class: inverse, center, middle

.training360-logo[Training360]

# JAVA-XML - XML kezelés és feldolgozás Java eszközökkel

## Tematika

* XML szerepe, szerkezete, well formed
* DOM, SAX, StAX
* XML tesztelés
* XML névterek
* DTD, XML séma, valid fogalma
* XPath
* XSLT, XML transzformációk
* JAXB

## Források

* Brett McLaughlin, Justin Edelson: Java and XML: Solutions to Real-World Problems 3rd Edition (2006, O'Reilly)
* https://www.w3schools.com/ - tutoriálok
* https://www.w3.org/ - referenciák

class: inverse, center, middle

.training360-logo[Training360]

# XML technológiák áttekintés

## XML formátum

* Általános célú leíró nyelv (W3C szabvány)
* Alapja az SGML, platform független leírónyelv
* Bővíthető
* Ember és számítógép által is feldolgozható szöveges formátum
* Fa hierarchia
* Adatátvitelre, és nem adattárolásra tervezték (platformfüggetlenség)

## Példa XML állomány

<?xml version="1.0" encoding="UTF-8" standalone="no"?>  
<!-- Catalog of books -->  
<catalog>   
 <book isbn10="059610149X">  
 <title>Java and XML</title>  
 <available />  
 </book>  
 <book isbn10="1590597060">  
 <title>Pro XML Development  
 with Java Technology</title>  
 </book>  
</catalog>

## Névterek

<?xml version="1.0" encoding="UTF-8" standalone="no"?>  
<!-- Catalog of books -->  
<c:catalog xmlns:c="http://training360.com/schemas/catalog"  
 xmlns:s="http://training360.com/schemas/stock">   
 <c:book isbn10="1590597060">  
 <c:title>Pro XML Development with Java Technology</c:title>   
 <s:stock>10</s:stock>  
 </c:book>  
</c:catalog>

## XML szerkesztő

* Legtöbb szövegszerkesztő támogatja
  + Pl. Notepad++ XML Tools Plugin
* Fejlesztőeszközök támogatják
* Altova XMLSpy
* Parancssori eszköz: XMLStarlet

## Validáció

* Szabályrendszerek
  + Elemek (*tag*) nevei
  + Mit tartalmazhatnak (más elemeket, vagy szövegeket - ezekhez típus is rendelhető)
  + Tartalmazott elemek multiplicitása
  + Attribútumok nevei - típus is rendelhető
* Séma: a dokumentumok felépítését és formáját leíró formális nyelv
* Valid, ha megfelel a sémának
* Csak jól formázott XML dokumentumot lehet validálni
* Séma leírása külön dokumentumban, DTD vagy XML séma (XSD)

## Példa XML séma

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">  
 <xs:element name="catalog">  
 <xs:annotation>  
 <xs:documentation>Catalog of books</xs:documentation>  
 </xs:annotation>  
 <xs:complexType>  
 <xs:sequence>  
 <xs:element name="book" minOccurs="0" maxOccurs="unbounded">  
 <xs:complexType>  
 <xs:sequence>  
 <xs:element type="xs:string" name="title"/>  
 <xs:element type="xs:string" name="available" minOccurs="0"/>  
 </xs:sequence>  
 <xs:attribute type="xs:string" name="isbn10" use="optional"/>  
 </xs:complexType>  
 </xs:element>  
 </xs:sequence>  
 </xs:complexType>  
 </xs:element>  
</xs:schema>

## XPath

* W3C szabvány
* Egy XML dokumentum elemei és attribútumai közötti navigációt biztosítja
* XPath szintaktika segítségével definiálhatjuk az XML dokumentum részeit
* Kifejezések segítségével mozoghatunk az XML dokumentumban
* Tartalmaz egy standard függvénykönyvtárat

## Példa XPath kifejezések

* catalog – csomópont neve, kiválasztja az összes catalog elemet
* /catalog – kiválasztja a gyökérelem alatt lévő összes catalog elemet
* catalog/book – kiválasztja a catalog elem összes book gyerek elemet
* //book – kiválasztja a dokumentum összes book elemét, függetlenül annak alhelyezkedésétől
* catalog//book – kiválasztja a catalog elem összes leszármazott book elemét
* //@lang – kiválasztja az összes lang nevű attribútumot

## Példa XSLT állomány

<?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>  
 <h2>Catalog</h2>  
 <table>  
 <xsl:for-each select="/catalog/book">  
 <tr>  
 <td><xsl:value-of select="title"/></td>  
 <td><xsl:if test="available">available</xsl:if></td>  
 </tr>  
 </xsl:for-each>  
 </table>  
 </body>  
 </html>  
</xsl:template>  
</xsl:stylesheet>

## Java XML kezelés

* JAXP (cserélhető implementációk, pluggable framework, factory)
* DOM
* SAX
* StAX (Java 6-tól): pull parser; kétszintű: cursor és iterator API
* XSLT, XPath

## JAXP verziók

* Java 8 - JAXP 1.6 - JSR 206
* https://en.wikipedia.org/wiki/Java\_API\_for\_XML\_Processing
* Xerces (DOM, SAX, StAX)
* [Xalan](https://xalan.apache.org/) (XSLT, XPath)
* StAX - JSR 173

class: inverse, center, middle

.training360-logo[Training360]

# XML formátum

## XML szintaktika

* Szigorú formai követelmények az értelmezhetőség miatt
* Elemek és attribútumaik (elemek nem előre definiáltak)
* Egy gyökér elem lehet
* Nyitó elemhez tartoznia kell egy lezáró elemnek is (vagy önzáró elem)
* A elemek egymásba ágyazhazók, de nem lehet átfedés
* Elemeknek lehet szöveges tartalmuk
* Jól formáltság (well-formed)
* Kis- és nagybetű különbség

## XML állomány értelmezése

* XML bevezetés (prolog)
* Fa hierarchia
* Attribútum értékek aposztrófok vagy idézőjelek között
  + Keverhető
  + Amelyikkel kezdődik, azzal kell lezárni
* Sortörések szöveges tartalomként jelennek meg
* Megjegyzések

## Elem vagy attribútum?

* Attribútumok kiválthatóak elemekkel
* Attribútumok nem tartalmazhatnak komplex értékeket, struktúrált értékeket, megjegyzéseket, stb.

## Elem elnevezési szabályok

* Tartalmazhatnak betűket, számokat és más karaktereket
* Nem kezdődhetnek számmal vagy írásjellel
* Nem kezdődhetnek az xml (vagy XML vagy Xml, stb.) karaktersorozattal
* Nem tartalmazhatnak szóközt
* Bármilyen név használható, nincs lefoglalt szó

## Elem elnevezési konvenciók

* Beszédesek, főleg főnevek, a kötőjellel elválasztott nevek jól használhatóak
* Legyenek rövidek és egyszerűek
* Kerüljük az alávonás használatát
* Kerüljük a pont használatát
* Kerüljük el a kettőspont használatát, ezt speciális esetekben (névtér megadása) használjuk csak
* Az angol ábécén kívüli karakterek szabályosak, de lehet, hogy az értelmező szoftver nem támogatja őket

## XML entity reference

* XML entity reference: olyan karaktersorozatok, melyeket ki kell cserélni másra a parse során
* Speciális karakterek karakterkóddal is megadhatóak, numeric character reference
  + Pl. &#245 decimális vagy &#xF5 hexadecimális számrendszerben
* Pl. a &lt;, mely a < karakterhez tartozó entity reference, előre definiáltak:
  + &lt; - <, &gt; - >, &amp; - &, &apos; - ', &quot; - "

class: inverse, center, middle

.training360-logo[Training360]

# DOM

## DOM

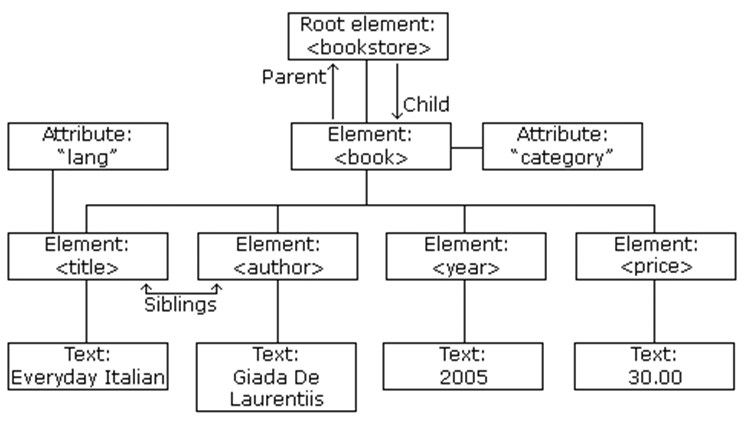
* Document Object Model
* W3C standard
* Platform és nyelv független interfész
* A dokumentum elemeinek a tulajdonságait és objektumait, valamint a hozzáférésükhöz használható metódusokat adja meg
* Fa hierarchiát kezel (bejárás és módosítás)
* Egyszerűbb implementálni (fabejárási algoritmusok)
* Nagy memóriaigény

## XML DOM

* A teljes dokumentum egy dokumentum csomópont
* Minden XML elem egy elem csomópont
* Az XML elemekben lévő szöveg mind szöveg csomópont
* Minden attribútum egy attribútum csomópont
* A megjegyzések megjegyzés csomópontok
* Interfész-hierarchia, API

## DOM interfészek

## DOM példa



DOM példa

## Csomópontok tulajdonságai

* Minden csomópont egy objektum
* Fontos tulajdonságok
  + nodeName
  + nodeValue
  + nodeType

## nodeName

* Csak olvasható
* Elem vagy attribútum (Element vagy Attr) csomópontnál a nodeName megegyezik az elem vagy attribútum nevével
* Szöveges csomópontnál (Text) mindig #text a neve
* Dokumentum (Document) csomópontnál mindig #document

## nodeValue

* Elem csomópontoknál (Element) az érték nem definiált
* Szöveg (Text) csomópontoknál az érték maga a szöveg
* Attribútum csomópontoknál (Attr) az érték az attribútum értéke

## nodeType

* Csak olvasható
* Konstansokkal

## Java és a DOM

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();  
DocumentBuilder documentBuilder = factory.newDocumentBuilder();  
Document document = documentBuilder.parse(inputStream);

* ParserConfigurationException, SAXException, IOException ellenőrzött kivételek
* Egy Document Node-nak egy Element gyermeke van, a gyökér elem, de lehet pl. több Comment gyermeke

## Node metódusok

* String getNodeName(), String getNodeValue(), short getNodeType()
* Node getParentNode(), Node getNextSibling(), Node getPreviousSibling()
* NodeList getChildNodes(), Node getFirstChild(), Node getLastChild()
* NameNodeMap getAttributes()
* String getTextContent()

## NodeList interfész

* Nem Iterable
* int getLength() és Node item(int index)

NodeList l = document.getElementsByTagName("book");  
for (int i = 0; i < l.getLength(); i++) {  
 Element bookElement = (Element) l.item(i);  
}

## Element metódusok

* NodeList getElementsByTagName(String)
* String getAttribute(String)

## XML betöltés tesztelés

* XML Stringként
  + Nehézkes a sortörések miatt
* XML beolvasás classpath-ról

class: inverse, center, middle

.training360-logo[Training360]

# DOM kiírása

## Fa létrehozása, módosítása

* DocumentBuilder.newDocument()
* Element Document.createElement(String)
* Attr Document.createAttribute(String)
* Text Document.createTextNode(String)
* Node.appendChild(Node), insertBefore(Node newChild, Node refChild)
* Node.removeChild(Node)
* Node.replaceChild(Node newChild, Node oldChild)
* Node.cloneNode(boolean deep)

## DOM kiírása

TransformerFactory transformerFactory = TransformerFactory.newInstance();  
Transformer transformer = transformerFactory.newTransformer();  
DOMSource source = new DOMSource(document);  
ByteArrayOutputStream baos = new ByteArrayOutputStream();  
StreamResult result = new StreamResult(baos);  
  
transformer.transform(source, result);  
return new String(baos.toByteArray());

## XML kiírás tesztelés

* XML Stringként
  + Nehézkes a sortörések miatt
  + Baj lehet a formázással (formázatlan, egy sorban; behúzások)
* XML classpath-on
  + Formázás probléma megmarad

## Eszközök XML tesztelésre

* [XMLUnit](https://www.xmlunit.org/)
  + XML összehasonlítás - akár formázás, megjegyzések, sorrend figyelmen kívül hagyásával
  + Validáció
  + XPath alapú tesztelés
  + Hamcrest matchereket is tartalmaz

class: inverse, center, middle

.training360-logo[Training360]

# Haladó DOM

## NodeIterator

* XML elemek sorban történő bejárása
* Nem kötelező implementálni, lekérdezhető

if (!documentBuilder.getDOMImplementation().hasFeature("traversal", "2.0")) {  
 throw new IllegalStateException("Invalid implementation");  
}

## NodeIterator használat

NodeIterator i = ((DocumentTraversal) document).createNodeIterator(document,  
 NodeFilter.SHOW\_ELEMENT, null, true);  
Node node;  
while ((node = i.nextNode()) != null) {  
 if (node.getNodeName().equals("book")) {  
 // ...  
 }  
}

* Node root
* NodeFilter.showXYZ
* NodeFilter interfész, short acceptNode(Node n) implementálandó
* entityReferenceExpansion - entity reference kifejtése

## TreeWalker

* Elemek bejárása fa hierarchiában
* Létrehozása, paraméterezése, mint a NodeIterator esetén

TreeWalker treeWalker = ((DocumentTraversal) document).createTreeWalker(document,  
 NodeFilter.SHOW\_ELEMENT, null, true);

## TreeWalker interfész

* getCurrentNode()
* parentNode(), firstChild(), lastChild()
* previousSibling(), nextSibling()
* previousNode(), nextNode()

## External entity reference

* Létezik external entity reference, ha külső tartalmat szeretnénk betölteni a parse során
  + <!ENTITY copyright SYSTEM "http://www.ibm.com/developerWorks/copyright.xml">
  + Hivatkozni rá &copyright; használatával
  + Itt system id kerül megadásra, mely a külső tartalomra hivatkozik
  + Megadható public id is, mely egy logikai név

## Java EntityResolver

* A Java parsernek megadható EntityResolver implementáció, mely a system és public id alapján maga tölti be a külső tartalmat, pl. url helyett lokális állományból (lokális cache)

https://www.ibm.com/developerworks/library/x-tipent/index.html

## Java EntityResolver implementáció

public class CopyrightResolver implements EntityResolver {  
 public InputSource resolveEntity(String publicID, String systemID) throws SAXException {  
 if (systemID.equals("http://www.ibm.com/developerWorks/copyright.xml")) {  
 return new InputSource("/localCopyright.xml");  
 }  
 return null;  
 }  
}

documentBuilder.setEntityResolver(new CopyrightResolver());

class: inverse, center, middle

.training360-logo[Training360]

# SAX

## SAX

* Dokumentum egy részét is képes feldolgozni
* Un. push parser
* Eseményvezérelt
* Nagyteljesítményű
* Nehezebb implementálni
* DOM is erre épül
* Csak olvasásra

## SAX példakód

SAXParserFactory factory = SAXParserFactory.newInstance();  
SAXParser saxParser = factory.newSAXParser();  
BookSaxHandler handler = new BookSaxHandler();  
saxParser.parse(inputStream, handler);  
return handler.getCatalog();

* ParserConfigurationException, SAXException, IOException ellenőrzött kivételek

## DefaultHandler

* A következő interfészeket implementálja üres metódusokkal
  + ContentHandler - eseménykezelő metódusok, ha a parser a megfelelő xml részhez ér
  + EntityResolver - entity reference feloldása
  + ErrorHandler - hibakezelés
  + DTDHandler - DTD-vel kapcsolatos események kezelésére

## DefaultHandler leszármazott

public class BookSaxHandler extends DefaultHandler {  
  
 // Attribútumokban az adatok gyűjtése  
  
 @Override  
 public void startElement(String uri, String localName, String qName, Attributes attributes)  
 throws SAXException {  
 // ...  
 }  
  
 @Override  
 public void endElement(String uri, String localName, String qName) throws SAXException {  
 // ...  
 }  
  
 @Override  
 public void characters(char[] ch, int start, int length) throws SAXException {  
 // ...  
 }  
}

class: inverse, center, middle

.training360-logo[Training360]

# StAX

## StAX

* Pull parser - iterálás az eseményekre
* Dokumentumot részekben tudja feldolgozni
* Streaming API, a parser az XML dokumentumnak csak egy részét látja egyszerre
* Gyorsabb
* Kevesebb memóriát használ
* Közepesen nehéz implementálni
* Olvasásra és írásra is

## StAX API-k

* Cursor API: alacsonyabb szintű, XMLStreamReader, és XMLStreamWriter
* Iterator API: XMLEventReader és XMLEventWriter

## Cursor API - parse

XMLInputFactory f = XMLInputFactory.newInstance();  
XMLStreamReader r = f.createXMLStreamReader(source);  
  
while (r.hasNext()) {   
 if (r.getEventType() == XMLStreamConstants.START\_ELEMENT) {  
 if ("book".equals(r.getName().getLocalPart())) {  
 String isbn = r.getAttributeValue(null, "isbn10"));  
 }  
 else if ("title".equals(r.getName().getLocalPart())) {  
 String title = r.getElementText();  
 }  
 }  
 r.next();  
}

## Cursor API - kiírás

StringWriter sw = new StringWriter();  
XMLOutputFactory output = XMLOutputFactory.newInstance();  
XMLStreamWriter writer = output.createXMLStreamWriter(sw);  
writer.writeStartDocument();  
writer.writeStartElement("catalog");  
for (Book book: catalog) {  
 writer.writeStartElement("book");  
 writer.writeAttribute("isbn10", book.getIsbn10());  
 writer.writeStartElement("title");  
 writer.writeCharacters(book.getTitle());  
 writer.writeEndElement();  
 writer.writeEndElement();  
}  
writer.writeEndElement();  
writer.flush();  
return sw.toString();

## Iterator API - olvasás

XMLInputFactory f = XMLInputFactory.newInstance();  
XMLEventReader r = f.createXMLEventReader(source);  
  
while (r.hasNext()) {  
 XMLEvent event = r.nextEvent();  
 if (event.getEventType() == XMLStreamConstants.START\_ELEMENT) {  
 if (event instanceof StartElement) {  
 StartElement element = (StartElement) event;  
 if ("book".equals(element.getName().getLocalPart())) {  
 String isbn = element.getAttributeByName(new QName("isbn10")).getValue();  
 }  
 else if ("title".equals(element.getName().getLocalPart())) {  
 String title = r.getElementText();  
 }  
 }  
 }  
}

## Melyiket használjuk?

* A Cursor API kevesebb memóriát használ, és a példányosítások hiánya miatt gyorsabb
* Az Iterator API XMLEvent osztályai nem módosíthatóak (immutable), a parse-olás után is megőrzik az értéküket

## Melyiket használjuk?

* Emiatt többlépéses (bővíthető, plug-inelhető) feldolgozásokat is könnyebben ki lehet dolgozni ezen példányok továbbadásával
* Az XMLEvent interfészt akár magunk is implementálhatjuk, akár teljesen új eseményt hozhatunk létre, akár egy meglévőt egészíthetünk ki utility metódusokkal
* Az Iterator API esetén új eseményeket szúrhatunk be, vagy eseményeket ugorhatunk át

Ha nem a teljesítmény az elsődleges szempont, érdemes a magasabb szintű iterator API-t használni

class: inverse, center, middle

.training360-logo[Training360]

# XML névterek

## XML névterek

* Két különböző dokumentumban ugyanaz a elem név hordozhat más jelentést
* Probléma lehet, ha a dokumentumokat egymásba kell ágyazni
* Névütközés
* Névtér legyen egyedi URL, amivel minősítjük a elemeket
  + Az adott URL-en nem kell tartalomnak lennie
* Teljes URL kiírása helyett használjunk rövidítéseket (prefix)

## Példa névtér

<?xml version="1.0" encoding="UTF-8" standalone="no"?>  
<!-- Catalog of books -->  
<c:catalog xmlns:c="http://training360.com/schemas/catalog">   
 <c:book isbn10="1590597060">  
 <c:title>Pro XML Development with Java Technology</c:title>  
 </c:book>  
</c:catalog>

## Alapértelmezett névtér

<?xml version="1.0" encoding="UTF-8" standalone="no"?>  
<!-- Catalog of books -->  
<catalog xmlns="http://training360.com/schemas/catalog">   
 <book isbn10="1590597060">  
 <title>Pro XML Development with Java Technology</c:title>  
 </book>  
</catalog>

Teljes dokumentumra, vagy részfára

## Névterek használata

<?xml version="1.0" encoding="UTF-8" standalone="no"?>  
<!-- Catalog of books -->  
<c:catalog xmlns:c="http://training360.com/schemas/catalog"  
 xmlns:s="http://training360.com/schemas/stock">   
 <c:book isbn10="1590597060">  
 <c:title>Pro XML Development with Java Technology</c:title>   
 <s:stock>10</s:stock>  
 </c:book>  
</c:catalog>

Alapértelmezett névtérrel kombinálható

## Névterek DOM esetén

* documentBuilderFactory.setNamespaceAware(true)
* root.setAttribute("xmlns:catalog", "http://training360.com/schemas/catalog");
* Node.getNamespaceURI(), Node.getLocalName() metódusok
* getElementsByTagNameNS(String namespaceURI, String localName) metódus
* createElementNS(String namespaceURI, String qualifiedName) metódus
* Node.setPrefix(String) metódus

## Névterek SAX esetén

@Override  
public void startElement(String uri, String localName, String qName, Attributes attributes)  
 throws SAXException {  
 // ...  
}

A uri és localName paraméterek hordozzák

## Névterek StAX esetén

* QName osztály
* String getNamespaceURI() és String getLocalPart() metódusokkal

class: inverse, center, middle

.training360-logo[Training360]

# DTD

## DTD

* DTD = Document Type Definition
* Speciális formátumú nyelv
* Régebbi, elavult
* XML dokumentumon belül is definiálhatjuk a DTD-t
* Külső állományban is tárolhatjuk (újrafelhasználható)
* Belső DTD felülbírálja a külsőt, kaszkád elv

## Belső DTD

<?xml version="1.0" encoding="UTF-8" standalone="no"?>  
<!DOCTYPE catalog [  
<!ELEMENT catalog (book)>  
<!ELEMENT book (title, available)>  
<!ATTLIST book isbn10 CDATA #REQUIRED>  
<!ELEMENT title (#PCDATA)>  
<!ELEMENT available EMPTY>  
]>  
<!-- Catalog of books -->  
<catalog>   
 <book isbn10="059610149X">  
 <title>Java and XML</title>  
 <available />  
 </book>  
</catalog>

## Standalone

* A standalone="no" Használja a DTD-t validálásra
* Nem használja:
  + Attribútumok alapértelmezett értékeinek meghatározására
  + Entity deklarációk feloldására
  + Nem normalizál

## Elemtípusok

* Empty: az elem üres, maximum attribútumai lehetnek
* Element-only: az elemnek csak gyerekeleme lehet
* Mixed: szöveget és gyerekelemet is tartalmazhat
* Any: bármi lehet benne, amit a DTD megenged

## Attribútum alapértelmezett értékek

* #REQUIRED – kötelező az attribútum megadása
* #IMPLIED – az attribútum megadása opcionális
* #FIXED - az attribútum rögzített értékkel rendelkezik
* Az attribútum alapértelmezett értéke

## Attribútum típusok 1.

* CDATA – nem értelmezett (unparsed) szöveges adat
* Enumerated – karakterláncok sorozata
* NOTATION – DTD más pontján megadott jelölés
* ENTITY – külső egyedhivatkozás
* ENTITIES – több külső egyedhivatkozás, üres karakterekkel elválasztva

## Attribútum típusok 2.

* ID – egyedi azonosító
* IDREF – DTD más pontján deklarált ID-ra mutató hivatkozás
* IDREFS – több különböző ID-ra mutató hivatkozás
* NMTOKEN – XML tokenekből felépített név
* NMTOKENS – több XML tokenekből felépített név

XML tokenek: számok, betűk, pontok, kötőjelek, kettőspontok és aláhúzásjelek

## Külső DTD

<?xml version="1.0" encoding="UTF-8" standalone="no"?>  
<!DOCTYPE catalog SYSTEM "catalog.dtd">  
<!-- Catalog of books -->  
<catalog>   
 <book isbn10="059610149X">  
 <title>Java and XML</title>  
 <available />  
 </book>  
</catalog>

<!ELEMENT catalog (book)>  
<!ELEMENT book (title, available)>  
<!ATTLIST book isbn10 CDATA "">  
<!ELEMENT title (#PCDATA)>  
<!ELEMENT available EMPTY>

## DTD és Java

* Nem külön validáció, hanem a parsing process része
* JDK-ban lévő SAX, DOM implementáció támogatja
* JDK-ban lévő StAX implementáció nem támogatja, [Woodstox](https://github.com/FasterXML/woodstox) igen

## DOM DTD ellenőrzéssel

DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();  
factory.setValidating(true);  
DocumentBuilder builder = factory.newDocumentBuilder();  
MyErrorHandler errorHandler = new MyErrorHandler();  
builder.setErrorHandler(errorHandler);  
builder.setEntityResolver(new DtdEntityResolver());  
builder.parse(new InputSource(new StringReader(xml)));

## ErrorHandler

public class MyErrorHandler implements ErrorHandler {  
  
 @Override  
 public void warning(SAXParseException exception) throws SAXException {  
 // ...  
 }  
  
 @Override  
 public void error(SAXParseException exception) throws SAXException {  
 // ...  
 }  
  
 @Override  
 public void fatalError(SAXParseException exception) throws SAXException {  
 // ...  
 }  
}

## EntityResolver

public class DtdEntityResolver implements EntityResolver {  
 @Override  
 public InputSource resolveEntity(String publicId, String systemId)  
 throws SAXException, IOException {  
 return new InputSource(ValidatorApi.class.getResourceAsStream("/catalog.dtd"));  
 }  
}

## SAX DTD ellenőrzéssel

SAXParserFactory factory = SAXParserFactory.newInstance();  
factory.setValidating(true);  
SAXParser saxParser = factory.newSAXParser();  
MyErrorHandler myErrorHandler = new MyErrorHandler();  
XMLReader xmlReader = saxParser.getXMLReader();  
xmlReader.setErrorHandler(myErrorHandler);  
xmlReader.setEntityResolver(new DtdEntityResolver());  
xmlReader.parse(new InputSource(new StringReader(xml)));

class: inverse, center, middle

.training360-logo[Training360]

# XML séma

## XML séma

* XSD szabvány (W3C)
* Önmaga is XML formátumú
* Adattípusok, pl. string, decimal, float, date
* Dokumentálásra is használható
* XML névterek használata

## Példa XML séma

<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">  
 <xs:element name="catalog">  
 <xs:annotation>  
 <xs:documentation>Catalog of books</xs:documentation>  
 </xs:annotation>  
 <xs:complexType>  
 <xs:sequence>  
 <xs:element name="book" minOccurs="0" maxOccurs="unbounded">  
 <xs:complexType>  
 <xs:sequence>  
 <xs:element type="xs:string" name="title"/>  
 <xs:element type="xs:string" name="available" minOccurs="0"/>  
 </xs:sequence>  
 <xs:attribute type="xs:string" name="isbn10" use="optional"/>  
 </xs:complexType>  
 </xs:element>  
 </xs:sequence>  
 </xs:complexType>  
 </xs:element>  
</xs:schema>

## Példa XML séma névtérrel

<?xml version="1.0" encoding="utf-8"?>  
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"  
 targetNamespace="https://training360.com/schemas/catalog"  
 xmlns="https://training360.com/schemas/catalog"  
 elementFormDefault="qualified" >  
 <xs:element name="catalog">  
 <!-- ... -->  
 </xs:element>  
</xs:schema>

## Séma hivatkozás

<catalog xmlns="https://training360.com/schemas/catalog"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation=  
 "https://training360.com/schemas/catalog https://training360.com/schemas/catalog/catalog.xsd">  
 <book isbn10="059610149X">  
 <title>Java and XML</title>  
 <available />  
 </book>  
</catalog>

## XML séma adattípusok

* Karakterláncok: xsd:string
  + Szöveges elemet vagy attribútumot adhatunk meg
* Logikai: xsd:boolean
  + Érték true (1) vagy false (0) lehet
* Számtípusok: (mind xsd: előtaggal) integer, decimal, float, double
  + Kiegészítő típusok: positiveInteger, negativeInteger, nonPositiveInteger, nonNegativeInteger, az utóbbi két típus a nullát is tartalmazza

## XML séma dátum és idő 1.

* xsd:time – tetszőleges időpont (pl. 17:30)
* xsd:timeInstant – időpont teljes leírása (pl. 2008-06-10 14:33)
* xsd:duration – időtartam (pl. P3Y4M2DT13H27M11S)
* xsd:date – dátum (pl. 2008-06-10)

## XML séma dátum és idő 2.

* xsd:month – év és hónap (pl. 2008-06)
* xsd:year – év (pl. 2008)
* xsd:century – század (pl. 20)
* xsd:recurringDate – dátum év nélkül (pl. 06-10)
* xsd:recurringDay – hónap napja hónap és év megadása nélkül (pl. 10)

## Egyéni típusok

<xsd:simpleType name="egytoltizigTipus">  
 <xsd:restriction base="xsd:integer">  
 </xsd:restriction>  
</xsd:simpleType>

* Érték korlátozás
* Számok pontosságának korlátozása
* Érték hosszának korlátozása
* Reguláris kifejezés
* Felsorolt típusok
* Listák

## Érték korlátozás

* xsd:minInclusive – legkisebb megengedett érték
* xsd:minExclusive – az értéknek ennél nagyobbnak kell lennie (alsó határ)
* xsd:maxInclusive – legnagyobb megengedett érték
* xsd:maxExlusive – az értéknek ennél kisebbnek kell lennie (felső határ)
* xsd:precision – a szám teljes hossza maximum
* xsd:scale – maximum hány tizedesjegy lehet a számban

## Hosszúság korlátozás

* xsd:length – pontos hossz
* xsd:minlength – legrövidebb megengedett hossz
* xsd:maxlength – legnagyobb megengedett hossz

## Reguláris kifejezés

* xsd:pattern használatával

## Felsorolásos típus

* a xsd:restriction elemen belül xsd:enumeration elemekként adjuk meg a lehetséges értékeket

## Listák

* xsd:list elemmel hozható létre, ahol megadhatjuk, hogy a lista milyen típusú elemeket és azokból hányat tartalmazhat
  + xsd:length, xsd:minlength, xsd:maxlength hossz szabályozásra

<xsd:element name="dailyValues">  
 <xsd:simpleType>  
 <xsd:list base="xsd:decimal">  
 <xsd:length value="12" />  
 </xsd:list>  
 </xsd:simpleType>  
</xsd:element>

## Összetett típusok

* Üres elemek
* Csak elemeket tartalmazó elemek
* Szöveges
* Kevert tartalmú elemek
* Indikátorok

Létrehozásuk xsd:complexType elemmel történik

## Üres elemek

* Nincs szöveges tartalmuk, sem gyerekelemeik, attribútumaik azonban lehetnek

## Csak elemeket tartalmazó elemek

* Csak gyerekelemeket és attribútumokat tartalmaznak

<xsd:element name="assets">  
 <xsd:complexType>  
 <xsd:element name="automobile" type="automobileType" />  
 </xsd:complexType>  
</xsd:element>

<xsd:complexType name="automobileType">  
 <xsd:complexContent>  
 <xsd:extension base="xsd:anyType">  
 <xsd:attribute name="regnumber" type="xsd:string" />  
 <xsd:attribute name="year" type="xsd:year" />  
 <xsd:attribute name="type" type="xsd:string" />  
 <xsd:attribute name="model" type="xsd:string" />  
 </xsd:extension>  
 </xsd:complexContent>  
</xsd:complexType>

## Szöveges

* xs:simpleContent használatával

## Kevert

* mixed="true" attribútum használatával
* Feldolgozása nehezebb

## Sorrenddel kapcsolatos indikátorok

* xs:all indikátor mutatja, hogy az elemek tetszőleges sorrendben szerepelhetnek
* xs:choice indikátor mutatja, hogy az elemek közül egyet kell választani
* xs:sequence indikátor mutatja, hogy az elemek a meghatározott sorrendben szerepelhetnek

## Előfordulással kapcsolatos indikátorok

* xs:minOccurs - minimum hányszor kell szerepelnie
* xs:maxOccurs - maximum hányszor kell szerepelnie

## Csoport indikátor

* Elemek csoportja, melyet más elemhivatkozásoknál lehet használni

<xs:group name="persongroup">  
 <xs:sequence>  
 <xs:element name="firstname" type="xs:string"/>  
 <xs:element name="lastname" type="xs:string"/>  
 </xs:sequence>  
</xs:group>  
  
<xs:element name="person" type="personinfo"/>  
  
<xs:complexType name="personinfo">  
 <xs:sequence>  
 <xs:group ref="persongroup"/>  
 <xs:element name="country" type="xs:string"/>  
 </xs:sequence>  
</xs:complexType>

## Validálás séma alapján Java-ban

SchemaFactory factory = SchemaFactory.newInstance(XMLConstants.W3C\_XML\_SCHEMA\_NS\_URI);  
Schema schema = factory.newSchema(  
 new StreamSource(ValidatorApi.class.getResourceAsStream("/catalog.xsd")));  
  
Validator validator = schema.newValidator();  
validator.validate(new StreamSource(new StringReader(xml)));

class: inverse, center, middle

.training360-logo[Training360]

# XPath

## Terminológia 1.

* Csomópont (node)
  + Elem
  + Attribútum
  + Szöveg
  + Névtér
  + Feldolgozó utasítás
  + Megjegyzés
  + Dokumentum (gyökér elem)

## Terminológia 2.

* Atomi érték (atomic values): gyerek és szülő csomópont nélküli értékek
* Adat: (item): atomi értékek vagy csomópontok

## Kiválasztás

* A csomópontot egy útvonal követésével vagy lépések sorozatával választhatjuk ki
* catalog – csomópont neve, kiválasztja az összes catalog elemet
* /catalog – kiválasztja a gyökérelem alatt lévő összes catalog elemet
* catalog/book – kiválasztja a catalog elem összes book gyerek elemét
* //book – kiválasztja a dokumentum összes book elemét, függetlenül annak alhelyezkedésétől
* catalog//book – kiválasztja a catalog elem összes leszármazott book elemét
* //@lang – kiválasztja az összes lang nevű attribútumot

## XPath predicates 1.

* /catalog/book[1] - A catalog első book gyerekelemét találja meg
* /catalog/book[last()] - a catalog utolsó book gyerekelemét találja meg
* /catalog/book[last()-1] – a catalog utolsó előtti book gyerekelemét találja meg
* /catalog/book[position()<3] - kiválasztja a catalog első két book gyerekelemét

## XPath predicates 2.

* //title[@lang] - kiválasztja az összes title elemet, aminek van lang attribútuma
* //title[@lang='eng'] – kiválasztja az összes title elemet, aminek eng értékű lang attribútuma van
* /catalog/book[price>35.00] – kiválasztja a catalog összes olyan book elemét, aminél az price attribútum értéke 35.00 felett van
* /catalog/book[price>35.00]/title – kiválasztja a catalog összes olyan book elemének a title elemét, amelyeknél a book elem price attribútumának értéke 35.00 felett van.

## XPath ismeretlen csomópontok

* /bookstore/\* - kiválasztja a bookstore összes gyerek elemét
* //\* - kiválasztja a dokumentum összes elemét
* //title[@\*] – kiválasztja az összes title elemet, ami attribútummal rendelkezik

## Több útvonal kiválasztása

* a | jel segítségével
* //book/title | //book/price – kiválasztja a book elemek összes title és price elemét
* //title | //price – kiválasztja a dokumentum összes title és price elemét
* /bookstore/book/title | //price – kiválasztja a bookstore elem összes book elemének összes title elemét és a dokumentum összes price elemét

## XPath tengelyek 1.

* Az aktuális csomóponthoz viszonyított csomópont csoportot definiálnak
* ancestor – kiválasztja a jelenlegi csomópont összes ősét
* ancestor-or-self – kiválasztja a jelenlegi csomópont összes ősét és az jelenlegi csomópontot
* attribute – kiválasztja az jelenlegi csomópont összes attribútumát
* child – kiválasztja az jelenlegi csomópont összes gyerekét
* descendant – kiválasztja a jelenlegi csomópont összes leszármazottját
* descendant-or-self – kiválasztja a jelenlegi csomópont összes leszármazottját és a jelenlegi csomópontot

## XPath tengelyek 2.

* following – kiválaszt mindent a dokumentumból a jelenlegi csomópont záró eleme után
* following-sibling – kiválasztja az összes testvért a jelenlegi csomópont után
* namespace – kiválasztja a jelenlegi csomópont összen névtér csomópontját
* parent – kiválasztja a jelenlegi csomópont szülőjét

## XPath tengelyek 3.

* preceding – kiválaszt mindent a dokumentumból ami a jelenlegi csomópont nyitó eleme előtt van
* preceding-sibling – kiválasztja a jelenlegi csomópont előtt lévő összes testvért
* self – kiválasztja a jelenlegi csomópontot

## Útvonalak

* Lehet relatív (/ jellel kezdődik) vagy abszolút
* Lépések sorozata, lépések / jellel elválasztva
* Mindegyik lépést az aktuális csomópont készlethez viszonyítva értékel ki
* Minden lépés a következőkből áll: tengely::predicate

## Útvonal példák 1.

* child::book – Kiválasztja az összes olyan book csomópontot, amelyek a jelenlegi csomópont gyerekei
* attribute::lang – kiválasztja a jelenlegi csomópont lang attribútumát
* child::\* - kiválaszja a jelenlegi csomópont összes gyerekét
* attribute::\* - kiválasztja a jelenlegi csomópont összes attribútumát
* child::text() – kiválasztja a jelenlegi csomópont összes szöveg típusú gyerekcsomópontját
* child::node() – kiválasztja a jelenlegi csomópont összes gyerek csomópontját

## Útvonal példák 2.

* descendant::book – kiválasztja a jelenlegi csomópont összes book leszármazottját
* ancestor::book – kiválasztja a jelenlegi csomópont összes book ősét
* ancestor-or-self::book – kiválasztja a jelenlegi csomópont összes book ősét – és a jelenlegi csomópontot is, ha az book csomópont
* child::\*/child::price – kiválasztja a jelenlegi csomópont összes price unokáját

## XPath operátorok

* | - két csomópont készletet számol ki
* + - összeadás, - - kivonás, \* - szorzás, div - osztás
* = - egyenlő
* != - nem egyenlő
* < - kisebb
* <= - kisebb vagy egyenlő
* > - nagyobb
* >= - nagyobb vagy egyenlő
* or - vagy, and - és, mod - modulus (osztás maradéka)

## Csomópontokkal kapcsolatos függévenyek

* position() – csomópont numerikus pozícióját határozza meg
* last() – csomópont halmaz utolsó elemét adja meg
* count() – csomópont halmaz elemeinek számát adja meg

## Karakterláncokat feldolgozó függvények

* concat() – karakterláncok összefűzése
* starts-with() – a karakterlánc a megadott karaktersorozattal kezdődik-e?
* contains() – karakterlánc tartalmazza-e az adott karaktersorozatot?
* substring-before() – megadott karaktersorozat előtti részét adja vissza a karakterláncnak
* substring-after() – megadott karaktersorozat utáni részét adja vissza a karakterláncnak
* substring() – adott pozícióban kezdődő, megadott hosszúságú részt ad vissza
* string-length() – a karakterlánc hosszát adja vissza

## Logikai függvények

* not() – érték tagadása
* lang() – használjuk-e az adott nyelvet a dokumentumban?

## Számokkal kapcsolatos függvények

* ceiling() – decimális számot egészre, felfelé kerekít
* floor() – decimális számot egészre, lefelé kerekít
* round() – decimális számot kerekít
* sum() – számhalmaz elemeit összegzi

## XPath Java-ban

XPathFactory xPathfactory = XPathFactory.newInstance();  
XPath xpath = xPathfactory.newXPath();  
XPathExpression expr = xpath.compile(xpathExpression);  
String value = (String) expr.evaluate(  
 new InputSource(new StringReader(xml)), XPathConstants.STRING);

Visszatérési típusok: NUMBER, STRING, BOOLEAN, NODESET, NODE, rendre Double, String, Boolean, NodeList, Node

## Function resolver

XPathFactory xPathfactory = XPathFactory.newInstance();  
xPathfactory.setXPathFunctionResolver(new XPathFunctionResolver() {  
 @Override  
 public XPathFunction resolveFunction(QName functionName, int arity) {  
 return new XPathFunction() {  
 @Override  
 public Object evaluate(List args) throws XPathFunctionException {  
 return new IsbnValidator().validateIsbn10(  
 ((NodeList) args.get(0)).item(0).getTextContent());  
 }  
 };  
 }  
 });  
XPathExpression expr = xpath.compile("tr:findInvalid(/catalog/book[position() = 1]/@isbn10)");  
return (boolean) expr.evaluate(new InputSource(new StringReader(xml)), XPathConstants.BOOLEAN);

class: inverse, center, middle

.training360-logo[Training360]

# XSLT

## XSLT

* XSLT - XML Stylesheet Language Transformation
* Lehetővé teszi az XML dokumentumok átalakítását más XML dokumentumokká
* Erőteljesen épít az XPath-ra

## Példa XSLT állomány

<?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>  
 <h2>Catalog</h2>  
 <table>  
 <xsl:for-each select="/catalog/book">  
 <tr>  
 <td><xsl:value-of select="title"/></td>  
 <td><xsl:if test="available">available</xsl:if></td>  
 </tr>  
 </xsl:for-each>  
 </table>  
 </body>  
 </html>  
</xsl:template>  
</xsl:stylesheet>

## Sablonok és elemek

* <xsl:template match="/"> - XPath kifejezéssel választjuk ki a környezetet
* <xsl:value-of select="title"/> - XPath kifejezéssel nyerjük ki a konkrét értéket
* <xsl:apply-templates select="title"/> - XPath kifejezéssel megadott csomópont(ok)ra alkalmazza a sablont

## Vezérlő elemek

* <xsl:for-each> elem – egy specifikus csomópontkészlet összes elemét válaszhatjuk ki
* <xsl:if> elem - feltételes kiválasztás
* Többszörös kiválasztáshoz az <xsl:choose>, <xsl:when> és <xsl:otherwise> elemeket használhatjuk

## Csomópontok rendezése

* <xsl:sort> segítségével végezzük

## XSLT Javaban

* Eredeti neve TrAX (Transformation API for XML)
* Factory method tervezési minta
* Source, Result
  + DOMSource, SAXSource, StreamSource
  + DOMResult, SAXResult, StreamResult

## XSLT Javaban példakód

try {  
 Transformer transformer = TransformerFactory.newInstance().newTransformer(  
 new StreamSource(XsltApi.class.getResourceAsStream("/catalog.xslt")));  
 StringWriter result = new StringWriter();  
 transformer.transform(  
 new StreamSource(new StringReader(input)), new StreamResult(result)  
 );  
 return result.toString();  
} catch (TransformerException e) {  
 throw new RuntimeException("Error transforming xml", e);  
}

class: inverse, center, middle

.training360-logo[Training360]

# JAXB

## JAXB

* Java Architecture for XML Binding
* XML séma – Java reprezentáció
* XML -> Java (unmarshalling)
* Java -> XML (marshalling)
* Java <-> XML séma
* Annotációk
* https://docs.oracle.com/javase/tutorial/jaxb/intro/index.html

## Implementációk

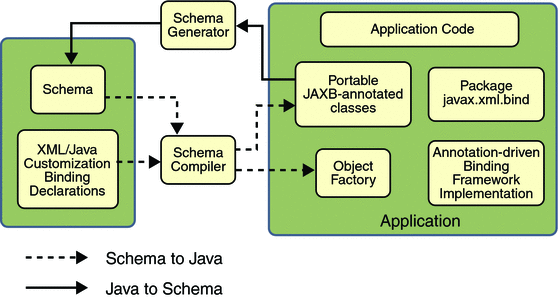
* JAXB Reference Implementation
  + https://github.com/javaee/jaxb-v2
* EclipseLink MOXy
  + http://www.eclipse.org/eclipselink/#moxy

## JAXB verziók

* Java 8 - JAXB 2.2 - JSR 222

$ java -version  
java version "1.8.0\_101"  
Java(TM) SE Runtime Environment (build 1.8.0\_101-b13)  
Java HotSpot(TM) 64-Bit Server VM (build 25.101-b13, mixed mode)  
$ schemagen -version  
schemagen 2.2.8-b130911.1802  
$ schemagen -fullversion  
schemagen full version "2.2.8-b130911.1802"  
$ xjc -version  
xjc 2.2.8-b130911.1802  
$ xjc -fullversion  
xjc full version "2.2.8-b130911.1802"

## JAXB architektúra



JAXB architektúra

## JAXB binding process



JAXB binding

## JAXB marshalling

JAXBContext ctx = JAXBContext.newInstance(Catalog.class, Book.class);  
Marshaller marshaller = ctx.createMarshaller();  
marshaller.setProperty(Marshaller.JAXB\_FORMATTED\_OUTPUT,  
 Boolean.TRUE);  
marshaller.setProperty(Marshaller.JAXB\_FRAGMENT,  
 Boolean.FALSE);  
  
  
StringWriter writer = new StringWriter();  
marshaller.marshal(catalog, writer);  
return writer.toString();

## JAXB működése, unmarshalling

* javax.xml.transform.Source: DOMSource, JAXBSource, SAXSource, StAXSource, StreamSource

JAXBContext ctx = JAXBContext.newInstance(Catalog.class, Book.class);  
Unmarshaller unmarshaller = ctx.createUnmarshaller();  
return (Catalog) unmarshaller.unmarshal(source);

## További osztályok

* JAXBElement: név/érték pár, ha nincs hozzá megfelelő Java reprezentáció
* ObjectFactory
  + POJO-k példányosítására alternatíva
  + Megfelelő JAXBElement példányok előállítására
* JAXBException

## JAXB binding

* Default binding
* Inline annotations
* Internal binding language (séma fájlon belül)
* External binding customization file (külső állomány, xjc programnak -b kapcsolóval átadandó)

## Annotációk

* @XmlElement
* @XmlElementWrapper
* @XmlAttribute
* @XmlEnum, @XmlEnumValue
* @XmlRootElement
* @XmlAccessorType: NONE, FIELD, PROPERTY, PUBLIC\_MEMBER
* @XmlAccessorOrder: ALPHABETICAL, UNDEFINED
* @XmlType propOrder
* @XmlTransient

## További annotációk

* @XmlSchema, package-info.java állományban adandó meg
* @XmlSchemaType, ha egy Java típushoz több séma típus tartozik, ezzel lehet választani, pl. XMLGregorianCalendar xs:date-hez

## Validation

Unmarshaller u = …  
Schema schema = SchemaFactory  
 .newInstance(XMLConstants.W3C\_XML\_SCHEMA\_NS\_URI)  
 .newSchema(new StreamSource(MyClass.class.getResourceAsStream("/schema.xsd")));  
u.setSchema(schema);

## Adapter

* Multivariate type mappings
* @XmlJavaTypeAdapter
* extends XmlAdapter<ValueType,BoundType>
* DatatypeConverter

## Körkörös referencia

* A cycle is detected in the object graph. This will cause infinitely deep XML
* @Transient
* afterUnmarshal metódus
* CycleRecoverable interfész (2.3-as verziótól kezdve)

## Ős és leszármazott

* Leszármazott: @XmlRootElement
* Hivatkozó attribútum: @XmlElementRef

## ANY

<xs:complexType name="AnyType">  
 <xs:sequence>  
 <xs:any minOccurs="0" maxOccurs="unbounded"  
 namespace="##any" processContents="skip" />  
 </xs:sequence>  
</xs:complexType>

* @XmlAnyElement

https://stackoverflow.com/questions/13941747/serializing-with-jaxb-and-the-any

## Újrafelhasználható complex type és JAXBElement<T>

* JAXB 2.0 – JDK 5 – generikusok megjelenésével
* Különböző element ugyanazon complex type-pal
* JAXBElement<T>
* External binding, simple-minded binding mode: <xjc:simple />
* https://stackoverflow.com/questions/20396375/why-and-when-jaxbelement-is-required-in-jaxb
* https://javaee.github.io/jaxb-v2/doc/user-guide/release-documentation.html#simple

## Catalog

* Séma hivatkozik másik sémára annak helyének megadása nélkül

<xs:import namespace="http://www.w3.org/1999/xlink" />

* Feloldása catalog fájllal, CatalogResolver

http://xerces.apache.org/xml-commons/components/resolver/resolver-article.html

## Sémából Java forrás

* Parancssorból

xjc -p package sema.xsd

## Java forrásból séma

* Parancssorból

schemagen package/\*.java

## Maven plugin

* maven-jaxb2-plugin (https://github.com/highsource/maven-jaxb2-plugin) - 2017. november 12., de nincs schemagen
* jaxb2-maven-plugin (http://www.mojohaus.org/jaxb2-maven-plugin/Documentation/v2.2/) - 2017. március 8.
* CXF XJC Maven Plugin (http://cxf.apache.org/cxf-xjc-plugin.html)