Arrays unidimensionales y bidimensionales

ARRAYS UNIDIMENSIONALES

La sintaxis para declarar e inicializar un array será:

```
Tipo_de_variable[ ]   Nombre_del_array = new  Tipo_de_variable[dimensión];
```

También podemos alternativamente usar esta declaración:

```
Tipo_de_variable[ ] Nombre_del_array;

Nombre_del_array = new Tipo_de_variable[dimensión];
```

El tipo de variable puede ser cualquiera de los admitidos por Java y que ya hemos explicado. Ejemplos de declaración e inicialización con valores por defecto de arrays usando todos los tipos de variables Java, serían:

```
- byte[ ] edad = new byte[4];

- short[ ] edad = new short[4];

- int[ ] edad = new int[4];

- long[ ] edad = new long[4];

- float[ ] estatura = new float[3];

- double[ ] estatura = new double[3];

- boolean[ ] estado = new boolean[5];

- char[ ] sexo = new char[2];

- String[ ] nombre = new String[2];
```

Aclarar que los valores por defecto son los siguientes:

a) Para números el valor cero “0”.

b) Para cadenas y letras el valor vacío.

c) Para booleanos el valor false.

En caso de que queramos inicializarlos con valores propios, haremos esto:

Para números enteros

int[ ] edad = {45, 23, 11, 9}; //Array de 4 elementos

De la misma forma procederíamos para los otros tipos de enteros : byte, short, long.

Para números reales

double[ ] estatura = {1.73, 1.67, 1.56}; //Array de 3 elementos

De la misma forma procederíamos para el tipo float, pero teniendo en cuenta que los números deberán llevar al final la letra “f” o “F”. Por ejemplo 1.73f o 1.73F.

Para cadenas

String[ ] nombre = {"María", "Gerson"}; //Array de 2 elementos

Para caracteres

char[ ] sexo = {'m', 'f'}; //Array de 2 elementos

Para booleanos

boolean[ ] pepe = {true,false,false,false,true}; //Array de 5 elementos

Cuando creamos un array de nombre “a” y de dimensión “n” (int[ ] a = new int[n]) estamos creando n variables que son a[0], a[1], a[2], ..., a[n-1]. Los arrays se numeran desde el elemento cero, que sería el primer elemento, hasta el n-1 que sería el último elemento. Es decir, si tenemos un array de 5 elementos, el primer elemento sería el cero y el último elemento sería el 4. Esto conviene tenerlo en cuenta porque puede dar lugar a alguna confusión.

Ejemplo:
Se pueden inicializar en un bucle **for** :

```
for(int i=0; i<4; i++){
          numeros[i]=i*i+4;
}
```

Mejorando el ejemplo anterior, no necesitamos saber el número de elementos del array, su miembro dato *length* nos proporciona la dimensión del array:

```
for(int i=0; i<numeros.length; i++){
          numeros[i]=i*i+4;
```

```
}
```

Para recorrer los elementos de array *nombres* para leerlo sería igual:

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

ARRAYS UNIDIMENSIONALES DE OBJETOS

Por ejemplo de la clase Rectángulo que tiene dos atributos y un método calcularArea().

- Declarar y Crear el array

```
Rectangulo[] rectangulos=new Rectangulo[3];
```

- Inicializar los elementos del array IMPORTANTE: No solo hay que construir el array (new array) sino que también hay que construir cada uno de los objetos que va a almacenar ese array (new clase)

```
rectangulos[0]=new Rectangulo(10, 20);
rectangulos[1]=new Rectangulo(30, 40);
rectangulos[2]=new Rectangulo(50, 80);
```

O bien, en una sola línea

```
Rectangulo[] rectangulos={new Rectangulo(10, 20),
        new Rectangulo(30, 40), new Rectangulo(50, 80)};
```

- Usar el array

Para calcular y mostrar el área de los rectángulos escribimos

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

- Si queremos rellenar de manera dinámica los objetos rectangulos, sería de la siguiente manera:

```
   Rectangulo[] rectangulos=new Rectangulo[3];
for(int i=0; i<rectangulos.length; i++){
   rectangulos[i] = new Rectangulo();
  rectangulos[i].setAlto(5);//el valor puede venir dado x pantalla
  rectangulos[i].setAncho(7);
  System.out.println(rectangulos[i].calcularArea());
}
```

Arrays multidimensionales

Una matriz bidimensional puede tener varias filas, y en cada fila no tiene por qué haber el mismo número de elementos o columnas. Por ejemplo, podemos declarar e inicializar la siguiente matriz bidimensional

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

- La primera fila tiene cuatro elementos {1,2,3,4}

- La segunda fila tiene dos elementos {5,6}
- La tercera fila tiene seis elementos {7,8,9,10,11,12}
- La cuarta fila tiene un elemento {13}

Para mostrar los elementos de este array bidimensional escribimos el siguiente código

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

Como podemos apreciar, *matriz.length* nos proporciona el número de filas (cuatro), y *matriz[i].length*, nos proporciona el número de elementos en cada fila.

Mostramos los elementos de una fila separados por un tabulador usando la función *print*. Una vez completada una fila se pasa a la siguiente mediante *println.*

Los arrays bidimensionales nos permiten guardar los elementos de una matriz. Queremos crear y mostrar una matriz cuadrada unidad de dimensión 4. Recordaremos que una matriz unidad es aquella cuyos elementos son ceros excepto los de la diagonal principal *i==j*, que son unos. Mediante un doble bucle **for** recorremos los elementos de la matriz especificando su fila *i* y su columna *j*. En el siguiente programa

- Se crea una matriz cuadrada de dimensión cuatro
- Se inicializa los elementos de la matriz (matriz unidad)
- Se muestra la matriz una fila debajo de la otra separando los elementos de una fila por tabuladores.

```
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("");
    }
  }
}
```

Arrays Multidimensionales en Java

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.

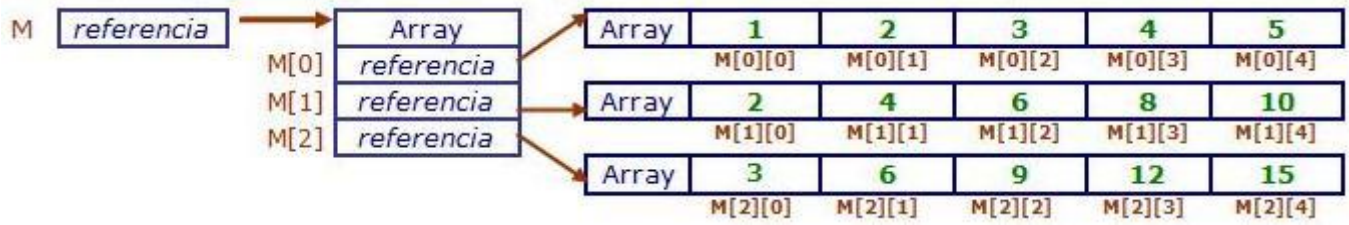
Una matriz necesita dos índices para acceder a sus elementos. Gráficamente podemos representar una matriz como una tabla de n filas y m columnas cuyos elementos son todos del mismo tipo.

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

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

Gráficamente podemos representar la disposición real en memoria del array anterior así:



La longitud del array M (M.length) es 3.

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.

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.

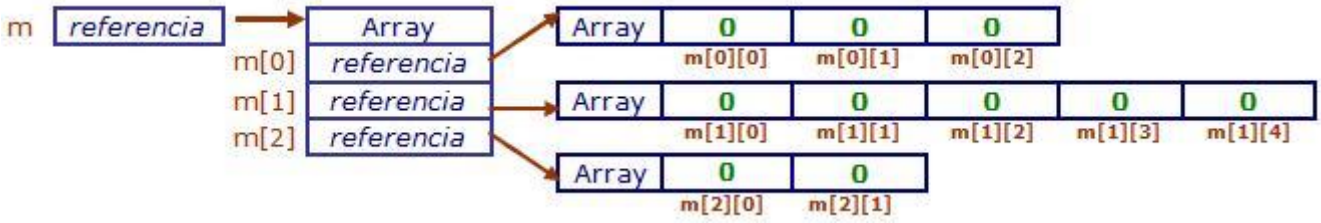
A cada fila se le puede asignar un número distinto de columnas:

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:



El método Sort de la clase Arrays

El método *sort* de la clase *Arrays*, que está en el paquete *java.util* que **se encargará de ordenar cualquier tipo de array unidimensional de tipo primitivo o String** que le pasemos como argumento.

Necesitaremos importar la clase **Arrays** de java.util, que es la que contiene el método estático **sort**(array) que nos permitirá hacer la ordenación del array.

Su uso se verá mejor con el siguiente ejemplo:

```
import java.util.Arrays;
public class Main {
    public static void main(String[] args) {
        //Array de String
        String[] nombres = {"Pepe", "Juan", "Alex",
            "Julian", "Francisco", "Luis"};

        //Ordena el array
        Arrays.sort(nombres);

        //Mostramos el array ya ordenado
        for (int i=0; i<nombres.length; i++) {
            System.out.print(nombres[i] + ", ");
        }
    }
}
```

A este método **se le puede pasar cualquier array** de cualquier tipo **básico**, ya probamos un tipo referencia con el tipo String, así que a continuación podrás ver el método sort en acción con un array de tipo primitivo **int**.

```
import java.util.Arrays;
public class Main {
    public static void main(String[] args) {
        //Array de String
        int[] numeros = {4, 2, 6, -3, 10, 11, 166, 1};

        //Ordena el array
        Arrays.sort(numeros);

        //Mostramos el array ya ordenado
```

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

videos ejemplo de uso de sort y toString
https://www.youtube.com/watch?v=L_KZ_f65GVs
https://www.youtube.com/watch?v=rOfOSR4T-1s

Cómo pasar argumentos al Main

El parámetro args es un array de Strings que debe aparecer obligatoriamente como argumento del método main en un programa Java.

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

Aunque se le suele dar el nombre args, no es obligatorio que este parámetro se llame así, podemos darle el nombre que queramos. Por ejemplo sería valido un método main escrito así:

```
public static void main(String[] argumentos){
    …
}
```

Para qué sirve el array String [] args de main?

Trabajando con un entorno de desarrollo a menudo nos olvidamos de que un programa java se puede ejecutar desde la línea de comandos del sistema operativo con la orden:

```
C:\> java nombrePrograma
```

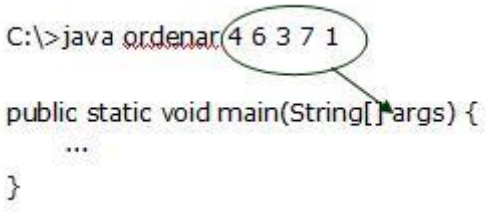
Además, mediante esta orden, podemos enviar valores al programa.

Por ejemplo, si tenemos un programa que se llama ***ordenar.java*** (es decir, el main está en una clase ordenar) que ordena 5 números enteros, pero en lugar de leerlos por teclado los números se le pasan al programa como argumentos, este programa lo ejecutaríamos así:

```
C:\> java ordenar 4 6 3 7 1
```

Los valores que se envían se deben escribir a continuación del nombre del programa y separados por un espacio en blanco.

El array **array args** que aparece como argumento del método **main** es el encargado de recoger y almacenar estos valores.



Como args es un **array de Strings** contendrá cada uno de estos valores como un String:

Contenido del Arrays args:

args[0]	"4"
args[1]	"6"
args[2]	"3"
args[3]	"7"
args[4]	"1"

La propiedad length del array args **(args.length)** contiene el **número de valores** enviados al programa.

Siempre que trabajemos con valores recibidos desde la línea de comandos debemos controlar el número de valores recibidos para evitar que se produzcan errores al procesar el array.

Ejemplo.

Programa llamado saludo.java que recibe desde la línea de comandos del sistema operativo un nombre de persona y lo muestra por pantalla.

```
public class saludo{
   public static void main(String[] args) {
      if (args.length > 1) { //si hay más de 1 parámetro
         System.out.println("Hay demasiados parámetros. Debe escribir: saludo nombrePersona");
      } else if (args.length == 0) { //si no hay parámetros
         System.out.println("Falta el nombre de la persona");
      } else {
         System.out.println("Buenos Días " + args[0]);
      }
   }
}
```

Para ejecutarlo se debe escribir en la línea de comandos:

>java saludo *nombreCualquiera*

Por ejemplo:

```
> java saludo Juan
```

La salida del programa sería en este caso:

```
Buenos Días Juan
```

Debemos tener en cuenta que **todos los valores que se envían a main desde la línea de comandos son de tipo String**. Si uno de los parámetros introducido en la línea de comandos es un número, el programa lo tratará como un String, por lo que para tratarlo como número y poder operar matemáticamente con él habrá que hacer la conversión al tipo numérico adecuado.

Ejemplo

El siguiente programa llamado ***longcir.java*** recibe desde la línea de comandos el valor del radio de una circunferencia y calcula y muestra la longitud de la circunferencia.

```
public class longcir {
   public static void main(String[] args) {
      double radio;
      if (args.length > 1) { //si hay más de 1 parámetro
         System.out.println("Hay demasiados parámetros. Debe escribir: longcir valorRadio");
      } else if (args.length == 0) { //si no hay parámetros
         System.out.println("Falta el valor del radio");
      } else {
         radio = Double.parseDouble(args[0]); //Se convierte el argumento a double
         System.out.println("Longitud de la circunferencia: " + 2 * Math.PI * radio);
      }
   }
}
```

Otro Ejemplo

Programa ***echo.java*** que muestra los argumentos introducidos en la línea de comandos.

```
public class echo {
   public static void main(String[] args) {
      int i;
      for (i = 0; i < args.length; i++) {
```

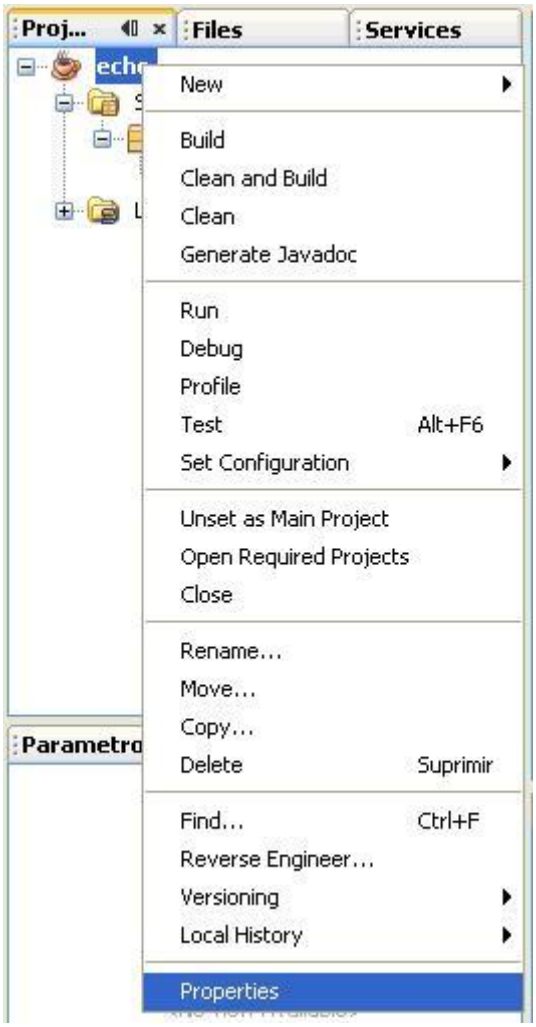
```
            System.out.print(args[i]+ " ");
        }
        System.out.println();
    }
}
```

Enviar valores a main en Netbeans y Eclipse

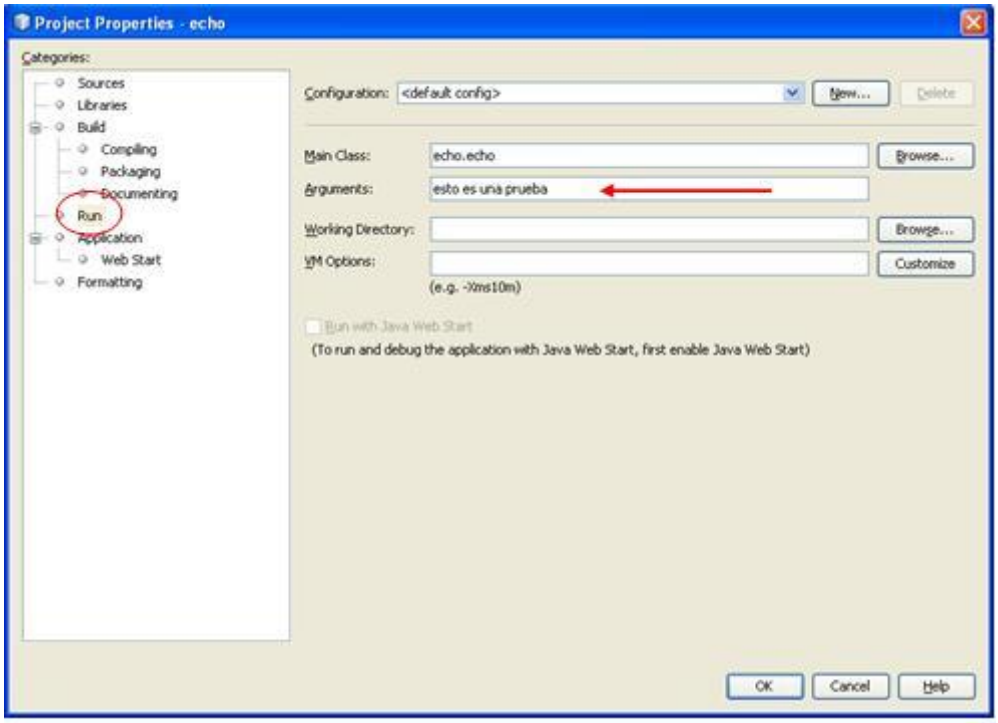
Si estamos utilizando un IDE podemos ejecutar programas que reciben valores desde la línea de comandos desde el propio IDE sin necesidad de ejecutarlo desde la línea de comandos.

En NetBeans se realiza de la siguiente forma (igual para Eclipse):

Botón derecho sobre el proyecto y seleccionamos *Properties*.



En la ventana que aparece pulsamos en *Run* y escribimos los parámetros en *Arguments*



Estos argumentos serán los que reciba main en el array args cuando ejecutemos el programa.

En Eclipse, a la hora de ejecutar con el botón ejecutar (flecha en circunferencia verde), si pinchamos en el desplegable del mismo, en la opción "Run configurations", dentro de la pestaña "Arguments", en el apartado "Program Arguments", podemos poner los argumentos necesarios, y pulsar el botón ejecutar de esa pantalla.

ArrayList (Arrays dinámicos)

ArrayList

La clase ArrayList en Java, es una clase que permite almacenar datos en memoria de forma similar a los Arrays, con la ventaja de que el número de elementos que almacena, lo hace de forma dinámica, es decir, que no es necesario declarar su tamaño como pasa con los Arrays. Los ArrayList nos permiten añadir, eliminar y modificar elementos (que pueden ser objetos o elementos primitivos) de forma trasparente para el programador.

Los principales métodos para trabajar con los ArrayList son los siguientes:

```
// Declaración de un ArrayList de "String". Puede ser de cualquier otro Elemento u Objeto (float, Boolean, Object, ...)
ArrayList<String> nombreArrayList = new ArrayList<String>();
// Añade el elemento al ArrayList
nombreArrayList.add("Elemento");
// Añade el elemento al ArrayList en la posición 'n'
nombreArrayList.add(n, "Elemento 2");
// Devuelve el numero de elementos del ArrayList
nombreArrayList.size();
// Devuelve el elemento que está en la posición '2' del ArrayList
nombreArrayList.get(2);
// Sustituye el elemento que está en la posición '2' del ArrayList por "Elemento sustituto"
nombreArrayList.set(2,"Elemento sustituto");
// Comprueba se existe el elemento ('Elemento') que se le pasa como parámetro
nombreArrayList.contains("Elemento");
// Devuelve la posición de la primera ocurrencia ('Elemento') en el ArrayList
nombreArrayList.indexOf("Elemento");
// Devuelve la posición de la última ocurrencia ('Elemento') en el ArrayList
nombreArrayList.lastIndexOf("Elemento");
// Borra el elemento de la posición '5' del ArrayList
nombreArrayList.remove(5);
```

```
// Borra la primera ocurrencia del 'Elemento' que se le pasa como parámetro.
nombreArrayList.remove("Elemento");
//Borra todos los elementos de ArrayList
nombreArrayList.clear();
// Devuelve True si el ArrayList esta vacío. Sino Devuelve False
nombreArrayList.isEmpty();
// Copiar un ArrayList
ArrayList arrayListCopia = (ArrayList) nombreArrayList.clone();
// Pasa el ArrayList a un Array
Object[] array = nombreArrayList.toArray();
```

Adicional:

Otra manera de trabajar con los ArrayList son los "Iteradores" (Iterator). Los Iteradores sirven para recorrer los ArrayList y poder trabajar con ellos. Los Iteradores solo tienen tres métodos que son el *"hasNext()"* para comprobar que siguen quedando elementos en el iterador, el *"next()"* para que nos de el siguiente elemento del iterador; y el *"remove()"* que sirve para eliminar el elemento del Iterador.

Bueno, si esto no te ha quedado muy claro, pasamos a poner el primer ejemplo. En el siguiente fragmento de código, declaramos un ArrayList de Strings y lo rellenamos con 10 Strings (Elemento i). Esto lo hacemos con el método *"add()"*. Después añadimos un nuevo elemento al ArrayList en la posición '2' (con el metodo *"add(posición,elemento)"*) que le llamaremos "Elemento 3" y posteriormente imprimiremos el contenido del ArrayList, recorriendolo con un Iterador. El fragmento de este código es el siguiente:

```
// Declaración el ArrayList
ArrayList<String> nombreArrayList = new ArrayList<String>();

// Añadimos 10 Elementos en el ArrayList
for (int i=1; i<=10; i++){
        nombreArrayList.add("Elemento "+i);
}

// Añadimos un nuevo elemento al ArrayList en la posición 2
nombreArrayList.add(2, "Elemento 3");

// Declaramos el Iterador e imprimimos los Elementos del ArrayList
Iterator<String> nombreIterator = nombreArrayList.iterator();
while(nombreIterator.hasNext()){
        String elemento = nombreIterator.next();
        System.out.print(elemento+" / ");
}
```

Ejecutando este código obtenemos por pantalla lo siguiente:

```
Elemento 1 / Elemento 2 / Elemento 3 / Elemento 3 / Elemento 4 / Elemento 5 / Elemento 6 / Elemento 7 / Elemento 8 / Elemento 9 / Elemento 10 /
```

Como se observa en el resultado tenemos repetido el elemento *"Elemento 3"* dos veces y esto lo hemos puesto a proposito para mostrar el siguiente ejemplo. Ahora para seguir trabajando con los ArrayList, lo que vamos a hacer es mostrar el número de elementos que tiene el ArrayList y después eliminaremos el primer elemento del ArrayList y los elementos del ArrayList que sean iguales a *"Elemento 3"*, que por eso lo hemos puesto repetido. El *"Elemento 3"* lo eliminaremos con el metodo *"remove()"* del iterador. A continuación mostramos el código que realiza lo descrito:

```
// Recordar que previamente ya hemos declarado el ArrayList y el Iterator de la siguiente
forma:
// ArrayList<String> nombreArrayList = new ArrayList<String>();
// Iterator<String> nombreIterator = nombreArrayList.iterator();

// Obtenemos el numero de elementos del ArrayList
int numElementos = nombreArrayList.size();
System.out.println("nnEl ArrayList tiene "+numElementos+" elementos");

// Eliminamos el primer elemento del ArrayList, es decir el que ocupa la posición '0'
System.out.println("n... Eliminamos el primer elemento del ArrayList
("+nombreArrayList.get(0)+")...");
nombreArrayList.remove(0);

// Eliminamos los elementos de ArrayList que sean iguales a "Elemento 3"
System.out.println("n... Eliminamos los elementos de ArrayList que sean iguales a
"Elemento 3" ...");
nombreIterator = nombreArrayList.iterator();
while(nombreIterator.hasNext()){
          String elemento = nombreIterator.next();
          if(elemento.equals("Elemento 3"))
                    nombreIterator.remove();      // Eliminamos el Elemento que hemos
obtenido del Iterator
}

// Imprimimos el ArrayList despues de eliminar los elementos iguales a "Elemento 3"
System.out.println("nImprimimos los elementos del ArrayList tras realizar las
eliminaciones: ");
nombreIterator = nombreArrayList.iterator();
while(nombreIterator.hasNext()){
          String elemento = nombreIterator.next();
          System.out.print(elemento+" / ");
}

// Mostramos el numero de elementos que tiene el ArrayList tras las eliminaciones:
numElementos = nombreArrayList.size();
System.out.println("nNumero de elementos del ArrayList tras las eliminaciones =
"+numElementos);
```

Como salida a este código tenemos lo siguiente:

```
El ArrayList tiene 11 elementos

... Eliminamos el primer elemento del ArrayList (Elemento 1)...

... Eliminamos los elementos de ArrayList que sean iguales a "Elemento 3" ...

Imprimimos los elementos del ArrayList tras realizar las eliminaciones:
Elemento 2 / Elemento 4 / Elemento 5 / Elemento 6 / Elemento 7 / Elemento 8 / Elemento 9 /
Elemento 10 /

Número de elementos del ArrayList tras las eliminaciones = 8
```

Si os fijais, hemos eliminado 3 elementos del ArrayList de dos formas distintas, preguntando por la posición que ocupa un elemento en el ArrayList y preguntando por el contenido de algún elemento del ArrayList. Como se observa es muy importante saber manejar los Iteradores ya que con ellos podemos tratar los elementos del ArrayList.

For extendido (each) en java

EL FOR EXTENDIDO O BUCLES FOR EACH EN JAVA

En las últimas versiones de Java se introdujo una nueva forma de uso del for, a la que se denomina “for extendido” o “for each”. Esta forma de uso del for, que ya existía en otros lenguajes, facilita el recorrido de colecciones de objetos existentes sin necesidad de definir el número de elementos a recorrer. Por ejemplo, arrays unidimensionales, bidimensionales, ArrayList, etc... La sintaxis que se emplea es:

```
for ( TipoColeccion nombreVariable : nombreDeLaColeccion ) {

                Instrucciones

}
```

La interpretación que podemos hacer de la sintaxis del for extendido es: “Para cada elemento del tipo TipoColeccion que se encuentre dentro de la colección nombreDeLaColección ejecuta las instrucciones que se indican”. La variable local-temporal del ciclo almacena en cada paso el objeto que se visita y sólo existe durante la ejecución del ciclo y desaparece después. Debe ser del mismo tipo que los elementos a recorrer. Ejemplo

```
//Ejemplo

public void listarTodosLosNombres () {

        String[] listaDeNombres = {"Pepe", "Paco", "Marisol"};

        for (String nombre: listaDeNombres) {

        System.out.println (nombre); //Muestra cada uno de los nombres dentro de
listaDeNombres

    }

  }
```

El for extendido tiene algunas ventajas y algunos inconvenientes. No se debe usar siempre. Su uso no es obligatorio, de hecho, como hemos indicado, en versiones anteriores ni siquiera existía en el lenguaje.

EJEMPLO FOR EACH PARA UN ARRAY BIDIMENSIONAL:

```
public static void main(String[] args)
{
   int[][] array1 = {{1, 2, 3, 4}, {5, 6, 7, 8}};
   for(int[] arr2: array1)  //recorre las filas
   {
       for(int val: arr2)    //recorre las columnas
           System.out.print(val);
   }
}
```