Tema 3.- La clase Arrays

Contenido

1. Introducción

Cuando en un programa tenemos que trabajar con un conjunto de datos del mismo tipo que se van a tratar de forma global en un proceso homogéneo que se repetirá, no sería operativo definir una variable distinta para cada elemento del conjunto pues habría que escribir el programa nombrando a todas y cada una de las variables.

Un **array o vector** es una estructura de datos que nos permite almacenar una gran cantidad de datos de un mismo tipo bajo el nombre que asignemos al array. El tamaño de los arrays es el número de elementos que se van a definir y se declara en un primer momento. Este valor no se puede cambiar sin que se pierda el contenido del array definido.

A los datos almacenados en un array se les denomina elementos; al número de elementos de un array se les denomina dimensión, tamaño o rango del vector. El tamaño se especifica con una expresión entera positiva encerrada entre paréntesis cuadrados.

En términos generales un array unidimensional puede expresarse como:

```
tipo-dato nombreArray[]; // Se declara el array
nombreArray = new tipo-dato[N]; // Se crea el array
```

donde:

- "tipo-dato" es el tipo de dato primitivo, String o de una clase predeterminada.
- "nombreArray" es el nombre del array por el que nos vamos a referir a él.
- "N" una expresión entera positiva que indica el número de elementos del array.

Ej
```
int numeros []; //también vale int [] numeros
numeros = new int[10]; //vector de 10 elementos de enteros
```

Hay otra forma de declarar y crear un array que consiste en hacer la declaración y asignarle los valores de los elementos separados por comas. El número de elementos que asignemos establecerá la dimensión del array.

```
tipo-dato nombreArray[]={dato_1, dato_2, dato_3,....,dato_n};
// vector con n elementos
```
Ej
```
int[] numeros={1,3,2,5,4,2,11,6,-3,0} // vector de 10 enteros
```

Para acceder a los elementos individuales de un array se emplea un índice que será un número entero no negativo que indicará la posición del elemento dentro del array.

Para referirse a una posición particular o elemento dentro del array, especificamos el nombre del array y el número de posición del elemento particular dentro del mismo, el índice. El índice comienza en 0 y termina en (tamaño-1) y se especifica entre corchetes [ ].

Así el elemento 7 de nuestro vector se podrá manejar o tratar declarando `numeros[6]`. Lo manejaremos como si fuera el nombre de una variable habitual del tipo de datos que sea `numeros[]`. El tratamiento de todos los elementos se podrá realizar denominado a cada elemento como `numeros[i]` donde i será un entero que pueda valer entre 0 y (tamaño-1).

Para referirnos a la colección completa lo haremos utilizando su nombre, vector.

Ej
```
System.out.println(numeros[6]); // imprimirá el 11
```

Para conocer o tratar el tamaño de vector, lo haremos accediendo a `numeros.length`, que nos facilitará el entero que indica el tamaño del vector.

2. Clase Arrays

Todos los arrays que podamos construir tienen en Java una clase que los puede manejar y que es la clase Arrays. Esta clase es de tipo estática por lo que para utilizar sus métodos (funciones) se les invocará de la siguiente forma:

`Arrays.metodo(nombreVector, otros parámetros)`.

En nombreVector es el vector que se va a manejar con el método indicado por ***`metodo`***.

3. Métodos de clase Arrays

3.1 copyOf

Este método permite crear un nuevo vector con el contenido que tuviera el vector que se pasa por parámetro con un tamaño que es el que se indica en el segundo parámetro. El vector que devuelve este método se puede asignar a la variable que definía el vector original con lo que estaremos cambiando el tamaño del vector original.

Si se pasan dos parámetros enteros tras el nombre del vector, lo que devolverá en este caso es el subvector formado por los elementos entre las posiciones desde y hasta.

3.2 fill

Este método rellena el vector que se pasa como parámetro con el valor establecido por valor. Si tiene tres parámetros además del vector, los dos siguientes a vector indican las posiciones que se van a cargar con el valor del tercer parámetro. La posición hasta queda excluida de la carga.

3.3 setAll

Este método permite cargar el vector pasado como parámetro con el resultado de la función generadora. Esta función podrá utilizar como valor para la generación del valor el índice del elemento para el que se esté generando el valor. La función se tiene que crear aparte. También se puede emplear notación lambda para definir la función:

```
numeros = new int[10];
Arrays.setAll(numeros, (i)-> i*i); // carga números con el cuadrado de
 cada posición
Arrays.setAll(numeros, (i)-> (int) (Math.random()*7)-3); // carga
 números con un valor aleatorio entre -3 y 3
```

3.4 sort

Este método permite ordenar el vector que se pasa como parámetro. Cuando la ordenación requiera de un criterio distinto al natural será necesario definir un método **Comparator** con el que se aplique el criterio con el que quiere ordenar.

3.5 toString

Este método devuelve una cadena con el contenido del vector separando el valor de los elementos por comas.

Result.	Método	Descripción
<T> [ ] (static)	Arrays.copyOf(<T> [ ] v, int longitud)	Devuelve un vector con el contenido de **v**, ampliado a la nueva longitud del tipo que se haya especificado en **longitud**. Si tiene que truncar elementos, los borra, si tiene que añadirlos, los inicializa.
<T> [ ] (static)	Arrays.copyOf(<T> [ ] v, int desde, int hasta)	Devuelve un vector con el contenido de **v**, empezando en el elemente **desde** y llegando al elemento **hasta**.
void (static)	Arrays.fill(<T> [ ] v, <T> valor)	Llena el vector **v** con el contenido de **valor**.
void (static)	Arrays.fill(<T> [ ] v, int desde, int hasta, <T> valor)	Llena el vector **v** entre las posiciones **desde** a **hasta** con el contenido de **valor**.
void (static)	Arrays.setAll(<T> [ ] v, IntFunction<T> generador)	Llena el vector **v** con el contenido de la función **generador**.
void (static)	Arrays.sort(<T> [ ] v)	Ordena el vector **v** en orden creciente natural de su contenido o el que defina según **compareTo()**.
void (static)	Arrays.sort(<T> [ ] v, Comparator<T> comp)	Ordena el vector **v** en orden de su contenido mediante el criterio de **comp**.
String (static)	Arrays.toString(<T> [ ] v)	Produce un String con el contenido del vector **v** en el que los elementos van separados por comas.

4. Ejemplo de algunas cosas de lo explicado en este tema

```
import java.util.Arrays;
public class PruebaArrays {
	public static void main(String[] args) {
// TODO Auto-generated method stub
		int numeros[];
		numeros = new int[10];
		// posiciones 3 a la 6 excluida con un 9
		Arrays.fill(numeros, 3, 6, 9);
		System.out.println(Arrays.toString(numeros));
		// rellena el vector con el cuadrado de cada posición
		Arrays.setAll(numeros, (i)-> i*i);
		System.out.println(Arrays.toString(numeros));
		// rellena el vector con un número aleatorio entre -3 y 3
		Arrays.setAll(numeros, (i)-> (int) (Math.random()*7)-3);
		System.out.println(Arrays.toString(numeros));
		// ordena el contenido generado en el paso anterior
		Arrays.sort(numeros);
		System.out.println(Arrays.toString(numeros));
	}// fin de main
}// fin de programa
```