TEMA 1.- MANEJO DE FICHEROS.

Objetivos

- Clases asociadas a las operaciones de gestión de ficheros y directorios: creación, borrado, copia, movimiento, entre otras.
- Flujos. Flujos basados en bytes y flujos basados en caracteres.
- Formas de acceso a un fichero.
- Clases para gestión de flujos de datos desde/hacia ficheros.
- Operaciones básicas sobre ficheros de acceso secuencial.
- Operaciones básicas sobre ficheros de acceso aleatorio.
- Trabajo con ficheros XML: analizadores sintácticos (parser) y vinculación (binding).
- Librerías para conversión de documentos XML a otros formatos.
- Excepciones: detección y tratamiento.

Contenidos

1.- Introducción.

Los programas usan variables para almacenar información:

- Datos de entrada.
- Resultados calculados.
- Valores intermedios generados a lo largo del cálculo.

Toda esta información es efímera: cuando acaba el programa, todo desaparece.

Para muchas aplicaciones, es importante poder almacenar datos de manera permanente. Una posibilidad es organizar esa información en uno o varios ficheros almacenados en algún soporte de almacenamiento persistente.

2.- Tipos de Ficheros.

Sabemos que es diferente manipular números que Strings, aunque en el fondo ambos acaben siendo bits en la memoria del ordenador. Por eso, cuando manipulamos archivos, distinguiremos dos clases de archivos dependiendo del tipo de datos que contienen:

- Los archivos de caracteres (o de texto)
- Los archivos de bytes (o binarios)

Fichero de texto es aquél formado exclusivamente por caracteres y que, por tanto, puede crearse y visualizarse usando un editor. Las operaciones de lectura y escritura trabajarán con caracteres. Por ejemplo, los ficheros con código java son ficheros de texto.

Fichero binario ya no está formado por caracteres sino que los bytes que contiene y pueden representar otras cosas como números, imágenes, sonido, etc.

En Java, los distintos tipos de ficheros se diferencian por las clases que usaremos para representarlos y manipularlos.

Las clases que usaremos pertenecen a la biblioteca estándar de java. Concretamente ubicadas en el paquete **java.io** por lo que deben ser importadas.

Además, el código que trabaja con archivos ha de considerar que muchas cosas pueden ir mal cuando se trabaja con ellos: Archivo corrupto, alguien ha desconectado el pendrive a medio ejecutar del programa, es un disco en red y ésta ha caído, o no tiene más espacio para almacenar información, etc. Para controlar estos errores usaremos **las excepciones.**

```
1 try {
2   Codigo que abre y trata el fichero
3 } catch (IOExcepction ex) {
4   Código que trata el error
5 }
```

3.- Lectura de un fichero de texto.

Podemos abrir un fichero de texto para leer usando la clase **FileReader**.
Esta clase tiene métodos que nos permiten leer caracteres. Suele ser habitual querer las líneas completas y **FileReader** no contiene métodos que nos permitan leer líneas completas, pero sí **BufferedReader**. Afortunadamente, podemos construir un **BufferedReader** a partir del **FileReader**.

Un bucle irá leyendo línea a línea la información del fichero de texto. La **condición de parada** del bucle debería ser **que la línea leída sea nula**. Por ejemplo, con un bucle **while**:

```
File archivo = new File ("archivo.txt");
try{
    FileReader fr = new FileReader (archivo);
    BufferedReader br = new BufferedReader(fr);
    String linea = "";
    do{
        linea = br.readLine();
        if(linea == null){
            break;
        }
        System.out.println(linea);
    }while(linea != null);

    br.close();
    fr.close();
}catch (Exception e) {
    System.out.println("Se ha producido el siguiente error al intentar leer el fichero: "
            + e.getClass());
}
```

4.- Escritura en un fichero de texto.

Igual que teniamos **FileReader** para abrir un fichero en modo lectura, tenemos el equivalente para modo escritura **FileWriter**.

De nuevo, esta clase tiene métodos que nos permiten escribir caracteres, pero siempre uno a uno. Como, al igual que pasaba en las lecturas, queremos escribir lineas completas, nos apoyaremos en un Buffer, en este caso **BufferedWriter**. Al igual que pasaba en la lectura, podemos construir un **BufferedWriter** a partir del **FileWriter**.

Si en el caso de la lectura era importante hacer uso del método close, en la escritura es doblemente importante, ya que es cuando se vacía el buffer y se escribe la información físicamente en el fichero.

Para cambiar de línea, se podría añadir un \n, pero no todos los editores de texto lo detectarían y puede dar problemas. Lo mejor es usar el método newLine de la clase BufferedReader para realizar el salto de línea.

El objeto FileWriter se puede instanciar con su segundo constructor **FileWriter(File f, boolean append)** donde el segundo parámetro indica si se abre para añadir información o no, siendo por defecto a false, es decir, **por defecto siempre se abre como un nuevo fichero y machaca la información**.

```
File archivo = new File ("archivoEscrito.txt");
try {
    FileWriter fw = new FileWriter(archivo, true);
    BufferedWriter bw = new BufferedWriter(fw);

    bw.write("weee");
    bw.newLine();
    bw.write("wuu");

    //Muy importante
    bw.close();
    fw.close();
} catch (IOException e) {
    e.printStackTrace();
}
```

5.- Ficheros binarios.

Las clases FileInputStream y FileOutputStream permite leer y escribir byte a byte ficheros de tipo binario.

FileInputStream dispone de los métodos read() y close() para leer y cerrar el flujo de datos. Cuando s termina el fichero, el método **read()** devuelve -1.

FileOutputStream también dispone de los métodos write() y close() para escribir y cerrar el flujo de datos.

Ejemplo: Realizar un duplicado de una imagen utilizando la lectura y escritura byte a byte.

```
package ficherobytes;
import java.io.*;
import java.io.IOException;

public class FicheroBytes {
    public static void main(String[] args) {
        Fichero miFichero = new Fichero();
        miFichero.leeBytes("C:/Users/Mario G/Desktop/paisaje.jpg");
        miFichero.escribeBytes("C:/Users/Mario G/Desktop/paisaje_C.jpg");
    }

}
class Fichero{
    int[] bytesImagen;
    public void leeBytes(String nombreFichero){
        try {
            File archivo = new File(nombreFichero);
            int tamano = (int)archivo.length();
            bytesImagen = new int[tamano];
            int cont = 0;
            FileInputStream fichero = new FileInputStream(nombreFichero);
            int miByte =  fichero.read();
            while (miByte!=-1){
                bytesImagen[cont]=miByte;
                miByte =  fichero.read();
                cont++;
            }
            fichero.close();
            System.out.println("El fichero ha sido leido.");
        } catch (IOException e) {
            System.out.println("Error en el Fichero");
        }
    }
    public void escribeBytes(String nombreFichero){
        try {
            FileOutputStream fichero = new FileOutputStream(nombreFichero);
            for(int i=0; i<this.bytesImagen.length; i++){
                fichero.write(bytesImagen[i]);
            }
            fichero.close();
            System.out.println("El fichero ha sido duplicado.");
        } catch (IOException e) {
            System.out.println("Error en el Fichero");
        }
    }
}
```

Ejemplo: Copiar fichero en Java

```
package copiarfichero;
import java.io.*;
public class CopiarFichero {
    public static void main(String[] args) {
        Fichero miFichero = new Fichero();
        miFichero.copiar("paisaje.jpg", "foto.jpg");
    }
}
class Fichero{
    public void copiar(String ficheroOrigen, String ficheroDestino){
        try {
            FileInputStream origen = new FileInputStream(ficheroOrigen);
            FileOutputStream destino = new FileOutputStream(ficheroDestino);
            byte[] buffer = new byte[1024];
            int longitud;
            while ((longitud = origen.read(buffer)) > 0) {
                destino.write(buffer, 0, longitud);
            }
            origen.close();
            destino.close();
            System.out.println("El fichero ha sido copiado con Éxito!!!");
        } catch (IOException e) {
            System.out.println("Error en el Fichero");
        }
    }
}
```

Otra posibilidad es usar el método readAllBytes de la clase Files del paquete java.nio.file.

```
package copiarfichero;
import java.io.*;
// importamos para poder volcar todos los bytes del fichero a un array de bytes
import java.nio.file.Files;
public class CopiarFichero {
    public static void main(String[] args) {
        Fichero miFichero = new Fichero();
        miFichero.copiar("paisaje.jpg", "foto.jpg");
    }
}
class Fichero{
    public void copiar(String ficheroOrigen, String ficheroDestino){
        try {
            // Creamos un puntero al fichero de origen
            File fichero = new File(ficheroOrigen);
            // Creamos flujo de datos para copiar los bytes leidos del fichero
            FileOutputStream destino = new FileOutputStream(ficheroDestino);
            // Creamos un array y volcamos todos los bytes del fichero de origen.
            byte[] buffer = Files.readAllBytes(fichero.toPath());
            // Escribimos los bytes en el flujo del fichero de destino
            destino.write(buffer);
            // Cerramos el flujo de destino
            destino.close();
```

```
            // Mostramos mensaje que la copia se ha realizado correctamente
            System.out.println("El fichero ha sido copiado con Éxito!!!");
        } catch (IOException e) {
            System.out.println("Error en el Fichero");
        }
    }
}
```

6.- Clase File.

La clase File para trabajar con ficheros, dispone de los siguientes métodos:

MÉTODO	DESCRIPCIÓN
boolean canRead()	Devuelve true si se puede leer el fichero
boolean canWrite()	Devuelve true si se puede escribir en el fichero
boolean delete()	Elimina el fichero o directorio. Si es un directorio debe estar vacío. Devuelve true si se ha podido eliminar.
boolean exists()	Devuelve true si el fichero o directorio existe
String getName()	Devuelve el nombre del fichero o directorio
String getAbsolutePath()	Devuelve la ruta absoluta asociada al objeto File.
boolean isDirectory()	Devuelve true si es un directorio válido
boolean isFile()	Devuelve true si es un fichero válido
long lastModified()	Devuelve un valor en milisegundos que representa la última vez que se ha modificado (medido desde las 00:00:00 GMT, del 1 de Enero de 1970). Devuelve 0 si el fichero no existe o ha ocurrido un error.
long length()	Devuelve el tamaño en bytes del fichero. Devuelve 0 si no existe. Devuelve un valor indeterminado si es un directorio.
String[] list()	Devuelve un array de String con el nombre de los archivos y directorios que contiene el directorio indicado en el objeto File. Si no es un directorio devuelve null. Si el directorio está vacío devuelve un array vacío.
boolean mkdir()	Crea el directorio. Devuelve true si se ha podido crear.
boolean renameTo(File dest)	Cambia el nombre del fichero por el indicado en el parámetro dest. Devuelve true si se ha realizado el cambio. de la ruta padre.

Ejemplo:

```
package ficheros;
import java.io.*;
public class Ficheros {
    public static void main(String[] args) {
        File miFichero = new File("datos/archivo.txt");
        File tuFichero = new File("datos/archivoC.txt");
        File miCarpeta = new File("curso");
        File origen = new File(".");
        System.out.println("Ruta Absoluta: "+miFichero.getAbsolutePath());
        System.out.println("Ruta Relativa: "+miFichero.getPath());
        System.out.println("Nombre Fichero: "+miFichero.getName());
        System.out.println("Directorio Padre: "+miFichero.getParent());
        System.out.println("Tamaño: "+miFichero.length()+" bytes");
        if(miFichero.canRead()){
            System.out.println("El fichero tiene permisos de Lectura.");
        }
        if(miFichero.canWrite()){
```

```
            System.out.println("El fichero tiene permisos de Escritura.");
        }
        if(miFichero.canExecute()){
            System.out.println("El fichero tiene permisos de Ejecución.");
        }
        if(miFichero.isFile()){
            System.out.println("El archivo es un fichero.");
        }
        if(miFichero.isDirectory()){
            System.out.println("El archivo es un directorio.");
        }
        if(miFichero.exists()){
            System.out.println("El fichero existe");
        }
        miFichero.renameTo(tuFichero);
        System.out.println("El fichero ha sido renombrado");
        tuFichero.delete();
        System.out.println("El fichero ha sido eliminado");
        miCarpeta.mkdir();
        System.out.println("La carpeta ha sido creada");

        String[] listado = origen.list();
        for (String lista : listado) {
            System.out.println(lista);
        }
    }
}
```

7.- Manejo de ficheros XML y Json.

A principios de los años 90 surgió el problema de que las maquinas pudieran entenderse entre sí. Utilizaban diferentes sistemas operativos y sus programas estaban escritos en diferentes lenguajes de programación.

Una de las soluciones fue crear el estándar XML y más tarde JSON, para a través de ellos poder intercambiar información entre máquinas independientemente del sistema operativo o del lenguaje de programación con el que estuviera hecho el software.

7.1.- String y StringBuilder.

Los objetos String disponen de un método split para realizar de forma más rápida la división por el carácter especificado.

Ejemplo:

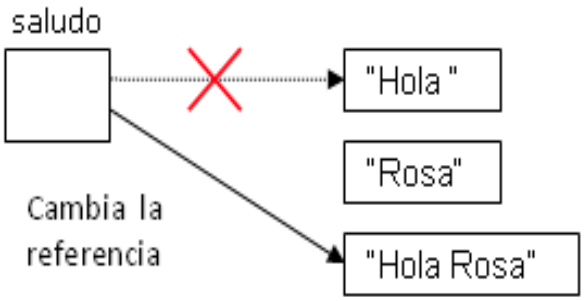
```
String cadena = "Buenos días, hoy vamos a ver como funciona el split";
String[] trozosCadena = cadena.split(" ");
for(int i = 0; i < trozosCadena.length; i++){
    System.out.println(trozosCadena[i]);
}
```

Resultado:

```
Buenos
días,
hoy
vamos
a
ver
como
funciona
el
split
```

Los objetos String son inmutables, es decir, al crearse e inicializarse no pueden cambiar su valor. Con el siguiente ejemplo, puede parecer que el String saludo sí cambia, pero lo que en realidad sucede es que la referencia se asigna a una nueva cadena creada:

```
public class Saludo{
    public static void main (String[] args){
        String saludo = "Hola ";
        saludo = saludo + "Rosa";
        System.out.println(saludo);
    }
}
```



Cuando manipulamos Strings, concatenando, insertando o reemplazando caracteres se crean muchos objetos que son descartados rápidamente. Esta creación de objetos puede producir un incremento en el uso de la memoria. El recolector de basura de Java, limpia la memoria pero esto tiene un costo de tiempo. Si creamos y destruimos muchos objetos, la performance de nuestro programa puede decaer sustancialmente.

StringBuilder es similar a la clase String en el sentido de que sirve para almacenar cadenas de caracteres. No obstante, presenta algunas características especiales a tener en cuenta:

- Su tamaño y contenido pueden modificarse.
- Debe crearse con alguno de sus constructores asociados. No se permite instanciar directamente a una cadena como sí permiten los String.
- Indexado. Cada uno de sus caracteres tiene un índice: 0 para el primero, 1 para el segundo, etc.

Posee varios constructores, como se puede observar en la siguiente tabla, aunque el más habitual es el primero (vacío).

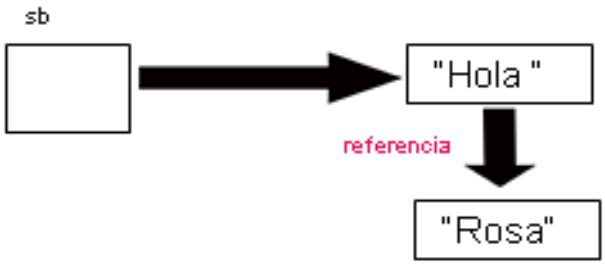
Constructor	Descripción	Ejemplo
StringBuilder()	Construye un StringBuilder vacío y con una capacidad por defecto de 16 carácteres.	StringBuilder s = new StringBuilder();
StringBuilder(int capacidad)	Se le pasa la capacidad (número de caracteres) como argumento.	StringBuilder s = new StringBuilder(55);
StringBuilder(String str)	Construye un StringBuilder en base al String que se le pasa como argumento.	StringBuilder s = new StringBuilder("hola");

Métodos de la clase StringBuilder.

Retorno	Método	Explicación
StringBuilder	append(...)	Añade al final del StringBuilder a la que se aplica, un String o la representación en forma de String de un dato asociado a una variable primitiva
int	length()	Devuelve el número de caracteres del StringBuilder
StringBuilder	reverse()	Invierte el orden de los caracteres del StringBuilder
void	setCharAt(int indice,char ch)	Cambia el carácter indicado en el primer argumento por el carácter que se le pasa en el segundo
char	charAt(int indice)	Devuelve el carácter asociado a la posición que se le indica en el argumento
void	setLength(int nuevaLongitud)	Modifica la longitud. La nueva longitud no puede ser menor
String	toString()	Convierte un StringBuilder en un String
StringBuilder	insert(int indicelni,String cadena)	Añade la cadena del segundo argumento a partir de la posición indicada en el primero
StringBuilder	delete(int indicelni,int indiceFin)	Borra la cadena de caracteres incluidos entre los dos indices indicados en los argumentos
StringBuilder	deleteChar(int indice)	Borra el carácter indicado en el índice
StringBuilder	replace(int indicelni, int indiceFin,String str)	Reemplaza los caracteres comprendidos entre los dos indices por la cadena que se le pasa en el argumento
int	indexOf (String str)	Analiza los caracteres de la cadena y encuentra el primer índice que coincide con el valor deseado
String	subString(int indicelni,int indiceFin)	Devuelve una cadena comprendida entre los dos índices

```java
public class Saludo {
    public static void main(String args[]){
        StringBuilder sb = new StringBuilder();
        sb.append("Hola ");
        sb.append("Rosa");

        //Al acabar de rellenar la cadena,
        //ya podemos transformarla a un único string
        String saludo = sb.toString();
        System.out.println(saludo);
    }
}
```



Podemos ver que no va dejando objetos zombies en memoria, sino que **va referenciando unos a otros**, igual que lo hace un ArrayList.

Por último, cuando sabemos que la cadena no va a ser modificada, se transforma a String con el método **toString**.

8.- XML (Extensible Markup Language)

Estándar desarrollado por la World Wide Web Consortium (W3C). Define reglas para codificar información legible por un ser humano y por un ordenador. Uno de los formatos más utilizados para el intercambio de información entre sistemas. Basado en texto para representar información estructurada: datos, documentos, configuración, etc..

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
<libro>
    <titulo>Odisea 2001</titulo>
    <numeroPaginas>400</numeroPaginas>
</libro>
```

Una etiqueta es un elemento que puede tener contenido, más elementos o estar vacía pero siempre deben cerrarse. <y> contenido</y>

Se compone de partes bien definidas y esas partes a su vez se componen de otras formando un árbol de información. Podemos encontrar atributos (tipo)

JAXB permite mapear clases Java a XML y viceversa, por lo que posee dos funciones fundamentales: capacidad de presentar un objeto Java en XML (Marshall) y presentar un XML como un objeto Java.

Las clases que se van a mapear deben tener añadidos las anotaciones que ayudarán a convertir la clase Java en XML:

- **@XmlRootElement**. Define la raíz del XML.
- **@XmlType(propOrder = { "campo1"," campo2",...})**. Orden de los elementos en el XML.
- **@XmlElement(name="nombre").** Define el elemento del XML. Nuevo nombre

Ejemplo:

```
package javatoxml;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement Anotaciones van con @
@XmlType(propOrder = {"titulo","paginas"})Orden de las propiedades de los atributos
public class Libro {
    private String titulo;
    private int paginas;

    public Libro(String titulo, int paginas) {
        this.titulo = titulo;
        this.paginas = paginas;
    }
    public Libro() {
    }
    public String getTitulo() {
        return titulo;
    }
    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }
    @XmlElement(name="numeroPaginas") Cambiamos el nombre
    public int getPaginas() {Lo cambiamos en el get
        return paginas;
    }
    public void setPaginas(int paginas) {
        this.paginas = paginas;
    }
    @Override
    public String toString() {
        return "Libro{" + "titulo=" + titulo + ", paginas=" + paginas + '}';
    }
}
```

```
package xmltojava;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;
import javax.xml.bind.PropertyException;
import java.io.File;

public class JavatoXML {
    public static void main(String[] args) {
        try {
            Libro libro= new Libro("Odisea 2001",400);
            //Instanciamos el contexto indicando la clase
```

```
            //que será el RootElement. Libro
            JAXBContext contexto = JAXBContext.newInstance(libro.getClass());
            // Creamos un Marshaller que convertirá la clase Java en XML
            Marshaller marshaller = contexto.createMarshaller();
            // Indicamos que queremos el XML en formato amigable
            // Con retornos de carro y tabulado
            marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
                Boolean.TRUE);
            // Llamando al método marshal hacemos la conversión
            // de Clase Java a XML.
            // Pasamos como parámetros la instancia de la clase
            // y la Salida que queremos para el XML
            // System.out sería la consola o new File("./miFichero.xml")
            // para guardar en un fichero.
            //marshaller.marshal(libro, System.out);
            marshaller.marshal(libro, new File("miFichero.xml"));
        } catch (PropertyException e) {
            e.printStackTrace();
        } catch (JAXBException e) {
            e.printStackTrace();
        }
    }
}
```

La misma operación se puede realizar al contrario, es decir leer un fichero XML y rellenar un objeto Java.

Ejemplo:

```
package javatoxml;
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

@XmlRootElement
@XmlType(propOrder = {"titulo","paginas"})
public class Libro {
    private String titulo;
    private int paginas;

    public Libro(String titulo, int paginas) {
        this.titulo = titulo;
        this.paginas = paginas;
    }
    public Libro() {
    }
    public String getTitulo() {
        return titulo;
    }
    public void setTitulo(String titulo) {
        this.titulo = titulo;
```

```
    }
    @XmlElement(name="numeroPaginas")
    public int getPaginas() {
        return paginas;
    }
    public void setPaginas(int paginas) {
        this.paginas = paginas;
    }
    @Override
    public String toString() {
        return "Libro{" + "titulo=" + titulo + ", paginas=" + paginas + '}';
    }
}
```

```
package xmltojava;

import java.io.File;
import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Unmarshaller;
public class XMLtoJava {
    public static void main(String[] args) {
        try {
            // Instanciamos el contexto, indicando la clase que será
            // el RootElement, en nuestro caso Libro
            JAXBContext context = JAXBContext.newInstance(Libro.class );
            // Se crea un unmarshaller para convertir XML a Java
            Unmarshaller unmarshaller = context.createUnmarshaller();
            // Utilizamos el método unmarshal para obtener los datos
            // del fichero xml
            Libro libro = (Libro)unmarshaller.unmarshal(new
File("miFichero.xml") );
            // Mostramos los atributos del Objeto libro generado a partir del
XML
            System.out.println("Titulo: "+libro.getTitulo());
            System.out.println("Páginas: "+libro.getPaginas());
        } catch (JAXBException e) {
                e.printStackTrace();
        }
    }
}
```

9.- Json

XML procesamiento muy lento con gran volumen de datos, lo que dio lugar a Json. Estándar abierto que utiliza texto plano para codificar información en la forma **atributo: valor**. Ampliamente usado para intercambio de información entre servicios web y APIs REST. Su simplicidad y facilidad de implementación le otorgan un gran desempeño y lo convierten en una de las alternativas ideales para reemplazar XML.

Ejemplo:

```
{
  "departamento": 8,
  "nombredepto": "Ventas",
  "director": "juan rodriguez",
  "empleados": [
    {
      "nombre": "Pedro",
      "apellido": "Fernandez"
    },
    {
      "nombre": "Jacinto",
      "apellido": "Benavente"
    }
  ]
}
```

Sintaxis Json:

- Siempre empiezan con { y acaban con }
- Para asignar a un nombre un valor debemos usar los dos puntos **':'** este separador es el equivalente al igual ('=') de cualquier lenguaje.

  {"Nombre" : "Pedro"}
- Posibles Valores:
  - **número** (entero o float)
  - **string** (entre comillas)
  - **booleano** (true o false)
  - **array** (entre corchetes [])
  - **objeto** (entre llaves {})
  - **Null**

Ejercicio: Identifica los posibles objetos y/o arrays

```
{"Fruteria":
  [
    {"Fruta":
       [
             {"Nombre":"Manzana","Cantidad":10},
             {"Nombre":"Pera","Cantidad":20},
             {"Nombre":"Naranja","Cantidad":30}
        ]
    },
    {"Verdura":
       [
             {"Nombre":"Lechuga","Cantidad":80},
             {"Nombre":"Tomate","Cantidad":15},
             {"Nombre":"Pepino","Cantidad":50}
        ]
    }
  ]
}
```

Podemos utilizar una buena herramienta online para visualizar mejor el **contenido de un Json**:
http://jsonviewer.stack.hu/

Las diferencias entre XML y Json son:

XML

Ventajas:

- Tiene un formato muy estructurado y fácil de comprender.
- Puede ser validado fácilmente mediante Schemas(XSD)
- Se pueden definir estructuras complejas y re utilizables.

Desventajas:

- Es mas complicado de entender
- El formato es sumamente estricto.
- Lleva mas tiempo procesarlo
- Un error con los namespace puede hacer que todo el documento sea invalido

JSON

Ventajas:

- Formato sumamente simple
- Velocidad de procesamiento alta
- Archivos de menor tamaño

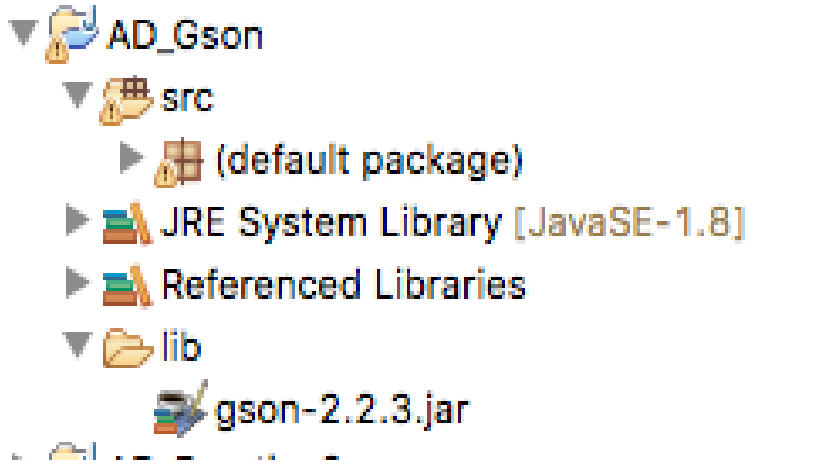
Desventajas:

- Tiene una estructura enredosa y difícil de interpretar a simple vista

XML es mucho mejor para la comunicación entre servidores y aplicaciones ya que permite mensajes mucho mas completos, tipificados y además pueden ser validados con facilidad.

JSON tiene un formato mas simple y fácil de procesar, por lo que es ideal para dispositivos de bajo nivel de procesamiento como los teléfonos móviles..

Para trabajar con **Json** desde **java o android** podemos usar la librería **Gson** de google subida a la plataforma con el nombre **gson-2.2.3.jar**.
Al crear el proyecto se debe añadir a una nueva carpeta llamada **libs**
Una vez añadida, botón derecho—Build Path – Add to build path



Gson tiene 2 métodos principales:

- **toJson**(Objeto): Recibe como parámetro el objeto a transformar en Json. Este método devuelve un String con el objeto parseado a Json.
- **fromJson**(String json, Clase): Recibe como parámetro el json en una variable de tipo String y la clase a la que tiene que parsearlo. Este método devuelve un objeto de la clase pasada como segundo parámetro con los valores del Json.

Ejemplo: Transformar XML a Json

XML.

```
<empresa>
   <departamento>8</departamento>
   <director>juan rodriguez</director>
   <empleados>
      <empleado>
         <apellido>Fernandez</apellido>
         <nombre>Pedro</nombre>
      </empleado>
      <empleado>
         <apellido>Benavente</apellido>
         <nombre>Jacinto</nombre>
      </empleado>
   </empleados>
   <nombredepto>Ventas</nombredepto>
</empresa>
```

Json.

```
{
  "departamento": 8,
  "nombredepto": "Ventas",
  "director": "juan rodriguez",
  "empleados": [
    {
      "nombre": "Pedro",
      "apellido": "Fernandez"
    },
```

```
    {
      "nombre": "Jacinto",
      "apellido": "Benavente"
    }
  ]
}
```

Ejemplo: Transformar Json a XML

Json:

```
{
    "marcadores": [
      {
        "latitude": 40.416875,
        "longitude": -3.703308,
        "city": "Madrid",
        "description": "Puerta del Sol"
      },
      {
        "latitude": 40.417438,
        "longitude": -3.693363,
        "city": "Madrid",
        "description": "Paseo del Prado"
      },
      {
        "latitude": 40.407015,
        "longitude": -3.691163,
        "city": "Madrid",
        "description": "Estación de Atocha"
      }
    ]
}
```

XML:

```
<marcadores>
      <marcador>
            <latitude>40.416875</latitude>
            <longitude>-3.703308</longitude>
            <city>Madrid</city>
            <description>Puerta del Sol</description>
      </marcador>
      <marcador>
            <latitude>40.417438</latitude>
            <longitude>-3.693363</longitude>
            <city>Madrid</city>
            <description>Paseo del Prado</description>
      </marcador>
      <marcador>
```

```
            <latitude>40.407015</latitude>
            <longitude>-3.691163</longitude>
            <city>Madrid</city>
            <description>Estación de Atocha</description>
      </marcador>
</marcadores>
```

Ejemplo: Convertir Json en Objeto y Objeto a Json.

Clase Libro.

```
package jsonobjeto;
public class Libro {
    private String titulo;
    private int paginas;

    public Libro(String titulo, int paginas) {
        this.titulo = titulo;
        this.paginas = paginas;
    }
    public Libro() {
    }
    public String getTitulo() {
        return titulo;
    }
    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }
    public int getPaginas() {
        return paginas;
    }
    public void setPaginas(int paginas) {
        this.paginas = paginas;
    }
    @Override
    public String toString() {
        return "Libro{" + "titulo=" + titulo + ", paginas=" + paginas + '}';
    }
}
```

Clase Principal.

```
package jsonobjeto;
import com.google.gson.Gson;
public class JsonObjeto {
    public static void main(String[] args) {
      // Json a Objeto
      String jsonT = "{\"titulo\":\"El lazarillo de Tormes\","
                  + "\"paginas\":23}";
      Gson miGson = new Gson();
      Libro miLibro = miGson.fromJson(jsonT, Libro.class);
      System.out.println("Título: " + miLibro.getTitulo());
      System.out.println("Páginas: " + miLibro.getPaginas());
```

```
        // Objeto a Json
        String jsonDevuelto = miGson.toJson(miLibro);
        System.out.println(jsonDevuelto);
    }
}
```

Transformar un listado a Json.

Cuando se quiere parsear un listado de varios objetos, en nuestro caso un ArrayList, se debe usar el tipo complejo al que se quiere transformar.

Ejemplo:

Clase Persona:

```
package listadojson;
public class Persona {
        private String nombre;
        private String apellido;
        private int edad;
        public Persona(String nombre, String apellido, int edad) {
                this.nombre = nombre;
                this.apellido = apellido;
                this.edad = edad;
        }
        public String getNombre() {
                return nombre;
        }
        public void setNombre(String nombre) {
                this.nombre = nombre;
        }
        public String getApellido() {
                return apellido;
        }
        public void setApellido(String apellido) {
                this.apellido = apellido;
        }
        public int getEdad() {
                return edad;
        }
        public void setEdad(int edad) {
                this.edad = edad;
        }
}
```

Clase Principal.

```
package listadojson;

import java.lang.reflect.Type;
import java.util.ArrayList;
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;
```

```
public class ListadoJson {

    public static void main(String[] args) {
        ArrayList<Persona> personas = new ArrayList<Persona>();
        personas.add(new Persona("Isaac", "Garcia", 30));
        personas.add(new Persona("Pepe", "Gonzalez", 20));
        personas.add(new Persona("Juan", "Pereira", 10));
        Gson gson = new Gson();
        Type tipoListado = new TypeToken<ArrayList<Persona>>() {
        }.getType();
        String json = gson.toJson(personas, tipoListado);
        System.out.println(json);
    }
}
```

Transformar un array Json a objetos.

Cuando se quiere parsear un array json a varios objetos, se debe usar el tipo complejo desde el que se quiere transformar.

Ejemplo:

Clase Alumno:

```
package arrayjson;
public class Alumno {
    int id;
    String nombre;
    String apellidos;

    public Alumno(int id, String nombre, String apellidos) {
        this.id = id;
        this.nombre = nombre;
        this.apellidos = apellidos;
    }

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
```

```
    public String getApellidos() {
        return apellidos;
    }

    public void setApellidos(String apellidos) {
        this.apellidos = apellidos;
    }

    @Override
    public String toString() {
        return "Alumno{" + "id=" + id + ", nombre=" + nombre + ", apellidos="
+ apellidos + '}';
    }
}
```

Clase Principal.

```
package arrayjson;
import com.google.gson.*;
import com.google.gson.reflect.*;
import java.util.List;
public class ArrayJson {
    public static void main(String[] args) {
        String jsonString =
          "[{\"id\":1,\"nombre\":\"Ana\",\"apellidos\":\"Ros Martí\"},"
          + "{\"id\":2,\"nombre\":\"Antonio\",\"apellidos\":\"Romero Pla\"},"
          + "{\"id\":3,\"nombre\":\"Eva\",\"apellidos\":\"Rui Pons\"}]";
        try{
            Gson gson = new Gson();
            List<Alumno> listAlumnos = gson.fromJson(jsonString, new
TypeToken<List<Alumno>>(){}.getType());
            if( listAlumnos!= null ){
                for(Alumno object : listAlumnos){
                    System.out.println("Alumno : " + object.getNombre() + " "
+ object.getApellidos() );
                }
            }
        }catch(JsonSyntaxException e){
            System.err.println("JsonSyntaxException: " + e.getMessage());
        }
    }
}
```

10.- Interfaces de Usuario

Java dispone de varias librerías de clase para implementar interfaces visuales. Vamos a utilizar los componentes del paquete Swing para el desarrollo de interfaces gráficas. Para utilizar los componentes del paquete Swing hay que importar del paquete javax.swing.

Para evitar advertencias en el código del entorno Netbeans en el desarrollo de interfaces debemos desactivar las opciones "Problematc call in the constructor" y "Parsing suspicious parameter in the constructor" del menú Herramientas /Opciones / Editor / Sugerencias / Inicialization .

JFrame.

La clase JFrame encapsula el concepto de una ventana. Para implementar una aplicación que muestre una ventana debemos crear una clase que herede de la clase JFrame.

```
public class Formulario extends JFrame{
```

Estamos creando una clase Formulario que hereda todos los métodos y propiedades de la clase JFrame.

La clase JFrame dispone de varios métodos para dar formato a la ventana:

Método	Descripción
setTitle(texto)	Título de la Ventana
setSize(ancho,alto)	Tamaño de la Ventana
setLocation(x, y)	Posición de la Ventana
setBounds(x, y, ancho, alto)	Posición y Tamaño de la Ventana
setResizable(boolean)	Permitir cambio de Tamaño
setLocationRelativeTo(null)	Posición centro de la pantalla
SetDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)	Acción al pulsar botón Cerrar
setVisible(boolean)	Permitir Visibilidad

Ejemplo:

```
package formulario.jframe;
import javax.swing.*;
public class FormularioJFrame {
    public static void main(String[] args) {
        Formulario miFormulario = new Formulario();
        miFormulario.setVisible(true);
    }
}
class Formulario extends JFrame{
    public Formulario() {
        this.setTitle("Mi Aplicación Java");
        this.setSize(500,300);
        this.setResizable(false);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

JPanel

La clase JPanel permite crear objetos contenedores, la finalidad de estos objetos es la agrupación de otros objetos como botones, campos de texto, etiquetas, selectores, etc... y permite manejar la agrupación de una mejor forma.

Podemos crear un JPanel creando una instancia de la clase con

```
JPanel miPanel = new JPanel();
```

Una vez creado el JPnanel podemos establecer el método setLayout a null para poder posicionar los componentes en las posiciones x, y que indiquemos con setLocation o setBounds.

```
miPanel.setLayout(null);
```

Una vez colocados los componentes en el panel añadimos el panel al formulario con add().

```
miFormulario.add(miPanel);
```

JLabel

La clase JLabel permite añadir texto. Podemos crear un JLabel creando una instancia de la clase y pasando como parámetro el texto que queremos mostrar.

```
JLabel miTexto = new JLabel("Este es el texto que se mostrará");
```

Para ubicar el texto en el JPanel utilizaremos el método setLocation o setBounds, para posteriormente añadir el texto al panel.

```
miTexto.setBounds(10,40,100,30);
miPanel.add(miTexto);
```

Podemos cambiar el tipo y tamaño de fuente creando una fuente con la clase Font y asignarlo posteriormente al texto con el método setFont(). El método setFont() recibe como parámetros el nombre de la fuente, el estilo y el tamaño. Los posibles estilos vienen definidos por las constantes Font.PLAIN, Font.BOLD, Font.ITALIC, ... Deberemos importar la clase Font del paquete java.awt

```
Font miFuente = new Font("Comic Sans Ms", Font.PLAIN, 24);
miTexto.setFont(miFuente);
```

También podemos asignar color al texto con el método setForeground() que recibe como parámetro el color definidos por las constantes Color.RED, Color.BLUE, Color.GREEN, ... o creando una instancia de Color y asignar la cantidad de cada una de las componentes RGB. Deberemos importar la clase Color del paquete java.awt

```
Color miColor = new Color(100,45,200);
miTexto.setForeground(miColor);
```

Si queremos asignar un texto al JLabel podemos utilizar el método setText();

```
miTexto.setText("Este es el nuevo texto");
```

Ejemplo:

```
package formulario.jframe;
import java.awt.Color;
import java.awt.Font;
import javax.swing.*;
public class FormularioJFrame {
    public static void main(String[] args) {
        Formulario miFormulario = new Formulario();
        miFormulario.setVisible(true);
    }
}

class Formulario extends JFrame{
    JPanel miCapa;
    JLabel textoTitulo;
    JLabel textoSubtitulo;

    public Formulario() {
        this.setTitle("Mi Aplicación Java");
        this.setSize(500,300);
        this.setResizable(false);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);

        // Creamos un panel para colocar componentes en el Formulario
        miCapa = new JPanel();
        // Establecemos el setLayout a null para poder posicionar los
        // componentes en las posiciones x, y indicadas en setBounds
        miCapa.setLayout(null);

        // Creamos un estilo de fuente con la clase Font y se lo asignaremos
        // con setFont a los componentes JLabel que se creen en el Panel
        // Debemos importar la clase Font del paquete java.awt
        Font miFuente = new Font("Comic Sans Ms", Font.PLAIN, 24);
        // Creamos un color con la clase Color y asigando la cantidad de cada
        // una de las componentes RGB. Posteriormente podemos asignarlo al
        // texto con el método setForeground()
        // Deberemos importar la clase Color del paquete java.awt
        Color miColor = new Color(100,45,200);
        // Creamos un texto, la posicionamos y la añadimos a la capa
        textoTitulo = new JLabel("Curso de Programación");
        textoTitulo.setBounds(10,20,300,40);
        textoTitulo.setFont(miFuente);
        textoTitulo.setForeground(miColor);
        miCapa.add(textoTitulo);
        // Creamos un texto, la posicionamos y la añadimos a la capa
        textoSubtitulo = new JLabel("Aula Campus");
        textoSubtitulo.setBounds(10,50,100,30);
        miCapa.add(textoSubtitulo);
        // Añadimos la capa al Formulario
        this.add(miCapa);
    }
}
```

JTextField

La clase JTextField permite introducir texto por teclado. Podemos crear un JTextField creando una instancia de la clase.

```
JTextField miCampoTexto = new JTextField();
```

Para ubicar el campo de texto en el JTextField utilizaremos el método setLocation o setBounds, para posteriormente añadir el texto al panel.

```
miCampoTexto.setBounds(10,40,100,30);
miPanel.add(miTexto);
```

Podemos acceder al contenido del JTextField con el método getText().

```
miCampoTexto.getText();
```

Ejemplo:

```
package formulario.jframe;
import javax.swing.*;
public class FormularioJFrame {
    public static void main(String[] args) {
        Formulario miFormulario = new Formulario();
        miFormulario.setVisible(true);
    }
}
class Formulario extends JFrame{
    JPanel miCapa;
    JLabel miTexto;
    JTextField miCampoTexto ;

    public Formulario() {
        this.setTitle("Mi Aplicación Java");
        this.setSize(500,300);
        this.setResizable(false);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);

        // Creamos un panel para colocar componentes en el Formulario
        miCapa = new JPanel();
        // Establecemos el setLayout a null para poder posicionar los
        // componentes en las posiciones x, y indicadas en setBounds
        miCapa.setLayout(null);

        // Creamos un texto, la posicionamos y la añadimos a la capa
        miTexto = new JLabel("Nombre:");
        miTexto.setBounds(10,20,50,30);
        miCapa.add(miTexto);
        // Creamos un Campo de Texto, lo posicionamos y añadimos a la capa
        miCampoTexto = new JTextField();
        miCampoTexto.setBounds(70,20,100,30);
        miCapa.add(miCampoTexto);

        // Añadimos la capa al Formulario
        this.add(miCapa);
    }
}
```

JButton.

La clase JButton muestra un botón. Podemos crear una instancia del botón, posicionarla con el método setBounds() y añadirla al panel con add().

```
JButton miBoton = new JButton("Finalizar");
miBoton.setBounds(10,100,100,30);
miPanel.add(miBoton);
```

El uso de botones precisa la captura de eventos como cuando hacemos clic sobre un botón. Java implementa el concepto de interfaces para poder llamar a métodos de una clase existente a una clase desarrollada por nosotros.

La captura del clic sobre el objeto de la clase JButton se hace mediante la implementación de la interface ActionListener tiene la siguiente estructura:

```
interface ActionListener {
    public void actionPerformed(ActionEvent e) {
}
```

Luego las clases que implementen la interface ActionListener deberán especificar el código del método actionPerformed.

Para indicar que una clase implementará una interface declaamos la clase con la sintaxis:

```
public class Formulario extends JFrame implements ActionListener {}
```

Estamos indicando que nuestra clase implementa la interface ActionListener, por lo que estamos obligados a desarrollar el código del método actionPerformed.

Podemos añadir al botón el método addActionListener() que queda esperando a que pulsemos sobre el botón. Cuando pulsemos sobre el botón, tomará el parámetro this que hace referencia al propio botón y ejecutará el método actionPerformed() que recibe como parámetro un objeto de la clase ActionEvent. La interface ActionListener y el objeto de la clase ActionEvent que llega como parámetro están definidos en el paquete java.awt.event.

```
miBoton.addActionListener(this);
```

El método actionPerformed() mediante el uso del método getSource() del objeto que llega como parámetro analizar que botón se presionó:

```
    public void actionPerformed(ActionEvent e) {
        if (e.getSource()==miBoton) {
            System.exit(0);
        }
    }
```

Si se pulsó miBoton se llama al método exit de la clase System se finaliza la ejecución del programa.

Ejemplo:

```
package formulario.jframe;
import java.awt.event.*;
import javax.swing.*;
public class FormularioJFrame {
    public static void main(String[] args) {
        Formulario miFormulario = new Formulario();
        miFormulario.setVisible(true);

    }
}
class Formulario extends JFrame implements ActionListener{
    JPanel miCapa;
    JButton miBoton;
    JButton salirBoton;
    JLabel miTexto;
    JLabel miMensaje;
    JTextField miCampoTexto;

    public Formulario() {
        this.setTitle("Mi Aplicación Java");
        this.setSize(500,300);
        this.setResizable(false);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);

        // Creamos un panel para colocar componentes en el Formulario
        miCapa = new JPanel();
        // Establecemos el setLayout a null para poder posicionar los
        // componentes en las posiciones x, y indicadas en setBounds
        miCapa.setLayout(null);

        // Creamos un Texto, lo posicionamos y añadimos a la capa
        miTexto = new JLabel("Nombre: ");
        miTexto.setBounds(10,20,100,30);
        miCapa.add(miTexto);

        miMensaje = new JLabel();
        miMensaje.setBounds(10,50,100,30);
        miCapa.add(miMensaje);

        // Creamos un Campo de Texto, lo posicionamos y añadimos a la capa
        miCampoTexto = new JTextField();
        miCampoTexto.setBounds(80,20,100,30);
        miCapa.add(miCampoTexto);

        // Creamos un botón, lo posicionamos y añadimos a la capa
        miBoton = new JButton("Mensaje");
        miBoton.setBounds(10,100,100,30);
        miCapa.add(miBoton);
```

```
        // Creamos un botón, lo posicionamos y añadimos a la capa
        salirBoton = new JButton("Salir");
        salirBoton.setBounds(150,100,100,30);
        miCapa.add(salirBoton);

        miBoton.addActionListener(this);
        salirBoton.addActionListener(this);

        // Añadimos la capa al Formulario
        this.add(miCapa);
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource()==miBoton) {
            String mensaje;
            mensaje = miCampoTexto.getText();
            miMensaje.setText("Hola "+mensaje);
        }
        if (e.getSource()==salirBoton) {
            System.exit(0);
        }
    }
}
```

Ejemplo: Aplicación con interface visual que permita introducir dos números en dos cajas de texto. Debe tener un botón llamado Sumar que suma los dos números introducidos en las cajas de texto y muestra el resultado en otra caja de texto no editable. Por último un botón de Salir permite cerrar la aplicación.

```
package formulario.jframe;
import java.awt.event.*;
import javax.swing.*;
public class FormularioJFrame {
    public static void main(String[] args) {
        Formulario miFormulario = new Formulario();
        miFormulario.setVisible(true);
    }
}

class Formulario extends JFrame implements ActionListener{
    JPanel miCapa;
    JButton sumarB;
    JButton salirB;
    JLabel num1L;
    JLabel num2L;
    JLabel resultadoL;
    JTextField num1TF;
    JTextField num2TF;
    JTextField resultadoTF;

    public Formulario() {
```

```
        this.setTitle("Calculadora");
        this.setSize(500,300);
        this.setResizable(false);
        this.setLocationRelativeTo(null);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);

        // Creamos un panel para colocar componentes en el Formulario
        miCapa = new JPanel();
        // Establecemos el setLayout a null para poder posicionar los
        // componentes en las posiciones x, y indicadas en setBounds
        miCapa.setLayout(null);

        // Creamos un Texto, lo posicionamos y añadimos a la capa
        num1L = new JLabel("Numero 1: ");
        num1L.setBounds(10,20,100,30);
        miCapa.add(num1L);

        num2L = new JLabel("Numero 2: ");
        num2L.setBounds(10,60,100,30);
        miCapa.add(num2L);

        resultadoL = new JLabel("Resultado: ");
        resultadoL.setBounds(10,100,100,30);
        miCapa.add(resultadoL);

        // Creamos un Campo de Texto, lo posicionamos y añadimos a la capa
        num1TF = new JTextField();
        num1TF.setBounds(80,20,100,30);
        miCapa.add(num1TF);

        // Creamos un Campo de Texto, lo posicionamos y añadimos a la capa
        num2TF = new JTextField();
        num2TF.setBounds(80,60,100,30);
        miCapa.add(num2TF);

        resultadoTF = new JTextField();
        resultadoTF.setBounds(80,100,100,30);
        resultadoTF.setEditable(false);
        miCapa.add(resultadoTF);

        // Creamos un botón, lo posicionamos y añadimos a la capa
        sumarB = new JButton("Sumar");
        sumarB.setBounds(140,180,100,30);
        miCapa.add(sumarB);

        // Creamos un botón, lo posicionamos y añadimos a la capa
        salirB = new JButton("Salir");
        salirB.setBounds(250,180,100,30);
        miCapa.add(salirB);

        sumarB.addActionListener(this);
```

```
        salirB.addActionListener(this);

        // Añadimos la capa al Formulario
        this.add(miCapa);
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource()==sumarB) {
            Double num1;
            Double num2;
            Double calculo;

            num1=Double.parseDouble(num1TF.getText());
            num2=Double.parseDouble(num2TF.getText());
            calculo = num1+num2;
            resultadoTF.setText(calculo.toString());
        }
        if (e.getSource()==salirB) {
            System.exit(0);
        }
    }
}
```