

INDICE

<u>1.7.</u>	<u>TRABAJO CON FICHEROS XML</u>	<u>1</u>
<u>1.7.1.</u>	<u>ACCESO A FICHEROS XML CON DOM</u>	<u>2</u>
<u>1.7.2.</u>	<u>ACCESO A FICHEROS XML CON SAX</u>	<u>8</u>
<u>1.7.3.</u>	<u>SERIALIZACIÓN DE OBJETOS A XML</u>	<u>12</u>
<u>1.8.</u>	<u>ACTIVIDADES</u>	<u>17</u>

1.7. Trabajo con Ficheros XML

XML (*eXtensible Markup Language- Lenguaje de Etiquetado Extensible*) es un metalenguaje, es decir, un lenguaje para la definición de lenguajes de marcado. Nos permite jerarquizar y estructurar la información y describir los contenidos dentro del propio documento. Los ficheros XML son ficheros de texto escritos en lenguaje XML, donde la información está organizada de forma secuencial y en orden jerárquico. Existen una serie de marcas especiales como son los símbolos menor que, < y mayor que, > que se usan para delimitar las marcas que dan la estructura al documento. Cada marca tiene un nombre y puede tener 0 más atributos. Un fichero XML sencillo tiene la siguiente estructura:

```
<?xml version="1.0"?>
<Empleados>
  <empleado>
    <id>1</id>
    <apellido>FERNANDEZ</apellido>
    <dep>10</dep>
    <salario>1000.45</salario>
  </empleado>
  <empleado>
    <id>2</id>
    <apellido>GIL</apellido>
    <dep>20</dep>
    <salario>2400.6</salario>
  </empleado>
  <empleado>
    <id>3</id>
    <apellido>LOPEZ</apellido>
    <dep>10</dep>
    <salario>3000.0</salario>
  </empleado>
</Empleados>
```

Los ficheros XML se pueden utilizar para proporcionar datos a una base de datos, o para almacenar copias de partes del contenido de la base de datos. También se utilizan para escribir ficheros de configuración de programas o en el protocolo SOAP (*Simple Object Access Protocol*), para ejecutar comandos en servidores remotos; la información enviada al servidor remoto y el resultado de la ejecución del comando se envían en ficheros XML.

Para leer los ficheros XML y acceder a su contenido y estructura, se utiliza un procesador de ML o parser. El procesador lee los documentos y proporciona acceso a su contenido y estructura. Algunos de los procesadores más empleados son: **DOM**: *Modelo de Objetos de Documento* y **SAX**: *API Simple para XML*. Son independientes del lenguaje de programación y existen versiones particulares para Java, VisualBasic, C, etc. Utilizan dos enfoques diferentes:

- ❖ **DOM**: un procesador XML que utilice este planteamiento almacena toda la estructura del documento en memoria en forma de árbol con nodos padre, nodos hijo y nodos finales (que son aquellos que no tienen descendientes). Una vez creado el árbol, se van recorriendo los diferentes nodos y se analiza a qué tipo particular pertenecen. Tiene su origen en el W3C. Este tipo de procesamiento necesita más recursos de memoria y tiempo sobre todo si los ficheros XML a procesar son bastante grandes y complejos.
- ❖ **SAX**: un procesador que utilice este planteamiento lee un fichero XML de forma secuencial y produce una secuencia de eventos (comienzo/fin del documento comienzo/fin de una etiqueta, etc.) en función de los resultados de la lectura. Cada evento invoca a un método definido por el programador. Este tipo de procesamiento prácticamente no consume memoria, pero por otra parte, impide tener una visión global del documento por el que navegar.

1.7.1. Acceso a ficheros XML con DOM

Para poder trabajar con DOM en Java necesitamos las clases e interfaces que componen el paquete **org.w3c.dom** (contenido en el JSDK) y el paquete **javax.xml.parsers** del API estándar de Java que proporciona un par de clases abstractas que toda implementación DOM para Java debe extender. Estas clases ofrecen métodos para cargar documentos desde una fuente de datos (fichero, **InputStream**, etc). Contiene dos clases fundamentales: **DocumentBuilderFactory** y **DocumentBuilder**.

DOM no define ningún mecanismo para generar un fichero XML a partir de un árbol DOM. Para eso usaremos el paquete **javax.xml.transform** que permite especificar una fuente y un resultado. La fuente y el resultado pueden ser ficheros, flujos de datos o nodos DOM entre otros.

Los programas Java que utilicen DOM necesitan estas interfaces (no se exponen todas, solo algunas de las que usaremos en los ejemplos):

- **Document**: Es un objeto del documento XML tiene un equivalente en un objeto de este tipo. Permite crear nuevos nodos en el documento.
- **Element**: Cada elemento del documento XML tiene un equivalente en un objeto de este tipo. Expone propiedades y métodos para manipular los elementos del documento y sus atributos.
- **Node**: Representa a cualquier nodo del documento.

- **NodoList**: Contiene una lista con los nodos hijos de un nodo.
- **Attr**: Permite acceder a los atributos de un nodo.
- **Text**: Son los datos carácter de un elemento.
- **CharacterData**: Representa a los datos carácter presentes en el documento. Proporciona atributos y métodos para manipular los datos de caracteres.
- **DocumentType**: Proporciona información contenida en la etiqueta **<!DOCTYPE>**.

A continuación vamos a crear un fichero XML a partir del fichero aleatorio de empleados creado en el epígrafe anterior. Lo primero que hemos de hacer es importar los paquetes necesarios:

```
import org.w3c.dom;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.parsers.dom;
import javax.xml.parsers.stream;
import java.io.*;
```

A continuación creamos una instancia de **DocumentBuilderFactory** para construir el parser, se debe encerrar entre **try-catch** porque se puede producir la excepción **ParserConfigurationException**:

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
try {
    DocumentBuilder builder = factory.newDocumentBuilder();
    .....
}
```

Creamos un documento vacío de nombre *document* con el nodo raíz de nombre Empleados y asignamos la version del XML, la interfaz **DOMImplementation** permite crear objetos **Document** con nodo raíz:

```
DOMImplementation implementation = builder.getDomImplementation();
Document document = implementation.createDocument (null, "Empleados", null);
Document.setXmlVersion("1.0"); //asignamos la versión de nuestro XML
```

El siguiente paso sería recorrer el fichero con los datos de los empleados y por cada registro crear un nodo empleado con 4 hijos (*id*, *apellido*, *dep* y *salario*). Cada nodo hijo tendrá su valor (por ejemplo: *1*, *FERNANDEZ*, *10*, *1000.45*). Para crear un elemento usamos el método **createElement(String)** llevando como parámetro el nombre que se pone entre las etiquetas menor que y mayor que. El siguiente código crea y añade el nodo **<empleado>** al documento:

```
//creamos el nodo empleado
Element raiz = document.createElement("empleado") ;
//lo pegamos a la raiz del documento
document.getDocumentElement().appendChild(raiz);
```

A continuación se añaden los hijos de ese nodo (raíz), estos se añaden en el método *CrearElemento()*:

```
//añadir ID
CrearElemento ("id", Integer.toString(id), raiz, document);
//añadir APELLIDO
CrearElemento ("apellido",apellidos.trim(), raiz, document);
//añadir DEP
CrearElemento ("dep",Integer.toString(dep), raiz, document);
//añadir SALARIO
CrearElemento ("salario", Double.toString(salario), raiz, document);
```

Como se puede ver el método recibe el nombre del nodo hijo (*id*, *apellido*, *dep* o *salario*) y sus textos o valores que tienen que estar en formato String (1, FERNANDEZ, 10, 1000.45), el nodo al que se va a añadir (*raiz*) y el documento (*document*). Para crear el nodo hijo (<id> o <apellido> o <dep> o <salario>) se escribe:

```
Element elem = document.createElement(datoEmple); //creamos un hijo
```

Para añadir su valor o su texto se usa el método *createTextNode(String)*:

```
Text text = document.createTextNode(valor); //damos valor
```

A continuación se añade el nodo hijo a la raíz (empleado) y su texto o valor al nodo hijo:

```
raiz.appendChild(elem); //pegamos el elemento hijo a la raiz
elem.appendChild(text); //pegamos el valor al elemento
```

Al final se generaría algo similar a esto por cada empleado:

```
<empleado>
<id>1</id><apellido>FERNANDEZ</apellido><dep>10</dep><Salario>1000.45</salario>
</empleado>
```

El método es el siguiente:

```
static void CrearElemento(String datoEmple, String valor,
                          Element raiz, Document document){
    Element elem = document.createElement(datoEmple);    //creamos hijo
    Text text = document.createTextNode(valor);           //damos valor
    raiz.appendChild(elem);                               //pegamos el elemento hijo a la raiz
    elem.appendChild(text);                               //pegamos el valor
}
```

En los últimos pasos se crea la fuente XML a partir del documento:

```
Source source = new DOMSource (document);
```

Se crea el resultado en el fichero *Empleados.xml*:

```
Result result = new StreamResult  
    (new java.io.File("Empleados.xml"));    //fichero XML
```

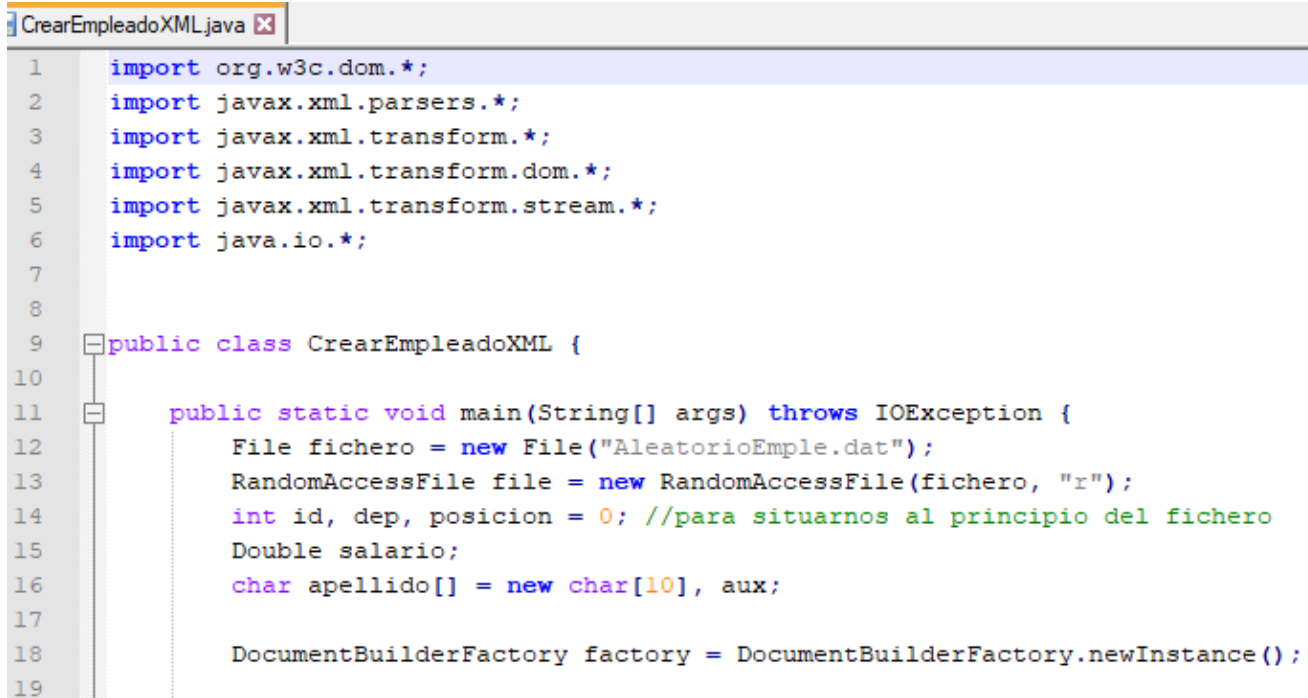
Se obtiene un **TransformerFactory**:

```
Transformer transformer =  
    TransformerFactory.newInstance().newTransformer();
```

Para mostrar el documento por pantalla podemos especificar como resultado el canal de salida *System.out*:

```
Result console = new StreamResult(System.out);  
transformer.transform(source, console);
```

El código completo es el siguiente:



```
CrearEmpleadoXML.java x
```

```
1  import org.w3c.dom.*;  
2  import javax.xml.parsers.*;  
3  import javax.xml.transform.*;  
4  import javax.xml.transform.dom.*;  
5  import javax.xml.transform.stream.*;  
6  import java.io.*;  
7  
8  
9  public class CrearEmpleadoXML {  
10  
11      public static void main(String[] args) throws IOException {  
12          File fichero = new File("AleatorioEmple.dat");  
13          RandomAccessFile file = new RandomAccessFile(fichero, "r");  
14          int id, dep, posicion = 0; //para situarnos al principio del fichero  
15          Double salario;  
16          char apellido[] = new char[10], aux;  
17  
18          DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();  
19
```

```
19 try{
20     DocumentBuilder builder = factory.newDocumentBuilder();
21     DOMImplementation implementation = builder.getDOMImplementation();
22     Document document = implementation.createDocument(null, "Empleados", null);
23     document.setXmlVersion("1.0");
24
25     for(;;){
26         file.seek(posicion); //nos posicionamos
27         id = file.readInt(); //obtengo el id de empleado
28         for (int i=0; i < apellido.length; i++){
29             aux = file.readChar();
30             apellido[i] = aux;
31         }
32         String apellidos = new String(apellido);
33         dep = file.readInt();
34         salario = file.readDouble();
35
36         if (id > 0) { //id validos a partir de 1
37             Element raiz =
38                 document.createElement("empleado"); //nodo empleado
39             document.getDocumentElement().appendChild(raiz);
40
41             //añadir ID
42             CrearElemento("id", Integer.toString(id), raiz, document);
43             //Apellido
44             CrearElemento("apellido", apellidos.trim(), raiz, document);
45             //DEP
46             CrearElemento("dep", Integer.toString(dep), raiz, document);
47             //añadir salario
48             CrearElemento("salario", Double.toString(salario), raiz, document);
49         }
50         posicion = posicion + 36;
51         if (file.getFilePointer() == file.length()) break;
52
53     } // fin del for que recorre el fichero
54     Source source = new DOMSource(document);
55     Result result =
56         new StreamResult(new java.io.File("Empleados.xml"));
57     Transformer transformer =
58         TransformerFactory.newInstance().newTransformer();
59     transformer.transform(source, result);
60
61
62 }catch (Exception e) {System.err.println("Error: "+e);}
63
64 file.close(); //cerrar fichero
65 } //fin del main
```

```

66
67 //Inserción de los datos del empleado
68 static void CrearElemento(String datoEmple, String valor, Element raiz, Document document){
69     Element elem = document.createElement(datoEmple);
70     Text text = document.createTextNode(valor); //damos valor
71     raiz.appendChild(elem); //pegamos el elemento hijo a la raiz
72     elem.appendChild(text); //pegamos el valor
73 }
74 } //fin de la clase

```

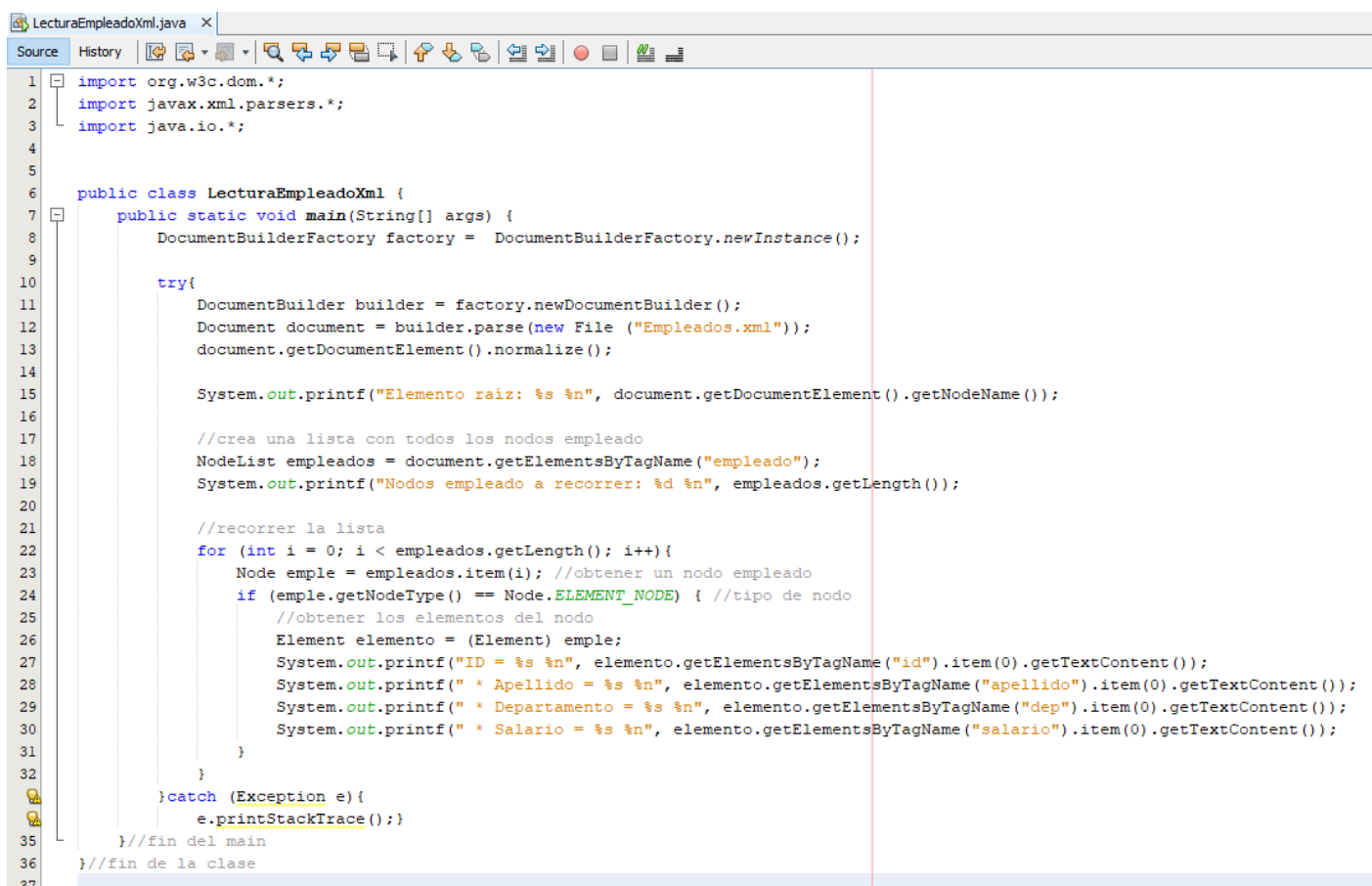
Para leer un documento XML, creamos una instancia de **DocumentBuilderFactory** para construir el parser y cargamos el documento con el método **parse()**:

```
Document document = builder.parse(new File ("Empleados.xml"));
```

Obtenemos la lista de nodos con nombre empleado de todo el documento:

```
NodeList empleados = document.getElementsByTagName ("empleado");
```

Se realiza un bucle para recorrer la lista d nodos. Por cada nodo se obtienen sus etiquetas y sus valores llamando a la función **getNode()**. El código es el siguiente:



```

1  import org.w3c.dom.*;
2  import javax.xml.parsers.*;
3  import java.io.*;
4
5
6  public class LecturaEmpleadoXml {
7      public static void main(String[] args) {
8          DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
9
10         try{
11             DocumentBuilder builder = factory.newDocumentBuilder();
12             Document document = builder.parse(new File ("Empleados.xml"));
13             document.getDocumentElement().normalize();
14
15             System.out.printf("Elemento raíz: %s %n", document.getDocumentElement().getNodeName());
16
17             //crea una lista con todos los nodos empleado
18             NodeList empleados = document.getElementsByTagName("empleado");
19             System.out.printf("Nodos empleado a recorrer: %d %n", empleados.getLength());
20
21             //recorrer la lista
22             for (int i = 0; i < empleados.getLength(); i++){
23                 Node emple = empleados.item(i); //obtener un nodo empleado
24                 if (emple.getNodeType() == Node.ELEMENT_NODE) { //tipo de nodo
25                     //obtener los elementos del nodo
26                     Element elemento = (Element) emple;
27                     System.out.printf("ID = %s %n", elemento.getElementsByTagName("id").item(0).getTextContent());
28                     System.out.printf(" * Apellido = %s %n", elemento.getElementsByTagName("apellido").item(0).getTextContent());
29                     System.out.printf(" * Departamento = %s %n", elemento.getElementsByTagName("dep").item(0).getTextContent());
30                     System.out.printf(" * Salario = %s %n", elemento.getElementsByTagName("salario").item(0).getTextContent());
31                 }
32             }
33         } catch (Exception e){
34             e.printStackTrace();
35         } //fin del main
36     } //fin de la clase
37

```

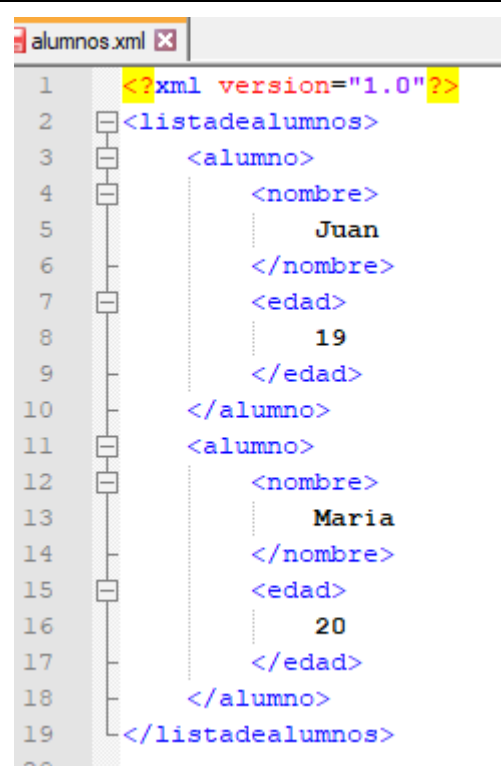
Referencia:

- API DOM: <https://docs.oracle.com/javase/8/docs/api/org/w3c/dom/bootstrap/package-tree.html>

1.7.2. Acceso a Ficheros XML con SAX

SAX (API simple para XML) es un conjunto de clases e interfaces que ofrecen una herramienta muy útil para el procesamiento de documentos XML. Te permite escanear documentos de forma secuencial (es decir, no carga todo el archivo en la memoria como lo hace DOM), esto implica poco consumo de memoria incluso si los documentos son grandes, por el contrario, evita tener una visión general del documento que está analizando. SAX es más complejo de programar que DOM, es una API totalmente escrita en Java e incluida dentro del JRE que nos permite crear nuestro propio analizador XML.

La lectura de un documento XML produce eventos que provocan la llamada al método, los eventos encuentran la etiqueta de inicio y finalización del documento (***startDocument()*** y ***endDocument()***), la etiqueta de inicio y final de un elemento (***startElement()*** y ***endElement()***), ***characters()***, etc.

Documento XML (alumnos.xml)	Métodos asociados a eventos del documento
 <pre> 1 <?xml version="1.0"?> 2 <listadealumnos> 3 <alumno> 4 <nombre> 5 Juan 6 </nombre> 7 <edad> 8 19 9 </edad> 10 </alumno> 11 <alumno> 12 <nombre> 13 Maria 14 </nombre> 15 <edad> 16 20 17 </edad> 18 </alumno> 19 </listadealumnos> </pre>	<pre> startDocument() startElement() startElement() startElement() characters() endElement() startElement() characters() endElement() endElement startElement() startElement() characters() endElement() startElement() characters() endElement() endElement endElement endElement </pre>

Vamos a construir un ejemplo sencillo en Java que muestra los pasos básicos necesarios para hacer que se puedan tratar los eventos. En primer lugar se incluyen las clases e interfaces de SAX:


```
import java.io.*;
import org.xml.sax.Attributes;
import org.xml.sax.InputSource;
import org.xml.sax.SAXException;
import org.xml.sax.XMLReader;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.helpers.XMLReaderFactory;
```

Se crea un objeto procesador de XML, es decir, un **XMLReader**, durante la creación de este objeto se puede producir una excepción (**SAXException**) que es necesario capturar (se incluye en el método *main()*):

```
XMLReader procesadorXML = XMLReaderFactory.createXMLReader() ;
```

A continuación hay que indicar al **XMLReader** qué objetos poseen los métodos que tratarán los eventos. Estos objetos serán normalmente implementaciones de las siguientes interfaces:

- ❖ **ContentHandler**: recibe las notificaciones de los eventos que ocurren en el documento.
- ❖ **DTDHandler**: recoge eventos relacionados con la DTD.
- ❖ **ErrorHandler**: define métodos de tratamientos de errores.
- ❖ **EntityResolver**: sus métodos se llaman cada vez que se encuentra una referencia a una entidad.
- ❖ **DefaultHandler**: clase que provee una implementación por defecto para todos sus métodos, el programador definirá los métodos que sean utilizados por el programa.

Esta clase es de la que extenderemos para poder crear nuestro parser de XML. En el ejemplo, la clase se llama *GestionContenido* y se tratan solo los eventos básicos: inicio y fin de documento, inicio y fin de etiqueta encontrada, encuentra datos carácter (*startDocument()*, *endDocument()*, *startElement()*, *endElement()*, *characters()*):

- **startDocument**: se produce al comenzar el procesamiento del documento XML.
- **endDocument**: se produce al finalizar el procesamiento del documento XML.
- **startElement**: se produce al comenzar el procesamiento de una etiqueta XML. Es aquí donde se leen los atributos de las etiquetas.
- **endElement**: se produce al finalizar el procesamiento de una etiqueta XML.
- **characters**: se produce al encontrar una cadena de texto.

Para indicar al procesador XML los objetos que realizarán el tratamiento se utiliza alguno de los siguientes métodos incluidos dentro de los objetos **XMLReader**: *setContentHandler()*, *setDTDHandler()*, *setEntityResolver()* y *setErrorHandler()*; cada uno trata un tipo de evento y está asociado con una interfaz determinada. En el ejemplo usaremos *setContentHandler()* para tratar los eventos que ocurren en el documento:

```
GestionContenido gestor = new GestionContenido() ;
procesadorXML.setContentHandler(gestor) ;
```

A continuación se define el fichero XML que se va a leer mediante un objeto *InputSource*:

```
InputSource fileXML = new InputSource("alumnos.xml");
```

Por último, se procesa el documento XML mediante el método *parse()* del objeto **XMLReader**, le pasamos un objeto **InputSource**:

El ejemplo completo se muestra a continuación:

```
Ejemplo17SAX_1.java X
1  package ejemplo17sax_1;
2
3  import java.io.*;
4  import org.xml.sax.Attributes;
5  import org.xml.sax.InputSource;
6  import org.xml.sax.SAXException;
7  import org.xml.sax.XMLReader;
8  import org.xml.sax.helpers.DefaultHandler;
9  import org.xml.sax.helpers.XMLReaderFactory;
10
11
12  public class Ejemplo17SAX_1 {
13
14      public static void main(String[] args) throws SAXException, IOException {
15          XMLReader procesadorXML = XMLReaderFactory.createXMLReader();
16          GestionContenido gestor = new GestionContenido();
17          procesadorXML.setContentHandler(gestor);
18          InputSource fileXML = new InputSource("alumnos.xml");
19          procesadorXML.parse(fileXML);
20      }
21  }
22
23
24  class GestionContenido extends DefaultHandler{
25      public GestionContenido(){
26          super();
27      }
28      public void startDocument(){
29          System.out.println("Comienzo del documento XML");
30      }
31
32      public void endDocument(){
33          System.out.println("Final del Documento XML");
34      }
35      public void startElement (String uri, String nombre, String nombreC, Attributes atts){
36          System.out.printf("\tFin Elemento:  %s %n", nombre);
37      }
38      public void characters (char[] ch, int inicio, int longitud){
39          String car = new String (ch, inicio, longitud);
40          //quitar saltos de línea
41          car = car.replaceAll("[\t\n]", "");
42          System.out.printf("\tCaracteres: %s %n", car);
43      }
44  } //fin GestionContenido
```

En el resultado de ejecutar el programa con el fichero *alumnos.xml* se puede observar cómo el orden de ocurrencia de los eventos está relacionado con la estructura del documento:

```
Output - ejemplo17SAX_1 (run)

run:
Comienzo del documento XML
    Fin Elemento:  listadealumnos
    Caracteres:
    Fin Elemento:  alumno
    Caracteres:
    Fin Elemento:  nombre
    Caracteres: Juan
    Caracteres:
    Fin Elemento:  edad
    Caracteres: 19
    Caracteres:
    Caracteres:
    Fin Elemento:  alumno
    Caracteres:
    Fin Elemento:  nombre
    Caracteres: Maria
    Caracteres:
    Fin Elemento:  edad
    Caracteres: 20
    Caracteres:
    Caracteres:
Final del Documento XML
BUILD SUCCESSFUL (total time: 0 seconds)
```

Referencia:

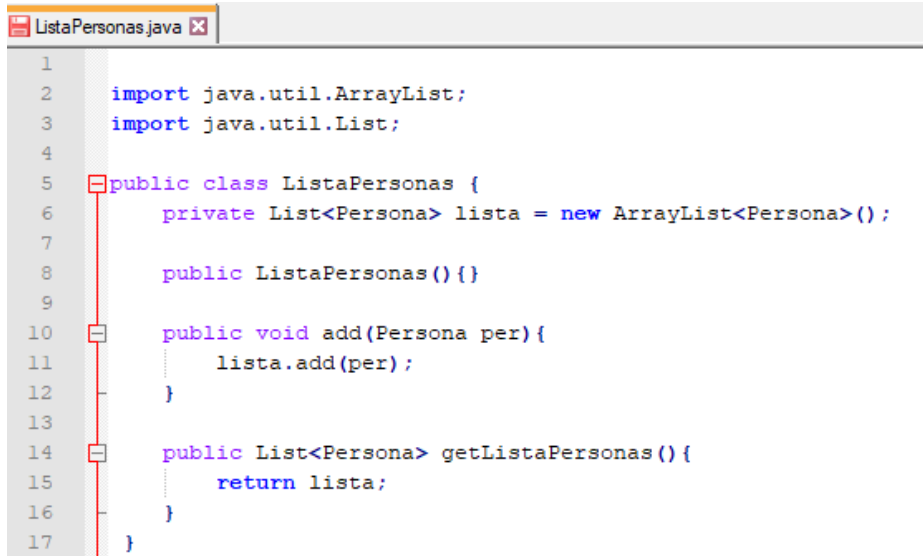
- API SAX: <http://www.saxproject.org/apidoc/>

1.7.3. Serialización de objetos a XML

A continuación vamos a ver cómo se pueden serializar de forma sencilla objetos Java a XML y viceversa; utilizaremos para ello la librería **XStream**. Para poder utilizarla hemos de descargar los JAR desde el sitio Web: <http://xstream.github.io/download.html>. Para el ejemplo se ha descargado el fichero **xstream-distribution-1.4.14-bin.zip** que hemos de descomprimir y buscar el JAR **xstream-1.4.14.jar** que está en la carpeta *lib* que es el que usaremos para el ejemplo. También necesitamos el fichero **kxml2-min-2.3.0.jar** que se localiza en la carpeta *lib/xstream*. Una vez que tenemos los dos ficheros los añadimos a nuestro proyecto Eclipse o NetBeans o los definimos en el CLASSPATH, por ejemplo, supongamos que tenemos los JAR en la carpeta D:\unil\xstream, el CLASSPATH nos quedaría:

```
SET CLASSPATH = .;D:\unil\xstream\kxml2-2.3.0.jar;D:\unil\xstream\xstream-1.4.14.jar
```

Partimos del fichero *FichPersona.dat* que utilizamos en epígrafes anteriores y contiene objetos *Persona*. Crearemos una lista de objetos *Persona* y la convertiremos en un fichero de datos XML. Necesitaremos la clase *Persona* (ya definida) y la clase *ListaPersonas* en la que se define una lista de objetos *Persona* que pasaremos al fichero XML:



```
1
2 import java.util.ArrayList;
3 import java.util.List;
4
5 public class ListaPersonas {
6     private List<Persona> lista = new ArrayList<Persona>();
7
8     public ListaPersonas() {}
9
10    public void add(Persona per) {
11        lista.add(per);
12    }
13
14    public List<Persona> getListaPersonas() {
15        return lista;
16    }
17 }
```

El proceso consistirá en recorrer el fichero *FichPersona.dat* para crear una lista de personas que después se insertarán en el fichero *Personas.xml*, el código Java es el siguiente (fichero *EscribirPersonas.java*):

```

Ejemplo18EscribirPersonas.java
1
2  import java.util.*;
3  import java.io.*;
4  import com.thoughtworks.xstream.XStream;
5
6  public class Ejemplo18EscribirPersonas{
7      public static void main(String[] args) throws IOException{
8          //Creamos unha lista de personas
9          //ArrayList<Persona> listaPersoa = new ArrayList<Persona>();
10         ListaPersonas listaper = new ListaPersonas();
11         try{
12             //Cargar o contido de ficheiroPersoas.dat nun ArrayList
13             FileInputStream filein = new FileInputStream("FichPersona.dat");
14             ObjectInputStream datosFich = new ObjectInputStream(filein);
15
16             System.out.println("Comienza el proceso.....");
17
18             while (filein.available() > 0){
19                 Persona persona = (Persona)datosFich.readObject();//Leo del fichero
20                 listaper.add(persona); //Añado a la lista
21             }//while
22             datosFich.close();
23         }catch(Exception error){
24             System.out.println("Error en la carga del listado con informacion procedente del fichero. "
25                 +error.getMessage());
26         }
27
28         try{
29             //Generar el XML con informacion que tenemos en la lista
30             XStream xstream = new XStream();
31             //Cambiar d nombre a las etiquetas XML
32             xstream.alias("ListaPersonasMunicipio",List.class);
33             xstream.alias("DatosPersona",Persona.class);
34             //quitar etiqueta lista (atributo de la clase ListaPersonas)
35             xstream.addImplicitCollection(ListaPersonas.class, "lista");
36             //Insertar los objetos en el XML
37             xstream.toXML(listaper,new FileOutputStream("PersonasXML.xml"));
38         }
39         catch(Exception error){
40             System.out.println("Error en la transformación a XML. "+error.getMessage());
41         }
42
43     }//main
44 }//class

```

El fichero 'PersonasXML.xml' generado tiene el siguiente aspecto:



En primer lugar para utilizar XStream, simplemente creamos una instancia de la clase XStream:

```
XStream xstream = new XStream();
```

En general las etiquetas XML se corresponden con el nombre de los atributos de la clase, pero se pueden cambiar usando el método **alias(String alias, Class clase)**. En el ejemplo se ha dado un alias a la clase *ListaPersonas*, en el XML aparecerá con el nombre *ListaPersonasMunicipio*:

```
xstream.alias("ListaPersonasMunicipio", ListaPersonas.class);
```

También se ha dado un alias a la clase *Persona*, en el XML aparecerá con el nombre *DatosPersona*:

```
xstream.alias("DatosPersona", Persona.class);
```

El método **aliasField (String alias, Class clase, String nombre campo)**, permite crear un alias para un nombre de campo. Por ejemplo, si queremos cambiar el nombre de los campos *nombre* y *edad* (de la clase *Persona*) crearíamos los siguientes alias:

```
xstream.aliasField("Nombre alumno", Persona.class, "nombre");
xstream.aliasField("Edad alumno", Persona.class, "edad");
```

Entonces en el fichero XML se crearían con las etiquetas `<Nombre alumno>` en lugar de `<nombre>` y `<Edad alumno>` en lugar de `<edad>`.

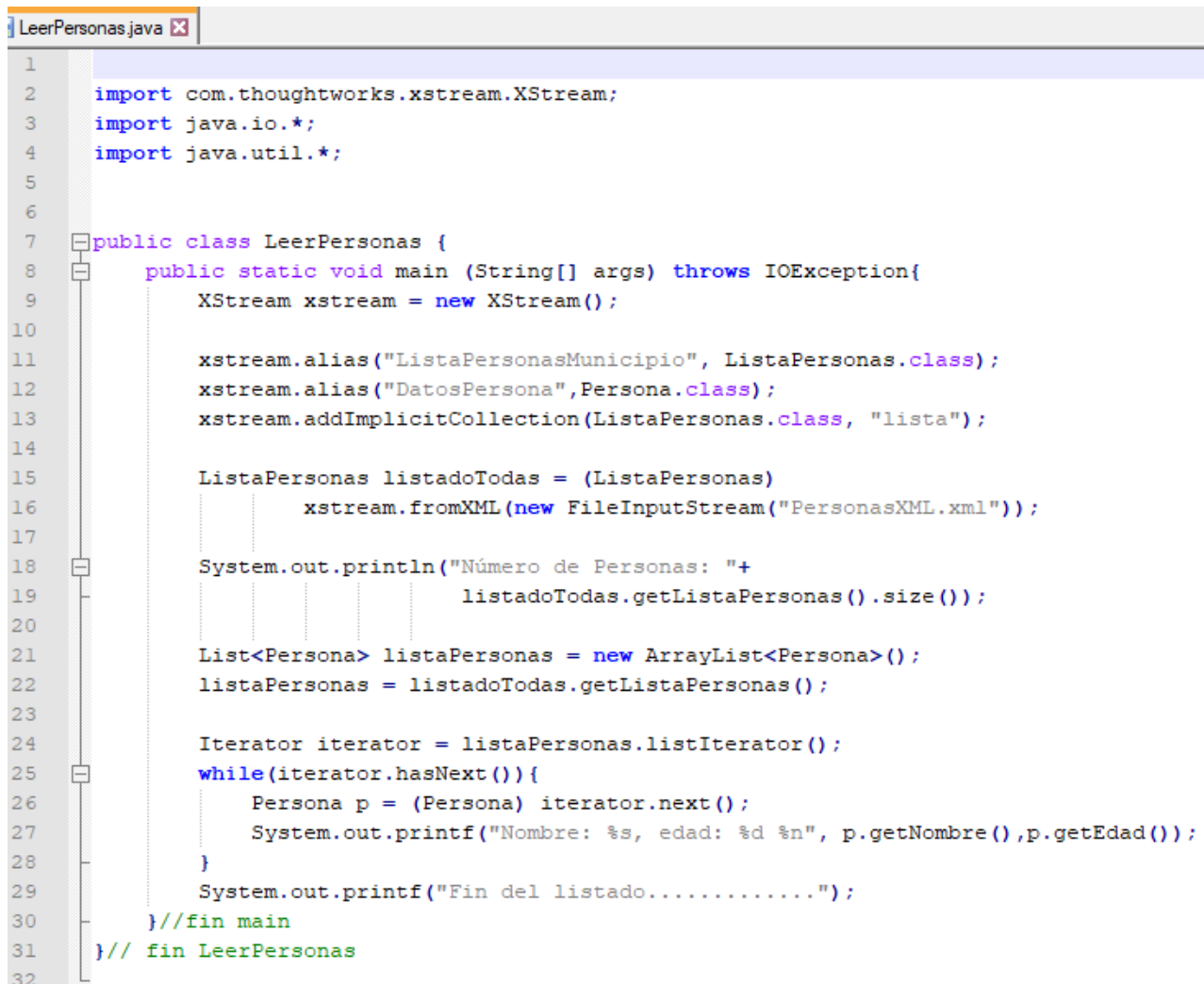
Para que no aparezca el atributo `lista` de la clase `ListaPersonas` en el XML generado se ha utilizado el método `addImplicitCollection (Class clase, String nombredecampo)`

```
xstream.addImplicitCollection(ListaPersonas.class, "lista");
```

Por último, para generar el fichero `Personas.xml` a partir de la lista de objetos se utiliza el método `toXML(Object objeto, OutputStream out)`:

```
xstream.toXML(listaper, new FileOutputStream("Personas.xml"));
```

El proceso para realizar la lectura del fichero XML generado es el siguiente:



```

1
2 import com.thoughtworks.xstream.XStream;
3 import java.io.*;
4 import java.util.*;
5
6
7 public class LeerPersonas {
8     public static void main (String[] args) throws IOException{
9         XStream xstream = new XStream();
10
11         xstream.alias("ListaPersonasMunicipio", ListaPersonas.class);
12         xstream.alias("DatosPersona", Persona.class);
13         xstream.addImplicitCollection(ListaPersonas.class, "lista");
14
15         ListaPersonas listadoTodas = (ListaPersonas)
16             xstream.fromXML(new FileInputStream("PersonasXML.xml"));
17
18         System.out.println("Número de Personas: "+
19             listadoTodas.getListaPersonas().size());
20
21         List<Persona> listaPersonas = new ArrayList<Persona>();
22         listaPersonas = listadoTodas.getListaPersonas();
23
24         Iterator iterator = listaPersonas.listIterator();
25         while(iterator.hasNext()){
26             Persona p = (Persona) iterator.next();
27             System.out.printf("Nombre: %s, edad: %d %n", p.getNombre(), p.getEdad());
28         }
29         System.out.printf("Fin del listado.....");
30     } //fin main
31 } // fin LeerPersonas
32

```


Se deben utilizar los métodos *alias()* y *addImplicitCollection()* para leer el XML ya que se usaron para hacer la escritura del mismo. Para obtener el objeto con la lista de personas o lo que es lo mismo para deserializar el objeto a partir del fichero, utilizamos el método *fromXML(InputStream input)* que devuelve un tipo **Object**:

```
ListaPersonas listadoTodas = (ListaPersonas)
    xstream.fromXML(new FileInputStream("PersonasXML.xml"));
```

Es necesario añadir dos librerías más: `'xmlpull'` en mi caso `'xmlpull-1.1.3.1.jar'` y `'xpp3_min'` en mi caso `'xpp3_min-1.1.4c.jar'`.

Referencia:

- API XStream: <http://x-stream.github.io/javadoc/index.html>

1.8. ACTIVIDADES

ACTIVIDAD 1.5

A partir del fichero de objetos Persona utilizado anteriormente crear un documento XML usando DOM.

ACTIVIDAD 1.6

Utilizar SAX para visualizar el contenido del fichero 'Empleados.xml' creado anteriormente.