Tema 5. Estructuras de datos – 1 Arrays unidimensionales.

1. ESTRUCTURAS ESTÁTICAS Y DINÁMICAS DE DATOS

En los programas que hemos visto hasta ahora hemos utilizado variables de un tipo de datos simple (int, double, char, etc.) para almacenar los datos del programa. Pero hay casos en los que necesitamos procesar una gran cantidad de datos del mismo tipo y en los que tener una gran cantidad variables simples no es práctico.

Por ejemplo, imaginemos el caso de leer por teclado los nombres de los 20 alumnos de una clase y sus notas finales y mostrarlos ordenados por notas de mayor a menor. Utilizando variables simples necesitaríamos 20 cadenas de caracteres y 20 variables numéricas para guardar todos los valores y además el proceso de ordenación sería bastante complicado.

Para resolver este problema los lenguajes de programación incorporan mecanismos para manipular grupo de datos. Son las **estructuras de datos**.

Un dato de tipo estructurado puede almacenar más de un valor a la vez, con la condición de que todos los elementos deben ser del mismo tipo.

Las estructuras de datos se dividen en dos tipos en función de cómo utilizan la memoria:

- **Estructura de Datos Estáticas**: Son aquellas en las que el espacio ocupado en memoria no puede ser modificado durante la ejecución del programa. Un ejemplo de estructuras estáticas son los arrays.
- **Estructuras de Datos Dinámicas:** Son aquellas en las que el espacio ocupado en memoria puede ser modificado durante la ejecución.

2. ARRAYS

Definición: Un array es una colección finita de datos del mismo tipo, que se almacenan en posiciones consecutivas de memoria y reciben un nombre común.

Por ejemplo, supongamos que queremos guardar las notas de los 20 alumnos de una clase.

Podemos representar gráficamente el array de notas de la siguiente forma:

Array **notas**:

8.50	6.35	5.75	8.50	· · ·	3.75	6.00	7.40
notas[0]	notas[1]	notas[2]	notas[3]	· · ·	notas[17]	notas[18]	notas[19]

Para acceder a cada elemento del array se utiliza el nombre del array y uno o más índices que indican la posición que ocupa el elemento dentro del array.

Cada índice se escribe entre corchetes.

El **primer elemento** del array ocupa la **posición 0**, el segundo la posición 1, etc.

En un array de N elementos el último ocupará la posición N-1.

En el ejemplo anterior, notas[0] contiene la nota del primer alumno y notas[19] contiene la del último

Los índices deben ser enteros no negativos.

3. CREAR ARRAYS UNIDIMENSIONALES

Para crear un array se deben realizar dos operaciones:

- Declaración
- Instanciación

Declarar un Array

En la declaración se crea la referencia al array.

La referencia será el nombre con el que manejaremos el array en el programa.

Además del nombre, en la declaración se debe indicar el tipo de datos que contendrá.

De forma general un array unidimensional se puede declarar de cualquiera de estas dos formas:

```
tipo []  nombreArray;
tipo nombreArray[];
```

Aunque la más habitual en java es hacerlo de la primera forma:

```
tipo []  nombreArray;
```

- tipo: indica el tipo de datos que contendrá. Un array puede contener elementos de tipo básico o referencias a objetos.
- nombreArray: es la referencia al array. Es el nombre que se usará en el programa para manejarlo.

Por ejemplo:

```
int [] ventas;  //array de datos de tipo int llamado ventas
double [] temperaturas;  //array de datos de tipo double llamado temperaturas
String [] nombres; //array de datos de tipo String llamado nombres
```

Instanciar un Array

Mediante la instanciación se reserva un bloque de memoria para almacenar todos los elementos del array.

La dirección donde comienza el bloque de memoria donde se almacenará el array, se asigna al nombre del array. Esto quiere decir que el nombre del array contiene la dirección de memoria donde se encuentra.

De forma general un array se instancia así:

```
nombreArray = new tipo[tamaño];
```

- *nombreArray:* es el nombre creado en la declaración.
- *tipo*: indica el tipo de datos que contiene.
- *tamaño*: es el número de elementos del array. Debe ser una expresión entera positiva. El tamaño del array **no se puede modificar** durante la ejecución del programa.
- *new*: operador para crear objetos. Mediante new **se asigna la memoria** necesaria para ubicar el objeto. Java implementa los arrays como objetos.

Por ejemplo:

```
ventas = new int[5]; //se reserva memoria para 5 enteros y
                     //se asigna la dirección de inicio del array a ventas.
```

La declaración y la instanciación se suelen hacer en una sola instrucción:

```
tipo []  nombreArray = new tipo[tamaño];
```

Por ejemplo:

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

El tamaño del array también se puede asignar durante la ejecución del programa, es decir, podemos leer por teclado el tamaño que tendrá el array y a continuación crearlo:

```
Scanner sc = new Scanner(System.in);
System.out.print("Número de elementos del array: ");
int numeroElementos = sc.nextInt(); //Se introduce por teclado el número de elementos
int [] ventas = new int[numeroElementos];//Se crea el array con el tamaño introducido
```

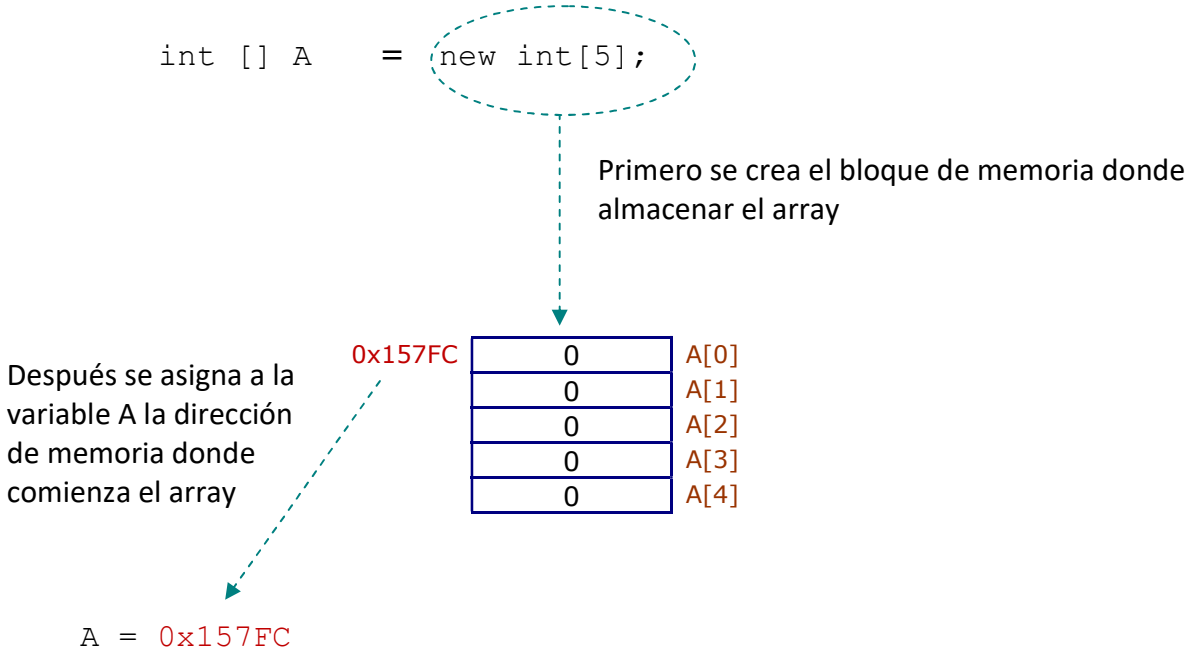
Si no hay memoria suficiente para crear el array, el operador *new* lanza la excepción java.lang.OutOfMemoryError.

Debe quedar clara la diferencia entre el nombre del array y el contenido del array.

Cuando por ejemplo se ejecuta una instrucción para crear un array de enteros llamado A:

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

se realizan dos operaciones: primero se crea un bloque de memoria donde almacenar el array de 5 enteros y después se asigna la dirección de inicio del bloque de memoria, también llamado referencia del array, a la variable A.



El nombre del array contiene la dirección de inicio del bloque de memoria donde se encuentra el array.

4. INICIALIZAR ARRAYS UNIDIMENSIONALES

Un array es un objeto, por lo tanto, cuando se crea, a sus elementos se les asigna automáticamente un valor inicial:

- 0 para arrays numéricos
- '\u0000' (carácter nulo) para arrays de caracteres
- false para arrays booleanos
- null para arrays de String y de referencias a objetos.

También podemos dar otros valores iniciales al array cuando se crea.

Los valores iniciales se escriben entre llaves separados por comas.

Los valores iniciales deben aparecer en el orden en que serán asignados a los elementos del array.

El número de valores determina el tamaño del array.

Por ejemplo:

```
double [] notas = {6.7, 7.5, 5.3, 8.75, 3.6, 6.5};
```

en la instrucción se declara el array *notas* de tipo double, se reserva memoria para 6 elementos y se les asignan esos valores iniciales.

Ejemplos:

```
//creación de un array de 4 elementos booleanos
boolean [] resultados = {true,false,true,false};

//creación de un array de 7 elementos de tipo String
String [] dias = {"Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"};
```

5. ACCEDER A LOS ELEMENTOS DE UN ARRAY

Para acceder a cada elemento del array se utiliza el nombre del array y un índice que indica la posición que ocupa el elemento dentro del array. El índice se escribe entre corchetes.

Se puede utilizar como índice un valor entero, una variable de tipo entero o una expresión de tipo entero.

El **primer elemento** del array ocupa la **posición 0**, el segundo la posición 1, etc. **En un array de N elementos el último ocupará la posición N-1**.

Un elemento de un array se puede utilizar igual que cualquier otra variable. Se puede hacer con ellos las mismas operaciones que se pueden hacer con el resto de variables (incremento, decremento, operaciones aritméticas, comparaciones, etc).

Por ejemplo:

int m = 5;

int [] a = new int[5];

0	0	0	0	0
a[0]	a[1]	a[2]	a[3]	a[4]

a[1] = 2;

0	2	0	0	0
a[0]	a[1]	a[2]	a[3]	a[4]

a[2] = a[1];

0	2	2	0	0
a[0]	a[1]	a[2]	a[3]	a[4]

a[0] = a[1] + a[2] + 2;

6	2	2	0	0
a[0]	a[1]	a[2]	a[3]	a[4]

a[0]++;

7	2	2	0	0
a[0]	a[1]	a[2]	a[3]	a[4]

int m = 5;

a[3] = m + 10;

7	2	2	15	0
a[0]	a[1]	a[2]	a[3]	a[4]

Si se intenta acceder a un elemento que está fuera de los límites del array (un elemento con índice negativo o con un índice mayor que el que corresponde al último elemento del array) **el compilador no avisa del error**. El error se producirá durante la ejecución. En ese caso se lanza una excepción java.lang.ArrayIndexOutOfBoundsException.

Se puede saber el número de elementos del array mediante el atributo **length**.

Podemos utilizar length para evitar errores de acceso al array.

Por ejemplo, leemos por teclado un valor y la posición del array donde lo vamos a guardar:

```
int i, valor;
int [] a = new int[10];
System.out.print("Posición: ");
i = sc.nextInt(); //pedimos una posición del array
if(i>=0 && i < a.length){
   System.out.print("Valor: ");
   valor = sc.nextInt();  //pedimos el valor
   a[i] = valor;
}else{
 System.out.println("Posición no válida");
}
```

6. RECORRER UN ARRAY UNIDIMENSIONAL

Para recorrer un array se utiliza una instrucción iterativa, normalmente una instrucción for, aunque también puede hacerse con while o do..while, utilizando una variable entera como índice que tomará valores desde el primer elemento al último o desde el último al primero.

Por ejemplo, el siguiente programa declara un array de 7 elementos de tipo double y le asigna valores iniciales. A continuación recorre el array para mostrar el contenido.

```
double[] notas = {2.3, 8.5, 3.2, 9.5, 4, 5.5, 7.0}; //array de 7 elementos
for (int i = 0; i < 7; i++) {
      System.out.print(notas[i] + " "); //se muestra cada elemento del array
}
```

Para evitar errores de acceso al array es recomendable **utilizar length para recorrer el array completo.**

```
double[] notas = {2.3, 8.5, 3.2, 9.5, 4, 5.5, 7.0}; //array de 7 elementos
for (int i = 0; i < notas.length; i++) {
      System.out.print(notas[i] + " "); //se muestra cada elemento del array
}
```

Ejemplo: Programa que lee por teclado la nota de los alumnos de una clase y calcula la nota media del grupo. También muestra los alumnos con notas superiores a la media. El número de alumnos se lee por teclado.

Este programa crea un array de elementos de tipo double que contendrá las notas de los alumnos. El tamaño del array será el número de alumnos de la clase.

Se realizan 3 recorridos sobre el array, el primero para asignar a cada elemento las notas introducidas por teclado, el segundo para sumarlas y el tercero para mostrar los alumnos con notas superiores a la media.

```
import java.util.Scanner;
public class Recorrido2 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int numAlum, i;
        double suma = 0, media;
        do {
            System.out.print("Número de alumnos de la clase: ");
            numAlum = sc.nextInt();
        } while (numAlum <= 0);
```

```
        double[] notas = new double[numAlum]; //se crea el array

        // Entrada de datos.
        // Se asigna a cada elemento del array la nota introducida por teclado
        for (i = 0; i < notas.length; i++) {
            System.out.print("Alumno " + (i + 1) + " Nota final: ");
            notas[i] = sc.nextDouble();
        }

        // Sumar todas las notas
        for (i = 0; i < notas.length; i++) {
            suma = suma + notas[i];
        }

        // Calcular la media
        media = suma / notas.length;

        // Mostrar la media
        System.out.printf("Nota media del curso: %.2f %n", media);

        // Mostrar los valores superiores a la media
        System.out.println("Listado de notas superiores a la media: ");

        for (i = 0; i < notas.length; i++) {
            if (notas[i] > media) {
                System.out.println("Alumno número " + (i + 1)+ " Nota final: " + notas[i]);
            }
        }
    }
}
```

Bucle for para colecciones (foreach)

A partir de java 5 se incorpora una instrucción *for* mejorada para recorrer arrays y contenedores en general.

Permite acceder secuencialmente a cada elemento del array.

La sintaxis es:

```
for(tipo nombreDeVariable : nombreArray){
     . . . .
}
```

- *tipo*: indica el tipo de datos que contiene el array.
- *nombreDeVariable*: variable a la que en cada iteración se le asigna el valor de cada elemento del array. Está definida dentro del for por lo que solo es accesible dentro de él.
- *nombreArray*: es el nombre del array que vamos a recorrer.

Mediante este bucle solo podemos acceder a los elementos del array. No podemos hacer modificaciones en su contenido.

Ejemplo:

El siguiente programa crea un array temperatura de 10 elementos. Lee por teclado los valores y a continuación los muestra por pantalla usando un bucle *foreach*.

```
import java.util.Scanner;
public class Recorrerforeach1 {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        double [] temperatura = new double[10];
        int i;
        for(i = 0; i<temperatura.length;i++){
            System.out.print("Elemento " + i + ": ");
            temperatura[i] = sc.nextDouble();
        }

        for(double t: temperatura){ //bucle foreach
            System.out.print(t + " ");
        }
        System.out.println();
    }
}
```

7. ARRAYS DE CARACTERES

Un array unidimensional de caracteres se crea de forma similar a un array unidimensional de cualquier otro tipo.

Ejemplo: Array de 8 caracteres llamado *cadena*.

```
char [] cadena = new char[8];
```

Por defecto se inicializa con el carácter nulo.

\u0000	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000
cadena[0]	cadena [1]	cadena [2]	cadena [3]	cadena [4]	cadena [5]	cadena [6]	cadena [7]

Ejemplo: Array de 5 caracteres llamado vocales. Se asignan valores iniciales: a, e, i, o, u

```
char [] vocales = {'a', 'e', 'i', 'o', 'u'};
```

a	e	i	o	u
vocales[0]	vocales [1]	vocales [2]	vocales [3]	vocales [4]

A diferencia de los demás arrays, **se puede mostrar su contenido completo** mediante una sola instrucción print o printf.

Por ejemplo, para mostrar el contenido de los arrays anteriores:

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

Mostrará 8 caracteres nulos (en blanco)

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

Mostrará: `aeiou`

El atributo length de un array de caracteres contiene el tamaño del array independientemente de que sean caracteres nulos u otros caracteres.

Por ejemplo:

```
char [] cadena = new char[8];
```

\u0000	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000
cadena[0]	cadena [1]	cadena [2]	cadena [3]	cadena [4]	cadena [5]	cadena [6]	cadena [7]

```
System.out.println(cadena.length);
```

Muestra: `8`

```
cadena[0] ='m';
cadena[1] ='n';
```

m	n	\u0000	\u0000	\u0000	\u0000	\u0000	\u0000
cadena[0]	cadena [1]	cadena [2]	cadena [3]	cadena [4]	cadena [5]	cadena [6]	cadena [7]

```
System.out.println(cadena.length);
Muestra: 8
```

```
System.out.print(cadena);
System.out.print(cadena);
System.out.println(".");
```

Mostrará:

mn*bbbbbb*mn*bbbbbb*. (Los espacios en blanco se han representado con el carácter *b)*

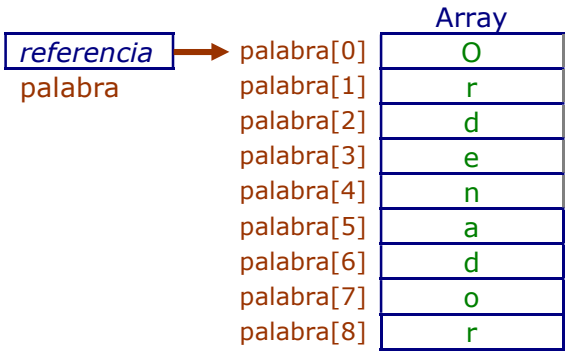
Se puede asignar un String a un array de caracteres mediante el método toCharArray() de la clase String.

Ejemplo:

```
String s = "Ordenador";
```


```
char [] palabra = s.toCharArray();
```

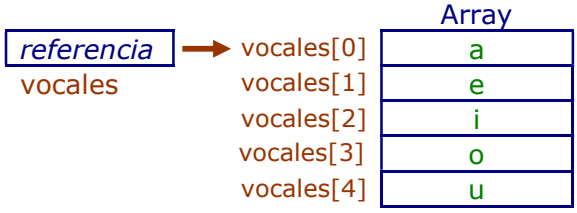
Se crea un nuevo array de caracteres con el contenido del String s y se asigna la dirección de memoria a palabra.



Se puede crear un String a partir de un array de caracteres.

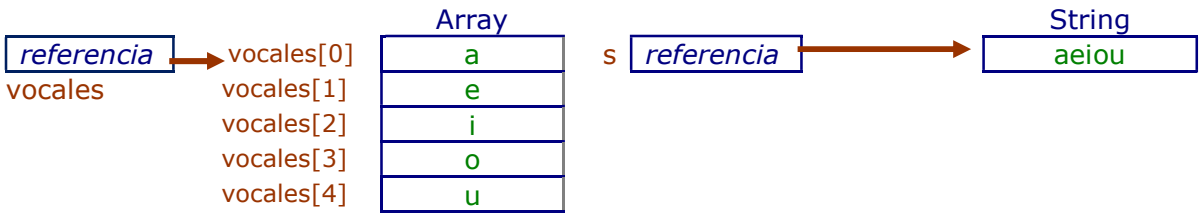
Ejemplo:

```
char [] vocales = {'a', 'e', 'i', 'o', 'u'};
```



```
String s = new String(vocales);
```

Se crea un nuevo String con el contenido del array vocales y se asigna la dirección de memoria a s.



8. RECORRER UN ARRAY DE CARACTERES UNIDIMENSIONAL

Se puede recorrer de forma completa utilizando una instrucción iterativa, normalmente un for.

Por ejemplo:

```
char [] s = new char[10];
s[0]='a';
s[1]='b';
s[2]='c';
for(int i = 0; i<s.length; i++) //Muestra todos los caracteres del array, incluidos los nulos.
    System.out.print(s[i]+ " ");
```

Si los caracteres no nulos se encuentran al principio del array se puede recorrer utilizando un while, mientras que no encontremos un carácter nulo.

Por ejemplo:

```
char [] s = new char[10];
s[0]='a';
s[1]='b';
s[2]='c';
int i = 0;
while(s[i]!='\0'){
      System.out.print(s[i]);
      i++;
}
```

9. COPIAR ARRAYS

Cuando nos surge la necesidad de realizar una copia de un array puede parecernos lógico utilizar una instrucción de asignación como haríamos con una variable de tipo primitivo.

Si disponemos de un array llamado A de 5 elementos de tipo entero:

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

y queremos copiar su contenido en otro array B mediante la instrucción:

```
int []  B = A;
```

en realidad no estamos haciendo una copia del array A. Lo que estamos haciendo es asignar a la variable B el contenido de la variable A y lo que contiene A es la dirección de memoria donde se encuentra el array.

Lo que hemos conseguido **no es una copia del array** A sino que hemos creado un **alias o referencia alternativa al array**.

De esta forma ahora tenemos dos formas de acceder al mismo array: utilizando la variable A o utilizando la variable B. Ambas nos llevan a la misma zona de memoria donde se encuentra el array.

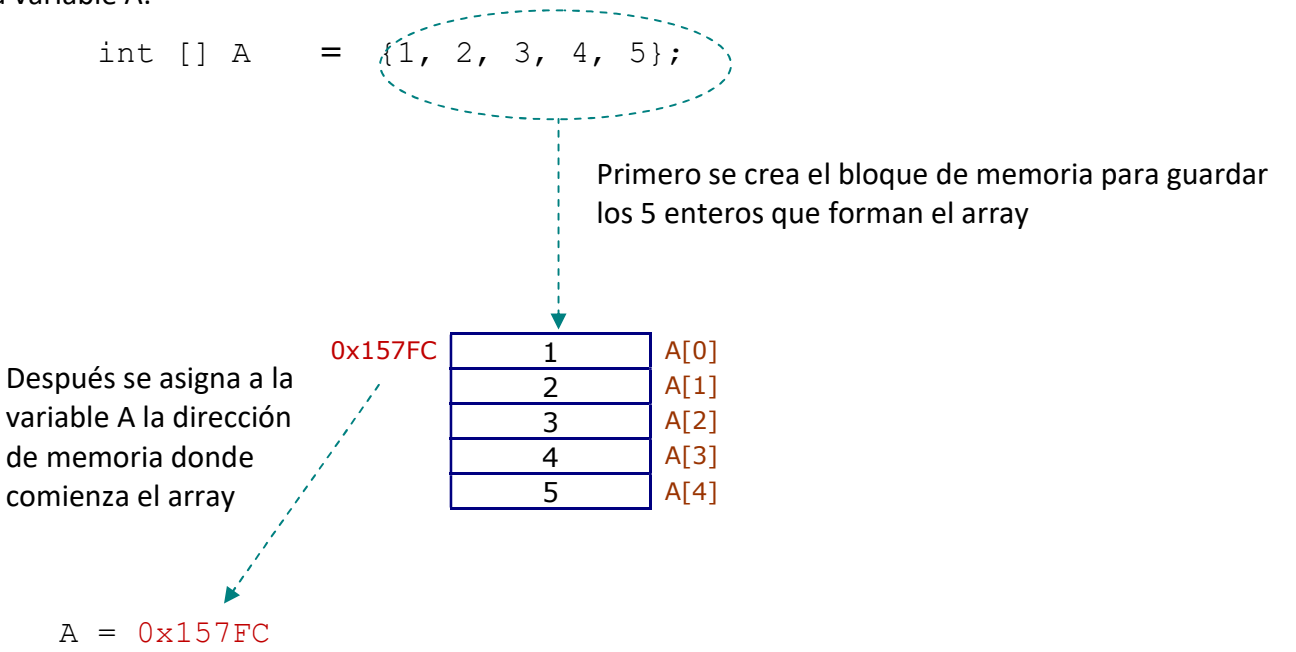
Tenemos **un solo array** al que podemos acceder mediante dos variables.

Para entenderlo mejor vamos a representarlo de forma gráfica.

Cuando se ejecuta la instrucción que crea el array A

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

se realizan dos operaciones: primero crea un bloque de memoria para guardar el array de 5 enteros y después se asigna la dirección de inicio del bloque de memoria (también llamado referencia del array) a la variable A.



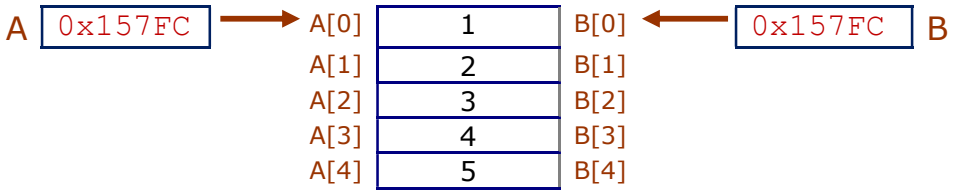
Si ahora declaramos la variable B y le asignamos el valor de A:

```
int []  B = A;
```

Esta instrucción crea una variable B y le asigna el contenido de la variable A. Lo que estamos haciendo es copiar en B la dirección de memoria del array A.

```
B = 0x157FC;
```

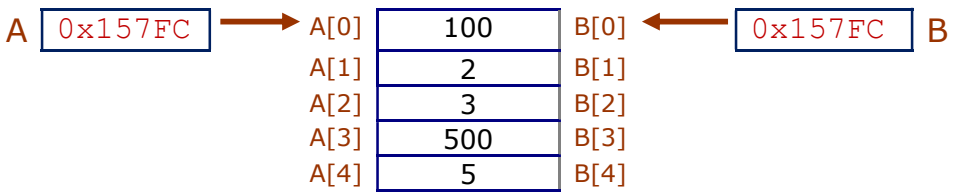
En memoria tenemos un solo array al que podemos acceder mediante dos variables: A y B.



Ahora podemos realizar operaciones sobre el array utilizando las dos variables:

```
A[0] = 100;
B[3] = 500;
```

Ambas modifican el mismo array:



Hemos podido comprobar que de esta forma no estamos copiando un array en otro.

Para copiar el contenido de un array en otro podemos hacerlo de varias maneras:

- Creando un nuevo array y mediante un bucle y realizar la copia elemento a elemento.
- Utilizando el método clone().
- Utilizando el método System.arraycopy().

Copiar un array unidimensional mediante un bucle elemento a elemento.

Esta es la forma trivial de hacerlo.

Si tenemos un array A de enteros

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

y queremos hacer una copia de este array A en otro array llamado B lo que haremos será crear el nuevo array B y copiar elemento a elemento:

```
int[] B = new int[5];     //se crea un nuevo array B
for (int i = 0; i < A.length; i++) {  //se copia cada elemento de A en B
     B[i] = A[i];
}
```

Copiar un array unidimensional utilizando el método clone.

De forma general, para copiar arrays con el método clone se escribe la instrucción:

```
arrayDestino = arrayOrigen.clone();
```

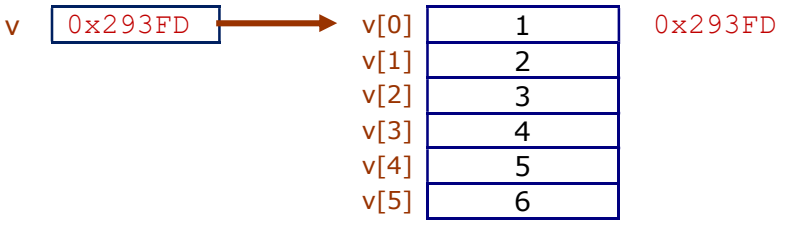
El método *clone* crea en memoria una copia del *arrayOrigen* . La dirección de memoria del array creado a *arrayDestino*. De esta forma **obtenemos dos array distintos con el mismo contenido**.

Ejemplo:

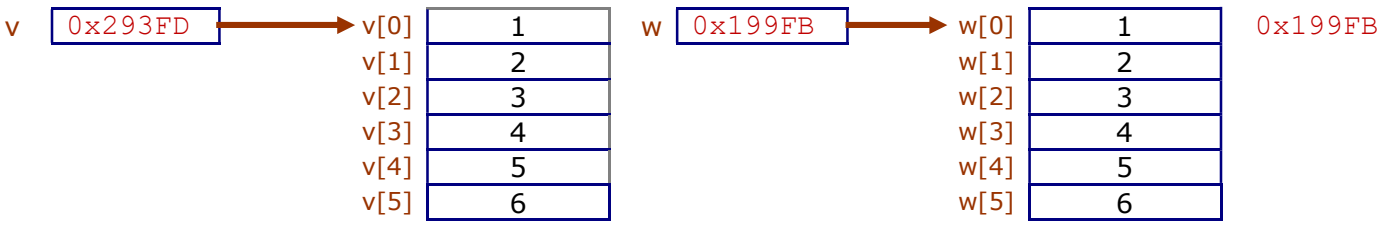
```
int[] v = {1, 2, 3, 4, 5, 6};
int [] w = v.clone();
```

El proceso realizado se puede representar de forma gráfica de esta forma:

```
int[] v = {1, 2, 3, 4, 5, 6};
```




```
int [] w = v.clone();
```



La instrucción `v.clone()` crea en memoria un nuevo array y copia en el nuevo array el contenido del array `v`. La dirección de inicio del array creado se asigna a la variable `w`.

Copiar un array unidimensional utilizando System.arraycopy().

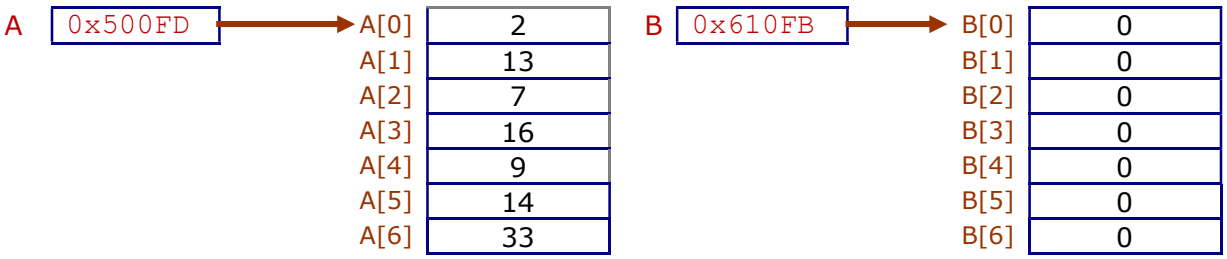
Este método copia el contenido de un array en otro array que ya debe existir.

Para realizar la copia hay que indicar al método arraycopy 5 argumentos en este orden:

1. el array origen
2. desde qué posición del array origen vamos a copiar
3. el array destino
4. a partir de qué posición vamos a copiar en el array de destino
5. cantidad de elementos a copiar

Ejemplo: disponemos de los arrays A y B

```
int[] A = {2, 13, 7, 16, 9, 14, 33};
int[] B = new int[7];
```

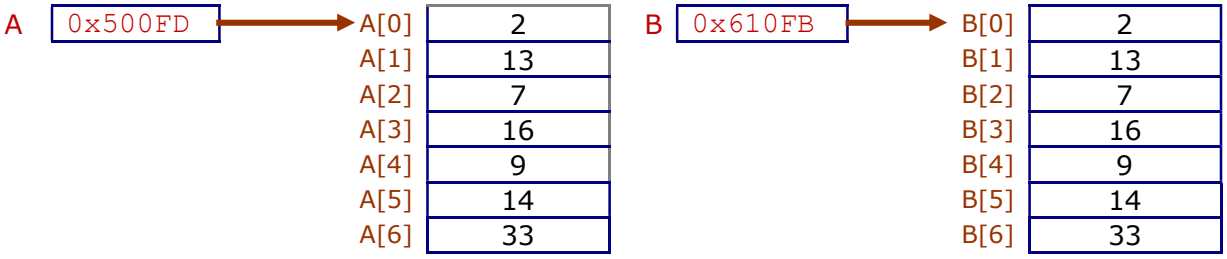


Para copiar el array A en el array B se escribe la instrucción:

```
System.arraycopy(A, 0, B, 0, A.length);
```

Los valores que hemos escrito indican el nombre del array de origen (A), a partir de qué posición del array A vamos a copiar, en este caso desde la primera posición (0), a continuación el array de destino (B), a partir de qué posición vamos a copiar en el array de destino, en este caso desde la primera posición (0) y finalmente el número de elementos a copiar, este caso queremos copiar todos los elementos de A por lo que hemos indicado A.length.

Como resultado se ha realizado una copia completa del array A en el Array B.



La ventaja que nos ofrece este método es que podemos copiar solo una parte de un array en otro. Además, los arrays origen y destino pueden tener tamaños diferentes.

Por ejemplo, disponemos de los siguientes arrays:

```
int[] A = {2, 13, 7, 16, 9, 14, 33};
int[] B = new int[3];
int[] C = new int[3];
int[] D = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
```

Vamos a copiar los tres primeros elementos del array A en el array B, los tres últimos elementos de A en el array C y los elementos de las posiciones 3 y 4 del array A en las posiciones 2 y 3 del array D.

Escribimos las instrucciones:

```
System.arraycopy(A, 0, B, 0, 3); //copia los tres primeros elementos de A en B
System.arraycopy(A, 4, C, 0, 3); //copia los tres últimos elementos de A en C
System.arraycopy(A, 3, D, 2, 2); //copia los elementos 3 y 4 de A en los
                                 //elementos 2 y 3 de D
```

El contenido final de los arrays después de realizar la copia es el siguiente:

```
Array B:
2 13 7
Array C:
9 14 33
Array D:
1 1 16 9 1 1 1 1 1 1
```

10. ARRAYS Y MÉTODOS

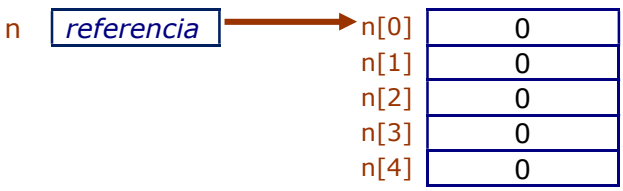
Cuando se pasa un array a un método **el método recibe una copia de la referencia** al array. Realmente estamos creando un alias del array de forma que dentro del método tenemos acceso al array original y se puede modificar su contenido.

Por ejemplo, el siguiente programa crea un array de 5 elementos de tipo int y mediante un método asigna a los elementos del array los valores 1, 2, 3, 4, 5.

```
public class PasarArrayMetodo {
    public static void main(String[] args) {
        int [] n = new int[5];
        for(int i = 0; i < n.length; i++){
            System.out.print(n[i] + " ");
        }
        System.out.println();
        inicializar(n);
        for(int i = 0; i < n.length; i++){
            System.out.print(n[i] + " ");
        }
    }
    public static void inicializar(int [] x){
        for(int i = 0; i < x.length; i++){
            x[i] = i+1;
        }
    }
}
```

De forma gráfica, este programa realiza lo siguiente:

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



Se crea un array de 5 elementos de tipo int y se asigna la dirección de inicio del array a la variable n

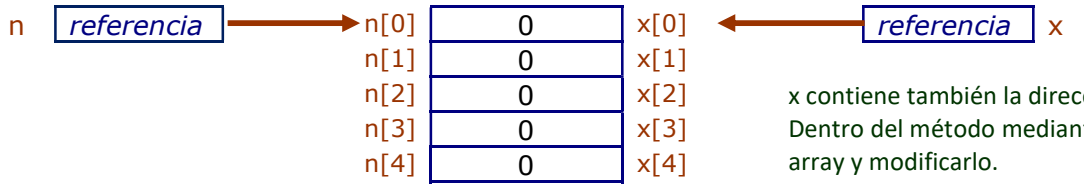
```
for(int i = 0; i < n.length; i++) {
      System.out.print(n[i] + " ");
}
```

Muestra: `0 0 0 0 0`

```
        inicializar(n);
        . . .

    }
    public static void inicializar(int [] x){
         . . .
```

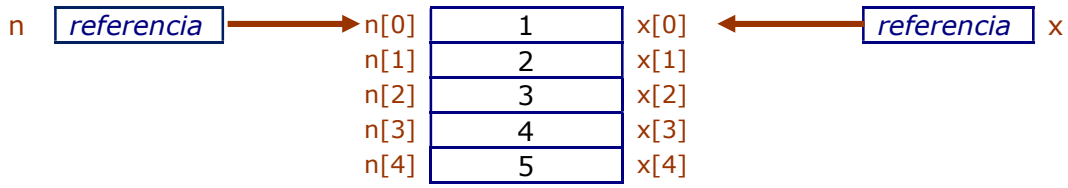
En la llamada al método se copia el contenido de n (referencia del array) en x
`int [] x = n`



x contiene también la dirección donde está el array.
Dentro del método mediante x podemos acceder al array y modificarlo.

```
public static void inicializar(int [] x){
        for(int i = 0; i < x.length; i++){
            x[i] = i+1;
        }
}
```

x contiene la dirección donde está el array.
En este for se modifica el contenido del array.




```
    ...
    inicializar(n);
    for(int i = 0; i < n.length; i++){
        System.out.print(n[i] + " ");
    }

}
```

Cuando volvemos al método main y recorremos el array comprobamos que el array se ha modificado.

Muestra: `1 2 3 4 5`

Un método además de recibir un array como parámetro, puede **devolver un array mediante la instrucción return**.

Ejemplo:

Método que crea un array de N elementos de tipo entero. A cada elemento le asigna un valor aleatorio entre 1 y 10. El valor de N lo recibe como parámetro. El método devuelve el array creado.

En el método main se llamará al método para crear el array y después se mostrará el contenido del array.

```
public class ArraysMetodos {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int elementos;
        do {
            System.out.print("Introduce tamaño del array: ");
            elementos = sc.nextInt();
            if (elementos < 1) {
                System.out.println("Valor no válido");
            }
        } while (elementos < 1);

        int[] numeros = crearArray(elementos); //se llama al método

        //mostrar el contenido del array
        for(int i = 0; i <numeros.length; i++){
            System.out.println(numeros[i]);
        }
    }
    //método para crear y devolver un aray de N elementos de tipo int
    //los valores de los elementos del array se asignan de forma aleatoria
    //con valores entre 1 y 10
    public static int[] crearArray(int N) {
        Random rnd = new Random();
        int[] A = new int[N]; //se crea un array de enteros de tamaño N
        //se asigna a cada elemento del array un número aleatorio entre 1 y 10
        for (int i = 1; i < A.length; i++) {
            A[i] = rnd.nextInt(10) + 1;
        }
        return A; //se devuelve el array creado
    }
}
```

Ejemplo: Programa que crea un array de enteros. El tamaño del array se pide por teclado. Después se introducen por teclado los valores de los elementos del array. Finalmente se muestra el contenido del array y el mayor valor que contienen el array. Se utilizarán los siguientes métodos:

- Método para leer el tamaño del array.
- Método para crear el array.
- Método para introducir por teclado los valores de los elementos del array.
- Método para mostrar el array.
- Método que calcula y devuelve el mayor elemento del array.

```
public class ArraysMetodos2 {
    public static void main(String[] args) {
        int elementos = leerTamaño();
        int[] numeros = crearArray(elementos);
        llenarArray(numeros);
        mostrarArray(numeros);
        System.out.println("El mayor elemento es: " + obtenerMayor(numeros));
    }
    //Método que lee y devuelve un entero que indica el tamaño de un array
    public static int leerTamaño() {
        Scanner sc = new Scanner(System.in);
        int N;
        do {
            System.out.print("Introduce tamaño del array > 0: ");
            N = sc.nextInt();
            if (N < 1) {
                System.out.println("Valor no válido");
            }
        } while (N < 1);
        return N;
    }
```

```
    //método para crear y devolver un aray de N elementos de tipo int
    public static int[] crearArray(int N) {
        return new int[N]; //se crea un array de enteros de tamaño N y se devuelve
    }

    //método que asigna valores a un array de enteros
    //los valores se introducen por teclado
    public static void llenarArray(int[] A) {
        Scanner sc = new Scanner(System.in);
        for (int i = 0; i < A.length; i++) {
            System.out.print("Elemento " + i + ": ");
            A[i] = sc.nextInt();
        }
    }

    //método para mostrar el contenido de un array de enteros
    public static void mostrarArray(int[] A) {
        for (int i = 0; i < A.length; i++) {
            System.out.print(A[i] + " ");
        }
    }

    //método que calcula y devuelve el mayor elemento de un array de enteros
    public static int obtenerMayor(int [] A){
        int mayor = A[0]; //tomamos el primer elemento como mayor
        for(int i = 1; i < A.length; i++){ //desde el segundo elemento hasta el final
            if(A[i] > mayor){ //comprobamos si es mayor que el mayor actual
                mayor = A[i];
            }
        }
        return mayor;  //devolvemos el valor mayor obtenido
    }
}
```