COMP9321 Web Application Engineering

Extensible Markup Language (XML)

Dr. Basem Suleiman
Service Oriented Computing Group, CSE, UNSW Australia

Semester 1, 2016, Week 4

http://webapps.cse.unsw.edu.au/webcms2/course/index.php?cid=2442

Acknowledgement/Contributions

Service Oriented Computing Group, CSE, UNSW Australia

- Dr. Helen Paik
- Prof. Boualem Bentallah
- Dr. Srikumar Venugopal
- Dr. Moshe Chai Barukh
- Dr. Amin Beheshti
- Dr. Basem Suleiman
- Many others from service oriented computing group

Extensible Markup Language (XML)

- XML is a markup language much like HTML
  - HTML:
    - HTML was designed to display data.
    - HTML tags are not predefined.
  - XML:
    - XML was designed to describe data.
    - XML tags are not predefined.
- XML is designed to be self-descriptive
- XML is a W3C Recommendation
- XML is derived from SGML (ISO 8879).
  - (Standard Generalized Markup Language) is a standard for how to specify a document markup language or tag set.

Extensible Markup Language (XML)

- XML originally designed to meet the challenges of large-scale electronic publishing.
- XML separates presentation issues from the actual data.
- XML plays an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere.
  - Needs a communication protocol?
  - e.g. SOAP stands for Simple Object Access Protocol
    - SOAP is based on XML
    - SOAP is a W3C recommendation
    - SOAP uses XML Information Set for its message format.

Extensible Markup Language (XML)

```
<!DOCTYPE html>
<html>
<body>

<h2>COMP9321 Students</h2>

<ul>
  <li>Student#: 50001</li>
  <li>Name: Adam B.</li>
  <li>Program: 8543</li>
  <li>Stage: 1</li>
</ul>

<ul>
  <li>Student#: 50002</li>
  <li>Name: Alex C.</li>
  <li>Program: 3978</li>
  <li>Stage: 3</li>
</ul>

...

</body>
</html>
```

COMP9321 Students

- Student#: 50001
- Name: Adam B.
- Program: 8543
- Stage: 1

- Student#: 50002
- Name: Alex C.
- Program: 3978
- Stage: 3

...

```
<?xml version="1.0" encoding="UTF-8"?>
<comp9321_students>
 <student>
   <id>50001</id>
   <name>Adam B.</name>
   <program>8543</program>
   <stage>1</stage>
 </student>
 <student>
   <id>50002</id>
   <name>Alex C.</name>
   <program>3978</program>
   <stage>3</stage>
 </student>
</comp9321_students>
```

```
<?xml version="1.0" encoding="UTF-8"?>
- <comp9321_students>
  - <student>
      <id>50001</id>
      <name>Adam B.</name>
      <program>8543</program>
      <stage>1</stage>
    </student>
  - <student>
      <id>50002</id>
      <name>Alex C.</name>
      <program>3978</program>
      <stage>3</stage>
    </student>
</comp9321_students>
```

XML: Tags, tags

Consider the following snippet of information from a staff list:

LName	Title	FName	School	Campus	Room
Edgar	Miss	Pam	Optometry	KG	B501
Edmond	Dr	David	Information Systems	GP	S842
Edmonds	Dr	Ian	Physical Sciences	GP	M206

In XML ...

```
<Phonebook>
 <Entry>
  <LastName>Edgar</LastName>
  <Title>Miss</Title>
  <FistName>Pam</FirstName>
  <School>Optometry</School>
  <Campus>KG</Campus>
  <Room>B501</Room>
 </Entry>
 <Entry>
  <LastName>Edmond</LastName>
  <Title>Dr</Title>
  <FistName>David</FirstName>
  <School>Information Systems</School>
  <Campus>GP</Campus>
  <Room>S842</Room>
 </Entry>
 <!-- Entry continues ... -->
 </Phonebook>
```

Why XML?

Early Web

- Used to publish documents to be read by humans
- HTML was designed for the purpose

Today's Web

- Many business activities are performed on the Web
  - Dynamic interactions:
    Web app $\Leftrightarrow$ people / Web app $\Leftrightarrow$ Web app
  - Web becomes a platform for data exchange
  - XML provides a simple, cross-platform data format
- Web contains vast amount of data published in HTML format
  - Many programs process or analyse such data
  - HTML changes ... (when data inside does not) $\rightarrow$ the program that reads the HTML page must change too
  - XML provides a long-term, reliable data format for publishing

Why XML?

Benefits of using XML in document (data) exchange

- Self-describing, modular and portable data
- A common, widely accepted data representation language
- Standard supports available for creating/parsing XML docs
- Standard supports for checking validity of data
- Efficient search of business information
  - standard support for querying XML docs
  - quick and simple search (XPath)
  - more comprehensive keyword + structure based search possible as well (XQuery)
- Extensible document descriptions
  - XML is flexible (cf. relational tables)!
  - reuse, adaptation of existing documents

Separating the Content from Presentation

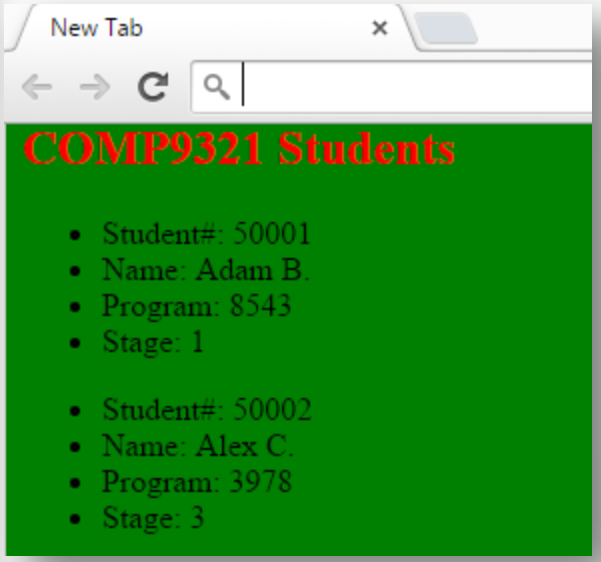
HTML was designed to display data.

- **CSS**: **C**ascading **S**tyle **S**heets
- CSS defines **how HTML elements are to be displayed**
- All formatting could be removed from the HTML document, and stored in a separate CSS file.

XML was designed to describe data.

Separating the Content from Presentation

```
<!DOCTYPE html>
<html>

<head>
  <style>
    body {background-color: green;}
    h2 {color: red;}
  </style>
</head>

<body>

<h2>COMP9321 Students</h2>

<ul>
  <li>Student#: 50001</li>
  <li>Name: Adam B.</li>
  <li>Program: 8543</li>
  <li>Stage: 1</li>
</ul>

<ul>
  <li>Student#: 50002</li>
  <li>Name: Alex C.</li>
  <li>Program: 3978</li>
  <li>Stage: 3</li>
</ul>

...

</body>
</html>
```



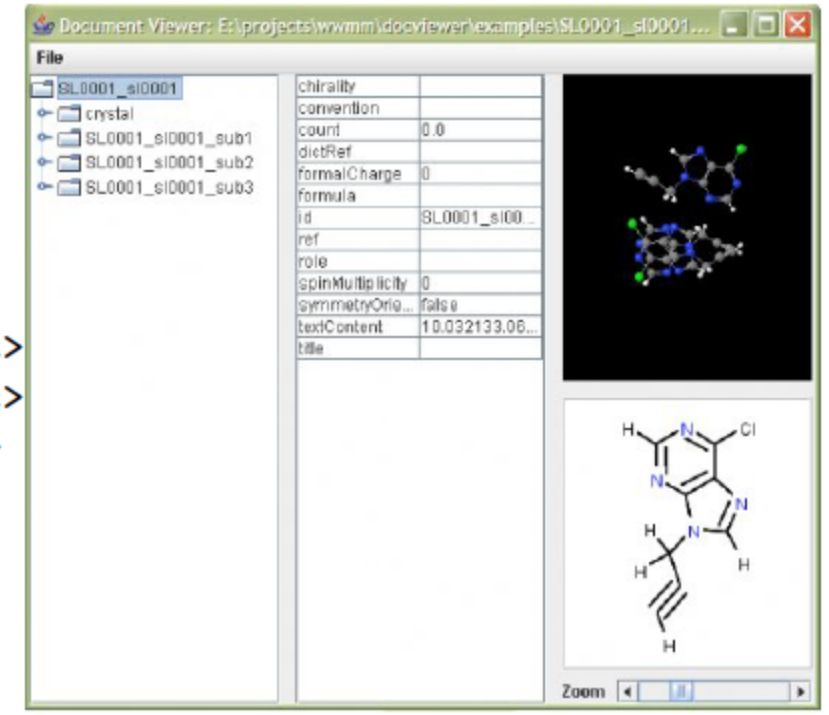
XML Applications

Like any other good inventions, XML is now used for things that are far beyond its creators original imagination.

- A set of 'tags' that are developed for specific types of documents.

e.g., Chemical Markup Language (CML)

```
<atom id="caffeine_karne_a_1">
 <float builtin="x3" units="A">-2.8709</float>
 <float builtin="y3" units="A">-1.0499</float>
 <float builtin="z3" units="A">0.1718</float>
 <string builtin="elementType">C</string>
</atom>
```



XML Applications

RSS : **R**eally **S**imple **S**yndication

With RSS it is possible to distribute up-to-date web content from one web site to thousands of other web sites around the world.

- RSS is written in XML
- RSS allows you to syndicate your site content
- RSS defines an easy way to share and view headlines and content
- RSS files can be automatically updated
- RSS allows personalized views for different sites
- RSS is useful for web sites that are updated frequently, like:
  - e.g. News sites, Companies, and Calendars.

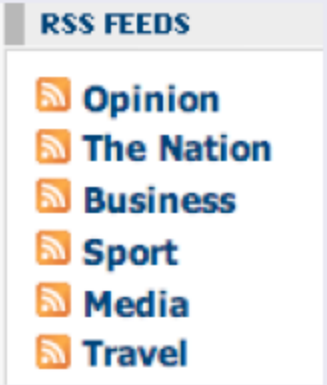
Without RSS, users will have to check your site daily for new updates.

XML Applications

RSS : **R**eally **S**imple **S**yndication

Really Simple Syndication (RSS)

```
<rss version="0.91">
<channel>
<title>CNN.com</title>
<item>
<title>July ends with 76 ... killed</title>
<link>http://www.cnn.com/.../story.html</link>
<description>Three U.S. soldiers were ...</description>
</item>
```



Without RSS, users will have to check your site daily for new updates.

XML is …

- Is a Language
  - there is a grammar, and it can be parsed by machines
- Is a Markup Language
  - XML looks a bit like HTML (tags).
  - But it describes what things are, not what they are supposed to do
- Is eXtensible
  - you can define more words and add to the language
- XML is for structuring data.
- XML is for describing data.
- XML is text, but isn't meant to be read.
- XML is verbose by design.
- XML is a family of technologies.
- XML is license-free, platform-independent and well-supported.
- **XML is NOT a programming language.**
  - it is not something you can 'compile'

Quick XML syntax

- All XML documents must have 'a' root element
- All XML elements must have a closing tag
  - Empty element tags end with / >
- XML tags are case sensitive (NAME vs. Name)
- All XML elements must be nested (<q> </q> ??)
- Element Naming
  - letters, numbers, and other characters
  - must not start with a number, '. (period)' or '- (hyphen)'
  - must not start with 'xml' (or XML or Xml ..)
  - cannot contain spaces
- Attribute values must always be quoted (single or double)
- Comments in XML: `<!-- This is a comment -->`

The XML Family

XML: a markup language used to describe information.

The XML Family

XML: a markup language used to describe information.

Document Object Model (DOM)

- XML DOM defines a standard for accessing and manipulating XML documents.
- The DOM presents an XML document as a tree-structure.
- The DOM is a W3C standard.
- The DOM is separated into 3 different parts / levels:
  - Core DOM - standard model for any structured document
  - XML DOM - standard model for XML documents
    - A standard object model for XML
    - A standard programming interface for XML
    - Platform- and language-independent
  - HTML DOM - standard model for HTML documents

The XML Family

XML: a markup language used to describe information.

Document Object Model (DOM):

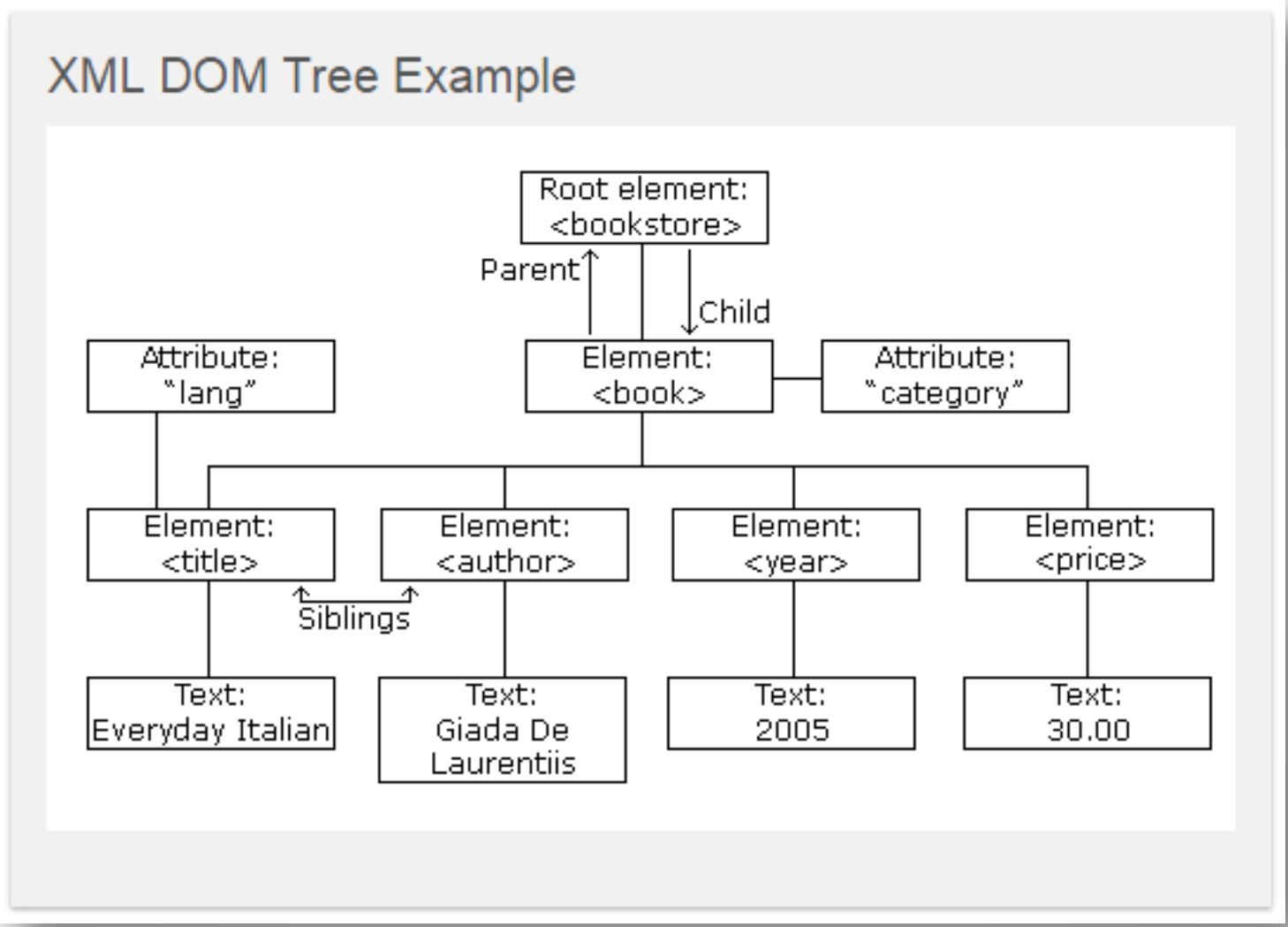
- XML DOM defines a standard for accessing and manipulating XML documents.
- The DOM presents an XML document as a tree-structure.
- The DOM is a W3C standard.
- The DOM is separated into 3 different parts / levels:
  - Core DOM - standard model for any structured document
  - XML DOM - standard model for XML documents
    - A standard object model for XML
    - A standard programming interface for XML
    - Platform- and language-independent
  - HTML DOM - standard model for HTML documents

XML Document: Bookstore Example

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

<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="Web Technologies">
        <title lang="en">Extensible Markup Language (XML) </title>
        <author>W3C</author>
        <year>1998</year>
        <price>0.00</price>
    </book>

    <book category="Web Development">
        <title lang="en">JavaServer Pages (JSP)</title>
        <author>Jason Hunter</author>
        <year>2000</year>
        <price>35.00</price>
    </book>

</bookstore>

```

Using xml-formatter, Web Toolkit Online http://www.webtoolkitonline.com/xml-formatter.html

XML Tree Structure: Bookstore Example



Source: http://www.w3schools.com/xml/xml_tree.asp

The XML Family – Basics

- XML documents are formed as **element trees**
- **XML prolog:** XML version and encoding character - <?xml version="1.0" encoding="UTF-8"?>
- An XML tree starts at a **root element** <bookstore> and branches from the root to **child elements (e.g., <book>)**
- All elements can have sub elements (child elements, e.g., **<author>)**
- The terms parent, child, and sibling describes the relationships between elements
- Parent have children. Children have parents. Siblings are children on the same level (brothers and sisters)
- All elements can have text content (e.g., Everyday Italian) and attributes (e.g., **<book category="cooking">**)
- Elements has starting and closing tags (**<price> </price>**)

The XML Family – Basics

- XML tags are case sensitive. <Student> is different from <student>
- All elements **must** be properly nested within each other
- Attribute values must always be quoted - <student id="3230813">
- Comments <!-- XML comments -->
- **White spaces** are preserved in XML (in contrast to HTML)
- XML Stores **New Line** as LF (carriage return and line feed)

- **Entity References**
  - Some characters are reserved and have special meaning in XML, e.g., <, >, &
  - Must be escaped using entity references

<Age> should be **<** 25 **</Age>**

Character	Entity reference
< (less than)	<
> (greater than)	>
& (ampersand)	&
‘ (apostrophe)	'
" (quotation)	"

XML – Document Type Definition

- An XML document with correct syntax is called "**Well Formed**"
- Errors (incorrect syntax) – application processing will trigger errors
- Well Formed XML document → it has valid XML syntax rules
- Well formed XML document → "valid" XML document ?
- A **valid XML** document must be:
  - Well formed AND
  - Conform to Document Type Definition (DTD)
- **Document Type Definition (DTD)**
  - Defines the structure and the legal elements and attributes of an XML document
  - DTD or XML Schema (XML alternative to DTD)
  - Internal DTD declaration or external DTD declaration (.dtd)

DTD Parsing

In DTD

- **Parsed Character DATA (PCDATA)**
  - Text that WILL be parsed by a parser. - tags inside the text will be treated as markup and entities will be expanded

- **Character Data (CDATA)**
  - Text that will NOT be parsed by a parser – tags inside the text will NOT be treated and entities will not be expanded.

Phonebook.xml with Internal DTD

```
<?xml version="1.0"?>
<!DOCTYPE Phonebook [
  <!ELEMENT Phonebook (Entry)+ >
  <!ELEMENT Entry (LastName, FirstName, School,  Campus, Room, Extension)>
  <!ELEMENT LastName  (#PCDATA)>
  <!ELEMENT FirstName (#PCDATA)>
  <!ELEMENT School    (#PCDATA)>
  <!ELEMENT Campus    (#PCDATA)>
  <!ELEMENT Room      (#PCDATA)>
  <!ELEMENT Extension (#PCDATA)>
  <!ATTLIST LastName Title (Miss | Ms | Mrs | Mr | Dr | Prof) #REQUIRED>
]>
<Phonebook>
   <Entry>
      <LastName Title="Miss">Edgar</LastName>
      <FirstName>Pam</FirstName>
      <School>Optometry</School>
      <Campus>GP</Campus>
      <Room>B501</Room>
      <Extension>5695</Extension>
   </Entry>  <!-- more entries not shown ... -->
</Phonebook>
```

Phonebook.xml with **External** DTD. Phonebook.dtd

Phonebook.xml

```
<?xml version="1.0"?>
<!DOCTYPE Phonebook SYSTEM "Phonebook.dtd">
<Phonebook>
  <Entry>
    <LastName Title="Miss">Edgar</LastName>
    <FirstName>Pam</FirstName>   <!-- rest of the entries -->
</Phonebook>
```

Phonebook.dtd

```
<!ELEMENT Phonebook (Entry+) >
<!ELEMENT Entry (LastName, FirstName, School,Campus, Room, Extension)>
<!ELEMENT LastName  (#PCDATA)>
<!ELEMENT FirstName (#PCDATA)>
<!ELEMENT School    (#PCDATA)>
<!ELEMENT Campus    (#PCDATA)>
<!ELEMENT Room      (#PCDATA)>
<!ELEMENT Extension (#PCDATA)>
<!ATTLIST LastName Title (Miss | Ms | Mrs | Mr | Dr | Prof) #REQUIRED>
```

CDATA Section

Sometimes the character data of an element might contain too many characters that need to be escaped (e.g., chunk of other XML parts or HTML code).

CDATA section lets you enclose the character data as literal.

Example

```
<p>You can use default <code>xmlns</code> to attribute to avoid
having to add the svg prefix to all your elements:</p>
   <![CDATA[
        <svg xmlns="http://www.w3.org/2000/svg"
                width="12cm" height="12cm"
             <eclipse rx="110" ry="130"/>
             <rect x = "4cm" y="1cm" width="3cm" height="6cm"/>
        </svg>
   ]]>
```

Everything between <!CDATA[and]]> is treated as raw characters, not markups.

Defining XML Content: Elements

A Book

```
<book>
<author>
    <name>J.K. Rowling</name>
</author>
<detail>
    <series>Seventh</series>
    <title>Harry Potter and the Deathly Hallows</title>
</detail>
</book>
```

Creating Elements:

```
<!ELEMENT book (author, detail)>
<!ELEMENT author (name)>
<!ELEMENT name (#PCDATA)>
```

```
<!ELEMENT detail (series, title)>
<!ELEMENT series (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```

Defining XML Content: Modifiers

A Book

```
<book>
<author> <!– more than one authors? –>
   <name>E. Harold</name> <name>S. Means</name>
</author>
<detail> <!– not every book is in a series –>
   <title>XML in a Nutshell</title>
</detail> </book>
```

1. ? : optional element (only once)
2. + : mandatory element (1 or more)
3. * : optional element (0 or more)

```
<!ELEMENT book (author, detail*)>
<!ELEMENT author (name+)>
<!ELEMENT name (#PCDATA)>
```

```
<!ELEMENT detail (series?, title)>
<!ELEMENT series (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```

Defining XML Content: Choices, Empty

Element Choices

```
<!ELEMENT newbooks (book+)>
<!ELEMENT book (author+, detail*)>
<!ELEMENT author (name | penname)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT penname (#PCDATA)>
<!ELEMENT detail ((series?, title) | (publisher, release*))>
<!ELEMENT series (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT release (#PCDATA)>
```

Empty Element Content

```
<!ELEMENT BR EMPTY>
```


 is called an empty element

Defining XML Content: Mixed content, Any

Mixed content: mixture of elements and text

```
<!ELEMENT message (#PCDATA | bold | italic)*>
```

```
<message>You <italic> really <bold>must</bold> try this delicious
<bold>new</bold> recipe for <italic>pudding </message>
```

ANY : Any predefined element could be included

```
<!ELEMENT book (author+, description, detail*)>
<!ELEMENT author (name+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT description ANY>
<!ELEMENT detail (series?, title)>
<!ELEMENT series (#PCDATA)>
<!ELEMENT title (#PCDATA)>
```

Defining XML Content: Creating Attributes

```
<book>
  <author period="classical" category="children">
    <name type="normal">J.K. Rowling</name>
  </author>
  <title>Harry Potter and the Half-Blood Prince</title>
</book>
```

Creating Attributes:

```
<!ELEMENT book (author, title)>
<!ELEMENT author (name+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT title (#PCDATA)>
<!ATTLIST name type (normal | penname) "normal" #REQUIRED>
<!ATTLIST author period CDATA #REQUIRED
                 category CDATA #IMPLIED>
```

Defining XML Content: Creating Attributes

Default values for attributes:

The default postcode in an address is to be 4001, state must be QLD.

```
<!ATTLIST Address Postcode CDATA "4001"
                  State CDATA #FIXED "QLD">
```

The above definition has the following effects on the source doc.

- <Address /> → <Address Postcode="4001" State="QLD">
- <Address Postcode="4010" State="QLD" /> → (no error)
- <Address Postcode="4001" State="NSW" /> → (error)

XML Custom Entities

Sometimes it might be desirable to construct a document from several (other) XML documents:

```
<!DOCTYPE sql [
<!ELEMENT sql (select, from)>
<!ELEMENT select (col+)>
<!ATTLIST select order CDATA #REQUIRED>
<!ELEMENT col (#PCDATA)>
<!ELEMENT from (table+)>
<!ELEMENT table (#PCDATA)>
<!ENTITY select SYSTEM "select.xml">
<!ENTITY from   SYSTEM "from.xml">]>
<sql>&select;&from;</sql>
```

Where "select.xml" contains:

```
<select order="cost">
<col>CarNr</col>
<col>Make</col>
<col>Cost</col>
</select>
```

And "from.xml" contains:

```
<from>
<table>Cars</table>
</from>
```

Parameter Entities

It might be a good idea to fragment the DTD in the same way that the document content is partitioned:

```
<!DOCTYPE sql [
<!ELEMENT sql (select, from)>
<!ENTITY % seldef SYSTEM "select.dtd">
%seldef;
<!ENTITY % fromdef SYSTEM "from.dtd">
%fromdef;
<!ENTITY select SYSTEM "select.xml">
<!ENTITY from   SYSTEM "from.xml">]>
<sql>&select;&from;</sql>
```

Where select.dtd is defined as:

```
<!ELEMENT select (col+)>
<!ATTLIST select order CDATA #REQUIRED>
<!ELEMENT col (#PCDATA)>
```

and "from.dtd" is:

```
<!ELEMENT from (table+)>
<!ELEMENT table (#PCDATA)>
```

Well-formedness and Validity of XML

Well-formedness Rules:

- Open and close all tags
- Empty-element tags end with />
- There is a unique root element
- Elements may not overlap
- Attribute values are quoted
- < and & are only used to start tags and entity references, respectively
- Only the five predefined entity references are used

Validity Rules:

- Well-formed
- Must have a Document Type Definition (DTD)
- Must comply with the constraints specified in the DTD

Limitations of DTD

- Limited support for constraining attribute values
- No limits on character data
- Limited support for namespaces
- No self-documentation, does not have XML syntax

The XML Family – XML Schema

XML Schema (or SML Schema Definition XSD)

- is an XML-based alternative to DTD.
- describes the structure of an XML document.
- defines elements and attributes that can appear in a document
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes
- defines the child elements, their orders, etc.
- XML Schemas are much more powerful than DTDs.
- The XML Schema language is also referred to as XML Schema Definition (XSD).

The XML Family – XML Schema

XML Schema (orXSD)

- W3C's recommendation for replacing DTD with features such as:
- Simple and complex data types
- Type derivation and inheritance
- Namespace-aware element and attributes
- Limits on number of appearances by an element
- Combining with regular expressions for finer control over document structure

- Most importantly, XML Schemas are well-formed XML documents themselves

XML Namespaces

- XML elements can have *any* names.
- What if a name could mean two different things (ie., name clash)?

The following two XML documents that describe student information.

From University X:

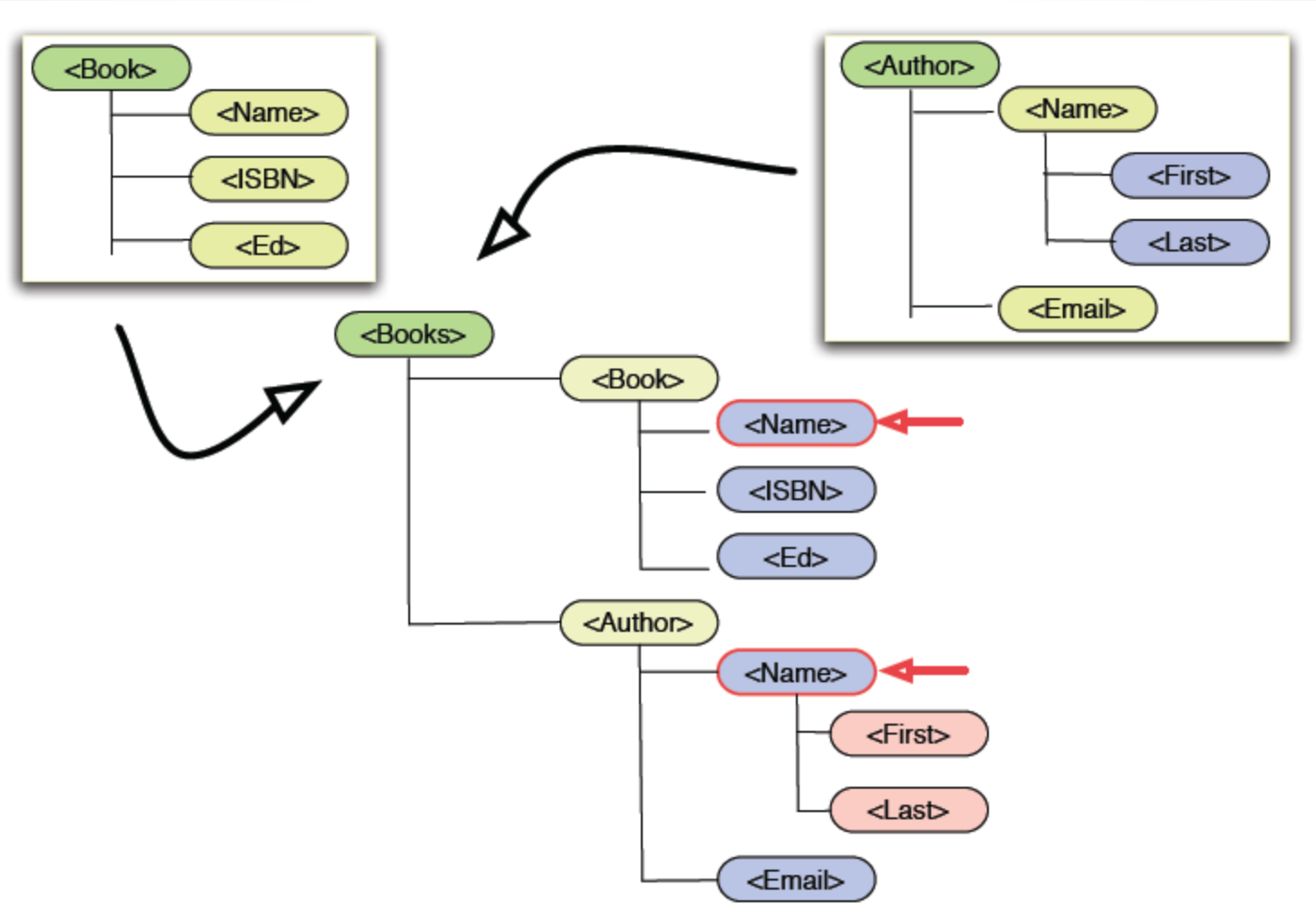
```
<student>
 <id>12345</id>
 <name>Jeff Smith</name>
 <language>C#</language>
 <rating>9.5</rating>
</student>
```

From University Y:

```
<student>
 <id>534-22-5252</id>
 <name>Bob Citizen</name>
 <language>Spanish</language>
 <rating>3.2</rating>
</student>
```

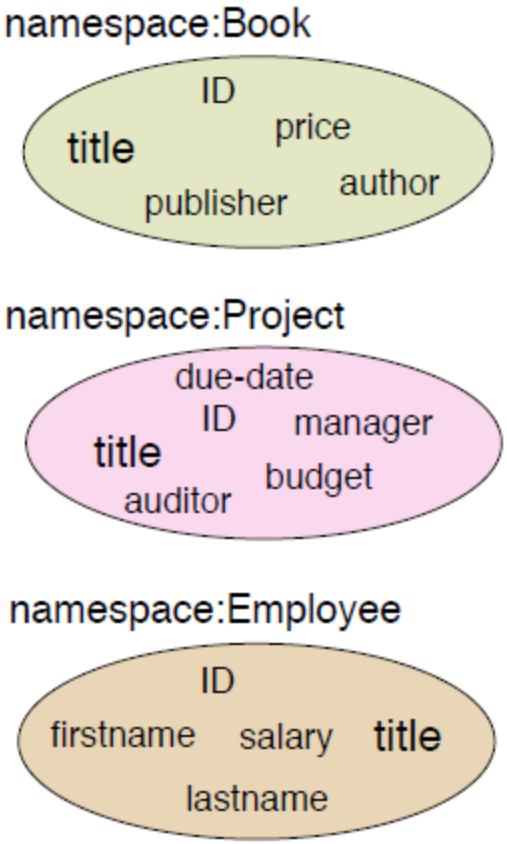
How could a program distinguish the different elements?

XML Namespaces (example)



XML Namespaces

A namespace is a set of *names* in which all names are unique.

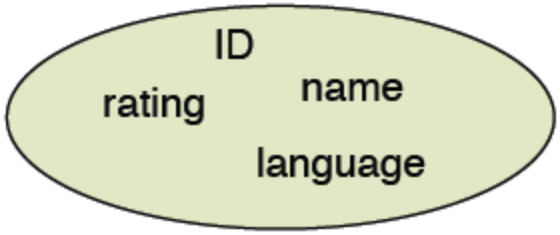


- The name 'title' can now be identified as: Book.title, Project.title, Employee.title ...
- These names are called "qualified names".
- XML namespaces give elements and attributes a unique name across the Internet.
- XML namespaces enable programmers to process the tags and attributes they care about and ignore those that don't matter to them.

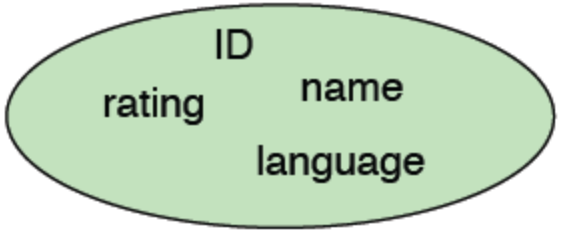
Previous examples can now be ...

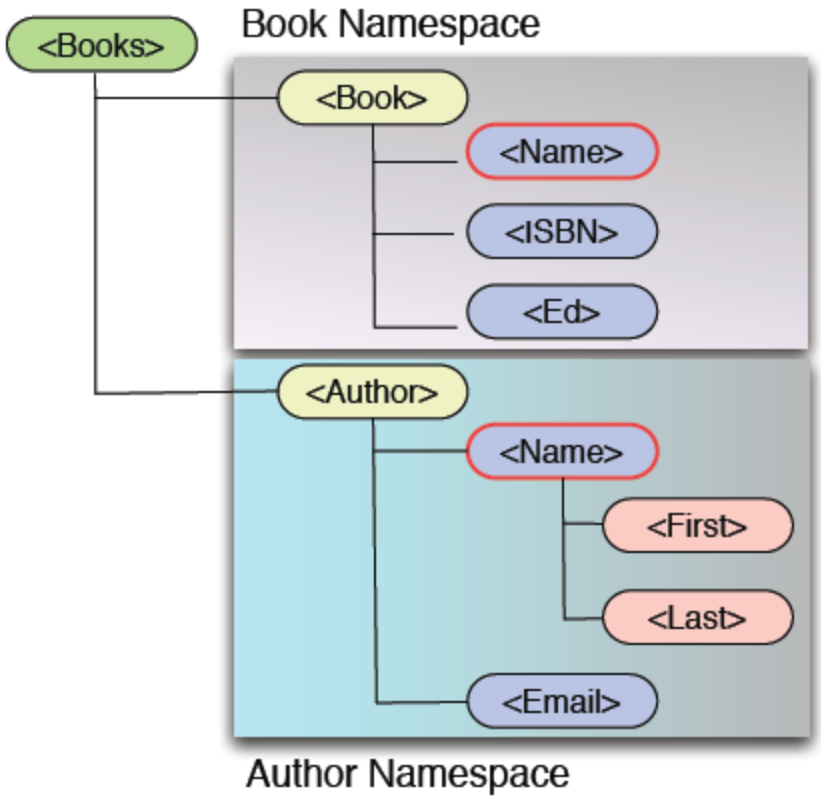
The previous examples can now have qualified names:

namespace:UniversityX/Student



namespace:UniversityY/Student





XML Namespace Syntax

- xmlns:<prefix>='namespace identifier'
- eg., <books xmlns:xdc="http://www.xml.com/books">
- not a normal XML attribute (treated differently)
- the URI must be unique, but may not represent a 'useful' resource
- the prefix is by convention or author's choice

Consider the following XML document: painting.xml

```
<catalog>
<rdf:RDF xmlns:rdf="http://www.w3.org/TR/REC-rdf-syntax#">
 <rdf:Description xmlns:dc="http://purl.org/dc/"  about="painting.xml">
   <dc:title>Impressionist Paintings</dc:title>
   <dc:creator>Elliotte Rusty Harold</dc:jcreator>
   <dc:description>impressionist paintings</dc:description>
   <dc:date>2000-08-22</dc:date>
 </rdf:Description>
</rdf:RDF>
<painting>
  <title>Memory of the Garden at Etten</title>
  <artist>Vincent Van Gogh</artist>
  <date>1888</date>
  <descrption>Two women look to the left.</description>
</painting> </catalog>
```

XML Schema Definition (XSD)

- a recommendation of the World Wide Web Consortium (W3C)
- specifies how to formally describe the elements in an Extensible Markup Language (XML) document.

Types and Declarations

- Simple Types - Basic data types such as strings, integers, boolean, etc.
- Complex Types - Composed of simple types. Consists of an arrangement of elements and attributes.
- Element declarations - Associates an element name to an instance of a simple or complex type
- Attribute declaration - Associates an attribute name with an instance of a simple type.

Simple Types

Simple Type Declaration

- Cannot contain elements or attributes
- Can be pre-defined or user-defined.

Examples:

```
<element name='sname' type="string" />
<element name='age' type="integer" />
<element name='course' type='string'/>
```

Which defines:

```
<sname>John Doe</sname>
<age>24</age>
<course>COMP9321</course>
```

Attributes

Attributes are simple types as well (however, simple types themselves CANNOT contain attributes)

```
<attribute name="currency" type="string"/>
```

Attributes can have default or fixed values and can be optional or required

```
<attribute name="currency" type="string" default="EUR"/>
<attribute name="currency" type="string" fixed="AUD"/>
<attribute name="currency" type="string" use="required"/>
```

- <attribute name="attribute-name" type="attribute-type" use="uu"/>
- Built-in data types
  - String, decimal, integer, boolean, date, time
- A default value is automatically assigned to the attribute when no other value is specified
- A fixed value is also automatically assigned to the attribute, and you cannot specify another value
- Attributes are optional by default, otherwise use="required"

Type Restrictions

Restrictions are used to specify a range of acceptable values for XML elements or attributes.

Examples:

```
<simpleType name="nameString">
<restriction base="string">
<pattern value="([a-zA-Z])+"/>
</restriction>
</simpleType>
<element name='sname' type="tns:nameString" />

<simpleType name="ageNum">
<restriction base="integer">
<minInclusive value="1"/>
<maxInclusive value="80"/>
</restriction>
</simpleType>
<element name='age' type="tns:ageNum" />

<simpleType name="courseString">
<restriction base="string">
<enumeration value="COMP9321"/>
<enumeration value="COMP9322"/>
<enumeration value="COMP9323"/>
</restriction>
</simpleType>
<element name='course' type="tns:courseString"/>
```

More examples : http://www.w3schools.com/xml/schema_facets.asp

Type Restrictions

Constraint	Description
enumeration	Defines a list of acceptable values
fractionDigits	Specifies the maxi number of decimal places allowed. Must be equal to or greater than zero
length	Specifies the exact number of characters or list items allowed. Must be >= than zero
maxExclusive	Specifies the upper bounds for numeric values (the value must be less than this value)
maxInclusive	Specifies the upper bounds for numeric values (the value must be <= to this value)
maxLength	Specifies the Max number of characters or list items allowed. Must be >= zero
minExclusive	Specifies the lower bounds for numeric values (the value must be greater than this value)
minInclusive	Specifies the lower bounds for numeric values (the value must be>= to this value)
minLength	Specifies the minimum number of characters or list items allowed. Must be >= 0
pattern	Defines the exact sequence of characters that are acceptable
totalDigits	Specifies the exact number of digits allowed. Must be greater than zero
whiteSpace	Specifies how white space (line feeds, tabs, spaces, and carriage returns) is handled

More examples : http://www.w3schools.com/xml/schema_facets.asp

Complex Types

Complex Types can be empty or composed of only elements, or only text, or a mix of both elements and text.
The number of elements are controlled by the *indicators* as below:
Order indicators:

- All - all elements specified in the type can occur in any order but must occur only once
- Choice - either one or the other element must be present
- Sequence - all elements must occur in the order specified. You can have more than one element.

Occurrence indicators:

- maxOccurs
- minOccurs

Complex Types

```
<complexType name="student">
<sequence>
<element name='sname' type="tns:nameString" />
<element name='age' type="tns:ageNum" />
<element name='course' type="tns:courseString"/>
</sequence>
</complexType>
```

which defines:

```
<student>
<sname>John Doe</sname>
<age>24</age>
<course>COMP9321</course>
</student>
```

XML Example - Creating XSD

Shiporder.xml

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

<shiporder orderid="889923"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="shiporder.xsd">
  <orderperson>John Smith</orderperson>
  <shipto>
    <name>Ola Nordmann</name>
    <address>Langgt 23</address>
    <city>4000 Stavanger</city>
    <country>Norway</country>
  </shipto>
  <item>
    <title>Empire Burlesque</title>
    <note>Special Edition</note>
    <quantity>1</quantity>
    <price>10.90</price>
  </item>
  <item>
    <title>Hide your heart</title>
    <quantity>1</quantity>
    <price>9.90</price>
  </item>
</shiporder>
```

Shiporder.xsd

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="shiporder">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="orderperson" type="xs:string"/>
      <xs:element name="shipto">
        <xs:complexType>
          <xs:sequence>
            <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:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="item" maxOccurs="unbounded">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="title" type="xs:string"/>
            <xs:element name="note" type="xs:string" minOccurs="0"/>
            <xs:element name="quantity" type="xs:positiveInteger"/>
            <xs:element name="price" type="xs:decimal"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
    <xs:attribute name="orderid" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

Source: http://www.w3schools.com/xml/schema_example.asp

The XML Family – XSL and XSLT

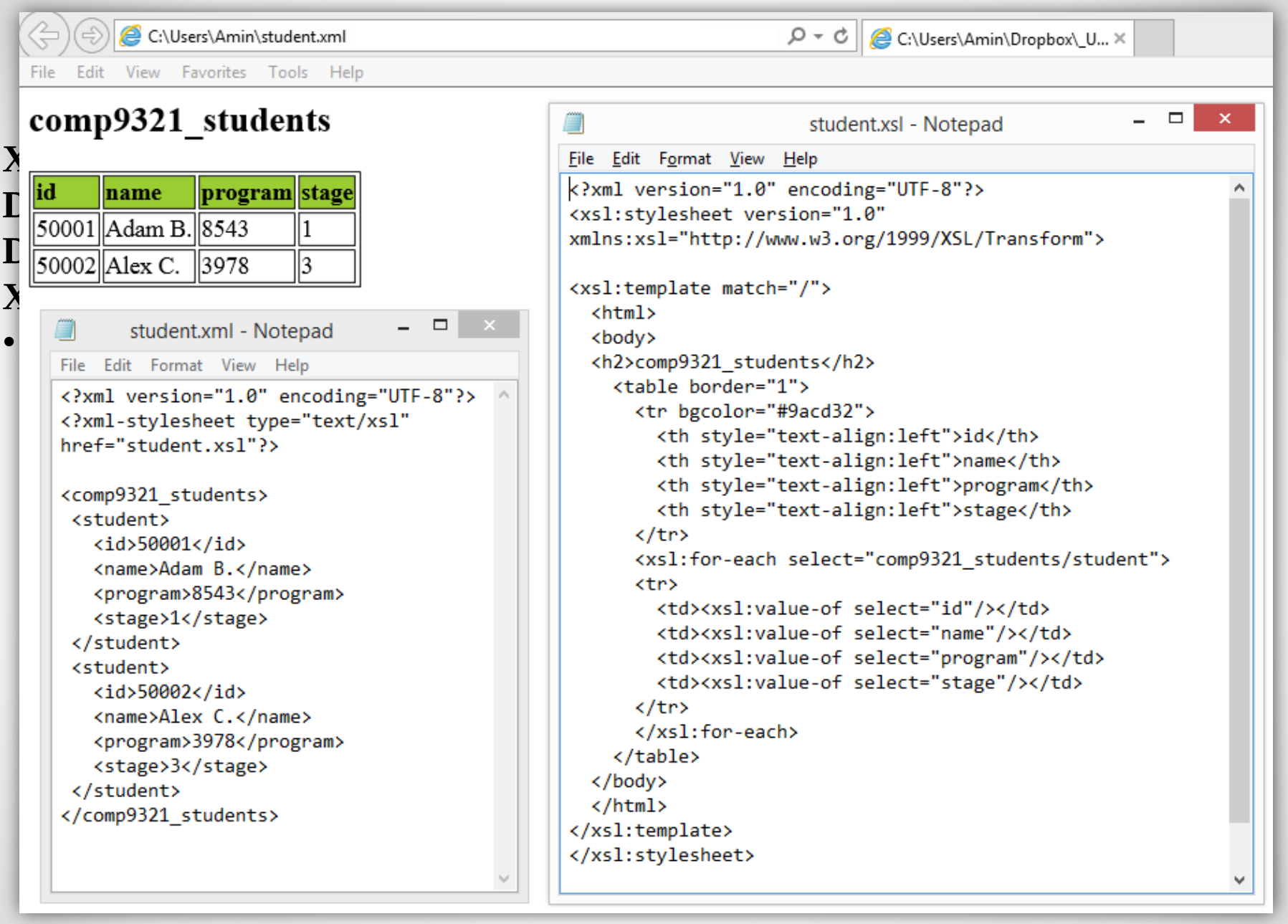
XML: a markup language used to describe information.

DOM: a programming interface for accessing and updating documents.

DTD and XML Schema: describes the structure and content of XML documents.

XSLT:

- XSL stands for eXtensible Stylesheet Language, and is a style sheet language for XML documents.
- CSS = Style Sheets for HTML
- XSL = Style Sheets for XML
  - XSL describes how the XML document should be displayed!
- XSLT (XSL Transformations) a language for transforming XML documents.


C:\Users\Amin\student.xml
C:\Users\Amin\Dropbox_U...
File Edit View Favorites Tools Help
comp9321_students
id name program stage
50001 Adam B. 8543 1
50002 Alex C. 3978 3
student.xml - Notepad
File Edit Format View Help
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl"
href="student.xsl"?>
<comp9321_students>
<student>
<id>50001</id>
<name>Adam B.</name>
<program>8543</program>
<stage>1</stage>
</student>
<student>
<id>50002</id>
<name>Alex C.</name>
<program>3978</program>
<stage>3</stage>
</student>
</comp9321_students>
student.xsl - Notepad
File Edit Format View Help
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<body>
<h2>comp9321_students</h2>
<table border="1">
<tr bgcolor="#9acd32">
<th style="text-align:left">id</th>
<th style="text-align:left">name</th>
<th style="text-align:left">program</th>
<th style="text-align:left">stage</th>
</tr>
<xsl:for-each select="comp9321_students/student">
<tr>
<td><xsl:value-of select="id"/></td>
<td><xsl:value-of select="name"/></td>
<td><xsl:value-of select="program"/></td>
<td><xsl:value-of select="stage"/></td>
</tr>
</xsl:for-each>
</table>
</body>
</html>
</xsl:template>
</xsl:stylesheet>

The XML Family - XPath

XML: a markup language used to describe information.

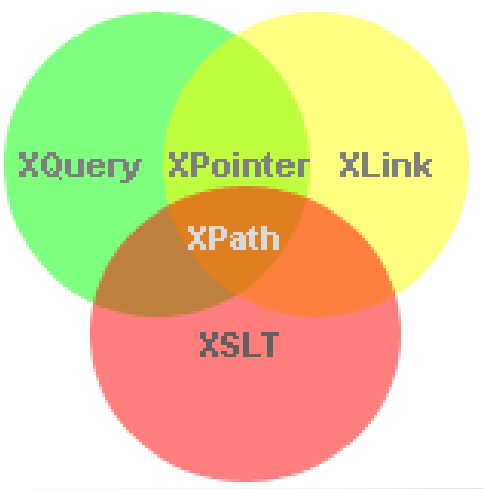
DOM: a programming interface for accessing and updating documents.

DTD and XML Schema: describes the structure and content of XML documents.

XSLT: a language for transforming XML documents

XPath:

- XPath (XML Path language) is a language for finding information in an XML document.
- XPath contains a library of standard functions
- XPath is a major element in XSLT
- XPath is also used in XQuery, XPointer and XLink
- XPath is a W3C recommendation



The XML Family – XPath Examples

```
<?xml version="1.0" ….>
<comp9321_students>
 <student>
  <id>50001</id>
  <name>Adam B.</name>
  <program>8543</program>
  <stage>1</stage>
 </student>
 <student>
  <id>50002</id>
  <name>Alex C.</name>
  <program>3978</program>
  <stage>3</stage>
 </student>
</comp9321_students>
```

some **XPath** expressions:

/comp9321_student/student[1]

- Selects the first "student" element that is the child of the "comp9321_student" element

/comp9321_student/student[last()]

- Selects the last "student" element that is the child of the "comp9321_student" element

/comp9321_student/student[position()<3]

- Selects the first two "student" element that is the child of the "comp9321_student" element

/comp9321_student/student[stage>2]

- Selects all the "student" elements of the "comp9321_student" element that have a "stage" element with a value greater than 2.

…

The XML Family

XML: a markup language used to describe information.

DOM: a programming interface for accessing and updating documents.

DTD and XML Schema: describes the structure and content of XML documents.

XSLT: a language for transforming XML documents

XPath: a query language for navigating XML documents.

XPointer: for identifying fragments of a document.

XLink: generalises the concept of a hypertext link.

XInclude: for merging documents.

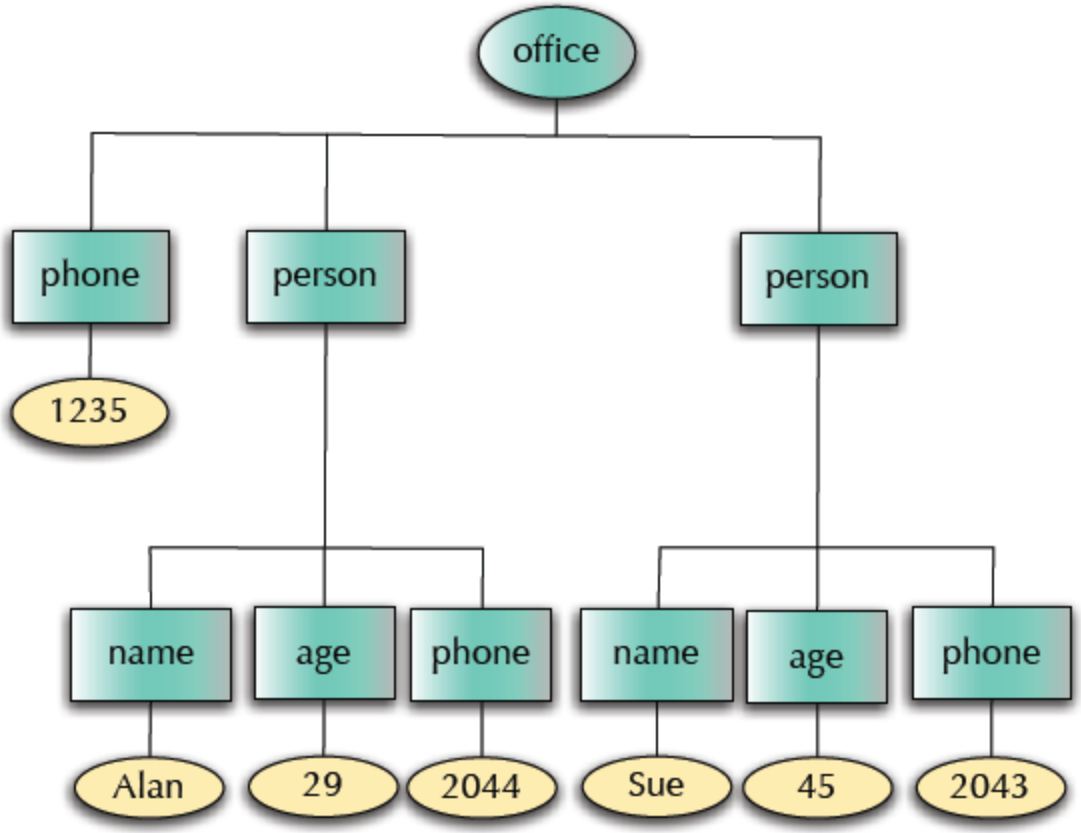
XQuery: a language for making queries across documents.

RDF: a language for describing resources.

An XML document is a tree ...

An XML document is a tree ...

```
<office>
   <phone>1235</phone>
   <person>
      <name>Alan</name>
      <age>29</age>
      <phone>2044</phone>
  </person>
  <person>
      <name>Sue</name>
      <age>45</age>
      <phone>2043</phone>
  </person>
</office>
```



Attributes in XML tags

LName	Title	FName	School	Campus	Room
Edgar	Miss	Pam	Optometry	KG	B501
...	...	...	...	...	...

Phonebook with Attributes

```
<Phonebook>
 <Entry entryNumber="001">
   <Name  Title="Miss">
      <Last>Edgar</Last>
      <First>Pam</First>
   </Name>
   <School Campus="KG">Optometry</School>
   <Room  Building="B" Level="5">01</Room>
 </Entry>
</Phonebook>
```

- Attribute order is not significant

Attributes in XML tags

LName	Title	FName	School	Campus	Room
Edgar	Miss	Pam	Optometry	KG	B501
...	...	...	...	...	...

Phonebook with many attributes ...

```
<Phonebook>
 <Entry entryNumber="001">
   <Name  Title="Miss" LName="Edgar" FName="Pam"/>
   <Location Campus="KG"  School="Optometry" Building="B" Room="501"/>
 </Entry>
</Phonebook>
```

- Avoid using too many (loses structure, more parsing effort ...)

Parsing XML documents with Java

Inside an XML file, there are ...

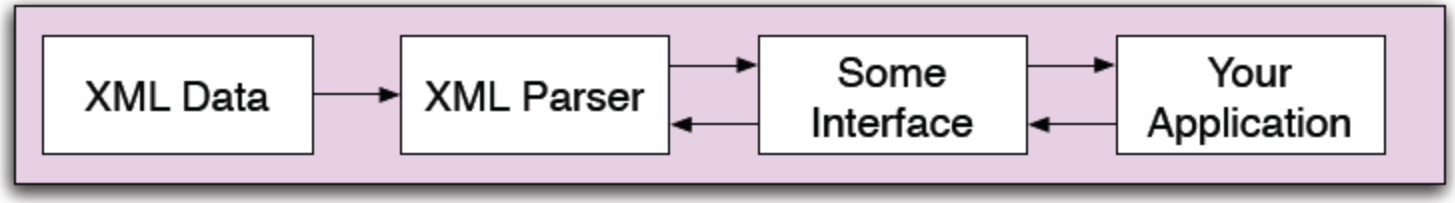
- Markup:
  - Tags, Entity References, Comments Processing Instructions, DTD declarations, XML declaration, and CDATA Section Delimiters
- and Character Data which includes everything else
  - Parsed Character Data (PCDATA): character data left after entity references are replaced with their text
  - e.g. Given `<PUBLISHER>A &amp; M Records</PUBLISHER>`, the parsed character data is `A & M Records`

```
<?xml version="1.0" encoding="UTF-8">
<!DOCTYPE SONG SYSTEM "song.dtd">
<SONG xmlns="http://www.cafeconleche.org/namespace/song">
<TITLE>New Slang</TITLE>
<ARTWORK ALT="Garden State" WIDTH="100" HEIGHT="200"/>
<ARTIST>The Shins</ARTIST>
 <!-- The publisher is actually Polygram but I needed  an example of a general entity reference. -->
  <PUBLISHER>A &amp; M Records</PUBLISHER>
  <LENGTH>3:29</LENGTH>
  <YEAR>2004</YEAR>
</SONG>
```

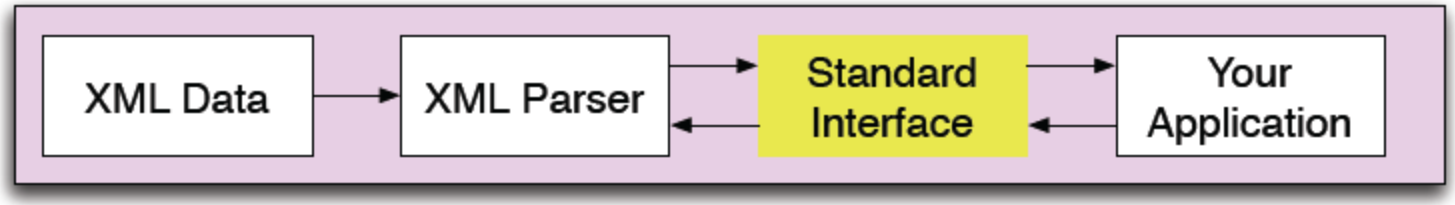
Parsing XML documents with Java

What do you mean by 'parsing (or processing) XML docs'?

- Parsing makes an interface available to your application that needs to make use of the document
- Through the interface, you can modify, retrieve the document contents.



What if the interface provided by the parser is parser-specific? → your application will have to be 'parser-specific'.
Obviously ... we want "STANDARD"!



SAX and DOM as the Standard Interfaces

- SAX - the **S**imple **A**PI for **X**ML
- DOM - the **D**ocument **O**bject **M**odel

Why two standards? → trade-off between control and performance

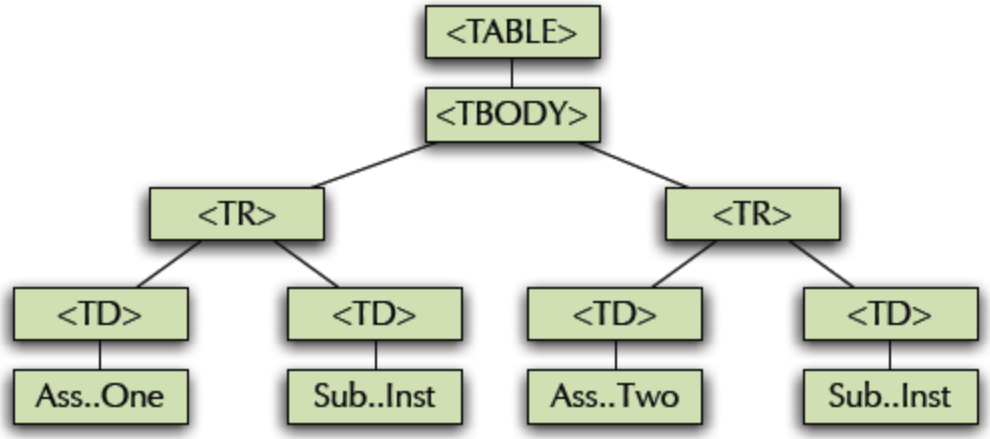
- DOM gives you a tree structure
  - you have a complete control over the structure
  - ie., traverse the tree, modify structure, etc.
  - the tree gets stored in memory all at once
- SAX lays out the document in time, as a sequence of 'events'
  - events are associated with each tag (open/close), each tag body, etc.
  - you will write event handlers (ie., you can ignore certain events)
  - It requires much less memory
  - It becomes difficult to use if processing an element depends on earlier/later elements

Document Object Model (DOM)

- DOM is an API for HTML and XML documents, its specification is developed by W3C (`http://www.w3.org/DOM/DOMTR`)
- It defines the logical structure of documents and the way a document is accessed and manipulated.

TABLE

```
<TABLE>
<TBODY>
<TR>
<TD>Assignment One</TD>
<TD>Submission Instructions</TD>
</TR>
<TR>
<TD>Assignment Two</TD>
<TD>Submission  Instructions</TD>
</TR>
</TBODY>
</TABLE>
```



http://java.sun.com/j2se/1.5.0/docs/api/org/w3c/dom/package-summary.html

Dealing with Nodes in DOM

In DOM, XML Documents are treated as a tree of nodes

Types of Nodes: Twelve different kinds of node are defined by the W3C DOM standard.

- elements
- attributes
- text
- CDATA
- entity reference
- entity
- processing instruction
- comment
- Document
- Document type
- Document fragment
- Notation

An example XML here ...

```
<?xml version="1.0"?>
<bibliography>
<book>
<author>North, Ken</author>
<title>Database magic with Ken North</title>
<address>Upper Saddle River, NJ</address>
<publisher>Prentice Hall</publisher>
<year>1999</year>
<isbn>0136471994</isbn>
</book>
<book>
<author> Elmasri, Ramez, and Shamkant B. Navathe</author>
<title>Fundamentals of database systems</title>
<address>Reading, Mass.</address>
<publisher>Addison-Wesley Pub Co</publisher>
<year>2000 </year>
<edition>3rd</edition>
<isbn>0201542633</isbn>
</book>
<book>
<author>Feiler, Jesse</author>
<title>Database-driven Web sites</title>
<address>San Francisco</address>
<publisher>Morgan Kaufmann</publisher>
<year>1999</year>
<isbn>0122513363</isbn>
</book>
<!-- more book -->
</bibliography>
% Text of an element node is stored in a text node.
```

DOM for XML

- The root node has no parent but every other node has *exactly one parent node*
- A node can have any number of children
- Two nodes that have the same parent are called *siblings*
- A node with no children is called a *leaf node*
- Among siblings, the node that appears first in sequence is called the first child and the node that appears last is the last child
- Important: Text of an element node is stored in a separate text node.

Using a DOM Parser (eg., Apache Xerces)

```
import org.w3c.dom.*;
import org.apache.xerces.parsers.DOMParser;

public class DOMCountNames {
   public static void main(String[] args) {
         try {
    DOMParser parser = new DOMParser();
    parser.parse(args[0]);
    Document doc = parser.getDocument();
    // do something ..
         }
         catch(Exception e){
            e.printStackTrace(System.err);
 }
   }
}
```

The DOMParser Class:

- DOMParser class is derived from the XMLParser class
- parse() method parses the input source given by a system identifier
- Document getDocument() method returns the document itself

Document Interface Methods

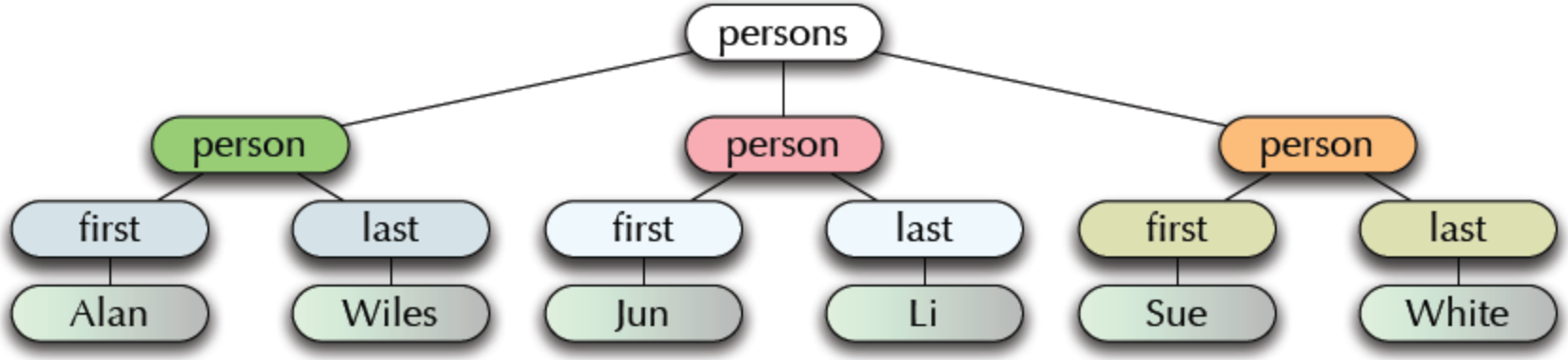
Once you have the **Document** object, you can:

- *Attr* createAttribute(String name): Creates an attribute
- *Element* createElement(String tagName): Creates an element
- *Text* createTextNode(String data): Creates a Text Node
- *Element* getDocumentElement(): Gets the root element of the document
- *Element* getElementById(String elementId): Get the element by ID
- *NodeList* getElementsByTagName(String tagname): Returns a NodeList of all the elements with a given tag name

NodeList Interface Methods:

- int getLength(): Gets the number of nodes in this list
- Node item(int index): Gets the item at the specified index value in the collection

Examples of Node Properties (XML), p.9.25



document.getElementsByTagName("person")[1]	person
document.getElementsByTagName("person")[1].parentNode	persons
document.getElementsByTagName("person")[1].childNodes	first last
document.getElementsByTagName("person")[1].firstChild	first
document.getElementsByTagName("person")[1].lastChild	last
document.getElementsByTagName("person")[1].previousSibling	person
document.getElementsByTagName("person")[1].lastChild.firstChild	Li

Count/Print the number of 'book' elements

```
import org.w3c.dom.*;
import org.apache.xerces.parsers.DOMParser;
public class DOMCountNames   {
   public static void main(String[] args) {
      try{
 DOMParser parser = new DOMParser();
 parser.parse(args[0]);
 Document doc = parser.getDocument();
 NodeList nodelist = doc.getElementsByTagName("book");
 System.out.println(args[0] + " has " + nodelist.getLength()
                    + " <book> elements.");
}
      catch(Exception e){
 e.printStackTrace(System.err);
}
   }
}
```

Compiling and Running

```
% javac -classpath ".:xerces.jar" DOMCountNames.java
% java -classpath ".:xerces.jar" DOMCountNames books.xml
```

Dealing with Nodes in DOM

There is a large range of methods that can be applied to the nodes:

Node Interface Methods

getNodeName()	getNodeValue()	setNodeValue(...)
getNodeType()	getParentNode()	getChildNodes()
getFirstChild()	getLastChild()	getPreviousSibling()
getNextSibling()	getAttributes()	getOwnerDocument()
insertBefore(...)	replaceChild(...)	removeChild(...)
appendChild(...)	hasChildNodes()	cloneNode(...)
normalize()	isSupported(...)	getNamespaceURI()
hasAttributes()	getLocalName()	

Node interface documentationhttps://www.w3.org/2003/01/dom2-javadoc/org/w3c/dom/Node.html

Dealing with Nodes

Consider the following program:

```
Document doc = parser.getDocument();
Element docRoot = doc.getDocumentElement();
String docRootName = docRoot.getTagName();
System.out.println("Doc root: "+docRootName);
int i =   0;
for (Node node = docRoot.getFirstChild();
     node != null;
     node = node.getNextSibling()) {
  if (node.getNodeType()==Node.ELEMENT_NODE) {
    System.out.println(i+": " + node.getNodeType()
                              + node.getNodeName());
  }
  else {
    System.out.println(i+": " + node.getNodeType());
  }
  i++;
}
```

The method getNodeType() returns a number in the range 1 to 12. Thus we can tell which kind of node we are dealing with. What will the output of this program be?

The Element interface

This interface outlines operations that are specific to elements:

- `getTagName()`: This method returns the name of the tag associated with the element.
- `getAttribute(name)`: This method returns a string containing the value of an attribute:
  - `name` is the name of the attribute.

Element Interface documentation – full list of methods and operations

https://www.w3.org/2003/01/dom2-javadoc/org/w3c/dom/Element.html

More with DOM ...

Heaps of hands on tutorials on the web ...

- DOM and Javascript: e.g.,
  `http://www.sitepoint.com/print/xml-javascript-mozilla`
  `https://developer.mozilla.org/en/The_DOM_and_JavaScript`
- DOM, XML, Javascript and Ajax: e.g.,
  `http://www.w3schools.com/Ajax/ajax_intro.asp`

References

- http://www.w3.org/XML/
- XML in a nutshell, Chapters 9 and 10
- http://www.ibm.com/developerworks/library/xml-schema/
- http://www.w3schools.com/xml/
- Some examples in these notes are originated from Dr. David Edmond from QUT, Brisbane