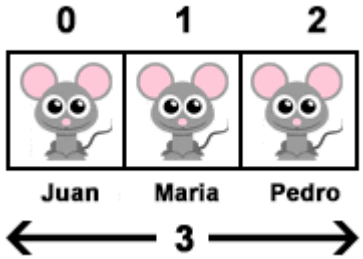
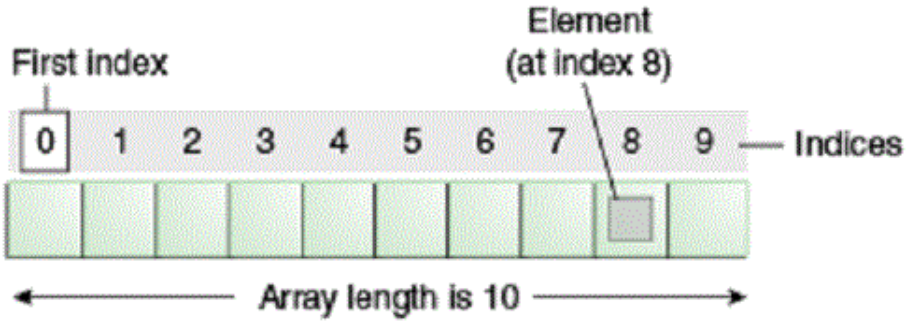
ARRAYS
UNIDIMENSIONALES

Encarnación Sánchez Gallego

ÍNDICE

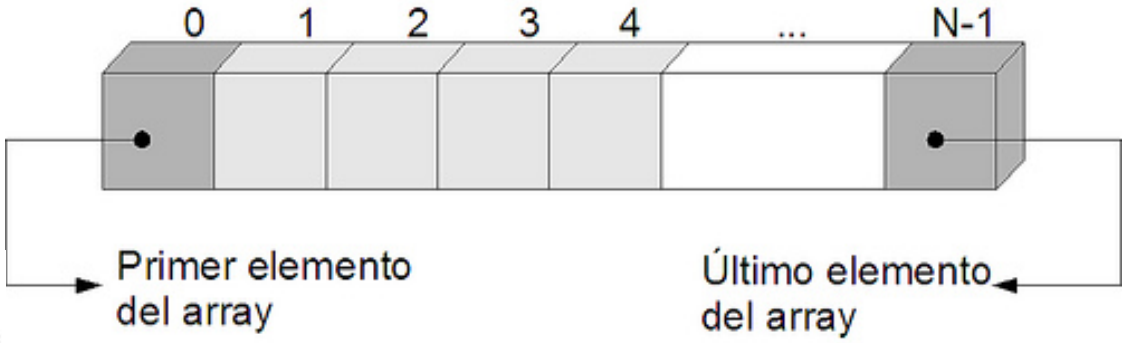


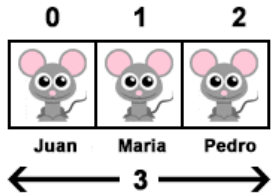
1. Introducción
2. Arrays en Java
3. Arrays unidimensionales.



INTRODUCCIÓN

Un array, arreglo o matriz es simplemente **una variable que puede contener valores múltiples**, a diferencia de una variable regular que solo puede contener un único valor.





INTRODUCCIÓN

Tipos Primitivos	referencias a Objetos
int, short, byte, long	Strings
char, boolean	Arreglos
float, double	otros objetos

A los arrays y otros tipos diferentes a los primitivos se les llama **tipos de datos referenciados o referencias**, porque se utilizan para almacenar la dirección de los datos en la memoria del ordenador.

Estos tipos de datos se construyen a partir de los tipos de datos primitivos o de otros tipos de datos referenciados.

Cuando el conjunto de datos utilizado tiene características similares se suelen agrupar en estructuras para facilitar el acceso a los mismos y por eso también se les llaman datos estructurados, como los arrays, listas, árboles, etc. Pueden estar en la memoria del programa en ejecución, guardados en el disco como ficheros, o almacenados en una base de datos.

INTRODUCCIÓN

Algunas consideraciones de los arrays en Java:

- Son objetos en Java y cada array tiene asociado una variable con la que se puede saber su longitud (length) con el valor máximo de elementos que puede contener, es decir, el tamaño del array.
- Se declara como otras variables pero con corchetes [] después del tipo de datos.
- El tamaño de un array debe especificarse mediante un valor int y no, *long* o *short*.
- Las variables en el array están ordenadas y cada una tiene un índice que **comienza desde 0**. El último índice será el máximo - 1.
- Al igual que otros tipos se puede usar como un campo estático, una variable local o un parámetro de método.
- El array puede contener tipos de datos primitivos así como también objetos de una clase según la definición del array.
- En Java, todas las matrices se asignan dinámicamente.

INTRODUCCIÓN

Además en Java se proporciona un tratamiento especial a los textos o cadenas de caracteres mediante el tipo de dato **String** (que puede entenderse como un array de caracteres) y se crea automáticamente un nuevo objeto de tipo String cuando se encuentra una cadena de caracteres encerrada entre comillas dobles.

Aunque en realidad se trata de objetos, y por tanto son tipos referenciados, se pueden utilizar de forma sencilla como si fueran variables de tipos primitivos:

```
String mensaje;
mensaje= "Mi primera cadena de texto";
```

Cuidado al utilizar la comparación “==” ya que no siempre funciona de manera correcta.

ARRAYS EN JAVA

En Java vamos a utilizar los arrays para guardar un conjunto de objetos de la misma clase. Luego se podrá acceder a cada elemento individual del array mediante un número entero denominado índice (que irá desde 0 para el primer elemento hasta n-1 para el índice del último elemento, siendo n la dimensión del array).

Los arrays son objetos en Java y como tales hay que seguir unos pasos para usarlos convenientemente:

1. **Declarar** el array. Consiste en decir “esto es un array”.
2. **Crear** el array. Consiste en decir el tamaño que tendrá el array, es decir, el número de elementos que contendrá. Una vez creado el array este no podrá cambiar de tamaño
3. **Inicializar** los elementos del array.
4. **Usar** el array.

ARRAYS EN JAVA. DECLARAR Y CREAR

Para declarar los arrays se puede hacer de las siguiente manera.

```
tipo_de_dato[] nombre_del_array;
tipo_de_dato nombre_del_array[];
```

Declaración de un array de números enteros:

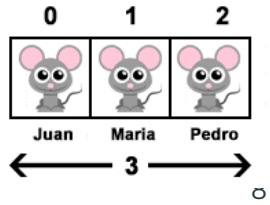
```
int[] arrayDeEnteros;
```

Para **crear** un array de 4 número enteros escribimos

```
arrayDeEnteros = new int[4];
```

La declaración y la creación del array se puede hacer en una misma línea.

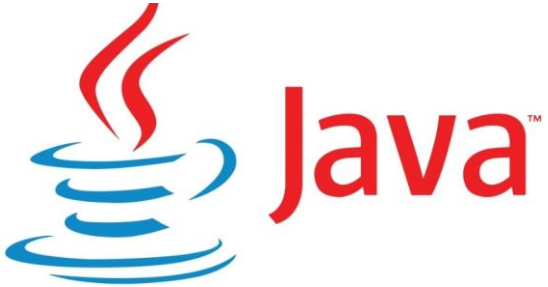
```
int[] arrayDeEnteros = new int[4];
```



ARRAYS EN JAVA. DECLARAR Y CREAR

También podemos **declarar** el array de un tipo de objeto no primitivo, en este caso del tipo Cuenta.

```
Cuenta[] cuentasCliente;
```



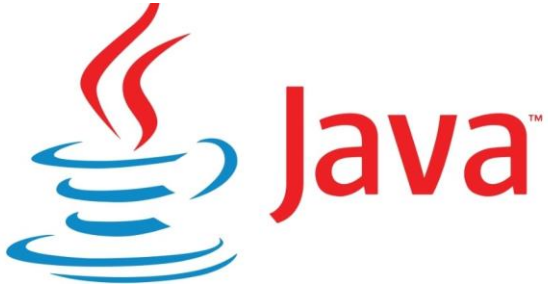
Para **crear** el array se hace de forma parecida:

```
cuentasCliente = new Cuenta[3];
```

La **declaración y** la **creación** del array se puede hacer en una misma línea.

```
Cuenta[] cuentasCliente = new Cuenta[3];
```

ARRAYS EN JAVA. INICIALIZAR Y USAR



Para **inicializar** el array de 4 enteros se puede hacer:

```
arrayDeEnteros[0]=2;
arrayDeEnteros[1]=-4;
arrayDeEnteros[2]=15;
arrayDeEnteros[3]=-25;
```

También podemos utilizar un bucle for (para) para realizar la **inicialización**:

```
for(int i=0; i<4; i++){
        arrayDeEnteros[i]=i+1;  //Se asigna el valor desde 1 hasta 4
}
```

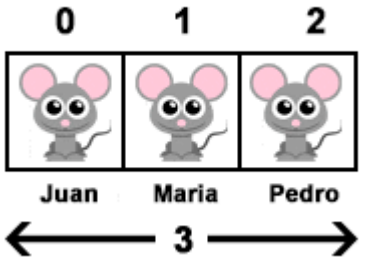
Incluso podemos inicializarlo en una línea y realizar la **declaración**, la **creación** y la **inicialización** en una misma línea, del siguiente modo:

```
int[] arrayDeEnteros={2, -4, 15, -25};
```

ARRAYS EN JAVA. INICIALIZAR Y USAR

Ejemplo con bucles (recordamos que a cada elemento del array se accede a través de su índice, desde 0 hasta el tamaño total del array -1):

```
//Acceder a cada elemento del array y lo mostramos por pantalla

for (int i = 0; i < array.length; i++){

    System.out.println("Índice " + i + " : "+ array[i]);

}
```



ARRAYS EN JAVA. INICIALIZAR Y USAR

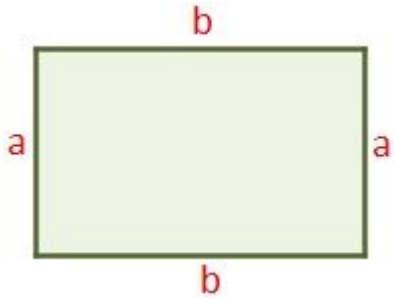
De hecho para hacer el recorrido no necesitamos saber el número de elementos del array, su miembro dato **length** nos proporciona la dimensión del array. De forma equivalente:

```
for(int i=0; i<arrayDeEnteros.length; i++){
        arrayDeEnteros[i]=i+1;
}
```

De igual manera podemos **declarar, crear, inicializar y usar** un array de cadenas de texto:

```
String[] nombres={"Juan", "José", "Miguel", "Antonio"};
for(int i=0; i<nombres.length; i++){
        System.out.println(nombres[i]);
}
```

ARRAYS EN JAVA. EJEMPLO



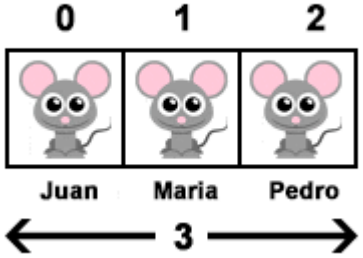
EJEMPLO: vamos a crear un array de tres objetos de la clase Rectangulo:

```
Rectangulo[] rectangulos; //Declaración
rectangulos=new Rectangulo[3]; //Creación
        rectangulos[0]=new Rectangulo(10, 20, 30, 40); //Inicializaciones de 2 formas diferentes
        rectangulos[1]=new Rectangulo(30, 40);
        rectangulos[2]=new Rectangulo(50, 80);
        //O bien, todo en una sola línea
        Rectangulo[] rectangulos={new Rectangulo(10, 20, 30, 40),
                                  new Rectangulo(30, 40), new Rectangulo(50, 80)};
        // Uso del array para calcular y mostrar el área de los rectángulos
        for(int i=0; i<rectangulos.length; i++){
                System.out.println(rectangulos[i].calcularArea());
        }
```

ARRAYS EN JAVA. EJERCICIO

EJERCICIO:

- Rellena un array con números del 1 al 10. Luego muéstralos del 10 al 1.



ARRAYS EN JAVA. EJERCICIO

- Rellena un array con números del 1 al 10. Luego muéstralos del 10 al 1. Solución:

```
int[] arrayDeEnteros = new int[10];
for(int i=0; i<arrayDeEnteros.length; i++){
        arrayDeEnteros[i]=i+1;  //Se asigna el valor desde 1 hasta 10
}
for(int i=arrayDeEnteros.length; i>0; i++){
        System.out.println(arrayDeEnteros[i-1]);  //Se muestran del 10 al 1
}
```

CONCLUSIONES SOBRE ARRAYS

Cuando un array se declara, solo se crea una referencia del array. El momento en que se crea, indicando el tipo de dato y el número de elementos, es cuando realmente se da memoria al array.

Los elementos en los arrays asignados se inicializarán automáticamente a cero (para tipos numéricos), false (para booleano) o null (para tipos de referencia) aunque en las últimas versiones será necesario hacer las inicializaciones implícitamente.

Por tanto, obtener un array es un proceso de dos pasos. Primero, se debe declarar una variable del tipo de array deseado. En segundo lugar, se debe asignar la memoria que mantendrá el array, usar y asignarla a la variable del array. Por lo tanto, en Java, todos los arrays **se asignan dinámicamente.**

MÁS EJEMPLOS

```
int[] arrayInt1; // Declaración del array.

arrayInt1 = new int[10]; //Creación del array reservando un espacio en memoria.

int[] arrayInt2=new int[10]; // Declaración y creación en un mismo lugar.

arrayInt2 = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100}; //Inicialización

float arrayFloat1[] = new float[5]; // Declaración y creación del otro array.

arrayFloat1 = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0}; //Inicialización
```

ARRAYS UNIDIMENSIONALES

Una vez que ya sabemos declarar y crear arrays nos vamos a centrar en los arrays **unidimensionales**. Estos arrays son aquellos que sólo tienen una dimensión (los puestos como ejemplo hasta ahora son arrays unidimensionales).

En las siguientes páginas vamos a ver varias formas en las que se suelen utilizar dichos arrays:

1. **Modificación** de una posición del array.
2. **Acceso** a una posición del array.
3. Array para el **paso de parámetros**.

ARRAYS UNIDIMENSIONALES

La primera **modificación** que tendremos que hacer a nuestro array es la **inicialización**. Para ello podemos hacerlo mediante una inicialización fija o un bucle de la siguiente manera.

```
int[] intArray1 = {10, 20, 30};                    // Declaración e inicialización fija
String[] diassemana= {"L", "M", "X", "J", "V", "S", "D"};
int[] intArray2 = new int[]{ 1,2,3,4,5,6,7,8,9,10 };

int[] intArray3=new int[5];                        // Declaración
for (int i=0;i<5;i++){                                    // Inicialización con bucle
        r[i] = 0;
}
```

ARRAYS UNIDIMENSIONALES

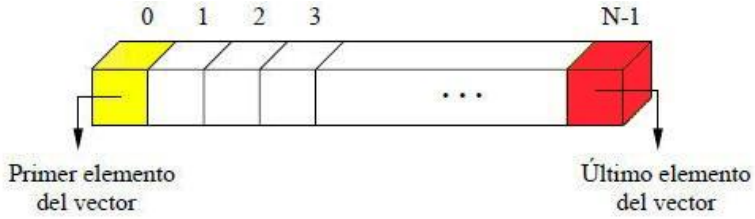
La **modificación** de una posición específica se realiza mediante la indicación entre corchetes de la posición a modificar después del nombre del array. Por ejemplo:

```
int[] Numeros=new int[3]; // Declaramos y creamos un array de 3 enteros

Numeros[0]=99; // Modificamos la primera posición del array, la 0.

Numeros[1]=120; // Modificamos la segunda posición del array, la 1.

Numeros[2]=33; // Modificamos la tercera y última posición del array, la 2

int suma=Numeros[0] + Numeros[1] + Numeros[2]; //Hacemos una suma de todos
```

ARRAYS UNIDIMENSIONALES

Como ya se ha mencionado debido a que los arreglos se implementan como objetos, cada array tiene asociado una variable de instancia de **longitud** (**length**) que contiene la cantidad máxima de elementos que el array puede contener. En otras palabras, length contiene el tamaño del array. Par poder utilizarla basta con hacer la siguiente llamada:

```
int numeros[]={1,2,3};      // Declaración

System.out.println("Longitud del array numeros: " +numeros.length);  //Uso
```



ARRAYS UNIDIMENSIONALES

Ejemplo de inicialización y uso de un array con length:

```
int numeros[] = new int[4];                 // Declaramos y creamos el array
for (int i=0; i < numeros.length; i++){     // Usando length para inicializar el array
        numeros[i]=i+1;
}
System.out.print("El array es: ");          // Ahora mostramos el array en una línea con print
for (int i=0; i < numeros.length; i++){     // Para ello usamos length
        System.out.print(numeros[i]+"  ");
}
System.out.println();          //Escribimos un println al final para añadir un ENTER
```

ARRAYS UNIDIMENSIONALES

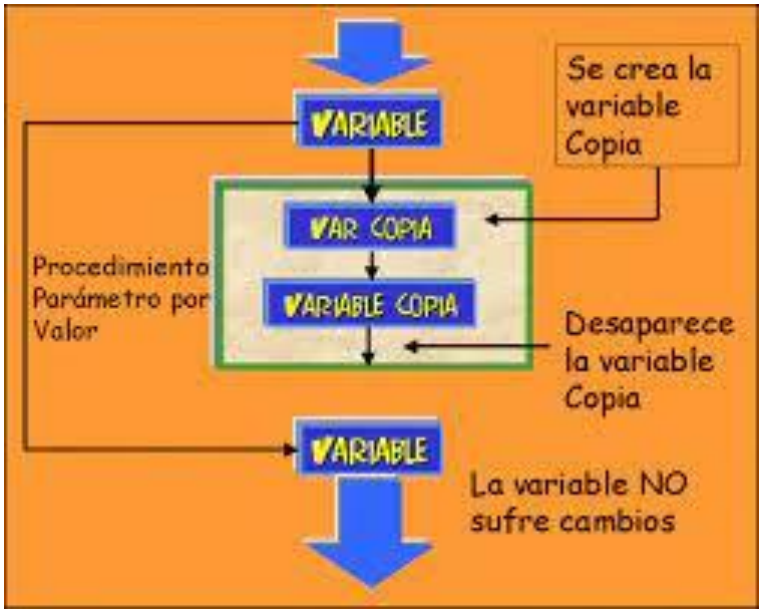
Ejemplo de recorrido:

```
int[] array;                    // Declaración de un array de enteros.
array = new int[5];             // Creamos el array en memoria para 5 enteros.
array[0] = 10;                  //Inicializamos todas las posiciones
array[1] = 20;
array[2] = 30;
array[3] = 40;
array[4] = 50;
for (int i = 0; i < array.length; i++)    // Accediendo a los elementos del array
        System.out.println("Elemento en el índice " + i + " : "+ array[i]);
}
```

ARRAYS UNIDIMENSIONALES

Al utilizar funciones se pueden pasar valores que hagan variar el funcionamiento de dicha función. Este **paso de argumentos** puede ser de cualquier tipo de variables, tanto tipos primitivos como tipos referenciados.

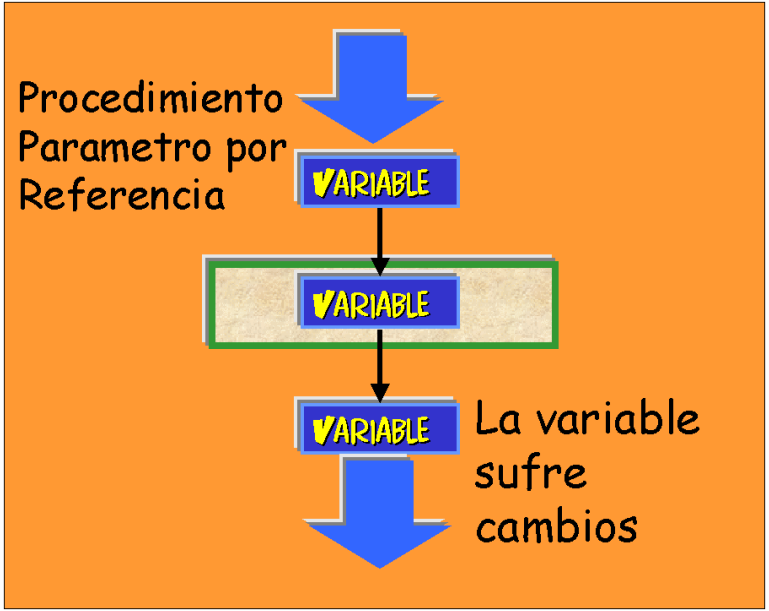
Cuando una variable de tipo primitivo se pasa como argumento, en realidad se realiza una copia que es la que se usará en la función. A este tipo de paso de argumentos se llama **paso por valor**.



ARRAYS UNIDIMENSIONALES

En cambio, cuando una variable de tipo referenciado se pasa como argumento se está pasando una referencia y por tanto en la función se estará usando esa misma variable. A este tipo de paso de argumentos se le llama **paso por referencia**.

En concreto, cuando se pasa un array se está haciendo de esta forma y por tanto, cuando se modifique el array pasado por argumento, se estará modificando el array original.



ARRAYS UNIDIMENSIONALES

Así **declaramos una función** en la que se pasa un array como argumento. La función calculará la suma de todos los números que contiene un array pasado como argumento:

```
int funcionSumaArray (int[] arrayEntrada) {          // Pasamos un array de enteros
        int suma=0;
        for (int i=0; i<arrayEntrada.length; i++) {  // Recorrido del array
                suma=suma+arrayEntrada[i];           // Calculamos la suma 1 a 1
        }
        return suma;
        // Devolvemos el valor total
}
```

Cuando se especifica que el argumento es un array es igual que declarar un array, sin la creación del mismo.

ARRAYS UNIDIMENSIONALES

Para **utilizar** la función y **pasar como argumento** un array a esta función simplemente se pone el nombre del array:

```
int[] arrayNumeros= {1, 2, 3, 4, 5};

int suma=funcionSumaArray (arrayNumeros);

System.out.println("La suma es: " + suma);
```

Después de haber utilizado la función, en la variable "suma" tendremos el resultado de la función almacenado en dicha variable.

ARRAYS EN JAVA. INICIALIZAR Y USAR

Ejemplo completo de programa Java para demostrar el uso de array como parámetro:

```
class PruebaArray {
    public static void main(String args[]) {          // Método principal
        int array[] = {3, 1, 2, 5, 4};                 // Definición del array
        funcionSuma(array);                            // Uso de la función pasando el
    argumento
    }
    public static void funcionSuma(int[] argumentoArray) {     // Función de la suma
        int resultadoSuma= 0;
        for (int i = 0; i < argumentoArray.length; i++)
            resultadoSuma+=argumentoArray[i];
        System.out.println("Suma de valores del array: " + resultadoSuma);
    }
}
```

ARRAYS UNIDIMENSIONALES

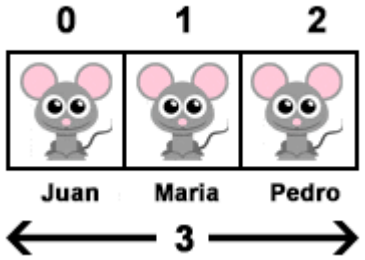
Un método también puede **devolver o retornar** un array. Por **ejemplo**, en el siguiente programa se devuelve (mediante el return) un array desde el método:

```
class PruebaArray { // Programa completo para demostrar el retorno de un array
        public static void main(String args[]) { // Método principal
                int arrayDevuelto[] = funcionRetornadora();          // Llamada a la función
                for (int i = 0; i < arrayDevuelto.length; i++)       // Recorrido del array
                        System.out.print(arrayDevuelto+" ");         // Imprimimos los valores
                }
        public static int[] funcionRetornadora() {                    // Declarando la función
                return new int[]{1,2,3};                             // Devuelve un array nuevo
        }
}
```

ARRAYS UNIDIMENSIONALES

Ejercicio. Ejecuta el siguiente programa. ¿Qué error devuelve? ¿Cómo se puede arreglar?

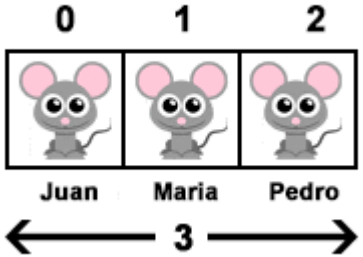
```
public class Array {
    public static void main(String arg[]) {
        int [] arrayValores= null;
        arrayValores[4] = 100;
        System.out.println(arrayValores[4]);
    }
}
```



ARRAYS UNIDIMENSIONALES

Ejercicio. Ejecuta el siguiente programa. ¿Qué error devuelve? ¿Cómo se puede arreglar?

```
public class Array {
    public static void main(String arg[]) {
        int [] arrayValores= new int[10];  //Cualquier valor por encima de 4 vale
        arrayValores[4] = 100;
        System.out.println(arrayValores[4]);
    }
}
```



WEBGRAFÍA

- https://w3api.com/Java/System/out/
- https://www.discoduroderoer.es/arrays-de-objetos-en-java/
- https://javadesdecero.es/arrays/unidimensionales-multidimensionales/