

**Enunciado.**

**Apartado 1) (5 puntos) Realiza las siguientes acciones utilizando NetBeans:**

**(2,5 puntos) Crear un fichero EMPLEADOS.DAT de acceso aleatorio, que contenga al menos cinco empleados. Dicho fichero contendrá los campos siguientes: CODIGO (int), NOMBRE (string), DIRECCION (string), SALARIO (float) y COMISION (float).**

**(2,5 puntos) A partir de los datos del fichero EMPLEADOS.DAT crear un fichero llamado EMPLEADOS.XML usando DOM.**

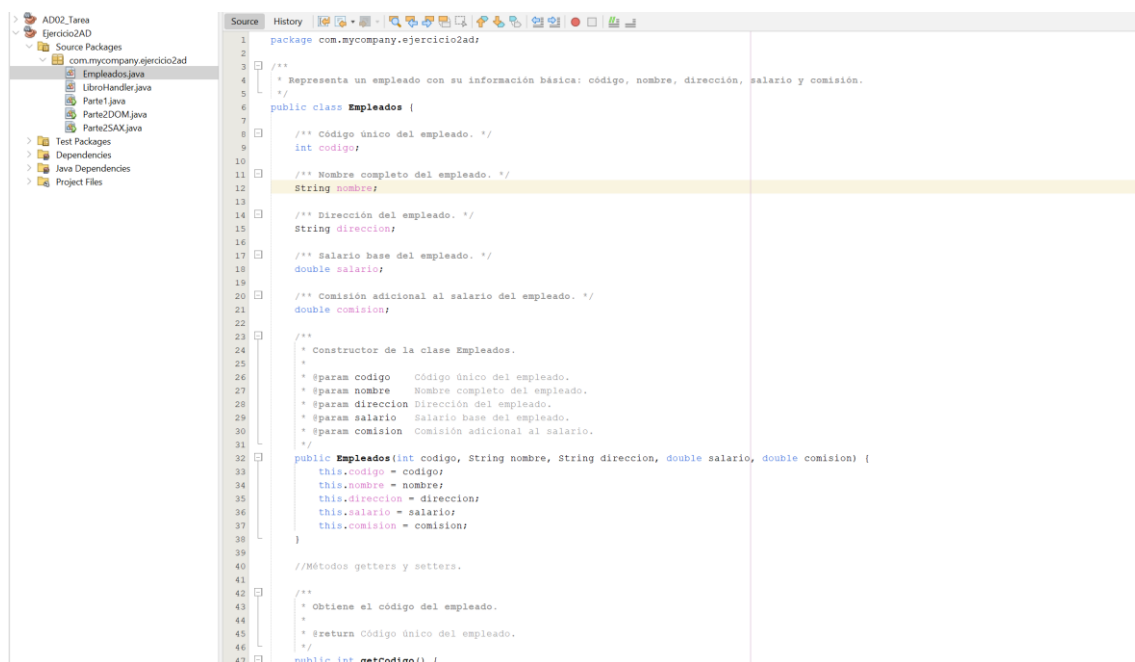
**Apartado 2) (5 puntos) Visualizar todas las etiquetas del fichero LIBROS.XML utilizando las técnicas DOM y SAX.**

**Realizar todos los ejercicios en un mismo proyecto, y subir el mismo comprimido.**

**EJERCICIO 1. Crear un fichero EMPLEADOS.DAT de acceso aleatorio, que contenga al menos cinco empleados. Dicho fichero contendrá los campos siguientes: CODIGO (int), NOMBRE (string), DIRECCION (string), SALARIO (float) y COMISION (float).**

### 1. Diseño de la Clase Empleados

- Se creó una clase Empleados para modelar los datos de un empleado.
- Los atributos principales son:
  - codigo (int): Identificador único del empleado.
  - nombre (String): Nombre completo.
  - direccion (String): Dirección del empleado.
  - salario (double): Salario base.
  - comision (double): Comisión adicional al salario.
- Se definió un constructor para inicializar los atributos.
- Se incluyeron métodos getter y setter para permitir la encapsulación y manipulación de los datos.



```
1 package com.myccompany.ejercicio2ad;
2
3 /**
4  * Representa un empleado con su información básica: código, nombre, dirección, salario y comisión.
5  */
6 public class Empleados {
7
8     /** Código único del empleado. */
9     int codigo;
10
11     /** Nombre completo del empleado. */
12     String nombre;
13
14     /** Dirección del empleado. */
15     String direccion;
16
17     /** Salario base del empleado. */
18     double salario;
19
20     /** Comisión adicional al salario del empleado. */
21     double comision;
22
23     /**
24      * Constructor de la clase Empleados.
25      *
26      * @param codigo    Código único del empleado.
27      * @param nombre    Nombre completo del empleado.
28      * @param direccion Dirección del empleado.
29      * @param salario    Salario base del empleado.
30      * @param comision  Comisión adicional al salario.
31      */
32     public Empleados(int codigo, String nombre, String direccion, double salario, double comision) {
33         this.codigo = codigo;
34         this.nombre = nombre;
35         this.direccion = direccion;
36         this.salario = salario;
37         this.comision = comision;
38     }
39
40     //Métodos getters y setters.
41
42     /**
43      * Obtiene el código del empleado.
44      *
45      * @return Código único del empleado.
46      */
47     public int getCodigo() {
```

## 2. Creación de los Datos de Prueba

- En el método main, se crearon cinco objetos de tipo Empleados con datos ficticios.
- Los objetos se almacenaron en una lista ArrayList llamada plantilla.

```
public class Partel {  
  
    /**  
     * Punto de entrada principal del programa.  
     *  
     * @param args Argumentos de la línea de comandos.  
     */  
    public static void main(String[] args) {  
  
        // Crear objetos Empleados con datos ficticios  
        Empleados empleado1 = new Empleados(1, "David Garcia", "Calle falsa 1", 1000.00, 100.00);  
        Empleados empleado2 = new Empleados(2, "Alba Calzas", "Calle falsa 2", 2000.00, 200.00);  
        Empleados empleado3 = new Empleados(3, "Manolo Cabezabolo", "Calle falsa 3", 3000.00, 300.00);  
        Empleados empleado4 = new Empleados(4, "Juanra Bogordo", "Calle falsa 4", 4000.00, 400.00);  
        Empleados empleado5 = new Empleados(5, "Tomás Turbado", "Calle falsa 5", 5000.00, 500.00);  
  
        // Crear una lista de empleados  
        ArrayList<Empleados> plantilla = new ArrayList<>(Arrays.asList(empleado1, empleado2, empleado3, empleado4, empleado5));  
        plantilla.add(empleado5);  
  
        /**  
         * Alternativamente, se pueden agregar empleados usando el método add:  
         * plantilla.add(empleado1); plantilla.add(empleado2);  
         * plantilla.add(empleado3); plantilla.add(empleado4);  
         * plantilla.add(empleado5);  
         */  
    }  
}
```

## 3. Creación del Archivo Binario

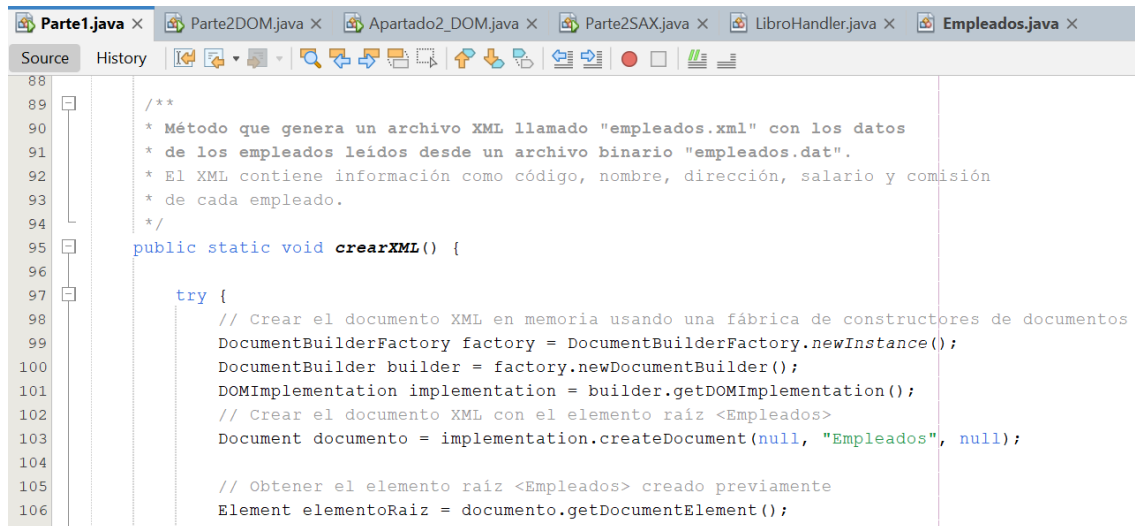
- Se utilizó la clase RandomAccessFile para manejar el archivo binario empleados.dat:
  - writeInt para escribir el código.
  - writeChars (con StringBuffer) para escribir nombre y dirección con un tamaño fijo de 20 caracteres.
  - writeDouble para escribir salario y comisión.

```
// Crear el archivo binario "empleados.dat"  
try (RandomAccessFile raf = new RandomAccessFile("empleados.dat", "rw")) {  
  
    for (Empleados e : plantilla) {  
  
        raf.writeInt(e.codigo); // Escribir el código  
  
        // Escribir el nombre (20 caracteres)  
        StringBuffer sb = new StringBuffer(e.nombre);  
        sb.setLength(20);  
        raf.writeChars(sb.toString());  
  
        // Escribir la dirección (20 caracteres)  
        sb = new StringBuffer(e.direccion);  
        sb.setLength(20);  
        raf.writeChars(sb.toString());  
  
        // Escribir salario y comisión  
        raf.writeDouble(e.salario);  
        raf.writeDouble(e.comision);  
  
    }  
  
    crearXML();  
  
} catch (FileNotFoundException ex) {  
    Logger.getLogger(Partel.class.getName()).log(Level.SEVERE, null, ex);  
} catch (IOException ex) {  
    Logger.getLogger(Partel.class.getName()).log(Level.SEVERE, null, ex);  
}
```

## EJERCICIO 2. A partir de los datos del fichero EMPLEADOS.DAT crear un fichero llamado EMPLEADOS.XML usando DOM.

### 1. Creación del Documento XML:

- Utilizas DocumentBuilderFactory y DocumentBuilder para crear un nuevo documento XML en memoria.
- Añades un elemento raíz <empleados> como contenedor principal de todos los empleados.



```
88
89
90  /**
91   * Método que genera un archivo XML llamado "empleados.xml" con los datos
92   * de los empleados leídos desde un archivo binario "empleados.dat".
93   * El XML contiene información como código, nombre, dirección, salario y comisión
94   * de cada empleado.
95   */
96
97  public static void crearXML() {
98
99      try {
100          // Crear el documento XML en memoria usando una fábrica de constructores de documentos
101          DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
102          DocumentBuilder builder = factory.newDocumentBuilder();
103          DOMImplementation implementation = builder.getDOMImplementation();
104          // Crear el documento XML con el elemento raíz <Empleados>
105          Document documento = implementation.createDocument(null, "Empleados", null);
106
107          // Obtener el elemento raíz <Empleados> creado previamente
108          Element elementoRaiz = documento.getDocumentElement();
```

### 2. Lectura del archivo binario (empleados.dat):

- Lees el archivo secuencialmente utilizando RandomAccessFile, lo que te permite manejar el acceso aleatorio y leer los campos individuales en el orden esperado.
- Los datos de cada empleado se leen y se asignan a nodos XML:
  - <codigo>
  - <nombre>
  - <direccion>
  - <salario>
  - <comision>

### 3. Estructuración del XML:

- Cada empleado se encapsula en un nodo <empleado>, y los campos son sus hijos directos.
- Al final de cada iteración, el nodo <empleado> se añade al nodo raíz <empleados>.

```
// Leer los datos desde el archivo binario "empleados.dat"
try (RandomAccessFile rf = new RandomAccessFile("empleados.dat", "r")) {

    // Mientras no se haya alcanzado el final del archivo, leer los datos de los empleados
    while (rf.getFilePointer() < rf.length()) {
        // Crear un nuevo elemento <Empleado> que representará un empleado en el XML
        Element elementoEmpleado = documento.createElement("Empleado");
        elementoRaiz.appendChild(elementoEmpleado); // Añadir el elemento <Empleado> al elemento raíz <Empleados>

        // Leer el código del empleado y añadirlo como un subelemento <Código>
        Element elementoCodigo = documento.createElement("Código"); // Crear el elemento <Código>
        int codigo = rf.readInt(); // Leer el código del archivo binario (tipo int)
        Text textoCodigo = documento.createTextNode(String.valueOf(codigo)); // Crear un nodo de texto con el valor del código
        elementoCodigo.appendChild(textoCodigo); // Añadir el texto al elemento <Código>
        elementoEmpleado.appendChild(elementoCodigo); // Añadir <Código> al <Empleado>

        // Leer el nombre del empleado (20 caracteres) y añadirlo como un subelemento <Nombre>
        Element elementoNombre = documento.createElement("Nombre"); // Crear el elemento <Nombre>
        String nombre = "";
        for (int i = 0; i < 20; i++) {
            nombre += rf.readChar(); // Leer cada carácter y construir el nombre
        }
        Text textoNombre = documento.createTextNode(nombre.trim()); // Eliminar espacios en blanco y crear el nodo de texto
        elementoNombre.appendChild(textoNombre); // Añadir el texto al elemento <Nombre>
        elementoEmpleado.appendChild(elementoNombre); // Añadir <Nombre> al <Empleado>

        // Leer la dirección del empleado (20 caracteres) y añadirla como un subelemento <Dirección>
        Element elementoDireccion = documento.createElement("Dirección"); // Crear el elemento <Dirección>
        String direccion = "";
        for (int i = 0; i < 20; i++) {
            direccion += rf.readChar(); // Leer cada carácter y construir la dirección
        }
        Text textoDireccion = documento.createTextNode(direccion.trim()); // Eliminar espacios en blanco y crear el nodo de texto
        elementoDireccion.appendChild(textoDireccion); // Añadir el texto al elemento <Dirección>
        elementoEmpleado.appendChild(elementoDireccion); // Añadir <Dirección> al <Empleado>

        // Leer el salario del empleado y añadirlo como un subelemento <Salario>
        Element elementoSalario = documento.createElement("Salario"); // Crear el elemento <Salario>
        double salario = rf.readDouble(); // Leer el salario del archivo binario (tipo double)
        Text textoSalario = documento.createTextNode(String.valueOf(salario)); // Crear un nodo de texto con el valor del salario
        elementoSalario.appendChild(textoSalario); // Añadir el texto al elemento <Salario>
        elementoEmpleado.appendChild(elementoSalario); // Añadir <Salario> al <Empleado>

        // Leer la comisión del empleado y añadirla como un subelemento <Comisión>
        Element elementoComision = documento.createElement("Comisión"); // Crear el elemento <Comisión>
        double comision = rf.readDouble(); // Leer la comisión del archivo binario (tipo double)
        Text textoComision = documento.createTextNode(String.valueOf(comision)); // Crear un nodo de texto con el valor de la comisión
        elementoComision.appendChild(textoComision); // Añadir el texto al elemento <Comisión>
        elementoEmpleado.appendChild(elementoComision); // Añadir <Comisión> al <Empleado>
    }
}

} catch (FileNotFoundException ex) {
    // Manejar el caso en que el archivo binario no exista
    Logger.getLogger(Parte1.class.getName()).log(Level.SEVERE, null, ex);
} catch (IOException ex) {
    // Manejar errores de entrada/salida al leer el archivo binario
    Logger.getLogger(Parte1.class.getName()).log(Level.SEVERE, null, ex);
}
}
```

#### 4. Escritura del XML:

- Utilizas Transformer para transformar el documento DOM en un archivo físico empleados.xml.

```
// Guardar el documento XML generado en un archivo físico "empleados.xml"
TransformerFactory tf = TransformerFactory.newInstance(); // Crear una fábrica de transformadores
Transformer transformer = tf.newTransformer(); // Crear un transformador

// Crear el origen (documento DOM) y el destino (archivo XML)
DOMSource source = new DOMSource(documento);
StreamResult result = new StreamResult(new File("empleados.xml"));

// Realizar la transformación y guardar el XML en el archivo
transformer.transform(source, result);

} catch (ParserConfigurationException | TransformerConfigurationException ex) {
    // Manejar errores relacionados con la configuración del parser o el transformador
    Logger.getLogger(Parte1.class.getName()).log(Level.SEVERE, null, ex);
} catch (TransformerException ex) {
    // Manejar errores al transformar y guardar el documento XML
    Logger.getLogger(Parte1.class.getName()).log(Level.SEVERE, null, ex);
}
}
```

### Invocación del método:

- Llamas a CrearXML() al final del main para ejecutar el proceso después de crear el archivo empleados.dat.

```
crearXML();
```

### EJERCICIO 3. Visualizar todas las etiquetas del fichero LIBROS.XML utilizando las técnicas DOM.

```
▼ <libros>
  ▼ <libro año="1994">
    <titulo>TCP/IP Illustrated</titulo>
    ▼ <autor>
      <apellido>Stevens</apellido>
      <nombre>W.</nombre>
    </autor>
    <editorial>Addison-Wesley</editorial>
    <precio> 65.95</precio>
  </libro>
  ▼ <libro año="1992">
    <titulo>Advan Programming for Unix environment</titulo>
    ▼ <autor>
      <apellido>Stevens</apellido>
      <nombre>W.</nombre>
    </autor>
    <editorial>Addison-Wesley</editorial>
    <precio>65.95</precio>
  </libro>
  ▼ <libro año="2000">
    <titulo>Data on the Web</titulo>
    ▼ <autor>
      <apellido>Abiteboul</apellido>
      <nombre>Serge</nombre>
    </autor>
    ▼ <autor>
      <apellido>Buneman</apellido>
      <nombre>Peter</nombre>
    </autor>
    ▼ <autor>
      <apellido>Suciu</apellido>
      <nombre>Dan</nombre>
    </autor>
    <editorial>Morgan Kaufmann editorials</editorial>
    <precio>39.95</precio>
  </libro>
</libros>
```

## Paso 1: Inicializar el Parser DOM

1. Se creó un objeto `DocumentBuilderFactory` para generar un parser DOM.
2. Se utilizó `DocumentBuilder` para construir un objeto `Document` a partir del archivo `libros.xml`.

```
package com.mycompany.ejercicio2ad;

//En este apartado he procedido en primer lugar a
import java.io.File;
import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

/**
 * La clase Parte2DOM se encarga de procesar un archivo XML llamado
 * "libros.xml", mostrando en consola la estructura del documento. Utiliza la
 * API DOM para trabajar con el archivo, imprimiendo el elemento raíz, los
 * atributos de cada elemento "libro" y las propiedades de sus nodos hijos.
 */
public class Parte2DOM {

    /**
     * Método principal que ejecuta el programa. Lee el archivo "libros.xml", lo
     * analiza y muestra en consola su contenido.
     *
     * @param args Argumentos de línea de comandos (no utilizados en este
     * programa).
     */
    public static void main(String[] args) {

        try {
            /**
             * Obtener una Instancia de DocumentBuilderFactory: Esta clase se
             * utiliza para crear objetos DocumentBuilder, necesarios para
             * analizar el archivo XML y se utiliza el método
             * newDocumentBuilder() para inicializar un parser DOM::
             */
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder builder = factory.newDocumentBuilder();

            //El archivo libros.xml se convierte en un objeto Document para su manipulación:
            Document documento = builder.parse(new File("libros.xml"));
```

## Paso 2: Normalizar el Documento

El documento XML fue normalizado para eliminar nodos redundantes, facilitando la manipulación de datos.

```
//Se utiliza el método normalize() para limpiar el árbol de nodos XML, eliminando redundancias como nodos de texto innecesarios:
documento.getDocumentElement().normalize();
```

### Paso 3: Procesar el Elemento Raíz

El elemento raíz (<libros>) se extrajo utilizando `getDocumentElement()` y se imprimió su nombre en consola.

```
//El elemento raíz del archivo XML (<libros> en este caso) se obtiene con getDocumentElement() y se muestra en consola:  
Element elementoRaiz = (Element) documento.getDocumentElement();  
System.out.println("El elemento raíz es: " + elementoRaiz.getNodeName());
```

### Paso 4: Obtener y Recorrer Elementos <libro>

1. Se obtuvo una lista de nodos <libro> mediante `getElementsByTagName("libro")`.
2. Cada nodo <libro> se recorrió con un bucle `for`.
3. Se verificó si cada nodo <libro> tenía atributos y, de ser así, se recorrieron e imprimieron en consola.
4. Se obtuvieron los nodos hijos de <libro> utilizando `getChildNodes()`.
5. Cada hijo fue procesado:
  - Caso general: Se imprimió el nombre y el contenido del nodo.
  - Caso especial (<autor>): Si el nodo era <autor>, se recorrieron sus subnodos (<nombre>, <apellido>, etc.) y se imprimieron sus propiedades.

```
//Iterar Sobre Cada Nodo <libro>: Un bucle for recorre cada nodo de la lista:  
for (int i = 0; i < listaLibros.getLength(); i++) {  
    // Obtener el nodo actual de la lista  
    Node nodo = listaLibros.item(i);  
  
    //Si el nodo actual tiene atributos, se recorren e imprimen:  
    if (nodo.hasAttributes()) {  
        for (int j = 0; j < nodo.getAttributes().getLength(); j++) {  
            System.out.println("Atributo: " + nodo.getAttributes().item(j).getNodeName()  
                               + ", Valor: " + nodo.getAttributes().item(j).getTextContent());  
        }  
    }  
  
    // Verificar que el nodo es un elemento (tipo ELEMENT_NODE)  
    if (nodo.getNodeType() == Node.ELEMENT_NODE) {  
        Element libro = (Element) nodo;  
  
        // Obtener los hijos del elemento <libro>  
        NodeList hijosLibro = libro.getChildNodes();  
        for (int j = 0; j < hijosLibro.getLength(); j++) {  
            Node hijoLibro = hijosLibro.item(j);  
  
            // Verificar que el hijo es un elemento  
            if (hijoLibro.getNodeType() == Node.ELEMENT_NODE) {  
                // Caso especial: procesar el nodo <autor> con subelementos  
                if (hijoLibro.getNodeName().equals("autor")) {  
                    Element e2 = (Element) hijoLibro;  
                    NodeList hijosAutor = e2.getChildNodes();  
                    for (int k = 0; k < hijosAutor.getLength(); k++) {  
                        Node hijoAutor = hijosAutor.item(k);  
                        if (hijoAutor.getNodeType() == Node.ELEMENT_NODE) {  
                            System.out.println("\tPropiedad: " + hijoAutor.getNodeName() + ", Valor: "  
                                               + hijoAutor.getTextContent());  
                        }  
                    }  
                }  
                // Caso general: imprimir el nombre del nodo y su valor  
            } else {  
                System.out.println("Propiedad: " + hijoLibro.getNodeName() + ", Valor: "  
                                   + hijoLibro.getTextContent());  
            }  
        }  
    }  
}
```



## Paso 5: Manejo de Excepciones

Se implementó un bloque try-catch para capturar errores relacionados con:

- Configuración del parser.
- Errores en el formato del archivo XML.
- Problemas de lectura/escritura del archivo.

```
        }  
    } catch (ParserConfigurationException | SAXException | IOException ex) {  
        Logger.getLogger(Parte2DOM.class.getName()).log(Level.SEVERE, null, ex);  
    }  
}  
}
```

## Paso 6: Organización de la Salida

Se agregó un salto de línea después de procesar cada libro para mejorar la legibilidad de la salida.

```
        System.out.println("Propiedad: " + hijoLibro.getTagName() + ", Valor: " +  
            + hijoLibro.getTextContent());  
    }  
}  
// Separar visualmente los libros en la salida  
System.out.println("");  
}  
}  
} catch (ParserConfigurationException | SAXException | IOException ex) {  
    Logger.getLogger(Parte2DOM.class.getName()).log(Level.SEVERE, null, ex);  
}  
}
```

Salida:

```
El elemento raiz es: libros  
Atributo: a❖o, Valor: 1994  
Propiedad: titulo, Valor: TCP/IP Illustrated  
    Propiedad: apellido, Valor: Stevens  
    Propiedad: nombre, Valor: W.  
Propiedad: editorial, Valor: Addison-Wesley  
Propiedad: precio, Valor: 65.95  
  
Atributo: a❖o, Valor: 1992  
Propiedad: titulo, Valor: Advan Programming for Unix environment  
    Propiedad: apellido, Valor: Stevens  
    Propiedad: nombre, Valor: W.  
Propiedad: editorial, Valor: Addison-Wesley  
Propiedad: precio, Valor: 65.95  
  
Atributo: a❖o, Valor: 2000  
Propiedad: titulo, Valor: Data on the Web  
    Propiedad: apellido, Valor: Abiteboul  
    Propiedad: nombre, Valor: Serge  
    Propiedad: apellido, Valor: Buneman  
    Propiedad: nombre, Valor: Peter  
    Propiedad: apellido, Valor: Suciu  
    Propiedad: nombre, Valor: Dan  
Propiedad: editorial, Valor: Morgan Kaufmann editorials  
Propiedad: precio, Valor: 39.95
```

## EJERCICIO 3. Visualizar todas las etiquetas del fichero LIBROS.XML utilizando las técnicas SAX.

### Clase Parte2SAX

#### 1. Importación de Librerías

- Se importaron las clases necesarias para configurar el parser SAX (SAXParserFactory, SAXParser) y manejar excepciones específicas (SAXException, IOException).

#### 2. Configuración del Parser

- Se utilizó SAXParserFactory para crear una instancia de SAXParser.
- Se instanció un objeto LibroHandler para delegar el manejo de los eventos SAX.

#### 3. Procesamiento del Archivo XML

- El método parse del parser procesa el archivo libros.xml utilizando el handler.

#### 4. Manejo de Excepciones

- Se implementó un bloque try-catch para capturar errores relacionados con la configuración del parser, la estructura del archivo o problemas de entrada/salida.

```
package com.mycompany.ejercicio2ad;

import java.io.IOException;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.SAXException;

/**
 * La clase Parte2SAX utiliza un parser SAX para leer y procesar el archivo
 * XML "libros.xml". Este enfoque procesa el archivo de forma secuencial,
 * invocando métodos específicos según los eventos encontrados en el documento.
 */
public class Parte2SAX {

    /**
     * Método principal que ejecuta el programa.
     * Configura el parser SAX y delega el procesamiento al handler LibroHandler.
     *
     * @param args Argumentos de línea de comandos (no utilizados en este programa).
     */
    public static void main(String[] args) {

        try {
            // Crear una instancia de SAXParserFactory
            SAXParserFactory factory = SAXParserFactory.newInstance();

            // Crear un objeto SAXParser
            SAXParser parser = factory.newSAXParser();

            // Crear una instancia de LibroHandler para manejar los eventos
            LibroHandler handler = new LibroHandler();

            // Parsear el archivo XML utilizando el handler
            parser.parse("libros.xml", handler);

        } catch (ParserConfigurationException | SAXException | IOException ex) {
            // Manejar excepciones relacionadas con el parser SAX
            Logger.getLogger(Parte2SAX.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
}
```

## Clase LibroHandler

### 1. Extensión de DefaultHandler

- La clase extiende DefaultHandler y sobrescribe los métodos startElement, characters y endElement.

### 2. Manejo de Eventos SAX

- startElement: Se procesa el inicio de cada elemento, incluyendo sus atributos. En el caso de <libro>, se imprime el atributo año.
- characters: Se acumula el contenido del texto en un StringBuilder para su procesamiento posterior.
- endElement: Según el nombre del elemento (qName), se imprime su contenido al finalizar la lectura del elemento.

### 3. Buffer de Texto

- Se utiliza un StringBuilder (valor) para acumular el contenido de texto entre etiquetas y evitar problemas con caracteres fragmentados.

```
package com.mycompany.ejercicio2ad;

import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

/**
 * La clase LibroHandler extiende DefaultHandler y se encarga de manejar los
 * eventos SAX al procesar un archivo XML. Sobrescribe métodos para actuar
 * en el inicio y final de elementos, así como para procesar el contenido de texto.
 */
public class LibroHandler extends DefaultHandler {

    // Buffer para acumular el contenido de texto de los nodos
    private StringBuilder valor;

    /**
     * Constructor de la clase. Inicializa el buffer para procesar el texto.
     */
    public LibroHandler() {
        this.valor = new StringBuilder();
    }
}
```

```

/**
 * Método invocado al inicio de un elemento XML.
 *
 * @param uri El namespace del elemento (si aplica).
 * @param localName El nombre local del elemento (sin prefijo).
 * @param qName El nombre calificado del elemento (incluyendo prefijo).
 * @param attributes Los atributos asociados al elemento.
 * @throws SAXException Si ocurre un error durante el manejo del evento.
 */
@Override
public void startElement(String uri, String localName,
    String qName, Attributes attributes)
    throws SAXException {

    // Reiniciar el buffer de texto
    this.valor.setLength(0);

    // Procesar atributos del elemento <libro>
    if (qName.equals("libro")) {
        String anio = attributes.getValue("año");
        System.out.println("Atributo año: " + anio);
    }
}

/**
 * Método invocado al leer el contenido de texto entre elementos.
 *
 * @param ch Array de caracteres del contenido.
 * @param start Índice de inicio del contenido en el array.
 * @param length Longitud del contenido a procesar.
 * @throws SAXException Si ocurre un error durante el manejo del evento.
 */
@Override
public void characters(char ch[], int start, int length)
    throws SAXException {
    // Acumular el contenido del texto en el buffer
    this.valor.append(ch, start, length);
}

```

```

/**
 * Método invocado al final de un elemento XML.
 *
 * @param uri El namespace del elemento (si aplica).
 * @param localName El nombre local del elemento (sin prefijo).
 * @param qName El nombre calificado del elemento (incluyendo prefijo).
 * @throws SAXException Si ocurre un error durante el manejo del evento.
 */
@Override
public void endElement(String uri, String localName, String qName)
    throws SAXException {

    // Procesar el contenido al final de cada elemento
    switch (qName) {
        case "libro":
            System.out.println("");
            break;
        case "titulo":
            System.out.println("Título: " + this.valor.toString());
            break;
        case "apellido":
            System.out.println("Apellido autor: " + this.valor.toString());
            break;
        case "nombre":
            System.out.println("Nombre autor: " + this.valor.toString());
            break;
        case "editorial":
            System.out.println("Editorial: " + this.valor.toString());
            break;
        case "precio":
            System.out.println("Precio: " + this.valor.toString());
            break;
    }
}

```