# Grado en Ingeniería Informática

## Metodología de la Programación

## Sesión 3

*Arrays*

Docentes:

Raúl Marticorena

# Índice de contenidos

# 1. Introducción

En este tema se introduce el concepto de *arrays*[1] en Java: su declaración, reserva de memoria, inicialización y manejo de *arrays* de una y varias dimensiones en el lenguaje Java. Para una consulta exhaustiva se recomienda [Arnold et al., 2001, Eckel, 2007] o bien la documentación en línea de Oracle "Arrays".

# 2. Objetivos

- Familiarizarse con la sintaxis en Java para la declaración y uso de *arrays.*
- Estudiar el manejo interno de *arrays* en Java, en particular con *arrays* multidimensionales.

# 3. Contenidos específicos del tema

La definición habitual de *array* es: *"Conjunto de un número fijo de datos del mismo tipo relacionados entre sí, ordenados en posición"*.

Una de sus dimensiones ocupa posiciones contiguas en memoria. El *tipo relacionado* puede ser primitivo, referencia a objeto, o variable de tipo (en genericidad, se verá en un tema posterior).

En Java los *arrays* **son objetos** y se manipulan **siempre por referencia** (concepto similar al puntero, como dirección de memoria, del lenguaje C, pero con ciertas limitaciones en cuanto a las operaciones a poder realizar). Cualquier variable (ya sea variable local, argumento formal de método, variable de retorno de método, de instancia, de clase, etc.) puede ser declarada de tipo *array*.

A la hora de trabajar con *arrays* hay que distinguir varios pasos:

- 1º) **Declarar** la variable de tipo *array* (solo crea espacio para la referencia o puntero, 4 bytes en Java) de la forma:
  - `tipoBase[] nombreVariableArray`[2]`;`
- 2º) **Reservar** o **asignar** la **memoria** para los elementos del array:
  - Con el operador **`new`**.
  - O bien inicializando directamente los valores de los *arrays* con la notación de llaves y valores iniciales. Esto implícitamente realiza la correspondiente reserva de memoria.
- 3º) **Inicializar** los valores de las posiciones del array.
  - Si los elementos son tipo referencia a objeto, hay que **inicializar** cada uno (por defecto tomarán valores **`null`**)
  - En caso contrario, usando tipos primitivos, toman valores por defecto (ver valores en tabla del guión de tipos primitivos en sesiones previas).

1 También se utilizan los términos: arreglo, vector, lista, matriz o tabla.
2 La sintaxis Java permite colocar los corchetes después del nombre de variable, pero NO se recomienda y la mayoría de herramientas de comprobación de calidad de código lo señalan como un error.



**Resumiendo:** con un *array* debemos **declarar la variable, reservar memoria e inicializar** con los valores del *array*. Java permite realizar estos tres pasos en una única línea, o bien paso a paso.

Los *arrays* son entidades **estáticas**, esto es, una vez declarada una dimensión (en tiempo de ejecución) **NO pueden cambiar de tamaño** (salvo que se vuelva a crear uno nuevo, y se sustituya).

Tienen siempre una propiedad (atributo) de nombre `length`, accesible a través del nombre del *array* con la notación del punto, que devuelve la longitud reservada al *array* inicialmente.

## 3.1 Ejemplos de declaración y reserva.

Los siguientes ejemplos trabajan con *arrays* de elementos de tipo **`int`**, pero se puede utilizar cualquier tipo primitivo[3] como tipo base del *array*, y como se verá en temas posteriores, tipos más complejos e implementados por el propio programador. La longitud del *array* es fija y no depende del número de elementos introducidos en el *array* posteriormente. El tamaño de un *array* se define con un valor entero que puede ser una constante o literal, pero también puede ser un dato calculado (e.g. una variable).

En la reserva de memoria para tipos primitivos se realiza una reserva del espacio físico necesario (su valor), mientras que con *arrays* de tipo referencia a objetos, solo se reserva el espacio para las referencias (los punteros), no el espacio realmente necesario para los distintos objetos.

A continuación se muestra el Código 1 con ejemplos de declaración e inicialización de *arrays.*

```
// declara el array c
int[] c; // valor null en c incialmente
// asigna memoria, 12 posiciones para enteros (12 * 4 bytes) con valores por defecto
c = new int[12];
// declaracion y asignacion de memoria en un solo paso
int[] d = new int[12]; // 12 valores cero
// declaración, reserva y asignación con tres valores enteros
int[] e = { 1, 2, 3 };
// imprime el número de elementos declarados del array, en el ejemplo 12
// al ser tipo primitivo int tenemos 12 ceros por defecto
System.out.printf("%d", c.length);
```

*Código 1: Ejemplo de declaración e inicialización de arrays*

## 3.2 Acceso a los elementos de un array

La notación utilizada para acceder a los elementos del *array* utiliza los corchetes (e.g. `nombreVariableTipoArray[índice]`). El valor del índice **solo puede ir** desde `0` hasta `n-1` siendo `n` la longitud/reserva de memoria del *array*.

Si el valor utilizado como índice **excede** el límite de las posiciones reservadas o es **negativo**, Java lanza una **excepción** (error en tiempo de ejecución, que por el momento, interrumpirá bruscamente la ejecución).

Cuando se accede a un elemento de un *array:*

- Si se utiliza en la **parte derecha de una asignación** se **lee el valor.**
- Si se utiliza en la **parte izquierda de una asignación** se está **escribiendo** sobre dicha posición un nuevo valor.

Por otro lado, no se pueden comparar dos referencias a arrays directamente con el operador == para comprobar que los valores contenidos coinciden (estaríamos comparando las referencias o punteros), sino que habrá que comparar la igualdad de sus elementos, uno a uno, con la notación de corchetes.[4]

3 Se recuerdan los ocho tipos primitivos existentes en Java: **`byte`**, **`short`** , **`int`**, **`long`**, **`float`**, **`double`**, **`char`** y **`boolean`**. Siempre van en minúscula (mientras que los tipos referencia a objeto empiezan siempre con mayúscula).

4 Existe una clase de utilidad `java.lang.Arrays` para facilitar este tipo de operaciones, pero por el momento se trabajará en un estilo menos "orientado a objeto" con la notación de arrays clásica.



En el Código 2, se muestra un ejemplo de código parcial de la clase `RupturaArray` en el que se declaran y manipula un *array,* accediendo a una posición **no permitida** (no reservada).

```
public static void main(String[] args) {
	short[] a = {1, 2, 3, 4}; // local al metodo main

	// OJO con <= se genera excepcion
	for ( int i = 0; i <= a.length; i ++ ) {
		System.out.printf("%d%n", a[i]); // leemos el valor en la posición i
		a[i] = a[i] + 1; // escribimos en la posición i incrementando el valor en 1
	} // for

} // main
```

*Código 2: Ejemplo de recorrido de un array con excepción (error)*

El resultado en pantalla se muestra en el Código 3, obteniendo la traza del error.

El tipo de excepción originada (`java.lang.ArrayIndexOutOfBoundsException`) indica que se ha intentado acceder a una posición fuera de los límites del *array* (en la posición 4). La línea donde se ha producido el acceso incorrecto en la línea 21 del código (`RupturaArray.java:21`). Este error es habitual en el manejo de bucles y *arrays.*

```
1
2
3
4
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
	at RupturaArray.main(RupturaArray.java:21)
```

*Código 3: Traza en pantalla de la ejecución del código*

**Nota**: si se utiliza un entorno integrado, como Eclipse, los textos subrayados son realmente hiperenlaces que permiten acceder rápidamente al código correspondiente, facilitando la navegación a la fuente del error.

## 3.3 Paso de arrays como argumentos[5]

Mientras que los tipos primitivos se pasan por valor, los objetos se pasan "por referencia" aunque **realmente Java pasa las referencias de los objetos, por valor, pasando el valor del "puntero"**.

Los *arrays*, al ser objetos, se pasan de igual forma, a través del nombre de la entidad (identificador de variable) con que se declaró. La referencia al *array* no puede ser modificada en el cuerpo del método (si se modifica no tiene efecto al retornar el método), pero sí los valores almacenados en el *array* (sean tipo primitivo o referencia a objetos):

Ej: `mostrar(unArray); // se pasa la referencia al array manejado con la variable unArray al método`

En la declaración del método (signatura o firma del método) que recibe la referencia al *array,* se especifica el tipo de elementos y dimensiones (ojo, no el número de elementos del *array*, sino las

5 A lo largo de la asignatura se utilizará el término **argumento** en lugar de ***parámetro*** para no confundir con los ***parámetros genéricos,*** que se verán en el Tema 4. Genericidad.



dimensiones, tantas como pares de corchetes). Debe existir coincidencia entre la declaración de tipo del argumento actual y formal (como en cualquier paso de argumentos).

Ej: `void mostrar(int[] elArray) {} // el tipo de la variable unArray debe coincidir con el tipo de argumento formal elArray. En este caso ambos son un array de una dimensión de enteros.`

## 3.4 Ejemplos de uso de arrays de una dimensión

El presente ejemplo (ver Código 4) muestra los resultados de una encuesta realizada a una muestra de 40 clientes. Se valora la calidad de un local con valores de 0 a 10. El módulo genera un resumen de la encuesta indicando la frecuencia de cada una de las calificaciones.

Tal y como se indicó en el guión de tipos primitivos, el valor por defecto para el tipo **`int`** es cero. Por lo tanto, un *array* de enteros sin inicializar, como el *array* `frecuencias` en el ejemplo, tendrá inicialmente valores cero en todas las posiciones del mismo. Por ejemplo, si se utiliza un array de 10 valores de tipo `boolean`, tendremos por defecto 10 valores **`false`**.

Se debe poner especial atención a la declaración, reserva, inicialización, y paso por valor de la referencia a los métodos auxiliares.



```java
public class Encuesta {

	/**
	 * Valor constante con el numero de pesos (votaciones). En Java se utiliza la
	 * palabra reservada final para indicar que es constante (similar al uso de
	 * const en C). La convención de nombres es utilizar mayúsculas.
	 */
	private static final int VALORES = 11;

	/**
	 * Función principal (Método raíz).
	 *
	 * @param args
	 *            parametros de entrada introducidos en línea de comandos
	 */
	public static void main(String[] args) {
		// Respuestas introducidas por los clientes
		int[] respuestas = { 1, 2, 6, 4, 4, 4, 3, 7, 9, 0, 1, 2, 6, 4, 8, 5, 3,
				7, 9, 0, 1, 2, 6, 4, 8, 5, 3, 3, 9, 0, 1, 2, 6, 4, 8, 5, 3, 2,
				9, 10 };
		// Tabla de frecuencias
		int[] frecuencias = new int[VALORES];

		calcular(frecuencias, respuestas); // calcular frecuencias
		mostrar(frecuencias); // calcular valores
	}

	/**
	 * Calcula la frecuencia de las apariciones de cada peso.
	 *
	 * @param frecuencias
	 *            frecuencia de aparición de cada número / voto
	 * @param respuestas
	 *            valores de números / votos recibidos en la encuesta
	 */
	private static void calcular(int[] frecuencias, int[] respuestas) {
		for (int j = 0; j < respuestas.length; j++) {
			++frecuencias[respuestas[j]];
		}
	}

	/**
	 * Muestra por pantalla las frecuencias de las votaciones.
	 *
	 * @param frecuencias
	 *            frecuencias de aparición de cada número / voto
	 */

	private static void mostrar(int[] frecuencias) {
		System.out.printf("--------------------%n");
		System.out.printf("Tabla de frecuencias%n");
		System.out.printf("--------------------%n");
		for (int j = 0; j < frecuencias.length; j++) {
			System.out.printf("Valor: %d \t Frecuencia: %d%n", j, frecuencias[j]);
		}
	}
}
```

*Código 4: Ejemplo de manejo de arrays con valores de una encuesta*

## 3.5 Arrays de varias dimensiones

Podemos utilizar *arrays* con dos o mas subíndices para representar *arrays* multidimensionales, como por ejemplo tablas de valores de varias dimensiones. Para la declaración se utiliza un par de corchetes por cada dimensión, y preferiblemente, colocados todos siempre junto al tipo de elementos del array[6].

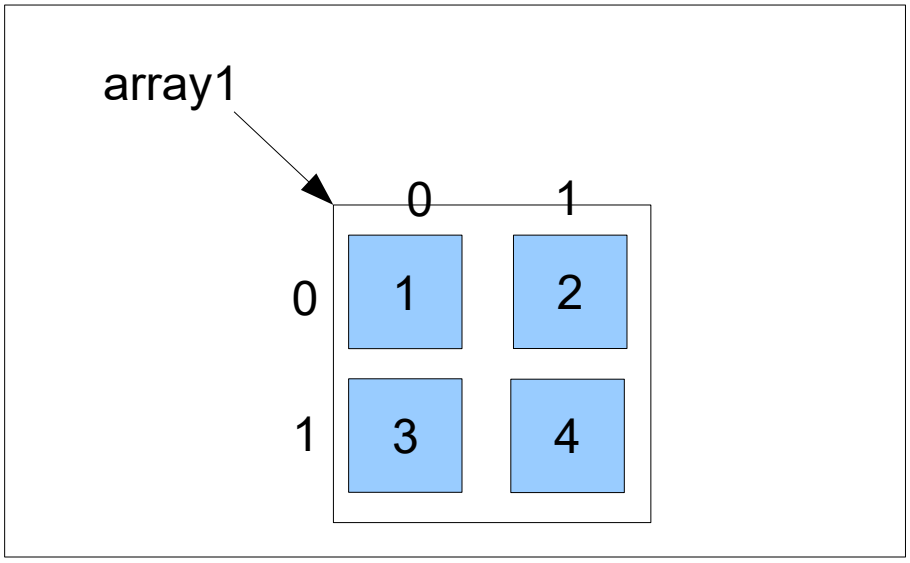
6 Java permite sintácticamente declaraciones de *arrays* multidimensionales colocando los corchetes de múltiples formas, como por ejemplo `char[][] a[];` o `char a[][][];` pero **no se recomienda su uso**.



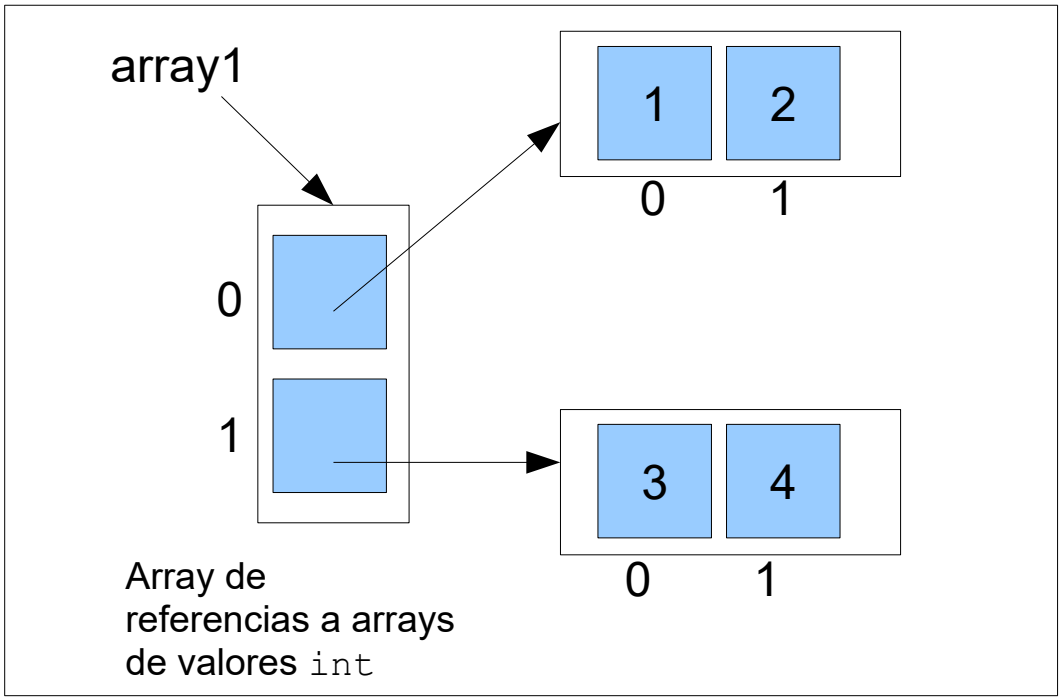
Ej: `char[][][] arrayDeTresDimensionesDeCaracteres;`

Java **no maneja directamente *arrays* de múltiples subíndices**, sino que utiliza internamente la idea de ***arrays* de *arrays***, donde cada elemento se identifica de la forma `nombreArray[indiceDim1]...[indiceDimN]`.

Supongamos que declaramos un *array* de dos dimensiones: `int[][] array1 = { {1, 2}, {3, 4} };` aunque la representación mental que podemos realizar es que el *array* es una tabla de 2x2 con cuatro enteros almacenados en el *array*, tal y como se muestra a continuación (ojo, podemos utilizar esta representación visual para representar y facilitarnos el manejo del array pero internamente, J**ava NUNCA lo maneja así)**:




**realmente** en memoria, Java lo gestiona como un ***array* de una dimensión, donde cada uno de sus elementos es a su vez un "*array* de enteros"**, tal y como se muestra a continuación.




Si el *array* fuera de más de dos dimensiones, se vuelve a aplicar el razonamiento previo.

En el Código 5, podemos ver la declaración, inicialización y acceso a *arrays* de dos dimensiones en Java:

```java
// array de 3 filas por 4 columnas
int[][] b = new int[3][4];
// array de 2 filas por 3 columnas
int[][] c = { { 1, 2, 3} , {2, 4, 6} };
// asignamos 5 al elemento de la fila 1, columna 0
b[1][0]=5;
```

*Código 5: Declaración, inicializacion y acceso a un array de dos dimensiones*

En Java, solo es obligatorio indicar el índice de la primera dimensión en la reserva de memoria inicial (e.g. `new int[3][];`), aunque no sea tan habitual. A posteriori, se podría declarar el tamaño del array en la segunda dimensión (asignando el array a la posición) y así sucesivamente.

Debemos consultar la dimensión de los *arrays* multidimensionales, teniendo en cuenta la idea de *array* de *arrays*. Esto es especialmente útil cuando tenemos *arrays* que no forman rectángulos o se trabaja en n-dimensiones. En el Código 6, podemos ver el acceso a las longitudes de cada una de las dimensiones del *array* previamente declarado, así como la declaración e inicialización de un *array* triangular.

```java
System.out.printf("%d%n", c.length); // valor 2

System.out.printf("%d%n", c[0].length); // valor 3

//ejemplo de un array triangular

int [][] triangular = { { 1, 2, 3} , { 4, 5 }, { 6 } };
```

*Código 6: Longitudes en las dimensiones y array triangular.*

El *array* `triangular` tendría un aspecto similar a:

1 2 3

4 5

6

## 3.6 Ejercicios

- Escribir un programa `Palindromo.java`, con un método `main` que compruebe si un array de caracteres (tipo `char`), inicializado en el propio código, es un palíndromo o no, mostrando el resultado en pantalla (Ej: "Sí es palíndromo" o "No es palíndromo"). Un palíndromo se lee igual de izquierda a derecha que de derecha a izquierda. Ej: *rodador, ana, otto, atar a la rata*. Para simplificar el ejercicio se considera que no hay espacios en blanco ni tabuladores en el array (solo letras).
  - Para su resolución se debe resolver el problema con un método estático, a utilizar desde el método `main`, con las correspondiente signatura. Dado un *array* de caracteres comprueba si es o no palíndromo, devolviendo un valor lógico (`true` o `false`):
    - `static boolean esPalindromo(char[] texto)`
- Escribir un programa `Tabla.java` con un método `main` que genera una tabla de m x n que almacene los números consecutivos desde 1 hasta (m x n). Una vez inicializada la matriz mostrar sus valores en pantalla. Después intercambiar los valores por sus cuadrados y volver a mostrar la matriz. Nota: los valores n y m se fijan en el código. En el ejemplo con m=3, n=3, se almacenan los valores hasta 9, y posteriormente se sustituyen por sus valores elevados al cuadrado como se muestra a continuación:



| | | | | | |
|---|---|---|---|---|---|
| 1 | 2 | 3 | ----------> 1 | 4 | 9 |
| 4 | 5 | 6 | ----------> 16 | 25 | 36 |
| 7 | 8 | 9 | ----------> 49 | 64 | 81 |

- Para su resolución se deben implementar tres métodos estáticos, a reutilizar en el cuerpo de un método `main`, como el siguiente:

```
public static void main(String[] args) {
	int[][] tabla = new int[3][3]; // declaración y reserva
	inicializar(tabla); // inicializa el array
	mostrar(tabla); // muestra en pantalla el estado actual
	elevarAlCuadrado(tabla); // eleva al cuadrado cada posición
	mostrar(tabla); // muestra en pantalla el estado actual
}
```

- Se deja como ejercicio deducir la signatura de los correspondientes tres métodos: `inicializar`, `mostrar` y `elevarAlCuadrado`. Como pista, los tres métodos no devuelven ningún valor. Tanto `inicializar` como `elevarAlCuadrado` modifican el estado del array recibido.

- Escribir el código Java que permita pintar en pantalla el *array* triangular declarado anteriormente en la Sec. 3.5, utilizando siempre la propiedad `length` para consultar la longitud de cada una de sus dimensiones.
- Escribir un programa `Estadistico.java`, con un método `main`, que dado un *array* de valores en punto flotante (`double`) codificado directamente en el código, muestra el valor máximo, mínimo y la media aritmética o promedio del *array* de números.
  - Para su resolución se deben implementar tres métodos estáticos, a reutilizar en el cuerpo de un método `main`, como el siguiente:

```
public static void main(String[] args) {
	double[] numeros = { 1.0, 1.0, 2.0, 3.0 };
	System.out.printf("El valor máximo es %.2f%n", obtenerMaximo(numeros));
	System.out.printf("El valor mínimo es %.2f%n", obtenerMinimo(numeros));
	System.out.printf("La media arimética es %.2f%n", obtenerMedia(numeros));
}
```

  - Las signaturas de los tres métodos deben ser:
    - `static double obtenerMaximo(double[] numeros)`
    - `static double obtenerMinimo(double[] numeros)`
    - `static double obtenerMedia(double[] numeros)`
  - Como valor mínimo de un valor `double`, existe una constante definida en Java como `Double.MIN_VALUE`. En el caso especial de que un *array* no tenga elementos se devolverá este valor como máximo.
  - Como valor máximo de un valor `double`, existe una constante definida en Java como `Double.MAX_VALUE`. En el caso especial de que un *array* no tenga elementos se devolverá este valor como valor mínimo.
  - Si el *array* no tiene elementos, su media aritmética debería ser 0.0.
  - **Nota**: en siguientes lecciones se profundizará sobre el uso de las clases de envoltura como `Double` y el uso de constantes.



# 4. Resumen

En esta sesión de prácticas se introduce el concepto de *array* en Java, su declaración (incluyendo la reserva de memoria), su inicialización y acceso a sus datos (tanto lectura como escritura) a través de la notación de corchetes.

Los *arrays* en Java son objetos que pueden contener valores tanto de tipo primitivo, como referencias a objeto. Eso incluye una única propiedad (atributo) con la longitud de cada una de las dimensiones, denominada `length`. La propiedad es de sólo lectura.

# 5. Glosario

**Array:** conjunto de un número fijo de datos del mismo tipo relacionados entre sí, ordenados en posición.

# 6. Bibliografía

[Arnold et al., 2001] Arnold, K., Gosling, J., and Holmes, D. (2001). El Lenguaje de Programacion JAVA. Serie Java. Pearson Educacion – Addison Wesley, 3a edition.

[Eckel, 2007] Eckel, B. (2007). Piensa en Java. Prentice Hall, 4 edition.

[Prieto el al., 2012] Prieto, N., Casanova, A., Marqués, F. Llorens, M., Galiano, I., Gómez, J.A., González, J., Herrero, C., Martínez-Hinajero, C., Moltó, G. y Piris, J. (2012) Empezar a programar usando Java. Editorial Universitat Politécnica de Valencia

# 7. Recursos

**Bibliografía complementaria**:

[Oracle, 2017] Lesson: Language Basics. The Java Tutorials. Arrays (2017). Disponible en

http://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html



# Licencia

Autor: Raúl Marticorena

Área de Lenguajes y Sistemas Informáticos

Departamento de Ingeniería Civil

Escuela Politécnica Superior

UNIVERSIDAD DE BURGOS

2019