XML

Rodrigo García Carmona
Universidad San Pablo-CEU
Escuela Politécnica Superior

INTRODUCCIÓN A XML

EL LENGUAJE XML

- **XML:** Extensible Markup Language
  - Estándar para la representación y envío de información
  - Formato de documento similar a HTML
    - Las etiquetas describen el contenido…
    - **…no la forma en la que es formateado**.
  - Es un lenguaje apto para *streaming*.
- Seguir explicación con el ejemplo ***Bookstore-noDTD.xml***

REGLAS DE XML

- Un fichero XML es un texto sobre el cual se superponen etiquetas.
- **Etiqueta:** marca sobre un elemento.
  - Delimitada por “<“ y “>”.
  - Se cierra con “/” delante de “<“.
- **Bloque:** zona delimitada por el inicio-final de una etiqueta.
  - Ejemplo: <Title> </Title>
- **Atributo:** información adicional asociada a una etiqueta.
  - Delimitada por las comillas (“”).
  - Si aparecen deben llevar un valor
  - <Book Price=“100”>
- Todas las etiquetas abren y cierran.
  - <Remark />
  - <Book> </Book>
- **Se distinguen mayúsculas de minúsculas.**

EJEMPLO DE XML

```
<Bookstore> <!-- This is a bookstore -->
   <Book ISBN="ISBN-0-13-713526-2" Price="85" Edition="3rd">
      <Title>A First Course in Database Systems</Title>
      <Authors>
         <Author>
            <First_Name>Jeffrey</First_Name>
            <Last_Name>Ullman</Last_Name>
         </Author>
         <Author>
            <First_Name>Jennifer</First_Name>
            <Last_Name>Widom</Last_Name>
         </Author>
      </Authors>
      <Import />
   </Book>
   <Book ISBN="ISBN-0-13-815504-6" Price="100">
      <Title>Database Systems: The Complete Book</Title>
      <Remark>Buy this book bundled with "A First Course"!</Remark>
         <Authors><Author>
            <First_Name>Hector</First_Name>
            <Last_Name>Garcia</Last_Name>
         </Author>
      </Authors>
   </Book>
</Bookstore>
```

MODELO RELACIONAL VS. XML

Modelo relacional

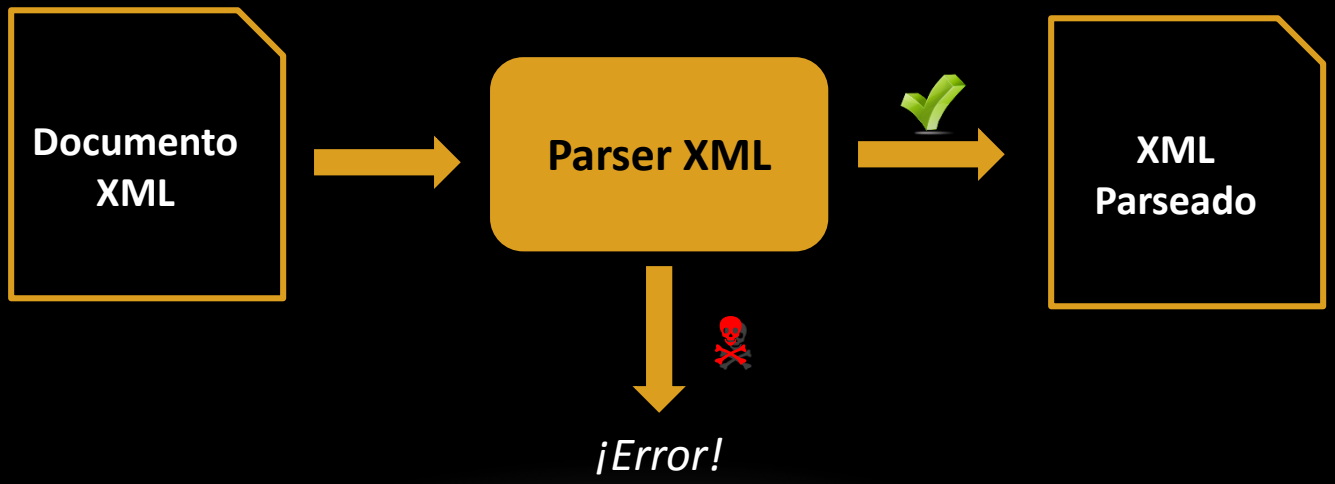
- Estructura:
  - Tablas
- Esquema:
  - Fijado con antelación
- Consultas:
  - Sencillas e intuitivas
- Ordenación:
  - Ninguna
- Implementación:
  - Soporte nativa

XML

- Estructura:
  - Jerárquica en árbol
- Esquema:
  - Flexible, auto-descriptivo
- Consultas:
  - Algo más complicadas…
- Ordenación:
  - Implícita
- Implementación:
  - A través de añadidos

XML BIEN FORMADO

- Un documento XML está **bien formado** (*well-formed*) si se adhiere a unos **requisitos estructurales básicos:**
  - Un único elemento raíz.
  - Etiquetas emparejadas. Anidamiento apropiado.
  - Atributos únicos dentro de cada elemento.
- **Parsers:** DOM, SAX…

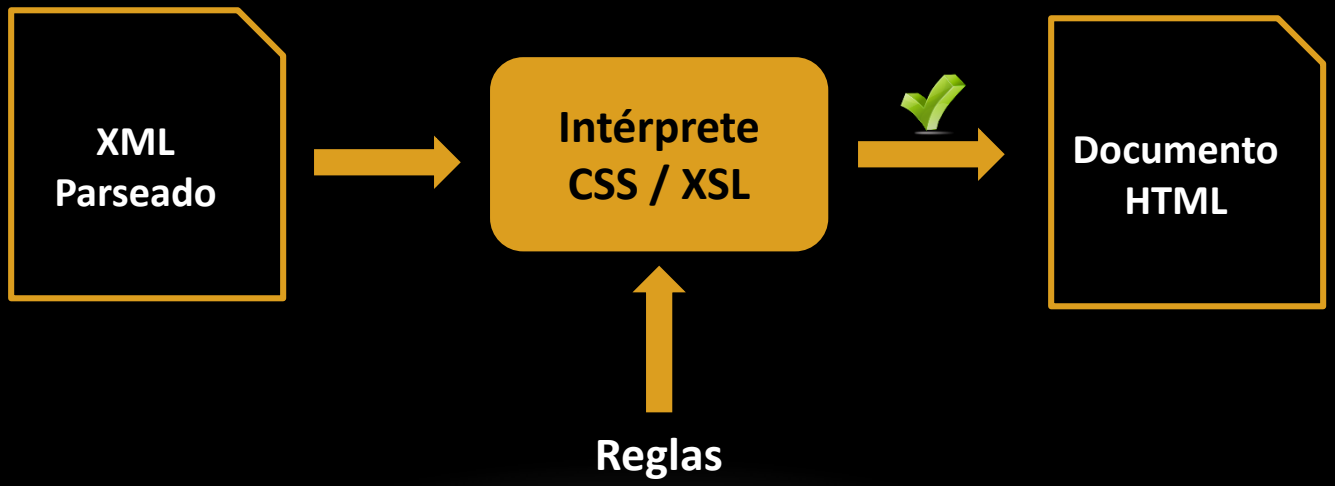


USANDO LIBXML2

- Para trabajar con documentos XML utilizaremos la biblioteca ***libxml2***.
- El programa en su versión para Windows se encuentra en la carpeta compartida de la asignatura.
- Es necesario añadir la carpeta ***bin*** de libxml2 al PATH de Windows.
- Usamos el programa ***xmllint***, de la siguiente forma:
  - `xmllint –-noout ArchivoXML.xml`
- Si está mal formado aparecerán errores.

MOSTRANDO XML

- Se utilizan lenguajes de reglas para transformar XML en HTML:
  - **CSS:** Hojas de estilo en cascada (*Cascading Style Sheets*).
  - **XSL:** Lenguaje de hojas de estilo extensible (*eXtensible Stylesheet Language*).
- Pasa previamente por el parser.



ESTÁNDARES XML

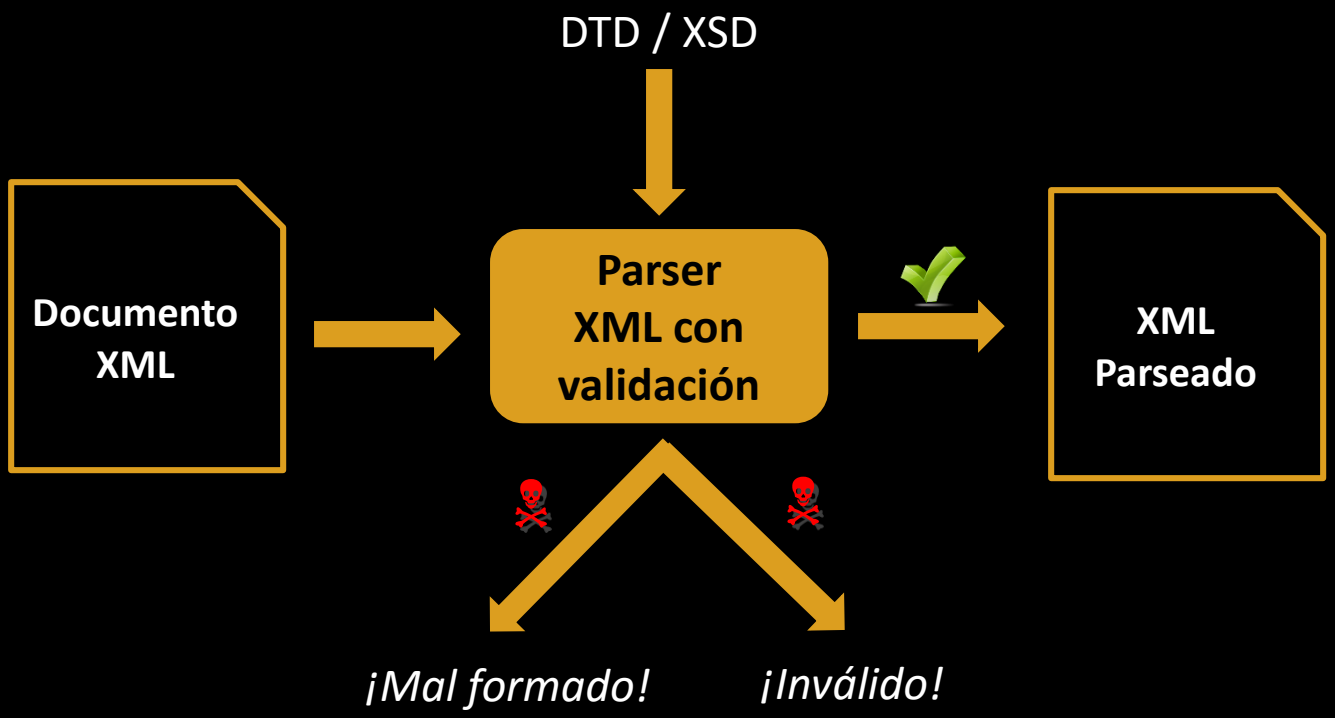
- XML es el estándar de representación e intercambio de datos más extendido.
- Cuenta con un número de estándares asociados enorme.
- En esta asignatura nos centraremos en **los más fundamentales**:
  - DTD
  - XSD
  - XPath
  - Xquery
  - XSL

DTD Y XML SCHEMA

XML VÁLIDO

- Un documento XML está **bien formado** (*well-formed*) si se adhiere a unos **requisitos estructurales básicos:**
  - Un único elemento raíz.
  - Etiquetas emparejadas. Anidamiento apropiado.
  - Atributos únicos dentro de cada elemento.
- Un documento XML es **válido** (*valid*) si se adhiere a unos **requisitos específicos de contenido**. Veremos dos formas de especificar estos requisitos:
  - **DTD:** *Document Type Descriptor*.
  - **XSD:** XML Schema.

VALIDANDO XML



VALIDANDO CON LIBXML2

- También podemos validar documentos XML contra un DTD o un XSD usando ***xmllint***.
- Usamos el programa ***xmllint***, de la siguiente forma:
  - Para validar un XML con un DTD incorporado:
    - `xmllint --noout --valid ArchivoXMLconDTD.xml`
  - Para validar un XML con un DTD en otro archivo:
    - `xmllint` --dtdvalid DTD.dtd `--noout ArchivoXML.xml`
  - Para validar un XML con un XSD en otro archivo:
    - `xmllint` --schema XSD.xsd `--noout ArchivoXML.xml`
- Si no es válido aparecerán errores.

DTD

- Descriptor de tipo de documento (*Document Type Descriptor*).
- Estándar para validar XML.
- **Lenguaje que proporciona una gramática para especificar:**
  - Elementos
  - Atributos
  - Anidado
  - Ordenación
  - Número de apariciones
- Posee tipos de atributos especiales (punteros **sin tipo**):
  - ID
  - IDREF / IDREFS

EJEMPLO DE DTD

```
<!DOCTYPE Bookstore [
   <!ELEMENT Bookstore (Book | Magazine)*>
   <!ELEMENT Book (Title, Authors, Remark?)>
   <!ATTLIST Book ISBN CDATA #REQUIRED
                  Price CDATA #REQUIRED
                  Edition CDATA #IMPLIED>
   <!ELEMENT Magazine (Title)>
   <!ATTLIST Magazine Month CDATA #REQUIRED Year CDATA #REQUIRED>
   <!ELEMENT Title (#PCDATA)>
   <!ELEMENT Authors (Author+)>
   <!ELEMENT Remark (#PCDATA)>
   <!ELEMENT Author (First_Name, Last_Name)>
   <!ELEMENT First_Name (#PCDATA)>
   <!ELEMENT Last_Name (#PCDATA)>
]>
```

XML SCHEMA

- XSD: Estándar para validar XML
- Lenguaje muy amplio, escrito en XML.
- **Lenguaje que proporciona una gramática para especificar:**
  - Elementos
  - Atributos
  - Anidado
  - Ordenación
  - Número de apariciones
  - Tipos de datos
  - Claves
  - Punteros (**con tipo**)
  - …

EJEMPLO DE XSD

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<xsd:element name="Bookstore">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="Book" type="BookType“ minOccurs="0" maxOccurs="unbounded" />
      <xsd:element name="Author" type="AuthorType" minOccurs="0" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:key name="BookKey">
    <xsd:selector xpath="Book" /><xsd:field xpath="@ISBN" />
  </xsd:key>
  <xsd:key name="AuthorKey">
    <xsd:selector xpath="Author" /><xsd:field xpath="@Ident" />
  </xsd:key>
  <xsd:keyref name="AuthorKeyRef" refer="AuthorKey">
    <xsd:selector xpath="Book/Authors/Auth" /><xsd:field xpath="@authIdent" />
  </xsd:keyref>
  <xsd:keyref name="BookKeyRef“ refer="BookKey">
    <xsd:selector xpath="Book/Remark/BookRef" /><xsd:field xpath="@book" />
  </xsd:keyref>
</xsd:element>
```

VENTAJAS E INCONVENIENTES DE DTD/XSD

Ventajas

- Las aplicaciones pueden asumir que existe una estructura concreta.
- Se puede usar CSS/XML para dar forma a esa estructura.
- Es más fácil escribir documentación.
- Resto de ventajas del tipado fuerte.

Inconvenientes

- Un documento XML simplemente bien formado es más flexible y fácil de modificar.
- Los DTDs /XSDs pueden llegar a ser muy complejos y difíciles de manipular.
- Resto de desventajas del tipado débil.

XPATH

XPATH: RUTAS

- Hay que pensar en el documento XML como en un árbol.
- **Construcciones básicas para rutas:**
  - /: Separador:
  - //: Yo, y cualquier elemento descendiente.
  - /Book: Etiquetas.
  - /@ISBN: Atributos.
    - Se obtiene su valor con /data(@ISBN)
  - |: OR lógico, se usa con paréntesis.
  - *: Comodín.

XPATH: CONDICIONES

- **Condiciones:**
  - []: Separador de condiciones.
    - Se pueden anidar para agrupar condiciones.
    - Incluyen un “/” implícito.
  - [Remark]: Existencia.
  - <, >, =, !=: Comparadores.
  - and: AND lógico, enlaza condiciones.
  - or: OR lógico, enlaza condiciones.
  - [número]: Contador.

XPATH: FUNCIONES

- **Funciones** incluidas en XPath:
  - Cuenta con multitud.
  - Dentro de condiciones.
  - Ejemplos:
    - Contiene: contains(elemento, “texto”)
    - Nombre: name()
    - Contador: count()

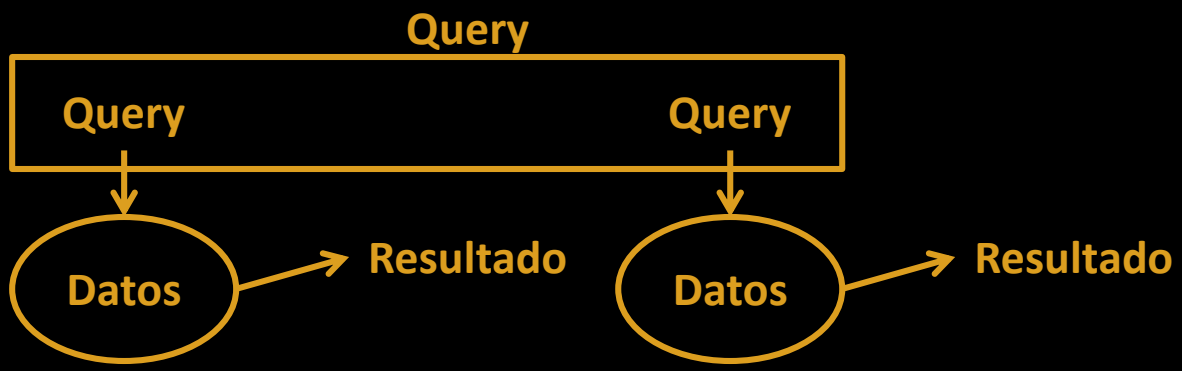
XPATH: EJES DE NAVEGACIÓN

- **Ejes de navegación** en Xpath:
  - Incluye 13.
  - Ejemplos:
    - Padre: parent::
    - Hermano que le precede: preceding-sibling::
    - Hermano que le siga: following-sibling::
    - Descendientes: descendents:
    - Propia etiqueta: self:

XQUERY

XQUERY

- Lenguaje para componer expresiones.
- Cada expresión **opera sobre** y **devuelve** una secuencia de elementos:
  - Documento XML o *Stream* XML.
- XPath es un subconjunto de XQuery.
- XPath es uno de los tipos de expresiones que soporta.
  - Podemos incluir XPath dentro de XQuery.



XSLT

XSLT

- **XSL:** *Extensible Stylesheet Language*: Versión inicial.
- **XSLT:** *XSL (with) Transformations*: Versión mejorada.
- XSLT se escribe utilizando XML.
- Estructura el documento en nodos: elementos, atributos, texto, comentarios…
- Sirve para encontrar partes de un documento XML (usando XPath) y reemplazarlas por otras.
- Busca conforme a una plantilla y reemplaza el resultado entero.
- Pueden aplicarse plantillas recursivamente.
- Usa construcciones típicas de los lenguajes de programación:
  - Condicionales: (if-else)
  - Iteraciones: (for-each)
- Al usarlo se debe tener cuidado con:
  - Comportamientos extraños con los espacios en blanco.
  - Prioridad implícita de las plantillas.

ELEMENTOS XSLT (I)

- **<xsl:template>**
  - Para construir plantillas.
  - El contenido es lo que se escribe a la salida.
  - También se puede usar para descartar.
  - El atributo **match** la asocia a un nodo o nodos del XML.
  - El valor de match es una expresión XPath.
  - match="/" cubre todo el documento.
  - match="text()" cubre el texto, ni las etiquetas ni los atributos.
  - match="*|@*|text()" cubre todo el documento, pero cada ente por separado.
  - Cuidado con el orden de las plantillas.
- **<xsl:value-of>**
  - Extrae el valor de un nodo.
  - El atributo **select** especifica lo que se extrae.
  - El valor de select es una expresión XPath.

ELEMENTOS XSLT (II)

- **<xsl:for-each>**
  - Extrae todos los valores de un nivel.
  - El atributo **select** especifica lo que se extrae.
  - El valor de select es una expresión XPath.
- **<xsl:sort>**
  - Ordena los valores.
  - Se usa dentro de un for-each.
  - El atributo **select** especifica lo que se ordena.
  - El valor de select es una expresión XPath.
- **<xsl:if>**
  - Impone una condición.
  - Se usa dentro de un for-each.
  - El atributo **test** especifica lo que se comprueba.

INTEROPERABILIDAD ENTRE XML Y JAVA

XML Y UML

- **XML es un lenguaje adecuado para:**
  - Interoperabilidad entre plataformas.
  - Envío de información *(streaming).*
  - Representación de la información en forma de árbol.
- **UML es un lenguaje adecuado para:**
  - Diseño de alto nivel.
  - Manejo de la información en lenguajes orientados a objetos.
  - Representación de la información en forma visual.
- Resulta interesante **poder transformar información de XML a UML y viceversa**.
  - Podemos utilizar la solución más adecuada en cada momento.
  - Explicaremos cómo llevar a cabo estas transformaciones usando Java.
    - La traducción entre Java y UML es inmediata.
    - Es extrapolable a cualquier otro lenguaje orientado a objetos.

XML Y JAVA

- Java ofrece varias bibliotecas para trabajar con XML.
- Tenemos dos alternativas a la hora de trabajar con XML en Java:
  - **Procesar XML directamente:**
    - Utilizaremos **JAXP**: Java API for XML Processing.
  - **Traducir entre documentos XML y objetos Java:**
    - Utilizaremos **JAXB**: Java Architecture for XML Binding.
- JAXP y JAXB son las bibliotecas más populares para trabajar con XML en Java, aunque no son las únicas.
- También podemos utilizar Java junto a los estándares XML que ya hemos visto:
  - XPath.
  - XQuery.
  - XSLT.

JAXB

- **JAXB:** Java Architecture for XML Binding
- Está incluido en la JDK estándar de Java a partir de la versión 6.
- Permite realizar dos operaciones:
  - **Marshalling:** Convertir objetos Java en documentos XML.
  - **Unmarshalling:** Convertir documentos XML en objetos Java.
- **Usamos las anotaciones definidas en el estándar JAXB:**
  - *@XmlRootElement:* java.xml.bind.annotation.XmlRootElement
  - *@XmlElement:* java.xml.bind.annotation.XmlElement
  - *@XmlElementWrapper:* java.xml.bind.annotation.XmlElementWrapper
  - *@XmlAttribute:* import java.xml.bind.annotation.XmlAttribute
  - *@XmlType:* java.xml.bind.annotation.XmlType
  - *@XmlTransient:* java.xml.bind.annotation.XmlTransient

ANOTANDO CLASES JAVA (I)

- Tendremos que anotar, usando las anotaciones ya presentadas, las clases Java que queremos que representen los elementos del documento XML.
- **Las clases tienen que cumplir las siguientes condiciones:**
  - Poseer un constructor sin parámetros o un método factoría.
  - Métodos *get* y *set* públicos para los atributos anotados de la clase.
- Anotamos la clase que va a representar **el elemento raíz del XML** con *@XmlRootElement*
  - Atributos opcionales:
    - *name:* nombre de la etiqueta XML.
- Ejemplo de uso:
  - *@XmlRootElement(name = "book“)*

ANOTANDO CLASES JAVA (II)

- Opcionalmente, anotamos todas las clases **tanto la que van a representar el elemento raíz del XML como las que no**, con *@XmlType* para especificar el orden de las etiquetas*:*
  - Atributo *propOrder:* especifica el orden en el que tienen que aparecer las etiquetas hijas en el XML.
    - En *propOrder* ponemos el nombre de los atributos Java, no de los elementos del documento XML.
- Ejemplo de uso independiente:
  - *@XMLType(propOrder = { “firstName", “lastName" })*
- Ejemplo de uso en clase con *@XmlDocumentRoot*:
  - *@XmlRootElement(name = "book“)*
    @XMLType(propOrder = { “title", “author", “price" })

ANOTANDO ATRIBUTOS

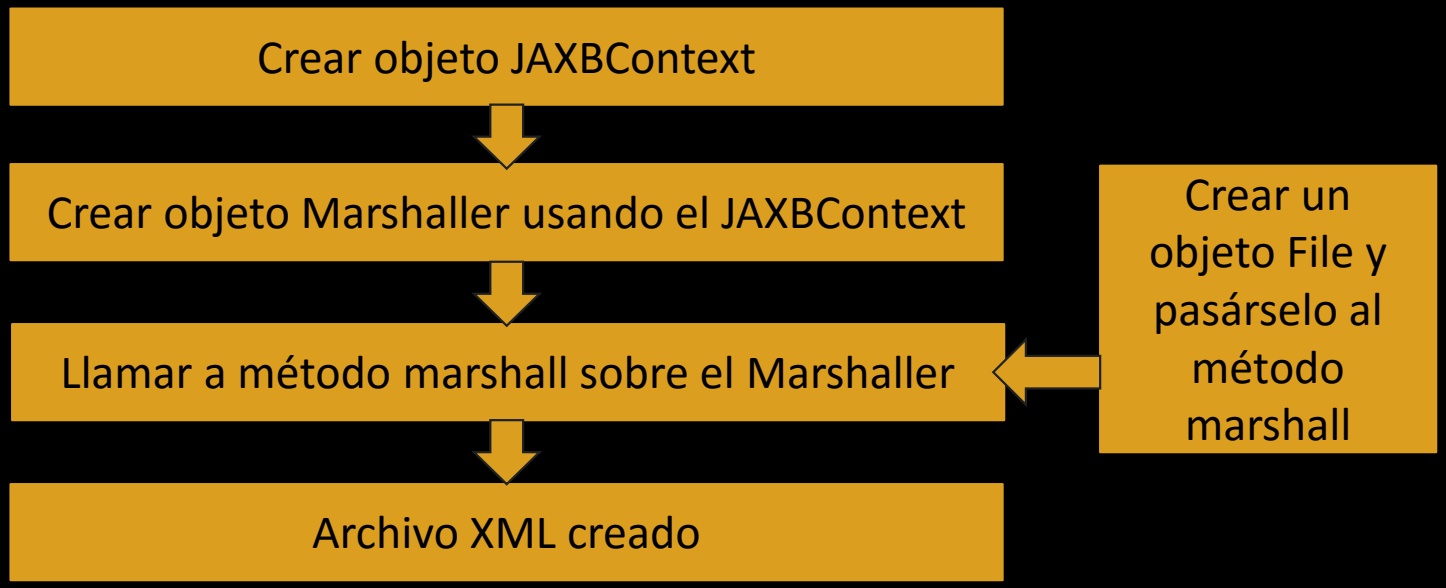
- Cada uno de los atributos de la clase que deba estar representado en el XML deberá estar anotado de una de las dos siguientes formas:
  - ***@XmlElement:*** Si se trata de una etiqueta XML.
  - ***@XmlAttribute:*** Si se trata de un atributo XML.
  - Ambas pueden tener los siguientes atributos opcionalmente:
    - *name*: para especificar el nombre de la etiqueta.
    - *required: true/false* para indicar si el elemento es obligatorio.
- Cuando se anota un atributo que **es una List,** utilizamos la anotación ***@XmlElementWrapper:***
  - Con el atributo *name* para especificar el nombre.
- Se usa en conjunción con *@XmlElement*.
- Ejemplo de uso:
  - ```
    @XmlElement(name = "Auth", required = true)
    @XmlElementWrapper(name = "Authors")
    private Vector<Auth> authorsVector;
    ```
    ```

EXCLUYENDO ATRIBUTOS DEL XML

- Si queremos que un determinado atributo de la clase **no aparezca en el XML** también tendremos que etiquetarlo.
- Usamos la anotación *@XmlTransient*.
- Ejemplo de uso:
  - @XmlTransient
    private String internalID;

MARSHALLING

- El *marshalling* es un proceso compuesto de los siguientes pasos:

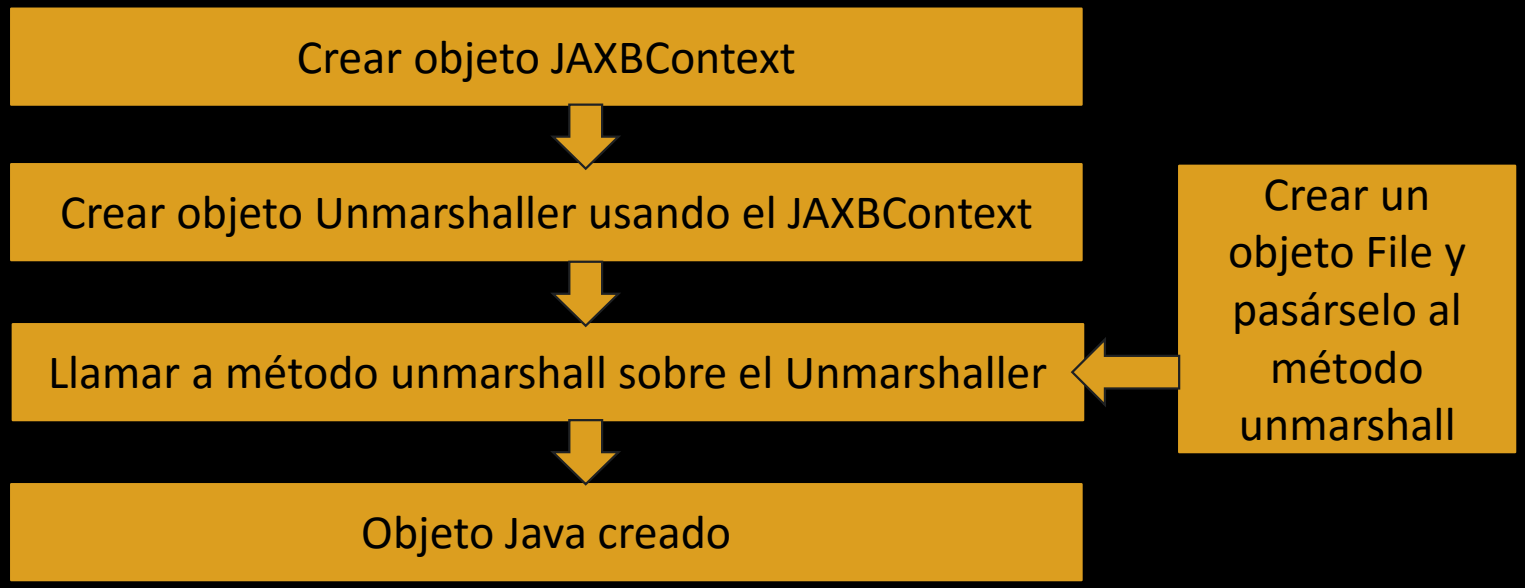


EJEMPLO DE MARSHALLING

```
// Creamos el objeto
Book book = new book();
book.setName(“Dune”);
Book.setAuthor(“Frank Herbert”)
// Creamos el JAXBContext
JAXBContext jaxbC = JAXBContext.newInstance(Book.class);
// Creamos el JAXBMarshaller
Marshaller jaxbM = jaxbC.createMarshaller();
// Formateo bonito
jaxbM.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,Boolean.TRUE);
// Escribiendo en un fichero
File XMLfile = new File(“Book.xml");
jaxbM.marshal(book, XMLfile);
// Escribiendo por pantalla
jaxbM.marshal(book, System.out);
```

UNMARSHALLING

- El *unmarshalling* es un proceso compuesto de los siguientes pasos:



EJEMPLO DE UNMARSHALLING

```
// Creamos el JAXBContext
JAXBContext jaxbC = JAXBContext.newInstance(Book.class);
// Creamos el JAXBMarshaller
Unmarshaller jaxbU = jaxbC.createUnmarshaller();
// Leyendo un fichero
File XMLfile = new File(“Book.xml");
// Creando el objeto
Book book = (Book) jaxbU.unmarshal(book, XMLfile);
// Escribiendo por pantalla el objeto
System.out.println(book);
```