

Estructuras de Datos

TDA Pila

Adriana Collaguazo Jaramillo, Mg.

LA PILA: UN TDA SIMPLE

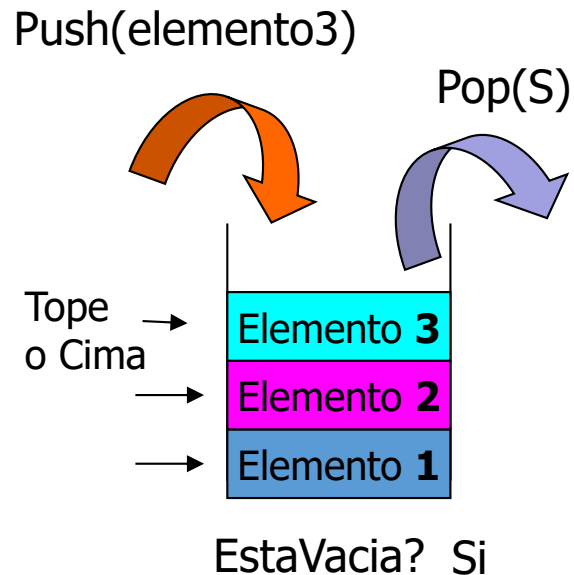
- Uno de los conceptos más útiles en computación es la pila o stack
 - Es un conjunto de elementos, en la que:
 - Los elementos se añaden y se remueven por un solo extremo
 - Este extremo es llamado “tope” de la pila
- Ejemplo:
- Cuando un empleado se va de vacaciones, le llega correo a su escritorio.
 - Las cartas se van “apilando”.
 - Al regresar de vacaciones, la última carga en llegar, será la primera que revisará
 - Al terminar de revisarla, la nueva carta del tope de la pila habrá cambiado
 - Del “pilo” de cartas, la más nueva que queda, será la siguiente en ser revisada

La última en llegar,
será la primera en
salir:
LAST IN, FIRST OUT
LIFO



TDA PILA: DEFINICION

- Dada una Pila llamada S
 - ¿Qué datos serian importantes conocer sobre la Pila?
 - ¿Y que operaciones podríamos efectuar a la misma?



□ Usemos el ejemplo del correo:

□ Al acumularse,

□ Cada carta(elemento), era “metida” a la pila:
pila.push(elemento)

□ La operación push aumenta un elemento a la pila, y esta aumenta en su tamaño

□ Al revisar c/carta, se la “sacaba” de la pila

□ **elemento = pila.pop()**

□ La operación pop remueve el elemento **Tope** de la pila y lo retorna. La pila disminuye su tamaño

PILA: OPERACIONES

size() retorna -> int

Efecto: Devuelve el tamaño

pop() retorna -> *elemento*

Efecto: Remueve el elemento tope y lo **retorna**

Excepcion: Si la pila esta vacía, produce error

push(elemento)

Efecto: Toma la pila y aumenta su tamaño, poniendo el elemento en la cima de la pila

peek() retorna -> *elemento*

Efecto: Devuelve el elemento cima/tope de la pila

Excepcion: Si la pila esta vacía produce error

OPERACIONES BASICAS DE LA PILA

□ Push

- Inserta un elemento en la pila, cuando se puede
 - Si la pila esta llena, no se puede insertar
 - Error
- Este elemento es el nuevo tope
- El tope aumenta

□ Pop

- Remueve el elemento tope de la pila, y lo devuelve.
- Recuerde, el tope cambia
- Si la pila esta vacía, no se puede sacar nada
 - Error, y devuelve valor invalido

Clase Stack – Impl. estática

Constructors

Constructor and Description

`Stack()`

Creates an empty Stack.

Methods

Modifier and Type

Method and Description

boolean

`empty()`

Tests if this stack is empty.

E

`peek()`

Looks at the object at the top of this stack without removing it from the stack.

E

`pop()`

Removes the object at the top of this stack and returns that object as the value of this function.

E

`push(E item)`

Pushes an item onto the top of this stack.

int

`search(Object o)`

Returns the 1-based position where an object is on this stack.

Java Interface Deque

`java.util`

Interface Deque<E>

Type Parameters:

`E` - the type of elements held in this collection

All Superinterfaces:

`Collection<E>`, `Iterable<E>`, `Queue<E>`

All Known Subinterfaces:

`BlockingDeque<E>`

All Known Implementing Classes:

`ArrayDeque`, `ConcurrentLinkedDeque`, `LinkedBlockingDeque`, `LinkedList`

Java Interface Deque

java.util

Interface Deque<E>

Type Parameters:

E - the type of elements held in this collection

All Superinterfaces:

Collection<E>, Iterable<E>, Queue<E>

All Known Subinterfaces:

BlockingDeque<E>

All Known Implementing Classes:

ArrayDeque, ConcurrentLinkedDeque, LinkedBlockingDeque, LinkedList


```
Deque<Integer> s1 = new ArrayDeque<>();  
    s1.push(1);  
    s1.push(2);  
    s1.push(3);
```


```
Deque<Integer> s2 = new LinkedList<>();  
    s1.push(4);  
    s1.push(5);  
    s1.push(6);
```

Estática

```
Deque<Integer> s1 = new ArrayDeque<>();  
    s1.push(1);  
    s1.push(2);  
    s1.push(3);
```

Dinámica




```
Deque<Integer> s2 = new LinkedList<>();  
    s1.push(4);  
    s1.push(5);  
    s1.push(6);
```



[ArrayDeque](#) and [LinkedList](#) are implementing [Deque](#) interface but implementation is different.

3

Key differences:

- 
- 
- 
1. The *ArrayDeque* class is the resizable array implementation of the *Deque* interface and *LinkedList* class is the list implementation
 2. NULL elements can be added to *LinkedList* but not in *ArrayDeque*
 3. *ArrayDeque* is more efficient than the *LinkedList* for add and remove operation at both ends and *LinkedList* implementation is efficient for removing the current element during the iteration
 4. The *LinkedList* implementation consumes more memory than the *ArrayDeque*

So if you don't have to support NULL elements && looking for less memory && efficiency of add/remove elements at both ends, *ArrayDeque* is the best

Refer to [documentation](#) for more details.

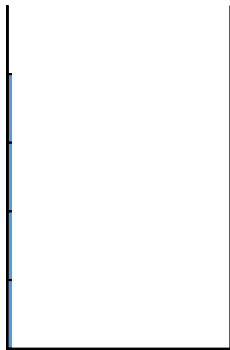
EJERCICIO EN CLASE

- Las pilas se usan para
 - Recuperar un conjunto de elementos en **orden inverso** a como se introdujeron
- Un programa debe
 - Leer una secuencia de elementos enteros por teclado
- Ejemplo: Luego mostrarlos en orden inverso al ingresado
 - Si se ingresa: 1, 3, 5, 7
 - Se mostrará: 7, 5, 3, 1

ANALISIS

- Cada elemento ingresado puede ser “metido” en la pila

- Ejemplo:



7 5 3 1

- Una vez llenada la pila,
 - Solo hay que “sacar”, elemento tras elemento
 - Hasta que la pila quede vacía

SOLUCION?

```
Begin{  
    Stack s;  
    Object dato;  
    s = new Stack( );  
    while( true ){  
        write("Ingrese elemento:");  
        read( dato )  
        if(dato == centinela) break;  
        s.push( dato );  
    }  
    while(!s.empty( )){  
        write(s.pop( ));  
    }  
}
```

Ejemplo: Pila de llamadas del sistema

De AR o LL a Pila

- Pila extends LL
 - isEmpty() se hereda de LL
 - push(e) = addLast(e)
 - pop() = removeLast(e)
 - peek() = getLast(e)