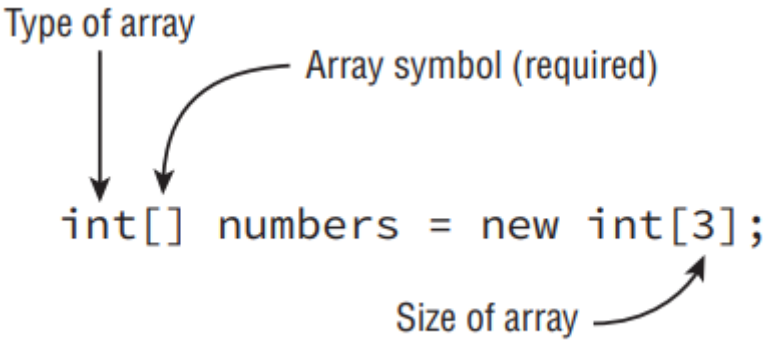
Arrays en Java:

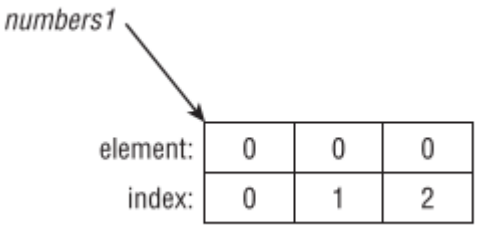
1. Introducción

Un array o vector es un tipo de datos complejo que almacena un número determinado de elementos del mismo tipo (primitivo o no) en posiciones sucesivas de memoria. Se corresponde con el concepto de lista ordenada, que puede contener elementos duplicados y permite operaciones de: creación/inicialización, búsqueda, ordenación, lectura, escritura, entre otras.

Estructura básica de un array en Java



Este array **numbers** almacena hasta un máximo de tres enteros (int) mediante las posiciones en memoria 0, 1 y 2. Puesto que el tipo base de elementos que almacena este array es un tipo primitivo (int), cada posición del array se inicializa con su valor por defecto (0).
Si por el contrario, fuesen Objetos, la inicialización sería a **null**.



También podemos inicializar el array durante su creación y en ese caso no necesitamos establecer el tamaño directamente. Las sentencias siguientes producen el mismo resultado:

Declaración + inicialización

```
int[] numbers = new int[] {1, 3, 5, 7, 9};
```

Declaración + inicialización

```
int[] numbers =  {1, 3, 5, 7, 9};
```

Declaración + inicialización (corchetes desplazados en declaración)

```
int numbers[] = new int[] {1, 3, 5, 7, 9};
```

Declaración estableciendo el tamaño máximo y asignación posterior de valores

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

numbers[0]=1;
numbers[1]=3;
numbers[2]=5;
numbers[3]=7;
numbers[4]=9;
```

Además de con tipos primitivos, también podemos construir arrays con Clases de la API de Java o clases propias. Por ejemplo:

```
String[] nombres = {"Juan", "Manuel", "Antonio", "Juan"};

System.out.println(nombres[0].equals(nombres[3]));
```

Donde creamos un array de 4 cadenas y posteriormente llamamos a los métodos que tienen cada una de las variables. En este caso podemos llamar al .equals porque contiene un array de String (que extienden de Object).

Si queremos conocer el tamaño de un array podemos usar la propiedad **length**, que nos devuelve un entero int con el número de elementos que puede almacenar el array. **No** se accede a esta propiedad como la llamada a un método (con paréntesis).

2. Acceso indexado:

el acceso a los elementos de un array se realiza utilizando el nombre del array y la posición del elemento al que queremos acceder entre corchetes:

```
nombres[0]
```

3. Actualización de una posición:

Para cambiar el valor de una posición del array utilizamos el operador de asignación =; la única diferencia con una asignación a variable es que en un array debemos indicar qué posición queremos actualizar. Por ejemplo, en este caso solo actualiza el primer nombre del array:

```
nombres[0] = "Ernesto";
```

4. Recorrer un array:

Para recorrer un array podemos hacer uso del acceso indexado, o bien, de la estructura for reducida.

for - Acceso indexado	for reducido

`for (int i=0; i< nombres.length; i++) {`
`  System.out.println(nombres[i]);`
`}`
	`for(String nombre: nombres) {`
	`  System.out.println(nombre)`
	`}`

5. Ordenar un array:

Para ordenar los elementos de un array (orden lexicográfico) podemos hacer uso del método estático contenido en la clase java.util.Arrays ⇒ Arrays.sort.

```
Arrays.sort(nombres);
```

Tras la llamada a este método el array quedará ordenado crecientemente.
Búsqueda de un elemento:

Para buscar un elemento en un array podemos hacer uso de las estructuras habituales, por ejemplo, para ver si un elemento se encuentra entre los valores de un array y devolver su posición:

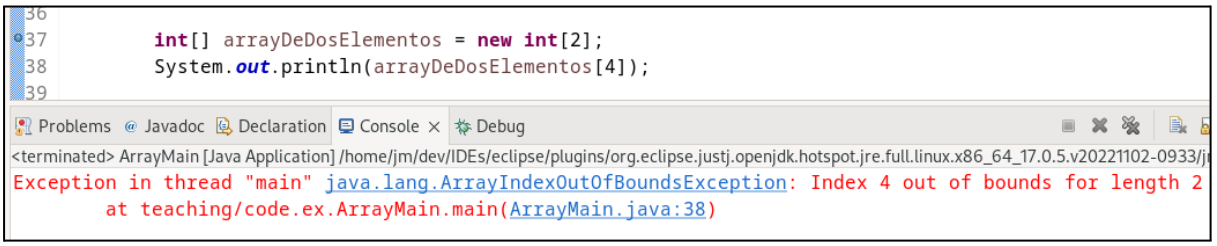
```
public int buscarElemento(int[] numbers, int elemento) {

    int i =0, posicion =-1;

    while (i<numbers.length && posicion==-1) {
        posicion = elemento == numbers[i]? i : -1;
        i++;
    }

    return posicion;
}
```

También podemos hacer uso de la función **Arrays.binarySearch** que está sobrecargada, pero requiere que el array esté ordenado previamente de forma creciente.

6. Excepciones:

Existen dos excepciones habituales con el manejo de arrays. Por un lado, si intentamos acceder a un elemento cuya posición exceda el tamaño del array, nuestro programa lanzará un java.lang.ArrayIndexOutOfBoundsException

```
int[] arrayDeDosElementos = new int[2];
System.out.println(arrayDeDosElementos[4]);
```



Para evitar este error tenemos que hacer uso de la propiedad length que nos da el tamaño del array. Por ejemplo:

```
int[] arrayDeDosElementos = new int[2];
int ultimaPosicion = arrayDeDosElementos.length-1;
System.out.println(arrayDeDosElementos[ultimaPosicion]);
```

La otra excepción habitual es tratar de ejecutar un método en un elemento del array que no se ha inicializado, la conocida java.lang.NullPointerException

```
String[] nombres = new String[3];
System.out.println(nombres[0].length());
```

```
Exception in thread "main" java.lang.NullPointerException: Cannot
        at teaching/code.ex.ArrayMain.main(ArrayMain.java:41)
```

Para evitarlo, primero tenemos que asignar un valor a ese elemento del array.