

MP0486
Acceso a datos
UF4. Bases de datos XML

**4.2. XML y
JAVA (parsers)**

Índice

☰	Objetivos	3
☰	¿Qué es un parser?	4
☰	Obtener el árbol DOM	6
☰	Iterando los cruceros	9
☰	Escribir documentos XML	16
☰	Resumen	20

Objetivos

En esta lección perseguimos los siguientes objetivos:

- 1 Comprender el concepto de *parser*.
 - 2 Construir un modelo de objeto Java a partir de un documento XML con ayuda del *parser DOM*.
 - 3 Crear un archivo XML a partir de un modelo de objetos Java con ayuda del *parser DOM*.
-

¡Ánimo y adelante!

¿Qué es un parser?

Un **parser o analizador sintáctico** es una herramienta software capaz de analizar el contenido de un documento XML y generar, a partir de él, un modelo de objetos Java. También puede realizar la operación inversa, es decir, construir a partir de un modelo de objetos Java un documento XML.

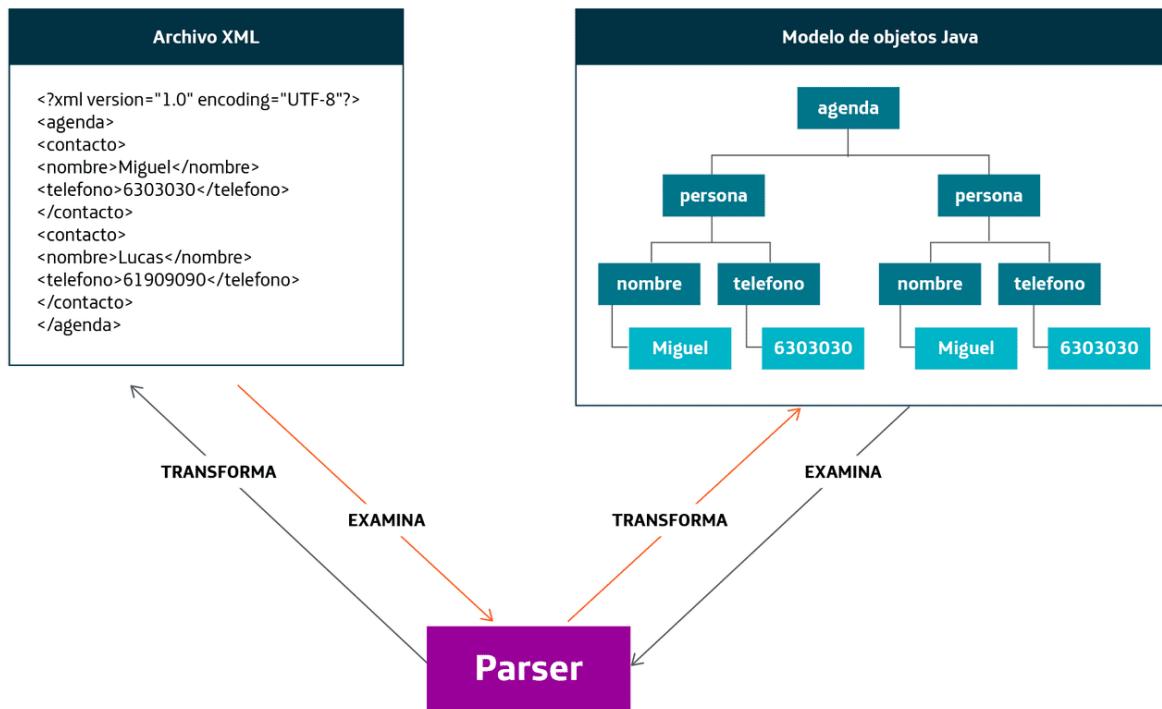
Un parser realiza **dos tipos de tareas**:

1

Primero, a partir de un documento XML, lo examina para ver si su sintaxis es correcta. Después, una vez analizada la sintaxis, construye a partir de él un modelo de objetos que podrá ser manipulado por un programa.

2

A partir de un modelo de objetos Java, lo examina con el fin de construir a partir de él un documento XML o editar uno existente.



Modelo de funcionamiento de un *parser*.

En esta lección utilizaremos el parser DOM (*Document Object Model*) que construye un modelo de objetos Java que replica la estructura del documento XML.

Obtener el árbol DOM

En este apartado utilizaremos el *parser DOM* para leer el documento *cruceros.xml* y construir, a partir de él, un árbol jerárquico de objetos Java denominado **árbol DOM** (*Document Object Model*).

Recorreremos el árbol de objetos obtenido a través del *parser DOM* para mostrar información al usuario sobre cada uno de los cruceros.

Para lograr el objetivo necesitaremos dos librerías distintas:

1

javax.xml.parsers: provee clases que permiten el procesamiento de documentos XML.

2

org.w3c.dom: proporciona las interfaces para la representación del DOM (*Document Object Model*).

Comenzaremos por un ejemplo simple que muestra todo el contenido de texto de la etiqueta raíz (*cruceros*), es decir, sin incluir las etiquetas, sólo los textos.

Para ponerlo en práctica, puedes abrir el proyecto de la lección anterior denominado *ProyectoXML* y añadir la clase Java siguiente:

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;
import org.w3c.dom.Node;

public class LeerCruceros {
    public static void main(String[] args) {
        DocumentBuilderFactory fabrica = DocumentBuilderFactory.newInstance();
        DocumentBuilder analizador;
        Document doc;
        Node raiz;

        try {
            analizador = fabrica.newDocumentBuilder();
            doc = analizador.parse("cruceros.xml");
            raiz = doc.getDocumentElement();
            System.out.println(raiz.getTextContent());
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Vamos a analizar detenidamente el ejemplo:

DocumentBuilderFactory fabrica = DocumentBuilderFactory.newInstance();

La clase **DocumentBuilderFactory**, situada en el paquete **javax.xml.parsers**, nos permite obtener el objeto **DocumentBuilder** a partir de su método **newInstance()**. El objeto **DocumentBuilder** es imprescindible para analizar un documento XML y construir a partir de él un árbol DOM.

DocumentBuilder analizador = fabrica.newDocumentBuilder();

La clase **DocumentBuilder**, situada en el paquete **javax.xml.parsers**, representa un analizador o **parser** cuyos objetos nos permiten construir el árbol DOM a partir del documento XML por medio de su método **parse()**.

Document doc = analizador.parse("cruceros.xml");

La clase **Document**, situada en el paquete **org.w3c.dom.Document**, representa un modelo de objetos como replica de un documento XML. Es tarea del objeto **DocumentBuilder** analizar el contenido del documento XML y devolver el objeto **Document** con el árbol DOM. El objeto que en nuestro ejemplo hemos denominado **doc** ya contiene toda la estructura del documento XML.

```
Node raiz = doc.getDocumentElement();
```

Un documento XML está formado por nodos o elementos que pueden, a su vez, contener otros nodos. El método `getDocumentElement()` de la clase `Document` devuelve el objeto `Node` que representa el nodo raíz, que para nuestro ejemplo es el nodo `cruceros`.

```
System.out.println(raiz.getTextContent());
```

Nuestra variable `raiz` es una referencia al objeto `Node` que representa el nodo `cruceros`. El método `getTextContent()` muestra todo el contenido de texto sin incluir las etiquetas.

Al ejecutar el programa, la información de los cruceros se muestra más o menos así:

```
Problems @ Javadoc Declaration Console
<terminated> LeerCruceros [Java Application] C:\Program Files\Java\jdk1.8.0_6

Mediterraneo (Grecia, Italia)
Costa cruceros
6 días
2018-12-26

Venecia
18:00

Navegación

Agostini
7:00
14:00

Santorini
9:00
20:00

Bari
8:00
14:00

Venecia
```

En el siguiente apartado mejoraremos la **salida a pantalla** de la información de los cruceros.

Iterando los cruceros

Ya tenemos el árbol DOM del documento *cruceros.xml*, y, a partir de él, recuperamos el nodo raíz, es decir, el nodo *cruceros*.

Ahora, mejoraremos la salida en pantalla, iterando los cruceros y mostrando de cada uno los datos que nos interesen.

En el ejemplo anterior utilizamos la expresión *raiz = doc.getDocumentElement()* para obtener el nodo raíz (*cruceros*) y a continuación mostramos, sin más, el texto contenido dentro de dicho nodo.

Sabemos que la variable *raiz* representa al nodo *cruceros*, que está compuesto por nodos hijo (etiquetas *crucero*). En esta ocasión, vamos a iterar los nodos hijos (*crucero*) y, por cada nodo *crucero*, accederemos de momento al *destino* y los *detalles*.

```
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;

import org.w3c.dom.Document;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

public class LeerCruceros {
    public static void main(String[] args) {
        DocumentBuilderFactory fabrica = DocumentBuilderFactory.newInstance();
        DocumentBuilder analizador;
        Document doc;
```

```
Node raiz;

try {
    analizador = fabrica.newDocumentBuilder();
    doc = analizador.parse("cruceros.xml");
    raiz = doc.getDocumentElement();
    recorrerNodos(raiz);
} catch (Exception e) {
    System.out.println(e.getMessage());
}

private static void recorrerNodos(Node raiz) {
    NodeList nodos = raiz.getChildNodes();
    System.out.println("Elementos en el nodo raíz: " + nodos.getLength());
    for (int i=0; i<nodos.getLength();i++) {
        // Iteración por los elementos crucero.
        Node nodoHijo = nodos.item(i);
        if (nodoHijo.getNodeType() == Node.ELEMENT_NODE) {
            Node destino = nodoHijo.getChildNodes().item(1);
            System.out.println(" Destino: " + destino.getTextContent());
            Node detalles = nodoHijo.getChildNodes().item(3);
            System.out.println(" Detalles: " + detalles.getTextContent());
        }
    }
}
```

Iteración de los nodos hijo (*crucero*) y acceso a *destino* y *detalles* de cada uno.

El resultado del programa queda así:

```
Problems @ Javadoc Declaration Console 
<terminated> LeerCruceros [Java Application] C:\Program Files\Java\jdk1.8.0_65\bin\
Elementos en el nodo raíz: 5
Destino: Mediterraneo (Grecia, Italia)
Detalles:
    Costa cruceros
    6 días
    2018-12-26

Destino: Atlántico (España, Marruecos, Portugal)
Detalles:
    MSC Cruceros
    7 días
    2019-02-13
```

Resultado.

Si observas el código, verás que nada más obtener el nodo raíz, invocamos al método *recorrerNodos()*, donde realizamos el resto de la tarea.

Céntrate en estas dos líneas:

```
NodeList nodos = raiz.getChildNodes();
System.out.println("Elementos en el nodo raíz: " + nodos.getLength());
```

Primero, obtenemos un objeto de tipo *NodeList* con la colección de nodos hijos, y después utilizamos el método *getLength()* para obtener el número total de nodos de dicha colección.

Pero, ¿por qué aparece en el resultado que hay cinco elementos, cuando sabemos que sólo hay dos cruceros?

La respuesta está en que un documento XML, además de etiquetas, contiene textos, y también admite que entre el cierre de una etiqueta y la apertura de la siguiente exista texto. Estos textos también son elementos o nodos.

En teoría, nuestro documento XML no contiene textos entre el cierre de una etiqueta y la apertura de la siguiente, pero sí contiene un retorno de carro, y eso o un simple espacio ya es considerado un texto.

Con esta imagen, que representa un documento XML muy simple, lo comprenderás mejor:

<agenda>	
Nodo de texto 1	Nodo 1: TEXT_NODE
<contacto>	
primer contacto	
<nombre>Miguel</nombre>	Nodo 2: ELEMENT_NODE
<telefono>6303030</telefono>	
</contacto>	
Nodo de texto 2	Nodo 3: TEXT_NODE
<contacto>	
Segundo contacto	
<nombre>Lucas</nombre>	Nodo 4: ELEMENT_NODE
<telefono>61909090</telefono>	
</contacto>	
Nodo de texto 3	Nodo 5: TEXT_NODE
</agenda>	

Documento XML simple.

Tenemos un nodo raíz *agenda* y sólo dos contactos, sin embargo, hemos metido textos entre medias de las etiquetas, con color rojo en la imagen. Estos textos también son considerados nodos. Si no estuvieran estos textos, bastaría con que hubiera un retorno de carro o un espacio en blanco en su lugar para que se considere un texto.

La clase *Node* cuenta con un método denominado *getNodeType()* que informa, mediante un número entero, de qué tipo de nodo se trata (*ELEMENT_NODE* o *TEXT_NODE*).



Vamos a terminar de analizar **el resto de código:**

```
for (int i=0; i<nodos.getLength();i++) {  
    // Iteración por los elementos crucero.  
    Node nodoHijo = nodos.item(i);  
    if (nodoHijo.getNodeType() == Node.ELEMENT_NODE) {  
        Node destino = nodoHijo.getChildNodes().item(1);  
        System.out.println(" Destino: " + destino.getTextContent());  
        Node detalles = nodoHijo.getChildNodes().item(3);  
        System.out.println(" Detalles: " + detalles.getTextContent());  
    }  
}
```

Recuerda que la variable *nodos* representaba a la colección de nodos hijo dentro del nodo raíz, es decir, los elementos *crucero* y los elementos de tipo texto que no nos interesan.

Lo primero que hacemos es una iteración de tipo *for* para acceder cada uno de los nodos a través del método *item(index)*, que devuelve un elemento a través de su índice. Por cada nodo hijo iterado, sólo realizaremos el resto de las acciones si se trata de un *ELEMENT_NODE*, en cuyo caso sabemos que se trata de un elemento *crucero* con sus etiquetas internas *destino* y *detalles* que ocuparan las posiciones 1 y 3, ya que también tienen elementos de tipo texto entre medias.

Ahora, **completaremos el programa** para que también muestre las escalas de cada crucero.

Ya conoces la dinámica de trabajo; ahora tendrás que considerar que cada *crucero*, además de *destino* y *detalles*, consta de un elemento *escalas* compuesto por una colección de elementos *escala*, por lo que habrá que iterar de nuevo para obtener la información de cada una de las escalas:

```
import javax.xml.parsers.DocumentBuilder;  
import javax.xml.parsers.DocumentBuilderFactory;  
  
import org.w3c.dom.Document;  
import org.w3c.dom.Node;  
import org.w3c.dom.NodeList;  
  
public class LeerCruceros {
```

```
public static void main(String[] args) {  
    DocumentBuilderFactory fabrica = DocumentBuilderFactory.newInstance();  
    DocumentBuilder analizador;  
    Document doc;  
    Node raiz;  
  
    try {  
        analizador = fabrica.newDocumentBuilder();  
        doc = analizador.parse("cruceros.xml");  
        raiz = doc.getDocumentElement();  
        recorrerNodos(raiz);  
    } catch (Exception e) {  
        System.out.println(e.getMessage());  
    }  
}  
  
private static void recorrerNodos(Node raiz) {  
    NodeList nodos = raiz.getChildNodes();  
    for (int i=0; i<nodos.getLength();i++) {  
        // Iteración por los elementos crucero.  
        Node nodoHijo = nodos.item(i);  
        if (nodoHijo.getNodeType() == Node.ELEMENT_NODE) {  
            Node destino = nodoHijo.getChildNodes().item(1);  
            System.out.println("Destino: " + destino.getTextContent());  
            System.out.println("-----");  
            Node detalles = nodoHijo.getChildNodes().item(3);  
            System.out.println(" Detalles: " + detalles.getTextContent());  
            Node escalas = nodoHijo.getChildNodes().item(5);  
            recorrerEscalas(escalas);  
        }  
    }  
}  
  
private static void recorrerEscalas(Node escalas) {  
    NodeList nodos = escalas.getChildNodes();  
    System.out.println(" Escalas:");  
    for (int i=0; i<nodos.getLength();i++) {  
        Node escala = nodos.item(i);  
        if (escala.getNodeType() == Node.ELEMENT_NODE) {  
            String dia = escala.getAttributes().item(0).getNodeValue();  
            Node parada = escala.getChildNodes().item(1);  
            Node llegada = escala.getChildNodes().item(3);  
            Node salida = escala.getChildNodes().item(5);  
            System.out.print(" ");  
            System.out.print("Día " + dia + ": ");  
            System.out.print(parada.getTextContent() + " ");  
            System.out.print(llegada.getTextContent() + " - ");  
            System.out.println(salida.getTextContent());  
        }  
    }  
    System.out.println();  
}
```

Iteración de elementos *escala*.

Nuestro programa arroja el siguiente **resultado**:

```
Problems @ Javadoc Declaration Console X
<terminated> LeerCruceros [Java Application] C:\Program Files\Java\jdk1.8.0_65\bin\javaw.exe (15/6/2018)
Destino: Mediterraneo (Grecia, Italia)
-----
Detalles:
    Costa cruceros
    6 días
    2018-12-26

Escalas:
    Día 1: Venecia - 18:00
    Día 2: Navegación -
    Día 3: Agostini 7:00 - 14:00
    Día 4: Santorini 9:00 - 20:00
    Día 5: Bari 8:00 - 14:00
    Día 6: Venecia 8:30 -

Destino: Atlántico (España, Marruecos, Portugal)
-----
Detalles:
    MSC Cruceros
    7 días
    2019-02-13

Escalas:
    Día 1: Barcelona - 18:00
    Día 2: Navegación -
    Día 3: Casablanca 7:00 - 22:00
    Día 4: Navegación -
    Día 5: Santa Cruz de Tenerife 9:00 - 16:00
    Día 6: Funchal 8:00 - 19:00
    Día 7: Barcelona 17:00 -
```

Vista del resultado final.

Escribir documentos XML

La tecnología DOM también nos permite construir archivos XML a partir de un modelo de objetos Java.

Aprenderás cómo a partir de un sencillo ejemplo, que crea un documento XML para guardar los datos de una agenda telefónica.

Comienza por crear un proyecto Java y la siguiente clase principal, luego analizaremos el código detenidamente.

```
import java.io.File;  
  
import javax.xml.parsers.DocumentBuilder;  
import javax.xml.parsers.DocumentBuilderFactory;  
import javax.xml.transform.Transformer;  
import javax.xml.transform.TransformerException;  
import javax.xml.transform.TransformerFactory;  
import javax.xml.transform.dom.DOMSource;  
import javax.xml.transform.stream.StreamResult;  
  
import org.w3c.dom.Document;  
import org.w3c.dom.Element;  
  
  
public class CrearAgenda {  
    public static void main(String[] args) {  
        DocumentBuilderFactory fabrica = DocumentBuilderFactory.newInstance();  
    }  
}
```

```
DocumentBuilder analizador;
Document doc;

try {
    analizador = fabrica.newDocumentBuilder();
    // Creamos nuevo documento
    doc = analizador.newDocument();
    // Añadimos elemento raiz
    Element agenda = doc.createElement("agenda");
    doc.appendChild(agenda);
    // Añadimos tres contactos al elemento raíz agenda.
    agregarContactos(agenda, doc);
    // Guardamos en disco el nuevo documento XML.
    guardar(doc);

    System.out.println("El archivo se ha creado con éxito");
} catch (Exception e) {
    System.out.println(e.getMessage());
}
}

public static void agregarContactos(Element agenda, Document doc) {
    // Agregamos primer contacto
    Element contacto = doc.createElement("contacto");
    agenda.appendChild(contacto);
    Element nombre = doc.createElement("nombre");
    nombre.appendChild(doc.createTextNode("AMELIA GONZALEZ LOPEZ"));
    contacto.appendChild(nombre);
    Element telefono = doc.createElement("telefono");
    telefono.appendChild(doc.createTextNode("612333333"));
    contacto.appendChild(telefono);

    // Agregamos segundo contacto
    contacto = doc.createElement("contacto");
    agenda.appendChild(contacto);
    nombre = doc.createElement("nombre");
    nombre.appendChild(doc.createTextNode("PEDRO BOTIJO MONTERA"));
    contacto.appendChild(nombre);
    telefono = doc.createElement("telefono");
    telefono.appendChild(doc.createTextNode("622333444"));
    contacto.appendChild(telefono);

    // Agregamos tercer contacto
    contacto = doc.createElement("contacto");
    agenda.appendChild(contacto);
    nombre = doc.createElement("nombre");
    nombre.appendChild(doc.createTextNode("MIGUEL MALATESTA SENTADO"));
    contacto.appendChild(nombre);
    telefono = doc.createElement("telefono");
    telefono.appendChild(doc.createTextNode("655444333"));
    contacto.appendChild(telefono);
}

private static void guardar(Document doc) throws TransformerException {
    TransformerFactory fabricaConversor = TransformerFactory.newInstance();
    Transformer conversor = fabricaConversor.newTransformer();
    DOMSource fuente = new DOMSource(doc);
    StreamResult resultado = new StreamResult(new File("agenda.xml"));
    conversor.transform(fuente, resultado);
}
```

Vamos a estudiar detenidamente el código anterior:

doc = analizador.newDocument();

De nuevo volvemos a utilizar una referencia a un objeto de tipo *DocumentBuilder* (*analizador*) para obtener un objeto *Document*. Esta vez queremos construir un objeto *Document* nuevo para después ir añadiéndole nuevos elementos o nodos.

Element agenda = doc.createElement("agenda");

Los objetos *Document* cuentan con el método *createElement()* capaz de crear un objeto de tipo *Element*. La clase *Element* representa un elemento de un documento XML, o lo que es lo mismo, una etiqueta con su apertura y cierre. Un elemento también es un nodo, sin embargo, un objeto de la clase *Node* puede representar a cualquier tipo de nodo (elementos o textos), mientras que un objeto de tipo *Element* representa a una etiqueta.

doc.appendChild(agenda);

Con la sentencia "*Element agenda = doc.createElement("agenda")*", creamos un elemento XML virtual, y ahora con el método *appendChild(agenda)* lo estamos agregando al documento. Hasta aquí, nuestro documento virtual está así:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<agenda>
</agenda>
```

A partir de ahora, el programa deberá ir añadiendo nuevos elementos hijos dentro del elemento raíz *agenda*.

agregarContactos(agenda, doc);

Dentro de este método agregamos tres contactos, ejecutando las siguientes sentencias:

```
Element contacto = doc.createElement("contacto");
agenda.appendChild(contacto);
Element nombre = doc.createElement("nombre");
nombre.appendChild(doc.createTextNode("AMELIA GONZALEZ LOPEZ"));
```

```
contacto.appendChild(nombre);
Element telefono = doc.createElement("telefono");
telefono.appendChild(doc.createTextNode("61233333"));
contacto.appendChild(telefono);
```

Para añadir cada uno de los contactos, realizamos las siguientes operaciones:

- Creamos un elemento *contacto* y lo añadimos al elemento *agenda*.
- Creamos un elemento *nombre* y le añadimos un elemento de texto con el valor "AMELIA GONZALEZ LOPEZ".
- Añadimos el elemento *nombre* al contacto.
- Creamos un elemento *telefono* y le añadimos un elemento de texto con el valor "61233333".
- Añadimos el elemento *telefono* al contacto.

guardar(doc);

Hasta aquí, se ha creado el árbol DOM para el nuevo documento XML, pero en memoria, no se ha guardado en un archivo de disco. Justo ésa es la misión del método *guardar*. Vamos a recordar la implementación del método *guardar()*.

```
TransformerFactory fabricaConversor = TransformerFactory.newInstance();
Transformer conversor = fabricaConversor.newTransformer();
DOMSource fuente = new DOMSource(doc);
StreamResult resultado = new StreamResult(new File("agenda.xml"));
conversor.transform(fuente, resultado);
```

La clase *Transformer* dispone de métodos que transforman un árbol DOM a un formato de documento XML. Hay que tener en cuenta que la estructura DOM no es exclusiva de los documentos XML; hay otros formatos a partir de los que se puede obtener un árbol DOM, por ejemplo, un documento HTML.

Para obtener una referencia a un objeto *Transformer* hay que invocar al método *newTransformer()* de la clase *TransformerFactory*.

Este objeto *Transformer* provee del método *transform()* que realiza la transformación al formato XML, generando así el documento físico. El método *transform()* requiere dos argumentos:

- Un objeto *DOMSource*, que representa el árbol DOM de origen.
- Un objeto *StreamResult*, que representa un flujo de datos de escritura asociado al fichero físico donde deseamos escribir los datos XML.

Resumen

Has terminado la lección, repasemos los puntos más importantes que hemos tratado.

- Un **parser o analizador sintáctico** es una herramienta software capaz de analizar el contenido de un documento XML y generar a partir de él un modelo de objetos Java. También puede realizar la operación inversa, es decir, construir a partir de un modelo de objetos Java un documento XML.
- Uno de los *parsers* más empleados en las aplicaciones Java es el **parser DOM** (*Document Object Model*). El parser DOM requiere estas librerías: *javax.xml.parsers*, y provee clases que permiten el procesamiento de documentos XML y *org.w3c.dom*, que proporciona las interfaces para la representación del DOM.



PROEDUCA