Universidad Europea

# Fundamentos de programación

## Arrays y Matrices

# Índice

# Objetivos de aprendizaje

Los objetivos que se pretenden alcanzar con este recurso son los siguientes:

- Entender el funcionamiento de los arrays de datos en Java.
- Conocer cómo declarar e instanciar arrays, tanto de una dimensión como multidimensionales (matrices).
- Conocer cómo acceder a los datos almacenados en los arrays, tanto para su lectura como para su modificación.

# 1. Introducción

- Un array (o "arrelgo") almacena una serie de valores **pertenecientes al mismo tipo**, en un bloque continuo de memoria.
- Cada valor puede ser accedido a través de un índice.
- El array que se muestrea en el ejemplo:
  - Sus elementos son valores de tipo entero (`int`).
  - Está almacenado a partir de la dirección de memoria `1000`.
  - Tiene un tamaño (número de elementos) de `5` elementos.
  - Su primer elemento, el `10` está almacenado en el índice `0`.
  - Su segundo elemento, el `-5`, está almacenado en el índice `1`.
  - …
  - Su último elemento, el `6`, está almacenado en el índice `4`. (tamaño -1, al empezar a contar en 0).

**Ejemplo**

Ejemplo de array de enteros en memoria:

| Dirección | | Índice |
|---|---|---|
| 1000 | 10 | 0 |
| 1004 | -5 | 1 |
| 1008 | 23 | 2 |
| 1012 | 17 | 3 |
| 1016 | 6 | 4 |

# 2. Declaración

Su declaración es similar a la de cualquier otra variable:

- En primer lugar, se especifica el **tipo** de datos para los elementos.
- A continuación, se escribe el **nombre** que se le quiere dar al array.

Se diferencia de la declaración de las variables "corrientes", porque **hay que escribir un par de corchetes**, bien delante o detrás del nombre que se proporciona al array:

```
int elementos[];        double numeros[];        String palabras[];        boolean mascara[];

int []elementos;        double []numeros;        String []palabras;        boolena []mascara;
```

# 3. Instanciación

- Para instanciar (crear) el array, se debe especificar su tamaño (número de elementos) mediante el uso de un constructor.

**Recuerda**

**Importante:** Los elementos estarán **inicializados** al **valor por defecto** del tipo definido en la declaración

- Creación de un array para 10 elementos de tipo entero:

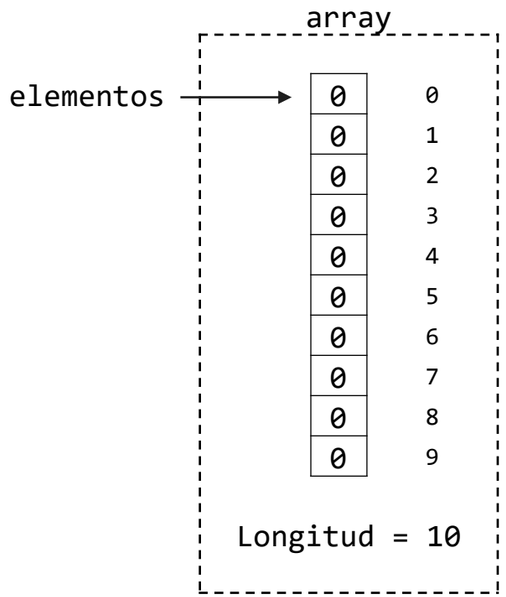
```
// declaración
int elementos[];
// instanciación (new)
elementos = new int[10];
```

- También se puede instanciar al declarar:

```
int elementos[] = new int[10];
```

**Ejemplo**

Ejemplo de array para 10 elementos:



# 4. Acceso

- Cada elemento del array tiene asociado un índice que permite su acceso individual:
  - Estos índices son una secuencia de números enteros que comienzan en **0** y llegan hasta (**tamaño-1**).
  - El acceso a un elemento de un array sirve tanto para **lectura** como para **escritura** (asignación).

**Ejemplo de acceso a arrays:**

```
int notas[] = {5,8,7,9}; // 4 elementos
// Dos lecturas, una asignación, dos lecturas
System.out.println(notas[0]); // primero
System.out.println(notas[3]); // último
notas[2] = 67; // Asigación. Antes era 7
System.out.println(notas[2]); // tercero
System.out.println(notas[5]); // ¿Qué pasa?
```

**Salida por consola de la ejecución del código:**

```
5
9

67
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 5
        at MiClase.main(MiClase.java:22)
```

# 5. Longitud

- Cualquier objeto definido como un array tiene un campo **length** que almacena su tamaño (especialmente útil en bucles)
- Para acceder a dicho campo, se debe usar la sintaxis:

```
nombreArray.length
```

**Ejemplo de acceso a arrays:**

```
int notas[] = {5,8,7,9}; // 4 elementos

System.out.println(notas[0]); //primero
//System.out.println(notas[3]);
System.out.println(notas[notas.length-1]);
System.out.println(notas[notas.length]);
```

**lenth** del array vale **4**
Se intenta acceder al **5º** elemento (1º índice 0)

**Salida por consola de la ejecución del código:**

```
5

9
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 5
        at MiClase.main(MiClase.java:22)
```

# 6. Arrays multidimensionales: Matrices

Se conocen como arrays multidimensionales o matrices a aquellos de más de una dimensión.

- Todas las propiedades y conceptos de los arrays de una dimensión (vistos anteriormente), aplican a las matrices.
- Conceptualmente son arrays de arrays: Un array que en cada uno de sus elementos contiene otro array de los tipos de datos que se hayan definido.
- En la práctica, consiste en añadir un par de corchetes por cada dimensión extra.
- Pueden ser de N-Dimensiones:
    - 2 dimensiones: Matrices rectangulares o cuadradas.
    - 3 dimensiones: Matrices tridimensionales (cubos o prismas rectangulares).
    - Etc …

# 6. Arrays multidimensionales: Matrices

- Ejemplos de matrices N-Dimensiones:
  - **2 Dimensiones:** Matrices rectangulares o cuadradas.

```
// Declaración matriz cuadrada orden 10
int matriz[][] = new int[10][10];

//Declaración e inicialización matriz 2x3 (filas x columnas
String nombres[][]= {{"Luis","Ana"}, {"María","Juan"}, {"Javier","Sandra"},};
```

  - **3 Dimensiones:** Matrices tridimensionales (cubos o prismas rectangulares).

```
// Declaración matriz cúbica orden 10
int matriz[][][] = new int[10][10][10];

/* Ejemplo: Matriz 3D de pixels de una TV HD: 1920 x 1080 x 3 (3 puntos para colores RGB)
 * La 3ª dimensión representa los valores de Red, Green & Blue
 * Cada uno de los valores de R, G, B es un entero entre 0 y 255  */
int matriz[][][] = new int[1920][1080][3]; // la 3ª dimensión representa los valores de RGB
```

# 7. Gestión de memoria en arrays multidimensionales

**Conceptualmente**

Al crear un array multidimensional, por ejemplo una matriz de 5 filas y 10 columnas de valores enteros:

```
int miMatriz[][] = new int[5][10];

// Se asignan valores a la matriz...
```

Se tiende a pensar que la información se almacena en memoria de acuerdo al formato de las matrices (como si se tratase de una tabla de datos).

**Realmente sucede**

Se crea una variable que hace referencia (puntero) a un array de arrays (tantos arrays como filas tenga la matriz). Este array de arrays contiene referencias a los arrays de datos.

Cada uno de estos últimos arrays (correspondiente a una fila), almacena los valores enteros (en este caso) consecutivamente, en memoria (porque los enteros son de tipos simples, si no también serian referencias a los objetos contenidos).

# 7. Gestión de memoria en arrays multidimensionales

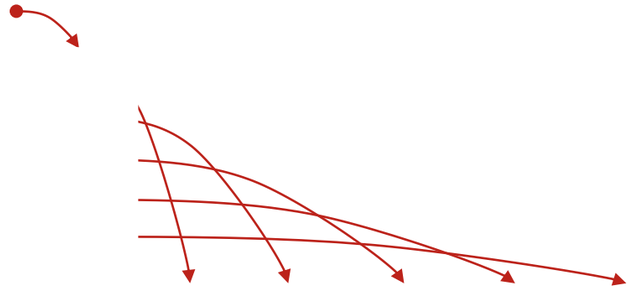
### Conceptualmente

**miMatriz:**

columnas

| filas | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 45 | -49 | 71 | -20 | 132 | -60 | -119 | 105 | 75 | -58 |
| 1 | 111 | 34 | -84 | -117 | 131 | 99 | -63 | 11 | 20 | 98 |
| 2 | 125 | 15 | 12 | -131 | 113 | -129 | 75 | -150 | -92 | -67 |
| 3 | 99 | -23 | -2 | 58 | 69 | 99 | -17 | -1 | -25 | 86 |
| 4 | -59 | 31 | 43 | -30 | 51 | 30 | 136 | 104 | -107 | 40 |

### Realmente



**`miMatriz`**

- `5 filas (de la 0 a la 4)`
- `10 columnas (de la 0 a la 9)`

# 7. Gestión de memoria en arrays multidimensionales

### Conceptualmente

**miMatriz:**

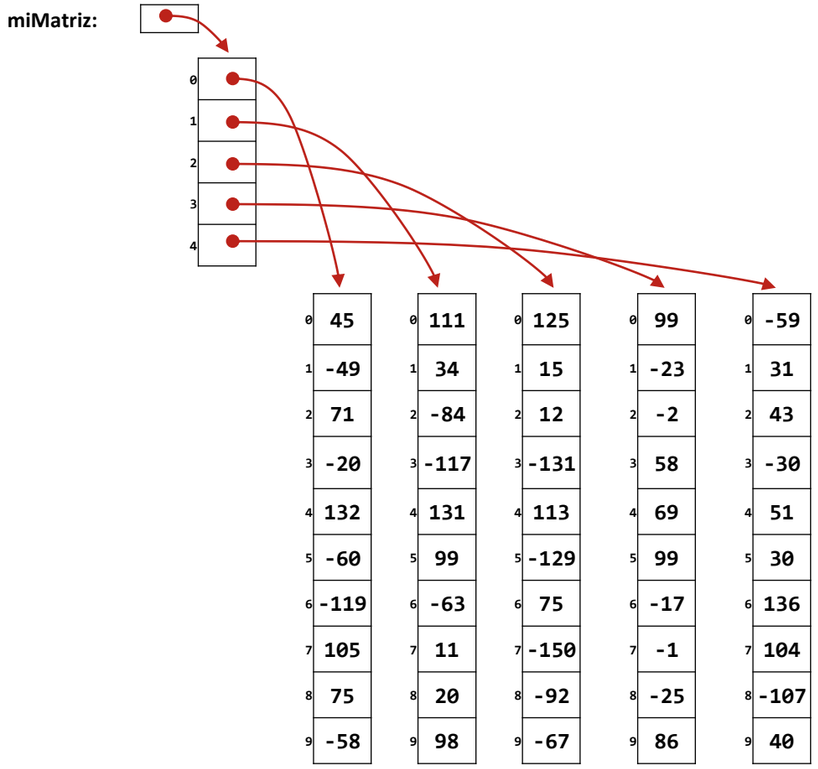
columnas

| filas | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 45 | -49 | 71 | -20 | 132 | -60 | -119 | 105 | 75 | -58 |
| 1 | 111 | 34 | -84 | -117 | 131 | 99 | -63 | 11 | 20 | 98 |
| 2 | 125 | 15 | 12 | -131 | 113 | -129 | 75 | -150 | -92 | -67 |
| 3 | 99 | -23 | -2 | 58 | 69 | 99 | -17 | -1 | -25 | 86 |
| 4 | -59 | 31 | 43 | -30 | 51 | 30 | 136 | 104 | -107 | 40 |

**`miMatriz`**

- `5 filas (de la 0 a la 4)`
- `10 columnas (de la 0 a la 9)`

### Realmente



# 8. Acceso en Arrays multidimensionales

Para acceder a elementos de una matriz (array multidimensional) se sigue la misma mecánica que con los arrays, pero en este caso se debe definir el índice de la fila, columna, … de forma muy similar a la notación matemática.

- **Ejemplos de acceso en Matrices:**

```
int notas[] = {5,8,7,9}; // 4 elementos

// Dos lecturas, una asignación, dos lecturas

System.out.println(notas[0]); // primero

System.out.println(notas[3]); // último

notas[2] = 67; // Asigación. Antes era 7

System.out.println(notas[2]); // tercero

System.out.println(notas[5]); // ¿Qué pasa?
```

**Salida por consola de la ejecución del código:**

```
5
9

67
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 5
        at MiClase.main(MiClase.java:22)
```

# 8. Acceso en Arrays multidimensionales

Los tipos de datos permiten definir y conocer:

- Se prefiere instanciar el array después de declararlo.

```
int datos[] = new int[10];
```

```
int datos[];
datos = new int[10];
```

- Los elementos de un array de tamaño **N** tienen índices con un rango de **0…N-1**. Si se intenta acceder a elemento en posición mayor o igual que **N** (posicion≥ **N**) o menor que 0 (posicion **<0**), se producirá un error de ejecución: **Excepción.**
- No se puede redimensionar un array. Si se necesita modificar su tamaño, hay que crear uno nuevo con el tamaño deseado y copiar los datos que se deseen mantener, así como añadir los nuevos.
- Al crear bucles que recorran un array, se debe utilizar siempre el campo **length** en la condición del bucle.

# Referencias bibliográficas

Chapter 10. Arrays: <https://docs.oracle.com/javase/specs/jls/se7/html/jls-10.html>

Eckel, Bruce (2008). "Thinking in Java" 4th ed. Prentice Hall.

Horstmann, C.S. (2018). "Core Java I – Fundamentals" 11th ed. Prentice Hall.

Horstmann, C.S. (2016). "Core Java II – Advanced Features" 10th ed. Prentice Hall.

Schildt, H. (2018). "Java. A Beginner's Guide" 8th ed. Oracle Press.

Schildt, H. (2018). "Java. The Complete Reference" 11 th ed. Oracle Press.

Universidad Europea



Ve más allá