# 6.1 Arrays unidimensionales



## 1. Introducción

Los tipos de datos que conocemos hasta ahora no permiten solucionar problemas que requieren gestionar muchos datos a la vez. Por ejemplo, imaginemos que deseamos almacenar las notas de una clase de 25 alumnos, para ello, con lo que conocemos hasta ahora, no habrá más remedio que declarar 25 variables. Eso es tremendamente pesado de programar. Manejar esos datos significaría estar continuamente manejando 25 variables. Por ello, en casi todos los lenguajes se pueden agrupar una serie de variables del mismo tipo en una misma estructura que comúnmente se conoce como array. Esa estructura permite referirnos a todos los elementos pero también nos permite acceder individualmente a cada uno de ellos.

Los ***arrays*** son una colección de datos del mismo tipo al que se le pone un nombre. Para acceder a un dato individual de la colección hay que utilizar su posición. La posición es un número entero, normalmente se le llama ***índice***. Hay que tener en cuenta que en los arrays el primer elemento tiene como índice el número cero, el segundo el uno y así sucesivamente.

Un array unidimensional es aquel que tiene una única dimensión, es decir, un único índice. También se le suele llamar *vector*.

## 2. Declaración

Existen dos formas de declarar un array unidimensional:

- `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 de la siguiente manera: `nombre_array = new tipo[tamaño];`

Ejemplo:

```
int[] grades;//declaración del array de enteros llamado grades
grades = new int[3];//creación del array grades reservando en memoria espacio
para 3 enteros
```

Se puede realizar lo mismo en una única línea de código:

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

## 4. Referencia a los elementos del array

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

`grades[0]` : es el primer elemento del array, es decir, la primera nota.

`grades[1]` : es el segundo elemento del array, es decir, la segunda nota.

`grades[2]` : es el tercer elemento del array, es decir, la tercera nota.

## 5. Asignación de valores

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

```
grades[2] = 8;//Se asigna un 8 a la tercera nota
```

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

```
int[] grades = new int[] {8, 7, 9};//{8, 7, 9} es un literal array
```

No es necesario escribir ***new int[]*** en las últimas versiones de Java:

```
int[] grades = {8, 7, 9};
```

En este caso, se está asignando un 8 a la primera nota, un 7 a la segunda y un 9 a la tercera.

## 6. El atributo length

Los arrays poseen el atributo ***length*** que contiene el tamaño del array:

```
System.out.println(grades.length);//Muestra 3
```

## 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:

```
package tema6_1_ArraysUnidimensionales;

public class ArraysFor {

    public void show() {

        int[] grades = { 8, 7, 9 };

        for (int i = 0; i < grades.length; i++) {
            System.out.printf("Nota en el índice %d: %d\n", i, grades[i]);
        }

    }

    public static void main(String[] args) {

        new ArraysFor().show();

    }

}
```

Los índices del array van desde 0 hasta length-1. En el ejemplo, el array es de tamaño 3 y sus índices son 0, 1 y 2. Por eso en la condición del for se itera mientras i sea menor que *grades.length*, es decir, el último índice que se procesa es el 2. Si pusiéramos la condición del for como menor o igual, nos lanzaría la excepción ***ArrayIndexOutOfBoundsException*** ya que estaríamos accediendo al array con el índice 3, el cual es un índice no válido:

```
for(int i=0;i<=grades.length;i++) {//Se lanza la excepción
ArrayIndexOutOfBoundsException
```

También se pueden utilizar con los arrays los bucles for-each:

```
package tema6_1_ArraysUnidimensionales;

public class ArraysForEach {

    public void show() {

        int[] grades = { 8, 7, 9 };

        for (int grade : grades) {
            System.out.printf("Nota: %d\n", grade);
        }

    }
```

```
    public static void main(String[] args) {

        new ArraysForEach().show();

    }

}
```

## 8. Método split

Las clases *String* y *Pattern* poseen el método ***split*** que divide una cadena en subcadenas en función de una expresión regular y devuelve un array con las subcadenas resultantes.

```
package tema6_1_ArraysUnidimensionales;

import java.util.Arrays;

public class Split {

    public void show() {

        String string = "Esto:es:una:cadena:dividida:por:split";
        String[] stringArray = string.split(":");
        System.out.println(Arrays.toString(stringArray));//[Esto, es, una,
cadena, dividida, por, split]

    }

    public static void main(String[] args) {

        new Split().show();

    }

}
```

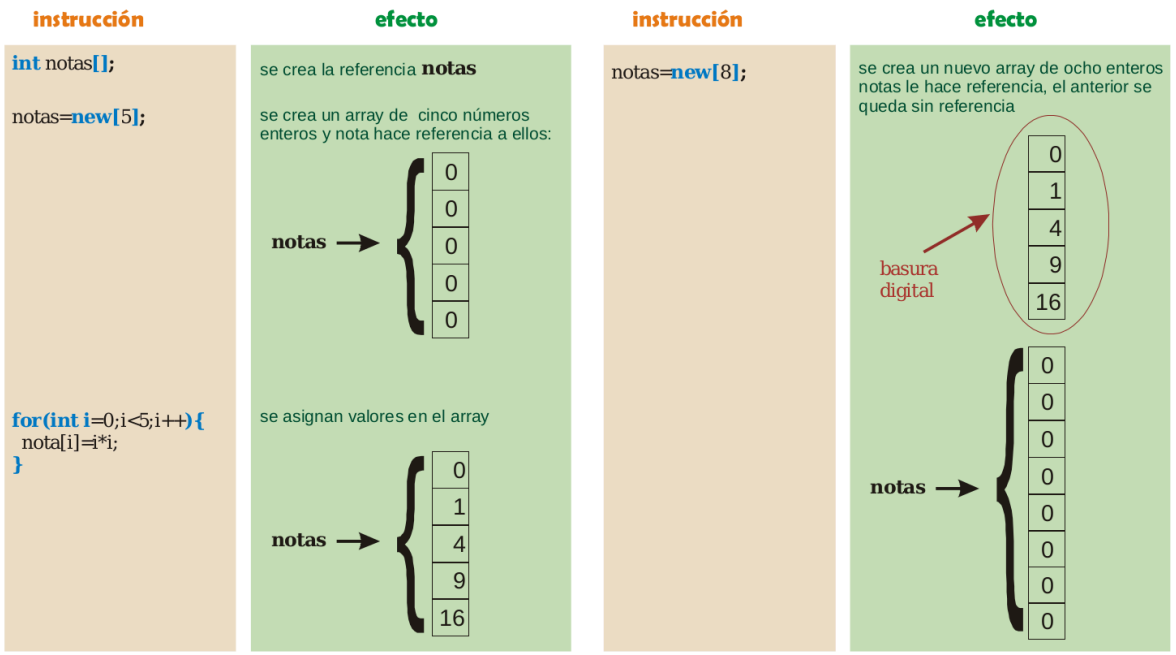
Fíjense en el método estático *toString* de la clase *Arrays*. Devuelve una cadena con los elementos del array entre corchetes y separados por comas.

## 9. Los arrays se implementan como objetos

En Java, los arrays se implementan como objetos. Una de las ventajas que tiene esto es que los arrays que pierden la referencia pueden ser recolectados como basura. (Ver *Tema 4  Programación Orientada a Objetos 13 Referencias*).

En la perspectiva de Java, un array es una referencia a una serie de valores que se almacenan en la memoria. El operador *new* en realidad lo que hace es devolver dicha referencia para poder leer y escribir en esos valores.

Veamos el efecto del uso del operador new en los arrays y cómo afectan en la memoria:




Vemos como el anterior array se ha quedado sin referencia y se marca como elegible para el recolector de basura.

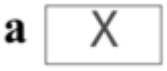
# 10. Qué puede contener un array

El  array puede contener objetos o tipos de datos primitivos. En el caso de los tipos de datos primitivos, los valores reales se almacenan en ubicaciones de memoria contigua. En el caso de los objetos de una clase, los objetos reales se almacenan en *heap*.

## 10.1 Arrays de tipos primitivos

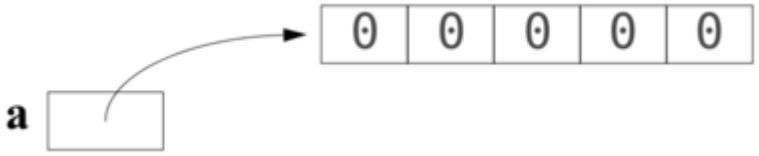
- Crear una variable de tipo array sólo crea una referencia:

```
int[] a;
```




- El espacio en memoria para el array se crea con *new*. Cuando se trata de un array de un tipo primitivo, se crean los elementos del array y se inicializan a sus valores por defecto:

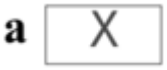
```
a = new int[5];
```




## 10.2 Arrays de objetos

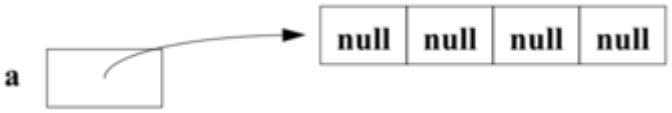
- Crear una variable de tipo array de objetos sólo crea la referencia:

```
Complejo[] a;
```

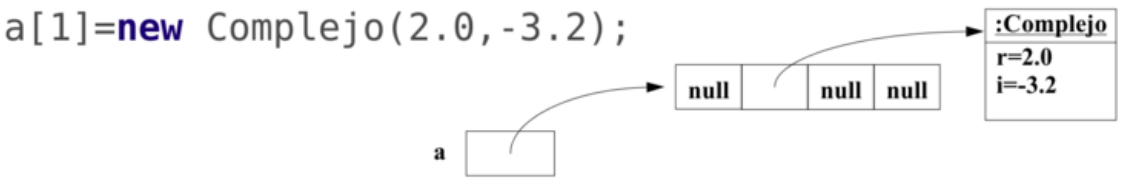



- Con new se reserva el espacio para el array de referencias pero no los objetos a los que apuntarán las referencias. Las referencias se inicializan a su valor por defecto (*null*):

```
a = new Complejo[4];
```




- Los objetos hay que crearlos posteriormente:

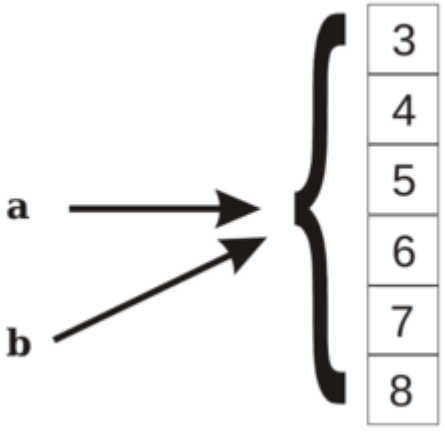



## 11. Asignación de arrays

Como en Java los arrays son objetos, la asignación se realiza como los objetos (Ver *Tema 4 Programación Orientada a Objetos 13 Referencias*).

Un array se puede asignar a otro array (si son del mismo tipo). La asignación solo copia referencias, es decir, la dirección de memoria. Por lo tanto, ambos arrays apuntarán al mismo sitio:

```
int[] a;
int[] b={3,4,5,6,7,8};
a=b;
```




Esta asignación provoca que cualquier cambio en *a* también cambie el array *b* ya que de hecho, son el mismo array:

```
package tema6_1_ArraysUnidimensionales;

import java.util.Arrays;

public class Assignment {
```

```
    public void show() {

        int[] a;
        int[] b = { 3, 4, 5, 6, 7, 8 };
        a = b;
        System.out.println(Arrays.toString(b));//[3, 4, 5, 6, 7, 8]
        a[0] = 1;
        System.out.println(Arrays.toString(b));//[1, 4, 5, 6, 7, 8]

    }

    public static void main(String[] args) {

        new Assignment().show();

    }

}
```

## 12. Comparación de arrays

Como en Java los arrays son objetos, la comparación se realiza como los objetos (Ver *Tema 4 Programación Orientada a Objetos 13 Referencias*).

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*:

```
package tema6_1_ArraysUnidimensionales;

import java.util.Arrays;

public class Comparison {

    public void show() {

        int[] array1;
        int[] array2 = { 3, 4, 5, 6, 7, 8 };
        int[] array3 = { 3, 4, 5, 6, 7, 8 };
        array1 = array2;
        System.out.println(array1 == array2);//true porque apuntan al mismo array
        System.out.println(array2 == array3);//false porque no apuntan al mismo
array
        System.out.println(array1.equals(array2));//true porque apuntan al mismo
array
        System.out.println(array2.equals(array3));//false porque no apuntan al
mismo array
        System.out.println(Arrays.equals(array1, array2));//true porque el
contenido es el mismo ya que apuntan al mismo array
        System.out.println(Arrays.equals(array2, array3));//true porque el
contenido es el mismo

    }
```

```
    public static void main(String[] args) {

        new Comparison().show();

    }

}
```

## 13. Arrays como parámetros

Al igual que las variables, también podemos pasar arrays a los métodos, es decir, se pueden definir parámetros de tipo array. En la llamada al método, lo que el método recibe como argumento es la dirección de memoria del array, por lo que si el método modifica el array, el array del método que efectúa la llamada también se ve afectado por dichos cambios.

```
package tema6_1_ArraysUnidimensionales;

import java.util.Arrays;

public class ArraysAsParameters {

    public void show() {

        int[] array = { 3, 4, 5, 6, 7, 8 };
        System.out.println(Arrays.toString(array));//[3, 4, 5, 6, 7, 8]
        method(array);
        System.out.println(Arrays.toString(array));//[6, 8, 10, 12, 14, 16]

    }

    public void method(int[] array) {

        for (int i = 0; i < array.length; i++) {
            array[i] *= 2;
        }

    }

    public static void main(String[] args) {

        new ArraysAsParameters().show();

    }

}
```

## 14. Cómo retornar un array

Un método también puede devolver un array, en cuyo caso, lo que retorna es la dirección de memoria de dicho array.

```
package tema6_1_ArraysUnidimensionales;
```

```
import java.util.Arrays;

public class ReturningAnArray {

    public void show() {

        int[] a;
        a = method();
        System.out.println(Arrays.toString(a));//[3, 4, 5, 6, 7, 8]

    }

    public int[] method() {

        int[] array = { 3, 4, 5, 6, 7, 8 };

        return array;

    }

    public static void main(String[] args) {

        new ReturningAnArray().show();

    }

}
```

En este caso, el método *show* define una variable *a* de tipo array de enteros pero no hace el new del array, sino que recibe la dirección de memoria que le devuelve el método *method*. Sin embargo, el método *method* sí que hace el new del array porque su responsabilidad es crear el array, darle valores y devolverlo.

## 15. Excepción ArrayIndexOutOfBoundsException

Cuando se intenta acceder a un elemento del array que no existe, java nos lanza la excepción *ArrayIndexOutOfBoundsException*. El programador no debe capturar con un try-catch la excepción sino que debe corregir el error de programación que ha producido dicha excepción. Veamos un ejemplo:

```
package tema6_1_ArraysUnidimensionales;

public class ArrayIndexOutOfBoundsException1 {

    public void show() {

        int[] grades = { 8, 7, 9 };

        for (int i = 0; i <= grades.length; i++) {
            System.out.printf("Nota en el índice %d: %d\n", i, grades[i]);
        }

    }
```

```
    public static void main(String[] args) {

        new ArrayIndexOutOfBoundsException1().show();

    }

}
```

La salida por consola es la siguiente:

```
Nota en el índice 0: 8
Nota en el índice 1: 7
Nota en el índice 2: 9
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index 3 out
of bounds for length 3
    at
tema6_1_ArraysUnidimensionales.ArrayIndexOutOfBoundsException1.show(ArrayIndexOut
OfBoundsException1.java:10)
    at
tema6_1_ArraysUnidimensionales.ArrayIndexOutOfBoundsException1.main(ArrayIndexOut
OfBoundsException1.java:17)
```

El programador no debe hacer ésto:

```
package tema6_1_ArraysUnidimensionales;

public class ArrayIndexOutOfBoundsException2 {

    public void show() {

        int[] grades = { 8, 7, 9 };

        try {
            for (int i = 0; i <= grades.length; i++) {
                System.out.printf("Nota en el índice %d: %d\n", i, grades[i]);
            }
        } catch (ArrayIndexOutOfBoundsException e) {//Esto no se debe hacer
            System.err.println("Esto no se debe hacer");
        }

    }

    public static void main(String[] args) {

        new ArrayIndexOutOfBoundsException2().show();

    }

}
```

Sino que debe corregir el error, es decir, quitar el = de la condición del for, ya que en la última iteración, *i* toma el valor de *grades.length* que es 3, cuando *grades[3]* no es un elemento válido ya que los elementos válidos van de 0 a 2. Por lo tanto, el programador debe corregir el error:

```java
package tema6_1_ArraysUnidimensionales;

public class ArrayIndexOutOfBoundsException3 {

    public void show() {

        int[] grades = { 8, 7, 9 };

        for (int i = 0; i < grades.length; i++) {
            System.out.printf("Nota en el índice %d: %d\n", i, grades[i]);
        }

    }

    public static void main(String[] args) {

        new ArrayIndexOutOfBoundsException3().show();

    }

}
```