

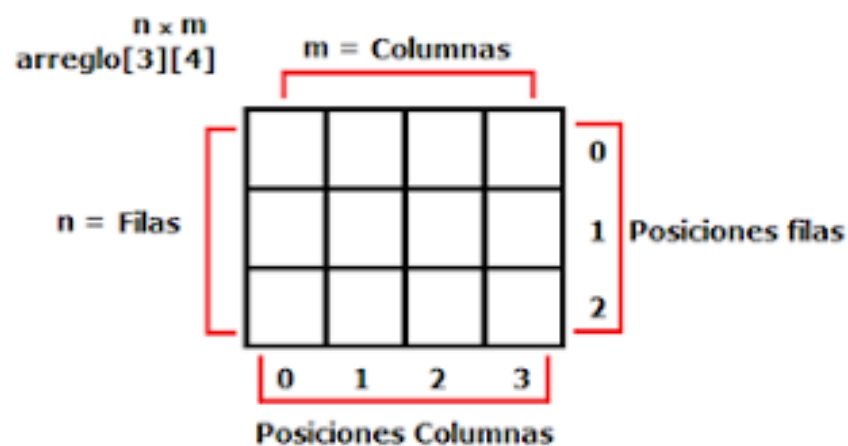


Matrices en Java

En la guía anterior trabajamos con los arreglos unidimensionales, en esta ocasión vamos a continuar con las estructuras de datos pero ahora conociendo las Matrices o arreglos bidimensionales.

¿Que son?

Este tipo de arreglos son conocidos como matrices y corresponden a una estructura de datos que puede almacenar muchos más datos que los arreglos unidimensionales, pues estos últimos como ya se mencionó se componen de una fila por n columnas, mientras que los bidimensionales se componen de n filas por m columnas.



Como se ve en la imagen, se tiene un arreglo de 3 filas por 4 columnas siendo $n=3$ y $m=4$, la lógica de la estructura es muy similar a los arreglos unidimensionales, cada índice inicia en 0 hasta el tamaño-1 por esa razón las posiciones de las filas van de 0 a 2 y el de las columnas de 0 a 3.

Se puede ver que la matriz anterior es como si fueran 3 arreglos de tamaño 4 juntos, pues se puede entender cada fila como uno de ellos, por lo tanto la declaración, construcción e inicialización es muy similar.

Declaración de Matrices.

Las matrices se identifican porque al momento de su creación se utilizan doble corchetes ([] []), al usarlos java automáticamente identifica que se va a trabajar con matrices, ya que representan el tamaño de filas por columnas que puede tener, igual que los arreglos unidimensionales se tienen 2 formas generales para su creación.

<tipoDato> identificador[] []; Ej: **int matrizDeEnteros[] [];**

O también

<tipoDato>[] [] identificador; Ej: **int[] [] matrizDeEnteros;**

Donde tipo de dato define el tipo de dato de cada uno de los valores que puede contener la matriz.

Construcción de la Matriz.

Después de haber declarado la matriz se puede construir e inicializar de 2 maneras.

Forma1.

la primera se usa cuando inicialmente no sabemos cuáles son los valores que va a contener la matriz, ya que luego serán ingresados, se crea con la siguiente estructura:

Identificador = new <tipoDato> [filas] [columnas]; Ej. **matrizDeEnteros = new int[3] [4];**

Podemos observar como la matrizDeEnteros declarada previamente, ahora es creada con un tamaño de 3 x 4 (3 filas 4 columnas) creándose un total de posiciones de memoria equivalente a la multiplicación de estos valores, así estas posiciones corresponden a un espacio donde se pueden almacenar 12 datos de tipo int.

Inicialización de la Matriz.

Igual que con los arreglos, se deben tener presente el tamaño asignado para las filas y columnas pues cada posición almacenará un valor del tipo de dato declarado para la matriz, el llenado se realiza de la siguiente manera:

```
identificador[filas] [columnas]=dato;
```

Sabemos que el identificador corresponde al nombre de la matriz, posición a alguna de las casillas y dato el valor a asignar, que debe ser del tipo de dato definido al momento de la creación.

Ej:

Llenado fila 0 columnas del 0 al 3
matrizDeEnteros[0][0]=1;
matrizDeEnteros[0][1]=3;
matrizDeEnteros[0][2]=5;
matrizDeEnteros[0][3]=7;

Llenado fila 1 columnas del 0 al 3
matrizDeEnteros[1][0]=5;
matrizDeEnteros[1][1]=4;
matrizDeEnteros[1][2]=1;
matrizDeEnteros[1][3]=16;

Llenado fila 2 columnas del 0 al 3
matrizDeEnteros[2][0]=7;
matrizDeEnteros[2][1]=9;
matrizDeEnteros[2][2]=61;
matrizDeEnteros[2][3]=13;

		Columnas			
		0	1	2	3
Filas	0	1	3	5	7
	1	5	4	1	16
	2	7	9	61	13

matrizDeEnteros [3][4]

Forma 2.

De la misma manera que la segunda forma de llenado de arreglos, para las matrices se usa cuando sabemos con exactitud cuáles son los valores que va a contener la matriz, aquí el proceso de construcción e inicialización se hace directo y se realiza de la siguiente manera:

```
Identificador = { {valor, valor,valor}, {valor, valor,valor}, {valor, valor,valor} };
```

Dónde:

- Identificador:** nombre de la matriz
- Llaves externas:** representa toda la matriz en general
- Llaves internas:** representan cada una de las filas que puede contener la matriz
- Valores:** representan los valores de cada columna en la fila respectiva

Como vemos en la estructura, se está creando una matriz de 3x3, pues hay 9 valores repartidos en 3 grupos correspondientes a las filas los cuales se muestran encerrados en las llaves internas.

Ejemplo: La matriz 3x4 usada en la forma 1 se puede representar de la siguiente manera.

```
matrizDeEnteros = { {1, 3, 5, 7} , {5, 4, 1, 16} , {7, 9, 61, 13} };
```

		Columnas			
		0	1	2	3
Filas	0	1	3	5	7
	1	5	4	1	16
	2	7	9	61	13

matrizDeEnteros [3][4]

Como se puede observar no fue necesario indicar cuál es el tamaño de la matriz, ya que java identifica el tamaño gracias a las posiciones y cantidad de valores separados por coma.

Acceso a los datos de una matriz.

Para acceder a la información almacenada dentro de una matriz se debe tener presente el nombre de la matriz, tamaño y el tipo de datos que contiene.

Por ejemplo, si queremos almacenar un dato de una variable, la forma de acceder es por medio de los índices de fila y columna que corresponde a la posición del valor a obtener:

variable = Identificador[fila] [columna];

donde la variable corresponde a una variable del tipo de dato que se quiere almacenar, el identificador corresponde al nombre de la matriz y la posición a alguno de los valores entre 0 y tamaño-1 (tanto para fila como para la columna)

Tomando el ejemplo de la matriz anterior, queremos obtener el valor en la posición (2,3) de la matriz (ver imagen anterior).

Entonces:

int dato= matrizDeEnteros [2] [3];

Por lo tanto dato, almacenará el valor 16.

Ejemplo Forma 1.

Ejemplo: el siguiente ejercicio permite crear una matriz de tipo String usando la primera forma explicada.

```
14 public static void main(String[] args) {
15
16     String estaciones[][]=new String[2][2];
17
18     //llenado de la matriz
19     estaciones[0][0]="Otoño";
20     estaciones[0][1]="Verano";
21     estaciones[1][0]="Invierno";
22     estaciones[1][1]="Primavera";
23
24     //Obteniendo información de la matriz
25     System.out.println("estación en la posición (0,0): "+estaciones[0][0]);
26     System.out.println("estación en la posición (0,1): "+estaciones[0][1]);
27     System.out.println("estación en la posición (1,0): "+estaciones[1][0]);
28     System.out.println("estación en la posición (1,1): "+estaciones[1][1]);
29 }
```

Lo anterior crea una matriz de tipo String y tamaño 2x2 con posiciones de 0 a 1 en filas y 0 a 1 en columnas

estaciones =

"Otoño"	"Verano"	0
"Invierno"	"Primavera"	1
0	1	

Ejemplo Forma 2.

Ahora miremos el siguiente ejemplo donde se creará la misma matriz anterior pero usando la segunda forma trabajada.

Ejemplo: el siguiente ejercicio permite crear 1 arreglo de tipo String usando la segunda forma explicada.

```
14 public static void main(String[] args) {
15
16     String estaciones[][]={{ "Otoño", "Verano"},
17                             {"Invierno", "Primavera"}};
18
19     //Obteniendo información de la matriz
20     System.out.println("estación en la posición (0,0): "+estaciones[0][0]);
21     System.out.println("estación en la posición (0,1): "+estaciones[0][1]);
22     System.out.println("estación en la posición (1,0): "+estaciones[1][0]);
23     System.out.println("estación en la posición (1,1): "+estaciones[1][1]);
24 }
```

Lo anterior crea una matriz de tipo String y tamaño 2x2 con posiciones de 0 a 1 en filas y 0 a 1 en columnas

estaciones =

"Otoño"	"Verano"	0
"Invierno"	"Primavera"	1
0	1	

Llenado y consulta de datos del arreglo por medio de ciclos.

Cuando se desea llenar un arreglo de muchas posiciones, los ciclos juegan un papel muy importante ya que nos permitirán hacer este proceso más dinámico, pues podemos recorrer cada posición usando la variable de incremento, tanto para asignar como para

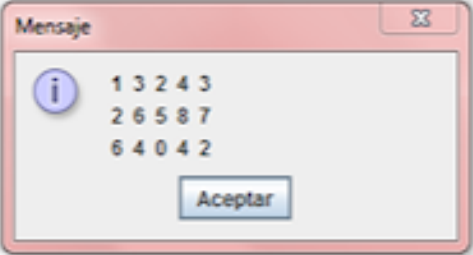
obtener.

Teniendo en cuenta que siempre cuando asignamos o consultamos datos del arreglo, debemos indicar cuál es la posición o índice que vamos a usar, la posición podemos trabajarla como una variable que toma cada uno de los valores posibles que a tomar.

Ej: arreglo[posición]=valor //asignación valor en el arreglo Variable= arreglo[posición] //asignación valor del arreglo en la variable.

Ejemplo 3: Algoritmo que solicita el ingreso de un matriz 3x5, posteriormente lo recorre y lo imprime.

```
14 public static void main(String[] args) {
15
16     int matriz[][]=new int[3][5];
17
18     for (int i = 0; i < 3; i++)
19     {
20         for (int j = 0; j < 5; j++)
21         {
22             matriz[i][j]=Integer.parseInt(JOptionPane.showInputDialog("Ingrese el numero en la posicion (" +i+" "+j+"")));
23         }
24     }
25
26     String cad="";
27     for (int i = 0; i < 3; i++)
28     {
29         for (int j = 0; j < 5; j++)
30         {
31             cad+=matriz[i][j]+" ";
32         }
33         cad+="\n";
34     }
35
36     JOptionPane.showMessageDialog(null, cad);
37
38 }
39 }
```



The screenshot shows a Java IDE with the code for Example 3. The code creates a 3x5 integer matrix, prompts the user to input values for each element, and then prints the matrix. A dialog box titled 'Mensaje' is shown, displaying the resulting matrix:

1	3	2	4	3
2	6	5	8	7
6	4	0	4	2

El ejemplo anterior realiza un proceso similar al llenado y búsqueda de un arreglo, esta vez note que se utilizan 2 ciclos anidados tanto para el llenado como la consulta de los datos.

El primer ciclo for lo que hace es recorrer cada una de las filas, caso contrario el segundo recorre las columnas, de este modo se puede usar el proceso `matriz[i][j]` para ir almacenando los datos en cada posición dado el vector (i,j) que representa (filas,columnas).

En el siguiente video puedes ver el trabajo con matrices paso a paso: <https://www.youtube.com/watch?v=kt7AODSgu7A>

Instructor: Cristian David Henao Hoyos