UT 4: Estructuras de almacenamiento.

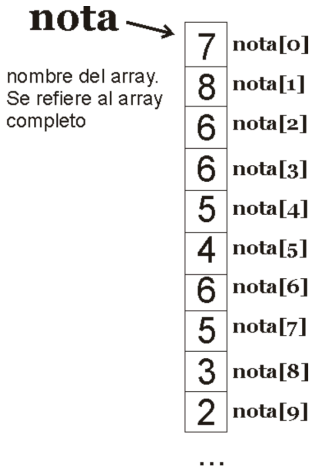
1. Introducción

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 leer las notas de una clase de 25 alumnos. Desearemos por tanto almacenarlas y para ello, con lo que conocemos hasta ahora, no habrá más remedio que declarar 25 variables.

Para resolver este tipo de problemas, prácticamente todos los lenguajes de programación dispone de un mecanismo que permite agrupar variables del mismo tipo bajo una estructura común, denominada *array*.

2. Arrays

Un *array* es una estructura de datos formada por una cantidad fija de datos de un mismo tipo (homogénea). Todos los elementos de un *array* se almacenan en posiciones contiguas de memoria y para poder acceder a un elemento concreto se debe conocer su posición o índice.



Hay varios tipos de arrays:

- **Unidimensionales**: Para acceder a un elemento solo se necesita un índice, es decir, son vectores de una sola dimensión.
- **Multidimensionales**: Se trata de un *array* formado con varios arrays a su vez. En el caso de un *array* bidimensional o matriz, para poder acceder a un elemento se deben utilizar dos índices, uno por cada dimensión.

2.1. Arrays unidimensionales

Para declarar un *array* unidimensional en Java, se necesita conocer el tipo de dato y el nombre:

```
tipo-o-clase[] identificador-array;
o
tipo-o-clase identificador-array[];
```

Por ejemplo, un *array* de enteros se puede declarar de dos formas (equivalentes):

```
int[] numeros;
o
int numeros[];
```

Esto son ejemplos de arrays de distintos tipos:

```
int[] numerosEnteros;    // array de tipo int
double[] numerosReales;  // array de tipo double
String[] nombres;        // array de tipo String
Object[] objetos;        // array de la clase Object
```

Para poder utilizar un *array*, no solo hay que declararlo, sino que hay que instanciar el objeto *array*, e indicar el tamaño del mismo:

```
nombres = new String[100];
```

```
numerosEnteros = new int[50];
```

De forma similar al resto de objetos de Java, un *array* se puede inicializar al momento de la declaración

```
int[] numerosEnteros = new int[10];
String[] nombres = new String[100];
```

Un *array* también se puede inicializar indicando la lista de valores que va a almacenar:

```
String[] diasLaborables = {"Lunes",
                    "Martes",
                    "Miércoles",
                    "Jueves",
                    "Viernes";

int[] enteros = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
```

El *array* "diasLaborables" almacena 5 objetos de tipo *String* con los nombres de los días laborables de la semana. El *array* "enteros" almacena 11 números enteros con valores de 0 a 10.

NOTA: Hay que conocer el tamaño del *array* previamente e indicarlo, salvo que se inicialice como el caso anterior, es decir, indicando el listado de valores, en cuyo caso se sabría también el número de elementos que contiene.

Para hacer referencia a cada elemento de un *array* es necesario indicar la posición que ocupa en la estructura de almacenamiento. Esta posición se denomina índice. El primer elemento de un *array* se almacena en la posición cero y el último elemento en la posición n-1, donde n es el tamaño del *array*

Por ejemplo:

La declaración del array:

```
int[] notas = {4, 9, 7, 5, 10};
```

Para poder recorrer un array se necesita conocer su tamaño y utilizar un bucle, normalmente for:

```
for (int i=0; i<=notas.length - 1; i++)
   System.out.println("notas[" + i + "] es " + notas[i]);
```

El atributo *length* almacena el tamaño de un *array* (notas.length = 5). El siguiente ejemplo sería equivalente:

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

Y la salida del *array* por consola sería:

```
notas[0] es 4
notas[1] es 9
notas[2] es 7
notas[3] es 5
notas[4] es 10
```

Otra forma de recorrer un *array* es utilizar el bucle "*for-each*":

```
for (tipo variable : array)
{
    declaraciones usando variable;
}
```

Ejemplos:

```
for (int nota : notas)
   System.out.println(nota);
```

```
for (String dia: diasLaborables)
   System.out.println(dia);
```

2.2. Arrays multidimensionales

Un *array* de dos dimensiones es un *array* que contiene otro *array* en cada uno de sus elementos.

Por ejemplo, un *array* de dos dimensiones de números enteros de tamaño 2 x 3 se declara:

```
int[][] notas = new int[2][3];
```

Un *array* de dos dimensiones tiene forma de tabla. Para indicar la posición de uno de sus elementos es necesario indicar dos índices, uno para la fila y otro para la columna. La fila indica la posición en el primer *array* y la columna la posición en el segundo *array*.

Para mostrar los elementos del *array* notas es necesario utilizar dos bucles *for* anidados.

```
for (int i = 0; i < notas.length; i++)
   for (int j = 0; j < notas[i].length; j++)
     System.out.println("notas[" + i + "][" + j + "] " +
                                        "es " + notas[i][j]);
```

También se puede utilizar un *for-each:*

```
for (int[] fila : notas)
   for (int nota : fila)
      System.out.println(nota);
```

Para inicializar un array de dos dimensiones tendríamos la siguiente opción:

```
private static void inicializaMatrices() {
    int[][] mat = new int[][] {
            {1, 2, 3, 4, 5},
            {1, 2, 3, 4, 5},
            {1, 2, 3, 4, 5},
            {1, 2, 3, 4, 5},
            {1, 2, 3, 4, 5},
            {1, 2, 3, 4, 5},
            {1, 2, 3, 4, 5},
            {1, 2, 3, 4, 5},
            {1, 2, 3, 4, 5}
    };

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

En general, un *array* multidimensional es aquel que dentro de uno de sus elementos almacena uno o más *arrays* de objetos. Por ejemplo, un *array* de tres dimensiones de números enteros de tamaño 3 x 3 x 3 se declara:

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

Para poder recorrerlo, se necesitaría un *for* anidado por cada dimensión del *array*.

Por ejemplo, para su inicialización:

```
int num = 1;

for (int i=0; i<numeros.length; i++)
   for (int j=0; j<numeros[i].length; j++)
        for (int k=0; k<numeros[j].length; k++)
         numeros[i][j][k] = num++;
```

Y para mostrar su información:

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

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

Ojo: en el último bucle, podemos poner el tamaño "números[j].length" porque todas las dimensiones son iguales, si no fuera así, tendríamos que tener el índice concreto para las dos dimensiones anteriores: "números[i][j].length".

La salida por consola sería:

```
 1  2  3
 4  5  6
 7  8  9


10 11 12
13 14 15
16 17 18

19 20 21
22 23 24
25 26 27
```

Para inicializar un array de 3 dimensiones:

```
int[][][] arr3 = new int[][][] {
        {
            {1, 2},
            {3, 4},
            {3, 4},
            {3, 4},
            {3, 4}
        },
        {
            {1, 2},
            {3, 4},
            {3, 4},
            {3, 4},
            {3, 4}
        },
        {
            {1, 2},
            {3, 4},
            {3, 4},
            {3, 4},
            {3, 4}
        },
        {
            {1, 2},
            {3, 4},
            {3, 4},
            {3, 4},
            {3, 4}
        }
};
```

Para recorrerlo hay que tener en cuenta el “anidamiento de tamaños”:

```
for(int i=0; i < arr3.length; i++) {
    for (int j = 0; j < arr3[i].length; j++) {
        for (int k = 0; k < arr3[i][j].length; k++) {
            System.out.print(arr3[i][j][k]);
        }
        System.out.println();
    }
    System.out.println();
}
```

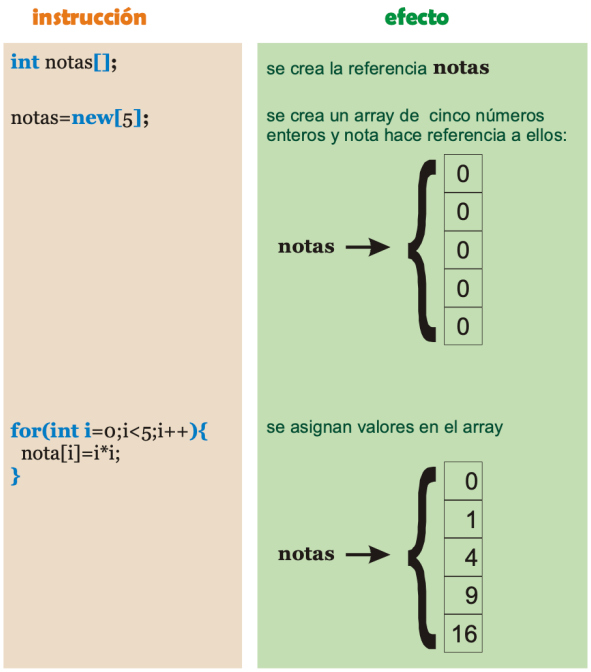
3. Particularidades y características especiales de los arrays

A un *array* se le puede inicializar las veces que haga falta:

```
int notas[]=new notas[16];
...
notas=new notas[25];
```

Pero hay que tener en cuenta que el segundo *new* hace que se pierda el contenido anterior. De hecho elimina ese contenido.

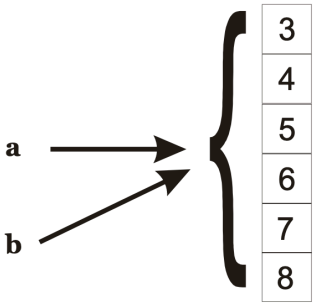
Paso a paso, las instrucciones de uso de un *array* tienen este efecto:



Otra característica importante es la asignación de un array a otro (solo si son del mismo tipo):

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

Lo que ocurre en el código anterior es que tanto a como b hacen referencia al mismo *array* :



Esta asignación provoca que cualquier cambio en a también cambie el *array* b (ya que, de hecho, es el mismo *array*). Ejemplo:

```
int a[]={3,3,3};
int b[];
b= a;
b[0]=8;
System.out.println(a[0]);//Escribirá el número 8
```

Otro detalle importante es que el operador de igualdad (==) se puede utilizar con arrays, pero no compara el contenido sino si las referencias señalan al mismo *array*. Es decir:

```
int a[]={3,3,3};
int b[]={3,3,3};
System.out.println(a==b);   //escribe false, aunque ambos arrays
                            //tienen el mismo contenido
int c[]=b;
System.out.println(b==c);   //escribe true
```

Ojo, no hay que confundir la comparación de arrays con la comparación de cada elemento de un *array*.

4. Recursividad

Se dice que un Procedimiento o una Función es recursivo cuando se llama o invoca así mismo.

La recursividad es una alternativa realmente "potente" a las estructuras repetitivas pero tiene el inconveniente de que consume mucha memoria principal (memoria RAM).

Ejemplo: Implementar la ***función potencia (solución normal, no recursiva).***

La potencia de un número ($base^{exp}$) se calcula como la multiplicación de la base tantas veces como indique el exponente. Ejemplos:

$2^3 = 2 * 2 * 2 = 8$

$3^4 = 3 * 3 * 3 * 3 = 81$

```
Funcion potencia (base: entero, exp: entero): entero
Var
        i, res: entero
Inicio
        res ← 1
        Desde i = 1 hasta exp hacer
                res ← res * base
        Fin Desde
        potencia ← res
Fin
```

```java
private static int potencia(int base, int exponente) {
    int resultado=1;

    for(int i=1; i <= exponente; i++) {
        // Ejecuto el bucle el número de veces indicado por el exponente
        resultado = resultado * base;
    }

    return resultado;
}
```

Ejemplo: Implementar la función **potencia de forma recursiva**.

La potencia de un número de forma recursiva se calcula como:

$$base^{exp} = base * base^{exp-1}$$

En la resolución del problema vuelve a aparecer el mismo problema: para calcular $base^{exp}$ hay que calcular el resultado de $base^{exp-1}$.

A la hora de dar una solución recursiva hay que indicar también una condición de fin de recursividad. De otra forma tendríamos un proceso infinito. En este caso la condición de fin de recursividad sería (**caso base**):

$$base^0 = 1$$

Cualquier número (base) elevado al exponente 0, su resultado por definición es 1 (condición de fin de recursividad).

Ejemplos:

$2^3 = 2 * 2^2$

$2^2 = 2 * 2^1$

$2^1 = 2 * 2^0$

$2^0 = 1$ (Condición de Fin de recursividad)

$3^4 = 3 * 3^3$

$3^3 = 3 * 3^2$

$3^2 = 3 * 3^1$

$3^1 = 3 * 3^0$

$3^0 = 1$ (Fin de recursividad)

Función potenciaRecursiva (base: entero, exp: entero): entero
Inicio
    Si (exp = 0) entonces
        potenciaRecursiva ←1
    Sino
        potenciaRecursiva ← base * potenciaRecursiva (base, exp-1)
    Fin si
Fin

```java
private static int potenciaRecursiva(int base, int exponente) {
    // Caso base
    if(exponente == 0) {
        // Cualquier número elevado a 0 es igual a 1
        return 1;
    } else {
        // En cada llamada, se descuenta uno al exponente
        // Ejemplo:
        //      3^4 = 3 * 3^3;
        //          3^3 = 3 * 3^2; ...
        return base * potenciaRecursiva(base, exponente-1);
    }
}
```