Estructura de almacenamiento: arrays

Arrays

Los arrays, vectores o arreglos son estructuras de datos que representan un contenedor de elementos y que en Java son del mismo tipo de datos.

Podemos tener dos tipos de arrays dependiendo de si su tamaño es fijo o no.

- Arrays Estáticos: Su tamaño es fijo y se indica al crear el array. Al definir el array se reserva espacio en memoria para su uso. Estos arrays son los que veremos en este tema.
- Arrays Dinámicos: El tamaño del array (y el espacio de memoria en uso) varía a lo largo de la ejecución. Estos arrays se verán más adelante y son los ***ArrayList***.

Dimensiones de un array

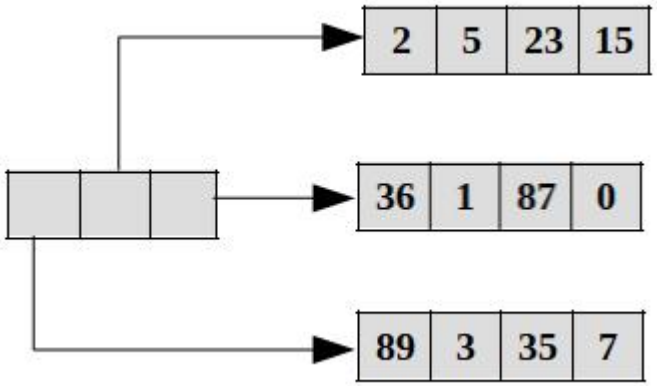
Los datos de un array también pueden almacenarse en varias dimensiones

1. Arrays Unidimensionales: Los elementos que se manejan están en una sola dimensión

Dimensiones de un array

Los datos de un array también pueden almacenarse en varias dimensiones

2. Arrays Multidimensionales: Los elementos del array son a su vez arrays de elementos (También se llaman Matrices)



Declaración y creación

La declaración

tipo dimensión nombre o *tipo nombre dimension*

la dimensión se indica mediante pares de corchetes []

```
short [] edades;
int [] pesos;
int precios[];
char [] caracteres;
int [] [] matriz;
```

Declaración y creación

Una vez declarado, se reserva espacio en memoria para los datos que va a almacenar

. La dimensión se indica mediante los pares de corchetes []

```
//Diferentes formas de instanciar los arrays
//Se crean con el tamaño definido y con el
//valor 'neutro' para el tipo (0 para los numéricos,
//'' para los char,"" para los string,...)
edades= new short[10];
pesos= new int[100];
precios = new int[10];
matriz = new int [4] [2];
caracteres = new char[5];
```

Inicialización y acceso

La inicialización de datos de un array se puede realizar o bien mediante asignación directa de cada posición en el array:

```
edades[0]=1;
matriz[0][1]=2;
caracteres[0]='a';
```

Inicialización y acceso

También se puede realizar una inicialización de todos los valores del array, pero debe hacerse junto a la declaración

```
char[] letras= {'a','b','c'};
byte [] [] arrayBidimensional= {
    {1,2,3},
    {4,5}
};
```

Modificación del tamaño del array

Una vez definido el tamaño del array, **no se puede modificar**. Lo que sí se puede hacer es indicar en tiempo de ejecución este tamaño.

```
Scanner sc= new Scanner(System.in);
System.out.println("Indica el tamaño:");
int tamano= sc.nextInt();
int []array= new int[tamano];
```

Métodos con arrays

El array es un objeto y como tal podemos utilizarlo como tipo en el envío de parámetros o en el retorno de datos de funciones y procedimientos.

```
//Llamada al método
int [][] retorno= miMetodo(edades,pesos);


// declaración del método
public static int[][] miMetodo(short[] array1, int[] array2) {
    /// ….
    int arrayRetorno[][] = new int[4][2];
    ///
    return arrayRetorno;
}
```

Métodos con arrays (2)

El caso de los objetos es diferente a los tipos primitivos ya que se pasa como valor (una copia) el puntero de la dirección de memoria del array indicado.

Modificar un valor del array lo actualiza en el 'original'

Cambiar a dónde apunta esa variable (con un new) no afecta al 'original'.

```java
public static void main(String[] args) {
    int[] array= new int[]{1,2,3};
    System.out.println("Antes de modificar");//[1, 2, 3]
    System.out.println(Arrays.toString(array));

    modificarArray(array);

    System.out.println("Después de modificar");
    System.out.println(Arrays.toString(array));//[1, 4, 3]
}

private static void modificarArray(int[] miArray){
    miArray[1]=4;

    miArray= new int[]{6,12,14};
}
```

Operaciones sobre arrays

Para realizar operaciones con arrays podemos utilizar métodos propios del objeto o métodos estáticos de la clase (accesibles desde **java.util.Arrays**)

Operaciones sobre arrays

Recorrido de un Array

Al igual que los String, los Arrays comienzan en la posición 0 y terminan en la anterior a su longitud.
Tenemos acceso a un campo length dentro de la instancia creada de un array (en arrays, al contrario que con los Strings, es una propiedad, no una función → no se invoca).

```
int longitud= edades.length;//Es 10. Es igual al tamaño del array
```

Si tenemos un array bidimensional, cada array que lo compone tendrá su propio campo length.

```
longitud = matriz.length; //4
longitud = matriz[0].length; //2
longitud = matriz[1].length; //2
```

Operaciones sobre arrays

La forma más sencilla de recorrer los arrays, al igual que los String es mediante un ***for***

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

for(int i=edades.length-1;i>=0;i--)
{
    System.out.println(edades[i]);
}
```

Operaciones sobre arrays

Con los arrays veremos una alternativa para el uso del for. El ***for each***

```
for(char letra:letras)
{
    System.out.print(letra);

}
```

Este uso del for así, lo que hace, es ir iterando los elementos de la colección indicada (letras) uno a uno e introduciéndolos en la variable definida en el for (letra).

Operaciones sobre arrays

Búsqueda en un array

La forma básica para encontrar si un elemento está en un array es recorrerlo e ir comparando el valor existente con aquél que queremos encontrar.
Sin embargo, la clase de utilidad Arrays proporciona el método binarySearch, para buscar un valor en un array ordenado:

```
String[] alumnos= {"Luis","Miguel","Rodrigo"};
int posicion=Arrays.binarySearch( alumnos, "Luis");
System.out.println("Encontrado en la posición " + posicion); //0
posicion=Arrays.binarySearch(alumnos, "Rodrigo");
System.out.println("Encontrado en la posición " + posicion); //2
posicion=Arrays.binarySearch(alumnos, "luis");
System.out.println("Encontrado en la posición " + posicion); //-4
```

Operaciones sobre arrays

Impresión de elementos de un array

Para imprimir los elementos de un array unidimensional, o bien lo recorremos e imprimimos o bien podemos utilizar el método *Arrays.toString()*:

```
System.out.println(Arrays.toString(alumnos));
```

```
[Luis, Miguel, Rodrigo]
```

Para arrays multidimensionales o bien imprimimos dimensión a dimensión o usamos el método *Arrays.deepToString()*

Operaciones sobre arrays

Ordenación de elementos de un array

La clase java.util.Arrays permite ordenar ascendentemente los elementos de un array unidimensional.

```
int[] numeros = {121,12,33,1,55};
java.util.Arrays.sort(numeros);
System.out.println(Arrays.toString(numeros));
```

```
[1, 12, 33, 55, 121]
```

Operaciones sobre arrays

Comparación de arrays

El método Arrays.equals compara si dos arrays tienen los mismos elementos, mientras que si usamos el comparador lógico == se compara si los dos arrays apuntan a la misma dirección de memoria. Con arrays, el método equals de la instancia funciona igual que ==.

```
int[] array1 = {4,2,5,7};
int[] array2 = {4,2,5,7};
int[] array3 = array1; //array3 y array1 apuntan a la misma dirección de memoria

System.out.println(array1==array2); //false
System.out.println(array1.equals(array2)); //false. En arrays es lo mismo que ==
System.out.println(java.util.Arrays.equals(array1, array2)); //true

System.out.println(array1==array3); //true
System.out.println(array1.equals(array3));  //true
System.out.println(java.util.Arrays.equals(array1, array2)); //true
```

Operaciones sobre arrays

Inicialización masiva de arrays

El método Arrays.fill permite rellenar un array unidimensional con los valores pasados por argumento.

```
int[][] miArray=new int[2][4];
Arrays.fill(miArray[0], 2);
Arrays.fill(miArray[1], 3);
System.out.println(Arrays.toString(miArray[0]));
System.out.println(Arrays.toString(miArray[1]));
```

```
[2, 2, 2, 2]
[3, 3, 3, 3]
```

Operaciones sobre arrays

Copia de arrays

Existen varias formas de realizar copias de los elementos de un array, dando diferentes resultados dependiendo del tipo de datos que componen dicho array.

1. Copia mediante el método ***clone***
   Devuelve una copia exacta del array desde el que se realiza la invocación.

Operaciones sobre arrays

```
int array[]= {4,7,14,9};
int arrayClonado[]=array.clone();
System.out.println(array==arrayClonado); //false. Diferentes posiciones de memoria
arrayClonado[0]=2;

System.out.println(Arrays.toString(array));
System.out.println(Arrays.toString(arrayClonado));
```

```
[4, 7, 14, 9]
[2, 7, 14, 9]
```

Si los datos del array a copiar son básicos (primitivos o envoltorio), los datos del nuevo array serán independientes de los datos del antiguo (cambiar los datos del nuevo no afecta a los del original).

Operaciones sobre arrays

```
Integer arrayBidimensional[][]= {{4,7,14,9},{2,5,8,14}};
Integer arrayBidimensionalClonado[][]=arrayBidimensional.clone();
System.out.println(arrayBidimensional==arrayBidimensionalClonado); //false. Diferentes posiciones de memoria
System.out.println(arrayBidimensional[0]==arrayBidimensionalClonado[0]); //true. Misma posición de memoria
System.out.println(arrayBidimensional[1]==arrayBidimensionalClonado[1]); //true. Misma posición de memoria

arrayBidimensionalClonado[0][1]=6;

System.out.println(Arrays.toString(arrayBidimensional[0]) + ' ' + Arrays.toString(arrayBidimensional[1]));
System.out.println(Arrays.toString(arrayBidimensionalClonado[0]) + ' ' + Arrays.toString(arrayBidimensionalClonado[1]));
```

```
[4, 6, 14, 9] [2, 5, 8, 14]
[4, 6, 14, 9] [2, 5, 8, 14]
```

Si los datos del array a copiar no son básicos (son objetos, como pueden ser arrays o instancias de clases), se copia la dirección de memoria, con lo que los cambios sobre un array repercuten en el otro. Es lo que se conoce como **Shallow copy**

Operaciones sobre arrays

Copia de arrays

2. Métodos ***Arrays.copyOf*** y ***Arrays.copyOfRange***
   Devuelven una copia (total o parcial) del array pasado como argumento.
   Al igual que con ***clone*** la copia de datos no primitivos resulta en una Shallow copy.

   type[] Arrays,copyOfRange (arrayOrigen,posInicio,posFinal);
   type[] Arrays,copyOf (arrayOrigen,longitud);

Operaciones sobre arrays

```
int a[]= {1,2,3,4,5};
int b[]=Arrays.copyOf(a, a.length);
int c[]=Arrays.copyOf(a, 2);
int d[]=Arrays.copyOfRange(a, 0, a.length);
int e[]=Arrays.copyOfRange(a, 2, 4);
System.out.println("Array a:" + Arrays.toString(a)); Array a:[1, 2, 3, 4, 5]
System.out.println("Array b:" + Arrays.toString(b)); Array b:[1, 2, 3, 4, 5]
System.out.println("Array c:" + Arrays.toString(c)); Array c:[1, 2]
System.out.println("Array d:" + Arrays.toString(d)); Array d:[1, 2, 3, 4, 5]
System.out.println("Array e:" + Arrays.toString(e)); Array e:[3, 4]
```

Cadenas de caracteres

Además de las formas de inicialización de Strings vistas en la primera evaluación, existe una sobrecarga del constructor de String que acepta un array de caracteres.

```
String cadena1="Hola mundo";
System.out.println(cadena1);

char arrayCaracteres[]= {'H','o','l','a',' ','M','u','n','d','o'};
String cadena2= new String(arrayCaracteres);
System.out.println(cadena2);
```

Métodos de String

Además de los métodos de String vistos anteriormente, existe un método interesante que, dado un patrón (en nuestro podemos usarlo con un caracter o cadena de caracteres), devuelve esa cadena como un array dividido por el patrón indicado.
Por ejemplo:

```
String cadena = "Esta es una cadena de texto";
String[] cadenaPartida= cadena.split(" ");
System.out.println(Arrays.toString(cadenaPartida));
```

En la salida tendremos un array en el cual cada posición es uno de los elementos en los que se ‘parte’

```
[Esta, es, una, cadena, de, texto]
```

Métodos de String

String[] split(String regex)	Busca las coincidencias de la expresion regular regex en la cadena y parte dicha cadena en subcadenas a partir de esas coincidencias.