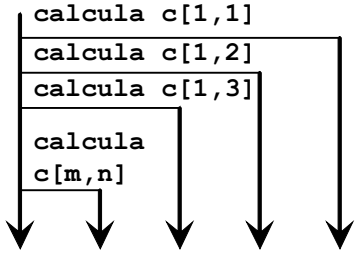
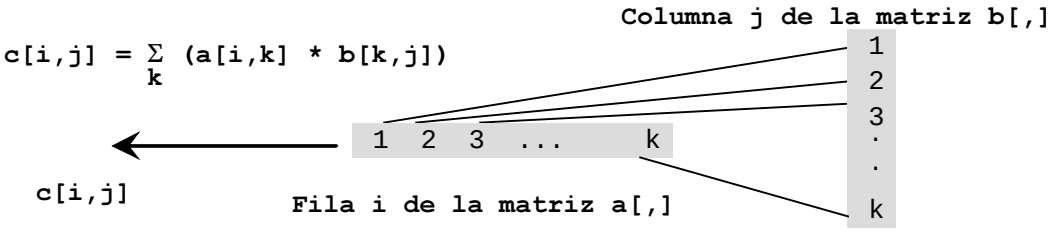
Ejercicio 1

La multiplicación de matrices bidimensionales se puede realizar mediante el uso de hilos. Desde un hilo de ejecución, se pueden crear varios hilos, uno por cada entrada de la matriz resultante. Dos sentencias **`for`** pueden utilizarse para crear **`m x n`** hilos.

```
for (int i = 1; i <= m; i++)
 for (int j = 1; j <= n; j++)
  Crea un hilo para calcular c[i,j];
```



Cada elemento de la matriz resultante **`c[i,j]`**, es el resultado de la suma de los elementos obtenidos de la multiplicación, elemento a elemento, de la fila de la primera matriz multiplicado por la columna de la segunda matriz. Es decir, dadas dos matrices $\mathbf{a}_{m\times k}$ (*m* filas y *k* columnas) y $\mathbf{b}_{k\times m}$ (*k* filas y *m* columnas), se desea calcular el producto de **`a`** por **`b`** en una matriz **`c`** de *m* filas y *n* columnas. La entrada de la fila *i* y la columna *j* es la suma del producto de los elementos correspondientes de la fila *i* de la matriz **`a`** y la columna *j* de la matriz **`b`** como se muestra en la siguiente figura.



Se pide implementar en el lenguaje de programación Java un programa que realice la operación indicada. De tal forma que la llamada y el resultado de la ejecución sean de la siguiente forma:

```
>java Principal 2 3 2 1 2 3 4 5 6 7 8 9 10 11 12
a =
 1.0 2.0 3.0
 4.0 5.0 6.0
b =
 7.0 8.0
 9.0 10.0
 11.0 12.0
c =
 58.0 64.0
 139.0 154.0
```

Donde se muestra invocación al intérprete de la clase **`Principal`**. A continuación, se proporcionan los tres primeros valores (es decir, **`2 3 2`**) son las dimensiones de las matrices. Es decir, **`a`** es una matriz de 2 filas y 3 columnas, y **`b`** es una matriz de 3 filas y 2 columnas. Se muestran las dos matrices leídas **`a`** y **`b`**, y **`c`** es la matriz de resultados.

`static int parseInt(String s) throws NumberFormatException` y **`static double parseDouble(String s) throws NumberFormatException`** Son métodos de la clases **`Integer`** y **`Double`**, respectivamente, que analiza un **`String`** para devolver un entero con signo (**`int`**) o un **`double`** respectivamente.
`java.lang.ArrayIndexOutOfBoundsException` Lanza para indicar que se ha accedido a un índice ilegal de un array.

Ejercicio 2

Escriba el código Java que pueda generar mediante la herramienta **`javadoc`** la documentación de la clase mostrada a continuación.

Package **Class** **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS — **FRAMES** **NO FRAMES** **All Classes**

SUMMARY: NESTED | FIELD | CONSTR | METHOD — DETAIL: FIELD | CONSTR | METHOD

Ejemplos

Class Atributo

java.lang.Object

└ Ejemplos.Atributo

---

public final class **Atributo**
extends java.lang.Object

Un objeto Atributo define un atributo como una pareja nombre/valor, donde nombre es un String y valor es un Object.

Since:

1.0

Constructor Summary

Atributo(java.lang.String nombre) Crea un nuevo atributo con el nombre dado y un valor inicial de null.
Atributo(java.lang.String nombre, java.lang.Object valor) Crea un nuevo atributo con el nombre dado y un valor inicial.

Method Summary

java.lang.String	**getNombre**() Devuelve el nombre del atributo actual.
java.lang.Object	**getValor**() Devuelve el valor del atributo actual.
java.lang.Object	**setValor**(java.lang.Object nuevoValor) Escribe el valor del atributo actual.
java.lang.String	**toString**() Devuelve una cadena de texto de la forma nombre=valor

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

Constructor Detail

Atributo

public Atributo(java.lang.String nombre)

Crea un nuevo atributo con el nombre dado y un valor inicial de null.

See Also:

Atributo(String,Object)

Atributo

public Atributo(java.lang.String nombre,
java.lang.Object valor)

Crea un nuevo atributo con el nombre dado y un valor inicial.

See Also:

Atributo(String)

Method Detail

getNombre

public java.lang.String getNombre()

Devuelve el nombre del atributo actual.

Returns:

un String que contiene el nombre.

---

getValor

public java.lang.Object getValor()

Devuelve el valor del atributo actual.

Returns:

un Object que contiene el valor.

---

setValor

public java.lang.Object setValor(java.lang.Object nuevoValor)

Escribe el valor del atributo actual. Cambia el valor devuelto por la llamada a getValor().

Parameters:

nuevoValor - un Object que contiene el valor.

Returns:

un Object que contiene el valor original.

See Also:

getValor().

---

toString

public java.lang.String toString()

Devuelve una cadena de texto de la forma nombre=valor

Returns:

un Object que contiene el valor.

Ejercicio 3

Se proporciona el siguiente documento **XML**:

```
<?xml version="1.1"?>
<!DOCTYPE listin SYSTEM "listin.dtd">
<listin>
    <persona sexo="hombre" id="ricky">
       <nombre>Ricky Martin</nombre>
       <email>ricky@puerto-rico.com</email>
       <relacion amigo-de="laetitia"/>
    </persona>
    <persona sexo="mujer" id="laetitia">
       <nombre>Laetitia Casta</nombre>
       <email>castal@micasa.com</email>
    </persona>
</listin>
```

- ¿Es un documento bien formado? ¿Porqué? En caso negativo modifíquelo para que lo sea.

- Escriba una DTD (**Document Type Definition**) que valide el documento **XML** de arriba. Teniendo en cuenta que el documento está considerado para un número ilimitado de elementos personas **<persona>** (al menos uno). Sabiendo que un elemento **<persona>** puede no tener elemento **<email>** o tener varios. Que el elemento **<relacion>** es opcional. Y que el atributo **amigo-de**, es un atributo del tipo IDREFS.

- ¿Qué significado tienen #REQUIRED y #IMPLIED en una DTD?

Ejercicio 1: Solución

```
import matrices.Multiplicacion;
import matrices.LecturaMatricesBidimensionales;

public class Principal {
    public static void main(String args[]){
        LecturaMatricesBidimensionales in =
                     new LecturaMatricesBidimensionales(args);
        double[][] c = null;
        in.init();

        in.imprimeMatriz("a",in.getMatrizA());
        in.imprimeMatriz("b",in.getMatrizB());

        Multiplicacion m = new Multiplicacion(in.getMatrizA(), in.getMatrizB());
        m.multiplica();

        c = m.getMatrizResul();
        in.imprimeMatriz("c",m.getMatrizResul());
    }
}
```

```
package matrices;

public class Multiplicacion {
    private double [][]a   = null;
    private double [][]b   = null;
    private double [][]c   = null;
    private ElementoResultante [][]t   = null;

    public Multiplicacion(double [][]mat1, double [][]mat2){
        this.a = mat1;
        this.b = mat2;
    }

    public void multiplica(){
       //Instanciación de la matriz resultante
       c = new double[ a.length ][ b[0].length ];
       t = new ElementoResultante[ a.length ][ b[0].length ];

       int n = a.length;
       int m = b[0].length;

        for (int i = 0; i < n; i++){
            for (int j = 0; j < m; j++){
                //Crea un hilo para calcular c[i,j];
                // se necesita la columna de b[j]
                t[i][j] = new ElementoResultante (a,b,c,i,j);
            }
        }

        for (int i = 0; i < n; i++){
            for (int j = 0; j < m; j++){
                try{
                    t[i][j].join();
                }catch(InterruptedException e){}
            }
        }    }
    public double [][] getMatrizResul(){

        return c;
    }
}
```

```
package matrices;

public class LecturaMatricesBidimensionales{
    private double a[][]     = null,
                   b[][]     = null;
```

```
    private int    filasA    = 0, columnasA = 0,
                   filasB    = 0, columnasB = 0,
                   filasC    = 0, columnasC = 0;
    private String args[] = null;
    private int    n      = 0;

    public LecturaMatricesBidimensionales(String arg[]){
        this.args = arg;
    }

    public void init(){
        try { // Lee todas las entradas desde la línea de comandos
            filasC    =
            filasA    = Integer.parseInt(args[n++]);
            filasB    =
            columnasA = Integer.parseInt(args[n++]);
            columnasC =
            columnasB = Integer.parseInt(args[n++]);

            if (filasA    <= 0 || columnasA <= 0 || columnasB  <= 0) {
                System.out.println("Dimension incorrecta");
            }
            a = lecturaMatriz(filasA,columnasA);
            b = lecturaMatriz(filasB,columnasB);
      } catch (NumberFormatException e) {
            e.printStackTrace();
      } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("No hay suficiente datos.");
            e.printStackTrace();
      }
    }
    private double [][]lecturaMatriz(int filas, int columnas){
         double local[][]  = new double[filas][columnas];
         for (int i = 0; i < filas; i++)
            for (int j = 0; j < columnas; j++)
               local[i][j] = Double.parseDouble(args[n++]);
         return local;
    }

    public double [][] getMatrizA(){ return a;         }
    public double [][] getMatrizB(){ return b;         }
    public int    getfilasA()      { return filasA;    }
    public int    getcolumnasA()   { return columnasA; }
    public int    getfilasB()      { return filasB;    }
    public int    getcolumnasB()   { return columnasB; }
    public int    getfilasC()      { return filasC;    }
    public int    getcolumnasC()   { return columnasC; }

    public void imprimeMatriz(String nombre, double[][] c) {
        System.out.println(nombre + " = ");
        for (int i = 0; i < c.length; i++) {
            for (int j = 0; j < c[i].length; j++)
                System.out.print(" " + c[i][j]);
            System.out.println();
        }
    }

}
```

```
package matrices;

public class ElementoResultante extends Thread{
    private double a[][]     = null;
    private double b[][]     = null;
    private double c[][]     = null;
    private int    i         = 0;
    private int    j         = 0;

    public ElementoResultante (double a[][], double b[][], double c[][], int i , int j){
        this.a = a;
        this.b = b;
        this.c = c;
        this.i = i;
        this.j = j;
        start();
    }

    public void run(){
        for(int k = 0 ; k < a[0].length; k++)
            c[i][j] += a[i][k] * b[k][j];
    }
}
```

Ejercicio 2: Solución

```
/*
 * No Copyright
 * ABSOLUTELY NO WARRANTIES; USE AT YOUR OWN RISK
 *
 */

package Ejemplos;

/**
 * Un objeto <code>Atributo</code> define un atributo como
 * una pareja nombre/valor, donde nombre es un <code>String</code>
 * y valor es un <code>Object</code>.
 * <p>
 *
 * @author  Antonio J. Sierra
 * @version 12/11/07 (1.0)
 * @since   1.0
 */
public final
class Atributo {

    /** El nombre del atributo. */
    private final String nombre;

    /** El valor del atributo. */
    private Object valor = null;

    /**
     * Crea un nuevo atributo con el nombre dado y un valor
     * inicial de <code>null</code>.
     * @see Atributo#Atributo(String,Object)
     */
    public Atributo(String nombre) {
        this.nombre = nombre;
    }

    /**
     * Crea un nuevo atributo con el nombre dado y un valor
     * inicial.
     * @see Atributo#Atributo(String)
     */
    public Atributo(String nombre, Object valor) {
        this.nombre = nombre;
        this.valor  = valor;
    }

    /**
     * Devuelve el nombre del atributo actual.
     *
     * @return un <code>String</code> que contiene el nombre.
     */
    public String getNombre() {
        return nombre;
    }

    /**
     * Devuelve el valor del atributo actual.
     *
     * @return un <code>Object</code> que contiene el valor.
     */
    public Object getValor() {
```

```
        return valor;
    }

    /**
     * Escribe el valor del atributo actual. Cambia el valor devuelto
     * por la llamada a {@link #getValor}.
     *
     * @param nuevoValor un <code>Object </code> que contiene el valor.
     * @return un <code>Object</code> que contiene el valor original.
     * @see  #getValor().
     */
    public Object setValor(Object nuevoValor) {
        Object valAnt = valor;
        valor = nuevoValor;
        return valAnt;
    }

    /**
     * Devuelve una cadena de texto de la forma
     * <code>nombre=valor</code>
     *
     * @return un <code>Object </code> que contiene el valor.
     */
    public String toString() {
        return nombre + "'" + valor + "'";
    }

}
```

Ejercicio 3: Solución

El documento está bien formado ya que tomando el documento como un todo, cumple con las etiquetas de producción de un documento. Cumple todas las restricciones dadas por la especificación. Cada una de las entidades analizadas que se referencian directa o indirectamente en el documento están bien formadas.

```
<!ELEMENT listin (persona)+>
<!ELEMENT persona (nombre, email*, relacion?)>
<!ATTLIST persona id ID #REQUIRED>
<!ATTLIST persona sexo (hombre | mujer) #IMPLIED>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT email  (#PCDATA)>
<!ELEMENT relacion EMPTY>
<!ATTLIST relacion amigo-de IDREFS #IMPLIED>
```

Un atributo **#REQUIRED** significa que el atributo debería de proporcionarse, **#IMPLIED** que no tiene valor por defecto proporcionado.

Por definición, si la declaración no es ni **#IMPLIED** ni **#REQUIRED**, entonces el valor del atributo contiene el valor por defecto declarado; la palabra **#FIXED** indica que el atributo debería siempre que tener el valor por defecto.