Java XML Parsing

Laboratory of Service Design and Engineering

2011/2012

Outline

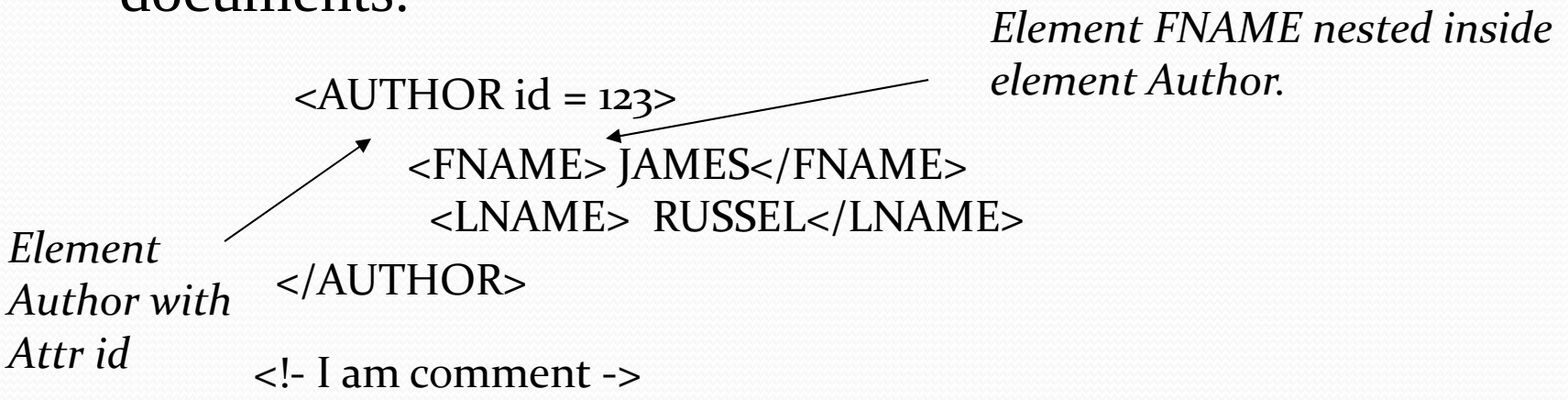
- XML Overview
- Simple API for XML (SAX)
- Document Object Model (DOM)
- Exercises

Introduction to XML

- XML stands for E**X**tensible **M**arkup **L**anguage
- XML was designed to describe data.
- XML tags are not predefined unlike HTML
- XML DTD and XML Schema define rules to describe data
- XML example of semi structured data

XML Building Blocks

- Elements (Tags) are the primary components of XML documents.



- Attributes provide additional information about Elements. Values of the Attributes are set inside the Elements
- Comments stats with <!- and end with ->

Well Formed Document

- An XML document MUST be well formed
- There is exactly one root element
- All elements that contain data must have start and end tags

```
<author>
.....
</author>
```

- Empty elements are allowed

```
<author> </author>
<author/>
```

Well Formed Document

- Elements must nest properly
  - If one element contains another element, the entire second element must be defined inside the start and end tags of the first element.
- All attribute values must be in quotes

<author id=”123”>

XML Parsing

- A software which parse the XML file in order to get contents.
- Event Driven Based
  - SAX (Simple API for XML)
  - Process the XML file when an event occurs.
- Tree Based
  - DOM (Document Object Model)
  - Process and reads the entire document into memory
  - CPU and memory intensive

SAX – Event Based

- It turns a document into a stream of events
  - e.g. starting and closing tags are events
- It handles one event at a time, “pulling” the next one from the stream as needed.
- The event handling does not depend on the events that came before (state independent processing)
- You see the data as it streams in: you cannot go back to an earlier position or jump ahead
- When an event comes in you can
  - Take some action in response to it
  - Build up a data structure

When to use SAX

- State independent processing
- To parse documents much larger than your available system memory
- Fast

SAX Java Packages

Packages	Description
org.xml.sax	Defines the **SAX interfaces**.
org.xml.sax.ext	Defines **SAX extensions** that are used for doing more sophisticated SAX processing-for example, to process a document type definition (DTD) or to see the detailed syntax for a file.
org.xml.sax.helpers	Contains helper classes that make it easier to use SAX-for example, by defining a **default handler** that has null methods for all the interfaces, so that you only need to override the ones you actually want to implement.
javax.xml.parsers	Defines the SAXParserFactory class, which returns the SAXParser. Also defines exception classes for reporting errors.

SAX ContentHandler

- ContentHandler methods
  - void startDocument()
  - void endDocument()
  - startElement(String uri, String localName, String qName, Attributes attributes)
  - void endElement(String uri, String localName, String qName)
  - void characters(char[] ch, int start, int length)
  - ……
- DefaultContentHandler implements them empty (do nothing)

SAX Example

//get a factory

- SAXParserFactory spf = SAXParserFactory.*newInstance();*

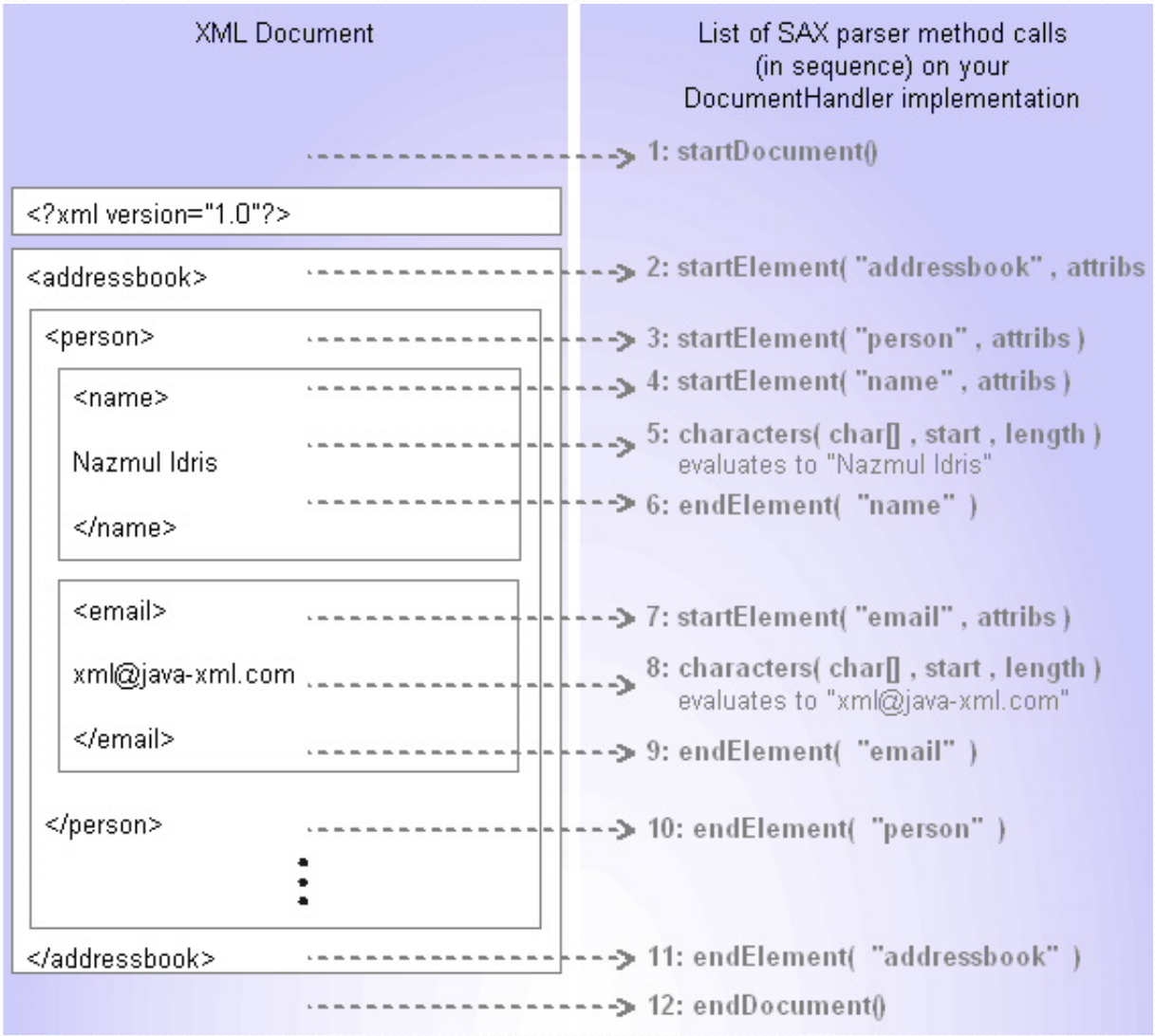
//get a new instance of parser

- SAXParser sp = spf.newSAXParser();

//parse the file and also register this class for call backs

- sp.parse("employees.xml", **this);**

SAX Example


XML Document
List of SAX parser method calls (in sequence) on your DocumentHandler implementation
1: startDocument()
<?xml version="1.0"?>
<addressbook>
2: startElement("addressbook" , attribs
<person>
3: startElement("person" , attribs)
<name>
4: startElement("name" , attribs)
Nazmul Idris
5: characters(char[] , start , length) evaluates to "Nazmul Idris"
</name>
6: endElement("name")
<email>
7: startElement("email" , attribs)
xml@java-xml.com
8: characters(char[] , start , length) evaluates to "xml@java-xml.com"
</email>
9: endElement("email")
</person>
10: endElement("person")
</addressbook>
11: endElement("addressbook")
12: endDocument()

SAX Exercise 15 Min

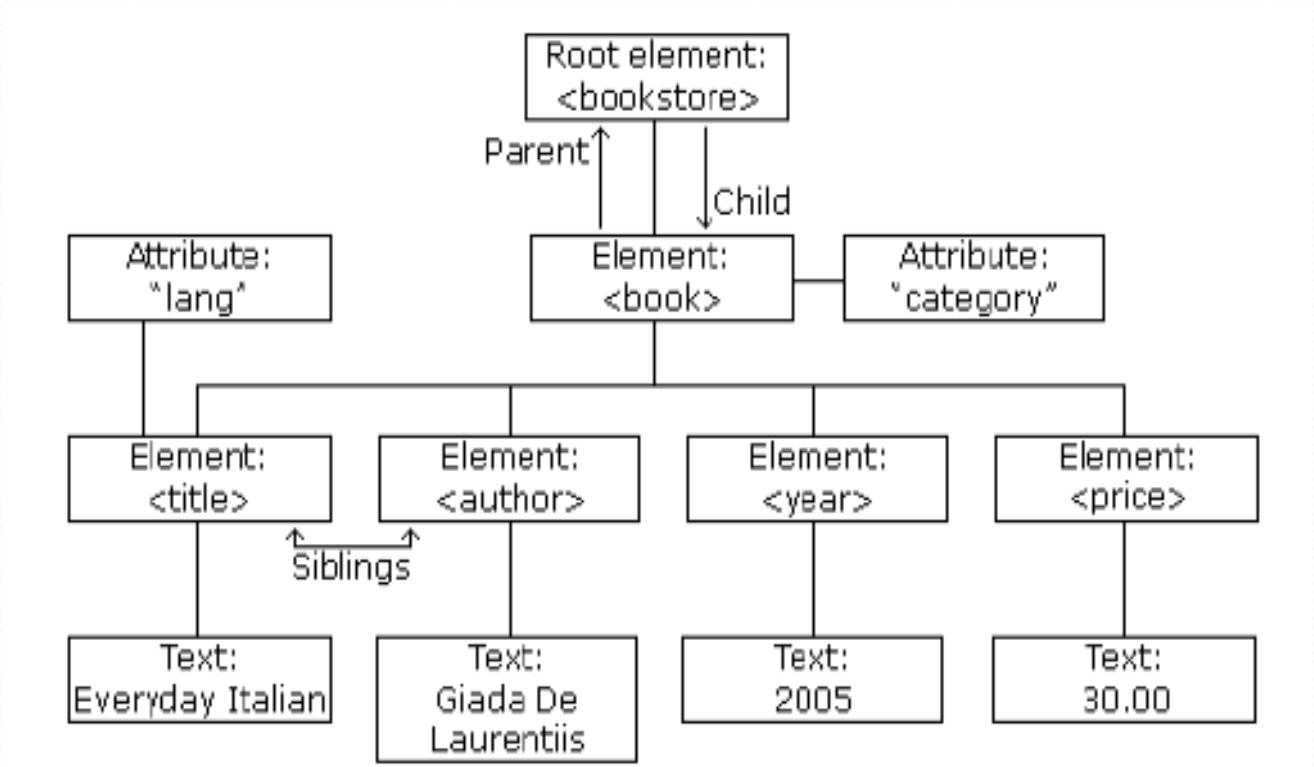
- Please download the code from
  https://sites.google.com/site/mimran15/links-for-classes/lab-documents
- Use given java classes to parse “employee.xml” file
- Add jar files to the project
- Add salary element in xml file.
- Get all employees with Age < = 25 and Salary > 3000

Note: This code will be removed before next lecture, due to space limitations.

Document Object Model (DOM)

- Home page: http://www.w3.org/DOM
- The goal of the W3C DOM specification is to define a programmatic interface to access and update **XML and HTML documents**
- The DOM API defines a set of **interfaces for tree-based XML parsing**
- A tree-based API compiles an XML document into an internal tree structure. This makes it possible for an application program to navigate the tree to achieve its objective.

DOM



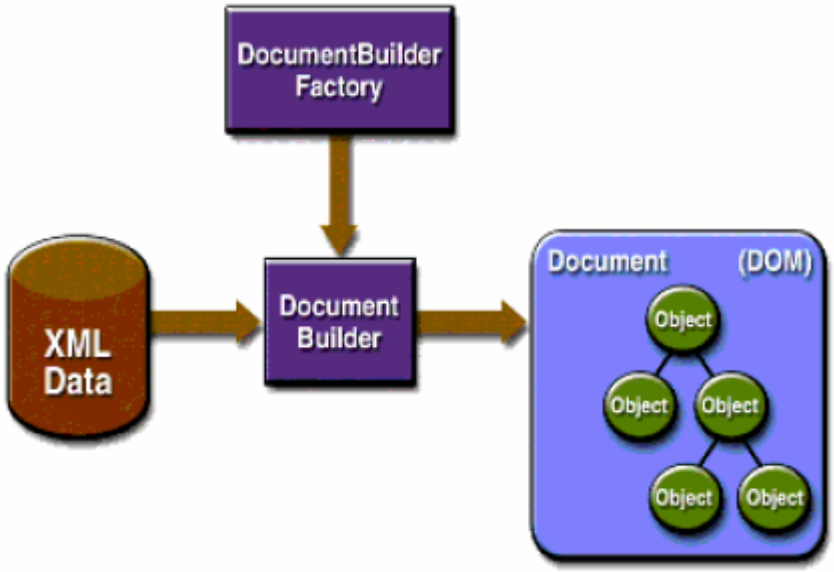
- Presents an XML document as a tree structure, with elements, attributes, comments, text as nodes

When to use DOM

- The DOM API is generally an easier API to use than SAX.
- It is ideal for interactive applications: the entire object tree is held in memory, where it can be accessed and manipulated by the user.
- It is much more CPU and memory intensive than SAX.

DOM

- *DocumentBuilderFactory* creates an instance of DocumentBuilder
- *DocumentBuilder* produces a DOM Document from the XML document
- By default, Document implements the org.w3c.dom.Document interface



Dom Java Packages

Package	Description
org.w3c.dom	Defines the DOM programming interfaces for XML (and, optionally, HTML) documents, as specified by the W3C.
javax.xml.parsers	Defines the DocumentBuilderFactory class and the DocumentBuilder class, which returns an object that implements the W3C Document interface. **The factory that is used to create the builder is determined by the javax.xml.parsers system property**, which can be set from the command line or overridden when invoking the new Instance method. This package also defines the ParserConfigurationException class for reporting errors.

DOM: Node

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

DOM: Node Methods

- Navigation Methods
  - getParentNode()
  - getChildNodes()
  - getNextSibling()
  - ….
- Inspection Methods
  - getNodeName()
  - getNodeType()
  - getNodeValue()
  - …

DOM: Node Methods

- Editing Methods
  - cloneNode(…)
  - setNodeValue(String value)
  - ….
- Editing Structure Methods
  - appendChild(Node newChild)
  - removeChild(Node oldChild)
  - replaceChild(Node newChild, Node oldChild)
  - ….

DOM: Document Methods

- Creating Methods
  - createElement(String tagName)
  - createAttribute(String name)
  - createTextNode(String data)
  - createComment(String data)

DOM

- DocumentBuilderFactory dbf = DocumentBuilderFactory.*newInstance();*

// Using factory get an instance of document builder

- DocumentBuilder db = dbf.newDocumentBuilder();

// parse using builder to get DOM representation of the XML file

- dom = db.parse("employees.xml");

DOM Exercise 15Min

- Use the DOM class from downloaded code.
- Parser the “employee.xml” file
- Get all employees with Age < = 25 and Salary > 3000

DOM Exercise

- Create another empty XML file called “Report”
- Parse the “employee.xml” by SAX parser
- Increase the salary of employee by 10% who are “permanent”
- Put those employees into “Report.xml” which are permanent and got increment in salary.