



ESCUELA DE
INGENIERÍA EN CIENCIAS Y SISTEMAS
FACULTAD DE INGENIERÍA
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA



Día, Fecha:

Miércoles, 26 de febrero

Hora de inicio:

4:30 – 6:10 pm

Introducción a la Programación y Computación 1 Sección G

Max Rodrigo Durán Canteo

Clase 6

Manejo de memoria

Agenda

La Memoria

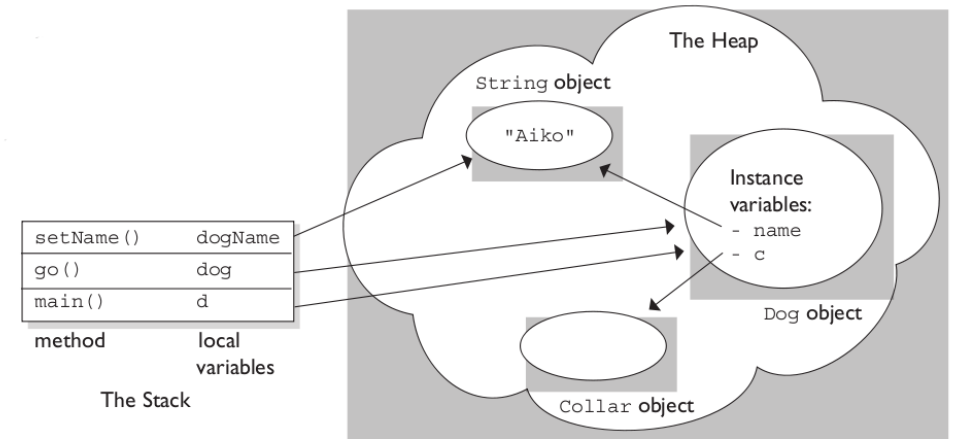
Estructuras de Datos

Arreglos

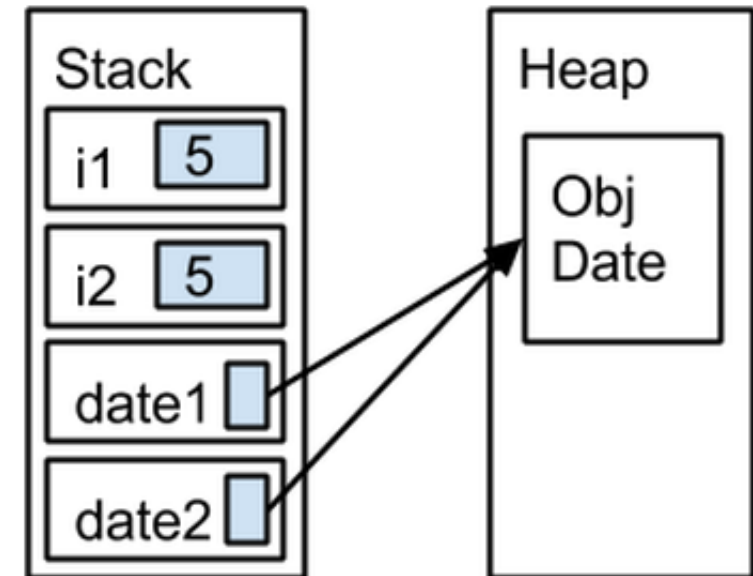
Listas

Ordenamiento

La Memoria

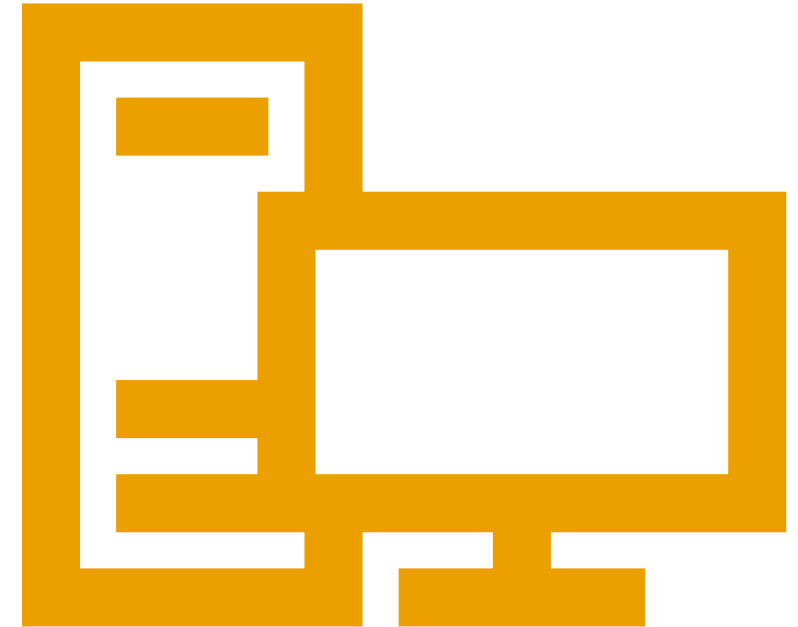


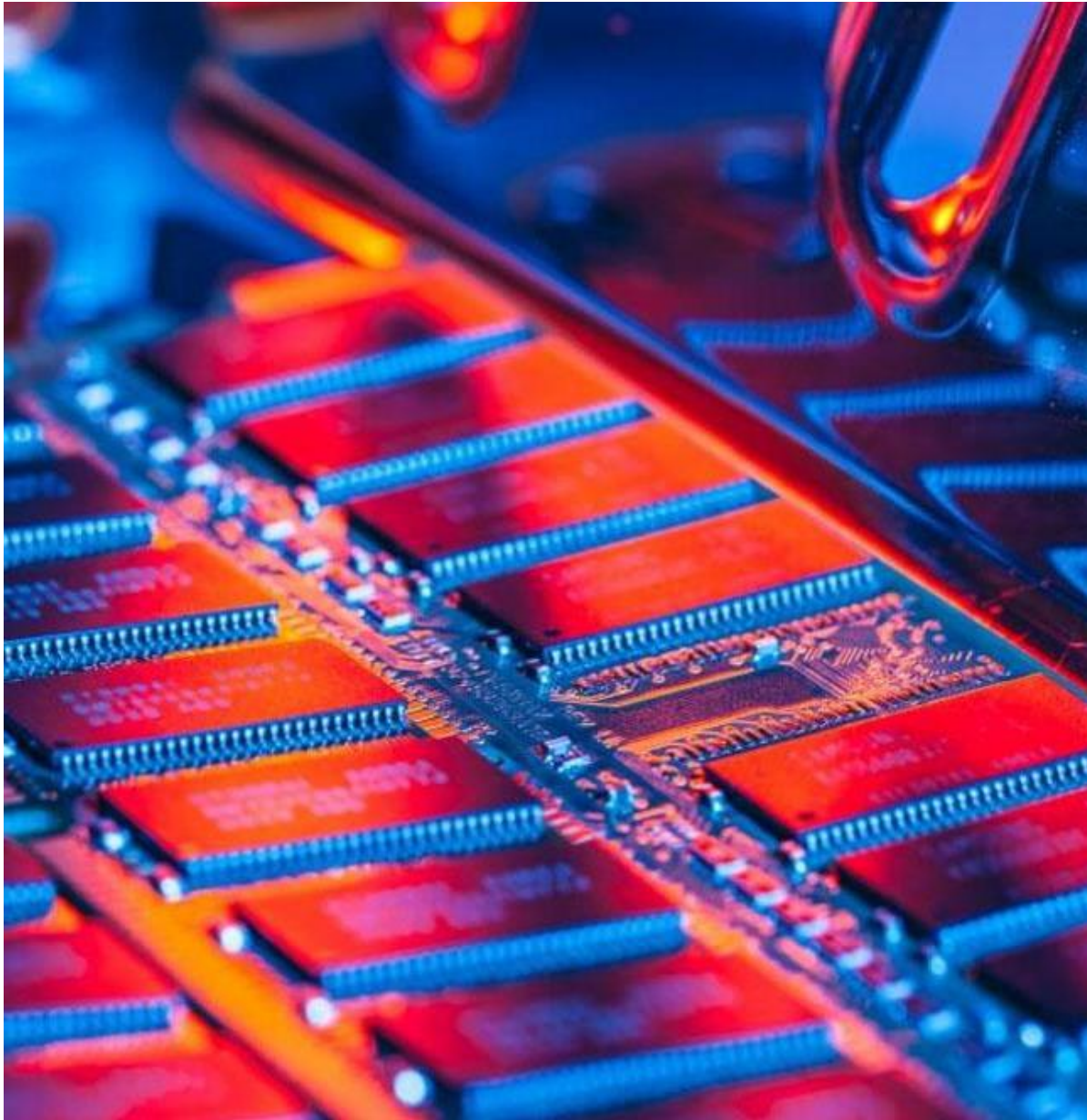
*Definición, Memoria Estática/Dinámica,
Stack, Heap*



Memoria

Es el dispositivo que retiene, memoriza o almacena datos informáticos durante algún periodo de tiempo. La memoria proporciona una de las principales funciones de la computación moderna: el almacenamiento de información y conocimiento. Es uno de los componentes fundamentales de la computadora, que interconectada a la unidad central de procesamiento y los dispositivos de entrada/salida, implementan lo fundamental del modelo de computadora de la arquitectura de von Neumann.





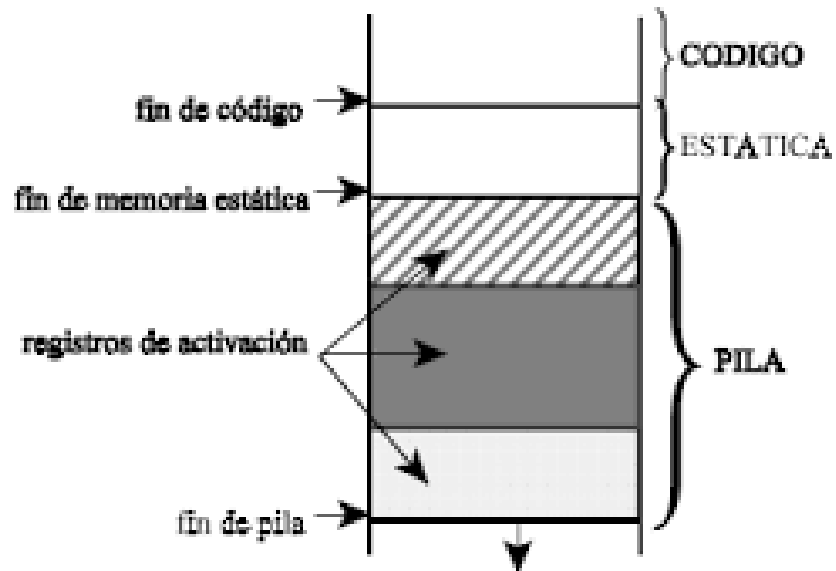
Memoria

La ejecución de un programa requiere que diversos elementos se almacenen en la memoria:

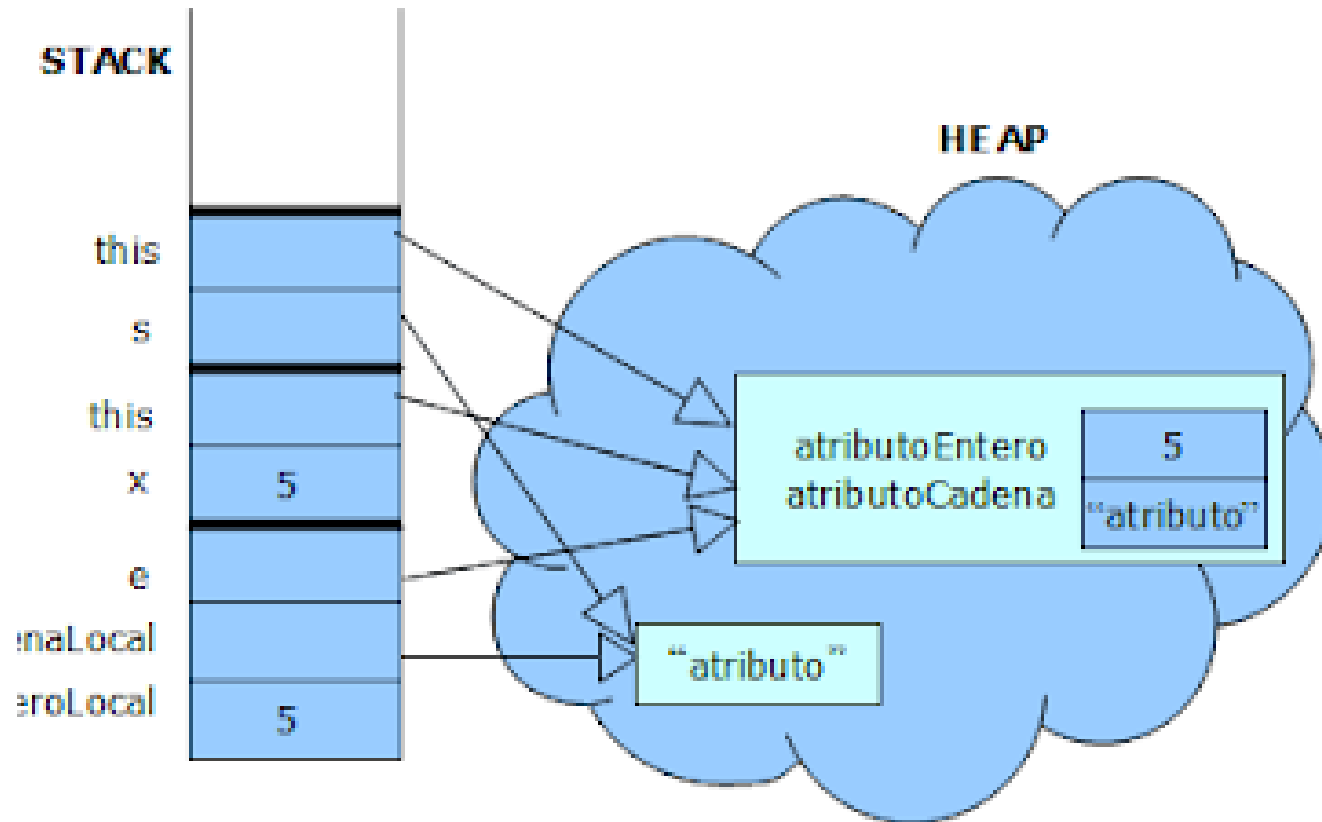
- Código del programa (Instrucciones)
- Datos
 - Permanentes
 - Temporales
- Direcciones para controlar de flujo de ejecución del programa

Memoria Estática

Se refiere a la asignación de memoria en tiempo de compilación o en tiempo de carga del programa. La cantidad de memoria reservada es fija y no cambia durante la ejecución del programa. Ejemplos incluyen variables globales, variables locales estáticas y constantes.



Memoria Dinámica



Se refiere a la asignación de memoria en tiempo de ejecución. La cantidad de memoria puede variar según las necesidades del programa. Esta memoria se gestiona explícitamente mediante funciones como `malloc/free` en C o `new/delete` en C++.

Stack

Es una región de memoria que sigue una estructura LIFO (Last In, First Out). Es utilizada principalmente para:

Almacenar variables locales de funciones.

Mantener información sobre el contexto de ejecución (como direcciones de retorno).

Gestionar llamadas a funciones.

Consideraciones del Stack



La memoria en el stack se asigna y libera automáticamente cuando se entra o sale de una función.



Su acceso es muy rápido.



Es liberada al terminar la ejecución.



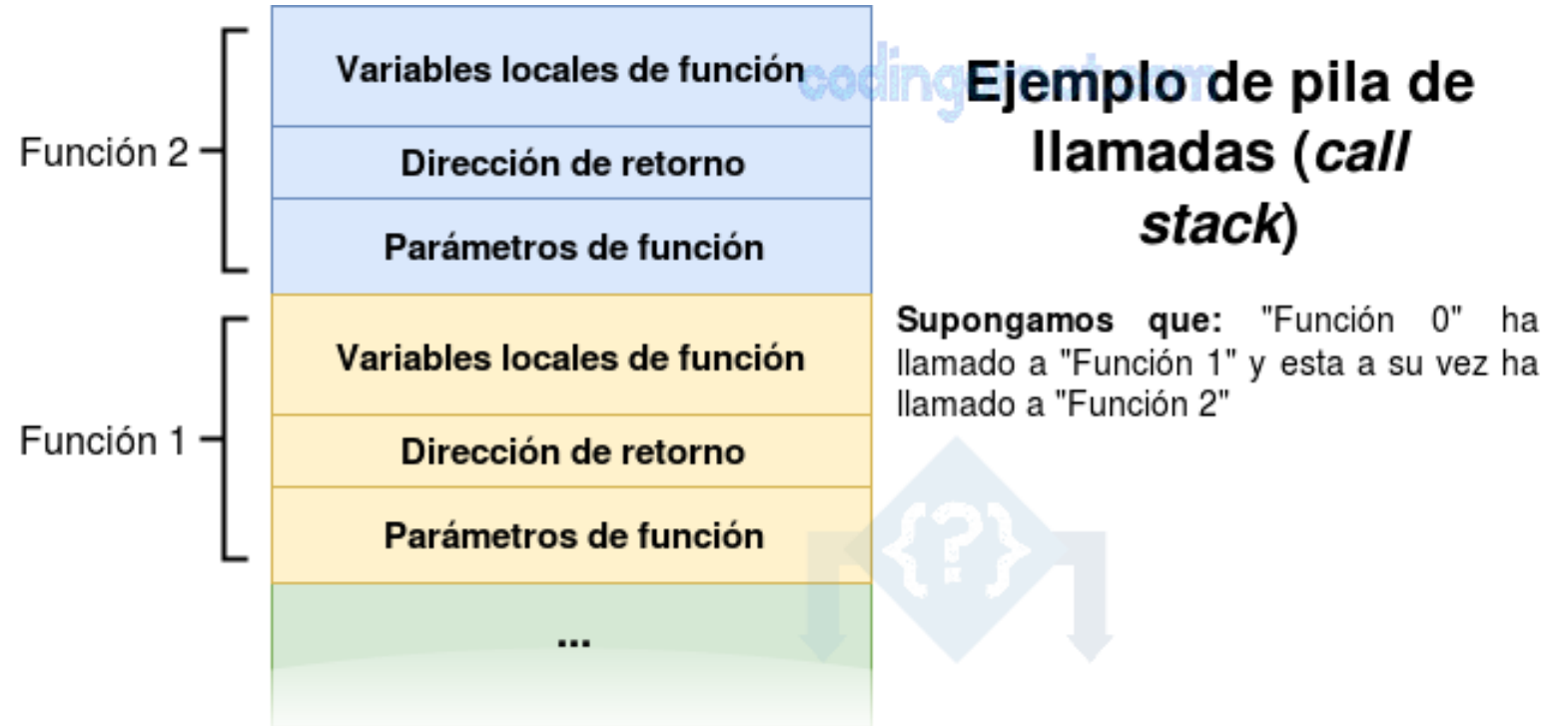
Fragmentos grandes de memoria, como arrays de gran envergadura, no deberían ser almacenados en el Stack, para prevenir desbordamientos del mismo (Stack Overflow).



Las variables almacenadas en el Stack solamente son accesibles desde el bloque de código donde fueron declaradas.

Ejemplo Stack

```
void main(){
    if (true) {
        int x = 0;
    }
    x = 1;
}
```



Heap

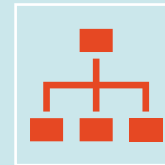
Es una región de memoria más flexible y menos estructurada que el stack. Se utiliza para:

- Asignar memoria dinámicamente durante la ejecución del programa.
- Almacenar datos cuyo tamaño no se conoce en tiempo de compilación o que deben persistir más allá del ámbito de una función.

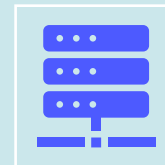
Características del Heap



La memoria en el heap debe ser gestionada manualmente (en lenguajes como C/C++) o automáticamente (en lenguajes con recolección de basura, como Java o Python).



Es más lenta que el stack debido a la complejidad de su gestión.



Tiene un tamaño mucho mayor que el stack, pero también es más propenso a fragmentación.

Stack vs Heap

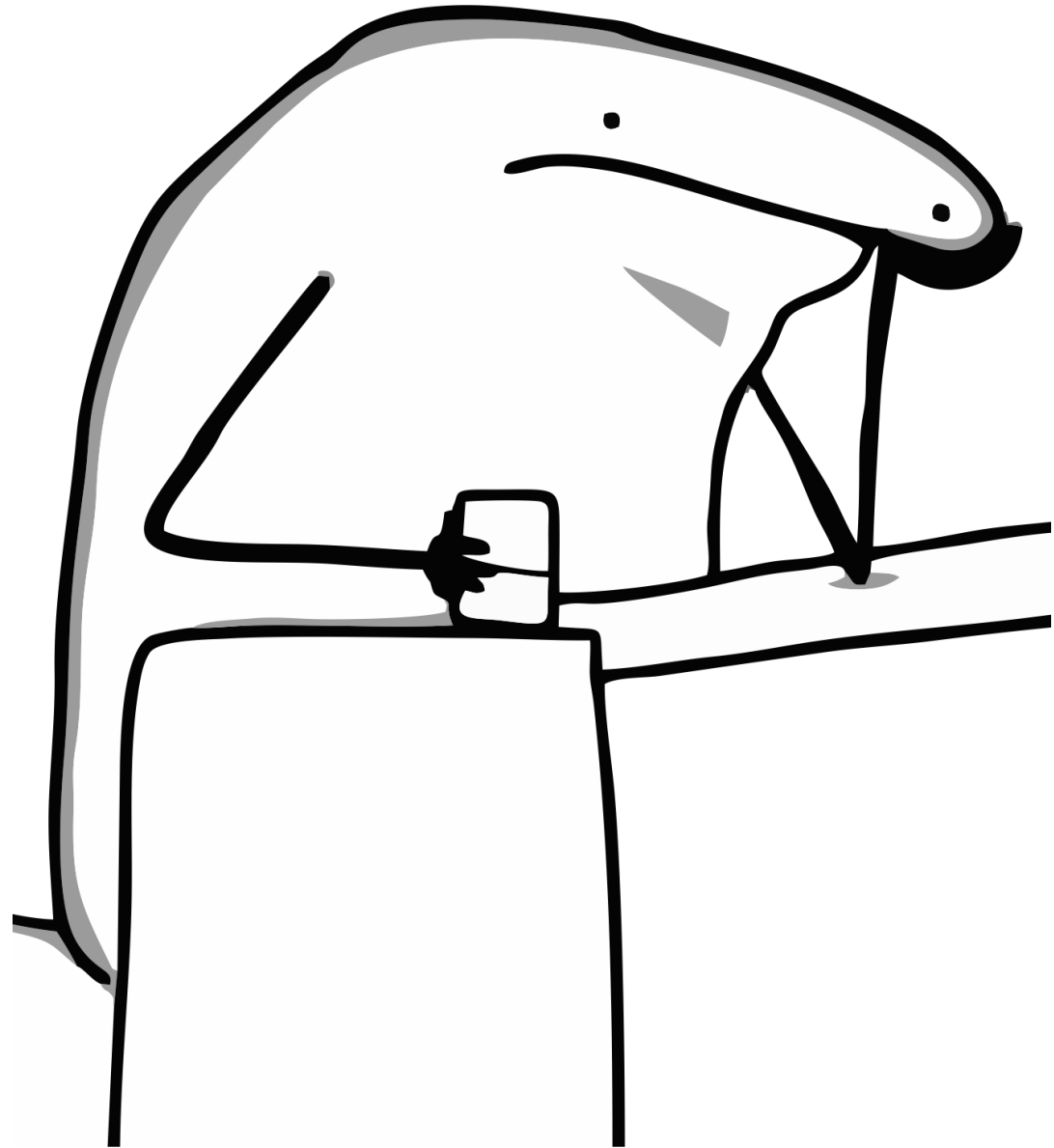
Aspecto	Stack (Pila)	Heap (Espacio Libre)
Organización	Estructura ordenada tipo LIFO (Last In, First Out).	Estructura desordenada; bloques se manejan dinámicamente.
Almacenamiento	Variables locales, parámetros de funciones, y datos temporales.	Objetos, variables globales, estáticas, y memoria dinámica.
Reserva de Memoria	Automática durante la ejecución de funciones.	Manual mediante operaciones como malloc o new.
Tamaño	Limitado y definido en tiempo de compilación.	Mayor y definido en tiempo de ejecución.
Velocidad	Más rápido debido a su organización simple.	Más lento por la necesidad de manejar fragmentación.
Acceso	Directo y rápido.	Más complejo, requiere punteros para referencia.
Manejo	Administrado automáticamente por el sistema.	Requiere gestión explícita para asignar y liberar memoria.
Errores Comunes	Desbordamiento de pila (stack overflow).	Fugas de memoria (memory leaks).

Relación entre memoria estática/dinámica y stack/heap

Concepto	STACK	HEAP
Memoria Estática	Variables locales y parámetros de funciones (asignación fija en tiempo de compilación).	Variables globales y estáticas (no estrictamente en el heap, pero gestionadas de forma estática).
Memoria Dinámica	No aplica (el stack no maneja memoria dinámica).	Asignación explícita en tiempo de ejecución mediante funciones como malloc o new .

Estructuras de Datos

*¿Cómo guardar la
información?*



Definiciones

Estructura

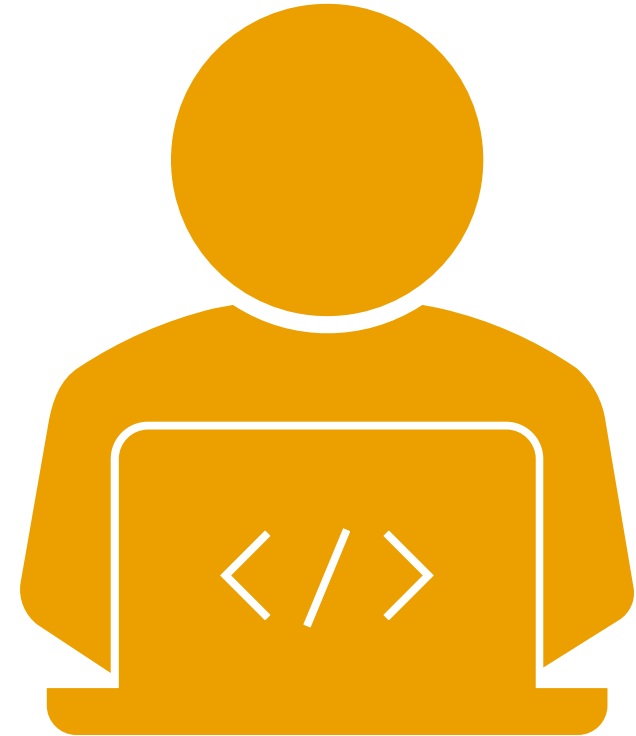
La estructura es el conjunto de elementos que caracterizan un determinado ámbito de la realidad o sistema. Los elementos estructurales son permanentes y básicos, no son sujetos a consideraciones circunstanciales ni coyunturales, sino que son la esencia y la razón de ser del mismo sistema.

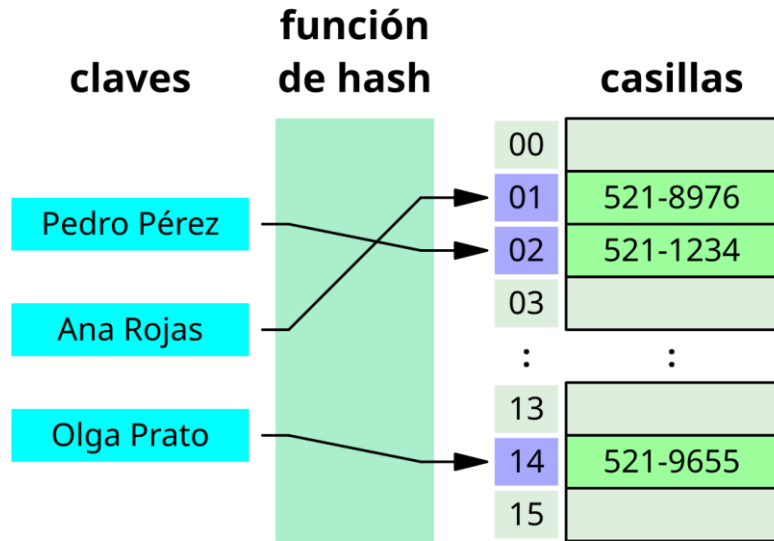
Datos

Un dato es una representación simbólica (numérica, alfabética, algorítmica, espacial, etc.) de un atributo o variable cuantitativa o cualitativa. Los datos describen hechos empíricos, sucesos y entidades. Es un valor o referente que recibe el computador por diferentes medios. Los datos representan la información que el programador manipula en la construcción de una solución o en el desarrollo de un algoritmo.

Estructura de Datos

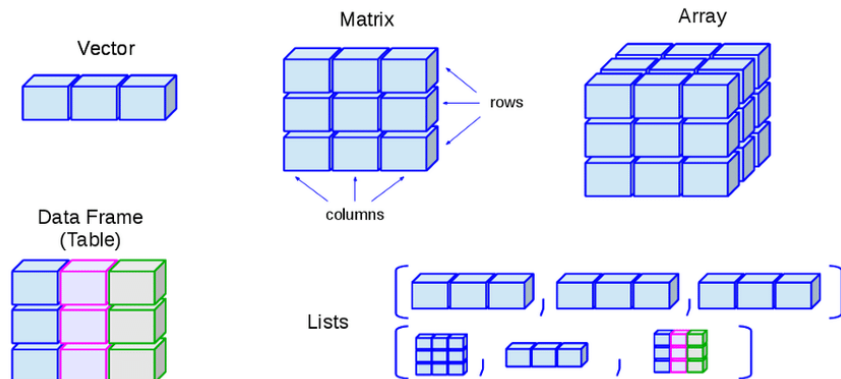
En ciencias de la computación, una estructura de datos es una forma particular de organizar datos en una computadora para que puedan ser utilizados de manera eficiente. Diferentes tipos de estructuras de datos son adecuados para diferentes tipos de aplicaciones, y algunos son altamente especializados para tareas específicas.





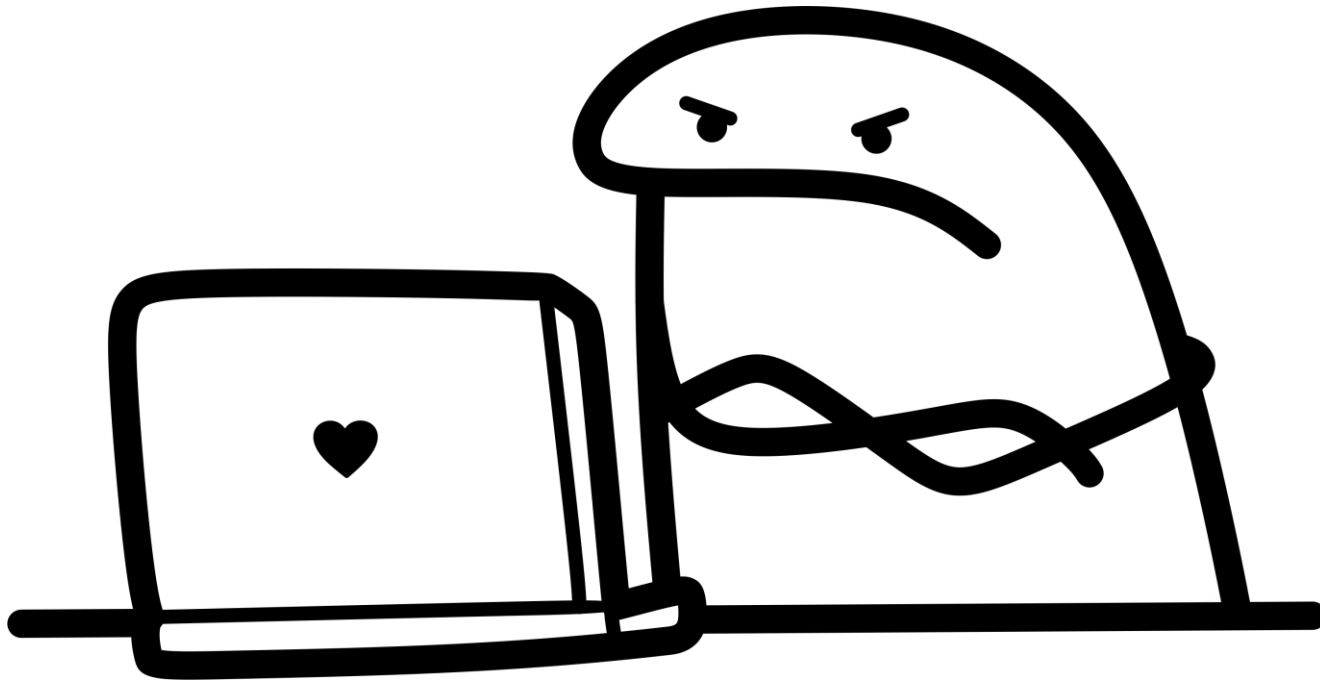
Ejemplos de Estructuras de Datos

- Vector
- Arreglo
- Case
- Árbol
- Estructura (struct)
- Lista
- Cadena



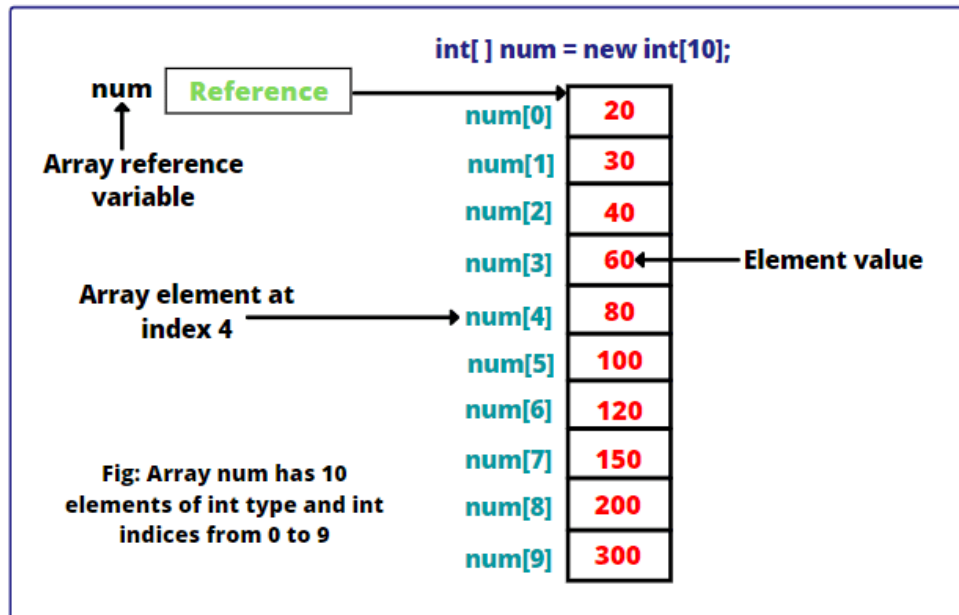
Arreglos

- Definición
- Sintaxis
- Valores Default
- Tips
- Métodos



Arreglos

Un array (arreglo) en Java es una estructura de datos que nos permite almacenar un conjunto de datos de un mismo tipo. El tamaño de los arrays se declara en un primer momento y no puede cambiar luego durante la ejecución del programa, como sí puede hacerse en otros lenguajes.




Sintaxis

<Tipo de dato> "[" <ID> = new <Tipo de dato> "[" "<Longitud>"]"


Ejemplos



```
int[] numeros = new int[5];
```



```
int[][] matriz = new int[3][3];
```



```
string[] nombres = new string[12];
```

Valores por defecto en arreglos

Tipo de Dato

Números

- Valor por defecto: 0

Cadenas y Letras

- Valor por defecto: ""

Booleanos

- Valor por defecto: false
-

Tips para conocer Arreglos

Indices

El índice comienza desde $n = 0$ y termina con el tamaño $n - 1$.

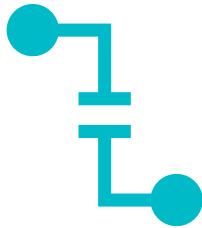
Verlo (Imprimirlo)

Utilizar ciclo for para recorrer el arreglo e imprimirlo.

Acceso

`Nombre_arreglo[n]`

Métodos para Arreglos



Split

Divide una cadena en subcadenas utilizando un delimitador definido mediante una expresión regular o caracter.



toCharArray

Convierte una cadena en un array de caracteres.

split vs toCharArray



split

```
String saludo = "hola mundo";  
String[] dividir = saludo.split(' ');  
  
> ["hola", "mundo"]
```

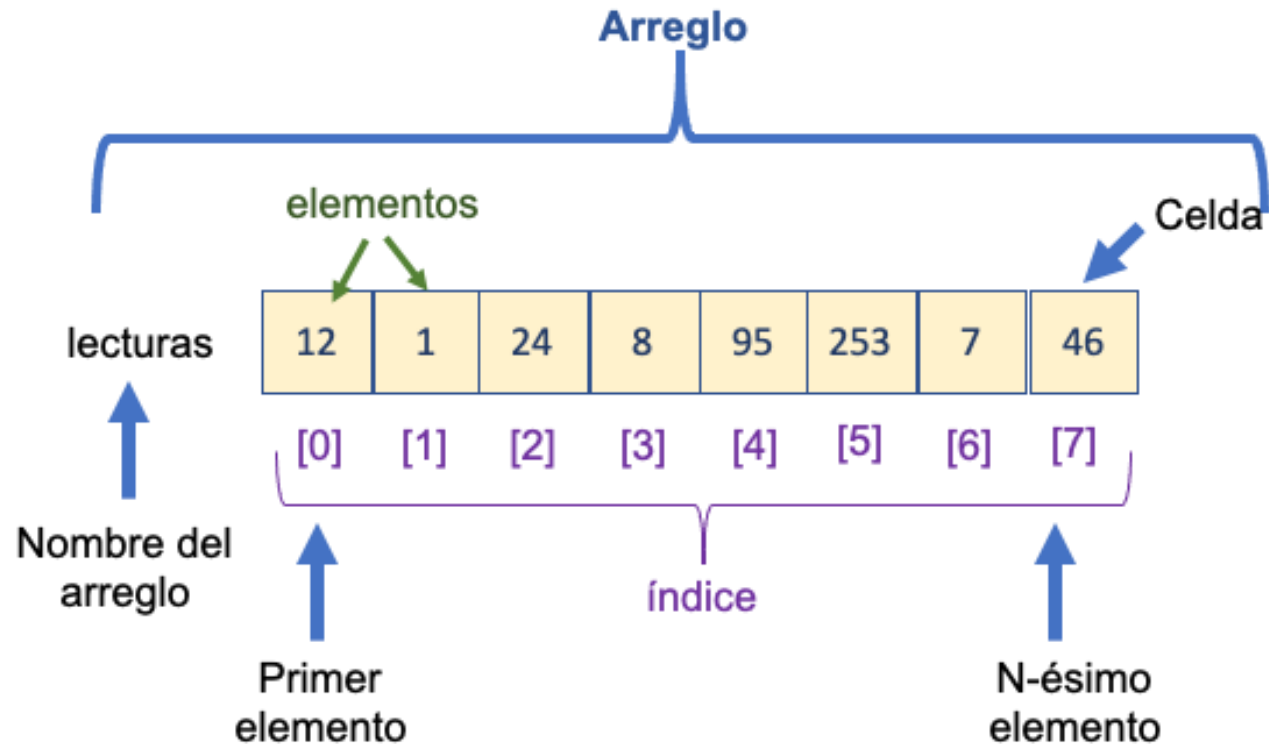


toCharArray

```
String saludo = "hola";  
Char[] dividir = saludo.toCharArray();  
  
> ["h", "o", "l", "a"]
```

Arreglos Unidimensionales

Un arreglo unidimensional es una estructura de datos que almacena una colección de elementos del mismo tipo en una secuencia lineal. Los elementos se organizan en posiciones consecutivas de memoria, y cada uno se accede mediante un índice.

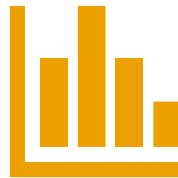


Características



Tamaño fijo

El tamaño del arreglo debe definirse al momento de su creación y no puede cambiar.



Índices basados en cero

El primer elemento tiene índice 0, el segundo 1, y así sucesivamente.



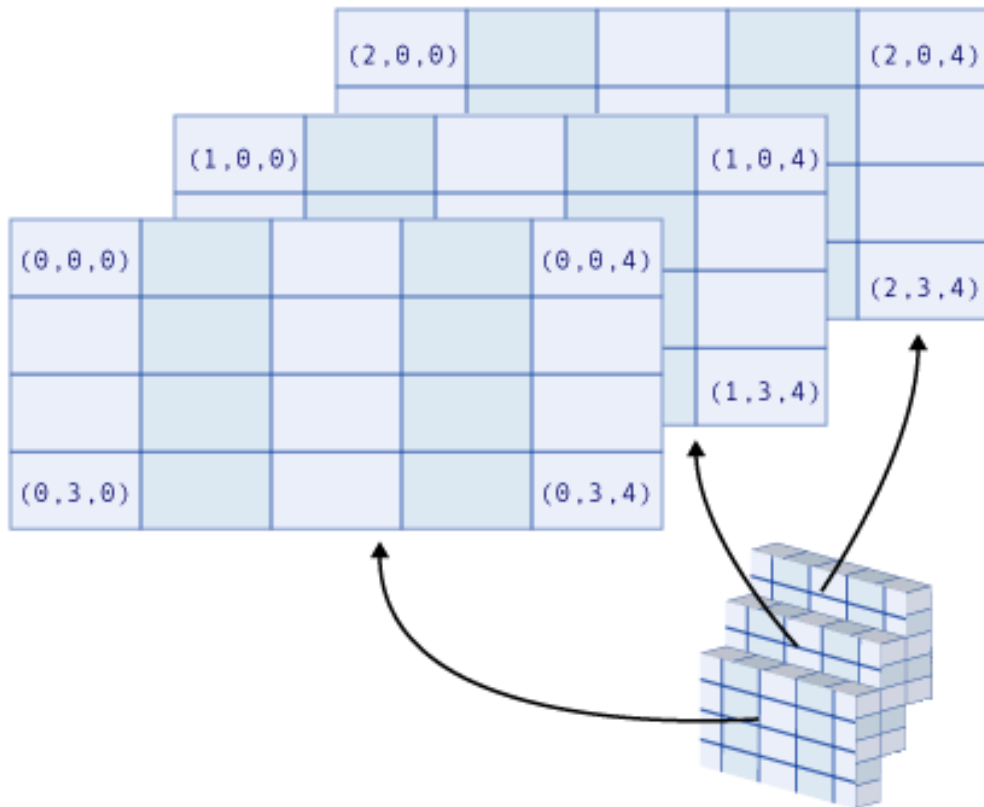
Acceso directo

Los elementos se acceden en tiempo constante ($O(1)$) utilizando su índice.

Declaración

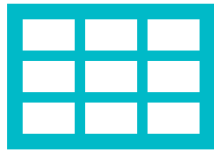
```
// Declaración  
int[] arreglo = new int[5];  
  
// Inicialización con valores  
int[] numeros = {1, 2, 3, 4, 5};
```

Arreglos Multidimensionales



Un arreglo multidimensional es una estructura de datos que organiza elementos en más de una dimensión. Los arreglos bidimensionales (matrices) son los más comunes, pero también existen arreglos tridimensionales o de más dimensiones.

Características



Matrices rectangulares

En Java, las matrices bidimensionales suelen ser rectangulares (todas las filas tienen la misma longitud).



Acceso mediante índices

Se utiliza un par de índices para acceder a los elementos: `[fila][columna]`.

Declaración

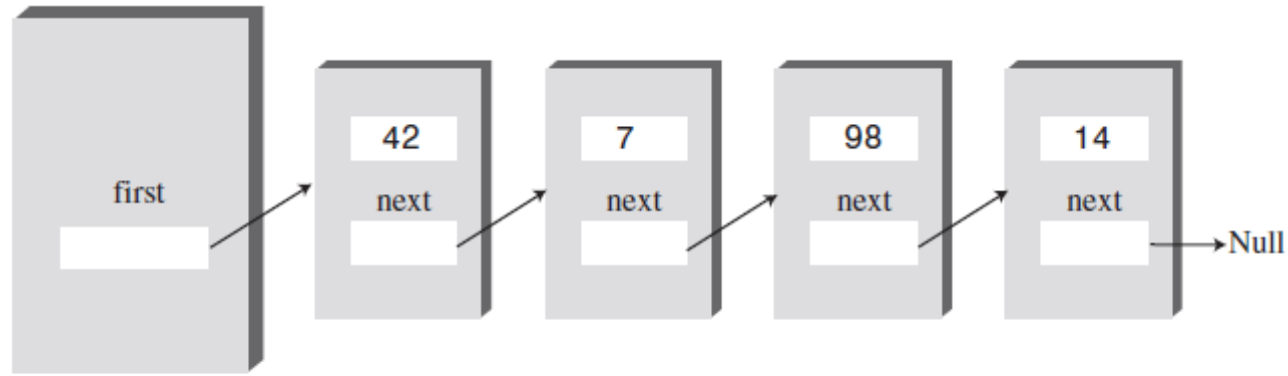
```
// Matriz de 3x3
int[][] matriz = new int[3][3];

// Inicialización con valores
int[][] matrizNumeros = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
```

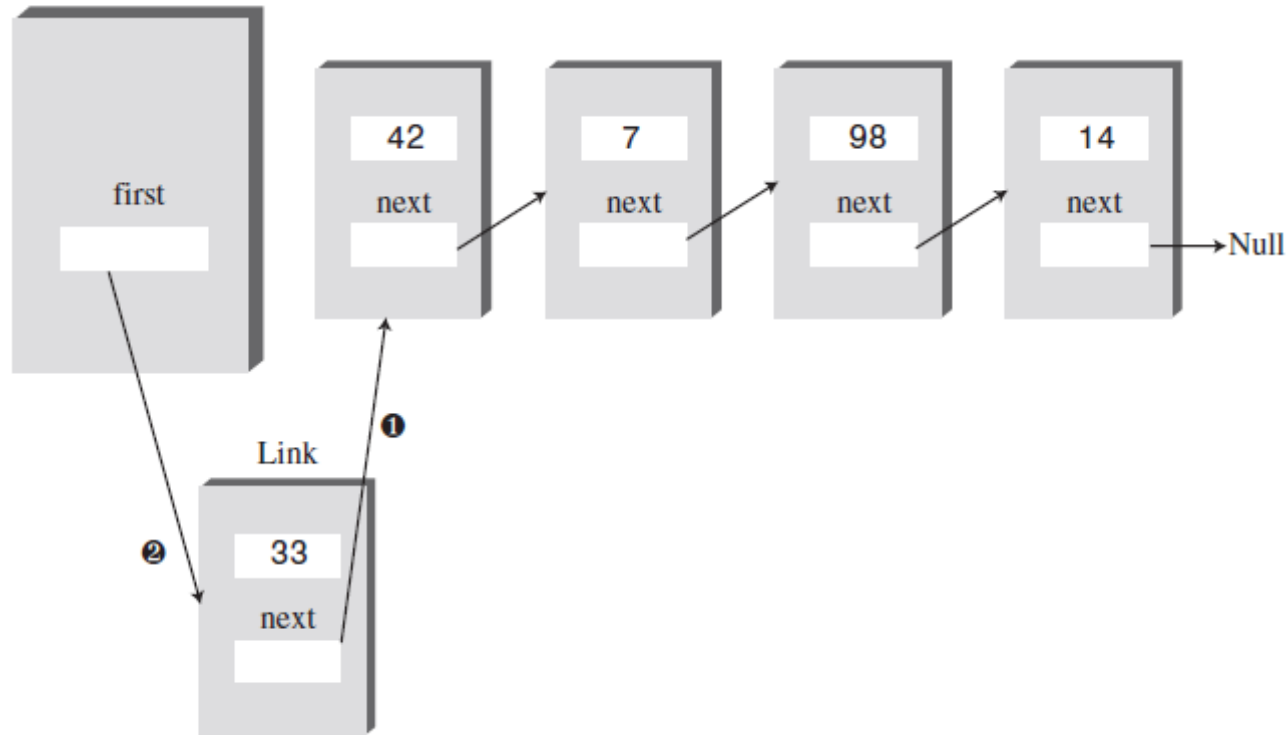
Listas

Definición, ordenamientos





a) Before Insertion



b) After Insertion

Listas

La lista tiene un enlace al primer contenedor y cada contenedor tiene un enlace al siguiente contenedor en la lista. Para agregar un elemento a la lista, el elemento se coloca en un nuevo contenedor y ese contenedor está vinculado a uno de los otros contenedores en la lista.

Listas Dinámicas



Las listas dinámicas son estructuras de datos que permiten almacenar colecciones de elementos del mismo tipo, pero a diferencia de los arreglos, su tamaño puede cambiar dinámicamente durante la ejecución del programa.



En Java, las listas dinámicas se implementan principalmente mediante la interfaz `List` y sus implementaciones como `ArrayList` y `LinkedList`.

Características

Listas Dinámicas

Tamaño variable

- Pueden crecer o reducirse según sea necesario.

Acceso por índice

- Al igual que los arreglos, los elementos se pueden acceder mediante índices.

Métodos útiles

- Proporcionan métodos como `add()`, `remove()`, `get()`, etc., para manipular los elementos.

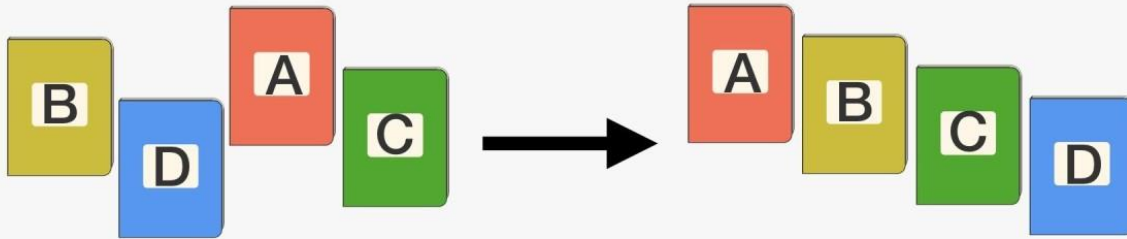
Arreglos

Vs Listas

Característica	Arreglos	Listas Dinámicas
Tamaño	Fijo (No se puede modificar luego de ser creado)	Variable (Puede crecer o reducirse)
Acceso	Por medio de índices	Por medio de índices
Inserción/Eliminación	Costoso (Requiere un nuevo arreglo)	Eficiente (por métodos “add”, “remove”)
Memoria	Menor sobrecarga de memoria	Mayor sobrecarga debido a la gestión interna
Uso común	Datos de tamaño conocido	Datos de tamaño desconocidos

Ordenamiento

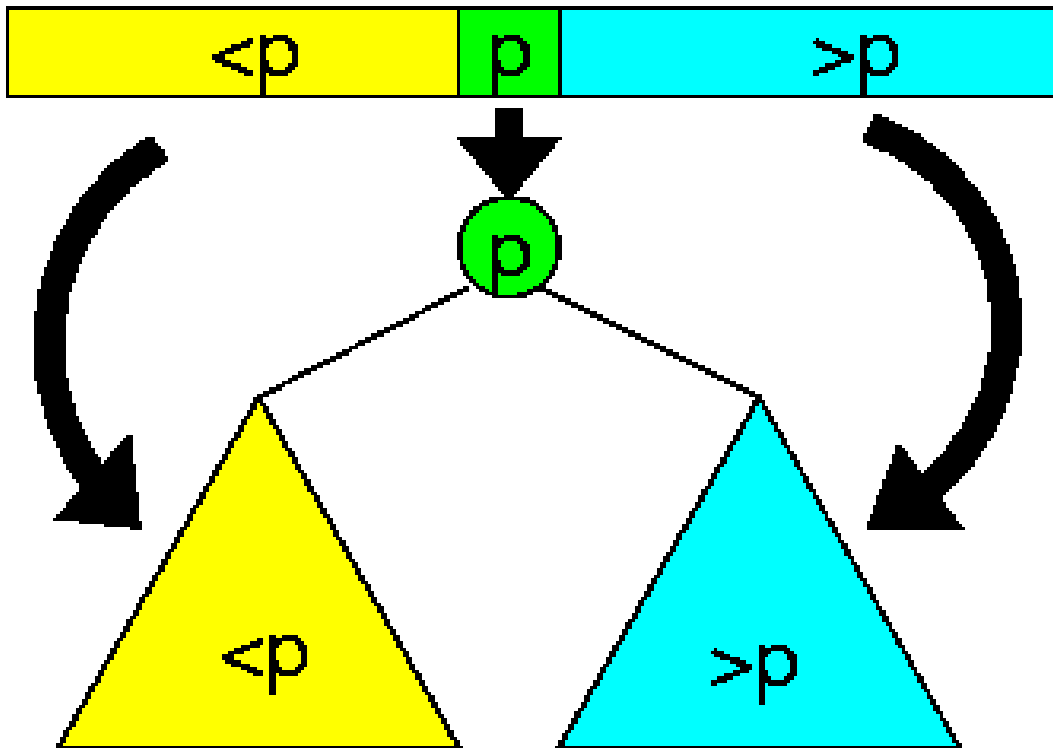
Sorting Algorithms



Es la operación de arreglar los elementos en algún orden secuencial de acuerdo a un criterio de ordenamiento. El propósito principal de un ordenamiento es el de facilitar las búsquedas de los miembros del conjunto ordenado. Ordenar un grupo de datos significa mover los datos o sus referencias para que queden en una secuencia por categorías y en forma ascendente o descendente.

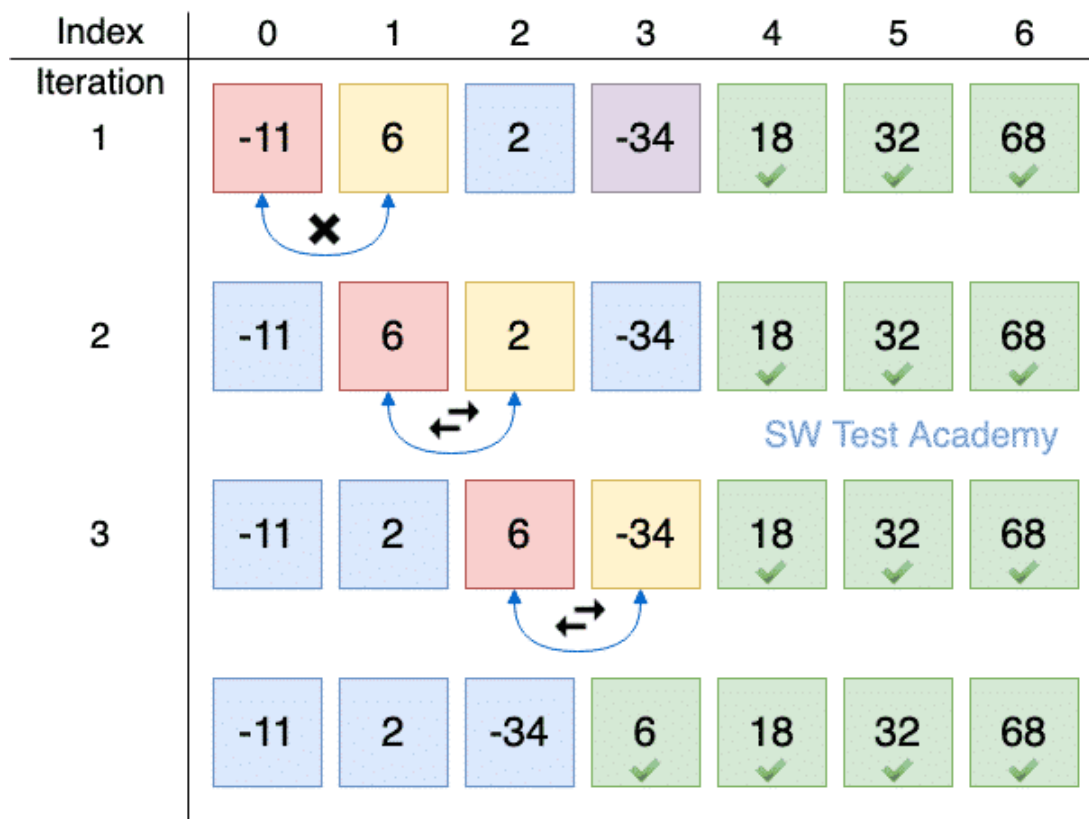
Métodos de Ordenamiento

Los métodos de ordenamiento son algoritmos que organizan los elementos de una colección en un orden específico (ascendente o descendente).

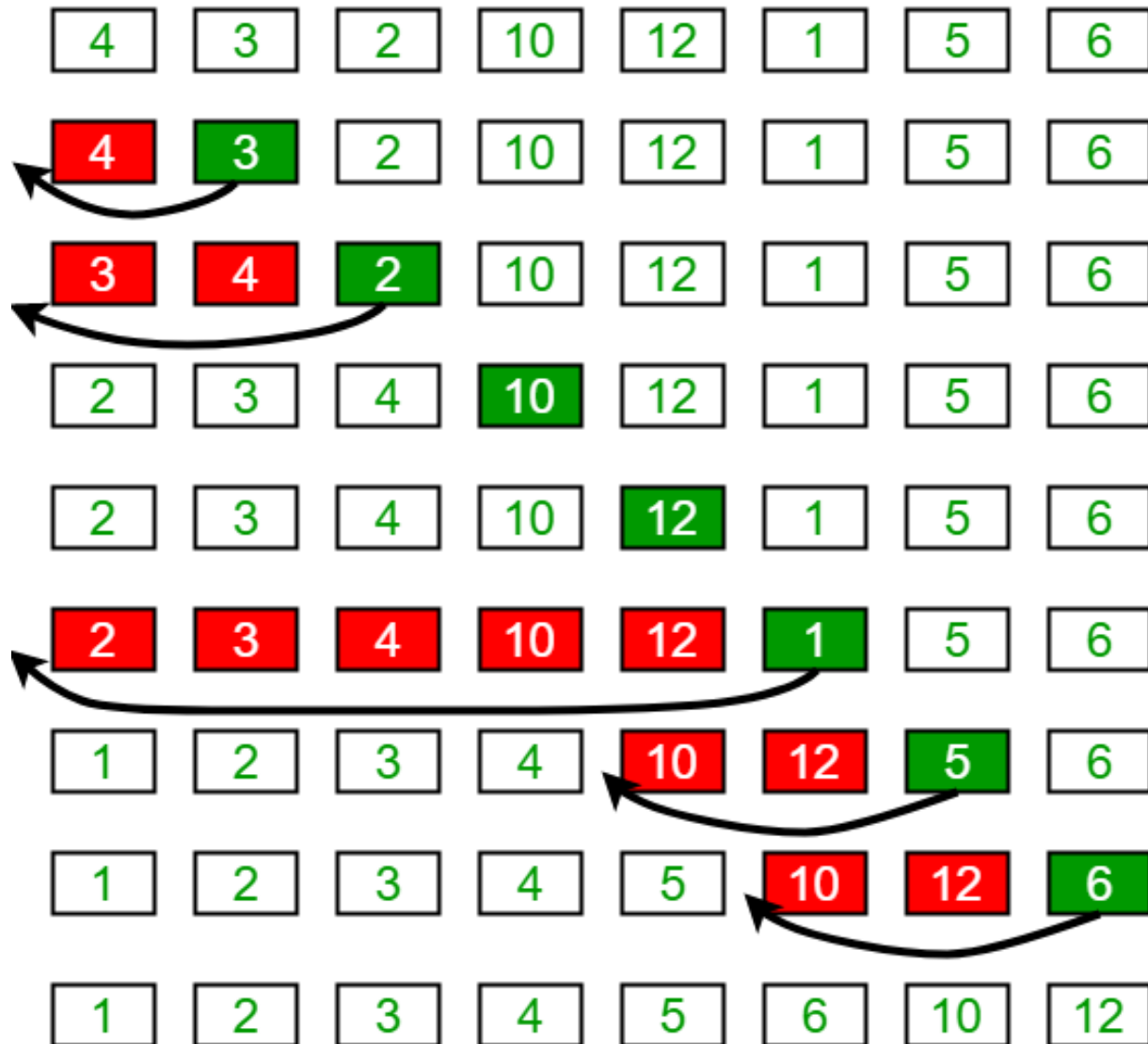


Ordenamiento Burbuja

El ordenamiento burbuja hace múltiples pasadas a lo largo de una lista. Compara los ítems adyacentes e intercambia los que no están en orden. Cada pasada a lo largo de la lista ubica el siguiente valor más grande en su lugar apropiado. En esencia, cada ítem “burbujea ” hasta el lugar al que pertenece.



Insertion Sort Execution Example



Ordenamiento por Inserción

En este tipo de algoritmo los elementos que van a ser ordenados son considerados uno a la vez. Cada elemento es “insertado ” en la posición apropiada con respecto al resto de los elementos ya ordenados.

Ordenamiento por Selección

El ordenamiento por selección mejora el ordenamiento burbuja haciendo un sólo intercambio por cada pasada a través de la lista. Para hacer esto, un ordenamiento por selección busca el valor mayor a medida que hace una pasada y, después de completar la pasada, lo pone en la ubicación correcta. Al igual que con un ordenamiento burbuja, después de la primera pasada, el ítem mayor está en la ubicación correcta. Después de la segunda pasada, el siguiente mayor está en su ubicación.

8	12	25	29	32	17	40
---	----	----	----	----	----	----

8	12	25	29	32	17	40
---	----	----	----	----	----	----

8	12	17	29	32	25	40
---	----	----	----	----	----	----

8	12	17	29	32	25	40
---	----	----	----	----	----	----

8	12	17	29	32	25	40
---	----	----	----	----	----	----

8	12	17	25	32	29	40
---	----	----	----	----	----	----

8	12	17	25	32	29	40
---	----	----	----	----	----	----

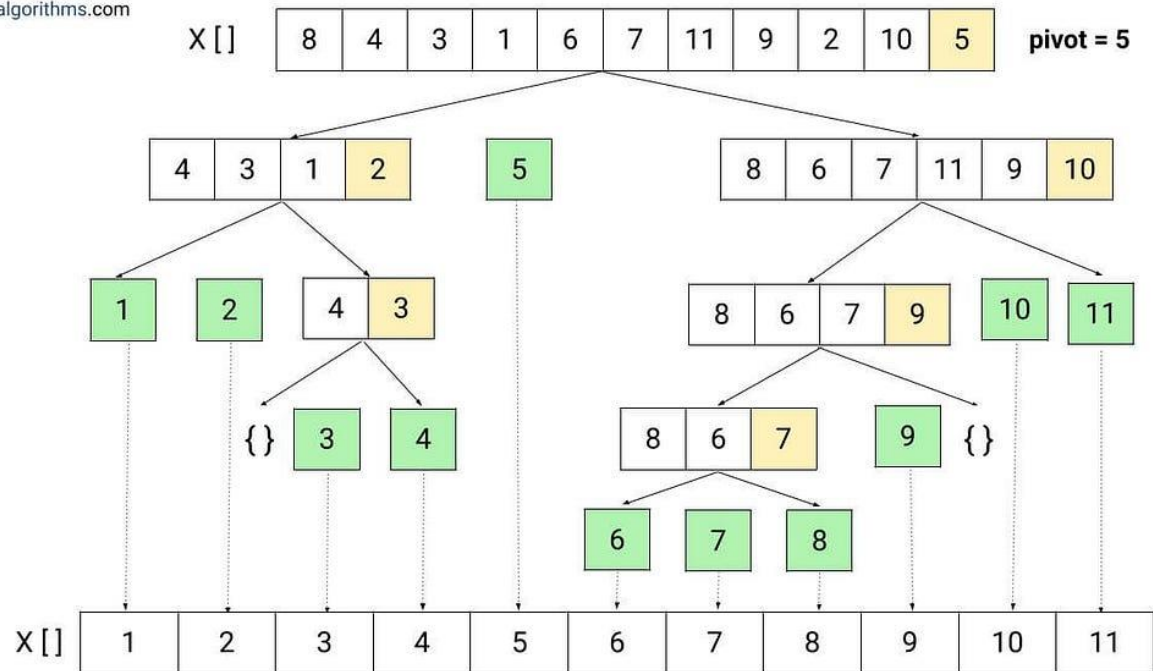
8	12	17	25	32	29	40
---	----	----	----	----	----	----

8	12	17	25	29	32	40
---	----	----	----	----	----	----

8	12	17	25	29	32	40
---	----	----	----	----	----	----

Ordenamiento Rápido

enjoyalgorithms.com



El método Quick Sort es actualmente el mas eficiente y veloz de los método de ordenación interna. Es tambien conocido con el nombre del método rápido y de ordenamento por partición. Este método es una mejora sustancial del método de intercambio directo y recibe el nombre de Quick Sort, por la velocidad con la que ordena los elementos del arreglo. Quicksort es un algoritmo basado en la técnica de divide y vencerás, que permite, en promedio, ordenar n elementos en un tiempo proporcional a $n \log n$.



Dudas o Comentarios
