

Unidad 5

Arrays



Contenidos

- **Conceptos básicos**
 - **Crear Array**
 - **Acceso y modificación**
 - **Tipos de Array**
 - **Referencia**
 - **Longitud de un Array**
 - **Recorrer un Array**
 - **Copiar un Array**
 - **Comparar dos Array**
 - **Comparar elementos de un Array**
 - **Array en funciones**
-
- **Métodos de Arrays**
 - Arrays (clase) java.util.Arrays
 - Generar String
 - Rellenar
 - Copiar
 - Copiar rango
 - Copia de origen a destino
 - Igual
 - Ordenar
 - Ordenar rango
 - Búsqueda binaria
 - Otros métodos para arrays multidimensionales

Contenidos

- **Arrays Multidimensionales**
 - Recorrer un array
 - Recorrer un array 3D
 - Recorrer un array 3D usando propiedad longitud
 - Array longitud de longitud variable
 - Arrays de diferentes tipos

Conceptos básicos

Variables escalares

Las variables escalares solo pueden tener un valor cada vez.

```
double nota = 4,25;  
nota = 8,75;
```

Una vez se sobreescribe el valor de la nota, desaparece el anterior. No se pueden tener dos notas simultáneamente. Habría que crear dos variables:

```
double nota1 = 4.25;  
double nota2 = 8.75;
```

¿qué pasa si necesitamos almacenar 100 notas? ¿creamos 100 variables?

Conceptos básicos

Variables vectoriales (array)

Representan una secuencia lineal y finita de elementos del mismo tipo.

nota =	4.25	8.75		
--------	------	------	--	--

(tamaño: 4)

Estos elementos pueden ser de tipo primitivo o pertenecientes a otras clases.

Los arrays son objetos con características propias. Además del conjunto de elementos del mismo tipo, un objeto de tipo array almacena en memoria una constante numérica entera que representa el número de elementos o tamaño del array.

Conceptos básicos

¿Para qué sirven?

Se usan para almacenar datos que están relacionados y que queremos procesar en su conjunto.

Por ejemplo:

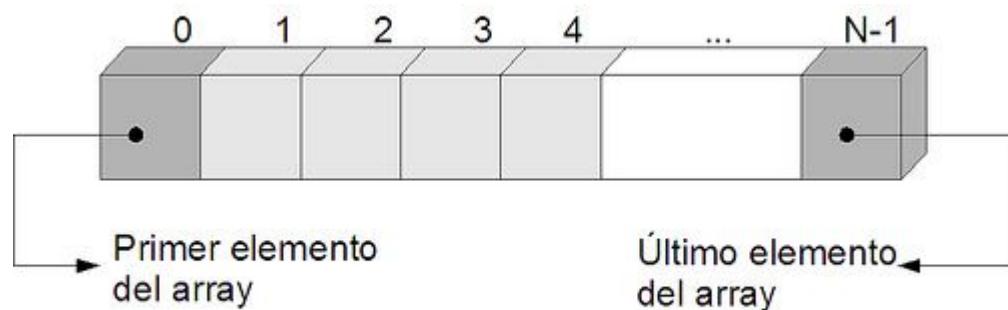
- El peso de un niño en sus 36 primeros meses.
- Los nombres de los 30 alumnos de clase.
- Los aciertos o fallos en un test de 50 preguntas.
- La ficha que está posicionada en cada celda de un tablero de ajedrez.

Conceptos básicos

Índice

Se usa para distinguir entre cada uno de los elementos o componentes que constituyen una tabla.

Un array de N elementos, tiene un índice que va del 0 al $N-1$, para así tener identificados cada elemento.



Conceptos básicos

Array Unidimensional

Índice	0	1	2	3	4	5	6	7	8	9
Datos	7	4	10	15	1	20	18	4	27	17

Tipo de dato: Entero

Tamaño: 10

A los arrays se les asigna un nombre (identificador) como a otra variable más.

La forma de acceder a cada uno de estos valores es indicando el nombre de identificador y el índice de la forma:

nombre[índice].

Por ejemplo, si el array anterior se llama “temperaturas”, se accedería a una posición del mismo de la siguiente manera:

temperaturas[3] sería el valor **15**.

Conceptos básicos

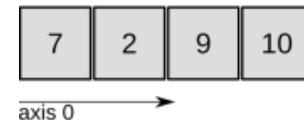
Los arrays pueden ser de 1 dimensión (se pueden llamar lista).

De 2 dimensiones (se pueden llamar tablas).

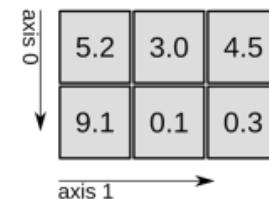
O de más dimensiones

- Tridimensionales
- Multidimensionales

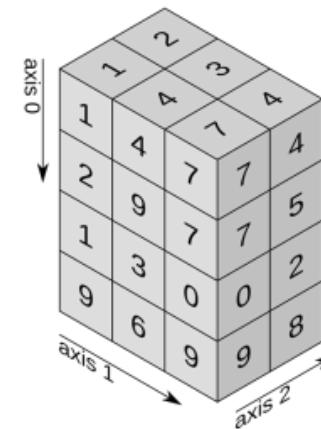
1D array



2D array



3D array



Crear arrays

Hay que tener en cuenta:

- Tipo de datos de los elementos que queremos almacenar.
 - Todos deben ser del mismo tipo.
- Cantidad de elementos.
- Declarar la variable.
 - Como otra variable cualquiera.
- Crear el array.

Crear arrays

1. Declaración (existen dos formas equivalentes):

```
tipo[] identificador □ int[] edad;
```

```
tipo identificador[] □ int edad[];
```

2. Asignación (para crear un objeto Array se usa new):

```
identificador = new tipo[longitud]
```

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

Ejemplo.

Queremos crear una variable notas para almacenar 5 números enteros.

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

También se puede crear un array al mismo tiempo que se rellena.

```
int notas[] = { 2, 5, 7, 9, 8 };
```

Acceso y modificación

Acceso a los valores

Puedes acceder a los valores con ***identificador[índice]***

```
int notas[] = {2, 5, 7, 9, 8};  
notas[0];    // vale 2.  
notas[1];    // vale 5.  
notas[2];    // vale 7.  
notas[3];    // vale 9.  
notas[4];    // vale 8.
```

Podemos usarlos como otra variable más. Por ejemplo:

```
System.out.println(notas[3]);  
int maximo = notas[4];  
int media = (notas[0]+notas[1]+notas[2]+notas[3]+notas[4]) / 5;
```

Acceso y modificación

Asignar valores

Como podemos usarlos como una variable más, también le podemos asignar valores usando el índice:

identificador[índice] = valor;

Por ejemplo:

```
int[] notas = new int[5];
notas[0] = 2;
notas[1] = 2+3;
notas[2] = notas[1]+2;
notas[3] = 9;
notas[4] = 8;
```

Acceso y modificación

Fuera de rango

Si intentamos acceder a un índice que no está en el rango de 0 a Longitud N-1, nos da un error en tiempo de ejecución.

Por ejemplo:

```
int[] notas = {1,2,3,4,5};  
System.out.println(notas[7]);
```

No da problemas en compilación. Por tanto, el entorno de desarrollo no va a mostrar ningún mensaje de error, pero al ejecutar sale el siguiente error:

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 7 out of bounds for length 5

Por tanto, cualquier valor entre 0 y 4 es correcto pero menor a 0 o igual o mayor que 5 da error.

Tipos de array

Arrays de números:

```
byte byteArray[];  
short shortArray[];  
int intArray[];  
long longArray[];  
float floatArray[];  
double doubleArray[];
```

Arrays de booleans:

```
boolean booleanArray[];
```

Arrays de Caracteres o cadena de caracteres:

```
char charArray[];  
String stringArray[];
```

También de Objetos o clases propias:

```
Object objectArray[];  
MiClass miClaseArray[];
```

Referencia

Cuando se declara una variable escalable, se reserva en memoria un espacio para almacenar su valor (boolean, byte, int, float, ...). El espacio depende del tipo de la variable. Por ejemplo un short ocupa menos que un float.

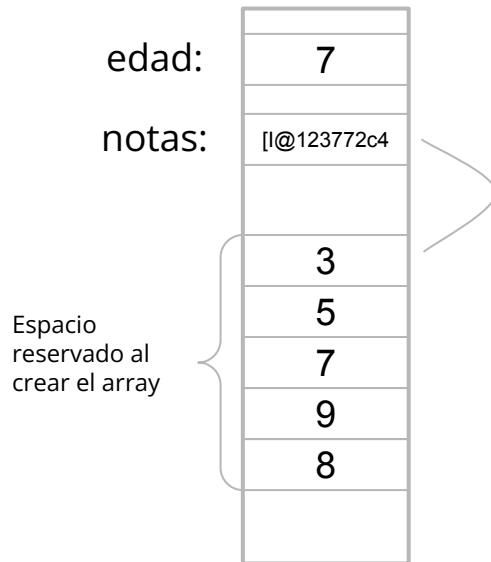
```
int edad = 7;
```

En cambio, cuando se declara un array se reserva en memoria principal un espacio de memoria que va a almacenar una dirección a otra posición de memoria que es donde se creará el array.

```
int notas[];
```

Cuando se crea el array, se reserva el espacio de memoria que corresponda según el tipo de los elementos del array y la longitud del mismo (tamaño un elemento * Número de elementos). La variable del array toma el valor de la dirección de memoria donde se ha creado el array.

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



Referencia

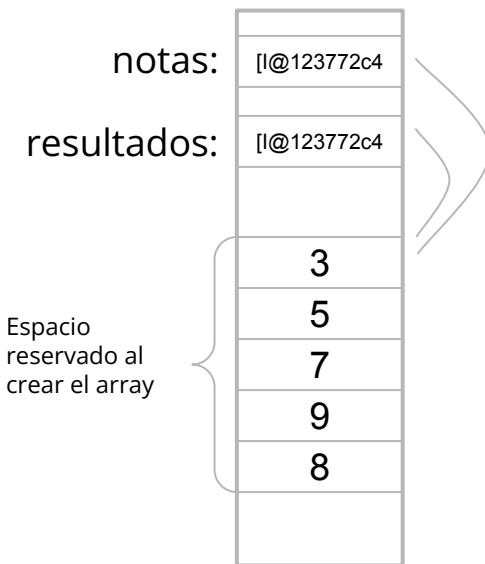
Si declaramos dos variables de tipo array.

```
int notas[];  
int resultados[];
```

Podemos crear un array y asignar el valor de este a otro array:

```
notas = new int[5];  
  
resultados = notas;
```

Realmente copiamos la referencia al array y no el propio array. Es decir, tenemos dos variables apuntando al mismo array.



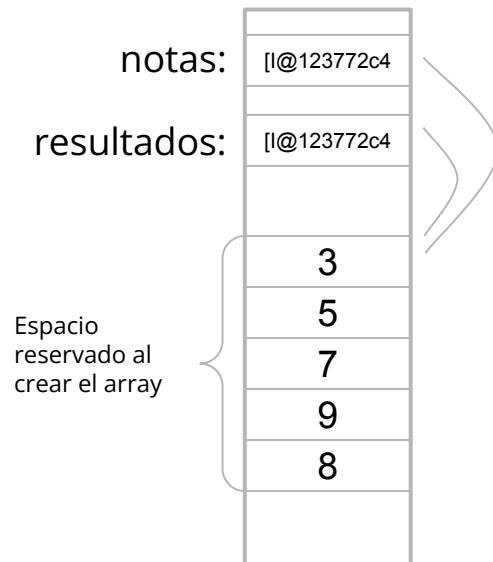
Referencia

Por tanto, para el siguiente ejemplo:

```
int notas[];
int resultados[];
notas = new int[5];
resultados = notas;
```

Los dos vectores apuntan al mismo array y si se cambia los valores en uno, también se cambian en el otro.

```
System.out.println(notas[0]);          // = 3
System.out.println(resultados[0]);      // = 3
resultados[0] = 6;
System.out.println( notas[0] );          // = 6
System.out.println( resultados[0] );     // = 6
```



Referencia

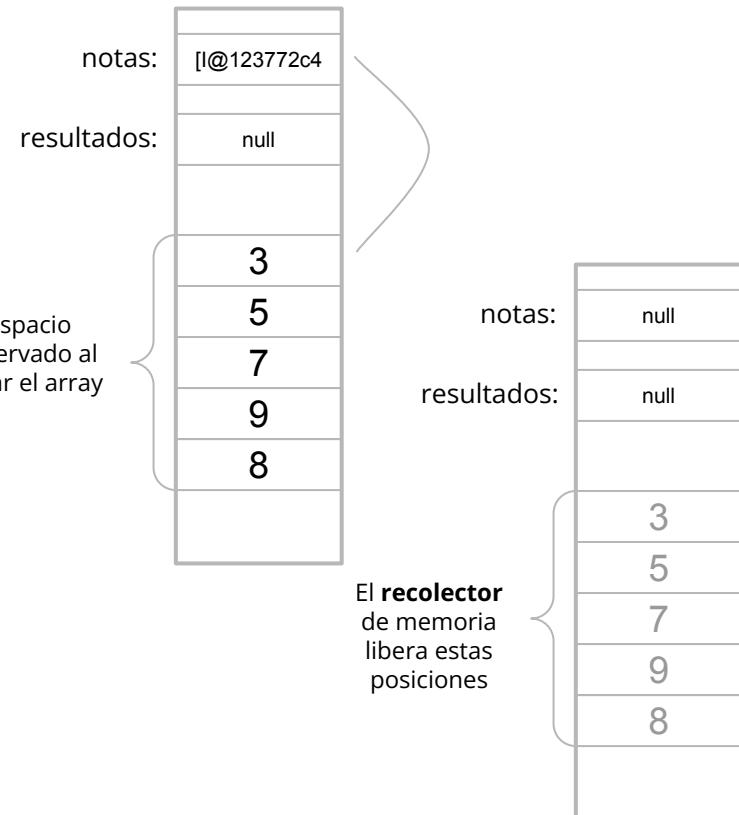
A null

Cuando queremos que una variable se inicie sin apuntar a ninguna posición de memoria, o si se quiere que deje de apuntar a un array, en ese caso se puede indicar que apunta a **null**, que quiere decir que apunta a ningún sitio.

```
int resultados[] = null;  
int notas[] = new int[5];  
// notas apunta a un array.
```

```
notas = null;  
// Ahora notas apunta a ningún sitio.
```

Cuando no hay ninguna variable que apunte a las posiciones de memoria de un array, este array es liberado por el **recolector de basura**.



Longitud de un Array

Propiedad **length**

Cuando se crea un array se ha de indicar además del tipo de los elementos, la cantidad elementos que queremos almacenar. Esto se llama longitud del array.

Se puede consultar la longitud mediante la propiedad **length**.

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

```
int longitud = notas.length;
```

```
System.out.println(longitud); // Devuelve 5.
```

Recorrer un Array

Para recorrer un array lo mejor es usar bucles.

Si queremos pasar por todos los elementos de un array de longitud N, podemos hacer un recorrido desde el elemento de la posición 0 al N-1.

Por ejemplo, podemos usar un bucle for donde el contador del bucle es el índice del array.

```
int notas[] = {3, 7, 2, 9, 6};  
for (int i=0; i<notas.length; i++) {  
    System.out.println( notas[i] );  
}
```

Ejemplo

Imagina que tengo las notas del alumnado obtenidas en una prueba almacenadas en un array de enteros.

La prueba tiene una puntuación de 0 a 5. Pero el alumnado está acostumbrado a una puntuación de 0 a 10. Por tanto, voy a modificar todas las notas para multiplicar todos los números por 2.

El programa puede quedar de la siguiente forma:

```
int notas[] = {2, 4, 5, 3, 5, 0, 3, 4, 1, 5, 4, 5, 3};  
for (int i=0; i<notas.length; i++) {  
    notas[i] = notas[i] * 2;  
}
```

Ahora notas será: {4, 8, 10, 6, 10, 0, 6, 8, 2, 10, 8, 10, 6}

Copiar un array

Cómo se podría hacer una copia de un array en lugar de copiar la referencia.

El programa puede quedar de la siguiente forma:

```
int notas[] = {3, 5, 7, 9, 8};  
int resultados[] = new int[notas.length];  
for (int i=0; i<notas.length; i++) {  
    resultados[i] = notas[i];  
}
```

Ahora hay dos arrays distintos con los mismos elementos.

Comparar dos arrays

Cuando se compara (==) dos arrays, realmente estamos comparando las referencias, es decir, si ambas variables apuntan al mismo array.

```
int notas[] = {3, 5, 7, 9, 8};  
int resultados[] = new int[notas.length];  
for (int i=0; i<notas.length; i++) {  
    resultados[i] = notas[i];  
}
```

```
notas == resultados // false
```

Aunque tengan el mismo tamaño y los mismos valores, no son el mismo array.

```
int notas[] = {3, 5, 7, 9, 8};  
int resultados[] = notas;
```

```
notas == resultados // true
```

Ambas variables apuntan al mismo array.

Comparar elementos de un array

Para comparar que dos arrays tienen los mismos elementos, habrá que comparar que tienen la misma longitud y que tienen los mismos elementos en las mismas posiciones.

Vamos a crear una variable (`iguales`) en la que suponemos que son iguales y tratamos de demostrar lo contrario.

```
int notas[] = {3, 5, 7, 9, 8};  
int resultados[] = {3, 5, 7, 9, 8};  
boolean iguales = true;  
if (notas.length != resultados.length) iguales = false;  
for (int i=0; iguales && i<notas.length; i++) {  
    if (notas[i] != resultados[i]) iguales = false;  
}  
System.out.println(iguales);
```

Arrays en funciones

Se pueden declarar arrays dentro de las funciones al igual que otras variables.

También se pueden pasar arrays como parámetros en la llamada de una función y recibirlas como resultado de la ejecución de la función.

```
static int[] duplicar(int[] lista)
```

Pero hay que tener en cuenta que realmente lo que se pasa como parámetro no es una copia del array sino una copia de la referencia al array. Es decir, los elementos del array serán los mismos dentro y fuera de la función.

Arrays en funciones

Por tanto, debemos tener cuidado de que si se modifican los elementos del array dentro de la función, se está modificando el array original.

Por ejemplo, en el ejemplo, el array **notas** se pasa como parámetro (aunque dentro se llame **lista**), se modifica dentro de la función y se devuelve.

El array **resultados** recoge ese array, pero no se crea un array nuevo, sigue siendo el mismo. Simplemente hay otra variable que apunta a él.

```
static int[] duplicar(int[] lista) {  
    for (int i=0; i<lista.length; i++) {  
        lista[i] = lista[i] * 2;  
    }  
    return lista;  
}  
  
public static void main(String[] args) {  
    // TODO code application logic here  
    int notas[] = {3, 5, 7, 9, 8};  
    int resultados[] = duplicar(notas);  
}
```

Las variables **notas** y **resultados** apuntan al mismo array que ahora tiene los valores:

{4, 10, 14, 18, 16}

Arrays (clase) java.util.Arrays

```
import java.util.Arrays;
```

Esta clase proporciona múltiples métodos para manipular Arrays

Los métodos están sobrecargados para poder ser aplicados sobre múltiples tipos de datos

Tipos de datos primitivos (enteros, reales, caracteres y booleanos) y objetos.

Generar String

static String toString(T[] datos)

Genera una cadena para impresión.

Ejemplo:

```
int[] lista = {1, 2, 3, 4, 5};  
System.out.println(Arrays.toString(lista));  
// muestra por consola [1, 2, 3, 4, 5];
```

Rellenar

```
static void fill(T[] array, T valor)
```

Llena el array con el valor indicado. Por tanto, todos los elementos serán iguales.

Ejemplo:

```
int[] unos = new int[10];  
Arrays.fill(unos, 1);  
// el array unos será [1, 1, 1, 1, 1, 1, 1, 1, 1, 1];
```

Copiar

static T[] copyOf(T[] datos, int n)

Genera una copia de los datos. Si n es mayor que datos.length, rellena con null o valor por defecto del tipo. Si n es menor que datos.length, se ignora el exceso (es decir, trunca).

Ejemplo:

```
int[] lista = {1, 2, 3, 4, 5};  
int[] listaCorta = Arrays.copyOf(lista, lista.length-1);  
// el array listaCorta será [1, 2, 3, 4];  
int[] listaLarga = Arrays.copyOf(lista, lista.length+1);  
// el array listaLarga será [1, 2, 3, 4, 5, 0];
```

Copiar rango

```
static T[] copyOfRange(T[] origen, int posOrigen, int hasta)
```

Copia un segmento de los datos desde una posición (incluido) hasta otra (sin incluirla)..

Desde el elemento situado en la posición indicado en **desde**.

Hasta el elemento anterior al situado en la posición indicado en **hasta**.

Ejemplo:

```
int[] lista = {1, 2, 3, 4, 5, 6, 7, 8, 9};  
int[] copia = Arrays.copyOfRange(lista, 3, 6);  
System.out.println(Arrays.toString(copia));  
// copia contiene [4, 5, 6]
```

Copia de origen a destino

Esta método no es de Arrays sino de **System**.

```
static void arraycopy(T[] origen, int posOrigen, T[] destino, int postDestino,  
                      int longitud)
```

Copia el número de elementos indicado en longitud desde el array de origen al array de destino.

En el array de destino empiezan a copiarse en la posición indicada en **postDestino**, y en el array de origen comienzan a leerse en

posOrigen. Por tanto, puede haber un desplazamiento si ambos números son distintos.

Ejemplo:

```
int[] lista = {1, 2, 3, 4, 5, 6, 7, 8, 9};  
int[] copia = new int[9];  
System.arraycopy(lista, 2, copia, 4, 3);  
System.out.println(Arrays.toString(copia));  
// copia contiene [0, 0, 0, 0, 3, 4, 5, 0, 0]
```

Igual

```
static boolean equals(T[] a1, T[] a2)
```

Chequea si los arrays son idénticos, comprobando valores primitivos (==) y referencias (con equals).

Ejemplo:

```
int[] lista1 = {1, 2, 3, 4, 5};  
int[] lista2 = {1, 2, 3, 4, 5};  
int[] lista3 = {1, 2, 3, 5, 4};  
Arrays.equals(lista1, lista2); // true  
Arrays.equals(lista1, lista3); // false
```

Ordenar

static void sort(T[] datos)

Ordena el array.

Ejemplo:

```
int[] valores = {4, 7, 1, 9, 2, 8, 3, 6, 5};  
Arrays.sort(valores);  
System.out.println(Arrays.toString(valores));  
// el array valores será [1, 2, 3, 4, 5, 6, 7, 8, 9];
```

Ordenar rango

```
static void sort(T[] datos, int desde, int hasta)
```

Ordena el array entre las posiciones indicadas.

Ejemplo:

```
int[] valores = {4, 7, 1, 9, 2, 8, 3, 6, 5};  
Arrays.sort(valores, 3, 7);  
System.out.println(Arrays.toString(valores));  
// el array valores será [4, 7, 1, 2, 3, 8, 9, 6, 5];
```

Búsqueda binaria

```
static int binarySearch(T[] datos, T clave)
```

Busca en qué posición del array datos se encuentra la clave dada. El array debe estar ordenado.

Devuelve la posición en la que se encuentre el elemento (clave).

Si no encuentra el elemento, devuelve un valor negativo. Ese número si se pone a positivo y se le resta 1, indica en qué lugar podríamos insertar dicho valor.

Ejemplo:

```
int[] valores = {1, 2, 3, 4, 6, 7, 8, 9};  
int cuatro = Arrays.binarySearch(valores, 4);           // devuelve 3  
int dos = Arrays.binarySearch(valores, 2);             // devuelve 1  
int once = Arrays.binarySearch(valores, 5);            // devuelve -5
```

Otros métodos para arrays multidimensionales

static boolean deepEquals(Object[] a1, Object[] a2)

Chequea si los arrays son idénticos, comprobando valores primitivos (==) y referencias (con equals). Además, si el array es multidimensional, profundiza en las sucesivas dimensiones.

static String deepToString(T[] datos)

Genera una cadena para impresión, incluso si se trata de un array multidimensional.

Arrays Multidimensionales

Nos referimos a arrays multidimensionales cuando tienen más de una dimensión.

Mientras que los arrays de una dimensión representan listas o vectores, los de dos dimensiones representan tablas o matrices. También pueden tener más de 2 dimensiones y representar matrices tridimensionales o más.

Se definen de la siguiente forma:

Tipo[][]... [] nombre = new Tipo[dim1][dim2]...[dimN];

Donde Tipo es el tipo de los elementos del array y se debe poner unos corchetes [] por cada dimensión. Tanto en la declaración como en la creación. En la creación se debe indicar el tamaño de cada dimensión.

Ejemplo de matriz de enteros

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

//Se genera una matriz de 3x4.

Se puede acceder a cada elemento de la tabla mediante:

a[i][j];

Donde i es un valor de la primera dimensión [3] y podrá tomar valores de 0 a 2.

Y j es un valor de la segunda dimensión [4] y podrá tomar valores de 0 a 3.

	Column 1	Column 2	Column 3	Column 4
Row 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 3	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Recorrer un array

Si queremos recorrer un array de múltiples direcciones, hay que crear tantos bucles como dimensiones tiene. Los bucles están anidados unos a otros.

```
String[][] a = new String[3][4];
for (int i=0; i<3; i++) {
    for (int j=0; j<4; j++) {
        a[i][j] = i+"-"+j;
    }
}
System.out.println(Arrays.deepToString(a));
```

Muestra el array de la forma:

```
[ [0-0, 0-1, 0-2, 0-3],
  [1-0, 1-1, 1-2, 1-3],
  [2-0, 2-1, 2-2, 2-3] ]
```

Recorrer un array 3D

Ejemplo de un array de 3 dimensiones.

```
String[][][] a = new String[2][3][4];
for (int i=0; i<2; i++) {
    for (int j=0; j<3; j++) {
        for (int k=0; k<4; k++) {
            a[i][j][k] = i+"-"+j+"-"+k;
        }
    }
}
System.out.println(Arrays.deepToString(a));
```

Muestra el array de la forma:

```
[[[0-0-0, 0-0-1, 0-0-2, 0-0-3], [0-1-0, 0-1-1, 0-1-2, 0-1-3], [0-2-0, 0-2-1, 0-2-2, 0-2-3]],  
 [[1-0-0, 1-0-1, 1-0-2, 1-0-3], [1-1-0, 1-1-1, 1-1-2, 1-1-3], [1-2-0, 1-2-1, 1-2-2, 1-2-3]]]
```

Recorrer un array 3D usando propiedad longitud

Lo mejor es no usar una longitud fija y siempre usar la propiedad length. Pero hay que tener cuidado en los arrays multidimensionales. Hay que preguntar por la longitud en cada dimensión.

```
String[][][] a = new String[2][3][4];
for (int i = 0; i < a.length; i++) {
    for (int j = 0; j < a[i].length; j++) {
        for (int k = 0; k < a[i][j].length; k++) {
            a[i][j][k] = i + "-" + j + "-" + k;
        }
    }
}
System.out.println(Arrays.deepToString(a));
```

Array multidimensional de longitud variable

Se pueden crear arrays de longitud variable mediante asignación directa:

```
int a[][] = {  
    {1, 2, 3},  
    {11, 12},  
    {},  
    {31, 32, 33, 34 ,35, 36, 37}  
};  
System.out.println(Arrays.deepToString(a));
```

Se crea el array a con el contenido:

```
[[1, 2, 3], [11, 12], [], [31, 32, 33, 34, 35, 36, 37]]
```

Donde la primera dimensión es de 4 elementos, pero la segunda es de 3, 2, 0 y 7, es decir, no es fija para la segunda dimensión.

Si queremos leer a[3][5] devuelve el valor 36 pero si queremos leer a[2][5] dará un error de fuera de rango.

En estos casos siempre hay que recorrerlos usando la propiedad length.

Arrays de diferentes tipos

NO ES POSIBLE

Alternativa:

Pero se pueden usar varios arrays de diferentes tipos donde relacionamos un elemento con otro mediante el índice.

Todos tienen la misma longitud

Es decir, en el índice 0 está el nombre, la edad y las notas del primer alumno, en el índice 1, están almacenados los datos del segundo alumno, y así sucesivamente.

Más adelante veremos cómo meter objetos en los arrays.

Arrays de diferentes tipos

Por ejemplo: Alumnos, edades y notas.

Se pueden almacenar los datos de la forma:

```
String[] nombres = { "Marta",
    "Yaiza", "Anibal", "Rosendo",
    "Susana" };

int[] edades = { 18, 21, 19, 28, 20};

int[][] notas = {
    { 7, 8, 8},
    { 6, 4, 7},
    { 3, 4, 6},
    { 5, 5, 5},
    { 8, 9, 10},
};
```

Se pueden mostrar los datos de la siguiente forma:

```
for(int i=0; i<nombres.length; i++) {
    System.out.println( nombres[i]
        + " tiene "
        + edades[i]
        + " años y ha obtenido las notas "
        + Arrays.toString(notas[i]));
}
```

Se obtiene como resultado:

```
Marta tiene 18 años y ha obtenido las notas [7, 8, 8]
Yaiza tiene 21 años y ha obtenido las notas [6, 4, 7]
Anibal tiene 19 años y ha obtenido las notas [3, 4, 6]
Rosendo tiene 28 años y ha obtenido las notas [5, 5, 5]
Susana tiene 20 años y ha obtenido las notas [8, 9, 10]
```