Acceso a datos.

UT 1. Manejo de ficheros.

Temporalización: 10 horas.	
OBJETIVOS DIDÁCTICOS	
RESULTADOS DE APRENDIZAJE	RA1. Desarrolla aplicaciones que gestionan información almacenada en ficheros identificando el campo de aplicación de los mismos y utilizando clases específicas.
OBJETIVOS GENERALES	h,p
COMPETENCIAS	o, p, w
CONTENIDOS	
1. Introducción. 2.Ficheros. 2.1 operaciones básicas. 2.2 librerías i/o Java 2.3 Excepciones. 2.4 Flujo de datos. De bytes y de caracteres. 2.5 Entrada y salida estándar. 2.6 Almacenamiento d objetos en ficheros. 3. Ficheros y XML. 3.1 Trabajan con ficheros XML. 3.2 Librerías para conversión de documentos XML a otros formatos. 4. JSON.	
ORIENTACIONES METODOLÓGICAS	
CRITERIO DE EVALUACIÓN	a) Se han utilizado clases para la gestión de ficheros y directorios. b) Se han valorado las ventajas y los inconvenientes de las distintas formas de acceso. c) Se han utilizado clases para recuperar información almacenada en ficheros. d) Se han utilizado clases para almacenar información en ficheros. e) Se han utilizado clases para realizar conversiones entre diferentes formatos de ficheros. f) Se han previsto y gestionado las excepciones. g) Se han probado y documentado las aplicaciones desarrolladas.

1 Introducción.

El almacenamiento y tratamiento de información es una de las tareas más destacadas en la informática. Los inicios de la informática están ligados a este tratamiento de la información, por ejemplo, los orígenes de IBM son el tratamiento de datos para diferentes gobiernos.

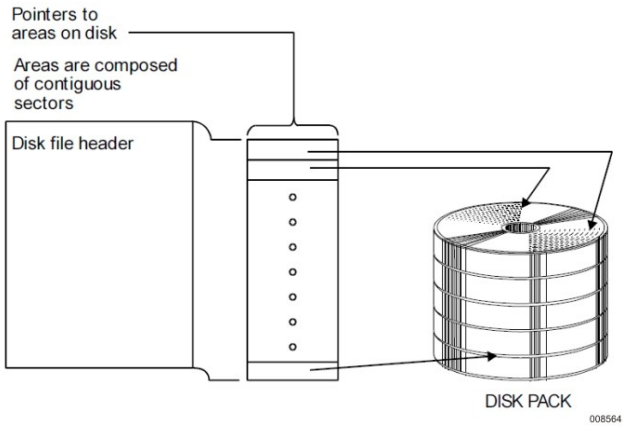
En la actualidad la IA y las tecnologías Big Data se basan el almacenamiento y análisis de gran cantidad de datos, en diferentes formatos y de diferentes fuentes como son las bases de datos relacionales, base de datos no relacionales o ficheros entre muchos otros.

Este tema trata de como trabajar con el elemento clásico y básico para gestionar información: los ficheros en el lenguaje de programación Java, aunque en el resto de los lenguajes actuales se tienen prácticamente las mismas funcionalidades. Se tratan la jerarquía de clases que ofrece Java para trabajar con ficheros, tratando las diferentes librerías que ofrece Java de forma estándar, y así como los diferentes paquetes para trabajar con datos en formato XML y JSON, tanto de forma local como de forma remota.

2 Ficheros.

Es necesario definir qué es un fichero, los equipos informáticos poseen hardware en el que es posible almacenar conjunto de bits, ya sea de forma magnético o eléctrica. Este hardware utiliza direcciones para su gestión, por ejemplo, la dirección típica en disco magnéticos es indicar: cilindro, pista y sector.

Un fichero es un concepto lógico que permite a los humanos gestionar la información, siendo el sistema operativo, más concretamente el subsistema de archivos el encargado de gestionar la información en los dispositivos hardware (discos duros, dispositivos de almacenamiento, ficheros remotos) y presentarlos ante el humano como un conjunto de ficheros y directorios. En caso de no existir el sistema de archivos, los programas tendrían que gestionar: búsqueda de huecos, evitar la sobreescritura, gestionar el dispositivo...y un largo etcétera.



Existen diferentes sistemas de archivos que determinan como aspectos como el tamaño máximo de los ficheros, la velocidad de lectura/escritura o la seguridad ante fallos, los sistemas de archivos más conocidos:

Windows:

- FAT32: Sistema de archivos más moderno que FAT16, pero aún con ciertas limitaciones, como un tamaño máximo de archivo de 4 GB. Es compatible con múltiples sistemas operativos y sigue siendo utilizado en dispositivos como memorias USB.
- exFAT (Extended File Allocation Table): Evolución de FAT32, diseñado para superar las limitaciones del tamaño máximo de archivo. Es compatible con archivos de hasta 16 EB (Exabytes), aunque no es tan eficiente como otros sistemas en almacenamiento de archivos pequeños.
- NTFS (New Technology File System): Sistema de archivos principal utilizado en las versiones modernas de Windows. Soporta tamaños de archivo y particiones muy grandes, permisos avanzados y características como cifrado, compresión y tolerancia a fallos.

Linux:

- ext4 (Fourth Extended File System): Una mejora sobre ext3, que soporta tamaños de archivo de hasta 16 TB y particiones de hasta 1 EB. Es el sistema de archivos por defecto en muchas distribuciones modernas de Linux, y ofrece mejor rendimiento y eficiencia que sus predecesores.

- ReiserFS: Sistema de archivos con buenas capacidades de manejo de archivos pequeños y técnicas avanzadas de journaling. No es tan utilizado como ext4, pero se considera innovador.
- XFS: Un sistema de archivos de alto rendimiento utilizado en Linux, conocido por su manejo eficiente de grandes archivos y su escalabilidad.
- Btrfs (B-tree File System): Sistema de archivos moderno con características avanzadas como instantáneas (snapshots), compresión, verificación de integridad de datos y soporte para almacenamiento en múltiples discos. Diseñado para mejorar sobre los sistemas ext4 y XFS en Linux.

El sistema operativo ofrece una interfaz para poder interactuar con los dispositivos, existiendo en los lenguajes de programación librerías que permiten abstraerse de los sistemas de archivo y de las características físicas de los dispositivos, incluyendo sistemas de archivos en red.

2.1 Operaciones básicas.

En todos los lenguajes de programación existen librerías para la gestión de archivos y directorios que permiten entre otros:

- Listar directorios.
- Crear, cambiar de nombre o borrar directorios.
- Crear, cambiar de nombre o borrar ficheros.
- Leer ficheros.
- Escribir en ficheros.
- Copiar ficheros

La librería básica que permite realizar estas acciones en Java es **java.io**, dentro de este paquete destaca la clase File con una gran cantidad de métodos destacando:

Type	Method	Description
`boolean`	`canExecute()`	Tests whether the application can execute the file denoted by this

		abstract pathname.
`boolean`	`canRead()`	Tests whether the application can read the file denoted by this abstract pathname.
`boolean`	`canWrite()`	Tests whether the application can modify the file denoted by this abstract pathname.
`int`	`compareTo(File pathname)`	Compares two abstract pathnames lexicographically.
`boolean`	`createNewFile()`	Atomically creates a new, empty file named by this abstract pathname if and only if a file with this name does not yet exist.
`static File`	`createTempFile(String prefix, String suffix)`	Creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name.
`static File`	`createTempFile(String prefix, String suffix, File directory)`	Creates a new empty file in the specified directory, using the given prefix and suffix strings to generate its name.
`boolean`	`delete()`	Deletes the file or directory denoted by this abstract pathname.
`void`	`deleteOnExit()`	Requests that the file or directory denoted by this abstract pathname be deleted when the virtual machine terminates.
`boolean`	`equals(Object obj)`	Tests this abstract pathname for

		equality with the given object.
`boolean`	`exists()`	Tests whether the file or directory denoted by this abstract pathname exists.
`File`	`getAbsoluteFile()`	Returns the absolute form of this abstract pathname.
`String`	`getAbsolutePath()`	Returns the absolute pathname string of this abstract pathname.
`File`	`getCanonicalFile()`	Returns the canonical form of this abstract pathname.
`String`	`getCanonicalPath()`	Returns the canonical pathname string of this abstract pathname.
`long`	`getFreeSpace()`	Returns the number of unallocated bytes in the partition named by this abstract path name.
`String`	`getName()`	Returns the name of the file or directory denoted by this abstract pathname.
`String`	`getParent()`	Returns the pathname string of this abstract pathname's parent, or `null` if this pathname does not name a parent directory.
`File`	`getParentFile()`	Returns the abstract pathname of this abstract pathname's parent, or `null` if this pathname does not name a parent directory.
`String`	`getPath()`	Converts this abstract pathname into

		a pathname string.
`long`	`getTotalSpace()`	Returns the size of the partition named by this abstract pathname.
`long`	`getUsableSpace()`	Returns the number of bytes available to this virtual machine on the partition named by this abstract pathname.
`int`	`hashCode()`	Computes a hash code for this abstract pathname.
`boolean`	`isAbsolute()`	Tests whether this abstract pathname is absolute.
`boolean`	`isDirectory()`	Tests whether the file denoted by this abstract pathname is a directory.
`boolean`	`isFile()`	Tests whether the file denoted by this abstract pathname is a normal file.
`boolean`	`isHidden()`	Tests whether the file named by this abstract pathname is a hidden file.
`long`	`lastModified()`	Returns the time that the file denoted by this abstract pathname was last modified.
`long`	`length()`	Returns the length of the file denoted by this abstract pathname.
`String[]`	`list()`	Returns an array of strings naming the files and directories in the directory denoted by this abstract pathname.

`String[]`	`list(FilenameFilter filter)`	Returns an array of strings naming the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
`File[]`	`listFiles()`	Returns an array of abstract pathnames denoting the files in the directory denoted by this abstract pathname.
`File[]`	`listFiles(FileFilter filter)`	Returns an array of abstract pathnames denoting the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
`File[]`	`listFiles(FilenameFilter filter)`	Returns an array of abstract pathnames denoting the files and directories in the directory denoted by this abstract pathname that satisfy the specified filter.
`static File[]`	`listRoots()`	List the available filesystem roots.
`boolean`	`mkdir()`	Creates the directory named by this abstract pathname.
`boolean`	`mkdirs()`	Creates the directory named by this abstract pathname, including any necessary but nonexistent parent directories.
`boolean`	`renameTo(File dest)`	Renames the file denoted by this

		abstract pathname.
boolean	`setExecutable(boolean executable)`	A convenience method to set the owner's execute permission for this abstract pathname.
boolean	`setExecutable(boolean executable, boolean ownerOnly)`	Sets the owner's or everybody's execute permission for this abstract pathname.
boolean	`setLastModified(long time)`	Sets the last-modified time of the file or directory named by this abstract pathname.
boolean	`setReadable(boolean readable)`	A convenience method to set the owner's read permission for this abstract pathname.
boolean	`setReadable(boolean readable, boolean ownerOnly)`	Sets the owner's or everybody's read permission for this abstract pathname.
boolean	`setReadOnly()`	Marks the file or directory named by this abstract pathname so that only read operations are allowed.
boolean	`setWritable(boolean writable)`	A convenience method to set the owner's write permission for this abstract pathname.
boolean	`setWritable(boolean writable, boolean ownerOnly)`	Sets the owner's or everybody's write permission for this abstract pathname.
Path	`toPath()`	Returns a `java.nio.file.Path` object constructed from this abstract

		path.
`String`	`toString()`	Returns the pathname string of this abstract pathname.
`URI`	`toURI()`	Constructs a `file:` URI that represents this abstract pathname.
`URL`	`toURL()`	Deprecated. This method does not automatically escape characters that are illegal in URLs.

Un ejemplo de uso de la clase File:

```
File f = new File("C:\\Users\\Pedro\\Desktop\\ejemplojava.txt");
System.out.println("Existe el fichero:" + f.exists());
if (f.createNewFile()) {
    System.out.println("Se ha creado el fichero" +
f.getAbsolutePath());
}
System.out.println("Comprobar si ahora existe:" + f.exists());
System.out.println("¿Es un fichero?" + f.isFile());
System.out.println("¿Es un directorio'" + f.exists());
System.out.println("La longitud es:" + f.length());
System.out.println("La URI es:" + f.toURI().toString());
f.renameTo(new File("C:\\Users\\Pedro\\Desktop\\ejemplojava2.txt"));
f.delete();
System.out.println("Existe el fichero:" + f.exists());
```

Siendo la salida:

```
Existe el fichero:false
Se ha creado el ficheroC:\Users\Pedro\Desktop\ejemplojava.txt
Comprobar si ahora existe:true
¿Es un fichero?true
¿Es un directorio'true
La longitud es:0
La URI es:file:/C:/Users/Pedro/Desktop/ejemplojava.txt
```

```
Existe el fichero:false
```

También exista la clase estática Files, con funcionalidad más similar a los comandos que se pueden realizar desde el terminal, pero en este caso se encuentra en el paquete **java.nio** (new input/output). Este paquete aparece en la versión 7 de Java, alguna de la funcionalidad que ofrece esta clase son:

- static long copy(InputStream in, Path target, CopyOption... options)
- static Path createDirectories(Path dir, FileAttribute<?>... attrs)
- static Path createFile(Path path, FileAttribute<?>... attrs)
- static Path createLink(Path link, Path existing)
- static boolean deleteIfExists(Path path)
- static long size(Path path)

Como se puede observar la clase no define ningún método para leer o escribir, Java define una abstracción para trabajar con ficheros o con cualquier flujo de datos, los Streams, tratados en puntos posteriores.

2.2 Librerías I/O en Java.

Una de las tareas más importante que realizan algunas aplicaciones es el manejo de la entrada y salida, ya sea al sistema de ficheros o a la red. Desde las versiones iniciales de Java se ha mejorado el soporte, añadiendo programación asíncrona de E/S, permitir obtener información de atributos propios del sistema de archivos, reconocimiento de enlaces simbólicos y facilitado de algunas operaciones básicas entre otras.

Java NIO.2

En las primeras versiones de Java el sistema de entrada/salida proporcionado en el paquete **java.io** era básico. En la versión 1.4 de Java se añadió un nuevo sistema de entrada/salida llamado **NIO** para suplir algunas de sus deficiencias, que posteriormente en Java 7 se mejoró aún más con NIO.2. Entre las mejoras se incluyen:

- Permitir navegación de directorios sencillo.
- Soporte para reconocer enlaces simbólicos.
- Leer atributos de ficheros como permisos e información como última fecha de modificación.
- Soporte de entrada/salida asíncrona.
- Capacidad para operaciones básicas sobre ficheros como copiar y mover ficheros.

2.2.1 NIO.

Este paquete define a su vez más paquetes relacionados con la entrada salida, se centra en faciliatar la concurrencia, el uso de la red y la ejecución de hilos.

Se basa en:

- Uso del concepto de buffer (almacenamiento temporal con estructura en cola, en la que se tiene productores y consumidores), el procesamiento no es instantaneo. El ejemplo más claro es cuando se tiene un vídeo que se quiere

reproducir de internet, en los primeros instantes se llena el buffer (solo el productor deja datos) hasta llegar un punto que el consumidor comienza a consumirlo, todo ello de forma asíncrona, cada elemento trabaja de forma independiente.

- Charset, encargado de los diferentes conjuntos de caracteres unicode y bytes.
- Channels (Canales), nuevo concepto que amplia el concepto de fichero representando elementos con capacidad de operaciones de I/O, con selectores, multiplexación y operaciones no bloqueantes de I/O, en concreto los ficheros y los sockets (IP+PUERTO).
- Selectors (Selectores) que junto con los canales definen la forma de multiplexión y bloqueo.

Los paquetes más destacados son:

PAQUETE	DESCRIPCIÓN
java.nio	Posee el resto de paquetes, y se centra en la definición de diferentes tipos de buffers.
java.nio.channels	Define el concepto de canal, más abstracto que el fichero, ya que puede ser ficheros o conexionescon el exterior denominadas sockets (conexión a nivel TCP o UDP). **Son operaciónes multiplexadas y sin bloqueo.**
java.nio.charset	Relacionadado con los juegos de caracteres, la codificación y descodificación.
java.nio.file	Amplia la funcionalidad de java.io con respecto a los ficheros, estableciendo diferentes interfaces o clases como Files

	(estática), FielStore, Paths o FileSystem entre otras.
java.nio.file.attribute	Permite interactuar con le sistema de ficheros

Este tema se centra en las clase java.nio.file, quedando los detalles de buffers y canales fuera de los contenidos del módulo.

Las clases principales de java.io.file son las siguientes:

Path: Es una interfaz sobre una ruta de un sistema de ficheros. No tiene porque existir en el sistema de ficheros, pero si cuando se hacen algunas operaciones como la lectura del fichero que representa. Puede usarse como reemplazo completo de java.io.File, aunque si fuera necesario con los métodos File.toPath() y Path.toFile() se ofrece compatibilidad entre ambas representaciones.Los sistemas de archivos poseen una estructura en árbol, denomiándose Path al caminio ya sea desde la raíz (absoluto, C:/ (depende unidad) o / dependiendo del sistema) o desde el punto actual (relativo, con , o ./) Se dispone de la clase Path para trabajar con el sistema de ficheros ofreciendo funcionalidad para :

- Crear paths.
- Obtener información.
- Eliminar redundancia.
- Convertir path de relativos a absolutos y a la inversa.
- Unir paths o comparar paths entre otros.

Con la clase Path se pueden hacer operaciones sobre rutas como obtener la ruta absoluta de un Path relativo o el Path relativo de una ruta absoluta, cuantos elementos se compone la ruta, obtener el Path padre o una parte de una ruta. Otros métodos interesantes son relativize(), normalize(), toAbsolutePath(), resolve(), startsWith() y endsWith().

Un ejemplo de algunos de los métodos de la clase:

```
Path camino= Paths.get("C:\\ti\\ccs1040\\ccs\\doc\\css.png");
//informacion
System.out.println("Ruta:"+camino);
System.out.println("Es ruta absoluta:"+camino.isAbsolute());
System.out.println("Root:"+camino.getRoot());
System.out.println("Padre:"+camino.getParent());
System.out.println("Subcamino:"+camino.subpath(1,2));
System.out.println("Numero de nodos del
camino:"+camino.getNameCount());
```

Cuya salida es:

```
Ruta:C:\ti\ccs1040\ccs\doc\css.png
Es ruta absoluta:true
Root:C:\
Padre:C:\ti\ccs1040\ccs\doc
Subcamino:ccs1040
Numero de nodos del camino:
```

Files: Clase que posee **todos sus métodos estáticos** con utilizades para realizar operaciones básicas sobre ficheros. Con la clase Files es posible obtener información del fichero como nombre, enlace simbólico o permisos entre otros. Esta clase tiene decenas de métodos para realizar operaciones con ficheros.

Modifier and Type	Method and Description
static long	copy(InputStream in, Path target, CopyOption... options) Copies all bytes from an input stream to a file.
static long	copy(Path source, OutputStream out) Copies all bytes from a file to an output stream.
static Path	copy(Path source, Path target, CopyOption... options) Copy a file to a target file.
static Path	createDirectories(Path dir, FileAttribute<?>... attrs) Creates a directory by creating all nonexistent parent directories first.
static Path	createDirectory(Path dir, FileAttribute<?>... attrs) Creates a new directory.
static Path	createFile(Path path, FileAttribute<?>... attrs) Creates a new and empty file, failing if the file already exists.
static Path	createLink(Path link, Path existing) Creates a new link (directory entry) for an existing file *(optional operation)*.
static Path	createSymbolicLink(Path link, Path target, FileAttribute<?>... attrs) Creates a symbolic link to a target *(optional operation)*.
static Path	createTempDirectory(Path dir, String prefix, FileAttribute<?>... attrs) Creates a new directory in the specified directory, using the given prefix to generate its name.
static Path	createTempDirectory(String prefix, FileAttribute<?>... attrs) Creates a new directory in the default temporary-file directory, using the given prefix to generate its name.
static Path	createTempFile(Path dir, String prefix, String suffix, FileAttribute<?>... attrs) Creates a new empty file in the specified directory, using the given prefix and suffix strings to generate its name.
static Path	createTempFile(String prefix, String suffix, FileAttribute<?>... attrs) Creates an empty file in the default temporary-file directory, using the given prefix and suffix to generate its name.
static void	delete(Path path) Deletes a file.
static boolean	deleteIfExists(Path path) Deletes a file if it exists.

Un ejemplo de uso de Files que incluye también uso de clases FileStore y Path:

```
  public static void main(String[] args) throws IOException {

        Path directoriopath;

        directoriopath = Files.createDirectories(Paths.get("C:\\Users\\
Pedro\\Desktop\\ejemploFiles"));

        System.out.println("Se ha creado el directorio:" +
directoriopath.toAbsolutePath() + "?" + Files.exists(directoriopath));

        Path fichero = directoriopath.resolve("nuevo.txt");

        //por si se ha creado antes

        if (fichero.toFile().exists()) {

            fichero.toFile().delete();

        }

        Files.createFile(fichero);

        System.out.println("Se ha creado el fichero:" +
fichero.toAbsolutePath() + "?" + Files.exists(fichero));

        FileStore almacen = Files.getFileStore(directoriopath);

        System.out.println("El almacenamiento tiene un espacio total
de:" + almacen.getTotalSpace());

        System.out.println("Y libre:" + almacen.getUsableSpace());

        String extension = "js";

        //se hace uso de la programación funcional y los streams
```

```
        Files.find(Paths.get("C:\\Users\\Pedro\\Desktop"), 5, (path,
atributos) -> {


            if (path.getFileName().toString().lastIndexOf(".") > -1
&& !path.toFile().isDirectory()) {
                String tempoextension =
path.getFileName().toString().substring(path.getFileName().toString().l
astIndexOf(".") + 1);
                return extension.equals(tempoextension);
            } else {
                return false;
            }
        }).forEach(cnsmr -> {
            System.out.println("Encontrado fichero con extensón (" +
extension + "):" + cnsmr.toAbsolutePath());
        });
    }
```

FileSystem: Interfaz para interacturar con sistema de archivos. En cada sistema operativo la implementación es diferente, es usado por la clase FileSystems.

FileSystems: otra clase de utilidad como punto de entrada para obtener referencias a sistemas de archivos. Representa el sistema de ficheros y permite principalmente:

- Que el software Java interactue con el sistema de ficheros de forma sencilla.
- Una factoría para crear diferentes objetos y servicios relacionados con el sistema de ficheros.

All Methods | Static Methods | Concrete Methods

Modifier and Type	Method and Description
static FileSystem	getDefault() Returns the default FileSystem.
static FileSystem	getFileSystem(URI uri) Returns a reference to an existing FileSystem.
static FileSystem	newFileSystem(Path path, ClassLoader loader) Constructs a new FileSystem to access the contents of a file as a file system.
static FileSystem	newFileSystem(URI uri, Map<String,?> env) Constructs a new file system that is identified by a URI
static FileSystem	newFileSystem(URI uri, Map<String,?> env, ClassLoader loader) Constructs a new file system that is identified by a URI

Un ejemplo de uso de FileSystems en Windows:

```
    public static void main(String[] args) {
        FileSystem sf = FileSystems.getDefault();
        System.out.println("Es de solo lectura?" + sf.isReadOnly());
        System.out.println("Está abierto?" + sf.isOpen());
        System.out.println("Unidades montandas (en Windows letras)");
        System.out.println("El separador del sistema
es"+sf.getSeparator());

        sf.getRootDirectories().forEach(rd -> {
            System.out.println(rd.toAbsolutePath() + "\n");
        });
        System.out.println("Características de los sistemas de
ficheros"+sf.getSeparator());

       sf.getFileStores().forEach(fs -> {
            try {
                System.out.println("Tamaño del
bloque"+fs.getBlockSize());
                System.out.println("Tipo"+fs.type());
            } catch (IOException ex) {

Logger.getLogger(ejemploFilesystems.class.getName()).log(Level.SEVERE,
null, ex);
            }
        });
    }
Y la salida (depende del equipo).
Es de solo lectura?false
Está abierto?true
Unidades montandas (en Windows letras)
El separador del sistema es:\
     C:\
     D:\
     E:\
     F:\
     G:\
Características de los sistemas de ficheros:
```

```
     Tamaño del bloque:512  Tipo:NTFS
     Tamaño del bloque:512  Tipo:NTFS
     Tamaño del bloque:512  Tipo:NTFS
     Tamaño del bloque:512  Tipo:NTFS
     Tamaño del bloque:512  Tipo:FAT32
```

2.2.2 NIO2.

Destacar que NIO.2 se ha desarrollado **pensando en entrada/salida asíncrona** como puede ser la solicitud de un fichero en red, o la petición de una web. Para ello define dos paradigmas, uno basado en la clase Future y otro en funciones de rellamada o callbacks. La programación asíncrona evita bloquear el hilo que ejecuta el código y aprovecha mejor los procesadores multinúcleo con lo que se mejora el rendimiento de las aplicaciones. Para los ficheros se usa la clase AsynchronousFileChannel y para flujos de red AsynchronousSocketChannel. Estos contenidos se tratarán en mayor profundidad en otros módulos.

Un ejemplo de uso de NIO.2:

```
public static void main(String[] args) throws Exception {
        // Path
        System.out.println("# info");
        Path relative = Paths.get(".");
        Path absolute = relative.toAbsolutePath().normalize();
        System.out.printf("Relative: %s%n", relative);
        System.out.printf("Absolute: %s%n", absolute);
        System.out.printf("Name count: %d%n", absolute.getNameCount());
        System.out.printf("Parent: %s%n", absolute.getParent());
       System.out.printf("Subpath(0, 2): %s%n", absolute.subpath(0,
2));
        // ls -la
        System.out.println();
        System.out.println("# ls -la");
        Files.walk(relative, 1).sorted((p1, p2) -> {
return p1.getFileName().toString().compareTo (p2.getFileName().
ToString() );
        }).forEach(path -> {
            try {
                System.out.println(toLine(path));
```

```
            } catch (Exception e) {
                e.printStackTrace();
            }
        });
        // File operations
        Path file = Paths.get("build.gradle");
        Path backup = Paths.get("build.gradle.backup");
        Path rename = Paths.get("build.gradle.backup.1");
        Files.copy(file, backup, StandardCopyOption.REPLACE_EXISTING);
        Files.move(backup, rename,
StandardCopyOption.REPLACE_EXISTING);
        Files.delete(rename);

        // Read
        System.out.println("");
        System.out.println("# build.gradle");
        Files.readAllLines(file).stream().forEach(l -> {
            System.out.println(l);
        });
        //
        System.out.println("");
        System.out.println("# async with Future");
        {
            AsynchronousFileChannel channel =
AsynchronousFileChannel.open(file, StandardOpenOption.READ);
            ByteBuffer buffer = ByteBuffer.allocate(100_000);
            Future<Integer> result = channel.read(buffer, 0);
            // ...
            Integer bytesRead = result.get();
            System.out.println(new String(buffer.array(), "utf-8"));
        }
        System.out.println("");
        System.out.println("# async with callback");
        {
            AsynchronousFileChannel channel =
AsynchronousFileChannel.open(file, StandardOpenOption.READ);
            ByteBuffer buffer = ByteBuffer.allocate(100_000);
```

```
            channel.read(buffer, 0, buffer, new
CompletionHandler<Integer, ByteBuffer>() {
                public void completed(Integer result, ByteBuffer
buffer) {
                    try {
                        System.out.println(new String(buffer.array(),
"utf-8"));
                    } catch (UnsupportedEncodingException e) {
                        e.printStackTrace();
                    }
                }

                public void failed(Throwable exception, ByteBuffer
attachment) {
                    System.out.println(exception.getMessage());
                }
            });

            Thread.sleep(2000);
}
```

En el ejemplo anterior se ven dos técnicas que no casan exactamente con la programación OO:

Future objeto que representa un resultado que no se tiene en el momento, sino que se tendrá disponible en el futuro, pudiendo establecer métodos para los casos de cancelación o de finalización, los métodos son cancel(), get(), get (sobrecargado), isCancelled() y isDone(), forma parte del paquete java.util.concurrent,

CallBack: Se le pasa un objeto al que se llamará cuando finalice la acción.

Mencionar que la programación asíncrona o reactiva en los lenguajes actuales está cada vez más presente, por ejemplo en node.js(Javascript en el servidor), javascript o las librerías RXJava o el framework Spring.

2.3 Excepciones.

Una excepción es una error en **tiempo de ejecución** que se produce de manera inesperada, las causas pueden ser ajenas al software o por falta de previsión de ciertas situaciones.

El ejemplo clásico es la división de un número entre 0, que provoca un fallo. Al crear el software o bien comprueba antes de hacer la división que el divisor no sea 0 o insertar código que trate el posible error en caso de que se produzca.

En Java cuando se produce un error en tiempo de ejecución, la JVM realiza un primer tratamiento devolviendo un objeto de tipo excepción, ya que como se ha tratado en Java todo son objetos.

```
public class Ejemplo1 {
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        int dividendo,divisor,resultado;
        Scanner input= new Scanner(System.in);
        dividendo=input.nextInt();
        divisor=input.nextInt();
        resultado=dividendo/divisor;
        System.out.println("El resultado es "+resultado);
    }
}
```

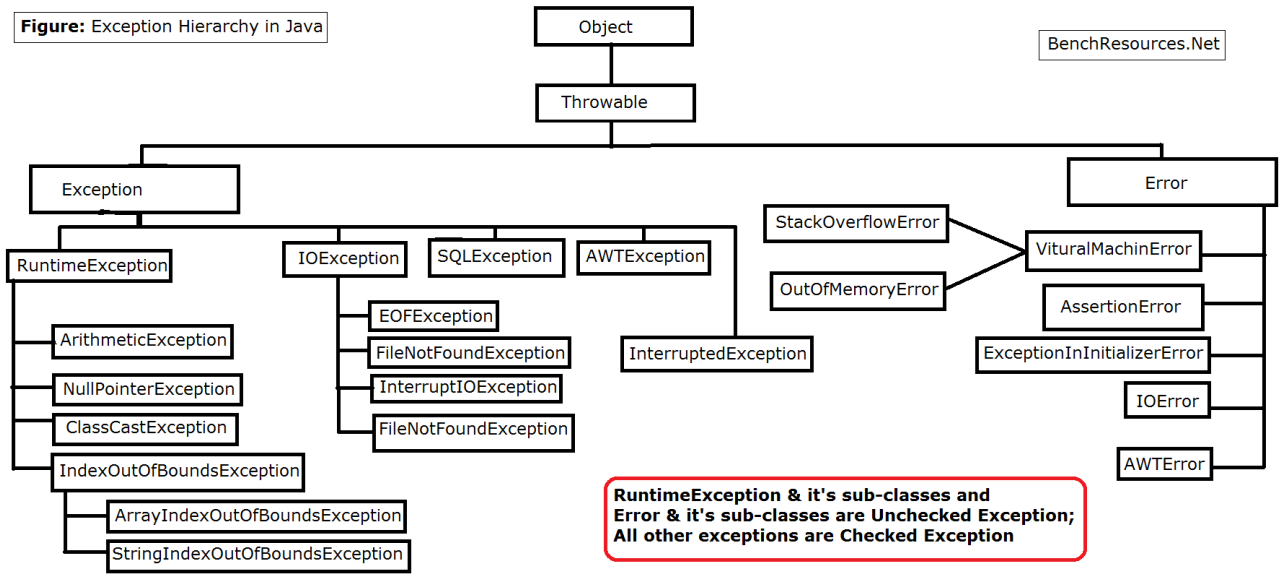
Al introducir los valores 5 y 0 se produce el siguiente error:

```
5
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
     at
pedro.ieslaencanta.com.excepciones.Ejemplo1.main(Ejemplo1.java:23)
Command execution failed.
```

La línea 23 del código es resultado=dividendo/divisor;

Es posible que en una misma instrucción se produzcan diferentes tipos de excepción, en el ejemplo anterior se pueden dar fallos y lanzarse excepciones si se divide entre 0 y también si lo que se introduce por teclado es diferente a un entero.

Existe una jerarquía de excepciones dependiendo de la causa que desencadene dicha excepción, esta jerarquía por supuesto utiliza herencia. El diagrama de clases de los errores y excepciones en Java es el siguiente:



La primera clase para los errores es Thowable, la API de dicha clase se encuentra en: https://docs.oracle.com/javase/9/docs/api/java/lang/Throwable.html, esta clase contiene la pila completa (llamada a métodos de diferentes clases desde el principal hasta la llamada que produce el error) entre otra información, un mensaje de error descriptivo de esta heredan 2 nuevas clases:

Error: Se producen cuando es un error grave y difícilmente es posible su tratamiento para que el software no falle, por ejemplo fallo de la máquina virtual, desbordamiento de la memoria o problemas con hilos. **No se suele gestionar las situaciones que hacen lanzar este tipo de error, ya que no es responsabilidad del software y poco se puee hacer.**

Por ejemplo el siguiente código hace salta un error por desbordamiento del heap (memoría dinámica que se usa en tiempo de ejecución y que és limitada).

```
public static void main(String[] args) {
      String str="desbordar";
      while(true){
          str+=str+"al montón";
      }
  }
```

Lanzándose el siguiente error:

```
Exception in thread "main" java.lang.OutOfMemoryError: Overflow: String
length out of range
      at
java.base/java.lang.StringConcatHelper.checkOverflow(StringConcatHelper
.java:57)
      at
java.base/java.lang.StringConcatHelper.mix(StringConcatHelper.java:138)
      at
pedro.ieslaencanta.com.excepciones.DesbordamientoHeap.main(Desbordamien
toHeap.java:19)
```

Exception: Errores producidos en tiempo de ejecución por el software, por ejemplo intentar acceder a una posición de un vector que no existe. **Estas si han de ser tratadas y previstas, al menos en los puntos críticos.**

Java define a partir de esta clase una gran cantidad de subclases para el tratamiento de una gran cantidad de situaciones, en la siguiente imagen se pueden observar todas las subclases de Exception.

Cada subclase añade información particular que permite tratar de forma más detallada, en la siguiente imagen se observan todas las clases que heredan de Exception:

Direct Known Subclasses:

AbsentInformationException, AclNotFoundException, ActivationException, AgentInitializationException, AgentLoadException, AlreadyBoundException, ApplicationException, AttachNotSupportedException, AWTException, BackingStoreException, BadAttributeValueExpException, BadBinaryOpValueExpException, BadLocationException, BadStringOperationException, BrokenBarrierException, CardException, CertificateException, ClassNotLoadedException, CloneNotSupportedException, DataFormatException, DatatypeConfigurationException, DestroyFailedException, ExecutionControl.ExecutionControlException, ExecutionException, ExpandVetoException, FontFormatException, GeneralSecurityException, GSSException, IllegalClassFormatException, IllegalConnectorArgumentsException, IncompatibleThreadStateException, InterruptedException, IntrospectionException, InvalidApplicationException, InvalidMidiDataException, InvalidPreferencesFormatException, InvalidTargetObjectTypeException, InvalidTypeException, InvocationException, IOException, JAXBException, JMException, JShellException, KeySelectorException, LambdaConversionException, LastOwnerException, LineUnavailableException, MarshalException, MidiUnavailableException, MimeTypeParseException, MimeTypeParseException, NamingException, NoninvertibleTransformException, NonInvertibleTransformException, NotBoundException, NotOwnerException, ParseException, ParserConfigurationException, PrinterException, PrintException, PrivilegedActionException, PropertyVetoException, ReflectiveOperationException, RefreshFailedException, RemarshalException, RuntimeException, SAXException, ScriptException, ServerNotActiveException, SOAPException, SQLException, StringConcatException, TimeoutException, TooManyListenersException, TransformerException, TransformException, UnavailableServiceException, UnmodifiableClassException, UnsupportedAudioFileException, UnsupportedCallbackException, UnsupportedFlavorException, UnsupportedLookAndFeelException, URIReferenceException, URISyntaxException, UserException, VMStartException, XAException, XMLParseException, XMLSignatureException, XMLStreamException, XPathException

---

Los más destacados son RunTimeException y IOException.

AnnotationTypeMismatchException, ArithmeticException, ArrayStoreException, BufferOverflowException, BufferUnderflowException, CannotRedoException, CannotUndoException, CatalogException, ClassCastException, ClassNotPreparedException, CMMException, CompletionException, ConcurrentModificationException, DataBindingException, DateTimeException, DOMException, DuplicateRequestException, EmptyStackException, EnumConstantNotPresentException, EventException, FileSystemAlreadyExistsException, FileSystemNotFoundException, FindException, IllegalArgumentException, IllegalCallerException, IllegalMonitorStateException, IllegalPathStateException, IllegalStateException, IllformedLocaleException, ImagingOpException, InaccessibleObjectException, IncompleteAnnotationException, InconsistentDebugInfoException, IndexOutOfBoundsException, InternalException, InvalidCodeIndexException, InvalidLineNumberException, InvalidModuleDescriptorException, InvalidModuleException, InvalidRequestStateException, InvalidStackFrameException, JarSignerException, JMRuntimeException, JSException, LayerInstantiationException, LSException, MalformedParameterizedTypeException, MalformedParametersException, MediaException, MirroredTypesException, MissingResourceException, NashornException, NativeMethodException, NegativeArraySizeException, NoSuchDynamicMethodException, NoSuchElementException, NoSuchMechanismException, NullPointerException, ObjectCollectedException, ProfileDataException, ProviderException, ProviderNotFoundException, RangeException, RasterFormatException, RejectedExecutionException, ResolutionException, ResourceRequestDeniedException, SecurityException, SPIResolutionException, SystemException, TypeConstraintException, TypeNotPresentException, UncheckedIOException, UndeclaredThrowableException, UnknownEntityException, UnknownTreeException, UnmodifiableModuleException, UnmodifiableSetException, UnsupportedOperationException, VMDisconnectedException, VMMismatchException, VMOutOfMemoryException, WebServiceException, WrongMethodTypeException, XPathException

- **RunTimeException**. Lanza excepciones relacionadas con la programación, como división entre 0 o intento de acceso a indices de vector que no existen. Las subclases de RuntimeException son:

- **<u>IOException. Muchos de los errores de los programas provienen por fallos de entrada salida, el caso más común son los ficheros, pero pueden ser otros como todos los derivados con conexiones de red, como HttpTimeoutException, que se lanza cuando una conexión HTTP tarde más de lo esperado.</u>**

Direct Known Subclasses:

AttachOperationFailedException, ChangedCharSetException, CharacterCodingException, CharConversionException, ClosedChannelException, ClosedConnectionException, EOFException, FileLockInterruptionException, FileNotFoundException, FilerException, FileSystemException, HttpRetryException, HttpTimeoutException, IIOException, InterruptedByTimeoutException, InterruptedIOException, InvalidPropertiesFormatException, JMXProviderException, JMXServerErrorException, LoadException, MalformedURLException, ObjectStreamException, ProtocolException, RemoteException, SaslException, SocketException, SSLException, SyncFailedException, TransportTimeoutException, UnknownHostException, UnknownServiceException, UnsupportedDataTypeException, UnsupportedEncodingException, UserPrincipalNotFoundException, UTFDataFormatException, WebSocketHandshakeException, ZipException

---

Las excepciones de tipo RuntimeException **no existe obligación de ser capturadas**, en cambio el resto son de tipo **Checked**, lo que **obliga a que sea capturad**a , en caso de no serlo se obtiene un error de compilación, por ejemplo, en el ejericio de generar figuras geométricas al guardar la información en un archivo:

```
            FileWriter fichero = null;
            fichero = new FileWriter(ruta);
             fichero.write(this.aSVG());
```

```
            fichero.close();
```

Al compilar se obtiene el siguiente error (además el IDE también marca el error):

```
*.java:89: error: unreported exception IOException; must be caught or
declared to be thrown
            fichero = new FileWriter(ruta);
*.java:90: error: unreported exception IOException; must be caught or
declared to be thrown
            fichero.write(this.aSVG());
*.java:91: error: unreported exception IOException; must be caught or
declared to be thrown
            fichero.close();
3 errors
```

Indicando que o bien se ha de capturar o bien ha de poder lanzarse, los detalles de como captar o indicar que ese método puede lanzar excepciones se tratan en el siguiente punto.

2.3.1 Manejo y captura.

Ya se conoce el proceso que se desencadena al producirse un error, en caso de ser un error no controlable se lanza un objeto de la clase Error, en caso de ser otro tipo de error se lanza un objeto de tipo Exception o derivado, en este caso si es posible su tratamiento.

En primer lugar se ha de identificar los lugares en que se pueden lanzar excepciones, por ejemplo, en los lugares en que se realiza una división.

Caso 1:Capturar y manejar la excepción. Existen una estructura para la captura y manejo de excepciones:

```
try{
     //código que puede hacer que se lance una exepción
}catch (Exception e1){
     //manejo de la excepción e1, que será alguno de los tipos que
heredan de exception
}catch(Exception e2){
     //manejo de la excepción e2, diferente a e1, que será alguno de
los tipos  que heredan de exception
}catch(Exception….
```

```
     //se manejan todos los tipos que se deseee
finally{
     //código que se ejecuta se produzca o no la excepción
}
```

Al manejar diferentes tipos de excepciones, el anidamiento a de ir de la más concreta a la menos concreta, ya que se captura la primera que coincide con la jerarquía de clases.

En el ejemplo anterior, para capturar 2 tipos de excepciones:

```
    public void guardarDibujo(String ruta) {
        FileWriter fichero = null;
        try {

            fichero = new FileWriter(ruta);
            fichero.write(this.aSVG());



        } catch (IOException ex) {
            System.err.println("Excepción por IO");
        } catch (Exception ex2) {
            System.err.println("Excepción general, la más alta");

        } finally {
            try {
                fichero.close();
                System.out.println("Este código se ejeucta siempre,
salte o no           salte");
            } catch (IOException ex) {

Logger.getLogger(Dibujo.class.getName()).log(Level.SEVERE, null, ex);
            }
        }

    }
```

Los métodos más destacados de una excepción son: String getMessage que da una descripción del error y printlStackTrace que muestra la pila de llamadas a objetos y métodos que ha desencadenado la expceción.

En el siguiente enlace se listan algunas buenas prácticas con resptecto al tratamiento de excepciones.

https://www.clubdetecnologia.net/blog/2017/java-buenas-practicas-para-el-manejo-de-excepciones/

2. Propagación de la excepción. En caso no tratarse el error, la excepción se propaga al método siguiente hasta que o bien encuentra un try… o se llega al método static main. En ocasiones al usar una librería, clase o método el compilador indica que es posible que se produzca una excepción de algún tipo y que esta debe tratarse **obligatoriamente**, teniendo 2 alternativas:

tratarla con try catch.

Indicar que en el método a su vez puede lanzar excepciones del tipo que se obliga a lanzar, en el ejemplo del fichero es obligatorio tratar excepciones de entrada/salida, sino se trata en el método es necesario indicar en la defínción del mismo que quien lo use habrá de tratarlo:

```
public void guardarDibujo(String ruta) throws IOException {
        FileWriter fichero = null;
        fichero = new FileWriter(ruta);
        fichero.write(this.aSVG());
        System.out.println("Este código se ejeucta siempre, salte o no salte");
    }
```

Ahora quien llame al método guardarDibujo(String ruta) o bien sigue propagando la excepción.

2.3.2 Lanzamiento de excepciones.

Hasta ahora se han capturado y tratado las excepciones, pero también es posible lanzarlas para avisar de un uso incorrecto de por ejemplo, las librerías creadas.

Para lanzar una excepción, ya sea de las predefinidas o de las personalizadas se utiliza la palabra reservada:

throws

Por ejemplo, en la división por 0 se captura la excepción Aritmética y a continuación con la información se lanza la nueva excepción creada anteriormente:

```
   int dividendo,divisor,resultado;
        Scanner input= new Scanner(System.in);
        dividendo=input.nextInt();
        divisor=input.nextInt();
        try{
        resultado=dividendo/divisor;
        }catch ( ArithmeticException e){
            throw  new DivisionporCeroException("División por 0
personalizado "+e.getMessage());
        }
        System.out.println("El resultado es "+resultado);
```

Al dividir por 0 se obtiene la siguiente salida:

```
Exception in thread "main"
pedro.ieslaencanta.com.excepciones.DivisionporCeroException: División
por 0 personalizado / by zero
     at
pedro.ieslaencanta.com.excepciones.Ejemplo1.main(Ejemplo1.java:26)
```

En las buenas prácticas de excepciones se recomienda añadir la máxima información posible, se añade la excepción inicial a la personalizada

Otra opción es no esperar a que se lance la excepción, sino detectar el problema y hacer saltar la personalizada.

```
     int dividendo, divisor, resultado;
      Scanner input = new Scanner(System.in);
      dividendo = input.nextInt();
      divisor = input.nextInt();
      if (divisor == 0) {
        throw new DivisionporCeroException("División por 0
personalizado ");
```

```
        }
        resultado = dividendo / divisor;
        System.out.println("El resultado es " + resultado);
```

Al ejecutar el código anterior:

```
Exception in thread "main"
pedro.ieslaencanta.com.excepciones.DivisionporCeroException: División
por 0 personalizado
      at
pedro.ieslaencanta.com.excepciones.Ejemplo1.main(Ejemplo1.java:24)
```

2.3.3 Excepciones relacionadas con ficheros.

En Java, las excepciones relacionadas con el manejo de ficheros son comunes cuando trabajas con operaciones de entrada/salida (I/O) como la lectura y escritura de archivos. Las excepciones que más frecuentemente son:

IOException:Es la excepción base para todos los problemas de entrada/salida en Java.Se lanza cuando ocurre un error en las operaciones de I/O, como la lectura o escritura de archivos.

FileNotFoundException: Es una subclase de IOException. Se lanza cuando el programa intenta abrir un archivo que no existe o no tiene permisos suficientes para accederlo.

SecurityException: Se lanza cuando el programa no tiene permisos para acceder o modificar un archivo (por ejemplo, cuando no tiene permisos de lectura o escritura).

Cuando se trabaja con ficheros, Java obliga a gestionar la posible excepción, ya sea con un bloque try cath o propagando la excepción.

Un ejemplo:

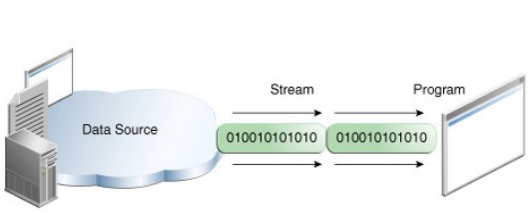
```
if (file != null) {
    try {
        this.index.saveIndex(file);
    } catch (IOException e) {
        a.show();
    }
```

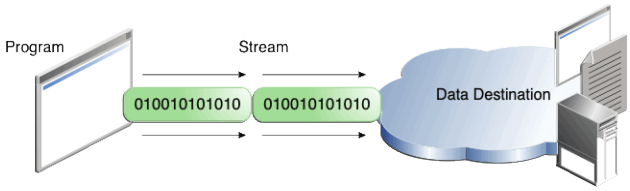
```
}
```

El método de index, saveIndex obliga a gestionar la posible excepción, pero solo si es de tipo IOExcepction o derivadas.

2.4 Flujos de datos. De bytes y de caracteres.

Java se basa en el uso de los Streams o flujos de datos para abstraer la naturaleza de la fuente o destino de los datos, pudiendo se entre otros ficheros, dispositivos de I/O, otros programas o estructuras de datos en memoria. El concepto es similar al de los Streams vistos en el tema anterior.





Estos flujos de datos hacen uso de la herencia para implementar los detalles de cada una de las clases, clasificándose en función de dos características:

- **Si es de entrada o de salida.**
- **Si trabaja con bytes o con caracteres.**

Se definen 4 interfaces básicas a partir de las cuales se establecen el resto de las clases para la entrada/salida, siendo estas y los métodos que implementan los siguientes:

InputStream:

- int available()
- void close()
- void mark(int readlimit) Marks the current position in this input stream.
- boolean markSupported() Tests if this input stream supports the mark and reset methods.
- int read()
- int read(byte[] b)
- int read(byte[] b, int off, int len)
- void reset() Repositions this stream to the position at the time the mark method was last called on this input stream.

- long skip(long n) Skips over and discards n bytes of data from this input stream.

OutputStream:

- void close()
- void flush()
- void write(byte[] b)
- void write(byte[] b, int off, int len)
- void write(int b)

Reader:

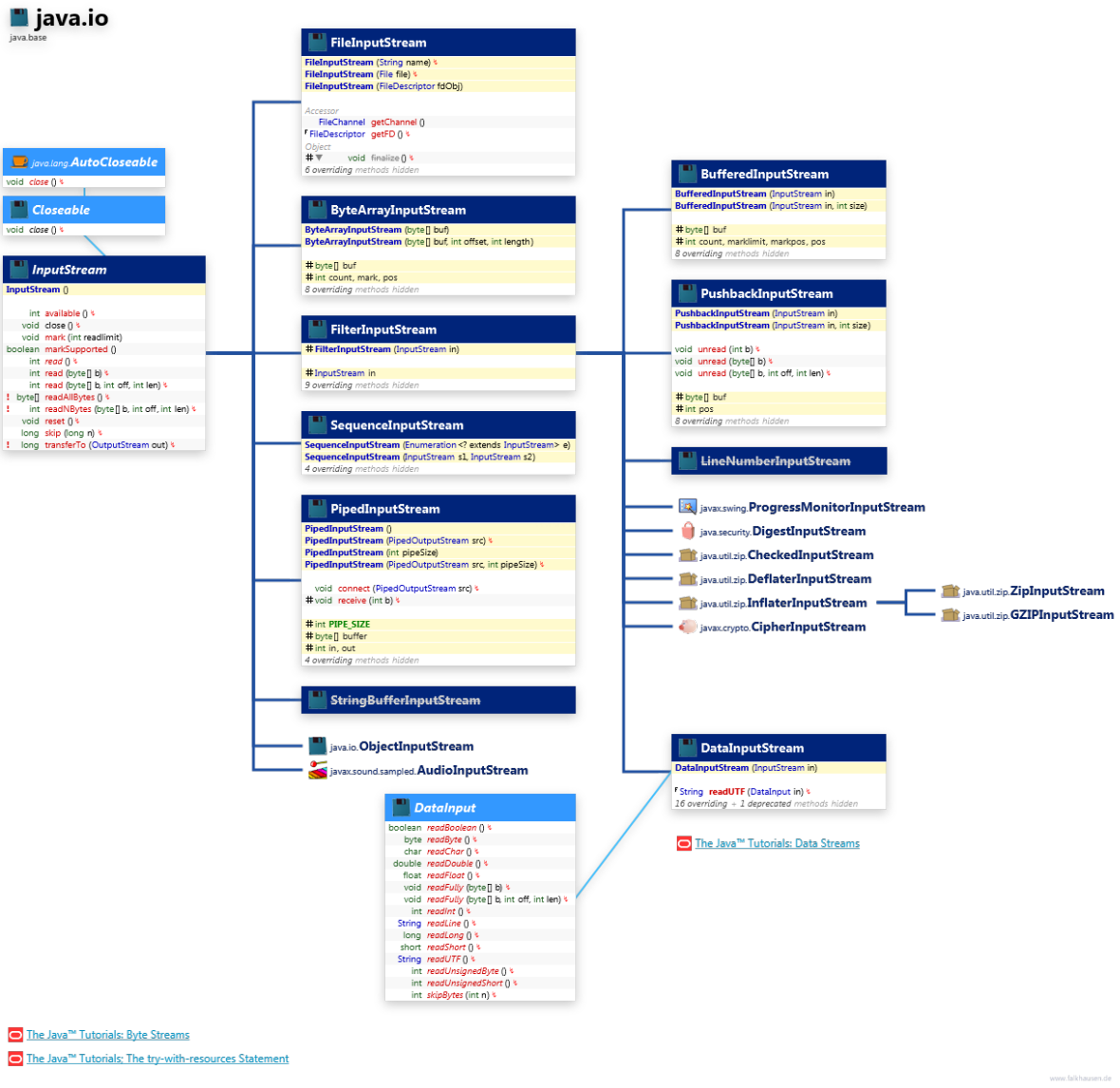
- void close()
- void mark(int readAheadLimit)
- boolean markSupported()
- int read()
- int read(char[] cbuf)
- int read(char[] cbuf, int off, int len)
- int read(CharBuffer target)
- boolean ready()
- void reset()
- long skip(long n)

Writer:

- Writer append(char c)
- Writer append(CharSequence csq)
- Writer append(CharSequence csq, int start, int end)
- void close()
- void flush()
- void write(char[] cbuf)
- void write(char[] cbuf, int off, int len)
- void write(int c)
- void write(String str)
- void write(String str, int off, int len)

En las siguientes imágenes se pueden ver la jerarquía de clases en las 4 posibles combinaciones:

Lectura de bytes:



Todas las clases heredan de InputStream y son las siguientes:

- FileInputStream
- BynaryArrayIntput
- FilterInputStream.
  - BufferedInputStream.
  - PushbackInputStream.
  - LineNumberInputStream(Deprecated).
  - DataInputStream.
- SequenceInputStream.
- PipeInputStream.
- StringBufferInputStream(Deprecated)
- ObjectInputStream.

Ejemplo de uso de InputStream, que permite leer un fichero PDF:

```
 public static void main(String[] args) throws MalformedURLException,
IOException {
        File file = new File("C:\\Users\\Pedro\\Desktop\\ejemploFiles\\
salida.pdf");
        try {
            FileInputStream input = new FileInputStream(file);
            int character;
            //-1 en caso de llegar al final del fichero
            while ((character = input.read()) != -1) {
                System.out.print((char) character);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Obteniendo el contenido del fichero PDF, un fragmento:

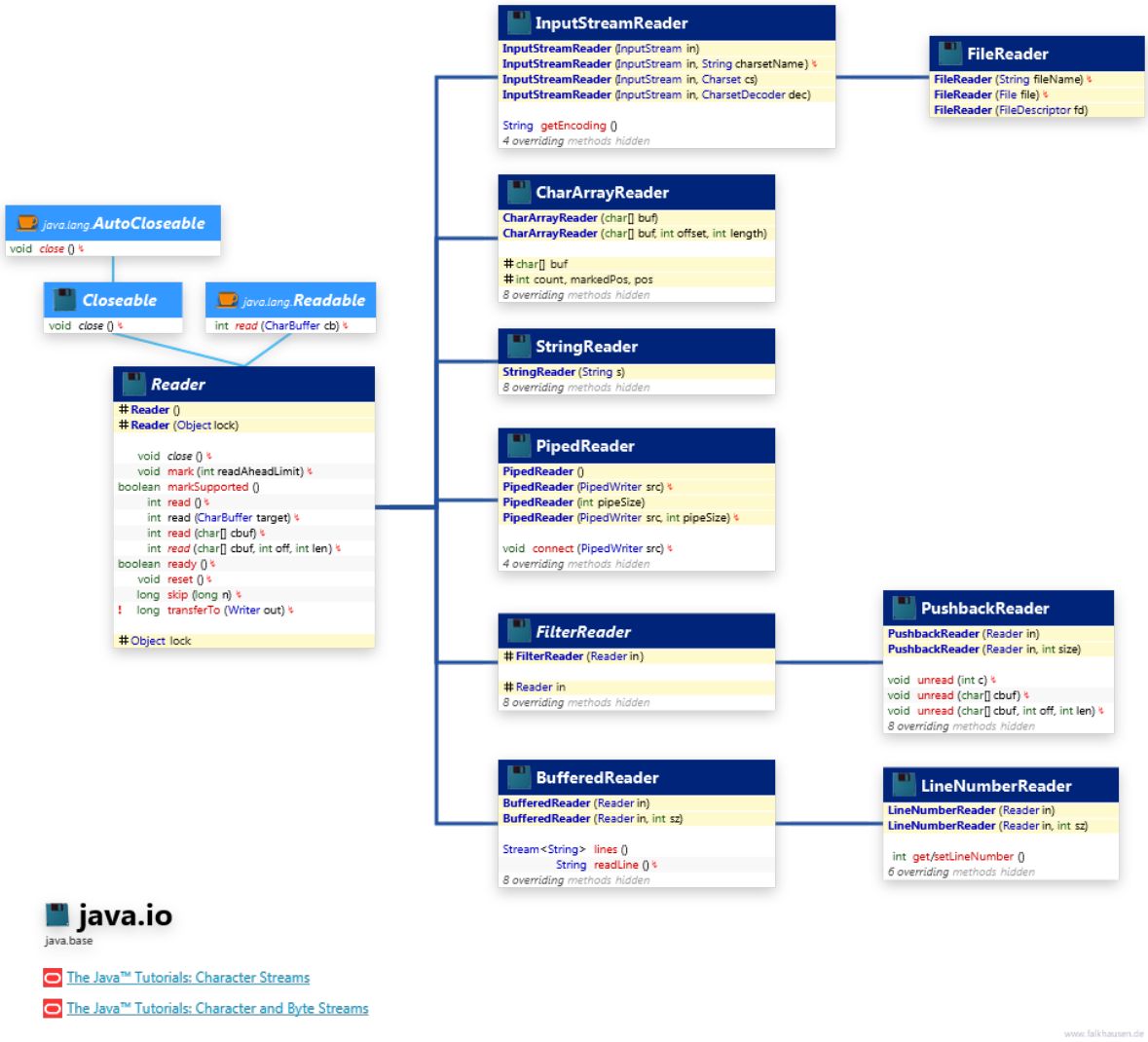
```
%PDF-1.3
%âãÏÓ

1 0 obj
<<
/Type /Catalog
/Outlines 2 0 R
/Pages 3 0 R
>>
endobj

2 0 obj
<<
/Type /Outlines
/Count 0
>>
endobj
```

- **Observar que no se lee y escribe sobre el fichero directamente, sino que es el objeto flujo de datos de entrada el que se encarga de leer.**

Ejemplo en el que se obtiene también un PDF pero en este caso de una dirección web:

```
 public static void main(String[] args) throws MalformedURLException, IOException {
        System.out.println("Iniciando la descarga");
        URL url = new URL("http://africau.edu/images/default/sample.pdf");
        //se abre el flujo de datos
        InputStream in = url.openStream();
        //se crea tambiñen un flujo de salida
        FileOutputStream fos = new FileOutputStream(new File("C:\\Users\\Pedro\\Desktop\\ejemploFiles\\salida.pdf"));

        int length = -1;
        //buffer para ir leyendo
        byte[] buffer = new byte[1024];
        //se va leyendo en un buffer de bytes y escribieno en el flujo de salida
        while ((length = in.read(buffer)) > -1) {
            fos.write(buffer, 0, length);
        }
        //se cierran los flujos
        fos.close();
        in.close();

    }
```

Lectura de caracteres:



En este caso la clase base es Reader, de las que heredan:

- InputStreamReader.
  - FileReader
- CharArrayReader.
- StringReader.
- PipedReader
- FilterReader.
  - PushbackReader.
- BufferedReader
  - LineNumberReader

Un ejemplo de lectura de un fichero de texto similar al anterior en binario, pero que lee una página web del disco duro:

```
public static void main(String[] args) throws MalformedURLException,
IOException {
        try {
            FileReader in = new
FileReader("C:/ti/ccs1040/ccs/doc/CCS_10.4.0_ReleaseNotes.htm");
            BufferedReader br= new BufferedReader(in);
            String linea;
            while((linea=br.readLine())!=null){
                System.out.println(linea);
            }
            br.close();
            in.close();

        } catch (FileNotFoundException e1) {
            System.err.println("Error: No se encuentra el fichero");
        } catch (IOException e2) {
            System.err.println("Error leyendo/escribiendo fichero");
        }
    }
```

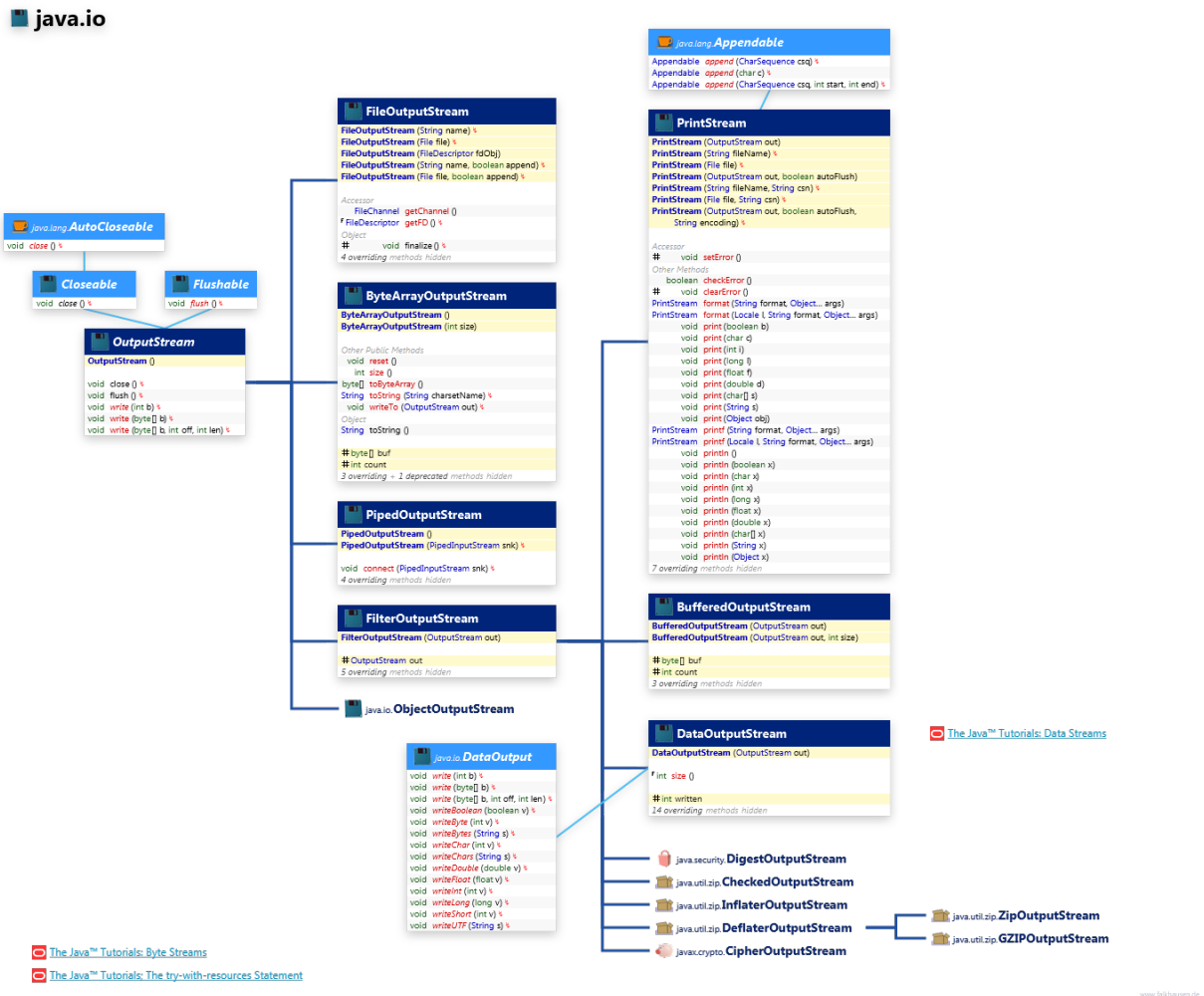
Observar como se define en primer lugar el FileReader para a continuación pasarlo a BufferedReader para poder leer línea a línea, en caso de no usar BufferedReader la lectura sería más complicada.

Otro ejemplo pero en este caso se lee una página web:

```
 public static void main(String[] args) throws MalformedURLException,
IOException {
        try {
            URL url = new URL("http://www.ieslaencanta.com/web/");
            InputStreamReader in = new
InputStreamReader(url.openStream());
            BufferedReader br = new BufferedReader(in);
            String linea;
            while ((linea = br.readLine()) != null) {
                System.out.println(linea);
            }
            br.close();
            in.close();
```

```
        } catch (FileNotFoundException e1) {
            System.err.println("Error: No se encuentra el fichero");
        } catch (IOException e2) {
            System.err.println("Error leyendo/escribiendo fichero");
        }
    }
```

Escritura de bytes:

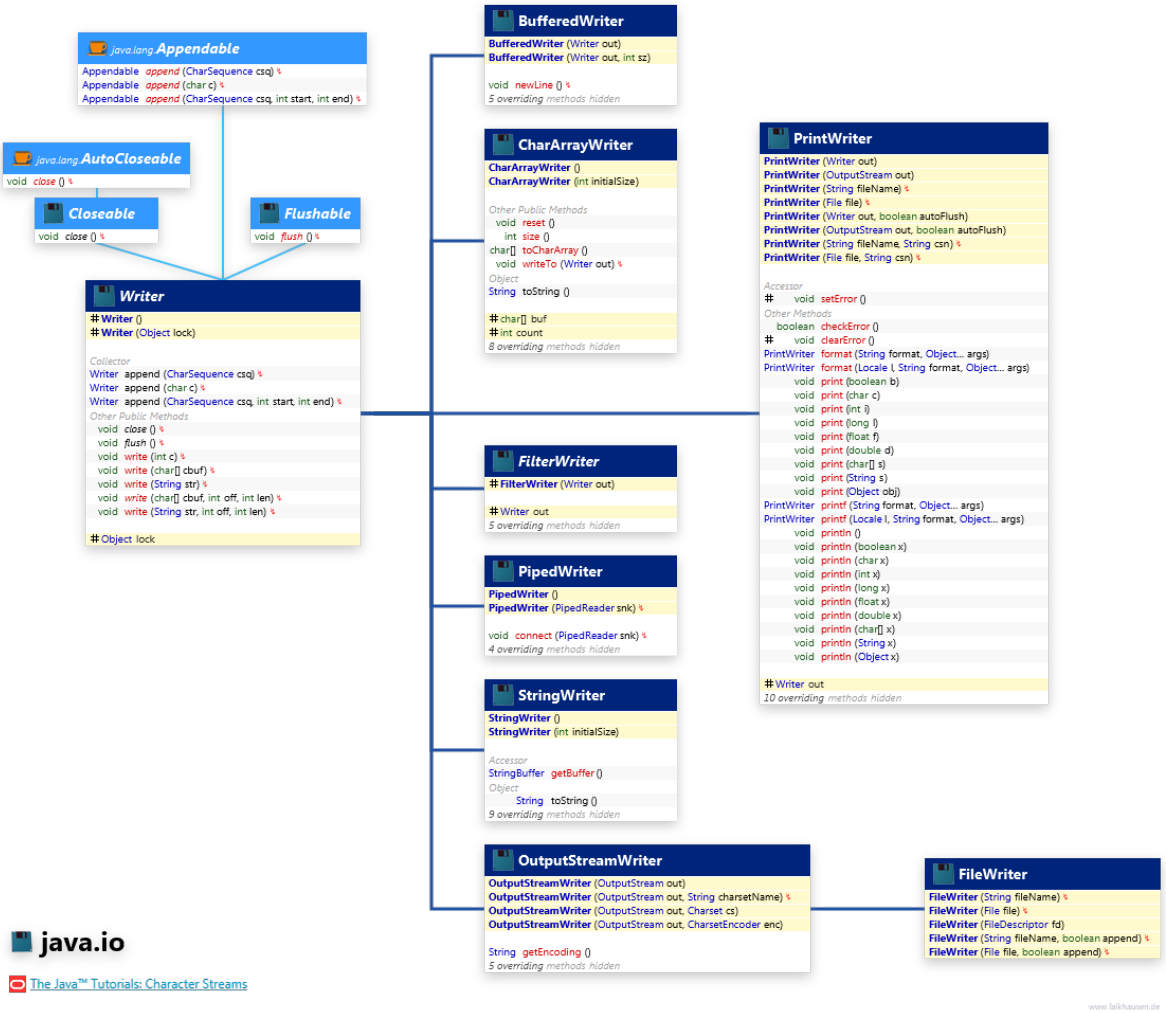


La clase base es OutpuStream de la que heredan:

- FileOutputStream
- ByteArrayOutputStream
- PipeOutputStream

- FilterOutputStream.
  - PrintStream
  - BufferedOutputStream
  - DataOutputStream
- ObjectOutputStream

Escritura de caracteres:



Al igual que en el caso de Read, la clase base es Writer y de esta heredan:

- BufferedWriter
- CharArrayWriter
- PrintWriter.
- FilterWriter.
- PipleWriter
- StrinigWriter

- OutputStreamWriter
  - FileWriter

Un ejemplo para de escritura de caracteres, por ejemplo un programa que genere una quiniela de forma aleatorio con resuntado 1,X,2 para 15 partidos:

```
    public static void main(String[] args) throws IOException {
        FileWriter out = null;
        PrintWriter p_out = null;
        char[] simbolos={'1','X','2'};
        int tope=15;
        try {
            out = new FileWriter("C:\\Users\\Pedro\\Desktop\\
result.txt");
            p_out = new PrintWriter(out);
            for(int i=0;i<15;i++){
                p_out.println((i+1)+":"+ simbolos[ (int)
(Math.random()*3)]);
            }
        } catch (IOException e) {
            System.err.println("Error al escribir en el fichero");
        } finally {
            p_out.close();
        }
    }
```

Al comparar las diferentes clases de las cuatro combinaciones:

InputStream	Reader	OutpuStream	Writer

FileInputStream BynaryArrayIntput FilterInputStream. BufferedInputStream. PushbackInputStream. LineNumberInputStream(Deprecated). DataInputStream. SequenceInputSteam. PipeInputStream. StringBufferInputStream(Deprecated) ObjectInputStream	InputStreamReader. FileReader CharArrayReader. StringReader. PipedReader FilterReader. PushbackReader. BufferedReader LineNumberReader	FileOutputStream ByteArrayOutputStream PipeOutputStream FilterOutputStream. PrintStream BufferedOutputStream DataOutputStream ObjectOutputStream	BufferedWriter CharArrayWriter PrintWriter. FilterWriter. PipleWriter StrinigWriter OutputStreamWriter FileWriter

Siempre que se abra un fichero se ha de cerrar, ya que en caso contrario se queda bloqueado(en especial la escritura) y consume recursos.

2.5 Entrada y salida estándar.

Al igual que en C, en Java también existen los conceptos de entrada, salida, y salida de error estándar. La entrada estándar normalmente se refiere a lo que el usuario escribe en la consola, aunque el sistema operativo o el programa, puede hacer que se tome de otra fuente. De la misma forma la salida estándar y la salida de error estándar normalmente es muestran los mensajes y los errores del programa respectivamente en la consola, siendo posible (aunque el sistema operativo también podrá) redirigirlas a otro destino.

En Java esta entrada, salida y salida de error estándar **se tratan de la misma forma que cualquier otro flujo de datos**, estando estos tres elementos encapsulados en tres objetos de flujo de datos que se encuentran como propiedades estáticas de la clase System:

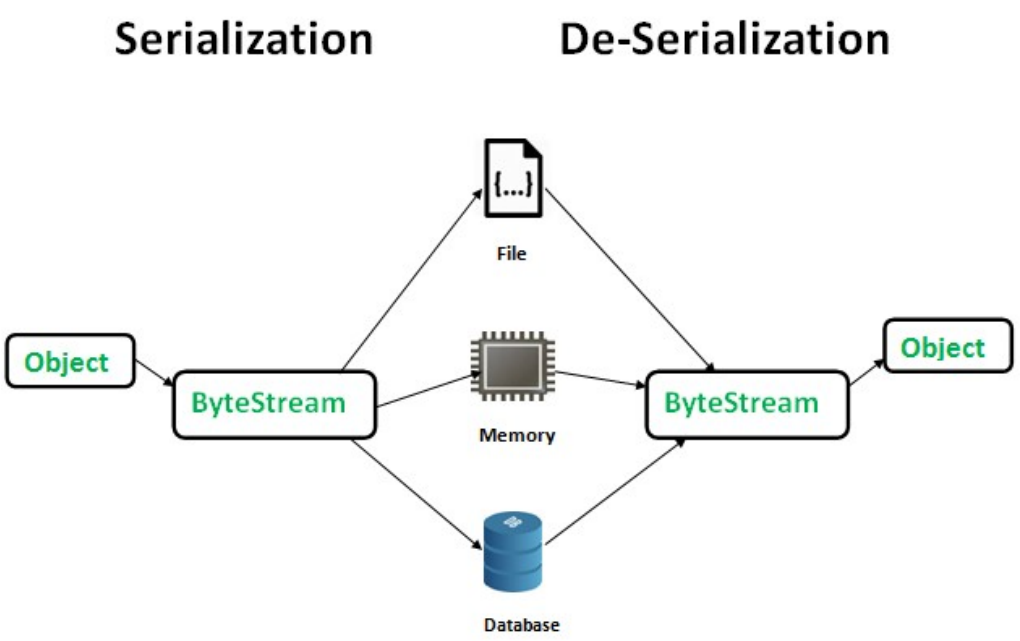
Definición	Tipo	Objeto
Stdin	InputStream	System.in
Stdout	PrintStream	System.out
Stderr	PrintStream	System.err

Para la entrada estándar se utiliza un objeto InputStream básico, sin embargo para la salida se utilizan objetos **PrintWriter que facilitan la impresión de texto** ofreciendo a parte del método común de bajo nivel write para escribir bytes, dos métodos más: print y println. Estas funciones permitirán escribir cualquier cadena, tipo básico, o bien cualquier objeto que defina el método toString que devuelva una representación del objeto en forma de cadena. La única diferencia entre los dos métodos es que el segundo añade automáticamente un salto de línea al final del texto impreso, mientras que en el primero deberemos especificar explícitamente este salto.

2.6 Almacenamiento de objetos en ficheros.

La serialización es la transformación de un objeto en una secuencia de bytes que pueden ser posteriormente leídos para reconstruir el objeto original.

El objeto serializado pueda guardarse en un fichero o puede enviarse por red para reconstruirlo en otro lugar. Puede crearse en un sistema Windows y enviarlo. por ejemplo, a otro sistema que utilice Linux.



Guardar objetos de forma que existan cuando la aplicación haya terminado se conoce como persistencia. Para poder transformar el objeto en una secuencia de bytes, el objeto debe ser serializable.

Un objeto es serializable si su clase **implementa la interface Serializable**.La interface Serializable se encuentra en el paquete java.io Es una interface vacía. No contiene ningún método.

```
public interface Serializable{
}
```

Sirve para indicar que los objetos de la clase que lo implementa se pueden serializar. Solo es necesario que una clase la implemente para que la máquina virtual pueda serializar los objetos.

Si un objeto contiene atributos que son referencias a otros objetos éstos a su vez deben ser serializables.

Todos los tipos básicos Java son serializables, así como los arrays y los String.

Persistencia de Objetos en Ficheros

Para escribir objetos en un fichero binario en Java se utiliza la clase **ObjectOutputStream** derivada de OutputStream. Un objeto ObjectOutputStream se crea a partir de un objeto **FileOutputStream** asociado a un fichero, aunque también se puede crear a partir de un socket .

El constructor de la clase es:

```
ObjectOutputStream(OutputStream nombre);
```

Por ejemplo, las instrucciones para crear el fichero personas.dat para escritura de objetos serían éstas:

```
FileOutputStream fos = new FileOutputStream ("/ficheros/personas.dat");
ObjectOutputStream salida = new ObjectOutputStream (fos);
```

En caso de un socket:

```
socket1 = new Socket(InetAddress.getLocalHost(), portNumber);
ObjectOutputStream oos = new
ObjectOutputStream(socket1.getOutputStream());
```

La clase proporciona el método writeObject(Object objeto) para escribir el objeto en el fichero. El método defaultWriteObject() de la clase ObjectOutputStream realiza de forma automática la serialización de los objetos de una clase. Este método se invoca en el método writeObject().

defaultWriteObject() escribe en el stream de salida todo lo necesario para reconstruir los objetos:

- La clase del objeto.
- Los miembros de la clase (atributos).
- Los valores de los atributos que no sean static o transient.

Para leer objectos, es decir serializar, se utilizan clases similares pero a la inversa, en concreto el método **defaultReadObject()** de la clase **ObjectInputStream** que realiza la deserialización de los objetos de una clase. Este método se invoca en el método readObject().

```
public class Persona implements Serializable{
   private String nif;
   private String nombre;
   private int edad;
   public Persona() {
   }
   public Persona(String nif, String nombre, int edad) {
       this.nif = nif;
```

```
        this.nombre = nombre;
        this.edad = edad;
    }
    public int getEdad() {
        return edad;
    }
    public void setEdad(int edad) {
        this.edad = edad;
    }
    public String getNif() {
        return nif;
    }
    public void setNif(String nif) {
        this.nif = nif;
    }
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
public class Serial1 {
    public static void main(String[] args) {

        FileOutputStream fos = null;
        ObjectOutputStream salida = null;
        Persona p;
        try {
            //Se crea el fichero
            fos = new FileOutputStream("/ficheros/personas.dat");
            salida = new ObjectOutputStream(fos);
            //Se crea el primer objeto Persona
            p = new Persona("12345678A","Lucas González", 30);
            //Se escribe el objeto en el fichero
            salida.writeObject(p);
```

```
            //Se crea el segundo objeto Persona
            p = new Persona("98765432B","Anacleto Jiménez", 28);
            //Se escribe el objeto en el fichero
            salida.writeObject(p);
            //Se crea el tercer objeto Persona
            p = new Persona("78234212Z","María Zapata", 35);
            //Se escribe el objeto en el fichero
            salida.writeObject(p);

        } catch (FileNotFoundException e) {
            System.out.println("1"+e.getMessage());
        } catch (IOException e) {
            System.out.println("2"+e.getMessage());
        } finally {
            try {
                if(fos!=null) fos.close();
                if(salida!=null) salida.close();
            } catch (IOException e) {
                System.out.println("3"+e.getMessage());
            }
        }

    }
}
```

Leer objetos de ficheros

Para leer los objetos contenidos en un fichero binario que han sido almacenados mediante ObjectOutputStream se utiliza la clase ObjectInputStream derivada de InputStream. Un objeto ObjectInputStream se crea a partir de un objeto FileInputStream asociado al fichero, a un socket o cualquier clase que herede de inputStream.

El constructor de la clase es:

```
ObjectInputStream(InputStream nombre);
```

Por ejemplo, las instrucciones para crear el objeto ObjectInputStream para lectura de objetos del fichero personas.dat serían éstas:

```
FileInputStream fis = new FileInputStream ("/ficheros/personas.dat");
ObjectInputStream entrada = new ObjectInputStream (fis);
```

La clase proporciona el método readObject() que devuelve el objeto del fichero (tipo Object). Es necesario hacer un **casting** para guardarlo en una variable del tipo adecuado.

Siguiendo con el ejemplo anterior:

```
Persona p;
        try {
            fis = new FileInputStream("/ficheros/personas.dat");
            entrada = new ObjectInputStream(fis);
            p = (Persona) entrada.readObject(); //es necesario el
casting
            System.out.println(p.getNif() + " " + p.getNombre() + " " +
p.getEdad());
            p = (Persona) entrada.readObject();
            System.out.println(p.getNif() + " " + p.getNombre() + " " +
p.getEdad());
            p = (Persona) entrada.readObject();
            System.out.println(p.getNif() + " " + p.getNombre() + " " +
p.getEdad());
        } catch (FileNotFoundException e) {
            System.out.println(e.getMessage());
        } catch (ClassNotFoundException e) {
            System.out.println(e.getMessage());
        } catch (IOException e) {
            System.out.println(e.getMessage());
        } finally {
            try {
                if (fis != null) {
                    fis.close();
                }
                if (entrada != null) {
                    entrada.close();
                }
            } catch (IOException e) {
                System.out.println(e.getMessage());
```

```
            }
        }
```

3 Ficheros, XML y JSON.

XML, es un **metalenguaje** extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C) que permite definir la gramática de lenguajes específicos. Por lo tanto, XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi en cualquier software imaginable. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil, por ejemplo, la factura electrónica (https://www.facturae.gob.es).

3.1 Trabajo con ficheros XML.

XML, es un **metalenguaje** extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C) que permite definir la gramática de lenguajes específicos. Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi en cualquier software imaginable. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil, por ejemplo la factura electrónica (https://www.facturae.gob.es).

3.1.1 Analizadores sintácticos (parser) .

Los analizadores sintácticos XML (también conocidos como parsers XML) son programas o componentes de software diseñados para leer, interpretar y procesar

documentos XML, verificando su estructura y extrayendo información útil de ellos. El propósito de un analizador sintáctico XML es transformar un archivo XML, que es simplemente texto, en una estructura de datos que pueda ser utilizada por aplicaciones o programas.

Existen 2 filosofías a la hora de analizar SAX (basados en eventos) y DOM(basados en árboles cargados de forma completa en memoria) como es una página web en un navegador.

Se recomienda utilizar el estándar DOM con documentos XML pequeños, cuando se quiera analizar el documento múltiples veces o editarlo, ya que se encuentra cargado en memoria, también cuando se desee generar un documento XML desde cero.

En cambio es recomendable utilizar el estándar SAX con documentos XML grandes, cuando se quiera analizar el documento una sola vez o por partes (capturando los elementos importantes). También se usa cuando no se requiere una modificación estructural.

Un ejemplo de parseo con DOM:

```
<?xml version="1.0"?>
<company>
        <staff id="1001">
                <firstname>yong</firstname>
                <lastname>mook kim</lastname>
                <nickname>mkyong</nickname>
                <salary>100000</salary>
        </staff>
        <staff id="2001">
                <firstname>low</firstname>
                <lastname>yin fong</lastname>
                <nickname>fong fong</nickname>
                <salary>200000</salary>
        </staff>
</company>
```

```
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;
import org.w3c.dom.Element;
import java.io.File;
public class ReadXMLFile {

  public static void main(String argv[]) {

    try {

      File fXmlFile = new File("/Users/mkyong/staff.xml");
      DocumentBuilderFactory dbFactory =
DocumentBuilderFactory.newInstance();
      DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
      Document doc = dBuilder.parse(fXmlFile);
      doc.getDocumentElement().normalize();
      System.out.println("Root element :" +
doc.getDocumentElement().getNodeName());
      NodeList nList = doc.getElementsByTagName("staff");
      System.out.println("----------------------------");
      for (int temp = 0; temp < nList.getLength(); temp++) {
            Node nNode = nList.item(temp);
            System.out.println("\nCurrent Element :" +
nNode.getNodeName());
            if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                  Element eElement = (Element) nNode;
                  System.out.println("Staff id : " +
eElement.getAttribute("id"));
                  System.out.println("First Name : " +
eElement.getElementsByTagName("firstname").item(0).getTextContent());
                  System.out.println("Last Name : " +
eElement.getElementsByTagName("lastname").item(0).getTextContent());
                  System.out.println("Nick Name : " +
eElement.getElementsByTagName("nickname").item(0).getTextContent());
```

```
                System.out.println("Salary : " +
eElement.getElementsByTagName("salary").item(0).getTextContent());


            }
      }
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

}
```

Análisis del XML:

Es necesario tener claro qué librería se adapta con mayor facilidad a las necesidades del proyecto.

- **Xerces2** permite el procesamiento de documentos XML tanto con el estándar DOM o con el estándar SAX. Además, integra a otras librerías independientes de parseado.
- **JDOM** permite leer, escribir, crear y manipular ficheros XML de forma sencilla e intuitiva. Está totalmente programada en Java, lo que le permite utilizar las capacidades particulares del lenguaje, simplificando significativamente su manejo para programadores expertos en Java. Se basa en el procesamiento de un documento XML y la construcción de un árbol. Una vez construido el árbol se puede acceder directamente a cualquiera de sus componentes.
- **JAXP** (Java API for XML Processing) permite procesar tanto el estándar DOM como el estándar SAX. Este API está diseñado para ser flexible y uniformar el desarrollo de aplicaciones Java con Xml. Además, proporciona una capa intermedia que nos permite usar cualquier analizador XML compatible dentro de nuestra aplicación, reduciendo el acoplamiento de los componentes de la aplicación con la implementación del analizador.

Serialización del XML:

La serialización es un mecanismo ampliamente usado para transportar objetos a través de una red, para hacer persistente un objeto en un archivo o base de datos, o para distribuir objetos idénticos a varias aplicaciones o localizaciones.

Se recomienda utilizar las librerías Xerces2, Xstream o JAXB para serializar objetos Java en un medio de almacenamiento, como puede ser un archivo o un buffer de memoria, con el fin de transmitirlo a través de una conexión en red.

Para llevar a cabo este proceso se han de definir mediante reglas tanto el proceso de transformación de objetos Java a XML como el proceso de transformación inversa.

Un ejemplo de serialización utilizando JAXB para **JEE**:

```
import javax.xml.bind.annotation.XmlElement;
import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Libro {
    private String titulo;
    private int paginas;

    public String getTitulo() {
        return titulo;
    }
    public Libro(String titulo, int paginas) {
        super();
        this.titulo = titulo;
        this.paginas = paginas;
    }
    public Libro() {
        super();
    }
    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }
    @XmlElement(name="numeroPaginas")
    public int getPaginas() {
```

```
        return paginas;
    }
    public void setPaginas(int paginas) {
        this.paginas = paginas;
    }
}
```

Destacar el uso de anotaciones par conseguir la serialización como @XmlRootElement o @XmlElement(name="numeroPaginas")

Para serializar un elemento:

```
    public static void main(String[] args) {


        try {
            Libro libro= new Libro("Odisea 2001",400);
            JAXBContext contexto = JAXBContext.newInstance(
                    libro.getClass() );
                Marshaller marshaller = contexto.createMarshaller();

marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
                    Boolean.TRUE);
                marshaller.marshal(libro, System.out);
        } catch (PropertyException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } catch (JAXBException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

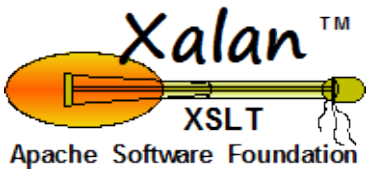
Para cargarlo:

```
  public static void main(String[] args) {


        try {
            JAXBContext context =
JAXBContext.newInstance( Libro.class );
```

```
            Unmarshaller unmarshaller = context.createUnmarshaller();
            Libro libro = (Libro)unmarshaller.unmarshal(
                new File("src/Libro.xml") );
            System.out.println(libro.getTitulo());
            System.out.println(libro.getPaginas());

        } catch (JAXBException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }
```

3.2 Librerías para conversión de documentos XML a otros formatos.

En ocasiones es necesario realizar transformaciones de documentos XML con lenguajes basados en XSL (XSLT, XPath). Un documento XML puede ser transformado en distintos lenguajes derivados de XML, como HTML, o en otro tipo como Json. Para realizar estas transformaciones basadas en XSLT, se recomienda el empleo de la librería Xalan.



Aunque no es la única existente, existen otras como:

JAXP (API estándar de Java): Fácil de usar para transformaciones básicas, incluida en el JDK.

Saxon: Implementación potente y de alto rendimiento con soporte para XSLT 2.0/3.0.

XmlBeans y JDOM: Bibliotecas auxiliares para trabajar con XML, que pueden combinarse con XSLT para transformaciones.

Un ejemplo de transformación con Xalan:

Fichero xml:

```
<?xml version="1.0"?>
```

```
<catalog>
   <book id="bk101">
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>An in-depth look at creating applications
      with XML.</description>
   </book>
   <book id="bk102">
      <author>Ralls, Kim</author>
      <title>Midnight Rain</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2000-12-16</publish_date>
      <description>A former architect battles corporate zombies,
      an evil sorceress, and her own childhood to become queen
      of the world.</description>
   </book>
   <book id="bk103">
      <author>Corets, Eva</author>
      <title>Maeve Ascendant</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2000-11-17</publish_date>
      <description>After the collapse of a nanotechnology
      society in England, the young survivors lay the
      foundation for a new society.</description>
   </book>
   <book id="bk104">
      <author>Corets, Eva</author>
      <title>Oberon's Legacy</title>
      <genre>Fantasy</genre>
      <price>5.95</price>
      <publish_date>2001-03-10</publish_date>
```

```
    <description>In post-apocalypse England, the mysterious
    agent known only as Oberon helps to create a new life
    for the inhabitants of London. Sequel to Maeve
    Ascendant.</description>
</book>
<book id="bk105">
    <author>Corets, Eva</author>
    <title>The Sundered Grail</title>
    <genre>Fantasy</genre>
    <price>5.95</price>
    <publish_date>2001-09-10</publish_date>
    <description>The two daughters of Maeve, half-sisters,
    battle one another for control of England. Sequel to
    Oberon's Legacy.</description>
</book>
<book id="bk106">
    <author>Randall, Cynthia</author>
    <title>Lover Birds</title>
    <genre>Romance</genre>
    <price>4.95</price>
    <publish_date>2000-09-02</publish_date>
    <description>When Carla meets Paul at an ornithology
    conference, tempers fly as feathers get ruffled.</description>
</book>
<book id="bk107">
    <author>Thurman, Paula</author>
    <title>Splish Splash</title>
    <genre>Romance</genre>
    <price>4.95</price>
    <publish_date>2000-11-02</publish_date>
    <description>A deep sea diver finds true love twenty
    thousand leagues beneath the sea.</description>
</book>
<book id="bk108">
    <author>Knorr, Stefan</author>
    <title>Creepy Crawlies</title>
```

```
    <genre>Horror</genre>
    <price>4.95</price>
    <publish_date>2000-12-06</publish_date>
    <description>An anthology of horror stories about roaches,
    centipedes, scorpions  and other insects.</description>
</book>
<book id="bk109">
    <author>Kress, Peter</author>
    <title>Paradox Lost</title>
    <genre>Science Fiction</genre>
    <price>6.95</price>
    <publish_date>2000-11-02</publish_date>
    <description>After an inadvertant trip through a Heisenberg
    Uncertainty Device, James Salway discovers the problems
    of being quantum.</description>
</book>
<book id="bk110">
    <author>O'Brien, Tim</author>
    <title>Microsoft .NET: The Programming Bible</title>
    <genre>Computer</genre>
    <price>36.95</price>
    <publish_date>2000-12-09</publish_date>
    <description>Microsoft's .NET initiative is explored in
    detail in this deep programmer's reference.</description>
</book>
<book id="bk111">
    <author>O'Brien, Tim</author>
    <title>MSXML3: A Comprehensive Guide</title>
    <genre>Computer</genre>
    <price>36.95</price>
    <publish_date>2000-12-01</publish_date>
    <description>The Microsoft MSXML3 parser is covered in
    detail, with attention to XML DOM interfaces, XSLT processing,
    SAX and more.</description>
</book>
<book id="bk112">
```

```
      <author>Galos, Mike</author>
      <title>Visual Studio 7: A Comprehensive Guide</title>
      <genre>Computer</genre>
      <price>49.95</price>
      <publish_date>2001-04-16</publish_date>
      <description>Microsoft Visual Studio 7 is explored in depth,
      looking at how Visual Basic, Visual C++, C#, and ASP+ are
      integrated into a comprehensive development
      environment.</description>
   </book>
</catalog>
```

Fichero xsl:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:fo="http://www.w3.org/1999/XSL/Format">


    <xsl:output method="html" indent="no" omit-xml-declaration="yes"
        doctype-public="-//W3C//DTD XHTML 1.0 Transitional//EN"
        doctype-system="http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd"
        encoding="iso-8859-1" />
    <xsl:template match="catalog">
        <html lang="en">
            <head>
                <meta charset="UTF-8" />
                <title>Level up lunch book transformation with
XSLT</title>
                <!-- Latest compiled and minified CSS -->
                <link rel="stylesheet"

href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.1/css/bootstrap.min
.css" />
                <!-- Optional theme -->
                <link rel="stylesheet"
href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.1/css/bootstrap-
theme.min.css" />
```

```
                <!-- Latest compiled and minified JavaScript -->
                <script

src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.1/js/bootstrap.min.j
s"></script>
            </head>
            <body>
                <h2>Level up lunch - XSL + XML = Bootstrap</h2>
                <table class="table table-striped">
                    <thead>
                        <tr>
                            <th>Title</th>
                            <th>Author</th>
                            <th>Genre</th>
                            <th>Description</th>
                        </tr>
                    </thead>
                    <tbody>
                        <xsl:for-each select="book">
                            <tr>
                                <td>
                                    <xsl:value-of select="title" />
                                </td>
                                <td>
                                    <xsl:value-of select="author" />
                                </td>
                                <td>
                                    <xsl:value-of select="genre" />
                                </td>
                                <td>
                                   <xsl:value-of
select="description" />
                                </td>
                            </tr>
                        </xsl:for-each>
                    </tbody>
                </table>
```

```
            </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

Transformación a HTML:

```
public static void main(String[] args) throws TransformerException {
    StreamSource xlsStreamSource = new StreamSource(Paths
            .get("src/test/resources/books.xsl")
            .toAbsolutePath().toFile());
    StreamSource xmlStreamSource = new StreamSource(Paths
            .get("src/test/resources/books.xml")
            .toAbsolutePath().toFile());
    TransformerFactory transformerFactory =
TransformerFactory.newInstance(
            "org.apache.xalan.processor.TransformerFactoryImpl", null);
    Path pathToHtmlFile = Paths.get("src/test/resources/myfile.html");
    StreamResult result = new StreamResult(pathToHtmlFile.toFile());
    Transformer transformer =
transformerFactory.newTransformer(xlsStreamSource);
    transformer.transform(xmlStreamSource, result);

}
```

Si se usa Saxon con los mismos ficheros de entrada:

```
import net.sf.saxon.TransformerFactoryImpl;
import javax.xml.transform.*;
import javax.xml.transform.stream.StreamSource;
import javax.xml.transform.stream.StreamResult;
import java.io.File;

public class SaxonTransformacionXSLT {
    public static void main(String[] args) throws Exception {
        // Crear fábrica de transformadores con Saxon
        TransformerFactory factory = new TransformerFactoryImpl();

        // Cargar hoja de estilos XSLT
        StreamSource xslt = new StreamSource(new
File("transformacion.xsl"));

        // Crear un transformador
        Transformer transformer = factory.newTransformer(xslt);
```

```
        // Cargar el archivo XML
        StreamSource xml = new StreamSource(new File("archivo.xml"));

        // Especificar el resultado de la transformación
        StreamResult salida = new StreamResult(new
File("salida.html"));

        // Aplicar la transformación
        transformer.transform(xml, salida);
    }
}
```

4 JSON.

Si bien XML es uno de los estándares de tratamientos de información estructurada en ficheros su complejidad y tamaño de las librerías para su manejo hicieron aparecer otros formatos más sencillos entre los que destaca JSON, estándar de facto para el manejo de información, intercambio entre aplicaciones o consumo de servicios, si bien XML sigue utilizándose cuando se necesita su potencia.

JSON (JavaScript Object Notation) es un formato para intercambio de datos liviano, basado en texto e independiente del lenguaje de programación, que resulta fácil de escribir y leer tanto para los seres humanos como para las máquinas. JSON puede representar dos tipos estructurados: **objetos y matrices**. Un objeto es una colección no ordenada de cero o más pares de nombres/valores. Una matriz es una secuencia ordenada de cero o más valores. Los valores pueden ser cadenas, números, booleanos, nulos y estos dos tipos estructurados.

Un ejemplo de un fichero JSON:

```
{
     "firstName": "John",
     "lastName": "Smith",
     "age": 25,
     "address": {
        "streetAddress": "21 2nd  Street",
```

```
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [
        {
            "type": "home",
            "number": "212  555-1234"
        },
        {
            "type": "fax",
            "number": "646  555-4567"
        }
    ]
}
```

Procesamiento JSON:

La API de Java para procesamiento JSON (JSR 353) proporciona rutinas que permiten analizar, generar, transformar y consultar JSON usando la API de modelos de objetos y de streaming.

- **La API de modelos de objetos** crea una estructura de árbol, **de acceso aleatorio**, que representa los datos JSON almacenados en la memoria. Es posible **recorrer el árbol y formular consultas**. Este modelo de programación es el más flexible y posibilita el procesamiento en casos en que se requiera **acceso aleatorio** a la totalidad del contenido de la memoria. Sin embargo, a menudo no es tan eficiente como el modelo de streaming y requiere más memoria.

- **La API de streaming** ofrece un modo de analizar y generar JSON en streams. Le otorga al programador el control sobre el análisis y la generación. La API de streaming ofrece un analizador **basado en eventos** y brinda al desarrollador de aplicaciones la posibilidad de "pedir" el evento siguiente en lugar de tener que ocuparse del evento en una devolución de llamada. De este modo, el desarrollador cuenta con **mayor control procedimental** del procesamiento JSON. El código de aplicación puede procesar o descartar el evento del analizador y pedir el siguiente evento (extraer el evento). El modelo de streaming es **adecuado para el procesamiento local cuando no se requiere acceso aleatorio** a otras porciones de la información. De manera similar, la API de streaming **permite generar JSON bien formado en stream** escribiendo un evento por vez.

Clases principales de la API de modelos de objetos

Clase o interfaz	Descripción
Json	Contiene métodos estáticos para crear lectores, escritores, constructores de JSON y sus objetos de fábrica.
JsonGenerator	Escribe datos JSON en forma de stream, con un valor por vez.
JsonReader	Lee datos JSON de un stream y crea un modelo de objeto en la memoria.
JsonObjectBuilder JsonArrayBuilder	Crean un modelo de objeto o un modelo de matriz en la memoria agregando valores del código de aplicación.
JsonWriter	Escribe un modelo de objeto de la memoria en un stream.
JsonValue JsonObject JsonArray JsonString JsonNumber	Representan tipos de datos para valores en datos JSON.

Las clases principales de la API Stream:

Clase o interfaz	Descripción
Json	Contiene métodos estáticos para crear lectores, escritores, constructores de JSON y sus objetos de fábrica.
JsonParser	Representa eventos del análisis al leer datos del flujo de datos
JsonGenerator	Escribe datos Json en el flujo de datos

Un ejemplo de lectura de JSON usando la API de modelo:

```
String personJSONData =
          "  {" +
          "   \"name\": \"Jack\", " +
          "   \"age\" : 13, " +
          "   \"isMarried\" : false, " +
          "   \"address\": { " +
          "     \"street\": \"#1234, Main Street\", " +
          "     \"zipCode\": \"123456\" " +
          "   }, " +
```

```
            "   \"phoneNumbers\": [\"011-111-1111\", \"11-111-1111\"] "
+
            " }";
        JsonReader reader = Json.createReader(new
StringReader(personJSONData));
        JsonObject personObject = reader.readObject();
        reader.close();
       System.out.println("Name    : " +
personObject.getString("name"));
       System.out.println("Age     : " + personObject.getInt("age"));
       System.out.println("Married: " + personObject.getBoolean
("isMarried"));
        JsonObject addressObject =
personObject.getJsonObject("address");
        System.out.println("Address: ");
        System.out.println(addressObject.getString("street"));
        System.out.println(addressObject.getString("zipCode"));
        System.out.println("Phone   : ");
       JsonArray phoneNumbersArray =
personObject.getJsonArray("phoneNumbers");
        for (JsonValue jsonValue : phoneNumbersArray) {
            System.out.println(jsonValue.toString());
        }
```

Un ejemplo extraido de Oracle Usando la API de Stream, dirigida por eventos (próximo tema).

```
URL url = new URL("https://graph.facebook.com/search?
q=java&type=post");
 try (InputStream is = url.openStream();
      JsonParser parser = Json.createParser(is)) {
     while (parser.hasNext()) {
         Event e = parser.next();
         if (e == Event.KEY_NAME) {
             switch (parser.getString()) {
                 case "name":
                    parser.next();
                    System.out.print(parser.getString());
                    System.out.print(": ");
```

```
                    break;
                case "message":
                    parser.next();
                    System.out.println(parser.getString());
                    System.out.println("---------");
                    break;
            }
        }
    }
}
```

Esto es un poco engorroso, Google provee de una librería similar a la vista en XML que facilita la lectura y escritura de objetos a json y viceversa Gson.
https://github.com/google/gson.

Algunos ejemplos extraidos de la documentación oficial:
Clase a serializar/deserialitar

```
public class BagOfPrimitives {
    private int value1 = 1;
    private String value2 = "abc";
    //no lo pasa a json, ni serializa
    private transient int value3 = 3;

    BagOfPrimitives() {

    }

    public static void main(String[] args) {
        //crear el objeto
        BagOfPrimitives obj = new BagOfPrimitives();
        //instanciar un objecto de la librería
        Gson gson = new Gson();
        //pasar a json
        String json = gson.toJson(obj);
        System.out.println(json);
    }
}
```

La salida del programa anterior:

```
{"value1":1,"value2":"abc"}
```

En caso de tener un array de elementos:

```
public class BagOfPrimitives {


    private int value1 =1;
    private String value2 = "abc";
    private transient int value3 = 3;


    BagOfPrimitives() {
        this.value1=(int)(Math.random()*20);
        // no-args constructor
    }


    public static void main(String[] args) {
        //crear el objeto
        BagOfPrimitives[] vector= new BagOfPrimitives[10];
        for(int i=0;i<vector.length;i++)
            vector[i]= new BagOfPrimitives();
        //instanciar un objecto de la librería
        Gson gson = new Gson();
        //pasar a json
        String json = gson.toJson(vector);
        System.out.println(json);
    }


}
```

Con la salida:

```
[{
      "value1": 18,
      "value2": "abc"
}, {
      "value1": 0,
      "value2": "abc"
}, {
      "value1": 11,
      "value2": "abc"
}, {
```

```
      "value1": 16,
      "value2": "abc"
}, {
      "value1": 0,
      "value2": "abc"
}, {
      "value1": 11,
      "value2": "abc"
}, {
      "value1": 8,
      "value2": "abc"
}, {
      "value1": 2,
      "value2": "abc"
}, {
      "value1": 8,
      "value2": "abc"
}, {
      "value1": 2,
      "value2": "abc"
}]
```

Ahora el paso contrario, se tiene el array en json y se desea pasar a objectos:

```
        String json = "[{\n"
                + "    \"value1\": 18,\n"
                + "    \"value2\": \"abc\"\n"
                + "}, {\n"
                + "    \"value1\": 0,\n"
                + "    \"value2\": \"abc\"\n"
                + "}, {\n"
                + "    \"value1\": 11,\n"
                + "    \"value2\": \"abc\"\n"
                + "}, {\n"
                + "    \"value1\": 16,\n"
                + "    \"value2\": \"abc\"\n"
                + "}, {\n"
                + "    \"value1\": 0,\n"
```

```
                + "    \"value2\": \"abc\"\n"
                + "}, {\n"
                + "    \"value1\": 11,\n"
                + "    \"value2\": \"abc\"\n"
                + "}, {\n"
                + "    \"value1\": 8,\n"
                + "    \"value2\": \"abc\"\n"
                + "}, {\n"
                + "    \"value1\": 2,\n"
                + "    \"value2\": \"abc\"\n"
                + "}, {\n"
                + "    \"value1\": 8,\n"
                + "    \"value2\": \"abc\"\n"
                + "}, {\n"
                + "    \"value1\": 2,\n"
                + "    \"value2\": \"abc\"\n"
                + "}]";

        //instanciar un objecto de la librería
        Gson gson = new Gson();
        //pasar a objetos
        BagOfPrimitives[] vector = gson.fromJson(json,
BagOfPrimitives[].class);
        System.out.println(vector.length);
```