7.2 Arrays multidimensionales

1. Introducción
2. Declaración
3. Creación
4. Referencia a los elementos del array
5. Asignación de valores
6. El atributo length
7. Utilización del bucle for
8. Comparación de arrays
9. Arrays irregulares

1. Introducción

Los arrays multidimensionales son aquellos que tienen más de una dimensión, es decir, son arrays que contienen otros arrays. Tienen más de un índice, uno por cada dimensión.

Un array con dos dimensiones es un array bidimensional y también se le suele llamar *matriz*. Si además, la matriz tiene el mismo número de filas que de columnas, entonces se llama *matriz cuadrada*.

2. Declaración

Se declaran como los arrays unidimensionales añadiendo tantos corchetes `[ ]` como dimensiones haya. Por ejemplo, veamos las dos formas de declarar un array bidimensional:

- `tipo nombre_array[][];`
- `tipo [][]nombre_array;`

tipo declara el tipo de elemento del array, es decir, el tipo de datos de cada elemento que comprende el array. Dicho tipo de datos puede ser un tipo primitivo o un objeto.

Esta declaración le dice al compilador que dicha variable va a contener un array con elementos de dicho tipo pero todavía no se reserva espacio en la memoria RAM ya que no se conoce el tamaño del mismo.

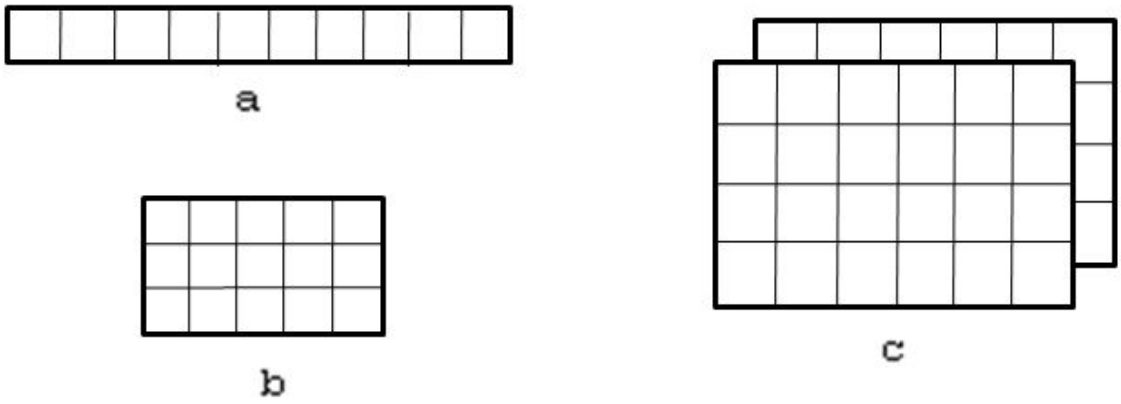
3. Creación

Se realiza con el operador ***new***, que es el que realmente crea el array indicando un tamaño. Cuando se usa new es cuando se reserva el espacio necesario en memoria para el array.

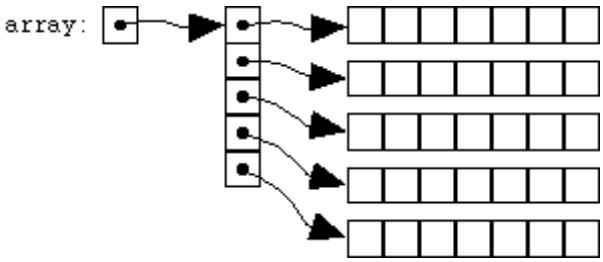
Se crea un array bidimensional de la siguiente manera: `nombre_array = new tipo[tamaño][tamaño];`

Ejemplos:

```
int a[]= new int[10];//Array de una dimensión
int b[][]= new int[3][5];//Array bidimensional (dos dimensiones)
int c[][][]= new int[4][6][2];//Array tridimensional (tres dimensiones)
```



Los arrays multidimensionales son arrays que contienen arrays, por lo tanto un array bidimensional se puede también representar de la siguiente manera:



4. Referencia a los elementos del array

Para referenciar los elementos del array se utiliza el índice de los mismos entre corchetes:

`a[9]` : es el último elemento del array unidimensional *a*.

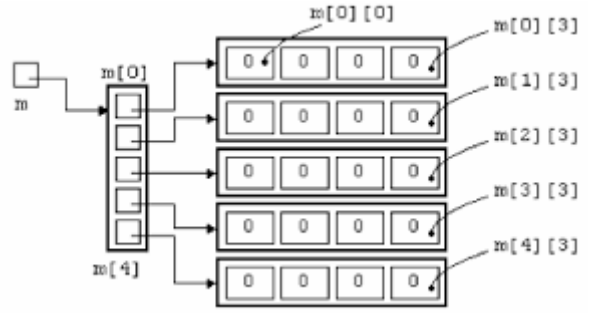
`b[1][2]` : es el elemento que está justo en medio del array bidimensional *b*.

`c[0][0][0]` : es el primer elemento del array tridimensional *c*.

Supongamos que tenemos el siguiente array bidimensional:

```
int m[][]= new int[5][4];
```

Es un array que contiene 5 arrays unidimensionales de 4 posiciones cada uno. Veamos en la imagen la referencia de sus elementos:



- *m*: contiene la referencia al array completo
- *m[0]*: contiene la referencia del primer array unidimensional.
- *m[4]*: contiene la referencia del quinto array unidimensional.
- *m[0] [0]*: contiene el primer elemento del primer array unidimensional.
- *m[0] [3]*: contiene el último elemento del primer array unidimensional.

- *m[4] [3]*: contiene el último elemento del último array unidimensional.

5. Asignación de valores

Se pueden asignar valores a los elementos del array utilizando el signo **=**:

```
a[9] = 8;//Se asigna un 8 al último elemento del array unidimensional a
b[1][2] = 9;//Se asigna un 9 al elemento que está justo en medio del array
bidimensional b
c[0][0][0] = 1;//Se asigna un 1 al primer elemento del array tridimensional c
```

También se pueden asignar valores a todos los elementos del array utilizando ***literales array***:

```
//Array de dos dimensiones [3][5]
int twoDimensions[][]= { {0, 1, 2, 3, 4},
                         {5, 6, 7, 8, 9},
                         {10, 11, 12, 13, 14} };
//Array de tres dimensiones [2][3][2]
int threeDimensions[][][]={ { {0, 1}, {2, 3}, {4, 5} },
                            { {6, 7}, {8, 9}, {10, 11} } };
```

6. El atributo length

Los arrays poseen el atributo ***length*** que contiene el tamaño del array. Como los arrays multidimensionales son arrays de arrays, se puede aplicar a cualquier posición que contenga una referencia. Si se aplica a una posición que contenga un elemento, entonces da error de compilación.

```
package tema7_2_ArraysMultidimensionales;

public class LengthMultidimensional {

    public void showLengthMultidimensional() {

        //Array de dos dimensiones [3][5]
        int twoDimensions[][] = { { 0, 1, 2, 3, 4 }, { 5, 6, 7, 8, 9 }, { 10,
11, 12, 13, 14 } };

        //Array de tres dimensiones [2][3][2]
        int threeDimensions[][][] = { { { 0, 1 }, { 2, 3 }, { 4, 5 } }, { { 6,
7 }, { 8, 9 }, { 10, 11 } } };

        System.out.println(twoDimensions.length);//3
        System.out.println(twoDimensions[0].length);//5
        System.out.println(twoDimensions[1].length);//5
        System.out.println(twoDimensions[2].length);//5

        System.out.println(threeDimensions.length);//2
        System.out.println(threeDimensions[0].length);//3
        System.out.println(threeDimensions[1].length);//3
        System.out.println(threeDimensions[0][0].length);//2
        System.out.println(threeDimensions[1][2].length);//2
        System.out.println(threeDimensions[1][2][0].length); //Error de
compilación

    }

    public static void main(String[] args) {

        new LengthMultidimensional().showLengthMultidimensional();
```

```
    }

}
```

7. Utilización del bucle for

La ventaja de usar arrays es que gracias a un simple bucle for se pueden recorrer fácilmente todos los elementos de un array multidimensional:

```
package tema7_2_ArraysMultidimensionales;

public class ForMultidimensional {

    public void showForMultidimensional() {

        //Array de dos dimensiones [3][5]
        int twoDimensions[][] = { { 0, 1, 2, 3, 4 }, { 5, 6, 7, 8, 9 }, { 10,
11, 12, 13, 14 } };

        //Array de tres dimensiones [2][3][2]
        int threeDimensions[][][] = { { { 0, 1 }, { 2, 3 }, { 4, 5 } }, { { 6,
7 }, { 8, 9 }, { 10, 11 } } };

        for (int i = 0; i < twoDimensions.length; i++) {//Recorrido de arrays
bidimensionales
            for (int j = 0; j < twoDimensions[i].length; j++) {
                System.out.printf("%2d ", twoDimensions[i][j]);
            }
            System.out.println();
        }

        System.out.println();

        for (int i = 0; i < threeDimensions.length; i++) {//Recorrido de
arrays tridimensionales
            for (int j = 0; j < threeDimensions[i].length; j++) {
                for (int k = 0; k < threeDimensions[i][j].length; k++) {
                    System.out.printf("%2d ", threeDimensions[i][j][k]);
                }
                System.out.println();
            }
            System.out.println();
        }

    }

    public static void main(String[] args) {

        new ForMultidimensional().showForMultidimensional();
    }

}
```

También se pueden utilizar los bucles for-each pero solamente con arrays unidimensionales. En un array multidimensional, se puede utilizar un for-each en cualquiera de los arrays unidimensionales que forman parte del array multidimensional.

```
package tema7_2_ArraysMultidimensionales;

public class ArraysForEach {
```

```
    public void showArraysForEach() {

        //Array de dos dimensiones [3][5]
        int twoDimensions[][] = { { 0, 1, 2, 3, 4 }, { 5, 6, 7, 8, 9 }, { 10,
11, 12, 13, 14 } };
        //Recorrido del segundo array unidimensional del array bidimensional:
        for (int number : twoDimensions[1]) {
            System.out.printf("%d ", number);//5 6 7 8 9
        }
        System.out.println();
        for (int i = 0; i < twoDimensions.length; i++) {
            //Recorrido de los arrays unidimensionales que forman el array
bidimensional
            for (int number : twoDimensions[i]) {
                System.out.printf("%d ", number);
            }
            System.out.println();
        }

    }

    public static void main(String[] args) {

        new ArraysForEach().showArraysForEach();

    }

}
```

Para mostrar un array multidimensional, también se puede utilizar el método estático *deepToString* de la clase Arrays. Devuelve una cadena con los elementos del array entre corchetes y separados por comas. Se utiliza con arrays multidimensionales. Para los arrays unidimensionales se usa el método *Arrays.toString*.

```
package tema7_2_ArraysMultidimensionales;

import java.util.Arrays;

public class DeepToString {

    public void showDeepToString() {

        int twoDimensions[][] = { { 0, 1, 2, 3, 4 }, { 5, 6, 7, 8, 9 }, { 10,
11, 12, 13, 14 } };

        System.out.println(Arrays.deepToString(twoDimensions));
        //[[0, 1, 2, 3, 4], [5, 6, 7, 8, 9], [10, 11, 12, 13, 14]]

    }

    public static void main(String[] args) {

        new DeepToString().showDeepToString();

    }

}
```

8. Comparación de arrays

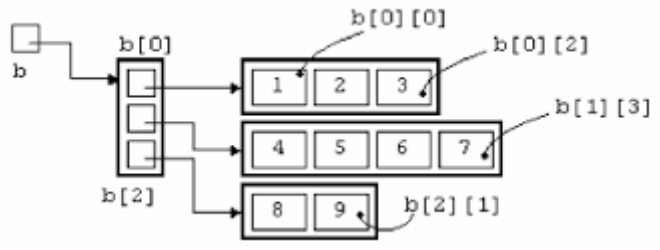
Como en Java los arrays son objetos, la comparación se realiza como los objetos (*Ver el apartado referencias del tema 4 Programación Orientada a Objetos*).

El operador de igualdad **==** cuando se utiliza con arrays, no compara el contenido de los arrays sino sus direcciones de memoria o referencias, es decir, si apuntan al mismo array. Lo mismo ocurre con el método *equals* de los arrays, que compara las direcciones de memoria. Si queremos comparar el contenido de los arrays, tendremos que utilizar el método estático *equals* de la clase *Arrays* para los unidimensionales y el método ***deepEquals*** para los multidimensionales:

```
package tema7_2_ArraysMultidimensionales;

import java.util.Arrays;

public class ComparisonMultidimensional {

    public void showComparisonMultidimensional() {

        int twoDimensions1[][];

        int twoDimensions2[][] = { { 0, 1, 2, 3, 4 }, { 5, 6, 7, 8, 9 }, { 10,
11, 12, 13, 14 } };

        int twoDimensions3[][] = { { 0, 1, 2, 3, 4 }, { 5, 6, 7, 8, 9 }, { 10,
11, 12, 13, 14 } };

        twoDimensions1 = twoDimensions2;

        System.out.println(twoDimensions1 == twoDimensions2);//true porque
apuntan al mismo array
        System.out.println(twoDimensions2 == twoDimensions3);//false porque no
apuntan al mismo array
        System.out.println(twoDimensions1.equals(twoDimensions2));//true
porque apuntan al mismo array
        System.out.println(twoDimensions2.equals(twoDimensions3));//false
porque no apuntan al mismo array
        System.out.println(Arrays.deepEquals(twoDimensions1,
twoDimensions2));//true porque el contenido es el mismo ya que apuntan al
mismo array
        System.out.println(Arrays.deepEquals(twoDimensions2,
twoDimensions3));//true porque el contenido es el mismo
        System.out.println(Arrays.equals(twoDimensions2,
twoDimensions3));//false porque para arrays multidimensionales es DeepEquals
en lugar de equals

    }

    public static void main(String[] args) {

        new ComparisonMultidimensional().showComparisonMultidimensional();

    }

}
```

9. Arrays irregulares

Los arrays unidimensionales que forman un array multidimensional no tienen por qué tener todos el mismo tamaño.




```java
package tema7_2_ArraysMultidimensionales;

public class IrregularArrays {

    public void showIrregularArrays() {

        int b[][] = { { 1, 2, 3 }, { 4, 5, 6, 7 }, { 8, 9 } };

        for (int i = 0; i < b.length; i++) {
            for (int j = 0; j < b[i].length; j++) {
                System.out.printf("%2d ", b[i][j]);
            }
            System.out.println();
        }

    }

    public static void main(String[] args) {

        new IrregularArrays().showIrregularArrays();

    }

}
```