UF6.1
Arrays en Java

JAVA™

Universidad Europea de Madrid

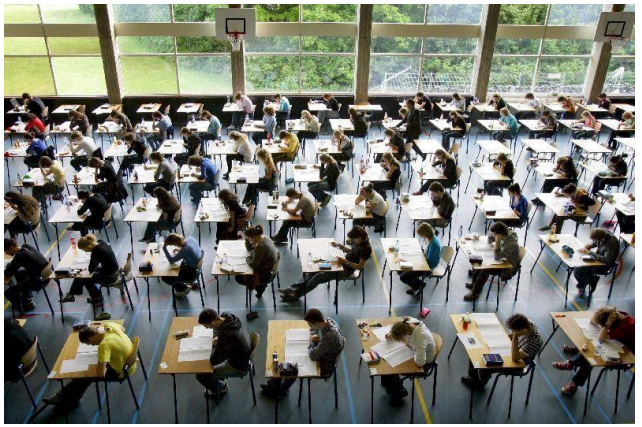
LAUREATE INTERNATIONAL UNIVERSITIES

CONTENIDOS

1. Introducción
2. Declaración
3. Características
4. Inicialización
5. Arrays bidimensionales
6. Copying and cloning arrays
7. Ordenar Vectores

Almacenar información

Hasta ahora, hemos utilizado variables (atributos, parámetros o variables locales) para almacenar los datos que nos hacían falta. Imagina que en un programa nos piden almacenar la nota de 100 alumnos que van a hacer un examen y además nos piden que obtengamos la nota media. No parece muy operativo tener 100 variables en nuestro programa y si además el número de alumnos aumentase hasta 200 ó 1000…¿qué harías?



La solución a esto se llama arrays o vectores (son sinónimos).

Array

Un array se compone de una serie de posiciones consecutivas en memoria.
A los vectores se accede mediante un índice: mi_vector[índice]

array

índice	0	1	2	3	4	5	6	7	8	9
contenido	13	21	34	92	0	6	76	4	0	84

matriz

índice \ índice	0	1	2	3	4	5	6	7	8
0	76	4	0	84	21	34	92	0	6
1	1	2	3	4	5	6	7	8	9
2	21	34	92	0	6	76	4	0	84

contenido

Los arrays puden ser de varias dimensiones.

ARRAY

Declaración

En Java se pueden declarar vectores de dos formas diferentes:

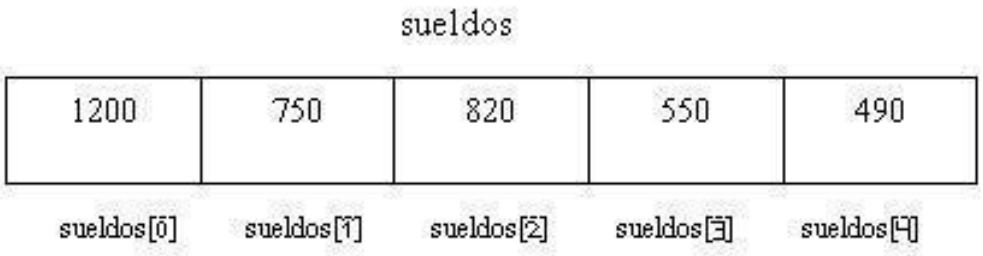
```
<tipo> [] <nombre> ;
<tipo> <nombre> [] ;
```

Por ejemplo, para declarar un array de números enteros utilizaremos la siguiente sentencia:

```
int   [] sueldos;
```

El array aún no ha sido creado, sino meramente declarado. Java trata los vectores como si fueran objetos, por lo tanto para crear el Array (reservar su memoria e inicializarlo) deberemos recurrir al operador new:

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

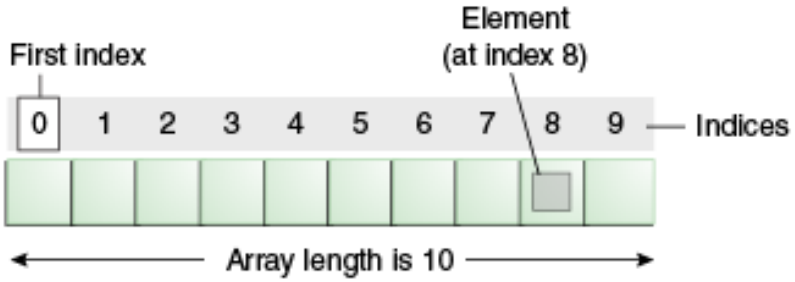


ARRAY

Características 1/2

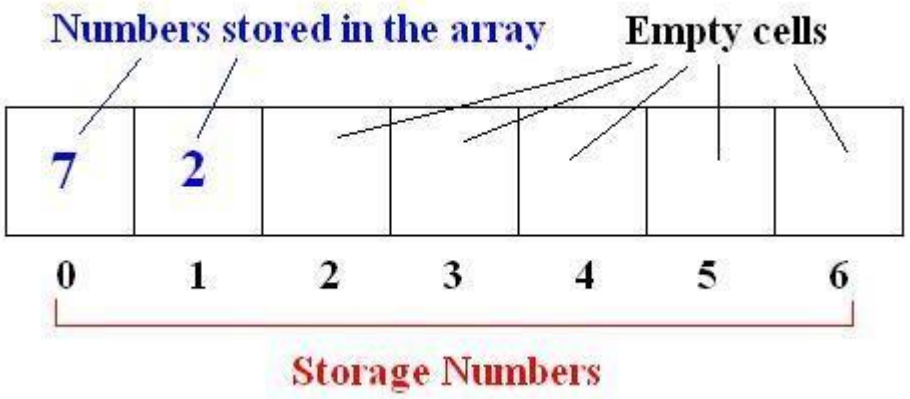
Algunas de sus características más importantes de los arrays son las siguientes:

1. Los arrays se crean con el operador new seguido del tipo y número de elementos.
2. Se puede acceder al número de elementos de un array con la variable miembro implícita length (por ejemplo, vect.length).
3. Se accede a los elementos de un array con los corchetes [] y un índice que varía de 0 a length-1.
4. Se pueden crear arrays de objetos de cualquier tipo. En principio un array de objetos es un array de referencias que hay que completar llamando al operador new.



Características 2/2

5. Los elementos de un array se inicializan al valor por defecto del tipo correspondiente (cero para valores numéricos, la cadena vacía para Strings, false para boolean, null para referencias).
6. Como todos los objetos, los arrays se pasan como argumentos a los métodos por referencia.
7. Como todo objeto Java, hereda de la clase Object sus métodos. Especial interés tienen:
   - equals: permite saber si dos referencis son el mismo objeto.
   - clone: duplica un objeto



ARRAY

Inicialización

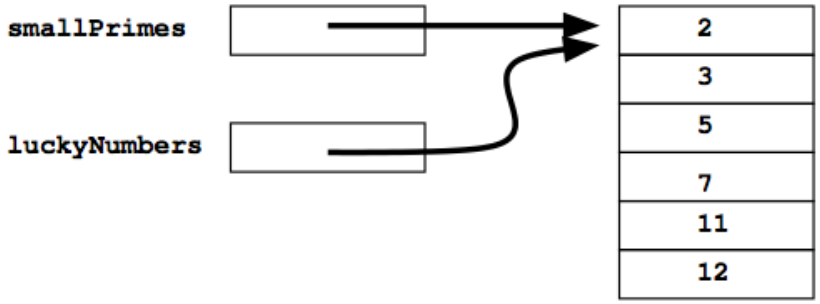
Existen diferentes formas :

1. Los arrays se pueden inicializar con valores entre llaves {...} separados por comas.

```
int c = 15;
int[] primes = { 1, 2, 3, 5, 7, 9, 11 };
int[] a = { 1, primes[2], c, (int) Math.pow(2,2) };
```

2. También los arrays de objetos se pueden inicializar con varias llamadas a new dentro de unas llaves {...}.
3. Si se igualan dos referencias a un array no se copia el array, sino que se tiene un array con dos nombres, apuntando al mismo y único objeto:

```
luckyNumbers = smallPrimes;
```



Acceso – Para practicar

Accessing Array Elements Example

```
int c = 15, j = 1;

int[] primes = { 1, 2, 3, 5, 7, 9, 11 };
int[] a = { 1, primes[2], c, (int)Math.pow(2,2) };

System.out.println( "\n Before: \na[0] is " + a[0] );
System.out.println( "a[1] is " + a[1] \n "a[2] is " + a[2] );
System.out.println( "a[3] is " + a[3] );

a[ 0 ] = 0;

a[ j++ + a[0] ] = 1;          // a[1 + 0] = a[1] = 1, j = 2
a[ j-- ]++;                   // a[2] = a[2] + 1 = 16, j=1
a[ 3 ] %= 5;                  // a[3] = a[3] % 5 = 4 % 5 = 4;

//  a[ 0 ] = a[3] + a[1] - a[0] = 4 + 1 - 0 = 5, j=0
a[ j-1 ] = a[3] + a[j] - a[--j];

System.out.println( "\n After: \n a[0] is " + a[0] );
System.out.println( "a[1] is " + a[1] \n "a[2] is " + a[2] );
System.out.println( "a[3] is " + a[3] );
```

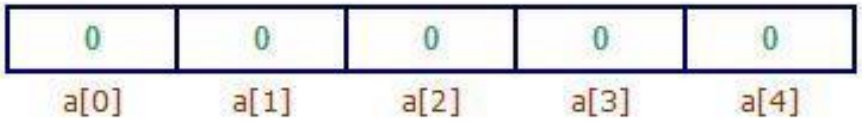
Output

```
Before:
a[0] is 1
a[1] is 3
a[2] is 15
a[3] is 4

After:
a[0] is 5
a[1] is 1
a[2] is 16
a[3] is 4
```

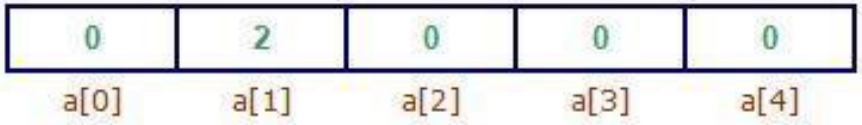
ARRAY

Un ejemplo – Para practicar

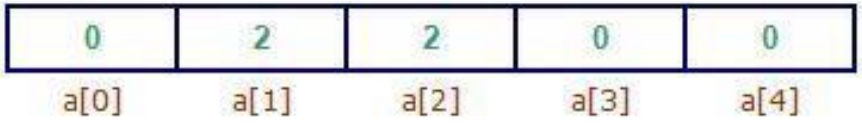
```
int m = 5;
int [] a = new int[5];
```



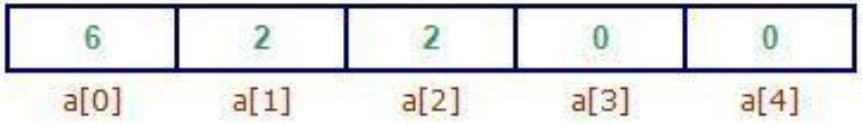

```
a[1] = 2;
```



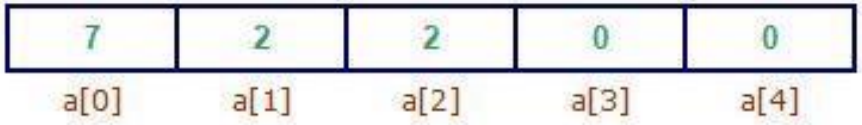

```
a[2] = a[1];
```



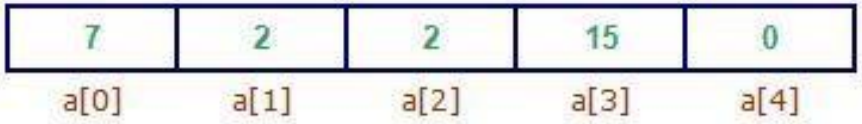

```
a[0] = a[1] + a[2] + 2;
```




```
a[0]++;
```




```
int m = 5;
a[3] = m + 10;
```

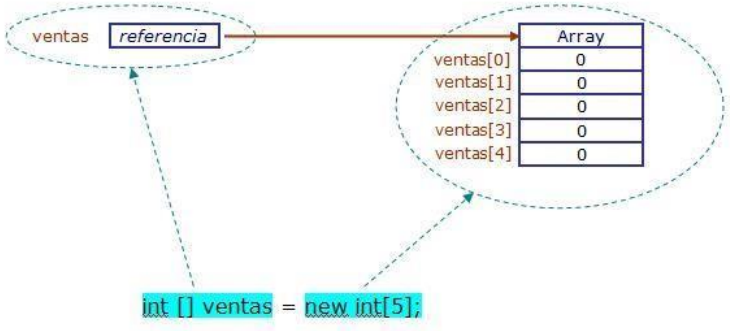


ARRAY

Más ejemplos – Para practicar

A continuación se presentan algunos ejemplos de creación de arrays:

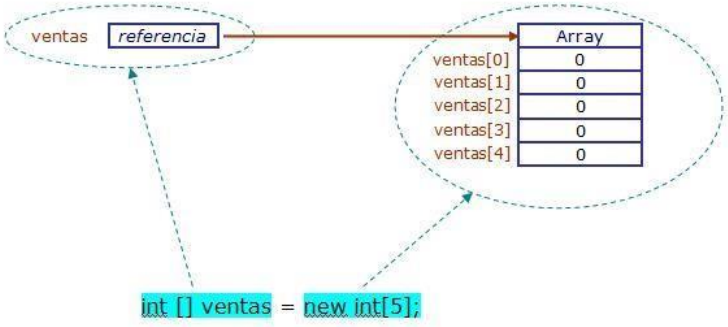
```
// crear un array de 10 enteros, que por defecto se inicializan a cero
int v[] = new int[10];

// crear arrays inicializando con determinados valores
int v[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
String dias[] = {"lunes", "martes", "miercoles", "jueves", "viernes", "sabado",
"domingo"};

// array de 5 objetos
MiClase listaObj[] = new MiClase[5]; // de momento hay 5 referencias a null
for( i = 0 ; i < 5;i++)
        listaObj[i] = new MiClase(...);

// array anónimo
obj.metodo(new String[]={"uno", "dos", "tres"});
```



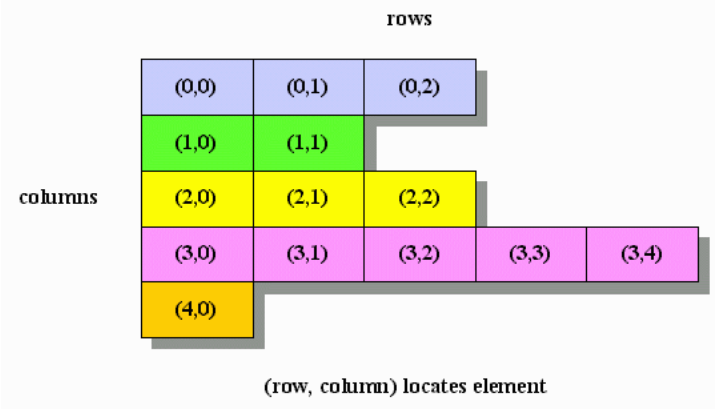
ARRAYS BIDIMENSIONALES

Definición

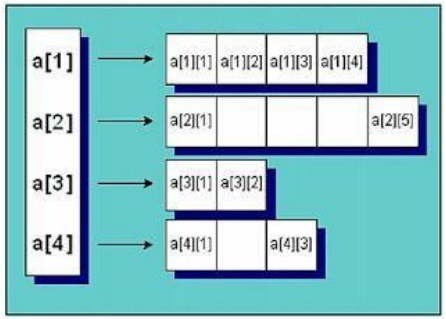
Los arrays bidimensionales de Java se crean de un modo muy similar al de C++ (con reserva dinámica de memoria).



En Java una matriz es un vector de vectores fila, o más en concreto un vector de referencias a los vectores fila. Con este esquema, cada fila podría tener un número de elementos diferente.



ARRAYS BIDIMENSIONALES



Creación

Una matriz se puede crear directamente en la forma:

```
int [][] mat = new int[3][4];
```

o bien se puede crear de modo dinámico dando los siguientes pasos:

1. Crear la referencia indicando con un doble corchete que es una referencia a matriz:

   ```
   int[][] mat;
   ```

2. Crear el vector de referencias a

   las filas: mat = new int[nfilas][];

3. Reservar memoria para los vectores correspondientes a

   las filas: for (int i=0; i<nfilas; i++);
   mat[i] = new int[ncols];

ARRAYS BIDIMENSIONALES

Ejemplos

A continuación se presentan algunos ejemplos de creación de arrays bidimensionales:

```
// crear una matriz 3x3
// se inicializan a cero
double mat[][] = new double[3][3];
int [][] b = {      {1, 2, 3},
                    {4, 5, 6}, // esta coma es permitida
            };
int c = new[3][]; // se crea el array de referencias a arrays
c[0] = new int[5];
c[1] = new int[4];
c[2] = new int[8];
```

En el caso de una matriz b, b.length es el número de filas y b[0].length es el número de columnas (de la fila 0).

Por supuesto, los arrays bidimensionales pueden contener tipos primitivos de cualquier tipo u objetos de cualquier clase.

ARRAYS BIDIMENSIONALES

Ejemplo: Matriz Unidad

```
package matrizUnidad;

public class MatrizUnidadApp {
  public static void main (String[] args) {
   double[][] mUnidad= new double[4][4];

   for (int i=0; i < mUnidad.length; i++) {
      for (int j=0; j < mUnidad[i].length; j++) {
         if (i == j) {
            mUnidad[i][j]=1.0;
         }else {
            mUnidad[i][j] = 0.0;
         }
      }
   }

   for (int i=0; i < mUnidad.length; i++) {
      for (int j=0; j < mUnidad[i].length; j++) {
         System.out.print(mUnidad[i][j]+"\t");
      }
      System.out.println("");
   }

   try{
//espera la pulsación de una tecla y luego RETORNO
      System.in.read();
   }catch (Exception ex) {  }
  }
}
```

COPYING ARRAYS

System method arraycopy()

Arrays can be copied using the Java System method arraycopy():

public static native void **arraycopy** (Object src, int src_position, Object dst, int dst_position, int length)

Copies a region of the source array, src, beginning at the array cell src_position, to the destination array, dst, beginning at the cell dst_position in the destination. The number of cells copied is equal to the length argument.

Parameters:

src - the source array.
src_position - start position (first cell to copy) in the source array.
dst - the destination array.
dst_position - start position in the destination array.
length - the number of array elements to be copied.

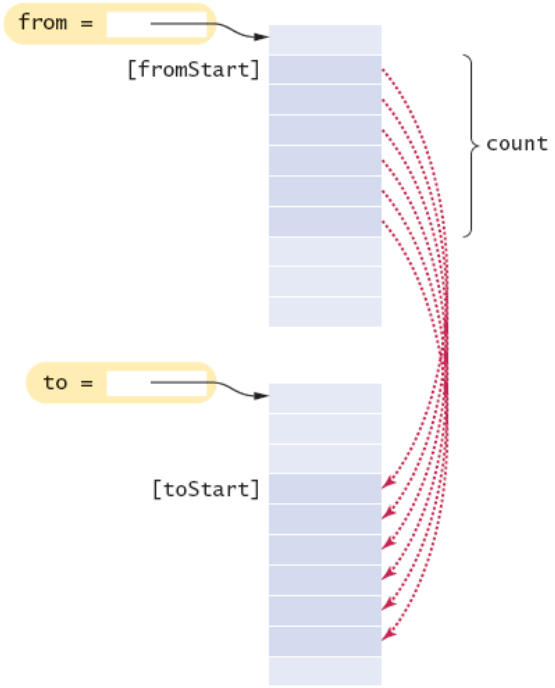
Note: arraycopy does not allocate memory for the destination array; the memory must already be allocated.

COPYING ARRAYS

System method arraycopy()

Example, **arraycopy**()

```
int[] primes = { 1, 2, 3, 5, 7, 9, 11 };
int[] c = new int[ primes.length ];
System.arraycopy( primes, 0, c, 0, primes.length);     // copy array primes to array c
```



The System.arraycopy Method

CLONING ARRAYS

Clone ()

By default, all Java arrays support the clone method

```
public class ArrayClone
{
    public static void main( String[] args )
    {
        int[] primes = { 1, 2, 3, 5, 7, 11, 13, 17 };
        int[] backup;

        backup = (int[]) primes.clone();
        backup[0] = 0;

        System.out.println( "Primes: " );
        for( int i=0 ; i < primes.length ; i++ )
                System.out.print( " " + primes[i] );

        System.out.println( "\nbackup: With first Element Modified" );
        for( int j=0 ; j < backup.length ; j++ )
                System.out.print(" " + backup[j] );
    }
}
```

- Declare reference to an int array.
- Call the clone() method for the Java array.
- Cast Object to array.
- Set the first element of the cloned array to zero.

ORDENAR VECTORES

Estrategia

El ordenamiento de un vector se logra intercambiando las componentes de manera que:

vec[0] <= vec[1] <= vec[2] etc.

El contenido de la componente vec[0] sea menor o igual al contenido de la componente vec[1] y así sucesivamente.

Si se cumple lo dicho anteriormente decimos que el vector está ordenado de menor a mayor.

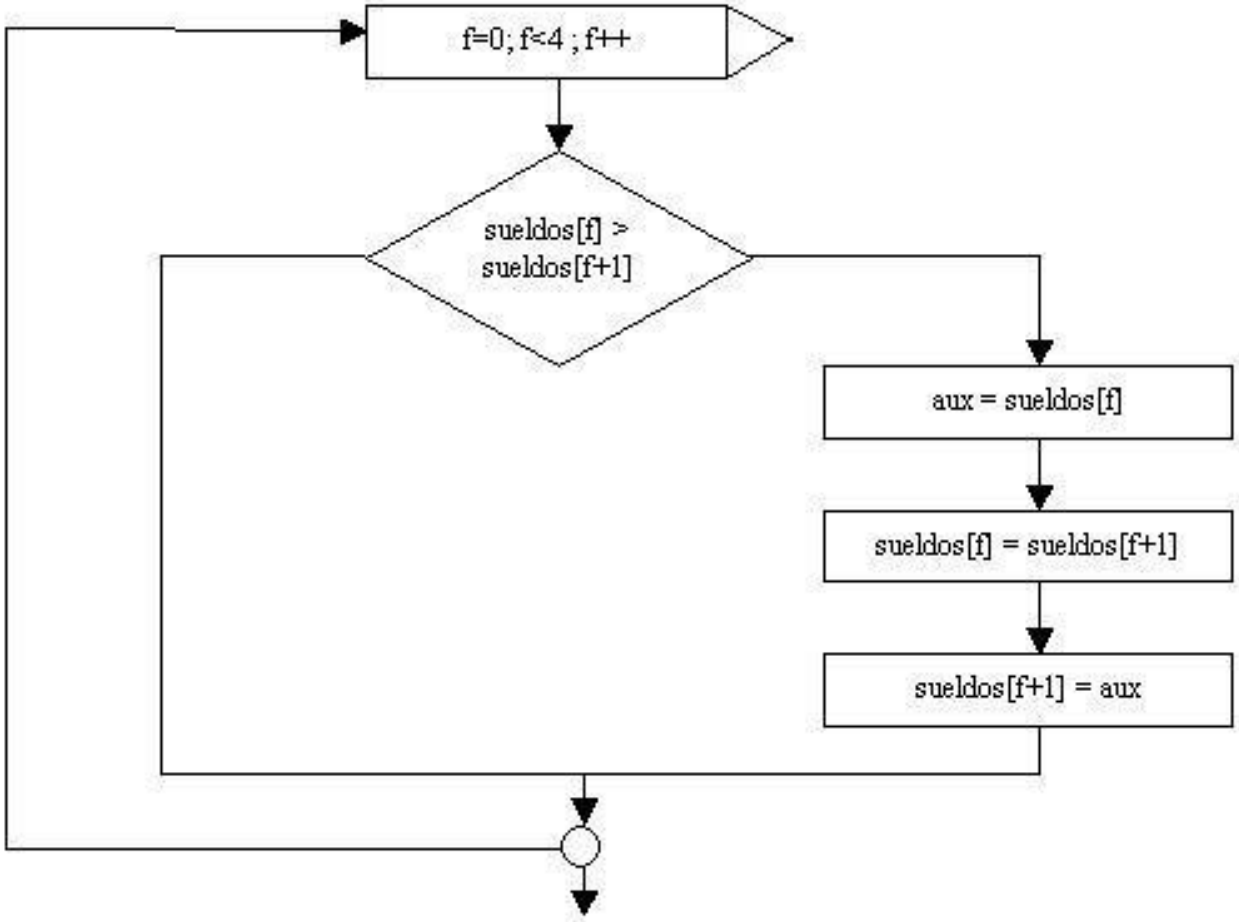
Igualmente podemos ordenar un vector de mayor a menor.

Se puede ordenar tanto vectores con componentes de tipo int, float como String (en este último caso el ordenamiento es alfabético).

ORDENAR VECTORES

Planteamiento del problema

Se debe crear un vector donde almacenar 5 sueldos. Ordenar el vector sueldos de menor a mayor.



ORDENAR VECTORES

Estrategia

Esta primera aproximación tiene por objetivo analizar los intercambios de elementos dentro del vector.

El algoritmo consiste en comparar si la primera componente es mayor a la segunda, en caso que la condición sea verdadera, intercambiamos los contenidos de las componentes.

Vamos a suponer que se ingresan los siguientes valores por teclado:

1200
750
820
550
490

En este ejemplo: ¿es 1200 mayor a 750? La respuesta es verdadera, por lo tanto intercambiamos el contenido de la componente 0 con el de la componente 1.

Luego comparamos el contenido de la componente 1 con el de la componente 2: ¿Es 1200 mayor a 820?

La respuesta es verdadera entonces intercambiamos.

ORDENAR VECTORES

Estrategia

Si hay 5 componentes hay que hacer 4 comparaciones, por eso el for se repite 4 veces.

Generalizando: si el vector tiene N componentes hay que hacer N-1 comparaciones.

Cuando	f = 0	f = 1	f = 2	f = 3
	750	750	750	750
	1200	820	820	820
	820	1200	550	550
	550	550	1200	490
	490	490	490	1200

Podemos ver cómo el valor más grande del vector desciende a la última componente. Empleamos una variable auxiliar (aux) para el proceso de intercambio:

```
aux=sueldos[f];
sueldos[f]=sueldos[f+1];
sueldos[f+1]=aux;
```

ORDENAR VECTORES

Estrategia

Al salir del for en este ejemplo el contenido del vector es el siguiente:

750
820
550
490
1200

Analizando el algoritmo podemos comprobar que el elemento mayor del vector se ubica ahora en el último lugar.

Podemos definir otros vectores con distintos valores y comprobar que siempre el elemento mayor queda al final.

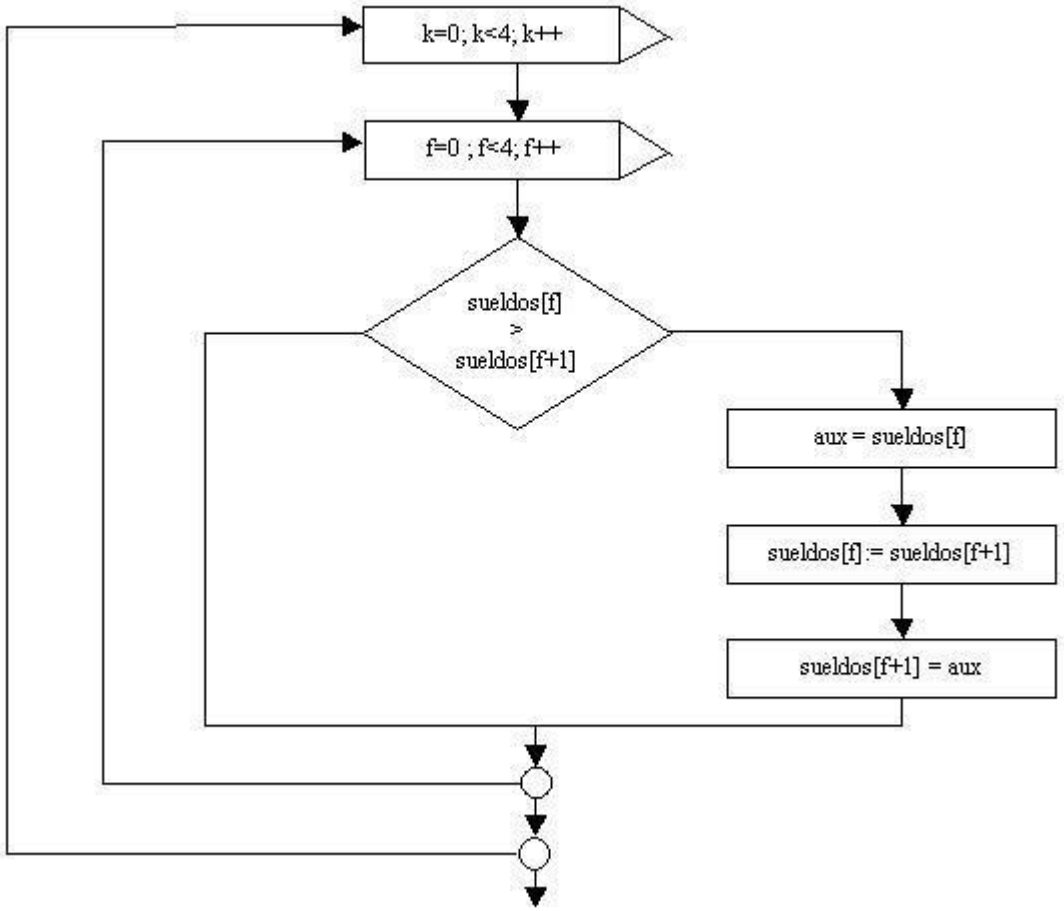
Pero todavía con este algoritmo no se ordena un vector. Solamente está ordenado el último elemento del vector.

Ahora bien, con los 4 elementos que nos quedan podemos hacer el mismo proceso visto anteriormente, con lo cual quedará ordenado otro elemento del vector. Este proceso lo repetiremos hasta que quede ordenado por completo el vector.

ORDENAR VECTORES

Estrategia

Como debemos repetir el mismo algoritmo podemos englobar todo el bloque en otra estructura repetitiva.



ORDENAR VECTORES

Estrategia

Realicemos una prueba del siguiente algoritmo:

Cuando k = 0

f = 0	f = 1	f = 2	f = 3
750	750	750	750
1200	820	820	820
820	1200	550	550
550	550	1200	490
490	490	490	1200

Cuando k = 1

f = 0	f = 1	f = 2	f = 3
750	750	750	750
820	550	550	550
550	820	490	490
490	490	820	820
1200	1200	1200	1200

Cuando k = 2

f = 0	f = 1	f = 2	f = 3
550	550	550	550
750	490	490	490
490	750	750	750
820	820	820	820
1200	1200	1200	1200

ORDENAR VECTORES

Estrategia

Cuando k = 2

f = 0	f = 1	f = 2	f = 3
550	550	550	550
750	490	490	490
490	750	750	750
820	820	820	820
1200	1200	1200	1200

Cuando k = 3

f = 0	f = 1	f = 2	f = 3
490	490	490	490
550	550	550	550
750	750	750	750
820	820	820	820
1200	1200	1200	1200

ORDENAR VECTORES

Estrategia

¿Porque repetimos 4 veces el for externo?

Como sabemos cada vez que se repite en forma completa el for interno queda ordenada una componente del vector. A primera vista diríamos que deberíamos repetir el for externo la cantidad de componentes del vector, en este ejemplo el vector sueldos tiene 5 componentes.

Si observamos, cuando quedan dos elementos por ordenar, al ordenar uno de ellos queda el otro automáticamente ordenado (podemos imaginar que si tenemos un vector con 2 elementos no se requiere el for externo, porque este debería repetirse una única vez)

Una última consideración a este ALGORITMO de ordenamiento es que los elementos que se van ordenando continuamos comparándolos.

ORDENAR VECTORES

Estrategia

Ejemplo: En la primera ejecución del for interno el valor 1200 queda ubicado en la posición 4 del vector. En la segunda ejecución comparamos si el 820 es mayor a 1200, lo cual seguramente será falso.

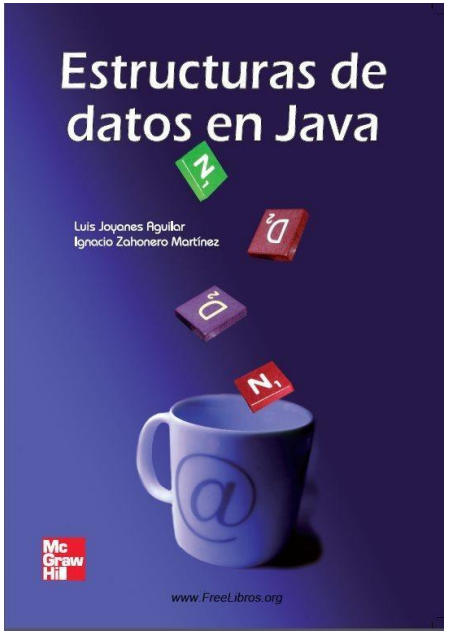
Podemos concluir que la primera vez debemos hacer para este ejemplo 4 comparaciones, en la segunda ejecución del for interno debemos hacer 3 comparaciones y en general debemos ir reduciendo en uno la cantidad de comparaciones.

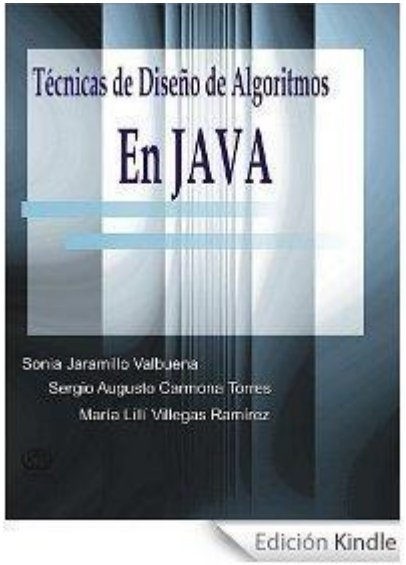
Si bien el algoritmo planteado funciona, un algoritmo más eficiente, que se deriva del anterior es el plantear un for interno con la siguiente estructura: (f=0 ; f<4-k; f++)

Es decir restarle el valor del contador del for externo.

ESTRUCTURAS DE DATOS

Y algoritmos de ordenación





Pulsa aquí->