

**PROGRAMACIÓN**

## **UP3. ESTRUCTURAS DE DATOS**



**1º CFGS DAW**

Curso 2024-25



# Arrays

## 1. Estructura de los vectores (arrays)

### ¿Qué es un Array?

Un *array* se puede definir como una colección de datos (de tamaño fijo) que contiene valores 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]**

## 1.1. Creación de vectores en Java

---

### ¿Cómo se declara un Array?

---

Para declarar un *array* se tiene que indicar el tipo de dato de los elementos que contendrá y el nombre:

```
1 | int[] numeros;
```

Aunque la línea de código anterior es válida, aún no se podría utilizar el *array*. Necesitamos crear una nueva instancia y asignarla a la "variable". Esto se hace usando la palabra reservada *new*, seguida del tipo de dato, así como indicando el tamaño que tendrá el *array* (entre corchetes),

```
1 | numeros = new int[10];
```

Una vez creado un *array* en Java, este ya no puede cambiar su tamaño.

Los *arrays* pueden estar formados por cualquier tipo de datos:

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

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

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

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];
```

Para insertar un elemento en un *array* se debe especificar entre corchetes en qué posición se necesita insertar el elemento y el valor:

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

## 1.2. Inicialización

---

### Inicializar y recorrer un Array

---

#### Inicializar un array

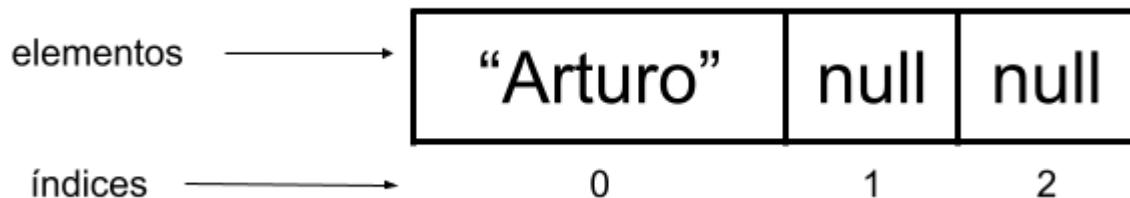
Dado el siguiente array de caracteres (tipo *String*),

```
1 | String[] nombres = new String[3];
```

Para insertar el valor “Arturo” en la posición 0, se hace de la siguiente forma:

```
1 | nombres[0] = "Arturo";
```

A continuación se muestra una representación visual del array después de insertar el valor “Arturo” en la posición 0:



Observa como los índices 1 y 2 contienen *null*, esto es porque Java otorga esos valores cuando se instancia el *array*. Por lo tanto, si no inicializamos los elementos de un *array*, se almacenará el valor por defecto según el tipo de dato:

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

También se pueden declarar, instanciar, y asignar elementos a los *arrays* en una sola línea. Por ejemplo:

```
1 | int[] numeros = {2,4,6,8};
```

observa cómo los elementos con los que se está inicializando el *array* van entre llaves {2,4,6,8}. Además, en este caso no se debe utilizar la palabra reservada *new*, ni corchetes junto al tamaño.

## Excepciones y errores

- Dado que el tipo de dato del *array* es *String*, sólo se pueden insertar elementos de tipo *String*. **Si se intenta insertar un elemento de otro tipo, Java devuelve un error.**

Por ejemplo, al intentar insertar un *número* en el *array nombres*,

```
1 | nombres[1] = 4;
```

se obtiene un error similar al siguiente,

```
C:\Users\Patricia\IdeaProjects\Ejemplo_repo\untitled\src\main\java\org\example>Main.java:55:22
java: incompatible types: int cannot be converted to java.lang.String
```

Y en concreto, *IntelliJ IDEA* nos lo notifica directamente mediante el intérprete:

The screenshot shows a Java code editor with the following code:

```
String[] nombres = new String[3];
nombres[1] = 4;
```

A tooltip is displayed over the line `nombres[1] = 4;`, indicating a type mismatch:

- Required type: String
- Provided: int

Buttons at the bottom of the tooltip include "Wrap using 'String.valueOf()'" (Alt+Mayús+Intro), "More actions..." (Alt+Intro), and a menu icon (three dots).

- Por otro lado, si se intenta insertar un elemento en una posición que no existe en el *array*, Java devolverá una excepción llamada *ArrayIndexOutOfBoundsException*. Este error puede que lo hayas visto antes, ya que trabajando con cadenas de caracteres puede ser que hayas utilizado alguna función que se comporte igual (como *charAt(posicion)*).

Por ejemplo, se sabe que el índice 3 no está disponible en el *array* *nombres* porque la numeración empieza en 0 y es de tamaño 3, por lo tanto, los índices disponibles son 0, 1, y 2. Si se intenta insertar un elemento en el índice 3:

```
1 | nombres[3] = "José";
```

se obtiene la siguiente excepción,

```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2022.3
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException Create breakpoint : Index 3 out of bounds for length 3
at org.example.Main.main(Main.java:55)
```

## Recorrer un *array*

Para recorrer los elementos de un *array* podemos hacerlo mediante un bucle *for*. Al igual que con la clase *String*, también podemos utilizar la propiedad *length* para obtener la cantidad de elementos que forman 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
```

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

**OJO** con el signo de la condición en el bucle *for*. Si elegimos *i <=*, debemos repetir el bucle hasta *notas.length - 1*.

## EJEMPLOS

- Si nos ceñimos al programa del punto anterior, en caso de querer imprimir directamente los valores del *array* sin haberlos inicializado, obtendríamos valores nulos:

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

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

- Podemos inicializar manualmente 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]
```

```
1 | int[] notas = {6,7,8,1,10,5}; //Array de 6 enteros
2 | for(int i = 0; i < notas.length; i++) {
3 |     System.out.print(notas[i]);
4 | } // 6781105
```

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

## 1.3. Tratamiento de la información almacenada en vectores

---

- Recorrer *arrays* con *ForEach*

Existen formas alternativas de recorrer un *array*. Podemos hacerlo de forma abreviada mediante un bucle tipo *forEach*:

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

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

Este tipo de bucle no necesita condiciones, simplemente irá guardando en la variable de control (*n*) los valores del *array* para la posición dada con cada iteración (recorre todas las posiciones disponibles en el *array*). Así nos ahorraremos también calcular el tamaño (*length*).

- Si lo que queremos hacer es **imprimir el contenido de un *array***, podemos hacerlo así:

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

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

De este modo, nos ahorraremos escribir un bucle.

- **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, por lo tanto, esto implica que

si hacemos una copia de un *array* y modificamos alguno de los elementos de la copia, también modificaremos 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]
```

```
1 int[] nums = {1, 2, 3, 4, 5};  
2 int[] copiaArray;  
3 copiaArray = nums;  
4 copiaArray[0] = 0; //modifica al Array original(nums)  
5 System.out.println(Arrays.toString(nums)); // [0, 2, 3, 4, 5]  
6 System.out.println(Arrays.toString(copiaArray)); // [0, 2, 3, 4, 5]
```

De hecho, al intentar imprimir el valor de un *array*, nos muestra su espacio de memoria. Observa que es exactamente el mismo:

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

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

- **Clonado de arrays**

Dado el problema visto en el punto anterior, 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 que ya podremos modificar a nuestro antojo.

```

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, 2, 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]

```

```

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

```

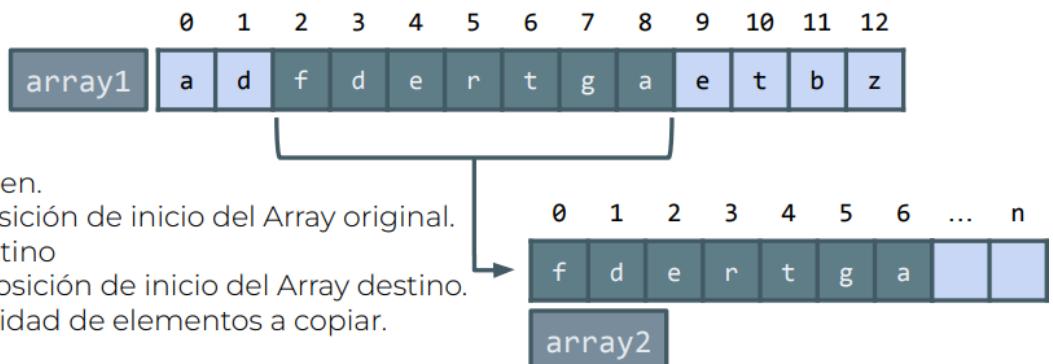
- Copiar un **array** significa pasar los datos de un **array** a otro (todos o una parte). Existe un método en **System** que permite copiar elementos de un **array** a otro:

```
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.



```
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]  
    }  
}
```

```
1 | public class EjemploCopiaArray {  
2 |     public static void main(String[] args) {  
3 |  
4 |         //declaración e inicialización  
5 |         int[] primos = {1, 2, 3, 5, 7, 11, 13};  
6 |         int[] copia = new int[primos.length];  
7 |  
8 |         //copia  
9 |         System.arraycopy(primos, 1, copia, 3, 3);  
10 |  
11 |         //salida  
12 |         System.out.println(Arrays.toString(primos)); // [1,2,3,5,7,11,13]  
13 |         System.out.println(Arrays.toString(copia)); // [0,0,0,2,3,5,0]  
14 |     }  
15 | }
```

# Batería de ejercicios sobre arrays

---



## Ejercicio 1. Suma de todos los elementos de un array

---

Crea un array de 8 números enteros e inicialízalo con números aleatorios [0,500]. Calcula y muestra la suma de todos los elementos del array.

La suma de los elementos es: 75



## Ejercicio 2. Invierte un array

---

Crea un array de números enteros (los que quieras) y luego invierte sus elementos. Imprime el array invertido.

Array original: 1, 2, 3, 4, 5  
Array invertido: 5, 4, 3, 2, 1



## Ejercicio 3. Contar ocurrencias de un número en un array

---

Crea un array de 25 números enteros e inicialízalo con valores aleatorios [0,100]. Luego, pide al usuario que ingrese un número [0,100] y muestra cuántas veces aparece ese número en el array.

Introduce un número para buscar [0,100]: 5  
El número 5 aparece 3 veces en el array.



## Ejercicio 4. Elimina un elemento de un array

---

Crea un array de enteros y permite al usuario ingresar una posición. Luego, crea un nuevo array sin el elemento en esa posición.

```
Array original: 1, 3, 5, 7, 9  
Introduce el índice a eliminar: 2  
Array resultante: 1, 3, 7, 9
```



## Ejercicio 5. Rota un array hacia la derecha

---

Crea un array y rota todos sus elementos una posición hacia la derecha. El último elemento debe moverse a la primera posición.

```
Array original: 1, 2, 3, 4, 5  
Array rotado: 5, 1, 2, 3, 4
```



## Ejercicio 6. Comprueba si el array es simétrico (palíndromo)

---

Crea un array de números enteros y verifica si es simétrico (es decir, se lee igual de izquierda a derecha que de derecha a izquierda).

```
Array: 1, 2, 3, 2, 1  
Es simétrico: SI
```



## Ejercicio 7. Combina dos arrays en uno

---

---

Crea dos arrays de enteros y combina sus elementos en un solo array más grande. Luego, muestra el nuevo array.

```
Array 1: 1, 2, 3  
Array 2: 4, 5, 6  
Array combinado: 1, 2, 3, 4, 5, 6
```



## Ejercicio 8. Inserta un elemento en un array

---

Crea un array e inserta un nuevo elemento en una posición específica sin sobrescribir el contenido.

```
Array original: 1, 2, 3, 5  
Inserta el número 4 en la posición 3  
Array resultante: 1, 2, 3, 4, 5
```



## Ejercicio 9. Palabra más larga

---

Crea un array de tipo String y muestra la palabra más larga que contenga.

---



## Ejercicio 10. Identificar primera letra y contar

---

Crea un array de tipo *String* que guarde palabras y pide una letra (*char*) al usuario. El programa deberá mostrar todas las palabras cuya primera letra sea el valor del *char* que ha introducido el usuario.

---



# PRÁCTICA 1. BATALLA DE SAMURÁIS



## → INTRODUCCIÓN. El método *Split()*

En Java, el método *split()* perteneciente a la clase *String* divide una cadena utilizando un delimitador definido, como por ejemplo \^\$. / ? \* + () [ ]. También espacios.

```
String cadena = "Hola,Soy,Una,Cadena";
String[] subcadenas = cadena.split(",");
System.out.println(Arrays.toString(subcadenas));
```

Ejecutando el código anterior observaremos que las subcadenas resultado de dividir la cadena principal se guardan en un *array* de la siguiente forma:

```
"C:\Program Files\Java\jdk-23\bin\java.exe" "-javaagent:D:\NetBeans\lib\ext\jpda.jar" -Dfile.encoding=UTF-8
[Hola, Soy, Una, Cadena]

Process finished with exit code 0
```

Es decir, ha troceado la frase teniendo el delimitador (,) como referencia y ha guardado cada palabra obtenida en una posición distinta del *array*. El tamaño del array resultado será la cantidad de subcadenas obtenidas.

## → PROBLEMA A RESOLVER

Se trata de un juego para dos jugadores, en el que cada uno de ellos dispone de un equipo de 7 samuráis. Cada samurái es identificado por un número entre el 1 y el 7, y tiene un valor de potencia entre 1 y 24. El jugador debe asignar un total de 30 unidades de potencia entre sus siete samuráis. Una vez creados los dos equipos de guerreros, empieza el combate.

- Se obtendrá un número aleatorio para indicar cual es el orden del guerrero con el que se inicia el combate. Luego, se seguirá luchando en orden consecutivo.
- Muere el samurái que tiene menor orden de potencia (y su potencia pasa a valer 0).
- En caso de empate, mueren ambos samuráis. Cuando un equipo tiene más de la mitad de sus samuráis muertos, pierde la partida.

## EJEMPLO.

```
> Equipo 1
> Introduce potencia de los samurais: 1 3 5 5 7 8 2
> ERROR. La potencia total no suma 30.
> Introduce potencia de los samurais: 1 3 5 5 7 8 1
> Equipo completado.
> Equipo 2
> Introduce potencia de los samurais: 6 6 6 6 1 1 4
> Equipo completado.
> ¡Empieza la batalla!
> La batalla inicia con el Samurai 3.
> Samurai 3. Gana Equipo 2. 5 vs 6
> Samurai 4. Gana Equipo 2. 5 vs 6
> Samurai 5. Gana Equipo 1. 7 vs 1
> Samurai 6. Gana Equipo 1. 8 vs 1
> Samurai 7. Gana Equipo 2. 1 vs 4
> Samurai 1. Gana Equipo 2. 1 vs 6
> ¡Equipo 2 GANA! Equipo 1 ha tenido 4 bajas.
```

Realiza un programa en *Java* que implemente la lógica del juego explicada anteriormente, usando vectores.

## → REALIZACIÓN DE LA PRÁCTICA

Sigue los siguientes pasos para realizar la práctica. ¡Ve guardando tu trabajo de vez en cuando para evitar que se borre el avance si se cierra el editor de textos u ocurre cualquier problema en tu equipo!

1. Programa en Java la aplicación requerida
2. Plan de pruebas. Realiza las pruebas necesarias para comprobar que el programa funciona bien



## ENTREGA

**REALIZA UN INFORME EN PDF CON LA INFO GENERADA Y LOS PASOS SEGUIDOS PARA REALIZAR ESTA PRÁCTICA. EXPLICA TU CÓDIGO. SÚBELO TODO A LA TAREA DE AULES DISPONIBLE.**

**ADEMÁS, PEGA LA URL DE TU PROYECTO EN GITHUB.**

# Ampliación split() y uso de asList()

## Ampliación split()

### El método *split()*

Como ya hemos visto en la práctica de los samuráis, el método *split()* es muy útil cuando necesitamos trocear un *String* para guardarnos sus valores separados en un array.

Este método divide una cadena utilizando un delimitador definido, como por ejemplo `| ^ $ . / ? * + () [ { - /`. También espacios.

```
1 | String cadena = "Hola,Soy,Una,Cadena";
2 | String[] subcadenas = cadena.split(",");
3 | System.out.println(Arrays.toString(subcadenas));
```

**IMPORTANTE.** Este método solamente se puede aplicar a variables de tipo *String*.

Ejecutando el código anterior, observaremos que las subcadenas resultado de dividir la cadena principal se guardan en un *array* de la siguiente forma:

```
"C:\Program Files\Java\jdk-23\bin\java.exe
[Hola, Soy, Una, Cadena]

Process finished with exit code 0
```

Es decir, ha troceado la frase teniendo el delimitador (,) como referencia y ha guardado cada palabra obtenida en una posición distinta del array. El tamaño del array resultado será la cantidad de subcadenas obtenidas.

Para practicar...

Queremos hacer un programa que muestre nuestra alegría de que llega la NAVIDAD y permita imprimir esa palabra con distinta cantidad de letras. Tendremos como entrada 7 números enteros, que indicarán el número de "N"s, el número de "A"s, el número de "V"s, el número de "I"s, etc.

Por ejemplo, si los números son:

n = 2

a = 1

v = 3

i = 2

d = 4

a = 5

d = 1

deberemos escribir la palabra NNAVVVIIDDDDAAAAAD que tiene 2 "N"s, 1 "A", 3 "V"s y 2 "I"s,... Es decir, leeremos esos 7 números y tendremos que escribir correctamente la palabra resultante.

Ejemplo de entrada:

```
Introduce cuantas letras quieres en la palabra NAVIDAD...
3 4 5 2 3 2 1
[3, 4, 5, 2, 3, 2, 1]
[N, A, V, I, D, A, D]
```

Ejemplo de salida:

```
Introduce cuantas letras quieres en la palabra NAVIDAD...
3 4 5 2 3 2 1
[3, 4, 5, 2, 3, 2, 1]
[N, A, V, I, D, A, D]
NNNAAAAAVVVVVIIDDDAAD

Process finished with exit code 0
```

[Ejercicio adaptado de la [Olimpiada Informática <https://oicv.org/valencia/wp-content/uploads/2023/02/olimpiadas\\_2023\\_es.pdf>](https://oicv.org/valencia/wp-content/uploads/2023/02/olimpiadas_2023_es.pdf) ]

# Uso de `asList()`

---

## Buscar valores en un array

---

`asList` es un método de la clase `java.util.Arrays` que convierte un *array* en una lista (*List*). La lista como tal de momento no nos interesa, pero sí el uso que este método permite hacer para hacer búsquedas rápidas entre los valores de un *array* para no tener que recorrerlo con un bucle constantemente.

### Uso de `asList` con Strings

Imagina que tienes un *array* de colores, y necesitas saber si el color que ha introducido el usuario está "apuntado" en esa lista:

```
1 | Scanner teclado = new Scanner(System.in);
2 | String colores[] = {"azul", "rojo", "amarillo", "verde", "negro"};
3 | System.out.println("Introduce un color: ");
4 | String color = teclado.next();
```

La forma de usar el método es la siguiente:

```
1 | boolean color_encontrado = Arrays.asList(colores).contains(color);
```

De esta forma, si el color introducido por el usuario existe en el *array* definido con los colores permitidos, la variable `color_encontrado` nos devolverá `true`. En caso contrario, nos devolverá `false`.

### Uso de `asList` con números

Este método también se puede usar con valores numéricos, pero hay que tener en cuenta que las listas no pueden guardar tipos de datos primitivos (`int, char, etc`), así que deberemos definir nuestros *arrays* como de tipo `Integer` en lugar de `int` si queremos que nos funcione.

Por ejemplo, si queremos saber si un número se encuentra dentro de un rango de valores:

```
1 Integer numeros[] = new Integer[50];
2
3     for (int i = 1; i <= numeros.length; i++) {
4         numeros[i] = i;
5     }
6
7 System.out.println("Introduce un número: ");
8 int num = teclado.nextInt();
9
10 boolean num_encontrado = Arrays.asList(numeros).contains(num);
11
12 if (!num_encontrado){
13     System.out.println("Lo siento, valor número " + num + " no en-
14
15 }else{
16     System.out.println("Valor número " + num + " encontrado!");
17 }
```

---

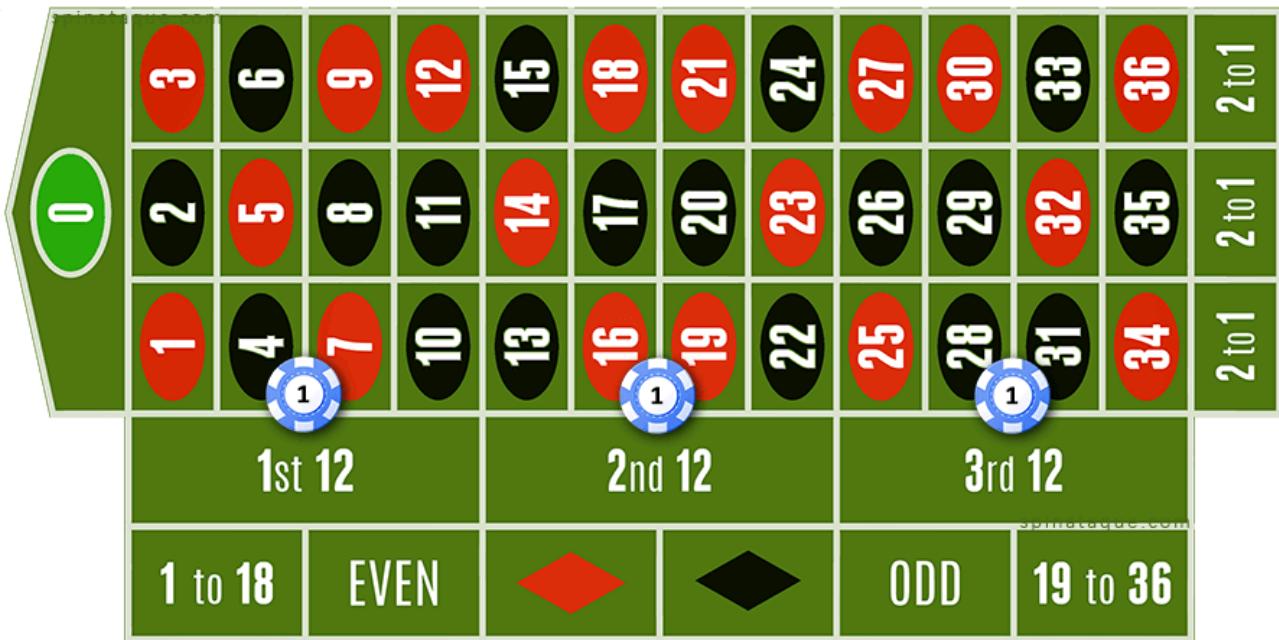
Vuelve a AULES e intenta resolver la actividad  
<https://aulas.edu.gva.es/fp/mod/assign/view.php?id=7355925> de la ruleta del casino...

---

## Actividad. La ruleta del casino (asList() + return)



### Actividad. La ruleta del casino (asList() + return)



1. Definir colores válidos (rojo y negro).
2. Definir números validos (0-36).
3. Definir valores par/impar.
4. Pedir al usuario un número (0-36).
5. Comprobar si el número del usuario está dentro del rango válido (0-36). Si no, notificar ERROR y salir del programa.
6. Pedir al usuario un color (se debe dar por bueno en mayúsculas y minúsculas). **En caso de haber introducido un 0 como número, no pediremos el color ni par/impar.**
7. Comprobar si el color está dentro de los colores válidos (rojo-negro). Si no, notificar ERROR y salir del programa.
8. Pedir al usuario opción par/impar (se debe dar por bueno en mayúsculas y minúsculas)
9. Comprobar si la opción par/impar está dentro de las opciones permitidas. Si no, notificar ERROR y salir del programa.

8. El programa debe sortear al azar un color y un número aleatoriamente. Si es par o impar se debe calcular.

9. Mostrar qué color, opción par/impar y número han tocado tras "girar" la ruleta.

10. Verificar si el usuario ha tenido suerte.

- Si acierta tanto número como color y que es par/impar (mostrar que HA GANADO).
  - Si acierta sólo el color (mostrar que ha acertado el color).
  - Si acierta sólo si es par o impar (mostrar que ha acertado que es par/impar).
  - Si acierta el número (mostrar que ha acertado el número). En caso de ser 0, notificar que HA GANADO y los demás usuarios de la mesa pierden.
  - Si no acierta nada, mostrar que ha perdido.
- 



Obra publicada con Licencia Creative Commons Reconocimiento Compartir igual 4.0  
<<http://creativecommons.org/licenses/by-sa/4.0/>>

## 2. Ordenación de arrays

### 2. Ordenación de arrays

#### Ordenar Arrays en JAVA

---

¿Por qué ordenar los datos?

→ Es mucho más eficiente trabajar con datos ordenados.

¿Qué podemos ordenar?

→ Cualquier estructura con elementos ordenables, ya sean números, caracteres u objetos.

---

Los problemas más comunes en la informática son la búsqueda y la ordenación. En este caso, el proceso de ordenar un *array* consiste en recolocar los elementos que lo componen (ya sea de mayor a menor o viceversa), con el fin de acelerar la búsqueda posterior de la información.

En este aspecto, existe una multitud de algoritmos de ordenamiento, como por ejemplo el de *Intercambio/Selección*, que se comporta de la siguiente manera.

#### Cómo ordenar *arrays* en Java con bucles y variables auxiliares

Esta opción es la más manual, puesto que tendremos que programar todo el algoritmo, pero es una buena práctica que nos ayudará a comprender mejor el manejo de *arrays*.

El **algoritmo de Selección** es el algoritmo más sencillo que podemos utilizar a la hora de ordenar *arrays*. Básicamente, consiste en intercambiar elementos dentro del *array* hasta conseguir que este quede ordenado.

La idea es recorrer el *array* y comparar cada elemento con todos los siguientes para buscar el apropiado con el que intercambiárselo. Si ordenamos de forma ascendente (de menor a mayor), entonces buscaremos elementos menores para hacer el intercambio. Por el contrario, si ordenamos de forma descendente (de mayor a menor), buscaremos elementos mayores para intercambiar.

Vamos a verlo directamente con un ejemplo para que lo entendamos mejor. Utilizaremos el siguiente array:

```
1 | int array[] = {4,6,2,8,7};
```

0	1	2	3	4
4	6	2	8	7

### - De forma ascendente (de menor a mayor)

Lo vamos a ordenar de forma ascendente (de menor a mayor) utilizando el siguiente código:

```
1 | for (int i = 0; i < array.length-1; i++) {  
2 |     for (int j = i+1; j < array.length ; j++) {  
3 |         if (array[j]<array[i]){  
4 |             int aux = array[j];  
5 |             array[j] = array[i];  
6 |             array[i] = aux;  
7 |         }  
8 |     }  
9 | }
```

Vamos a analizarlo paso a paso:

- Como puedes comprobar hay dos bucles *for* y un *if* anidados.
- El *for* externo nos permite recorrer todos los elementos del *array* desde el primero (posición 0) hasta el penúltimo, por eso en la condición hemos puesto *array.length-1*. Está hecho así intencionadamente, puesto que queremos comparar cada elemento con todos los siguientes (todos los que están a su derecha). El último elemento no tiene más elementos detrás, por lo que no tiene sentido compararlo con nada. El dato que se encuentre en la posición *i* será el dato de referencia en cada iteración de este bucle.

- El bucle *for* interno sirve para recorrer todos los elementos que se encuentran posicionados después del dato de referencia, es decir, todos los que están a la derecha de la posición *i*. Cada uno de estos elementos los tenemos que comparar con el dato que se encuentra en la posición *i*, es decir, con el dato de referencia.

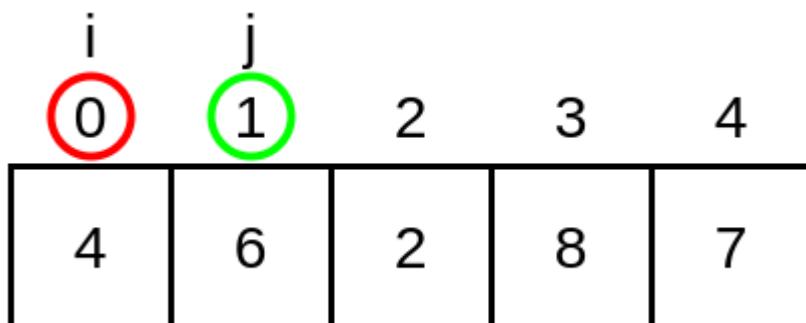
- De esto se encarga la instrucción *if*. Si encontramos en la posición *j* algún dato con valor menor al de referencia, lo intercambiamos con el que está en la posición *i*. Si no, no hacemos nada. Por eso la condición del *if* es  $array[j] < array[i]$ . Si se cumple la condición, se ejecutan las instrucciones de las líneas 4, 5 y 6, que básicamente realizan el intercambio con ayuda de una variable auxiliar. Si se produce un intercambio, entonces el valor de referencia cambia, puesto que se ha sustituido lo que había en la posición *i*.

Vamos a ver la **ejecución paso a paso** y cómo se va modificando el *array* en cada iteración de los bucles...

- Iteración 1. Primera iteración del *for* externo.

La variable *i* toma valor 0.

El *for* interno también comienza y la variable *j* toma valor *i+1*, es decir 1.

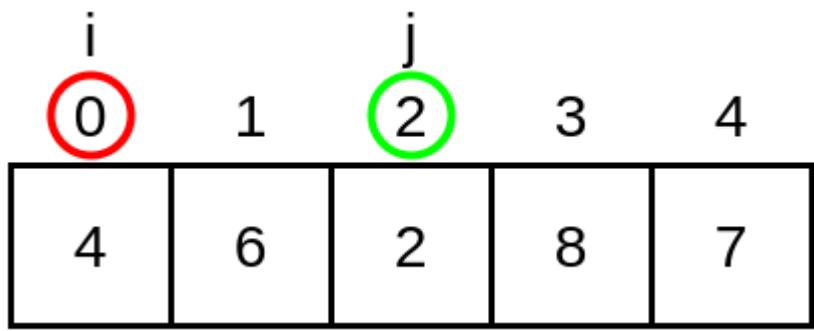


Ahora es el turno del *if*. Comparamos  $array[j] < array[i]$ . En este caso, la condición no es cierta porque  $array[j]$  tiene valor 6, y  $array[i]$  tiene valor 4. No se cumple que  $array[j]$  sea menor, por lo que no se ejecuta el cuerpo del *if*, y por lo tanto de momento no se realiza ningún intercambio.

El *for* interno avanza. El valor de *j* se incrementa una unidad (*j++*) y como sigue siendo menor que *length* (*j < array.length*), continúa su ejecución.

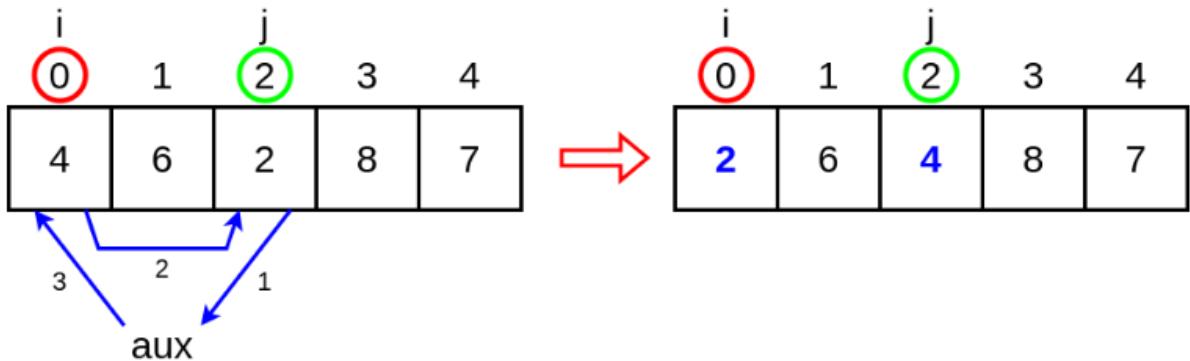
La variable *i* sigue siendo 0.

La variable *j* ahora es 2.



Se ejecuta de nuevo la instrucción *if*, y en este caso la condición es verdadera ( $\text{array}[j] < \text{array}[i]$ ) puesto que el dato que se encuentra en la posición *j* (2) es menor que el almacenado en la posición *i* (4). En este caso sí se ejecuta el cuerpo del *if*, y se produce un intercambio:

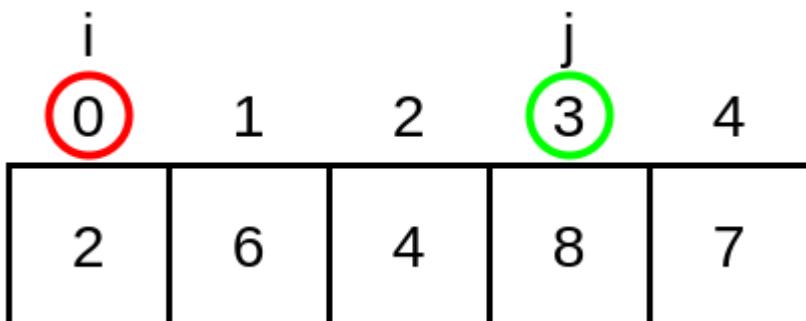
1. Se copia en la variable *aux* lo que hay en  $\text{array}[j]$  (2)
2. Se copia en  $\text{array}[j]$  lo que hay en  $\text{array}[i]$  (4)
3. Se copia en  $\text{array}[i]$  lo que hay en la variable *aux* (2)



El *for* interno avanza de nuevo, incrementando el valor de *j* en una unidad. Ahora *j* tiene valor 3 y se repite de nuevo la comprobación del *if*.

La variable *i* sigue siendo 0.

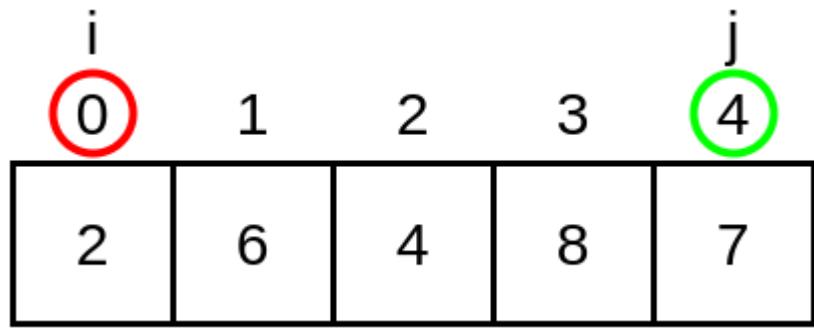
La variable *j* ahora vale 3.



En este caso no se cumple la condición  $\text{array}[j] < \text{array}[i]$ , puesto que 8 no es menor que 2. No se ejecuta ninguna de las instrucciones del cuerpo del *if*, y el *for* interno avanza de nuevo incrementando el valor de *j*.

La *i* sigue siendo 0.

La *j* ahora es 4.



En este caso ocurre lo mismo. El valor que hay en la posición *j* no es menor al que se encuentra en la posición *i*, por lo que la condición del *if* no se cumple y no se ejecuta nada.

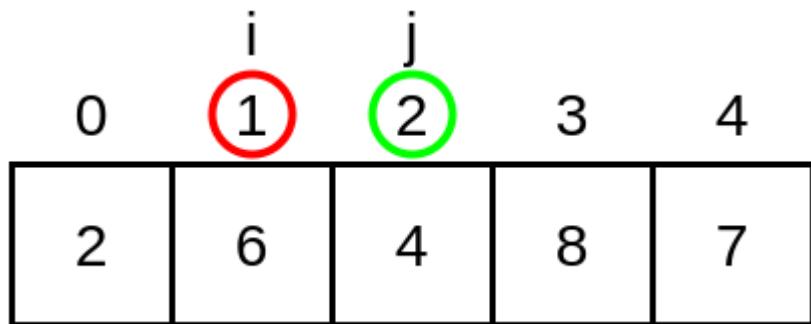
El *for* interno incrementa de nuevo el valor de *j* en una unidad, por lo que pasa a valer 5. Como *length* en este ejemplo también es 5, el bucle interno se detiene. Ha completado todas sus iteraciones.

**¿Qué hemos conseguido hasta aquí?** Comparar el dato de referencia (posición 0) con todos los demás e intercambiarlo por otro menor siempre que ha sido posible. Ahora toca avanzar el bucle externo.

#### - Iteración 2: Segunda iteración del *for* externo.

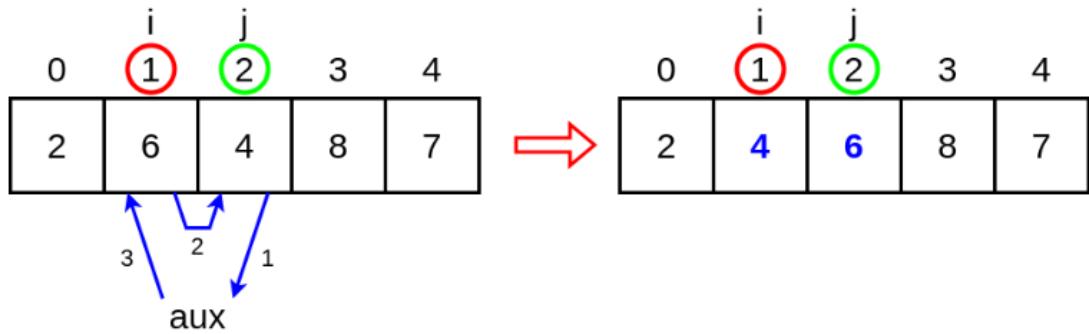
La variable *i* se incrementa en una unidad (*i++*) y pasa a valer 1.

El *for* interno empieza de nuevo. La *j* toma valor *i+1*, es decir 2.



¿Qué ocurre si ahora comprobamos la condición del *if*. *array[j]<array[i]*? Pues que es cierta, porque 4 es menor que 6. En este caso, se ejecutan las instrucciones del *if* y se intercambian los valores de las dos posiciones:

1. El valor 4 que se encuentra en la posición *j* se copia en la variable *aux*.
2. El valor 6 de la posición *i* se copia en la posición *j*.
3. Por último el valor de *aux*, que es 4, se copia en la posición *i*.



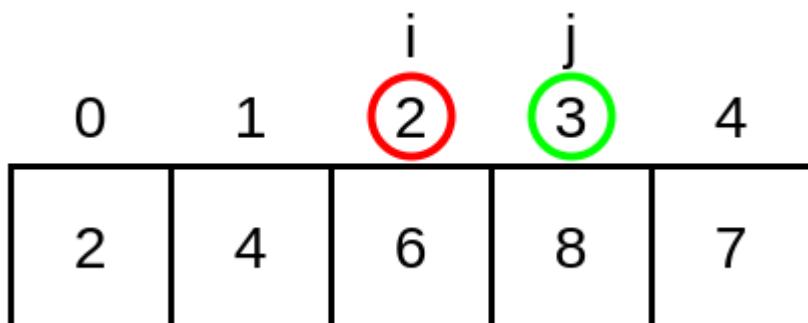
Las dos iteraciones que quedan del bucle interno no van a producir intercambios, puesto que 8 y 7 son mayores que 4, que es el valor de referencia actual, por lo que este bucle interno completará sus iteraciones sin más cambios.

#### - Iteración 3: Tercera iteración del *for* externo.

El *for* externo avanza de nuevo incrementando el valor de *i* en una unidad y el *for* interno comienza de nuevo desde la posición *i*+1.

La variable *i* ahora vale 2.

La variable *j* ahora vale 3.



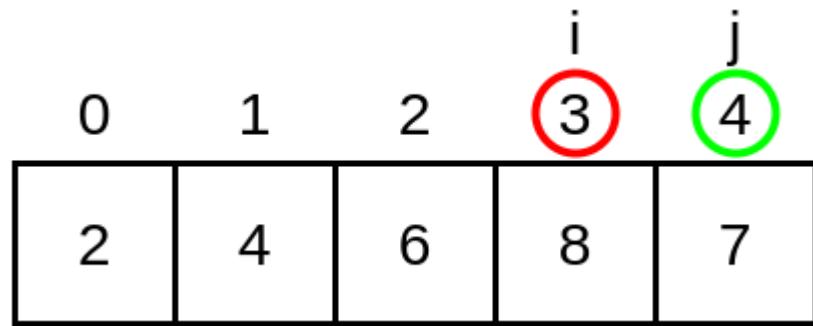
Como puedes comprobar, en esta iteración no se realizará ningún intercambio puesto que el valor que hay en la posición 2 (6) es menor que todos los elementos siguientes: 8 y 7. El bucle interno completará sus iteraciones sin cambios.

#### - Iteración 4: Cuarta y última iteración del *for* externo.

Hemos llegado a la última iteración del *for* externo. El valor de *i* se incrementa pasando a valer 3, y el *for* interno comienza de nuevo inicializando el valor de *j* a *i*+1.

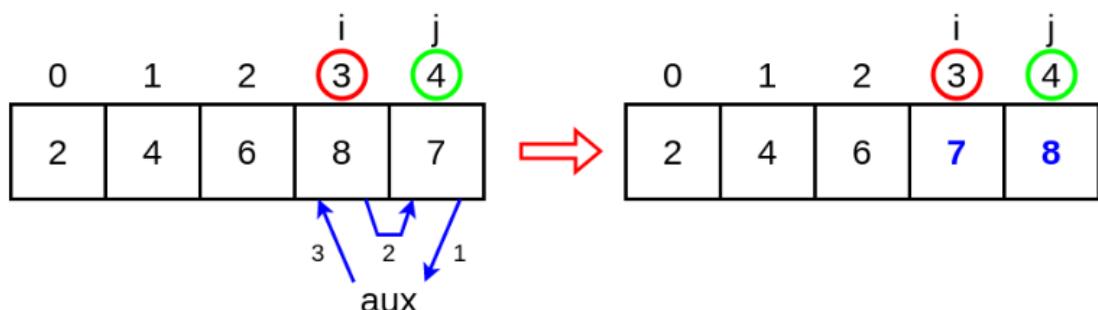
El valor de *i* ahora es 3.

El valor de *j* ahora es 4.



Supongo que ya te habrás dado cuenta. El valor que hay en la posición  $j$  (7) es menor que el valor que hay en la posición  $i$  (8), por tanto, la condición del `if (array[j]<array[i])` se cumple y se produce el intercambio entre ambas posiciones:

1. El valor 7 de la posición 4 se copia en la variable `aux`.
2. El valor 8 de la posición 3 se copia en la posición 4.
3. El valor 7 almacenado en `aux` se copia en la posición 3.



El bucle interno finaliza sus iteraciones y el bucle externo también, puesto que ha alcanzado la penúltima posición (tal y como vimos al principio). La ejecución termina y el `array` queda ordenado de menor a mayor. Compruébalo:

```

1 | for (int i = 0; i < array.length; i++) {
2 |     System.out.println(array[i]);
3 |

```

### - De forma descendente (de mayor a menor)

¿Cómo sería el algoritmo si lo que buscamos es ordenar el `array` de forma descendente? Pues exactamente igual, la única diferencia es que la comprobación que tenemos que hacer en el `if` ahora es: `array[j]>array[i]` en vez de `array[j]<array[i]`:

```
1 | for (int i = 0; i < array.length-1; i++) {  
2 |     for (int j = i+1; j < array.length ; j++) {  
3 |         if (array[j]>array[i]){  
4 |             int aux = array[j];  
5 |             array[j] = array[i];  
6 |             array[i] = aux;  
7 |         }  
8 |     }  
9 | }
```

De esta forma, el intercambio se produce cuando el valor de la posición  $j$  sea menor que el valor de referencia de la posición  $i$ . Así conseguimos que quede ordenado de mayor a menor.

---

**ACTIVIDAD.** Analiza paso a paso el algoritmo descendente de la misma forma que lo hemos hecho en el caso anterior.

---

## 2.1. Otros algoritmos de ordenación

---

¿Qué es un algoritmo de ordenación?

- Es una forma estructurada que indica cómo ordenar los elementos que hay dentro.

¿Son todos los algoritmos iguales?

- No, algunos son más eficientes que otros.
- 

### Algoritmos '*in-place*' y algoritmos '*no in-place*'

Un algoritmo «*no in-place*» es un algoritmo que no puede realizarse sin utilizar un espacio adicional para guardar datos mientras se procesa. Por ejemplo, si se tiene una lista de números y se quiere ordenar utilizando un algoritmo «*no in-place*», se necesitaría un *array* adicional para almacenar los números mientras se van ordenando, ya que no puede realizar la operación de ordenación sin modificar los datos originales y, por lo tanto, necesita un lugar donde almacenarlos mientras los procesa.

En cambio, un algoritmo «*in-place*» es un algoritmo que puede realizar una operación sin necesidad de espacio adicional (puede procesar los datos directamente en el lugar donde se almacenan, sin necesidad de utilizar una estructura de datos adicional).

---

**ACTIVIDAD. ¿Qué tipo de algoritmo es el método por Selección visto en el apartado anterior?**

---

Los algoritmos «*in-place*» son más eficientes en términos de espacio, ya que no requieren espacio adicional para realizar la operación. Sin embargo, algunos algoritmos «*no in-place*» pueden ser más eficientes en términos de tiempo o pueden tener otras ventajas que los hagan más adecuados para ciertas tareas.

---

### El método burbuja (*Bubble Sort*)

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, intercambiar las posiciones hasta que terminen todos ordenados.

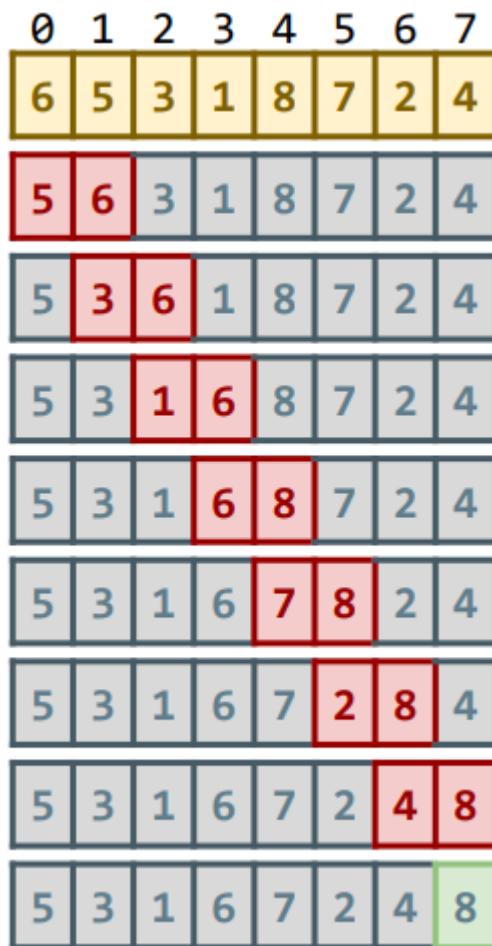
En cada iteración se comprueban los elementos adyacentes, de este modo, el máximo lo acabaremos situando a la derecha del *array*. Seguiremos haciendo lo mismo para el resto

del *array*, pero hasta una posición menos con cada iteración.

El código Java para la ordenación burbuja, de forma ascendente:

```
1 int aux;
2 for (int i = 0; i < vector.length; i++) {
3     for (int j = 0; j < vector.length - i - 1; j++) {
4         //El valor máximo lo más a la derecha posible
5         if (vector[j] > vector[j + 1]) {
6             aux = vector[j];
7             vector[j] = vector[j + 1];
8             vector[j + 1] = aux;
9         }
10    }
11 }
```

En el siguiente ejemplo se puede observar un ejemplo de cómo se guarda en la última posición el valor máximo. Luego, se repetirá el proceso descartando mirar dicho elemento (recorriendo hasta *length - 1*), y así sucesivamente:



Supongamos que tenemos el siguiente *array*.

```
1 | int vector[] = {5, 3, 8, 4, 2};
```

El algoritmo realizará las siguientes comparaciones y cambios:

- **Primera iteración (i = 0):**

Compara 5 y 3, intercambia → {3, 5, 8, 4, 2}

Compara 5 y 8, no hace nada.

Compara 8 y 4, intercambia → {3, 5, 4, 8, 2}

Compara 8 y 2, intercambia → {3, 5, 4, 2, 8}

- **Segunda iteración (i = 1):**

Compara 3 y 5, no hace nada.

Compara 5 y 4, intercambia → {3, 4, 5, 2, 8}

Compara 5 y 2, intercambia → {3, 4, 2, 5, 8}

- **Tercera iteración (i = 2):**

Compara 3 y 4, no hace nada.

Compara 4 y 2, intercambia → {3, 2, 4, 5, 8}

- **Cuarta iteración (i = 3):**

Compara 3 y 2, intercambia → {2, 3, 4, 5, 8}

Al final del proceso, el *array* estará ordenado: {2, 3, 4, 5, 8}.

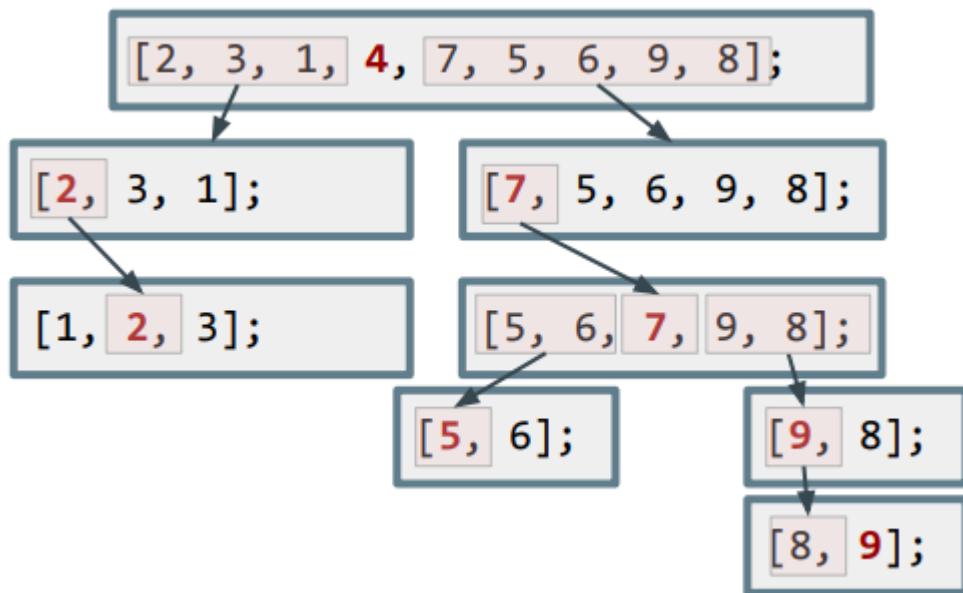
---

## Algoritmo de ordenación rápida - Quicksort

*Quicksort* es uno de los algoritmos de ordenación más rápidos que existe, y hace uso de la estrategia **divide y vencerás**. Trabaja de la siguiente forma:

```
1 | 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.

La eficiencia de este 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 el **peor caso**, el pivote termina en un extremo de la lista.

Ejemplo detallado de cómo ordenar este *array*: [4, 2, 7, 11, 1, 3, 10, 0, 9, 8, 6, 5]:

1. Seleccionamos un pivote: por ejemplo, el primer elemento de la lista, el 4 (podría ser cualquiera otra posición, incluso aleatoria).
2. Partimos la lista en dos sublistas: una con los elementos menores que 4 y otra con los elementos mayores que 4. La nueva lista se divide en dos sublistas, [2, 1, 3, 0] y [7, 11, 10, 9, 8, 6, 5].
3. Ordenamos cada sublista hasta que todos los elementos estén ordenados:
  - a. Ordenamos la sublista [2, 1, 3, 0] siguiendo los mismos pasos. Seleccionamos el primer elemento, 2, como pivote. Particionamos de nuevo la lista en [1, 0] y [3] y ordenamos re nuevo cada sublista hasta que todos los elementos estén ordenados.
  - b. Ordenamos la sublista [7, 11, 10, 9, 8, 6, 5] siguiendo los mismos pasos. Seleccionamos el primer elemento, 7, como pivote. Dividimos la lista en dos sublistas más [4, 6, 5] y [11, 10, 9, 8]. Continuamos ordenando cada sublista hasta que todos los elementos estén ordenados.
4. Una vez que todas las sublistas estén ordenadas, se combinan entre ellas para formar la lista completa ordenada: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11].

*Quicksort* es un algoritmo «in-place», lo que significa que no requiere espacio adicional para ordenar los datos. Además, suele tener un rendimiento excepcional en la mayoría de los casos. Es **especialmente adecuado en entornos con restricciones de espacio y en**

situaciones en las que se necesita un algoritmo rápido y eficiente para ordenar los datos. Sin embargo, es importante tener en cuenta que *Quicksort* no es un algoritmo estable y que puede tener un rendimiento peor en casos concretos, como cuando el pivote seleccionado es muchas veces el elemento más pequeño o más grande del conjunto de datos.

---

## Otros algoritmos de ordenación

Existen otros algoritmos de ordenación que son igual de rápidos que *Quicksort*, incluso a veces mejorando el peor caso de *Quicksort*. Sin embargo, en la práctica *Quicksort* suele ser el mejor para arrays medianos o grandes. Entre los más conocidos están *Mergesort* y *Heapsort*.

### Ordenar 100 Arrays de un millón de enteros.

**8.24s**  
Quicksort

**10.94s**  
Heapsort

**12.07s**  
Mergesort

### Ordenar 1.000 Arrays de 100.000 enteros.

**7.16s**  
Quicksort

**10.27s**  
Heapsort

**13.45s**  
Mergesort

### Ordenar un millón de Arrays de 100 enteros.

**3.34s**  
Quicksort

**3.14s**  
Heapsort

**5.71s**  
Mergesort

### Ordenar 1 Array de 300 millones de enteros.

**34.43s**  
Quicksort

**136.88s**  
Heapsort

**51.82s**  
Mergesort

## 2.2. Cómo ordenar arrays en Java con el método sort() de la clase Arrays

---

Como ya sabemos, existe una clase en Java llamada *Arrays* que tiene algunos métodos ya implementados con funcionalidades útiles para trabajar con *arrays*. Uno de ellos es el método *sort()*. Esta forma de ordenar *arrays* es mucho más sencilla y automática que las anteriores, y además no tenemos que hacer prácticamente nada: Java lo hace todo por nosotros.

Si buscamos un poco en alguno de nuestros proyectos, podemos ver que método *sort()* utiliza internamente el algoritmo *Mergesort*,

```
@Contract(mutates = "param1")
public static void sort( @NotNull Object[] a) {
    if (LegacyMergeSort.userRequested)
        legacyMergeSort(a);
    else
        ComparableTimSort.sort(a, lo: 0, a.length, work: null, workBase: 0, workLen: 0);
}
```

```
private static void mergeSort(Object[] src,
                               Object[] dest,
                               int low,
                               int high,
                               int off) {
    int length = high - low;

    // Insertion sort on smallest arrays
    if (length < INSERTIONSORT_THRESHOLD) {
        for (int i=low; i<high; i++)
            for (int j=i; j>low &&
                 ((Comparable) dest[j-1]).compareTo(dest[j])>0; j--)
                swap(dest, j, b: j-1);
        return;
    }

    // Recursively sort halves of dest into src
    int destLow  = low;
    int destHigh = high;
    int srcLow   = off;
    int srcHigh  = srcLow + length;
```

Bueno, vamos al grano. ¿Cómo se usa?

```
1 | int [] array = {4,6,2,8,7};
2 |
```

```
Arrays.sort(array);
```

Simplemente con eso, el *array* quedará ordenado de forma ascendente. Basta decirle el *array* que queremos ordenar al método *sort()*, y el cambio quedará hecho sobre el propio *array*.

Podemos comprobar que ha funcionado mostrando el *array* por pantalla tras la ejecución de *sort()*:

```
1 | for (int i = 0; i < array.length; i++) {  
2 |     System.out.println(array[i]);  
3 | }
```

Si lo que buscamos es **ordenarlo de forma descendente (de mayor a menor)** debemos decirle al método *sort()* que lo ordene al revés, pero **OJO**.

Este proceso sólo funciona con *arrays* de objetos, no con *arrays* de tipos primitivos (como nos sucede con el método *.asList()*). Por tanto, si lo que buscamos es ordenar descendentemente el *array* del ejemplo (*int*), tenemos que cambiar el tipo y usar ***Integer*** en vez de *int* de la siguiente forma:

```
1 | Integer [] array = {4,6,2,8,7};  
2 | Arrays.sort(array, Collections.reverseOrder());
```

Como puedes comprobar, además del *array*, le debemos indicar *Collections.reverseOrder()*. De esta forma, la ordenación se realizará en orden inverso.

## 2.3. Cómo buscar en un array ordenado con el método `binarySearch()`

---

Como ya hemos ido comentando hasta ahora, cuando el *array* está ordenado, la búsqueda es mucho más eficiente. Partiendo de que el *array* ya está ordenado usando alguna de las opciones explicadas en los puntos anteriores, la forma más fácil de realizar la búsqueda es utilizar el método `binarySearch()` de la clase *Arrays*.

El método `binarySearch()` de la clase *Arrays* realiza una búsqueda binaria. Este tipo de búsqueda reduce las iteraciones que se realizan para encontrar el dato, puesto que va acotando los elementos del *array* en función del dato a buscar. Un ejemplo,

10	33	45	56	57	87	90	93	96	99
v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]	v[9]

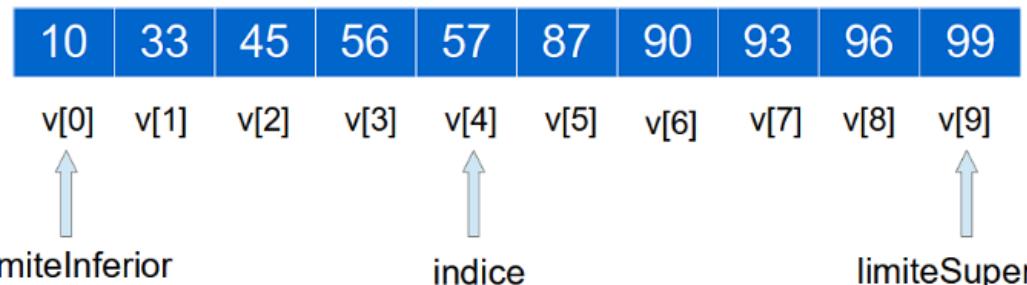
Como se puede apreciar, son 10 elementos de tipo entero ordenados de manera ascendente.

**Vamos a suponer que buscamos el elemento 45.** Nuestro algoritmo necesitará 3 variables de control:

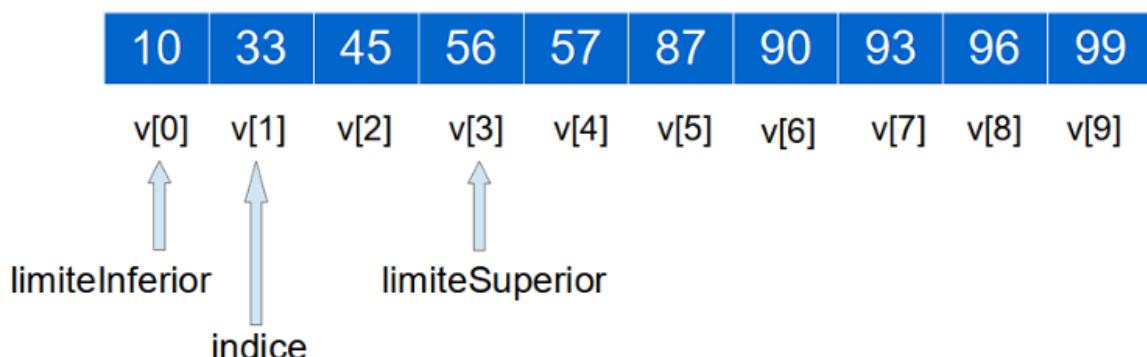
- Una variable llamada «`limiteInferior`». Esta variable la utilizaremos para delimitar desde qué elementos empezamos a buscar, empezando por el valor 0. En cada iteración el valor de esta variable puede cambiar.
- Una variable llamada «`limiteSuperior`». Esta variable la utilizaremos para delimitar desde qué elementos empezamos a buscar empezando por el valor número de elementos - 1. En cada iteración el valor de esta variable puede cambiar.
- Una variable llamada «`indice`». Esta variable será el elemento que buscaremos en cada iteración. Sí, se trata de un algoritmo iterativo. En cada iteración el valor de esta variable cambiará.

**Empezamos a buscar...**

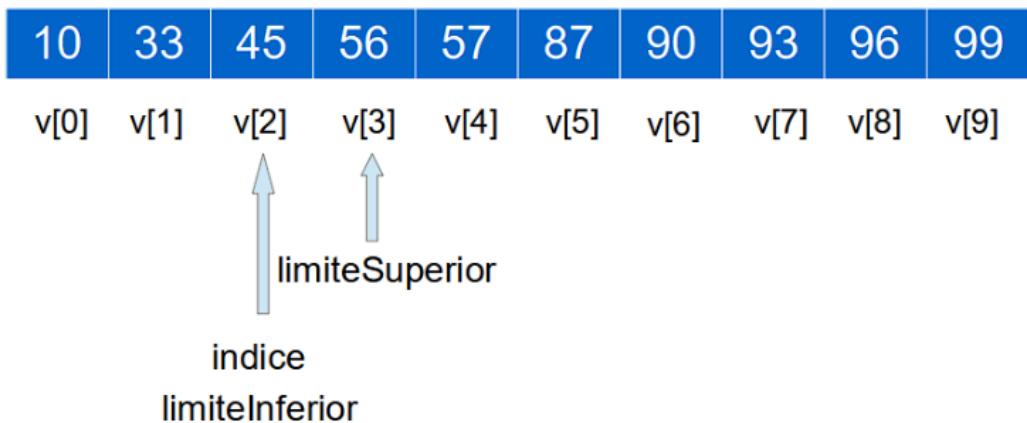
1. Seleccionaremos como «`limiteInferior`» la posición 0 de nuestro *array*.
2. Seleccionaremos como «`limiteSuperior`» la última posición de nuestro *array*. En este caso 9.
3. Calcularemos el valor de nuestra variable índice. «`indice`» = (`limiteInferior` + `limiteSuperior`) / 2. En este caso (0 + 9) / 2 = 4.5, al ser un entero se redondea a 4.



1. Comprueba si «v[indice]» es igual al elemento buscado. Recordad que estamos buscando el elemento «45».
  2. Como «v[4]» tiene un valor de «57», nuestro número, en caso de estar en el array tiene que estar hacia la izquierda de «v[4]», porque «v[4]=57» es mayor que «45».
  3. Cambiamos el valor del «límiteSuperior» al valor de «indice» de la iteración anterior restando 1. En este caso,  $4-1 = 3$ .
  4. «límiteInferior» se queda como está.
  5. Calcularemos el nuevo valor de nuestra variable índice. «indice» = ( $\text{límiteInferior} + \text{límiteSuperior}$ ) / 2. En este caso  $(0+3) / 2 = 1.5$ , al ser un entero se redondea a 1.



1. Comprueba si «v[indice]» es igual al elemento buscado. Recordad que estamos buscando el elemento «45».
  2. Como «v[1]» tiene un valor de «33», nuestro número en caso de estar en nuestro array tiene que estar hacia la derecha de «v[1]», porque «45» es mayor que «v[1]=33».
  3. Cambiamos el valor del «limiteInferior» al valor de «indice» de la iteración anterior sumando 1. En este caso  $1 + 1 = 2$ .
  4. «limiteSuperior» se queda como está, en 3.
  5. Calcularemos de nuevo el valor de nuestra variable índice. «indice» = ( $«limiteInferior» + «limiteSuperior») / 2$ . En este caso  $(2 + 3) / 2 = 2.5$ , al ser un entero se redondea a 2.



Venga, que en esta iteración ya vamos a encontrar el elemento buscado...

1. Comprueba si «v[indice]» es igual al elemento buscado. Recordad que estamos buscando el elemento «45».
2. En este caso «v[indice]», que es «v[2]» tiene un valor de «45», que es el número que estamos buscando, por lo tanto, aquí acaba nuestro algoritmo.

**ACTIVIDAD.** Busca en *Internet* algún algoritmo programado en *Java* que implemente la búsqueda binaria de forma "casera".

¡No te asistes! Aunque la lógica es bastante más compleja que la de una búsqueda normal, Java ya nos lo da hecho. Sólo tendremos que llamar al método *binarySearch()* para que haga la búsqueda binaria por nosotros. Ejemplo de cómo utilizarlo:

```

1  /* Declaraciones necesarias: */
2  int n = 9;
3  int [] nums = {2,2,4,4,7,9};
4  int pos = -1;
5
6
7  /* Búsqueda: */
8  int pos = Arrays.binarySearch(nums, n);
9
10 /* Comprobación del resultado: */
11 if(pos >= 0){
12     System.out.println("Número encontrado en la posición " + pos);
13 }else{
14     System.out.println("No se ha encontrado el número");
15 }
```

Necesitaremos tres variables:

1. El dato a buscar (variable *n*).
2. El *array* sobre el que realizar la búsqueda, que tendrá que estar ordenado (variable *nums*).
3. Un entero en el que guardar la posición que ocupa el dato en caso de existir en el *array* (variable *pos*).

La lógica es la misma que en el caso de la búsqueda normal: si el valor existe en el *array*, la variable *pos* tomará el valor de la primera posición en la que se encuentre dicho valor. En caso contrario, la variable *pos* tomará un valor negativo.

# Actividad. Eliminar duplicados de un array

---



## Actividad. Eliminar duplicados de un array

---

- Eliminar duplicados usando un array auxiliar y copiando elementos a array nuevo.
  - Eliminar duplicados modificando en directo el mismo array y copiando elementos a array nuevo.
  - Volver a recalcular el valor duplicado aleatoriamente.
- 



Obra publicada con Licencia Creative Commons Reconocimiento Compartir igual 4.0  
<http://creativecommons.org/licenses/by-sa/4.0/>

# Eliminar duplicados de un array con el método `.distinct()`

## Eliminar duplicados de un array con el método `.distinct()`

Vale, ahora la forma fácil...

---

Pues sí, en este caso, igual que cuando ordenamos o buscamos, existe un método que nos simplifica nuestro código cuando necesitamos eliminar elementos duplicados, incluso sin estar ordenados. Se trata del método `.distinct()`, perteneciente a la clase `Arrays`.

La forma de usarlo es similar a los métodos vistos anteriormente. Llamaremos al método con el *array* al que necesitemos eliminarle los elementos duplicados, y nos devolverá otro *array* nuevo que esté "limpio":

```
1 | int vector[] = {3,3,7,8,8,9,10,15,15};  
2 | int vector2[] = Arrays.stream(vector).distinct().toArray();  
3 | System.out.println(Arrays.toString(vector));  
4 | System.out.println(Arrays.toString(vector2));
```



Obra publicada con Licencia Creative Commons Reconocimiento Compartir igual 4.0  
<http://creativecommons.org/licenses/by-sa/4.0/>

## PRÁCTICA 2. SIMULACIÓN DE SORTEO



**LA PRIMITIVA**

### → INTRODUCCIÓN. Método *matches()*

El método *matches* de la clase *String* nos permite comprobar si una cadena cumple con un formato. Si coincide devuelve TRUE, y si no, FALSE. Por ejemplo, si quisiéramos controlar que lo que tiene la cadena son números y no otra cosa, utilizaremos la expresión `\d` (dígito) en el rango [0-9].

Para esta práctica es necesario que controlemos un formato numérico con ciertos separadores. Por no hacerlo igual, y remitiéndonos a un ejemplo parecido, volveremos a la práctica del Número de la Suerte del tema anterior. En aquella práctica debíamos controlar que el formato de la fecha introducida por el usuario fuera *dd/mm/aaaa*. Si hubiéramos querido controlarlo con el método *matches*,

***fecha.matches("|\d{1,2}/|\d{1,2}/|\d{4}")***

de esta forma, le estamos indicando que debe revisar si el *String* introducido contiene de 1 a 2 números antes de la primera barra, de 1 a 2 números después de la primera barra y 4 números sí o sí después de la segunda barra.

Si sólo fuera válido un dígito, no haría falta indicarle nada entre los corchetes {}. Por ejemplo:

***boolean formato = prueba.matches("|\d-|\d-|\d");***

En este caso, estaríamos comprobando que el formato del *String* tuviera una estructura tal que así: 4-3-3. Cualquier cosa distinta a eso, devolverá valor FALSE.

Iremos introduciendo poco a poco las posibilidades que tenemos utilizando este método, pero de momento, para esta práctica es suficiente.

### → EL JUEGO DE LA PRIMITIVA

La Lotería Primitiva es un juego de azar regulado por *Loterías y Apuestas del Estado (LAE)*, que consiste en elegir 6 números diferentes entre 1 y 49, con el objetivo de acertar la combinación ganadora en el sorteo correspondiente.

El juego está formado por **7 bolas**, de las cuales:

- 6 bolas se extraen de un bombo con 49 números, y
- 1 bola se extrae de otro bombo con 10 bolas (con números que van desde 0 a 9), correspondiente al «**reintegro**».

También se extrae una **bola extra como número complementario** del bombo de 49 números. Por lo tanto, en el sorteo **se extraen un total de 8 bolas (6 números + 1 reintegro + 1 complementario)**.

Ejemplo de sorteo:

Lotería Primitiva de España / Sorteo 136 (11/11/24)

06 14 22 35 36 37 23\* 01\*

Complementario: 23  
Reintegro: 01

Existen varias categorías de acertantes, dependiendo de los números que se aciertan:

- Categoría Especial: acertar los seis números de la combinación ganadora y el reintegro.
- 1<sup>a</sup> Categoría: acertar los seis números de la combinación ganadora.
- 2<sup>a</sup> Categoría: acertar cinco números de la combinación y el número complementario.
- 3<sup>a</sup> Categoría: acertar cinco números de la combinación.
- 4<sup>a</sup> Categoría: acertar cuatro números de la combinación.
- 5<sup>a</sup> Categoría: acertar tres números de la combinación.
- Reintegro: acertar el número del reintegro.
- No premiado.

#### → PROBLEMA A RESOLVER

Realiza un programa que simule el sorteo de “La Primitiva” y permita validar al usuario su boleto, indicándole qué categoría de acertante ha obtenido.

- a) El boleto se rellena de la siguiente forma:

El usuario debe seleccionar 6 números comprendidos entre el 1 y el 49. Además, debe elegir un número del 0 al 9 como reintegro. Es decir, el usuario debe introducir 7 números con el siguiente formato: **N-N-N-N/R**

- b) El programa debe validar el formato de entrada (números y separadores) usando el método `.matches()`. En caso de no ser un formato válido, el programa debe finalizar.
- c) En caso de ser válidos los datos introducidos por el usuario, deben guardarse en un vector. Usa el método `.split` con doble separador: `.split("-/-")`.
- d) El programa debe sortear las 6 bolas entre una lista de 49 números y guardar los valores en un vector, de forma ordenada. En caso de que alguno de los números del sorteo se repita (haya duplicados en el vector), se debe volver a recalcular. Por ejemplo,

```
Introduce los datos de tu boleto:  
23-44-11-7-34-2/7  
[23, 44, 11, 7, 34, 2, 7]  
Ha salido:  
[4, 24, 24, 31, 39, 39]
```

- e) El programa sorteará también un número complementario de forma aleatoria (1-49). Si el número complementario sorteado coincide con alguno de los números ya sorteados anteriormente (guardados en el vector), debe volver a recalcularlo.
- f) Por último, el programa sorteará el reintegro entre una lista con números del 0 al 9.

```
Introduce los datos de tu boleto:  
23-44-11-7-34-2/7  
[23, 44, 11, 7, 34, 2, 7]  
Ha salido:  
[4, 9, 20, 26, 31, 34]  
Complementario: 48  
Reintegro: 7  
  
Process finished with exit code 0
```

- g) Con todos los datos recogidos y sorteados, el programa debe comprobar los resultados del sorteo con los datos del boleto introducido por el usuario, y mostrar por pantalla la categoría de acertantes a la que pertenece.

## EJEMPLO.

```
Introduce los datos de tu boleto:  
23-44-11-7-34-2/7  
[23, 44, 11, 7, 34, 2, 7]  
  
SORTEO:  
[2, 11, 25, 33, 41, 47]  
Complementario: 39  
Reintegro: 7  
  
RESULTADOS:  
1 acierto.  
Reintegro.
```

Realiza un programa en *Java* que implemente la lógica del sorteo explicada anteriormente, usando vectores y los métodos mencionados.

### → REALIZACIÓN DE LA PRÁCTICA

Sigue los siguientes pasos para realizar la práctica. **¡Ve guardando tu trabajo de vez en cuando para evitar que se borre el avance si se cierra el editor de textos u ocurre cualquier problema en tu equipo!**

1. Programa en Java la aplicación requerida
2. Plan de pruebas. Realiza las pruebas necesarias para comprobar que el programa funciona bien



### ENTREGA

**REALIZA UN INFORME EN PDF CON LA INFO GENERADA Y LOS PASOS SEGUIDOS PARA REALIZAR ESTA PRÁCTICA. EXPLICA TU CÓDIGO. SÚBELO TODO A LA TAREA DE AULES DISPONIBLE.**

**ADEMÁS, PEGA LA URL DE TU PROYECTO EN GITHUB.**

# Etiquetar bucles anidados

## BONUS [pre-matrices]. Etiquetar (bautizar) bucles

### Etiquetas en bucles anidados

En Java, las etiquetas se pueden usar con los comandos *continue* y *break*. Una etiqueta se coloca antes de la sentencia a etiquetar seguida de ":".

Son útiles cuando tenemos bucles anidados y queremos especificar en cuál de ellos queremos hacer un *break* o *continue*. Por ejemplo, si tenemos 3 bucles anidados y cuando se cumple alguna condición necesitamos salir de dos de ellos, pero no de todos. Las etiquetas son una forma "elegante" de hacerlo.

```
1  bucle1:  
2      for (int i = 0; i < 10; i++){  
3          bucle2:  
4              for (int j = 0; j < 10; j++){  
5                  bucle3:  
6                      for (int k = 0; k < 10; k++){  
7                          if (i == j && j == k){  
8                              break bucle2;  
9                          }  
10                     }  
11                 }  
12             }  
13         }  
14     }
```

En este ejemplo, cuando se ejecuta la instrucción *break*, no salimos del bucle con la variable *k*, sino del bucle etiquetado como *bucle2*, es decir, salimos inmediatamente de los dos bucles con *k* y *j*. El funcionamiento normal del *break* normal sería salir del bucle más interno, el etiquetado como *bucle3* en este caso. Sin embargo, el *break* etiquetado hace que rompa el bucle etiquetado, es decir el *for* etiquetado como *bucle2*.

Las etiquetas funcionan para cualquier tipo de bucle, no solamente para los de tipo *for*. Otro ejemplo:

```
1 boolean esVerdadero = true;
2     externo: //etiqueta la siguiente sentencia, es decir el for.
3     for (int i = 0; i<5; i++){
4         while (esVerdadero){
5             System.out.println("Hola!");
6             break externo; //break con etiqueta, hace que rompa
7         }
8         System.out.println("Despues del while!");
9     }
10    System.out.println("Despues del for!");
11 }
```

# Batería de ejercicios pre-matrices: bucles anidados

---



## Ejercicio 1. Realiza una traza de los siguientes programas:

---

a)

```
int suma;
for (int i=0;i<4;i++){
    for (int j=3;j>0;j--){
        suma=i*10+j;
        System.out.println(suma);
    }
}
```

b)

```
int j;
for (int i=0;i<3;i++){
    j=i+1;
    while(j<4){
        System.out.println(j-i);
        j++;
    }
}
```



## Ejercicio 2. Triángulo.

---

Crea un programa que imprima un triángulo de asteriscos con una altura definida por el usuario.

Entrada:

Altura: 4

Salida:

```
*  
**  
***  
****
```



## Ejercicio 3. Tablas de multiplicar.

Diseña un programa que muestre las tablas de multiplicar del 1 al 10.



## Ejercicio 4. Números primos.

Se pide realizar un programa que calcule los números primos que se encuentran en el rango desde 2 hasta  $m$ . El programa debe pedir un número máximo  $m$  e imprimir todos los primos en el rango indicado.

```
Introduzca el valor m: 21  
Números primos: 2 3 5 7 11 13 17 19
```



Obra publicada con Licencia Creative Commons Reconocimiento Compartir igual 4.0  
<http://creativecommons.org/licenses/by-sa/4.0/>

### 3. Matrices

## 3. Matrices (arrays bidimensionales)

### Introducción a las matrices

Una matriz o *array* multidimensional es aquel que para acceder a una posición concreta, en vez de utilizar un único valor como índice (por ejemplo,  $v[2]$ ), se utiliza una secuencia de varios índices ( $m[2][4]$ ). Cada índice sirve como coordenada para una dimensión diferente. En esta unidad nos centraremos en los de 2 dimensiones.

		Columnas			
		0	1	2	3
Filas	0	2	4	5	8
	1	6	3	1	9

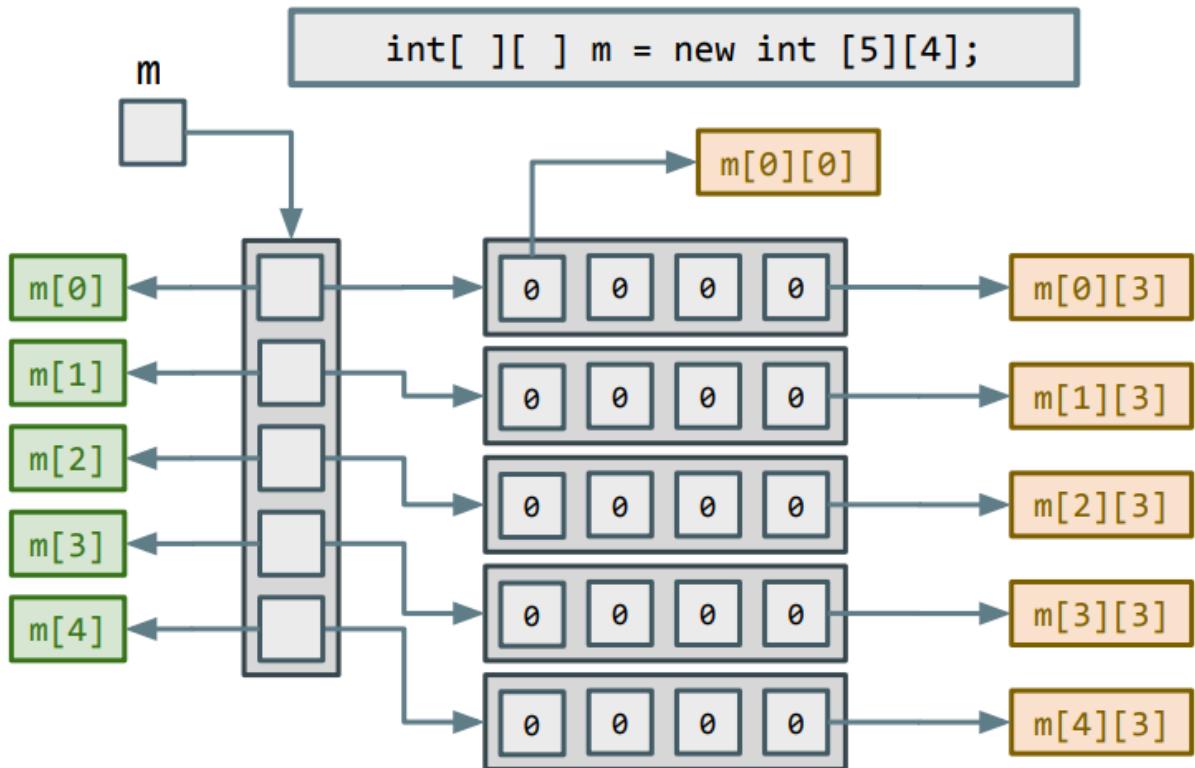
La sintaxis es como la de los vectores o *arrays*, pero se le añaden unos corchetes adicionales:

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

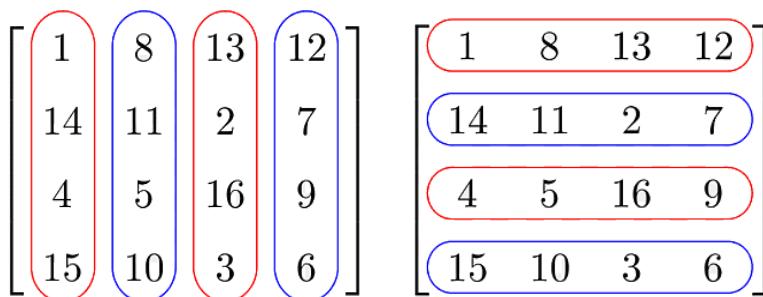
```
char letras[][];  
int precios[][] = new int[5][6];  
double[][] notas = new double[10][10];  
String ciudades[][] = new String[10][5];
```

y en este caso, también da igual si los corchetes se escriben junto al tipo de dato o junto al identificador (nombre) de la matriz. A tu gusto.

Como puedes ver en el ejemplo de la creación, **una matriz se puede interpretar como una tabla cuya primera dimensión serían las filas y la segunda serían las columnas.**



También se puede ver como una representación de diferentes vectores juntos, ya sean horizontales o verticales:



### 3.1. Inicializar una matriz

---

Podemos inicializar los valores de una matriz poniendo los elementos de cada vector (o *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

Los índices de las posiciones de cada elemento en una matriz como la anterior, de 4 filas y 5 columnas, serían:

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 lo tanto, como ves, el acceso a las posiciones de una matriz se hace de la siguiente manera:

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

Otro ejemplo:

```
1 |     int matriz[][] = {{1,4,5},{6,7,2},{8,3,8}};  
2 |     System.out.println(matriz[0][1]); //4
```

```
3 |     System.out.println(matriz[1][2]); //2
4 |     System.out.println(matriz[2][0]); //8
```

## 3.2. Recorrer una matriz

---

Como es de esperar, en el caso de las matrices también necesitaremos ayudarnos con bucles *for* para recorrerlas. Pero, ¿con qué condición?

### Comportamiento de *length*

Como siempre, podemos obtener tanto la cantidad de filas como la cantidad de elementos de cada fila (columnas) con el método *length*. Por ejemplo, si a una matriz la identificamos como *a*:

<i>a[0][0]</i>	<i>a[0][1]</i>	<i>a[0][2]</i>	<i>a[0][3]</i>	<i>a[0][4]</i>
<i>a[1][0]</i>	<i>a[1][1]</i>	<i>a[1][2]</i>	<i>a[1][3]</i>	<i>a[1][4]</i>
<i>a[2][0]</i>	<i>a[2][1]</i>	<i>a[2][2]</i>	<i>a[2][3]</i>	<i>a[2][4]</i>
<i>a[3][0]</i>	<i>a[3][1]</i>	<i>a[3][2]</i>	<i>a[3][3]</i>	<i>a[3][4]</i>

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

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

### Recorrer matriz

Para recorrer un *array* de varias dimensiones lo podemos hacer con dos bucles *for* anidados.

- Los bucles anidados utilizarán dos índices, y 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 matrices, la condición del bucle externo debe hacer referencia a la cantidad de filas, mientras que la condición del bucle interno debe hacer referencia a la cantidad de elementos de la fila actual (columnas).

```

1 //declaración y inicialización
2 int[][] matriz = {{1,4,5},{6,7,2},{8,3,8}};
3
4 //recorrido de la matriz
5 for (int fila = 0; fila < matriz.length; fila++) {
6
7     for (int columna = 0; columna < matriz[fila].length; columna++) {
8         System.out.print(matriz[fila][columna]+" ");
9     }
10
11    System.out.println();
12
13 }
```

## Rellenar matriz desde teclado

Puede ser que nos interese llenar los valores de una matriz desde el teclado, pidiéndoselo al usuario. La forma de hacerlo es la siguiente:

```

Scanner sc = new Scanner(System.in);

System.out.print("Introduce el número de filas: ");
int filas = sc.nextInt();
System.out.print("Introduce el número de columnas: ");
int columnas = sc.nextInt();

int matriz[][] = new int[filas][columnas];

//proceso de llenar la matriz
for (int i = 0; i < filas; i++) {
    for (int j = 0; j < columnas; j++) {
        System.out.print("Valor para la posición [" + i + "][" +
        matriz[i][j] = sc.nextInt();
    }
}

//mostramos el resultado
```

```
20     System.out.println("La matriz ha quedado de la siguiente manera");
21     for (int i = 0; i < filas; i++) {
22         for (int j = 0; j < columnas; j++) {
23             System.out.print(matriz[i][j] + " ");
24         }
25     }
26     System.out.println(); //insertamos un salto de línea al final
```

Fíjate que en este ejemplo no hemos hecho referencia al método *length*, ya que como teníamos guardado el tamaño en las variables *filas* y *columnas*, no es necesario calcularlos para recorrer la matriz.

### 3.3. Recorrer e imprimir array bidimensional con forEach

Como ya vimos con vectores, existen formas alternativas de recorrerlos. Para *arrays* de 2 dimensiones podemos hacerlo de forma abreviada de la siguiente manera:

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

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

### Imprimir matriz

Si lo que queremos es imprimir el contenido de una matriz podemos hacerlo usando el método *.toString()* de la clase *Arrays*, pero recorriendo las filas previamente:

```
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]
```

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

# Batería de ejercicios sobre matrices (nivel 1)

---



## Ejercicio 1: Búsqueda en una matriz

---

Usa etiquetas para salir de los bucles anidados cuando encuentres el número.

1. Sigue al usuario que llene una matriz de tamaño 3x3 con números enteros.
2. Luego, pide al usuario un número a buscar.
3. Indica si el número se encuentra en la matriz y muestra las coordenadas donde aparece (si existe).

```
Matriz:  
1 2 3  
4 5 6  
7 8 9  
Número a buscar: 5  
  
Salida:  
El número 5 se encuentra en la posición (1, 1).
```



## Ejercicio 2. Suma de filas y columnas de una matriz

---

Crea un programa que:

1. Cree una matriz de tamaño aleatorio (entre 1 y 5 filas/columnas) y la rellene de números aleatorios.
2. Calcule la suma de cada fila y de cada columna.
3. Imprima los resultados.

```

Matriz:
1 2 3
4 5 6
7 8 9

Suma de filas:
Fila 1: 6
Fila 2: 15
Fila 3: 24

Suma de columnas:
Columna 1: 12
Columna 2: 15
Columna 3: 18

```

---



## Ejercicio 3: Tabla de calificaciones

---

Un profe necesita ayuda para almacenar las notas de su alumnado.

Quiere guardar en una matriz el nombre de sus estudiantes junto a las notas obtenidas por asignatura, de la siguiente manera:

Estudiantes	Mates	Biología	Inglés	Programación
Juan	2	5	5	3
Pepe	7	7	7	7
Luis	5	5	7	4
Marta	6	7	8	5
Carla	4	6	6	5

- a) Diseña un programa que pregunte cuántos estudiantes tiene al usuario. Pregunta también cuántas asignaturas.
- b) Crea una matriz lo suficientemente grande para almacenar la información que necesita el usuario.
- c) Rellena la matriz con los nombres y asignaturas requeridas por el profesor (debe introducirlos por teclado). Los nombres se deben almacenar a lo largo de la primera columna, y las asignaturas a lo largo de la primera fila (a excepción de la posición 0, donde aparecen los nombres del alumnado).

Estudiantes	Mates	Biología	Inglés	Programación
Juan				
Pepe				
Luis				
Marta				
Carla				

```
Hola! Cuántos estudiantes tienes?
3
Cuántas asignaturas tienen?
4
Introduce el nombre del estudiante 1
juan
Introduce el nombre del estudiante 2
carlos
Introduce el nombre del estudiante 3
pepe
Introduce la asignatura 1
mat
Introduce la asignatura 2
bio
Introduce la asignatura 3
ing
Introduce la asignatura 4
pro
Inserta la nota de juan para la asignatura de mat
3
```

- d) Una vez tenemos la matriz como en la tabla anterior, pediremos las notas por alumno. Es decir, deberemos insertar todas las materias para Juan antes de pasar a insertar todas las materias de Pepe.

Estudiantes	Mates	Biología	Inglés	Programación
Juan	2	5	5	3
Pepe	7	7	7	7
Luis	5	5	7	4
Marta	6	7	8	5
Carla	4	6	6	5

```
pro
Inserta la nota de juan para la asignatura de mat
3
Inserta la nota de juan para la asignatura de bio
4
Inserta la nota de juan para la asignatura de ing
5
Inserta la nota de juan para la asignatura de pro
4
Inserta la nota de carlos para la asignatura de mat
5
Inserta la nota de carlos para la asignatura de bio
4
Inserta la nota de carlos para la asignatura de ing
6
Inserta la nota de carlos para la asignatura de pro
7
Inserta la nota de pepe para la asignatura de mat
4
Inserta la nota de pepe para la asignatura de bio
3
```

```
Inserta la nota de pepe para la asignatura de mat
4
Inserta la nota de pepe para la asignatura de bio
3
Inserta la nota de pepe para la asignatura de ing
2
Inserta la nota de pepe para la asignatura de pro
4
Estudiantes mat bio ing pro
juan 3 4 5 4
carlos 5 4 6 7
pepe 4 3 2 4
```

e) Cuando ya tengamos la matriz completa, realizaremos los siguientes cálculos:

- Nota media de todas las asignaturas por alumno.
- Nota media por asignaturas.

```
Estudiantes mat bio ing pro
juan 3 4 5 4
carlos 5 4 6 7
pepe 4 3 2 4
La nota media del alumno juan es 4.0
La nota media del alumno carlos es 5.5
La nota media del alumno pepe es 3.25
La nota media de la asignatura mat es 4.0
La nota media de la asignatura bio es 3.6666667
La nota media de la asignatura ing es 4.3333335
La nota media de la asignatura pro es 5.0

Process finished with exit code 0
```



Obra publicada con Licencia Creative Commons Reconocimiento Compartir igual 4.0  
<http://creativecommons.org/licenses/by-sa/4.0/>

## 4. Tratamiento de vectores en matrices

### 4. Tratamiento de vectores en matrices

#### Rellenar matriz con vectores

Hasta ahora, hemos hablado todo el rato de que una matriz está compuesta de vectores... pero nunca hemos creado una a partir de estos.

Para poder hacerlo tenemos varias opciones. Desde la más fácil hasta la más difícil...

- **Opción fácil:** ya tengo mi vector y sé en qué fila de mi matriz lo quiero introducir.

```
1     int vector[] = {3,5,4,1};  
2     int matriz[][] = new int[2][4];  
3  
4     for (int i = 0; i < matriz[0].length; i++) {  
5         matriz[0][i] = vector[i];  
6     }  
7  
8     for (int[] filas : matriz) {  
9         for (int columnas : filas){  
10             System.out.print(columnas + " ");  
11         }  
12         System.out.print("\n");  
13     }
```

3	5	4	1
0	0	0	0

Sólo nos hace falta un bucle para recorrer las columnas, ya que la fila donde se insertará el vector es fija.

- **Opción 2:** voy creando vectores y guardándolos en una matriz automáticamente.

```

1     Scanner teclado = new Scanner(System.in);
2
3     int matriz[][] = new int[4][4];
4
5     for(int i=0; i<matriz.length; i++){
6         System.out.print("Ingresa el vector: ");
7         String[] lectura = teclado.next().split(",");
8         for(int j=0; j<matriz[i].length; j++){
9             matriz[i][j] = Integer.parseInt(lectura[j]);
10        }
11    }
12
13    for(int[] filas : matriz){
14        for(int columnas : filas){
15            System.out.print(columnas + " ");
16        }
17        System.out.print("\n");
18    }

```

```

Ingresá el vector: 2,3,4,5
Ingresá el vector: 3,2,4,5
Ingresá el vector: 2,2,2,2
Ingresá el vector: 6,7,8,4
2 3 4 5
3 2 4 5
2 2 2 2
6 7 8 4

```

En este caso tendremos que llevar cuidado, ya que, **si la cadena que introducimos y convertimos en vector con `.split()` no tiene el mismo tamaño que número de columnas de nuestra matriz, los valores no se insertarán correctamente en la matriz (o directamente aparecerán errores de que está intentando acceder a posiciones que no existen en el vector).**

```

Ingresá el vector: 1,2,3,4
Ingresá el vector: 1,2,3
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3 out of bounds for length 3
    at org.example.Matrices.insertar_vector(Matrices.java:218)
    at org.example.Main.main(Main.java:59)

```

## Comparar valores de un vector con los de una matriz

Si necesitamos recorrer una matriz comprobando si los valores en las distintas posiciones coinciden con los de un vector, solamente necesitaremos un bucle (para dejar fija bien la

fila o bien la columna).

- Para comparar valores en una fila concreta (horizontalmente),

```
1 int vector[] = {1,4,5,8};
2     System.out.println(Arrays.toString(vector));
3
4 int matriz[][] = {{3,2,5,4},{1,4,5,8},{9,4,3,5}};
5
6 for(int[] filas : matriz){
7     for(int columnas : filas){
8         System.out.print(columnas + " ");
9     }
10    System.out.print("\n");
11 }
12
13 System.out.println("Voy a comprobar la fila con índice 1");
14 for (int i = 0; i < matriz[1].length; i++) {
15     if(matriz[1][i]!=vector[i]){
16         System.out.println("No son iguales.");
17         return;
18     }
19 }
20
21 System.out.println("Son iguales.");
22 }
```

```
[1, 4, 5, 8]
3 2 5 4
1 4 5 8
9 4 3 5
Voy a comprobar la fila 1
Son iguales.
```

- Para comparar valores en una columna concreta (verticalmente),

```
1 int vector[] = {4,8,5};
2     System.out.println(Arrays.toString(vector));
3
4 int matriz[][] = {{3,2,5,4},{1,4,5,8},{9,4,3,5}};
5
6 for(int[] filas : matriz){
```

```
7         for(int columnas : filas){
8             System.out.print(columnas + " ");
9         }
10        System.out.print("\n");
11    }
12
13    System.out.println("Voy a comprobar la columna con índice 3");
14    for (int i = 0; i < matriz.length; i++) {
15        if(matriz[i][3]!=vector[i]){
16            System.out.println("No son iguales.");
17            return;
18        }
19    }
20
21    System.out.println("Son iguales.");
22
23 }
```

```
[4, 8, 5]
3 2 5 4
1 4 5 8
9 4 3 5
Voy a comprobar la columna con índice 3
Son iguales.
```



Obra publicada con Licencia Creative Commons Reconocimiento Compartir igual 4.0  
<http://creativecommons.org/licenses/by-sa/4.0/>

## PRÁCTICA 3. SOPA DE LETRAS AUTOMATIZADA

E	R	Í	A	Ü	R	O	B	O	T	C	B
Y	Z	F	B	D	X	X	É	D	E	A	E
S	O	R	W	Y	K	L	Z	Ú	W	B	Ü
C	N	O	L	O	G	Í	A	I	I	L	P
M	Í	D	Í	J	F	M	U	K	T	E	R
O	Ñ	O	Ü	R	C	K	K	B	J	S	O
T	Y	M	Ó	F	Ñ	H	V	É	E	I	Y
O	O	C	Í	X	P	Ñ	I	L	Á	A	E
R	Y	É	Ü	Y	S	D	Ñ	P	P	Ñ	C
U	Í	V	A	P	A	T	D	Z	Á	Í	T
V	G	Á	H	V	D	F	J	É	N	Y	O

### → INTRODUCCIÓN. El método *matches()* vol. 2

Como ya sabemos, el método *matches()* de la clase *String* nos permite comprobar que el formato de una cadena introducida es correcto gracias a las expresiones de formato que es capaz de validar. En este caso, como lo que va a introducir el usuario son los datos de una sopa de letras, deberemos controlar que los valores son todos caracteres.

La forma de validar letras con el método *matches*, dando por buenas tanto minúsculas como mayúsculas, es la siguiente:

```
lectura.matches("[a-zA-Z]+")
```

En caso de no corresponder con el formato esperado, el método devuelve el valor *false*. Utilízalo para implementar el programa de la práctica que se explica a continuación.

### → PROBLEMA A RESOLVER

Dada una matriz de M x N letras y una palabra, tenemos que programar un código que encuentre la ubicación en la matriz en la que se puede localizar dicha palabra.

La palabra debe coincidir con una línea recta e ininterrumpida de letras en la matriz (mayúsculas y minúsculas se consideran equivalentes). Es decir, si buscamos *sopa*, podemos considerar *Sopa* o *sOPa* como coincidencias válidas.

La coincidencia solamente se puede dar en dirección horizontal derecha y dirección verticales abajo (las coincidencias hacia arriba o hacia la izquierda y las diagonales no se deben tener en cuenta).

## ENTRADA

El programa recibirá una entrada con el siguiente formato:

1. La primera línea preguntará y leerá un par de números enteros M (filas) y N (columnas).
2. Las siguientes M líneas de entrada contendrán N letras cada una, representando la matriz de letras donde debe buscarse la palabra. Las letras pueden estar en mayúsculas o minúsculas.

```
Introduce el número de filas :  
3  
Introduce el número de columnas:  
4  
Introduce las letras de la fila 1:aGfj  
Introduce las letras de la fila 2:hola  
Introduce las letras de la fila 3:RthV  
a G f j  
h o l a  
R t h V
```

En caso de no haber introducido **N letras**, se debe mostrar un error y finalizar el programa.

```
Introduce el número de filas:  
2  
Introduce el número de columnas:  
3  
Introduce las letras de la fila 1:1aa  
ERROR. Introduce datos válidos: 3 letras.
```

3. La siguiente línea pedirá una palabra a buscar. Esta palabra sólo puede contener letras mayúsculas y minúsculas (sin espacios, guiones u otros caracteres no alfabéticos).

```
Introduce el número de filas :  
3  
Introduce el número de columnas:  
4  
Introduce las letras de la fila 1:aGfj  
Introduce las letras de la fila 2:hola  
Introduce las letras de la fila 3:RthV  
a G f j  
h o l a  
R t h V  
Introduce la palabra a buscar:  
hola
```

## SALIDA

El programa devolverá un par de números enteros que representan la ubicación en la matriz de la palabra (estarán separados por un espacio):

- El primer entero es la fila de la matriz donde se puede encontrar la primera letra de la palabra dada.
- El segundo entero es la columna de la matriz donde se puede encontrar la primera letra de la palabra dada.

Si una palabra se puede encontrar más de una vez en la matriz se debe devolver la ubicación de la ocurrencia más alta de la palabra, es decir, aquella que sitúa la primera letra de la palabra más cerca de la parte superior izquierda de la matriz. Se asumirá que la palabra se encuentra al menos una vez en la matriz.

## EJEMPLOS

```
Introduce el número de filas (:  
3  
Introduce el número de columnas:  
4  
Introduce las letras de la fila 1:aGfj  
Introduce las letras de la fila 2:hola  
Introduce las letras de la fila 3:RthV  
a G f j  
h o l a  
R t h V  
Introduce la palabra a buscar:  
hola  
Encontrada!!! En la posición 1 0
```

```
Introduce el número de filas:  
8  
Introduce el número de columnas:  
11  
Introduce las letras de la fila 1:abcDEFGhigg  
Introduce las letras de la fila 2:hEbKWalDorf  
Introduce las letras de la fila 3:FtaAwaldORM  
Introduce las letras de la fila 4:FtmimrLqsrc  
Introduce las letras de la fila 5:byBArBeTTYv  
Introduce las letras de la fila 6:KLIBqwikomk  
Introduce las letras de la fila 7:strEBGadhrb  
Introduce las letras de la fila 8:yUiqlxcnBjk  
a b c D E F G h i g g  
h E b k W a l D o r f  
F t a A w a l d O R m  
F t m i m r L q s r c  
b y B A r B e T T Y v  
K l I b q w i k o m k  
s t r E B G a d h r b  
y U i q l x c n B j k  
Introduce la palabra a buscar:  
bambi  
Encontrada!!! En la posición 1 2
```

Realiza un programa en *Java* que implemente la lógica del juego explicada anteriormente, usando matrices.

## → REALIZACIÓN DE LA PRÁCTICA

Sigue los siguientes pasos para realizar la práctica. ¡Ve guardando tu trabajo de vez en cuando para evitar que se borre el avance si se cierra el editor de textos u ocurre cualquier problema en tu equipo!

1. Programa en Java la aplicación requerida
2. Plan de pruebas. Realiza las pruebas necesarias para comprobar que el programa funciona bien



#### ENTREGA

**REALIZA UN INFORME EN PDF CON LA INFO GENERADA Y LOS PASOS SEGUIDOS PARA REALIZAR ESTA PRÁCTICA. EXPLICA TU CÓDIGO. SÚBELO TODO A LA TAREA DE AULES DISPONIBLE.**

**ADEMÁS, PEGA LA URL DE TU PROYECTO EN GITHUB.**

## Programación

### EXAMEN TEMA 3 – ESTRUCTURAS DE DATOS

(27/11/2024)

1. (0,75p) ¿Qué devuelve la siguiente sentencia?

```
String[] nombres = new String[2];
nombres[2] = "Pepe";
for(int n : nombres) {
    System.out.println(n + " ");
}
```

2. (0,75p) ¿Qué sacará por pantalla el siguiente código Java?

```
int[] nums = {1, 2, 3, 4, 5};
int[] copiaArray;
copiaArray = nums;
copiaArray[3] = 0;
System.out.println(Arrays.toString(nums));
System.out.println(Arrays.toString(copiaArray));
```

3. (0,75p) ¿Qué sacará por pantalla el siguiente código Java? Identifica sobre el método `.arraycopy()` lo que significa cada uno de los elementos que contiene.

```
int[] primos = {2, 3, 5, 7, 11, 13, 17};
int[] copia = new int[primos.length];
System.arraycopy(primos, 2, copia, 4, 3);
System.out.println(Arrays.toString(primos));
System.out.println(Arrays.toString(copia));
```

4. (1,5p) ¿Qué 3 opciones tenemos si queremos buscar un elemento dentro de un vector o *array*? Explica el tipo de dato que devuelve cada una.

5. (0,5p) ¿Qué hace Java cada vez que usamos la palabra **return**?

6. (1,5p) Explica paso a paso qué haría el algoritmo SELECCIÓN DESCENDENTE para ordenar el siguiente vector:

[34,23,10,45,4,6,0]

- ¿Es un tipo de algoritmo *in-place* o *no-in-place*? ¿Por qué?
- ¿Por qué este tipo de algoritmo es muy lento si lo comparamos con otros más eficientes, como por ejemplo el *QuickSort*?
- ¿Qué método podemos utilizar en *Java* para realizar una ordenación rápidamente?

7. (1,5p) Explica paso a paso qué haría el algoritmo de búsqueda binaria para encontrar los valores **8** y **24** dentro del vector dado.

[3,8,14,19,24,35,48]

- ¿Por qué es tan eficiente?
- ¿Qué devuelve el método *.binarySearch()* si decidimos aplicarla en *Java* para buscar el número **8**?

8. (0,5p) ¿Qué sacará por pantalla el siguiente código *Java*?

```
int vector[] = {3,3,7,8,8,9,10,15,15};
int vector2[] = Arrays.stream(vector).distinct().toArray();
System.out.println(Arrays.toString(vector));
System.out.println(Arrays.toString(vector2));
```

9. (1,5p) Realiza una traza de los siguientes programas:

a)

```
for (int i = 1; i <= 5; i++) {
    for (int j = 1; j <= i; j++) {
        System.out.print(j + " ");
    }
    System.out.println();
}
```

b)

```
bucle1:
for (int i = 5; i >= 1; i--) {
    bucle2:
    for (int j = 1; j <= i; j++) {
        if (i == 3 && j == 3) {
            break bucle1;
        }
        System.out.print(j + " ");
    }
    System.out.println();
}
```

10. (0,75p) Dibuja la matriz resultante de ejecutar el siguiente código:

```
String vector[] = {"Pepe", "Juan", "Patri", "Carlos", "Bruna"};
String matriz[][] = new String[3][5];
for (int i = 0; i < matriz[0].length; i++) {
    matriz[2][i] = vector[i];
}
```

# Programación

## EXAMEN PRÁCTICO TEMA 3 – ESTRUCTURAS DE DATOS

(28/11/2024)



**LEE ATENTAMENTE LAS SIGUIENTES INSTRUCCIONES ANTES DE EMPEZAR:**



- **Recopila en un documento de texto las evidencias de todo el examen.** Guárdalo de vez en cuando para no perder el avance de tu trabajo.
- Cuando termines, **pásalo a PDF y sube el documento creado a la entrega de AULES.**

### PARTE 1: Configuración del entorno (1p)

1. Crea un nuevo repositorio llamado “*EXAMEN\_UD3\_[nombre]*” desde *SourceTree*. El repositorio debe crearse en local y tener su espejo en remoto, por lo tanto, sincronízalo con *GitHub*.

**Pega a continuación la URL a tu nuevo repositorio de GitHub:**

2. Crea un nuevo proyecto Java (*Maven*) con *IntelliJ* -o el IDE que utilices- dentro del repositorio que acabas de crear. Llámalo “*EXAMEN UD3*”.
3. Crea en el proyecto una nueva clase Java llamada “*Examen*”.
4. Crea dentro de la clase “*Examen*” un método nuevo para el ejercicio propuesto. Lo puedes llamar como quieras.
5. Añade al *Main* principal la llamada al método creado (*Examen.ejercicio1()*) para poder ejecutar los ejercicios que vas a programar a continuación.

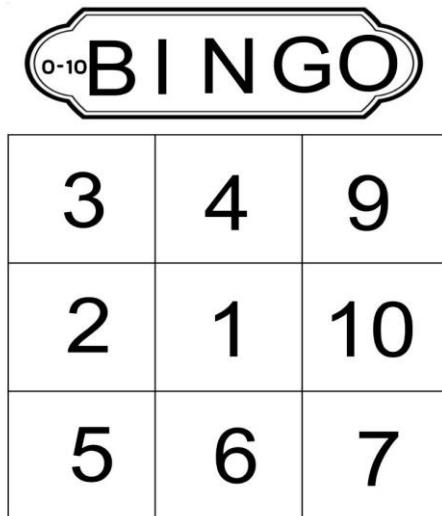
**Sincroniza los cambios en tu repositorio remoto.**

### PARTE 2: Resolución de problemas

Programa en *Java* la solución a los siguientes ejercicios. Usa el proyecto que te acabas de crear en el apartado anterior. **Si no has conseguido crearlo correctamente, utiliza alguno de los proyectos que ya tenías para los ejercicios de clase y pega la URL de GitHub del repositorio al que vas a subir los cambios.**

## 1. (9p) BINGO.

Hoy estamos en el *Casino Cantábrico*, donde una señora acaba de cantar BINGO... Pero los responsables de la sala no se fían mucho de ella, porque ya está mayor y a veces canta líneas y bingos incorrectos. Para ayudar al Casino, nos piden implementar un programa que sea capaz de validar si un cartón está premiado o no.



El BINGO es un juego de azar que consiste en un bombo con un número determinado de bolas numeradas en su interior. Los jugadores juegan con cartones con números aleatorios escritos en ellos, dentro del rango correspondiente (1-90). Un locutor va sacando bolas del bombo, anunciando los números en voz alta. Si un jugador tiene dicho número en su cartón lo tacha, y el juego continúa así hasta que alguien consigue marcar todos los números de su cartón. Para simplificar el tema del cartón en nuestro programa, supondremos que son siempre cuadrados de tamaño 3x3.

### Entrada

El programa pedirá los números del cartón a consultar al usuario (por filas) y los leerá por teclado.

### Salida

El programa mostrará al usuario si los números introducidos son premiados con LINEA o BINGO.

## Salida de ejemplo

\*\*\* BIENVENIDO AL BINGO DEL CASINO CANTÁBRICO \*\*\*

25 bolas extraídas hasta ahora: [7, 8, 9, 81, 10, 30, 2, 27, 1, 90, 41, 31, 49, 72, 50, 4, 69, 14, 42, 1, 85, 17, 48, 65, 78]

\*\*\* Introduce los datos de tu cartón \*\*\*

Fila 1:

45-33-12

Fila 2:

67-89-90

Fila 3:

7-8-9

Datos del cartón introducido:

45 33 12

67 89 90

7 8 9

PREMIOS:

No hay BINGO.

Línea 1: NO

Línea 2: NO

Línea 3: CORRECTA!!

Realiza un programa en Java que cumpla con los requisitos planteados y comenta sobre el código que has escrito alguna explicación breve de lo que hace. Sigue los siguientes pasos:

### a) (1,5p) Bolas del sorteo.

Lo primero que debe hacer el programa es calcular un número aleatorio de “bolas” entre 10 y 40 para simular el número de bolas que han salido en el sorteo hasta el momento.

Con el número de bolas calculado, deberemos crear un vector de tamaño “bolas” con números aleatorios del 1-90, que son los números oficiales del sorteo del Bingo.

Una vez hecho, mostraremos el resultado al usuario:

```
*** BIENVENIDO AL BINGO DEL CASINO CANTÁBRICO ***  
29 bolas extraídas hasta ahora: [7, 8, 9, 58, 41, 8, 65, 44, 8, 72, 76, 5, 2, 4, 54, 58, 28, 82, 78, 89, 83, 90, 18, 28, 77, 19, 78, 46, 81]
```

**NOTA (0,5p):** si no has conseguido hacerlo, introduce los datos como se pasen en el vector para poder seguir con los siguientes puntos.

**b) (1p) Buscar repetidos y recalcular.**

Comprueba que en la lista de números que se genera en el apartado anterior no haya valores repetidos. En caso de haberlos, recalculalos. (Tú eliges si quieres comprobarlo después de hacer el sorteo o mientras se está sorteando).

**c) (2p) Introducir datos del cartón**

Lo siguiente que debe hacer el programa es pedir al usuario los datos del cartón que quiere validar y almacenarlos en una matriz de tamaño 3x3.

Estos datos se introducirán por filas, validando el formato “**N-N-N**” donde el tipo de dato del valor **N** es forzosamente numérico y de 1 ó 2 dígitos.

```
*** BIENVENIDO AL BINGO DEL CASINO CANTÁBRICO ***  
20 bolas extraídas hasta ahora: [7, 8, 9, 74, 15, 79, 4, 51, 21, 23, 21, 22, 39, 40, 57, 77, 77, 8, 36, 25]  
*** Introduce los datos de tu cartón ***  
Fila 1:  
74-23-77  
Fila 2:  
7-8-9  
Fila 3:  
36-25-40
```

En caso de introducirse un valor con formato no válido, reportaremos un **ERROR** y finalizaremos el programa:

```
*** BIENVENIDO AL BINGO DEL CASINO CANTÁBRICO ***  
15 bolas extraídas hasta ahora: [8, 74, 18, 34, 72, 30, 42, 55, 12, 32, 67, 19, 89, 45, 61]  
*** Introduce los datos de tu cartón ***  
Fila 1:  
dd  
Cerrando programa... Introduce valores con el formato correcto (N-N-N).
```

**NOTA (0,5p):** si no has conseguido hacerlo, introduce los datos como sepas en la matriz 3x3 para poder seguir con los siguientes puntos.

**d) (0,5p) Imprimir datos del cartón introducido**

Imprime la matriz generada en el apartado anterior con los datos del cartón mediante el método que quieras.

```
*** BIENVENIDO AL BINGO DEL CASINO CANTÁBRICO ***

20 bolas extraídas hasta ahora: [7, 8, 9, 74, 15, 79, 4, 51, 21, 23, 21, 22, 39, 40, 57, 77, 77, 8, 36, 25]

*** Introduce los datos de tu cartón ***
Fila 1:
74-23-77
Fila 2:
7-8-9
Fila 3:
36-25-40
Datos del cartón introducido:
74 23 77
7 8 9
36 25 40
```

**e) (1,5p) Comprobar el premio BINGO.**

Deberemos comprobar que todos los números del cartón introducido estén entre las bolas sorteadas hasta el momento.

```
*** BIENVENIDO AL BINGO DEL CASINO CANTÁBRICO ***

18 bolas extraídas hasta ahora: [7, 8, 9, 53, 14, 19, 49, 74, 55, 36, 88, 15, 82, 40, 19, 38, 87, 79]

*** Introduce los datos de tu cartón ***
Fila 1:
53-14-87
Fila 2:
7-8-9
Fila 3:
79-38-14
Datos del cartón introducido:
53 14 87
7 8 9
79 38 14

PREMIOS:

HAY BINGO!!
```

En caso de no contener alguno de los números, informaremos que no hay BINGO y pasaremos a comprobar si hay LINEA en el siguiente apartado.

**f) (1,5p) Comprobar el premio LINEA.**

Solamente en caso de no haberse producido el BINGO en el apartado anterior, deberemos comprobar si el cartón introducido contiene alguna línea (fila completa) cuyos números aparezcan entre los sorteados hasta el momento.

```
*** BIENVENIDO AL BINGO DEL CASINO CANTÁBRICO ***

35 bolas extraídas hasta ahora: [7, 8, 9, 81, 9, 79, 59, 5, 49, 89, 39, 82, 16, 46, 22, 60, 2, 41, 25, 84, 18, 55, 27, 34, 10, 62, 30, 79, 5, 38, 30, 60, 20, 53, 30]

*** Introduce los datos de tu cartón ***
Fila 1:
44 56 32
Fila 2:
89 35 6
Fila 3:
7 8 9
Datos del cartón introducido:
44 56 32
89 35 6
7 8 9

PREMIOS:

No hay BINGO.
Línea 1: NO
Línea 2: NO
Línea 3: CORRECTA!!
```

**g) (1p) Pega a continuación las capturas de pantalla de tus pruebas:**



