1. Introducción

Ante la limitación de las variables que solo pueden almacenar un valor, aparecen los arrays, también conocidos como tablas o vectores en castellano. En este caso, los arrays serán conjuntos de variables, siempre del **mismo tipo**, en las que almacenar valores distintos.

En el caso de los arrays, cada valor irá identificado por un **índice**, que siempre **empezará por 0**. Tenemos que tener cuidado con dichos índices, pues referenciar un valor fuera de rango de dicho array dará una excepción en tiempo de ejecución de nuestro programa, por lo que terminará de manera súbita.

2. Uso de arrays

Para declarar un array en Java, tendremos que decidir el **tipo** y el **tamaño** de dicho array. Con esta información, podremos **declarar** un array de cualquiera de las siguientes formas:

```
int[] numeros;    //Declaración del array numeros
int numerosB[];
```

A continuación, para **inicializarlo** tendremos que reservar memoria para tantos elementos como vaya a tener dicho array, utilizando el operador de construcción *new*:

```
numeros = new int[5];   //Array para 5 posiciones
```

Con esta última sentencia, he creado una estructura en la que podré almacenar 5 números enteros. Lo podemos visualizar como un vector o tabla de 5 posiciones:

Por último, para **asignar valores**, iremos accediendo a través de los índices a cada una de sus posiciones. Recordemos que los índices empiezan en 0, y por tanto el índice máximo será longitud-1:

```
numeros[0] = 6;
numeros[1] = -3;
numeros[2] = 3;
numeros[3] = 21;
numeros[4] = 47;
//numeros[5] = 9; //Error: está fuera de rango
```

Siguiendo con el símil de la tabla o vector anterior, ahora tendrá este contenido:

6	-3	3	21	47

Otra opción de asignación de valores sería directamente a la hora de su declaración, en el que la parte de asignación de valores e inicialización se convierten en la misma:

```
double[] pesos = { 56.9, 87, 69.1 };
char[] vocales = { 'a', 'e', 'i', 'o', 'u' };
```

Una vez creado un array, **no se puede modificar su tamaño**. Si tuviéramos necesidad de cambiar el tamaño de un array, tendríamos que crear uno nuevo con el tamaño final, y copiar los elementos de uno a otro.

Hay que tener en cuenta a la hora de utilizar arrays que para empezar a utilizarlos tengo que inicializarlos (de ahí el operador *new*). Si no se inicializan, también es posible utilizar variables de tipo array que hagan **referencia** al array inicializado que ya tenía. Veamos este ejemplo:

```
int[] edades = new int[3];
int[] copia = edades;
edades[0] = 20;
copia[0] = 25; // Sobrescribe edades[0]: es la misma referencia
System.out.println(edades[0]); // Se imprime 25
```

2.1. Recorrer un array

Para recorrer cada posición de un array, podemos utilizar cualquiera de los bucles conocidos hasta ahora (for, while, do-while). Para obtener la longitud de un array, accedemos a su propiedad .length:

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

O con while:

```
int indice = 0;
while(indice < numeros.length) {
      System.out.println(numeros[indice]);
      indice++;
}
```

También existe otra forma de iteración que solamente sirve para elementos iterables (como los arrays con los que estamos trabajando). Se le conoce como **foreach** y en cada iteración, la variable indicada irá tomando cada uno de los valores de las posiciones del array.

Su declaración es la siguiente:

```
for (tipo variable: array) { ... }
```

Así, en el ejemplo que estamos utilizando:

```
for (int n: numeros) {
      System.out.println(n);
}
```

3. Operaciones con arrays

Para obtener la **longitud** de un array, utilizamos su propiedad length. Al ser una propiedad y no un método, no lleva los paréntesis al final:

```
int longitud = numeros.length;
System.out.println("La longitud del array es " + longitud);      //5
```

Para utilizar una serie de operaciones con arrays, importaremos y utilizaremos los métodos que proporciona java.util.Arrays. Gracias a ella, podremos hacer una serie de operaciones como:

- **toString()**: Mostrar el contenido de un array:

```
System.out.println(numeros);  //Muestra tipo y memoria
System.out.println(Arrays.toString(numeros));  //Contenido
```

- **fill()**: Inicializar con un valor distinto a los valores por defecto:

```
Arrays.fill(numeros, 10); //Todos los valores a 10
Arrays.fill(numeros, 1, 3, 5);//Posiciones 1 y 2 con valor 5
```

- **sort()**: Ordenar el array. Tenemos que tener en cuenta que los algoritmos de ordenación son bastante complejos y suelen requerir bastante tiempo de computación. Por tanto, tendremos que decidir si en realidad queremos ordenar un array antes de hacerlo.

```
int[] nums = { 6, 5, 90, -1, 0 };
Arrays.sort(nums);      //[-1, 0, 5, 6, 90]
```

- **binarySearch():** Buscar en un array. Podemos utilizar algoritmos de búsqueda de la clase Arrays siempre que el **array esté ordenado**. De no estarlo, tendremos que recorrerlo manualmente buscando el elemento que queramos. Si está ordenado, podemos utilizar binarySearch, que devolverá la posición en la que se encontró el elemento, o un número negativo en caso de no encontrarlo. Este número negativo representa la posición menos uno en la que deberíamos introducir el elemento buscado para que el array siga ordenado:

```
System.out.println(Arrays.binarySearch(nums, 6));     //3
System.out.println(Arrays.binarySearch(nums, 4));     //-3
(-(insertion point) - 1)
```

- **copyOf():** Copiar. El método copyOf de la clase Arrays devolverá una copia del array de origen con la longitud especificada. Si la longitud del nuevo array es menor que la original, solo se copian los valores que caben. Si es mayor, se inicializan a su valor por defecto. La estructura sería `copyOf(tipo origen[], int longitud)`. Veamos un ejemplo:

```
int[] original = {1, 2, 3, 4, 5};
int[] a, b;
a = Arrays.copyOf(original, 3);     //a = [1, 2, 3]
b = Arrays.copyOf(original, 7);     //b = [1, 2, 3, 4, 5, 0, 0]
```

- **copyOfRange():** Copiar un rango. Indicamos el rango a copiar (el índice inicial sí se inserta, el índice final no), y devolviendo el array resultante. La estructura sería `copyOfRange(tipo origen[], int desde, int hasta)`:

```
int[] c = Arrays.copyOfRange(original, 2, 5);  //c = [3, 4, 5]
```

- **System.arraycopy():** También podemos utilizar arraycopy de la clase System. La novedad en este caso es que no devuelve un array, sino que se lo tenemos que pasar como parámetro. Tiene la siguiente estructura: `arraycopy(tipo origen[], int posOrigen, tipo destino[], int posDestino, int longitud)`. Copiará en destino desde el índice posDestino tantos datos de origen como diga longitud:

```
int[] orig = { 10, 20, 30, 40, 50, 60, 70, 80, 90, 100 };
int[] d = new int[7];
System.arraycopy(orig, 6, d, 2, 4); // d = [0, 0, 70, 80, 90,
100, 0]
//Desde la posición 6 de orig, copia 4 elementos en d desde su
posición 2
```

- **.equals():** Comparar la igualdad de contenido. Hay que tener en cuenta, que al igual que pasa con String y otras clases, en los arrays no podemos utilizar los operadores lógicos ==, >, !=, <=, etc. Porque comparan direcciones en memoria, no contenido. Por tanto:ç

```
int[] a1 = {1, 2 ,3};
int[] a2 = {1, 2, 3};
int[] a3 = a1;
System.out.println(a1 == a2); //false
System.out.println(a1 == a3); //true
System.out.println(Arrays.equals(a1, a2)); //true
```

4. Arrays n-dimensionales

Podemos tener arrays de varias dimensiones, es decir, en el caso de que sea bidimensional, estaremos ante un array de arrays.

```
int[][] matriz = new int[3][5];

//Relleno las posiciones del array
for (int i = 0; i < matriz.length; i++) {
	for (int j = 0; j < matriz[i].length; j++) {
		matriz[i][j] = (int) (Math.random() * 10);
	}
}

//Recorro con for anidados:
for (int i = 0; i < matriz.length; i++) {
	for (int j = 0; j < matriz[i].length; j++) {
		System.out.print(matriz[i][j] + " ");
	}
	System.out.println("");
}

//Otra opción: for-each
for (int[] fila : matriz) {
	for (int posicion : fila) {
		System.out.print(posicion + " ");
	}
	System.out.println("");
}

//Arrays.deepToString para imprimir todos los arrays anidados
System.out.println(Arrays.deepToString(matriz));
```

No todos los subarrays tienen que tener el mismo tamaño, puedo declarar cada subarray de manera distinta:

```
int[][] irregular =new int[3][];
for(int i =0; i < irregular.length; i++){
	irregular[i] = new int[i + 3];  //tamaño i+3 de cada subarray
}
System.out.println(Arrays.deepToString(irregular));
//[[0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0, 0]]
```

Pueden tener tantas dimensiones como queramos.

```
double[][][][] cuatridimensional = new double[4][3][2][5];
cuatridimensional[2][1][0][4] = 5;
```