TEMA 7. Procesamiento de archivos.

Estructuras de datos avanzadas y POO. Interfaces funcionales y Streams.

Índice.

Estructuras de datos avanzadas y POO. Colecciones en Java.

1. Clase File.

Diferentes sistemas de archivos: NTFS, FAT3, ext4…

Conjunto de clases para abstraer del sistema de archivos.

Clase principal de abstracción File.

- Limitada.
- Información y modificar ficheros y directorios.
- Gran cantidad métodos.
- Coincide con propiedades fichero Linux

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.
boolean	delete()	Deletes the file or directory denoted by this abstract pathname.
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.
long	length()	Returns the length of the file denoted by this abstract pathname.
boolean	mkdir()	Creates the directory named by 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.

1. Clase File.

Ejemplo:

```
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());
```

Con 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
f.delete();
System.out.println("Existe el fichero:" +
f.exists());
```

Estructuras de datos avanzadas y POO. Colecciones en Java.

1. Clase File.

Ejercicio: Crear un programa simple que liste los archivos y directorios a partir de un objeto de tipo File. Un ejemplo de salida:

```
.installedComponents.properties directorio:false
.installedProductFamilies directorio:false
ccs_base directorio:true
doc directorio:true
eclipse directorio:true
install_info directorio:true
install_logs directorio:true
tirex3 directorio:true
tirex4 directorio:true
tools directorio:true
uninstallers directorio:true
uninstall_ccs.dat directorio:false
uninstall_ccs.exe directorio:false
utils directorio:true
```

Estructuras de datos avanzadas y POO. Colecciones en Java.

2. Librerias de I/O: I/O, NIO y NIO.2.

Evolución desde los inicios para introducir mejoras y nuevos conceptos:

- IO. Inicial
- NIO. Introducido en la versión 1.4
- NIO.2 A partir de Java 7.

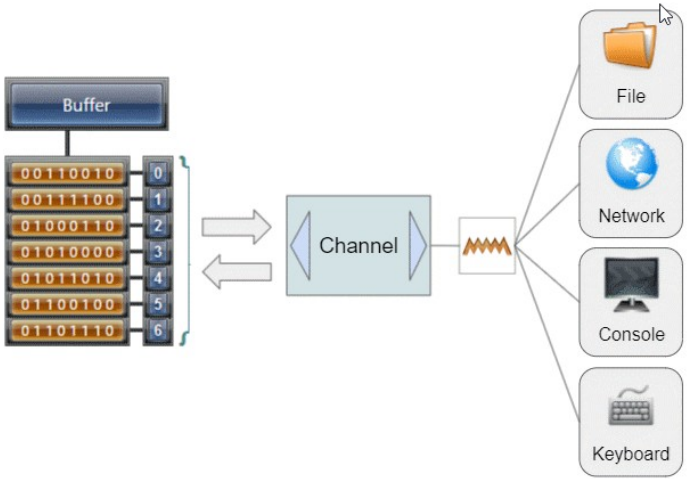
Mejoras:

- Programación asíncrona.
- Información própia del sistema de archivos.
- Enlace simbólicos.
- Mejora de velocidad.
- Red
- ...

Java NIO.2

Estructuras de datos avanzadas y POO. Colecciones en Java.

2. Librerias de I/O: I/O, **NIO** y NIO.2.



Mejoras:

- Concurrencia.
- Uso de red.
- Ejecución de hilos.

Fundamentos:

- Uso de buffers (cola con productor/consumidor).
- Charset. Diferentes conjunto de caracteres (unicode, bytes).
- Channels. Nueva abstracción con capacidad de selección, multiplexación y operaciones no bloqueantes (ficheros y sockets(IP+puerto).
- Selectors. Permiten junto con los canales la multiplexación y el bloqueo.

Estructuras de datos avanzadas y POO. Colecciones en Java.

2. Librerias de I/O: I/O, **NIO** y NIO.2.

Paquetes destacados:

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

2. Librerias de I/O: I/O, **NIO** y NIO.2.

Clases más destacadas:

Path. Interfaz sobre la ruta de ficheros. Funcionalidad:

- Crear paths.
- Obtener información.
- Eliminar redundancia.
- Convertir paths relativos en absolutos.
- Unir paths o comparar paths.

Métodos destacados:

- Relativize().
- Normalize().
- ToAbsolutePath().

Ejemplo:

```
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());
```

Salida:

```
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:
```

Estructuras de datos avanzadas y POO. Colecciones en Java.

2. Librerias de I/O: I/O, **NIO** y NIO.2.

Clases más destacadas:

Files. Clase con métodos estáticos.

- Operaciones **básicas** sobre ficheros:
  - Nombre, enlace simbólico, permisos...
- Múltitud de métodos.

All Methods | Static Methods | Concrete Methods

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.

2. Librerias de I/O: I/O, **NIO** y NIO.2.

```
public static void main(String[] args) throws IOException {
        Path directoriopath;
        directoriopath = Files.createDirectories(Paths.get("C:\\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().lastIndexOf(".") + 1);
                return extension.equals(tempoextension);
            } else {
                return false;
            }
        }).forEach(cnsmr -> {
            System.out.println("Encontrado fichero con extensón (" + extension + "):" + cnsmr.toAbsolutePath());
        });
    }
```

Estructuras de datos avanzadas y POO. Colecciones en Java.

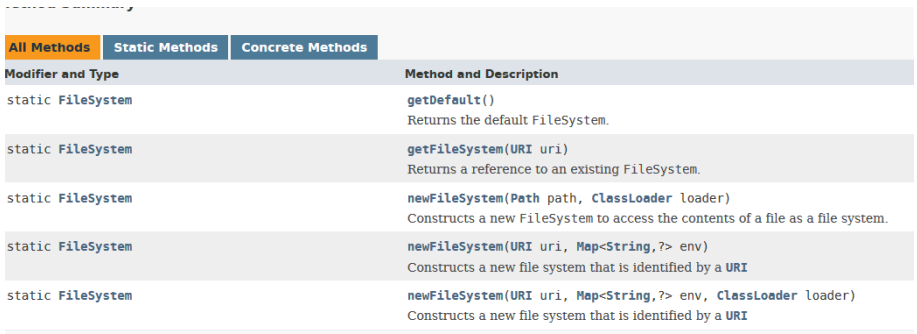
2. Librerias de I/O: I/O, **NIO** y NIO.2.

Clases más destacadas:

FileSystem. Interfaz para interacturar con sistema de archivos. En cada sistema operativo la implementación es diferente.

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

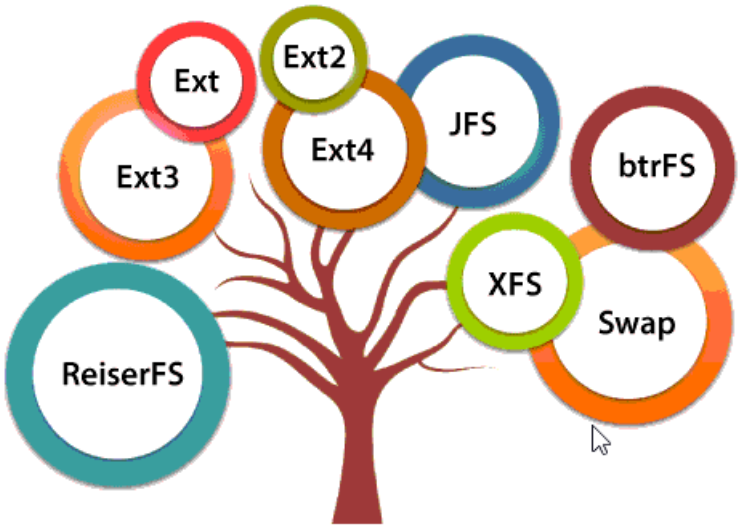
2. Librerias de I/O: I/O, **NIO** y NIO.2.

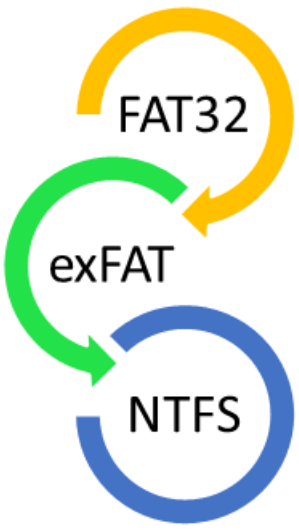
```
public static void main(String[] args) {
        FileSystem sf = FileSystems.getDefault();
        System.out.println("Es de solo lectura?" + sf.isReadOnly());
        System.out.println("Está abierto?" + sf.isOpen());
        System.out.println("Unidades montandas (en Windows letras)");
       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

2. Librerias de I/O: I/O, **NIO** y NIO.2.

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. Librerias de I/O: I/O, NIO y **NIO.2**.

I/O asíncrona:

- Fichero en red.
- Petición HTTP.

Fundamentos:

- Clase future.
- Callbacks.
- Conceptos similares→ Mandar a hacer una operación y continuar con la ejecución del programa, avisando de alguna forma cuando la tarea ha fininalizados.
- Ficheros: AsynchronousFileChannel.
- Flujos de red: AsynchronousSocketChannel.

https://www.youtube.com/watch?v=M0_fzAXIp50

2. Librerias de I/O: I/O, NIO y **NIO.2**.

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), insCancelled() y isDone(), forma parte del paquete java.util.concurrent.

CallBack: Se le pasa un objeto (en estructurada una función) al que se llamará cuando finalice la acción.

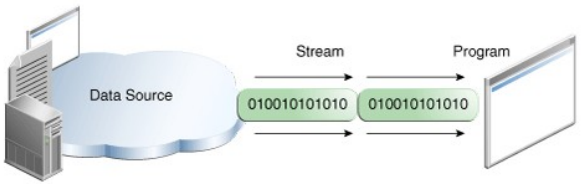
https://www.youtube.com/watch?v=zw2nUJM81FQ

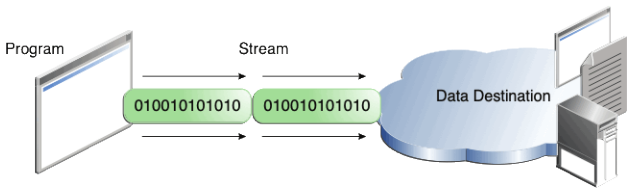
Estructuras de datos avanzadas y POO. Colecciones en Java.

3. Flujos de datos.

Abstracción de la fuente de los datos:

- Ficheros.
- Dispositivos I/O.
- Otros programas
- Datos en memoria.
- …





Similiar a streams de colecciones.

Se clasifican en binarios y de tipo texto.

Define 4 interfaces básicas a partir de las que se crean el resto de las clase:

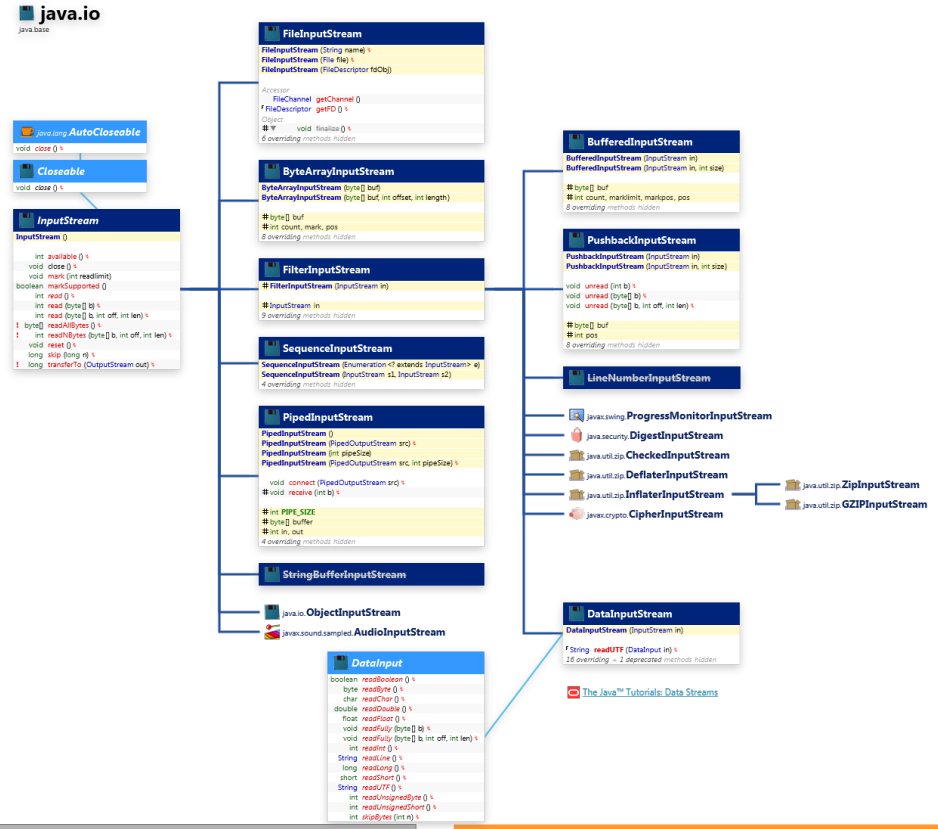
- InputStream.
- OutputStream.
- Reader.
- Writer.

Estructuras de datos avanzadas y POO. Colecciones en Java.

3. Flujos de datos.

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.



The Java™ Tutorials: Byte Streams

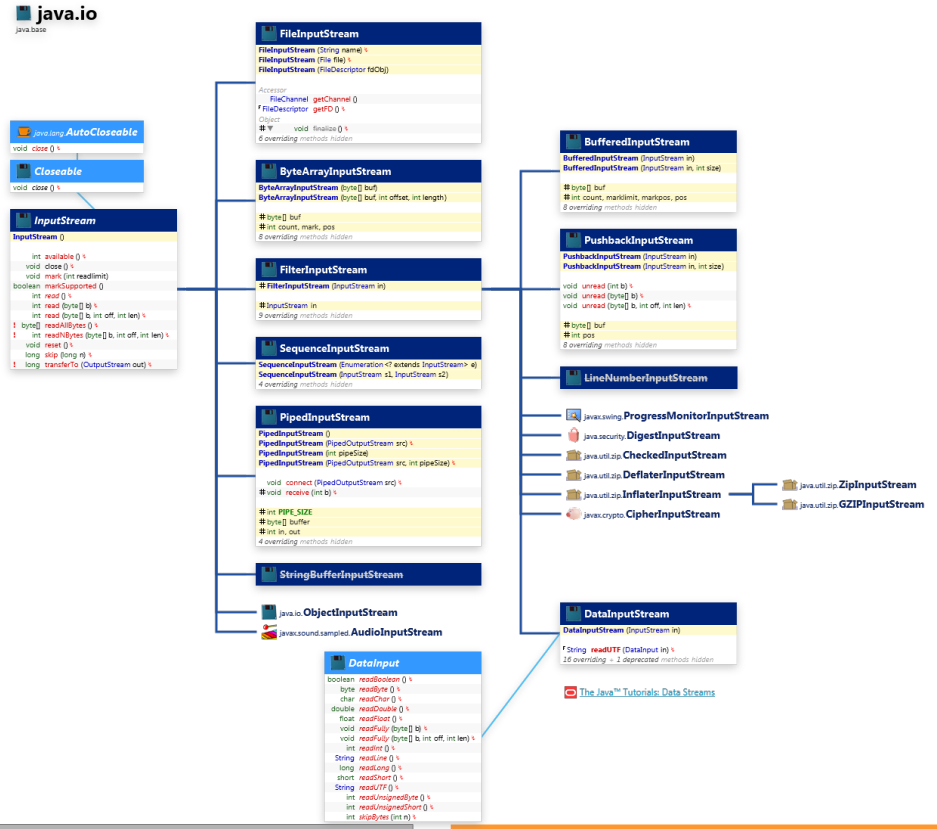
The Java™ Tutorials: The try-with-resources Statement

Estructuras de datos avanzadas y POO. Colecciones en Java.

3. Flujos de datos.

InputStream, heredan:

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



The Java™ Tutorials: Byte Streams

The Java™ Tutorials: The try-with-resources Statement

3. Flujos de datos.

InputStream, ejemplo:

```
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();}
   }
   URL url = new URL("http://africau.edu/images/default/sample.pdf");
   InputStream in = url.openStream();
   FileOutputStream fos = new FileOutputStream(new File("C:\\ejemploFiles\\salida.pdf"));
   int length = -1;
   byte[] buffer = new byte[1024];
   while ((length = in.read(buffer)) > -1) {
            fos.write(buffer, 0, length);
        }
        fos.close();
        in.close();
  }
```

```
%PDF-1.3
%âãÏÓ

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

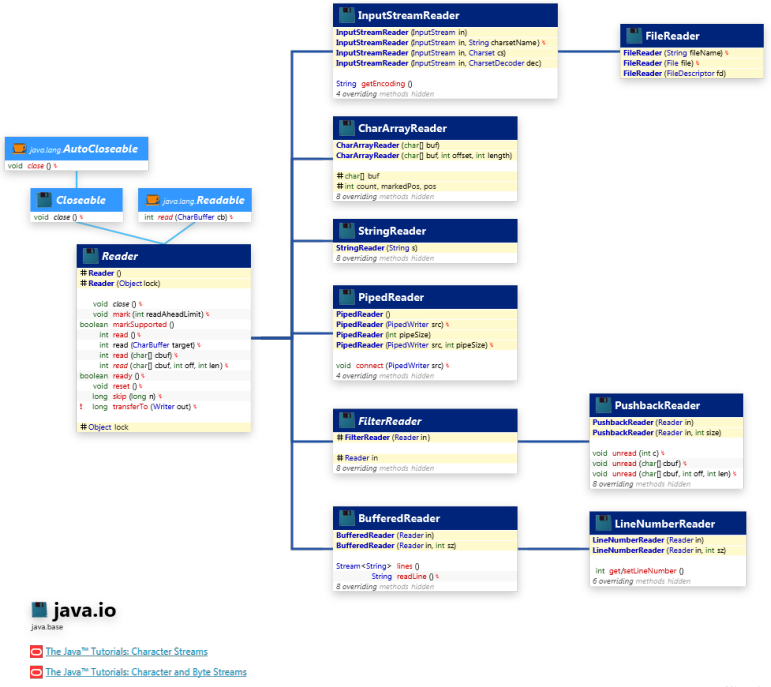
2 0 obj
<<
/Type /Outlines
/Count 0
>>
endobj
```

Estructuras de datos avanzadas y POO. Colecciones en Java.

3. Flujos de datos.

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)`

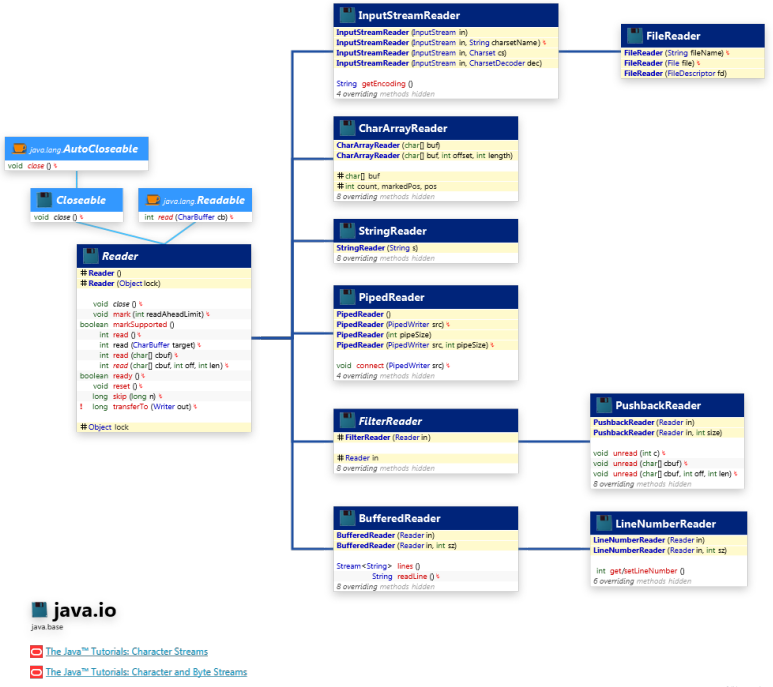


Estructuras de datos avanzadas y POO. Colecciones en Java.

3. Flujos de datos.

Reader, heredan:

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



3. Flujos de datos.

Reader, ejemplo:

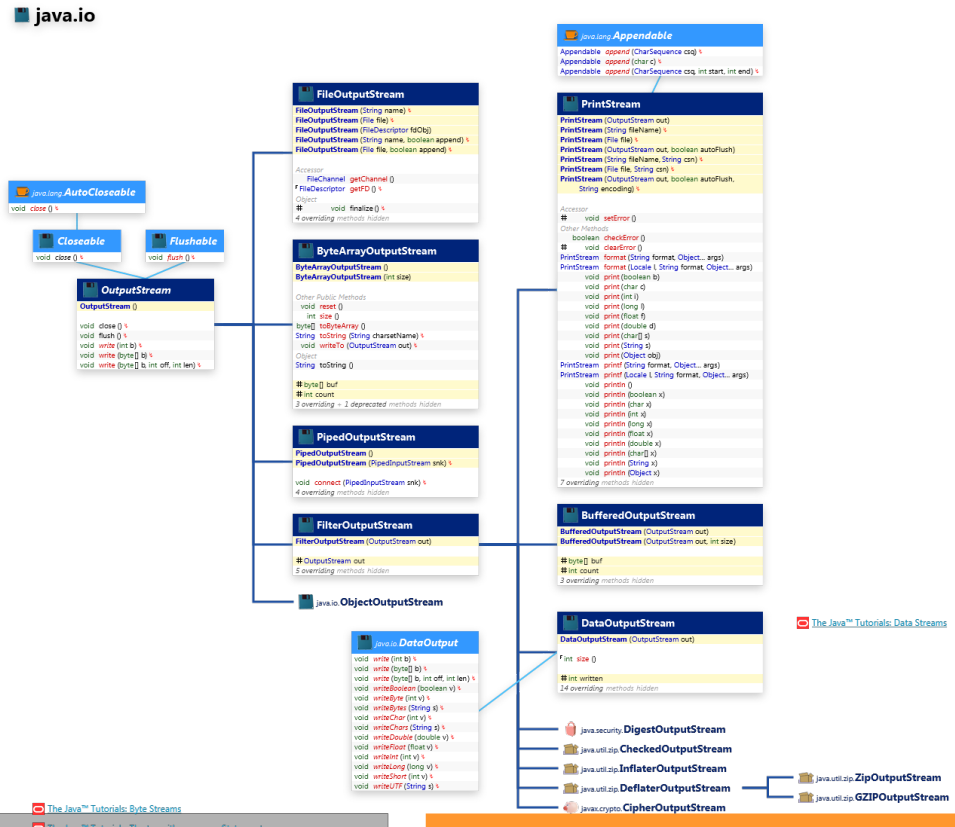
```
public static void main(String[] args) throws MalformedURLException, IOException {
        try {
            URL url = new URL("http://www.ieslaencanta.com/web/");
            InputStreamReader in = new InputStreamReader(url.openStream());
            BufferedReader br = new BufferedReader(in);
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

Estructuras de datos avanzadas y POO. Colecciones en Java.

3. Flujos de datos.

OutputStream:

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

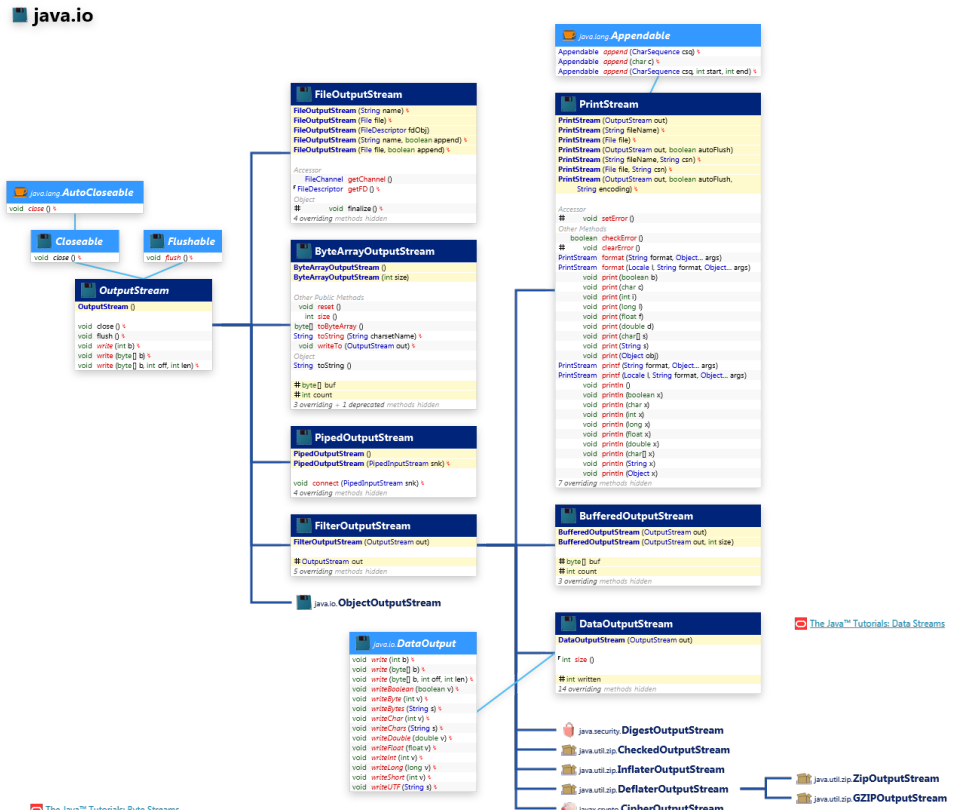


Estructuras de datos avanzadas y POO. Colecciones en Java.

3. Flujos de datos.

OutputStream, heredan:

```
FileOutputStream
ByteArrayOutputStream
PipeOutputStream
FilterOutputStream.
  PrintStream
  BufferedOutputStream
  DataOutputStream
ObjectOutputStream
```

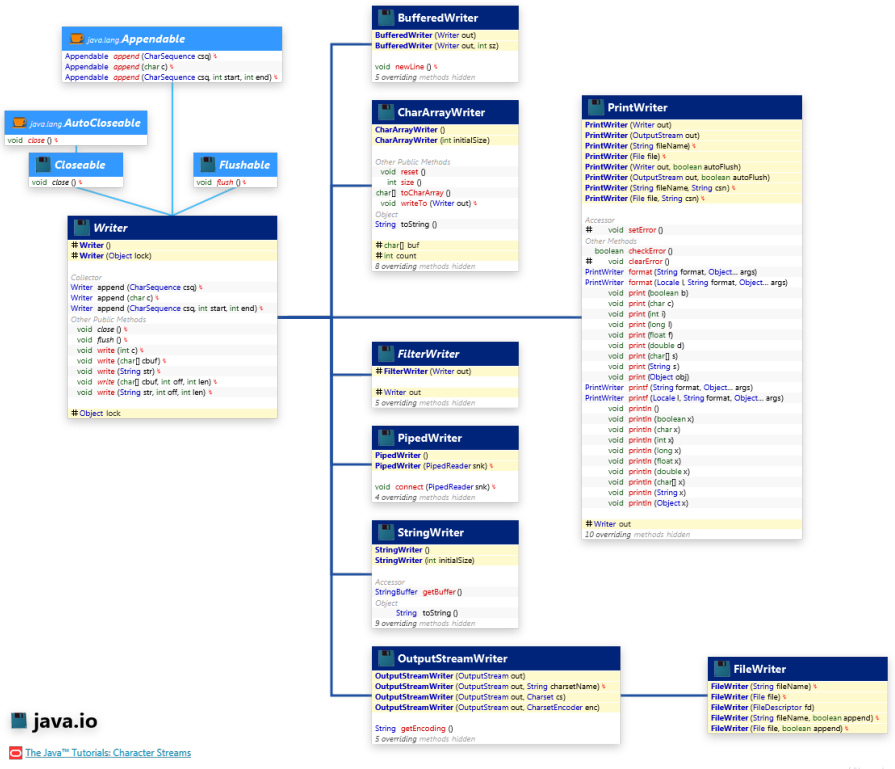


Estructuras de datos avanzadas y POO. Colecciones en Java.

3. Flujos de datos.

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)`

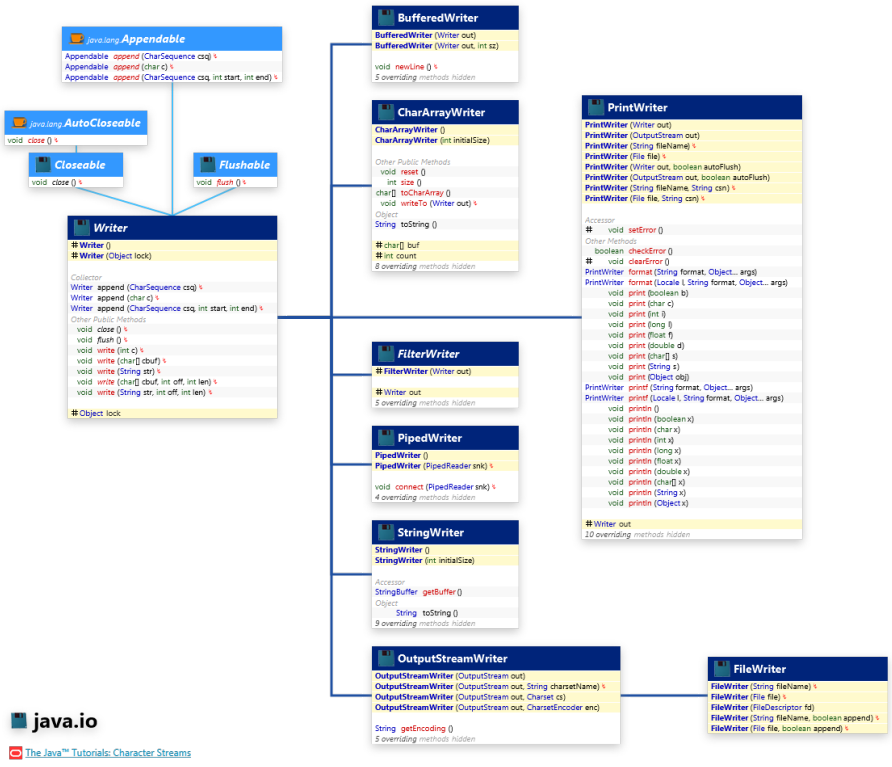


Estructuras de datos avanzadas y POO. Colecciones en Java.

3. Flujos de datos.

Writer, heredan:

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



3. Flujos de datos.

Writer, ejemplo quiniela:

```
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(   );
            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();
        }
    }
```

Estructuras de datos avanzadas y POO. Colecciones en Java.

3. Flujos de datos.

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

4. I/O estándar.

Todo proceso 3 ficheros abiertos por defecto: entrada, salida, error, en Java se tratan como otro flujo más.

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

Estructuras de datos avanzadas y POO. Colecciones en Java.

5. Almacenamiento de objetos en ficheros.

Serialización→ Transformar un objeto en una secuencia de bytes.

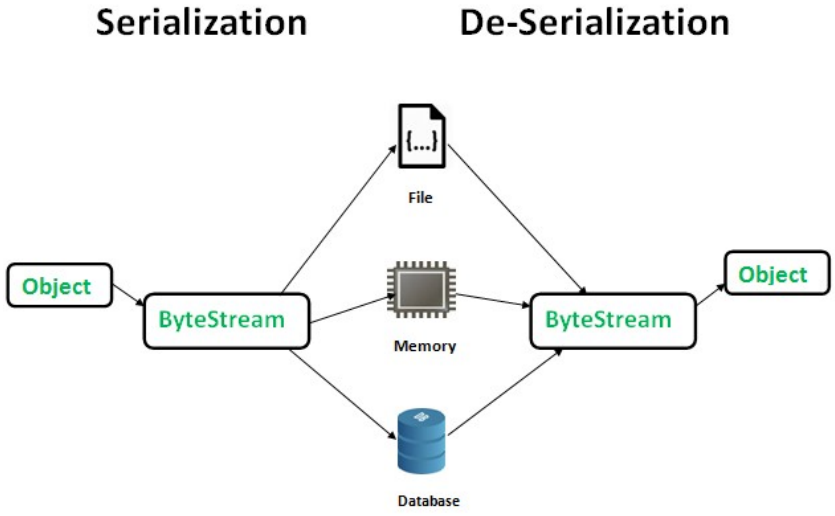
Guardarse en un fichero.

En base de datos (no es común).

Enviarlo por red.

Persistencia → Guardar los objetos de forma permanente.

Serializable→ Interfaz que se ha de implementar para que un objeto sea serializable y se pueda hacer persistente.



5. Almacenamiento de objetos en ficheros.

Serializable:

- Está vacia, no tiene ningún método.
- Sirver para indicar que la clase que la implementa se puede serializar
- Es necesario definirla para evitar problemas.
- **Todos los tipos básicos Java son serializables, así como los array y las cadenas.**
- **Si una clase tiene atributos que son a su vez clases, estas han de implementar serializable.**

```
public interface Serializable{
}
```

5. Almacenamiento de objetos en ficheros.

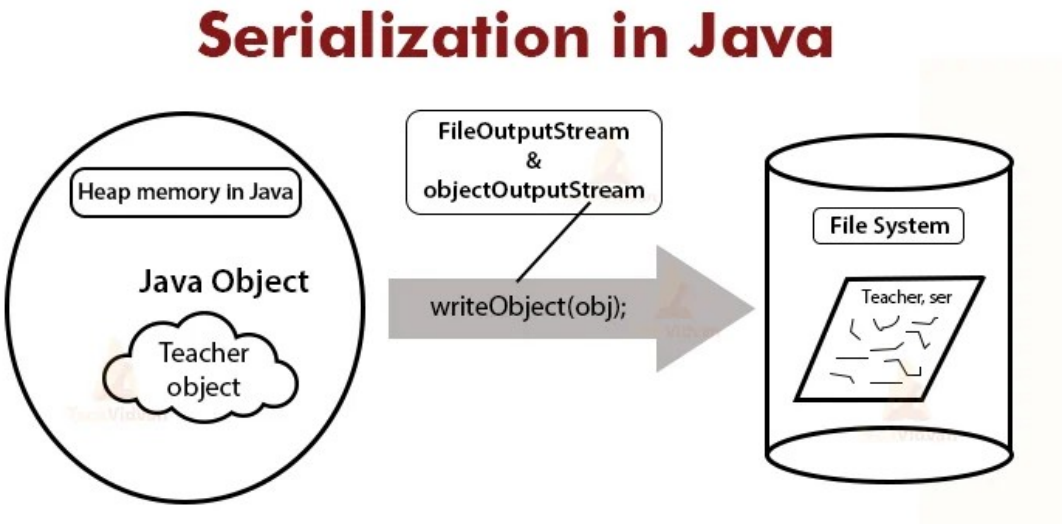
Persistencia de objetos en ficheros (escritura):

- Fichero binario.
- ObjectOutputStream que hereda de OutputStream.
- Se construye pasandole un FileOutputStream, asociado a un fichero o un socket.
  - `ObjectOutputStream(OutputStream nombre);`
  - `FileOutputStream fos = new FileOutputStream ("/ficheros/personas.dat");`
    `ObjectOutputStream salida = new ObjectOutputStream (fos);`
  - `socket1 = new Socket(InetAddress.getLocalHost(), portNumber);`
    `ObjectOutputStream oos = new ObjectOutputStream(socket1.getOutputStream());`
- Método writeObject(Object o).

5. Almacenamiento de objetos en ficheros.

Ejemplo:

```
FileOutputStream fos = null;
ObjectOutputStream salida = null;
Persona p;
fos = new FileOutputStream("/ficheros/personas.dat");
salida = new ObjectOutputStream(fos);
p = new Persona("12345678A","Lucas González", 30);
salida.writeObject(p);
p = new Persona("98765432B","Anacleto Jiménez", 28);
salida.writeObject(p);
p = new Persona("78234212Z","María Zapata", 35);
salida.writeObject(p);
```



5. Almacenamiento de objetos en ficheros.

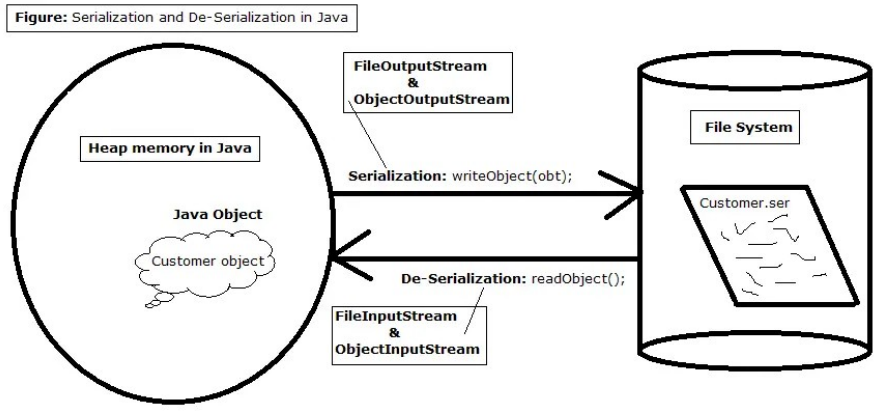
Persistencia de objetos en ficheros (lectura):

- Fichero binario.
- ObjectInputStream que hereda de InputStream.
- Se construye pasandole un FileInputStream, asociado a un fichero o un socket.
  - `ObjectInputStream(InputStream nombre);`
  - ```
    FileInputStream fis = new FileInputStream ("/ficheros/personas.dat");
    ObjectInputStream entrada = new ObjectInputStream (fis);
    ```
- Método readObject().
    ```

5. Almacenamiento de objetos en ficheros.

Ejemplo:

```
Persona p;
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());
```



6. Tratamiento de documentos XML.

Metalenguaje.

Estándar intercambio de información estructurada.

Uso diverso: Base de datos, editores de texto, hojas de cálculo, imágenes vectoriales….

Estándares de parseo (Transformar una entrada de texto en una estructura de datos (usualmente un árbol) que es apropiada para ser procesada), dos filosofías:

- DOM. En memoria, documentos pequeños y procesamiento intensivo. Se modifica la estructura.
- SAX. Documentos grandes, solo se procesa una vez o por partes. No se modifica la estructura.

6. Tratamiento de documentos XML.

Ejemplo DOM:

```
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");
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());
```

6. Tratamiento de documentos XML.

Librerías en Java.

Variedad, las más destacadas:

- **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. 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.

6. Tratamiento de documentos XML.Serialización en XML.

Librerías en Java: Xerces2, Xstream o JAXB.

Reglas transformación de objecto a XML como a la inversa.

Ejemplo JAXB para JEE(Jakarta):

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

```
Libro libro= new Libro("Odisea 2001",400);
JAXBContext contexto = JAXBContext.newInstance(
libro.getClass() );
Marshaller marshaller = contexto.createMarshaller();
marshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTP
UT,
Boolean.TRUE);
marshaller.marshal(libro, System.out);
```

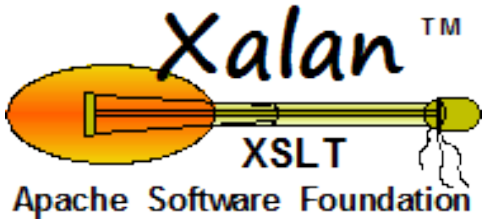
6. Tratamiento de documentos XML. Transformación.

Lenguajas basados en XSL (XSLT, Xpath…)

A otro XML, JSON, HTML….

Uso mayoritario libreriala Xalan:

XML→ Xalan+XSL→ HTML,JSON...




```
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);
```

Estructuras de datos avanzadas y POO. Colecciones en Java.

7. JSON.

- Estándar actual para manejo, intercambio de información o consumo de servicios entre otros.
- Formato para el intercambio de datos **liviano** basado en textos**.**
- Representa: **Objetos y matrices.**
  - **Objecto: Colección no ordenada de cero o más pares de nombre/valor.**
  - **Matriz: Secuencia ordenada de cero o más valores.**

```
{
    "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"
       }
    ]
}
```

Estructuras de datos avanzadas y POO. Colecciones en Java.

7. JSON.

API de Java para procesamiento JSON (JSR 353)

Analizar, generar, transformar y consultar.

Dos filosofías:

Modelo de objetos: Crea en memoria estructura árbol, modificar y acceso aleatorio.

Streams. Analizar y generar con flujo de datos.

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

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.

Estructuras de datos avanzadas y POO. Colecciones en Java.

7. JSON.

Ejemplo Object:

```
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"));
 ystem.out.println("Phone  : ");
JsonArray phoneNumbersArray = personObject.getJsonArray("phoneNumbers");
for (JsonValue jsonValue : phoneNumbersArray) {
    System.out.println(jsonValue.toString());}
```

Estructuras de datos avanzadas y POO. Colecciones en Java.

7. JSON.

Ejemplo Stream:

```
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;
             }
         }
     }
 }
```

Estructuras de datos avanzadas y POO. Colecciones en Java.

7. JSON.

Librería GSON de Google. Similiar a XML.

https://github.com/google/gson.

```
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);
    }
}
```