

# Tema 3

## ESTRUCTURAS LINEALES



TEMA	SUBTEMA
<p data-bbox="112 232 645 361">3 ESTRUCTURAS LINEALES</p>	<p data-bbox="736 139 938 187">3.1 Pilas</p> <ul data-bbox="768 218 1644 429" style="list-style-type: none"><li data-bbox="768 218 1644 265">3.1.1 Representación en memoria</li><li data-bbox="768 297 1450 344">3.1.2 Operaciones básicas</li><li data-bbox="768 375 1238 422">3.1.3 Aplicaciones</li></ul> <p data-bbox="736 454 973 501">3.2 Colas</p> <ul data-bbox="768 532 1644 901" style="list-style-type: none"><li data-bbox="768 532 1644 579">3.2.1 Representación en memoria</li><li data-bbox="768 611 1450 658">3.2.2 Operaciones básicas</li><li data-bbox="768 689 1495 822">3.2.3 Tipos de colas: simples, circulares y bicolas</li><li data-bbox="768 853 1238 901">3.2.4 Aplicaciones</li></ul> <p data-bbox="736 932 958 979">3.3 Listas</p> <ul data-bbox="768 1011 1760 1372" style="list-style-type: none"><li data-bbox="768 1011 1450 1058">3.3.1 Operaciones básicas</li><li data-bbox="768 1089 1760 1293">3.3.2 Tipos de listas: simplemente enlazadas, doblemente enlazadas y circulares</li><li data-bbox="768 1325 1209 1372">3.3.3 Aplicaciones</li></ul>

## 3.1 Pilas estáticas y dinámicas





➤ Pilas.

➤ Representación en memoria  
estática y dinámica.

➤ Operaciones básicas con pilas.

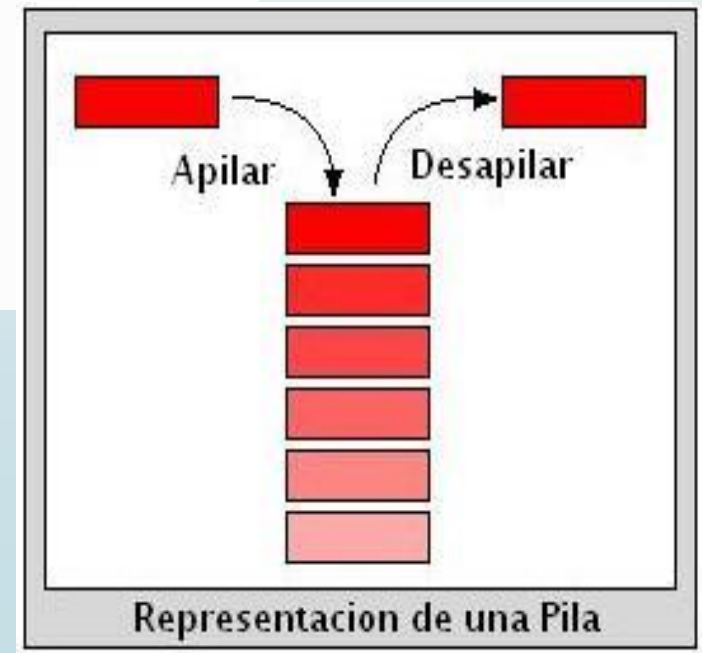
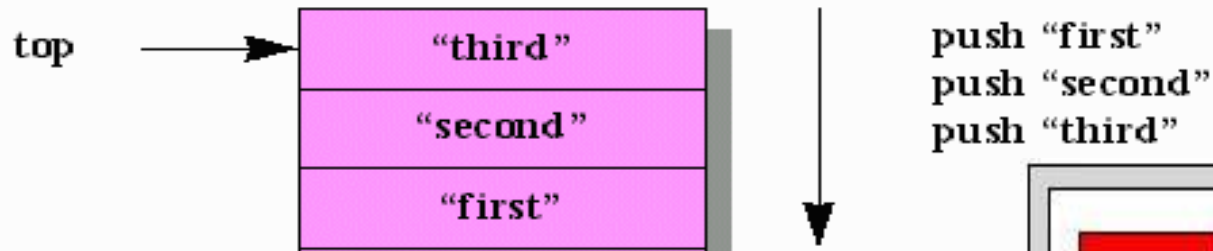
➤ Aplicaciones.

➤ Notación infija y postfija.

➤ Recursividad con ayuda de pilas.

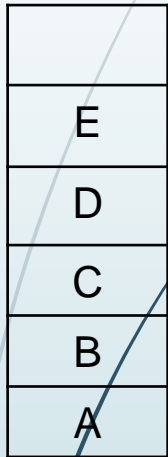
# Definición de Pila

- Una pila es una estructura de datos en la cual el acceso está limitado al elemento más recientemente insertado y solamente puede crecer y decrecer por uno de sus extremos.



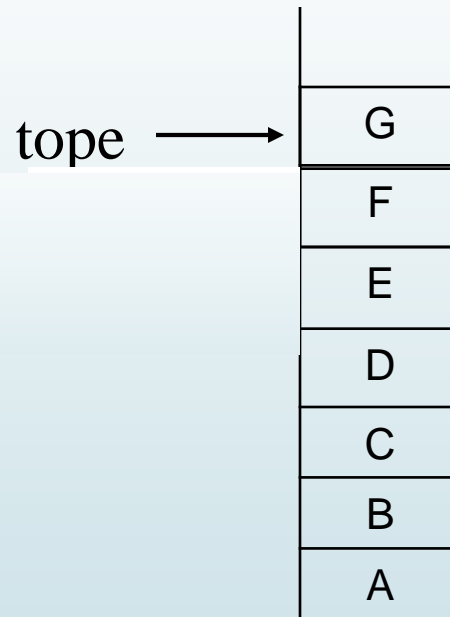
# Definición de pila

- Es un conjunto ordenado de elementos en el cual se pueden agregar y eliminar elementos en un extremo, llamado **tope** de la pila.

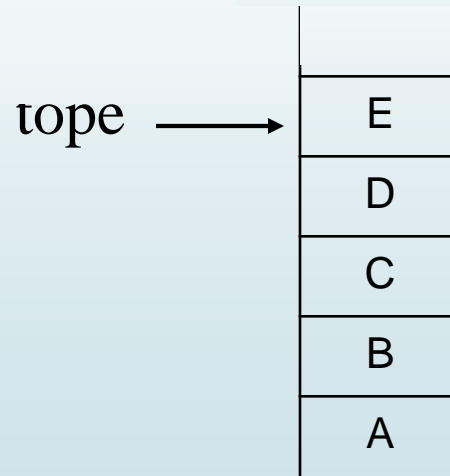


- A diferencia del arreglo la definición de la pila considera la inserción y eliminación de elementos, por lo que una pila es un objeto dinámico en constante cambio.

# Inserciones en una pila




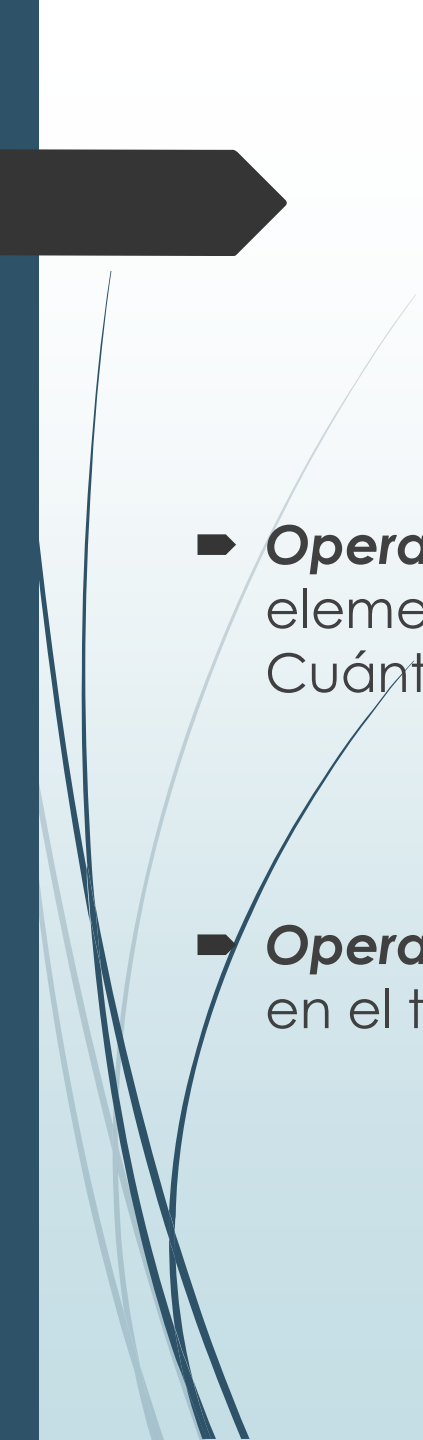
# eliminaciones en una pila



- Las pilas se denominan también estructuras LIFO (Last-In-First-Out), porque su característica principal es que el último elemento en llegar es el primero en salir.



- 
- En todo momento, el único elemento visible de la estructura es el último que se colocó.
  - Se define el **tope** de la pila como el punto donde se encuentra dicho elemento.

- 
- **Operaciones de acceso:** Pila vacía(empty), Qué elemento está en el Entope(Top), Pila llena(Full), Cuántos elementos hay en la pila(Total)
  - **Operaciones de transformación:** poner un elemento en el tope(push), quitar un elemento del tope(pop)

# Métodos

- `push( x )` --> (apilar) Inserta x
- `pop( )` --> (desapilar) Elimina el último elemento insertado
- `info( )` --> Retorna el último elemento insertado, sin eliminarlo.
- `pilallena( )` --> Retorna true si no existen espacios para un nuevo elemento; en caso contrario.
- `pilavacia( )` --> Retorna true si no existen elementos ; false en caso contrario
- `vaciar( )` --> Elimina todos los elementos .
- `contarl ( )` --> Regresa el numero. De elementos en la pila.
- `Top ( )` --> Regresa posición del último elemento insertado.
- `Imprime( )` --> Muestra contenido de la lista.
- `Invierte( )` --> Genera una nueva pila con el contenido en orden inverso.
- `Busca(x )` --> Retorna true si el elemento x esta en la pila, false en caso contrario.



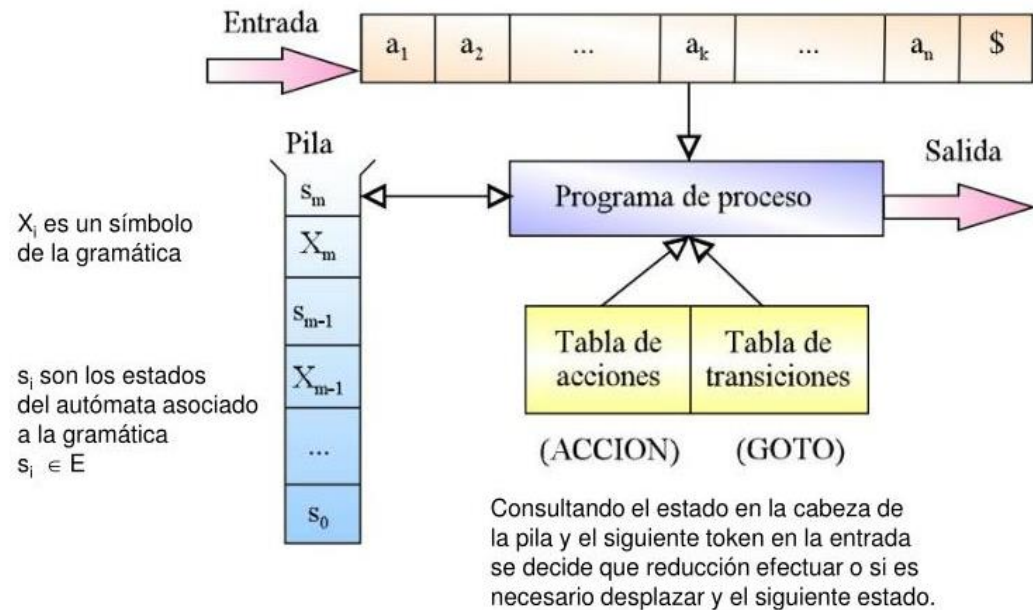
# Algunas aplicaciones

## Lenguajes de programación

- Los compiladores comprueban los programas buscando errores sintácticos. Se puede utilizar una pila para comprobar si hay símbolos desequilibrados.



- Estructura general de un analizador LR

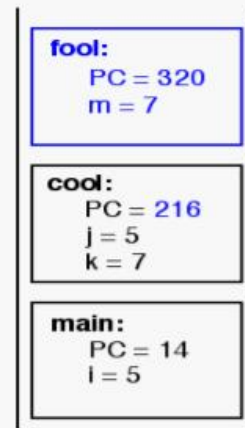


# Algunas aplicaciones

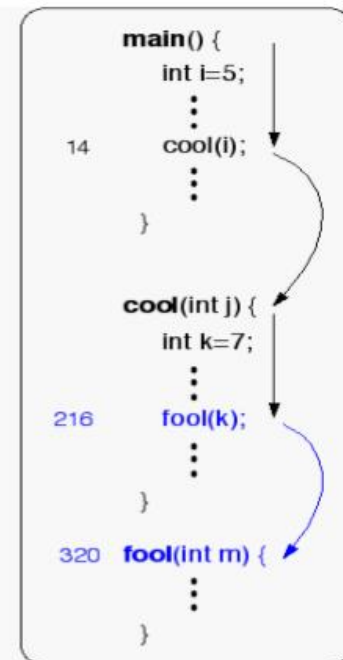
## Lenguajes de programación

- Implementación de llamadas a procedimientos. Las pilas son utilizadas en la mayoría de los lenguajes para implementar las llamadas a métodos. (JVM)

### Pila de métodos en Java



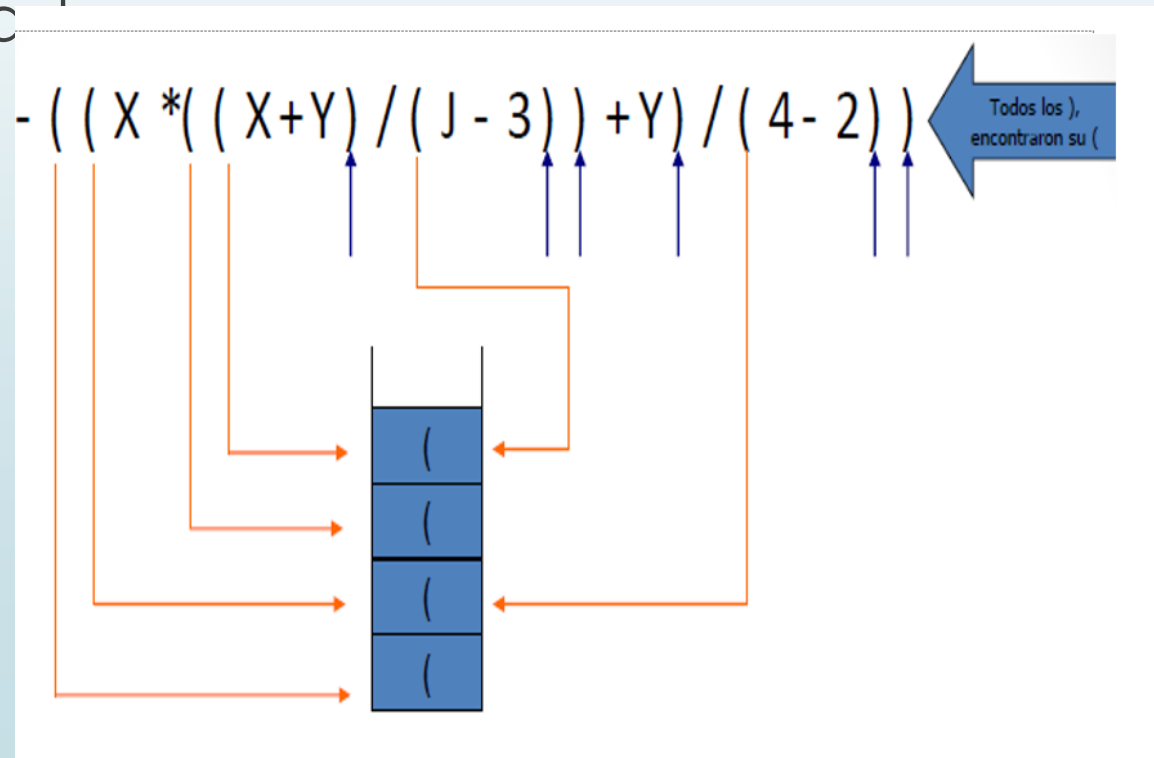
Java Stack



# Algunas aplicaciones

## Lenguajes de programación

- Evaluación de expresiones aritméticas en lenguajes de programación. Se utiliza una pila para la evaluación con precedencia entre operadores.



# Aplicaciones

- Historia de las páginas visitadas en un Web browser.
- Secuencia deshacer en un editor de texto.
- Llamadas a métodos en Java Virtual Machine.
- Estructura de datos auxiliar para algoritmos.
- Recursividad

# Operaciones

- Diseño de algoritmos





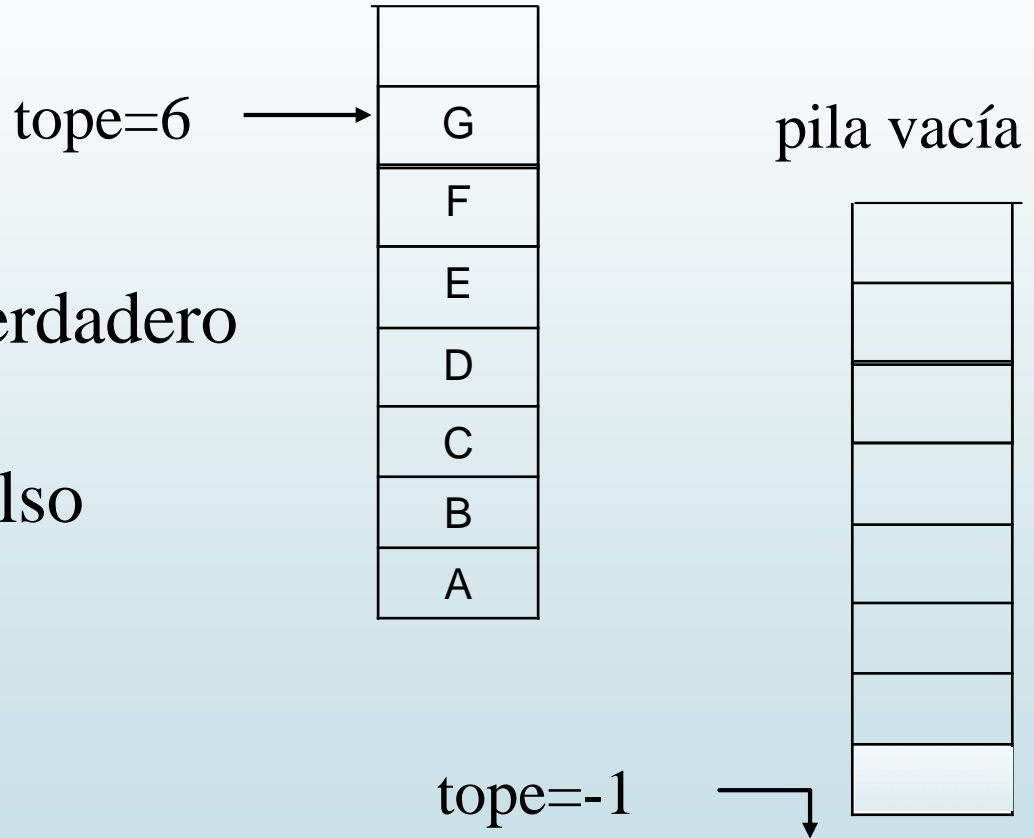
# PILAVACIA()

{Este algoritmo verifica si la pila esta vacía, asignando a band el valor de verdad correspondiente}

- ```

1. Inicio
2. Si tope == -1           tope = 6
   entonces
       hacer band = verdadero
   si no
       hacer band = falso
3. Fin_condicional_1
4. Regresa band
5. Fin

```



# PILALLENA( )

{este algoritmo verifica se la pila esta llena, asignando a band el valor de verdad correspondiente}

1. Inicio

2. Si  $\text{tope} == \text{max} - 1$

entonces

hacer  $\text{band} = \text{verdadero}$

sino

hacer  $\text{band} = \text{falso}$

3. Fin del condicional\_1

4. Regresa band

5. Fin

$\text{tope} = 7$



|   |
|---|
| H |
| G |
| F |
| E |
| D |
| C |
| B |
| A |

# Apilar( dato)

{Este algoritmo pone el elemento dato en PILA. Actualiza el valor de tope. max el número máximo de elementos que puede almacenar PILA}

1. Inicio
2. Llamar a PILALLENNA( )
3. Si band == verdadero  
entonces  
    Escribe “desbordamiento”  
sino  
    hace tope = tope++  
    elem[tope]= dato
4. Fin del condicional\_paso\_2
5. Fin

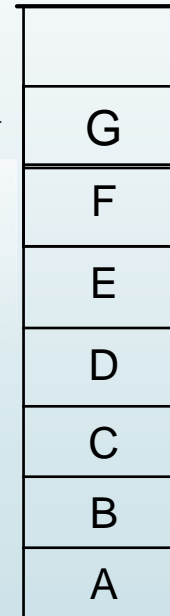
```
Apilar(int dato){  
    if (pilallena( )) {  
  
sout(“desbordamiento”);  
    else {  
        tope++;  
        elem[tope]=dato;  
    }  
  
}
```

## Apilar( dato)

{Este algoritmo pone el elemento dato en PILA. Actualiza el valor de tope. max es el número máximo de elementos que puede almacenar PILA}

1. Inicio
2. Llamar a PILALLENNA( )
3. Si band == verdadero  
entonces  
    Escribe “desbordamiento”  
sino  
    hace tope = tope++  
    elem[tope]= dato
4. Fin del condicional\_paso\_2
5. Fin

tope=6



# desapilar( )

{Este algoritmo borra el elemento dato de PILA }

1. Inicio
2. Si ( PILAVACIA( ) )  
    entonces  
        escribe "subdesbordamiento"  
    sino  
        hace dato = elem[tope]  
        tope= tope -1
3. Fin del condicional\_1
4. Regresa dato
5. Fin

tope=6

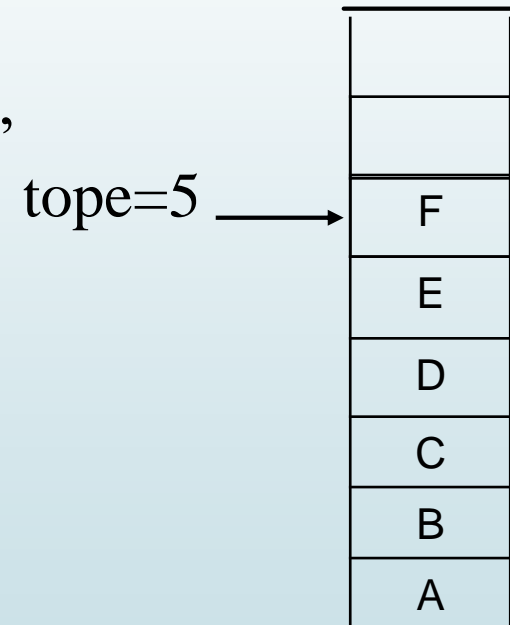


|   |
|---|
|   |
| G |
| F |
| E |
| D |
| C |
| B |
| A |

## Info( )

*{retorna el último elemento en la PILA, sin eliminarlo}*

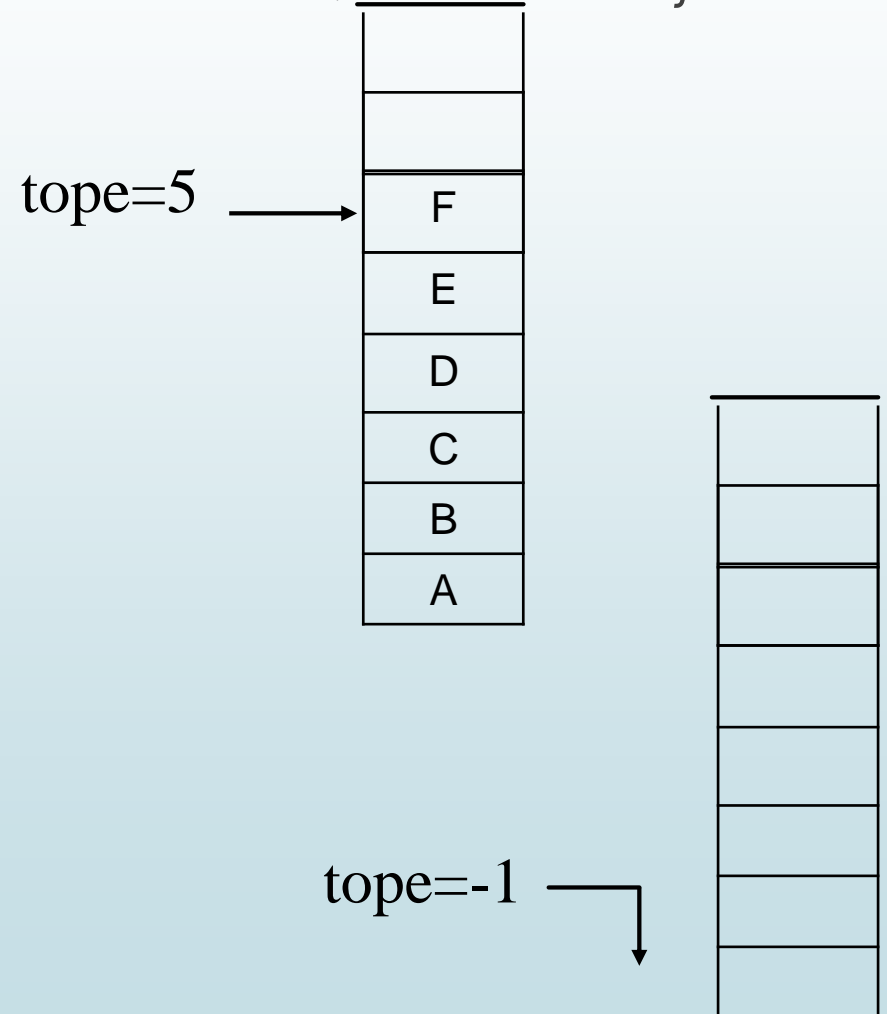
1. Inicio
2. Llamar a PILAVACIA( )
3. Si band == verdadero  
entonces  
    Escribe“ERROR NO HAY DATOS”  
sino  
    dato= elem[tope]
4. Fin del condicional\_paso\_2
5. Retorna dato
6. Fin



# Ejercicios en clase:

- 1) Diseña un algoritmo para contar los elementos de una pila.
- 2) Diseña un algoritmo para imprimir el contenido de una pila.
- 3) Diseña un algoritmo para invertir una pila





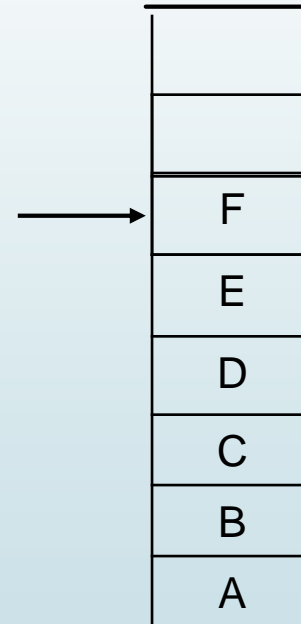


## contar( )

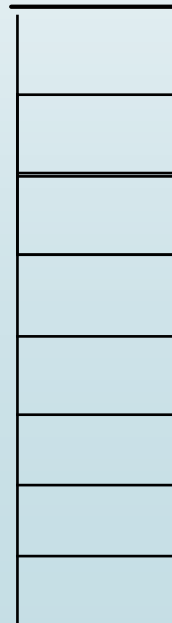
{retorna el número de elementos en la PILA, sin eliminarlos}

1. Inicio
2. `cont=0`
3. Llamar a `PILAVACIA( )`
4. Si `band == verdadero`  
entonces  
Escribe "ERROR NO HAY DATOS"  
sino  
repetir para `i` desde 0 hasta `tope`, `i++`  
`cont++`
5. Fin del condicional\_paso\_2
6. Retorna `cont`
7. Fin

`tope=5`



`tope=-1`

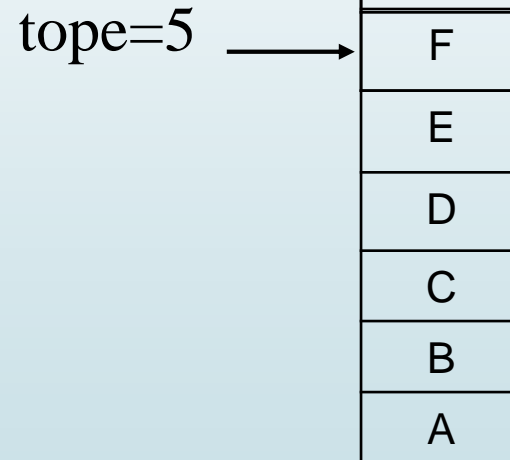


## contar( )

{retorna el número de elementos en la PILA, sin eliminarlos}

("versión mejorada")

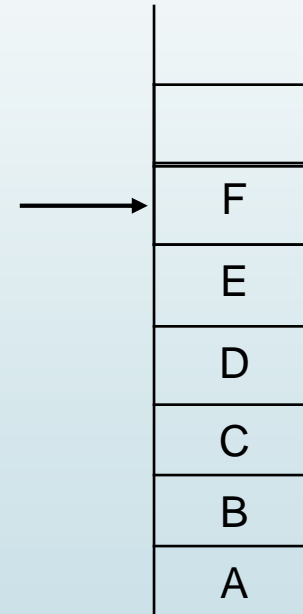
1. Inicio
2. retorna  $\text{tope}+1$
3. Fin



## imprimepila( )

*{muestra los elementos en la PILA, sin eliminarlos}*

1. Inicio
2. Llamar a PILAVACIA( )
3. Si band == verdadero  
entonces  
    Escribe“ERROR NO HAY DATOS”  
sino  
    tope=5  
    repetir para i desde 0 hasta tope, i++  
        Escribe elem[i]
4. Fin del condicional\_paso\_2
5. Retorna cont
6. Fin

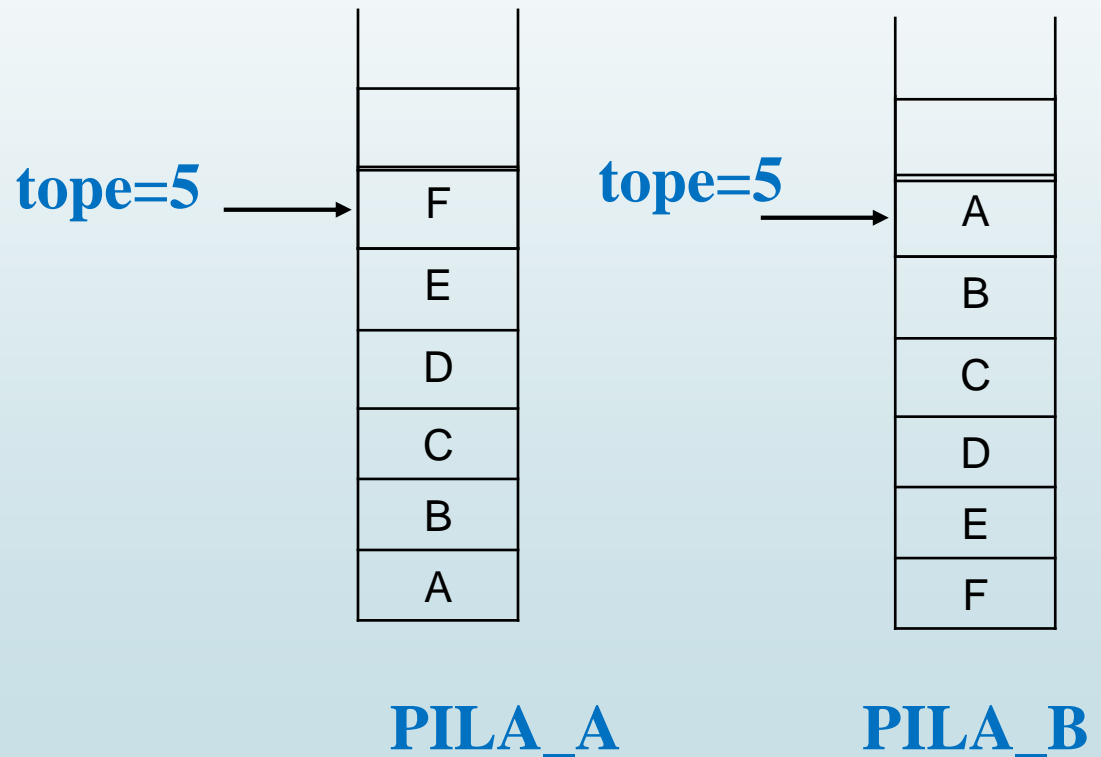


## Ejercicio

Diseña un algoritmo para invertir una pila

`inviertepila1 ( )`

*{invierte el contenido de una pila}*



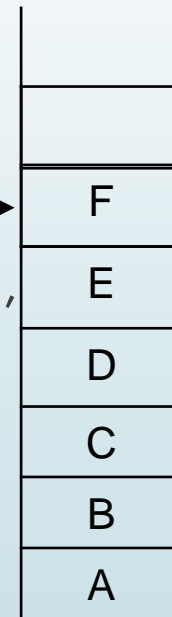
## Ejercicio

Diseña un algoritmo para invertir una pila

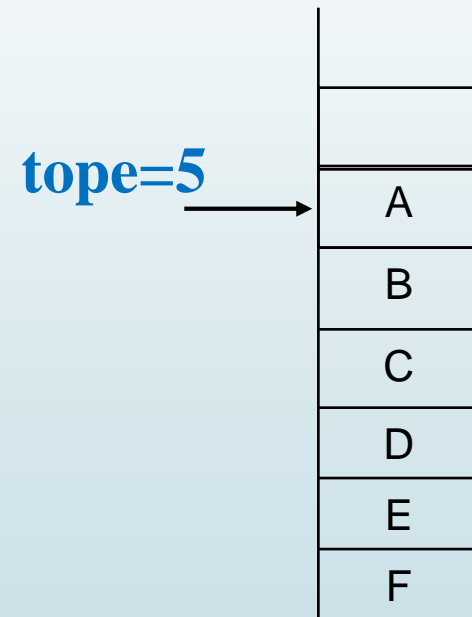
inviertepila1 ( )

{invierte el contenido de una pila}

1. Inicio
2. Si pilavacia()
3. Escribe "no hay datos"
4. sino **tope=5** →
5. crea una nueva pila (pila1),
6. repetir para i=0 hasta tope
7. d=pila.desapilar()
8. pila1.apilar(d)
9. Fin\_condicional\_paso\_2
10. Fin\_algoritmo



**PILA**



**PILA1**



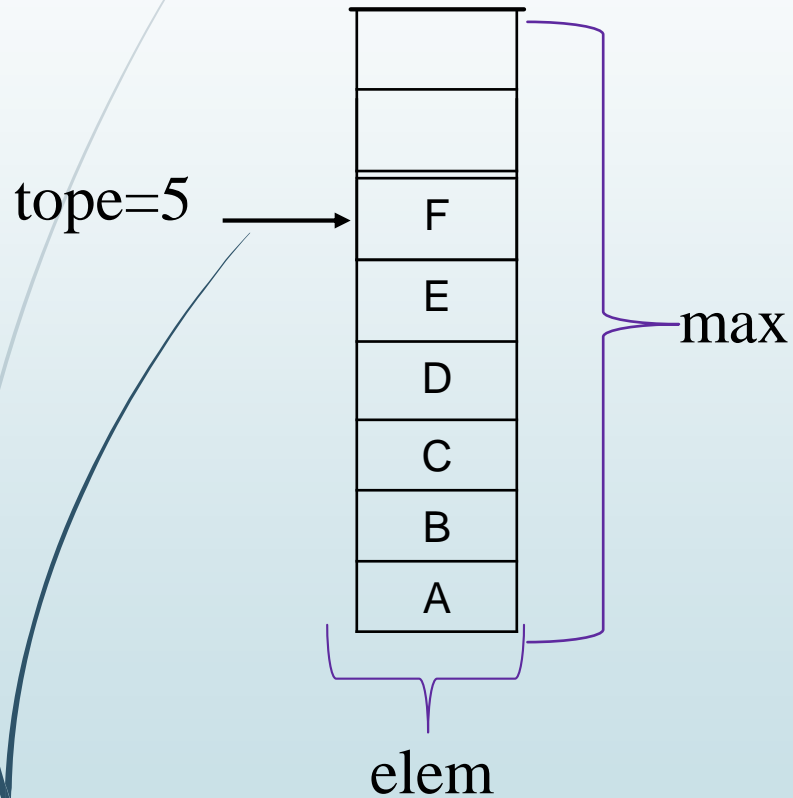
## Ejercicio

Diseña un algoritmo para buscar un elemento en una pila



## Diagrama UML de la clase Pila

Pila: A



| Pila                                                                                                                                                   |
|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| max<br>tope<br>elem[ ]                                                                                                                                 |
| Pilavacia ( )<br>Pilallena( )<br>apilar( )<br>desapilar( )<br>Imprimepila( )<br>Vaciar( )<br>Top( )<br>Info( )<br>Invertir( )<br>Total( )<br>Buscar( ) |

► Declaración en Java de la clase Pila

```
public class Pila
{
    // atributos
    private int tope;
    private int max;
    private int[ ] elem;
    // constructor; crea una pila vacía
    public Pila(int n ) {
        this.max= n;
        elem= new int [max] ;
        tope= -1;
    }
    // métodos
    public int top( )
    {
        return tope;
    }
    . . .
} // fin de la clase Pila
```





# Ejercicio práctico

- Implementa en Java la clase Pila y sus operaciones
- 



# FUENTES DE INFORMACION

- [CairoGuardati. Estructura de datos. Mc Graw Hill](#)
- [<https://estructura09110907.blogspot.com/2011/10/pilas.html>](#)