

# Listas !!

Los objetos pueden contener una referencia a otros objetos en el mismo tipo.

LISTAS SIMPLEMENTE ENLAZADA



## [LA CLASE NODO]

public class Nodo {

int Carga;

Nodo prox;

public Nodo() {

Carga = 0;

prox = null;

public Nodo (int Carga, Nodo prox) {

this.Carga = Carga;

this.prox = prox;

public String toString () {

return Carga + ", "

metodos

tilibra

↳ convierte un entero a string

main

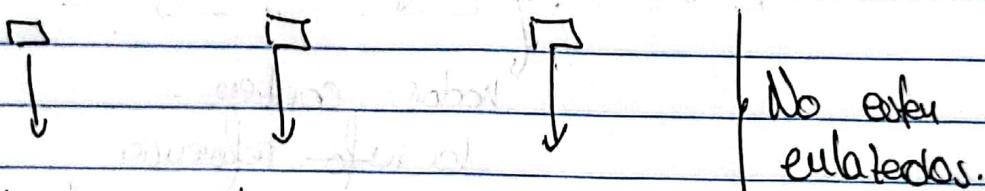
Nodo nodo = new Nodo(1, null);

System.out.print(nodo);

Nodo nodo1 = new Nodo(1, null)

Nodo nodo2 = new Nodo(2, null)

Nodo nodo3 = new Nodo(3, null)



Nodo1 Nodo2 Nodo3

1 2 3

Para enlazarlos

Nodo1.prox = nodo2

Nodo2.prox = nodo3

Nodo3.prox = null

Nodo1 → Nodo2 → Nodo3

LISTA

Son objetos que  
son una serie  
de enlaces

entre ellos  
objetos  
en una sola  
estructura

Osi que si queremos  
poner la lista como  
parámetro en algún  
metodo solo tiene  
referencia al  
primer nodo.

El primer nodo  
tiene que apuntar  
a la lista,

externa.



Tipo: Public static void imprimir lista (Nodo lista);

Nodo nodo = lista;

while (nodo != null) {

System.out.print (nodo);

nodo = nodo.prox;

Llamar el metodo

imprimir lista (lista)

listas → recursión

Caso base y dejar

Imprimir reverso de una lista

public static void (Nodo lista);

if (lista == null) return;

Nodo cabecera = lista;

Nodo colar = lista.prox;

imprimir\_reverso (cola);

System.out.print (cabecera);

# Pilas:

TAD Tipo abstracto de datos  
especifica un conjunto de operaciones o métodos, los devueltan o los operaciones.

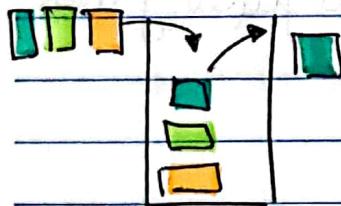
↳ Colección.

↳ Simplifica la tarea.

↳ estructura de datos que contiene múltiples elementos.

Operaciones:

- ⇒ Constructor (crea una pila vacía).
- ⇒ Apilar: Agregar un elemento al final.
- ⇒ Desapilar: Quitar el último que fue agregado.
- ⇒ Estado vacío: responder si es o no vacía.



Objeto Stack en Java

Stack → implementa el TAD pila.

Estructura LIFO

"last in, first out"

Apilar ⇒ Push

Desapilar ⇒ Pop

Estado vacío ⇒ isEmpty

↳ se importa desde java.util.

Stack pila = new Stack(); inicialmente la pila está vacía.

Podemos agregar cualquier tipo de item a ella.  
En java → objetos.

Libreria lista = new LinkedList();  
lista.agregar adelante (3);  
lista.agregar " " (1);  
lista.introducir ();

libreria

for (Nodo nodo = lista.Cabeza; nodo != null; nodo = nodo.siguiente) {  
pila.push (nodo);}

↓ 1, 2, 3

Borrar  $\Rightarrow$  Object obj = pila.pop();

→ xq' no  
dame que tipo  
de objeto contiene.



Nodo nodo = (Nodo) obj; (convierte q  
nodo de  
mejor.)

Borrar pila  $\Rightarrow$  desapilar todo

while (!pila.isEmpty()) {

Nodo nodo = (Nodo) pila.pop();

System.out.println(nodo + " ");

1321

### Implementación del TAD pila usando arreglo.

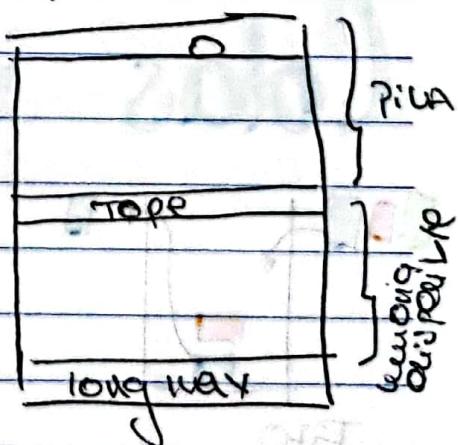
Arreglo de objetos: que contiene los elementos  
que lleva una cuenta de  
cuál es el próximo espacio  
disponible.

public Stack() {

this.array = new Object[128];

this.index = 0;

PILA



public void push (Object elem)

array[index] = elem;

index++

public Object pop ()

index--

return array[index].

tilibra

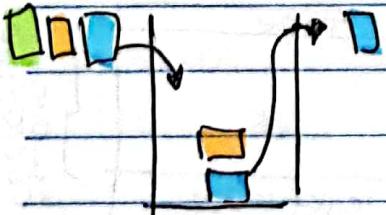
Permiten insertar en el arreglo. si que para si  
hay mas items que 128.

```
public void push (Object item) {  
    if (ultimo ()) reaumentar ();  
    indice < arreglo.length,  
    arreglo [indice] = item;
```

```
private boolean ultimo ()  
return indice = arreglo.length.
```

Private void reaumentar () {  
 ↓  
 Object [] nuevoArreglo = new Object [arreglo.length \* 2]  
 for (int i=0 ; i < arreglo.length ; i++) {  
 nuevoArreglo [i] = arreglo [i]  
 }  
 arreglo = nuevoArreglo.

## Colas y Colas Prioridad



FIFO

Encolado con prioridad  
↳ a cada cliente se le  
asigna una prioridad. El  
cliente con la prioridad +  
alta va primero. Es importante  
cuando llega.

El TAD Cola tiene las siguientes operaciones para la pila. java.util.LinkedList (vt).

public class Cola {

    private LinkedList<List> lista;

    public Cola () {

        lista = new LinkedList<List>();

    public boolean esVacia() {

        return lista.isEmpty();

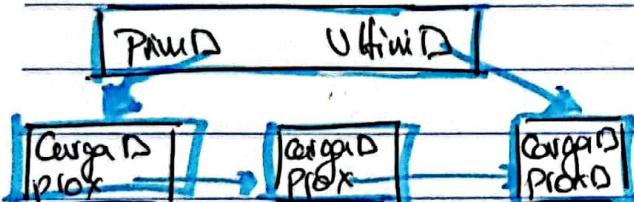
    public void agregar (Object obj) {

        lista.addLast (obj);

    public Object quitar () {

        return lista.removeFirst();

**Cola enlazada.** Similar a la lista enlazada pero tiene referencias del primero y ultimo.



public class Cola {

    public Nodo primero, ultimo;

    public void agregar (Object obj) {

        Nodo nodo = new Nodo (obj, null);

        if (ultimo != null) {

            ultimo.siguiente = nodo;

            ultimo = nodo;

        if (primero == null) {

            primero = ultimo;

        }

    }

# Unidad Temática 2. Ordenes.

→ Elegir el algoritmo a usar siguiendo los siguientes objetivos

→ ALGORITMO FÁCIL DE ENTENDER, CODIFICAR Y DEPURAR.

→ USO EFICIENTE DE LOS RECURSOS Y EFICIENCIA VELOC.

FACTORES QUE AFECTAN

AL TIEMPO DE EJECUCIÓN

→ Tamaño de entrada al programa.

→ Calidad del código generado

→ La complejidad del tiempo

→ el compilador.

del algoritmo tiene que depender de la velocidad, rapidez de la memoria y velocidad de las instrucciones.

Tiempo de ejecución entrada → El tiempo de ejecución depende de la entrada significa que debe definirse como una función de la entrada (o tamaño)

$T(n)$  → representa el tiempo de ejecución de programas con entrada tamaño  $n$ .

→ indica el número de instrucciones ejecutadas por la computadora.

→ Definiremos como el tiempo de ejecución en el peor caso esto es el último entre las entradas de tamaño  $n$  del tiempo de ejecución del programa.



Se puede calcular también el  $T_{\text{prom}}(n)$ , que es el promedio entre los entradas de acuerdo al tiempo de ejecución en esa entrada.

En vez de expresar el tiempo como segundos se expresa:

"El tiempo de ejecución de tal programa es proporcional a  $n^2$ ".

**Notación**  $\mathcal{O}$  → de orden  $n$ .

$T(n) = \mathcal{O}(f(n))$  si existen constantes positivas  $C$  y  $n_0$  tal que

$T(n) \leq C f(n)$  para todo  $n \geq n_0$

**Algoritmo**

→ ¿cuanto demora en el tiempo mientras cambia  
b) datos de entrada?

= costo del tiempo  
en el "orden"  
clase de  
notación  
asintótica.

$2^n$   
 $n^2$   
 $n$   
 $\log(n)$   
 $1$

mayor

menor

La complejidad temporal se puede definir de la siguiente manera:

$$T(n) = O(g(n))$$

$\Leftrightarrow$  Si y solo si

$$C \times g(n) \geq T(n)$$

$C > 0, n \geq n_0$

Existe  $c$  constante +

que multiplicada por el

crecimiento es mayor o

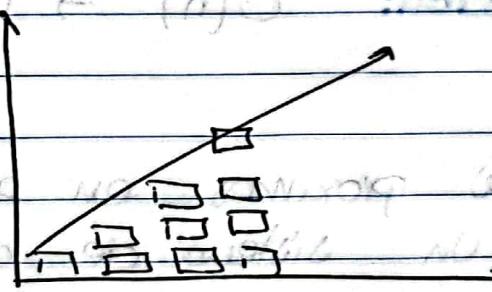
igual a  $T(n)$

para todo  $n \geq n_0$ .

$$\text{for} = O(n)$$

$$\text{for} = O(n^2)$$

$$O(n)$$



$$\text{if } n > 0 \text{ then } \rightarrow S(n-1)$$

Algunas operaciones que no se consideran en la complejidad:

(a)  $O(1)$

## Ejemplos de tiempos de ejecución de diversos algoritmos

- ① Encuentra mínimo de una matriz

Solución: ① Partir de una variable min que almacena el elemento mínimo.

- ② Inicializar min con el valor del primer elemento
- ③ Hacer un bucle iterativo de la matriz, actualizar el mínimo.

Tiempo ejecución:  $O(n)$  → lineal.

- ② Puntos más próximos en el plano. (dados los puntos en un sistema de coordenadas  $x-y$  encontrar la pareja de puntos + próximos)

Solución: ① Calcular la distancia entre cada pareja de puntos

- ② Quedarse con la distancia mínima

$$N(n-1)/2 \leftarrow \boxed{\text{Cálculo cartesiano}}$$

Hay  $n^2$  parejas de puntos.

Hacer todo esto requiere un tiempo cuadrático.

Algoritmos mejores  $\rightarrow O(n \log n)$



③ Puntos colineales en el plano (determinar si).  
3 puntos forman una linea recta.

$$\text{MAN CARA COMPUTACIONAL} \rightarrow n(n-1)(n-2)/6$$

(Computacional)

Punto de menor distancia entre los vértices

metodo Punto de menor distancia entre los vértices

ordenando los vértices de menor a mayor

(ordenar los vértices a menor distancia)

comparar con los demás vértices

RECUSIVIDAD → cuando un método se llama a si mismo.  
(CASO BASE)

¿CUANDO UN OBJETO Q → en parte está formado  
POR RECURSOS? X SI MISMO  
↳ está definido en función  
de SI MISMO.

Tener un uso SI existe una solución  
No recursiva aceptable.

Tipos Direcamente recursivo. → Modelo P tiene referencia  
explícita a SI MISMO.

Indirectamente recursivo. → Modelo P contiene  
referencia a Q que a su vez referencia a P. ; Cuidadada aplicación!

FORMA USUAL.

Función A ( $N$ : tamaño del problema).  
COM

< bloque >

Si < condición > entonces

< bloque >

A( $N-1$ )

< bloque >

SINO



< bloques >

Fin si

< bloques >

Fin.

Ejemplo sumar de los primeros  $n$  enteros  
en A.

Com

Si  $n=1$

devolver AT0]

Sino.

devolver SumaLineal(A, n-1) + (ATn-1)

Recursión de rob.  $\Rightarrow$  bases o lazo. como  
último paso

4 Reglas Lazo

Algo Difícil

Progres

Puede creerlo

Nunca olvides el trabajo.

