XML dokumentumok feldolgozása Java-ban

XML, DTD, XSD, XSLT, JAXP, DOM, SAX, JDOM

Simon Károly
simon.karoly@codespring.ro

XML

- EXtensible Markup Language (kiterjeszthető jelölőnyelv): W3C (1998).
- Főként adatreprezentációra alkalmas, platformfüggetlen, alkalmazástól független adatcserét tesz lehetővé.
- HTML ↔ XML: adatok megjelenítése ↔ adatok leírása.
- Az XML tag-ek nincsenek előre meghatározva, egyszerű szintaxis, szigorú szabályok.
- Az XML állományban tárolt adat szerkezete leírható DTD (Document Type Definition) vagy XML séma (XSD) segítségével.
- Új nyelvek definiálhatóak a segítségével (XHTML, WML stb.).

```
<?xml version="1.0" encoding="UTF-8"?>
<Personnel>
  <Employee type="full-time">
              <Name>Juliska</Name>
              <Id>1234</Id>
              <Age>23</Age>
  </Employee>
  <Employee type="part-time">
              <Name>Jancsika</Name>
              <Id>1235</Id>
              <Age>34</Age>
  </Employee>
</Personnel>
```

XML - szabályok

- Minden elemnek kell legyen záró tag-je.
- Számít a kis- vagy nagybetű.
- A tag-eket helyesen kell egymásba ágyazni (nem lehetnek egymásba ékelve).
- A dokumentumnak egy és csakis egy gyökér eleme lehet.
- Az attribútumok értékeit kötelező idézőjelbe (” vagy ’) tenni.
- A fehér karakterek figyelembe lesznek véve.
- Újsor: LF, Megjegyzés: <!– – XML comment – – >.
- Elemek közti viszonyok: szülő, gyerek, testvér.
- Elem felépítése: kezdő tag, törzs, záró tag (lehet üres is <tagnev ... /tagnev>). Az elemnek lehetnek attribútumai.
- A különböző állományokból származó azonos nevű elemek esetében a névkonfliktusok feloldására XML névtereket alkalmazunk.

```
<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>
```

```
<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

XML - névterek

- Egy elem kezdő tag-jébe (vagy a dokumentum gyökerébe) helyezett xmlns attribútum.
- Szintaxisa: xmlns:namespace-prefix=”namespaceURI”.
- Az illető elembe ágyazott összes elem, melynek ugyanaz a prefix-e, ugyanahhoz a névtérhez fog tartozni.
- Az URI egy egyedi nevet rendel a névterülethez.
- Alapértelmezett névtér (prefix nélkül): xmlns=”namespaceURI”.

```
<root>
 <h:table xmlns:h="http://www.w3.org/TR/html4/">
                          <h:tr>
                          <h:td>Apples</h:td>
                          <h:td>Bananas</h:td>
                          </h:tr>
 </h:table>

 <f:table xmlns:f="http://www.w3schools.com/furniture">
                          <f:name>African Coffee Table</f:name>
                          <f:width>80</f:width>
                          <f:length>120</f:length>
 </f:table>
</root>
```

Jól formált és érvényes XML dokumentumok

- Egy XML dokumentum jól formált (well formed): ha megfelel az XML szintaktikai szabályainak.
- Egy XML dokumentum érvényes (valid): ha jól formált, és megfelel a dokumentum séma definíciójának.
- Séma definíció: egy bizonyos DTD-ben vagy XML sémában (XSD) megadott szabályok.
- DTD (Document Type Definition): elterjedt séma-leíró módszer, amely megadja az XML dokumentum érvényes építőelemeit (elemek, attribútumok), illetve felépítését.
  - Szabványos, de nem XML alapú.
  - Megadható az XML állományon belül (ritkábban használt) (<DOCTYPE gyoker-elem [elem-deklaraciok]>), vagy külön dtd kiterjesztésű állományban (pl. a web.xml szerkezetét leíró DTD).
- XSD (XML Schema Definition): a DTD-nek XML alapú alternatívája, meghatározza, hogy milyen elemek és attribútumok szerepelhetnek egy dokumentumban, milyen beágyazott (gyerek) elemek vannak, és meghatározza ezek számát, illetve előfordulásának sorrendjét.
  - Az elemek illetve attribútumok típusa is definiálható, megadhatóak alapértelmezett, illetve rögzített értékek.

DTD

- Példa:

```
<!DOCTYPE TVSCHEDULE [
<!ELEMENT TVSCHEDULE (CHANNEL+)>
<!ELEMENT CHANNEL (BANNER, DAY+)>
<!ELEMENT BANNER (#PCDATA)>
<!ELEMENT DAY ((DATE, HOLIDAY) | (DATE, PROGRAMSLOT+))+>
<!ELEMENT HOLIDAY (#PCDATA)>
<!ELEMENT DATE (#PCDATA)>
<!ELEMENT PROGRAMSLOT (TIME, TITLE, DESCRIPTION?)>
<!ELEMENT TIME (#PCDATA)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT DESCRIPTION (#PCDATA)>
<!ATTLIST TVSCHEDULE NAME CDATA #REQUIRED>
<!ATTLIST CHANNEL CHAN CDATA #REQUIRED>
<!ATTLIST PROGRAMSLOT VTR CDATA #IMPLIED>
<!ATTLIST TITLE RATING CDATA #IMPLIED>
<!ATTLIST TITLE LANGUAGE CDATA #IMPLIED>
]>
```

- Elemek (!ELEMENT + elem név, gyerekek).

  Gyerekek: +→egy vagy több, ANY (bármilyen), EMPTY (üres), vagy (|), PCDATA (parsed character data), CDATA (character data).

- Attribútumok (!ATTLIST + elem név, attribútum név, típus, alapért. érték).

  Típus: CDATA, felsorolás: (en1| en2 | . . .), ID, IDREF, IDREFS, NMTOKEN, NMTOKENS, ENTITY, ENTITIES, NOTATION, xml: stb. Alapértelmezett érték: érték, REQUIRED, FIXED érték, IMPLIED.

XSD

- Egyszerű elem: <xs:element name=”xx” type=”yy”/>
  Beépített típusok: xs:string, xs:decimal, xs:integer, xs:boolean, xs:date, xs:time. Egyszerű elemnek lehet alapértelmezett (default=”. . . ”), vagy rögzített (fixed=”. . . ”) értéke.
- Attribútum: <xs:attribute name=”xx” type=”yy”/>.
  Lehet alapértelmezett, vagy rögzített értéke. Kötelező: use="required"
- Összetett elem: más beágyazott elemeket és attribútumokat tartalmazó elem.
  - Indikátorok: sorrendet meghatározó (All, Choice, Sequence), előfordulást meghatározó (maxOccurs, minOccurs), csoport (Group name, attributeGroup name).
  - Sequence elem: meghatározza, hogy a beágyazott elemek egy adott sorrendben kell legyenek.
  - Any, illetve anyAttribute elemekkel kibővíthetővé tehetjük a dokumentumot.
- Megszorítások (facets): megadhatjuk az elemek és attribútumok elfogadható értékeit.
  - <xs:restriction base=”xs:integer”> <xs:minInclusive value=”1990”/> <xs:maxInclusive value=”2010”/> </xs:restriction>
  - Felsorolás (xs:enumeration value="ertek"), reguláris kifejezés (xs:pattern value="[a-z]"), hossz (xs:length value="…")(+minLength, maxLength), fehér karakterek (xs:whiteSpace value="preserve") (replace, collapse) stb.
- Név hozzárendelése egyszerű típusokhoz, attribútumokhoz, hozzáférés más elemekből.
- Nevek hozzárendelése típusokhoz.

XSD - példa

- Példa:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

<!-- definition of simple elements -->
<xs:element name="orderperson" type="xs:string"/>
<xs:element name="name" type="xs:string"/>
<xs:element name="address" type="xs:string"/>
<xs:element name="city" type="xs:string"/>
<xs:element name="country" type="xs:string"/>
<xs:element name="title" type="xs:string"/>
<xs:element name="note" type="xs:string"/>
<xs:element name="quantity" type="xs:positiveInteger"/>
<xs:element name="price" type="xs:decimal"/>

<!-- definition of attributes -->
<xs:attribute name="orderid" type="xs:string"/>

<!-- definition of complex elements -->
<xs:element name="shipto">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="name"/>
      <xs:element ref="address"/>
      <xs:element ref="city"/>
      <xs:element ref="country"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

XSD - példa

```
<xs:element name="item">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="title"/>
      <xs:element ref="note" minOccurs="0"/>
      <xs:element ref="quantity"/>
      <xs:element ref="price"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="orderperson"/>
      <xs:element ref="shipto"/>
      <xs:element ref="item" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute ref="orderid" use="required"/>
  </xs:complexType>
</xs:element>

</xs:schema>
```

- Néha egyszerűbb, de rosszabb megoldás: elem definíciója az összetett elemen belül (ref alkalmazásának mellőzése).
- További lehetőség: nevek hozzárendelése típusokhoz:

```
<xs:simpleType name="stringtype">   <xs:restriction base="xs:string"/>  </xs:simpleType>
…
<xs:complexType name="shiptotype">
  <xs:sequence>  <xs:element name="name" type="stringtype"/> … </xs:sequence>
…
</xs:complexType>
```

XML – DTD/XSD megfeleltetés

- Közvetlen megfeleltetés: <!DOCTYPE root-element SYSTEM "dtdfile.dtd" >, ahol dtdfile.dtd a dtd állomány, vagy a schemaLocation attribútum használata: <xsi:schemaLocation = "http://www.ibm.com/schema.xsd">, ahol schema.xsd az XML séma neve.
- XML naplóbejegyzés (catalog entry): a DTD és XSD állományok regisztrációja az XML Catalog-ban, azonosítók (kulcsok) hozzárendelése.
- Azonosítók: public identifier (több rendszeren belül érvényes), system identifier (csak az adott rendszeren belül érvényes).
- Példák:
  - `<!DOCTYPE root-name PUBLIC "InvoiceId" "C:\mydtds\Invoice.dtd">` - ellenőrzi, hogy talál-e InvoiceId kulccsal rendelkező publikus azonosítót a katalógusban, ha igen, akkor az annak megfelelő URI-t használja, ha nem, akkor a rendszer azonosítót (amely közvetlenül az állományra mutat).
  - `<!DOCTYPE root-name SYSTEM "MyDTD.dtd">` - a rendszer azonosító használata a katalógusban. A MyDTD.dtd azonosítóval rendelkező katalógusbejegyzésnek megfelelő URI lesz használva.
  - `<purchaseOrder xmlns="http://www.ibm.com"`

    `xsi:schemaLocation="http://www.ibm.com C:\myschemas\PurchaseOrder.xsd">` …

    a schemaLocation-on belül az első rész az URI, a második a fájl neve (direkt megfeleltetés)
  - `<purchaseOrder xmlns="http://www.ibm.com" xsi:schemaLocation = "http://www.ibm.com PO.xsd">` …

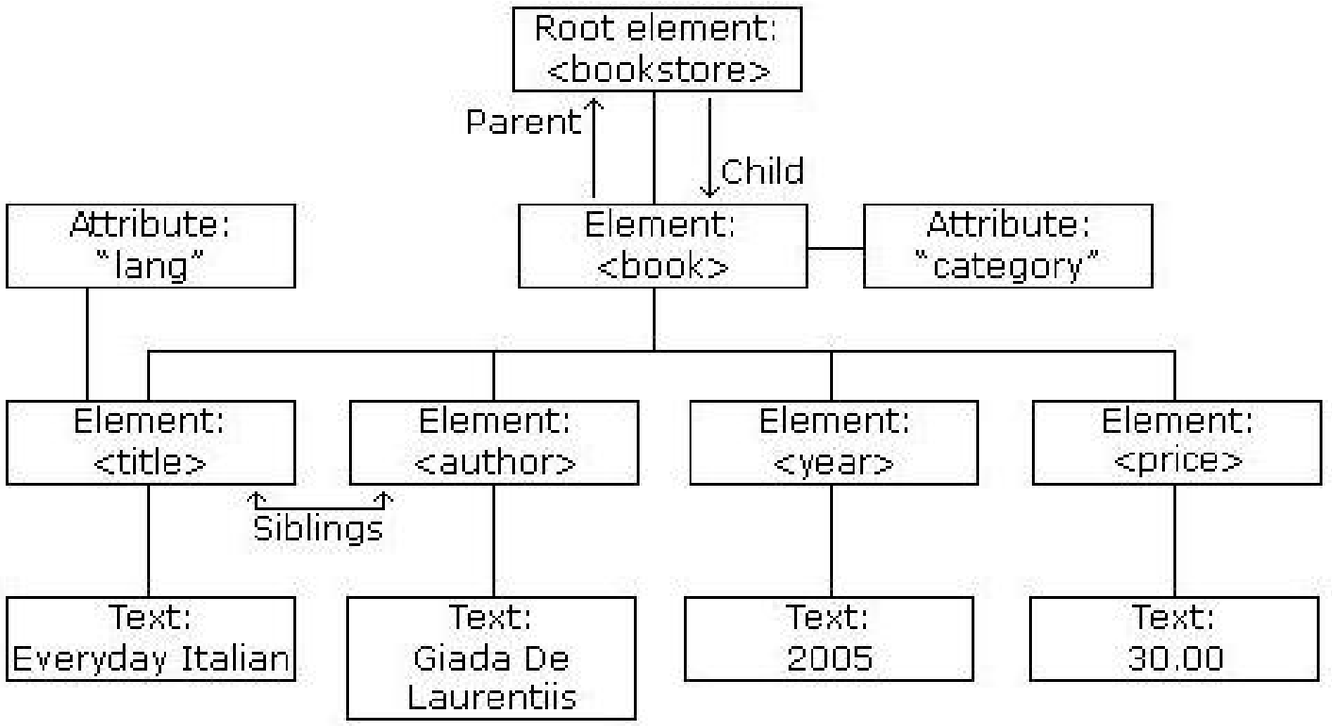
    a schemaLocation-on belül az első rész az URI, a második a kulcs az XML katalóguson belül

XSL

- XSL (EXtensible Stylesheet Language): XML alapú nyelvek XML dokumentumok bejárására, transzformálására, megjelenítésének meghatározására
- Három részből áll:
  - XPath: XML dokumentumok bejárására szolgáló, lekérdezésekre alkalmas nyelv. Kifejezéseket használ a bejárásra, és standard függvénykönyvtárakat biztosít. Több mint 100 beépített függvényt biztosít adatok (numerikus, string, dátum, idő stb.) feldolgozására.
  - XSLT: az XML dokumentumok transzformálására szolgáló nyelv (pl. XHTML dokumentumba történő transzformálás). Xpath-ot használ, a bejárás során azonosítja azokat a részeket, amelyek egy bizonyos sablonra illeszkednek, és ennek megfelelően végzi el az átalakítást.
  - XSL-FO: XML dokumentumok formázására szolgáló nyelv.

XML feldolgozás - DOM

- DOM (Document Object Model) – Platform- és nyelv-független standard XML dokumentumok feldolgozására. A dokumentumot hierarchikus formában ábrázolja, a csomópontok az elemek, attribútumok, illetve szövegrészek. Standard API-t biztosít az XML dokumentumok feldolgozására. W3C standard.



XML feldolgozás - SAX

- SAX (Simple API for XML): XML dokumentumok szekvenciális feldolgozására szolgáló API, a DOM egy igen elterjedt alternatívája.
- A DOM-tól eltérően nincs neki megfelelő formális specifikáció, a Java implementációt tekintik iránymutatónak.
- SAX feldolgozó (parser): egy SAX-et implementáló feldolgozó, adatfolyam feldolgozóként működik, eseményvezérelt API-val.
- Egy-egy esemény generálódik a következő elemek feldolgozása esetén: XML elem csomópontok, szöveget tartalmazó XML csomópontok, XML feldolgozó utasítások, XML megjegyzések.
- Az egyes eseményekre a felhasználó által definiált "callback"-metódusokat fogja meghívni a feldogozó.
- A feldolgozás egyirányú: a már feldolgozott adatot nem lehet újraolvasni (csak ha újrakezdjük a feldolgozást).
- Előnyök: kevesebb memóriát igényel, mint a DOM (ahol a teljes hierarchikus szerkezetet a memóriában kell tárolni), gyorsabb feldolgozást tesz lehetővé, nagyméretű dokumentumok esetében is használható
- Hátrány: nem tudjuk módosítani/menteni a forrás állományt, csak szekvenciálisan feldolgozni.

Java és XML feldolgozás

- JAXP (Java API for XML processing) - a javax.xml csomag. Ezen belül: javax.xml.datatypes (XML/Java típusmegfeleltetés), javax.xml.namespace (XML névterek), javax.xml.parsers (XML feldolgozás), javax.xml.transform (transzformációk), javax.xml.validation (XML validáció), javax.xml.xpath (Xpath kifejezések kiértékelésére szolgáló API).
- javax.xml.parsers:
  - DocumentBuilder: DOM reprezentáció, org.w3c.dom.Document objektum felépítése az XML állományból. Absztrakt osztály, példányra mutató referenciát a DocumentBuilderFactory getInstance gyártómetódusával kérhetünk.
  - SAXParser: SAX parser, absztrakt osztály, példányra mutató referenciát a SAXParserFactory.newSAXParser() metódussal kérhetünk. A feldolgozás közben egy adott Handler (pl. org.xml.sax.helper.DefaultHandler) megfelelő metódusai lesznek meghívva (callback). Ennek megfelelően a feldolgozáshoz a handler osztályból kell származtatnunk, újradefiniálva a megfelelő metódusokat. Az org.xml.sax csomagot használjuk.
- További lehetőségek: parser-ek (pl. Xerces) alkalmazása (pl. mentéshez), alternatív modellek, API-k, keretrendszerek (pl. Apache JDOM, XOM, dom4j) alkalmazása.

JDOM

- Apache JDOM: nyílt forráskódú Java keretrendszer optimalizált XML feldolgozásra.
- XML adatok reprezentációját szolgáló programozási modell, a DOM-hoz hasonló, de nem arra épül.
- Fejlesztését a SAX és DOM hiányosságai (pl. a SAX esetében nincs véletlen hozzáférés, mentési és módosítási lehetőség stb.) motiválták, de jó együttműködési lehetőséget is biztosít a SAX-el, illetve DOM-al.
- Csomagok: org.jdom (Document, Element, Attribute stb.), org.jdom.input (SAXBuilder, DOMBuilder, ResultSetBuilder), org.jdom.output (XMLOutputter, SAXOutputter, DOMOutputter, JTreeOutputter), org.jdom.adapters, org.jdom.transform (JDOMSource, JDOMResult).
- JAXP támogatás: bármilyen parser használható, de alapértelmezett a JAXP.
- Egyszerűbb és hatékonyabb manipuláció (pl. dokumentum felépítése, hozzáférés adatokhoz stb.), többféle, formázható kimenet, elemek lekérdezésének lehetősége egy "élő" listába (List típus, a módosítások a dokumentum objektumot is érintik), hatékony ellenőrzési lehetőségek (pl. jól formáltság), transzformációk stb.

Példák

```
<?xml version="1.0" encoding="UTF-8"?>
<Personnel>
        <Employee type="full-time">
                    <Name>Juliska</Name>
                    <Id>1234</Id>
                    <Age>23</Age>
        </Employee>
        <Employee type="part-time">
                    <Name>Jancsika</Name>
                    <Id>1235</Id>
                    <Age>34</Age>
        </Employee>
</Personnel>
```

- 1. Példa: feldolgozás – DOM alkalmazása (beolvasás, objektumok felépítése, kiírás a konzolra).
- 2. Példa: feldolgozás – SAX alkalmazása (beolvasás, objektumok felépítése, kiírás a konzolra).
- 3. Példa: XML állomány létrehozása, könyvek (Book példányok) adatainak tárolására.
- 4. Példa: az előző példák összevonása egy JDOM példába (beolvasás, objektumok felépítése, kiírás a konzolra, módosítás, mentés).

1. Példa

```
package examples;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

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

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

import core.Employee;

public class DomParserExample {

      private List<Employee> myEmployees;
      private Document dom;


      public DomParserExample () {
          myEmployees = new ArrayList<Employee> ();
      }
```

1. Példa

```
public void runExample () {
    parseXmlFile ();
    parseDocument ();
    printData ();
}

private void parseXmlFile () {
    //factory instance
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance ();
    try {
      //document builder instance
      DocumentBuilder db = dbf.newDocumentBuilder ();
      //dom representation
      dom = db.parse ("res/employees.xml");
    }catch (ParserConfigurationException pce) {
      pce.printStackTrace ();
    }catch (SAXException se) {
      se.printStackTrace ();
    }catch (IOException ioe) {
      ioe.printStackTrace ();
    }
}
```

1. Példa

```
private void parseDocument () {
      //the root element
      Element docEle = dom.getDocumentElement ();
      //employee node list
      NodeList nl = docEle.getElementsByTagName ("Employee");
      if (nl != null && nl.getLength () > 0) {
         for(int i = 0; i < nl.getLength (); i++) {
                  Element el = (Element) nl.item (i);
                  Employee e = getEmployee (el);
                  myEmployees.add (e);
            }
      }
}

/**
 *  Creating an Employee instance, using an employee element
 */
private Employee getEmployee (Element empEl) {
      String name = getTextValue (empEl, "Name");
      int id = getIntValue (empEl, "Id");
      int age = getIntValue (empEl, "Age");
      String type = empEl.getAttribute ("type");
      Employee e = new Employee (name, id, age, type);
      return e;
}
```

1. Példa

```
/**
 * Gets a String information from a specified text element
 * ex. <employee><name>Jancsika</name></employee>
 * If Element is a reference to this employee node,
 * the value of the tagName parameter is name,
 * the returned value will be Jancsika.
 *
 */
private String getTextValue (Element ele, String tagName) {
      String textVal = null;
      NodeList nl = ele.getElementsByTagName (tagName);
      if (nl != null && nl.getLength () > 0) {
         Element el = (Element) nl.item (0);
         textVal = el.getFirstChild ().getNodeValue ();
      }
      return textVal;
}


/**
 * Calls getTextValue and converts the result
 */
private int getIntValue (Element ele, String tagName) {
    return Integer.parseInt (getTextValue (ele, tagName));
}
```

1. Példa

```
private void printData (){
    System.out.println ("No of Employees '" + myEmployees.size() + "'.");
    Iterator<Employee> it = myEmployees.iterator ();
    while (it.hasNext ()) {
        System.out.println (it.next ());
    }
}


public static void main (String[] args){
    DomParserExample dpe = new DomParserExample ();
    dpe.runExample ();
}

}
```

2. Példa

```
package examples;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

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

import core.Employee;

public class SAXParserExample extends DefaultHandler {

      private List<Employee> myEmpls;
      private String tempVal;
      private Employee tempEmp;


      public SAXParserExample () {
          myEmpls = new ArrayList<Employee> ();
      }
```

2. Példa

```
public void runExample () {
    parseDocument ();
    printData ();
}

private void parseDocument () {
      //factory instance
      SAXParserFactory spf = SAXParserFactory.newInstance ();
      try {
         //SAX parser instance
         SAXParser sp = spf.newSAXParser ();
         //parsing the file, and registration for the callback methods
         sp.parse ("res/employees.xml", this);
      }catch (SAXException se) {
         se.printStackTrace ();
      }catch (ParserConfigurationException pce) {
         pce.printStackTrace ();
      }catch (IOException ie) {
         ie.printStackTrace ();
      }
}

private void printData (){
    System.out.println ("No of Employees '" + myEmpls.size() + "'.");
    Iterator<Employee> it = myEmpls.iterator ();
    while(it.hasNext ()) {
      System.out.println(it.next ());
    }
}
```

2. Példa

```
//Event Handlers
public void startElement (String uri, String localName, String qName, Attributes attributes)
                                                                    throws SAXException {
      tempVal = "";
      if (qName.equalsIgnoreCase ("Employee")) {
                  tempEmp = new Employee ();
                  tempEmp.setType (attributes.getValue ("type"));
      }
}
public void characters (char[] ch, int start, int length) throws SAXException {
      tempVal = new String (ch, start, length);
      }
      public void endElement (String uri, String localName, String qName) throws SAXException {
            if (qName.equalsIgnoreCase ("Employee")) {
               myEmpls.add(tempEmp);
            } else if (qName.equalsIgnoreCase ("Name")) {
               tempEmp.setName (tempVal);
            } else if (qName.equalsIgnoreCase ("Id")) {
               tempEmp.setId (Integer.parseInt (tempVal));
            } else if (qName.equalsIgnoreCase ("Age")) {
               tempEmp.setAge (Integer.parseInt (tempVal));
            }
      }
      public static void main (String[] args) {
          SAXParserExample spe = new SAXParserExample ();
          spe.runExample ();
      }
}
```

3. Példa

```
package examples;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

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

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Text;

import org.w3c.dom.bootstrap.DOMImplementationRegistry;
import org.w3c.dom.ls.DOMImplementationLS;
import org.w3c.dom.ls.LSSerializer;
import org.w3c.dom.ls.LSOutput;

import core.Book;

public class XMLCreatorExample {
```

3. Példa

```
private List<Book> myData;
private Document dom;

public XMLCreatorExample () {
    myData = new ArrayList<Book> ();
    loadData ();
    createDocument ();
}

public void runExample (){
    System.out.println ("Started .. ");
    createDOMTree ();
    printToFile ();
    System.out.println ("File generated successfully.");
}

/**
 * Generating a list of books
 */
private void loadData (){
    myData.add (new Book ("Java konyv", "Simon Károly", "Kenyerünk Java"));
    myData.add (new Book ("Mese", "Fiktiv Iro", "Blabla"));
}
```

3. Példa

```
//creating the DOM Document object
private void createDocument () {
    //creating a factory instance
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance ();
    try {
      //creating a builder instance
      DocumentBuilder db = dbf.newDocumentBuilder ();
      //creating a DOM object
      dom = db.newDocument ();
    } catch (ParserConfigurationException pce) {
      System.out.println ("Error while trying to instantiate DocumentBuilder " + pce);
      System.exit (1);
    }
}

//creating the XML structure
private void createDOMTree () {
    //creating the root element
    Element rootEle = dom.createElement ("Books");
    dom.appendChild (rootEle);
    Iterator<Book> it  = myData.iterator ();
    while (it.hasNext ()) {
      Book b = (Book) it.next ();
      //for each Book creates a new Book element and adds it to the root
      Element bookEle = createBookElement (b);
      rootEle.appendChild (bookEle);
    }
}
```

3. Példa

```
/**
 * Creates a Book XML element
 */
private Element createBookElement (Book b) {
      Element bookEle = dom.createElement ("Book");
      bookEle.setAttribute ("Subject", b.getSubject ());
      Element authEle = dom.createElement ("Author");
      Text authText = dom.createTextNode (b.getAuthor ());
      authEle.appendChild (authText);
      bookEle.appendChild (authEle);
      Element titleEle = dom.createElement ("Title");
      Text titleText = dom.createTextNode (b.getTitle ());
      titleEle.appendChild (titleText);
      bookEle.appendChild (titleEle);
      return bookEle;
}
```

3. Példa

```
//creating the XML file
private void printToFile () {
      try {
            //factory for creating DOMImplementation instance
            DOMImplementationRegistry registry = DOMImplementationRegistry.newInstance ();
            //factory for creating load/save objects
            DOMImplementationLS impl = (DOMImplementationLS) registry.getDOMImplementation ("LS");
            //creating serializer and output
            LSSerializer writer = impl.createLSSerializer ();
            LSOutput output = impl.createLSOutput ();
            //writing the file
            output.setByteStream (new FileOutputStream (new File ("book.xml")));
                  writer.write(dom, output);
            } catch (IOException ie) {
               ie.printStackTrace ();
            } catch (IllegalAccessException iae) {
               iae.printStackTrace ();
            } catch (InstantiationException ine) {
               ine.printStackTrace ();
            } catch (ClassNotFoundException cnfe) {
               cnfe.printStackTrace ();
            }
      }
      public static void main (String[] args) {
          XMLCreatorExample xce = new XMLCreatorExample ();
          xce.runExample ();
          }
}
```

4. Példa

```
package examples;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import java.util.ListIterator;
import java.util.Random;

import org.jdom.Document;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;
import org.jdom.output.XMLOutputter;
import org.jdom.Element;

import core.Employee;

public class JDomExample {

      private List<Employee> myEmployees;
      private Document dom;
      private List<Element> ndList;
      private Random rnd;
```

4. Példa

```
public JDomExample (){
    myEmployees = new ArrayList<Employee> ();
    rnd = new Random();
}

public void runExample () {
    parseXmlFile ();
    parseDocument ();
    printData ();
    modifyDocument ();
    parseDocument ();
    printData ();
    saveDocument ();
}

private void parseXmlFile () {
    SAXBuilder db = new SAXBuilder ();
    try {
      dom = db.build ("res/employees.xml");
    } catch (JDOMException e) {
      e.printStackTrace();
    } catch (IOException e) {
      e.printStackTrace ();
    }
}
```

4. Példa

```
@SuppressWarnings("unchecked")
private void parseDocument () {
      Element root = dom.getRootElement ();
      //getting the list of children:
      //(the list is "alive"! modifications will affect the Document object)
      ndList = root.getChildren ();
      //use iterator instead for (ndList.size ())
      ListIterator<Element> iterator = ndList.listIterator ();
      while (iterator.hasNext ()) {
         buildEmployeeInstance(iterator.next ());
      }
}

private void buildEmployeeInstance (Element e) {
      String name = e.getChildText ("Name");
      int id = Integer.parseInt (e.getChildText ("Id"));
      int age = Integer.parseInt (e.getChildText ("Age"));
      String type = e.getAttributeValue ("type");
      Employee emp = new Employee (name, id, age, type);
      myEmployees.add (emp);
}

private void printData () {
      System.out.println ("No of Employees '" + myEmployees.size () + "'.");
      for (Employee e:myEmployees)
         System.out.println (e);
}
```

4. Példa

```
private void modifyDocument () {
      ListIterator<Element> iterator = ndList.listIterator ();
      while (iterator.hasNext ())
         iterator.next ().getChild ("Age").setText (Integer.toString (rnd.nextInt (65)));
}

private void saveDocument () {
      XMLOutputter outp = new XMLOutputter ();
            try {
               outp.output (dom, new FileOutputStream (new File ("res/employees.xml")));
            } catch (FileNotFoundException e) {
               e.printStackTrace ();
            } catch (IOException e) {
               e.printStackTrace();
            }
      }

      public static void main (String[] args) {
            JDomExample jde = new JDomExample ();
            jde.runExample ();
      }

}
```

BiblioSpring – XML import/export

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns="http://www.bibliospring.edu.codespring.ro/Book"
              xmlns:xs="http://www.w3.org/2001/XMLSchema"
              targetNamespace="http://www.bibliospring.edu.codespring.ro/Book">

      <!-- Definition of simple elements -->
      <xs:element name="title" type="xs:string"></xs:element>
      <xs:element name="isbn" type="xs:string"></xs:element>
      <xs:element name="publishingDate" type="xs:string"></xs:element>
      <xs:element name="editorName" type="xs:string"></xs:element>
      <xs:element name="editorInfo" type="xs:string"></xs:element>
      <xs:element name="authorFirstName" type="xs:string"></xs:element>
      <xs:element name="authorLastName" type="xs:string"></xs:element>

      <!--  Definition of attributes -->

      <!--  Definition of complex types -->
      <xs:complexType name="EditorType">
            <xs:sequence>
                  <xs:element ref="editorName" minOccurs="1" maxOccurs="1"></xs:element>
                  <xs:element ref="editorInfo" minOccurs="0" maxOccurs="1"></xs:element>
            </xs:sequence>
      </xs:complexType>

      <xs:complexType name="AuthorType">
            <xs:sequence>
                  <xs:element ref="authorFirstName" minOccurs="1" maxOccurs="1"></xs:element>
                  <xs:element ref="authorLastName" minOccurs="0" maxOccurs="1"></xs:element>
            </xs:sequence>
      </xs:complexType>
```

BiblioSpring – XML import/export

```
<xs:complexType name="AuthorListType">
	<xs:sequence>
		<xs:element ref="author" maxOccurs="unbounded"></xs:element>
	</xs:sequence>
</xs:complexType>

<xs:complexType name="BookType">
	<xs:sequence>
		<xs:element ref="title"></xs:element>
		<xs:element ref="authors"></xs:element>
		<xs:element ref="isbn"></xs:element>
		<xs:element ref="editor"></xs:element>
		<xs:element ref="publishingDate"></xs:element>
	</xs:sequence>
</xs:complexType>

<xs:complexType name="BookListType">
	<xs:sequence>
		<xs:element ref="book" maxOccurs="unbounded"></xs:element>
	</xs:sequence>
</xs:complexType>

<!--  Definition of complex elements -->
<xs:element name="editor" type="EditorType"></xs:element>
<xs:element name="author" type="AuthorType"></xs:element>
<xs:element name="authors"  type="AuthorListType"></xs:element>
<xs:element name="book"  type="BookType"></xs:element>
<xs:element name="books"  type="BookListType"></xs:element>


</xs:schema>
```

BiblioSpring – XML import/export

```
package edu.codespring.bibliospring.swingclient.controll;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.Namespace;
import org.jdom2.output.Format;
import org.jdom2.output.XMLOutputter;

import edu.codespring.bibliospring.backend.model.Author;
import edu.codespring.bibliospring.backend.model.Book;

public class XMLExporter {

    private List<Book> books;
    private Document   dom;
    private Namespace  bsbNameSpace;

    public XMLExporter () {
        this (new ArrayList<Book> ());
    }

    public XMLExporter (final List<Book> books) {
        this.books = books;
        buildDOM ();
    }
```

BiblioSpring – XML import/export

```
public List<Book> getBooks () {
    return books;
}

public void setBooks (final List<Book> books) {
    this.books = books;
}

public void exportData (final File f) {
    final XMLOutputter writer = new XMLOutputter ();
    writer.setFormat (Format.getPrettyFormat ());
    try {
        writer.output (dom, new FileOutputStream (f));
    } catch (final FileNotFoundException ex) {
        ex.printStackTrace ();
    } catch (final IOException ex) {
        ex.printStackTrace ();
    }
}

private void buildDOM () {
    dom = new Document ();
    final Element rootElement = new Element ("books");
    bsbNameSpace = Namespace.getNamespace ("http://www.bibliospring.edu.codespring.ro/Book");
    rootElement.setNamespace (bsbNameSpace);
    dom.setRootElement (rootElement);
    for (final Book b : books) {
        rootElement.addContent (createBookElement (b));
    }
}
```

BiblioSpring – XML import/export

```
private Element createBookElement (final Book b) {
   final Element bookElement = new Element ("book", bsbNameSpace);
   final Element titleElement = new Element ("title", bsbNameSpace);
   titleElement.addContent (b.getTitle ());
   bookElement.addContent (titleElement);
   final Element authorsElement = new Element ("authors", bsbNameSpace);
   for (final Author a : b.getAuthors ()) {
       final Element authorElement = new Element ("author", bsbNameSpace);
       final Element firstNameElement = new Element ("authorFirstName", bsbNameSpace);
       firstNameElement.addContent (a.getFirstName ());
       final Element lastNameElement = new Element ("authorLastName", bsbNameSpace);
       lastNameElement.addContent (a.getLastName ());
       authorElement.addContent (firstNameElement);
       authorElement.addContent (lastNameElement);
       authorsElement.addContent (authorElement);
   }
   bookElement.addContent (authorsElement);
   final Element isbnElement = new Element ("isbn", bsbNameSpace);
   isbnElement.addContent (b.getIsbn ());
   bookElement.addContent (isbnElement);
   final Element editorElement = new Element ("editor", bsbNameSpace);
   final Element editorNameElement = new Element ("editorName", bsbNameSpace);
   final Element editorInfoElement = new Element ("editorInfo", bsbNameSpace);
   editorNameElement.addContent (b.getEditor ().getName ());
   editorInfoElement.addContent (b.getEditor ().getInfo ());
   editorElement.addContent (editorNameElement);
   editorElement.addContent (editorInfoElement);
   bookElement.addContent (editorElement);
   final Element dateElement = new Element ("publishingDate", bsbNameSpace);
   dateElement.addContent (b.getPublishingDate ());
   bookElement.addContent (dateElement);
   return bookElement;
}
```

BiblioSpring – XML import/export

```
package edu.codespring.bibliospring.swingclient.controll;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import org.jdom2.Document;
import org.jdom2.Element;
import org.jdom2.JDOMException;
import org.jdom2.Namespace;
import org.jdom2.input.SAXBuilder;
import edu.codespring.bibliospring.backend.model.Author;
import edu.codespring.bibliospring.backend.model.Book;
import edu.codespring.bibliospring.backend.model.Editor;
```

```
public class XMLImporter {

    private final File file;
    private Document   dom;
    private List<Book> books;

    public XMLImporter (final File file) {
        this.file = file;
    }

    private void buildDOM () {
        dom = new Document ();
        final SAXBuilder builder = new SAXBuilder ();
        try {
            dom = builder.build (file);
        } catch (JDOMException | IOException ex) {
            ex.printStackTrace ();
        }
    }
```

BiblioSpring – XML import/export

```
private void buildBookList () {
    books = new ArrayList<Book> ();
    final Element root = dom.getRootElement ();
    final Namespace ns = root.getNamespace ();
    final List<Element> liveList = root.getChildren ();
    for (final Element e : liveList) {
        final Book b = new Book ();
        b.setTitle (e.getChildText ("title", ns));
        final List<Element> authorList = e.getChild ("authors", ns).getChildren ();
        for (final Element a : authorList) {
            b.addAuthor (new Author (a.getChildText ("authorFirstName", ns), a.getChildText ("authorLastName", ns)));
        }
        b.setIsbn (e.getChildText ("isbn", ns));
        final Element editorElement = e.getChild ("editor", ns);
        b.setEditor (new Editor (editorElement.getChildText ("editorName", ns), editorElement.getChildText (
                "editorInfo", ns)));
        b.setPublishingDate (e.getChildText ("publishingDate", ns));
        books.add (b);
    }
}

public List<Book> getBookList () {
    buildDOM ();
    buildBookList ();
    return books;
}

}
```