

Acceso a Datos.

Tarea para AD02.

Apartado 1) Realiza las siguientes acciones utilizando NetBeans:

- ✓ 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).

Para la resolución de esta parte de la tarea, he generado un proyecto con la clase principal, llamada `Empleados_aleatorio.java` y una clase dentro del mismo paquete, llamada `Empleados.java`, donde recoger la estructura funcional de los datos de cada empleado.

Además de lo que se argumente aquí, he intentado que quede suficientemente documentado el código y sus capturas para que pueda ser entendido de la mejor manera posible.

Empleados.java

```
/*
 * TAREA AD02.EJERCICIO 1.
 * 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).
 * 2- A partir de los datos del fichero EMPLEADOS.DAT crear un fichero
 * llamado EMPLEADOS.XML usando DOM.
 *
 * La clase Empleados determina los campos a recoger y el acceso a sus datos.
 */
package empleados_aleatorio;

/**
 *
 * @author juang <juangmuelas@gmail.com>
 * @since 11/11/2021
 */
public class Empleados {
    //Declaramos las variables solicitadas en el ejercicio.
    private int codigo;
    private String nombre, direccion;
    private float salario, comision;

    //Constructor
    public Empleados(int codigo, String nombre, String direccion,
        float salario, float comision) {

        this.codigo = codigo;
        this.nombre = nombre;
        this.direccion = direccion;
        this.salario = salario;
        this.comision = comision;
    }

    //getter y setter (aunque para este caso nos llegaría con getters
    public int getCodigo() {
        return codigo;
    }

    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }

    public String getNombre() {
        return nombre;
    }
}
```

```

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getDireccion() {
        return direccion;
    }

    public void setDireccion(String direccion) {
        this.direccion = direccion;
    }

    public float getSalario() {
        return salario;
    }

    public void setSalario(float salario) {
        this.salario = salario;
    }

    public float getComision() {
        return comision;
    }

    public void setComision(float comision) {
        this.comision = comision;
    }
} //Fin clase Empleados

```

Empleados_aleatorio.java (primera parte del ejercicio)

```

/*
 * TAREA AD02.EJERCICIO 1.
 * 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).
 * 2- A partir de los datos del fichero EMPLEADOS.DAT crear un fichero
 * llamado EMPLEADOS.XML usando DOM.
 */
package empleados_aleatorio;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.ArrayList;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.transform.Result;
import javax.xml.transform.Source;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import org.w3c.dom.DOMImplementation;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Text;

/**
 *
 * @author juang <juangmuelas@gmail.com>
 * @since 11/11/2021
 */
public class Empleados_aleatorio {

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

```

```

/**
 * PARTE 1.
 * @param empleados Array de datos para empleados.
 * Se ha generado una clase Empleados.java para operar con sus datos.
 */
ArrayList<Empleados> empleados = new ArrayList<>();

/**
 * Añadimos los datos para 5 empleados.
 */
empleados.add(new Empleados(1,"Antonio Perez","Av Principal 9",
985.12f,11));
empleados.add(new Empleados(2,"Manuel Perez","Av Principal 9",
1102.23f,15));
empleados.add(new Empleados(3,"Ana Perez","Plaza Ayuntamiento 8",
985.12f,11));
empleados.add(new Empleados(4,"María Perez","C Los Obeliscos 5",
1102.23f,15));
empleados.add(new Empleados(5,"Carla Perez","Av Mar 21", 1114.89f,5));

/**
 * Creamos nuestro archivo EMPLEADOS.dat desde un try-catch
 * que envuelve a nuestro randomAccessFile.
 * Haciéndolo así hacemos un cerrado automático.
 * Valores en bytes a recordar:
 * double (8 bytes), long (8 bytes),
 * int (4 bytes), float(4 bytes),
 * char (2 bytes), short (2 bytes),
 * byte (1 byte), boolean (1 bit)
 */
try (RandomAccessFile raf = new RandomAccessFile("EMPLEADOS.dat",
"rw")) {
    //recorremos los array
    for( Empleados e : empleados){
        raf.writeInt(e.getCodigo()); //4 bytes
        //Para un correcto acceso aleatorio, limitamos los campos
        StringBuffer sb = new StringBuffer(e.getNombre()); //40 bytes
        sb.setLength(20);
        raf.writeChars(sb.toString());
        StringBuffer sbD = new StringBuffer(e.getDireccion());
        sbD.setLength(20);
        raf.writeChars(sbD.toString());
        raf.writeFloat(e.getSalario()); //4 bytes
        raf.writeFloat(e.getComision()); //4 bytes
    }

    /**
     * La suma de los campos nos da 92 bytes por registro.
     * Realizo una prueba para comprobar el correcto acceso.
     */
    raf.seek(92); //posicion 2
    System.out.println("Datos de posición " + raf.readInt() + ".");
    /**
     * Para los Strings se realiza un for que recorre los bytes
     * delimitados, sumando sus caracteres
     */
    String nombre = "";
    for(int i = 0; i < 20; i++){
        nombre += raf.readChar();
    }
    String direccion = "";
    for(int i = 0; i < 20; i++){
        direccion += raf.readChar();
    }

    System.out.println(nombre);
    System.out.println(direccion);
    System.out.println(raf.readFloat());
}

```

```

        System.out.println(raf.readFloat());

        System.out.println("Datos de posición mostrados.
        \n*****");

    } catch (FileNotFoundException e) {
        System.out.println(e.getMessage());
    } catch (IOException ex) {
        System.out.println(ex.getMessage());
        System.err.println("\nError de escritura\n");
    } //fin bloque try-catch para EMPLEADOS.dat

```

Como puede observarse en la captura del código, la resolución de este apartado se basa en la creación de un ArrayList que recoge los datos para 5 empleados.

A continuación, accedemos a ellos de forma aleatoria mediante el uso de la clase [RandomAccessFile](#), creando también en ese momento el fichero [EMPLEADOS.dat](#). Todo ello lo vamos a envolver en bloque try-catch, con el que aseguramos la recogida de posibles excepciones, y que nos permite así, desde esa cabecera, inicializarlo apuntando al principio del fichero y cerrarlo de forma automática al acabar, optimizando así recursos.

Recorremos mediante un bucle for los arrays, recogiendo los datos y prestando atención a los Strings, que para evitar errores recogemos sus caracteres mediante un objeto [StringBuffer](#), parseando el resultado de nuevo a String.

Si bien, con esto ya tendríamos el archivo pedido, realizo la comprobación por consola, para estar seguro en tiempo de ejecución.

```

89         raf.seek(92); //posicion 2
90         System.out.println("Datos de posición " + raf.readInt() + ".");
91         /**
92          * Para los Strings se realiza un for que recorre los bytes
93          * delimitados, sumando sus caracteres
94          */
95         String nombre = "";
96         for(int i = 0; i < 20; i++){
97             nombre += raf.readChar();
98         }
99         String direccion = "";
100        for(int i = 0; i < 20; i++){
101            direccion += raf.readChar();
102        }
103
104        System.out.println(nombre);
105        System.out.println(direccion);
106        System.out.println(raf.readFloat());
107        System.out.println(raf.readFloat());
108
109        System.out.println("Datos de posición mostrados. \n*****");
110
111    } catch (FileNotFoundException e) {
112        System.out.println(e.getMessage());
113    } catch (IOException ex) {
114        System.out.println(ex.getMessage());
115        System.err.println("\nError de escritura\n");
116    } //fin bloque try-catch para EMPLEADOS.dat

```

empleados_aleatorio.Empleados_aleatorio > main > try >

Salida - Empleados_aleatorio (run) x

```

run:
Datos de posición 2.
Manuel Perez
Av Principal 9
1102.23
15.0
Datos de posición mostrados.
*****

```

- ✓ A partir de los datos del fichero EMPLEADOS.DAT crear un fichero llamado EMPLEADOS.XML usando DOM.

Para esta segunda parte, también he intentado documentarla de la forma más extensa posible para que la captura pueda ser suficientemente relevante como para comprender su ejecución.

Empleados_aleatorio.java (segunda parte del ejercicio y continuando en el main)

```
/**
 * PARTE 2.
 * CREANDO XML usando DOM.
 * Declaramos las variables necesarias.
 * @param posicion integer para posicionar nuestro puntero.
 * Los siguientes parametros recogen los datos de empleados.
 * @param codigo integer.
 * @param nombre String.
 * @param direccion String.
 * @param salario float.
 * @param comision float.
 * @param nombreN array que recoge los caracteres del campo nombre.
 * @param direccionD array que recoge los caracteres del campo
 * direccion.
 */

int posicion=0;
int codigo;
String nombre="";
String direccion="";
float salario;
float comision;
char nombreN[] = new char[20], aux;
char direccionD[] = new char[20], aux1;
/**
 * Se crea una instancia DocumentBuilderFactory, construyendo el
 * parser en un try-catch, donde accedemos al archivo EMPLEADOS.dat,
 * igual que hicimos en el punto anterior, pero en este caso,
 * solo para lectura.
 */
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
try (RandomAccessFile raf = new RandomAccessFile("EMPLEADOS.dat",
"r")) {
    DocumentBuilder builder = factory.newDocumentBuilder();
    //Creamos un documento con el nodo raiz "Empleados".
    DOMImplementation implementation = builder.getDOMImplementation();
    Document document = implementation.createDocument(null,
"Empleados", null);
    document.setXmlVersion("1.0"); //Asignamos versión XML

    /**
     * Para la recogida de datos, uso casi la misma estructura
     * utilizada en la parte anterior.
     * Se envuelve todo en un for que recorre mientras encuentre datos
     */
    for(;;){
        raf.seek(posicion); //nos posicionamos
        codigo = raf.readInt(); //obtenemos codigo empleado
        for(int i = 0; i < 20; i++){
            aux = raf.readChar();
            nombreN[i]=aux;
            nombre = new String(nombreN);
        }
        for(int i = 0; i < 20; i++){
            aux1 = raf.readChar();
            direccionD[i]=aux1;
            direccion = new String(direccionD);
        }
    }
}
```

```

        salario = raf.readFloat();
        comision = raf.readFloat();

        //los muestro en consola para seguir la correcta toma de datos
        System.out.println("posicion "+ posicion);
        System.out.println("CODIGO: " + codigo + ", NOMBRE: "
            + nombre + ", DIRECCION: " + direccion + ", SALARIO: "
            + salario + ", COMISION: " + comision);

        //guardo la siguiente posicion
        posicion +=32;

        /**
         * Recogidos los datos, creamos cada nodo.
         * Para ello, se crea la función CrearElemento();
         */
        if(codigo>0){
            Element raiz = document.createElement("empleado"); //nodo
            empleado
            document.getDocumentElement().appendChild(raiz); //lo
            pegamos a la raiz
            CrearElemento("codigo",Integer.toString(codigo), raiz,
            document); //añadir codigo
            CrearElemento("nombre",nombre.trim() , raiz, document);
            //nombre
            CrearElemento("direccion",direccion.trim() , raiz,
            document); //direccion
            CrearElemento("salario",Float.toString(salario), raiz,
            document); //salario
            CrearElemento("comision",Float.toString(comision), raiz,
            document); //comision
        }
        posicion = posicion + 60;
        if (raf.getFilePointer() == raf.length() ) break;
    } //fin bucle for

    //Creamos la fuente XML a partir del documento
    Source source = new DOMSource(document);
    Result result = new StreamResult(new
    java.io.File("EMPLEADOS.xml"));
    Transformer transformer =
    TransformerFactory.newInstance().newTransformer();
    transformer.transform(source, result);

    //mostrar doc por consola
    Result console = new StreamResult(System.out);
    transformer.transform(source, console);

    } catch (Exception e) {
        System.err.println("Error: " + e );
    } //Fin try-catch para generar el XML
} //fin main

//Inserción de los datos del empleado
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
}

} //fin clase Empleados_aleatorio

```

Vemos como en este segundo punto se utiliza una forma similar para acceder al archivo [EMPLEADOS.dat](#), solo que lo ponemos como para acceder en modo de lectura tras generar la instancia para el [DocumentBuilderFactory](#).

Lo siguiente es generar el documento, con el elemento raíz y darle la versión de XML a utilizar.

Recogemos los datos mediante un bucle for que leerá mientras encuentre datos.

Prestamos también especial atención a los Strings, para evitar errores de lectura mediante el escaneo de sus caracteres y pasándolos a Strings luego.

Para asegurar que los datos van a llegar de forma correcta al nuevo documento, mostramos los datos por la consola, prestando atención a que el puntero se posiciona en el lugar correcto.

```
posicion 0
CODIGO: 1, NOMBRE: Antonio Perez, DIRECCION: Av Principal 9, SALARIO: 985.12, COMISION: 11.0
posicion 92
CODIGO: 2, NOMBRE: Manuel Perez, DIRECCION: Av Principal 9, SALARIO: 1102.23, COMISION: 15.0
posicion 184
CODIGO: 3, NOMBRE: Ana Perez, DIRECCION: Plaza Ayuntamiento 8, SALARIO: 985.12, COMISION: 11.0
posicion 276
CODIGO: 4, NOMBRE: Maria Perez, DIRECCION: C Los Obeliscos 5, SALARIO: 1102.23, COMISION: 15.0
posicion 368
CODIGO: 5, NOMBRE: Carla Perez, DIRECCION: Av Mar 21, SALARIO: 1114.89, COMISION: 5.0
```

Tras ello, creamos un condicional donde recorrer los códigos e ir creando cada elemento de nuestro XML, mediante una función que creamos tras nuestro main, y que se encarga de crear y recoger los valores de cada nodo y pegarlos al elemento raíz.

Por último, creamos la fuente XML para nuestro documento y creamos el archivo [EMPLEADOS.xml](#).

Para asegurarnos en ejecución, lo mostramos por consola:

```
204 //Creamos la fuente XML a partir del documento
205 Source source = new DOMSource(document);
206 Result result = new StreamResult(new java.io.File("EMPLEADOS.xml"));
207 Transformer transformer = TransformerFactory.newInstance().newTransformer();
208 transformer.transform(source, result);
209
210 //mostrar doc por consola
211 Result console = new StreamResult(System.out);
212 transformer.transform(source, console);
213
214 } catch (Exception e) {
215     System.err.println("Error: " + e);
216 } //Fin try-catch para generar el XML
217 } //fin main
218
```

empleados_aleatorio.Empleados_aleatorio > main > try >

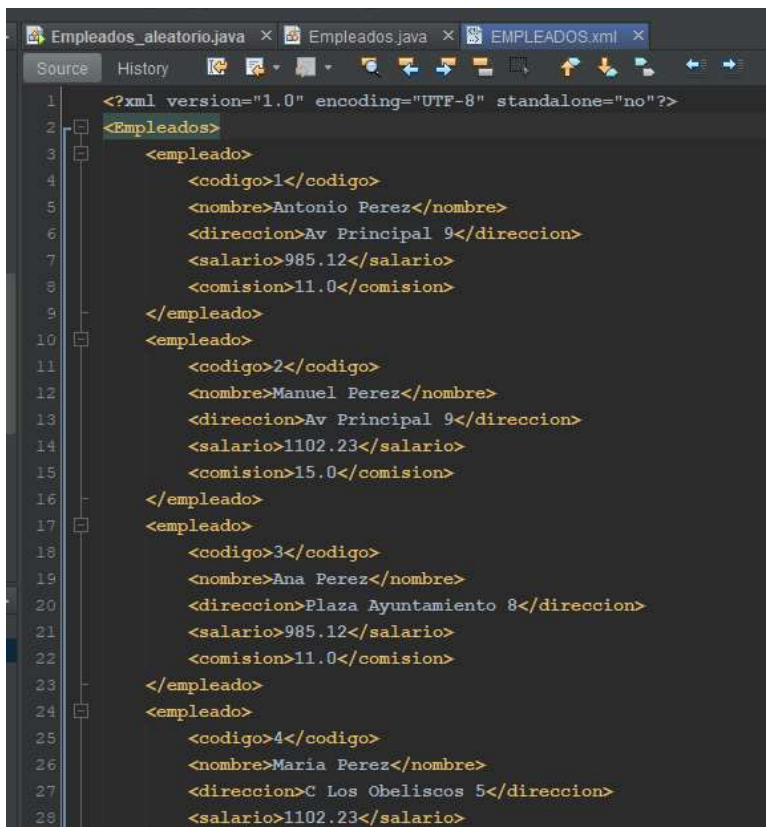
Salida - Empleados_aleatorio (run) x

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><Empleados><empleado><codigo>1</codigo><nombre>Antonio Perez</nombre><direcc
```

Si revisamos el directorio de nuestro proyecto, podemos ver los archivos creados:

Nombre	Fecha de modificación	Tipo
build	11/11/2021 13:39	Carpeta de archivos
nbproject	11/11/2021 12:41	Carpeta de archivos
src	11/11/2021 12:41	Carpeta de archivos
test	11/11/2021 12:43	Carpeta de archivos
build	11/11/2021 12:41	Documento XML
EMPLEADOS.dat	18/11/2021 12:05	Archivo DAT
EMPLEADOS	18/11/2021 12:45	Documento XML
manifest.mf	11/11/2021 12:41	Archivo MF

Y también visualizar nuestro xml a través de NetBeans:



Apartado 2) 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.

Fichero de trabajo (Libros.xml)

Este apartado precisa de crear un nuevo proyecto para mantener un orden y mejor presentación de lo que pretendemos mostrar, y en el he añadido el archivo facilitado para la tarea "Libros.xml".

He creado una clase principal `ParserXML.java`, donde se recorre dicho fichero mediante las técnicas DOM, en primer término, y SAX.

ParserXML.java (DOM)

```
/*
 * TAREA AD02.EJERCICIO 2.
 * Visualizar todas las etiquetas del fichero LIBROS.XML
 * utilizando las técnicas DOM y SAX.
 */
package parserxml;

import java.io.File;
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
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;
import org.xml.sax.helpers.DefaultHandler;

/**
 *
 * @author juang <juangmuelas@gmail.com>
 * @since 11/11/2021
 */
public class ParserXML {

    /**
     * Esta primera parte aprovecha el ejemplo mostrado en el punto 6.4.2
     * adaptándolo a nuestro fichero libros.xml, que hemos añadido a
     * nuestro directorio, y así mostrar los datos mediante el DOM
     */
    public static void main(String[] args) throws ParserConfigurationException
    {
        try {
            File inputFile = new File("Libros.xml");
            DocumentBuilderFactory dbFactory =
                DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document doc = dBuilder.parse(inputFile);
            doc.getDocumentElement().normalize();
            System.out.println("ELEMENTO RAIZ : " +
                doc.getDocumentElement().getNodeName()); //retorna el elemento raiz
            NodeList nList = doc.getElementsByTagName("libro");

            System.out.println("-----");
            for (int temp = 0; temp < nList.getLength(); temp++) {
                Node nNode = nList.item(temp);
                System.out.println("\nElemento hijo: " + nNode.getNodeName());
                if (nNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element eElement = (Element) nNode;
                    NodeList autorList = eElement.getElementsByTagName("autor");
                    System.out.println("Año : " + eElement.getAttribute("año"));
                    System.out.println("Titulo : " +
                        eElement.getElementsByTagName("titulo").item(0).getTextContent());

                    /**
                     * Al tener varios elementos autor en la última tag, se
                     * crea este nuevo for para recorrerlo, y se apunta al
                     * item a mostrar según lo encuentre.
                     */
                    for(int cont=0; cont < autorList.getLength();cont++){
                        Node node1 = autorList.item(cont);
                        if (node1.getNodeType() == node1.ELEMENT_NODE) {
                            System.out.println("Autor:");
                            System.out.println("Nombre:" +
                                eElement.getElementsByTagName("nombre").item(cont).getTextContent());
                            System.out.println("Apellido:" +
                                eElement.getElementsByTagName("apellido").item(cont).getTextContent());
                        }
                        System.out.println("Editorial : " +
                            eElement.getElementsByTagName("editorial").item(0).getTextContent());
                        System.out.println("Precio : " +
                            eElement.getElementsByTagName("precio").item(0).getTextContent());
                    }
                }
            } catch (Exception e) {
                e.printStackTrace();
            } //fin try-catch DOM
        }
    }
}

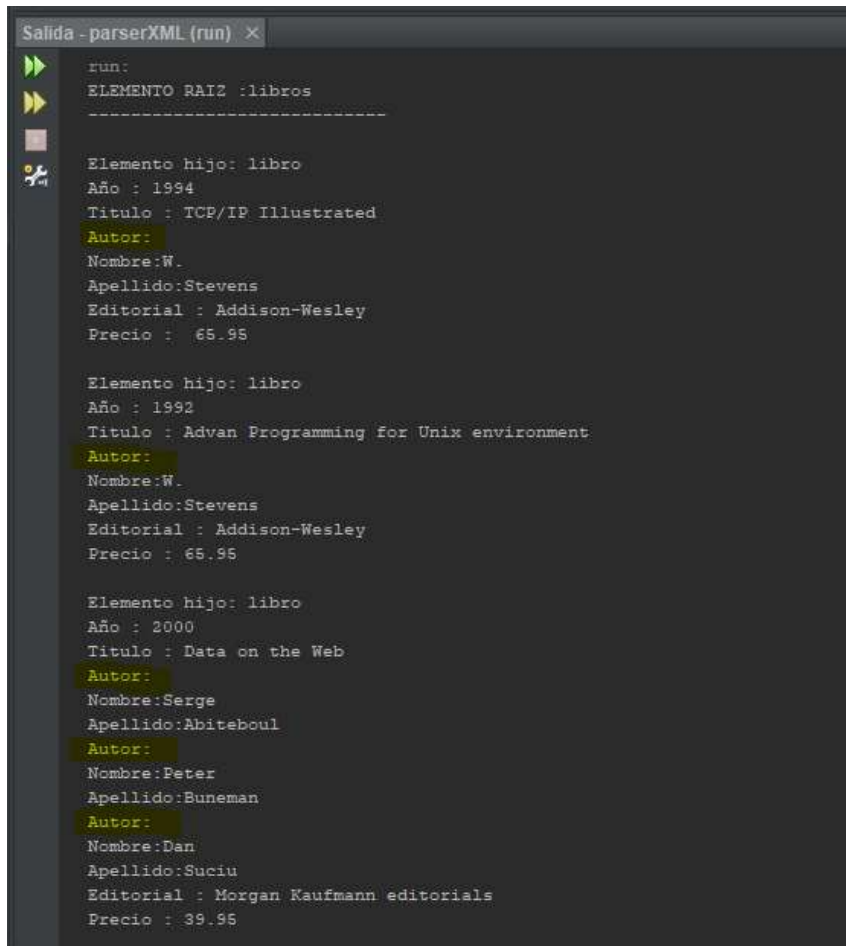
```

Como base, he tomado el ejemplo facilitado en el tema, donde vemos como genera la nueva instancia, luego el documento y comienza a recoger, tras el elemento raíz, los distintos nodos.

La mayor parte del trabajo se puede resolver con un primer bucle for que recorre los elementos y vamos pidiendo las distintas tags por su nombre y recogiendo el texto que incorporan.

Debemos eso sí, añadir un segundo bucle, que se centre en los autores, para así recoger todos los datos, apuntando al item.

Al ejecutar, podemos ver el resultado en consola, donde he resaltado ese último aspecto:



```
Salida - parserXML (run) X
run:
ELEMENTO RAIZ :libros
-----
Elemento hijo: libro
Año : 1994
Titulo : TCP/IP Illustrated
Autor:
Nombre:W.
Apellido:Stevens
Editorial : Addison-Wesley
Precio : 65.95

Elemento hijo: libro
Año : 1992
Titulo : Advan Programming for Unix environment
Autor:
Nombre:W.
Apellido:Stevens
Editorial : Addison-Wesley
Precio : 65.95

Elemento hijo: libro
Año : 2000
Titulo : Data on the Web
Autor:
Nombre:Serge
Apellido:Abiteboul
Autor:
Nombre:Peter
Apellido:Buneman
Autor:
Nombre:Dan
Apellido:Suciu
Editorial : Morgan Kaufmann editorials
Precio : 39.95
```

ParserXML.java (SAX)

```
/**
 * RECORRIENDO ELEMENTOS CON SAX
 * según ejemplo video plataforma
 */

try {
    //Obtención del Parser
    SAXParserFactory spf = SAXParserFactory.newInstance();
    SAXParser saxParser = spf.newSAXParser();

    //Obtenemos la clase creado para manejar los eventos
    DefaultHandler LibrosSAX = new LibrosSAX();
    System.out.println("\nVOLCADO SAX\n*****");
    //Lanzo el parseado
    saxParser.parse(new File("Libros.xml"), LibrosSAX);
} catch (ParserConfigurationException | IOException | SAXException e)
{
    System.out.println(e.getStackTrace());
} //Fin try-catch SAX

} //fin main
} //fin clase ParserXML
```

Además de la clase principal, necesitamos crear una clase que maneje los distintos eventos.

LibrosSAX.java

```
package parserxml;

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

/**
 *
 * @author juang <juangmuelas@gmail.com>
 * @since 11/11/2021
 */
public class LibrosSAX extends DefaultHandler{
    public LibrosSAX(){ super();}
    //Handler para el evento comienzo de documento
    @Override
    public void startDocument() throws SAXException{
        super.startDocument();
        System.out.println("Comienza parseado del documento.");
    }

    //Handler para el evento fin de documento
    @Override
    public void endDocument() throws SAXException{
        super.endDocument();
        System.out.println("\nFin del parseado del documento.");
    }

    //Handler etiqueta apertura
    @Override
    public void startElement(String uri,String localName, String qName,
        Attributes attributes) throws SAXException{
        super.startElement(uri, localName, qName, attributes);

        //Abro etiqueta
        System.out.print("<" + qName);

        //Recorro los atributos si los hay
        if(attributes != null){
            for (int i = 0; i < attributes.getLength(); i++) {
                System.out.print(" "
                    + attributes.getQName(i) + "=" + "\"" + attributes.getValue(i) + "\"");
            }
        }

        //cierro etiqueta
        System.out.print(">");
    }

    //Handler etiqueta cierre
    @Override
    public void endElement(String uri,String localName, String qName) throws
        SAXException{
        super.endElement(uri, localName, qName);
        System.out.print("</" + qName + ">");
    }

    //Handler para el evento Nodo texto encontrado
    @Override
    public void characters(char[] ch,int start, int length) throws
        SAXException{
        super.characters(ch, start, length);

        String content = new String(ch, start, length);
        System.out.print(content);
    }
} //Fin clase LibrosSAX
```

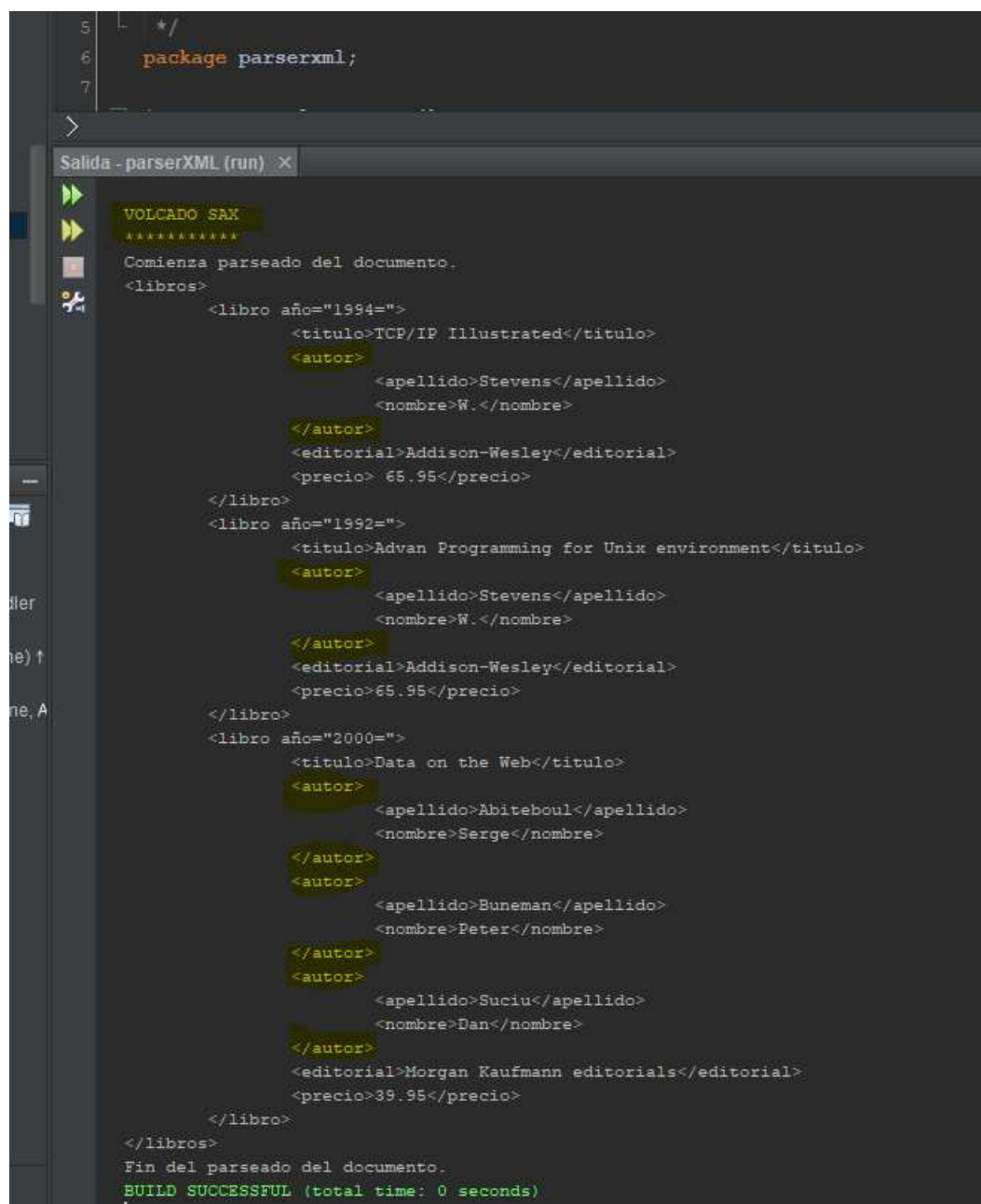
Para el último punto, he seguido un tutorial que nos muestran mediante un enlace a youtube en el apartado 6.2 del tema.

Es bastante práctico y está muy bien explicado, lo que ha ayudado mucho a sortear los cambios necesarios para adaptarlo a esta tarea.

Requiere, como decíamos, de la creación de esta última clase que maneja los evento que incorpora java a través de [DefaultHandler](#) y que se encargan de cada una de las partes del documento parseado. Por no ser muy redundante en las explicaciones, puede verse en la propia captura cada una de las funciones y su utilidad.

La clase principal encierra en un try-catch la obtención del parser de SAX, la llamada a la clase creada para los eventos, y el volcado de los datos.

El resultado por consola:



```
5  */
6  package parserxml;
7
>
Salida - parserXML (run) X
VOLCADO SAX
*****
Comienza parseado del documento.
<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>
Fin del parseado del documento.
BUILD SUCCESSFUL (total time: 0 seconds)
```