

Imaginemos que queremos guardar muchos valores en una sola variable, aquí es donde entran los *arrays*, y tienen una gran cantidad de usos y aplicaciones, es el tema (para mí) más importante de Programación 1, y conviene entenderlo bien, ya que en el examen final seguramente todo se base en *arrays* (listas indexadas), imaginemos que tenemos 3 variables que queremos meterlas en una sola:

```
static int variable1 = 5;
static int variable2 = 7;
static int variable3 = 2;
```

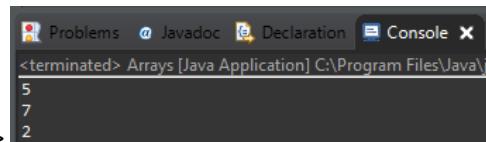
Para esto podemos crear un *array* que contenga éstos datos así:

```
static int[] variables = {5,7,2};
```

Indicamos que es un *array* poniendo `[]` antes o después del nombre de la variable, en este caso *variables*, es indiferente, aunque yo siempre se lo pongo antes, ahora después del `=` abrimos llaves `{` y enumeramos los datos, tienen que coincidir con el tipo que le hemos dado al *array*, es decir, si ponemos *boolean[]* todos los datos deben ser *true* o *false*, si ponemos *char[]* todos deben ser *chars*, ya me entendéis, los datos se enumeran separados por comas, cerramos llaves `}`, y ya está, tenemos un *array* de tamaño 3, ahora, ¿Cómo nos referimos a los elementos de nuestro *array*?, pues bien, el primer elemento, en este caso el 5, será el 0, el 7 será el elemento 1, y el 2 → el 2, es decir, si tenemos un *array* de 500 elementos se enumeran desde el 0 hasta el 499, dicho esto vamos a imprimir un éstos elementos por pantalla:

```
System.out.println(variables[0]);
System.out.println(variables[1]);
System.out.println(variables[2]);
```

>>>

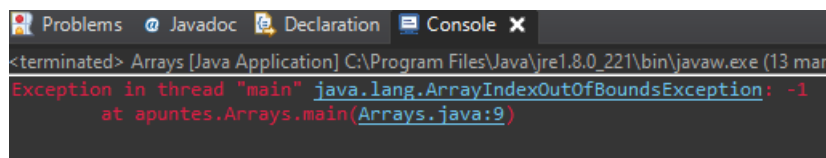


Problems Javadoc Declaration Console X
<terminated> Arrays [Java Application] C:\Program Files\Java\j...
5
7
2

Para referirnos al elemento 0 de la variable *variables*, ponemos el nombre de la variable y luego entre corchetes el número del elemento, si nos salimos de rango pasará esto:

```
9 System.out.println(variables[-1]);
```

>>>



Problems Javadoc Declaration Console X
<terminated> Arrays [Java Application] C:\Program Files\Java\jre1.8.0_221\bin\javaw.exe (13 mar
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: -1
at apuntes.Arrays.main(Arrays.java:9)

Como veis nos dice que nos hemos salido de rango en un *array*, lo que viene en azul, a la derecha vemos el número que hemos tratado de invocar, en este caso el -1, y luego abajo en azul nos dice la línea en la que ha ocurrido esto, la 9, de tal forma que sabemos que en la línea 9 se nos va de rango, ahora es fácil ver cuál es el error, pero más adelante tal vez no tenemos un número, ni una variable, tal vez tenemos una fórmula que se aplica 5000 veces en un *array*, y os aseguro que va a ser complicado encontrar el fallo, para que se salga de rango de un *array* de tamaño *n*, el número que tratamos de invocar debe ser menor que 0, o mayor o igual que *n*.

Hay otra forma de crear *arrays*, la que hemos hecho es por enumeración, diciendo que elementos hay exactamente y el propio programa calcula el tamaño, ahora vamos a ver como solo decirle el tamaño:

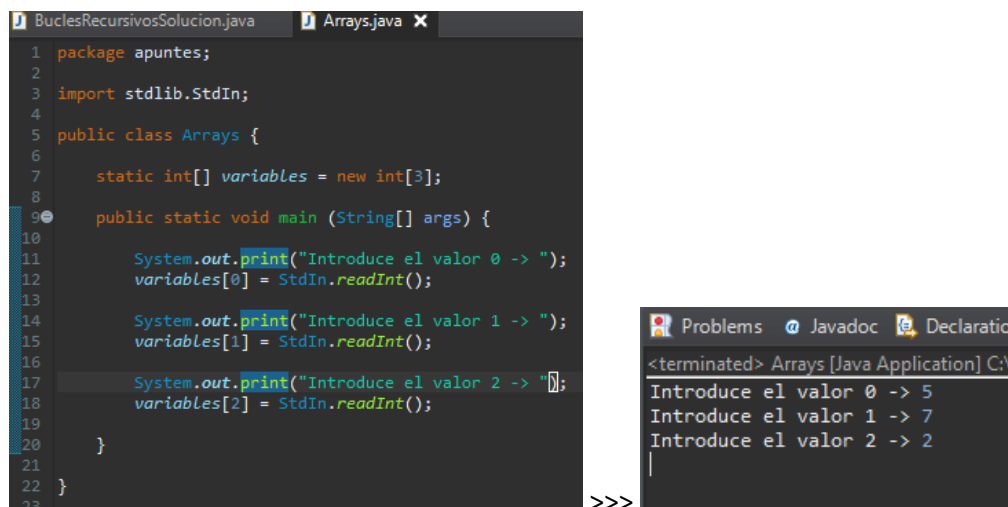
```
static int[] variables = new int[3];
```

La primera parte es igual, pero después del =, vamos a poner *new TIPO_DE_DATO[TAMAÑO]*, es decir, si fuese un *array* de *booleans*, pondría *new boolean[TAMAÑO]*, dentro de los corchetes ponemos el tamaño de nuestro *array*, cuando creamos un *array* de ésta forma los elementos que lo contienen van a tener un valor predefinido, es decir, si yo no le asigno nada a *variables[0]* cuando trate de mostrarla en pantalla me imprimirá un 0, ya que rellena el *array* con ceros, aquí os dejo los valores predefinidos:

-int → 0	-double → 0.0	-boolean → false
-Double → null	-char → "	-String → null

Cuando creamos una variable de cualquier tipo también se le asigna el valor predefinido, es decir, si yo pongo *static int x;* e imprimo *x*, me va a mostrar un 0.

Ahora vamos a convertir nuestra variable *variables* a la que solo se le ha asignado el tamaño 3 números introducidos por teclado:



The screenshot shows an IDE with two tabs: 'BuclesRecursivosSolucion.java' and 'Arrays.java'. The 'Arrays.java' tab is active, displaying the following code:

```
1 package apuntes;
2
3 import stdlib.StdIn;
4
5 public class Arrays {
6
7     static int[] variables = new int[3];
8
9     public static void main (String[] args) {
10
11         System.out.print("Introduce el valor 0 -> ");
12         variables[0] = StdIn.readInt();
13
14         System.out.print("Introduce el valor 1 -> ");
15         variables[1] = StdIn.readInt();
16
17         System.out.print("Introduce el valor 2 -> ");
18         variables[2] = StdIn.readInt();
19
20     }
21 }
22
23
```

To the right of the code editor, the 'Problems' tab is open, showing the execution output of the 'Arrays [Java Application]' program:

```
<terminated> Arrays [Java Application] C:\V
Introduce el valor 0 -> 5
Introduce el valor 1 -> 7
Introduce el valor 2 -> 2
```

Como veis si tuviésemos que introducir 100 datos, ya no solo el hecho de introducirlos sería tedioso, sino también escribir el código, y por eso dije que los bucles *for* eran tan importantes en el manejo de *arrays*, veamos un ejemplo:



The screenshot shows an IDE with two tabs: 'BuclesRecursivosSolucion.java' and 'Arrays.java'. The 'Arrays.java' tab is active, displaying the following code:

```
5 public class Arrays {
6
7     static int[] variables = new int[100];
8
9     public static void main (String[] args) {
10
11         for (int i = 0; i < variables.length; i++) {
12
13             System.out.print("Introduce el valor " + i + " -> ");
14             variables[i] = StdIn.readInt();
15
16         }
17
18     }
19 }
20
```

To the right of the code editor, the 'Problems' tab is open, showing the execution output of the 'Arrays [Java Application]' program:

```
Arrays [Java Application] C:\Program Fi
Introduce el valor 0 -> 5
Introduce el valor 1 -> 7
Introduce el valor 2 -> 2
Introduce el valor 3 -> 5
Introduce el valor 4 -> 7
Introduce el valor 5 -> 9
Introduce el valor 6 -> 0
Introduce el valor 7 -> 5
Introduce el valor 8 ->
```

En la línea 7 he cambiado el tamaño de 3 a 100, ahora vemos más en profundidad el bucle *for*, vamos a inicializar *i* en 0, igual que los elementos de nuestro *array*, ahora mientras *i < variables.length*, se ejecuta el bucle, si escribimos el nombre de nuestra variable, en este caso *variables*, seguido del método *.length* nos devolverá el tamaño de nuestro *array*, en este caso el 100, por ello podemos decir que para *i = 0*, mientras *i < 100*, con paso *i+1* se ejecutan las líneas entre la 12 y la 15, en la línea 13 como veis hago que se escriba el valor *i*, ya que de ésta forma podemos verlo en la salida (valor 0,1,2, ... , n-1), la más interesante es la línea 14, en ella veis como pongo *variables[i]*, mientras lo que haya entre los corchetes sea un número entero, podemos hacer lo que queramos ahí dentro, multiplicar, restar, realizar el módulo...

Por último, voy a realizar un último ejercicio, donde le voy a asignar a cada elemento su propio índice:

```
1 package apuntes;
2
3 import stdlib.StdIn;
4
5 public class Arrays {
6
7     static int[] variables = new int[100];
8
9     public static void main (String[] args) {
10
11         for (int i = 0; i < variables.length; i++) {
12             variables[i] = i;
13         }
14     }
15 }
16
17
18
19 }
```

El índice de un elemento, sería el número que ponemos entre los corchetes, por ejemplo, en el primer *array* que creamos →

```
static int[] variables = {5,7,2};
```

En el índice 0 tenemos el elemento 5, en el índice 1 el elemento 7 y en índice 2 el elemento 2, esto no es más que un poco de vocabulario.

He de mencionar que el tamaño de un *array* también puede ser 0, y se puede representar de dos formas:

- static int[] nums = new int[0];
- static int[] nums = {};

He de mencionar que si tenemos una función que tiene que buscar un valor dentro del *array*, tenemos que poner una variable de tipo *boolean* para que se salga del bucle *for*, esto es muy importante en optimización, ya que si tenemos un *array* de 1.000.000 de elementos, y queremos buscar el número 2, y resultado que el primer elemento del *array* es ése número, no nos sirve de nada recorrer los otros 999.999 números, por lo que el bucle debe acabar, ya explicaré optimización más adelante.

CONCEPTO IMPORTANTE

Los *arrays* son objetos, y los objetos no pueden ser comparados directamente, vamos a ver esto un poco mejor con un ejemplo:

```
1 package apuntes;
2
3 public class Arrays {
4
5     static int[] array1 = {1,2,3};
6     static int[] array2 = {1,2,3};
7
8     public static void main (String[] args) {
9
10        System.out.println(array1 == array2);
11
12    }
13
14 }
```

>>> false

Ambos *arrays* tienen el mismo tamaño, y contienen los mismos elementos, el caso es que al compararlos estarás comparando las direcciones de memoria:

```
System.out.println(array1);
System.out.println(array2);
```

>>> [I@15db9742
[I@6d06d69c

Como veis al tratar de imprimir los *arrays* por pantalla nos muestra la dirección de memoria de cada uno, y al compararlas directamente con `==` nos devuelve *false*, debido a que tienen direcciones diferentes, lo mismo ocurre con los *Double*, no pueden ser comparados directamente, y los *Strings* tampoco **en la mayoría de los casos**, no estoy muy seguro de como funcionan los *Strings*, ya que al hacer esto:

```
1 package apuntes;
2
3 public class Arrays {
4
5     static String str1 = "Hola";
6     static String str2 = "Hola";
7
8     public static void main (String[] args) {
9
10        System.out.println(str1 == str2);
11
12    }
13
14 }
```

>>> true

Pero al hacer esto:

```
static String str1 = "Hola";
static String str2;

public static void main (String[] args) {

    str2 = StdIn.readLine();

    System.out.println(str1 == str2);

}
```

>>> <terminated> Arrays [Java Ap
Hola
false

Al introducirlo por teclado nos devuelve *false*, lo mismo pasa si es el *String* se creo por medio de una suma de *Strings*, que aunque sean el mismo da *false*, veamos un par de casos:

```

1 package apuntes;
2
3 import stdlib.StdIn;
4
5 public class Arrays {
6
7     static String str1 = "Hola";
8
9     public static String devolverHola() {
10
11         String res = "";
12         res += "Hola";
13         return res;
14     }
15
16     public static void main (String[] args) {
17
18         System.out.println(str1 == devolverHola());
19     }
20 }

```

>>> false

```

1 package apuntes;
2
3 import stdlib.StdIn;
4
5 public class Arrays {
6
7     static String str1 = "Hola";
8
9     public static String devolverHola() {
10
11         return "Hola";
12     }
13
14     public static void main (String[] args) {
15
16         System.out.println(str1 == devolverHola());
17     }
18 }

```

>>> true

No estoy muy seguro de por qué al sumar *Strings*, aunque sean el mismo en esencia no se pueden comparar, ya que yo no creé la clase *String.java*, pero puedo asegurar que para comparar *Strings* lo mejor es utilizar el método *.equals*, solo tenemos que poner: *str1.equals(str2)*, y nos devolverá *true* aunque alguno de ellos haya sido creado mediante sumas, para los *arrays* no existe este método, por lo que los ejercicios que os ponga vendrán con una función hecha que te lo diga directamente, he de mencionar que esto de los métodos quedará mejor explicados en el PDF de “Clases y Objetos”, un tema muy importante y extenso.

EFICIENCIA

Imaginaos que quiero hacer una función que me diga si se encuentra un número en un *array*, es importante poner un condicional extra para que nos saque si lo encontramos, esto de buscar un número es lo que se llama algoritmos de búsqueda:

```

/*
 * seEncuentra (int[] col; int num) : boolean
 * POST: Indica si se encuentra num en col
 */
public static boolean seEncuentra (int[] col, int num) {
    boolean res = false;

    for (int i = 0; i < col.length && !res; i++) {
        if (col[i] == num) {
            res = true;
        }
    }

    return res;
}

```

Al encontrar el número, *res* pasa a valer *true*, y la condición del bucle deja de cumplirse, por lo que se sale.