

1º CFGS DESARROLLO DE APLICACIONES MULTIPLATAFORMA

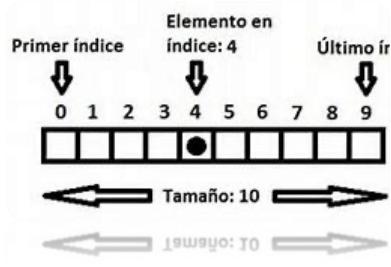
UD 7. Estructuras de almacenamiento.

Módulo: Programación



Centro de Enseñanza
Gregorio Fernández

Arrays



- Un **array** (o **vector**) es una estructura que sirve para agrupar y organizar datos.
- Adecuados para manejar una gran cantidad de información, ya que en estos casos no es viable utilizar variables independientes para cada dato.
- Los valores que almacena un array son **del mismo tipo**.
- Cada uno de los valores es almacenado en una posición identificada mediante un número llamado **índice**.
- Analicemos la siguiente figura de un array de enteros:



Arrays

I - Creación

- Los arrays son objetos, por lo que para crear un array hay que seguir los pasos:

1. Declaración

tipo[] nombreArray;

2. Instanciación

nombreArray = new tipo[tamaño];

- Lo habitual es hacer los dos pasos a la vez en una sola línea.

```
int[] alturas = new int[10];
String[] nombres = new String[tam];
boolean[] resultados = new boolean[30];
```



Centro de Enseñanza
Gregorio Fernández

Arrays

I - Creación

- Un array puede ser de **cualquier tipo** de dato Java, primitivo u objeto:

```
Persona[] personas = new Persona[100];
```

- Todos los valores almacenados en un array deben ser del **mismo tipo**.
- Una vez que un array es instanciado, su **tamaño es fijo**.
- Puede consultarse su tamaño mediante la constante **length**:

```
int L = lista.length;
```



Centro de Enseñanza
Gregorio Fernández

Arrays

II – Acceso a los elementos

- Para **acceder a un elemento de un array** se pone el nombre del array y entre corchetes el índice del elemento :

`alturas[4]` → 173

- La expresión anterior se refiere a un único elemento almacenado en una posición de memoria y por tanto puede usarse como cualquier variable de tipo entero.
- En particular su valor puede ser volcado a otras variables, usado en expresiones y modificada.
- El índice puede ser tanto una constante entera como una variable o expresión entera.
- Ejemplos:

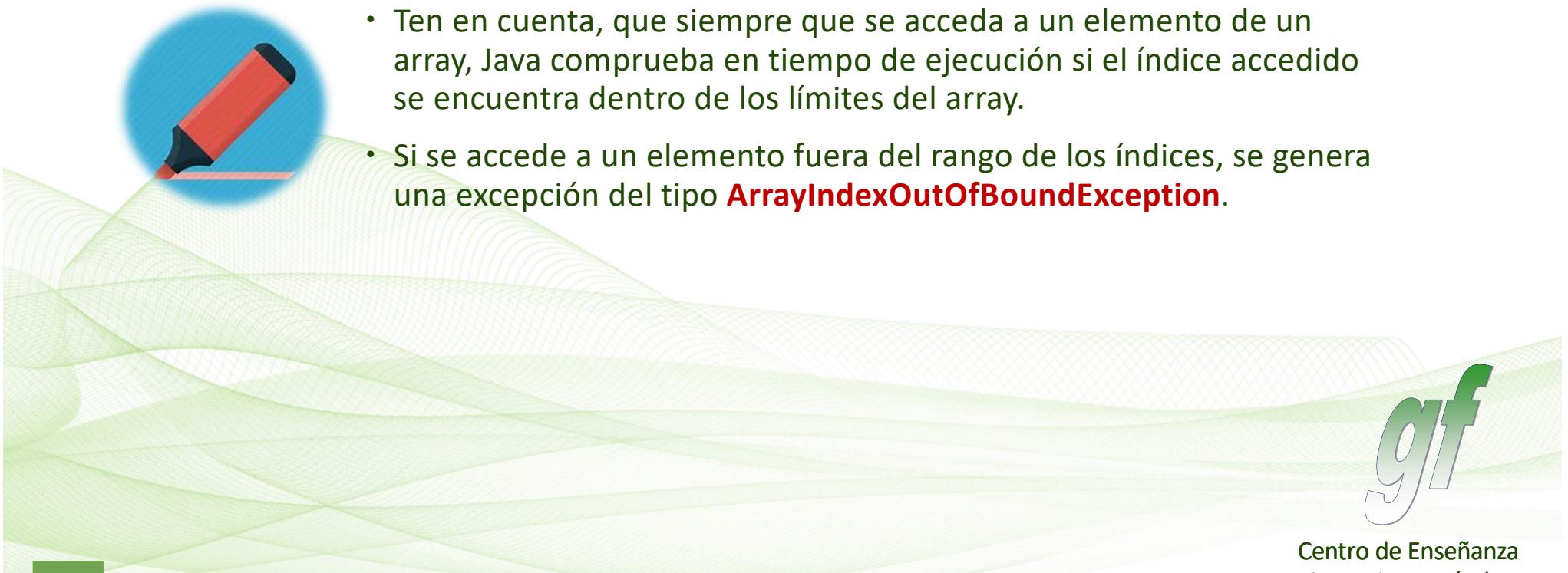
```
alturas[4] = 72;  
alturas[numero] = 12 * valor;  
media = (alturas[0] + alturas[1] + alturas[2])/3;  
System.out.println("Altura = " + alturas[MAX/2]);
```



Arrays

II – Acceso a los elementos

- Ten en cuenta, que siempre que se acceda a un elemento de un array, Java comprueba en tiempo de ejecución si el índice accedido se encuentra dentro de los límites del array.
- Si se accede a un elemento fuera del rango de los índices, se genera una excepción del tipo **ArrayIndexOutOfBoundsException**.



Arrays

III – Recorrido

- Para recorrer una array desde el principio hasta el final, utilizaremos un bucle.
- El más adecuado es el bucle **for**.
- En cada iteración del bucle, se accede a la posición i-ésima del array.
- Por ejemplo:

```
L=vector.length;  
  
for (int i=L-1; i >= 0; i--) {  
  
    System.out.print(vector[i] + " ");  
}
```



Arrays

IV - Inicialización

- Al instanciar un array Java inicializa automáticamente todos sus elementos a **null**, **0** o **false**, según que los datos que contenga sean de tipo objeto, numérico o booleano respectivamente.
- Después de instanciar el array se puede inicializar como se deseé. Por ejemplo, con los números del 0 al 9.
- También se puede **inicializar el array en la propia declaración**, indicando la lista de valores entre llaves y separados por comas. Por ejemplo:

```
int[] lista = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
```

- Cuando se crea un array de esta manera, ya no es necesario el uso del operador **new**.



Centro de Enseñanza
Gregorio Fernández

Arrays

V – Como argumentos

- Ya que los arrays son objetos, cuando un array es pasado como argumento a un método **es pasado por referencia**, es decir, lo que se pasa es una copia de la dirección de memoria donde empieza el array y no una copia completa de todos los valores del array.
- Esto implica que los cambios que se hagan en el array en el interior del método **permanecerán** tras su finalización.



Centro de Enseñanza
Gregorio Fernández

Arrays

VI – De objetos

- Los arrays también **pueden almacenar objetos**, para ser más exactos, referencias a objetos.
- Por ejemplo, si quisiéramos hacer un programa para almacenar los alumnos de una clase, podríamos crear un array de objetos Alumno:

```
public class Alumno {  
    private String nombre;  
    private double nota;  
    private String calificación;  
    ...  
}
```

- Si la clase es de 30 alumnos, creamos el array de tamaño 30:

```
Alumno[] clase = new Alumno[30];
```



Centro de Enseñanza
Gregorio Fernández

Arrays

VI – De objetos

- A continuación, se crearán los objetos Alumno y se introducen en el array:

```
clase[0] = new Alumno("Luis Pérez",5,"Aprobado");  
clase[1] = new Alumno("Marta Cuesta",8,"Notable");  
...  
...
```

- Si queremos visualizar todos los alumnos de la clase, lo haríamos de la siguiente forma:

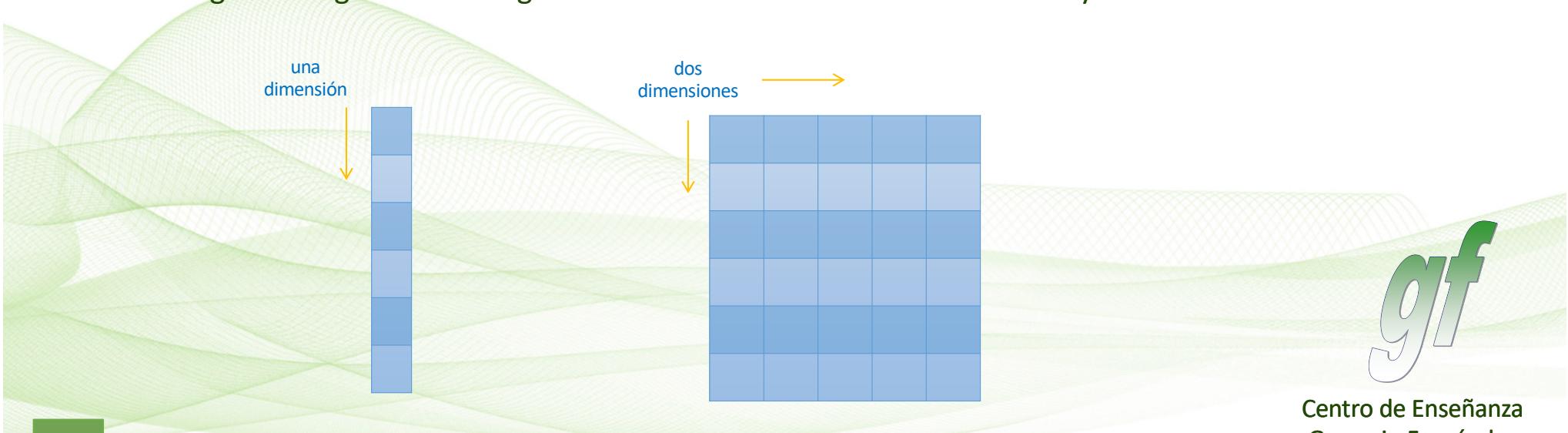
```
for (int i=0; i<clase.length; i++) {  
    System.out.println(clase[i].nombre);  
    System.out.println(clase[i].nota);  
    System.out.println(clase[i]);  
}
```



Centro de Enseñanza
Gregorio Fernández

Arrays bidimensionales

- Los arrays bidimensionales, almacenan sus valores en dos dimensiones que pueden verse como las filas y columnas de una **tabla**.
- También se les denomina **matrices**.
- La siguiente figura muestra gráficamente la diferencia entre un **vector** y una **matriz**:



Arrays bidimensionales

I – Miscelánea

- Para **representar cada dimensión** se usa un par de corchetes.
- Para **declarar e instanciar** un array bidimensional, hay que hacer referencia a las filas y columnas:

```
int[][] array2D = new int[3][2]; //3 filas 2 columnas
```

- Es posible inicializar el array directamente en su declaración:

```
int[][] array2D = {{0,1},{2,3},{4,5}};
```

- El **número de filas** puede conocerse mediante la propiedad **length** aplicada al array:

```
int NF = array2D.length;
```

- El **número de columnas** puede conocerse utilizando la propiedad **length** aplicada a cualquiera de las filas. En particular sobre la primera fila:

```
int NC = array2D[0].length;
```



Arrays bidimensionales

I – Miscelánea

- El **acceso a los elementos** se hará indicando los dos índices, el de la fila y el de la columna:

```
System.out.println(array2D[2][1]);
```

- Para **imprimir** los elementos de un array bidimensional hay que utilizar dos bucles anidados, el externo para recorrer las filas y el interno para recorrer las columnas dentro de cada fila.

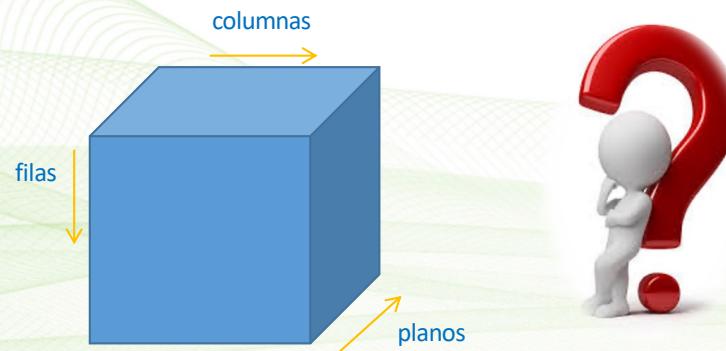
```
for (int i=0; i<NF; i++) {  
    for (int j=0; j<NC; j++) {  
        System.out.print (array2D[i][j]);  
    }  
    System.out.println();  
}
```



Centro de Enseñanza
Gregorio Fernández

Arrays multidimensionales

- Un array puede tener una, dos, tres o incluso más dimensiones.
- Cualquier array con más de una dimensión es llamado en general **array multidimensional**.
- Es fácil imaginar un array de dos dimensiones como una tabla, un array de tres dimensiones podría ser dibujado como un cubo, pero una vez pasadas las tres dimensiones es difícil imaginar los arrays multidimensionales.

The logo consists of the lowercase letters 'gf' in a bold, italicized, green font.

Centro de Enseñanza
Gregorio Fernández

Buscar en un array

- Los dos algoritmos más típicos para buscar en arrays son el de **búsqueda secuencial** para arrays desordenados y el de **búsqueda binaria** para arrays ordenados.

Búsqueda secuencial

- Consiste en buscar el valor empezando por el principio del array, comparando elemento a elemento hasta que coincida con el valor buscado, entonces finaliza la búsqueda.
- Válido tanto para arrays ordenados y desordenados.**
- Veamos su implementación en el siguiente método:

```
public int busquedaSecuencial(int[] lista, int dato){  
    for (int i = 0; i < lista.length; i++) {  
        if (lista[i] == dato) return i;  
    }  
    return -1;  
}
```



Buscar en un array

- También podemos implementar el algoritmo de **búsqueda secuencial** mediante un bucle **while** de la siguiente forma:

```
public int busquedaSecuencial(int[] lista, int dato) {  
    int i=0;  
    boolean encontrado=false;  
    while ((i<lista.length) && (encontrado==false)) {  
        if (lista[i] == dato) encontrado=true;  
        i++;  
    } //while  
    if (encontrado) return (i-1);  
    else return -1;  
}
```



Buscar en un array

- En el siguiente código se usa el método anterior para buscar un número entero generado aleatoriamente entre 0 y 19, en un array que contiene los 10 dígitos decimales desordenados:

```
int[] LISTA = {0,5,7,8,1,3,2,4,6,9};  
int DATO = (int) (20 * Math.random());  
int indice = busquedaSecuencial(LISTA, DATO);  
if (indice >= 0) {  
    System.out.println("Dato encontrado en " + indice);  
} else {  
    System.out.println("Dato no encontrado");  
}
```



Buscar en un array

- **Búsqueda binaria**

- ✓ Para realizar búsquedas sobre **arrays ordenados**, Java posee el método estático **binarySearch(int[] lista, int dato)** de la clase **java.util.Arrays**, el cual implementa de forma eficiente el algoritmo de búsqueda binaria.
- ✓ Requisito: que el **array esté ordenado**.
- ✓ Este método devuelve el índice en el que se encuentra el elemento a buscar ó -1 en caso de no encontrarlo.
- ✓ El siguiente código hace uso de dicho método para realizar la misma búsqueda que en el ejemplo anterior:

```
int[] LISTA = {0,1,2,3,4,5,6,7,8,9}; //array ordenado
int DATO = (int) (20 * Math.random());
int indice = Arrays.binarySearch(LISTA,DATO);
if (indice >= 0) {
    System.out.println("Dato encontrado en " + indice);
} else {
    System.out.println("Dato no encontrado");
}
```



Ordenar un array

- La ordenación de un array consiste en disponer los elementos del array de una forma bien definida.
- Se han desarrollado y criticado muchos algoritmos de ordenación a lo largo de los años.
- De hecho la ordenación es un área clásica en el estudio de la Informática.
- Existen numerosos algoritmos de ordenación de arrays: *inserción, selección, burbuja, quicksort, mergesort*, etc.
- Uno de los métodos de ordenación más sencillos es el **algoritmo de la burbuja (bubble sort)**.



Ordenar un array

I - Burbuja

- **Objetivo:** llevar el **máximo al final** del array progresivamente. Para ello, comparar cada elemento con el siguiente e intercambiar si hace falta.
- **Implementación:** son necesarios dos bucles anidados:

✓ **Un bucle externo:** donde se irán comparando todos los elementos de un *subarray* (que cada vez es más pequeño).

En cada pasada se establece el mayor de dicho subarray
(o menor si la ordenación es descendente).

Si el número de elementos del array es N se harán **$N-1$ pasadas**.

✓ **Un bucle interno:** para comparar parejas dentro de un subarray. En la pasada i -ésima se compararán **$N-i$ parejas**. Es decir, en cada pasada se va reduciendo una pareja que comparar.

- Ordenación ascendente: si el de la izquierda es mayor que el de la derecha se intercambian.
- Ordenación descendente: si el de la izquierda es menor que el de la derecha se intercambian.



Ordenar un array

I - Burbuja

- Burbuja ascendente en un array de enteros:

```
public void burbuja(int[] lista){  
    int N = lista.length;  
    for (int i=1; i<=N-1; i++) { //pasadas  
        for (int j=1; j<=N-i; j++) { //parejas  
            if (lista[j-1] > lista[j]) {  
                int tmp = lista [j-1];  
                lista [j-1] = lista [j];  
                lista [j] = tmp;  
            }  
        }  
    }  
}
```

Macarena Cuenca Carbajo



Clase java.util.Arrays

- Clase estática con métodos muy interesantes para utilizar sobre arrays.

Método fill

- Permite llenar un array unidimensional con un determinado valor.

```
int valores[] = new int[20];
Arrays.fill(valores, -1); //todo el array vale -1
```

- También permite indicar desde qué índice hasta qué índice rellenamos el array:

```
Arrays.fill(valores, 5, 8, -1); //elementos del 5 al 7
```

Método equals

- Compara dos arrays y devuelve true si son iguales. Se consideran iguales si son del mismo tipo, tamaño y contienen los mismos valores.



Centro de Enseñanza
Gregorio Fernández

Clase java.util.Arrays

Método sort

- Permite ordenar un array de forma ascendente. Se puede indicar también que sólo se quiere ordenar una parte del array, indicando los índices.

```
int x[]={4,5,2,3,7,8,2,3,9,5};  
Arrays.sort(x); //ordena el array completo  
Arrays.sort(x,2,5); //ordena del 2º al 4º elemento
```

Método binarySearch

- Permite buscar un elemento de forma ultrarrápida en un **array ordenado** (en un array desordenado sus resultados son impredecibles). Devuelve el índice en el que está colocado el elemento a buscar ó -1 si no lo encuentra. Ejemplo:

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



Centro de Enseñanza
Gregorio Fernández