Programación de sistemas

Pilas y Colas

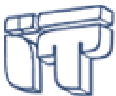
Julio Villena Román
<jvillena@it.uc3m.es>

MATERIALES BASADOS EN EL TRABAJO DE DIFERENTES AUTORES:
Carlos Delgado Kloos, Jesús Arias Fisteus, Carlos Alario Hoyos

Universidad Carlos III de Madrid

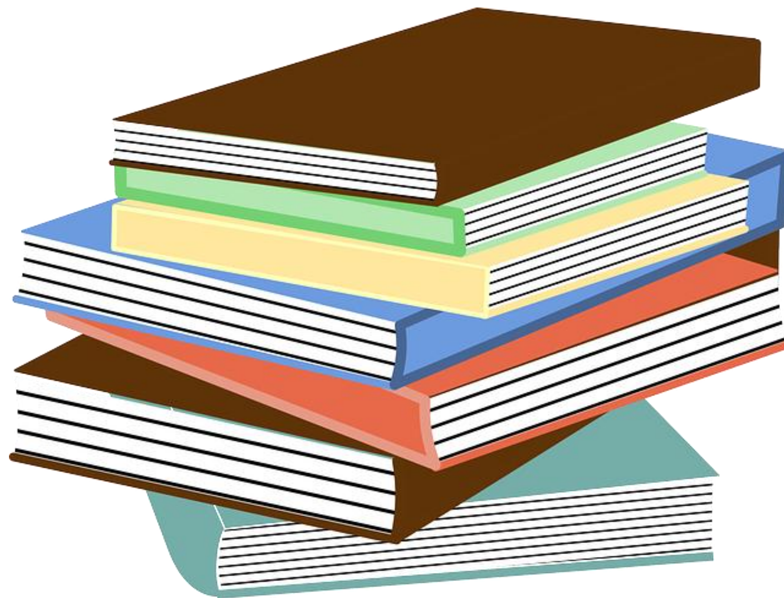
Contenidos

- Pilas (*stacks*)
- Colas (*queues*)
- Colas dobles (*deques – double-ended queues*)



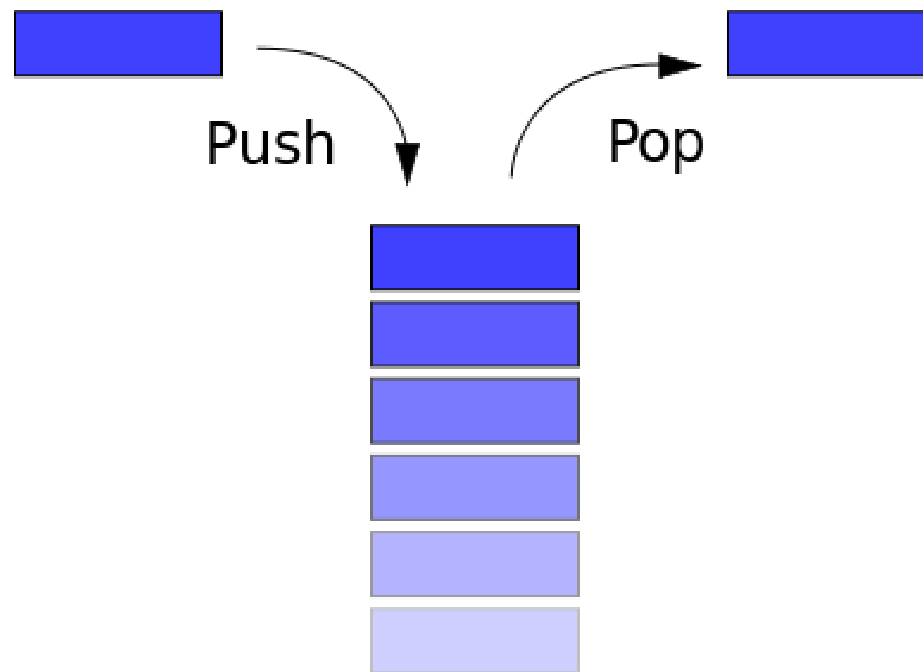
Pilas

- Estructura de datos lineal
- Inserción y extracción por un único extremo
  - ✓ LIFO (*Last-In-First-Out*)



Pilas

- Insertar por un extremo: `push(x)`
- Extraer por el mismo extremo: `pop()`



Ejemplo:
Revisión de paréntesis

- Correcto:
  -
  - ()
  - (()(()))

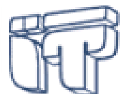
- Incorrecto:
  -)(
  - (()
  - ())

- Reglas:
  - Básica: apertura + cierre
  - Secuenciación: ()()
  - Anidamiento: (())

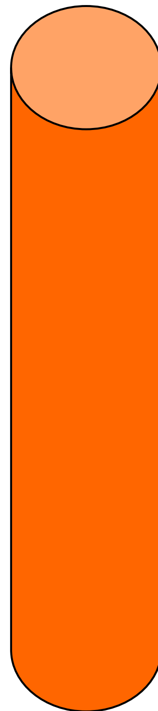
Ejemplo:
Revisión de paréntesis

Reglas:

- Cada vez que encontremos un “(“ se añade a la pila
- Cada vez que encontremos un “)” se extraerá el “(“ de arriba de la pila
- La expresión con paréntesis es correcta si la pila está vacía al acabar la expresión y siempre hemos encontrado un “)” correspondiente a un “(”



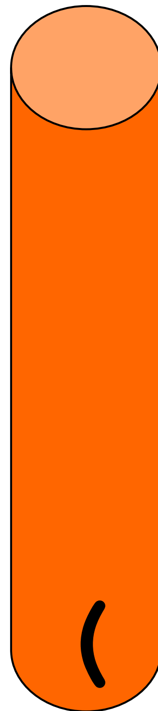
Ejemplo:
(()(()())())



(( ) (( ) ( )) ( ))

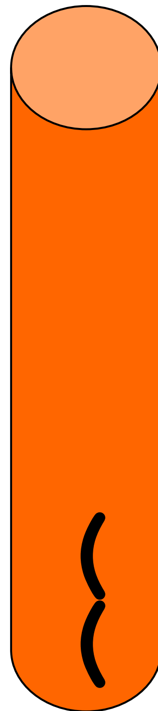
Universidad Carlos III de Madrid

Ejemplo:
(()(()())(())



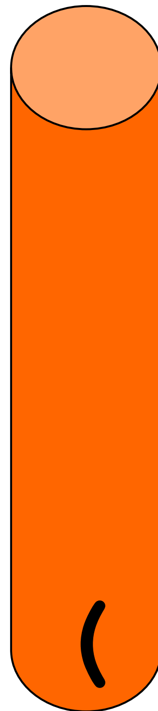
~~(~~ ( ) (( ) ( )) ( ))

Ejemplo:
(()(()())())



~~(~~ ~~(~~) (( ) ( )) ( ))

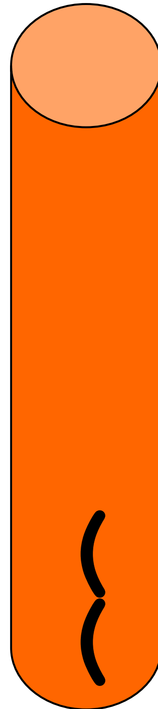
Ejemplo:
(()((())())(()))



(( ) (( ) ( )) ( ))

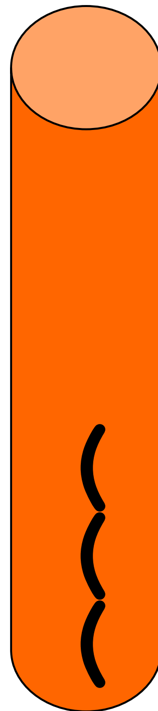
Universidad Carlos III de Madrid

Ejemplo:
(()(()()))(()))

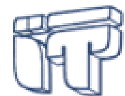


(( ) (( ) ( )) ( ))

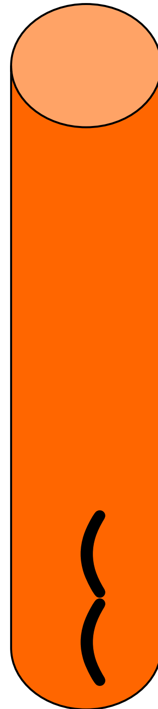
Ejemplo:
(()(()())(()))



(( ) (( ) ( )) ( ))

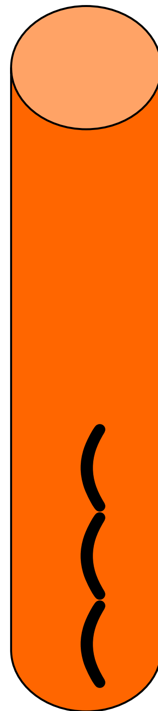


Ejemplo:
(()((())())(()))



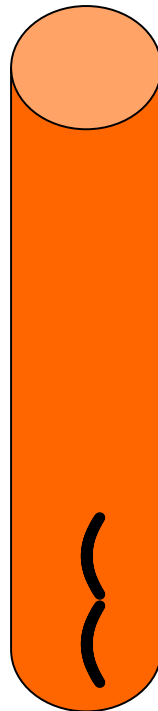
(( ) (( ) ( )) ( ))

Ejemplo:
(()(()())())



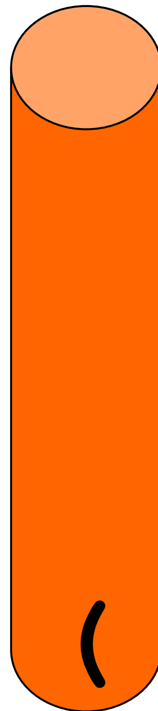
(( ) (( ) ( )) ( ))

Ejemplo:
(()(()())(()))



(( ) (( ) ( )) ( ))

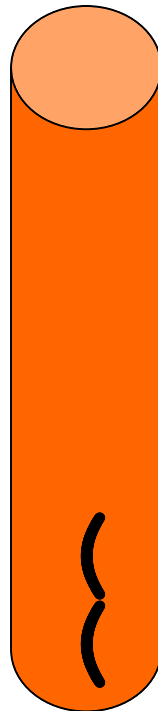
Ejemplo:
(()(()())(()))



(( ) (( ) ( )) ( ))

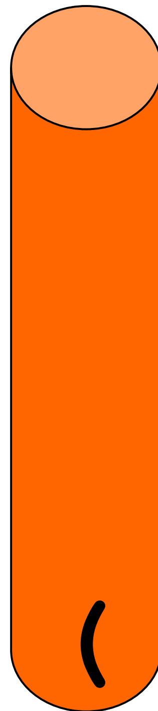
Universidad Carlos III de Madrid

Ejemplo:
(()(()())())



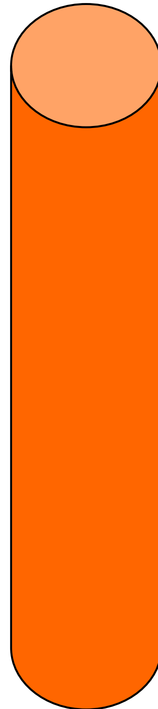
(( ) (( ) ( )) ( ))

Ejemplo:
(()(()())())



(( ) (( ) ( )) ( ))

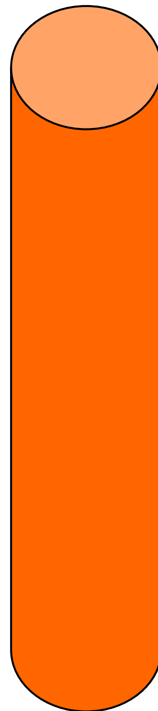
Ejemplo:
(()((())())(()))



Correcto: Expresión completa y pila vacía!!

(( ) (( ) ( )) ( ))

Ejemplo:
([]{()<>}())



Correcto: Expresión completa y pila vacía!!

~~([ ] { ( ) < > } ( ))~~

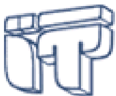
Ejemplo: HTML

```
<b><i>hello</b></i>
```

- Correcto en HTML 1.0-4.0
- Incorrecto en XHTML

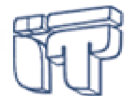
```
<b><i>hello</i></b>
```

- Correcto en ambos



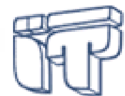
Interfaz para pilas

```
public interface Stack<E> {
    boolean isEmpty();
    int size();
    E top();
    void push(E info);
    E pop();
}
```



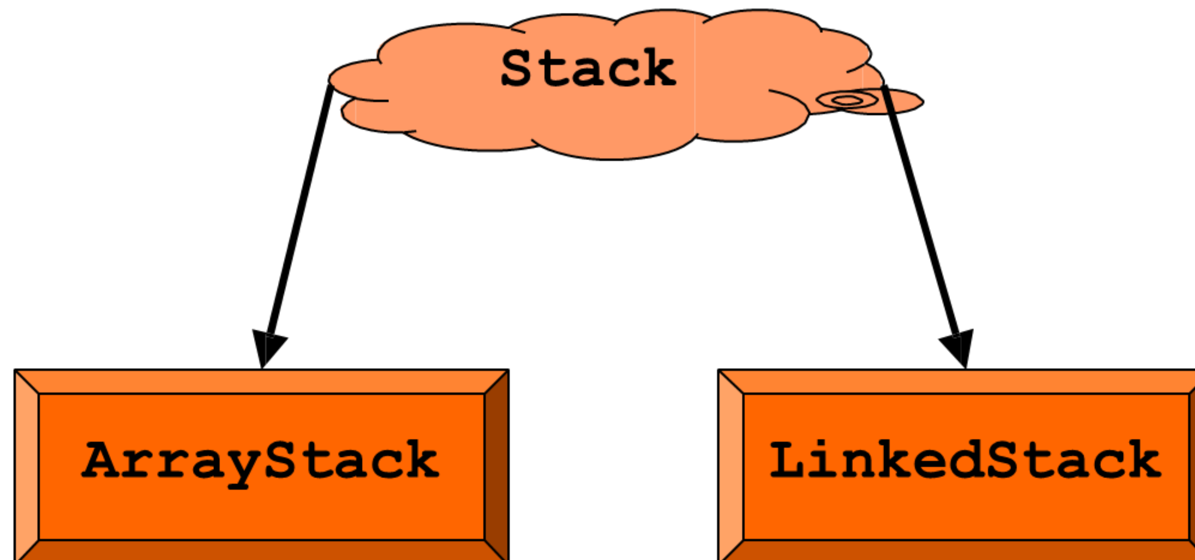
Interfaz para pilas (con excepciones)

```
public interface Stack<E> {
    boolean isEmpty();
    int size();
    E top() throws
        EmptyStackException;
    void push(E info) throws
        StackOverflowException;
    E pop() throws
        EmptyStackException;
}
```

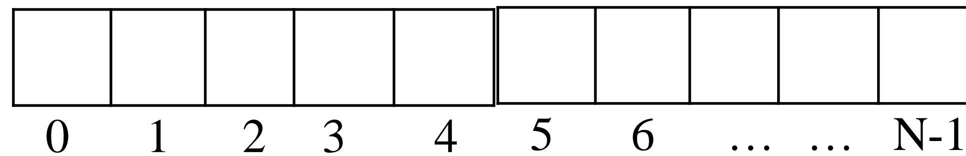


Una interfaz dos implementaciones

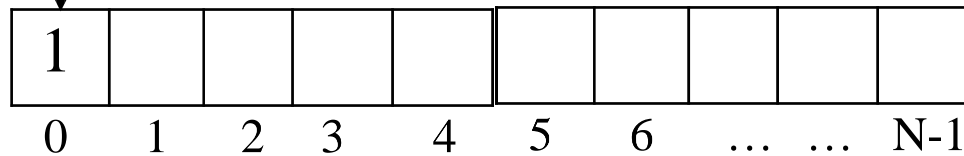
- Implementación basada en arrays:
  - ✓ `ArrayStack`
- Implementación basada en listas enlazadas:
  - ✓ `LinkedStack`



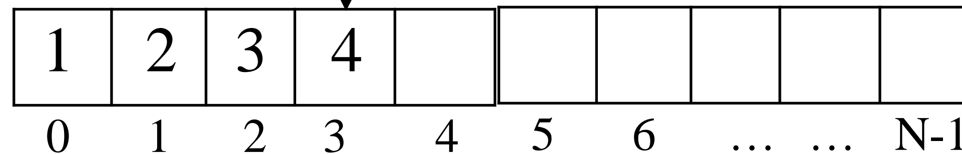
ArrayStack



Pila vacía



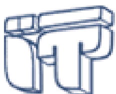
Pila con 1 elemento



Pila con 4 elementos

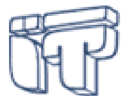
ArrayStack (I)

```
public class ArrayStack<T> implements Stack<E> {
  public static final int DEFAULT_CAPACITY = 1000;
  private int capacity;
  private E data[];
  private int top = -1;
  public ArrayStack() {
    this(DEFAULT_CAPACITY);
  }
  public ArrayStack(int capacity) {
    this.capacity = capacity;
    data = new E[capacity];
  }
...
```



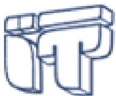
ArrayStack (II)

```
…
public int size() {
   return (top + 1);
}
public boolean isEmpty() {
   return (top < 0);
}
public E top() throws EmptyStackException {
   if (isEmpty())
      throw new EmptyStackException("Empty");
   return data[top];
}
…
```



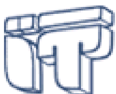
ArrayStack (III)

```
…
public void push(E o)
   throws StackOverflowException {
   if (size == capacity)
     throw new StackOverflowException();
   data[++top] = o;
}
…
```



ArrayStack (IV)

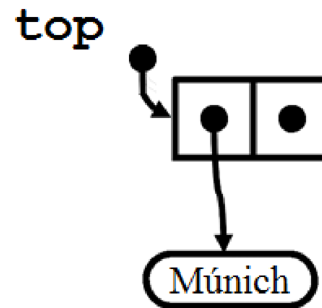
```
…
public E pop() throws StackEmptyException {
    E o;
    if (top == -1)
        throw new EmptyStackException();
    o = data[top];
    data[top--] = null;
    return o;
  }
}
```



LinkedStack

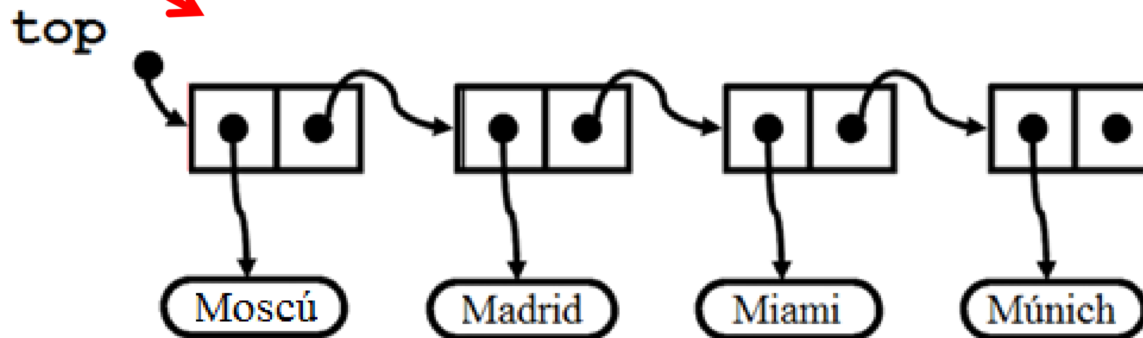
top

Pila vacía

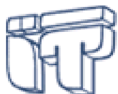


Pila con 1 elemento

Extremo de inserción y extracción

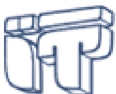


Pila con 4 elementos



Recordando la clase Node

```
public class Node<E> {
    private E info;
    private Node<E> next;

    public Node()    {…}
    public Node(E info)   {…}
    public Node(E info, Node<E> next)    {…}

    public Node<E> getNext() {…}
    public void setNext(Node<E> next) {…}
    public E getInfo() {…}
    public void setInfo(E info) {…}

}
```



LinkedStack (I)

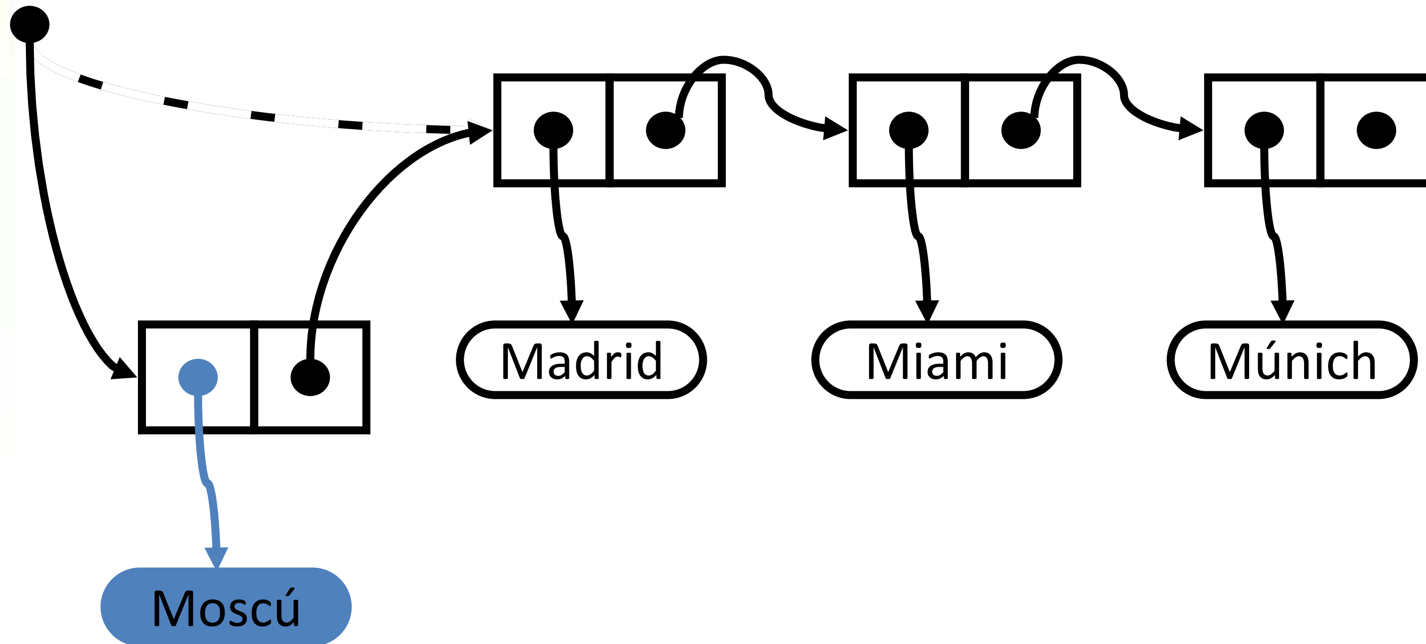
```
public class LinkedStack<E> implements Stack<E> {
    private Node<E> top;
    private int size;
    public LinkedStack() {
        top = null;
        size = 0;
    }

    public boolean isEmpty() {
        return (top == null);
    }

    public int size() {
        return size;
    }

    public E top() {
        if(isEmpty()){
            return null;
        }
        return top.getInfo();
    }
…
```

Atributos

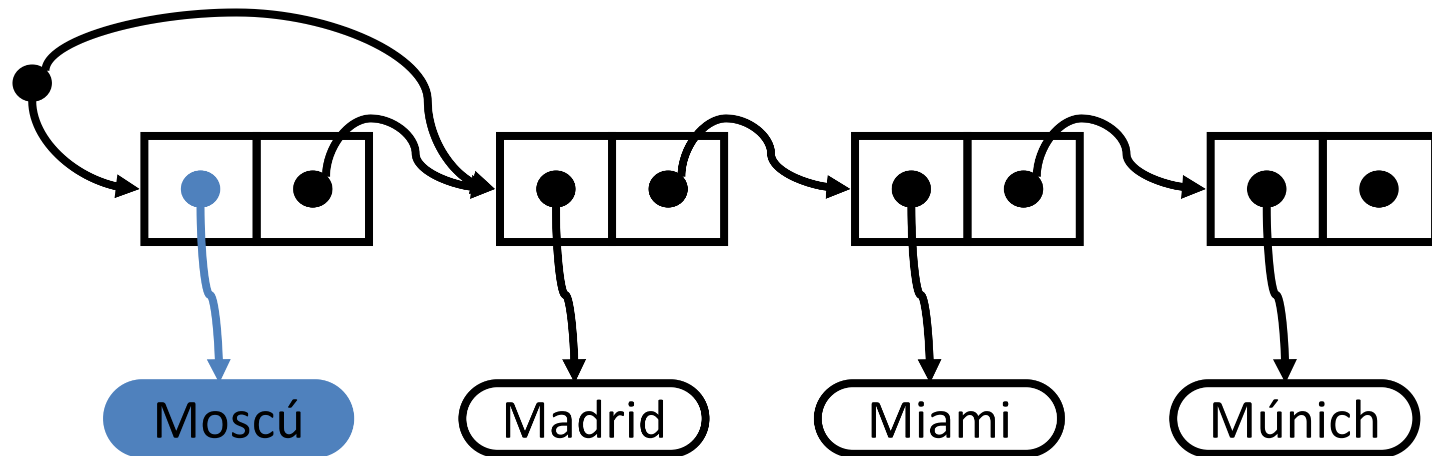
Constructor

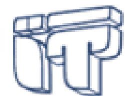
Métodos de la interfaz Stack a implementar (I)

Inserción (push)



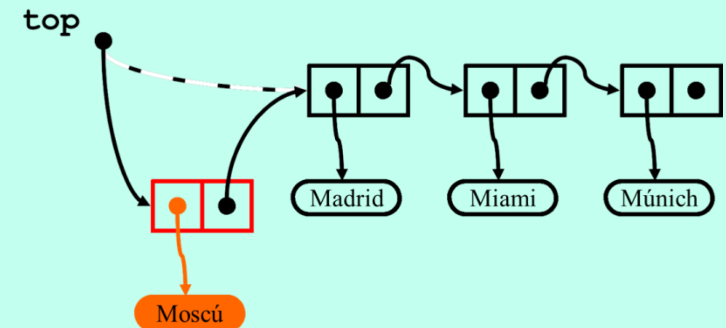
Extracción (pop)



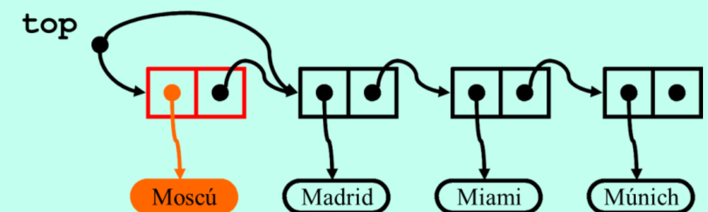


LinkedStack (II)

```
…
    public void push(E info){
        Node<E> n = new Node<E>(info, top);
        top = n;
        size++;
    }

    public E pop() {
        E info;
        if(isEmpty()){
            return null;
        } else{
            info = top.getInfo();
            top = top.getNext();
            size--;
            return info;
        }
    }
}
```



Métodos de la interfaz Stack a implementar (II)

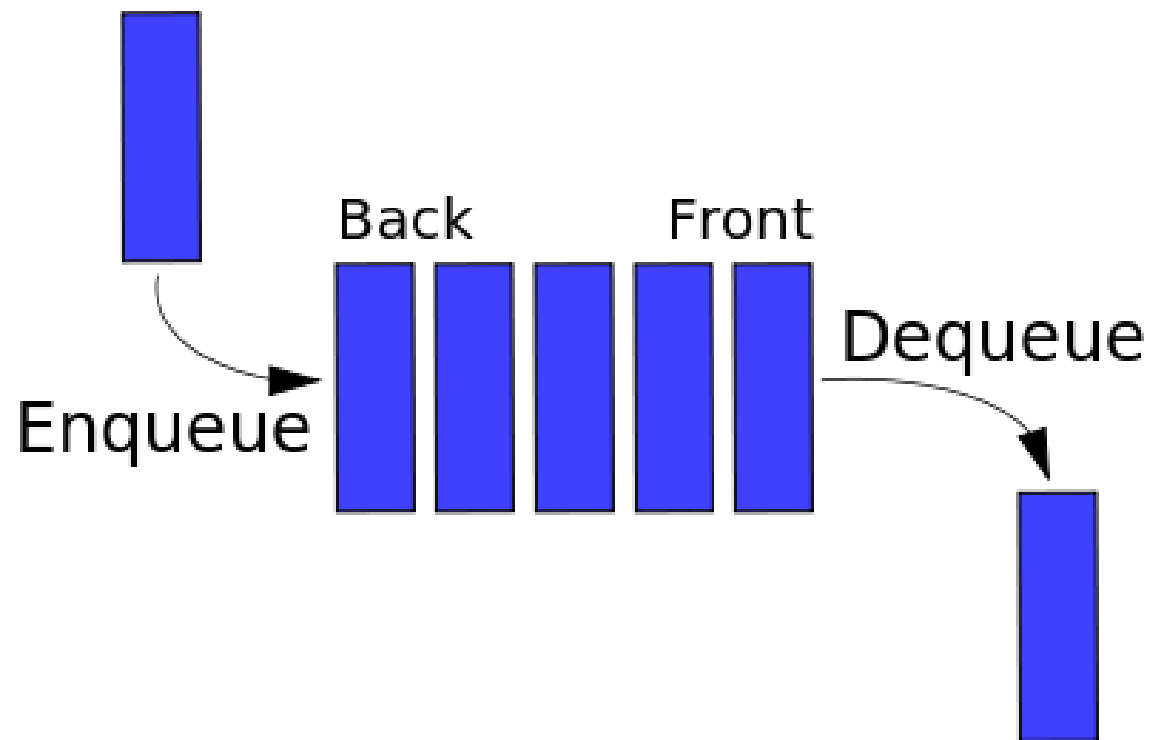


Colas

- Estructura de datos lineal
- Inserción por un extremo y extracción por el exremo opuesto
  - ✓ FIFO (*First-In-First-Out*)

Colas

- Insertar por un extremo: `enqueue(x)`
- Extraer por el extremo opuesto: `dequeue()`

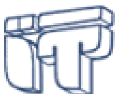


Interfaz para colas

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

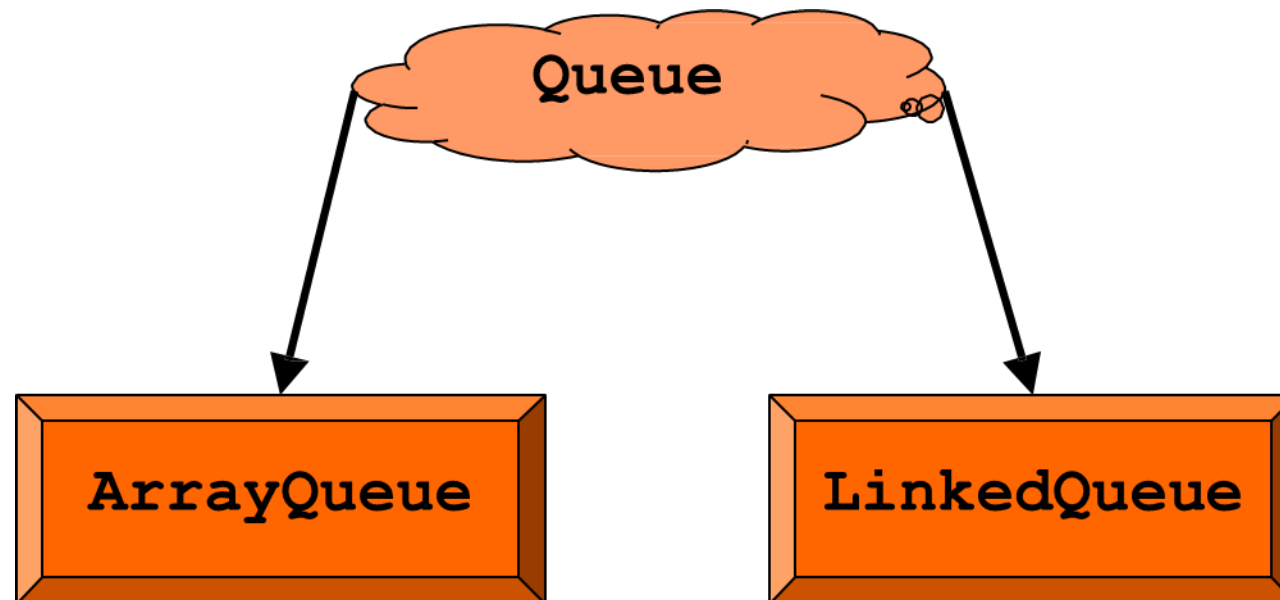
Interfaz para colas (con excepciones)

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

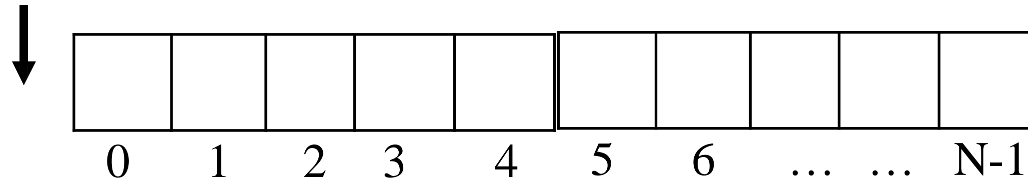


Una interfaz dos implementaciones

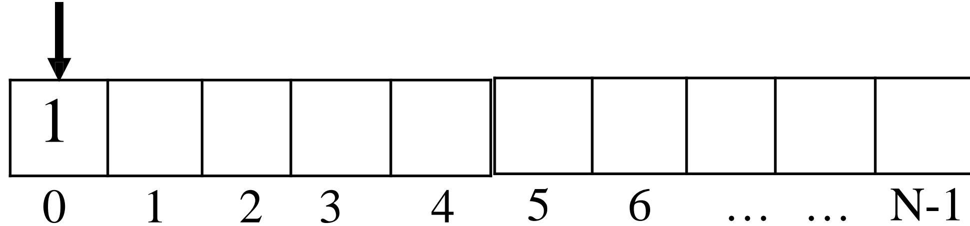
- Implementación basada en arrays:
  - ✓ `ArrayQueue`
- Implementación basada en listas enlazadas:
  - ✓ `LinkedQueue`



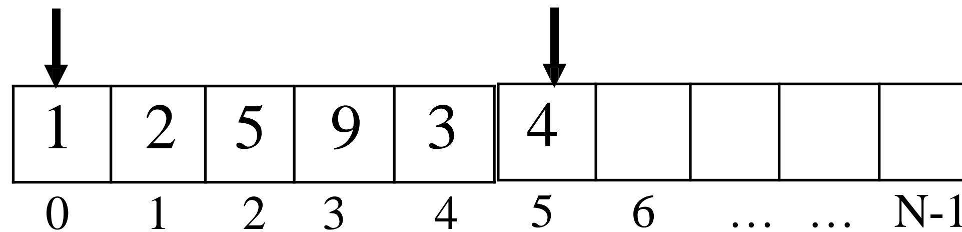
ArrayQueue



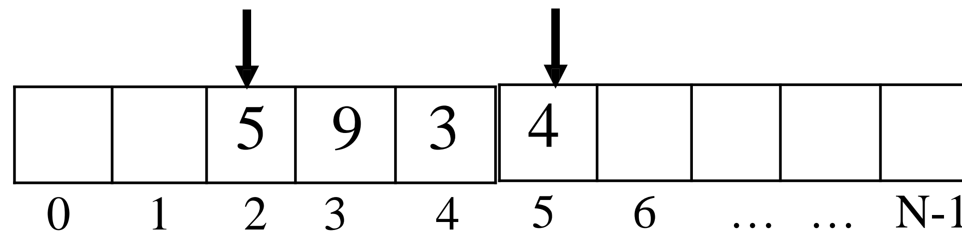
Cola vacía



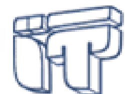
Insertamos 1 elemento



Insertamos 5 elementos más



Extraemos dos elementos



Universidad Carlos III de Madrid

LinkedQueue

top tail

Cola vacía

top

tail

Múnich

Cola con 1 elemento

Extremo de extracción

Extremo de inserción

tail

top

Múnich Miami Madrid Moscú

Cola con 4 elementos

LinkedQueue (I)

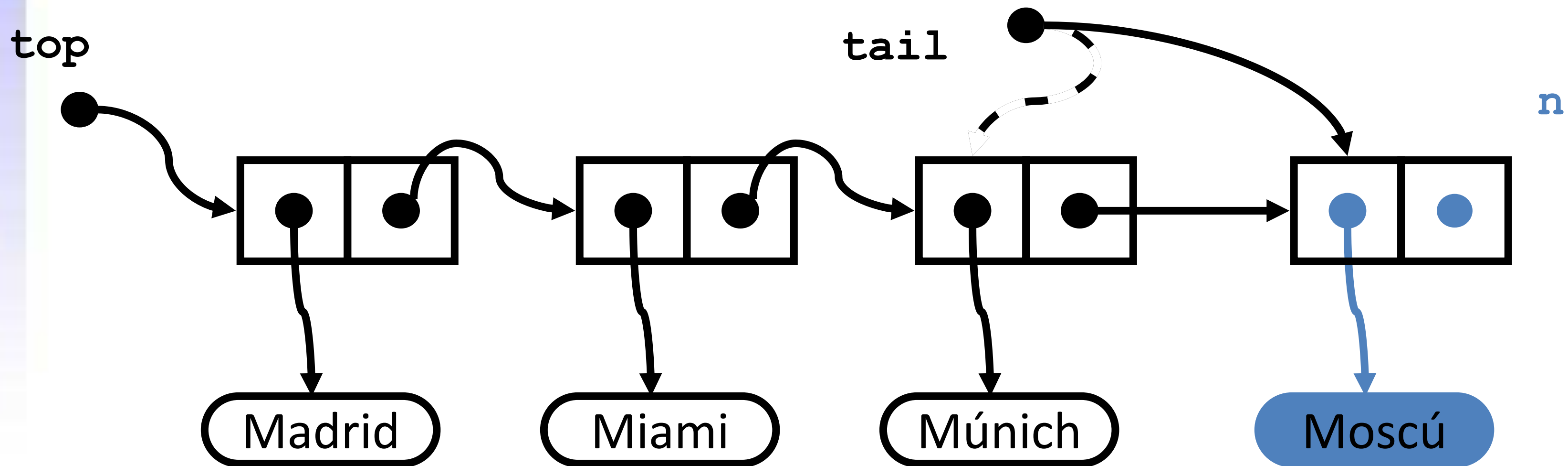
```
public class LinkedQueue<E> implements Queue<E> {
    private Node<E> top = null;
    private Node<E> tail = null;
    private int size = 0;
    public LinkedQueue(){
        top = null;
        tail = null;
        size = 0;
    }

    public boolean isEmpty() {
        return (top == null);
    }

    public int size() {
        return size;
    }

    public E front() {
        if (isEmpty()){
            return null;
        } else {
            return top.getInfo();
        }
    }
…
```

Atributos

Constructor

Métodos de la interfaz Queue a implementar (I)

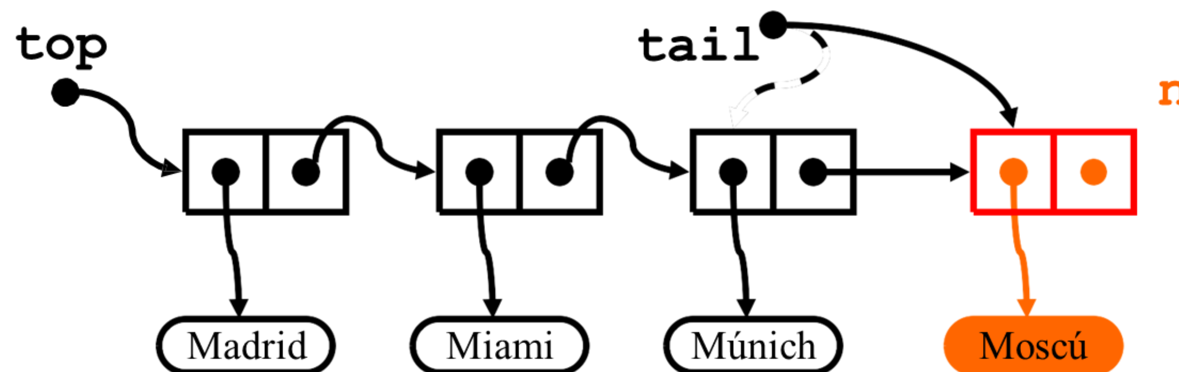
Inserción (enqueue)



LinkedQueue (II)

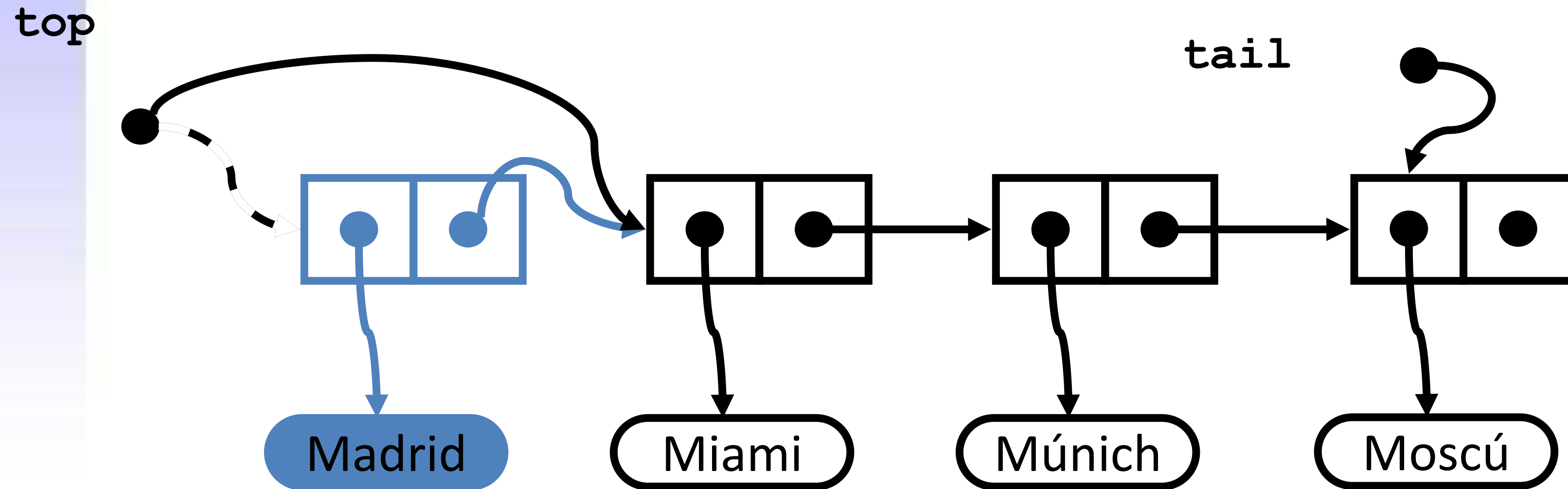
```
…
    public void enqueue (E info){
        Node<E> n = new Node<E>(info, null);
        if (isEmpty()){
            top = n;
        } else{
            tail.setNext(n);
        }
        tail = n;
        size++;
    }
…
```

Métodos de la interfaz Queue a implementar (II)



Universidad Carlos III de Madrid

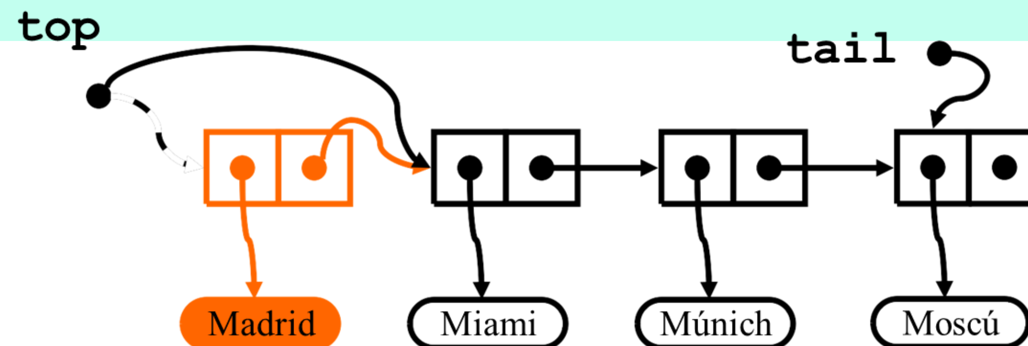
Extracción (dequeue)

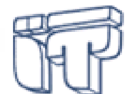


LinkedQueue (III)

```
…
    public E dequeue(){
        E info;
        if (isEmpty()){
            info = null;
        } else{
            info = top.getInfo();
            top = top.getNext();
            size--;
            if (isEmpty()){
                tail = null;
            }
        }
        return info;
    }
}
```

Métodos de la interfaz Queue a implementar (III)

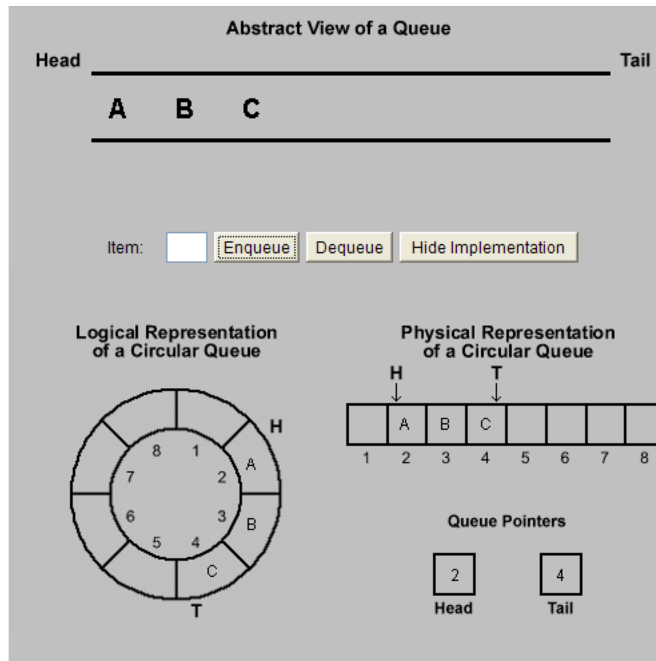




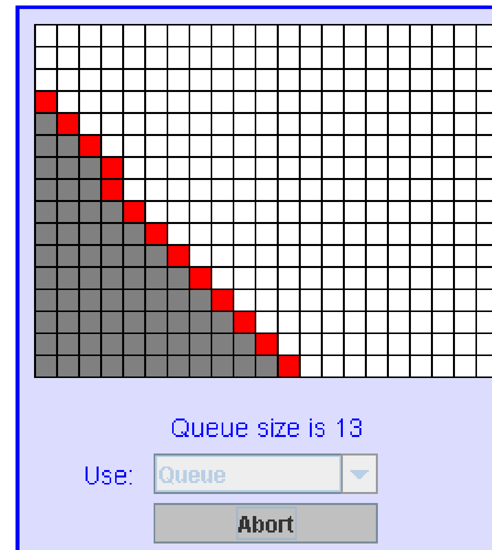
Universidad Carlos III de Madrid

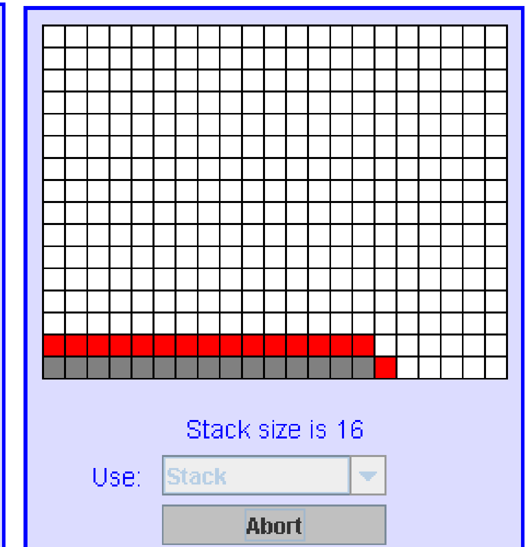
Actividad

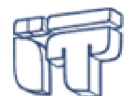
http://courses.cs.vt.edu/csonline/DataStructures/Lessons/QueuesImplementationView/applet.html



DepthBreadth.java en **http://www.faqs.org/docs/javap/c11/s3.html**

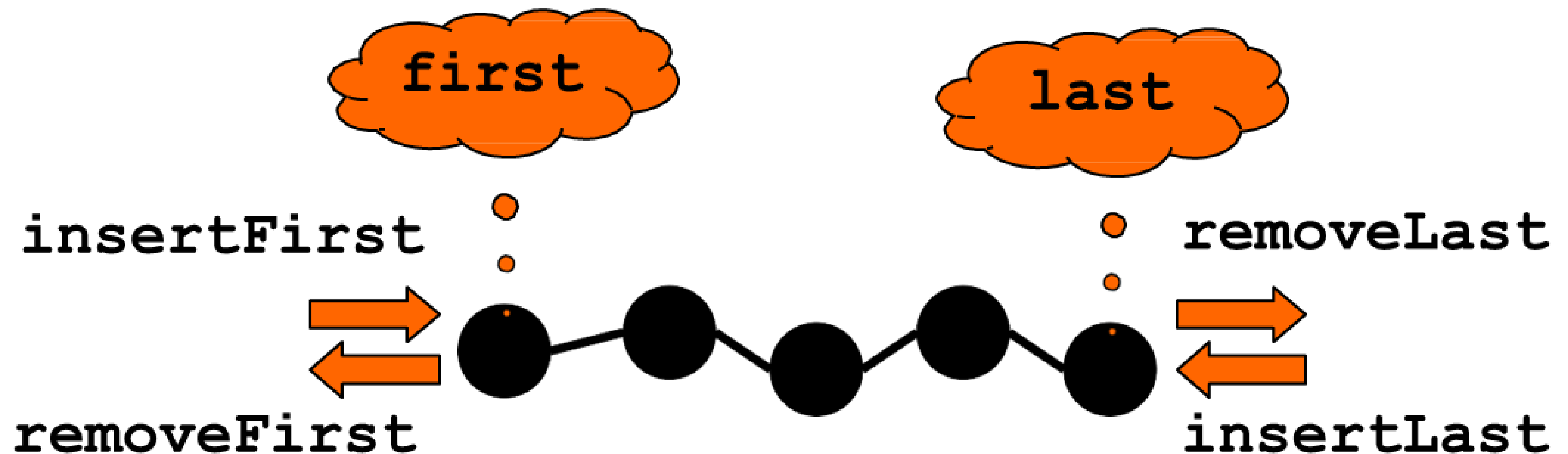






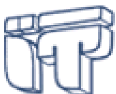
Colas dobles (*deques*)

- Estructura de datos lineal
  - *Deque (double-ended queue)*
- Inserción y extracción por cualquiera de los extremos



Interfaz para colas dobles

```
public interface Deque<E> {
    public boolean isEmpty();
    public int size();
    public E first();
    public E last();
    public void insertFirst(E info);
    public void insertLast(E info);
    public E removeFirst();
    public E removeLast();
}
```



Universidad
Carlos III de Madrid

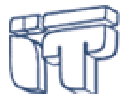
Interfaz para colas dobles

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

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

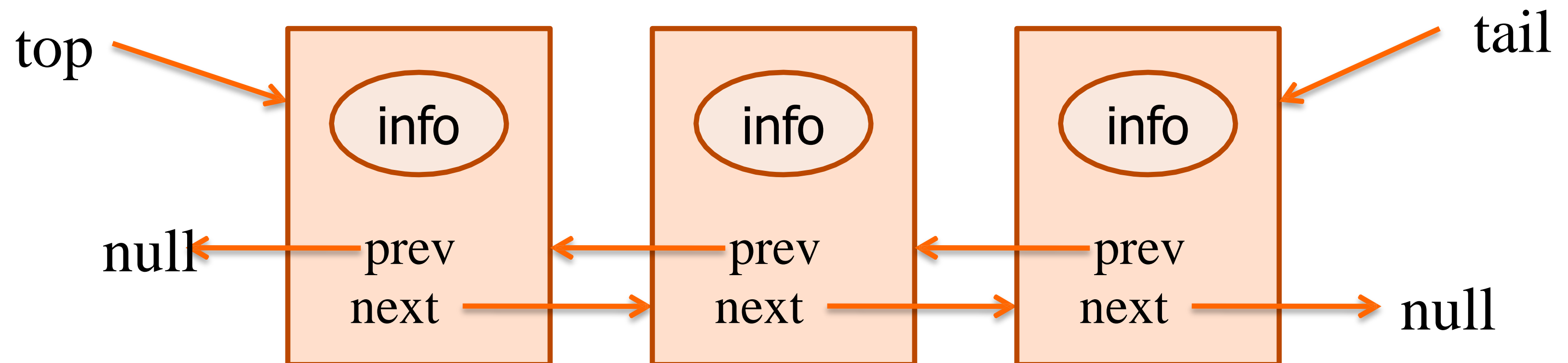
Implementación de colas dobles

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



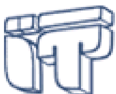
Listas doblemente enlazadas

- 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<E> {
    private E info;
    private DLNode<E> prev;
    private DLNode<E> next;

    public DLNode()  {…}
    public DLNode(E info)  {…}
    public DLNode(E info, DLNode<E> prev, DLNode<E> next){…}

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

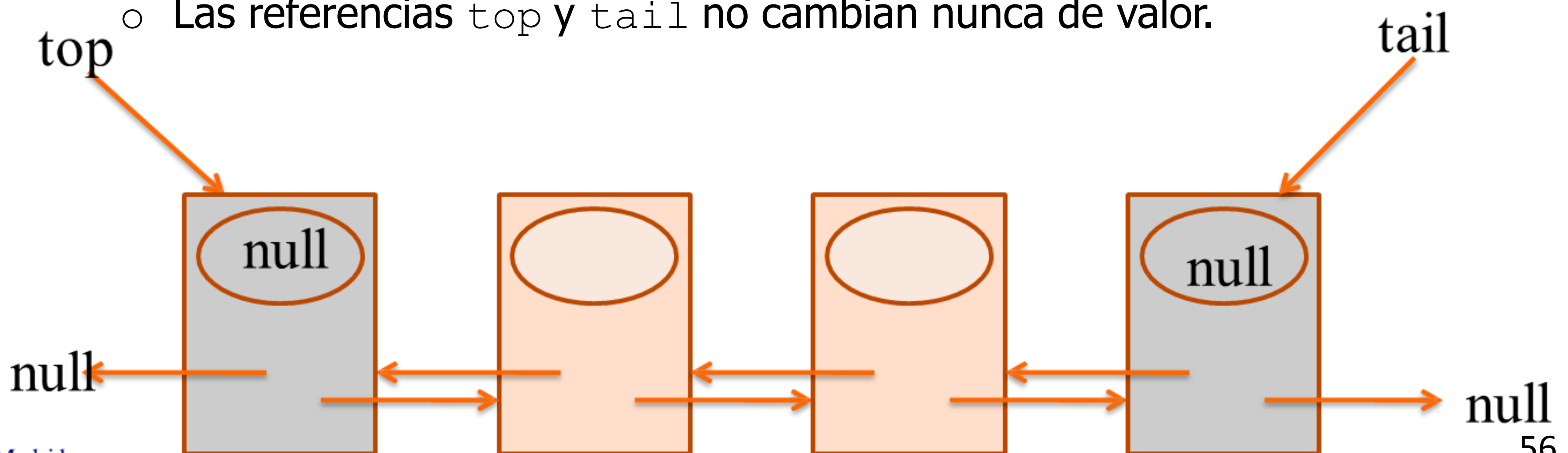


Ejercicio 1

- Completa el código de la clase **DLNode**. Incluye tres constructores: uno que no recibe información para inicializar ningún atributo; otro que permite inicializar el atributo **info**, y otro que permite inicializar los tres atributos

Listas doblemente enlazadas

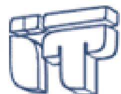
- La implementación de colas dobles basada en listas enlazadas se complica debido a la necesidad de comprobar siempre que existen los nodos anterior y posterior
- **Simplificación:** Crear dos nodos especiales, sin datos, de tal forma que uno esté siempre al principio y el otro siempre al final:
  - 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.



Universidad Carlos III de Madrid

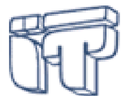
La clase cola doble (DLDeque) con listas doblemente enlazadas

```
public class DLDeque<E> implements Deque<E>{
    private DLNode<E> top;
    private DLNode<E> tail;                    Atributos
    private int size;

    public DLDeque(){
        top = new DLNode<E>();
        tail = new DLNode<E>();                Constructor
        tail.setPrev(top);
        top.setNext(tail);
        size = 0;
    }
…
```



Ejercicio 2

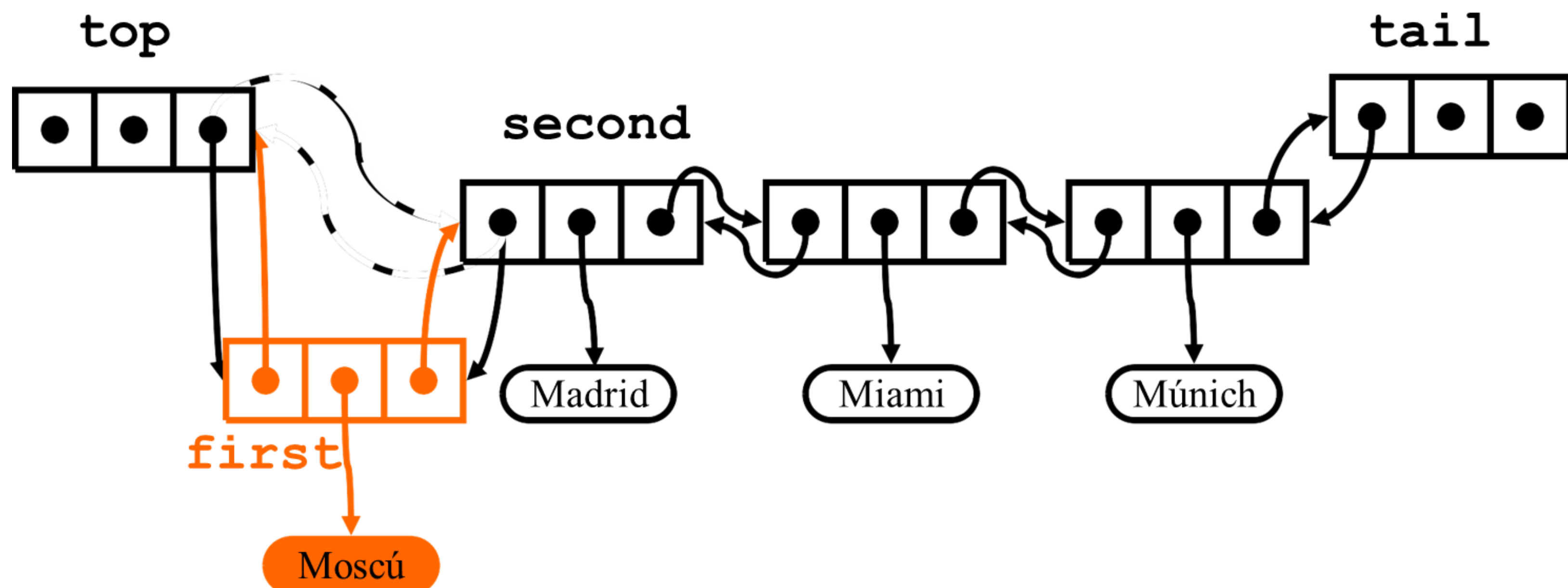
- Implementa los siguientes métodos de la clase `DLDeque`:
  - `boolean isEmpty()`
  - `int size()`
  - `E first()`
  - `E last()`



Universidad Carlos III de Madrid

La clase cola doble (DLDeque) con listas doblemente enlazadas: Inserción por el principio

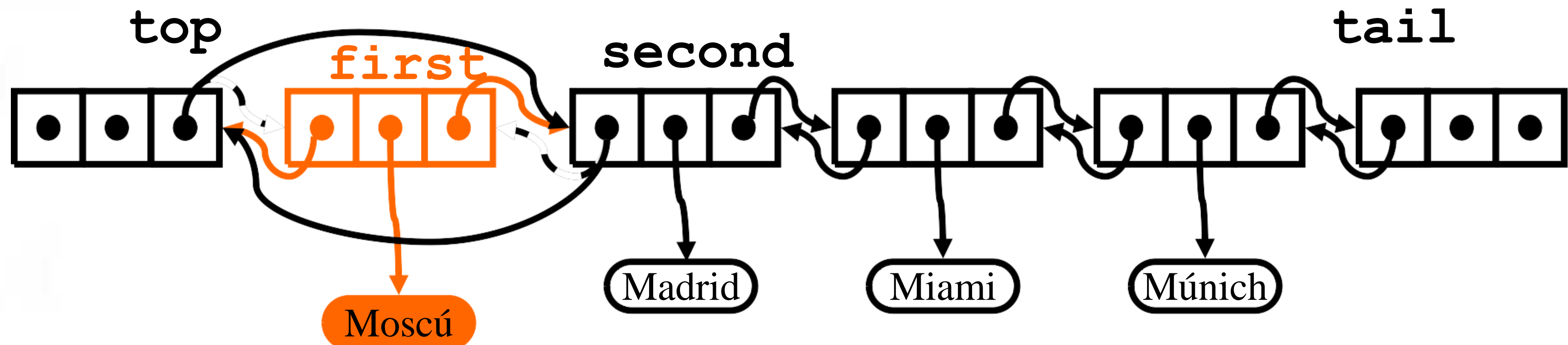
```
public void insertFirst(E info) {
    DLNode<E> second = top.getNext();
    DLNode<E> first = new DLNode<E>(info, top, second);
    second.setPrev(first);
    top.setNext(first);
    size++;
}
```



Universidad Carlos III de Madrid

La clase cola doble (`DLDeque`) con listas doblemente enlazadas: Extracción por el principio

```
public E removeFirst() {
    E info;
    if (top.getNext() == tail){
       info = null;
    } else {
        DLNode<E> first = top.getNext();
        info = first.getInfo();
        DLNode<E> second = first.getNext();
        top.setNext(second);
        second.setPrev(top);
        size--;
    }
    return info;
}
```

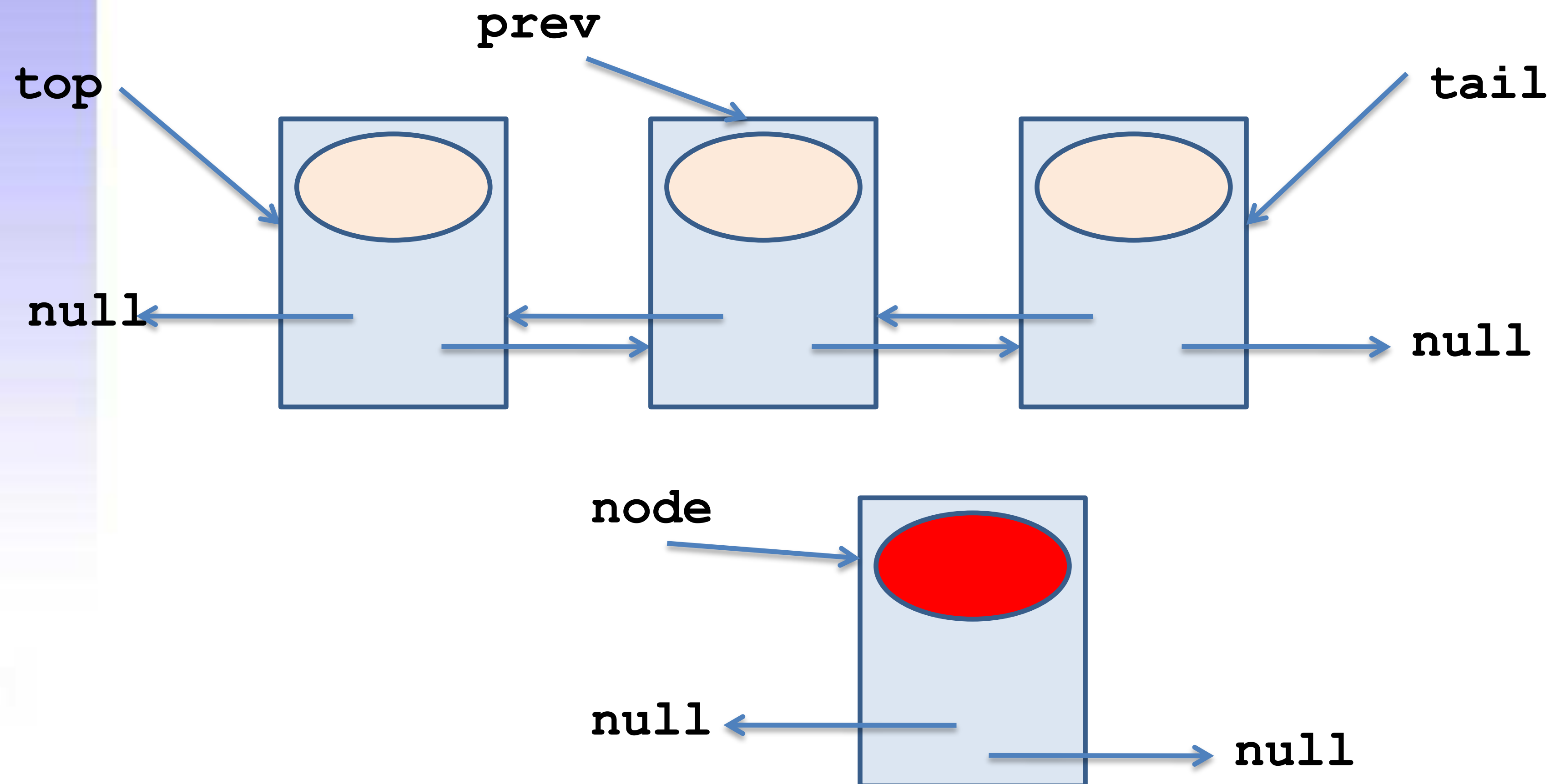


Ejercicio 3

- Implementa los siguientes métodos de la clase **`DLDeque`**:
  - **`void insertLast (E info)`**

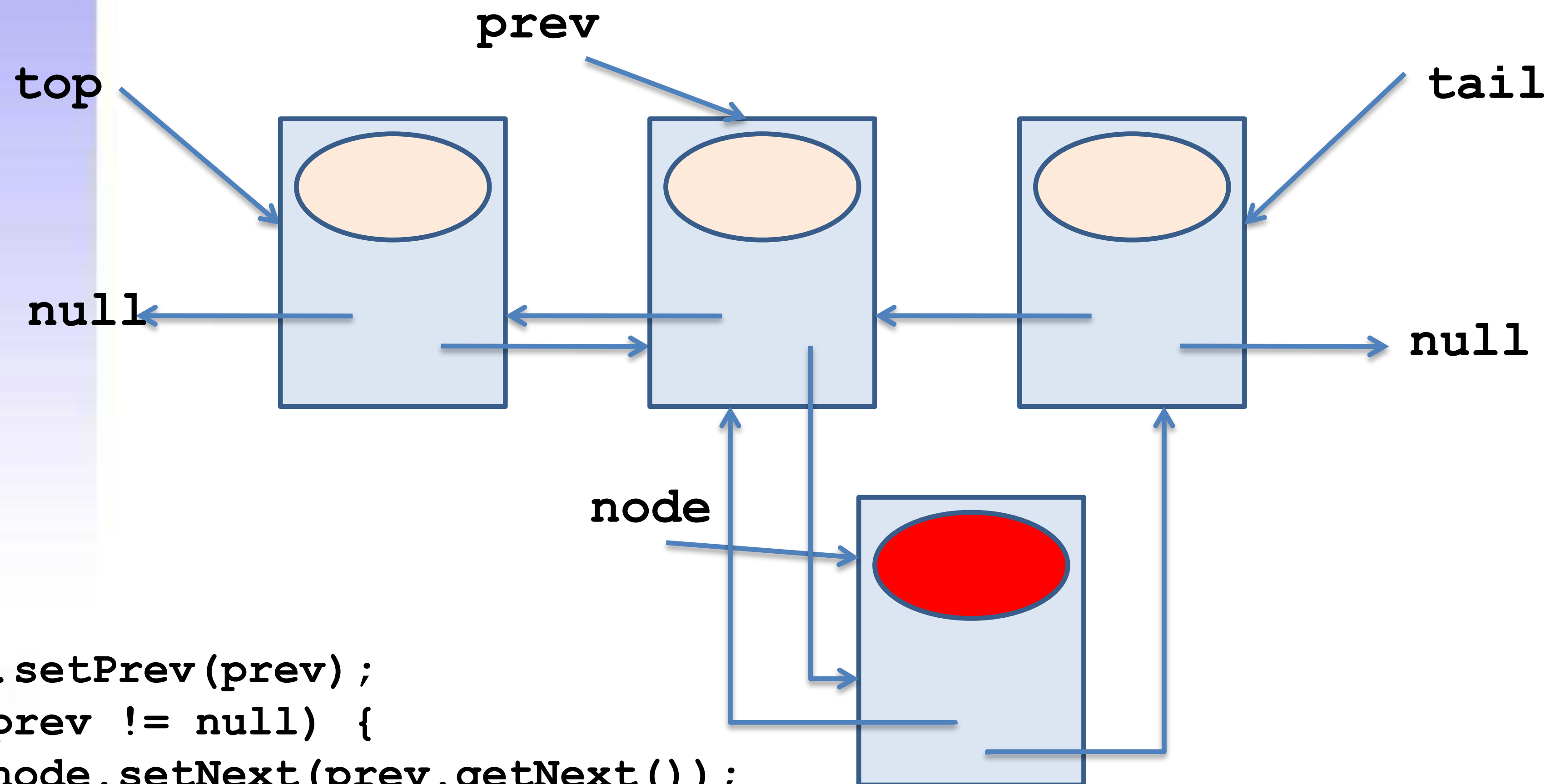
- *Para casa:*
  - ***`E removeLast()`***

Inserción (sin nodos *dummy*)




```
DLNode<E> node = new DLNode<E>(data);
```

Inserción (sin nodos *dummy*)




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

Inserción (sin nodos *dummy*)


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