FAMILIA PROFESIONAL:	**Informática y Comunicaciones**
CICLOS FORMATIVOS:	**Desarrollo de Aplicaciones Multiplataforma, Desarrollo de Aplicaciones Web**
MÓDULO:	**Programación**

UNIDAD 5: VECTORES Y MATRICES

CONTENIDOS

AUTORES: **Fernando Rodríguez Alonso**
Sonia Pasamar Franco

ÍNDICE DE CONTENIDOS

1. TABLAS O ARRAYS

Una **tabla** o ***array*** es una secuencia ordenada que está formada por uno o más datos pertenecientes al mismo tipo de datos. Los elementos de la tabla están almacenados en memoria de forma contigua y pueden ser accedidos de forma directa mediante índices de posición. En Java, la primera posición de cualquier tabla o *array* está referenciada con la posición 0.

Las tablas más comunes son las de una dimensión, denominadas vectores y las de dos dimensiones, denominadas matrices. Aunque se pueden utilizar tablas de cualquier dimensión, no es habitual trabajar con más de tres dimensiones, dado que el ser humano no es capaz de representar en el espacio más de tres dimensiones. A las tablas de tres dimensiones se les denomina cubos.

1.1. TABLAS DE UNA DIMENSIÓN (VECTORES)

Un vector es una secuencia de n elementos del mismo tipo dispuestos uno tras otro a los que se accede mediante su índice.

1.1.1. Declaración y Asignación de Valores

La declaración de un vector se realiza con la sintaxis:

```
tipoElemento[] nombreVector;
tipoElemento nombreVector[];
```

Donde `tipoElemento` es el tipo de dato del vector y `nombreVector` es la denominación que se le da al vector. Los corchetes denotan que se está declarando una tabla de una dimensión y pueden ir antes o después del nombre del vector.

Tras declarar el vector, es preciso indicar de cuántos elementos se compone, para lo que utiliza la sintaxis:

```
nombreVector = new tipoVector[cantidadElementos];
```

Donde cantidadElementos es el número de elementos que contendrá el vector.

Al declarar un vector no se realiza una reserva de memoria ya que se desconoce la cantidad de elementos que va a contener. Mediante el operador `new` precedido por el tipo de dato y con el número de elementos entre corchetes a continuación, se especifica la longitud del vector, que no podrá ser modificada en toda la ejecución del programa.

Mediante la siguiente expresión se declara un vector de enteros con 5 elementos.

```
int[] vectorEnteros = new int[5];
```

Para los vectores de tipo primitivo, tras la reserva de memoria, todos los elementos de tipo numérico tendrán el valor cero, los booleanos tendrán el valor false.

Mediante la siguiente expresión se declara un vector de enteros con unos valores iniciales dados.

```
int[] dias = {1, 2, 3, 4, 5, 6, 7};
```

1.1.2. Recorrido

Para acceder al contenido de un vector se indicará entre corchetes la posición del elemento, tanto para leer su valor como para asignar uno.

`System.out.println(dias[0]);` Mostraría por pantalla el valor 1, que se corresponde con la primera posición del vector, que es la posición cero.

`vectorEnteros[2] = 99;` Asignaría el valor 99 a la tercera posición del vector.

Los vectores tienen la propiedad `length` a la que se accede de la siguiente manera:

```
vectorEnteros.length
```

Para recorrer todos los elementos del vector se puede utilizar un bucle `for` que vaya desde la posición cero hasta la longitud del vector menos uno, de forma muy similar a la utilizada para recorrer una cadena de caracteres.

1.1.3. Copia de Vectores

No es posible realizar la asignación de un vector a otro vector, ya que, de hacerlo, ambos tendrían la misma referencia.

Dados dos vectores inicializados con valores distintos:

```
int[] vector1 = {1, 2, 3};
int[] vector2 = {4, 5, 6};
```

Se realiza una asignación del segundo vector sobre el primero.

```
vector1 = vector2;
```

Si se muestra el contenido del primer elemento de cada uno de los vectores:

```
System.out.println(vector1[0]); // Mostrará 4
System.out.println(vector2[0]); // Mostrará 4
```

Se obtendrá en ambos casos lo mismo, lo que hará pensar erróneamente que se ha realizado una copia de los valores, aunque lo que realmente ha sucedido es que ha hecho una copia de la referencia del segundo vector al primero.

Tras modificar el contenido del primer elemento del vector1 y mostrar de nuevo el contenido del primer elemento de cada uno de los vectores, se puede observar que ambos valores son iguales.

```
vector1[0] = 10;
System.out.println(vector1[0]); // Mostrará 10
System.out.println(vector2[0]); // Mostrará 10
```

Para copiar un vector en otro vector se puede utilizar el método estático `arraycopy` de la clase `System`. Este método recibe cinco parámetros: vector original a copiar, índice del vector origen desde el que se copia, vector destino de la copia, índice del vector destino donde se copia y número de elementos a copiar.

Para copiar el vector2 en el vector1 del ejemplo anterior:

```
System.arraycopy(vector2, 0, vector1, 0, 3);
```

1.2. TABLAS DE DOS DIMENSIONES (MATRICES)

Una matriz es una tabla de dos dimensiones o, dicho de otro modo, es un vector de vectores. Si la cantidad de elementos de las dos dimensiones es la misma, se dice que es una matriz cuadrada.

1.2.1. Declaración y Asignación de Valores

La declaración de una matriz se realiza con la sintaxis:

```
tipoElemento[][] nombreMatriz;
tipoElemento nombreMatriz[][];
```

Donde `tipoElemento` es el tipo de dato de los elementos de la matriz y `nombreMatriz` es la denominación que se le da a la matriz. Los corchetes denotan que se está declarando una tabla de dos dimensiones y pueden ir antes o después del nombre de la matriz.

Tras declarar la matriz, es preciso indicar de cuánto elementos se compone en cada dimensión, para lo que utiliza la sintaxis:

```
nombreMatriz = new tipoMatriz[cantidadFilas][cantidadColumnas];
```

Donde `cantidadFilas` es el número de elementos que contendrá la matriz en la primera dimensión y `cantidadColumnas` es el número de elementos que contendrá la matriz en la segunda dimensión.

1.2.2. Recorrido

El recorrido de las matrices es similar al recorrido de los vectores, pero al tener dos dimensiones, en lugar de utilizar un bucle `for`, se hace con dos bucles, uno para cada dimensión.

Si se piensa en una matriz como en una tabla de dos dimensiones, se puede determinar que una de las dimensiones son las filas y la otra son las columnas.

El ejemplo siguiente muestra como rellenar con todo 5 una matriz de 2 x 2:

```
int fila, columna;
for (fila = 0 ; fila < 2 ; fila++) {
     for (columna = 0 ; columna < 2 ; columna++) {
          matriz[fila][columna] = 5;
     }
}
```

1.2.3. Matrices Irregulares

Dado que una matriz se puede considerar un vector de vectores, es posible que una vez definida la dimensión del primer vector, haya un vector de distinta dimensión para cada una de las dimensiones del primer vector.

Con la siguiente sentencia se declara una matriz que contendrá tres elementos en una dimensión.

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

Con las siguientes expresiones se declaran las diferentes dimensiones de los vectores que están dentro de la matriz:

```
matrizIrregular[0] = new int[3];
matrizIrregular[1] = new int[2];
matrizIrregular[2] = new int[4];
```

Con el siguiente bucle, se rellenaría la matriz con valores aleatorios entre 1 y 10:

```
Random aleatorio = new Random();
int fila, columna;
for (fila = 0 ; fila < matrizIrregular.length ; fila++) {
     for (columna = 0 ; columna < matrizIrregular[fila].length ;
          columna++) {
           matrizIrregular[fila][columna] = aleatorio.nextInt(10) + 1;
     }
}
```

Mediante las siguientes instrucciones, se visualiza el contenido de la matriz:

```
int fila, columna;
for (fila = 0 ; fila < matrizIrregular.length ; fila++) {
     for (columna = 0 ; columna < matrizIrregular[fila].length ;
          columna++) {
           System.out.printf("%3d ", matrizIrregular[fila][columna]);
     }
     System.out.println();
}
```

2. CLASE ARRAYS

La clase **`Arrays`** del paquete **java.util** de Java contiene una gran variedad de métodos de clase para realizar operaciones sobre tablas o *arrays*.

2.1. OPERACIONES DE MANIPULACIÓN

Las **operaciones de manipulación de tablas o *arrays*** se realizan mediante métodos estáticos de la clase **`Arrays`**.

2.1.1. fill

El método **`fill`** permite rellenar un vector con un determinado valor. Recibe como parámetros el vector y el valor a asignar.

En el siguiente ejemplo, se rellena un vector de 23 elementos con el valor -1.

```
int[] vector = new int[23];
Arrays.fill(vector, -1);
```

Este método tiene una sobrecarga que permite indicar el intervalo de las posiciones a rellenar, el valor que se indica como final está fuera del intervalo.

En el siguiente ejemplo, se rellena el vector anterior con el valor -1 desde el elemento de la posición 5 hasta la 7.

```
Arrays.fill(vector, 5, 8, -1);
```

2.1.2. copyOf

El método **`copyOf`** permite copiar un vector a otro con la cantidad de elementos que se establezca. Recibe como parámetros el vector origen de la copia y la cantidad de elementos a copiar y devuelve el vector resultante.

Si la cantidad de elementos a copiar es menor que la dimensión del vector original, trunca los elementos a partir de la posición indicada.

Si la cantidad es mayor que la dimensión del vector original, los nuevos elementos se inicializan según el tipo de elementos del vector.

En el siguiente ejemplo, el vector resultante tendrá 5 elementos, que serán los 5 primeros del vector original.

```
int[] vector = {4, 5, 2, 3, 7, 8, 2, 3, 9, 5};
vector = Arrays.copyOf(vector, 5);
```

En el siguiente ejemplo, el vector resultante tendrá 20 elementos, los 10 primeros se corresponderán con los que tenía el vector original y el resto serán ceros.

```
int[] vector = {4, 5, 2, 3, 7, 8, 2, 3, 9, 5};
vector = Arrays.copyOf(vector, 20);
```

2.1.3. sort

El método **sort** permite ordenar un vector de tipos primitivos en orden ascendente. Tiene algunas sobrecargas, una de las cuales permite ordenar solamente los elementos de un intervalo dado.

En el siguiente ejemplo, el método sort recibe como parámetro un vector de enteros. No devuelve nada. Tras su ejecución, el vector estará ordenado de forma ascendente.

```
int[] vector = {4, 5, 2, 3, 7, 8, 2, 3, 9, 5};
Arrays.sort(vector);
```

En el siguiente ejemplo, el método sort recibe como parámetro un vector de enteros y dos posiciones del vector. No devuelve nada. Tras su ejecución, el vector tendrá ordenadas de forma ascendente las posiciones que van de la 2 a la 4.

```
Arrays.sort(vector, 2, 5);
```

2.2. OPERACIONES DE CONSULTA

Las **operaciones de consulta de tablas o *arrays*** se realizan mediante métodos estáticos de la clase **Arrays**.

2.2.1. binarySearch

El método **binarySearch** permite buscar un elemento de forma muy rápida en un vector ordenado. Es necesario que el vector esté ordenado, en caso contrario el resultado es impredecible. Devuelve el índice en el que está colocado el elemento o un valor negativo si no lo encuentra.

En el siguiente ejemplo, se imprimirá el valor 7, que es la posición en la que se encuentra el valor 8.

```
int[] vector = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12};
Arrays.sort(vector);
System.out.println(Arrays.binarySearch(vector, 8));
```

2.2.2. equals

El método **equals** compara dos vectores y devuelve true si son iguales. Se consideran iguales si son del mismo tipo, tamaño y contienen los mismos valores.

En el siguiente ejemplo se comparan dos vectores del mismo tipo y tamaño, pero con diferentes valores, lo que devolverá un booleano con el valor false y lo mostrará por consola.

```
int[] vector1 = {4, 5, 2, 3, 7};
int[] vector2 = {8, 2, 8, 2, 8};
System.out.println(Arrays.equals(vector1, vector2));
```

Si se utiliza == en lugar de equals para comparar los dos vectores lo que estaría comparando sería las direcciones de memoria de los vectores, que en este caso también serían distintas.

En el siguiente ejemplo se comparan dos vectores del mismo tipo, tamaño y valores. Si la comparación se hace con equals el resultado será true y si se hace con ==, el resultado será false.

```
boolean iguales;
int[] vector1 = {4, 5, 2, 3, 7};
int[] vector2 = {4, 5, 2, 3, 7};
iguales = Arrays.equals(vector1, vector2);
iguales = (v1 == v2);
```

2.2.3. toString

El método **`toString`** devuelve una cadena de caracteres con la lista completa de los elementos del vector.

El formato de salida contiene entre los valores del vector separados por comas y entre corchetes.

El siguiente ejemplo muestra por consola: `[4, 5, 2, 3, 7, 8, 2, 3, 9, 5]`

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

3. CASOS PRÁCTICOS

3.1. RECORRIDO DE UN VECTOR

El siguiente ejemplo muestra la declaración de un vector de enteros de 10 elementos.

En primer lugar, se recorre el vector para asignar un valor aleatorio entre 1 y 10 a cada uno de sus elementos. Las posiciones del vector van del cero hasta la longitud del vector menos uno. Para determinar la longitud del vector se hace uso de la propiedad `length`.

Tras rellenar el vector con valores aleatorios, se vuelve a recorrer para acceder de nuevo a todos sus elementos y mostrar por consola su valor.

```
import java.util.Random;

public class Ejemplo01 {

    public static void main(String[] args) {
        Random aleatorio = new Random();
        int[] vectorEnteros = new int [10];
        for(int i = 0; i < vectorEnteros.length; i++) {
            vectorEnteros[i] = aleatorio.nextInt(10) + 1;
        }
        for(int i = 0; i < vectorEnteros.length; i++) {
            System.out.println("Elemento:" + i + " = " + vectorEnteros[i]);
        }
    }

}
```

3.2. ORDENACIÓN DE UN VECTOR

El siguiente ejemplo muestra la declaración de un vector de enteros de 10 elementos y su posterior recorrido y asignación de valores aleatorios entre 1 y 10 a todos sus elementos.

En primer lugar, se muestra el contenido del vector usando el método `toString` de la clase Arrays.

A continuación, se realiza una ordenación del vector utilizando el método `sort` de la clase Arrays.

Finalmente, se vuelve a mostrar el contenido del vector para observar la ordenación.

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

public class Ejemplo02 {

    public static void main(String[] args) {
        Random aleatorio = new Random();
        int[] vectorEnteros = new int [10];

        for(int i = 0; i < vectorEnteros.length; i++) {
            vectorEnteros[i] = aleatorio.nextInt(10) + 1;
        }
        System.out.println("Antes: " + Arrays.toString(vectorEnteros));
        Arrays.sort(vectorEnteros);
        System.out.println("Después: " + Arrays.toString(vectorEnteros));
    }

}
```

3.3. RECORRIDO DE UNA MATRIZ

El siguiente ejemplo muestra la declaración de una matriz de números reales con 3 elementos en una dimensión y 2 elementos en la otra.

Para rellenar la matriz se utiliza un método auxiliar llamado `rellenarTabla`, que recibe como parámetro una matriz de elementos double y asigna valor a todos los elementos pidiendo los datos por teclado. Dado que este método tiene como parámetro una tabla, se copia la dirección de memoria donde se posiciona la tabla en el `main` y, por tanto, las modificaciones que se hagan dentro del método se verán reflejadas fuera del mismo.

Tras rellenar la matriz con datos, se llama al método auxiliar `mostrarTabla`, que recibe como parámetro una matriz de elementos double y la recorre para mostrar su contenido por consola.

Como puede observarse, en el recorrido, tanto para la asignación como para la lectura de los valores de cada uno de los elementos, se utilizan dos bucles for, uno para cada una de las dimensiones.

El bucle externo itera desde 0 hasta alcanzar la longitud de la matriz, que determina la cantidad de elementos en la primera dimensión (`tablaReales.length`).

El bucle interno itera para cada uno de los elementos de la primera dimensión, desde 0 hasta la cantidad de elementos que haya en dicha dimensión (`tablaReales[fila].length`).

```
import entrada.Teclado;

public class Ejemplo03 {

    public static void main(String[] args) {
        double[][] tablaReales = new double [3][2];

        rellenarTabla(tablaReales);

        System.out.println("Datos de la tabla");
        mostrarTabla(tablaReales);

    }

    private static void rellenarTabla(double[][] tablaReales) {
        for(int fila = 0; fila < tablaReales.length; fila++) {
            for(int columna = 0; columna < tablaReales[fila].length; columna++) {
                tablaReales[fila][columna] =
                        Teclado.leerReal("¿Real posicion["+ fila + "]["+ columna + "]? ");
            }
        }
    }

    private static void mostrarTabla(double[][] tablaReales) {
        for(int fila = 0; fila < tablaReales.length; fila++) {
            for(int columna = 0; columna < tablaReales[fila].length; columna++) {
                System.out.printf("%7.2f ",tablaReales[fila][columna]);
            }
            System.out.println();
        }
    }
}
```