IES DR LLUÍS SIMARRO

ENSENYAMENT PÚBLIC

XÀTIVA

CFGS Desarrollo de aplicaciones multiplataforma

Módulo profesional: Programación

GENERALITAT VALENCIANA

Conselleria d'Educació, Investigació, Cultura i Esport

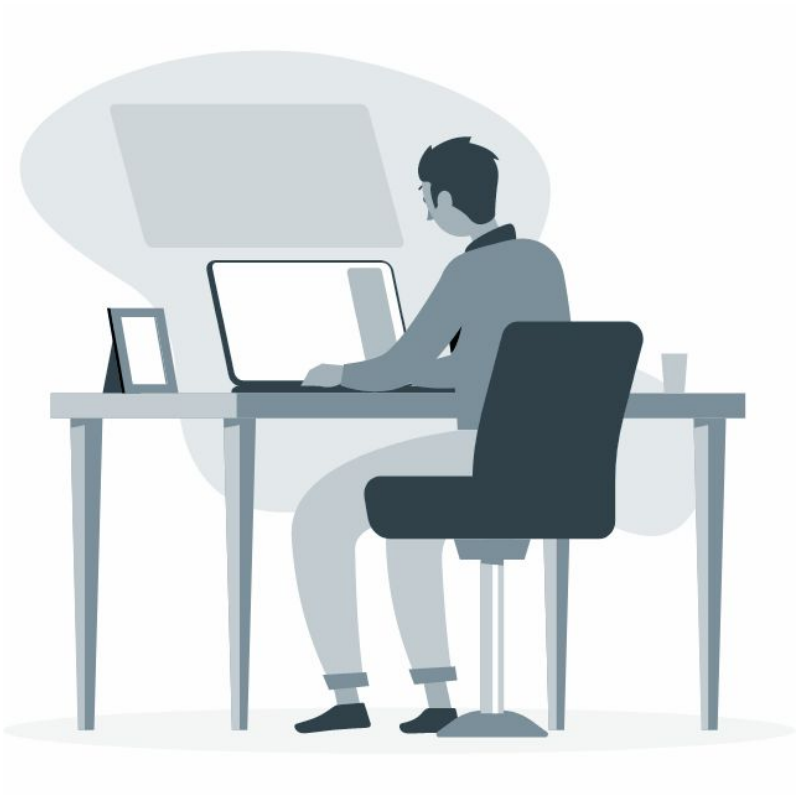
GOBIERNO DE ESPAÑA

MINISTERIO DE EDUCACIÓN Y FORMACIÓN PROFESIONAL

Unió Europea

Fons Social Europeu

L'FSE inverteix en el teu futur



Material elaborado por:

Edu Torregrosa Llácer

Modificado por Joan Carrillo

Arrays en JAVA

1. Qué es un Array?
2. Cómo declarar una Array?
3. Recorrido e inicialización de Arrays.
4. Array bidimensionales.
5. Arrays bidimensionales irregulares.
6. Copiar y clonar Arrays.
7. Ordenación de Arrays.

Arrays en JAVA

Qué es un Array?

Un Array se puede definir como una colección de un tamaño fijo que contiene variables del mismo tipo.

A continuación se muestra un ejemplo de Array para almacenar 8 notas sin decimales:

```
int[] notas = new int[8];
```

notas

	Array
notas[0]	0
notas[1]	0
notas[2]	0
notas[3]	0
notas[4]	0
notas[5]	0
notas[6]	0
notas[7]	0

En un Array el primer elemento siempre tiene índice 0:
notas[0]

En un Array el último elemento siempre tiene índice longitud-1:
notas[notas.length-1]

Cómo se declara un Array?

Los corchetes para indicar que se trata de un Array se pueden poner después del tipo, o después del identificador, de las dos formas es válido:

```
int[] notas = new int[8];
```

```
int notas[] = new int[8];
```

Los Arrays pueden estar formados por cualquier tipo primitivo, o por Objetos:

```
String nombres[] = new String[8];
```

```
boolean condiciones[] = new boolean[8];
```

```
char letras[] = new char[8];
```

Recorrer e inicializar un Array

Para recorrer los elementos de un Array podemos hacerlo mediante un bucle for. En un Array podemos utilizar la propiedad **length** para obtener la cantidad de elementos de dicho Array:

```
int[] notas = new int[5];
for(int i = 0; i < notas.length; i++) {
   notas[i] = i*2;
} //los valores almacenados en el Array serán 0,2,4,6,8
```

Si no inicializamos los elementos de un Array, se almacenará el valor por defecto según el tipo de dato.

```
int[] notas = new int[5];
for(int i = 0; i < notas.length; i++) {
   System.out.print(notas[i]);
} //se muestra 00000
```

```
byte → 0
short → 0
int → 0
long → 0
float → 0.0f
double → 0.0d
boolean → false
char → '\u0000'
String → null
Object → null
```

Declarar e inicializar un Array

Podemos inicializar los valores que tendrá un Array, poniéndolos entre llaves y separándolos por comas.

```
int[] notas = {6,7,8,1,10,5}; //Array de 6 enteros
for(int i = 0; i < notas.length; i++) {
   System.out.print(notas[i]);
} // 6781105
```

```
String[] nombres = {"Pep", "Tom", "Kal"}; //Array de 3 Strings
for(int i = 0; i < nombres .length; i++) {
   System.out.print("[" + nombres[i] +"]");
} // [Pep][Tom][Kal]
```

Ejercicios iniciación Arrays en JAVA

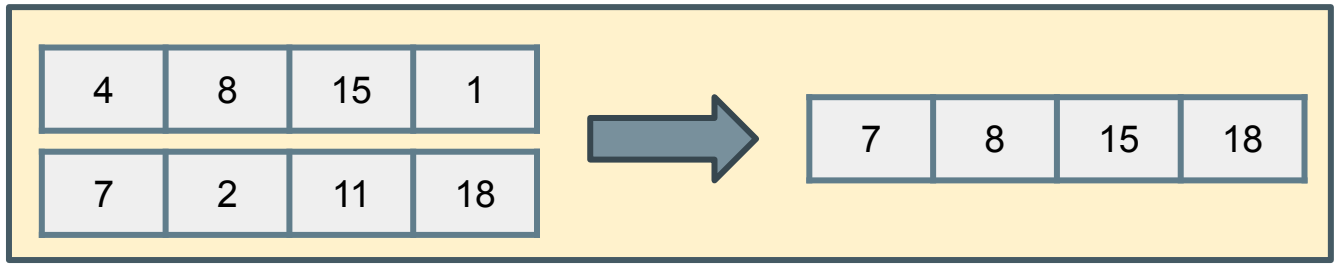
Ejercicios iniciación Arrays en JAVA

1. Crea un método que dado un Array de enteros y un valor entero, muestre la cantidad de veces que aparece el entero en el Array.

2. Crea un método que reciba un Array de Strings y muestre la String más larga.

3. Crea un método que reciba un Array de Strings y un char. Deberá mostrar todas las Strings cuya primera letra sea el char pasado como parámetro.

4. Crea un método que reciba dos Arrays como parámetros, y devuelva un Array con los valores máximos en cada una de las posiciones. Se debe tener en cuenta que los Arrays podrán ser de tamaños distintos.



Arrays bidimensionales en JAVA (matrices)

Array bidimensionales

- Un array multidimensional es aquel que para acceder a una posición concreta, en vez de utilizar un único valor como índice, se utiliza una secuencia de varios índices. Cada índice sirve como coordenada para una dimensión diferente.
- La sintaxis es como la de los arrays unidimensionales pero se le añaden unos corchetes adicionales.

```
Tipo[][] identificadorVariable = new Tipo[numeroFilas][numeroColumnas];
```

```
char letras[][];
int precios[][] = new int[5][6];
double[][] notas = new double[10][10];
String ciudades[][] = new String[10][5];
```

Array multidimensionales

- Si por ejemplo, necesitáramos almacenar las ventas por ciudad, podríamos necesitar una dimensión adicional. En el siguiente ejemplo podríamos representar 10 ciudades, y de cada ciudad tener 5 ventas:

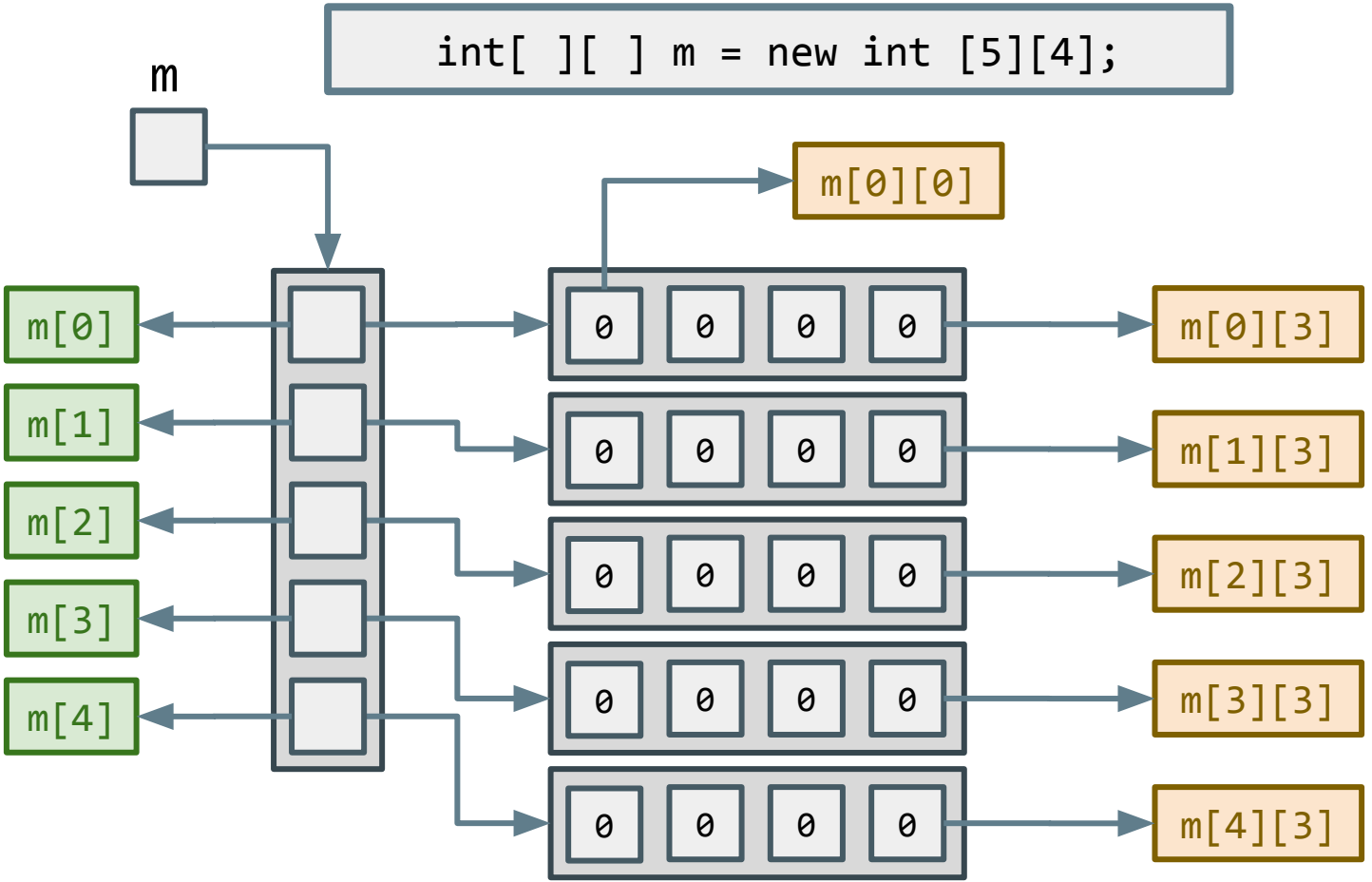
```
int[][] ventas2 = new int[10][5];
```

- Un Array bidimensional(matriz) se puede interpretar como una tabla cuya primera dimensión serían las **filas**(10) y que la segunda serían las **columnas**(5).

- Se pueden seguir añadiendo dimensiones, por ejemplo para incluir los años. En el siguiente ejemplo podríamos tener la información de los últimos 5 años.

```
int[][][] ventas3 = new int[4][10][5];
```

Declaración de arrays bidimensionales



Declaración de arrays bidimensionales

Podemos inicializar los valores de un Array bidimensional poniendo los elementos de cada Array interno entre llaves.

```
int[][] matriz = {
   {1,2,3,4,5},
   {6,7,8,9,10},
  {11,12,13,14,15}
        };
```

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15

Uso de arrays bidimensionales

El acceso a las posiciones de un array bidimensional sería:

```
identificadorArray[índiceFila][índiceColumna]
```

A continuación se muestran los índices de las posiciones de cada elemento en una matriz de 4 filas y 5 columnas.

```
int[][] a = new int[4][5];
```

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]
a[3][0]	a[3][1]	a[3][2]	a[3][3]	a[3][4]

Uso de arrays bidimensionales

Comportamiento de **length**:

Podemos obtener tanto la cantidad de filas, como la cantidad de elementos de cada fila:

a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]
a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]
a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]
a[3][0]	a[3][1]	a[3][2]	a[3][3]	a[3][4]

Por ejemplo, si dicho Array bidimensional lo identificamos como **a**:

- **a.length** vale 4 (hay 4 filas)
- **a[0].length**, **a[1].length**, etc. vale 5 (hay 5 columnas).

Recorrer arrays bidimensionales

- Para recorrer un array de varias dimensiones lo podemos hacer con dos bucles for anidados.
- Un bucle anidado utilizará dos índices, cada índice se utilizará para una dimensión diferente.
- Si tuviéramos un Array de 3 dimensiones, se necesitan 3 bucles anidados con 3 índices, aunque no es una estructura muy habitual.
- En Arrays bidimensionales, la condición del bucle externo debe hacer referencia a la cantidad de filas del Array. Mientras que la condición del bucle interno debe hacer referencia a la cantidad de elementos de la fila actual.

```
int[][] matriz = {{1,4,5},{6,7,2},{8,3,8}};
int filas = matriz.length; //3
int elementosFila0 = matriz[0].length; //3
int elementosFila1 = matriz[1].length; //3
int elementosFila2 = matriz[2].length; //3
```

Uso de arrays bidimensionales

```
public class Ejemplo_Recorrer {
    public static void main(String[] args) {
        //declaración y inicialización
        int[][] matriz = {{1,4,5},{6,7,2},{8,3,8}};
        //recorrido de la matriz
        for (int fila = 0; fila < matriz.length; fila++) {
            for (int columna = 0; columna < matriz[fila].length; columna++) {
                System.out.print(matriz[fila][columna]+" ");
            }
            System.out.println();
        }
    }
}
```

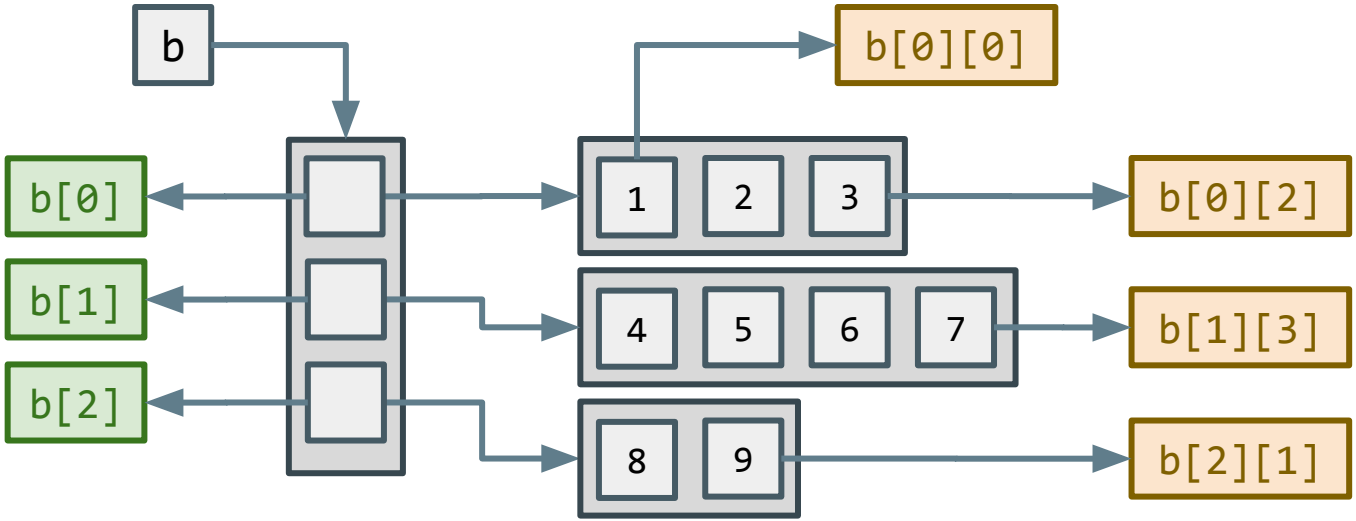
Arrays bidimensionales irregulares en JAVA

Uso de arrays bidimensionales irregulares

- El número de elementos de cada una de las filas de un Array puede especificarse de forma individual.
- La ejecución de la sentencia:

```
int[ ][ ] b = { {1, 2, 3}, {4, 5, 6, 7}, {8, 9} };
```

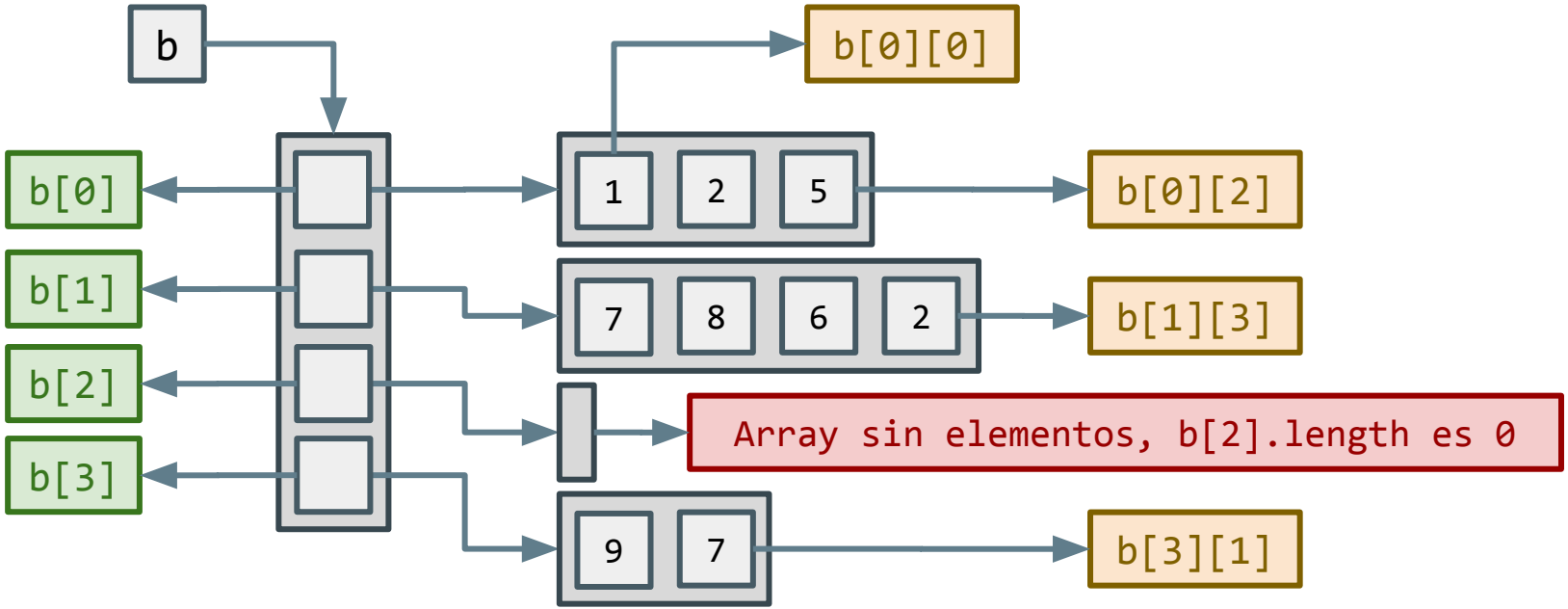
Genera un Array bidimensional irregular. En él, por ejemplo, no existen los elementos **b[0][3]** ni **b[2][2]**.



Uso de arrays bidimensionales irregulares

Podemos declarar de forma individual cada Array que forma parte de una matriz, y tener **algún Array sin elementos**.

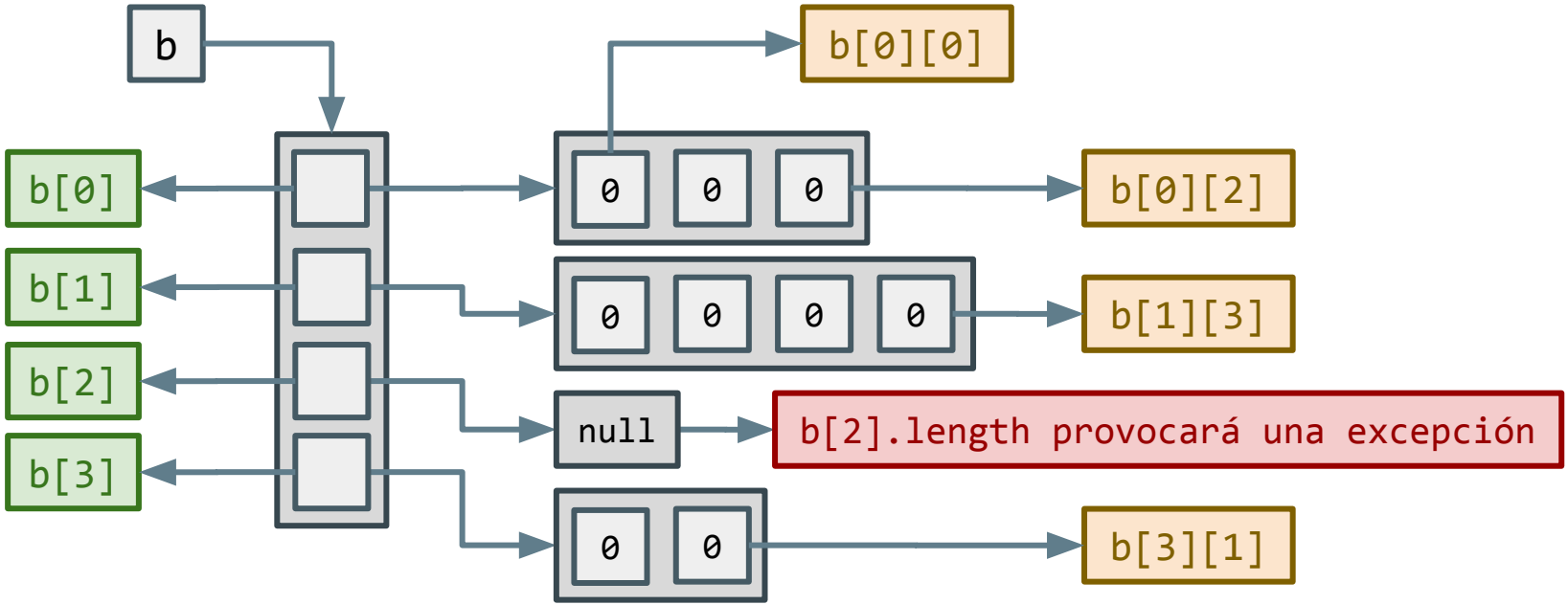
```
int[][] b =  {
  {1,2,5},{7,8,6,2},{},{9,7}
};
```



Uso de arrays bidimensionales irregulares

Podemos declarar de forma individual cada Array que forma parte de una matriz, y tener **algún Array nulo**.

```
int[][] b = new int[4][];
b[0] = new int[3];
b[1] = new int[4];
b[3] = new int[2];
```



Recorrer Arrays con foreach en JAVA

Recorrer arrays con foreach

Existen formas alternativas de recorrer un Array, para Arrays de 1 dimensión podemos hacerlo de forma abreviada de la siguiente forma:

```
int[] array = {1,4,5,6,7,8,3,8};
for(int n : array) {
    System.out.print(n);
}  //14567838
```

Si lo que queremos hacer es imprimir el contenido de un Array, podemos hacerlo de forma abreviada de la siguiente forma:

```
int[] array2 = {1,4,5,6};
System.out.println(Arrays.toString(array2));  // [1, 4, 5, 6]
```

Recorrer arrays bidimensional con foreach

Existen formas alternativas de recorrer un Array, para Arrays de 2 dimensiones podemos hacerlo de forma abreviada de la siguiente forma:

```
int[][] array3 = {{1,4,5},{6,7,8},{3,8}};
for(int[] fila : array3) {
    for(int columna : fila) {
        System.out.print(columna);
    }
} //14567838
```

Si lo que queremos hacer es imprimir el contenido de un Array bidimensional, podemos hacerlo de forma abreviada de la siguiente forma:

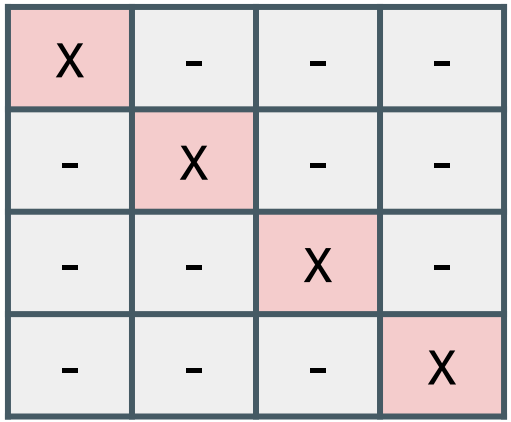
```
int[][] array4 = {{3,8,5},{4,1,8,4},{5,2}};
for(int[] fila : array4) {
    System.out.println(Arrays.toString(fila));
}
```

```
// [3, 8, 5]
// [4, 1, 8, 4]
// [5, 2]
```

Ejercicios matrices en JAVA

Ejercicio matrices

Crea y carga una matriz de n filas por n columnas. Para n = 4:



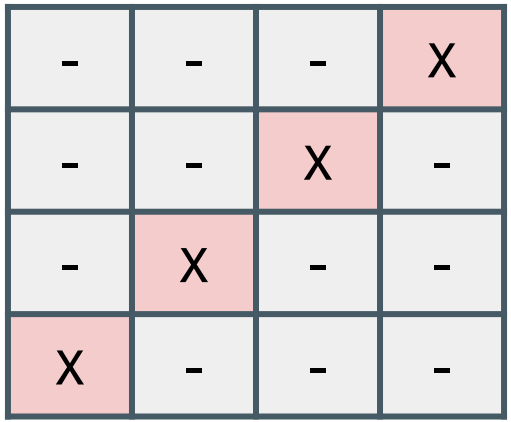
Debes utilizar los siguientes métodos:

- static int[ ][ ] **crearMatriz**(int n), para crear la matriz de tamaño n.
- static void **imprimirMatriz**(int[ ][ ] m), para mostrar el contenido.

Piensa una posible solución para imprimir la diagonal de forma invertida, la matriz debe ser la misma.

Ejercicio matrices

Crea y carga una matriz de n filas por n columnas. Para n = 4:



Debes utilizar los siguientes métodos:

- static int[ ][ ] crearMatriz(int n), para crear la matriz de tamaño n.
- static void imprimirMatriz(int[ ][ ] m), para mostrar el contenido.

Piensa una posible solución para imprimir la diagonal de forma invertida, la matriz debe ser la misma.

Ejercicio matrices

Crea un método que dadas dos matrices A y B, las compare elemento a elemento y muestre otra matriz M. Dicha matriz debe tener el valor máximo en cada una de las posiciones.

Las matrices se deben pasar como parámetros al método y deben ser exactamente iguales. Tanto en filas, como en elementos de cada fila. Por ejemplo:

A		
1	2	3
6	7	8
10	8	15

B		
5	1	4
7	3	11
11	12	13

M		
5	2	4
7	7	11
11	12	15

*Si las matrices no son iguales, se debe mostrar un mensaje indicando que no se pueden comparar. Se deberá utilizar un método para realizar dicha comprobación.

Ejercicio matrices irregulares

A partir del ejercicio anterior, mejorar el método para que funcione con matrices irregulares y de distinto tamaño. Podrán ser distintas tanto en la cantidad de filas, como en los elementos de cualquiera de sus filas.

Se deberá tener en cuenta que si hay una posición que únicamente existe en una de las dos matrices, dicho valor no se deberá comparar y deberá aparecer en la matriz resultante(M).

A		
1	2	
7	7	8
10	8	

B		
5	1	4
6	3	
11		
1	2	

M		
5	2	4
7	7	8
11	8	
1	2	

El famoso String[ ] args en JAVA

El famoso String[ ] args en JAVA

- Cabecera: **public static void main (String[] args)**
- Su tipo de retorno es **void**, es decir, no retorna ningún valor.
- Su único parámetro es un **Array de Strings**. Esta lista se recibe al ejecutar la aplicación con el comando java por consola, veamos algunos ejemplos:

```
class HolaMundo {
  public static void main (String args[])  {
    System.out.print("Hola");
    for(int i = 0; i < args.length; i++)  System.out.print(" " + args[i]);
    System.out.println(", qué tal estás? ");
  }
}
```

```
java HolaMundo                  // Hola, qué tal estás?
java HolaMundo Pepe             // Hola Pepe, qué tal estás?
java HolaMundo Pepe García      // Hola Pepe García, qué tal estás?
```

El famoso String[ ] args en JAVA

- Veamos un ejemplo adicional:

```
class HolaMundo {
    public static void main(String[] args) {
      if (args.length > 1) {
            for (int i = 0; i < Integer.parseInt(args[1]); i++)
                System.out.println("Hola " + args[0]);
      }
      else System.out.println("Hola mundo simple");
    }
}
```

```
java HolaMundo                  // Hola mundo simple
java HolaMundo Pepe             // Hola mundo simple
java HolaMundo Pepe 3           // Hola Pepe repetido en 3 líneas
java HolaMundo Pepe García      // ERROR en ejecución
```

Varags en JAVA

Varags en JAVA

Los **vargars** son parámetros de longitud variable en JAVA, para utilizarlos se añaden 3 puntos suspensivos después de indicar el tipo de datos del parámetro

- Internamente se transforma en un Array. De hecho se puede utilizar incluso en el main.
- Podemos acceder a través de índices a las posiciones.
- Podemos obtener la longitud con .length.
- El método únicamente puede tener un vargars, y debe ser siempre el último parámetro.

```
static void mostrarEnteros(int... nums) {
    System.out.print("Hay " + nums.length + " enteros: ");
    for (int i = 0; i < nums.length; i++) {
        System.out.print(nums[i] + " ");
    }
}
```

MAIN

```
int a = 2, b = 3;
mostrarEnteros();                // Hay 0 enteros:
mostrarEnteros(1,2,3);           // Hay 3 enteros: 1 2 3
mostrarEnteros(b,a++,a*b,++b);   // Hay 4 enteros: 3 2 9 4
```

main con varargs

- Veamos un ejemplo adicional de main con varargs:

```
class HolaMundo {
    public static void main(String... args) {
        if (args.length > 0) {
            System.out.println("Hola mundo con " + args.length + " args");
            for (int i = 0; i < args.length; i++)
                System.out.println("arg[" + i + "] = " + args[i]);
        }
        else System.out.println("Hola mundo sin args");
    }
}
```

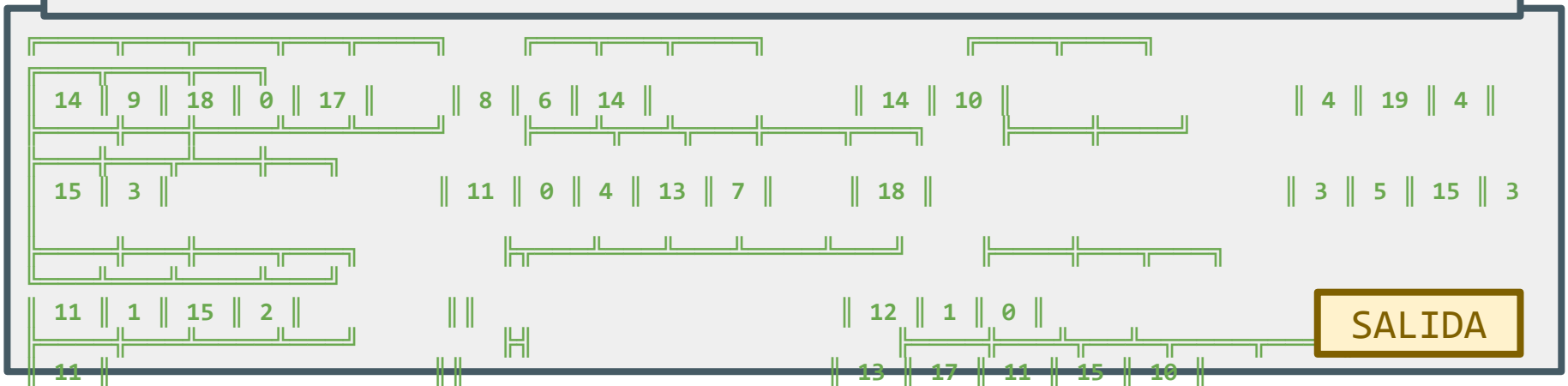
```
java HolaMundo                  // Hola mundo sin args

java HolaMundo Pepe García      // Hola mundo con 2 args
                                // args[0] = Pepe
                                // args[1] = García
```

Ejercicio bordes en matrices irregulares con varargs

Diseña un método para ampliar la funcionalidad del ejercicio de bordes realizado anteriormente. Ahora deberá ser capaz de imprimir una cantidad indeterminada de matrices irregulares una al lado de la otra. Dicho método recibirá un **vararg** de matrices de enteros como parámetro y un entero adicional. El entero se utilizará para indicar la cantidad de espacios de separación entre cada una de las matrices.

```
int[][] nums1 = { {14, 9, 18, 0, 17}, {15, 3}, {11, 1, 15, 2}, {11} };
int[][] nums2 = { {8, 6, 14}, {11, 0, 4, 13, 7}, {}, {}, {3} };
int[][] nums3 = { {14, 10}, {18}, {12, 1, 0}, {13, 17, 11, 15, 10} };
int[][] nums4 = { {4, 19, 4}, {3, 5, 15, 3} };
mostrarMatricesConBordes(5,nums1,nums2,nums3,nums4);
```



Ordenar Arrays en JAVA

Ordenación de Arrays

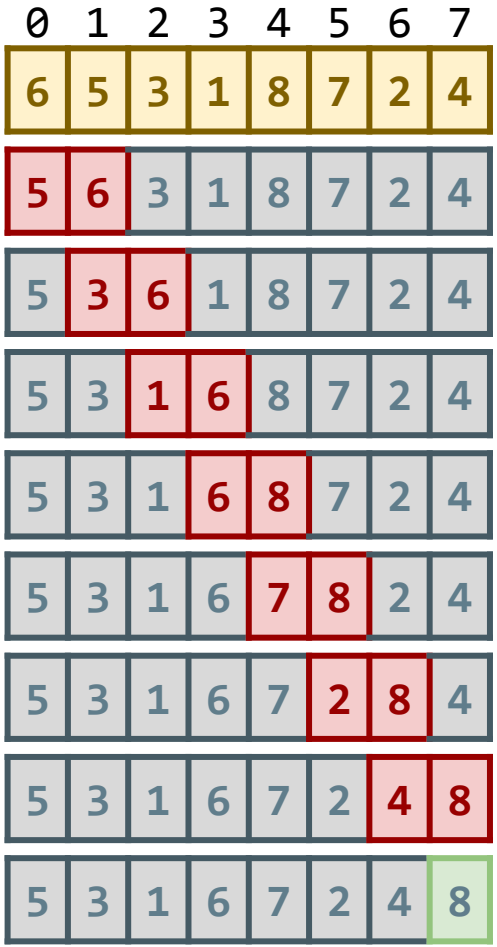
- **¿Para qué sirven?**
  - Para reorganizar el orden de los elementos de una colección.
- **¿Por qué ordenar los datos?**
  - Es mucho más eficiente trabajar con datos ordenados.
- **¿Qué es un algoritmo de ordenación?**
  - Forma estructurada que indica cómo ordenar los elementos que hay dentro.
- **¿Qué podemos ordenar?**
  - Cualquier estructura con elementos ordenables, ya sean números, Strings, u objetos.
- **¿Son todos los algoritmos iguales?**
  - No, algunos son más eficientes que otros.

El método burbuja

El algoritmo de la burbuja es uno de los métodos de ordenación más conocidos, y de los primeros en ser usados. Consiste en comparar pares de elementos adyacentes en un array y si están desordenados intercambiarlos hasta que terminen todos ordenados.

En cada pasada se comprueban los elementos adyacentes, de este modo el máximo lo situamos a la derecha del Array, a continuación hacemos lo mismo para el resto del elemento. Al estar ordenado el último elemento, ordenamos luego el resto de elementos.

A la derecha se puede observar un ejemplo de cómo se guarda en la última posición el valor máximo. Luego se repetirá en proceso descartando dicho elemento y así sucesivamente.



El método burbuja

```
public static int[] burbuja(int[] vector) {
   int aux;
   for (int i = 0; i < vector.length; i++) {
        for (int j = 0; j < vector.length - i - 1; j++) {
            //El valor máximo lo más a la derecha posible
            if (vector[j] > vector[j + 1]) {
                aux = vector[j];
                vector[j] = vector[j + 1];
                vector[j + 1] = aux;
            }
        }
    }
    return vector;
}
```

6	5	3	1	8	7	2	4

El método burbuja

```
//Declaración e inicialización
int[] arrayEjemplo = {3, 6, 9, 0, 1, 2, 3};
int[] arrayOrdenado;

//Salida inicial
System.out.println("Sin ordenar: ");
System.out.println(Arrays.toString(arrayEjemplo));

//Ordenación
arrayOrdenado = burbuja(arrayEjemplo);

//Salida final
System.out.println("Ordenado: ");
System.out.println(Arrays.toString(arrayOrdenado));
```

Quicksort en JAVA

El algoritmo de ordenación rápida Quicksort

Quicksort es uno de los algoritmos de ordenación más rápidos que existe, es capaz de ordenar un Array con un coste promedio de O(n*log n). Este método hace uso de la estrategia recursiva **divide y vencerás**. El algoritmo trabaja de la siguiente forma:

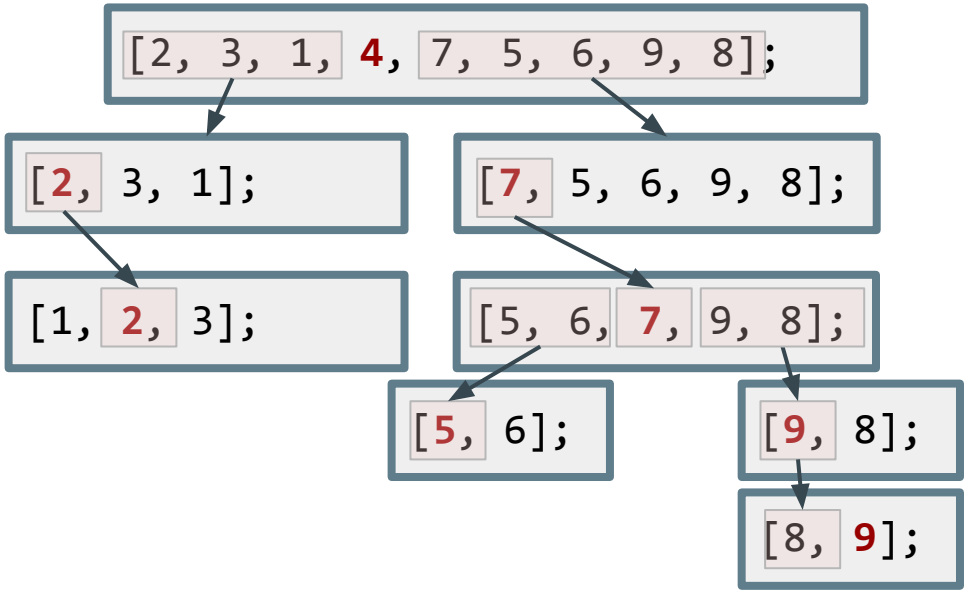
- Se debe elegir un elemento de la colección al que llamaremos pivote.
- Colocamos los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. En este momento, el pivote ocupa exactamente el lugar que le corresponde en la lista ordenada.
- La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
- Repetir este proceso de forma recursiva para cada sublista mientras tengan más de un elemento. Una vez terminado el proceso todos los elementos estarán ordenados.
- La eficiencia del algoritmo depende de la posición en la que termine el pivote elegido. En el mejor caso, el pivote termina en el centro de la lista, dividiéndola en dos sublistas de igual tamaño. En este caso, el coste del algoritmo es O(n*log n).En el peor caso, el pivote termina en un extremo de la lista, el coste del algoritmo es entonces de $O(n^2)$.

El algoritmo de ordenación rápida Quicksort

Veamos un ejemplo del funcionamiento del algoritmo Quicksort:

```
int[] nums = {4, 7, 5, 6, 2, 3, 1, 9, 8};
```

Elegimos como pivote el elemento de más a la izquierda, a partir de ahí colocamos a su izquierda los menores, y a su derecha los mayores



Repetimos el mismo proceso con las 2 sublistas de la izquierda y de la derecha.

Y así sucesivamente sobre cada una de las sublistas mientras las sublistas tengan más de un elemento.

El método Quicksort

```
public static void quicksort(int A[], int izq, int der) {
  int pivote = A[izq];    // tomamos primer elemento como pivote
  int i = izq;            // i realiza la búsqueda de izquierda a derecha
  int j = der;            // j realiza la búsqueda de derecha a izquierda
  int aux;

  while(i < j){          // mientras no se crucen las búsquedas
    while(A[i] <= pivote && i < j) i++;   // busca elemento mayor que pivote
    while(A[j] > pivote) j--;             // busca elemento menor que pivote
    if (i < j) {                          // si no se han cruzado
        aux  = A[i];                      // los intercambia
        A[i] = A[j];
        A[j] = aux;
    }
   }
   A[izq] = A[j];     // se coloca el pivote en su lugar de forma que tendremos
   A[j]   = pivote;   // los menores a su izquierda y los mayores a su derecha
   if(izq < j-1) quicksort(A, izq, j-1);        // ordenamos subarray izquierdo
   if(j+1 < der) quicksort(A, j+1, der);        // ordenamos subarray derecho
}
```

Búsqueda binaria en JAVA

Búsqueda binaria en un Array ordenado

Ahora que ya sabemos ordenar un Array de forma rápida, podemos hacer búsquedas de elementos de forma mucho más eficiente. Un algoritmo que se encarga de hacerlo es la búsqueda binaria. Veamos un ejemplo de este algoritmo con un método recursivo.

```
static boolean busquedaB(int[] nums, int numBuscado, int limiteInf, int limiteSup) {

if (limiteInf > limiteSup)
       return false; //no quedan elementos por analizar, NO ENCONTRADO

  int mitad = (limiteInf + limiteSup) / 2;

  if (nums[mitad] < numBuscado)
    return busquedaB(nums, numBuscado, mitad+1, limiteSup); //buscamos por la mitad der

  else if (nums[mitad] > numBuscado)
    return busquedaB(nums, numBuscado, limiteInf, mitad-1); //buscamos por la mitad izq

  else return true; //el elemento de la mitad coincide con numBuscado, ENCONTRADO
}
```

Búsqueda binaria en un Array ordenado

Se debe tener en cuenta que los 2 parámetros finales se deben inicializar siempre a **0** y a .**length-1**. De hecho, la forma más correcta para implementar el método sería a través de un método intermedio, por ejemplo.

```
static void mostrarBusquedaBinaria(int[] numeros, int numeroBuscado) {

    if (busquedaBinaria(numeros, numeroBuscado, 0, numeros.length - 1))
       System.out.println("El número " + numeroBuscado + " está en el Array.");
    else
       System.out.println("El número " + numeroBuscado + " NO está en el Array.");
}
```

MAIN

```
int[] nums = {1,2,3,4,5,6,7,8};
mostrarBusquedaBinaria(nums, 4); //El número 4 está en el Array.
mostrarBusquedaBinaria(nums, 9); //El número 9 NO está en el Array.
```

La clase Arrays

La clase Arrays.

- En el paquete java.utils se encuentra una clase estática llamada Arrays.
- Se utiliza como si fuera un objeto (como ocurre con Math).
- Su uso es:

```
Arrays.método(argumentos);
```

Copiar y clonar Arrays en JAVA

Clonado de Arrays

Todos los objetos en JAVA tienen el método **clone( )** que realiza una copia exacta del contenido en otro espacio de memoria. Si se utiliza en los Arrays, nos permitirá hacer una copia del mismo y de esta forma crear otro Array independiente del anterior.

```
int[] primos = {1, 2, 3, 5, 7, 11, 13, 17, 19, 23};
int[] copiaClonado;
int[] copiaReferencia;
//clonación
copiaClonado = primos.clone();
//cambio en el clon, sin afectar al original
copiaClonado[0] = 0;
//copia de referencia
copiaReferencia = primos;
//cambio en la copia, afecta al original
copiaReferencia[1] = 0;
//salida
System.out.println(Arrays.toString(primos));          //[1, 0, 3, 5, 7, 11, 13, 17, 19, 23]
System.out.println(Arrays.toString(copiaClonado));    //[0, 2, 3, 5, 7, 11, 13, 17, 19, 23]
System.out.println(Arrays.toString(copiaReferencia));//[1, 0, 3, 5, 7, 11, 13, 17, 19, 23]
```

Clonado

- Todos los objetos en java tienen el método **clone** que realiza una copia exacta del contenido en otro espacio de memoria.

- Si se utiliza en los arrays, nos permitirá hacer una copia del mismo y de esta forma crear otro array independiente del anterior.

Copia por referencia en Arrays

En JAVA en una variable no primitiva(objeto) se almacena la dirección de dicho objeto en memoria. Los Arrays se consideran tipos no primitivos en JAVA. Esto implica que si hacemos una copia de un Array, y modificamos alguno de los elementos de la copia, también modificamos el Array original.

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

copiaArray = nums;
copiaArray[0] = 0; //modifica al Array original(nums)

System.out.println(Arrays.toString(nums));        //[0, 2, 3, 4, 5]
System.out.println(Arrays.toString(copiaArray));  //[0, 2, 3, 4, 5]
```

De hecho al imprimir el valor de un Array, nos muestra su espacio de memoria

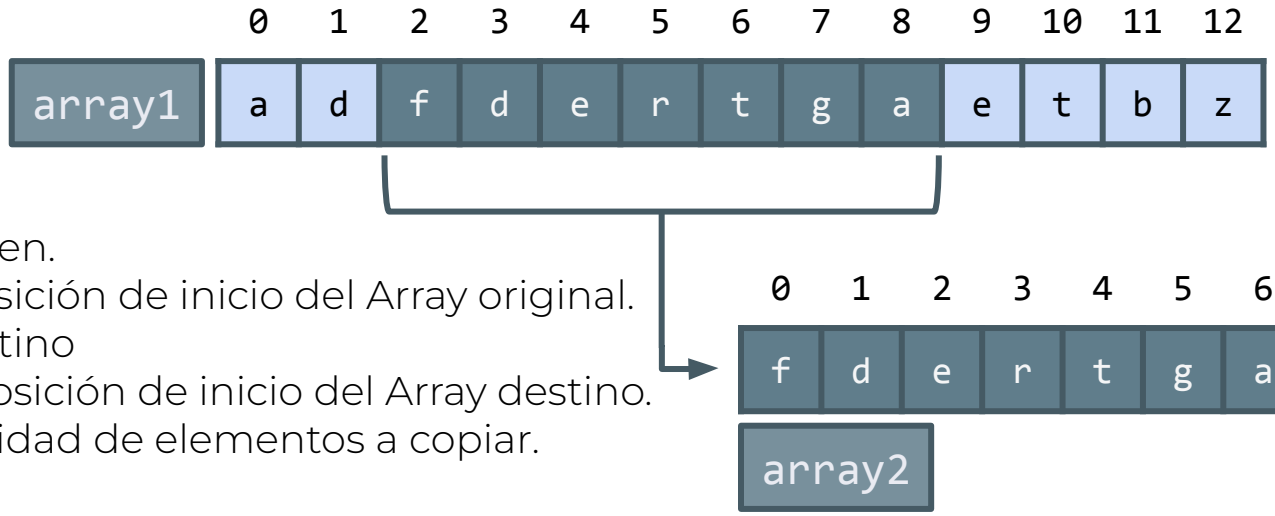
```
System.out.println(nums);       //[I@7440e464
System.out.println(copiaArray); //[I@7440e464
```

Copiar arrays

Copiar un array significa pasar los datos de un array a otro, existe un método en System que permite copiar elementos de un Array a otro:

```
arraycopy(array1, inicioOrigen, array2, inicioDestino, elementos)
```

```
arraycopy(array1, 2, array2, 0, 7);
```



Parámetros

- **array1**: Array origen.
- **inicioOrigen**: posición de inicio del Array original.
- **array2**: Array destino
- **inicioDestino**: posición de inicio del Array destino.
- **elementos**: cantidad de elementos a copiar.

Copiar arrays

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

        //declaración e inicialización
        int[] primos = {1, 2, 3, 5, 7, 11, 13};
        int[] copia = new int[primos.length];

        //copia
        System.arraycopy(primos, 1, copia, 3, 3);

        //salida
        System.out.println(Arrays.toString(primos)); // [1,2,3,5,7,11,13]
        System.out.println(Arrays.toString(copia));  // [0,0,0,2,3,5,0]
    }
}
```

Métodos

- **fill**: permite rellenar todo un array unidimensional con un determinado valor. Sus argumentos son el array a rellenar y el valor deseado:
  - Por ejemplo, llenar un array de 23 elementos enteros con el valor -1

```
int valores[] = new int[23];
Arrays.fill(valores, -1)
```

  - También permite decidir desde que índice hasta qué índice rellenamos:

```
Arrays.fill(valores, 5, 8, -1); //Almacena -1 desde el 5° al 7° elemento
```

Métodos

- **equals** : Compara dos arrays y devuelve true si son iguales (false en caso contrario). Se consideran iguales si son del mismo tipo, tamaño y contienen los mismos valores.

```
Arrays.equals(valoresA, valoresB);
```

Métodos

- **sort** : Permite ordenar un array en orden ascendente. Se pueden ordenar sólo una serie de elementos desde un determinado punto hasta un determinado punto.

```
int x[] = {4,5,2,3,7,8,2,3,9,5};
Arrays.sort(x); //Ordena de menor a mayor
Arrays.sort(x,2,5);//Ordena x solo desde el 2° hasta el 4° elemento
```

Métodos

- **binarySearch :** Permite buscar un elemento de forma ultrarrápida en un array ordenado. Devuelve el índice en el que está colocado el elemento buscado. Ejemplo:

```
int x[] = {1,2,3,4,5,6,7,8,9,10,11,12};
Arrays.sort(x); //Ordena de menor a mayor
System.out.println(Arrays.binarySearch(x,8);//Devuelve 7
```

Bibliografía:

Allen Weiss, M. (2007). *Estructuras de datos en JAVA*. Madrid: Pearson

Froufe Quintas, A. (2002). *JAVA 2: Manual de usuario y tutorial*. Madrid: RA-MA

J. Barnes, D. *Programación orientada a objetos en JAVA*. Madrid: Pearson

Desing Patterns. Elements of Reusable. OO Software

JAVA Limpio. Pello Altadi. Eugenia Pérez

Apuntes de Programación de Anna Sanchis Perales

Apuntes de Programación de Lionel Tarazón Alcocer

Ilustraciones:

https://pixabay.com/

https://freepik.es/

https://lottiefiles.com/

Preguntas