

XML-DOM Essentials

Web Languages Course 2009
University of Trento

Lab Objective

- Make basic operations on XML files by using the **J**ava **A**PI for **X**ML **P**rocessing



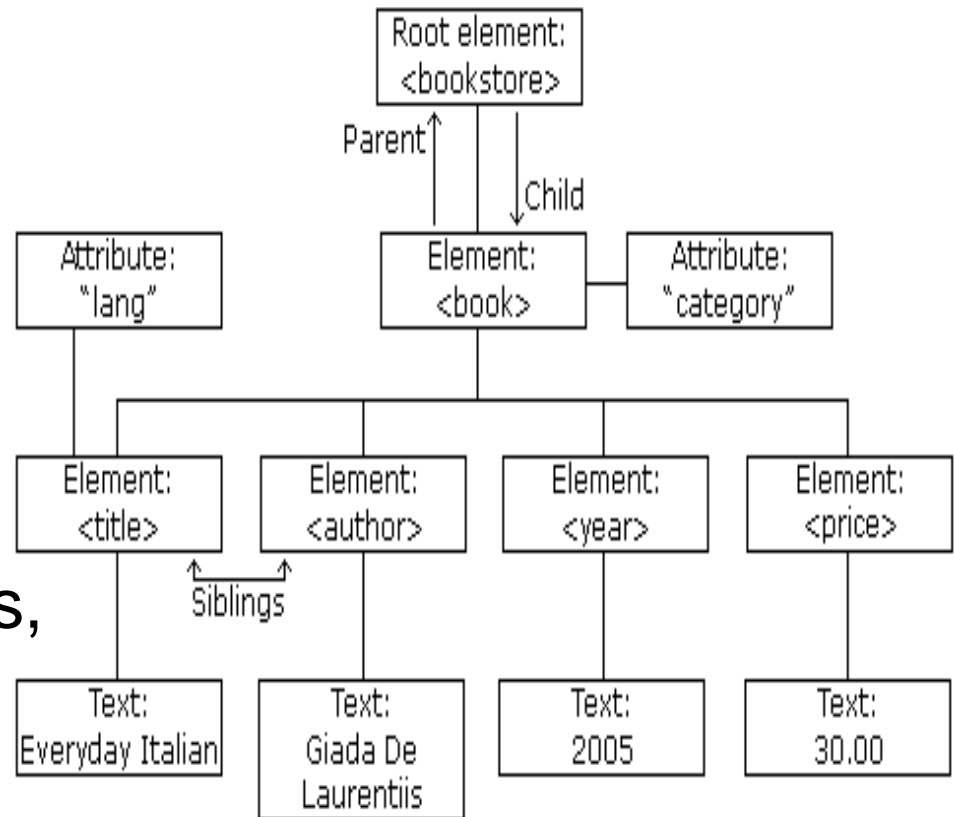
Lab Outline

- XML-DOM: intro
- Java XML API
 - DOM: parsing and manipulating
 - Class model
 - Parsing
 - Manipulating
 - XSL: transforming
 - The identity transformer
 - XPath: querying
 - Retrieving XML elements with XPath queries



XML- Document Object Model

- Defines a standard way for accessing and manipulating XML documents.
- Presents an XML document as a tree structure, with elements, attributes, comments, text as nodes



XML-DOM: The Class Model

- Each node is an instance of the Node interface
- Even the Document interface, representing the XML Document as a whole, is a Node:

```
interface Document implements Node
```

- Nodes possess the concept of “father” node and “children” nodes, as happens with any tree
- Node interface possess many sub-interfaces, specialized for the subtype of XML node they refer to



XML-DOM: The Class Model

- DOM main components are actually modeled as interfaces, and not as concrete classes
- As with any interface, this means you could not create an instance of it: instead, you have to ask the correspondent “Factory” class, which in turn creates the instance
- For example, you create a new Element class inside a Document class (which stands for creating a new node into the XML file) like this:

```
Element elem = myDocument.createElement("tagName");
```

- The goal is to decouple the actual implementation of the classes representing XML related concepts from their structure: this gives us the ability to choose a different implementation without changing existing code



XML-DOM: Parsing

1

- To parse an XML Document with DOM the class `DocumentBuilder` is used
- Since the class is abstract, the Factory class `DocumentBuilderFactory` is used

```
DocumentBuilderFactory dbf =  
DocumentBuilderFactory.newInstance();  
DocumentBuilder db = dbf.newDocumentBuilder();
```

- `DocumentBuilderFactory` as well as `DocumentBuilder` classes allow for fine tuning of the parsing process, in order to obtain the maximum flexibility when parsing an XML document into a DOM Document



XML-DOM: Parsing

2

- `DocumentBuilder` can parse from `File`, `InputStream`, or `URI` objects
- If something goes wrong, an `IOException` (e.g., unaccessible content) or a `SAXException` (e.g., not well-formed XML content) is thrown
- Finally, we call something like:

```
Document doc = db.parse(new File("myData.xml"));
```

- The `Document` instance created by parsing an XML file has no relation with the object on the disk, being a simple memory structure
 - Any transformation applied to the `Document` instance “in memory”, is not automatically reflected on the XML file “on the disk”



Exercise 1

- Understand the code
- Run the Java class `BuildXmlUsingDom` provided in the source code
- An xml file named *Employee.xml* should be created



XML-DOM: Manipulation

- A simple method for transforming every attribute of a certain Element object into a subElement of the same object

```
private static void removeAttributes(Document d, Element e) {
    NamedNodeMap attrList = e.getAttributes();
    while(attrList.getLength() > 0) {
        Attr attr = (Attr)attrList.item(0);
        // Adding the new Element
        Element newElement = d.createElement(attr.getName());
        newElement.setTextContent(attr.getValue());
        e.appendChild(newElement);
        // Removing the old Attribute
        e.removeAttribute(attr.getName());
    }
}
```



XSL: Transforming

1

- To save the result of a DOM XML tree back to a file (or string)

```
private static String transformXML(Document source)
    throws TransformerException, IOException {
```

```
    TransformerFactory tf =
        TransformerFactory.newInstance();
    Transformer ts = tf.newTransformer();
    StringWriter sw = new StringWriter();
    ts.transform(new DOMSource(source), new
        StreamResult(sw));
    return sw.toString();
}
```



XSL: Transforming

2

- The Transformer used here is the simplest of the Transformer object: it perform the so-called “identity-transformation”
- There's a lot of online documentation about this topic: I just add a little trick to get the output look nicer:

```
TransformerFactory tf = TransformerFactory.newInstance();  
// Number of spaces per tab  
tf.setAttribute("indent-number", new Integer(4));  
Transformer ts = tf.newTransformer();  
// Yes, we want indented output  
ts.setOutputProperty(OutputKeys.INDENT, "yes");
```



XPath: a (faster) way to query through XML documents trees 1

- Suppose you want to recover the textual content of the element *FirstName*, which is a child of *Student* element, which is a child of *Member* element inside the DOM document
- With DOM, you have to:
 - 1. Retrieve the *Member* element by calling
`doc.getElementsByTagName`
 - 2. Retrieve the *Student* element by calling
`member.getElementsByTagName`
 - 3. Retrieve the *FirstName* element by calling
`student.getElementsByTagName`
 - 4. Finally, retrieve the content by calling
`firstName.getTextContent`

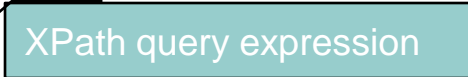
many
lines
of code!!



XPath: a (faster) way to query through XML documents trees 2

- XPath is a powerful language for addressing parts of an XML document: you could use XPath to extract information from XML documents just like you would use the SQL language to perform queries on RDBMS
- With XPath, you could obtain the same result in a more concise and elegant way:

```
XPathFactory xpf = XPathFactory.newInstance();  
xpath = xpf.newXPath();  
NodeList result =  
    (NodeList)xpath.evaluate("//Member/Student/FirstName", doc,  
    XPathConstants.NODESET);  
if(result.getLength() > 0) output =  
    result.item(0).getTextContent();
```



Exercise 2

- Implement a program that, given an xml file, read its content and create an array of Objects “Employee” and print it into the console
- **Suggestion #1:** make a Java bean called Employee according to the XML previously produced
- **Suggestion #2:** enrich the previous `BuildXmlUsingDom` class by implementing the methods appearing in the commented lines of the *main* method
 - `public Document readingXMLFromFile()`
 - `public ArrayList CreateListOfObjects(Document doc)`
- **Suggestion #3:** use DOM first, and XPath later



Exercise 3

- Write a function for calculating an XML tree maximum depth, that is, the distance of the farthest leaf node from the root
- Root distance is 0
- Distance of node “X” is the number of node met along the path from root to node “X”
- **Suggestion #1:** create a method called “maximumDepth” with the following signature:

```
private static int maximumDepth(Document doc)
```
- **Suggestion #2:** use recursion
- **Suggestion #3:** test the program with the xml file `rss.xml`



References

- JAXP
 - <https://jaxp.dev.java.net/>
- Xpath (from W3C site)
 - <http://www.w3.org/TR/xpath>
- XML DOM Tutorial (from W3C site)
 - <http://www.w3schools.com/dom/default.asp>
- Java DOM Tutorial
 - <http://www.roseindia.net/xml/dom/index.shtml>

