

Asignatura

Acceso a datos



UNIVERSAE
Instituto Superior de FP

Asignatura

Acceso a datos

UNIDAD 6

XML



UNIVERSAE
Instituto Superior de FP

XML

Las siglas de XML significan extensible markup language

Ventajas

- No necesita de conocimientos de programación para generar un documento simple
- Permite intercambios de datos entre diferentes programas de forma segura

Estructura

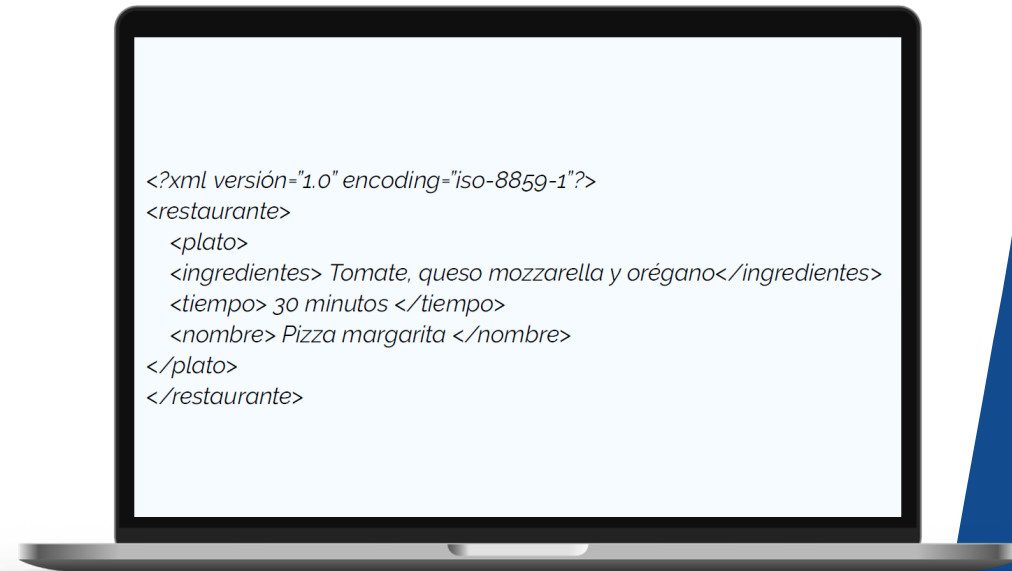
El documento XML contiene dos partes: el prólogo y el cuerpo

A continuación veremos un ejemplo.



XML: estructura y sintaxis

- Prólogo. Versión y codificación
- Cuerpo
 - Elementos. Etiquetas con apertura y cierre.
 - Atributos. Son propiedades de los elementos
 - Comentarios. `<!-- -->`



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```

▼<restaurante>
  ▼<plato>
    <ingredientes> Tomate, queso mozzarella y orégano</ingredientes>
    <tiempo> 30 minutos </tiempo>
    <nombre> Pizza margarita </nombre>
  </plato>
</restaurante>

```

DOM

Definición

- Document Object Model
- API que permite construir, navegar o modificar un documento HTML o XML
- Se realiza dinámicamente
- Representa una estructura jerárquica con todos los elementos del documento.

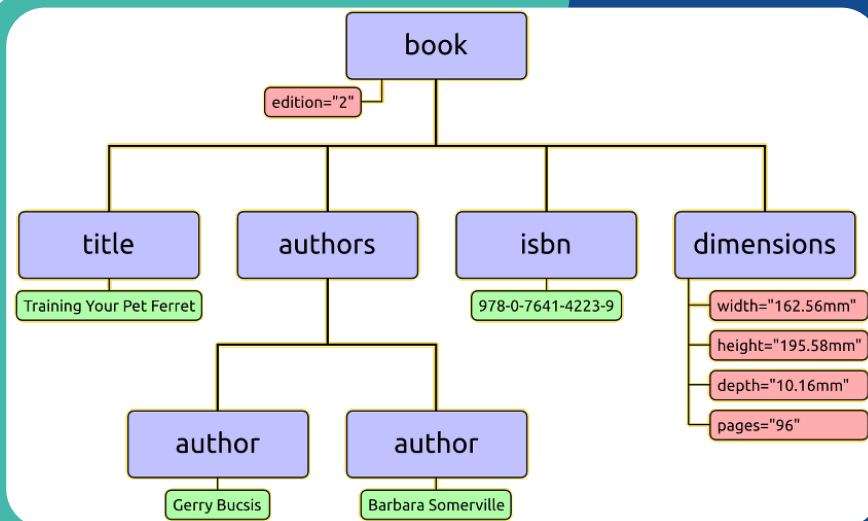
Estructura

- Esta compuesta de nodos
- Un nodo representa una etiqueta HTML o XML
- Los nodos se pueden clasificar en:
 - Document – Representa la raíz
 - Element – Representa una etiqueta
 - Atributo – Atributos que contienen los elementos
 - Text – Contiene el valor

Acciones

- Obtener y buscar nodos
- Añadir nodos
- Eliminar nodos
- Modificar propiedades de nodos

```
<?xml version='1.0' encoding='UTF-8' standalone="yes" ?>
<book edition="2">
  <title>Training Your Pet Ferret</title>
  <authors>
    <author>Gerry Bucsis</author>
    <author>Barbara Somerville</author>
  </authors>
  <isbn>9780764142239</isbn>
  <dimensions width="162.56mm" height="195.58mm" depth="10.16mm" pages="96" />
</book>
```





DOM. Parsing

```
try {
    File inputFile = new File("src/DOM/estudiantes.xml");
    DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
    DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
    Document doc = dBuilder.parse(inputFile);

    System.out.println("Elemento principal : " + doc.getDocumentElement().getNodeName());
    NodeList nodeList = doc.getElementsByTagName("estudiante");
    System.out.println("");

    for (int i = 0; i < nodeList.getLength(); i++) {
        Node nodo = nodeList.item(i);

        if (nodo.getNodeType() == Node.ELEMENT_NODE) {
            Element element = (Element) nodo;
            System.out.println("Número : " + element.getAttribute("numero"));
            System.out.println("Nombre : " +
                element.getElementsByTagName("nombre").item(0).getTextContent());
            System.out.println("Apellido : " +
                element.getElementsByTagName("apellido").item(0).getTextContent());
        }

        System.out.println();
    }
} catch (Exception e) {
    e.printStackTrace();
}
```

```
<?xml version='1.0' encoding='UTF-8' standalone="yes" ?>
<clase>
    <estudiante numero = "393">
        <nombre>Antonio</nombre>
        <apellido>García</apellido>
    </estudiante>
    <estudiante numero = "493">
        <nombre>Karen</nombre>
        <apellido>Ramírez</apellido>
    </estudiante>
    <estudiante numero = "593">
        <nombre>Sofia</nombre>
        <apellido>González</apellido>
    </estudiante>
</clase>
```

```
<terminated> ParsingDOM [Java Ap
Elemento principal :clase

Número : 393
Nombre : Antonio
Apellido : García
```



DOM. Crear documentos y serialización

```
try {  
  
    DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();  
    DocumentBuilder docBuilder = docFactory.newDocumentBuilder();  
  
    // Crear documento  
    Document doc = docBuilder.newDocument();  
    Element rootElement = doc.createElement("Empresas");  
    doc.appendChild(rootElement);  
  
    for (int i = 1; i <= 5; i++) {  
        Element empresa = doc.createElement("empresa");  
        empresa.setAttribute("CIF", "C"+i);  
  
        Element empleado = doc.createElement("empleado");  
        Element codigo = doc.createElement("codigo");  
        codigo.setTextContent("000"+i);  
        empleado.appendChild(codigo);  
        empresa.appendChild(empleado);  
  
        rootElement.appendChild(empresa);  
    }  
  
    // Serialización  
    TransformerFactory transformerFactory = TransformerFactory.newInstance();  
    Transformer transformer = transformerFactory.newTransformer();  
    transformer.setOutputProperty(OutputKeys.INDENT, "yes");  
    transformer.setOutputProperty(OutputKeys.METHOD, "xml");  
    DOMSource source = new DOMSource(doc);  
    StreamResult result = new StreamResult("src/DOM/empresas.xml");  
  
    transformer.transform(source, result);  
  
} catch (Exception e) {  
    e.printStackTrace();  
}
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>  
<Empresas>  
  <empresa CIF="C1">  
    <empleado>  
      <codigo>0001</codigo>  
    </empleado>  
  </empresa>  
  <empresa CIF="C2">  
    <empleado>  
      <codigo>0002</codigo>  
    </empleado>  
  </empresa>  
  <empresa CIF="C3">  
    <empleado>  
      <codigo>0003</codigo>  
    </empleado>  
  </empresa>  
  <empresa CIF="C4">  
    <empleado>  
      <codigo>0004</codigo>  
    </empleado>  
  </empresa>  
  <empresa CIF="C5">  
    <empleado>  
      <codigo>0005</codigo>  
    </empleado>  
  </empresa>  
</Empresas>
```

SAX

Definición

- Simple API for XML
- No es DOM
- Recorre el documento y genera eventos
- Los eventos son del tipo:
 - Comienzo de elemento
 - Final de elemento

Características

- No necesita cargar en memoria todo el contenido del documento
- Es mucho mas rápido que DOM
- Acceso secuencial
- No tiene métodos de creación o actualización

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<usuarios>
  <usuario id="1">
    <nombre>Jose</nombre>
    <username>jose</username>
    <password>1234</password>
  </usuario>
  <usuario id="2">
    <nombre>Carolina</nombre>
    <username>carolina</username>
    <password>5678</password>
  </usuario>
  <usuario id="3">
    <nombre>Pedro</nombre>
    <username>pedro</username>
    <password>9123</password>
  </usuario>
</usuarios>
```

```
startElement:usuarios
  startElement:usuario
    startElement:nombre
    endElement:nombre
    startElement:username
    endElement:username
    startElement:password
    endElement:password
  endElement:usuario
  startElement:usuario
  startElement:nombre
  endElement:nombre
  startElement:username
  endElement:username
  startElement:password
  endElement:password
endElement:usuario
  startElement:usuario
  startElement:nombre
  endElement:nombre
  startElement:username
  endElement:username
  startElement:password
  endElement:password
endElement:usuario
endElement:usuarios
```



SAX. Parsing



```
public class Manejador extends DefaultHandler {

    private boolean nombre = false;
    private boolean username = false;
    private boolean password = false;

    public void startElement(String uri, String localName, String qName, Attributes attributes)
        throws SAXException {
        System.out.println("startElement:" + qName);
        if (qName.equalsIgnoreCase("usuario")) {
            String id = attributes.getValue("id");
            System.out.println("id: " + id);
        }
        if (qName.equalsIgnoreCase("nombre")) {
            nombre = true;
        }
        if (qName.equalsIgnoreCase("username")) {
            username = true;
        }
        if (qName.equalsIgnoreCase("password")) {
            password = true;
        }
    }

    public void endElement(String uri, String localName, String qName) throws SAXException {
        System.out.println("endElement:" + qName);
    }

    public void characters(char ch[], int start, int length) throws SAXException {
        if (nombre) {
            System.out.println("nombre: " + new String(ch, start, length));
            nombre = false;
        }
        if (username) {
            System.out.println("username: " + new String(ch, start, length));
            username = false;
        }
        if (password) {
            System.out.println("password: " + new String(ch, start, length));
            password = false;
        }
    }
}
```

```
try {
    SAXParserFactory factory = SAXParserFactory.newInstance();
    SAXParser saxParser = factory.newSAXParser();

    DefaultHandler handler = new Manejador();

    saxParser.parse("src/SAX/usuarios.xml", handler);
} catch (Exception e) {
    e.printStackTrace();
}
```



Validación

- Un documento bien formado
- Se debe respetar la estructura, sintaxis y orden.

DTD

- Definición de Tipo de Documento
- Tiene su propia sintaxis SGML
- Se centra en la estructura de los elementos, orden, apariciones y tipos de datos

```
<!ELEMENT empleados (empleado+) >
<!ELEMENT empleado (id, nombre, email) >
<!ELEMENT id (#PCDATA)>
<!ELEMENT nombre (#PCDATA)>
<!ELEMENT email (#PCDATA)>
```

Esquema XSD

- XML Schema Definition
- La sintaxis es basada en XML
- Se centra en la estructura y contenido de forma más precisa
- Permiten especificar nuevos datos

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="agenda">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="contacto" minOccurs="1" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

DTD	XSD
ELEMENT	<element>
#PCDATA	Parte de un tipo simple
ANY	<any>
EMPTY	Soportado
Modelo de Contenido	<complexType>
, (Conector de secuencia)	<sequence>
(Conector de alternativas)	<disjunction>
? (Opcional)	Soportado
+ (Requerido y Repetible)	Soportado
* (Opcional y Repetible)	Soportado
ATTLIST	<attributeGroup>
Tipo de atributo CDATA, ID, IDREF, NOTATION...	Tipos <simpleType>predefinidos

Validación. DTD



```
try {  
  
    DocumentBuilderFactory domFactory = DocumentBuilderFactory.newInstance();  
    domFactory.setValidating(true);  
    DocumentBuilder builder = domFactory.newDocumentBuilder();  
    builder.setErrorHandler(new ErrorHandler() {  
        @Override  
        public void error(SAXParseException e) throws SAXException {  
            System.out.println("Error: " + e.getMessage());  
        }  
  
        @Override  
        public void fatalError(SAXParseException e) throws SAXException {  
            System.out.println("Error grave: " + e.getMessage());  
        }  
  
        @Override  
        public void warning(SAXParseException e) throws SAXException {  
            System.out.println("Aviso: " + e.getMessage());  
        }  
    });  
  
    Document doc = builder.parse("src/DTD/empleados.xml");  
  
} catch (Exception e) {  
    e.printStackTrace();  
}
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>  
<!DOCTYPE empleados SYSTEM "empleados.dtd">  
<empleados>  
    <empleado>  
        <id>001</id>  
        <nombre>Carolina</nombre>  
        <email>carol001@email.com</email>  
    </empleado>  
    <empleado>  
        <id>001</id>  
        <nombre>Jesús</nombre>  
        <apellido>Roldan</apellido>  
        <email>carol001@email.com</email>  
    </empleado>  
</empleados>
```

```
empleados.dtd X  
1 <!ELEMENT empleados (empleado+) >  
2 <!ELEMENT empleado (id, nombre, email) >  
3 <!ELEMENT id (#PCDATA)>  
4 <!ELEMENT nombre (#PCDATA)>  
5 <!ELEMENT email (#PCDATA)>
```

Problems @ Javadoc Declaration Console X Properties
<terminated> ValidatorDTD [Java Application] C:\Software\Eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.
Error: El tipo de elemento "apellido" debe declararse.
Error: El contenido del tipo de elemento "empleado" debe coincidir con "(id,nombre,email)".

Validación. Esquemas XSD



```
String xsd = "src/Esquemas/marcadores.xsd";
String xml = "src/Esquemas/marcadores.xml";

try {
    SchemaFactory factory = SchemaFactory.newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
    Schema schema = factory.newSchema(new File(xsd));
    Validator validator = schema.newValidator();
    validator.validate(new StreamSource(new File(xml)));
} catch (Exception e) {
    System.out.println("Exception: " + e.getMessage());
}
```

```
marcadores.xsd
1 <?xml version="1.0" encoding="UTF-8"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3   <xs:element name="marcadores">
4     <xs:complexType>
5       <xs:sequence>
6         <xs:element name="pagina" maxOccurs="unbounded">
7           <xs:complexType>
8             <xs:sequence>
9               <xs:element name="nombre" type="xs:string"/>
10              <xs:element name="descripcion" type="xs:string"/>
11              <xs:element name="url" type="xs:string"/>
12            </xs:sequence>
13          </xs:complexType>
14        </xs:element>
15      </xs:sequence>
16    </xs:complexType>
17  </xs:element>
18 </xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<marcadores xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="marcadores.xsd">
  <pagina>
    <nombre>XML</nombre>
    <descripcion>Tutorial de XML</descripcion>
    <url>https://developer.mozilla.org/es/docs/Web/XML/XML\_introduction/</url>
  </pagina>
  <pagina>
    <nombre>Wikipedia</nombre>
    <descripcion>La enciclopedia libre.</descripcion>
    <url>http://www.wikipedia.org/</url>
  </pagina>
  <pagina>
    <nombre>W3C</nombre>
    <tags></tags>
    <descripcion>World Wide Web Consortium.</descripcion>
    <url>http://www.w3.org/</url>
  </pagina>
</marcadores>
```

```
Problems @ Javadoc Declaration Console Properties
<terminated> Validador [Java Application] C:\Software\Eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.
Exception: cvc-complex-type.2.4.a: Se ha encontrado contenido no válido a partir del
elemento 'tags'. Se esperaba uno de '{descripcion}'.
```

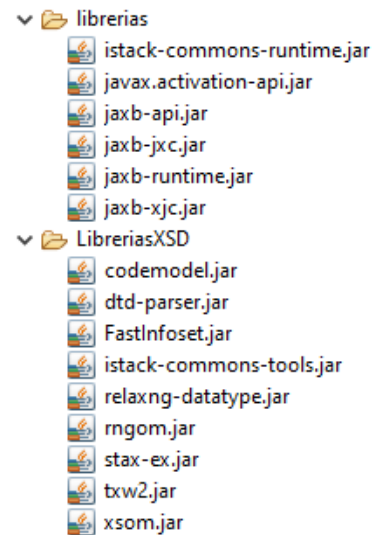
JAXB

¿Qué es?

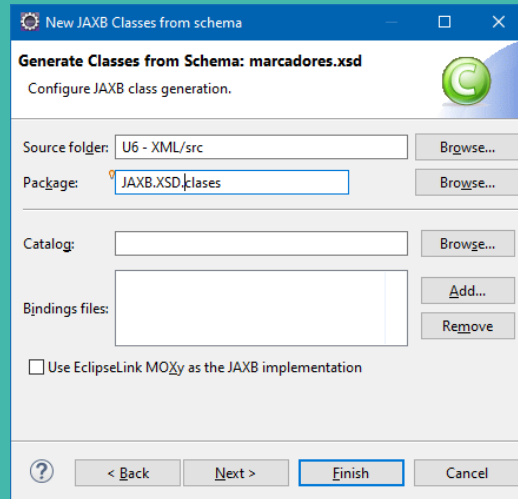
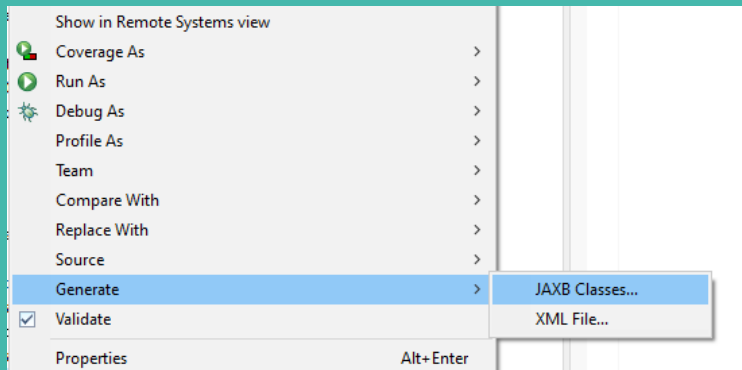
- Java Architecture for XML Binding
- Librería para traducir ficheros XML a objetos o viceversa
- Se puede realizar desde XML o esquemas XSD

Requerimientos

- Librería jaxb.
<https://javaee.github.io/jaxb-v2/>
- **Importante** dependiendo de la versión de java, pueden estar incluidas las librerías o no.



JAXB. Crear clases a partir de XSD



Problems @ Javadoc Declaration Console X
<terminated> C:\Software\Eclipse\plugins\org.eclipse.justjop
Analizando un esquema...
Compilando un esquema...
JAXB\XSD\clases\Marcadores.java
JAXB\XSD\clases\ObjectFactory.java

▼ JAXB.XSD.clases
> Marcadores.java
> ObjectFactory.java

```
package JAXB.XSD.clases;

import java.util.ArrayList;

* <p>Clase Java para anonymous complex type.
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "", propOrder = {
    "pagina"
})
@XmlRootElement(name = "marcadores")
public class Marcadores {

    @XmlElement(required = true)
    protected List<Marcadores.Pagina> pagina;

    * Gets the value of the pagina property.
    public List<Marcadores.Pagina> getPagina() {
        if (pagina == null) {
            pagina = new ArrayList<Marcadores.Pagina>();
        }
        return this.pagina;
    }

    * <p>Clase Java para anonymous complex type.
    @XmlAccessorType(XmlAccessType.FIELD)
    @XmlType(name = "", propOrder = {
        "nombre",
        "descripcion",
        "url"
    })
    public static class Pagina {

        @XmlElement(required = true)
        protected String nombre;
        @XmlElement(required = true)
        protected String descripcion;
        @XmlElement(required = true)
        protected String url;
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="marcadores">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="pagina" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="nombre" type="xs:string"/>
              <xs:element name="descripcion" type="xs:string"/>
              <xs:element name="url" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

JAXB. Crear objetos a partir de XML



```
@XmlRootElement(name = "empleado")
@XmlAccessorType(XmlAccessType.PROPERTY)
public class Empleado implements Serializable {

    private static final long serialVersionUID = 1L;

    private Integer id;
    private String nombre;
    private String apellido;
    private Departamento departamento;

    public Empleado() {
        super();
    }

    public Empleado(int id, String name, String apellido, Departamento departamento) {
        super();
        this.id = id;
        this.nombre = name;
        this.apellido = apellido;
        this.departamento = departamento;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getNombre() {}

    public void setNombre(String nombre) {}

    public String getApellido() {}

    public void setApellido(String apellido) {}

    public Departamento getDepartamento() {}

    public void setDepartamento(Departamento departamento) {}

    @Override
    public String toString() {
        return "Empleado [id=" + id + ", Nombre=" + nombre
            + ", Apellido=" + apellido + ", Departamento=" + departamento + "];"
    }
}
```

```
@XmlRootElement(name = "departamento")
@XmlAccessorType(XmlAccessType.PROPERTY)
public class Departamento implements Serializable {

    private static final long serialVersionUID = 1L;

    Integer id;
    String nombre;

    public Departamento() {
        super();
    }

    public Departamento(Integer id, String nombre) {
        super();
        this.id = id;
        this.nombre = nombre;
    }

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getNombre() {
        return nombre;
    }

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

    @Override
    public String toString() {
        return "Departamento [id=" + id + ", nombre=" + nombre + "];"
    }
}
```

```
File xmlFile = new File("src/JAXB/empleados.xml");

try {
    JAXBContext jaxbContext = JAXBContext.newInstance(Empleado.class);
    Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();

    Empleado employee = (Empleado) jaxbUnmarshaller.unmarshal(xmlFile);
    System.out.println(employee);
} catch (JAXBException e) {
    e.printStackTrace();
}
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<empleado>
  <departamento>
    <id>001</id>
    <nombre>Servicio técnico</nombre>
  </departamento>
  <id>001</id>
  <nombre>Jose</nombre>
  <apellido>Mardales</apellido>
</empleado>
```

Problems @ Javadoc Declaration Console × Properties

<terminated> Transformacion [Java Application] C:\Software\Eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.2.v20

Empleado [id=1, Nombre=Jose, Apellido=Mardales, Departamento=Departamento [id=1, nombre=Servicio técnico]]

JAXB. Crear XML a partir de objetos



```
try {
    Empleado empleado = new Empleado(001, "Jose", "Mardales", new Departamento(001, "Servicio técnico"));

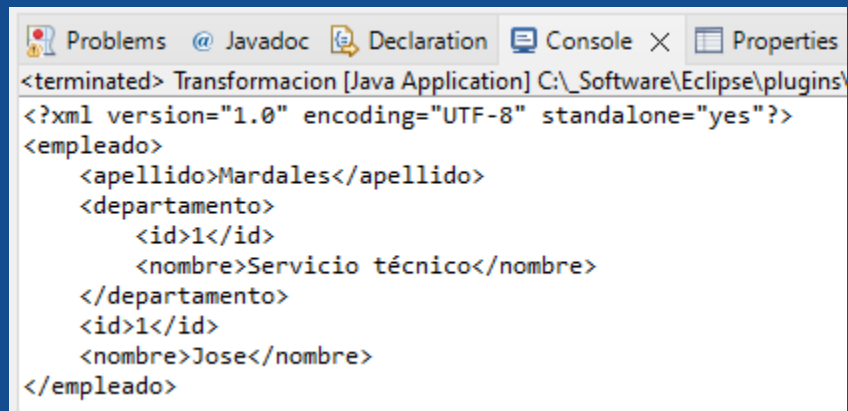
    JAXBContext jaxbContext = JAXBContext.newInstance(Empleado.class);
    Marshaller jaxbMarshaller = jaxbContext.createMarshaller();

    jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);

    StringWriter sw = new StringWriter();

    jaxbMarshaller.marshal(empleado, sw);

    String xmlContent = sw.toString();
    System.out.println(xmlContent);
} catch (JAXBException e) {
    e.printStackTrace();
}
```



The screenshot shows the Eclipse IDE's Console window. The title bar includes tabs for Problems, Javadoc, Declaration, Console, and Properties. The Console output displays the XML result of marshalling an Empleado object. The XML is formatted and includes a declaration at the top: `<?xml version="1.0" encoding="UTF-8" standalone="yes"?>`. The root element is `<empleado>`, which contains nested elements for `<apellido>`, `<departamento>` (with an `<id>` child), and `<nombre>`. The final output is:

```
<terminated> Transformacion [Java Application] C:\_Software\Eclipse\plugins\
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<empleado>
  <apellido>Mardales</apellido>
  <departamento>
    <id>1</id>
    <nombre>Servicio técnico</nombre>
  </departamento>
  <id>1</id>
  <nombre>Jose</nombre>
</empleado>
```


XPath

URL: <https://www.w3.org/TR/xpath/#predicates>

¿Qué es?

- Es un lenguaje estandarizado por W3C
- Permite buscar y seleccionar nodos de XML

Directivas

Expresiones de ruta	Descripción
Nombre_del_nodo	Selecciona los nodos con el nombre especificado
/	Selecciona el nodo raíz
//	Selecciona los nodos que coinciden con el nodo actual
.	Selecciona el nodo actual
..	Selecciona el nodo padre del actual
@	Selecciona atributos

Ejemplos

Expresiones	Significado
/Almacén/películas[1]/director	Se selecciona el director del nodo de películas 0, en este caso James Cameron. En ocasiones, dependiendo del navegador, los nodos se empiezan a contar desde el 0.
/Almacén/películas[last()]/director	El director de la última película.
/Almacén/películas[last()-1]/director	El director de la penúltima película.
//titulo[@lang="es"]	Solo los títulos de películas en español (atributo "es").
/Almacén/películas[año>1990]/director	Los directores de las películas del año 1991 en adelante.
/Almacén/películas[position()<3]/director	Los directores de las dos primeras películas.



Uso XPath



```
try {
    DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = builderFactory.newDocumentBuilder();
    Document xmlDocument = builder.parse("src/Xpath/tutorials.xml");

    XPathFactory xPathfactory = XPathFactory.newInstance();
    XPath xPath = xPathfactory.newXPath();

    String expression = "/Tutoriales/Tutorial";
    NodeList nodeList = (NodeList) xPath.compile(expression).evaluate(xmlDocument, XPathConstants.NODESET);

    System.out.println("/Tutoriales/Tutorial");
    for (int i = 0; i < nodeList.getLength(); i++) {
        Node nodo = nodeList.item(i);
        System.out.println(nodo.getTextContent());
    }

    System.out.println("/Tutoriales/Tutorial[@id='01']");
    String id = "01";
    expression = "/Tutoriales/Tutorial[@id=" + "'" + id + "'" + "]";
    Node node = (Node) xPath.compile(expression).evaluate(xmlDocument, XPathConstants.NODE);
    System.out.println(node.getTextContent());

    System.out.println("//Tutorial[descendant::title[text()='xml']]");
    String name = "XML";
    expression = "//Tutorial[descendant::title[text()=" + "'" + name + "'" + "]]";
    NodeList lista = (NodeList) xPath.compile(expression).evaluate(xmlDocument, XPathConstants.NODESET);

    for (int i = 0; i < lista.getLength(); i++) {
        Node nodo = lista.item(i);
        System.out.println(nodo.getTextContent());
    }
} catch (Exception e) {
    e.printStackTrace();
}
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<Tutoriales>
  <Tutorial id="01" type="java">
    <title>Clases</title>
    <description>Introducción a las clases</description>
    <date>20/06/2020</date>
    <author>JavaAuthor</author>
  </Tutorial>
  <Tutorial id="02" type="java">
    <title>XML</title>
    <description>Introducción a XPath</description>
    <date>01/12/2020</date>
    <author>XMLAuthor</author>
  </Tutorial>
  <Tutorial id="03" type="java">
    <title>XML</title>
    <description>Uso de validadores XSD</description>
    <date>15/01/2021</date>
    <author>XMLAuthor</author>
  </Tutorial>
</Tutoriales>
```

```
Problems @ Javadoc Declaration Console X
<terminated> Ejemplo (1) [Java Application] C:\Software\Eclip
/Tutoriales/Tutorial

Clases
Introducción a las clases
20/06/2020
JavaAuthor

XML
Introducción a XPath
01/12/2020
XMLAuthor

XML
Uso de validadores XSD
15/01/2021
XMLAuthor
```

```
Problems @ Javadoc Declaration Console X
<terminated> Ejemplo (1) [Java Application] C:\Software\Eclip
/Tutoriales/Tutorial[@id='01']

Clases
Introducción a las clases
20/06/2020
JavaAuthor
```

```
Problems @ Javadoc Declaration Console X
<terminated> Ejemplo (1) [Java Application] C:\Software\Eclip
//Tutorial[descendant::title[text()='xml']]

XML
Introducción a XPath
01/12/2020
XMLAuthor

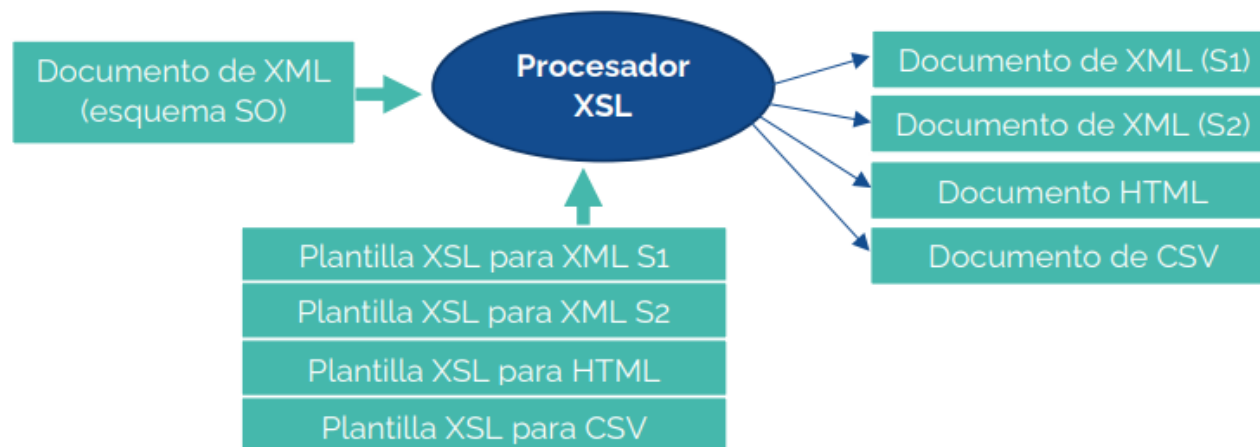
XML
Uso de validadores XSD
15/01/2021
XMLAuthor
```

XSL

¿Qué es?

- Proceso para generar documentos HTML, XML o CSV
- Es necesario una plantilla para especificar el formato de salida. **XSL**
- Los datos de entrada se originan por XML

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:template match="/">
    <html>
      <body>
        <h1>Detalles de empleados</h1>
        <table border="1">
          <tr>
            <th>Id</th>
            <th>Nombre</th>
            <th>Edad</th>
            <th>Salario</th>
          </tr>
          <xsl:for-each select="Empleados/Empleado">
            <tr>
              <td>
                <xsl:value-of select="Id" />
              </td>
              <td>
                <xsl:value-of select="Nombre" />
              </td>
              <td>
                <xsl:value-of select="Edad" />
              </td>
              <td>
                <xsl:value-of select="Salario" />
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```



XSL. Transformación a HTML



```
try {
    TransformerFactory tFactory = TransformerFactory.newInstance();

    Source xslDoc = new StreamSource("src/XSL/empleados.xsl");
    Source xmlDoc = new StreamSource("src/XSL/empleados.xml");

    String outputFileName = "src/XSL/empleados.html";

    OutputStream htmlFile = new FileOutputStream(outputFileName);
    Transformer transform = tFactory.newTransformer(xslDoc);
    transform.transform(xmlDoc, new StreamResult(htmlFile));
} catch (Exception e) {
    e.printStackTrace();
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform" >
  <xsl:template match="/">
    <html>
      <body>
        <h1>Detalles de empleados</h1>
        <table border="1">
          <tr>
            <th>Id</th>
            <th>Nombre</th>
            <th>Edad</th>
            <th>Salario</th>
          </tr>
          <xsl:for-each select="Empleados/Empleado">
            <tr>
              <td>
                <xsl:value-of select="Id" />
              </td>
              <td>
                <xsl:value-of select="Nombre" />
              </td>
              <td>
                <xsl:value-of select="Edad" />
              </td>
              <td>
                <xsl:value-of select="Salario" />
              </td>
            </tr>
          </xsl:for-each>
        </table>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Empleados>
  <Empleado>
    <Id>001</Id>
    <Nombre>Alfonso</Nombre>
    <Edad>30</Edad>
    <Salario>20000</Salario>
  </Empleado>
  <Empleado>
    <Id>002</Id>
    <Nombre>Resu</Nombre>
    <Edad>25</Edad>
    <Salario>30000</Salario>
  </Empleado>
  <Empleado>
    <Id>003</Id>
    <Nombre>Miguel</Nombre>
    <Edad>25</Edad>
    <Salario>26000</Salario>
  </Empleado>
</Empleados>
```

Detalles de empleados

Id	Nombre	Edad	Salario
001	Alfonso	30	20000
002	Resu	25	30000
003	Miguel	25	26000



Resumen

1. XML
2. XML: Estructura y sintaxis
3. DOM
4. DOM. Parsing
5. DOM. Crear documentos y serialización
6. SAX
7. SAX. Parsing
8. Validación de documentos XML
9. Validación. DTD
10. Validación. Esquemas XSD
11. JAXB
12. JAXB. Crear clases a partir de XSD
13. JAXB. Crear objetos a partir de XML
14. JAXB. Crear XML a partir de objetos
15. XPath
16. Uso XPath
17. XSL
18. XSL. Transformación a HTML

The background is a solid blue color. Overlaid on this are several faint, light-blue geometric patterns. These include a grid of small squares that form larger, irregular shapes, and numerous small, light-blue arrows pointing in various directions, some of which are slightly blurred, giving a sense of motion or data flow.

UNIVERSAE

— CHANGE YOUR WAY —