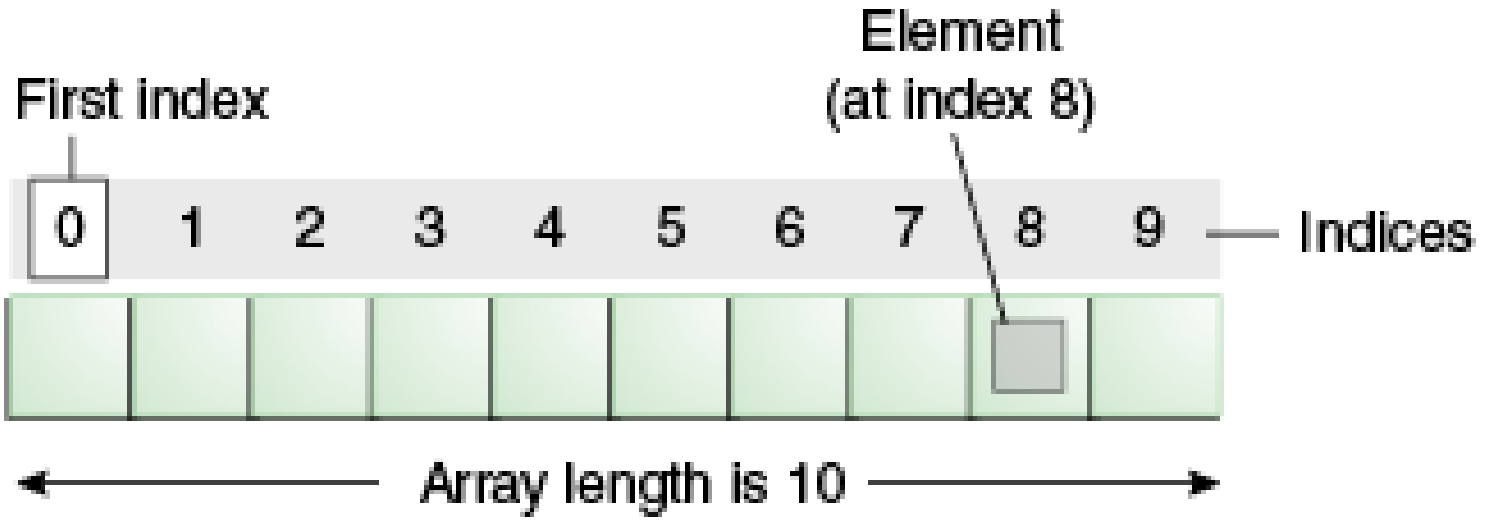
Unidad 4. Cadenas de caracteres y arrays



ARRAYS EN JAVA

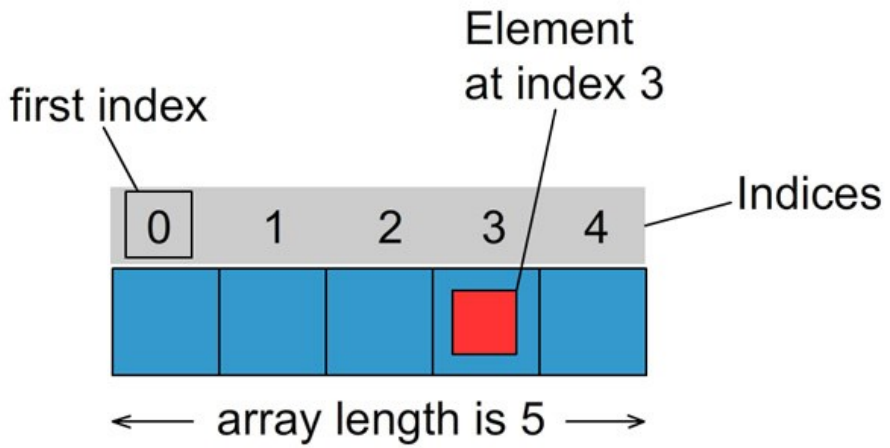
José L. Berenguel

Tabla de Contenidos

1. Creación e inicialización de arrays.
2. Dimensionado de arrays.
3. Funcionamiento de las referencias y la memoria.
4. Acceso y modificación de arrays.
5. Recorrido de arrays con for-each.
6. Arrays multidimensionales.
7. Arrays multidimensionales irregulares.
8. Array como argumento de un método.
9. Array como tipo de devolución de un método.
10. La clase java.util.Arrays del JDK.

Arrays: Introducción

- Una **variable** almacena **un único valor** simultáneamente.
- En ocasiones es necesario poder **almacenar más de un valor en una variable**, esto es posible con los **arrays** o tablas.
- Podemos decir que un **array es un conjunto de variables del mismo tipo** que ocupan posiciones contiguas de memoria.
- Cada elemento se accede a través de un índice, cuya numeración comienza en 0.



Arrays: creación e inicialización

- **Array**: Es un **objeto** en el que se puede almacenar un **conjunto de datos de un mismo tipo**.
- La declaración de un array es similar al de una variable añadiendo [ ] por cada dimensión del array.
- **Declaración** de una variable de tipo array:

```
tipo [] variableArray;
tipo variableArray[];
```

```
int a[];
int []b;
```

- **Dimensionado** o **reserva de memoria** de un array:

```
variableArray=new tipo[tamaño];
```

```
a = new int[10];
b = new int[10];
```

Arrays: dimensionado

- Cuando un array se dimensiona **todos sus elementos se inicializan a su valor por defecto según el tipo** (0 para tipos numéricos y *false* para *boolean*).
- Si es un **array de objetos**, será el constructor del tipo objeto el que inicializara cada uno de los elementos.

```
int [] k;
String [] p;
char cads[];
k=new int[5];
cads=new char[10];
String [] noms=new String[10];
```

```
int a[] = new int[10];
int []b = new int[10];
```

- **Declaración**, **dimensionado** e **inicialización** de un array:

```
int [] nums={10,20,30,40};
```

Arrays: referencias y memoria

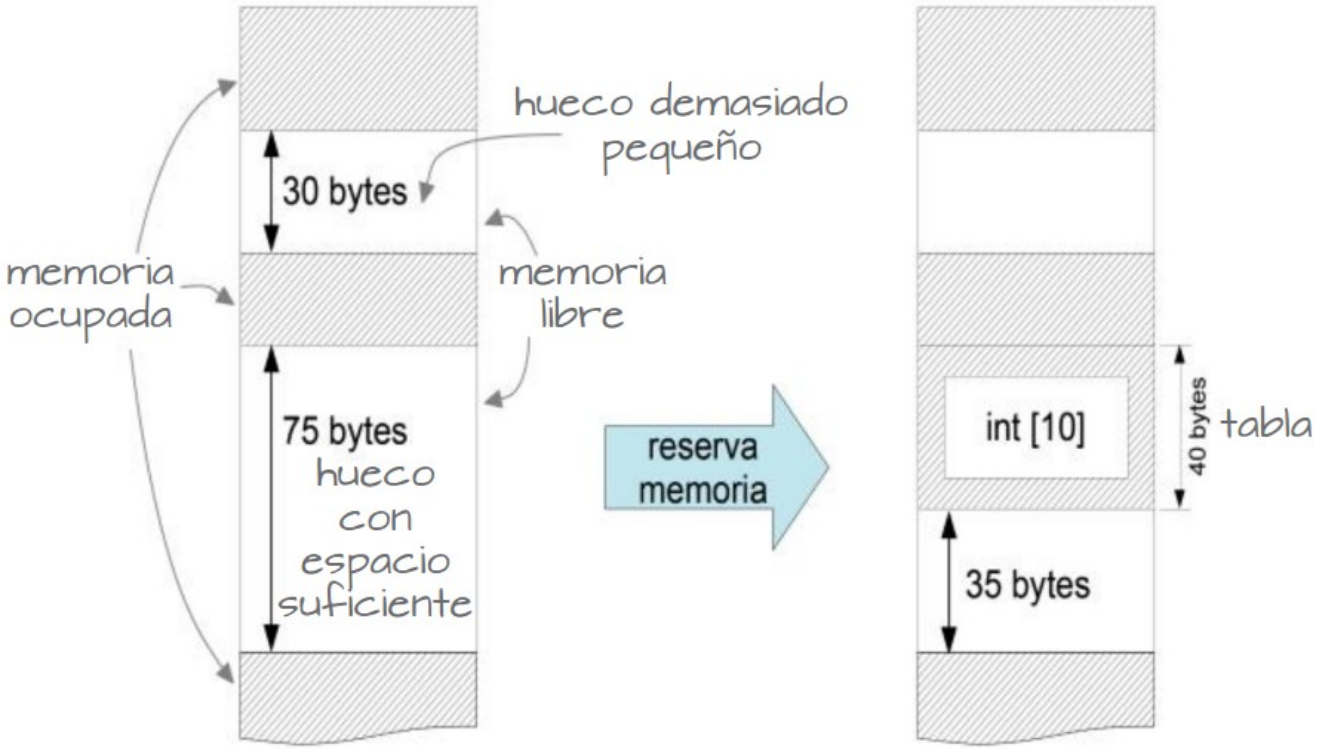
- El operador ***new*** es el encargado de **reservar espacio en memoria** para los tipos objeto.
- Devuelve una referencia (lugar en memoria) que se almacena en la variable identificadora.

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

Ejemplo: vamos a crear un array de 10 elementos de tipo *int*
¿Cuánto espacio en memoria necesitamos?

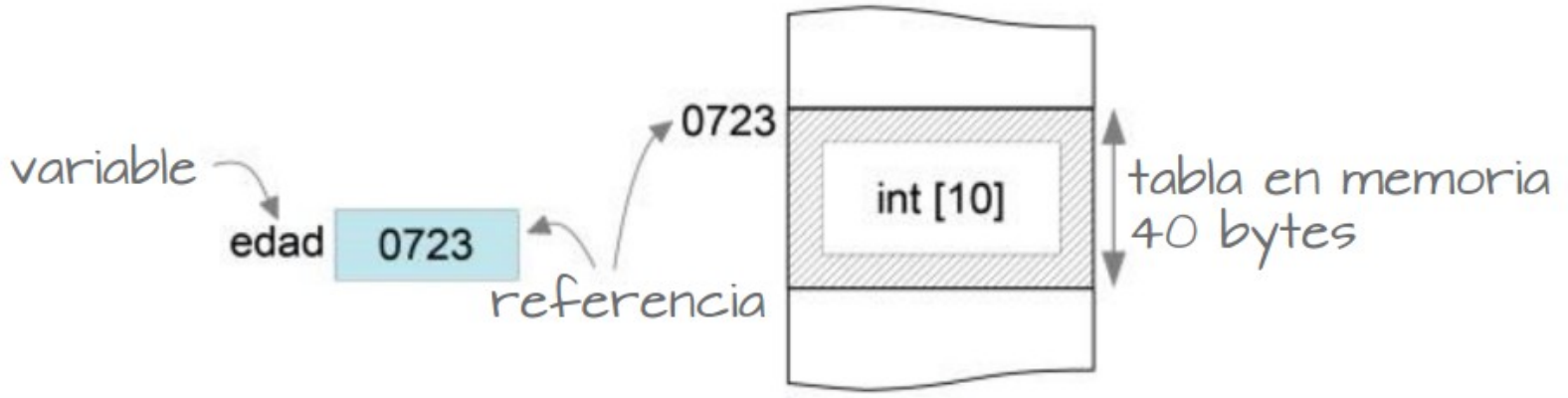
Arrays: referencias y memoria

- **Tamaño en memoria** = n.º elementos x tamaño tipo int = 10 x 4 bytes = **40 bytes**.
- El sistema operativo le facilita al programa un **espacio contiguo de memoria** de 40 bytes.



Arrays: referencias y memoria

- La referencia al comienzo de la tabla es almacenada en la variable del array, en nuestro ejemplo ***edad***.
- Recordemos que las variables declaradas tienen su espacio reservado en el contexto del programa, y aquellas que son de tipo objeto almacenarán las referencias de memoria cuando los objetos sean construidos con *new*.



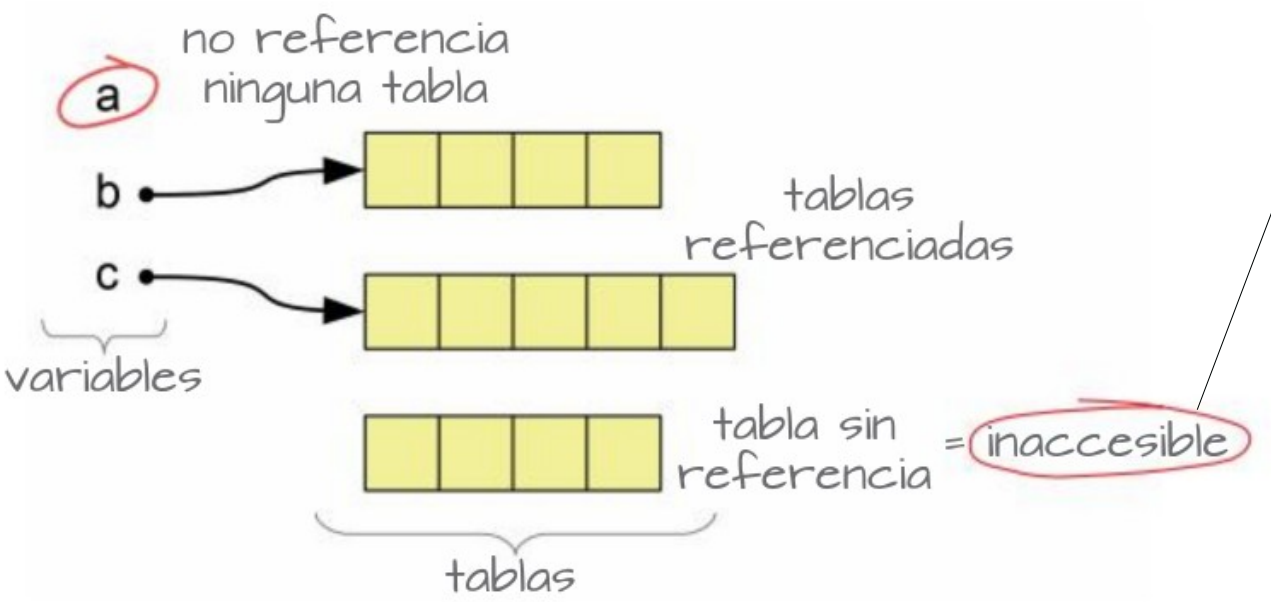

```
int t[] = new int[10];
System.out.println(t);
```

Este código no muestra todos los elementos del array.
Mostrará por consola la referencia guardad en la variable t.

Arrays: referencias y memoria

- Analicemos el siguiente código para ver cómo funcionarían las referencias.

```
int a[], b[], c[]; //variables
b = new int[4]; //tabla de cuatro enteros accesible mediante la variable b
c = new int[5]; //tabla de cinco enteros accesible mediante la variable c
new int[3]; //creamos una tabla cuya referencia no se asigna a ninguna variable
```

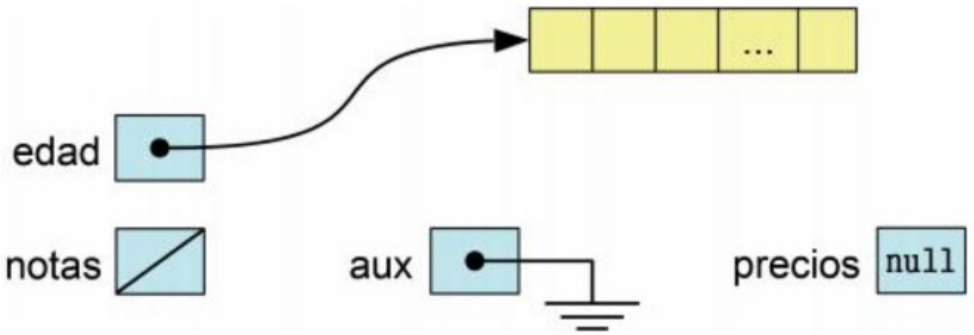


La memoria no referenciada es liberada por el recolector de basura o ***Garbage Collector***

Arrays: referencias y memoria

- En el ejemplo anterior, a no referenciaba a ningún lugar. ¿Cuál es su contenido? ¿0? ¿false?
- La referencia ***null*** es un literal que se asigna a los tipos objeto que no tienen referencias de memoria.
- Si intentamos acceder a algún método o posición de memoria a través de una variable no inicializada se lanzará la excepción ***NullPointerException***.

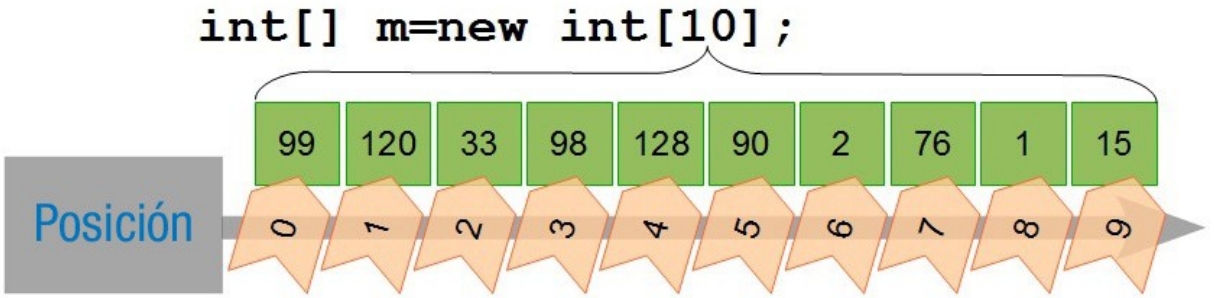
```
int t1[], t2[]; //variables de tipo tabla entera
t1 = new int[100]; //t1 referencia una tabla de 100 elementos
t2 = t1; //ahora t2 también referencia la misma tabla
t1 = null; //anulamos t1: no referencia nada
         //la tabla sigue siendo accesible desde t2
t2 = null; //anulamos t2: tampoco hace referencia a nada
//la tabla es inaccesible: el recolector de basura se encargará de ella
```



Arrays: acceso y modificación

- ***variableArray[índice]***: accede al elemento del array en la posición índice (entre [0,tamaño-1]).
- En caso de acceso a una posición fuera del rango del array se lanza la excepción ***ArrayIndexOutOfBoundsException***.
- ***variableArray.length***: contiene el tamaño del array.

```
int [] tablaMultiplicarDos=new int[10];
for(int i=0;i<tablaMultiplicadorDos.length;i++){
     tablaMultiplicarDos[i]=i*2;
}
```



Recorrido de arrays con for-each

- Facilita el recorrido de arrays y colecciones.
- Se emplea cuando se necesita acceder **solo para leer todos los elementos del array**.
- Palabra reservada: ***for*** (es una variante de este bucle):
  - ***tipo variable***: declaración de variable auxiliar del mismo tipo que el array que irá tomando cada valor del array de forma secuencial.
  - ***variableArray***: array que se va a recorrer.

```
for(tipo variable:variableArray){
    //instrucciones
}
```

Recorrido de arrays con for-each

- Comparación de recorrido de un array con ***for*** y ***for-each***.

```
//declaración e inicialización del array
int [] numeros={4,6,30,15};


//Mostrar los datos del array con un for
for(int i=0;i<numeros.length;i++){
    System.out.println(numeros[i]);
}


//Mostrar los datos del array con for-each
for(int n:numeros){
    System.out.println(n);
}
```

Arrays multidimensionales

- Para indicar que un array tiene más dimensiones, **se añaden más corchetes** a su declaración.

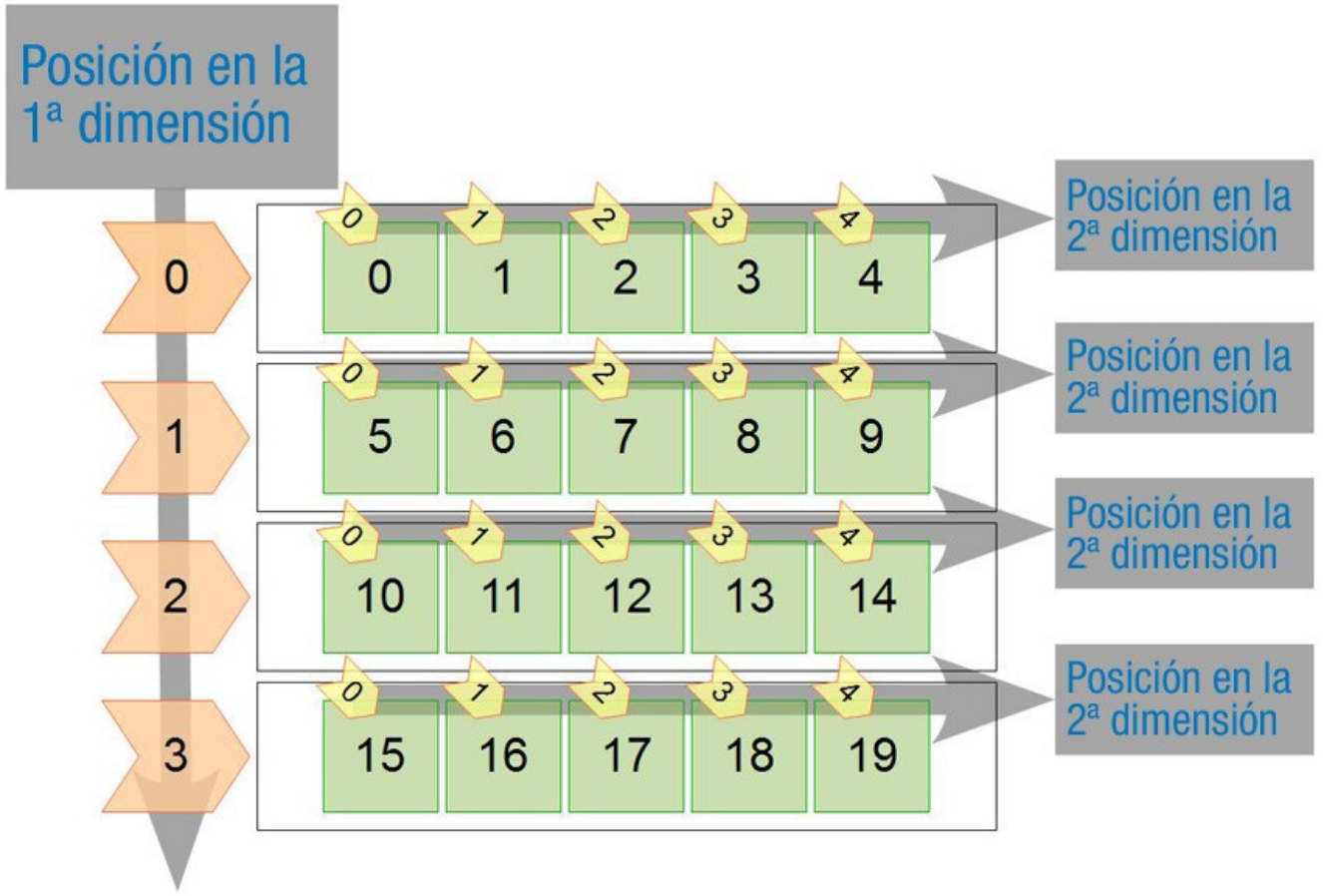
```
int [][] variableArray;
```

- Lo explicado para arrays de una dimensión es aplicable.

```
int [][] k = new int[3][5]; //declaración y dimensionado

k[1][3]=25; //acceder a una posición

k[3][2]=0; //Fuera de rango: excepción IndexOutOfBoundsException

k.length; //tamaño de la primera dimensión

k[0].length; //tamaño de la fila 0

int v[][] = {{1,2},{3,4}}; //declaración, inicialización y dimensionado
```

Arrays multidimensionales: recorrido

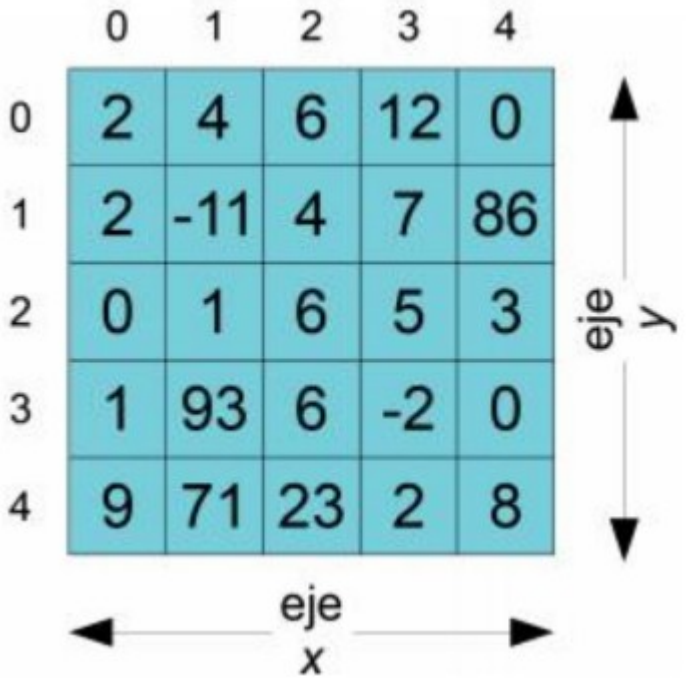
- Necesitamos un índice distinto para recorrer cada dimensión.



Arrays multidimensionales: recorrido

- Se utilizan **tantos bucles *for* anidados como dimensiones tenga el array**:

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

// --> Se rellena el array con valores

for(int i=0;i<k.length;++i)
    for(int j=0;j<k[0].length;++j)
        k[i][j]=i+j;
```



- Recorrido con ***for-each***:

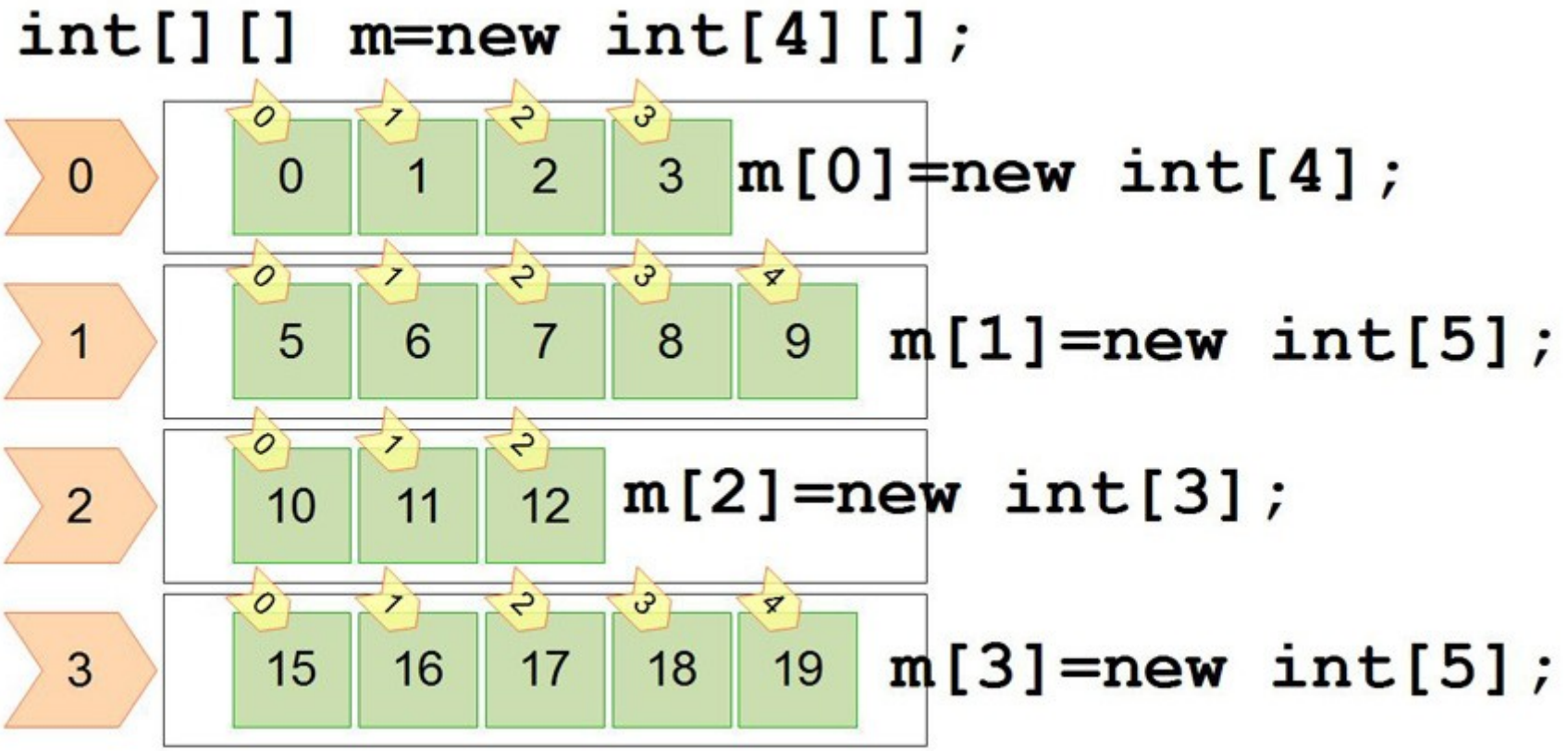
```
// Se recorre por filas
for(int []num:matriz){
    for(int elemento:num){
        System.out.print(elemento+" ");
    }
    System.out.println("\n");
}
```

Arrays multidimensionales irregulares

- Array en el que el número de elementos de la segunda y sucesivas dimensiones es distinto entre si.
- Equivale a tener un array de arrays donde cada posición del array contiene otro array.
  - Para crearlo, se debe **inicializar por separado cada fila o dimensión** con el número de elementos que contendrá.
  - También se puede crear e inicializar de manera explícita.

```
int [][] p = new int[2][]; //array con dos filas y número de columnas variable
p[0] = new int[3]; //la fila 0 tiene 3 elementos
p[1] = new int[5]; //la fila 1 tiene 5 elementos
//inicialización de valores
for(int i=0;i<p.length;++i){
    for(int j=0;j<p[i].length;++j){ //Importante la comprobación
        p[i][j]=i+j;
    }
}
```

Arrays multidimensionales irregulares



Array como argumento de un método

- **Cuidado**. El método puede modificar los valores contenidos en el array.

```
public class SumaArray{
     public static void main(String [] args){
          int suma;
          int [] numeros={3,4,5,10};
          //llamada al método
          suma=sumarArray(numeros);
          //mostramos el resultado
          System.out.println("La suma del vector es "+suma);
     }

     //Definición del método sumarArray
     public static int sumarArray(int [] vector){
          int total=0;
          //mejor si se emplea for-each
          for(int i=0;i<vector.length;i++){
               total+=vector[i];
          }
          return total;
     }
}
```

Array como tipo de devolución de un método

- El siguiente método devuelve un array con los siguientes cinco números enteros al recibido por parámetro:

```
int [] getNumeros(int n){
     int [] numeros=new int[5];
     for(int i=0;i<numeros.length;i++){
          numeros[i]=n+i+1;
     }
     return numeros
}
```

La clase Arrays del JDK

- La API de Java nos provee de una clase de utilidad para hacer operaciones sobre arrays:
  - ***java.util.Arrays***.
- Los métodos de esta clase son *static* por lo que no hay que construir un objeto para llamarlos, se invocan con ***Arrays.metodo()***.
- Lo habitual es que estos métodos reciban por parámetro el/los array/s con los que van a operar.
- Debemos comprobar en la documentación del método si se trabaja sobre una copia del array o directamente sobre los datos del array pasados por parámetro (en ese caso el array original será modificado.

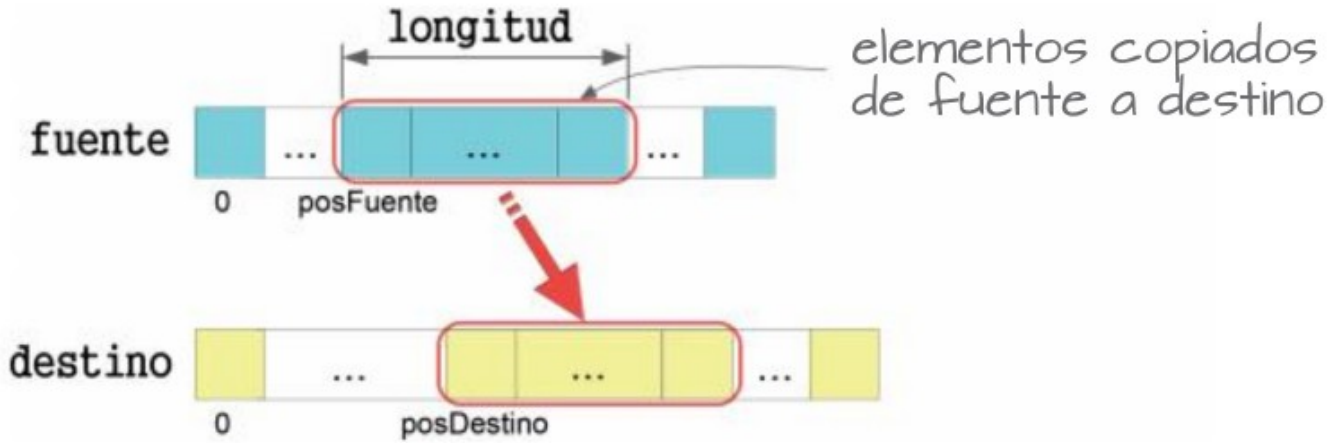
https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Arrays.html

La clase Arrays del JDK

- Alguno de los métodos de java.util.Arrays (todos los métodos están sobrecargados para soportar diferentes tipos):
  - ***void fill(array, valor)***. Inicializa todos los elementos del array a *valor*.
  - ***void fill(array, desde, hasta, valor)***. Inicializa los elementos [desde,hasta) del array a *valor*.
  - ***void sort(array)***. Ordena los elementos del array en orden numérico creciente.
  - ***int binarySearch(array, clave)***. Busca en un array ordenado la posición del elemento cuyo valor coincida con *clave*.
  - ***tipo[] copyOf(array, longitud)***. Devuelve una copia de array. Si longitud es menor que el array, se devolverán los elementos que caben en el nuevo array, si es mayor, los nuevos elementos se inicializan al valor por defecto.
  - ***tipo[] copyOfRange(origen, desde, hasta)***. Devuelve un nuevo array que contiene los elementos del array *origen[desde, hasta)*.

La clase Arrays del JDK

- Además, de los métodos descritos anteriormente, disponemos del método ***System.arraycopy()***.
  - ***void arraycopy(Object origen, int posOrigen, Object destino, int posDestino, int longitud)***.
- Este método **copia elementos consecutivos de un array en otro** (no crea un nuevo array). Por tanto, el array origen y destino deben estar creados.



Unidad 4. Cadenas de caracteres y arrays

DUDAS Y PREGUNTAS