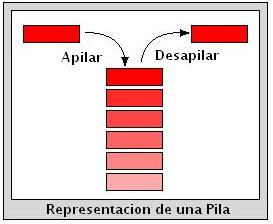
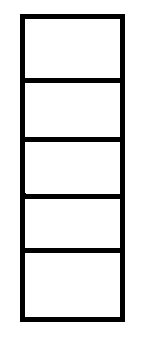
**PILAS**

Una **pila** (*stack* en [inglés](http://es.wikipedia.org/wiki/Idioma_ingl%C3%A9s)) es una lista ordinal o [estructura de datos](http://es.wikipedia.org/wiki/Estructura_de_datos) en la que el modo de acceso a sus elementos es de tipo [LIFO](http://es.wikipedia.org/wiki/LIFO) (del inglés *Last In First Out*, **último en *entrar*, primero en *salir***) que permite almacenar y recuperar datos. Se aplica en multitud de ocasiones en [informática](http://es.wikipedia.org/wiki/Inform%C3%A1tica) debido a su simplicidad y ordenación implícita en la propia estructura.

Para el manejo de los datos se cuenta con dos operaciones básicas: **apilar** (*push*), que coloca un objeto en la pila, y su operación inversa, **retirar** (o desapilar, *pop*), que retira el último elemento apilado.

En cada momento sólo se tiene acceso a la parte superior de la pila, es decir, al último objeto apilado (denominado **TOS**, *Top of Stack* en inglés). La operación **retirar** permite la obtención de este elemento, que es retirado de la pila permitiendo el acceso al siguiente (apilado con anterioridad), que pasa a ser el nuevo TOS.

Por analogía con objetos cotidianos, una operación **apilar** equivaldría a colocar un plato sobre una pila de platos, y una operación **retirar** a retirarlo.

Las **pilas** suelen emplearse en los siguientes contextos:

* Evaluación de expresiones en [notación postfija](http://es.wikipedia.org/wiki/Notaci%C3%B3n_postfija) ([notación polaca inversa](http://es.wikipedia.org/wiki/Notaci%C3%B3n_polaca_inversa)).
* Reconocedores sintácticos de [lenguajes independientes del contexto](http://es.wikipedia.org/wiki/Lenguaje_libre_de_contexto)
* Implementación de [recursividad](http://es.wikipedia.org/wiki/Recursividad).
* Llamadas a subprogramas

**Pila de llamadas**

La **pila de llamadas** es un segmento de [memoria](http://es.wikipedia.org/wiki/Memoria_de_ordenador) que utiliza esta estructura de datos para almacenar información sobre las llamadas a [subrutinas](http://es.wikipedia.org/wiki/Subrutina) actualmente en ejecución en un [programa](http://es.wikipedia.org/wiki/Programa_inform%C3%A1tico) en [proceso](http://es.wikipedia.org/wiki/Proceso_%28inform%C3%A1tica%29).

Cada vez que una nueva subrutina es llamada, se apila una nueva entrada con información sobre ésta tal como sus variables locales. En especial, se almacena aquí el punto de retorno al que regresar cuando esta subrutina termine (para volver a la subrutina anterior y continuar su ejecución después de esta llamada)..

**Pila como tipo abstracto de datos**

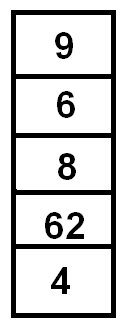
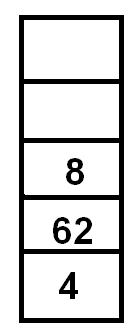
A modo de resumen tipo de datos, la pila es un contenedor de nodos y tiene dos operaciones básicas: **push** (o apilar) y **pop** (o desapilar). 'Push' añade un nodo a la parte superior de la pila, dejando por debajo el resto de los nodos. 'Pop' elimina y devuelve el actual nodo superior de la pila. Una metáfora que se utiliza con frecuencia es la idea de una pila de platos en una cafetería con muelle de pila. En esa serie, sólo la primera placa es visible y accesible para el usuario, todas las demás placas permanecen ocultas. Como se añaden las nuevas placas, cada nueva placa se convierte en la parte superior de la pila, escondidos debajo de cada plato, empujando a la pila de placas. A medida que la placa superior se elimina de la pila, la segunda placa se convierte en la parte superior de la pila. Dos principios importantes son ilustrados por esta metáfora: En primer lugar la última salida es un principio, la segunda es que el contenido de la pila está oculto. Sólo la placa de la parte superior es visible, por lo que para ver lo que hay en la tercera placa, el primer y segundo platos tendrán que ser retirados.

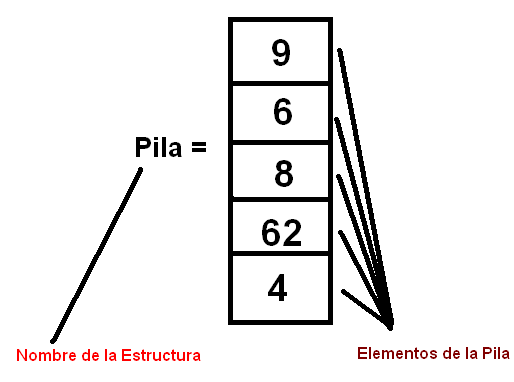
**Operaciones**

Una pila cuenta con 2 operaciones imprescindibles: apilar y desapilar, a las que en las implementaciones modernas de las pilas se suelen añadir más de uso habitual.

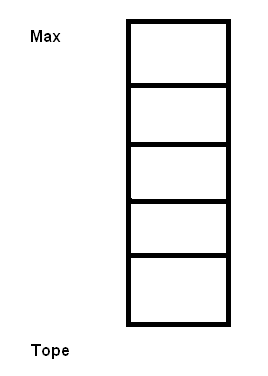
* **Crear:** se crea la pila vacía.
* **Apilar:** se añade un elemento a la pila.(push)
* **Desapilar:** se elimina el elemento frontal de la pila.(pop)
* **Cima:** devuelve el elemento que está en la cima de la pila. (top o peek)
* **Vacía:** devuelve cierto si la pila está vacía o falso en caso contrario.

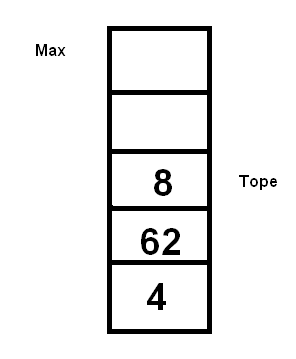
**Indexación base-cero (0):** En este modo el primer elemento del vector será la componente cero ('0') del mismo, es decir, tendrá el índice '0'. En consecuencia, si el vector tiene 'n' componentes la última tendrá como índice el valor 'n-1'. Java y el [lenguaje C](http://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n_C) es un ejemplo típico que utiliza este modo de indexación.

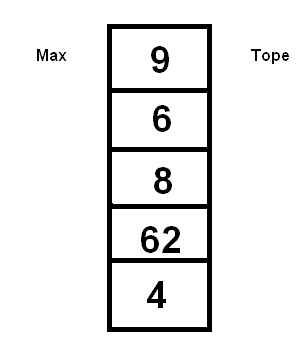




Nota: No confundir los elementos de la Pila con la posición o índice de la Misma







Para acceder a los elementos de la estructura es necesario hacer referencia al nombre e indicar la posición.

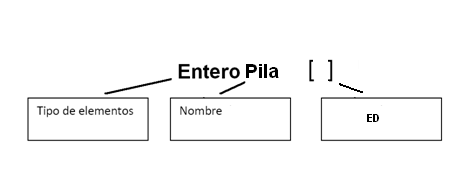
Pila [Tope]= 9

Nota: En adelante se hace referencia a las estructuras creadas con base a la utilización de clases, métodos y el nivel de accesibilidad de los mismos.

**I**

**DECLARACION DE LA ESTRUCTURA**

**1.1CREACION DE LA PILA**



**1.2 Definir las variables que manipularán la estructura.**

En esta parte del diseño es necesario definir las variables índices, variables de cálculos y operaciones que regirán los métodos con ámbito global y el nivel de visibilidad. Para darle una correcta documentación al diseño es recomendable comentar el uso que tendrá dicha variable en la estructura que se está diseñando

publico Entero Pila []; // Estructura de la pila

publico Entero Top, Max , Elem, pos; // variables para la pila

//Top : El Tope de la pila , referencia al último elemento que está en la pila

//Max : Máximo de espacios que tiene la pila

//Elem : Elemento que se agrega a la pila (Tecleado por el usuario)

**II**

**INICIALIZACIÓN DE LA ESTRUCTURA**

**Elaboración de un método constructor para inicializar la estructura.**

2.1 Se toman como referencia para el tamaño de la pila el parámetro contenido en las variables **Espacios**, el dato de este parámetro es obtenido a través de otra clase Que llamaremos MenuPpal donde el usuario determina el tamaño de la Pila.

2.2 Se indica nivel de acceso público a la estructura creada.

2.3 Creación de la Pila se representa como estructura de datos definida por el usuario

* 1. Se inicializan los valores de las variables de la estructura.

**//----- Contructor -------\\**

publico Pila (entero Espacios) // se recibe como parámetro los Espacios de la pila

Inicio

Pila = new entero [Espacios]; // Asignamos los espacios en la Pila

Max = Espacios - 1; // Se determina la capacidad máxima de almacenamiento

Top = -1; // Top se declara como -1 como referencia a que no existen datos en la pila

Fin

**//------- Fin del constructor ------\\**

**III**

**Métodos**

Los métodos son subrutinas que gobernaran a la estructura de datos creada, los métodos permitirán definir el comportamiento de la estructura durante la ejecución del programa.

Las operaciones cotidianas que realizan las pilas son las siguientes

* Inserción ( Push)
* Eliminación (Pop)
* Búsqueda
* Visualización

Inicialmente definiremos los métodos que indicarán el estado de la estructura en este caso nos interesa saber cuando la pila está llena y cuando esta vacía para lo cual realizaremos métodos booleanos que determinar si es falso o verdadero el estado consultado.

**3.1 Método para determinar si una pila esta vacía.**

Publico booleano PilaVacia()

Inicio

Si (Top==-1)

Inicio

retorne Verdadero;

sino

retorne falso;

Fin Si

Fin Metodo

**Explicación:** La variable Top es el índice de la pila y se está utilizando el sistema de indexación en base a 0, por lo cual si el valor del Top es de -1 indicaría que la estructura esta vacía.

**3.2 Método para determinar si una Pila está Llena.**

Publico booleano PilaLLena()

Inicio

Si (Top==Max)

Inicio

retorne Verdadero;

sino

returne falso;

Fin Si

Fin Metodo

**Explicación:** La variable Max está almacenado la capacidad máxima de almacenamiento de la Pila, por lo tanto se comparan el índice en este caso la variable Top con la capacidad en caso tal que sean iguales nos indicaría que la estructura está Llena.

**3.3 Método para insertar un elemento a la Pila.**

Publico Push()

Inicio

Si(!pilaLlena()) // Si la pila No esta llena .... Se puede agregar un nuevo dato a la pila

Elem = Leer(" Digite numero ");

Top = Top+1;; // Se incrementa Top como referencia de que se agrego un nuevo dato

Pila[Top] = Elem; // Se agrega el nuevo elemento a la pila

si (pilaLlena()) // Si la pila quedo llena entoces ... Imprimir mensaje

Escriba (" La Pila a quedado llena !! ");

Fin si

sino

Escriba(" Pila llena !! Imposible introducir un valor a la pila");

Fin Si

Fin Metodo

**Explicación:** Se verifica si se pueden insertar elementos invocando la negación del método pila llena, si es posible se incrementa el tope y se inserta el elemento capturado en la pila.

**3.4 Método para Eliminar un elemento de la Pila.**

Publico Pop ()

Inicio

Si(!pilaVacia()) // Si la pila no esta vacia entonces ....

Escriba("Se borro el Dato : " + Pila[Top]); // Eliminar dato

Pila[Top] = 0; // remplaza el valor por un cero (Eliminado)

Top= Top-1; // Top se reduce para decir que un elemento se borro

sino

// De lo contrario imprime .."que la pila esta vacia"

Escriba ("Pila Vacia... Imposible Eliminar");

Fin Si

Fin Metodo

**Explicación:** Se verifica si se pueden la pila esta vacía invocando la negación del método pila Vacía, si hay elementos se asigna cero o null según sea el caso para indicar la eliminación del elemento y se decrementa el tope liberando una posición dentro de la pila.