ACTIVIDAD 1.3

Consulta. Crea un programa Java que consulte los datos de un empleado del fichero aleatorio. El programa se ejecutará desde la línea de comandos y debe recibir un identificador de empleado. Si el empleado existe se visualizarán sus datos, si no existe se visualizará un mensaje indicándolo.

Inserción. Crea un programa Java que inserte datos en el fichero aleatorio. El programa se ejecutará desde la línea de comandos y debe recibir 4 parámetros: identificador de empleado, apellido, departamento y salario. Antes de insertar se comprobará si el identificador existe, en ese caso se debe visualizar un mensaje indicándolo; si no existe se deberá insertar.

Para modificar un registro determinado, accedemos a su posición y efectuamos las modificaciones. El fichero debe abrirse en modo "*rw*". Por ejemplo, para cambiar el departamento y salario del empleado con identificador 4 escribo lo siguiente:

```
int registro = 4;                        //id a modificar
long posicion = (registro -1 ) * 36; //calculo la posición
posición = posición + 4 + 20;          //sumo el tamaño de ID + apellido
file.seek(posicion);                     //nos posicionamos
file.writeInt(40);                       //modifico departamento
file.writeDouble(4000.87);               //modifico salario
```

ACTIVIDAD 1.4

Modificación. Crea un programa Java que reciba desde la línea de comandos un identificador de empleado y un importe. Se debe realizar la modificación del salario. La modificación consistirá en sumar al salario del empleado el importe introducido. El programa debe visualizar el apellido, el salario antiguo y el nuevo. Si el identificador no existe se visualizará mensaje indicándolo.

Borrado. Crea un programa Java que al ejecutarlo desde la línea de comandos reciba un identificador de empleado y lo borre. Se hará un borrado lógico marcando el registro con la siguiente información: el identificador será igual a -1, el apellido será igual al identificador que se borra, y el departamento y salario serán 0.

A continuación haz otro programa Java (o crea un método dentro del anterior programa) que muestre los identificadores de los empleados borrados .

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 o 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 XML 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 muy 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 que equivale a un ejemplar de un documento XML. 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.
- **NodeList**. 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 epígrade 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.transform.dom.*;
import javax.xml.transform.stream.*;
import java.io.*;
```

A continuación creamos una instancia de **DocumentBuilderFactory** para construir el parser, se debe encerrar entre **try-cath** 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 versión 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 version 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 (raiz), 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 **TransfomerFactory**:

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

Se realiza la transformación del documento a fichero:

```
transformer.transform(source, result);
```

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:

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

public class CrearEmpleadoXml {
 public static void main(String args[]) throws IOException{
   File fichero = new File("AleatorioEmple.dat");
   RandomAccessFile file = new RandomAccessFile(fichero, "r");

   int  id, dep, posicion=0; //para situarnos al principio del fichero
   Double salario;
   char apellido[] = new char[10], aux;

   DocumentBuilderFactory factory =
                  DocumentBuilderFactory.newInstance();

   try{
     DocumentBuilder builder = factory.newDocumentBuilder();
     DOMImplementation implementation = builder.getDOMImplementation();
     Document document =
          implementation.createDocument(null, "Empleados", null);
     document.setXmlVersion("1.0");

     for(;;) {
       file.seek(posicion); //nos posicionamos
       id=file.readInt();   // obtengo id de empleado
```

```
      for (int i = 0; i < apellido.length; i++) {
        aux = file.readChar();
        apellido[i] = aux;
      }
      String apellidos = new String(apellido);
      dep = file.readInt();
      salario = file.readDouble();

      if(id>0) { //id validos a partir de 1
        Element raiz =
                  document.createElement("empleado"); //nodo empleado
        document.getDocumentElement().appendChild(raiz);

        //añadir ID
        CrearElemento("id",Integer.toString(id), raiz, document);
        //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);
      }
      posicion= posicion + 36; // me posiciono para el sig empleado

      if (file.getFilePointer() == file.length()) break;

    }//fin del for que recorre el fichero

    Source source = new DOMSource(document);
    Result result =
          new StreamResult(new java.io.File("Empleados.xml"));
    Transformer transformer =
          TransformerFactory.newInstance().newTransformer();
    transformer.transform(source, result);

   }catch(Exception e){ System.err.println("Error: "+e); }

   file.close();  //cerrar fichero
 }//fin de main

 //Inserción de los datos del empleado
 static void  CrearElemento(String datoEmple, String valor,
                            Element raiz, Document document){
   Element elem = document.createElement(datoEmple);
   Text text = document.createTextNode(valor); //damos valor
   raiz.appendChild(elem); //pegamos el elemento hijo a la raiz
   elem.appendChild(text); //pegamos el valor
 }
}//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 esta lista de nodos. Por cada nodo se obtienen sus etiquetas y sus valores llamando a la función ***getNodo()***. El código es el siguiente:

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

public class LecturaEmpleadoXml {
 public static void main(String[] args) {

  DocumentBuilderFactory factory =
                        DocumentBuilderFactory.newInstance();

  try {
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document document = builder.parse(new File("Empleados.xml"));
    document.getDocumentElement().normalize();

    System.out.printf("Elemento raiz: %s %n",
                      document.getDocumentElement().getNodeName());
    //crea una lista con todos los nodos empleado
    NodeList empleados = document.getElementsByTagName("empleado");
    System.out.printf("Nodos empleado a recorrer: %d %n",
                      empleados.getLength());

    //recorrer la lista
    for (int i = 0; i < empleados.getLength(); i ++) {
      Node emple = empleados.item(i); //obtener un nodo empleado
      if (emple.getNodeType() == Node.ELEMENT_NODE) {//tipo de nodo
        //obtener los elementos del nodo
        Element elemento = (Element) emple;
        System.out.printf("ID = %s %n",
           elemento.getElementsByTagName("id").
                                        item(0).getTextContent());
        System.out.printf(" * Apellido = %s %n",
           elemento.getElementsByTagName("apellido").
                                        item(0).getTextContent());
        System.out.printf(" * Departamento = %s %n",
           elemento.getElementsByTagName("dep").
                                        item(0).getTextContent());
        System.out.printf(" * Salario = %s %n",
           elemento.getElementsByTagName("salario").
                                        item(0).getTextContent());
      }
    }
  } catch (Exception e)
    {e.printStackTrace();}

 }//fin de main
}//fin de la clase
```

¡¡INTERESANTE!!

API DOM: *http://docs.oracle.com/javase/8/docs/api/org/w3c/dom/package-tree.html.*

ACTIVIDAD 1.5

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

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. Permite analizar los documentos de forma secuencial (es decir, no carga todo el fichero en memoria como hace DOM), esto implica poco consumo de memoria aunque los documentos sean de gran tamaño, en contraposición, impide tener una visión global del documento que se va a analizar. SAX es más complejo de programar que DOM, es un API totalmente escrita en Java e incluida dentro del JRE que nos permite crear nuestro propio parser de XML.

La lectura de un documento XML produce eventos que ocasiona la llamada a métodos, los eventos son encontrar la etiqueta de inicio y fin del documento (***startDocument()*** y ***endDocument()***), la etiqueta de inicio y fin de un elemento (***startElement()*** y ***endElement()***), los caracteres entre etiquetas (***characters()***), etc:

Documento XML (alumnos.xml)	Métodos asociados a eventos del documento

`<?xml version="1.0"?>`
`<listadealumnos>`
`  <alumno>`
`  <nombre>`
`  Juan`
`  </nombre>`
`  <edad>`
`  19`
`  </edad>`
`  </alumno>`
`  <alumno>`
`  <nombre>`
`  Maria`
`  </nombre>`
`  <edad>`
`  20`
`  </edad>`
`  </alumno>`
`</listadealumnos>`
	`startDocument()`
	`startElement()`
	`  startElement()`
	`  startElement()`
	`  characters()`
	`  endElement()`
	`  startElement()`
	`  characters()`
	`  endElement()`
	`  endElement()`
	`  startElement()`
	`  startElement()`
	`  characters()`
	`  endElement()`
	`  startElement()`
	`  characters()`
	`  endElement()`
	`  endElement()`
	`endElement()`
	`endDocument()`

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 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 procesado del documento XML.
- ***endDocument***: se produce al finalizar el procesado del documento XML.
- ***startElement***: se produce al comenzar el procesado de una etiqueta XML. Es aquí donde se leen los atributos de las etiquetas.
- ***endElement***: se produce al finalizar el procesado 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**:

```
procesadorXML.parse(fileXML);
```

El ejemplo completo se muestra a continuación:

```
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;

public class PruebaSax1 {
     public static void main(String[] args)
             throws FileNotFoundException, IOException, SAXException{

      XMLReader  procesadorXML = XMLReaderFactory.createXMLReader();
      GestionContenido gestor = new GestionContenido();
      procesadorXML.setContentHandler(gestor);
      InputSource fileXML = new InputSource("alumnos.xml");
      procesadorXML.parse(fileXML);
     }
}//fin PruebaSax1

class GestionContenido extends DefaultHandler {
         public GestionContenido() {
             super();
         }
         public void startDocument() {
             System.out.println("Comienzo del Documento XML");
         }
         public void endDocument() {
             System.out.println("Final del Documento XML");
         }
         public void startElement(String uri, String nombre,
                   String nombreC, Attributes atts) {
             System.out.printf("\tPrincipio Elemento: %s %n",nombre);
         }
         public void endElement(String uri, String nombre,
                         String nombreC) {
             System.out.printf("\tFin Elemento: %s %n", nombre);
         }
         public void characters(char[] ch, int inicio, int longitud)
                                              throws SAXException {
              String car = new String(ch, inicio, longitud);
              //quitar saltos de línea
              car = car.replaceAll("[\t\n]","");
              System.out.printf ("\tCaracteres: %s %n", car);
         }

}//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:

```
Comienzo del Documento XML
  Principio Elemento: listadealumnos
  Caracteres:
  Principio Elemento: alumno
  Caracteres:
  Principio Elemento: nombre
  Caracteres:   Juan
  Fin Elemento: nombre
  Caracteres:
  Principio Elemento: edad
  Caracteres:   19
  Fin Elemento: edad
  Caracteres:
  Fin Elemento: alumno
  Caracteres:
  Principio Elemento: alumno
  Caracteres:
  Principio Elemento: nombre
  Caracteres:  Maria
  Fin Elemento: nombre
  Caracteres:
  Principio Elemento: edad
  Caracteres:  20
  Fin Elemento: edad
  Caracteres:
  Fin Elemento: alumno
  Caracteres:
  Fin Elemento: listadealumnos
Final del Documento XML
```

¡¡ INTERESANTE ! !

API SAX: *http://www.saxproject.org/apidoc/org/xml/sax/package-tree.*

ACTIVIDAD 1.6

Utiliza SAX para visualizar el contenido del fichero *Empleados.xml* creado anteriormente.

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 descargarnos los JAR desde el sitio Web: *http://x-stream.github.io/download.html*. Para el ejemplo se ha descargado el fichero **xstream-distribution-1.4.8-bin.zip** que hemos de descomprimir y buscar el JAR **xstream-1.4.8.jar** que está en la carpeta *lib* que es el que usaremos para el ejemplo. También necesitamos el fichero **kxml2-2.3.0.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:\uni1\xstream*, el CLASSPATH nos quedaría:

```
SET CLASSPATH =
   .;D:\uni1\xstream\kxml2-2.3.0.jar;D:\uni1\xstream\xstream-1.4.8.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:

```
import java.util.ArrayList;
import java.util.List;
public class ListaPersonas {
    private List<Persona> lista = new ArrayList<Persona>();

    public ListaPersonas(){ }

    public void add(Persona per) {
            lista.add(per);
    }
    public List<Persona> getListaPersonas() {
            return lista;
    }
}
```

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*):

```
import java.io.*;
import com.thoughtworks.xstream.XStream;

public class EscribirPersonas {
  public static void main(String[] args) throws IOException,
                                        ClassNotFoundException {
    File fichero = new File("FichPersona.dat");
    FileInputStream filein = new FileInputStream(fichero);
    ObjectInputStream dataIS = new ObjectInputStream(filein);

    System.out.println("Comienza el proceso...");

    //Creamos un objeto Lista de Personas
    ListaPersonas listaper = new ListaPersonas();

    try {
        while (true) { //lectura del fichero
           Persona persona= (Persona) dataIS.readObject();
           listaper.add(persona); //añadir persona a la lista
        }
    } catch (EOFException eo) {}
    dataIS.close();  //cerrar stream de entrada

    try {
       XStream xstream = new XStream();
       //cambiar de nombre a las etiquetas XML
       xstream.alias("ListaPersonasMunicipio", ListaPersonas.class);
       xstream.alias("DatosPersona", Persona.class);
       //quitar etiqueta lista (atributo de la clase ListaPersonas)
       xstream.addImplicitCollection(ListaPersonas.class, "lista");
       //Insertar los objetos en el XML
       xstream.toXML(listaper,new FileOutputStream("Personas.xml"));
       System.out.println("Creado fichero XML....");

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

```
  } // fin main
} //fin EscribirPersonas
```

El fichero generado tiene el siguiente aspecto:

```
<ListaPersonasMunicipio>
  <DatosPersona>
    <nombre>Ana</nombre>
    <edad>14</edad>
  </DatosPersona>
  <DatosPersona>
    <nombre>Luis Miguel</nombre>
    <edad>15</edad>
  </DatosPersona>
  <DatosPersona>
    <nombre>Alicia</nombre>
    <edad>13</edad>
  </DatosPersona>
  <DatosPersona>
    <nombre>Pedro</nombre>
    <edad>15</edad>
  </DatosPersona>
  <DatosPersona>
    <nombre>Manuel</nombre>
    <edad>16</edad>
  </DatosPersona>
  <DatosPersona>
    <nombre>Andrés</nombre>
    <edad>12</edad>
  </DatosPersona>
  <DatosPersona>
    <nombre>Julio</nombre>
    <edad>16</edad>
  </DatosPersona>
  <DatosPersona>
    <nombre>Antonio</nombre>
    <edad>14</edad>
  </DatosPersona>
  <DatosPersona>
    <nombre>María Jesús</nombre>
    <edad>13</edad>
  </DatosPersona>
</ListaPersonasMunicipio>
```

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 nombrecampo)***, 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:

```
import java.io.*;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import com.thoughtworks.xstream.XStream;

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

    XStream xstream = new XStream();

    xstream.alias("ListaPersonasMunicipio", ListaPersonas.class);
    xstream.alias("DatosPersona", Persona.class);
    xstream.addImplicitCollection(ListaPersonas.class, "lista");

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

    System.out.println("Numero de Personas: " +
                       listadoTodas.getListaPersonas().size());

    List<Persona> listaPersonas = new ArrayList<Persona>();
    listaPersonas = listadoTodas.getListaPersonas();

    Iterator iterador = listaPersonas.listIterator();
    while( iterador.hasNext() ) {
       Persona p = (Persona) iterador.next();
       System.out.printf("Nombre: %s, edad: %d %n",
                         p.getNombre(), p.getEdad());
    }
    System.out.println("Fin de listado .....");
 } //fin main
}//fin LeerPersonas
```

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("Personas.xml"));
```

¡¡ INTERESANTE ! !
API XStream: *http://x-stream.github.io/javadoc/index.html.*

1.7.4. Conversión de ficheros XML a otro formato

XSL (***Extensible Stylesheet Language***) es toda una familia de recomendaciones del *World Wide Web Consortium* (*http://www.w3.org/Style/XSL/*) 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 que representan a las reglas de construcción y elementos de reglas de estilo que representan a las reglas de mezcla de estilos. En el siguiente ejemplo vamos a ver cómo a partir de un fichero XML que contiene datos y otro XSL que contiene la presentación de esos datos, se puede generar un fichero HTML usando el lenguaje Java. Los ficheros son los siguientes:

FICHERO alumnos.xml:

```
<?xml version="1.0"?>
<listadealumnos>
   <alumno>
        <nombre>Juan</nombre>
        <edad>19</edad>
   </alumno>
   <alumno>
        <nombre>Maria</nombre>
        <edad>20</edad>
   </alumno>
</listadealumnos>
```

FICHERO alumnosPlantilla.xsl:

```
<?xml version="1.0" encoding='ISO-8859-1'?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match='/'>
      <html><xsl:apply-templates /></html>
 </xsl:template>
 <xsl:template match='listadealumnos'>
      <head><title>LISTADO DE ALUMNOS</title></head>
      <body>
        <h1>LISTA DE ALUMNOS</h1>
        <table border='1'>
          <tr><th>Nombre</th><th>Edad</th></tr>
          <xsl:apply-templates select='alumno' />
        </table>
      </body>
 </xsl:template>

 <xsl:template match='alumno'>
       <tr><xsl:apply-templates /></tr>
 </xsl:template>

 <xsl:template match='nombre|edad'>
       <td><xsl:apply-templates /></td>
 </xsl:template>

</xsl:stylesheet>
```

Para realizar la transformación se necesita obtener un objeto **Transformer** que se obtiene creando una instancia de **TransformerFactory** y aplicando el método ***newTransformer(Source source)*** a la fuente XSL que vamos a utilizar para aplicar la transformación del fichero de datos XML, o lo que es lo mismo, para aplicar la hoja de estilos XSL al fichero XML:

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

La transformación se consigue llamando al método ***transform(Source fuenteXml, Result resultado)***, pasándole los datos (el fichero XML) y el stream de salida (el fichero HTML):

```
transformer.transform(datos, result);
```

El código es el siguiente:

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

public class convertidor {
 public static void main(String[] args) throws IOException{
  String hojaEstilo = "alumnosPlantilla.xsl";
  String datosAlumnos = "alumnos.xml";
  File pagHTML = new File("mipagina.html");

  //crear fichero HTML
  FileOutputStream os = new FileOutputStream(pagHTML);
  Source estilos = new StreamSource(hojaEstilo); //fuente XSL
  Source datos = new StreamSource(datosAlumnos); //fuente XML

  //resultado de la transformación
  Result result = new StreamResult(os);

  try{
   Transformer transformer =
            TransformerFactory.newInstance().newTransformer(estilos);
   transformer.transform(datos, result);  //obtiene el HTML
  }
  catch(Exception e){System.err.println("Error: "+e);}

  os.close();  //cerrar fichero

 }//de main

}//de la clase
```

El fichero HTML generado se muestra en la Figura 1.3.

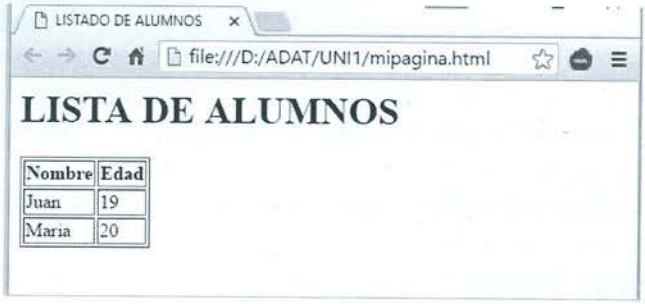

Figura 1.3. Fichero HTML generado.

¡¡ INTERESANTE ! !
API de Java: *http://docs.oracle.com/javase/8/docs/api/index.html.*

1.8. EXCEPCIONES: DETECCIÓN Y TRATAMIENTO

Una **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, es capturada por el gestor de excepciones por defecto que retorna un mensaje y detiene el programa. La ejecución del siguiente programa produce una excepción y visualiza un mensaje indicando el error:

```
public class ejemploExcepcion {
   public static void main(String[] args) {
      int nume = 10, denom = 0, cociente;
      cociente = nume / denom;
      System.out.printf("Resultado: %d", cociente);
   }
}//

D:\uni1>java ejemploExcepcion
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ejemploExcepcion.main(ejemploExcepcion.java:4)
```

Cuando dicho error ocurre dentro de un método Java, el método crea un objeto **Exception** y lo maneja fuera, en el sistema de ejecución. El manejo de excepciones en Java está diseñado pensando en situaciones en las que 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** que a su vez es una clase derivada de la clase base **Throwable**.

1.8.1. Capturar excepciones

Para capturar una excepción se utiliza el bloque **try-catch**. Se encierra en un bloque **try** el código que puede generar una excepción, este bloque va seguido por 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 el bloque **finally** para cerrar ficheros o liberar otros recursos del sistema después de que ocurra una excepción: