

11.1. ARRAYS

Un *array* (lista o tabla) es una secuencia de datos del mismo tipo. Los datos se llaman *elementos* del array y se numeran consecutivamente 0, 1, 2, 3... El tipo de elementos almacenados en el array puede ser cualquier tipo de dato simple de Java, o de tipo previamente declarado. Normalmente el array se utiliza para almacenar tipos tales como *char*, *int* o *float*.

Un array puede contener, por ejemplo, la edad de los alumnos de una clase, las temperaturas de cada día de un mes en una ciudad determinada, o el número de personas que residen en cada una de las diecisiete comunidades autónomas españolas. Cada ítem del array se denomina *elemento*.

Los elementos de un array se numeran, como ya se ha comentado, consecutivamente 0, 1, 2, 3... Estos números se denominan *valores índice* o *subíndice* del array. El término «subíndice» se utiliza, ya que se especifica igual que en matemáticas, como una secuencia tal como $a_0, a_1, a_2\dots$. Estos números localizan la posición del elemento dentro del array, proporcionando *acceso directo* al array.

Si el nombre del array es *a*, entonces *a[0]* es el nombre del elemento que está en la posición 0, *a[1]* es el nombre del elemento que está en la posición 1, etc. En general, el elemento *i*-ésimo está en la posición *i-1*. De modo que si el array tiene *n* elementos, sus nombres son *a[0], a[1], \dots, a[n-1]*. Gráficamente se representa así el array *a* con seis elementos.

a	25.1	34.2	5.25	7.45	6.09	7.54
0	1	2	3	4	5	

Figura 11.1. Array de seis elementos.

El array *a* tiene seis elementos: *a[0]* contiene 25.1, *a[1]* contiene 34.2, *a[2]* contiene 5.25, *a[3]* contiene 7.45, *a[4]* contiene 6.09 y *a[5]* contiene 7.54. El diagrama de la Figura 11.1 representa realmente una región de la memoria de la computadora, ya que un array se almacena siempre con sus elementos en una secuencia de posiciones de memoria contigua.

En Java, los índices de un array siempre tienen como límite inferior 0, como índice superior el tamaño del array menos 1.

11.1.1. Declaración de un array

Al igual que con cualquier tipo de variable, se debe declarar un array antes de utilizarlo. Un array se declara de modo similar a otros tipos de datos, excepto que se debe indicar al compilador que es un array y esto se hace con los corchetes.

```
int []v;  
float w[];
```

Los corchetes se pueden colocar de dos formas:

- Colocando los corchetes a continuación del tipo de datos.
- Colocando los corchetes a continuación del nombre del array

Así, la sintaxis de declaración de variables array en Java:

```
tipo [] identificador;  
tipo identificador [];
```

El primer formato indica que todos los identificadores son arrays del tipo. El segundo formato es array sólo el identificador al que le siguen los [].

EJEMPLO 11.1. Se escriben distintas declaraciones de arrays.

1. `char cad[], p;`
cad es un array de tipo char; p es una variable de tipo char.
2. `int [] v, w;`
tanto v como w son declarados arrays unidimensionales de tipo int.
3. `double [] m, t[], x;`
m y x son array de tipo double; t es un array de array con elementos de tipo double.

Precaución

Java no permite en la declaración de una variable array indicar el número de elementos. Así, por ejemplo, la declaración `int numeros[12];` el compilador producirá un error.

Para indicar el número de elementos que tiene un array se usa el operador new. Por ejemplo, para crear una array que guarde las notas de la asignatura de música en un aula de 26 alumnos:

```
float [] notas;  
notas = new float[26];
```

Se puede escribir en una misma sentencia:

```
float [] notas = new float[26];
```

La *sintaxis* para declarar y definir un array de un número de elementos determinado es:

```
tipo nombreArray[] = new tipo[numeroDeElementos];
```

o bien,

```
tipo nombreArray[];  
nombreArray = new tipo[numeroDeElementos];
```

Por ejemplo, para crear un array (lista) de diez variables de tipo entero se escribe:

```
int a[] = new int [10];
```

Esta declaración hace que el compilador reserve espacio suficiente para contener diez datos de tipo entero. En Java, los enteros ocupan, normalmente, cuatro bytes, de modo que un array de diez enteros ocupa 40 bytes de memoria. A esta memoria hay que añadir cuatro bytes más que Java utiliza para guardar el número de elementos del array. La Figura 11.2 muestra el esquema de un array de diez elementos; cada elemento puede tener su propio valor.

Se puede acceder a cada elemento del array utilizando un índice en el nombre del array. Por ejemplo,

```
System.out.print(a[4]);
```

visualiza el valor del elemento 5 del array. Los arrays siempre comienzan en el elemento 0. Así pues, el array a contiene los siguientes elementos individuales:

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]
a[8]	a[9]						

10										
	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Un array de enteros se almacena en bytes consecutivos de memoria. Cada elemento utiliza cuatro bytes. Se accede a cada elemento de array mediante un índice que comienza en cero. Así, el acceso al elemento quinto es a [4].

Figura 11.2. Almacenamiento de un array en memoria.

si, por ejemplo, se quiere crear un array de números reales. El tamaño es una constante representada por una variable protegida (`final`) .

```
final int N = 20;
float vector[];
vector = new float[N];
```

Para acceder al tercer elemento y leer un valor de entrada:

```
vector[2] = (Float.valueOf(entrada.readLine())).floatValue();
```

Precaución

Es un error frecuente acceder a un elemento de un array fuera del rango en que está definido. Java comprueba en tiempo de compilación que los índices estén dentro de rango, en caso contrario genera un error. Durante la ejecución del programa un acceso fuera de rango genera una excepción (error en tiempo de ejecución).

11.1.2. Subíndices de un array

El índice de un array se denomina, con frecuencia, *subíndice del array*. El término procede de las matemáticas, en las que un subíndice se utiliza para representar un elemento determinado.

numeros ₀	equivale a	numeros[0]
numeros ₃	equivale a	numeros[3]

El método de numeración del elemento *i*-ésimo con el índice o subíndice *i-1* se denomina *indexación basada en cero*. Su uso tiene el efecto de que el índice de un elemento del array es siempre el mismo que el número de «pasos» desde el elemento inicial a [0] a ese elemento. Por ejemplo, a [3] está a tres pasos o posiciones del elemento a [0].

Ejemplos

```
int []edad = new int[5];      Array edad contiene cinco elementos: el primero, edad[0], y el
                            último, edad[4].
int [] pesos, longitudes;    Declara dos arrays de enteros (no asigna tamaño).
```

```

float salarios[] ; Declarar un array de 25 elementos float .
salarios = new float[25] ;
double temperaturas[] ; Declarar un array de 50 elementos double .
temperaturas = new double[50] ;
char []letras = new char[25] ; Declarar un array de caracteres .
Racional []ra = new Racional[5] ; Declarar un array de cinco objetos racional .
final int MX = 20 ;
char buffer[] =new char[MX+1] ; Declarar un array de caracteres de tamaño MX+1, el primer
elemento es buffer[0] y el último buffer[MX].

```

En los programas se pueden referenciar elementos del array utilizando fórmulas para los subíndices. Mientras que el subíndice pueda ser evaluado a un entero, se puede utilizar una constante, una variable o una expresión para el subíndice. Así, algunas referencias individuales a elementos son:

```

edad[4]
ventas [total+5]
bonos [mes]
salario [mes[i]*5]

```

11.1.3. Almacenamiento de los arrays en memoria

Los elementos de los arrays se almacenan en bloques contiguos. Así, por ejemplo, los arrays

```

int edades[] ;
char codigos[] ;
edades = new int[5] ;
codigos = new char[5]

```

se representan gráficamente en memoria en la Figura 11.3.

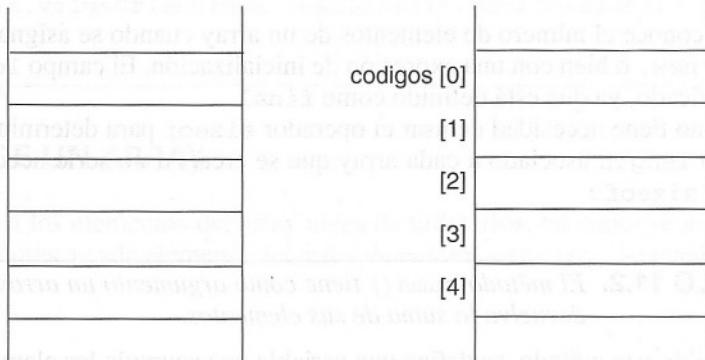


Figura 11.3. Almacenamiento de arrays en memoria.

Nota

Todos los subíndices de los arrays en Java comienzan con 0.

Los arrays de caracteres funcionan de igual forma que los arrays numéricos, partiendo de la base de que cada carácter ocupa normalmente dos bytes. Así, por ejemplo, un array llamado nombre se puede representar en la Figura 11.4.

nombre	
C	[0]
a	[1]
z	[2]
o	[3]
r	[4]
l	[5]
a	[6]

Figura 11.4. Almacenamiento de un array de caracteres en memoria.

11.1.4. El tamaño de los arrays. Atributo length

Java considera cada array un objeto, debido a ello se puede conocer el número de elementos de un array accediendo al campo `length`. Este campo resulta muy útil cuando se pasa un array a un método.

```
double [] v = new double[15];
System.out.print(v.length); //escribe 15, número de elementos de v.
```

Java conoce el número de elementos de un array cuando se asigna el número de elementos con el operador `new`, o bien con una expresión de inicialización. El campo `length` está protegido, no puede ser modificado, ya que está definido como `final`.

Java no tiene necesidad de usar el operador `sizeof` para determinar la longitud de un array, tiene el campo `length` asociado a cada array que se crea. Al no serlo necesario, Java no tiene definido el operador `sizeof`.

EJEMPLO 11.2. *El método `suma ()` tiene como argumento un array de tipo `double`. Se quiere que devuelva la suma de sus elementos.*

Para escribir este método, se define una variable que acumula los elementos del array. En un bucle con tantas iteraciones como elementos (longitud) se determina la suma.

```
double suma (double [] w)
{
    double s = 0.0;
    for (int i=0; i < w.length; i++)
        s += w[i];
    return s;
}
```

Precaución

El número de elementos de un array es un campo del array, no es un método:

```
w.length; // es correcto  
w.length(); // es un error
```

11.1.5. Verificación del rango del índice de un array

Java, al contrario que el lenguaje C, verifica el valor del índice de la variable que representa al array. Así, por ejemplo, en Java si se define un array a de seis elementos, índices 0 a 5, entonces el acceso a [6] es detectado por el compilador y genera un mensaje de error. Durante la ejecución del programa también puede ocurrir el acceso a un elemento fuera de los índices, esto producirá que el programa se «rompa» en tiempo de ejecución, generando una excepción.

EJEMPLO 11.3. *Protección frente a errores en el intervalo (rango) de valores de una variable de índice que representa un array.*

```
int datos(double a[]) throws IOException  
{  
    int n;  
    System.out.println("Entrada de datos, cuantos elementos: ? ");  
    System.out.flush();  
    n = Integer.parseInt(entrada.readLine());  
    if (n > a.length)  
        return 0;  
    for (int i = 0; i < n; i++)  
        a[i] = Double.valueOf(entrada.readLine()).doubleValue();  
    return 1;  
}
```

11.2. INICIALIZACIÓN DE UN ARRAY

Se deben asignar valores a los elementos del array antes de utilizarlos, tal como se asignan valores a variables. Para asignar valores a cada elemento del array de enteros precios, se puede escribir:

```
precios[0] = 10;  
precios[1] = 20;  
precios[3] = 30;  
precios[4] = 40;  
...
```

La primera sentencia fija precios[0] al valor 10, precios[1] al valor 20, etc. Sin embargo, este método no es práctico cuando el array contiene muchos elementos. El método utilizado, normalmente, es inicializar el array completo en una sola sentencia.

Cuando se inicializa un array, el tamaño del array se puede determinar automáticamente por las constantes de inicialización. Estas constantes se separan por comas y se encierran entre llaves, como en los siguientes ejemplos:

```
int numeros[] = {10, 20, 30, 40, 50, 60}; /* Define un array de 6
   elementos y se inicializan a las constantes */
int n[] = {3, 4, 5}                      // Define un array de 3 elementos
char c[] = {'L','u','i','s'};      // Define un array de 4 elementos
```

El array `numeros` tiene seis elementos, `n` tiene tres elementos y el array `c` tiene cuatro elementos.

Nota

La serie de valores entre corchetes sólo puede ser usada para inicializar un array y no en sentencias de asignación posteriores.

```
int cuenta[] = {15, 25, -45, 0, 50};
```

El compilador asigna automáticamente cinco elementos a `cuenta`.

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

```
edades[] = {14, 22, 17, 20, 50}; // es una sentencia errónea
```

El método de inicializar arrays mediante valores constantes después de su definición es adecuado cuando el número de elementos del array es pequeño. Por ejemplo, para inicializar un array (lista) de 10 enteros a los valores 10 a 1, y a continuación visualizar dichos valores en orden inverso, se puede escribir:

```
int cuenta[] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
for (i = 9; i >= 0; i--)
    System.out.println("cuenta descendente " + i + "=" + cuenta[i]);
```

Se pueden asignar valores numéricos representados mediante variables protegidas con la cláusula `final`, de modo que las sentencias siguientes son válidas:

```
final int ENE = 31, FEB = 28, MAR = 31, ABR = 30, MAY = 31,
JUN = 30, JUL = 31, AGO = 31, SEP = 30, OCT = 31,
NOV = 30, DIC = 31;
...
int meses[] = {ENE, FEB, MAR, ABR, MAY, JUN,
JUL, AGO, SEP, OCT, NOV, DIC};
```

Pueden asignarse valores a un array utilizando un bucle `for` o `while/do-while`, y éste suele ser el sistema más empleado normalmente. Por ejemplo, para inicializar todos los valores del array `numeros` al valor `-1`, se puede utilizar la siguiente sentencia:

```
for (i = 0; i < numeros.length; i++)
    numeros[i] = -1;
```

debido a que el valor del subíndice `i` varía de 0 al número de elementos `-1`, cada elemento del array `numeros` se inicializa y se establece a `-1`.

EJEMPLO 11.4. El siguiente programa lee `NUM` enteros; a continuación visualiza el producto de los números.

```
import java.io.*;
class Inicial
```

```
{  
    public static void main(String [] a) throws IOException  
    {  
        final int NUM = 10;  
        BufferedReader entrada = new BufferedReader(  
            new InputStreamReader(System.in));  
        int nums[] = new int[NUM];  
        int total = 1;  
        System.out.println("Por favor, introduzca " + NUM + " datos");  
        System.out.flush();  
        for (int i = 0; i < NUM; i++)  
        {  
            nums[i] = Integer.parseInt(entrada.readLine());  
        }  
        System.out.print("\nLista de números: ");  
        for (i = 0; i < NUM; i++)  
        {  
            System.out.print(" " + nums[i]);  
            total *= nums[i];  
        }  
        System.out.println("\nEl producto de los números es " + total);  
    }  
}
```

Los arrays en Java se inicializan a 0 por defecto. Por ello, la ejecución del siguiente programa visualiza 0 para los 10 valores del array:

```
int lista[] = new int[10];
for (int j = 0; j <= 9; j++)
    System.out.print("\n lista " + j + "=" + lista[j]);
```

Nota

El compilador Java inicializará cualquier array con un valor por defecto, ceros binarios, ya sea el array de tipo entero, real o de caracteres.

11.3. ARRAYS DE CARACTERES Y CADENAS DE TEXTO

Una cadena de texto es un conjunto de caracteres, tales como "ABCDEFG". Java soporta cadenas de texto utilizando la clase `String` y `StringBuilder` implementada en el paquete `java.lang`.

```
String cadena = "ABCDEFG";
```

Es importante comprender la *diferencia* entre un array de caracteres y una cadena de caracteres. Los *strings* son objetos en los que se almacenan las cadenas, tienen diversos constructores y métodos. Los arrays de tipo char son una secuencia de caracteres, con las mismas características que los arrays de otros tipos.

```
String dat = new String(datos); /* crea objeto cadena con el  
constructor que tiene como argumento un array de caracteres */  
System.out.println(dat); // muestra la cadena dat: "Fichero"  
System.out.println(datos); // es un error, datos no es una cadena
```

Las cadenas se deben almacenar en objetos de tipo `String` o de tipo `StringBuffer`.

Una vez que un objeto `String` es creado e inicializado con una cadena, no puede modificarse. Los métodos definidos en `String` para concatenar, cambiar caracteres... no modifican la cadena, sino que devuelven otro objeto `String`. Se puede decir que cadenas de tipo `String` son de sólo lectura.

Las cadenas definidas como objetos `StringBuffer` pueden cambiar la longitud de la cadena y el contenido.

```
StringBuffer cc = new StringBuffer("Cadena variable");  
cc.replace('v', 'V'); // cambia la letra v por V en el objeto cc.
```

Para concatenar constantes cadena se utiliza el operador `+`, como ya se ha hecho en la salida mediante `print()` o `println()`. El operador está sobrecargado para poder convertir otros tipos de datos a tipo cadena y realizar la concatenación. Así:

```
int h = 24;  
System.out.print("valor de h = " + h); /* concatena y forma la  
cadena "valor de h = 24" */  
System.out.print("valor de h = " + h+h); /* concatena y forma la  
cadena "valor de h = 2424" */  
System.out.print("valor de h = " + (h+h)); /* concatena y forma la  
cadena "valor de h = 48" */
```

La clase `String` tiene definido el método `concat()` para unir o concatenar dos cadenas. El método devuelve otro objeto `String` con las dos cadenas unidas.

```
String c1 = "Mi pueblo es ";  
String c2 = "Lupiana";  
System.out.println(c1.concat(c2)); /* se muestra en pantalla:  
"Mi pueblo es Lupiana" */
```

La longitud de una cadena se obtiene llamando al método `length()` de la clase `String`. Una confusión habitual es obtener la longitud de un array con una llamada a `length()`; o a la inversa, obtener la longitud de una cadena con el campo `length`.

```
String buf = " Cadena de oro";  
char acad[] = new char[6];  
for (int j= 0; j<6; j++)  
    acad[j] = (char) 'a'+j;  
System.out.println("Longitud de buf = " + buf.length());  
System.out.println("Longitud de acad = " + acad.length);
```

EJEMPLO 11.5. *El programa crea un array de cadenas con las líneas de texto leídas desde el teclado.*

El programa define un array de objetos cadena: `String cads[]`; el máximo de líneas que van a ser leídas es 23. Una vez terminada la entrada se muestran por pantalla.

```

import java.io.*;
class Cadenas
{
    public static void main(String [] a) throws IOException
    {
        final int NUM = 23;
        BufferedReader entrada = new BufferedReader(
                new InputStreamReader(System.in));
        String[] cads= new String[NUM];
        System.out.println("\nIntroduzca " + NUM + " líneas");
        for (int i = 0; i < NUM; ++i)
            cads[i] = entrada.readLine();
        System.out.println("\n\tLíneas tecleadas");
        for (int i = 0; i < NUM; ++i)
            System.out.println(i + ":" + cads[i]);
    }
}

```

EJEMPLO 11.6. Visualizar el array entero aleat después de asignar datos en el mismo con el método random() de la clase Math. El rango de números aleatorios que sea de 1 a 99.

El método random() genera números aleatorios entre 0.0 y 1.0 (no inclusive) de tipo double. Para generar números enteros entre 1 y 99 se multiplica por el límite superior y se suma 1:

```
1 + (int) Math.random()*99
```

```

import java.io.*;
class Aleatorio
{
    public static void main(String [] a)
    {
        final int TOPE = 99;
        final int N = 19;
        int [] v = new int[N];
        for (int i = 0; i < N; i++)
            v[i] = 1 + (int) Math.random()*TOPE;
        System.out.println("\n Valores generados aleatoriamente");
        for (int i = 0; i < N; i++)
            System.out.print(v[i] + " ");
    }
}

```

11.4. COPIA DE ARRAYS

Los elementos de un array se pueden asignar a otro array con un bucle que recorra cada elemento, el array destino tiene que estar definido con al menos los mismos elementos. Así, por ejemplo:

```

final int N = 12;
int v1[] = new int[N], v2[] = new int[N];
for (int i = 0; i < N; i++)
    v1[i] = (int) Math.random()*199 + 1 ;
// Los elementos de v1 son copiados a v2
for (int i = 0; i < N; i++)
    v2[i] = v1[i];

```

En vez de una copia de todos los elementos puede interesar copiar una parte de ellos, esto se hace variando el índice inicial o el índice final del bucle. Así, si se quiere copiar la primera mitad de los elementos:

```
for (int i = 0; i < v1.length/2; i++)
    v2[i] = v1[i];
```

Esta copia se puede hacer con un método de la clase `System`, `arrayCopy()`. Antes conviene recordar que los nombres de arrays son referencias a un bloque de memoria distribuida según el número de elementos, por lo que si se hace una asignación entre dos variables array éstas se refieren al mismo array. Por ejemplo:

```
double [] r, w;
r = new double[11];
w = new double[15];
for (int j = 0; j < r.length; j++)
    r[j] = (double) 2*j-1;
// asignación de r a w
w = r;
```

Esta asignación hace que se pueda acceder a los elementos desde `r` o desde `w`, pero no se ha creado un nuevo almacenamiento para los elementos; los 15 elementos que inicialmente se referencian desde `w` se han perdido.

Para copiar los 11 elementos que tiene al vector `r` en `w` con el método `arrayCopy()` hay que especificar la posición inicial del vector desde el que se copia, la posición del vector destino donde se inicia la copia y el número de elementos:

```
System.arraycopy(r, 0, w, 0, 11);
```

EJEMPLO 11.7

Se quiere definir dos arrays de tipo `double`, `v` y `w`, con 15 y 20 elementos, respectivamente. En el array `v` se van a guardar los valores de la función e^{2x-1} para valores de x mayor o igual de 1.0; el array `w` se inicializa cada elemento al ordinal del elemento. A continuación se copian los 10 últimos elementos de `v` a partir del elemento 11 de `w`. Por último se escriben los elementos de ambos arrays.

El programa que se escribe a continuación sigue los pasos indicados en el enunciado. Se usa la función `exp()` de la clase `Math` para el cálculo de la función e^{2x-1} , así como el método `arrayCopy()` para realizar la copia de elementos de array pedida.

```
import java.io.*;
class CopiArray
{
    public static void main(String [] a)
    {
        final int N = 15;
        final int M = 20;
        double [] v = new double[N], w = new double [M];
        double x = 1.0;
        for (int i = 0; i < N; x+=0.2, i++)
            v[i] = Math.exp(2*x-1);
        for (int i = 0; i < M; i++)
            w[i] = (double)i;
        // Se imprimen los elementos del vector v
```

```

        System.out.println("\n Valores del vector v");
        for (int i = 0; i < N; i++)
            System.out.print(v[i] + " ");
        System.out.flush();
        // Es realizada la copia de v a w
        System.arraycopy(v,N-10+1,w,10,11);
        // Se impimen los elementos del vector w
        System.out.println("\n Valores del vector w");
        for (int i = 0; i < M; i++)
            System.out.print(w[i] + " ");
        System.out.flush();
    }
}

```

La forma sintáctica de llamada al método arraycopy:

System.arraycopy(arrayOrigen, inicioOrigen, arrayDestino, inicioDestino, elementos)	
arrayOrigen:	nombre del array desde el que se va a copiar.
inicioOrigen:	posición del array origen desde la que se inicia la copia.
arrayDestino:	nombre del array en el que se hace la copia.
inicioDestino:	es la posición del array destino donde empieza la copia.
elementos:	número de elementos del array origen que se van a copiar.

Precaución

Ha de haber espacio suficiente en el array destino para realizar la copia de elementos desde el array fuente, en caso contrario se provoca un error en la ejecución.

En este ejemplo se copian elementos de un array w en un array p, se comete un error por no tener suficiente memoria reservada:

```

final int N =20;
double w[] = new double[N];
// asignación de valores a los elementos de w
// se define un array para copiar los 10 últimos elementos de w
double p[] = new double [10];
int j = 1;
System.arraycopy(w,N-10,p,j,10 );// la copia en p[] se sale de rango

```

11.5. ARRAYS MULTIDIMENSIONALES

Los arrays vistos anteriormente se conocen como arrays *unidimensionales* (una sola dimensión) y se caracterizan por tener un solo subíndice. Estos arrays se conocen también por el término *listas*. Los arrays *multidimensionales* son aquellos que tienen más de una dimensión y, en consecuencia, más de un índice. Los arrays más usuales son los de dos dimensiones, conocidos también por el nombre de *tablas*.

o matrices. Sin embargo, es posible crear arrays de tantas dimensiones como requieran sus aplicaciones, esto es, tres, cuatro o más dimensiones.

Un array de dos dimensiones equivale a una tabla con múltiples filas y múltiples columnas (Fig. 11.5).

	0	1	2	3	...	n
0						
1						
2						
3						
4						
m						

Figura 11.5. Estructura de un array de dos dimensiones.

Obsérvese que en el array bidimensional de la Figura 11.5, si las filas se etiquetan de 0 a m y las columnas de 0 a n , el número de elementos que tendrá el array será el resultado del producto $(m+1) * (n+1)$. El sistema de localizar un elemento será por las coordenadas representadas por su número de fila y su número de columna (a, b). La sintaxis para la declaración de un array de dos dimensiones es:

<tipo de datoElemento> <nombre array> [] [];
o bien
<tipo de datoElemento> [] []<nombre array>;

Algunos ejemplos de declaración de tablas:

```
char pantalla[] [] ;  
int puestos[] [] ;  
double [] [] matriz ;
```

Estas declaraciones no reservan memoria para los elementos de la matriz o tabla, realmente son referencias. Para reservar memoria y especificar el número de filas y de columnas se utiliza el operador new. Así, a partir de las declaraciones anteriores:

```
pantalla = new char[80] [24]; // matriz con 80 filas y 24 columnas  
puestos = new int[10] [5]; // matriz de 10 filas por 5 columnas  
final int N = 4;  
matriz = new double[N] [N]; // matriz cuadrada de N*N elementos
```

El operador new se puede aplicar a la vez que se hace la declaración. La sintaxis para definir una matriz:

```
<tipo de datoElemento> <nombre array>[][] =  
new<tipo de datoElemento> [<NúmeroDeFilas>] [<NúmeroDeColumnas>];
```

Atención

Java requiere que cada dimensión esté encerrada entre corchetes. La sentencia

```
int equipos[] [] = new int[4,5];
```

no es válida.

Un array de dos dimensiones en realidad es un *array de arrays*. Es decir, es un array unidimensional, y cada elemento no es un valor entero, o de coma flotante o carácter, sino que cada elemento es otro array.

Los elementos de los arrays se almacenan en memoria de modo que el subíndice más próximo al nombre del array es la fila y el otro subíndice, la columna. En la Tabla 11.1 se representan todos los elementos y sus posiciones relativas en memoria del array, `int tabla[] []=new int [4] [2]`, suponiendo que cada entero ocupa cuatro bytes.

Tabla 11.1. Un array bidimensional

Elemento	Posición relativa de memoria
tabla[0][0]	0
tabla[0][1]	4
tabla[1][0]	8
tabla[1][1]	12
tabla[2][0]	16
tabla[2][1]	20
tabla[3][0]	24
tabla[3][1]	28

11.5.1. Inicialización de arrays multidimensionales

Los arrays multidimensionales se pueden inicializar, al igual que los de una dimensión, cuando se declaran. La inicialización se hace utilizando llaves, encerrando entre llaves la lista de constantes separadas por comas de que consta cada fila, como en los ejemplos siguientes:

1. `int tabla1[] [] = { {51, 52, 53}, {54, 55, 56} };`

Se ha definido una matriz de dos filas por tres columnas cada fila.

O bien en estos otros formatos más amigables:

```
int tabla1[] []= { {51, 52, 53},
                  {54, 55, 56} };
int tabla1[] []= {
                  {51, 52, 53},
                  {54, 55, 56}
};
```

2. `int tabla2[] [] = {
 {1, 2, 3, 4},
 {5, 6, 7, 8},
 {9, 10, 11, 12}
};`

tabla1 [] []

Filas	Columns		
	0	1	2
0	51	52	53
1	54	55	56

tabla2 [] []

Filas	Columns				
	0	1	2	3	4
0	1	2	3	4	
1	5	6	7	8	
2	9	10	11	12	

Figura 11.6. Tablas de dos dimensiones.

Java trata los arrays de dos o más dimensiones como array de arrays, por ello se pueden crear arrays de dos dimensiones no cuadradas. En los siguientes ejemplos se crean arrays de distintos elementos cada fila.

1. double tb[] [] = { {1.5, -2.5}, {5.0, -0.0, 1.5} };

Se ha definido una matriz de dos filas, la primera con dos columnas y la segunda con tres.

2. int []a = {1,3,5}, b = {2,4,6,8,10};
int mtb[] [] = {a, b};

Se ha definido el array a de tres elementos, el b de cuatro elementos y la matriz mtb de dos filas, la primera con tres elementos o columnas, y la segunda con cuatro.

Se puede realizar la definición de una matriz en varias sentencias, primero especificando el número de filas y a continuación el número de elementos de cada fila. En los siguientes ejemplos se crean matrices de esta forma.

1. double [] []gr = new double[3] [];

Se define la matriz gr de tres filas. A continuación, los elementos de cada fila.

```
gr[0] = new double[3];
gr[1] = new double[6];
gr[2] = new double[5];
```

2. int [] []pres = new int[4] [];

Se define la matriz de tipo entero pres de cuatro filas. A continuación, los elementos de cada fila se definen con las sentencias:

```
pres[0] = v0;
pres[1] = v1;
pres[2] = v2;
pres[3] = v3;
```

Previamente hay que definir:

```
int V0 = {4, 3, 2, 1};  
int V1 = {7, 6, 5};  
int V2 = {9, 8};  
int V3 = {0};
```

Nota

En un array bidimensional `tabla`, al ser un array de arrays, el atributo `length` de `tabla` contiene el número de filas. El atributo `length` de cada array fila contiene el número de columnas.

```
float ventas[][] = {{0., 0., 0.}, {1.0, 1.0}, {-1.0}};  
  
System.out.print(ventas.length); // escribe 3  
System.out.print(ventas[0].length); // escribe 3  
System.out.print(ventas[1].length); // escribe 2  
System.out.print(ventas[2].length); // escribe 1
```

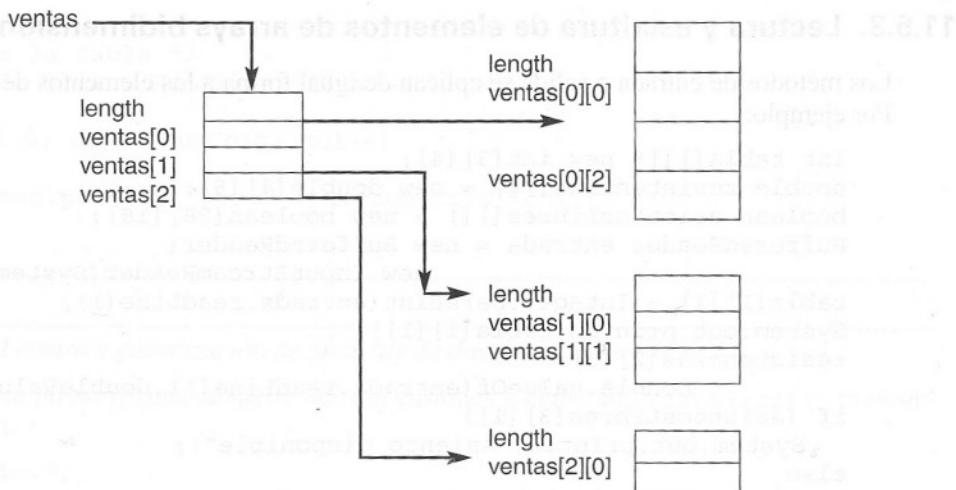


Figura 11.7. Disposición en memoria de `ventas [][]`.

Precaución

En la definición de un array bidimensional no es posible omitir el número de filas, así la declaración:

```
double vt[][] = new double[] [4];
```

es errónea, ya que no se ha especificado el número de filas y el tamaño queda indeterminado.

11.5.2. Acceso a los elementos de los arrays bidimensionales

Se puede acceder a los elementos de arrays bidimensionales de igual forma que a los elementos de un array unidimensional. La diferencia reside en que en los elementos bidimensionales deben especificarse los índices de la fila y la columna.

El formato general para asignación directa de valores a los elementos es:

inserción de elementos

```
<nombre array>[indice fila] [indice columna]=valor elemento;
```

extracción de elementos

```
<variable> = <nombre array> [indice fila] [indice columnna];
```

Algunos ejemplos de inserciones:

```
Tabla[2][3] = 4.5;  
Resistencias[2][4] = 50;  
AsientosLibres[5][12] = 5;
```

y de extracción de valores:

```
Ventas = Tabla[1][1];  
Dia = Semana[3][6];
```

11.5.3. Lectura y escritura de elementos de arrays bidimensionales

Los métodos de entrada o salida se aplican de igual forma a los elementos de un array bidimensional. Por ejemplo:

```
int tabla[][] = new int[3][4];  
double resistencias[][] = new double[4][5];  
boolean asientosLibres[][] = new boolean[28][18];  
BufferedReader entrada = new BufferedReader(  
                    new InputStreamReader(System.in));  
tabla[1][1] = Integer.parseInt(entrada.readLine());  
System.out.println(tabla[1][1]);  
resistencias[2][1] =  
    Double.valueOf(entrada.readLine()).doubleValue();  
if (asientosLibres[3][1])  
    System.out.println("Asiento disponible");  
else  
    System.out.println("Asiento ocupado");
```

11.5.4. Acceso a elementos mediante bucles

Se puede acceder a los elementos de arrays bidimensionales mediante bucles anidados. Su sintaxis es:

```
int fila, col;  
for (fila = 0; fila < NumFilas; ++fila)  
    for (col = 0; col < NumCol; ++col)  
        Procesar elemento Matriz[fila][col];
```

El número de filas y de columnas se puede obtener con el atributo `length`. Con este atributo, la sintaxis para acceder a los elementos:

```
<tipo> Matriz[] [] ;  
<especificación de filas y columnas con operador new>
```

```

for (fila = 0; fila < Matriz.length; ++fila)
    for (col = 0; col < Matriz[fila].length; ++col)
        Procesar elemento Matriz[fila] [col];

```

EJEMPLO 11.8. Define una tabla de discos, el número de filas y de columnas son datos de entrada. Rellena la tabla con datos de entrada y se muestran por pantalla.

```

float discos[][] ;
int fila, col, NumFilas, NumCols;
System.out.print("Número de filas: ");
System.out.flush();
NumFilas = Integer.parseInt(entrada.readLine());
System.out.print("Número de columnas: ");
System.out.flush();
NumCols = Integer.parseInt(entrada.readLine());
discos = new float [NumFilas] [NumCols];
// Entrada de datos
for (fila = 0; fila < NumFilas; fila++) {
    for (col = 0; col < NumCols; col++)
    {
        discos[fila] [col] =
            Float.valueOf(entrada.readLine()).floatValue();
    }
}
/* Visualizar la tabla */
for (fila = 0; fila < NumFilas; fila++)
{
    for (col = 0; col < NumCols; col++)
    {
        System.out.println("Pts " + discos[fila] [col]);
    }
}

```

EJERCICIO 11.1. Lectura y visualización de un array de dos dimensiones.

El método leer() lee un array (una tabla) de dos dimensiones y el método visualizar() presenta la tabla en la pantalla.

```

import java.io.*;
class Tabla
{
    public static void main(String [] a) throws IOException
    {
        int a[][]= new int[3] [5];
        leer(a);
        visualizar(a);
    }

    static void leer(int a[][])throws IOException
    {
        int i,j;
        BufferedReader entrada = new BufferedReader(
                new InputStreamReader(System.in));
        System.out.println("Entrada de datos de la matriz");
        for (i = 0; i < a.length; i++)
        {

```

```

11.5.2. Acceso
    System.out.println("Fila: " + i);
    System.out.flush();
    for (j = 0; j < a[i].length; j++)
        a[i][j] = Integer.parseInt(entrada.readLine());
    }
}

static void visualizar (int a[][])
{
    int i,j;
    System.out.println("\n\t Matriz leida\n");
    for (i = 0; i < a.length; i++)
    {
        for (j = 0; j < a[i].length; j++)
            System.out.print(a[i][j] + " ");
        System.out.println(" ");
    }
}

```

Ejecución

La traza (ejecución) del programa:

Entrada de datos de la matriz

Fila 0:

45
75
25
10
40

Fila 1:

Matriz leida
45 75 25 10 40
20 14 36 15 26
21 15 37 16 27

11.5.5. Arrays de más de dos dimensiones

Java proporciona la posibilidad de almacenar varias dimensiones, aunque raramente los datos del mundo real requieren más de dos o tres dimensiones. El medio más fácil de dibujar un array de tres dimensiones es imaginar un cubo tal como se muestra en la Figura 11.8.

Un array tridimensional se puede considerar como un conjunto de arrays bidimensionales combinados juntos para formar, en profundidad, una tercera dimensión. El cubo se construye con filas (dimensión vertical), columnas (dimensión horizontal) y planos (dimensión en profundidad). Por consiguiente, un elemento dado se localiza especificando su plano, fila y columna. Una definición de un array tridimensional `equipos` es:

```
int equipos[][][] = new int[3][15][10];
```

Un ejemplo típico de un array de tres dimensiones es el modelo *libro*, en el que cada página del libro es un array bidimensional construido por filas y columnas. Así, por ejemplo, cada página tiene cuarenta y cinco líneas que forman las filas del array y ochenta caracteres por línea, que forman las columnas del array. Por consiguiente, si el libro tiene quinientas páginas, existirán quinientos planos y el número de elementos será $500 \times 80 \times 45 = 1.800.000$.

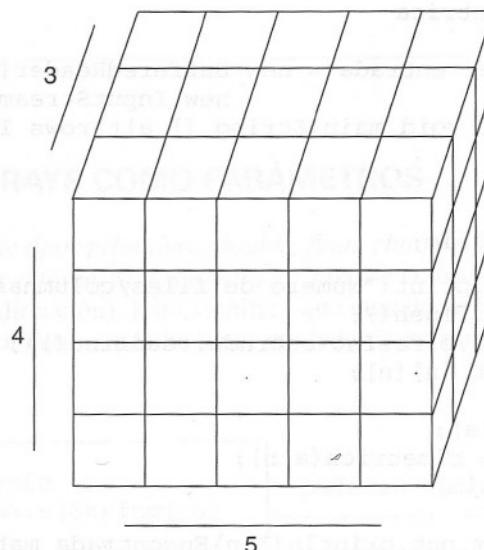


Figura 11.8. Un array de tres dimensiones ($4 \times 5 \times 3$).

11.5.6. Una aplicación práctica

El array *libro* tiene tres dimensiones [PAGINAS] [LINEAS] [COLUMNAS], que definen el tamaño del array. El tipo de datos del array es char, ya que los elementos son caracteres.

¿Cómo se puede acceder a la información del libro? El método más fácil es mediante bucles anidados. Dado que el libro se compone de un conjunto de páginas, el bucle más externo será el bucle de página, y el bucle de columnas será el bucle más interno. Esto significa que el bucle de filas se insertará entre los bucles página y columna. El código siguiente permite procesar el array:

```
int pagina, linea, columna;
final int PAGINAS = 500;
final int LINEAS = 45;
final int COLUMNAS = 80;
char libro[][][] = new char[PAGINAS][LINEAS][COLUMNAS];
for (pagina = 0; pagina < PAGINAS; ++pagina)
    for (linea = 0; linea < LINEAS; ++linea)
        for (columna = 0; columna < COLUMNAS; ++columna)
            <procesar libro[pagina][linea][columna]>
```