Programación II

Arrays

1 Los arrays

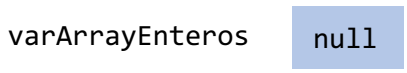
1.1 Declaración

Un array, como en C, se usa para almacenar de forma conjunta una serie de datos del mismo tipo, ya sea uno de los tipos básicos de Java (por ejemplo, `int`), objetos de una clase del sistema (`String`), u objetos de una clase defina por el programador.

Un array, al igual que en el caso de `String`, es una variable que almacena una referencia a un objeto. Por tanto, la información propiamente dicha sobre los componentes del array está en ese objeto, mientras que la variable sólo almacena una referencia al mismo. Es decir, cuando declaramos un array de cualquiera de las dos siguientes maneras:

```
int[] varArrayEnteros;   // preferida en Java
int varArrayEnteros[];   // al estilo de C
```

en realidad, estamos solamente reservando espacio para una variable que va a almacenar la dirección (referencia) del objeto array. Como no le hemos dado ningún valor, supondremos que inicialmente vale `null`.

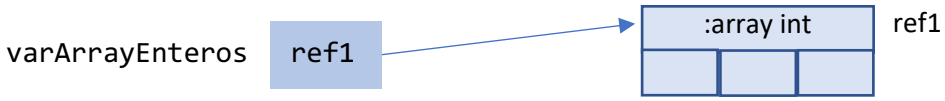


Como vemos, la variable `varArrayEnteros` tiene valor `null`, es decir, no apunta a ningún objeto array, ya que no hay todavía tal objeto. Hemos declarado y creado la variable que va a contener la referencia a un array de enteros, pero todavía no existe tal array, ni, por supuesto, se sabe nada de su tamaño.

1.2 Creación y asignación de valores al array

Para crear el array, dado que es un objeto, recurrimos al constructor new. Por ejemplo:

```
varArrayEnteros = new int[3];  // creamos un array de 3 enteros
```



Ahora vemos que se ha creado el objeto array con espacio para almacenar 3 enteros, aunque todavía no hemos almacenado ninguno[1].

Como en C, un array en Java tiene un tamaño fijo, definido en el momento de su creación, y que no es posible variar. El objeto que está en `ref1` tiene y tendrá siempre espacio para 3 enteros, no es posible cambiar su tamaño.

Y ahora que ya hemos creado el objeto array, ya podemos asignarles valores a los componentes del array, con la sintaxis tradicional:

```
varArrayEnteros[0] = 23;
varArrayEnteros[1] = 45;
varArrayEnteros[2] = 67;
```

[1] En realidad, sí que hay un valor. Al crear un array, si no especificamos su contenido, los arrays de números se inicializarán con ceros, los de `boolean` con `false`, y los de `char` con el carácter Unicode de 16 bits '\0000'.

Aunque también podemos asignarle valores en la sentencia de creación. Las siguientes dos sentencias tienen el mismo efecto que el código anterior:

```
int[] varArrayEnteros;
varArrayEnteros = new int[3]{23,45,67};
```

O podemos hacerlo todo a la vez, durante la declaración. Esta sentencia tiene el mismo efecto:

```
int[] varArrayEnteros = new int[3]{23,45,67};
```

Incluso podemos emplear una sintaxis abreviada sin el operador new (sólo válida en el momento de la declaración de la variable). En la siguiente sentencia, equivalente a las anteriores, el compilador entiende perfectamente que debe crear un objeto array de enteros con espacio para 3 elementos, y los valores especificados.

```
int[] varArrayEnteros = {23,45,67};  // forma abreviada sólo válida durante la declaración
```

Es interesante destacar que la variable `varArrayEnteros` guarda una referencia a un array de enteros. No a un array de 3 enteros, sino a un array de enteros en general. Es decir, a la misma variable podríamos asignarle la referencia a un objeto array de 5 enteros:

```
int[] varArrayEnteros = {23,45,67}; // varArrayEnteros apunta a un array de tamaño 3

int[] otroArray = {1,2,3,4,5};  // otroArray apunta a un array de tamaño 5

varArrayEnteros = otroArray; // varArrayEnteros apunta ahora al array de tamaño 5
```

1.3 Acceso a los componentes del array

Una vez creado el array, se puede conocer su longitud mediante su propiedad `length`:

```
int[] varArrayEnteros = {23,45,67};
varArrayEnteros.length;  // sería el número 3
```

Y se puede acceder a sus componentes mediante la sintaxis tradicional:

```
varArrayEnteros[1]; // sería el segundo componente del array, en este caso el número 45
```

Se puede recorrer todos sus miembros con un bucle `for` con la sintaxis habitual que conocemos de C:

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

Pero también podemos usar una sintaxis alternativa del `for` más simple (conocida como *enhanced for*):

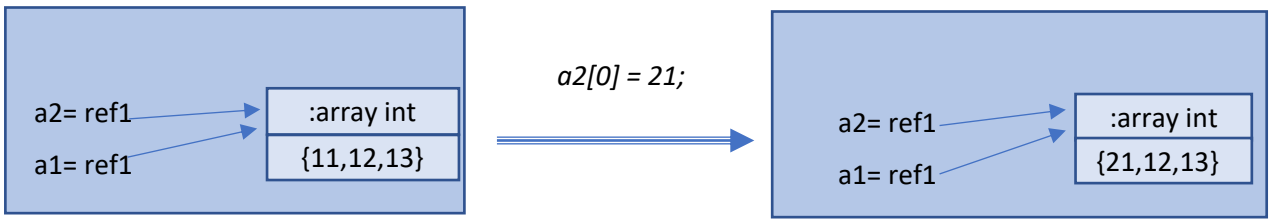
```
for (int var: varArrayEnteros)
      System.out.println(var);  // la variable var recorre todos los valores del array
```

1.4 Paso de un array a un método

Cuando se le pasa un array a un método, lo que realmente se le pasa es una referencia al objeto que almacena el contenido del array. Por ello, cualquier modificación de los valores del array en el método implica una modificación del array original (el único que existe).

```
int[] a1 = {11,12,13} ;
objeto.procesar(a1);
// ahora a1[0] es 21, no 11
```

```
public void procesar (int[] a2) {
            a2[0] = 21;
}
```



De la misma manera, un método puede devolver un array, y lo que devolverá será la referencia al array. Ese array puede haber sido creado dentro del método, y, por tanto, se habrá creado un objeto en la memoria que no desaparecerá tras terminar el método. La referencia que devuelve el método puede ser asignada a un array declarado en el programa llamante, sin que haya que crearlo ahí. Por ejemplo:

```
public class MiArray {

	public static void main (String[] args) {
		int[] miarray; // declaramos un array, pero no lo creamos

		miarray = crearArray();
		System.out.println(miarray[0]);  // imprimirá 51
	}

	public static int[] crearArray () {
		int[] a = {51,52,53};  // declaramos y creamos un objeto array
		int b=1;
		return a;  // devolvemos la referencia al array creado
		// las variables a y b desaparecen, pero el objeto array permanece
	}
}
```

2 Recursos útiles para trabajar con arrays

2.1.1 System.arraycopy()

Hay un método estático de la clase `System` llamado `arraycopy()` que permite copiar un conjunto de componentes de un array a otro:

```
public static void arraycopy(Object src, int sPos, Object dest, int dPos, int clength)
```

- `src` es el array origen y `dest` es el array destino
- `sPos` es la primera posición que se copia, y `dPos` es la primera posición a la que se copia
- `clength` es la cantidad de componentes que copian

Así:

- `System.arraycopy(s, 0, d, 0, s.length);` // copia todo el array `s` al array `d`
- `System.arraycopy(s, 2, d, 7, 3);` // copia `s[2]`,`s[3]` y `s[4]` a `d[7]`,`d[8]` y `d[9]`

2.1.2 java.util.Arrays

La clase `java.util.Arrays` tiene una serie de métodos de gran utilidad:

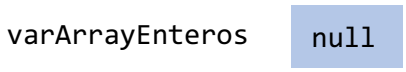
- `public static boolean equals(Object src1, Object src2);`
  Devuelve `true` si ambos arrays (`src1` y `src2`) son iguales componente a componente.
- `public static void toString(Object src);`
  Devuelve una cadena con una representación del array `src`.
- `public static void fill(Object src, type val);`
  Inicializa todos los componentes del array `src` al valor `val` (del tipo `type`).
- Otros métodos para buscar, ordenar…

3 Arrays multidimensionales

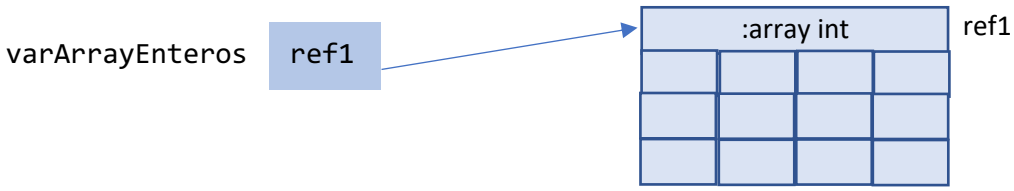
Igual que en C, en Java es posible trabajar con arrays multidimensionales. Veamos un ejemplo con arrays bidimensionales.

- `int[][] varArrayEnteros;`

  declara un array bidimensional de enteros, pero no crea el objeto array, aún no podemos almacenar nada, dado que no hay objeto array



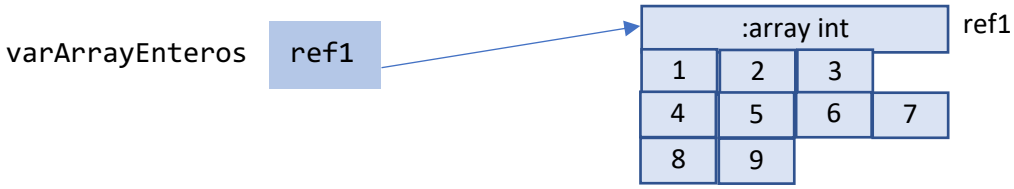
- `int[][] varArrayEnteros = new int[3][4];` // crea un array de 3 filas por 4 columnas



En Java, al contrario que en C, tenemos la ventaja de que cada dimensión puede tener un número distinto de componentes.

- `int[][] varArrayEnteros = {{1,2,3},{4,5,6,7},{8,9}};` // array de 3 filas

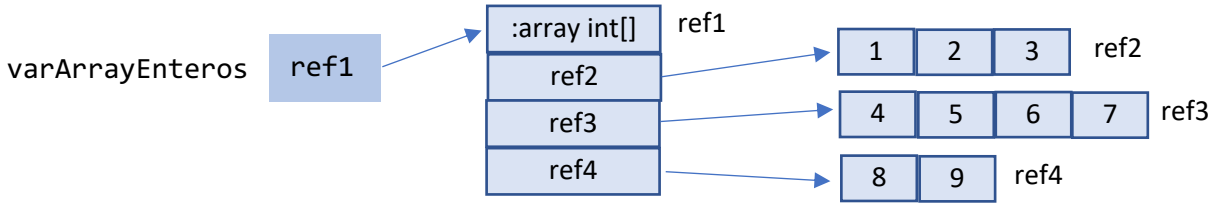
  En este último caso, se declara un array de 3 filas, con 3, 4 y 2 columnas respectivamente.



Es interesante entender que `varArrayEnteros[2][1]` es una variable de tipo `int`, que almacena el valor 9 (recordar que los índices comienzan en 0).

De la misma manera, `varArrayEnteros[2]` (cámbiese el 2 por 0, o por 1, si se quiere) es una variable que apunta a un array de enteros, en este caso el objeto array {8,9}.

En realidad, la forma más apropiada de verlo es la siguiente:



Es decir, `varArrayEnteros[0]` guarda en realidad `ref2`, que apunta a un objeto array de enteros.

Para averiguar el tamaño del array de cada fila se emplearía la propiedad `length`. Por ejemplo.

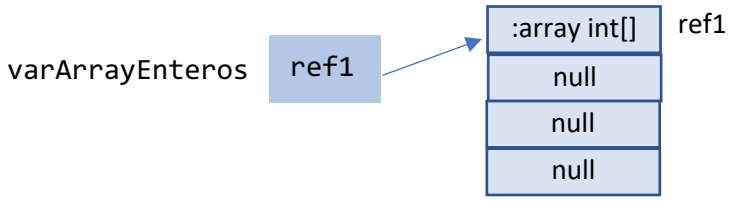
`varArrayEnteros[1].length;` // longitud del segundo array, es decir, 4

3.1.1 Creación de arrays multidimensionales paso a paso

En Java es posible ir creando los arrays de cada fila de forma independiente sobre la marcha, según vayamos sabiendo la longitud de cada fila. Veamos un ejemplo.

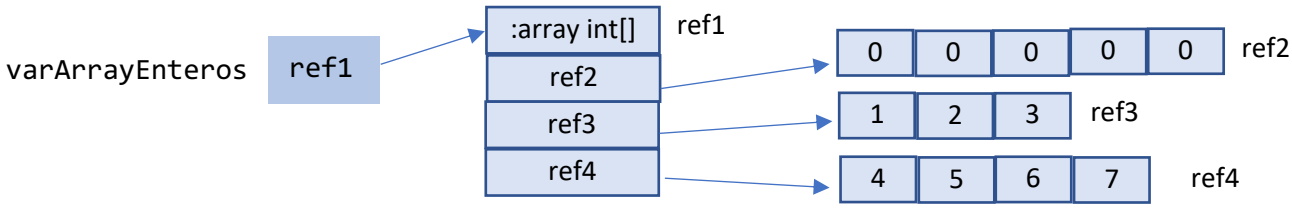
```
int[][]  varArrayEnteros = new int[3][];
```

Con esta sentencia estamos creando un array de 3 filas, pero no estamos diciendo nada sobre el número de columnas de cada fila, no hemos creado todavía las columnas. Así pues, lo que tenemos es:



Tenemos el espacio para guardar las referencias a cada array, pero aún no hemos creado esos arrays. Podemos hacerlo uno a uno.

```
varArrayEnteros[0] = new int[5]; // la primera fila tiene 5 columnas
varArrayEnteros[1] = new int[]{1,2,3}; // la segunda fila tiene 3 columnas
varArrayEnteros[2] = new int[]{4,5,6,7}; // la tercera fila tiene 4 columnas
```



3.1.2 Recorrer un array multidimensional

Para recorrer los valores del array podemos recurrir a un bucle `for` tradicional:

```
int[][] varArrayEnteros = {{1,2,3},{4,5,6,7}, {8,9}};
for (int x=0; x < varArrayEnteros.length; x++) { // 'x' recorre las filas
      for (int y=0; y < varArrayEnteros[x].length; y++) { // 'y' las columnas
            System.out.print(varArrayEnteros[x][y]+" ");
      }
      System.out.println();  // un retorno de carro tras cada fila
}
```

O también podemos recurrir al bucle `for` mejorado mencionado anteriormente:

```
for (int[] var1: varArrayEnteros) {

      // var1 irá tomando como valor las referencias a los arrays de cada dimensión
      for (int var2: var1)  // var2 será cada entero del array en curso (var1)
            System.out.print(var2+" ");

      System.out.println();
}
```