Departamento de Ingeniería Telemática
Escuela Superior de Ingenieros
Universidad de Sevilla

MATRICES

1. OBJETIVO

El objetivo de esta práctica es proporcionar estructuras y clases que están presentes en el funcionamiento básico de la mayoría de los programas escritos en el lenguaje de programación Java. Esta práctica muestra el uso de las matrices en el lenguaje de programación Java. Las matrices dentro de este lenguaje son objetos.

2. DIRECTORIO DE LA PRÁCTICA

El código fuente de esta práctica se deberá colocar en el subdirectorio **`fp2.poo.matrices`** del directorio **`src`**. Coloque en él todos los programas de esta práctica. Descargue el código de los ejemplos de esta práctica de la plataforma virtual.

3. Introducción. Conceptos Básicos.

Una matriz es un grupo de variables del mismo tipo a las que se hace referencia con el mismo nombre. Se pueden crear matrices de cualquier tipo y pueden tener una o más dimensiones. A un elemento específico de una matriz se accede por su *índice*. Las matrices ofrecen un medio de agrupar información relacionada.

Nota: Las matrices en Java funcionan de manera diferente a como lo hacen en el lenguaje C.

3.1 Matrices unidimensionales.

Una **matriz unidimensional** es, básicamente, una lista de variables del mismo tipo. Para crear una matriz, primero se tiene que crear una variable matriz del tipo deseado. La forma general de una declaración de matriz unidimensional es.

```
tipo nombreMatriz [];
```

Donde **`tipo`** declara el tipo básico de la matriz, que determina el tipo de cada elemento de la misma. El siguiente ejemplo muestra la declaración de una matriz llamada **`diasDelMes`** con el tipo "*matriz de int*":

```
int diasDelMes[];
```

Aunque esta declaración establece que **`diasDelMes`** es una variable matriz, realmente no existe ninguna matriz. De hecho, el valor de **`diasDelMes`** es asignado a **`null`**, que representa una matriz sin valor.

Para unir **`diasDelMes`** con una matriz de enteros "real", se debe reservar espacio utilizando **`new`** (operador de reserva de memoria) y asignándolo a **`diasDelMes`**.

El operador **new** tiene la siguiente forma general cuando aparece con matrices unidimensionales:

```
nombreMatriz = new tipo[tamaño];
```

Donde **tipo** especifica el tipo de los datos que van a ser asignados, **tamaño** especifica el número de elementos de la matriz y **NombreMatriz** es la variable matriz a la que se asigna la nueva matriz. Cuando se utiliza **new** para reservar espacio para una matriz, es necesario especificar el tipo y el número de elementos que se quieren reservar. Los elementos de una matriz reservada con **new** son inicializados automáticamente a cero.

En este ejemplo se reserva una matriz de enteros con 12 elementos y se asigna a **diasDelMes:**

```
diasDelMes = new int [12];
```

Cuando se ejecuta esta sentencia, **diasDelMes** se refiere a una matriz de 12 enteros. Además, todos los elementos de la matriz estarán inicializados a cero.

En resumen, la obtención de una matriz es un proceso en **dos** pasos.

1. En primer lugar es necesario declarar una variable del tipo de matriz deseado.

2. En segundo lugar, utilizando el operador **new**, se tiene que reservar la memoria en la que estará la matriz y se asigna a la variable matriz. En Java, la memoria necesaria para cada matriz se reserva dinámicamente.

Una vez que se ha reservado memoria para una matriz, se puede acceder a un elemento específico de la misma especificando su índice entre corchetes (`[]`). Al primer elemento de una matriz le corresponde el índice cero. Por ejemplo, esta sentencia asigna el valor 28 al segundo elemento **diasDelMes**.

```
diasDelMes[1] = 28;
```

La siguiente línea escribe el valor almacenado en el elemento con índice 3.

```
System.out.println(diasDelMes[3]);
```

Para resumir de manera práctica todo lo que hemos visto hasta ahora, el siguiente programa crea una matriz con los números de los días de cada mes.

```
/*
 *  Fichero: EjemploMatriz.java
 *
 *  Fundamentos de Programación II. GITT.
 *  Departamento de Ingeniería Telemática
 *  Universidad de Sevilla
 *
 */


package fp2.poo.matrices;

/**
 * Descripción: Esta es una clase de ejemplo para utilizar
 *              una matriz en Java.
 *
 * @version versión 1.0 Abril 2011
 * @author  Fundamentos de Programación II
 */
public class EjemploMatriz {

    /*
     *  Descripción: Método main que declara una variable matriz,
     *               realiza la instanciación y accede a sus elementos.
     */
    public static void main(String args[ ]) {
        int diasDelMes [] = null;

        diasDelMes = new int [12];
        diasDelMes[0]  = 31;
        diasDelMes[1]  = 28;
        diasDelMes[2]  = 31;
        diasDelMes[3]  = 30;
        diasDelMes[4]  = 31;
        diasDelMes[5]  = 30;
        diasDelMes[6]  = 31;
        diasDelMes[7]  = 31;
        diasDelMes[8]  = 30;
        diasDelMes[9]  = 31;
        diasDelMes[10] = 30;
        diasDelMes[11] = 31;
        System.out.println("Abril tiene " + diasDelMes[3] + " dias." );
      }
}
```

Cuando se ejecuta este programa se imprime el número de días de Abril. Al primer elemento de una matriz le corresponde el índice cero, por lo que el número de días de Abril es `diasDelMes[3]` o, lo que es lo mismo 30.

Ejercicios.

1. Utilice el siguiente makefile para probar el código de `EjemploMatriz.java` (este makefile se proporciona con los ficheros de la práctica).

```
#
#   Makefile de ejemplo para la compilación, creación del jar y ejecución
#

CLASE=EjemploMatriz
RUTACLASE=fp2/poo/matrices/

DIRSRC=./src/
DIRBIN=./bin/
DIRJAR=./jar/
PRINCIPAL=fp2.poo.matrices.EjemploMatriz

ejecutaJ: $(DIRJAR)$(CLASE).jar
	java -jar $(DIRJAR)$(CLASE).jar

$(DIRJAR)$(CLASE).jar: $(DIRBIN)$(RUTACLASE)$(CLASE).class
	jar cvfe  $(DIRJAR)$(CLASE).jar $(PRINCIPAL)  -C  $(DIRBIN)  $(RUTACLASE)$(CLASE).class

$(DIRBIN)$(RUTACLASE)$(CLASE).class: $(DIRSRC)$(RUTACLASE)$(CLASE).java
	javac -g -Xlint -d $(DIRBIN) $(DIRSRC)$(RUTACLASE)$(CLASE).java
```

2. Modifique el fichero **`EjemploMatriz.java`** para mostrar el número de días que tiene el mes de Enero y Diciembre.

3.2 Más sobre matrices.

Como se muestra a continuación, es posible combinar la declaración de una variable matriz con la operación de reservar memoria.

```
int diasDelMes [] = new int [12];
```

Esta es la forma que normalmente se utiliza en los programas Java. Las matrices se pueden inicializar cuando son declaradas. El mecanismo es similar al que se utiliza para inicializar los tipos simples. Un **inicializador de matriz** es una lista de expresiones separadas por comas y entre llaves ({}). Las comas separan los valores de los elementos de la matriz. Se creará una matriz lo suficientemente grande para contener el número de elementos que se especifiquen en la inicialización de la matriz. No es necesario utilizar el operador **`new`**.

Por ejemplo, para almacenar el número de días de cada mes, el siguiente código crea una matriz de enteros.

```
//version mejorada
class EjemploMatriz {
     public static void main(String args[ ]) {
          int diasDelMes [] =
              {31,28,31,30,31,30,31,31,30,31,30,31};
          System.out.println("Abril                              tiene"+
          diasDelMes[3]+"dias);
     }
}
```

Java comprueba estrictamente que no se intenta almacenar o hacer referencia accidentalmente a valores que estén fuera del rango de la matriz. El intérprete de Java se asegurará de que todos los índices de la matriz están dentro del rango correcto. En este aspecto, Java es diferente a C/C++,

ya que estos lenguajes no realizan comprobaciones en tiempo de ejecución. Por ejemplo, el intérprete de Java comprobará el valor de cada índice de **diasDelMes** para asegurarse de que está comprendido entre 0 y 11, ambos inclusive. Si se intenta acceder a elementos que están fuera del rango de la matriz, a números negativos o a números positivos mayores que la longitud de la matriz, se producirá un error en tiempo de ejecución.

Ejercicios.

3. Modifique el código fuente para que realice la inicialización de la matriz en una misma línea de código tal y como aparece en la versión mejora anterior.
4. Modifique el fichero **EjemploMatriz.java** para que acceda a una posición que esté fuera del rango de la matriz.

A continuación se muestra otro ejemplo que utiliza una matriz unidimensional. Este programa calcula la media de un conjunto de valores.

```
//Calcula la media de los valores de una matriz
class Media {
     public static void main(String args[ ]) {
          double nums[] = {10.1,11.2, 12.3,13.4,14.5};
          double result = 0;

          for (int i = 0; i<5; i++)
              result += nums[i];
          System.out.println("La media es "+ result/5);
     }
}
```

3.2 Matrices Multidimensionales.

En Java, las matrices multidimensionales realmente son matrices de matrices. Éstas, como se podría esperar, se parecen y funcionan como las matrices multidimensionales habituales. Sin embargo, como veremos, hay diferencias sutiles.
Para declarar una variable del tipo matriz multidimensional, es necesario especificar cada índice adicional utilizando otra pareja de corchetes ([]).

Por ejemplo, la siguiente línea declara la variable **dosDim** como matriz bidimensional.

```
int dosDim[][] = new int[4][5];
```

Este código reserva una matriz de 4 por 5 y la asigna a la variable **dosDim**. Internamente, esta matriz se implementa como una **matriz de matrices de int.** Conceptualmente, esta matriz se parece a la mostrada en la siguiente figura:

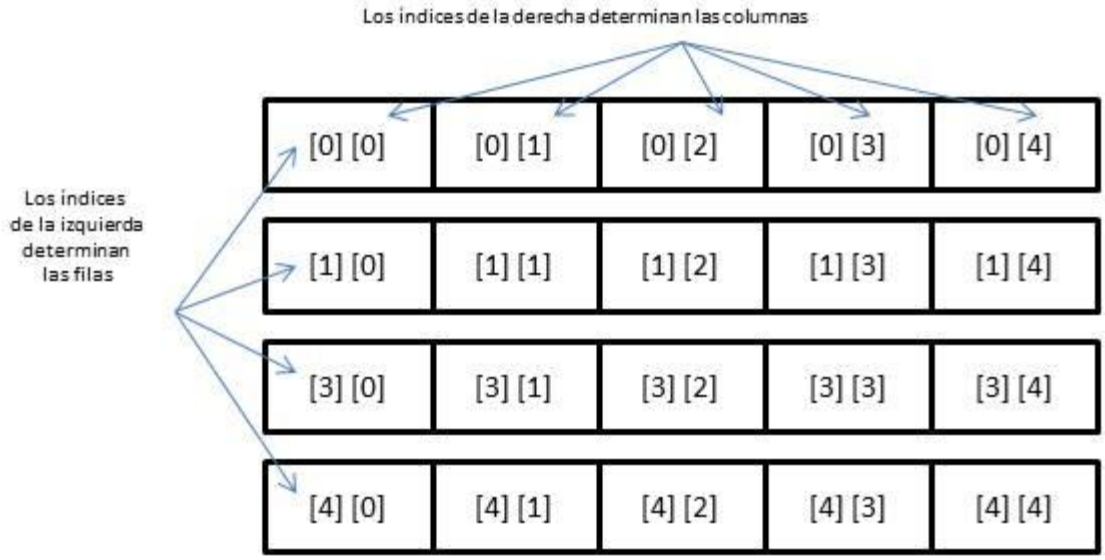

Figura 1: int biDim[][]=new[4][5]. Matriz bidimensional de 4x5.

El siguiente programa enumera los elementos de la matriz de izquierda a derecha y de arriba abajo y después escribe estos valores.

```
/*
 *  Fichero: MatrizBidimensional.java
 *
 *  Fundamentos de Programacion II. GITT.
 *  Departamento de Ingenieria Telematica
 *  Universidad de Sevilla
 *
 */


package fp2.poo.matrices;

/**
 * Descripcion: Esta clase mantiene una matriz bidimensional.
*
 * @version version 1.0 Abril 2011
 * @author  Fundamentos de Programacion II
 */
public class MatrizBidimensional {

    /** nœmero de filas de la matriz bidimensional */
    private int filas     = 0;

    /** numero de columnas de la matriz bidimensional */
    private int columnas  = 0;

    /** la matriz bidimensional */
    private int biDim[][] = null;

    /*
     *  Descripcion: Constructor de la clase.
     *               Inicializa los valores
     *  @param filas numero de filas de la matriz bidimensional
     *  @param columnas numero de columnas de la matriz bidimensional
     */
    public MatrizBidimensional( int filas, int columnas) {
        this.biDim    = new int[filas][columnas];
        this.filas    = this.biDim.length;
        this.columnas = columnas;
        this.inicializacion();
    }

    /*
     *  Descripcion: Inicializa los valores de la matriz.
```

```
     *
     *  @return Nada.
     */
    private void inicializacion(){
        for (int i = 0, k = 0; i < this.filas ; i++){
            for (int j = 0; j < this.columnas; j++) {
                this.biDim [i][j] = k;
                k++;
            }
        }
    }

    /*
     *  Descripcion: Muestra el contenido de una matriz.
     *
     *  @return Nada.
     */
    public void imprimeMatriz() {
        for (int i = 0; i < biDim.length; i++) {
            for (int j = 0; j < this.biDim[i].length; j++){
                System.out.print("\t" + this.biDim[i][j]);
            }
            System.out.println();
        }
    }

    /*
     *  Descripcion: Metodo getter.
     *
     *  @return Devuelve el numero de filas de la matriz.
     */
    public int getNumFilas(){
        return this.filas;
    }

    /*
     *  Descripcion: Metodo getter.
     *
     *  @return Devuelve el numero de columnas de la matriz.
     */

    public int getNumColumnas(){
        return this.columnas;
    }
}

class Bidimensional {
    public static void main(String args[ ]) {
        MatrizBidimensional matriz = new MatrizBidimensional (4, 5);
        matriz.imprimeMatriz();
    }
}
```

Al ejecutar este programa se generará la siguiente salida:

```
# java -classpath ./bin fp2.poo.matrices.Bidimensional
        0       1       2       3       4
        5       6       7       8       9
        10      11      12      13      14
        15      16      17      18      19
```

Cuando se crea la matriz multidimensional, sólo es necesario especificar la memoria que necesita la primera dimensión, es decir, la que está más a la izquierda. Después se pueden crear tablas unidimensionales para la segunda dimensión de manera independiente. Por ejemplo, el siguiente código crea la tabla para la primera dimensión de **`biDim`** en la misma línea de su declaración, y después, crea las tablas para la segunda dimensión:

```
int biDim[][] = new int[4][];
biDim[0] = new int [5];
biDim[1] = new int [5];
biDim[2] = new int [5];
biDim[3] = new int [5];
```

Nota: Al hacerlo de esta forma la longitud de cada matriz se puede establecer de forma independiente (no es necesario utilizar el mismo número de elementos para cada una de ellas). Aunque no se recomienda utilizar matrices multidimensionales donde la longitud de cada matriz sea diferente de manera generalizada, a menos que el contexto de uso de la matriz sea el más apropiado.

También es posible inicializar matrices multidimensionales. Para hacer esto, simplemente es necesario encerrar entre llaves el inicializador de cada dimensión.

El siguiente programa crea una matriz en la que cada elemento contiene el producto del índice de su fila por el índice de la columna. Observe que, además de valores literales, puede utilizar expresiones dentro de la inicialización de una matriz.

```
class DosDim {
    public static void main(String args[ ]) {
        int i, j;
        int m[][] = {
            { 0*0, 1*0, 2*0, 3*0},
            { 0*1, 1*1, 2*1, 3*1},
            { 0*2, 1*2, 2*2, 3*2},
            { 0*3, 1*3, 2*3, 3*3},
            };

        for (i = 0; i<4; i++){
             for (j = 0; j<4; j++) {
                System.out.print(m[i][j] + "\t");
             }
             System.out.println();
        }
    }
}
```

Cuando se ejecuta este programa se obtiene la siguiente salida.

```
0   0   0   0
0   1   2   3
0   2   4   6
0   3   6   9
```

Como se puede observar, cada fila de la matriz está inicializada con los valores especificados en las listas de inicialización.

Nota: Sintaxis alternativa para la declaración de una matriz

Aunque la sintaxis para declarar una matriz que se recomienda es la vista anteriormente se tiene esta otra forma: **`tipo[] nombreMatriz;`**. En este caso, los corchetes siguen al tipo y no al nombre de la matriz. Las siguientes dos declaraciones son equivalentes:

```
int a1[] = new int[3];
```

```
int[] a1 = new int[3];
```

Las siguientes declaraciones también son equivalentes:

```
char dosDim[][] = new char[3][4];
char[][] dosDim = new char[3][4];
```

Ejercicios.

5. Coloque el ejemplo de la clase bidimensional en el mismo paquete que la matriz unidimensional.
6. Implemente un makefile para la compilación, creación del fichero jar y ejecución del ejemplo de la clase bidimensional.
7. Use el código de la clase **`Cuenta`** que se muestra a continuación para crear desde el método **`main`** de otra clase, una tabla de 10 objetos del tipo **`Cuenta`**, unstancie 10 objetos y los inserte en la tabla.

```
/*
 *  Fichero: Cuenta.java
 *
 *  Fundamentos de Programacion II. GITT.
 *  Departamento de Ingenieria Telematica
 *  Universidad de Sevilla
 *
 */


package fp2.poo.utilidades;

/**
 * Descripcion: Esta es una clase que representa una cuenta bancaria.
 *              Mantiene una asociacion entre un usuario (nombre) y.
 *              su saldo (saldo).
 *
 * @version version 1.0 Abril 2011
 * @author  Fundamentos de Programacion II
 */
public class Cuenta {

    /** nombre es el atributo asociado al nombre de un usuario */
    private String nombre;

    /** saldo es el atributo asociado al dinero de una cuenta de usuario */
    private double saldo;

    /*
     *  Descripcion: Constructor de la clase.
     */
    public Cuenta ( String nombre, double saldo ) {
        this.nombre = nombre;
        this.saldo  = saldo ;
    }

    /*
     *  Descripcion: Este metodo configura el atributo nombre a un valor.
     */
    public void setNombre( String nombre ) {
        this.nombre = nombre;
    }
```

```
    /*
     *  Descripcion: Este metodo configura el atributo saldo a un valor.
     *
     */
    public void setSaldo( double saldo ) {
        this.saldo = saldo ;
    }

    /*
     *  Descripcion: Este metodo obtiene el atributo nombre.
     *
     */
    public String getNombre( ) {
        return this.nombre ;
    }

    /*
     *  Descripcion: Este metodo obtiene el valor del atributo saldo.
     *
     */
    public double getSaldo( ) {
        return this.saldo;
    }
}
```