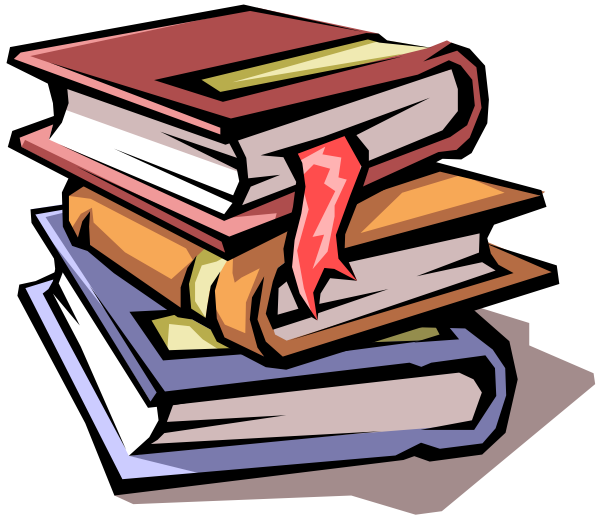
Pilas

Carlos Delgado Kloos

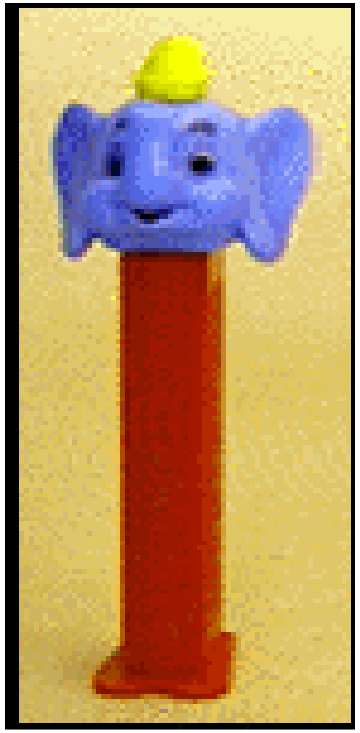
Dep. Ingeniería Telemática

Univ. Carlos III de Madrid

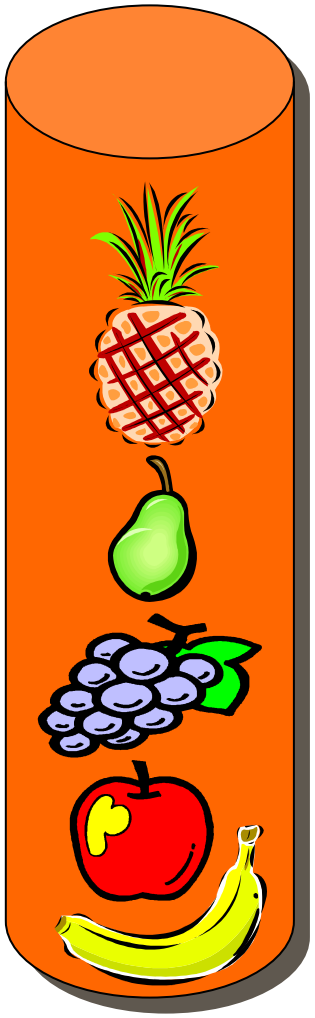
Ejemplo



Ejemplo

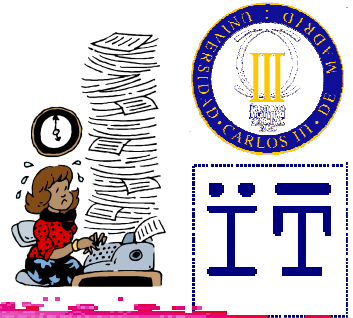


Ejemplo



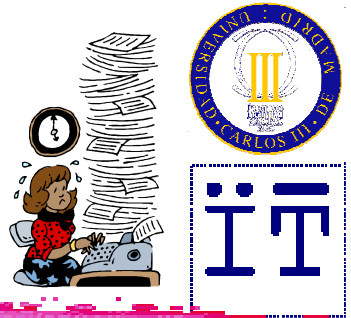
cdk@it.uc3m.es

Características



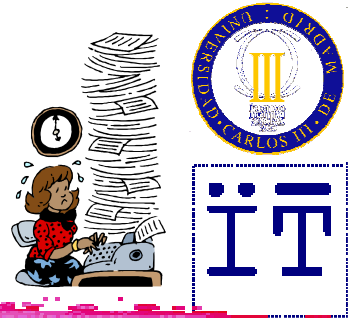
- Estructura lineal
- Acceso de inserción y eliminación por un solo extremo

Métodos principales



- Meter por un extremo: **`push(x)`**
- Sacar por el mismo extremo: **`pop()`**

Ejemplo: Comprobar paréntesis

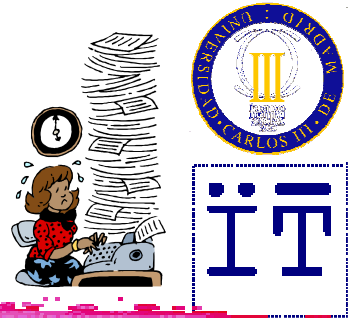


- Bien:
  -
  - ()
  - (()(()))

- Mal:
  -)(
  - (()
  - ())

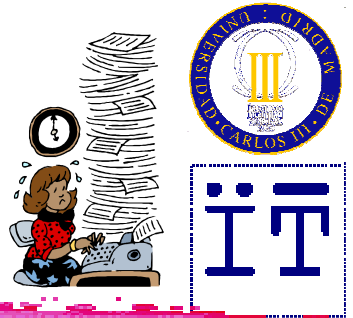
- Reglas:
  - Básico:
  - Secuenciación: ()()
  - Anidamiento: (())

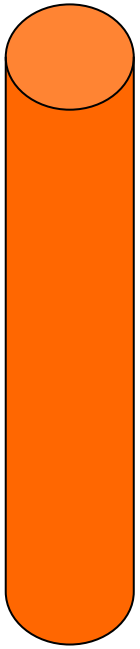
Ejemplo: Comprobar paréntesis



- Reglas:
  - Cada vez que nos encontremos "(" lo metemos en la pila.
  - Cada vez que nos encontremos ")" sacamos el "(" superior de la pila.
  - La cadena de paréntesis es correcta, si la pila está vacía al acabar de recorrer toda la cadena.

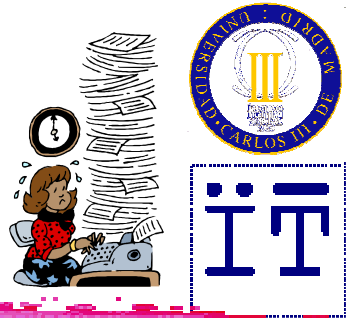
Ejemplo: comprobar (()(()()))())

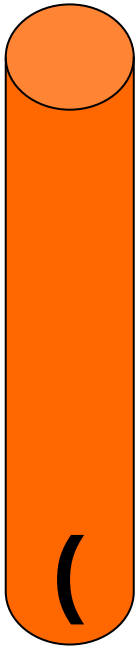




(( ) (( ) ( )) ( ))

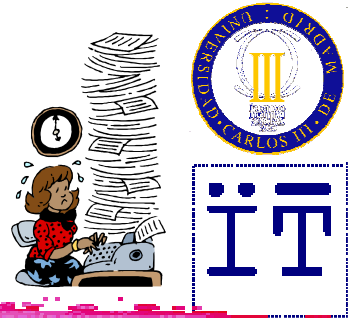
Ejemplo: comprobar (()(()()))

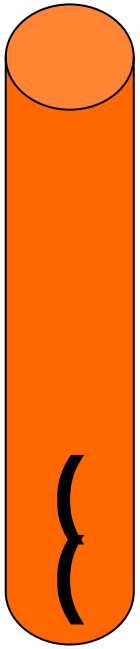




(( ) (( ) ( )) ( ))

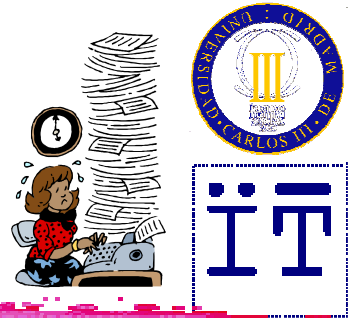
Ejemplo: comprobar (()(()()))())

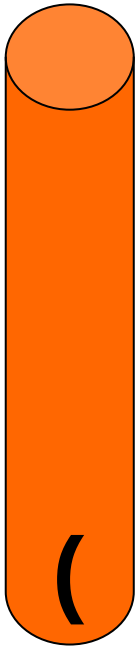




(((( ) (( ) ( )) ( ))

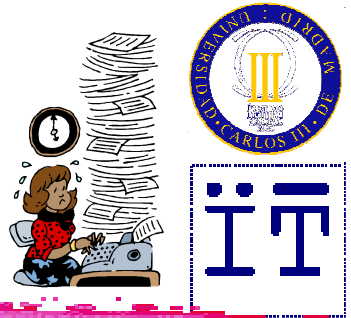
Ejemplo: comprobar (()(()()))

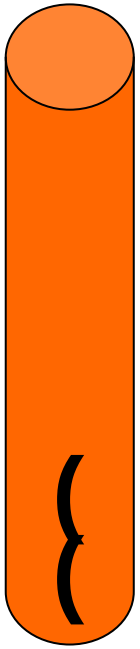




(((()()())))

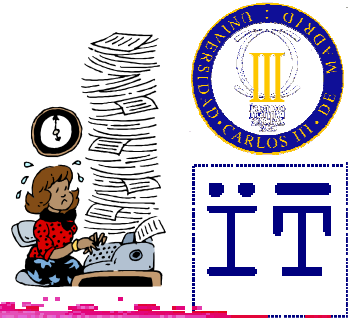
Ejemplo: comprobar (()(()()))

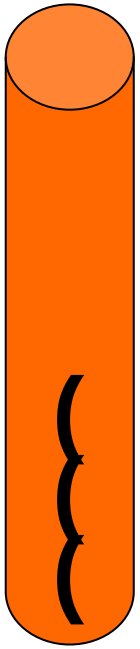




(( ) (( ) ( )) ( ))

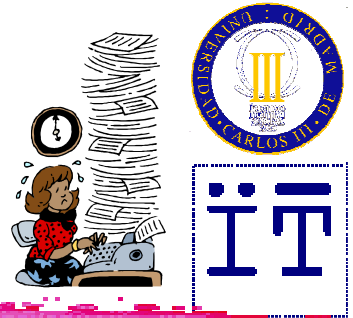
Ejemplo: comprobar (()(()()))())

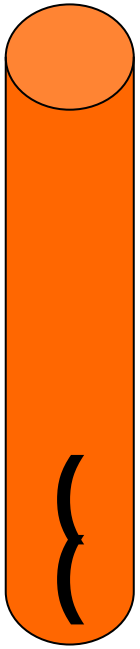




(( ) (( ) ( )) ( ))

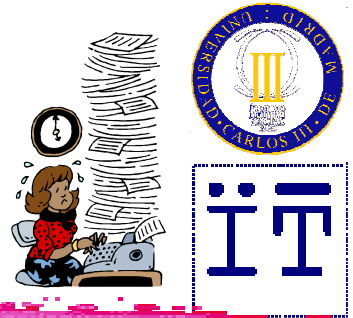
Ejemplo: comprobar (()(()()))())

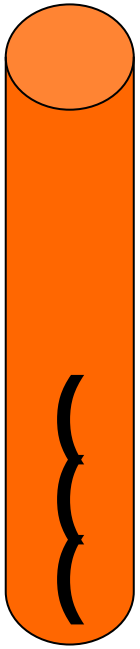




(( ) ((( ) ( )) ( ))

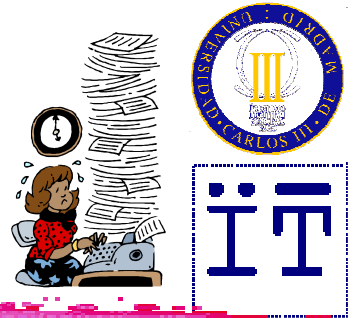
Ejemplo: comprobar (()(()()))())

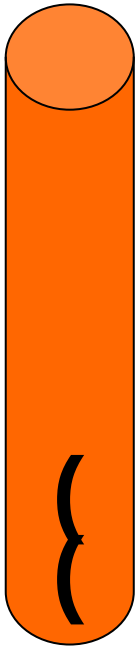




(( ) ((( ) ( )) ( ))

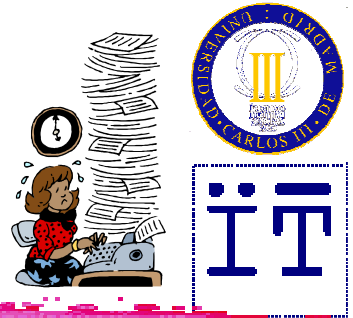
Ejemplo: comprobar (()(()()))())

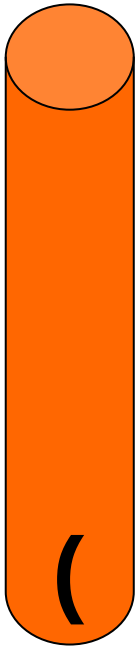




(( ) ((( ) (( )) ( ))

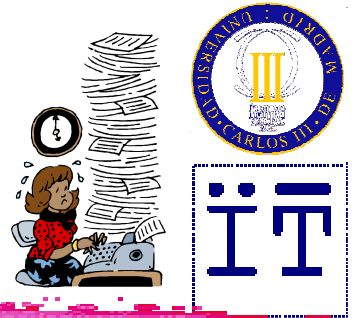
Ejemplo: comprobar (()(()()))())

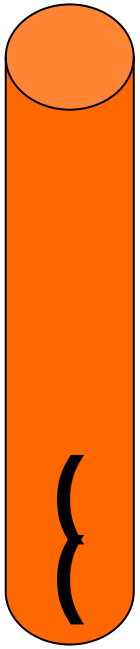




(()(()(())))()

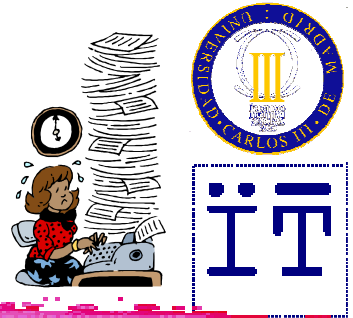
Ejemplo: comprobar (()(()()))())

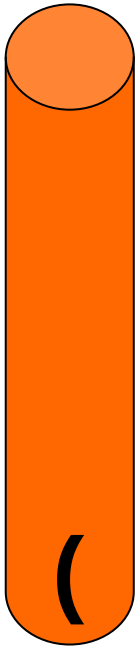




(()(()(()))())

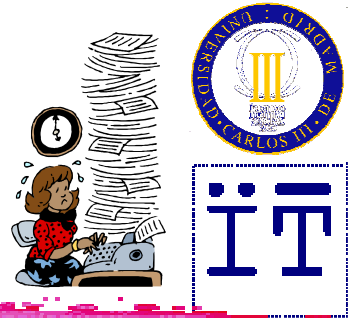
Ejemplo: comprobar (()(()()))())

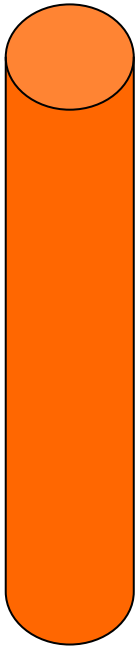




(()(()(()))()))

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

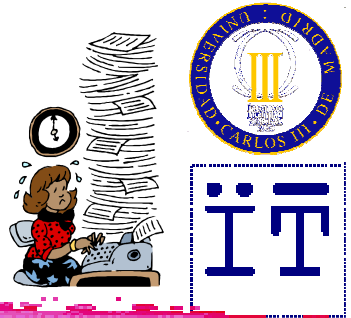


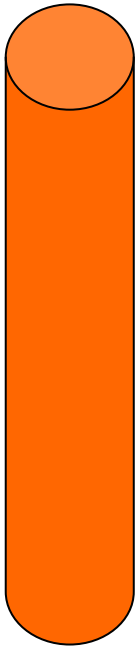


Correcto: Hemos recorrido toda la cadena y la pila está vacía

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

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





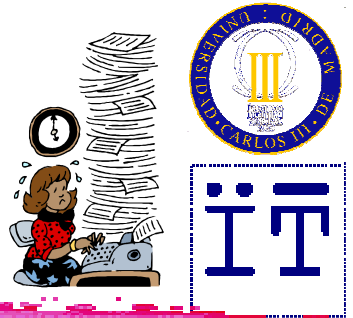
Correcto: Hemos recorrido toda la cadena y la pila está vacía

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

Ejemplo: HTML

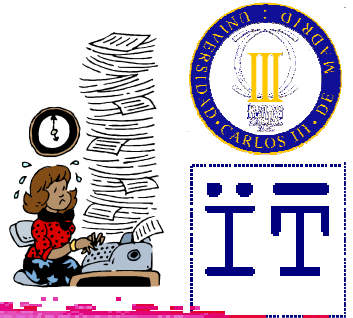
- **hola**
  - ([)]
  - Correcto con HTML 1.0-4.0
  - Incorrecto con XHTML
- **hola**
  - ([])
  - Correcto con HTML 1.0-4.0
  - Correcto con XHTML

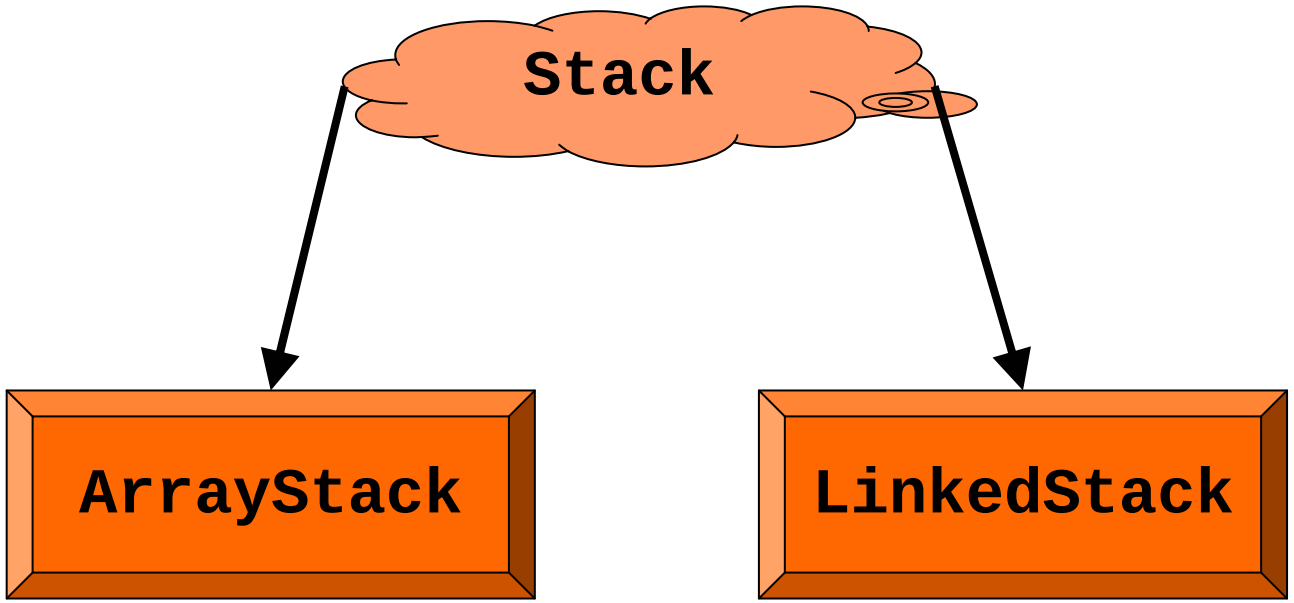
Interfaz para pilas



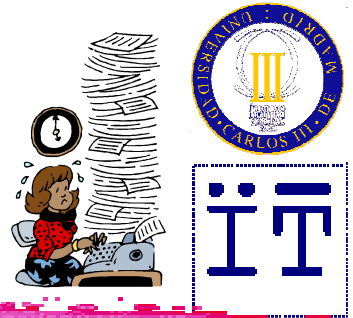
```
public interface Stack {
   public void push(Object o)
     throws StackOverflowException;
   public Object pop()
     throws EmptyStackException;
   public Object top()
     throws EmptyStackException;
   public int size();
   public boolean isEmpty();
}
```

Un interfaz y varias implementaciones

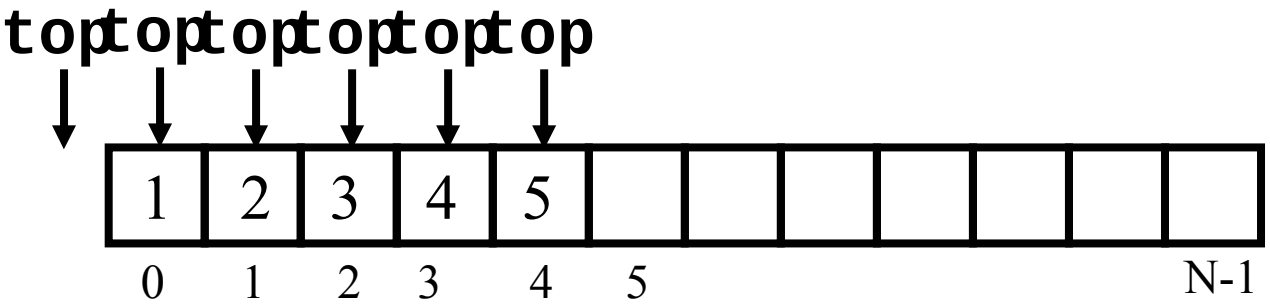




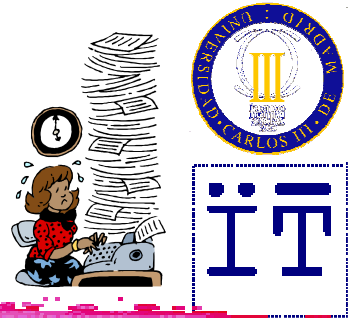
Implementación basada en arrays



S

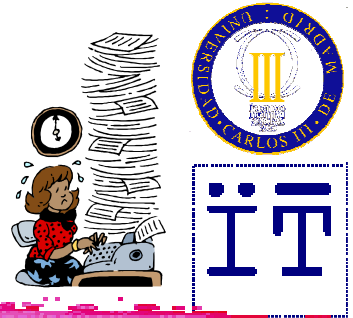


Implementación basada en arrays



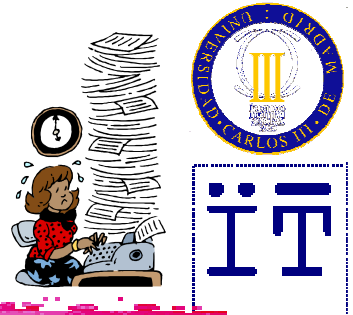
```
public class ArrayStack implements Stack {
  public static final int DEFAULT_CAPACITY = 1000;
  private int capacity;
  private Object data[];
  private int top = -1;
  public ArrayStack() {
    this(DEFAULT_CAPACITY);
  }
  public ArrayStack(int capacity) {
    this.capacity = capacity;
    data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

Implementación basada en arrays



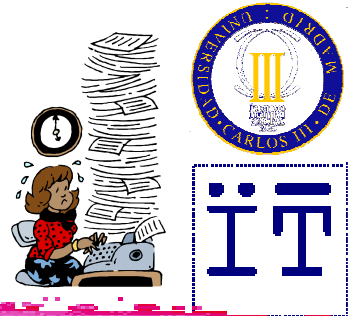
```
public int size() {
  return (top + 1);
}
public boolean isEmpty() {
  return (top < 0);
}
public Object top()
  throws EmptyStackException {
  if (top == -1)
    throw new EmptyStackException("empty");
  return data[top];
}
```

Implementación basada en arrays



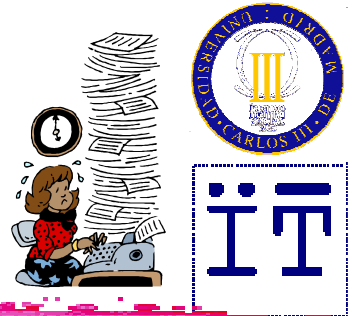
```
public void push(Object o)
  throws StackOverflowException {
  if (top == capacity - 1)
    throw new StackOverflowException();
  data[++top] = o;
}
```

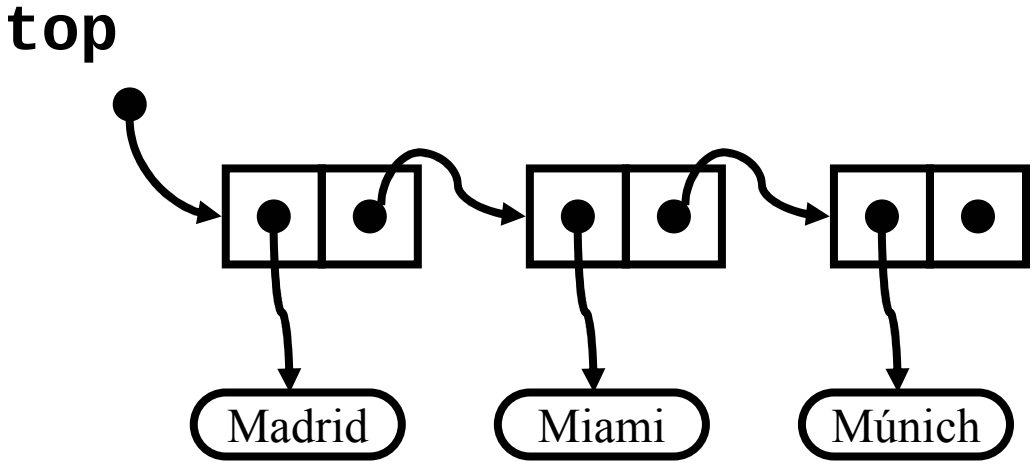
Implementación basada en arrays



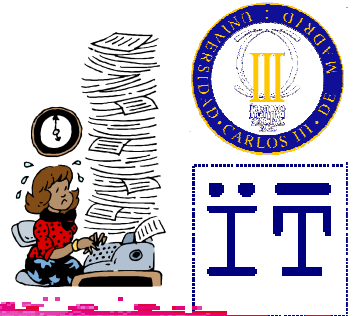
```
public Object pop()
  throws EmptyStackException {
  Object o;
  if (top == -1)
    throw new EmptyStackException();
  o = data[top];
  data[top--] = null;
  return o;
}
```

Implementación basada en listas encadenadas



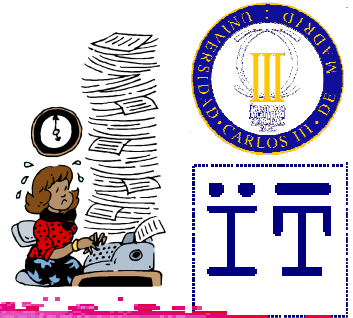


Implementación basada en listas



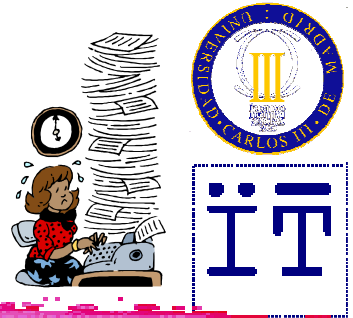
```
class Node {
  private Object info;
  private Node next;
  public Node(Object info, Node next) {
    this.info = info;
    this.next = next;
  }
  void setInfo(Object info) {this.info = info;}
  void setNext(Node next) {this.next = next;}
  Object getInfo() {return info;}
  Node getNext() {return next;}
}
```

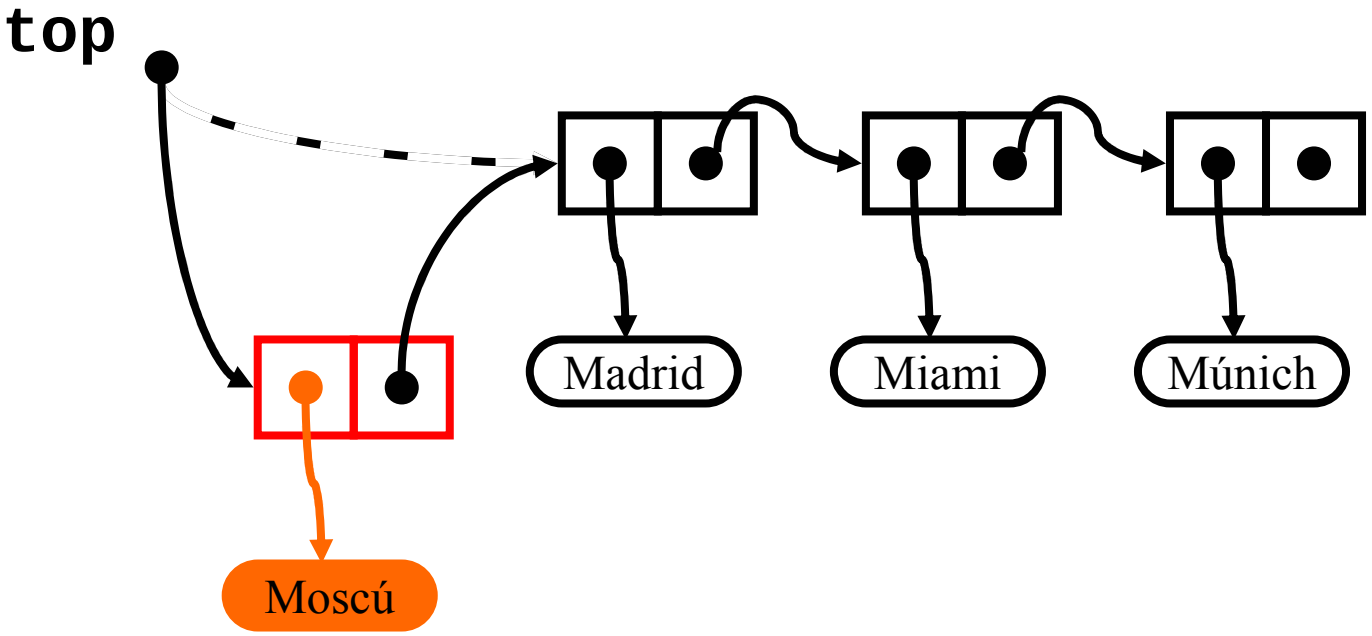
Implementación basada en listas



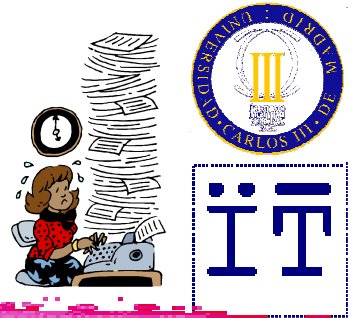
```
public class LinkedStack implements Stack {
  private Node top;
  private int size;
  public LinkedStack() {
    top = null;
    size = 0;
  }
  public int size() {
    return size;
  }
  public boolean isEmpty() {
    return (top == null);
  }
```

Inserción (push)



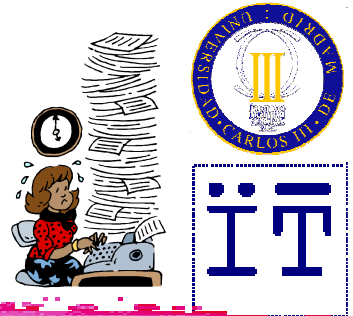


Implementación basada en listas



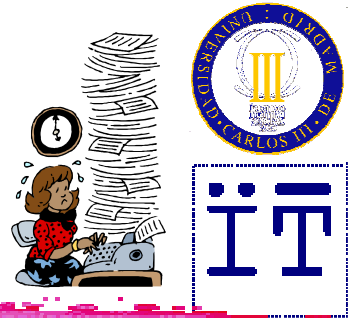
```
public void push(Object info) {
   Node n = new Node(info, top);
   top = n;
   size++;
}
```

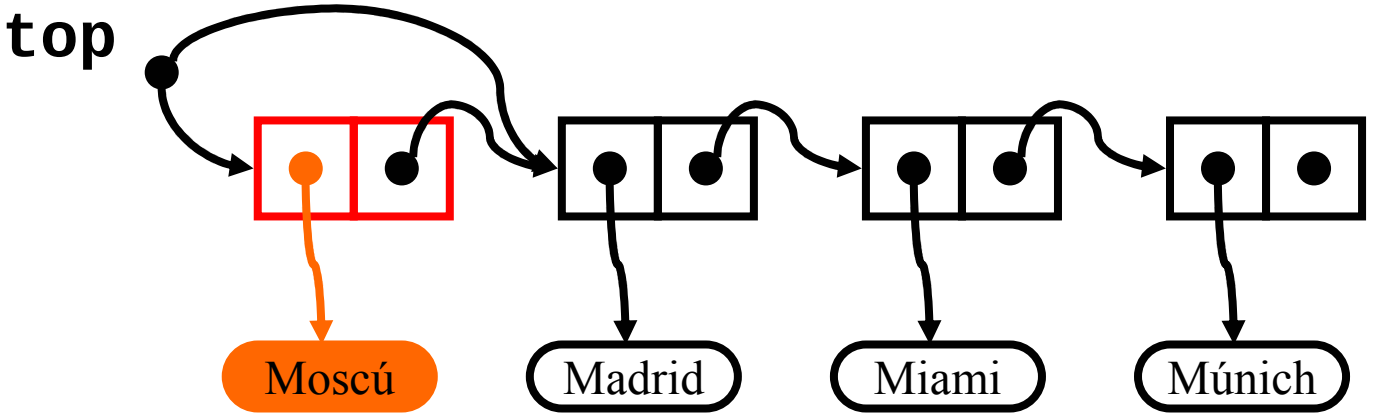
Implementación basada en listas



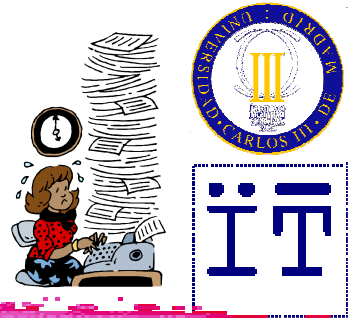
```
public Object top()
   throws EmptyStackException {
   if (top == null)
      throw new EmptyStackException();
   return top.getInfo();
}
```

Borrado (pop)



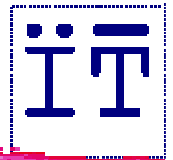


Implementación basada en listas



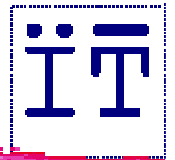
```
public Object pop()
   throws EmptyStackException {
   Object info;
   if (top == null)
      throw new EmptyStackException();
   info = top.getInfo();
   top = top.getNext();
   size--;
   return info;
}
```

Arrays vs. Listas encadenadas



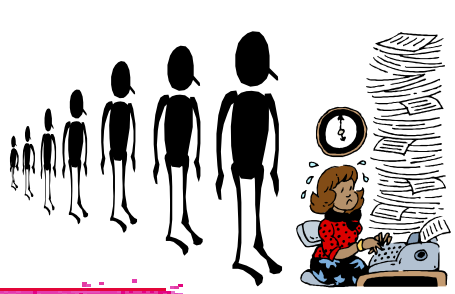
- Ventajas de los arrays:
  - Uso eficiente de memoria cuando en el momento de crearlo se conoce el número de datos a almacenar
  - Acceso muy rápido a posiciones arbitrarias
- Desventajas de los arrays:
  - Es necesario mover datos en inserciones, eliminaciones, concatenaciones, etc.
  - Tamaño estático (se puede redimensionar, pero esto requiere movimiento de datos)
  - Necesita que toda la memoria sea contigua

Arrays vs. listas encadenadas



- Ventajas de las listas encadenadas:
  - Inserciones, extracciones, concatenaciones, particiones sin movimiento de datos
  - Tamaño dinámico
  - No necesita memoria contigua
- Desventajas de las listas encadenadas:
  - Acceso lento a posiciones arbitrarias
  - Uso menos eficiente de memoria debido al espacio requerido por los objetos nodo

Pilas y recursión



```
public static long fac (int n) {
  if (n <= 1)
     return 1;
   else
     return n * fac(n - 1);
  }
```

Ejecución

- `fac(4)`
- `4*fac(3)`
- `4*(3*fac(2))`
- `4*(3*(2*fac(1)))`
- `4*(3*(2*1))`
- `4*(3*2)`
- `4*6`
- `24`