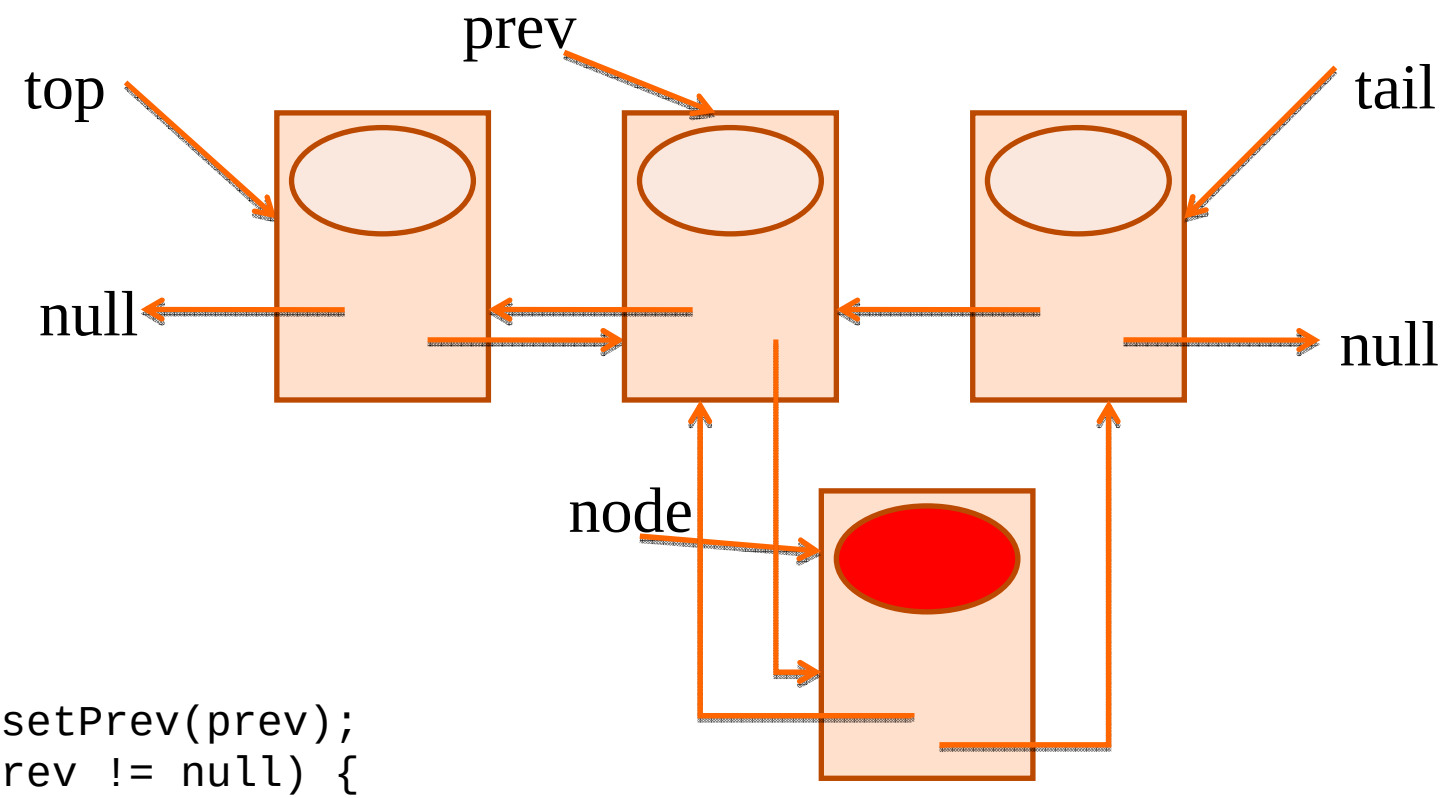
```
Public class DLNode {
    private Object info;
    private DLNode next;
    private DLNode prev;

    public DLNode(Object info) {…}
    public DLNode(Object info, DLNode prev, DLNode next) {…}

    public DLNode getNext() {…}
    public void setNext(DLNode next) {…}
    public DLNode getPrev() {…}
    public void setPrev(DLNode prev) {…}
    public Object getInfo() {…}
    public void setInfo(Object info) {…}
}
```

Inserción en cualquier posición



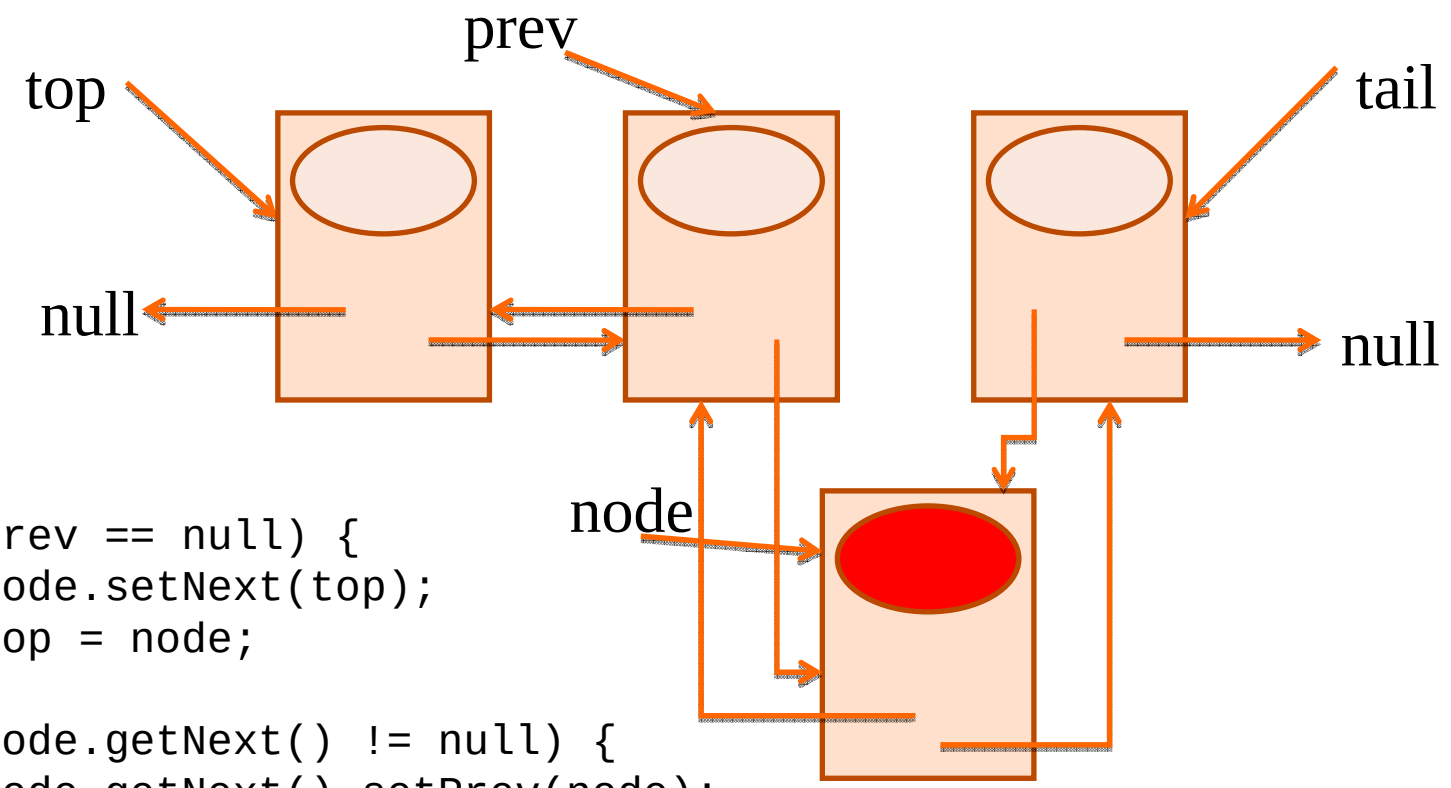

```
DLNode node = new DLNode(data);
```

Inserción en cualquier posición




```
node.setPrev(prev);
if (prev != null) {
    node.setNext(prev.getNext());
    prev.setNext(node);
}
```

Inserción en cualquier posición

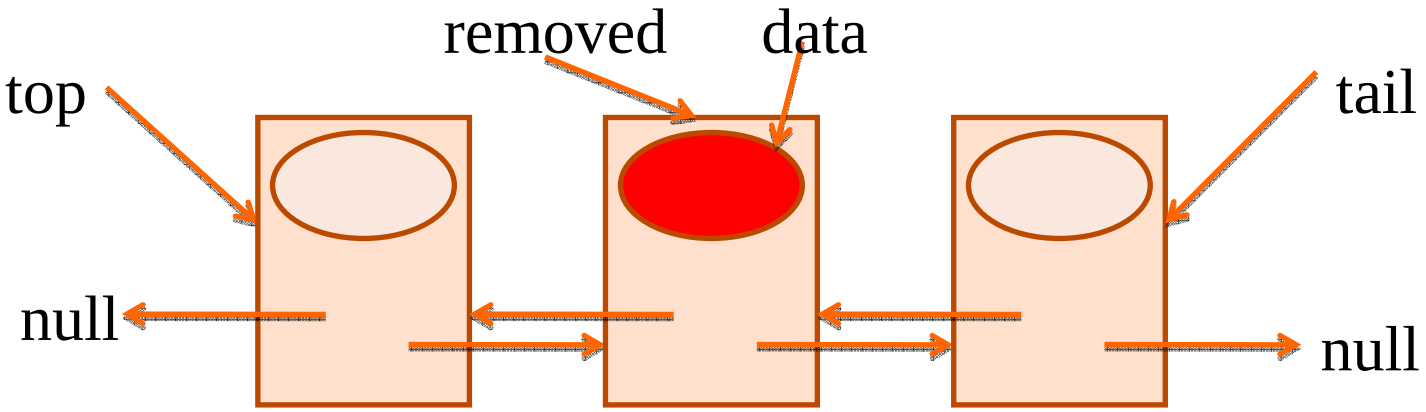



```
if (prev == null) {
    node.setNext(top);
    top = node;
}
if (node.getNext() != null) {
    node.getNext().setPrev(node);
} else {
    tail = node;
}
```

Inserción en cualquier posición

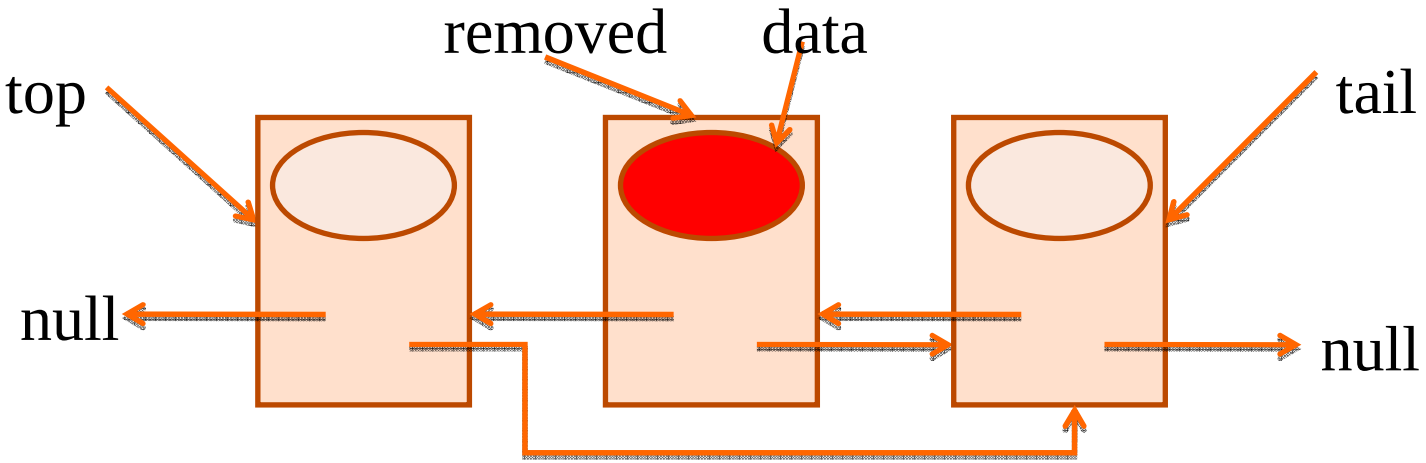
```
/**
  * Inserts 'data' after the 'prev' node. If 'prev'
  * is null, 'data' is inserted at the first position of
  * the list.
  */
public void insert(DLNode prev, Object data) {
    DLNode node = new DLNode(data);
    node.setPrev(prev);
    if (prev != null) {
        node.setNext(prev.getNext());
        prev.setNext(node);
    } else {
        node.setNext(top);
        top = node;
    }
    if (node.getNext() != null) {
        node.getNext().setPrev(node);
    } else {
        tail = node;
    }
}
```

Eliminación de un nodo



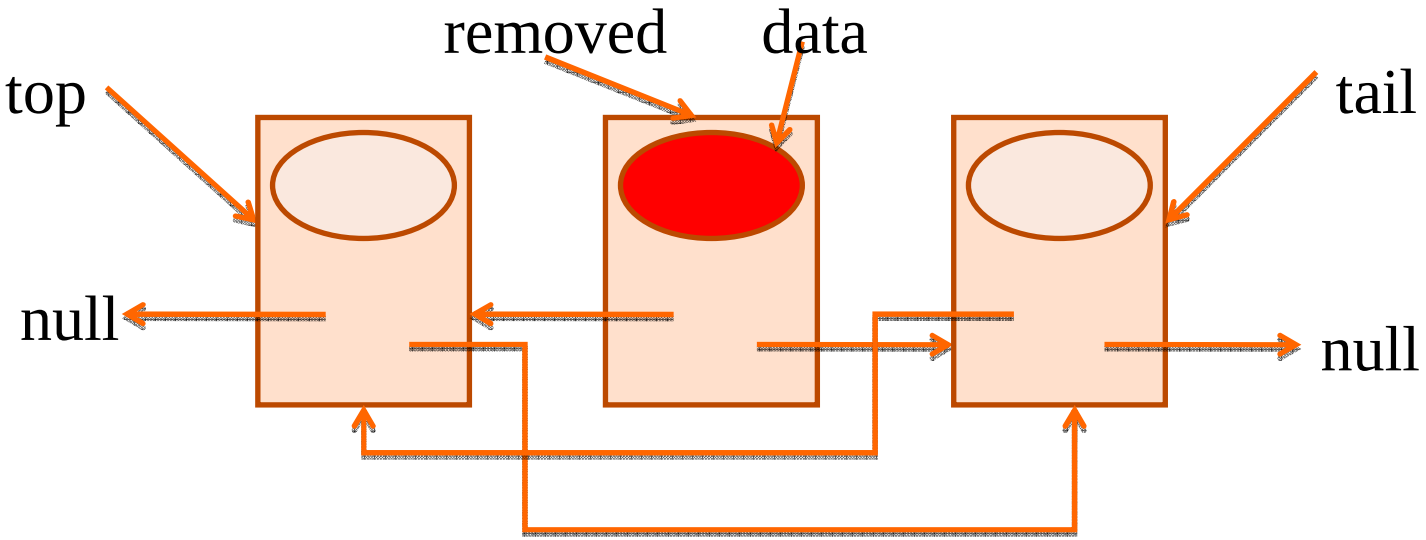

```
Object data = removed.getInfo();
```

Eliminación de un nodo



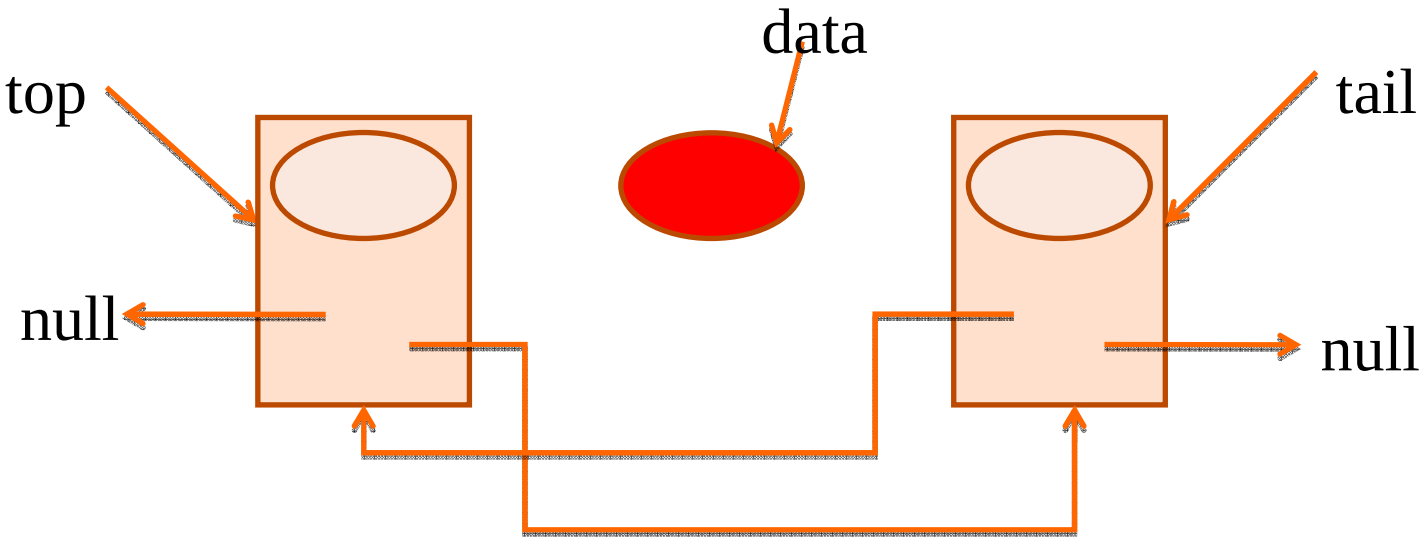

```
if (removed.getPrev() != null) {
    removed.getPrev().setNext(removed.getNext());
} else {
    top = removed.getNext();
}
```

Eliminación de un nodo




```
if (removed.getNext() != null) {
    removed.getNext().setPrev(removed.getPrev());
} else {
    tail = removed.getPrev();
}
```

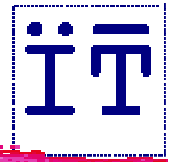
Eliminación de un nodo



Eliminación de un nodo

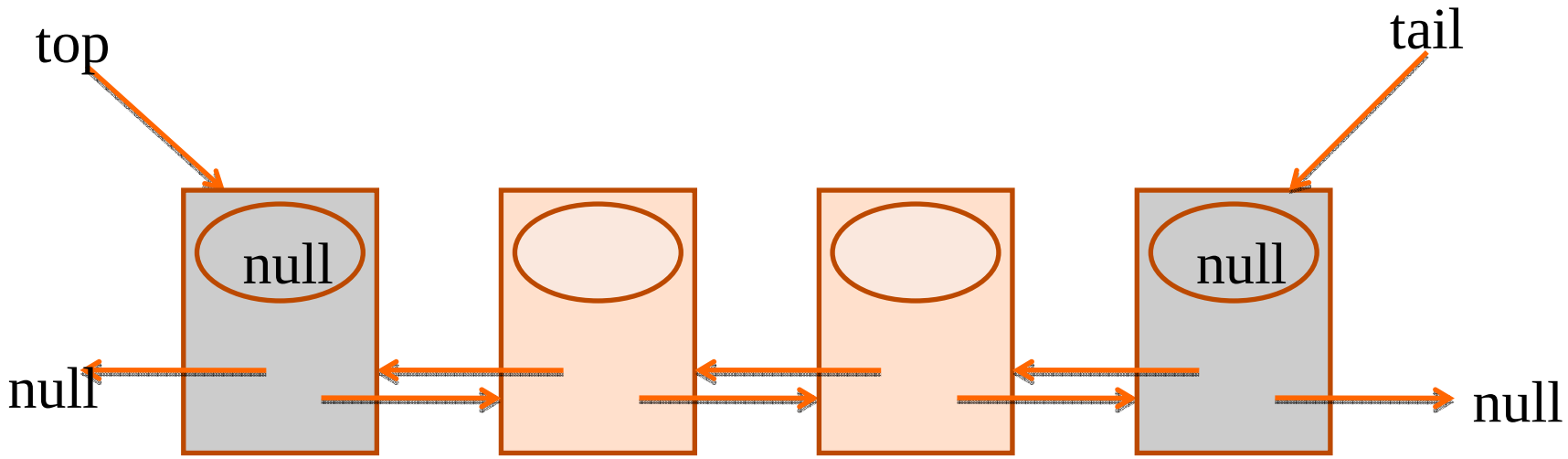
```
/**
  * Removes a node from the list and returns
  * the information it holds.
  */
public Object remove(DLNode removed) {
    Object data = removed.getInfo();
    if (removed.getPrev() != null) {
        removed.getPrev().setNext(removed.getNext());
    } else {
        top = removed.getNext();
    }
    if (removed.getNext() != null) {
        removed.getNext().setPrev(removed.getPrev());
    } else {
        tail = removed.getPrev();
    }
    return data;
}
```

Implementación alternativa



- La implementación anterior se complica debido a la necesidad de comprobar que existen los nodos anterior y posterior
- Posible simplificación:
  - Crear dos nodos especiales, sin datos, de tal forma que uno esté siempre al principio y el otro siempre al final:
    - Una lista vacía sólo contiene estos dos nodos.
    - Está garantizado en cualquier operación de inserción o extracción que siempre existen el nodo anterior y siguiente.
    - Las referencias `top` y `tail` no cambian nunca de valor.

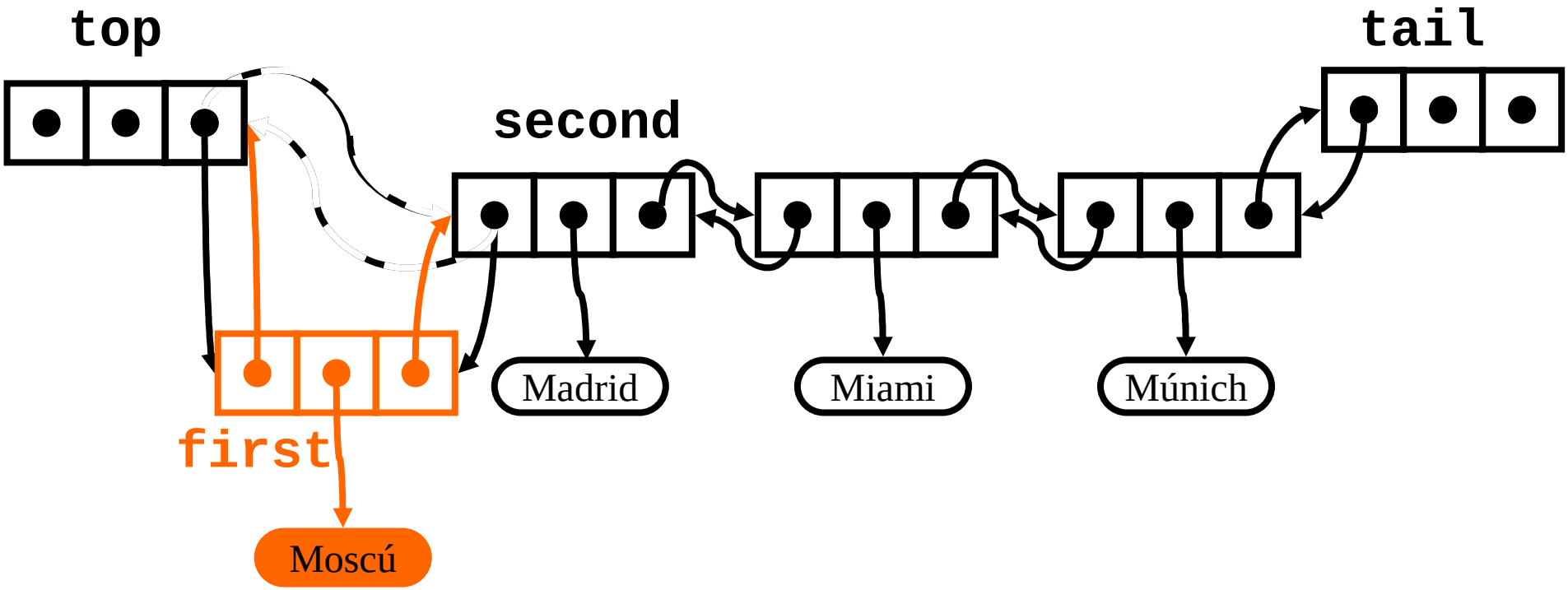
Implementación alternativa



Implementación basada en listas

```
public class DLDeque implements Deque {
  private DLNode top, tail;
  private int size;
  public DLDeque() {
    top = new DLNode();
    tail = new DLNode();
    tail.setPrev(top);
    top.setNext(tail);
    size = 0;
  }
```

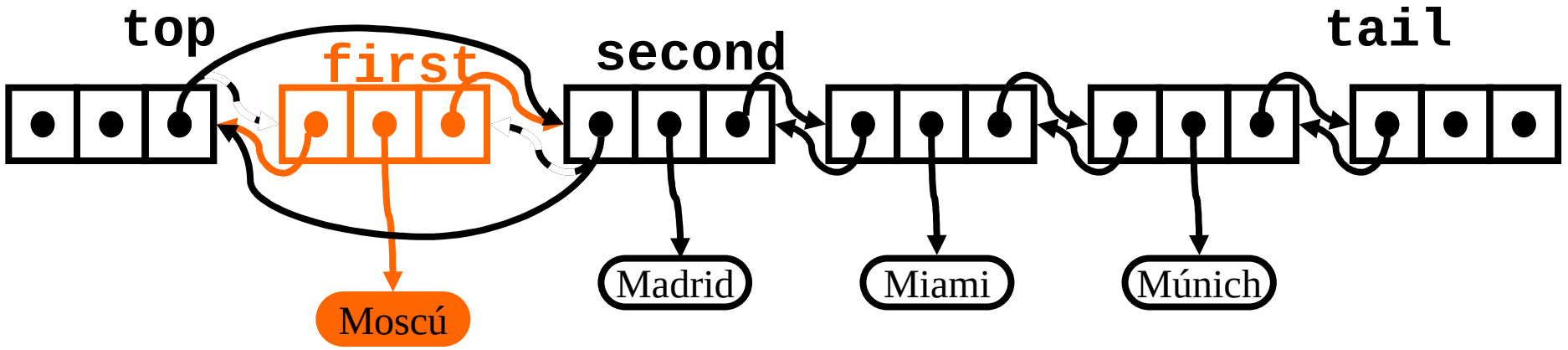
Inserción



Implementación basada en listas

```
public void insertFirst(Object info) {
  DLNode second = top.getNext();
  DLNode first = new DLNode(info, top, second);
  second.setPrev(first);
  top.setNext(first);
  size++;
}
```

Borrado



Implementación basada en listas

```
public Object removeFirst()
  throws EmptyDequeException {
  if (top.getNext() == tail)
    throw new EmptyDequeException();
  DLNode first = top.getNext();
  Object info = first.getInfo();
  DLNode second = first.getNext();
  top.setNext(second);
  second.setPrev(top);
  size--;
  return info;
```

Actividad

- Ver las "colas" en
  - **http://java.sun.com/docs/books/tutorial/collections/interfaces/queue.html**
  - **http://java.sun.com/javase/6/docs/api/java/util/Queue.html**

Method Summary	
boolean	add(E e) Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions, returning true upon success and throwing an IllegalStateException if no space is currently available.
E	element() Retrieves, but does not remove, the head of this queue.
boolean	offer(E e) Inserts the specified element into this queue if it is possible to do so immediately without violating capacity restrictions.
E	peek() Retrieves, but does not remove, the head of this queue, or returns null if this queue is empty.
E	poll() Retrieves and removes the head of this queue, or returns null if this queue is empty.
E	remove() Retrieves and removes the head of this queue.

Methods inherited from interface java.util.Collection
addAll, clear, contains, containsAll, equals, hashCode, isEmpty, iterator, remove, removeAll, retainAll, size, toArray, toArray

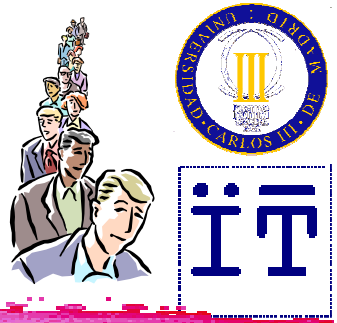
Colas con prioridad

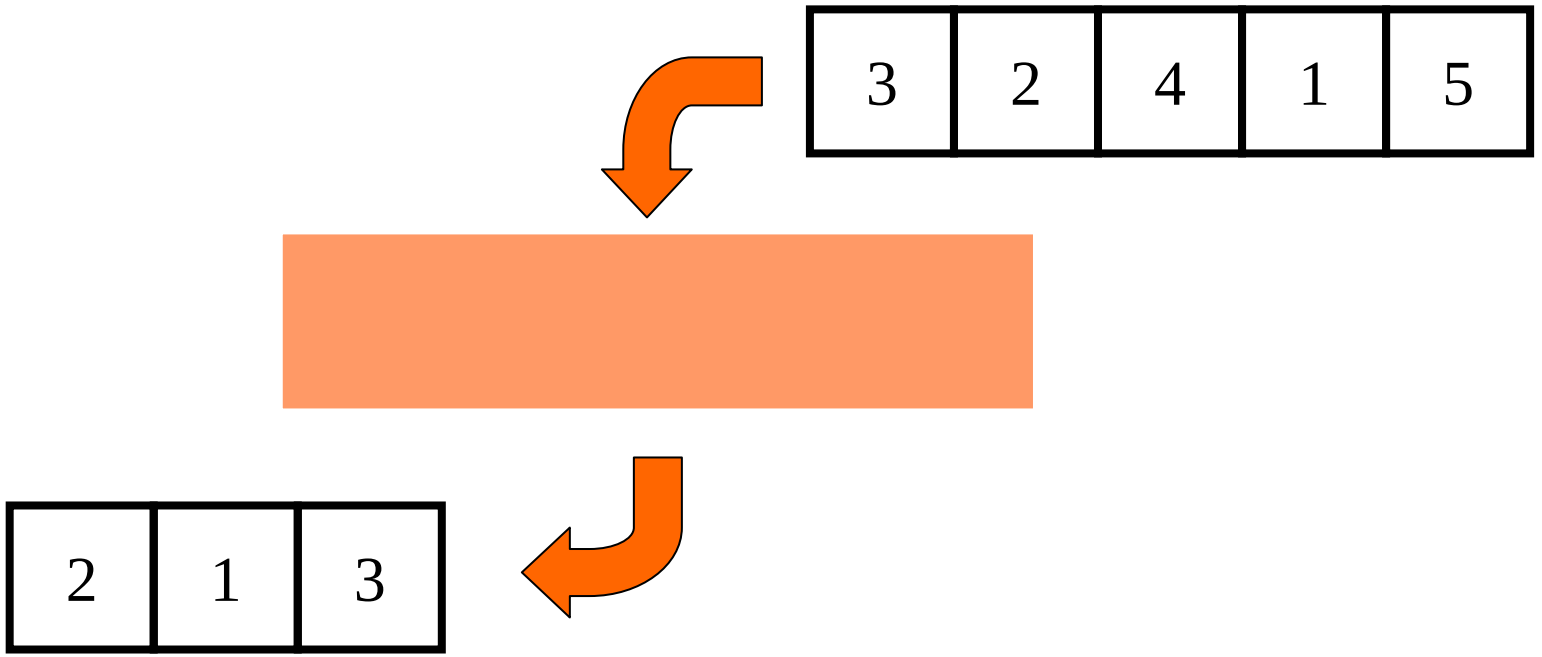
- Una cola con prioridad es una estructura de datos lineal que devuelve los elementos de acuerdo a un valor asociado a ellos (prioridad) (y no al orden en que fueron insertados).
- La prioridad puede coincidir con el valor del elemento, pero también puede diferir de él.

Interfaz

```
public interface PriorityQueue {
  public int size();
  public boolean isEmpty();
  public void insertItem(Comparable priority,
                         Object info);
  public Object minElem()
    throws EmptyPriorityQueueException;
  public Object removeMinElem()
    throws EmptyPriorityQueueException;
  public Object minKey()
    throws EmptyPriorityQueueException;
}
```

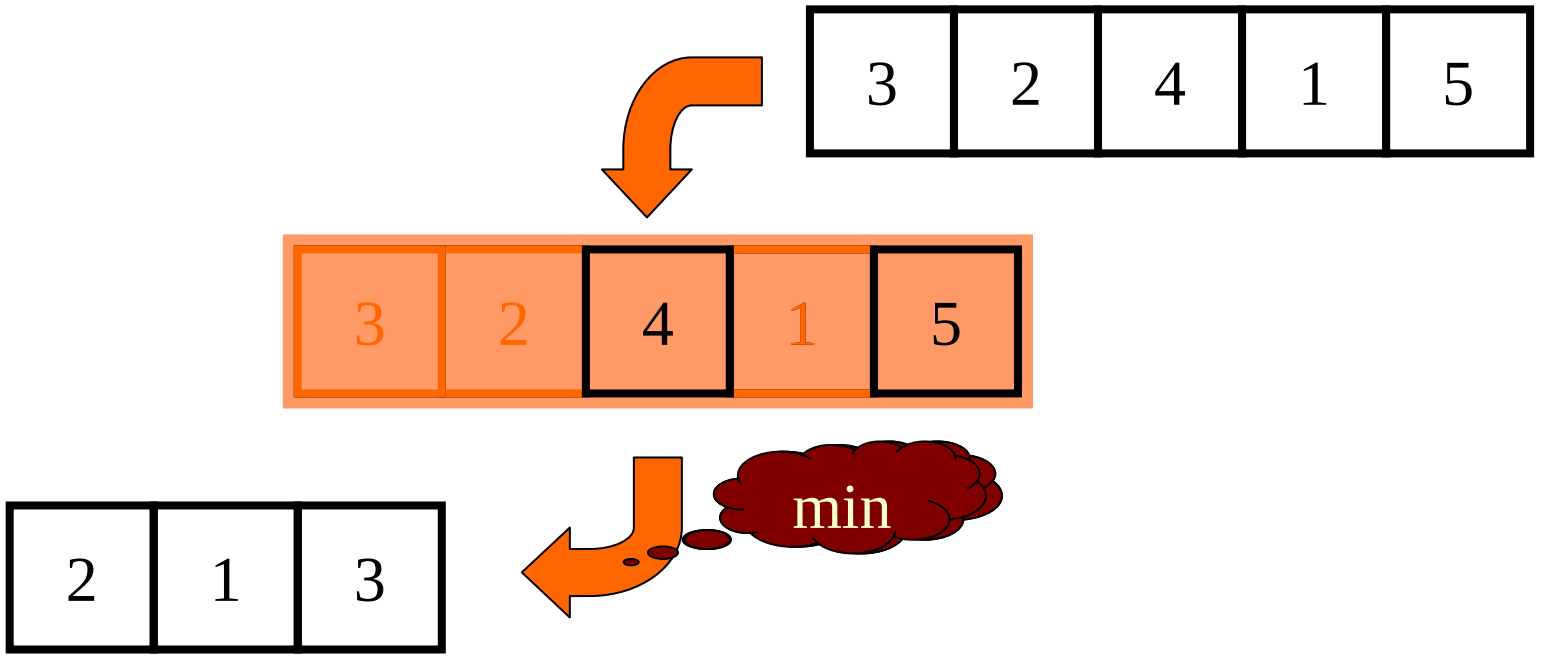
Ejemplo





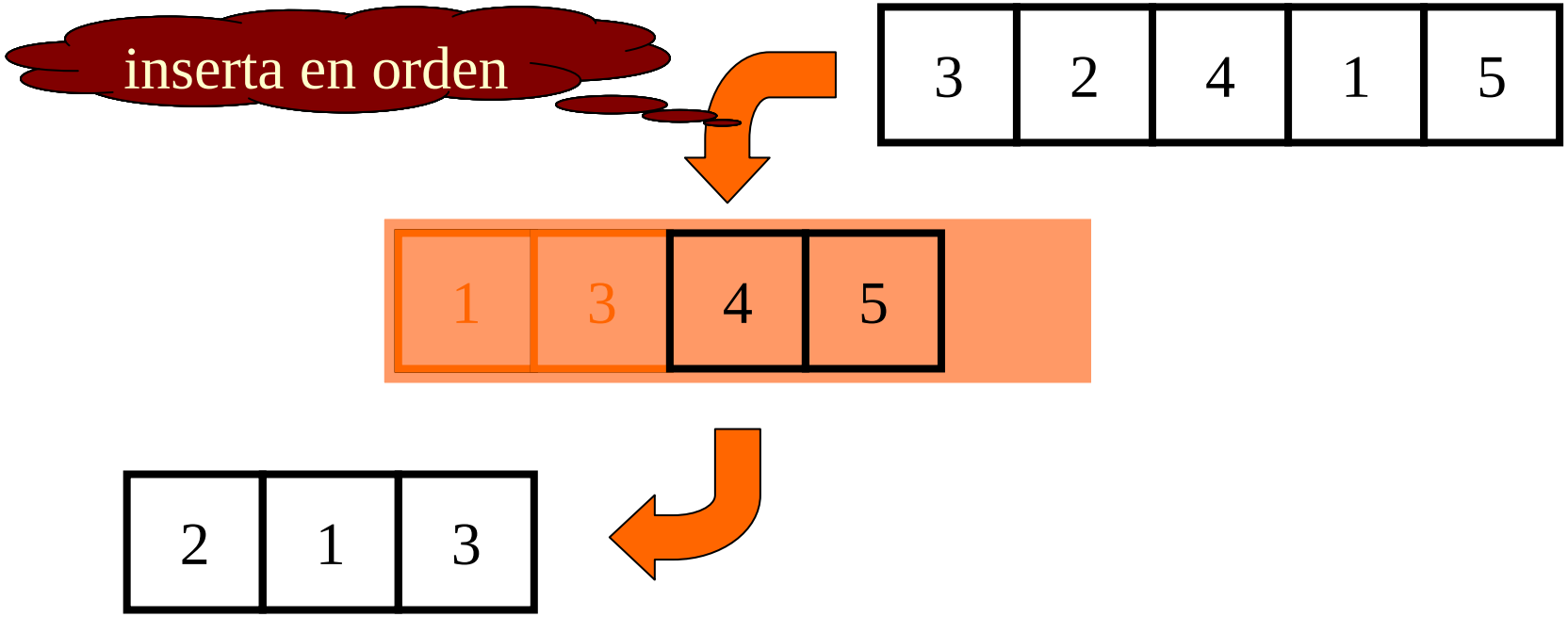
+ + + – + – + –

Ejemplo



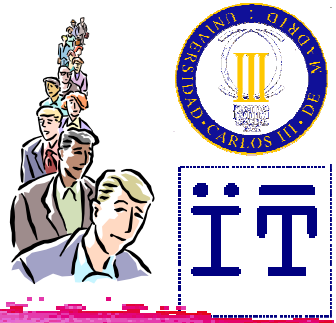
+ + + – + – + –

Ejemplo



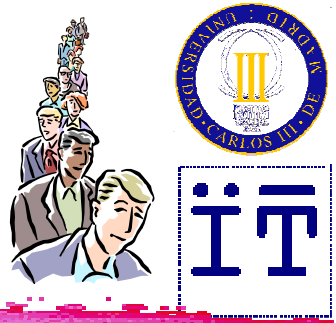
+ + + – + – + –

Implementaciones



- Con una secuencia sin ordenar
  - Inserción fácil 
  - Comparación al extraer 
- Con una secuencia ordenada
  - Comparación al insertar 
  - Extracción fácil 

Actividad



- Probar

  http://www.akira.ruc.dk/~keld/algoritmik_e99/Applets/Chap11/PriorityQ/PriorityQ.html