Web Data Integration

Web Data Formats

- Part 1 -
Web Data is heterogeneous with respect to the employed

1. **Exchange Format** (Technical Heterogeneity)
2. **Character Encoding** (Syntactical Heterogeneity)

- CSV
- JSON
- XML
- RDF
- Unicode
Outline

1. Web Data Formats Part I
   1. Character Encoding
   2. CSV
   3. XML
      1. Basic Syntax
      2. DTDs
      3. Namespaces
      4. XPath
      5. XSLT
      6. XML in Java

2. Web Data Formats Part II
   1. JSON
   2. RDF
Character Encoding

- Computers process only 0s and 1s ("bits")
- Every character has to be represented as a bit sequence, e.g., “A” = 0100 0001
- Character encoding: mapping of “real” characters to bit sequences
- A common problem in web data integration!

http://w3techs.com/technologies/overview/character_encoding/all
• ASCII („American Standard Code for Information Interchange“)  
  ISO 646 (1963), 127 characters (= 7 bits), 95 printable:  
  !"#$%&'()*+,-./0123456789:;<=?>@ABCDEFGHIJKLMNOPQRSTUVWXYZ\[_`abcdefghijklmnopqrstuvwxyz{|}~  

• Extension to 8 Bits: ISO 8859-1 to -16 (1998)  
  – Covers characters of European languages  
  – Well-known: 8859-1 (Latin-1)  
  – Including: Ä, Ö, Ü, ß, Ç, É, é, …  

• But the Web speaks more languages…
Character Encoding: Unicode

- ISO 10646
  - First version 1991 (Europe, Middle East, India)
  - 17 code pages of 16 bit
  - Covers even the most exotic languages

[Image: Tower of Babel comic]

Character Encoding: UTF-8

- UTF-8: Variable length encoding for Unicode
- Recommended encoding for the Internet
- Rationale:
  - Common characters only get one byte
  - Less common ones are encoded in 2-6 bytes
  - Fast transmission of files over the internet!

<table>
<thead>
<tr>
<th>Bits of code point</th>
<th>First code point</th>
<th>Last code point</th>
<th>Bytes in sequence</th>
<th>Byte 1</th>
<th>Byte 2</th>
<th>Byte 3</th>
<th>Byte 4</th>
<th>Byte 5</th>
<th>Byte 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>U+0000</td>
<td>U+007F</td>
<td>1</td>
<td>0xxxxxx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>U+0080</td>
<td>U+07FF</td>
<td>2</td>
<td>110xxxx</td>
<td>10xxxxx</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>U+0080</td>
<td>U+FFFF</td>
<td>3</td>
<td>1110xxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>U+10000</td>
<td>U+1FFFF</td>
<td>4</td>
<td>11110xx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>U+200000</td>
<td>U+3FFFF</td>
<td>5</td>
<td>11111xx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>U+4000000</td>
<td>U+7FFFFFFFF</td>
<td>6</td>
<td>1111110</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
<td>10xxxxx</td>
</tr>
</tbody>
</table>

Handling Character Encoding

- Editors such as Notepad++ support encoding conversion
Handling Character Encoding in Java

- FileInputStreams allow you to specify the character encoding.

```java
System.out.println("with correction:");
BufferedReader BR2 = new BufferedReader(
    new InputStreamReader(
        new FileInputStream("data/encoding_utf8.txt"), "UTF8")));
while(BR2.ready())
    System.out.println(BR2.readLine());
```

![Console output showing character encoding comparison](image)
2. Comma Separated Values (CSV)

- Data Model: Table
  - Used for data exported from RDBMs and spreadsheet applications.
  - Quite widely used on the Web and in Data Portals like datahub.io, data.gov.uk

- The first line is often used for headers (attribute names).

- Example:
  
<table>
<thead>
<tr>
<th>firstname,lastname,matriculation,birthday</th>
</tr>
</thead>
<tbody>
<tr>
<td>thomas,meyer,3298742,15.07.1988</td>
</tr>
<tr>
<td>lisa,müller,43287342,21.06.1989</td>
</tr>
</tbody>
</table>

- Advantage: Data representation with minimal overhead.

- Disadvantages
  - Restricted to tabular data
  - Hard to read for humans when tables get wider
Comma Separated Values (CSV) - Variations

- Field Separators
  - Comma,
  - Semicolon
  - Tab
  - ...

- Quotation marks
  - for marking strings

- Header included
  - nor not
Processing CSV Files in Java

- Apache Commons CSV
- Provides a simple CSV API
- [http://commons.apache.org/proper/commons-csv/](http://commons.apache.org/proper/commons-csv/)
- Example

Reader `in` = `new` FileReader("data/data.csv");

`Iterable<CSVRecord> parser = CSVFormat.EXCEL.parse(in);`

`for` (CSVRecord record : parser) {
    `if`(record.getRecordNumber()>1) {
        String firstname = record.get(0);
        String lastname = record.get(1);
        ...
    }
}

skip header line
3. XML - eXtensible Markup Language

- Widely used format for data exchange
- Data model: Tree
- Standardized by W3C in 1998
- Meta language
  - defines syntax standard
  - allows the definition of specific languages (XML applications)
3.1 XML – Basic Concepts and Syntax

1. Elements
   - Enclosed by pairs of tags:
     `<physician> ... </physician>`
   - Empty elements:
     `<young />`

2. Attributes
   `<physician id="D125436">`

3. Hierarchy
   - exactly one root element!
   `<physician>
      <address> ... </address>
   </physician>`

```xml
<physician id="D125436">
  <name>Dr. Mark Smith</name>
  <address>
    <street>Main St.</street>
    <number>14</number>
    <city>Smalltown</city>
  </address>
  <telephone>
    <number>+44 123 456789</number>
  </telephone>
  <hours>
    <monday>9-11 am</monday>
    <tuesday>9-11 am</tuesday>
    ...
  </hours>
</physician>
```
XML as a Tree

```xml
<physician>
  <name>Dr. Mark Smith</name>
  <address>
    <street>Main St.</street>
    <number>14</number>
    <city>Smalltowm</city>
  </address>
  <telephone>
    <number>+44 123 456789</number>
  </telephone>
  <hours>...
  </hours>
</physician>
```
Overall Structure of an XML Document

`CD2.xml`

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE CD SYSTEM "CD.dtd"> <!-- Description of a CD -->
<CD ArticleNo="2">
  <Artist>Moby</Artist>
  <Album>Play</Album>
  <ReleaseDate>03.06.2000</ReleaseDate>
  <Label>Mute (EDEL)</Label>
  <Format>CD</Format>
</CD>
```

- **Prolog** (required)
- **Document Type Definition** (optional)
- **Comments** (optional)
- **Root Element** (required)
- **Additional Elements** (optional)
Well-formed XML Documents

- Document that complies to the syntax requirements of XML
  - Closing tag for each opening tag
  - Proper nesting of tags
  - Only one attribute with a specific name, …

**Well-formed**

```xml
<physician id="D1254">
  <name>Dr. Mark Smith</name>
  <address>
    <street>Main St.</street>
    <number>14</number>
    <city>Smalltown</city>
  </address>
  <telephone>
    <number>+44 123 456789</number>
  </telephone>
  <hours>
    <monday>9-11 am</monday>
    <tuesday>9-11 am</tuesday>
    ...
  </hours>
</physician>
```

**Not well-formed**

```xml
<physician id="D1254" id="US43759">
  <name>Dr. Mark Smith</name>
  <address>
    <street>Main St.</street>
    <number>14</number>
    <city>Smalltown</city>
  </address>
  <telephone>
    <number>+44 123 456789</number>
  </telephone>
  <hours>
    <monday>9-11 am</monday>
    <tuesday>9-11 am</tuesday>
    ...
  </hours>
</physician>
```
**HTML vs. XML**

- **HTML**: Aimed at web browsers
  - Mixes structure, content, and presentation
- **XML**: Aimed at data exchange
  - Separates structure, content, and presentation

```html
<html>
  ...
  <b>Dr. Mark Smith</b>
  <i>Physician</i>
  Main St. 14
  Smalltown
  ...
</html>
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Physician SYSTEM "Medicine.dtd">
<physician>
  <name>Dr. Mark Smith</name>
  <address>
    <street>Main St.</street>
    <number>14</number>
    <city>Smalltown</city>
  </address>
  <telephone>
    <number>+44 123 456789</number>
  </telephone>
</physician>
```
XML References

- Sometimes, we need more than trees.

```xml
<student>
  <name>Stefanie Müller</name>
  <course>
    <title>Web Data Integration</title>
    <lecturer>
      <name>Christian Bizer</name>
    </lecturer>
    <lecturer>
      <name>Heiko Paulheim</name>
    </lecturer>
  </course>
  <course>
    ...
  </course>
</student>
...
```

If we organize the XML by students, we have to repeat the courses
If we organize the XML by courses, we have to repeat the students.
XML References

- Trees are limited when it comes to n:m relations
- Problem: data duplication
  - consistency
  - storage
  - transmission volume
- Solution: IDs and references

```xml
<student id="stud01">
  <name>Stefanie Müller</name>
</student>
@student id="stud02">
  <name>Franz Maier</name>
</student>
<course>
  <title>Data Integration</title>
  <lecturer>
    <name>Christian Bizer</name>
  </lecturer>
  <student ref="stud01">
  <student ref="stud02">
</course>
```
The XML Standards Family

- **XML**: Meta language for defining markup languages; provides standard syntax.
- **DTD**: Language for defining the structure of XML documents; XML applications.
- **XML Schema**: More expressive language for defining the structure of XML documents, includes data types.
- **Namespaces**: Mechanism for distinguishing between elements from different DTDs / Schemata.
- **XPath**: Language for selecting parts of an XML document.
- **XQuery**: Query language; more flexible than XPath; similar to SQL.
- **XSLT**: Template language for transforming XML documents; uses XPath.
- **DOM, SAX**: Standardized programming interfaces for accessing XML documents from within different programming languages.
- **XPointer**: XML application for defining hyperlinks between elements in different XML documents; combines URLs and Xpath.
3.2 Document Type Definition (DTD)

- Defines valid content structure of an XML document
  - Allowed elements, attributes, child elements, optional elements
  - Allowed order of elements

- DTDs can be used to validate an XML document from the Web before it is further processed.

- XML documents are called “valid” if they are
  - well-formed (syntactically correct)
  - and suit a DTD

- DTD is part of the W3C XML Specification
Referring from a Document to its DTD

```
physician.dtd

<!ELEMENT physician (  
  name,  
  address*,  
  telephone?, fax?,  
  hours)>  

<!ELEMENT address (  
  street,  
  number,  
  city)>  

<!ELEMENT street (#PCDATA)>  
<!ELEMENT number (#PCDATA)>  

]>"}

examp.xml

<!DOCTYPE physician SYSTEM  
  "physician.dtd">  

<physician>  
  <name>Dr. Mark Smith</name>  
  <address>  
    <street>Main St.</street>  
    <number>14</number>  
    <city>Smalltown</city>  
  </address>  
  <telephone>  
    <number>+44 123 456789</number>  
  </telephone>  
  <hours>  
    <monday>9-11 am</monday>  
    <tuesday>9-11 am</tuesday>  
    ...  
  </hours>  
</physician>
```
Document Type Definition (DTD)

1. Defining child elements and their order

   ```xml
   <!ELEMENT address(street,nr,addline*,zip,city,state?) >
   ```
   - ? and * mark optional and repeatable elements, + at least once
   - #PCDATA: Parsed character data that may include further elements.
   - #CDATA: Character data that is not parsed.
   - alternative elements

2. Defining attributes

   ```xml
   <!ATTLIST person number ID #REQUIRED
       title CDATA #IMPLIED
       supervisor IDREF #IMPLIED>
   ```
   - #REQUIRED = value necessary
   - #IMPLIED = no value necessary
   - ID and IDREF are used to define references
Example: A Complete DTD

CD.dtd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- Document Type Definition: CD-Example -->

<!ELEMENT CD (Interpret, Album, ReleaseDate, Label, Format)>
<!ATTLIST CD ArticleNo CDATA #REQUIRED>

<!ELEMENT ReleaseDate (#PCDATA)>
<!ELEMENT Format (#PCDATA)>
<!ELEMENT Interpret (#PCDATA)>
<!ELEMENT Label (#PCDATA)>
<!ELEMENT Album (#PCDATA)>
```
XML Schema

- More flexible than DTDs:
  - Minimum and maximum number of elements
  - Data types (numbers, dates, …)
  - Support for namespaces
  - Modular schemas are possible

- Standardized by W3C (2004)

- XML Schema documents are XML documents themselves
  - Unlike DTDs
  - More verbose syntax
XML Schema Data Types

- Simple types are built in
- Complex types may be defined by the user
- XML schema data types are also used by RDF

http://www.w3.org/TR/xmlschema-2/
3.3 XML Namespaces

- Problem: Elements with the **same name but different meaning** may occur in different schemata.
- How can we distinguish such elements?

```xml
<physician>
  <name>Dr. Mark Smith</name>
  <address>
    <street>Main St.</street>
    <number>14</number>
    <city>Smalltown</city>
  </address>
  <telephone>
    <number>+44 123 456789</number>
  </telephone>
  <hours>
    <monday>9-11 am</monday>
    <tuesday>9-11 am</tuesday>
    ...
  </hours>
</physician>
```
XML Namespaces

Mechanism for distinguishing between elements from different DTDs/schemata by naming them with **URIs**.

- Shorthand notation with prefix for **qualified names (QNames)**: `prefix:name`
- Default namespaces are possible

```xml
<physician xmlns = "http://www.med.com/physician"
            xmlns:addr = "http://www.med.com/addr">
  <name>Dr. Mark Smith</name>
  <addr:address>
    <addr:street>Main St.</addr:street>
    <addr:number>14</addr:number>
    <addr:city>Smalltown</addr:city>
  </addr:address>
  <telephone>
    <number>+44 123 456789</number>
  </telephone>
  <hours>
    <monday>9-11 am</monday>
    <tuesday>9-11 am</tuesday>
    ...
  </hours>
</physician>
```
3.4 XPath

Language for selecting sets of nodes from an XML document.

- W3C standard since 1999 (Version 2.0: 2010)
- Used by
  - XSLT
  - XPointer
  - XML Databases
  - Java JAXP API
- Result of a XPath expression: Node Set
- Tutorial: http://www.w3schools.com/XPath/
## XPath Node Types

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Root node</strong></td>
<td>Abstract root of XML tree. Note: This node lies one level above the root XML element and is used to access processing instructions.</td>
</tr>
<tr>
<td><strong>Processing instruction node</strong></td>
<td>Processing instructions are for instance references to style sheets. All lines that start with <code>&lt;?</code> And end with <code>?&gt;</code>.</td>
</tr>
<tr>
<td><strong>Element node</strong></td>
<td>Each element of the document (Start-Tag ... End-Tag).</td>
</tr>
<tr>
<td><strong>Attribute node</strong></td>
<td>Each attribute of an element (e.g. date = “5/1/2014“).</td>
</tr>
<tr>
<td><strong>Text node</strong></td>
<td>Largest possible connected character sequence. Example: The element <code>&lt;word&gt;&lt;b&gt;C&lt;/b&gt;hris&lt;/word&gt;</code> contains two text nodes: „C“ and „hris“</td>
</tr>
</tbody>
</table>
Example: XPath Node Types

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/css" href="wetter.css"?>
<wetter datum="1.5.2002">
  <stadt name="Berlin">
    <temp>22</temp>
    <wetterlage>sonnig</wetterlage>
  </stadt>
</wetter>
```
Selecting Node Sets with XPath

- Node are addressed using localization paths.
- Each path consists of a series of localization steps, similar to addressing files in the file system.
- Example: /wetter/stadt/wetterlage

Localization Step 1
- Verarbeitungsanw.: xml-stylesheet
- Attributknoten: datum="1.5.2002"
- Attributknoten: name="Berlin"

Localization Step 2
- Elementknoten: wetter
- Elementknoten: stadt

Localization Step 3
- Elementknoten: temp
- Textknoten: 22
- Textknoten: sonnig
- Elementknoten: wetterlage
### XPath Operators

<table>
<thead>
<tr>
<th>Example</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>wetter</td>
<td>Element nodes are addressed using their name.</td>
</tr>
<tr>
<td>@datum</td>
<td>Attribute nodes are addressed using @name (Result: datum = “1.5.2002“)</td>
</tr>
<tr>
<td>temp/text()</td>
<td>Text nodes are addressed using /text() (Result: 22)</td>
</tr>
<tr>
<td>/</td>
<td>Selection of the root node</td>
</tr>
<tr>
<td>//temp</td>
<td>Selection of all temp-nodes, no matter on which level(s) of the document (Result: temp).</td>
</tr>
<tr>
<td>stadt/*</td>
<td>The /* operator selects all child element nodes. The @* operator selects all child attribute nodes. (Result1: temp, wetterlage; Result2: name = “Berlin“; Result3: wetterlage).</td>
</tr>
<tr>
<td>stadt/@*</td>
<td></td>
</tr>
<tr>
<td>stadt/*[2]</td>
<td></td>
</tr>
<tr>
<td>stadt/node()</td>
<td>Selection of all child nodes independent of their type. (Result: temp, wetterlage, name= “Berlin“)</td>
</tr>
<tr>
<td>//stadt/..</td>
<td>Using .. it is possible to address the parent elements of an element (Result: wetter).</td>
</tr>
<tr>
<td>//temp</td>
<td>//wetterlage</td>
</tr>
</tbody>
</table>
XPath Example 1: Localization starts at Root Node

- **Example:** `/physician/address/street`
- **First / stands for root node**
• Example: number

• No / before first element: Start from context node (marked green)

• Context node exist for instance in XSLT
**Example:** `/physician/*/number`

- Asterisk (*) can be any node
XPath Example 3: Using the Order of Elements

- **Example:** `/physician/*[1]`
- `*[1]` returns the first descendant with whatever name
- **Note:** The tree is ordered!
XPath Example 4: Using the Order of Elements

- **Example**: `/physician//number`
- // stands for an arbitrary line of descendants
- Selected elements can be on different depths in the tree
**Example:** `/physician/../*[1]`

- .. goes up one level
Exercise 1: XPath

Which nodes sets are selected by the following Xpath expressions?

1. /wetter/stadt/temp
2. /wetter/stadt/@name
3. /wetter/stadt/*
4. //wetterlage/text()
5. //wetterlage//@datum
6. //@datum | /wetter/stadt/@name
Solution 1: XPath

1. `/wetter/stadt/temp`
   - Elementknoten: temp
2. `/wetter/stadt/@name`
   - Attributknoten: name = “Berlin“
3. `/wetter/stadt/`*
   - Elementknoten: temp, wetterlage
4. `//wetterlage/text()`
   - Textknoten: sonnig
5. `//wetterlage/../../@datum`
   - Attributknoten: datum = “1.5.2002“
6. `//@datum | /wetter/stadt/@name`
   - Attributknoten: datum = “1.5.2002“ und name = “Berlin“
Predicates

- Predicates allow you to further restrict the selection.
- Predicates are expressed using []

<table>
<thead>
<tr>
<th>Predicate</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>city[2]</code></td>
<td>Select second city child element according to order.</td>
</tr>
<tr>
<td><code>city[@name = &quot;Berlin&quot;]</code></td>
<td>Select only city elements which have a name attribute with the value Berlin.</td>
</tr>
<tr>
<td><code>city[@name != &quot;Berlin&quot;]</code></td>
<td>Select only city elements which not have a name attribute with the value Berlin.</td>
</tr>
<tr>
<td><code>temp[text() &gt; 22]</code></td>
<td>Select only temp elements with a value exceeding 22.</td>
</tr>
<tr>
<td><code>temp[text() &gt; 22 and text() &lt; 30]</code></td>
<td>Select only temp elements with a values between 22 and 30.</td>
</tr>
</tbody>
</table>

- `/wetter/stadt[@name = "Berlin"]/temp/text()` Result: 22
- `/wetter/stadt[1]/temp/text()` Result: 22
3.5 eXtensible Stylesheet Language Transformation (XSLT)

- Widely used in data integration for translating between different XML formats.

- Basic idea:
  1. Pick values out of XML documents using XPath
  2. build new documents from them using templates
     - other XML files
     - HTML files
     - text files
     - ...

- XSLT is a Turing complete language.
  - Tutorial: http://www.w3schools.com/xsl/default.asp
XSLT Templates

- **XSLT: Template**

  ```xml
  <xsl:template match="/physician">
    <html>
      <body>
        <h2>
          <xsl:value-of select="name"/>
        </h2>
      </body>
    </html>
  </xsl:template>
  ```

- **Output:**

  ![Output Example](image-url)

  - The match attribute is used to associate a template with an XML element.
  - `xsl:value-of` is used to extract a value from the XML document and fill it into the template.
Nesting XSLT Templates

- Templates can be nested with `xsl:apply-templates`

```xml
<xsl:template match="/physician">
  <html>
    <body>
      <h2><xsl:value-of select="name"/></h2>
      <xsl:apply-templates select="address"/>
    </body>
  </html>
</xsl:template>

<xsl:template match="address">
  <h3>Address:</h3>
  <p><xsl:value-of select="street"/></p>
  <p><xsl:value-of select="number"/></p>
  <p><xsl:value-of select="city"/></p>
</xsl:template>
```

- Output

```
Dr. Mark Smith

Address:
Main St.14
Smalltown
```
Alternative: Looping through all Child Elements

```xml
<xsl:template match="/">
  <html><body><h2>My CD Collection</h2>
  <table border="1">
    <tr bgcolor="#9acd32"><th>Title</th><th>Artist</th></tr>
    <xsl:for-each select="catalog/cd">
      <xsl:sort select="artist"/>
      <tr>
        <td><xsl:value-of select="title"/></td><td><xsl:value-of select="artist"/></td>
      </tr>
    </xsl:for-each>
  </table>
</body></html>
</xsl:template>
```

Source: http://www.w3schools.com/
3.6 XML in Java

- **JAXP**: Java API for XML Processing
- Provides for:
  - parsing and validating XML documents
  - DOM interface for accessing nodes using Xpath
  - transforming documents with XSLT
- Included since Java 1.4
- Tutorial:
  [http://docs.oracle.com/javase/tutorial/jaxp/](http://docs.oracle.com/javase/tutorial/jaxp/)
Example: Validating against a DTD in Java

- As simple as that:

  ```java
  DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
  factory.setValidating(true);
  DocumentBuilder builder = factory.newDocumentBuilder();
  Document doc = builder.parse("data/data.xml");
  ```

- What happens:

  ```java
  Error: URI=file://.../data.xml Line=21:
  The content of element type "physician" must match "(name,address*,telephone?,fax,hours)".
  ```
Example: Using XPath with Java

• Loading an XML document:

```java
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
DocumentBuilder builder = factory.newDocumentBuilder();
Document doc = builder.parse("data/data.xml");
```

• Defining an XPath Expression:

```java
XPathFactory xPathFactory = XpathFactory.newInstance();
XPath xpath = xPathFactory.newXPath();
XPathExpression expr = xpath.compile("/physician/name");
```

• Using an XPath Expression:

```java
String name = expr.evaluate(doc);
System.out.println(name);
```
References and Experimentation

- **Books**

- **Tutorials**
  - XML: http://zvon.org/comp/m/tutorial.html
  - XML Schema: http://www.w3schools.com/schema/
  - XPath: http://www.w3schools.com/XPath/
  - XSLT: http://www.w3schools.com/xsl/default.asp
  - JAXP: http://docs.oracle.com/javase/tutorial/jaxp/

- **Tools**
  - XTrans: Simple Tools for experimenting with XML, DTDs, XSLT: http://www.xtrans.com-about.com/
  - Altova XMLSpy: Powerful XML Editor supporting a wide range of XML technologies: http://www.altova.com/xmlspy.html