Arrays unidimensionales en Java

Concepto de Array:

Un **array** es una colección finita de datos del mismo tipo, que se almacenan en posiciones consecutivas de memoria y reciben un nombre común.

Por ejemplo, supongamos que queremos guardar las notas de los 20 alumnos de una clase.

Para almacenar las notas utilizaremos un array de 20 elementos de tipo double y en cada elemento del array guardaremos la nota de cada alumno.

Podemos representar gráficamente el array de notas de la siguiente forma:

Array **notas**:

8.50	6.35	5.75	8.50	· · ·	3.75	6.00	7.40
notas[0]	notas[1]	notas[2]	notas[3]	· · ·	notas[17]	notas[18]	notas[19]

Para acceder a cada elemento del array se utiliza el nombre del array y un índice que indica la posición que ocupa el elemento dentro del array.

El índice se escribe entre corchetes.

El **primer elemento** del array ocupa la **posición 0**, el segundo la posición 1, etc. **En un array de N elementos el último ocupará la posición N-1**.

En el ejemplo anterior, notas[0] contiene la nota del primer alumno y notas[19] contiene la del último.

Los índices deben ser enteros no negativos.

1. CREAR ARRAYS UNIDIMENSIONALES

Para crear un array se deben realizar dos operaciones (se consideran objetos):

- Declaración
- Instanciación

Declarar de un array

En la declaración se crea la **referencia** al array.
La referencia será el nombre con el que manejaremos el array en el programa.
Se debe indicar el nombre del array y el tipo de datos que contendrá.
De forma general un array unidimensional se puede declarar en java de cualquiera de estas dos formas:

```
tipo [] nombreArray;
tipo nombreArray[];
```

tipo: indica el tipo de datos que contendrá. Un array puede contener elementos de tipo básico o referencias a objetos.

nombreArray: es la referencia al array. Es el nombre que se usará en el programa para manejarlo.
Por ejemplo:

```
int [] ventas;  //array de datos de tipo int llamado ventas
double [] temperaturas;  //array de datos de tipo double llamado temperaturas
String [] nombres; //array de datos de tipo String llamado nombres
```

Instanciar un array

Mediante la instanciación se reserva un bloque de memoria para almacenar todos los elementos del array.

La dirección donde comienza el bloque de memoria donde se almacenará el array se asigna al nombre del array. Esto quiere decir que el nombre del array contiene la dirección de memoria donde se encuentra.

De forma general un array se instancia así:

nombreArray = new tipo[tamaño];

- nombreArray: es el nombre creado en la declaración.
- tipo: indica el tipo de datos que contiene.
- tamaño: es el número de elementos del array. Debe ser una expresión entera positiva. El tamaño del array no se puede modificar durante la ejecución del programa.
- new: operador para crear objetos. Mediante new se asigna la memoria necesaria para ubicar el objeto. Java implementa los arrays como objetos.

Por ejemplo:

```
ventas = new int[5]; //se reserva memoria para 5 enteros y
                     //se asigna la dirección de inicio del array a ventas.
```

Lo normal es que la declaración y la instanciación se hagan en una sola instrucción:
tipo [] nombreArray = new tipo[tamaño];

Por ejemplo:

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

El tamaño del array también se puede indicar durante la ejecución del programa, es decir, en tiempo de ejecución se puede pedir por teclado el tamaño del array y crearlo:

```
Scanner sc = new Scanner(System.in);
System.out.print("Número de elementos del array: ");
int numeroElementos = sc.nextInt();
int [] ventas = new int[numeroElementos];
```

Si no hay memoria suficiente para crear el array, new lanza una excepción java.lang.OutOfMemoryError.

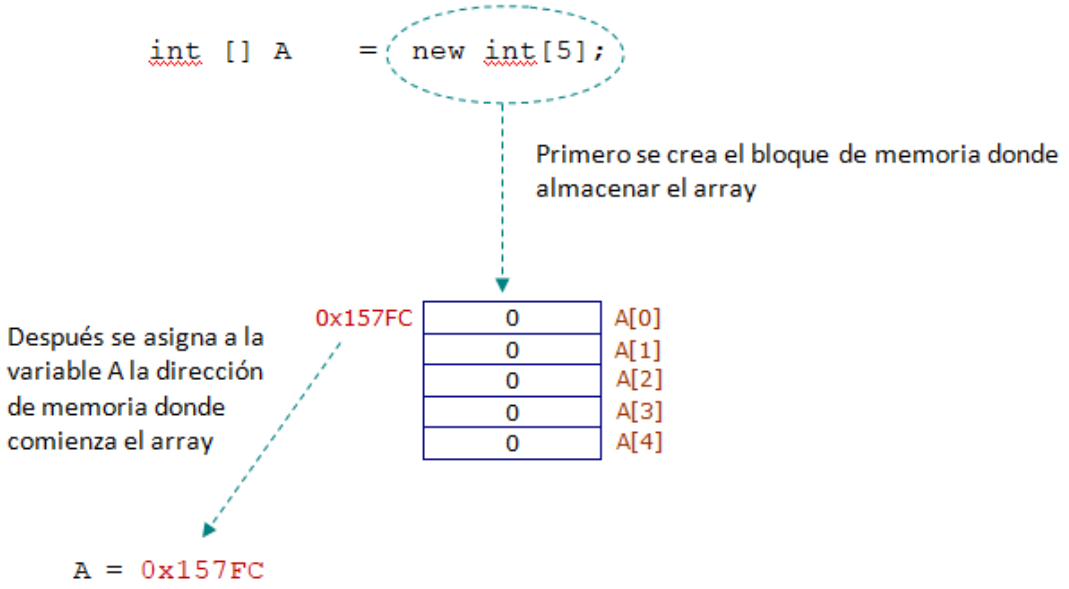
Diferencia entre la referencia y el contenido del array

Debe quedar clara la diferencia entre el nombre del array y el contenido del array.

Cuando por ejemplo se ejecuta una instrucción para crear un array de enteros llamado A:

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

se realizan dos operaciones: primero se crea un bloque de memoria donde almacenar el array de 5 enteros y después se asigna la dirección de inicio del bloque de memoria, también llamado referencia del array, a la variable A.



El nombre del array contiene la dirección de inicio del bloque de memoria donde se encuentra el contenido del array.

2. INICIALIZAR ARRAYS UNIDIMENSIONALES

Un array es un objeto, por lo tanto, cuando se crea, a sus elementos se les asigna automáticamente un valor inicial:

Valores iniciales por defecto para un array en java:

- 0 para arrays numéricos
- '\u0000' (carácter nulo) para arrays de caracteres
- false para arrays booleanos
- null para arrays de String y de referencias a objetos.

También podemos dar otros valores iniciales al array cuando se crea.

Los valores iniciales de un array se escriben entre llaves separados por comas.

Los valores iniciales de un array deben aparecer en el orden en que serán asignados a los elementos del array.

El número de valores determina el tamaño del array.

Por ejemplo:

```
double [] notas = {6.7, 7.5, 5.3, 8.75, 3.6, 6.5};
```

se declara el array notas de tipo double, se reserva memoria para 6 elementos y se les asignan esos valores iniciales.

Ejemplos:

```
//creación de un array de 4 elementos booleanos
boolean [] resultados = {true,false,true,false};
//creación de un array de 7 elementos de tipo String
String [] dias = {"Lunes", "Martes", "Miércoles", "Jueves", "Viernes",
"Sábado", "Domingo"};
```

3. ACCEDER A LOS ELEMENTOS DE UN ARRAY

Para acceder a cada elemento del array se utiliza el nombre del array y un índice que indica la posición que ocupa el elemento dentro del array.

El índice se escribe entre corchetes.

Se puede utilizar como índice un valor entero, una variable de tipo entero o una expresión de tipo entero.

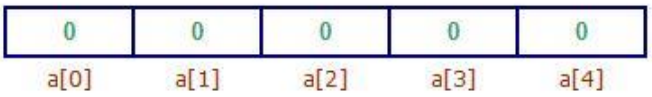
El **primer elemento** del array ocupa la **posición 0**, el segundo la posición 1, etc. **En un array de N elementos el último ocupará la posición N-1**.

Un elemento de un array se puede utilizar igual que cualquier otra variable.

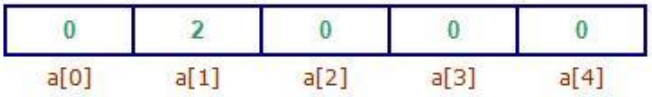
Se puede hacer con ellos las mismas operaciones que se pueden hacer con el resto de variables (incremento, decremento, operaciones aritméticas, comparaciones, etc).

Por ejemplo:

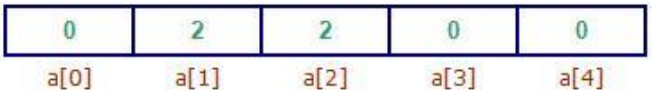
```
int m = 5;
int [] a = new int[5];
```



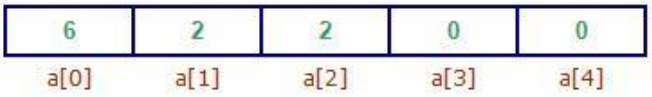

```
a[1] = 2;
```



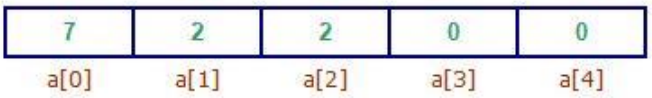

```
a[2] = a[1];
```



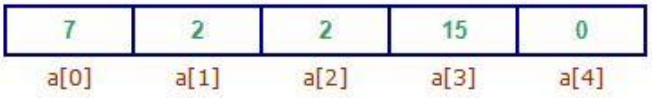

```
a[0] = a[1] + a[2] + 2;
```




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




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



Si se intenta acceder a un elemento que está fuera de los límites del array (un elemento con índice negativo o con un índice mayor que el que corresponde al último elemento del array) **el compilador no avisa del error**. El error se producirá durante la ejecución. En ese caso se lanza una excepción java.lang.ArrayIndexOutOfBoundsException.

Se puede saber el número de elementos del array mediante el atributo **length**.

Podemos utilizar length para comprobar el rango del array y evitar errores de acceso.

Por ejemplo, introducimos por teclado un valor y la posición del array donde lo vamos a guardar:

```
Scanner sc = new Scanner(System.in);
int i, valor;
int [] a = new int[10];
System.out.print("Posición: ");
i = sc.nextInt(); //pedimos una posición del array
if (i >= 0 && i < a.length) {  //si la posición introducida está
dentro de los límites del array
    System.out.print("Valor: ");
    valor = sc.nextInt();  //pedimos el valor
    a[i] = valor;
}else{
    System.out.println("Posición no válida");
}
```

4. RECORRER UN ARRAY UNIDIMENSIONAL

Para recorrer un array se utiliza una instrucción iterativa, normalmente una instrucción for, aunque también puede hacerse con while o do..while, utilizando una variable entera como índice que tomará valores desde el primer elemento al último o desde el último al primero.

Por ejemplo, el siguiente fragmento de programa Java declara un array de 7 elementos de tipo double y le asigna valores iniciales. A continuación, recorre el array, utilizando la instrucción for, para mostrar por pantalla el contenido del array.

```
double[] notas = {2.3, 8.5, 3.2, 9.5, 4, 5.5, 7.0}; //array de 7
elementos
for (int i = 0; i < 7; i++) {
     System.out.print(notas[i] + " "); //se muestra cada elemento del
array
}
```

Para evitar errores de acceso al array es recomendable **utilizar length para recorrer el array completo**.

Por ejemplo:

```
double[] notas = {2.3, 8.5, 3.2, 9.5, 4, 5.5, 7.0}; //array de 7
elementos
for (int i = 0; i < notas.length; i++) {
     System.out.print(notas[i] + " "); //se muestra cada elemento del
array
}
```

Ejemplo de recorrido de un array unidimensional en java:

Programa que lee por teclado la nota de los alumnos de una clase y calcula la nota media del grupo. También muestra los alumnos con notas superiores a la media. El número de alumnos se lee por teclado.

Este programa crea un array de elementos de tipo double que contendrá las notas de los alumnos. El tamaño del array será el número de alumnos de la clase.

Se realizan **3 recorridos** sobre el array, el primero para asignar a cada elemento las notas introducidas por teclado, el segundo para sumarlas y el tercero para mostrar los alumnos con notas superiores a la media.

```
import java.util.Scanner;

public class Recorrido2 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        int numAlum, i;
        double suma = 0, media;

        do {
            System.out.print("Número de alumnos de la clase: ");
            numAlum = sc.nextInt();
        } while (numAlum <= 0);

        double[] notas = new double[numAlum]; //se crea el array

        // Entrada de datos. Se asigna a cada elemento del array
        // la nota introducida por teclado
        for (i = 0; i < notas.length; i++) {
            System.out.print("Alumno " + (i + 1) + " Nota final: ");
            notas[i] = sc.nextDouble();
        }

        // Sumar todas las notas
        for (i = 0; i < notas.length; i++) {
            suma = suma + notas[i];
        }

        // Calcular la media
        media = suma / notas.length;

        // Mostrar la media
        System.out.printf("Nota media del curso: %.2f %n", media);

        // Mostrar los valores superiores a la media
        System.out.println("Listado de notas superiores a la media:");
        for (i = 0; i < notas.length; i++) {
            if (notas[i] > media) {
                System.out.println("Alumno numero " + (i + 1)+ " Nota
final: " + notas[i]);
            }
        }
    }
}
```

Recorrer un Array en java con foreach. Bucle for para colecciones

A partir de java 5 se incorpora una instrucción for mejorada para recorrer arrays y contenedores en general.

Permite acceder secuencialmente a cada elemento del array.

La sintaxis general es:

```
for(tipo nombreDeVariable : nombreArray){
   ......
}
```

tipo: indica el tipo de datos que contiene el array.

nombreDeVariable: variable a la que en cada iteración se le asigna el valor de cada elemento del array. Está definida dentro del for por lo que solo es accesible dentro de él.

nombreArray: es el nombre del array que vamos a recorrer.

Mediante este bucle solo podemos acceder a los elementos del array. No podemos hacer modificaciones en su contenido.

Ejemplo: El siguiente programa crea un array temperatura de 10 elementos. Lee por teclado los valores y a continuación los muestra por pantalla usando un bucle foreach.

```
import java.util.Scanner;

public class Recorrerforeach1 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        double [] temperatura = new double[10];
        int i;

        for(i = 0; i < temperatura.length; i++){
            System.out.print("Elemento " + i + ": ");
            temperatura[i] = sc.nextDouble();
        }

        for(double t: temperatura){
            System.out.print(t + " ");
        }
        System.out.println();
    }
}
```

Arrays de caracteres en Java

Un array de caracteres es un array unidimensional que contiene datos de tipo char.

Los arrays de caracteres en Java se crean de forma similar a un array unidimensional de cualquier otro tipo de datos.

Ejemplo: Array de 8 caracteres llamado cadena.

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

De forma gráfica el array de caracteres cadena se puede representar así:

\u0000	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000
cadena[0]	cadena [1]	cadena [2]	cadena [3]	cadena [4]	cadena [5]	cadena [6]	cadena [7]

Por defecto los elementos del array se inicializan con el carácter nulo (carácter \u0000 Unicode).

A diferencia de los demás arrays, se puede mostrar el contenido completo de un array de caracteres mediante una sola instrucción print, println o printf.

Para mostrar el contenido completo del array de caracteres:

```
System.out.println(cadena);
```

Mostrará 8 caracteres nulos (en blanco)

Ejemplo: Array de 5 caracteres llamado vocales. Se asignan valores iniciales: a, e, i, o, u

```
char [] vocales = {'a', 'e', 'i', 'o', 'u'};
```

a	e	i	o	u
vocales[0]	vocales [1]	vocales [2]	vocales [3]	vocales [4]

```
System.out.println(vocales);
```

Mostrará:

```
aeiou
```

El atributo length de un array de caracteres contiene el tamaño del array independientemente de que sean caracteres nulos u otros caracteres.
Por ejemplo:

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

\u0000	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000
cadena[0]	cadena [1]	cadena [2]	cadena [3]	cadena [4]	cadena [5]	cadena [6]	cadena [7]

```
System.out.println(cadena.length);
```
Muestra: 8

```
cadena[0] ='m';
```

```
cadena[1] ='n';
```

m	n	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000
cadena[0]	cadena [1]	cadena [2]	cadena [3]	cadena [4]	cadena [5]	cadena [6]	cadena [7]

```
System.out.println(cadena.length);
```
Muestra: 8

```
System.out.print(cadena);
System.out.print(cadena);
System.out.println(".");
```
Mostrará:
mn*bbbbbb*mn*bbbbbb*.
Los espacios en blanco se han representado por el carácter *b*

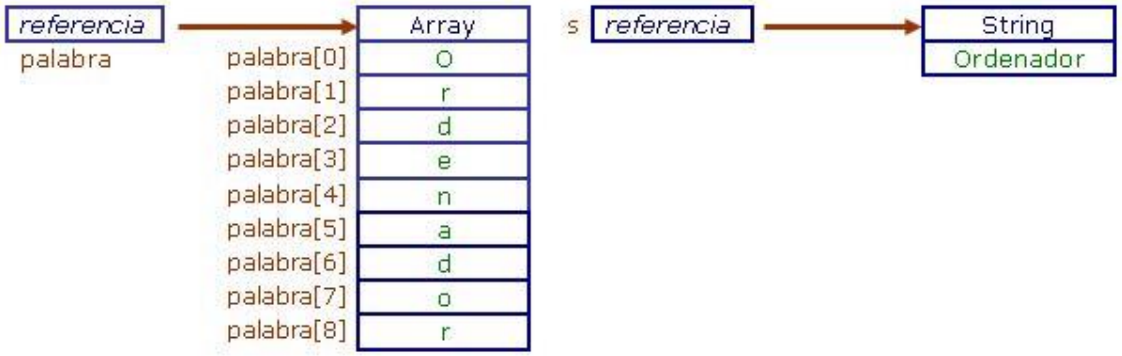
ASIGNAR UN STRING A UN ARRAY DE CARACTERES

Se puede asignar un String a un array de caracteres mediante el método toCharArray() de la clase String.
Ejemplo:

```
String s = "Ordenador";
```


```
char [] palabra = s.toCharArray();
```

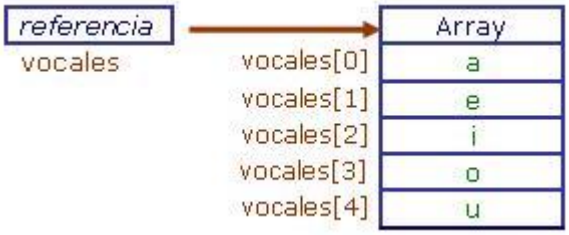
Se crea un nuevo array de caracteres con el contenido del String s y se asigna la dirección de memoria a *palabra*.



CREAR UN STRING A PARTIR DE ARRAY DE CARACTERES

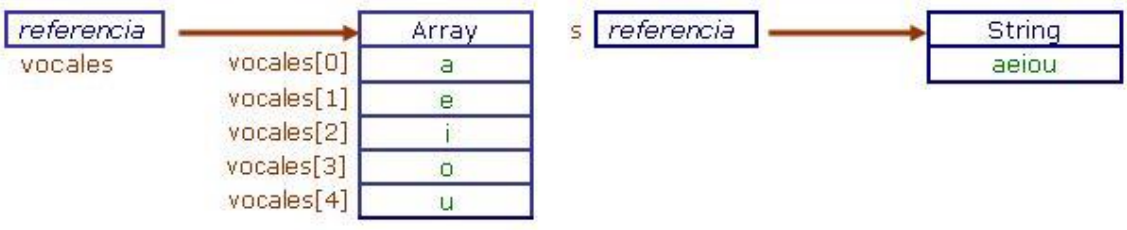
Se puede crear un String a partir de un array de caracteres utilizando el constructor de la clase String.
Ejemplo:

```
char [] vocales = {'a', 'e', 'i', 'o', 'u'};
```




```
String s = new String(vocales);
```

Se crea un nuevo String con el contenido del array vocales y se asigna la dirección de memoria a s.



RECORRER UN ARRAY DE CARACTERES UNIDIMENSIONAL

Un array de caracteres Java se puede recorrer de forma completa utilizando una instrucción iterativa, normalmente un for.

Por ejemplo:

```
char [] s = new char[10];
s[0]='a';
s[1]='b';
s[2]='c';
for(int i = 0; i < s.length; i++){
    System.out.print(s[i]+ " ");
}
```

Mostrará todos los caracteres del array, incluidos los nulos.

Si los caracteres no nulos se encuentran al principio del array se puede recorrer utilizando un while, mientras que no encontremos un carácter nulo.

Por ejemplo:

```
char [] s = new char[10];
s[0]='a';
s[1]='b';
s[2]='c';
int i = 0;
while(s[i]!='\0'){
      System.out.print(s[i]);
      i++;
}
```

Muestra los caracteres del array hasta que encuentra el primer nulo.

Matrices en Java

Un array en Java puede tener más de una dimensión. El caso más general son los arrays bidimensionales también llamados **matrices** o **tablas**.

La dimensión de un array la determina el número de índices necesarios para acceder a sus elementos.

Los vectores que hemos visto en otra entrada anterior son arrays unidimensionales porque solo utilizan un índice para acceder a cada elemento.

Una matriz necesita dos índices para acceder a sus elementos. Gráficamente podemos representar una matriz como una tabla de n filas y m columnas cuyos elementos son todos del mismo tipo.

La siguiente figura representa un array *M* de 3 filas y 5 columnas:

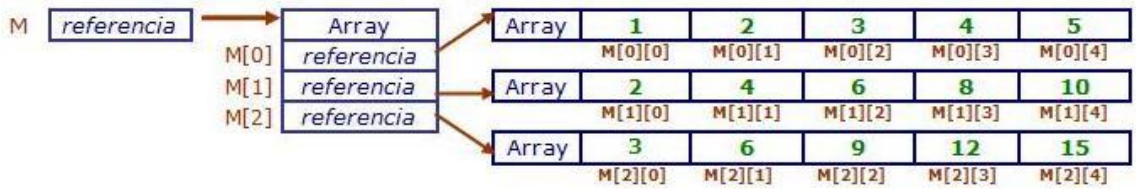
	0	1	2	3	4
0	1	2	3	4	5
1	2	4	6	8	10
2	3	6	9	12	15

A los elementos del array se accede mediante la fila y columna donde están situados.

A efectos prácticos, cuando trabajamos con arrays bidimensionales podemos pensar en una tabla como la que se muestra en la imagen anterior donde los elementos están distribuidos en filas y columnas.

Pero en realidad **una matriz en Java es un array de arrays**.

La disposición real en memoria del array anterior la podemos representar gráficamente de esta forma:



M es el nombre del array.
M contiene la dirección de memoria (referencia) de un array unidimensional de 3 elementos.
Cada elemento de este array unidimensional contiene la dirección de memoria de otro array unidimensional.
Cada uno de estos últimos arrays unidimensionales contiene los valores de cada fila de la matriz.
M.length indica el número de filas de la matriz. En este ejemplo el número de filas (M.length) es 3.
M[i].length indica el nímero de columnas de la fila i. En este ejemplo la longitud de cada fila del array (M[i].length) es 5.
Para acceder a cada elemento de la matriz se utilizan dos índices. El primero indica la fila y el segundo la columna.
M[0][2] = 9; //asigna el valor 9 al elemento situado en la primera fila (fila 0) y tercera columna (fila 2).
No debemos olvidar que la primera fila de una matriz es la fila 0 y la primera columna de una matriz es la columna 0.

CREAR MATRICES EN JAVA

Se crean de forma similar a los arrays unidimensionales, añadiendo un índice.

Por ejemplo:
matriz de datos de tipo int llamado ventas de 4 filas y 6 columnas:

```
int [][] ventas = new int[4][6];
```

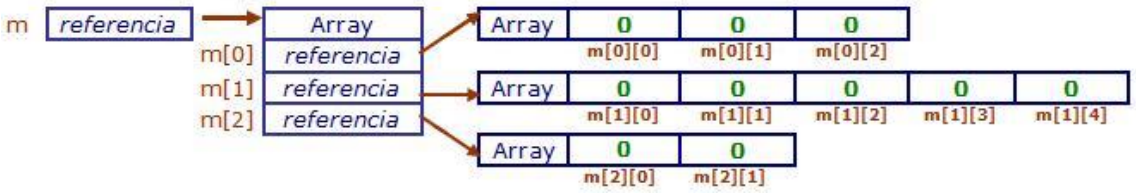
matriz de datos double llamado temperaturas de 3 filas y 4 columnas:

```
double [][] temperaturas = new double[3][4];
```

En Java se pueden crear **arrays *irregulares*** en los que el número de elementos de cada fila es variable. Solo es obligatorio indicar el número de filas.

Por ejemplo:
```
int [][] m = new int[3][];
```

crea una matriz m de 3 filas.
A cada fila se le puede asignar un número distinto de columnas:
```
m[0] = new int[3];
m[1] = new int[5];
m[2] = new int[2];
```
Gráficamente podemos representar la disposición real en memoria del array anterior así:



INICIALIZAR MATRICES

Un array es un objeto, por lo tanto, cuando se crea, a sus elementos se les asigna automáticamente un valor inicial:

- 0 para arrays numéricos
- '\u0000' (carácter nulo) para arrays de caracteres
- *false* para arrays booleanos
- *null* para arrays de String y de referencias a objetos

También podemos dar otros valores iniciales al array cuando se crea.
Los valores iniciales se escriben entre llaves separados por comas.
Los valores que se le asignen a cada fila aparecerán a su vez entre llaves separados por comas.
El número de valores determina el tamaño de la matriz.
Por ejemplo:

```
int [][] numeros = {{6,7,5},{3, 8, 4}, {1,0,2}, {9,5,2}};
```
se crea la matriz numeros de tipo int, de 4 filas y 3 columnas, y se le asignan esos valores iniciales.
Asignando valores iniciales se pueden crear también matrices irregulares.

Por ejemplo, la instrucción:

```
int [][] a = {{6,7,5,0,4}, {3, 8, 4}, {1,0,2,7}, {9,5}};
```
crea una matriz irregular de 4 filas. La primera de 5 columnas, la segunda de 3, la tercera de 4 y la cuarta de 2.

RECORRER MATRICES

Para recorrer una matriz se anidan dos bucles for. En general para recorrer un array multidimensional se anidan tantas instrucciones for como dimensiones tenga el array.

Ejemplo de recorrido de una matriz en Java:

Programa que lee por teclado números enteros y los guarda en una matriz de 5 filas y 4 columnas. A continuación, muestra los valores leídos, el mayor y el menor y las posiciones que ocupan.

```
import java.util.Scanner;

public class Bidimensional2 {

    public static void main(String[] args) {

        final int FILAS = 5, COLUMNAS = 4;
        Scanner sc = new Scanner(System.in);
        int i, j, mayor, menor;
        int filaMayor, filaMenor, colMayor, colMenor;

        int[][] A = new int[FILAS][COLUMNAS]; //Se crea una matriz de
5 filas y 4 columnas

        //Se introducen por teclado los valores de la matriz
        System.out.println("Lectura de elementos de la matriz: ");
        for (i = 0; i < FILAS; i++) {
            for (j = 0; j < COLUMNAS; j++) {
                System.out.print("A[" + i + "][" + j + "]= ");
                A[i][j] = sc.nextInt();
            }
        }

        //Mostrar por pantalla los valores que contiene la matriz
        System.out.println("valores introducidos:");
        for (i = 0; i < A.length; i++) {
            for (j = 0; j < A[i].length; j++) {
                System.out.print(A[i][j] + " ");
            }
            System.out.println();
        }

        //Calcular el mayor valor de la matriz y el menor.
        //Obtener las posiciones que ocupan el mayor y el menor dentro
de la matriz
        mayor = menor = A[0][0]; //se toma el primer elemento de la
matriz como mayor y menor
        filaMayor = filaMenor = colMayor = colMenor = 0;

        //mediante dos bucles for anidados recorremos la matriz
        //buscamos el mayor, el menor y sus posiciones
        for (i = 0; i < A.length; i++) {  //para cada fila de la
matriz
            for (j = 0; j < A[i].length; j++) { //para cada columna de
la matriz
                if (A[i][j] > mayor) {
                    mayor = A[i][j];
                    filaMayor = i;
                    colMayor = j;
                } else if (A[i][j] < menor) {
                    menor = A[i][j];
                    filaMenor = i;
                    colMenor = j;
                }
            }
```

```
        }

        //Mostrar por pantalla el mayor elemento de la matriz,el menor
y las posiciones que ocupan
        System.out.print("Elemento mayor: " + mayor);
        System.out.println(" Fila: "+ filaMayor + " Columna: " +
colMayor);
        System.out.print("Elemento menor: " + menor);
        System.out.println(" Fila: "+ filaMenor + " Columna: " +
colMenor);
    }
}
```

Ejemplo de recorrido de una matriz irregular en Java:

Programa que crea una matriz irregular de enteros. El número de filas se pide por teclado. Para cada fila se pedirá el número de columnas que tiene. El número mínimo de filas debe ser 2 y el número mínimo de columnas debe ser 1. A continuación asigna a cada elemento de la matriz un número aleatorio del 1 al 5. Finalmente muestra por pantalla el contenido de la matriz.

```
import java.util.Random;
import java.util.Scanner;

public class MatrizIrregular {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Random rnd = new Random();
        int filas, columnas;

        do { //lectura de las filas
            System.out.print("Introduce número de filas: ");
            filas = sc.nextInt();
            if (filas < 2) {
                System.out.println("Valor no válido");
            }
        } while (filas < 2);

        //se crea el array solo con el número de filas
        //el número de columnas se deja vacío
        //el número de columnas para cada fila lo indicaremos después
        int[][] A = new int[filas][];

        //para cada fila pedimos el número de columnas que tendrá
        for (int i = 0; i < A.length; i++) {
            do {
                System.out.print("Número de columnas para la fila " +
i + ": ");
                columnas = sc.nextInt();
                if (columnas < 1) {
                    System.out.println("Valor no válido");
                }
            } while (columnas < 1);
            //Se crea un array del tamaño indicado y se le asigna a la
fila
            A[i] = new int[columnas];
        }

        //A cada elemento de la matriz se le asigna un valor aleatorio
del 1 al 5
```

```
        for (int i = 0; i < A.length; i++) {
            for (int j = 0; j < A[i].length; j++) {
               A[i][j] = rnd.nextInt(5) + 1;
            }
        }

        //Se muestra el contenido de la matriz
        System.out.println("Contenido del array:");
        for (int i = 0; i < A.length; i++) {
            for (int j = 0; j < A[i].length; j++) {
                System.out.print(A[i][j] + " ");
            }
            System.out.println();
        }
    }
}
```

Copiar arrays en Java

CÓMO COPIAR UN ARRAY EN OTRO EN JAVA

Vamos a explicar diferentes formas de copiar arrays en Java.

Cuando nos surge la necesidad de realizar una copia de un array puede parecernos lógico utilizar una instrucción de asignación como haríamos con una variable de tipo primitivo.

Si disponemos de un array llamado A de 5 elementos de tipo entero:

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

y queremos copiar su contenido en otro array B mediante la instrucción:

```
int [] B = A;
```

en realidad, no estamos haciendo una copia del array A. Lo que estamos haciendo es asignar a la variable B el contenido de la variable A y lo que contiene A es la dirección de memoria donde se encuentra el array.

Lo que hemos conseguido **no es una copia del array** A sino que hemos creado un **alias o referencia alternativa al array**.

De esta forma ahora tenemos dos formas de acceder al mismo array: utilizando la variable A o utilizando la variable B. Ambas nos llevan a la misma zona de memoria donde se encuentra el array.

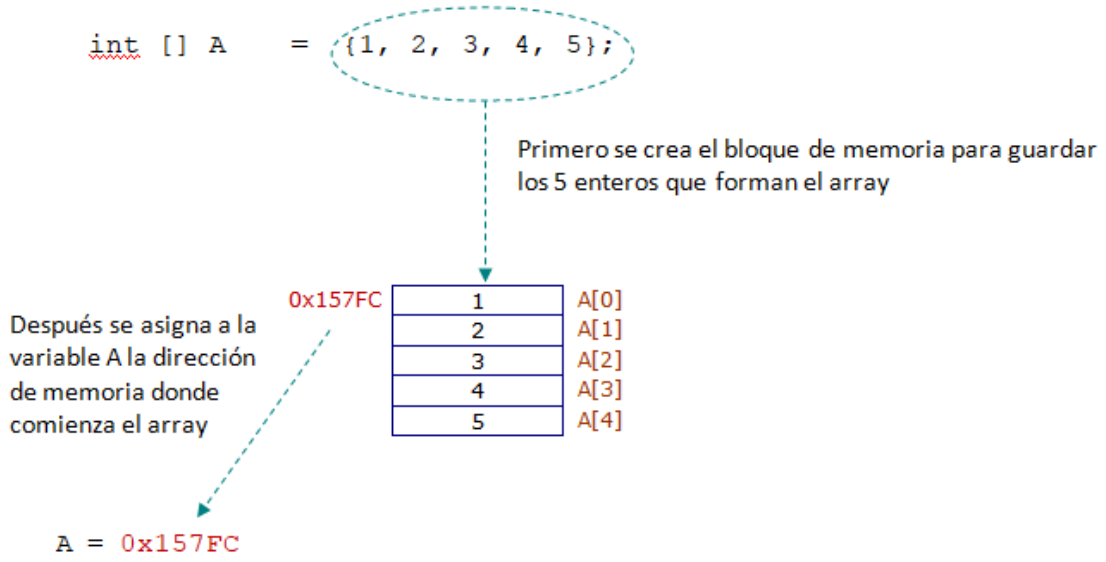
Tenemos **un solo array al que podemos acceder mediante dos variables**.

Para entenderlo mejor vamos a representarlo de forma gráfica.

Cuando se ejecuta la instrucción que crea el array A:

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

se realizan dos operaciones: primero crea un bloque de memoria para almacenar el array de 5 enteros y después se asigna la dirección de inicio del bloque de memoria, también llamado referencia del array, a la variable A.



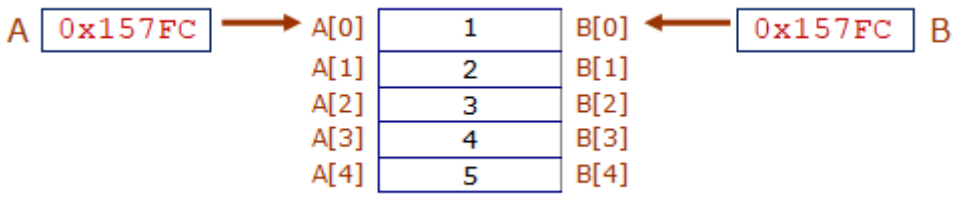
Si ahora declaramos la variable B y le asignamos el valor de A:

```
int [] B = A;
```

Esta instrucción crea una variable B y le asigna el contenido de la variable A. Lo que estamos haciendo es copiar en B la dirección de memoria del array A.

```
B = 0x157FC;
```

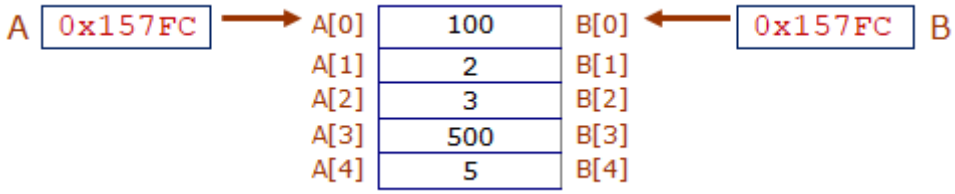
En memoria tenemos un solo array al que podemos acceder mediante dos variables: A y B.



Ahora podemos realizar operaciones sobre el array utilizando las dos variables:

```
A[0] = 100;

B[3] = 500;
```

Ambas modifican el mismo array:



Hemos podido comprobar que de esta forma no estamos copiando un array en otro.

Para copiar el contenido de un array en otro podemos hacerlo de varias maneras:

- Creando un nuevo array y mediante un bucle y realizar la copia elemento a elemento.
- Utilizando el método clone().
- Utilizando el método System.arraycopy().

Copiar un array unidimensional mediante un bucle elemento a elemento.

Esta es la forma trivial de hacerlo.

Si tenemos un array A de enteros

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

y queremos hacer una copia de este array A en otro array llamado B lo que haremos será crear el nuevo array B y copiar elemento a elemento:

```
int[] B = new int[5];     //se crea un nuevo array B

for (int i = 0; i < A.length; i++) {  //se copia cada elemento de A en B
     B[i] = A[i];
}
```

Copiar un array unidimensional utilizando el método clone.

De forma general, para copiar arrays con el método clone se escribe la instrucción:

```
arrayDestino = arrayOrigen.clone();
```

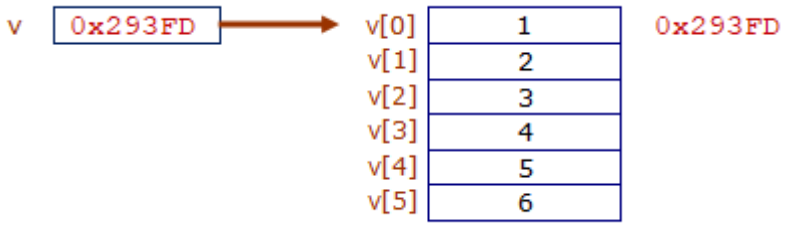
El método *clone* crea en memoria una copia del *arrayOrigen*. La dirección de memoria del array creado a *arrayDestino*. De esta forma **obtenemos dos arrays distintos con el mismo contenido**.

Ejemplo:

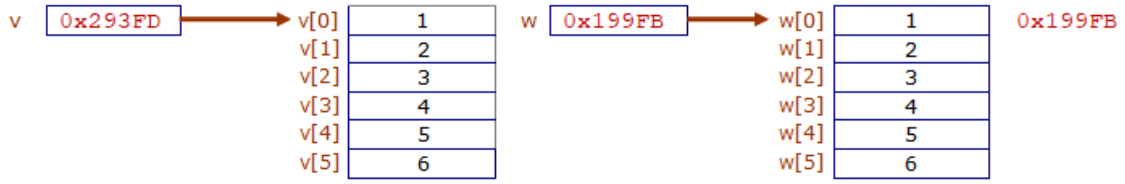
```
int[] v = {1, 2, 3, 4, 5, 6};

int [] w = v.clone();
```

El proceso realizado se puede representar de forma gráfica de esta forma:

```
int[] v = {1, 2, 3, 4, 5, 6};
```




```
int [] w = v.clone();
```



La instrucción v.clone() crea en memoria un nuevo array y copia en el nuevo array el contenido del array v. La dirección de inicio del array creado se asigna a la variable w.

Copiar un array unidimensional utilizando System.arraycopy().

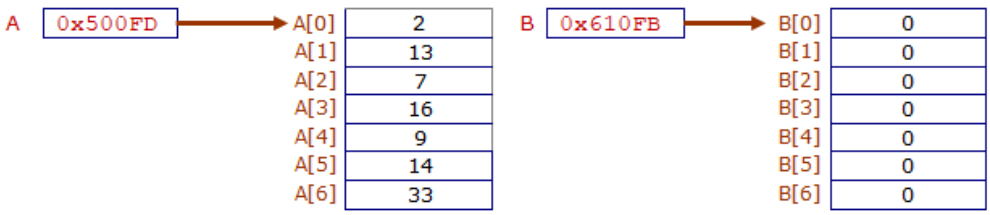
Este método copia el contenido de un array en otro array que ya debe existir.

Para realizar la copia hay que indicar al método arraycopy 5 argumentos en este orden:

1. el array origen
2. desde qué posición del array origen vamos a copiar
3. el array destino
4. a partir de qué posición vamos a copiar en el array de destino
5. cantidad de elementos a copiar

Ejemplo: disponemos de los arrays A y B

```
int[] A = {2, 13, 7, 16, 9, 14, 33};
int[] B = new int[7];
```

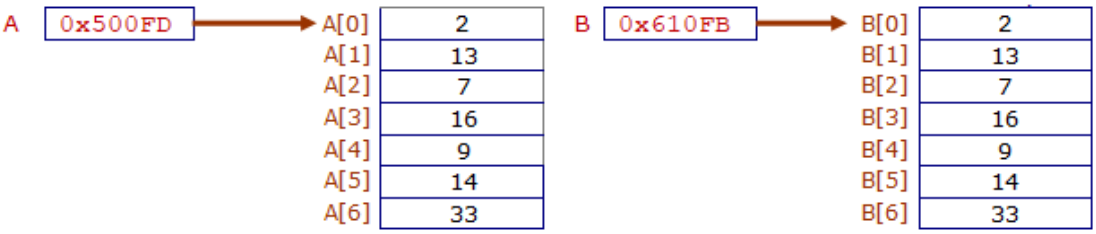


Para copiar el array A en el array B se escribe la instrucción:

System.arraycopy(A, 0, B, 0, A.length);

Los valores que hemos escrito indican el nombre del array de origen (A), a partir de qué posición del array A vamos a copiar, en este caso desde la primera posición (0), a continuación el array de destino (B), a partir de qué posición vamos a copiar en el array de destino, en este caso desde la primera posición (0) y finalmente el número de elementos a copiar, este caso queremos copiar todos los elementos de A por lo que hemos indicado A.length.

Como resultado se ha realizado una copia completa del array A en el Array B.



La ventaja que nos ofrece este método es que podemos copiar solo una parte de un array en otro. Además, los arrays origen y destino pueden tener tamaños diferentes.

Por ejemplo, disponemos de los siguientes arrays:

```
int[] A = {2, 13, 7, 16, 9, 14, 33};
int[] B = new int[3];
int[] C = new int[3];
int[] D = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
```

Vamos a copiar los tres primeros elementos del array A en el array B, los tres últimos elementos de A en el array C y los elementos de las posiciones 3 y 4 del array A en las posiciones 2 y 3 del array D.

Escribiremos las instrucciones:

```
System.arraycopy(A, 0, B, 0, 3); //copia los tres primeros elementos
de A en B
```

```
System.arraycopy(A, 4, C, 0, 3); //copia los tres últimos elementos de A en C
System.arraycopy(A, 3, D, 2, 2); //copia los elementos 3 y 4 de A en los elementos 2 y 3 de D
```

El contenido final de los arrays después de realizar la copia es el siguiente:

```
Array B:

2 13 7

Array C:

9 14 33

Array D:

1 1 16 9 1 1 1 1 1 1
```

Arrays y métodos en Java

Veamos cómo pasar un array como parámetro a un método y cómo devolver un array mediante la instrucción return.

Sabemos que el nombre de un array contiene la dirección de memoria donde se almacena el array. En la entrada Arrays unidimensionales tienes la explicación de esto.

Cuando se pasa un array a un método como parámetro, el método recibe la dirección de memoria donde se almacena el array. El parámetro formal del método se convierte en un alias del array, es decir, una forma alternativa de acceder al mismo array, de forma que dentro del método tenemos acceso al array original y se puede modificar su contenido.

Veamos un ejemplo para entenderlo mejor.

El siguiente programa crea un array de 5 elementos de tipo int y mediante un método asigna a los elementos del array los valores 1, 2, 3, 4, 5.

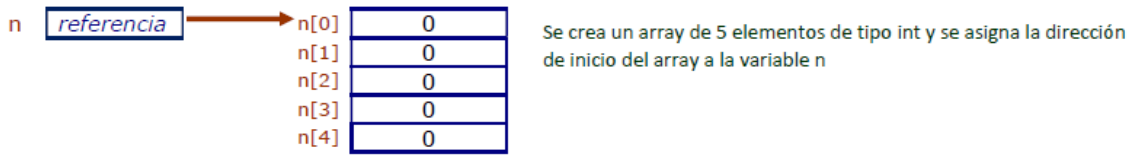
```
public class PasarArrayMetodo {
    public static void main(String[] args) {
        int [] n = new int[5];
        for(int i = 0; i < n.length; i++){
            System.out.print(n[i] + " ");
        }
        System.out.println();
        inicializar(n);
        for(int i = 0; i < n.length; i++){
            System.out.print(n[i] + " ");
        }
    }

    public static void inicializar(int [] x){
        for(int i = 0; i < x.length; i++){
            x[i] = i+1;
```

```
        }
    }
}
```

Para entender cómo funciona el paso de un array a un método vamos a representar de forma gráfica lo que realiza este programa.

En el método main se crea el array n:

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

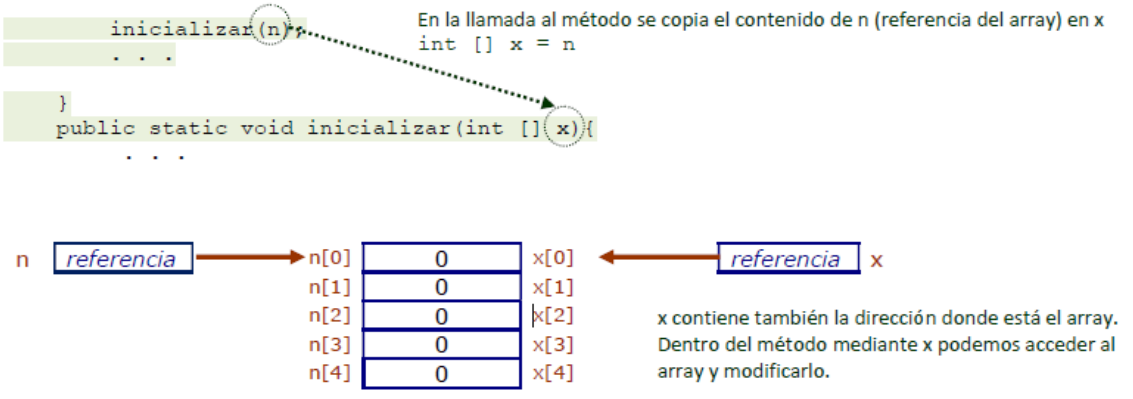


A continuación, se ejecuta el for:

```
for(int i = 0; i < n.length; i++) {
        System.out.print(n[i] + " ");
}
```

Muestra: 0 0 0 0 0

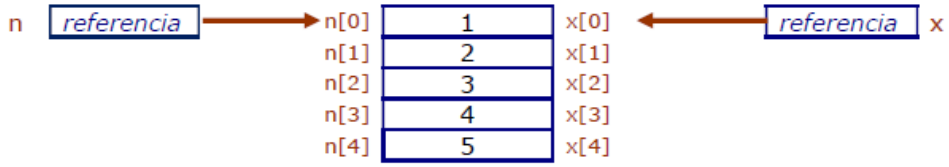
Después del salto de línea se llama al método inicializar y se le pasa el array n:



La ejecución continúa dentro del método inicializar con el for que asigna un valor a los elementos del array:

```
public static void inicializar(int [] x){
        for(int i = 0; i < x.length; i++){
            x[i] = i+1;
        }
}
```

x contiene la dirección donde está el array.
En este for se modifica el contenido del array.



El método finaliza y la ejecución del programa continúa en el método main. Se ejecuta el for que muestra el contenido del array:

```
...
inicializar(n);
for(int i = 0; i < n.length; i++){
    System.out.print(n[i] + " ");
}

}
```

Cuando volvemos al método main y recorremos el array comprobamos que el array se ha modificado.

Muestra: 1 2 3 4 5

En este ejemplo hemos visto cómo se pasa un array a un método en java y cómo desde dentro del método podemos modificar el array original.

Un método además de recibir un array como parámetro, puede **devolver un array mediante la instrucción return**.

Ejemplo:

Método que crea un array de N elementos de tipo entero. A cada elemento le asigna un valor aleatorio entre 1 y 10. El valor de N lo recibe como parámetro. El método devuelve el array creado.

En el método main se llamará al método para crear el array y después se mostrará el contenido del array.

```
public class ArraysMetodos {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int elementos;
        do {
            System.out.print("Introduce tamaño del array: ");
            elementos = sc.nextInt();
```

```
            if (elementos < 1) {
                System.out.println("Valor no válido");
            }
        } while (elementos < 1);
        int[] numeros = crearArray(elementos); //se llama al método
        //mostrar el contenido del array
        for(int i = 0; i < numeros.length; i++){
            System.out.println(numeros[i]);
        }
    }
    //método para crear y devolver un aray de N elementos de tipo int
    //los valores de los elementos del array se asignan de forma
aleatoria
    //con valores entre 1 y 10
    public static int[] crearArray(int N) {
        Random rnd = new Random();
        int[] A = new int[N]; //se crea un array de enteros de tamaño
N
        //se asigna a cada elemento del array un número aleatorio
entre 1 y 10
        for (int i = 1; i < A.length; i++) {
            A[i] = rnd.nextInt(10) + 1;
        }
        return A; //se devuelve el array creado
    }
}
```

Veamos ahora un ejemplo más completo de programa con métodos y arrays.

Ejemplo: Programa que crea un array de enteros. El tamaño del array se pide por teclado. Después se introducen por teclado los valores de los elementos del array. Finalmente se muestra el contenido del array y el mayor valor que contienen el array. Se utilizarán los siguientes métodos:

- Método para leer el tamaño del array.
- Método para crear el array.
- Método para introducir por teclado los valores de los elementos del array.
- Método para mostrar el array.
- Método que calcula y devuelve el mayor elemento del array.

```
public class ArraysMetodos2 {
    public static void main(String[] args) {

        int elementos = leerTamaño();
        int[] numeros = crearArray(elementos);
        llenarArray(numeros);
        mostrarArray(numeros);
        System.out.println("El mayor elemento es: " +
obtenerMayor(numeros));

    }
```

```
    //Método que lee y devuelve un entero que indica el tamaño de un
array
    public static int leerTamaño() {
        Scanner sc = new Scanner(System.in);
        int N;
        do {
            System.out.print("Introduce tamaño del array > 0: ");
            N = sc.nextInt();
            if (N < 1) {
                System.out.println("Valor no válido");
            }
        } while (N < 1);
        return N;
    }


    //método para crear y devolver un aray de N elementos de tipo int
    public static int[] crearArray(int N) {
        return new int[N]; //se crea un array de enteros de tamaño N y
se devuelve
    }

    //método que asigna valores a un array de enteros
    //los valores se introducen por teclado
    public static void llenarArray(int[] A) {
        Scanner sc = new Scanner(System.in);
        for (int i = 0; i < A.length; i++) {
            System.out.print("Elemento " + i + ": ");
            A[i] = sc.nextInt();
        }
    }

    //método para mostrar el contenido de un array de enteros
    public static void mostrarArray(int[] A) {
        for (int i = 0; i < A.length; i++) {
            System.out.print(A[i] + " ");
        }
    }

    //método que calcula y devuelve el mayor elemento de un array de
enteros
    public static int obtenerMayor(int [] A){
        int mayor = A[0]; //tomamos el primer elemento como mayor
        for(int i = 1; i < A.length; i++){ //desde el segundo elemento
hasta el final
            if(A[i] > mayor)
            {
                //comprobamos si es mayor que el mayor actual
                mayor = A[i];
            }
        }
        return mayor;  //devolvemos el valor mayor obtenido
    }

}
```

Java ArrayList. Estructura dinámica de datos

DECLARACIÓN Y CREACIÓN DE UN ARRAYLIST

ArrayList es una clase contenedora genérica que implementa arrays dinámicos de objetos de cualquier tipo.

De forma general un ArrayList en Java se crea de la siguiente forma:

```
ArrayList nombreArray = new ArrayList();
```
ArrayList nombreArray = new ArrayList();

Esta instrucción crea el ArrayList *nombreArray* vacío.

Un arrayList declarado así puede contener objetos de cualquier tipo.

Por ejemplo:

```
ArrayList a = new ArrayList();
a.add("Lenguaje");
a.add(3);
a.add('a');
a.add(23.5);
```
Los elementos del arrayList a son:
"Lenguaje" 2 'a' 23.5

Es decir, **un ArrayList puede contener objetos de tipos distintos**.

En este ejemplo, el primer objeto que se añade es el String "Lenguaje". El resto no son objetos. Son datos de tipos básicos pero el compilador los convierte automáticamente en objetos de su **clase envolvente** (clase contenedora o wrapper) antes de añadirlos al array.

Un array al que se le pueden asignar elementos de distinto puede tener alguna complicación a la hora de trabajar con él. Por eso, una alternativa a esta declaración es indicar el tipo de objetos que contiene. En este caso, el array solo podrá contener objetos de ese tipo.
De forma general:

```
ArrayList<tipo> nombreArray = new ArrayList();
```
tipo debe ser una clase. Indica el tipo de objetos que contendrá el array.
No se pueden usar tipos primitivos. Para un tipo primitivo se debe utilizar su clase envolvente.
Por ejemplo:
```
ArrayList<Integer> numeros = new ArrayList();
```
Crea el array *numeros* de enteros.

MÉTODOS DE ARRAYLIST

Algunos métodos que proporciona ArrayList son:

MÉTODO	DESCRIPCIÓN
size()	Devuelve el número de elementos (int)
add(X)	Añade el objeto X al final. Devuelve true.
add(posición, X)	Inserta el objeto X en la posición indicada.
get(posicion)	Devuelve el elemento que está en la posición indicada.

remove(posicion)	Elimina el elemento que se encuentra en la posición indicada. Devuelve el elemento eliminado.
remove(X)	Elimina la primera ocurrencia del objeto X. Devuelve true si el elemento está en la lista.
clear()	Elimina todos los elementos.
set(posición, X)	Sustituye el elemento que se encuentra en la posición indicada por el objeto X. Devuelve el elemento sustituido.
contains(X)	Comprueba si la colección contiene al objeto X. Devuelve true o false.
indexOf(X)	Devuelve la posición del objeto X. Si no existe devuelve -1

Los puedes consultar todos en:

http://docs.oracle.com/javase/9/docs/api/java/util/ArrayList.html

RECORRER UN ARRAYLIST

Podemos recorrerlo de forma clásica con un **bucle for:**

```
for(int i = 0;i < array.size();i++){
    System.out.println(array.get(i));
}
```

Con un **bucle foreach:**
Si suponemos el array de enteros llamado *numeros:*

```
for(Integer i: numeros){
    System.out.println(i);
}
```

Si el array contiene objetos de tipos distintos o desconocemos el tipo:

```
for(Object o: nombreArray){
    System.out.println(o);
}
```

Utilizando un **objeto Iterator**.
http://docs.oracle.com/javase/9/docs/api/java/util/Iterator.html
La ventaja de utilizar un Iterador es que no necesitamos indicar el tipo de objetos que contiene el array.

Iterator tiene como métodos:

hasNext: devuelve true si hay más elementos en el array.
next: devuelve el siguiente objeto contenido en el array.

Ejemplo:

```
ArrayList<Integer> numeros = new ArrayList();
......
//se insertan elementos
.....

Iterator it = numeros.iterator();        //se crea el iterador it para
recorrer el array numeros
while(it.hasNext()){                     //mientras queden elementos
      System.out.println(it.next());     //se obtienen y se muestran
}
```

EJEMPLOS DE USO DE ARRAYLIST

Ejemplo 1:

```
ArrayList<String> nombres = new ArrayList();
nombres.add("Ana");
nombres.add("Luisa");
nombres.add("Felipe");
System.out.println(nombres); // [Ana, Luisa, Felipe]
nombres.add(1, "Pablo");
System.out.println(nombres); // [Ana, Pablo, Luisa, Felipe]
nombres.remove(0);
System.out.println(nombres); // [Pablo, Luisa, Felipe]
nombres.set(0,"Alfonso");
System.out.println(nombres); // [Alfonso, Luisa, Felipe]
String s = nombres.get(1);
String ultimo = nombres.get(nombres.size() - 1);
System.out.println(s + " " + ultimo);  // Luisa Felipe
```

Ejemplo 2: Escribe un programa que lea números enteros y los guarde en un ArrayList hasta que se lea un 0 y muestra los números leídos, su suma y su media.

```
import java.util.*;

public class ArrayList2 {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        ArrayList<Integer> numeros = new ArrayList();
        int n;

        do {
            System.out.println("Introduce números enteros. 0 para
acabar: ");
            System.out.println("Numero: ");
            n = sc.nextInt();
            if (n != 0){
                numeros.add(n);
            }
        }while (n != 0);

        System.out.println("Ha introducido: " + numeros.size() + "
números:");

        //mostrar el arrayList completo
        System.out.println(numeros);

        //recorrido usando un iterador para mostrar un elemento por
línea
```

```
        Iterator it = numeros.iterator();
        while(it.hasNext()){
              System.out.println(it.next());
        }

        //recorrido usando foreach para sumar los elementos
        double suma = 0;
        for(Integer i: numeros){
            suma = suma + i;
        }
        System.out.println("Suma: " + suma);
        System.out.println("Media: " + suma/numeros.size());
    }
}
```

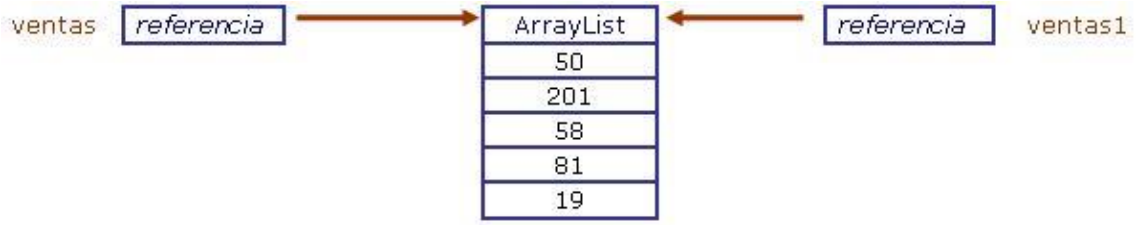
COPIAR UN ARRAYLIST

El nombre de un ArrayList contiene la referencia al ArrayList, es decir, la dirección de memoria donde se encuentra el ArrayList, igual que sucede con los arrays estáticos. Si disponemos de un ArrayList de enteros llamado ventas:

La instrucción:

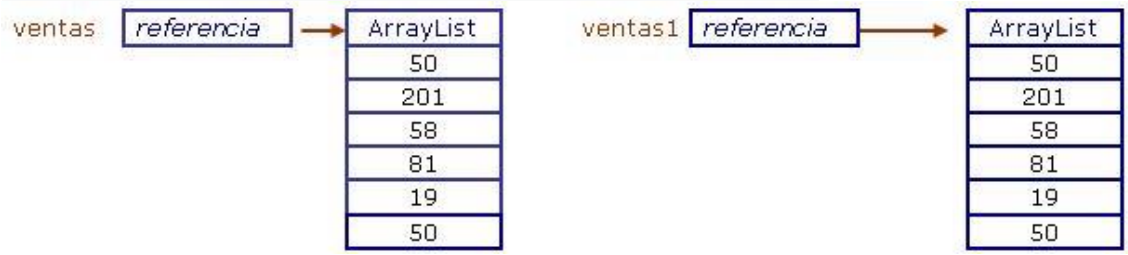
```
ArrayList<Integer> ventas1 = ventas;
```

No copia el array ventas en el nuevo array ventas1 sino que **crea un alias**:



De esta forma tenemos dos formas de acceder al mismo ArrayList: mediante la referencia ventas y mediante la referencia ventas1.
Para hacer una copia podemos hacerlo de forma manual elemento a elemento o se puede pasar la referencia del ArrayList original al constructor del nuevo:

```
ArrayList<Integer> ventas1 = new ArrayList(ventas);
```



ARRAYLIST COMO PARÁMETRO DE UN MÉTODO

Un ArrayList puede ser usado como parámetro de un método. Además, un método puede devolver un ArrayList mediante la sentencia return.

Ejemplo: Método que recibe un ArrayList de String y lo modifica invirtiendo su contenido:

```
import java.util.*;

public class ArrayList4 {

    public static void main(String[] args) {

        ArrayList<String> nombres = new ArrayList();
        nombres.add("Ana");
        nombres.add("Luisa");
        nombres.add("Felipe");
        nombres.add("Pablo");
        System.out.println(nombres);
        nombres = invertir(nombres);
        System.out.println(nombres);
    }

    public static ArrayList<String> invertir(ArrayList<String>
nombres) {
        // Crea una lista para el resultado del método
        ArrayList<String> resultado = new ArrayList();
        // Recorre la lista de nombres en orden inverso
        for (int i = nombres.size() - 1; i >= 0; i--) {
             // Añade cada nombre al resultado
              resultado.add(nombres.get(i));
        }
        return resultado;
    }
}
```

ARRAYS BIDIMENSIONALES UTILIZANDO ARRAYLIST

Un ArrayList es un array unidimensional, pero con ellos podemos *simular* arrays de dos o más dimensiones anidando ArrayLists.
Para crear una matriz lo que creamos es un ArrayList cuyos elementos son a su vez ArrayList. Esto se puede extender sucesivamente y obtener arrays de más dimensiones.
Ejemplo:
Programa que lee las notas de 10 alumnos y las guarda en un ArrayList Bidimensional. Cada alumno tiene un número indeterminado de notas. La lectura de notas de cada alumno acaba cuando se introduce un número negativo. Finalmente se muestran todas las notas de todos los alumnos.

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

        Scanner sc = new Scanner(System.in);
        final int numAlumnos = 10;  //número de alumnos
        int i, j, nota, cont = 1;

        //crear un ArrayList bidimensional de enteros vacío
        //Realmente se crea un ArrayList de ArrayLists de enteros
        ArrayList<ArrayList<Integer>> array = new ArrayList();
```

```
        //Se leen las notas de cada alumno.
        System.out.println("Introduzca notas. < 0 para acabar");
        for(i=0;i < numAlumnos;i++){
            cont = 1;
            System.out.println("Alumno " + (i+1) + ": ");
            System.out.print("Nota " + cont + ": ");
            nota = sc.nextInt();

           //para cada alumno se añade una nueva fila vacía
           //esto es necesario porque el arrayList se crea vacío
            array.add(new ArrayList<Integer>());

            while(nota >= 0){
                array.get(i).add(nota); //en la fila i se añade un
nueva nota
                cont++;
                System.out.print("Nota " + cont + ": ");
                nota = sc.nextInt();
            }
        }

        //Mostrar todas las notas
        System.out.println("Notas de alumnos");
        for(i=0;i < array.size();i++){                          //para
cada alumno (para cada fila)
            System.out.print("Alumno " + i + ": ");
            for(j=0;j < array.get(i).size();j++){               //se
recorre todas la columnas de la fila
                System.out.print(array.get(i).get(j) + " "); //se
obtiene el elemento i,j
            }
            System.out.println();
        }
}
```

ArrayList de Objetos en Java

En esta entrada vamos a escribir un programa Java que crea un ArrayList de Objetos de tipo Coche. El programa pide por teclado los datos de los coches y los guarda en el array. A continuación, utilizará el ArrayList para mostrar por pantalla lo siguiente:

- Todos los coches introducidos.
- Todos los coches de una marca determinada.
- Todos los coches con menos de un número determinado de Kilómetros.
- El coche con mayor número de Kilómetros.
- Todos los coches ordenados por número de kilómetros de menor a mayor.

Primero creamos la clase Coche:

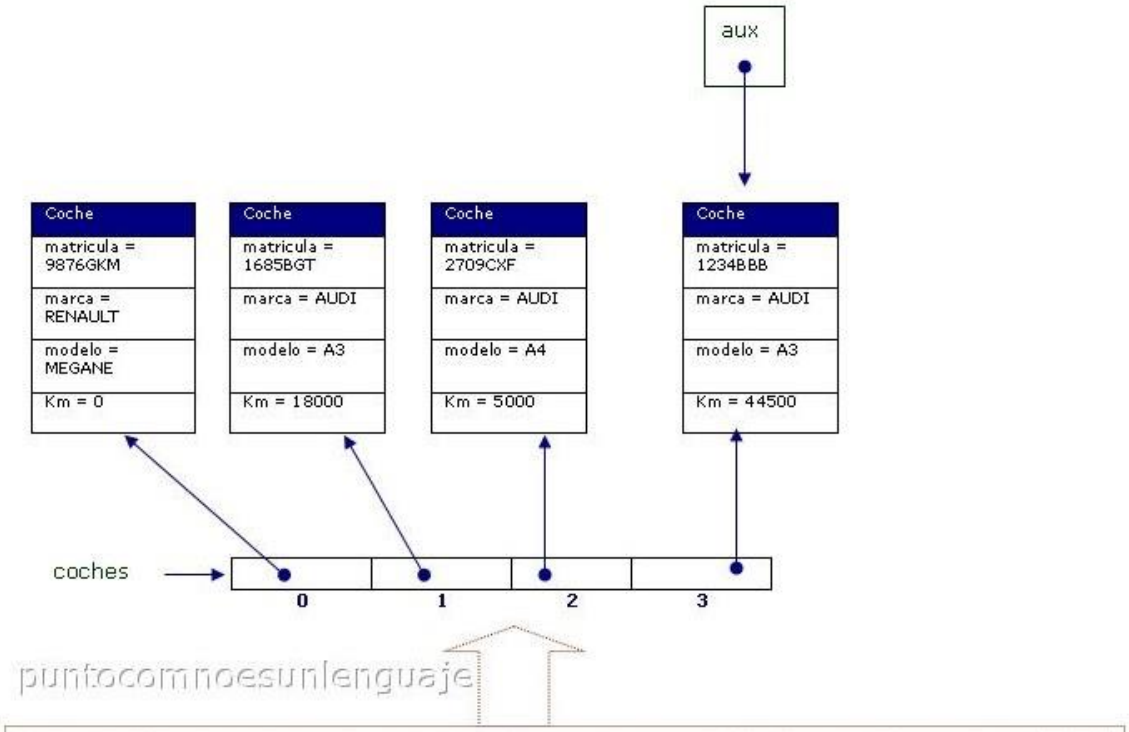
```
//Clase Coche
public class Coche {
    private String matricula;
    private String marca;
    private String modelo;
    private int Km;

    public int getKm() {
        return Km;
    }

    public void setKm(int Km) {
        this.Km = Km;
    }

    public String getMarca() {
        return marca;
    }

    public void setMarca(String marca) {
        this.marca = marca;
    }

    public String getMatricula() {
        return matricula;
    }

    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }

    public String getModelo() {
        return modelo;
    }

    public void setModelo(String modelo) {
        this.modelo = modelo;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append("\nMatrícula: ");
        sb.append(matricula);
        sb.append("\nMarca: ");
        sb.append(marca);
        sb.append("\nModelo: ");
        sb.append(modelo);
        sb.append("\nKm: ");
        sb.append(Km);
        return sb.toString();
    }
}
```

A continuación, creamos la clase principal del proyecto:

```
//Clase principal
public class Basico1 {

    static Scanner sc = new Scanner(System.in);

 //Se crea un ArrayList para guardar objetos de tipo Coche.
    static ArrayList<Coche> coches = new ArrayList();

    //método main
    public static void main(String[] args) {
        leerCoches();
        System.out.println("\nCoches introducidos:");
        mostrarCoches();
        mostrarPorMarca();
        mostrarPorKm();
        System.out.println("\nCoche con mayor número de Km: " +
mostrarMayorKm());
        System.out.println("\nCoches ordenados de menor a mayor número
de Km");
        mostrarOrdenadosPorKm();
    } //fin método main

    //Método para leer coches e introducirlos en el array
    public static void leerCoches(){

        //Declaración de variables para leer los datos de los coches
        String matricula;
        String marca;
        String modelo;
        int Km;

        //Variable auxiliar que contendrá la referencia a cada coche
nuevo.
        Coche aux;
        int i, N;

        //se pide por teclado el número de coches a leer
        do {
            System.out.print("Número de coches? ");
            N = sc.nextInt();
        } while (N < 0);
        sc.nextLine(); //limpiar el intro

        //lectura de N coches
        for (i = 1; i <= N; i++) {
            //leer datos de cada coche
            System.out.println("Coche " + i);
            System.out.print("Matrícula: ");
            matricula = sc.nextLine();
            System.out.print("Marca: ");
            marca = sc.nextLine();
            System.out.print("Modelo: ");
            modelo = sc.nextLine();
            System.out.print("Número de Kilómetros: ");
            Km = sc.nextInt();
            sc.nextLine(); //limpiar el intro

            aux = new Coche(); //Se crea un objeto Coche y se asigna
su referencia a aux
```

```
            //se asignan valores a los atributos del nuevo objeto
            aux.setMatricula(matricula);
            aux.setMarca(marca);
            aux.setModelo(modelo);
            aux.setKm(Km);

            //se añade el objeto al final del array
            coches.add(aux);
        }
    } //fin método leerCoches()
```

Podemos representar de forma gráfica el contenido del ArrayList según se van introduciendo los objetos:



Aspecto del ArrayList coches. Suponiendo que ya se habían insertado antes 3 objetos, añadir uno nuevo al final supone asignar la referencia del nuevo objeto (aux) al último elemento.

```
    //Método para mostrar todos los coches
    public static void mostrarCoches(){
        for(int i = 0; i< coches.size(); i++)
            System.out.println(coches.get(i));  //se invoca el método
toString de la clase Coche
    }
    //Método para mostrar todos los coches de una marca que se pide
por teclado
    public static void mostrarPorMarca(){
        String marca;
        System.out.print("Introduce marca: ");
        marca = sc.nextLine();
        System.out.println("Coches de la marca " + marca);
        for(int i = 0; i < coches.size(); i++){
            if(coches.get(i).getMarca().equalsIgnoreCase(marca))
                System.out.println(coches.get(i));
        }
    }
```

```
    //Método para mostrar todos los coches con un número de Km
inferior
    //al número de Km que se pide por teclado
    public static void mostrarPorKm(){
        int Km;
        System.out.print("Introduce número de kilómetros: ");
        Km = sc.nextInt();
        System.out.println("Coches con menos de " + Km + " Km");
        for(int i = 0; i < coches.size(); i++){
            if(coches.get(i).getKm() < Km)
                System.out.println(coches.get(i));
        }
    }
    //Método que devuelve el Coche con mayor número de Km
    public static Coche mostrarMayorKm(){
        Coche mayor = coches.get(0);
        for(int i = 0; i < coches.size(); i++){
            if(coches.get(i).getKm() > mayor.getKm())
                mayor = coches.get(i);
        }
        return mayor;
    }
    //Método que muestra los coches ordenados por número de Km de
menor a mayor
    public static void mostrarOrdenadosPorKm(){
        int i, j;
        Coche aux;
        for(i = 0; i < coches.size()-1; i++)
            for(j=0; j < coches.size()-i-1; j++)
                if(coches.get(j+1).getKm() < coches.get(j).getKm()){
                    aux = coches.get(j+1);
                    coches.set(j+1, coches.get(j));
                    coches.set(j, aux);
                }
        mostrarCoches();
    }
} //fin de la clase principal
```