Tema 8.- Vectores y matrices.La clase Arrays

Contenido

1 Vectores y matrices (arrays)

Ya vimos en el tema 2 la utilidad de los vectores y matrices para el manejo de conjuntos finitos de objetos o datos de un mismo tipo que se almacenan bajo el mismo nombre en posiciones consecutivas de memoria y se pueden localizar y diferenciar a través de un índice, de forma que se puede acceder a cada valor independientemente. Para Java, además un array es un objeto que tiene propiedades que se pueden manipular.

Gracias a los arrays se puede crear un conjunto de variables bajo un mismo nombre, la diferencia con una variable ordinaria es que será un número (índice del array) el que distinguirá a cada variable.

En el caso de las notas, se puede crear un array llamado `notas`, que representa a todas las notas de la clase. Para poner la nota del primer alumno se usaría `notas[0]`, el segundo sería `notas[1]`, etc. (los corchetes permiten especificar el índice en concreto del array).

La declaración de un array unidimensional se hace con esta sintaxis.

```
tipo nombre[];
```

Ejemplo:

```
double cuentas[]; //Declara un array que almacenará valores  doubles
```

Declara un array de tipo double. Esta declaración indica para qué servirá el array, pero no reserva espacio en la RAM al no saberse todavía el tamaño del mismo.

Tras la declaración del array, se tiene que inicializar. Eso lo realiza el operador `new`, que es el que realmente crea el array indicando un tamaño. Cuando se usa `new` es cuando se reserva el espacio necesario en memoria. Un array no inicializado es un array cuya dirección es `null`. Ejemplo:

```
int notas[]; //sería válido también int[] notas;
notas = new int[3]; //indica que el array constará de tres
//valores de tipo int
//También se puede hacer todo a la vez
//int notas[]=new int[3];
```

En Java (como en otros lenguajes) el primer elemento de un array es el cero. El primer elemento del array notas, es `notas[0]`. Se pueden declarar arrays a cualquier tipo de datos (enteros, booleanos, doubles, ... e incluso objetos).

En el ejemplo anterior se crea un array de tres enteros (con los tipos básicos se crea en memoria el array y se inicializan los valores, los números se inician a 0). Cuando se trata de un vector de objetos y una vez que se ha inicializado el vector, todos los elementos del vector que serán del tipo de la clase con la que se ha creado el vector, están inicializados a **`null`** al no haber sido instanciados.

Un array se puede inicializar las veces que haga falta:

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

Pero hay que tener en cuenta que el segundo new hace que se pierda el contenido anterior.

Un array se puede asignar a otro array (si son del mismo tipo):

```
int notas[];
int ejemplo[]=new int[18];
notas=ejemplo;
```

En el último punto, notas equivale a ejemplo. Esta asignación provoca que cualquier cambio en notas también cambie el array ejemplos. Es decir esta asignación anterior, no copia los valores del array, sino que notas y ejemplo son referencias al mismo array.

Ejemplo:

```
int notas[]={3,3,3};
int ejemplo[]=notas;
ejemplo= notas;

ejemplo[0]=8;
System.out.println(notas[0]);//Escribirá el número 8
```

Los arrays además pueden tener varias dimensiones como ya hemos visto en el tema 2. Entonces se habla de arrays de arrays (arrays que contienen arrays) Ejemplo:

```
int notas[][];
```

Se pueden utilizar más de dos dimensiones si es necesario.

Los arrays poseen un atributo público que permite determinar cuánto mide un array. Se trata de **length.** Ejemplo (continuando del anterior):

```
int notas[][]= new int[5][400]; // matriz de 5 filas por 400 columnas
System.out.println(notas.length); //Sale 5
System.out.println(notas[2].length); //Sale 400
```

2 La clase Arrays

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

Su uso es:

```
Arrays.método(argumentos);
```

Algunos de sus métodos son:

Resultado	Método	Descripción
void (**static**)	**fill**(tipo[] v, tipo valor)	Rellena un array con un valor que le indiquemos como parámetro. Los tipos de datos pueden ser boolean, byte, char, double, float, int, long, short u Object.
void (**static**)	**fill**(tipo[] v, int desde, int hasta, tipo valor)	Rellena los elementos del array comprendidos entre los índices que se pasan como parámetros con el valor indicado. Los tipos de datos pueden ser boolean, byte, char, double, float, int, long, short u Object.
boolean (**static**)	**equals**(tipo[] v1, tipo[] v2)	Compara dos arrays y devuelve true si son iguales. Se consideran iguales si son del mismo tipo, tamaño y contienen los mismos valores. Los tipos de datos pueden ser boolean, byte, char, double, float, int, long, short u Object.
int (**static**)	**binarySearch**(tipo[] v, tipo valorBuscado)	Busca un valor que le pasamos como parámetro, devuelve su posición. Debe estar ordenado. Un array y un valor. Los dos del mismo tipo. Estos pueden ser un byte, char, double, float, int, long, short u Object.
tipodato[ ] (**static**)	**copyOf**(tipo[] arrayOriginal, int nuevaLongitud)	Copia el array que se pasa como parámetro y lo devuelve en un nuevo array de tantos elementos como indique el segundo parámetro. Si se pasa del tamaño del array original, rellena con ceros las posiciones sobrantes, si la longitud es menor trunca el array original. Los tipos de datos pueden ser boolean, byte, char, double, float, int, long, short u Object.
tipodato[ ] (**static**)	**copyOfRange**(tipo[] arrayOriginal, int desde, int hasta)	Copia el rango especificado de elementos del arrayOriginal y lo devuelve en un nuevo array. Le indicamos los índices del primer y último elementos del rango que queremos copiar en el nuevo. Los tipos de datos pueden ser boolean, byte, char, double, float, int, long, short u Object.
void (**static**)	**sort**(tipo[] v)	Ordena el array que se pasa como parámetro en orden ascendente. Los tipos de datos pueden ser un byte, char, double, float, int, long, short u Object (*).
void (**static**)	**sort**(tipo[] v, int idesde, int hasta)	Ordena el array que se pasa como parámetro en orden ascendente. Ordena los elementos comprendidos entre los índices desde y hasta.
String (**static**)	**toString**(tipo[] v)	Devuelve una cadena con el contenido del array pasado como parámetro. Los tipos de datos pueden ser boolean, byte, char, double, float, int, long, short u Object.

(*) Si el método `sort` recibe como parámetro un vector de objetos, la clase a la que pertenecen los mismos debe implementar la interfaz `Comparable`. Más adelante veremos cómo ordenar un vector de objetos.

2.1 Ejemplos de algunos métodos

2.1.1 fill

Permite rellenar todo un array unidimensional con un determinado valor. Sus argumentos son el array a rellenar y el valor deseado:

```
int valores[]=new int[23];
Arrays.fill(valores,-1);//Todo el array vale -1
```

También permite decidir desde qué índice hasta qué índice rellenamos:

```
Arrays.fill(valores,5,8,-1);//Del elemento 5 al 7 valdrán -1
```

2.1.2 equals

Compara dos arrays y devuelve true si son iguales. Se consideran iguales si son del mismo tipo, tamaño y contienen los mismos valores.

```
int v1[]={4,5,2,3,7};
int v2[]={8,2,8,2,8};
Arrays.equals(int v1[],int v2[]);//devuelve false
```

Si hubiéramos utilizado **==** en vez de **.equals** para comparar los dos arrays también hubiera dado false pues las direcciones de los dos vectores son distintas en memoria.

```
boolean cierto;
int v1[]={4,5,2,3,7};
int v2[]={4,5,2,3,7};
cierto = Arrays.equals( v1, v2);//devuelve true
cierto = v1==v2;//devuelve false
```

2.1.3 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 v[]={4,5,2,3,7,8,2,3,9,5};
Arrays.sort(v);//Estará ordenado
Arrays.sort(v,2,5);//Ordena del 2º al 4º elemento
```

Si se tratase de objetos hay que establecer el criterio de ordenación pues lo que intentaría ordenar serían las direcciones de memoria de los objetos si no se ha indicado qué significa ordenar los objetos.

2.1.4 binarySearch

Permite buscar un elemento de forma ultrarrápida en un array ordenado (en un array desordenado sus resultados son impredecibles). Devuelve el índice en el que está colocado el elemento o un valor negativo si no está en el array. Ejemplo:

```
int v[]={1,2,3,4,5,6,7,8,9,10,11,12};
Arrays.sort(v);
System.out.println(Arrays.binarySearch(v,8));//Da 7 que es la posición donde
                                             //está el 8
```

2.1.5 copyOf

Permite copiar el vector a otro con la dimensión que se establezca. Si la dimensión es menor, trunca los elementos a partir de la posición indicada. Si la dimensión es mayor, los nuevos elementos se inicializan según el tipo de elementos del vector. Esto permite redimensionar el vector.

```
int v[]={4,5,2,3,7,8,2,3,9,5};
v = Arrays.copyOf(v, 20);//v tendrá ahora 20 elementos y los 10 primeros se
//corresponden con los antiguos al guardar el resultado en el vector de origen
```

2.1.6 toString

Devuelve la lista completa de los elementos/objetos del vector. Si está toString definido para la clase de los objetos del vector, además veremos el contenido de los objetos, si no, sus direcciones.

```
int v[]={4,5,2,3,7,8,2,3,9,5};
System.out.println(Arrays.toString(v));

//La salida será : [4,5,2,3,7,8,2,3,9,5]
```

3 El método System.arraycopy

La clase System (en el paquete java.lang) también posee un método relacionado con los arrays, dicho método permite copiar un array en otro. Recibe cinco argumentos: el array que se copia, el índice desde el que se empieza a copiar en el origen, el array destino de la copia, el índice desde el que se copia en el destino, y el tamaño de la copia (número de elementos de la copia).

```
int uno[]={1,1,2};
int dos[]={3,3,3,3,3,3,3,3,3};
System.arraycopy(uno, 0, dos, 0, uno.length);
for (int i=0;i<=8;i++){
     System.out.print(dos[i]+" ");
} //Sale 1 1 2 3 3 3 3 3 3
```

4 Otros ejemplos

Ejemplo de uso de sort para arrays

```
import java.util.Random;
import java.util.Arrays;

public class PruebaClaseArrays {
   public static void main(String[] args) {
      int indice, numElementos;

      Random azar = new Random();
      long instanteInicio, instanteFinal;

      // Generamos vectores con 1000000, 1100000, 1200000 elementos …

     for (numElementos = 1000000; numElementos <= 2000000; numElementos = numElementos + 100000) {
         // genero el vector
         int vector[] = new int[numElementos];
         for (indice = 0; indice < vector.length; indice++) {
            vector[indice] = azar.nextInt(11);
         }

         // devuelve el número de milisegundos desde el 1 de Enero de 1970
         instanteInicio = System.currentTimeMillis();

         // ordeno
         Arrays.sort(vector);
         instanteFinal = System.currentTimeMillis();
         System.out.println("tiempo para ordenar :" + numElementos
               + "  elementos:" + (instanteFinal - instanteInicio)
               + " milisegundos");
     }

   }

}
```

Ejemplo de generación aleatoria de datos en vectores y matrices

```
import java.util.Random;
import java.util.Arrays;
import java.util.Scanner;
public class Vectores {

	public static void main(String[] args) {
		int indice, z, filas, columnas;
		Random azar = new Random();
		int vector[] = new int[10];// vector de 25 elementos
		Scanner teclado = new Scanner(System.in);
		System.out.println("Valor de filas para la tabla:");
		filas = teclado.nextInt();
		System.out.println("Valor de columnas para la tabla:");
		columnas = teclado.nextInt();
		int tabla[][] = new int[filas][columnas];
		// genero el vector
		for (indice = 0; indice < vector.length; indice++) {
			vector[indice] = azar.nextInt(11);
		}

		mostrarArray(vector); // imprimo el vector

		//lo ordeno y lo vuelvo a imprimir
		Arrays.sort(vector);
		System.out.println("\nVector ordenado:");
		mostrarArray(vector);

		// genero la tabla
		for (indice = 0; indice < tabla.length; indice++){
			for (z = 0; z < tabla[0].length; z++) {
				tabla[indice][z] = azar.nextInt(41) + 150;
			}
		}
		//imprimo tabla
		System.out.println("\nValores de la tabla:\n");
		mostrarArray(tabla); //metodo sobrecargado

		System.out.println("\nNumero de filas: " + tabla.length);
		System.out.println("Numero de columnas " + tabla[0].length);
	}//main

	public static void mostrarArray(int v[]){
		int indice;
		for (indice = 0; indice < v.length; indice++) {
			System.out.println("Valor del elemento:" + indice + " es " + v[indice]);
		}
	}//mostrarArray

	public static void mostrarArray(int tabla[][]){
		int fila, columna;
		for (fila = 0; fila < tabla.length; fila++) {
			for (columna = 0; columna < tabla[0].length; columna++) {
				System.out.printf("%6d",tabla[fila][columna]);
			}
			System.out.println();
		}
	}//mostrarArray
} //class
```