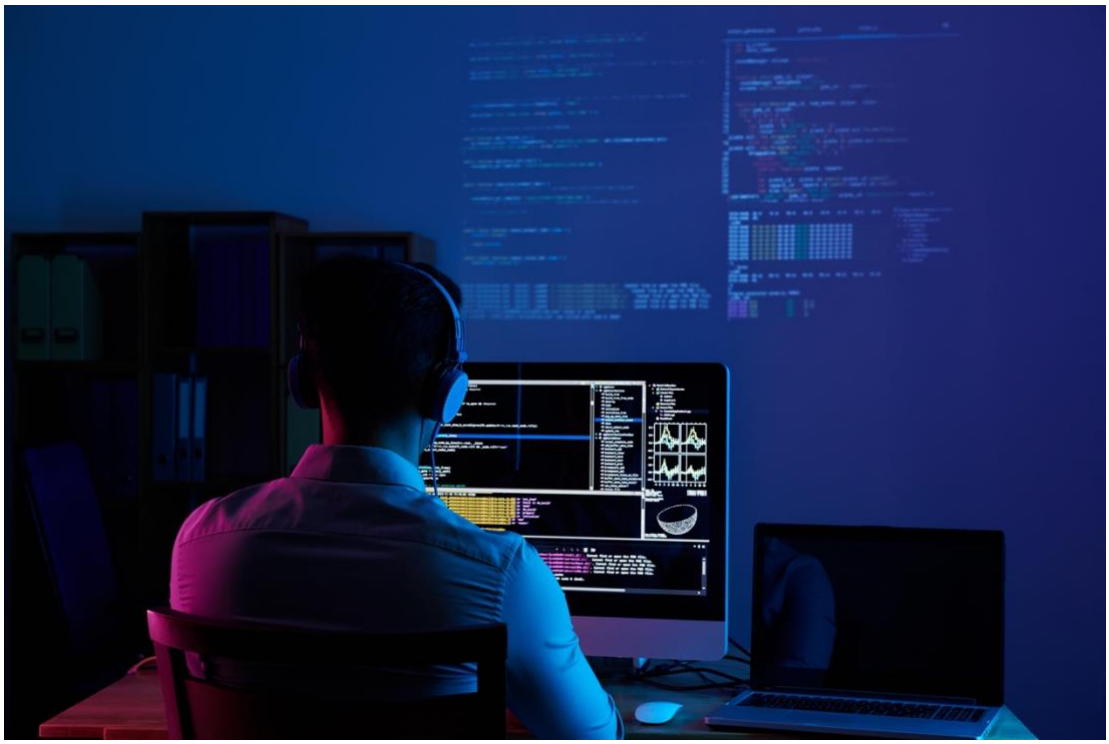
ESTRUCTURA DE ALMACENAMIENTO - ARRAYS

Programación



INDICE

ARRAY

Los arrays son estructuras que permiten guardar múltiples datos del mismo tipo.

Cuando se necesita procesar múltiples datos del mismo tipo, se pueden guardar en un array y procesar conjuntamente como una sola unidad. Es una forma conveniente de hacerlo cuando hay una gran cantidad de datos o cuando no se sabe a priori cuántos datos habrá.

Tienes que considerar un array como una colección de elementos del mismo tipo. Todos los elementos se guardan en la memoria de forma secuencial (uno detrás del otro).

Imagina un programa donde tenemos que procesar las notas de los alumnos. Para guardarlas en memoria podríamos utilizar diferentes variables de tipo float:

```
float nota1 = 5.6f;
float nota2 = 9.8f;
float nota3 = 7.75f;
float nota4 = 6.25f;
```

nota1	nota2	nota3	nota4
5.6f	9.8f	7.75f	6.25f

En casos como este, podemos utilizar un array para almacenar todas las notas en una única variable:

```
float [] notas = {5.6f, 9.8f, 7.75f, 6.25f};
```

notas	0	1	2	3
	5.6f	9.8f	7.75f	6.25f

Un array proporciona un único nombre para todos los elementos. La cantidad de elementos que se puede almacenar se establece cuando se crea el array y no se puede cambiar. Sí se puede cambiar un elemento guardado en el array.

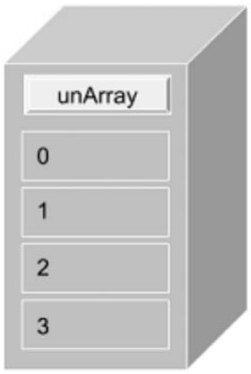
Para acceder a un elemento del array (para obtener su valor, o modificarlo) se debe utilizar su índice numérico. Los índices empiezan por 0.

El primer elemento de un array tiene índice 0, y el último tiene el índice igual al tamaño - 1.

Siguiendo la analogía de las cajas que vimos con las variables, podemos imaginar un array como una caja con múltiples cajones. La caja tiene un nombre identificador, y cada cajón tiene un número de índice:

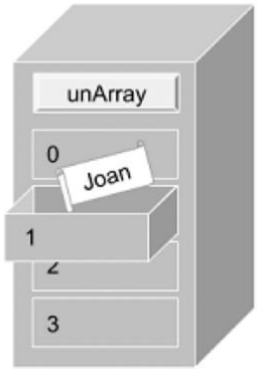
Por ejemplo, el siguiente array podemos guardar hasta 4 Strings:

String[] unArray = new String[4];



Para acceder a un cajón (elemento) del array, ponemos su índice entre corchetes [índice]:

unArray [1] = "Joan";



Podemos acceder para guardar un valor, y también para obtener el valor que hay guardado. Por ejemplo, para imprimirlo:

System.out.println(unArray[1]); // Joan

CREAR UN ARRAY

Para declarar un array debemos poner los caracteres [] después del tipo de los elementos del array:

```
tipoElementos [] nombreArray;
```

Por ejemplo, para declarar un array de enteros:

```
int [] arrayEnteros;
```

Para inicializar los valores de un array, podemos enumerarlos entre llaves {} y separados por comas:

```
int [] arrayEnteros = {23, 34, 45, 56};
String [] arrayStrings = { "this", "is", "an", "array", "of", "Strings"};
```

También se puede inicializar un array simplemente indicando su tamaño (la cantidad de elementos). En este caso el valor de los elementos del array será un valor predeterminado según el tipo.

```
tipoElementos [] nomArray = new tipoElementos [tamaño];
```

Para los tipos numéricos, el valor por defecto es 0, para los booleanos es false y para los Strings es el valor especial null.

```
int [] arrayEnters = new int [5]; // {0, 0, 0, 0, 0}
float [] arrayFloats = new float [5]; // {0.0, 0.0, 0.0, 0.0, 0.0}
char [] arrayChars = new char [5]; // {0, 0, 0, 0, 0}
boolean [] arrayBooleans = new boolean [5]; // {false, false, false, false}
String [] arrayStrings = new String [5]; // {null, null, null, null, null}
```

TAMAÑO DE UN ARRAY

El tamaño (o longitud) de un array se refiere al número de elementos que tiene.

Como hemos visto, un array no se puede cambiar de tamaño una vez creado.

Para obtener el tamaño de un array, se puede acceder a la propiedad especial length. Aquí tienes un ejemplo:

```
int [] arrayEnteros = {13, 25, 37, 49};

int tamaño = arrayEnteros.length; // cantidad de elementos del array

System.out.println (tamaño); // 4
```

Los índices de un array siempre van desde 0 hasta length - 1.

Al arrayEnteros del ejemplo anterior, los índices van de 0 hasta 3 (4 - 1).

Length	4
Valores	{13, 25, 37, 49}
índice	0 1 2 3

ACCEDER A LOS ELEMENTOS

Para acceder a un elemento de un array se pone el índice del elemento entre corchetes [] después del nombre del array.

```
nombreArray [indiceElemento]
```

De esta forma puedes obtener el valor de un elemento, o modificarlo.

```
String [] arrayStrings = { "hola", "que", "tal"};

// obtenemos el valor del elemento en la posición 0, y la imprimimos

System.out.println (arrayStrings [0]); // hola

// obtenemos el valor del elemento en la posición 1, y la asignamos a la variable elemento

String elemento = arrayStrings [1]; // elemento: "que"

// modificamos el valor del elemento en la posición 2

arrayStrings [2] = "que"; // arrayStrings: { "hola", "que", "que"}
```

Otro ejemplo:

```
int [] arrayEnteros = {111, 456, 888, 954};

System.out.println (arrayEnteros [2]); // 888

int elemento = arrayEnteros [3]; // 954

arrayEnteros [1] = 30000; // arrayEnteros: {111, 30000, 888, 954}
```

ARRAY INDEX OUT OF BOUNDS EXCEPTION

Si un programa trata de acceder a un elemento de un array utilizando un índice inferior a 0 o igual o superior al tamaño, el programa se detendrá con una excepción ArrayIndexOutOfBoundsException. Significa que el índice al que se está tratando de acceder está fuera de los límites del array:

```
int [] numeros = {13, 24, 35, 46};

System.out.println (numeros [-1]); // OutOfBounds, el índice más bajo es 0

System.out.println (numeros [4]); // OutOfBounds, el índice más alto se 3
```

Si ejecutamos este programa, se detendrá con la execepción ArrayIndexOutOfBoundsException:

```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: Index -1 out of bounds for length 4
```

	indices			
numeros	0	1	2	3
	13	24	35	46
	valores			

LEER UN ARRAY DE LA ENTRADA

Utilizando un bucle podemos leer una lista de datos de la entrada y guardarlas en un array.

Por ejemplo, la siguiente entrada consiste en dos líneas. La primera contiene el número de elementos, y en la segunda está la lista de elementos.

5

101 102 504 302 881

Podemos crear un array con length igual al número de elementos y luego utilizar un bucle para leer los elementos y guardarlos en el array. La variable y del bucle determina el índice (posición) del array donde se guardará el valor leído. Por lo tanto, la variable y deberá ir desde 0 hasta length - 1.

```
int len = scanner.nextInt ();
int [] array = new int [len];
for (int i = 0; i <len; i ++) {
        array [i] = scanner.nextInt ();
}
```

RECORRER UN ARRAY

Cuando decimos recorrer un array significa acceder uno por uno a cada uno de los elementos del array.

Podemos utilizar un bucle for con una variable y que vaya teniendo los valores desde 0 (la primera posición del array) hasta length - 1 (la última posición). Entonces, utilizaremos la variable y como índice, y así conseguiremos acceder a todos los elementos:

El siguiente código accede a cada uno de los elementos del array, para imprimirlos:

```
String [] palabras = { "vamos", "a", "recorrer", "este", "array"};

for (int i = 0; i <palabras.length; i ++) {
        System.out.println (i + "->" + palabras [i]);
}
```

```
0 -> vamos
1 -> a
2 -> recorrer
3 -> este
4 -> array
```

RECORRIDO DE ARRAYS CON FOR-EACH

A partir de la versión Java 5, el lenguaje Java incorpora una variante de la instrucción for, conocida como for-each, que facilita el recorrido de arrays y para la recuperación de su contenido, eliminando la necesidad de utilizar una variable de control que sirva de índice para acceder a las distintas posiciones.

En el caso de un array, el formato de la nueva instrucción for-each sería:

```
for(tipo variable:var_array){

        //instrucciones

}
```

donde, tipo variable representa la declaración de una variable auxiliar del mismo tipo que el array, variable que irá tomando cada uno de los valores almacenados en éste con cada iteración del for, siendo var_array la variable que apunta al array.

Por ejemplo, supongamos que tenemos el siguiente array:

```
int [] nums = {4, 6, 30, 15};
```

y queremos mostrar en pantalla cada uno de los números almacenados en el mismo.

Utilizando la versión tradicional de for, la forma de hacerlo sería:

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

Utilizado la nueva instrucción for-each, la misma operación se realizará de la siguiente forma:

```
for (int n:nums){
    System.out.println(n);
}
```

Obsérvese como, sin acceder de forma explícita a las posiciones del array, cada una de éstas es copiada automáticamente a la variable auxiliar n al principio de cada iteración.

OPERACIONES HABITUALES: BUSCAR, AÑADIR, INSERTAR, BORRAR

Algunas operaciones con datos pertenecientes a un array son especialmente frecuentes: buscar si existe un cierto dato, añadir un dato al final de los existentes, insertar un dato entre dos que ya hay, borrar uno de los datos almacenados, etc. Por eso, vamos a ver las pautas básicas para realizar estas operaciones, y una fuente de ejemplo.

BUSCAR

Para ver si un dato existe, habrá que recorrer todo el array, comparando el valor almacenado con el dato que se busca. Puede interesarnos simplemente saber si está o no (con lo que se podría interrumpir la búsqueda en cuanto aparezca una primera vez) o ver en qué posiciones se encuentra (para lo que habría que recorrer todo el array).

AÑADIR

Para poder añadir un dato al final de los ya existentes, necesitamos que el array no esté completamente lleno, y llevar un contador de cuántas posiciones hay ocupadas, de modo que seamos capaces de guardar el dato en la primera posición libre.

INSERTAR

Para insertar un dato en una cierta posición, los que queden detrás deberán desplazarse "hacia la derecha" para dejarle hueco. Este movimiento debe empezar desde el final para que cada dato que se mueve no destruya el que estaba a continuación de él. También habrá que actualizar el contador, para indicar que queda una posición libre menos.

BORRAR

Si se quiere borrar el dato que hay en una cierta posición, los que estaban a continuación deberán desplazarse "hacia la izquierda" para que no queden huecos. Como en el caso anterior, habrá que actualizar el contador, pero ahora para indicar que queda una posición libre más.

Vamos a verlo con un ejemplo:

```
int[] datos = {10, 15, 12, 0, 0};
int  capacidad = 5;        // Capacidad maxima del array
int  cantidad = 3;         // Número real de datos guardados
int  i;    // Para recorrer los elementos

// Mostramos el array
for (i=0; i<cantidad; i++){
        System.out.println(""+datos[i]);
}

// Buscamos el dato "15"
for (i=0; i<cantidad; i++){
         if (datos[i] == 15){
                System.out.println("15  encontrado  en  la  posición  "+( i+1));
        }
}

// Añadimos un dato al final
System.out.println("Añadiendo  6  al  final");
if (cantidad < capacidad)
{
        datos[cantidad] = 6;
        cantidad++;
}

// Y volvemos a mostrar el array
for (i=0; i<cantidad; i++){
        System.out.println (" "+datos[i]);
}

// Borramos el segundo dato
System.out.pritnln("Borrando  el  segundo  dato");
int posicionBorrar = 1;
for (i=posicionBorrar; i<cantidad-1; i++) {
        datos[i] = datos[i+1];
}
cantidad--;

// Y volvemos a mostrar el array
for (i=0; i<cantidad; i++){
        System.out.println(" "+datos[i]);
}
```

```
// Insertamos 30 en la tercera posición
 if (cantidad < capacidad)
{
        Systen.println("Insertando 30 en la posición 3");
        int posicionInsertar = 2;
        for (i=cantidad; i>posicionInsertar; i--) {
                datos[i] = datos[i-1];
        }
        datos[posicionInsertar] = 30;
        cantidad++;
}


// Y volvemos a mostrar el array
for (i=0; i<cantidad; i++){
        System.out.println(" "+datos[i]);
}
```

que tendría como resultado:

10 15 12

15 encontrado en la posición 2

El máximo es 15

Añadiendo 6 al final 10 15 12 6

Borrando el segundo dato 10 12 6

Insertando 30 en la posición 3

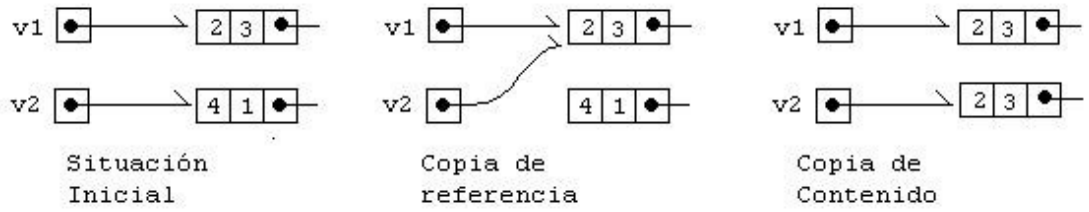
10 12 30 6

Este programa "no dice nada" cuando no se encuentra el dato que se está buscando. Se puede mejorar usando una variable "booleana" que nos sirva de testigo, de forma que al final nos avise si el dato no existía (no sirve emplear un "else", porque en cada pasada del bucle "for" no sabemos si el dato no existe, sólo sabemos que no está en la posición actual).

COPIA DE VECTORES

Para copiar vectores no basta con igualar un vector a otro como si fuera una variable simple.

Si partimos de dos vectores v1, v2 e hiciéramos v2=v1, lo que ocurriría sería que v2 apuntaría a laposición de memoria de v1. Eso es lo que se denomina un copia de referencia:



Si por ejemplo queremos copiar todos los elementos del vector v2 en el vector v1, existen dos formas para hacerlo:

- **Copiar los elementos uno a uno**

```
for (i = 0; i < v1.length; i++)
    v2[i] = v1[i];
```

- **Utilizar la función arraycopy**

```
System.arraycopy(v_origen, i_origen, v_destino, i_destino, length);
```

v_origen: Vector orígen

i_origen: Posición inicial de la copia

v_destino: Vector destino

i_destino: Posición final de la copia

length: Cantidad de elementos a copiar

```
// Copiamos todos los elementos de v1 en v2
System.arraycopy(v1, 0, v2, 0, v1.length);
```

LA CLASE ARRAYS

En el paquete java.utils se encuentra una clase estática llamada Arrays. Esta clase estática permite ser utilizada como si fuera un objeto (como ocurre con Math). Esta clase posee métodos muy interesantes para utilizar sobre arrays.

Su uso es:

Arrays.método(argumentos);

Algunos métodos son:

- **fill** : permite rellenar todo un array unidimensional con un determinado valor. Sus argumentos son el array a rellenar y el valor deseado:

  Por ejemplo, llenar un array de 23 elementos enteros con el valor -1

  int valores[] = new int[23];

  Arrays.fill(valores,-1); // Almacena -1 en todo el array 'valores'

  También permite decidir desde que índice hasta qué índice rellenamos:

  Arrays.fill(valores,5,8,-1); // Almacena -1 desde el 5º a la 7º elemento

- **equals** : Compara dos arrays y devuelve true si son iguales (false en caso contrario). Se consideran iguales si son del mismo tipo, tamaño y contienen los mismos valores.

  Arrays.equals(valoresA, valoresB); // devuelve true si los arrays son iguales

- **sort** : Permite ordenar un array en orden ascendente. Se pueden ordenar sólo una serie de elementos desde un determinado punto hasta un determinado punto.

  int x[]={4,5,2,3,7,8,2,3,9,5};

  Arrays.sort(x); // Ordena x de menor a mayor

  Arrays.sort(x,2,5); // Ordena x solo desde 2º al 4º elemento

- **binarySearch** : Permite buscar un elemento de forma ultrarrápida en un array ordenado.

  Devuelve el índice en el que está colocado el elemento buscado. Ejemplo:

  int x[]={1,2,3,4,5,6,7,8,9,10,11,12};

  Arrays.sort(x);

  System.out.println(Arrays.binarySearch(x,8)); //Devolvería 7