MANAGING INFORMATION (CSCU9T4)
Lecture 8: XML and Java (DOM)

Gabriela Ochoa
http://www.cs.stir.ac.uk/~goc/
RESOURCES

Books
- **Beginning XML, 5th Edition** (2012) by Joe Fawcett, Danny Ayers, Liam R. E. Quin

Links and websites
- [W3 ORG](#)
- [XML Tutorial](#): W3Schools
- Oracle: [XML Parser for Java](#)
- The Java Tutorials: [Reading XML Data into a DOM](#)
CONTENTS

- XML Schema Restrictions
- General programming languages (Java) and XML
- SAX and DOM are corresponding APIs that are language independent and supported by numerous languages:
  - Simple API for XML (SAX)
  - Document Object Model (DOM)
- Related tools
  - XPath - a language for navigating in XML documents.
- Examples and Demos
XML Schema Restrictions/ Facets

- Restrictions are used to define acceptable values for XML elements or attributes. Restrictions on XML elements are called facets.

- **Example**: restriction on value. “age”, cannot be lower than 0 or greater than 120.

```xml
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
      <xs:minInclusive value="0"/>
      <xs:maxInclusive value="120"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```
To limit the content of an XML element to a set of acceptable values, we would use the enumeration constraint.

**Example**: element called "car" with a restriction. The only acceptable values are: Audi, Golf, BMW:

```xml
<xs:element name="car">
  <xs:simpleType>
    <xs:restriction base="xs:string">
      <xs:enumeration value="Audi"/>
      <xs:enumeration value="Golf"/>
      <xs:enumeration value="BMW"/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```
DOM (Document Object Model)
**WHAT IS XML DOM?**

- A standard for accessing and manipulating XML documents.
  - A standard object model for XML
  - A standard programming interface for XML
  - Platform- and language-independent
- Defines the objects and properties of all XML elements, and the methods (interface) to access them.
- Presents an XML document as a tree-structure.
  - Everything in an XML document is a node.
**Tree Terminology**

- **Root**: node without parent (A)
- **Siblings**: nodes share the same parent
- **Internal node**: node with at least one child (A, B, C, F)
- **External node** (leaf): node without children (E, I, J, K, G, H, D)
- **Ancestors** of a node: parent, grandparent, grand-grandparent, etc.
- **Descendant** of a node: child, grandchild, grand-grandchild, etc.
- **Depth** of a node: number of ancestors
- **Height** of a tree: maximum depth of any node (3)
- **Degree** of a node: the number of its children
- **Degree** of a tree: the maximum number of its node.
- **Subtree**: tree consisting of a node and its descendants

A Tree is an abstract model of a hierarchical structure.
SAX vs. DOM

- **SAX** is relatively straightforward and handles documents of any size
  - Sometimes the application needs an intermediate data structure to hold elements of the input document
  - Doing this can be complex with SAX

- Alternatively, the XML parser itself can build a data structure for the whole document (if not too large)
  - the structure can be navigated using the methods provided
  - this saves the programmer from having to store the data

- **A DOM** parser does just this
DOM (Tree-Based) vs. SAX (Event-Based) APIs

```xml
<?xml version="1.0"?>
<EMPLIST>
  <EMP>
    <ENAME>MARY</ENAME>
  </EMP>
  <EMP>
    <ENAME>SCOTT</ENAME>
  </EMP>
</EMPLIST>
```

The DOM interface creates a TREE structure based on the XML Document.

The SAX interface creates a series of linear events based on the XML document.

Useful for applications that include changes e.g. reordering, adding, or deleting elements.

Useful for applications such as search and retrieval that do not change the "XML tree."
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
  <book category="children">
    <title lang="en">Harry Potter</title>
    <author>J K. Rowling</author>
    <year>2005</year>
    <price>29.99</price>
  </book>
  <book category="web">
    <title lang="en">XQuery Kick Start</title>
    <author>James McGovern</author>
    <author>Per Bothner</author>
    <author>Kurt Cagle</author>
    <author>James Linn</author>
    <author>Vaidyanathan Nagarajan</author>
    <year>2003</year>
    <price>49.99</price>
  </book>
  <book category="web" cover="paperback">
    <title lang="en">Learning XML</title>
    <author>Erik T. Ray</author>
    <year>2003</year>
    <price>39.95</price>
  </book>
</bookstore>
EXAMPLE: BOOXS.XML
**Node Parents, Children, and Siblings**

- In a node tree, the top node is called the root.
- Every node, except the root, has exactly one parent node.
- A node can have any number of children.
- A leaf is a node with no children.
- Siblings are nodes with the same parent.
Another example

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- A simple example of xml -->
<staff>
  <staffMember>
    <name>Ann Turner</name>
    <phone>7423</phone>
  </staffMember>
  <staffMember>
    <name>Dave Smith</name>
    <phone>7440</phone>
  </staffMember>
</staff>
```
A DOM parser builds a tree

<!-- A simple ... -->

<staff>
  <staffMember>
    <name>
      Ann Turner
    </name>
    <phone>
      7423
    </phone>
  </staffMember>
  <staffMember>
    <name>
      Dave Smith
    </name>
    <phone>
      7440
    </phone>
  </staffMember>
</staff>
XML DOM Parser

- The XML DOM contains methods to traverse XML trees, access, insert, and delete nodes.

- However, before an XML document can be accessed and manipulated, it must be loaded into an XML DOM object.

- An XML parser reads XML, and converts it into an XML DOM object that can be accessed.

- Let us learn how to do this with Java
Applying the DOM approach with Java

- A DOM-based parser reads XML and builds a data structure, so think of it as a document builder.

- A DocumentBuilder is obtained from a ‘Factory’:

  ```java
  DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance();
  DocumentBuilder builder = builderFactory.newDocumentBuilder();
  ```

- The DocumentBuilder is then given a file to read:

  ```java
  Document doc = builder.parse("Staff.xml");
  ```
After parsing an XML file, `DocumentBuilderFactory` returns a `Document` object as the root/top of the tree.

Everything in the tree is a `Node` object: `Node` subclasses include `Document`, `Element` and `Text`.

The `Node` class has many methods to explore the tree:

- obtain references to a node’s `first child`, `last child`, `next sibling`, `parent`, or `all` node's children.
- get the node's type: `element`, `attribute`, `text`, `comment`, `entity reference`, ...
DOM AND XPath

- An XPath expression can be used to find a node at a particular place in the DOM tree:
  
  ```java
  XPathFactory xpathFactory = XPathFactory.newInstance();
  XPath path = xpathFactory.newXPath();
  String result = path.evaluate("/staff/staffMember[1]/name", document);
  ```

- It is not necessary to navigate an XML tree to find a certain element. For example, this might be used to find a password in a configuration file:

  ```java
  Element elem = (Element) path.evaluate("/config/password");
  String password = elem.toString();
  ```
**XPath**

- XPath is an expression language for addressing parts of an XML document
- XPath data model provides a tree representation of XML documents as well as atomic values such as integers, strings, and Booleans.
- It uses path expressions to select a set of nodes or atomic values in an XML document
- Path expressions look very much like the expressions you see when you work with a traditional computer file system
- XPath contains a library of standard functions
- XPath is a W3C recommendation
- Tutorial: [http://www.w3schools.com/xsl/xpath_intro.asp](http://www.w3schools.com/xsl/xpath_intro.asp)
XPath data model

- XPath sees an XML document as a tree structure.
- The topmost element of the tree is called the root element.
- Each information (XML elements, attributes, text, etc.) is called a node.
- Nodes that XPath can see
  - Root node
  - Elements and attributes
  - Special nodes like comments, processing instructions, namespace declarations.
**XPath Syntax**

- Let us consider the following XML document

```xml
<?xml version="1.0" encoding="UTF-8"?>

<bookstore>
  <book>
    <title lang="en">Harry Potter</title>
    <price>29.99</price>
  </book>

  <book>
    <title lang="en">Learning XML</title>
    <price>39.95</price>
  </book>

</bookstore>
```
**XPATH: SELECTING NODES**

- XPath uses path expressions to select nodes in an XML document.
- The most useful path expressions are listed below:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodename</td>
<td>Selects all nodes with the name &quot;nodename&quot;</td>
</tr>
<tr>
<td>/</td>
<td>Selects from the root node</td>
</tr>
<tr>
<td>//</td>
<td>Selects nodes in the document from the current node that match the selection no matter where they are</td>
</tr>
<tr>
<td>.</td>
<td>Selects the current node</td>
</tr>
<tr>
<td>..</td>
<td>Selects the parent of the current node,</td>
</tr>
<tr>
<td>@</td>
<td>Selects attributes</td>
</tr>
</tbody>
</table>
**XPath: Examples of Path Expressions**

<table>
<thead>
<tr>
<th>Path Expression</th>
<th>Selects</th>
</tr>
</thead>
<tbody>
<tr>
<td>bookstore</td>
<td>All nodes with the name &quot;bookstore&quot;</td>
</tr>
<tr>
<td>/bookstore</td>
<td>The root element bookstore</td>
</tr>
<tr>
<td>bookstore/book</td>
<td>All book elements that are children of bookstore</td>
</tr>
<tr>
<td>//@lang</td>
<td>All attributes that are named lang</td>
</tr>
<tr>
<td>//book</td>
<td>Selects all book elements no matter where they are in the document</td>
</tr>
<tr>
<td>bookstore//@lang</td>
<td>All book elements that are descendant of the bookstore element, no matter where they are under the bookstore element</td>
</tr>
</tbody>
</table>
## Predicates

- Are used to find a specific node or a node that contains a specific value.
- Are always embedded in square brackets.

<table>
<thead>
<tr>
<th>Path Expression</th>
<th>Selects</th>
</tr>
</thead>
<tbody>
<tr>
<td>/bookstore/book[last()]</td>
<td>The last book element that is the child of the bookstore element</td>
</tr>
<tr>
<td>/bookstore/book[position() &lt; 3]</td>
<td>The first two book elements that are children of the bookstore element</td>
</tr>
<tr>
<td>//title[@lang]</td>
<td>All the title elements that have an attribute named &quot;lang&quot;</td>
</tr>
<tr>
<td>//title[@lang='en']</td>
<td>All the title elements that have a &quot;lang&quot; attribute with a value of &quot;en&quot;</td>
</tr>
<tr>
<td>/bookstore/book[price &gt; 35.00]</td>
<td>All the book elements of the bookstore element that have a price element with a value greater than 35.00</td>
</tr>
</tbody>
</table>
import java.io.*;  // import input-output
import javax.xml.parsers.*;  // import parsers
import javax.xml.xpath.*;  // import XPath
import org.w3c.dom.*;  // import DOM;

public class DOMDemo {
    DocumentBuilder builder = null;
    Document document = null;
    XPath path = null;
}
public void loadDocument(String filename) {
    try {
        // create a document builder and an XPath interpreter
        DocumentBuilderFactory builderFactory = DocumentBuilderFactory.newInstance();
        builder = builderFactory.newDocumentBuilder();
        XPathFactory xpathFactory = XPathFactory.newInstance();
        path = xpathFactory.newXPath();

        // parse the document for later searching
        document = builder.parse(new File(filename));
    }
    catch (Exception exc) {
        System.err.println("could not load document " + exc);
    }
}
public String query(String query) {
    String result = "";
    try {
        result = path.evaluate(query, document);
    }
    catch (Exception exception) {
        System.err.println("could not perform query - " + exception);
    }
    return(result);
}
public static void main(String[] args) {
    loadDocument("Staff.xml");

    Node staff = document.getFirstChild();
    Node staffMember = staff.getFirstChild().getNextSibling();
    System.out.println("First child is: " + staff.getNodeName());
    System.out.println("Child is: " + staffMember.getNodeName());

    Node name = staffMember.getFirstChild().getNextSibling();
    System.out.println("Name is: " + name.getFirstChild().getNodeValue());
    System.out.println("Staff count is: " + query("count(/staff/staffMember)"));
    System.out.println("Next name is: " + query("/staff/staffMember[2]/name");
}
} // DOMDemo

Output:

First child is: staff
Child is: staffMember
Name is: Ann Turner
Staff count is: 2
Next name is: Dave Smith
Modifying a DOM Tree

The Node class also provides methods to modify the tree (e.g. to add or delete branches):
- a new tree can be created using these methods
- there is a DocumentBuilder method called newDocument

A program could:
- read a file of data that is not in XML
- create a DOM tree from it
- convert it into an XML file

The following example creates a tree using code rather than reading a file.
public void createDocument() {
    try {
        // create a document builder
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        DocumentBuilder builder = factory.newDocumentBuilder();

        // get a new document
        document = builder.newDocument();

        // create library as the root element and add it to the document
        Element library = document.createElement("library");
        document.appendChild(library);

        // add a cooking book
        Element book1 = document.createElement("book");
        Element title = document.createElement("title");
        title.appendChild(document.createTextNode("Italian Recipes"));
        book1.appendChild(title);
        library.appendChild(book1);
    }
}
// add a children's Book
Element book2 = document.createElement("book");
Element title2 = document.createElement("title");
title2.appendChild(document.createTextNode("Harry Potter"));
book2.appendChild(title2);
library.appendChild(book2);

} // createDocument
private static void printNodes() {

    Node lib = document.getFirstChild();
    System.out.println("Document first child node is: " + lib.getNodeName());
    Node book = lib.getFirstChild();
    System.out.println("First Child");
    System.out.println(" Child node is: " + book.getNodeName());
    System.out.println(" Next child node is: " + book.getFirstChild().getNodeName());
    System.out.println(" child text content is: " + book.getFirstChild().getTextContent());

    book = book.getNextSibling();
    System.out.println("Sibling (2nd Child)");
    System.out.println(" Child node is: " + book.getNodeName());
    System.out.println(" Next child node is: " + book.getFirstChild().getNodeName());
    System.out.println(" child text content is: " + book.getFirstChild().getTextContent());

    System.out.println("Book count is: " + query("count(/library/book)");

}
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;

Transformer tf = TransformerFactory.newInstance().newTransformer();
tf.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
tf.setOutputProperty(OutputKeys.INDENT, "yes");
Writer output = new StringWriter();
tf.transform(new DOMSource(document), new StreamResult(output));
System.out.println(output.toString());
SUMMARY OF DOM

- DOM loads a complete XML tree into memory
- XPath expressions allow parts of the XML tree to be found
- DOM trees can be created from scratch using DOM-based Java methods
- DOM is most suitable when the XML structure is not too large (say, tens of thousands of elements)