XML vs. HTML

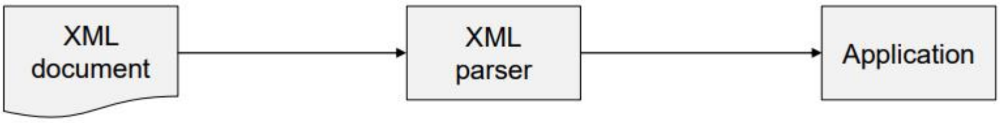
Superficially, the markup in XML looks like the markup in HTML

… but there are some crucial differences

XML	HTML
Structural and semantic language	Presentation language
No fixed set of tags that are supposed to work in every domain	Fixed set of tags with predefined semantics
Extensible - can be extended to meet different needs	Not extensible - it does web pages, but nothing else

How XML Works

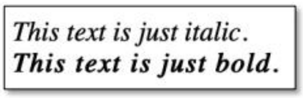
- Strict rules regarding the syntax of XML documents - allows for the development of XML parsers that can read documents
- Applications that need to understand an XML document will use a parser



Splits the document into individual pieces

Problems with loose syntax:

```
<html>
<body>
<i>This text is just italic.</italic>
<br>
<b>This text is just bold.</b>
</body>
</html>
```



This text is just italic.
This text is just bold.

Elements and Tags

- The content can be
  - Empty - an empty element is abbreviated as

    <element-name/>

    - e.g. <homework></homework> abbreviated as <homework/>
  - Simple content - consists of text
  - Element content - consists of one or more elements
  - Mixed content - consists of text and elements

ATTENTION: XML is case sensitive - <course> and <COURSE> are different

XML Names

- But, what can be used as XML names?
- XML names are:
  - Element names
  - Attribute names
  - Names for other constructs (later)

XML Names

- May contain only:
  - Alphanumeric characters (A-Z, a-z, 0-9)
  - Numbers
  - Underscore (_), hyphen (-), period (.)
  - Colon (:), but has special meaning for *namespaces* (discussed later), so be carefuly!
  - Non-English letters (δ, ü, ß, ж, etc.) and many other Unicode symbols are allowed (check the specification if interested)

- There is no limit to the length of an XML name

XML Names

Further restrictions:

- Names beginning with “XML” (in any combination of case) are forbidden
- XML names may only start with letters and underscore
  (and many other Unicode characters, check the specification if interested)

<course> ... </course>	<xml_course> ... </ xml_course >
<first_name> ... </first_name>	<first name> ... </first name>
<_1st-class> ... </_1st-class>	<1st-class> ... </1st-class>
✓	✗

Entity References

- XML predefines five entity references:

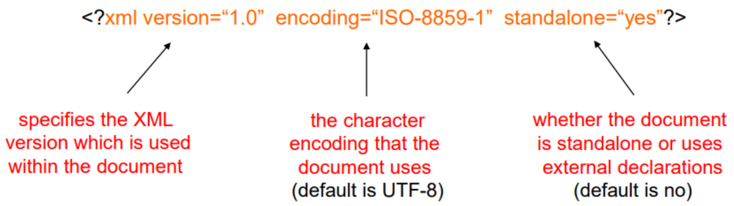
mandatory	&lt; for <	
	&amp; for &	
optional	&gt; for >	for symmetry with &lt;
	&quot; for “	needed in attribute values
	&apos; for ‘	

- Additional references can be defined in the document type definition (later)

ATTENTION: Entity references cannot be used in XML names

XML Declaration

- XML should begin with an XML declaration that declares the version used:



<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>

specifies the XML version which is used within the document

the character encoding that the document uses (default is UTF-8)

whether the document is standalone or uses external declarations (default is no)

- The XML declaration is optional.
- If given, the XML declaration must be the first thing in the document

ATTENTION: XML declaration is not an element or processing instruction

Well-formed XML Documents

- Every XML document must be well-formed - no exception
- It must adhere to some rules including:
  - Every start-tag has a matching end-tag
  - Elements may nest but not overlap
  - Exactly one document/root element
  - Attribute values are quoted
  - Attribute names in an element are unique
  - Comments and processing instruction not inside tags
  - No < or & inside the data character of an element or attribute
  - …

ATTENTION: Before publishing an XML document, check it for well-formedness

The Need for Namespaces

Namespaces have two purposes in XML:

- Disambiguating elements and attributes

  Distinguish between elements and attributes from different vocabularies that share the same name but are semantically different

- Grouping elements

  Group related elements and attributes together so that programs can easily recognize them

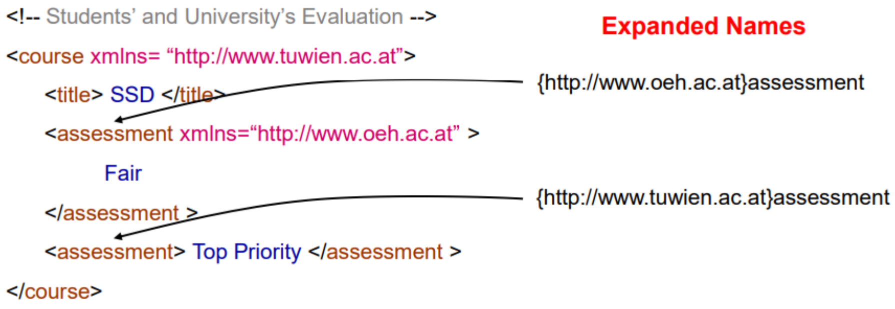
Default Namespace

- We can have a default namespace declared as xmlns=“name”
- We simply remove the prefix

```
<!-- Students’ and University’s Evaluation -->
<course
          xmlns=“http://www.oeh.ac.at”
          xmlns:univ= “http://www.tuwien.ac.at”>
    <title> SSD </title>
    <assessment> Fair </assessment >
    <univ:assessment> Top Priority </univ:assessment >
</course>
```

ATTENTION: Default namespace applies only to unprefixed elements, not attributes

Multiple Namespaces



```
<!-- Students’ and University’s Evaluation -->
<course xmlns= “http://www.tuwien.ac.at”>
    <title> SSD </title>
    <assessment xmlns=“http://www.oeh.ac.at” >
         Fair
    </assessment >
    <assessment> Top Priority </assessment >
</course>
```

Expanded Names
{http://www.oeh.ac.at}assessment
{http://www.tuwien.ac.at}assessment

- The closest ancestor with a namespace declaration takes precedence
- If there is no declaration among the ancestors:
  - For the default namespace the empty namespace is used
  - For a prefix we get an error (when the prefix is used)

DTDs at First Glance

- Schema - the markup permitted

- Many different XML schema languages available:
  - Document Type Definitions (DTDs)
  - W3C XML Schema
  - REgular LAnguage for XML Next Generation (RELAX NG)
  - Schematron
  - …

- In the context of this course we are going to see DTDs and W3C XML Schema

…but for the moment let us focus on DTDs

Validation

- Validating parsers - check both for well-formedness and validity

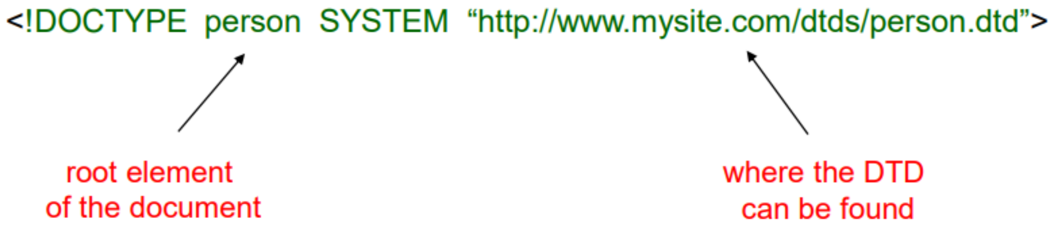
- Validating errors may be ignored (unlike well-formedness errors)

- Whether a validity error is serious depends on the application

ATTENTION: Validity errors are not necessarily fatal

Document Type Declaration

- a URL in an XML document indicating where its DTD can be found
- this is done via the document type declaration - after the XML declaration



```
<!DOCTYPE  person  SYSTEM  “http://www.mysite.com/dtds/person.dtd”>
```

root element
of the document

where the DTD
can be found

Document Type Declaration

- Relative URL - if the document and the DTD reside in the same base site

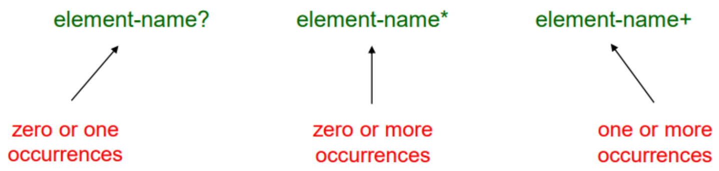
```
<!DOCTYPE  person  SYSTEM  “/dtds/person.dtd”>
```

- Just the file name - if the document and the DTD are in the same directory

```
<!DOCTYPE  person  SYSTEM  “person.dtd”>
```

Element Declarations: Number of Children

- Occurrence indicators (?,*,+)



element-name?

zero or one occurrences

element-name*

zero or more occurrences

element-name+

one or more occurrences

ATTENTION: DTDs cannot specify the exact number of occurrences, or say at most k or at least k occurrences

Element Declarations: Parentheses

- Individually the constructs of #PCDATA, sequences, ?, *, + and choices are rather limited

- E.g., we cannot say a name element may contain:
  - Just a first name,
  - Just a last name, or
  - A first and a last name with an arbitrary number of middle names

- Combine the above features in an arbitrary way - (nested) parentheses

Element Declarations: Empty Content

- Empty elements, i.e., without a content, are declared as

```
<!ELEMENT element-name EMPTY>
```

Valid: <element-name></element-name> or
<element-name/>

Invalid: <element-name> </element-name>

Element Declarations: Any Content

- We can say that an element simply exists, without any restrictions

```
<!ELEMENT  my-element-name  ANY>
```

- It is useful during the designing phase of a DTD
- In general, it is a bad design to use ANY in finished DTDs

ATTENTION: ANY does not allow undeclared child elements

Attribute Declarations: Attribute Types

- Up to now, attribute values can be any string of text
- … except the symbols <, ", ’, and & that need to be espaced using entity references
- DTDs can make stronger statements about the attribute values - attribute type
- There are ten attribute types in XML:
    - CDATA
    - NMTOKEN
    - NMTOKENS
    - Enumeration
    - ID
    - IDREF
    - IDREFS
    - ENTITY
    - ENTITIES
    - NOTATION

details follow

check out the textbook (XML in a Nutshell, Chapter 3)
or XML recommendation

Attribute Types: NMTOKEN

- XML name token - legal XML name, but can start with any allowed character
- Recall that XML names can start only with a letter or underscore
- NMTOKEN - an attribute can take XML name tokens

```
<!ATTLIST course date NMTOKEN #REQUIRED>
<!ELEMENT course (#PCDATA)>
```

Valid: `<course date="05-03-2025"> SSD </course>`

Invalid: `<course date="05/03/2025"> SSD </course>`

Attribute Types: Enumeration

- List of possible values (separated by |)

```
<!ATTLIST course day (Monday | Thursday) #REQUIRED>
<!ELEMENT course (#PCDATA)>
```

Valid: `<course day="Thursday"> SSD </course>`

Invalid: `<course day="Sunday"> SSD </course>`

ATTENTION: The only attribute type that is not an XML keyword

Attribute Types: ID

- An attribute must contain an XML name (not name token) that is unique
- Each element has at most one ID attribute - ID of an element

```
<!ATTLIST person id_number ID #REQUIRED>
<!ELEMENT person (#PCDATA)>
```

Invalid:

```
<person id_number="123456"> Tim Bray</person>
```

Valid:

```
<person id_number="_123456"> Tim Bray</person>
```

Attribute Types: IDREF

- An attribute must contain the value of some ID type attribute in the document

```
<!ATTLIST employee emp_id ID #REQUIRED>
<!ATTLIST project proj_id ID #REQUIRED>
<!ATTLIST manager mgr_id IDREF #REQUIRED>
<!ELEMENT employee (#PCDATA)>
<!ELEMENT project (#PCDATA)>
<!ELEMENT manager (#PCDATA)>
```

Valid:

```
<employee emp_id="e1"> E </employee>
      <project proj_id="p1"> P </project>
   <manager mgr_id="e1"> E </manager>
```

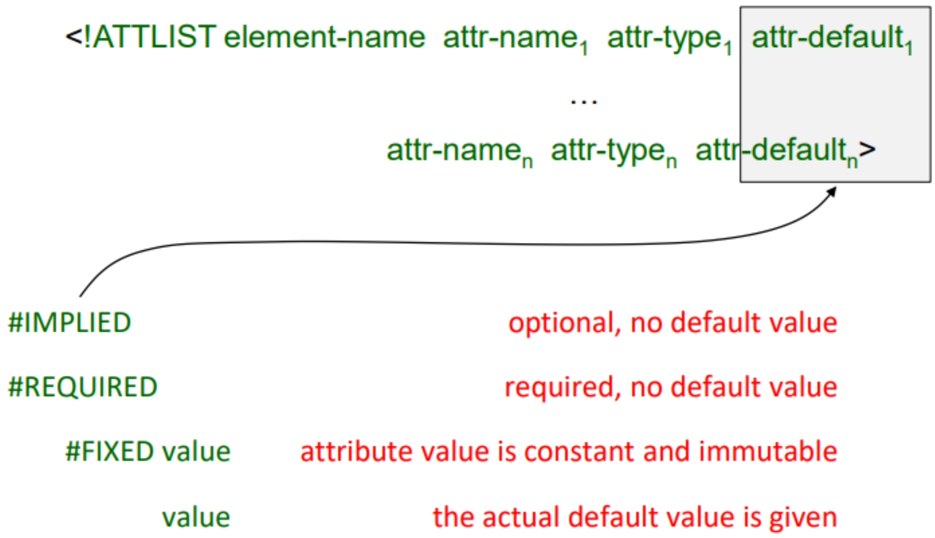
Other Attribute Types

- IDREFS - list of IDs occurring in the document
  - if you understand NMTOKENS, you understand IDREFS
- ENTITY – unparsed entity declared in the DTD
- ENTITIES - list of unparsed entities declared in the DTD
- NOTATION - name of a notation declared in the DTD

… for more details, check out the textbook (XML in a Nutshell, Chapter 3)

Attribute Declarations: Attribute Defaults

- Recall how an attribute declaration looks like



<!ATTLIST element-name attr-name$_1$ attr-type$_1$ attr-default$_1$

…

attr-name$_n$ attr-type$_n$ attr-default$_n$>

#IMPLIED	optional, no default value
#REQUIRED	required, no default value
#FIXED value	attribute value is constant and immutable
value	the actual default value is given

Limitations of DTDs

- Not in XML syntax
    - Different parsers for the document and the DTD
- A weak specification language
    - No control on the exact number of child elements
    - Limited selection of data types
    - The notion of inheritance does not exist
- No explicit support of namespaces
    - The validator is completely unaware of the existence of namespaces

… W3C XML Schema

Validation

- Validating parsers - check both for well-formedness and validity
- Validating errors may be ignored (unlike well-formedness errors)

Simple Elements

- Contain only text - no other elements or attributes
- "Only text" is a bit misleading - several different data types
    - Built-in types (e.g., boolean, string, integer, etc.)
- Facets - we can add restrictions to a data type
    - Limit its content (e.g., min/max value)
    - Match a certain pattern (e.g., €ddd.dd)

Defining Simple Elements – xsd:boolean

- An element of type xsd:boolean represents a logical Boolean that can be either true or false.
- There are four legal values
    - 0, 1, true, false
    - “0” is the same as “false”
    - “1” is the same as “true”

```
<xsd:element  name=“pass”  type=“xsd:boolean”/>
```

<pass> true </pass>	✓		
<pass> false </pass>	✓	<pass> -1 </pass>	✗
<pass> 0 </pass>	✓	<pass> </pass>	✗
<pass> 1 </pass>	✓		

Defining Simple Elements – xsd:integer

- An element of type xsd:integer represents an integer of arbitrary size.
- An element of type xsd:int represents a 4-byte integer, i.e., an integer between - 2147483648 and 2147483647.

- An element of type xsd:positiveInteger represents an integer larger or equal to 1
- An element of type xsd:nonNegativeInteger represents an integer larger or equal to 0
- An element of type xsd:negativeInteger represents an integer smaller than 0
- An element of type xsd:nonPositiveInteger represents an integer smaller or equal to 0

Default and Fixed Values for Simple Elements

- Default value - assigned to the element when no other value is specified

```
<xsd:element  name=“element-name”  type=“element-type”  default=“default-value”/>
```

- Fixed value - assigned to the element, and no other value can be specified

```
<xsd:element  name=“element-name”  type=“element-type”  fixed=“fixed-value”/>
```

Attributes

- Simple elements cannot have attributes
- If an element has attributes, then it is of complex type (later)
- But the attribute itself is always of simple type

Default and Fixed Values for Attributes

- Default value - assigned to the attribute when no other value is specified

```
<xsd:attribute  name=“attribute-name”  type=“attribute-type”  default=“default-value”/>
```

- Fixed value - assigned to the attribute, and no other value can be specified

```
<xsd:attribute  name=“attribute-name”  type=“attribute-type”  fixed=“fixed-value”/>
```

Restrictions on Values

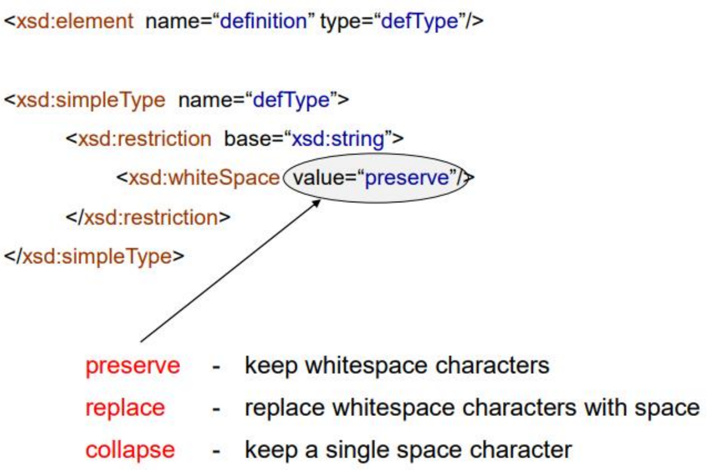
- minInclusive - greater than or equal
- maxInclusive - less than or equal
- minExclusive - greater than
- maxExclusive - less than

Example Regular Expressions

- “[A-Z][A-Z][A-Z]” - triples of uppercase letters from A to Z
- “[a-zA-Z][a-zA-Z][a-zA-Z]” - triples of lowercase/uppercase letters from A to Z
- “[abcd]” - one of the letters a, b, c or d
- “([a-z])*” - zero or more occurrences of lowercase letters from a to z
- “([a-z][A-Z])+” - one or more occurrences of pairs of letters (e.g., sToP, mOrE)
- “day | night” - either day or night
- “[a-zA-Z0-9]{5}” - exactly 5 characters of letters or numbers from 0 to 9

Restrictions on Whitespace Characters

- whiteSpace - specifies how whitespace characters (line feeds, tabs, spaces, and carriage returns) are handled



```
<xsd:element  name=“definition” type=“defType”/>

<xsd:simpleType  name=“defType”>
     <xsd:restriction  base=“xsd:string”>
          <xsd:whiteSpace value=“preserve”/>
     </xsd:restriction>
</xsd:simpleType>
```

preserve - keep whitespace characters

replace - replace whitespace characters with space

collapse - keep a single space character

Restrictions for Datatypes - Sum Up

Constraint	Description
minInclusive	Greater or equal than
maxInclusive	Less or equal than
minExclusive	Greater than
maxExclusive	Less than
enumeration	Set of acceptable values
pattern	Certain sequence of characters
whiteSpace	Specifies how whitespace characters are handled
length	Exact number of characters
minLength	Minimum number of characters
maxLength	Maximum number of characters

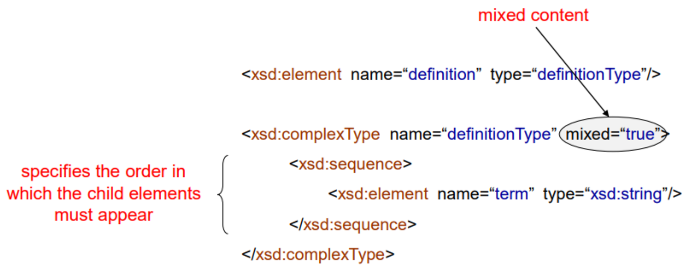
Complex Elements

- Contain other elements and/or attributes

- Four kinds of complex elements
  - Empty elements
  - Elements that contain only other elements (elements only)
  - Elements that contain only text (text only)
  - Elements that contain both elements and text (mixed)

Defining Complex “Mixed-content” Elements

```
<definition>
      The term <term> Semi-structured Data </term>
      refers to a form of structured data that does not
      conform with the formal structure of relational data
</definition>
```



mixed content

```
<xsd:element  name="definition"  type="definitionType"/>

<xsd:complexType  name="definitionType" mixed="true">
      <xsd:sequence>
            <xsd:element  name="term"  type="xsd:string"/>
      </xsd:sequence>
</xsd:complexType>
```

specifies the order in which the child elements must appear

Indicators

- Order indicators - to define the order of the elements

- Occurrence indicators - to define how often an element can occur

- Group indicators - to define related sets of elements
  - Check out the textbook (XML in a Nutshell, Chapter 17)

Occurrence Indicators

- minOccurs - the minimum number of times an element can occur

- maxOccurs - the maximum number of times an element can occur

xsd:key vs. xsd:unique

- xsd:key: The field exists in all selected elements and its value is unique
- xsd:unique: If the field exist for a selected element the its value is unique

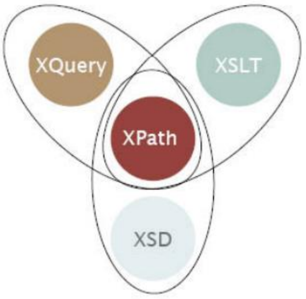
Assume not all employees have ids, but all managers have.

```
<xsd:element  name=“company”  type=“companyType”>
    <xsd:unique  name=“empKey”>
        <xsd:selector  xpath=“employees/employee”/>
        <xsd:field  xpath=“@emp_id”/>
    </xsd:unique>
    <xsd:keyref  name=“empRef”  refer=“empKey”>
        <xsd:selector  xpath=“managers/manager”/>
        <xsd:field  xpath=“@mgr_id”/>
    </xsd:keyref>
</xsd:element>
<xsd:complexType  name=“companyType”>
    …
</xsd:complexType>
```

What is XPath?

- A language for extracting parts of an XML document

- A basic query language for XML - plays the same role as the SQL SELECT statement plays for relational databases



Source: saxonica.com

- An important component of other XML-related technologies (such as XSD, XQuery and XSLT)

- As expected, XPath is a W3C standard

XPath Terminology

- XML documents are treated as trees of nodes

- There are seven kinds of nodes:
  - Document nodes
  - Element nodes
  - Attribute nodes
  - Text nodes
  - Namespace nodes
  - Processing-instruction nodes
  - Comment nodes

Relationships Among Nodes

- The terms parent, child, sibling, ancestor and descendant are describing the relationships among nodes

- In an XML tree:
  - Every node has exactly one parent (except the root)
  - A node can have an unbounded number of children
  - A leaf node has no children
  - Siblings have the same parent

Axes

- XPath defines 13 axes:
  - ancestor
  - ancestor-or-self
  - attribute
  - child
  - descendant
  - descendant-or-self
  - following
  - following-sibling
  - namespace
  - parent
  - preceding
  - preceding-sibling
  - self

Ancestor

- Selects all the nodes that are ancestors of the context node
- The first node on the axis is the parent of the context node, the second is its grandparent, and so on
- The last node on the axis is the root of the tree

Ancestor-or-self

- Selects the same nodes as the ancestor axis
- … but starting with the context node (instead of the parent of the context node)

Attribute

- If the context node is an element node, then this axis selects all its attribute nodes; otherwise, it selects nothing (empty sequence)
- The attributes will not necessarily be in the order in which they appear in the document
- Namespace nodes are not selected

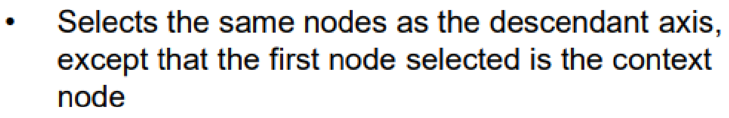
Child

- Selects all the children of the context node in document order
- If the context node is other than a document or element node, then this axis selects nothing
- The children of an element node do not include attribute or namespaces

Descendant

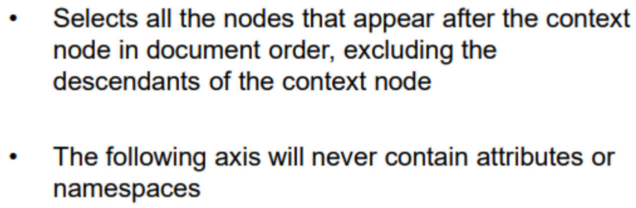
- Selects all the children of the context node, and their children, and so on recursively in document order

Descendant-or-self



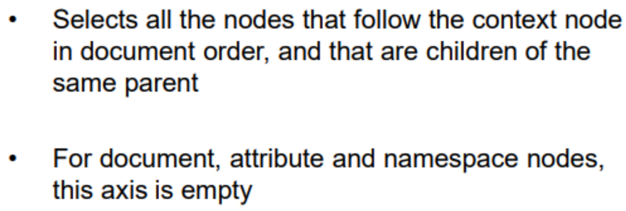
- Selects the same nodes as the descendant axis, except that the first node selected is the context node

Following



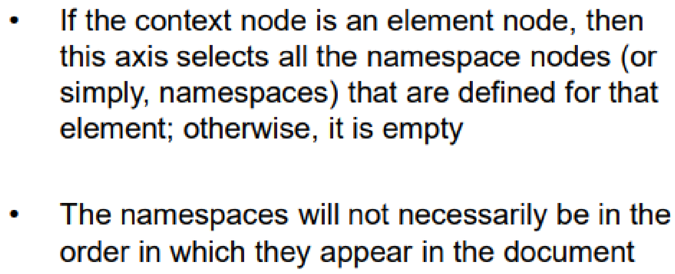
- Selects all the nodes that appear after the context node in document order, excluding the descendants of the context node
- The following axis will never contain attributes or namespaces

Following-sibling



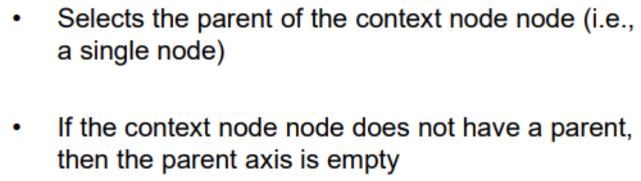
- Selects all the nodes that follow the context node in document order, and that are children of the same parent
- For document, attribute and namespace nodes, this axis is empty

Namespace



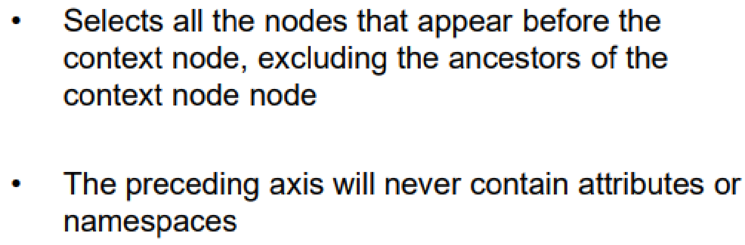
- If the context node is an element node, then this axis selects all the namespace nodes (or simply, namespaces) that are defined for that element; otherwise, it is empty
- The namespaces will not necessarily be in the order in which they appear in the document

Parent



- Selects the parent of the context node node (i.e., a single node)
- If the context node node does not have a parent, then the parent axis is empty

Precending



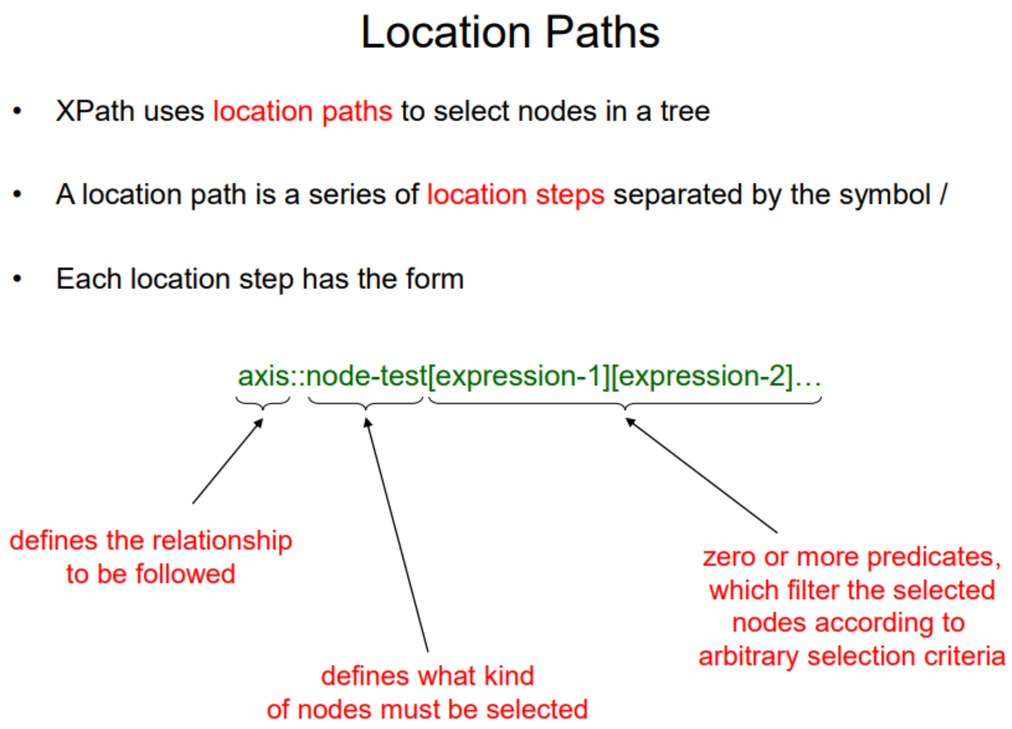
- Selects all the nodes that appear before the context node, excluding the ancestors of the context node node
- The preceding axis will never contain attributes or namespaces

Precending-sibling

- Selects all the nodes that precede the context node, and that are children of the same parent
- For document, attribute and namespace nodes, this axis is empty

Self

- Selects the context node node
- This axis is always non-empty
- Usually, this axis is used in a node-test in order to test whether the current node pass that node-test



Location Paths

- XPath uses location paths to select nodes in a tree
- A location path is a series of location steps separated by the symbol /
- Each location step has the form

axis::node-test[expression-1][expression-2]…

defines the relationship to be followed

defines what kind of nodes must be selected

zero or more predicates, which filter the selected nodes according to arbitrary selection criteria

Node Test

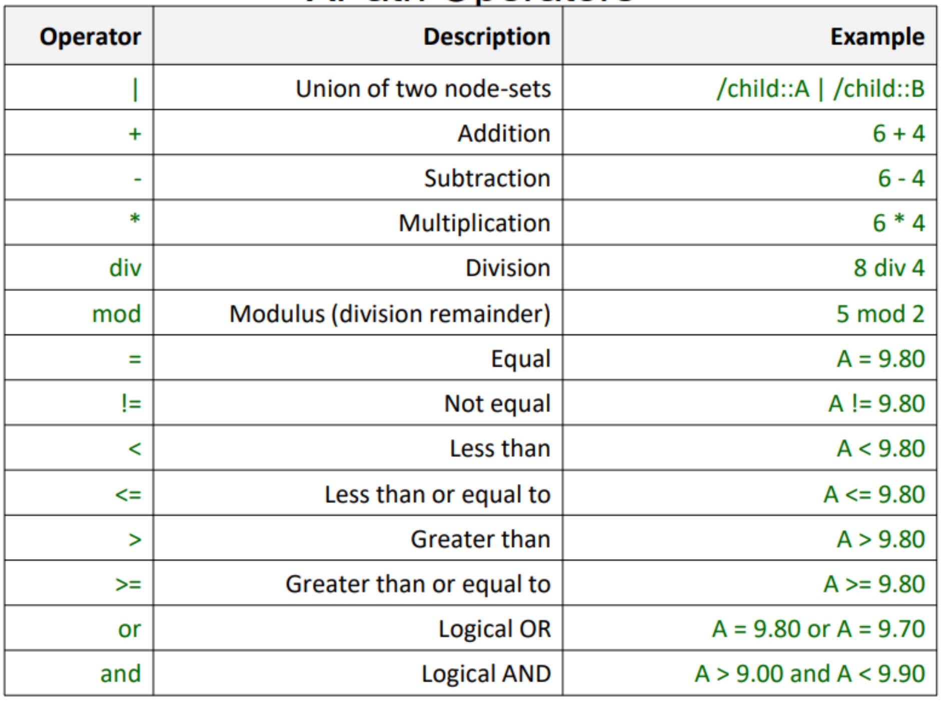
node()	selects all nodes
text()	selects only text nodes
name	selects only elements nodes with tag “name” (child::*name*) …but, if it is used with the attribute axis (attribute::*name*), then it selects the “*name*” attribute nodes …and if it is used with the namespace axis (namespace::*name*), then is selects the namespace nodes with prefix “*name*”
*	selects all element nodes (child::*) …but, if it is used with the attribute axis (attribute::*), then it selects all the attribute nodes …and if it is used with the namespace axis (namespace::*), then it selects all the namespace nodes

General XPath Expressions

- Location Paths are central subset of XPath and return node-sets
- General Xpath expressions can also return numbers, Booleans and strings

- Data-Types:
  - Numbers
  - Strings
  - Booleans
  - Node-Sets

XPath Operators



Operator	Description	Example
|	Union of two node-sets	/child::A | /child::B
+	Addition	6 + 4
-	Subtraction	6 - 4
*	Multiplication	6 * 4
div	Division	8 div 4
mod	Modulus (division remainder)	5 mod 2
=	Equal	A = 9.80
!=	Not equal	A != 9.80
<	Less than	A < 9.80
<=	Less than or equal to	A <= 9.80
>	Greater than	A > 9.80
>=	Greater than or equal to	A >= 9.80
or	Logical OR	A = 9.80 or A = 9.70
and	Logical AND	A > 9.00 and A < 9.90

XPath Functions

- Node-Set Functions

  count(/descendant-or-self::node()/course)

- String Functions

  starts-with(“Richard”,”Ric”)

- Boolean Functions

  not(attribute::age!=42)

- Number Functions

  floor(attribute::temperature)

Abbreviated Syntax

- The most commonly used location steps can be in an abbreviated syntax
- Simplify XPath expressions

/descendant-or-self::node()/	//
self::node()	.
parent::node()	..
child::	
attribute::	@
position() = n	n

XPath in XSD

- XSD uses XPath expressions in:
  - key elements
  - keyref elements
  - unique elements
- but only a subset of XPath is supported

Restricted XPath in xsd:selector

- Only the child axis, and the descendant-or-self axis at the begin of a path, are allowed (with special syntax)
- Selects Elements (not attributes)
- Legal:

`<xsd:selector xpath=“employees/employee”/>` ✓

`<xsd:selector xpath=“employees/*”/>` ✓

`<xsd:selector xpath=“.//employee”/>` ✓

`<xsd:selector xpath=“employees/employee | .//manager”/>` ✓

Restricted XPath in xsd:selector

- Only the child axis, and the descendant-or-self axis at the begin of a path, are allowed (with special syntax)
- Selects Elements (not attributes)
- Not legal:

`<xsd:selector xpath=“descendant::employee”/>` ✗

`<xsd:selector xpath=“employees//employee”/>` ✗

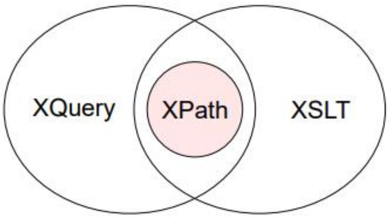
`<xsd:selector xpath=“parent::employee”/>` ✗

`<xsd:selector xpath=“employees/child::nodes()”/>` ✗

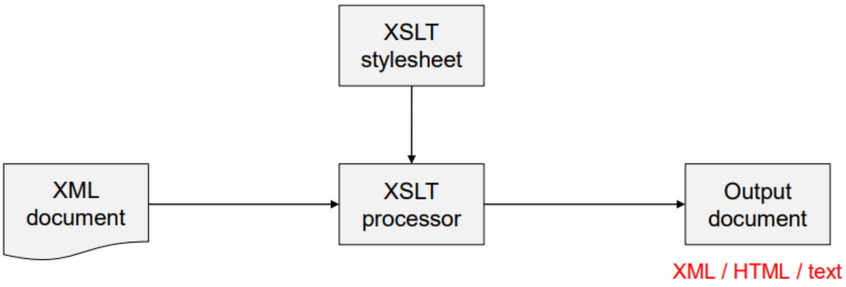
`<xsd:selector xpath=“employees/@employee”/>` ✗

What is XSLT?

- XSL = Extensible Stylesheet Language Family
- XSL = tools for styling XML documents (as CSS for HTML)
- XSLT = XSL Transformations
- XSLT is used to transform a source XML document into a target XML/HTML/text document
- XSLT uses XPath for navigation
- XSLT is a W3C recommendation



How XSLT Works?



- Define a transformation with an XSLT document (which is an XML document)
- Apply this transformation on an input document using an XSLT processor

Default Templates

- XSLT defines default templates that are always present

- Default templates are as follows
    - For root and elements: apply templates for child elements
    - For text elements: copy content to the output
    - For attributes: copy value to the output

- To override the behaviour of a default template create a template for an element

```
<xsl:template match=" * | / ">
  <xsl:apply-templates select="*"/>
</xsl:template>
```

```
<xsl:template match=" text( ) | @* ">
  <xsl:value-of select="."/>
</xsl:template>
```

Priorities

- Exactly one template is executed

- In case of more than one templates, a priority value decides which template is executed

- The XPath expression in the match attribute indicates the priority

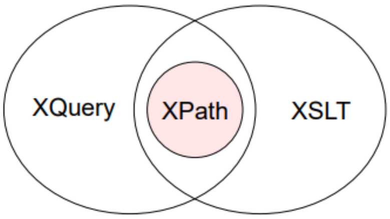
- More specific XPath expressions have higher priority

What is XQuery?

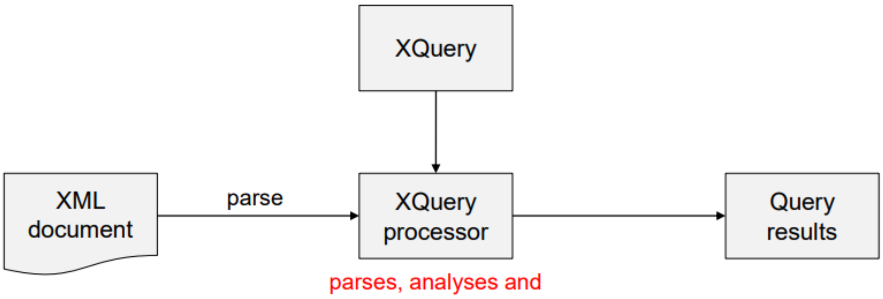
- XQuery is a language for querying XML data
- XQuery for XML is like SQL for relational databases
- XQuery is built on XPath expressions
- As expected, XQuery is a W3C recommendation

XQuery vs. XPath

- XPath is essentially a subset of XQuery
- XQuery has a number of features not supported by XPath
- XQuery can structure or sort query results (not just select elements and attributes)



Processing XQueries



XQuery

XML document

parse

XQuery processor

Query results

parses, analyses and evaluates the query

- Analysis phase: finds syntax errors and other static errors that do not depend on the input document

- Evaluation phase: may raise dynamic errors (e.g., missing input document or division by zero)

- A number of implementations available - http://www.w3.org/XML/Query

FLWOR Expressions

- The main engine of XQuery is FLWOR expressions

for …
let …
where …
order by …
return …

- Pronounced “Flower Expressions”

- Generalize Select-From-Having-Where in SQL

FLWOR Expressions: General Rules

- for and let may be used many times in any order

- Only one where is allowed

- More than one sorting criteria can be specified

  order by <expression> ascending, <expression> descending, …

Element Constructors

- An XQuery expression may construct a new XML element

- XML constructs can be used to create elements and attributes that appear in the query result
  - Wrapping results in a new element
  - Adding attributes to results

- A key difference compared to XPath

List Expressions

- XQuery expressions manipulate lists of items
  - Value lists: (1,2,3), ("a", "b")
  - Results of XPath expressions

- Many operators are supported
  - Range expressions (e.g., ”3 *to* 10”)
  - Concatenation using “**,**”
  - Set operators (union, intersect, except)

- Many functions are supported
  - count, avg, max, min, sum, distinct-values, …

Conditional Expressions

XQuery supports general if-then-else expressions

```
for $b in doc("books.xml")/bookstore/book
return
    if ($b/@category = "children")
    then <child> {$b} </child>
    else <adult> {$b} </adult>
```

ATTENTION: else is required, but it can be just else ()

Joins

```
for $i in doc("order.xml")//item
let $n := doc("catalog.xml")//product[number = $i/@num]/name
return
    <item  num = "{$i/@num}"
       name = "{$n}"
       quantity = "{$i/@quantity}"/>
```

```
<catalog>
    <product dept=“D1">
        <number> 130 </number>
        <name> N1 </name>
    </product>
    <product dept=“D2">
        <number> 230 </number>
        <name> N2 </name>
    </product>
</catalog>
```

```
<order>
    <item dept=“D1"  num="130"  quantity="5"/>
    <item dept="D2"  num="230"  quantity="10"/>
</order>
```

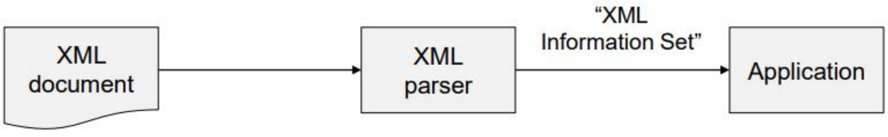
```
<item num="130" name="N1" quantity="5"/>

<item num="230" name="N2" quantity="10"/>
```

How XML Works

- Strict rules regarding the syntax of XML documents - allows for the development of XML parsers that can read documents

- Applications that need to understand an XML document will use a parser

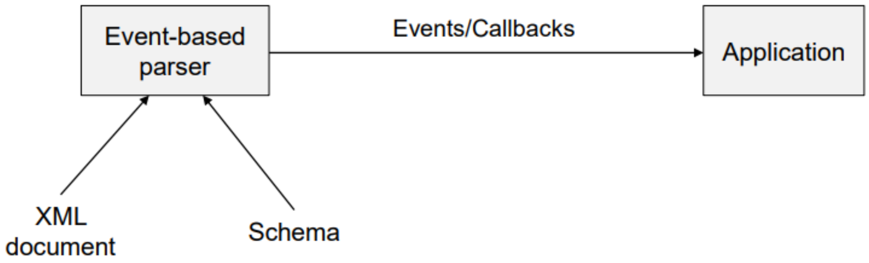


Splits the document into individual pieces

Event-Based Parsers

- Report parsing events, such as the start and end of elements, directly to the application

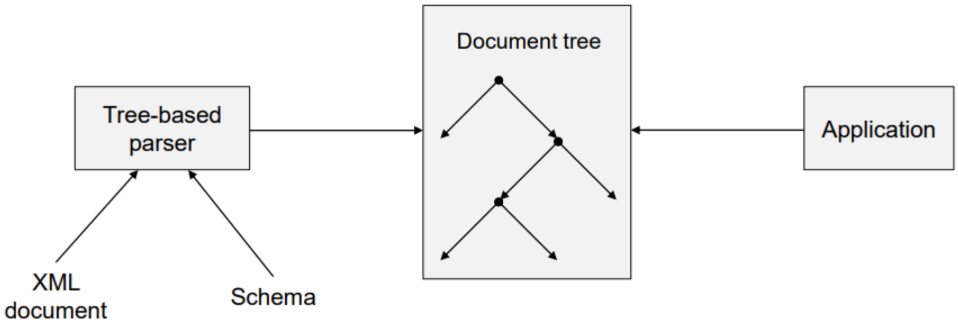
- The application implements handlers to deal with the different events



Tree-Based Parsers

- Map an XML document into an internal tree structure stored in main memory

- The application navigates that tree



Event-Based vs. Tree-Based Parsers

Event-based	Tree-based
• Sequential access	• Random access
• Fast	• Slow
• Constant memory	• Proportional to the document size

+

• Large documents	• Small documents
• Lack of data structure	• Ready-made data structure

Standards for XML Parsers

- SAX - Simple API for XML (event-based)
  - "De facto" standard

- DOM - Document Object Model (tree-based)
  - W3C standard

… APIs to read and interpret XML documents

SAX - Simple API for XML

- An event-based API for reading XML documents

- No W3C standard, but a "de facto" standard - very popular

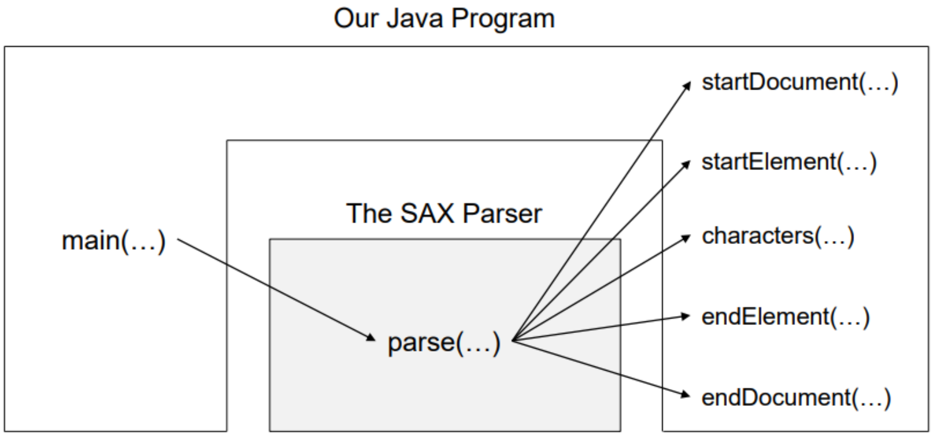
- Free and open source - http://www.saxproject.org

- Originally a Java-only API, but there are versions for several other programming languages (C++, Python, Perl, etc.)

ATTENTION: We focus on the Java version of the API

Callbacks

- SAX works through callbacks - we call the parser, it calls methods that we supply



Callbacks

- SAX works through callbacks - we call the parser, it calls methods that we supply

- Callback functions are divided into four event handlers:
  - ContentHandler - it handles basic parsing callbacks (e.g., element starts)
  - ErrorHandler - it handles parsing errors
  - DTDHandler - it handles notation and unparsed entity declarations
  - EntityResolver - customized handling for external entities

the crucial event handlers

Class DefaultHandler

- In package org.xml.sax.helpers
- Implements all the handlers mentioned before (ContentHandler, ErrorHandler, DTDHandler, EntityResolver)
- An adapter class - it provides empty methods for every method declared in each of the four interfaces
- Extend it and override the methods that are important for the current application

Features

- SAX uses features to control parser's behavior
- Each feature has an absolute URI as a name
- Features are either true or false

Some Features

- http://xml.org/sax/features/validation
  - Validate the document and report validity errors
  - Default value is false

- http://xml.org/sax/features/namespaces
  - The parser is namespace-aware
  - Default value is true

see https://xerces.apache.org/xerces2-j/features.html

Set Feature

```
public void setFeature(java.lang.String name, boolean value)
          throws SAXNotRecognizedException
          throws SAXNotSupportedException
```

- name - the name of the feature (an absolute URI)

- value - value of the feature (true or false)

- SAXNotRecognizedException - if the feature cannot be assigned
  - Turn on validation in a non-validating parser

- SAXNotSupportedException - if the feature cannot be activated
  - Turn on validation (in a validating parser) when part of the document has been already parsed

DOM - Document Object Model

- A tree-based API for reading and manipulating documents like XML and HTML

- A W3C standard

- The XML DOM is a standard for how to get, change, add or delete XML elements

DOM Nodes and Trees

All individual pieces of an XML document are represented as nodes of different types

- every element as an element node
- text in an element as a text node
- every attribute as an attribute node
- a comment as a comment node
- a document node denotes a document

- The whole XML document is seen as a tree of such nodes (node-tree)
- All nodes can be accessed through the node-tree
- Nodes can be modified/deleted, and new elements can be created

Relationships Among Nodes

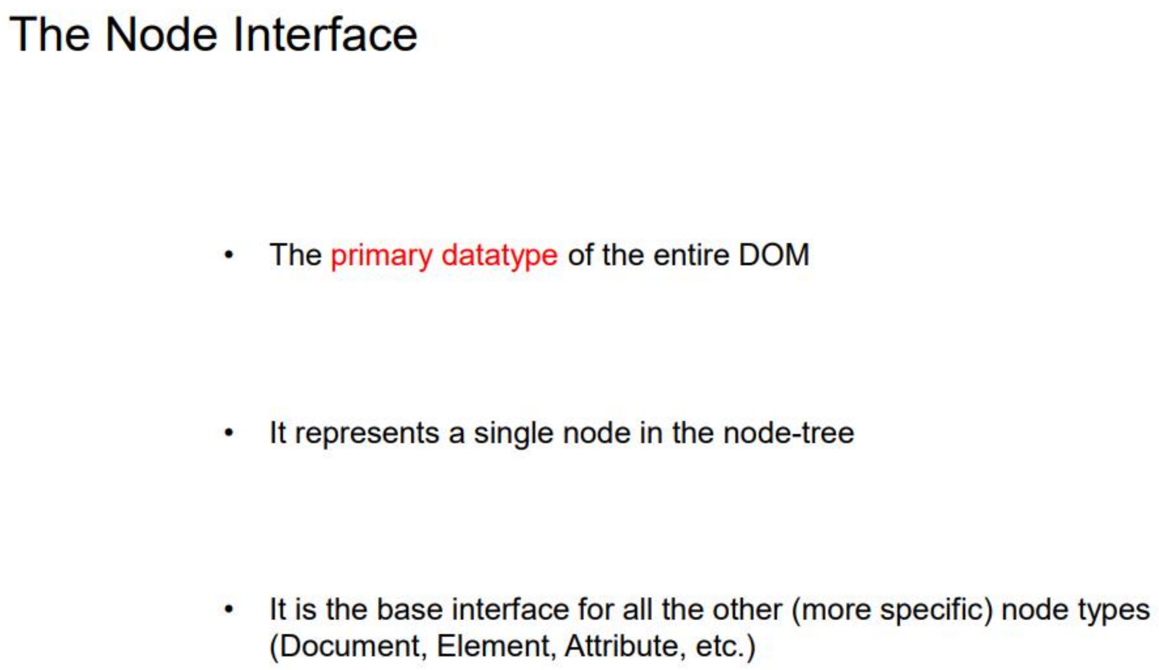
- The terms parent, child and sibling are describing the relationships among nodes

- In a node-tree:
  - Every node has exactly one parent (except the root of the tree)
  - A node can have an unbounded number of children
  - A leaf node has no children
  - Siblings have the same parent

XML DOM Parser

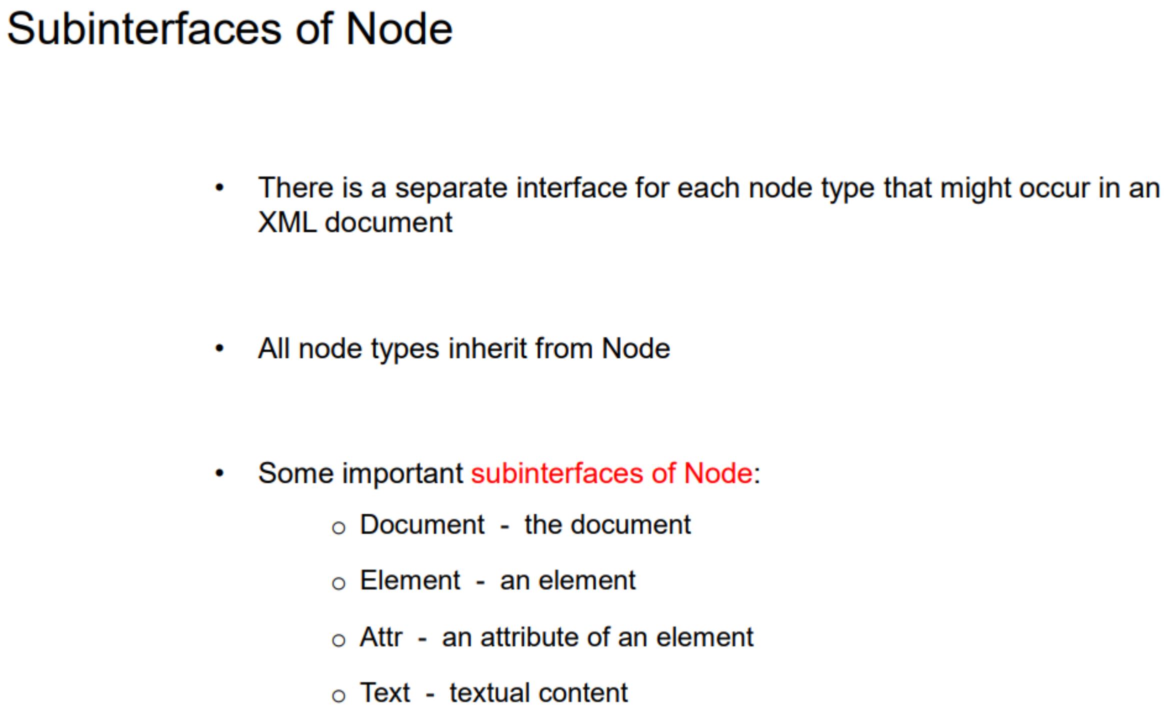
- A parser converts the document into an XML DOM object that can be accessed with Java

- XML DOM describes methods to traverse node-tree, access, insert and delete nodes



The Node Interface

- The primary datatype of the entire DOM
- It represents a single node in the node-tree
- It is the base interface for all the other (more specific) node types (Document, Element, Attribute, etc.)



Subinterfaces of Node

- There is a separate interface for each node type that might occur in an XML document
- All node types inherit from Node
- Some important subinterfaces of Node:
  - Document - the document
  - Element - an element
  - Attr - an attribute of an element
  - Text - textual content

```
HashMap<String, HashMap> selects = new HashMap<String, HashMap>();

for(Map.Entry<String, HashMap> entry : selects.entrySet()) {
   String key = entry.getKey();
   HashMap value = entry.getValue();

   // do what you have to do here
   // In your case, another loop.
}
```