Introducción

[CLASIFICACIÓN DE FICHEROS 2](#_Toc435544651)

[FLUJOS DE DATOS 2](#_Toc435544652)

[ACCESO SECUENCIAL TEXTO 2](#_Toc435544653)

[ACCESO SECUENCIAL BYTES 2](#_Toc435544654)

[ACCESO ALEATORIO 2](#_Toc435544655)

[SINTAXIS 2](#_Toc435544656)

[EJEMPLO DE USO DE FLUJOS 3](#_Toc435544657)

[ARCHIVOS XML 3](#_Toc435544658)

[APLICACIONES EJECUTABLES Y PROCESOS 4](#_Toc435544659)

[Aplicación 4](#_Toc435544660)

[Programa 4](#_Toc435544661)

[Programador 4](#_Toc435544662)

[Ejecutable 5](#_Toc435544663)

[Clasificación 5](#_Toc435544664)

[DOM (Document Object Model) 6](#_Toc435544665)

[Introducción 6](#_Toc435544666)

[DOM y Java 6](#_Toc435544667)

[ACCESO A FICHEROS XML CON SAX 10](#_Toc435544668)

[SAX (Simple API for XML) 10](#_Toc435544669)

## CLASIFICACIÓN DE FICHEROS

* Según su contenido
  + Binario o de byte
  + Caracteres irreconocibles
  + Representan información (Audio, Imagen, Vídeo)
  + Aplicaciones concretas: reconocer la información
* De caracteres o de texto
  + Caracteres reconocibles
  + Representan texto
  + Editor de texto
* Según su modo de acceso
  + Secuencial: Leer todo hasta llegar a donde quieres.
  + Aleatorio: Ir directamente donde quieres.

## FLUJOS DE DATOS

Acceder a un fichero = = = = => Flujo de información

Programa < = = = = = = = > Fichero destino / Origen información

El flujo es un objeto intermediario que sirve de intermediario entre el programa y el origen o destino de la información.

## ACCESO SECUENCIAL TEXTO

* Leer un fichero de texto
  + FileReader: Implementa un flujo de caracteres que los lee desde un fichero de texto. Con ello obtenemos un flujo de datos.
* Escribir un fichero de texto
  + FileWriter: Escribe un flujo de caracteres. Escribimos datos en el archivo que elijamos.

## ACCESO SECUENCIAL BYTES

* FileInputStream: Lee bytes.
* FileOutputStream: Escribe bytes.

## ACCESO ALEATORIO

* RandomAccessFile: Para acceso aleatorio a cualquier posición.

## SINTAXIS

**FileWriter**

FileWriter fileWriter = new FileWriter(String ruta, boolean añadir\_al\_final);

FileWriter fileWriter = new FileWriter(String ruta);

fileWriter.write([String[], String]);

**FileReader**

FileReader fileReader = new FileReader(String ruta);

FileReader fileReader = new FileReader(File fichero);

fileReader.read(); -> Array de caracteres

**FileOutputStream**

Constructor igual que FileWriter.

Flujo para ficheros binaries, escribe a través de arrays de caracteres.

**FileInputStream**

Constructor igual que FileReader.

Lee bytes secuencialmente.

**RandomAccessFile**

Acceder directamente a cualquier posición del fichero.

RandomAccessFile randomReader = new RandomAccessFile(File fichero, String modo[“r”, “rw”]);

## EJEMPLO DE USO DE FLUJOS

1º - Abrir el archivo

2º - Usar el fichero

3º - Gestionar excepciones

4º - Cerrar el archivo y destruirlo

## ARCHIVOS XML

* Es un **lenguaje de marcas extendido**
* Información de forma **neutra**: No depende ni del lenguaje de programación ni del sistema operativo
* Muy útil para desarrollar aplicaciones de software
  + Servidores web
* Un documento XML tiene dos afecciones
  + **A "bajo nivel"**
    - Fichero de texto: Accedemos con cualquier librería de acceso (FileWriter, FileReader..)
  + **A "alto nivel"**
    - Librerías específicas: Manipular más potente, más flexible y eficiente. Herramientas específicas o librerías.
      * Se reduce el tiempo de desarrollo de las aplicaciones
      * Se optimizan los accesos
      * Permite manejar los documentos de una forma simple y no cargar innecesariamente al sistema
* **La gran ventaja del XML** es que permite almacenar datos y luego acceder a ello fácilmente
* **Analizadores sintácticos o PARSERS** son las herramientas que se encargan de leer el lenguaje XML y comprueban si es válido o no
  + Es un módulo, una biblioteca o un programa
  + Herramientas que validan los documentos XML
    - Comprobar que el documento esté bien formado (well formed)
    - Que sea válido respecto al esquema XML
      * Ejemplo: JAXB
  + Herramientas que no validan los documentos XML
    - Comprueban que el documento esté bien formado (well formed)
    - No necesitan un esquema XML para comprobar que sea válido
      * Ejemplos: DOM y SAX

## APLICACIONES EJECUTABLES Y PROCESOS

## Aplicación

* Tipo de programa informático
* Herramienta para resolver de forma automática una necesidad específica del usuario
  + Escribir textos
  + Grabar vídeos
  + Hacer llamadas telefónicas

## Programa

* Conjunto de instrucciones secuenciales (están ordenadas) que ejecutadas desde un ordenador y que cumplen una función: Realizar una tarea y ayudar a realizarla.

## Programador

1. Escribir código fuente
2. Análisis
3. Pruebas
4. Obtener un código interpretado o binario (compilador)
5. Se guarda. Fichero ejecutable.

## Ejecutable

* Fichero compilado y preparado para ejecutar
* Un proceso es un programa en ejecución
  + Un proceso existe mientras se ejecuta una aplicación
    - Ejecución de una aplicación -> Arranque de varios programas
  + Compuestos de más ejecutables y librerías
* Uno o varios procesos
* ¿Cómo podemos identificar un fichero ejecutable?
  + En Windows por la extensión (.exe)
  + Ficheros con permiso de ejecución activado

### Clasificación

Según el tipo de código que contengan.

* **Binario**: Instrucciones directamente ejecutadas por la CPU del ordenador. Se obtiene al compilar el código fuente. Se ejecutará en equipos con plataforma compatible con la que ha sido compilada.
* **Interpretado**: Formado por códigos de operación que son interpretados por un intérprete (JWM – Joe’s Windows Manager). Los traduce al lenguaje máquina. Ejemplo scripts: Contiene instrucciones que ejecutará el intérprete una detrás de otra. No son compilados, se pueden ver con un editor de texto plano.
* **Librerías**: Son funciones que dan modularidad y reusabilidad a los programas. Su contenido es código ejecutable o interpretable.

# DOM (Document Object Model)

## Introducción

La tecnología DOM nos permite analizar y manipular dinámicamente el contenido, el estilo y la estructura del documento.

##### ¿Cómo trabajar con un documento XML?

Almacena en memoria en memoria en forma de árbol (nodos padres, hijos…). Cada estructura de datos se transforma en nodos, tenemos una organización jerárquica.

Los métodos DOM permiten:

* Recorrer los distintos nodos del árbol
* Analizar el tipo de partícula al que pertenecen

En función del tipo de nodo se ofrece una serie de funcionalidades u otras para poder trabajar con la información que contiene.



## DOM y Java

El principal inconveniente es que se el documento XML se guarda en memoria RAM o memoria principal, si es muy grande podría colapsar y hacerlo intratable.

##### JAXP (Java API for XML Processing)

Los parsers nos garantizan una interoperabilidad de Java.

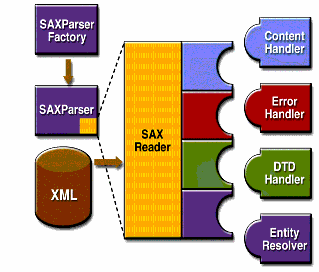
Estructura de árbol de nodos.

Buscamos parsear estructura jerárquica.

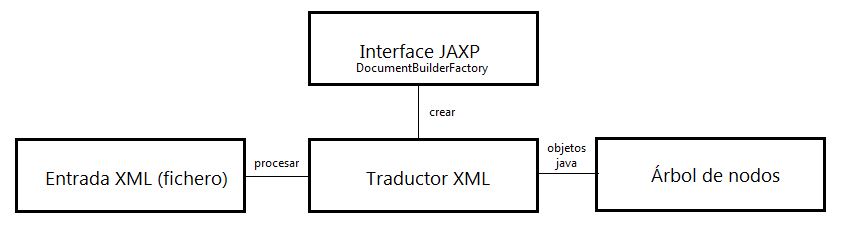
Parsers para DOM: XT, Xalan, Xerces.

Dentro de la librería Xerces usamos los siguientes paquetes:

* java.xml.parsers
  + javax.xml.parsers.DocumentBuilder
  + javax.xml.parsers.DocumentBuilderFactory
* org.w3c.clean: Nos representa el modelo DOM según la W3C.
* Para generar un fichero XML desde un árbol DOM javax.xml.transform especifican fuente y resultado.



##### Esquema que relaciona JAXP con el acceso a DOM



1. Desde la interfaz JAXP se crea un DocumentBuilderFactory
2. Se crea un DocumentBuilder que permite cargar en él la estructura de árbol DOM desde un fichero XML

##### javax.xml.parsers.DocumentBuilderFactory

Dentro de la clase tenemos métodos para indicar lo que nos interesa del fichero XML para ser incluida en el árbol DOM o si se desea validar el XML con respecto al esquema.

Métodos:

* **setIgnoringComments(boolean ignore)**: Ignora los comentarios de un fichero XML.
* **setIgnoringElementWhitespace(boolean ignore)**: Ignora los espacios en blanco.
* **setNamespaceAware(boolean aware)**: Interpretar el documento usando el espacio de nombres.
* **setValidating(boolean validate)**: Validar el documento XML según el esquema.

##### org.w3c.dom.Document

Métodos:

* **getFirstChild() / getNextSibling()**: Permiten obtener los nodos uno a uno: los nodos descendientes (hijos) y sus adyacentes (hermanos).
* **getNodeType()**: Devuelve una constante para distinguir entre los distintos tipos de nodos (elementos ELEMENT\_NODE, texto TEXT\_NODE). Útil para recorrer el árbol ya que nos permite ignorar los tipos de nodo que no nos interesan.
* **getAttributes()**: Devuelve un objeto NamedNodeMap (lista con sus atributos, si el nodo es de tipo elemento).
* **getNodeName / getNodeValue()**: Devuelve el nombre y el valor de un nodo. Útil para hacer búsquedas selectivas de un nodo concreto.

(\*) Error típico: Creer que getNodeValue() aplicado a un nodo de tipo elemento devuelve texto, cuando devolvería NULL o un DomException.

##### Abrir DOM desde Java

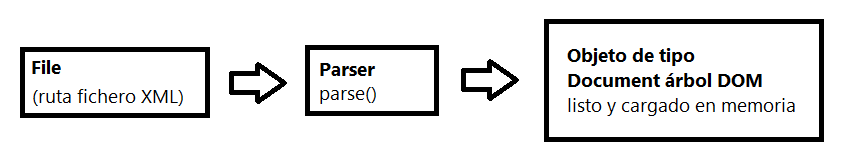
Para abrir un documento XML y crear un árbol DOM

* DocumentBuilderFactory (javax.xml.parsers)
* DocumentBuilder (javax.xml.parsers)
* Document (org.w3c.dom)

###### ¿Cómo se va a abrir?

1. Se crea Factory
2. Se prepara el parser para interpretar el fichero XML

Hay algunas cosas que no se tienen en cuenta para generar el DOM, los espacios en blanco y los comentarios.



##### Recorrer un árbol DOM

Se usan las siguientes clases

* Node (org.w3c.dom)
* NodeLast (org.w3c.dom)

Utilidad: Nos sirve para comprobar el tipo de nodo que está tratando, ya que un DOM tiene muchos tipos de nodo que no siempre tiene información que merezca la pena explorar.

##### Modificar y serializar

Además de recorrer en modo “sólo lectura” nuestro árbol DOM también se puede **modificar** y guardar en un fichero y más adelante veremos cómo crearlo.

* Se crea un añade elemento al árbol DOM y éste se guarda en un documento XML.
* Hay que tener en cuenta que hay que crear nodos de texto que desciendan de los nodos elementos para proporcionar información sobre los valores.
* Una vez modificado nuestro árbol DOM se guarda, se lleva a fichero, con esto se consigue la persistencia de los datos.

Para **serializar** (marshalling) el árbol se convierte en una cadena de caracteres o bytes. Es el proceso de convertir el estado de un objeto en un formato que se pueda almacenar. Ejemplo: Llevar los objetos que componen un árbol DOM a un fichero XML.

El un-marshalling pasa el contenido de un fichero a una estructura de objetos.

###### La clase XMLSerializer

Serializa el árbol DOM y lo lleva a un fichero XML.

###### La clase OutputFormat

Permite asignar diferentes propiedades del formato del fichero resultado. Ejemplo: Indentado, los elementos hijos aparecen con mayor tabulación.

###### ¿Qué necesitamos para serializar un árbol DOM?

* Un objeto File para presentar el fichero resultado
* La clase OutputFormat que nos permite indicar las pautas del formato para la salida
* Un objeto XMLSerializer que se construye con el File de salida y el formato definido por el OutputFormat
* Un método serialize() de XMLSerializer recibe como parámetro el documento que quieres llevar a fichero (el árbol DOM) y lo escribe

Los programas Java que utilizan DOM necesitan las siguientes interfaces:

* **Document** Permite crear nuevos nodos en el documento
* **Element** Expone las propiedades y métodos para manipular los elementos y sus atributos
* **Node** Representa cualquier nodo del documento
* **NodeList** Contiene una lista con los nodos hijos de cada nodo
* **Attr** Nos permite acceder a los atributos de un nodo
* **Text** Son los datos carácter de un nodo
* **CharacterData** Proporciona atributos y métodos para manipular los datos de los caracteres

# ACCESO A FICHEROS XML CON SAX

### SAX (Simple API for XML)

Conjunto de clases o interfaces que nos ofrecen una herramienta muy útil para el procesamiento de documentos XML.

Permite analizar los datos de forma secuencial. No hace falta cargar todo el fichero en memoria como pasa con DOM. Esto implica menos consumo de memoria aunque tengan un gran tamaño. La desventaja es que no te da una visión global del documento a analizar.

La programación de SAX es más compleja que la de DOM ya que es una API que está entera escrita en Java (está incluida dentro del JRE que nos permite crear nuestro parser XML).

Normalmente usaremos SAX cuando tengamos la información de los XML de una forma clara, bien estructurada y que no necesite modificaciones.

Pasos que sigue SAX

1. Se le dice al parser el fichero que tiene que ser leído
2. Traducimos el documento XML a una serie de eventos
3. Los eventos obtenidos pueden controlarse mediante métodos de control, se llaman callbacks
4. Para implementar los callbacks hay que implementar la interfaz ContentHandler

Veamos el proceso que sigue:

* SAX abre el fichero XML y coloca el puntero al inicio
* Según va leyendo el fichero el puntero va avanzando secuencialmente
* Cuando SAX encuentre un elemento propio del XML lanza un evento, veamos a que pueden deberse esos eventos:
  + Que SAX detecte el inicio del documento XML
  + Que SAX detecte el final del documento XML
  + Que SAX detecte una etiqueta de comienzo de un elemento, ejemplo <alumno>
  + Que SAX detecte una etiqueta de final de un elemento, por ejemplo </alumno>
  + Que SAX detecte un atributo
  + Que SAX detecte un Namespace
  + Que SAX detecte una cadena de caracteres
  + Que SAX detecte un error
* Cuando SAX devuelve que ha detectado un evento este evento podrá ser manejado por la clase DefaultHandler
* Cuando SAX detecta un evento de error o de final de documento se termina el recorrido

La lectura de un documento XML produce eventos, esto ocasiona la llamada a métodos. Algunos eventos son, encontrar:

* La etiqueta de inicio/fin del documento startDocument(), endDocument()
* La etiqueta de inicio/fin de un elemento startElement(), endElement()
* Los caracteres entre etiquetas characters(). Ejemplo:

<?xml version="1.0" encoding="UTF-8"?> startDocument()

**<clase> startElement()**

**<alumno> startElement()**

**<nombre>**Silvia**</nombre> startElement() -> characters() -> endElement()**

**<edad>**21**</edad> startElement() -> characters() -> endElement()**

**<lugar>**Ágreda**</lugar> startElement() -> characters() -> endElement()**

**</alumno> endElement()**

**</clase> endElement()**

endDocument()

Para abrir un documento XML se utilizan las clases:

* javax.xml.parsers.**SAXParserFactory**
* javax.xml.parsers.**SAXParser**
* org.xml.saxhelpers.**DefaultHandler**

Para la preparación de SAX hay que inicializar algunas variables y éstas serán utilizadas cuando se inicie el proceso de recorrido del fichero XML.

Veamos algunas de las variables:

* Un objeto parser
* Una instancia de clase que extienda de DefaultHandler

¿Cómo construimos un ejemplo para tratar eventos?

1. Incluir las clases e interfaces de SAX
   1. import org.xml.sax.Attributes
   2. import org.xml.sax.InputSource
   3. import org.xml.sax.SAXException
   4. import org.xml.sax.DefaultHandler
   5. import org.xml.sax.XMLReaderFactory
2. Se crea un objeto procesador, es decir, un XMLReader
3. Se indica al XMLReader, que objetos poseen los métodos que tratarán los eventos. Estos objetos suelen ser implementaciones de las siguientes interfaces:
   1. **ContentHandler** Recibe las notificaciones de los eventos que ocurren en el documento (startDocument, endDocument…)
   2. **DTDHandler** Recoge eventos relacionados con la DTD (Document Type Definition), descripción de la sintaxis y la estructura de un documento XML. Se emplea generalmente para determinar la estructura de un objeto por etiquetas
   3. **ErrorHandler** Define los métodos de tratamiento de errores
   4. **EntityResolver** Sus métodos se llaman cada vez que haya una referencia
   5. **DefaultHandler** Provee una implementación por defecto para todos sus métodos, el programador define los métodos que se utilizarán por el programa

Para indicar al procesador XML los objetos que realizarán el tratamiento utilizamos los siguientes métodos (incluidos dentro de los objetos XMLReader):

* setContentHandler()
* setDTDHandler()
* setEntityHandler()
* setErrorHandler()

1. Se define el fichero XML que se va a leer mediante un objeto InputSource
2. Se procesa el documento XML con un método parser

Para recorrer un documento XML una vez inicializado el parser, lo único que tenemos que hacer es lanzar el parser. Antes hay que definir una clase que extiende de DefaultHandler.

Ejemplo:

**public** **class** **GestionDeContenido** **extends** DefaultHandler {

@Override

**public** void startElement(String uri, String localName, String qName, Attributes attributes) {

*//Hacer algo*

}

@Override

**public** void endElement(String uri, String localName, String qName) **throws** SAXException {

*//Hacer algo*

}

@Override

**public** void characters(char ch[], int start, int length) **throws** SAXException {

*//Hacer algo*

}

}

Estos métodos se invocan al recorrer el documento XML y encontramos eventos de inicio de elemento de final de elemento o cadena de caracteres. Veamos que realiza cada método:

* **startElement()** Cuando se detecta con SAX un evento de inicio de elemento, se invoca este método que comprueba de qué tipo de elemento se trata:
  1. Si es <comunidad> saca el valor de su atributo y lo añade a una cadena (cadena\_final) que tendrá toda la salida después de recorrer todo el documento.
  2. Si es <idioma> a la cadena final se le añade el texto “Los idiomas son …”
  3. Si es <provincia> a la cadena final se le añade el texto “Las provincias son …”
  4. Si es otro tipo de elemento no hace nada
* **endElement()** Cuando se detecta con SAX un evento final de elemento se invoca a este método. El método comprueba si es el final del elemento, si es así se concatena el elemento de salto de línea
* **characters()** Cuando se detecta con SAX un evento de cadena de caracteres se invoca a este método. El método lee la cadena y añade a la cadena final cada uno de los distintos caracteres detectados

# CONVERSIÓN DE FICHEROS XML A OTRO FORMATO

### XSL (Extensible Stylesheet Language)

Son el conjunto de recomendaciones de la W3C (World Wide Web Consorcium) para expresar hojas de estilo en lenguaje XML.

Una hoja de estilo XSL describe el proceso de presentación a través de un pequeño conjunto de elementos XML. Esta hoja puede contener elementos de reglas de estilo.

A partir de un documento XML que contiene los datos y otro XSL que contiene la presentación de estos datos -> genera un fichero HTML usando el lenguaje Java.

Para realizar esta transformación necesitamos un objeto Transformer creando una instancia de un TransformerBuilder y aplicando el método newTransformer() al XML que vamos a utilizar para aplicar la transformación del fichero de datos XML.

La transformación se consigue llamando al método transform() pasándole los datos (el fichero XML) y el stream de salida.

Transformer transformer = new TransformerFactory.newInstance().newTransformer(estilos);

transformer.transformer(datos, resultado);

# Excepciones: Detección y tratamiento

Excepción es un evento que ocurre durante la ejecución del programa que interrumpe el flujo normal de las sentencias. Cuando no es capturada por el programa se captura por el gestor de excepciones por defecto que retorna un mensaje y detiene el programa.

Cuando la ejecución del programa produce una excepción visualiza un mensaje indicando el error.

Cuando dicho error ocurre dentro de un método Java, el método crea el objeto Exception y lo manda fuera. En Java el método que detecta un error no es capaz de manejarlo, un método así lanzará una excepción.

Las excepciones en Java son objetos de clases derivadas de la clase base Exception derivada a su vez de la clase Throwable.

### Capturar excepciones

Para capturar excepciones se utiliza un bloque try-catch. Se encierra en un bloque try el código que pueda generar una excepción y después añadir uno o más bloques catch.

Cada bloque catch especifica el tipo de excepción que puede atrapar y contiene un manejador de excepciones.

Después del último bloque catch puede aparecer un bloque finally (opcional) que siempre se ejecuta haya ocurrido o no la excepción. Se utiliza para cerrar ficheros o liberar otros recursos del sistema después de que ocurran las excepciones.

Para capturar cualquier excepción usamos la clase básica Exception en el caso de ponerla se sitúa al final de la lista de manejadores para evitar que los manejadores que vienen después queden informados.

Veamos algunos métodos de la clase Throwable:

* **exception.getMessage()**: Devuelve la cadena de caracteres del error
* **exception.getLocalizedMessage()**: Crea una descripción local del objeto
* **exception.toString()**: Devuelve una breve descripción del objeto
* **exception.printStackTrace(), excetion.printStackTrace(PrintStream), exception.printStackTrace(PrintWriter):** Imprime el objeto y la traza de pila de llamadas lanzada.

Una sentencia try puede estar dentro de un bloque try. Si la sentencia try de dentro no tiene manejador catch, se busca el manejador en las sentencias try más externas.

### Especificar excepciones

La palabra clave que usamos para especificar excepciones es throws seguida de la lista de todos los tipos de excepciones.

Para los métodos que pueden lanzar excepciones, deben saber cuáles son esas excepciones en su declaración, normalmente se compila el programa.

Si se produce una excepción que no está declarada, esta excepción debe ser capturada por un bloque try-catch o declarada para ser lanzada mediante throws.