

# Arrays en Java: Guía completa con ejemplos y ejercicios

Creación • Inicialización • Uso • Dimensiones • Métodos útiles • Ordenación • Búsqueda

## 1. Introducción

Un array en Java es una estructura de datos de tamaño fijo que almacena elementos del mismo tipo (primitivo u objetos) en posiciones contiguas de memoria. Proporciona acceso rápido por índice ( $O(1)$ ) y es la base para colecciones más complejas.

- Índices basados en 0 (el primer elemento está en la posición 0).
- La propiedad `length` indica el número de elementos (no confundir con `length()` de `String`).
- El tamaño es fijo: una vez creado, no puede cambiar.
- Los arrays de tipos primitivos contienen valores; los de objetos contienen referencias (pueden ser `null`).

## 2. Creación e inicialización

Formas comunes de crear e inicializar arrays:

Inicializar

```
int[] numeros = new int[5]; // [0,0,0,0,0]
```

```
String[] nombres = new String[3]; // [null,null,null]
```

```
int[] pares = {2, 4, 6, 8}; // Inicialización literal
```

~~`var datos = new double[]{1.5, 2.0, 3.14}; // Con 'new' + literal (útil en llamadas a métodos)`~~

Integer Boolean

```
Object[] mezcla = new Object[]{"hola", 42, true}; // Todos son objects
```

Asignación por índice:

```
numeros[0] = 10;
```

```
numeros[1] = 20;
```

```
// ...
```

```
System.out.println(numeros[0]); // 10
```

```
System.out.println(numeros.length); // 5
```

Inicialización por bucle:

```
for (int i = 0; i < numeros.length; i++) {
```

```
numeros[i] = i * i;
```

```
}
```

### 3. Recorrido y uso cotidiano

Formas de recorrer: for clásico, for-each y Streams.

```
int[] v = {3, 1, 4, 1, 5};
```

```
// for clásico
```

```
for (int i = 0; i < v.length; i++) {
```

```
    System.out.println("v[" + i + "] = " + v[i]);
```

```
}
```

```
// for-each
```

```
for (int x : v) {
```

```
    System.out.println(x);
```

```
}
```

```
// Streams (para agregados)
```

```
int suma = java.util.Arrays.stream(v).sum();
```

```
double media = java.util.Arrays.stream(v).average().orElse(0);
```

```
System.out.println("Suma=" + suma + ", media=" + media);
```

### 4. Arrays multidimensionales

Java soporta arrays multidimensionales (matrices) y arrays irregulares (jagged).

```
int[][] m = new int[2][3]; // 2 filas, 3 columnas
```

```
int[][] identidad = {
```

```
    {1, 0, 0},
```

```
    {0, 1, 0},
```

```
    {0, 0, 1}
```

```
};
```

```
// Jagged (filas de longitudes distintas)
```

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

```
jagged[0] = new int[2];
```

```
jagged[1] = new int[5];
```

```
jagged[2] = new int[]{7,8,9};
```

Siempre ponerlas así

Recorrido de una matriz:

```
for (int i = 0; i < m.length; i++) {           // filas
    for (int j = 0; j < m[i].length; j++) {     // columnas
        m[i][j] = i + j;
    }
}
```

## 5. Métodos y utilidades de Arrays

La clase `java.util.Arrays` proporciona utilidades muy útiles:

Métodos de clase:

```
import java.util.Arrays;
```

```
int[] a = {3, 1, 4, 1, 5};
```

```
int[] b = Arrays.copyOf(a, a.length);           // copia superficial
```

```
int[] c = Arrays.copyOfRange(a, 1, 4);           // [1,4,1]
```

```
Arrays.fill(b, 9);                               // [9,9,9,9,9]
```

```
boolean eq = Arrays.equals(a, b);                // false
```

```
String s = Arrays.toString(a);                   // "[3, 1, 4, 1, 5]"
```

```
// Para multidimensionales:
```

```
int[][] mm = {{1,2},{3,4}};
```

```
System.out.println(Arrays.deepToString(mm)); // "[[1, 2], [3, 4]]"
```

```
// Comparación / búsqueda avanzada (Java 9+):
```

```
int cmp = Arrays.compare(new int[]{1,2}, new int[]{1,3}); // < 0
```

```
int mis = Arrays.mismatch(new int[]{1,2,3}, new int[]{1,2,4}); // 2
```

```
Copias eficientes con System.arraycopy:
```

`compareTo`  
Devuelve 0 si los dos  
objetos son iguales.  
`compareToIgnoreCase`

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

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

```
System.arraycopy(origen, 1, destino, 0, 3); // copia 3 elementos desde índice 1
```

## 6. Ordenación

Ordenación natural y personalizada:

```
int[] nums = {5, 2, 9, 1};
```

```
java.util.Arrays.sort(nums);
```

```
// [1,2,5,9]
```

La matriz se organiza, se carga el orden anterior.

Hacer una copia

```
String[] palabras = {"sol", "luna", "estrella"};
```

```
java.util.Arrays.sort(palabras); // alfabético
```

~~// Con comparador (objetos):~~

~~java.util.Arrays.sort(palabras, java.util.Comparator.reverseOrder());~~

~~Ordenación en paralelo (arrays grandes):~~

~~int[] grande = java.util.stream.IntStream.range(0, 1\_000\_000).map(i -> 1\_000\_000 - i).toArray();~~

~~java.util.Arrays.parallelSort(grande);~~

## 7. Búsqueda

Búsqueda lineal vs. binaria (requiere array ordenado).

Búsqueda binaria (datos ordenados)

```
int[] datos = {1, 3, 4, 8, 10, 15};
```

n búsquedas:  $\log_2(n)$

```
int idx = java.util.Arrays.binarySearch(datos, 8); // devuelve índice ( $\geq 0$ ) o (-
```

(puntoInserción)-1) si no está

Si es negativo, no está y es la posición donde el dato estaría con el símbolo negativo.

$2^x = 100$

```
if (idx  $\geq$  0) System.out.println("Encontrado en " + idx);
```

En 7 búsquedas se encuentra el número.

## 8. Buenas prácticas y errores comunes

- Usa constantes para tamaños y evita números mágicos.
- Comprueba límites de índice: lanzará `ArrayIndexOutOfBoundsException` si te sales.
- Prefiere `Arrays.stream` para agregados y legibilidad.
- Cuando necesites tamaño dinámico, considera `ArrayList` u otras colecciones.
- Para arrays de objetos, considera null-safety (Optional, validaciones).

## 9. Ejemplos completos

1) Media, mínimo y máximo de un array:

```
import java.util.Arrays;

public class EstadisticasArray {

    public static void main(String[] args) {

        int[] v = {7, 2, 9, 4, 5};

        double media = Arrays.stream(v).average().orElse(0);

        int min = Arrays.stream(v).min().orElse(Integer.MIN_VALUE);

        int max = Arrays.stream(v).max().orElse(Integer.MAX_VALUE);

        System.out.printf("Media=%.2f, Min=%d, Max=%d\n", media, min, max);

    }

}
```

2) Rotación de elementos a la derecha k posiciones:

```
public class RotarArray {

    public static void rotarDerecha(int[] a, int k) {

        k = k % a.length;

        if (k == 0) return;

        reverse(a, 0, a.length - 1);

        reverse(a, 0, k - 1);

        reverse(a, k, a.length - 1);

    }

    private static void reverse(int[] a, int i, int j) {

        while (i < j) {

            int tmp = a[i]; a[i] = a[j]; a[j] = tmp;

            i++; j--;

        }

    }

    public static void main(String[] args) {
```

```
int[] a = {1,2,3,4,5};  
rotarDerecha(a, 2); // -> {4,5,1,2,3}  
System.out.println(java.util.Arrays.toString(a));  
}  
}
```

### 3) Ordenación de objetos con Comparator:

```
import java.util.Arrays;  
import java.util.Comparator;  
  
class Alumno {  
    String nombre;  
    int nota;  
    Alumno(String n, int nota){ this.nombre=n; this.nota=nota; }  
    public String toString(){ return nombre + "(" + nota + ")"; }  
}  
  
public class OrdenarAlumnos {  
    public static void main(String[] args) {  
        Alumno[] clase = {  
            new Alumno("Ana", 9),  
            new Alumno("Luis", 7),  
            new Alumno("Marta", 10)  
        };  
        Arrays.sort(clase, Comparator.comparingInt(a -> a.nota)); // asc  
        System.out.println(Arrays.toString(clase));  
        Arrays.sort(clase, Comparator.comparingInt((Alumno a) -> a.nota).reversed());  
        // desc  
        System.out.println(Arrays.toString(clase));  
    }  
}
```

## 10. Ejercicios propuestos

### Nivel 1 — Fundamentos

- Crea un array de 10 enteros, inicialízalo con los valores 1..10 e imprime sus elementos y su suma.
- Dado un array de doubles, calcula la media y cuenta cuántos elementos están por encima de la media.
- Crea un array de Strings con 5 nombres e imprime solo los que empiezan por vocal.

### Nivel 2 — Transformaciones y utilidades

- Escribe una función que invierta un array in-place (sin array auxiliar).
- Usa `System.arraycopy` para copiar una sub-sección de un array en otro.
- Dado un array de enteros, elimina duplicados dejando solo la primera aparición (puedes crear un nuevo array con el resultado).

### Nivel 3 — Multidimensionales y algoritmia

- Genera una matriz  $n \times n$  con 1s en la diagonal y 0s en el resto (matriz identidad).
- Calcula la suma por filas y por columnas de una matriz e imprime ambos vectores resultado.
- Implementa la búsqueda binaria recursiva sobre un array ordenado (devuelve índice o -1).

### Nivel 4 — Objetos y comparadores

- Define una clase `Producto(nombre, precio)`. Crea un array y ordénalo por precio ascendente y descendente.
- Ordena un array de Strings por longitud y, a igualdad, alfabéticamente.
- Dado un array de `Persona(nombre, edad)`, usa `Arrays.mismatch` para encontrar el primer índice que difiere respecto a otro array de `Persona` con el mismo tamaño (define `equals` de forma adecuada).

### Nivel 5 — Rendimiento y casos prácticos

- Crea un array de  $10^7$  enteros aleatorios y compara el tiempo de `Arrays.sort` vs `Arrays.parallelSort`.
- Implementa una rotación circular  $k$  a la derecha usando el método de tres reversos y mide tiempo para distintos  $k$ .
- Convierte una lista (`ArrayList<Integer>`) a array primitivo `int[]` de forma eficiente usando streams.

## 11. Apéndice: Conversión entre arrays y colecciones

```
String[] arr = {"a", "b", "c"};
```

```
java.util.List<String> lista = java.util.Arrays.asList(arr); // ¡tamaño fijo!
```

```
java.util.ArrayList<String> mutable = new java.util.ArrayList<>(lista); // ya es mutable
```

```
String[] deNuevo = mutable.toArray(new String[0]); // a array
```

## 12. Apéndice: Validaciones y defensas

- Comprueba null antes de acceder a arrays de objetos.
- Valida índices en métodos públicos: lanza `IllegalArgumentException` si procede.
- Para exponer datos internos, considera devolver copias (defensive copy) en lugar del array real.