M8 L1
XML Basics

Ira A. Fulton Schools of Engineering
Arizona State University

Lecture Outline

The Roles of XML

XML Nodes

- Elements
- Attributes

Representation

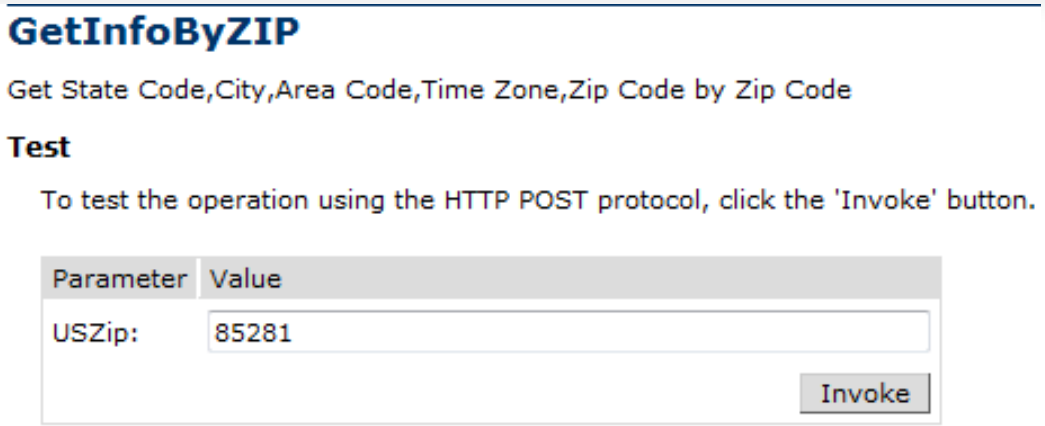
Special Elements and Characters

- Empty Element
- CDATE and PCDATA
- Whitespaces
- Namespace

The Role of XML in SOC

- XML is used for representing almost all languages, protocols, and data structures in SOC: SOAP, WSDL, UDDI, ebXML, BEPL, OWL, …
- WSDL Web services were also called XML services
- Many internal files in .Net are in XML
  - Configuration files Web.config, which stores application setting, user data, security options, …
  - XHTML: XML version of html. .Net uses XHTML, instead of html, for the page generated from aspx page. It is a stricter version of html based on XML standard;
  - DataSet in ADO .Net database management uses XML to represent collection of tables;
  - …

Processing DataSet from Web Services




```
<?xml version="1.0" encoding="utf-8" ?>
<NewDataSet>
    <Table>
        <CITY>Tempe</CITY>
        <STATE>AZ</STATE>
        <ZIP>85281</ZIP>
        <AREA_CODE>602</AREA_CODE>
        <TIME_ZONE>M</TIME_ZONE>
    </Table>
</NewDataSet>
```

XML: Extensible Markup Language

- XML is of plain text and with self-describing.
- It uses self-defined markup tags surrounding sentences, paragraphs, and even complete documents.
- The self-defined tags provide additional information about the data they envelope.
- It uses elements and attributes to provide both a logical structure and a physical structure to the document.
- XML contains metadata: Data about data.
- It is a meta language: a language used for defining other languages

XML Element, Attribute, and Document

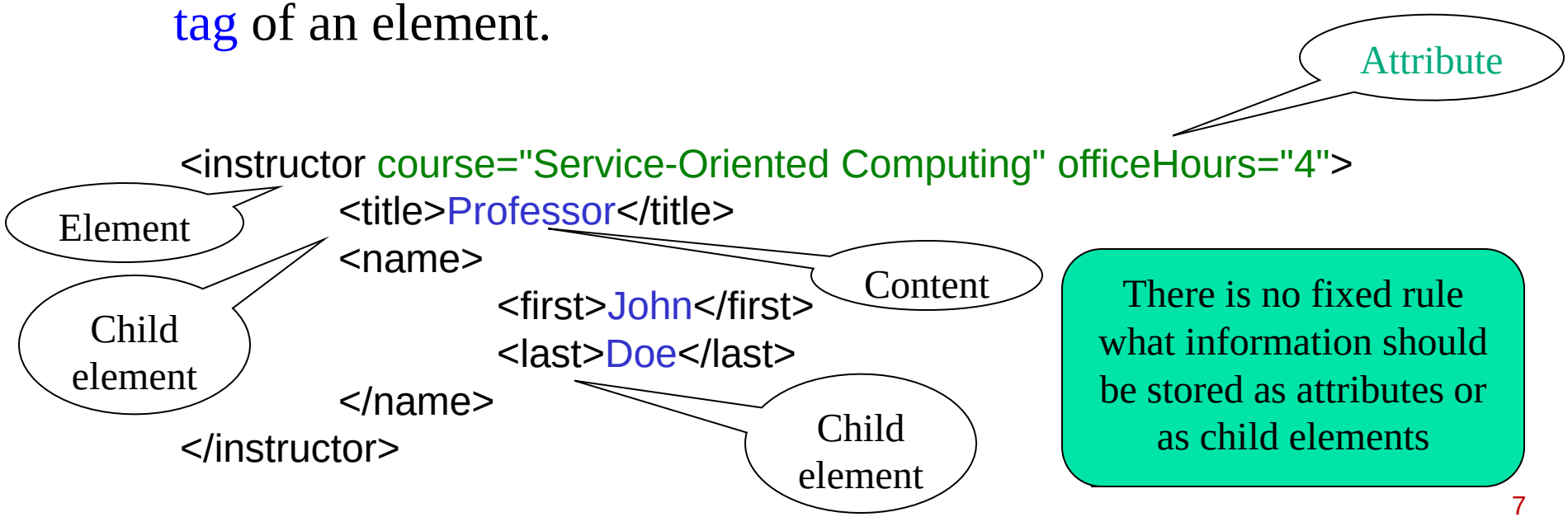
- An XML document starts with a prolog consisting of a declaration and optional references to external documents (namespaces):

```
<?xml version="1.0" encoding="UTF-8">
<instructor course="Service-Oriented Computing">
    <name>
        <first>John</first>
        <last>Doe</last>
    </name>
</instructor>
```

- The first line declares that the document follows XML version 1.0 and uses encoding method UTF-8.
- The remaining lines contain data, called "elements", stored in the XML file.

XML Element, Attribute, and Document

- Each **element** is quoted by a pair of tags (opening and closing tags). Tag names are not predefined and can be chosen freely except a few restrictions. A tag must
  - start with a letter, an underscore, or a colon.
  - not start with the reserved word xml (case insensitive).
- An **attribute** is a name = "value" equation inside the opening tag of an element.



XML Element, Attribute, and Document

There is no fixed rule what information should be stored as child elements or as attributes. However, normally

- **Elements** are used for defining data that are integral to the document. *Elements form a rooted tree*.
- **Attributes** are used for defining out-of-band data, which give "additional" information. *They are stored linearly in the element*.

```
<instructor >
        <title degree= "PhD" year = "2001" school = "ASU">Professor</title>
        <name>
                <first>John</first>
                <last>Doe</last>
        </name>
        <course level = "Senior">Service-Oriented Computing</course>
        <officeHours>4</officeHours>
</instructor>
```

XML Syntax: Well-Formed XML Doc

An XML document must be written following certain syntax rules.

- Must conform to XML specification at: www.w3.org/TR/REC-xml
- There is a unique root element. All other elements are children or descendants of the root element.
- Each element is quoted between an opening and a closing tag.
- Nested tags are allowed but tags may not overlap. For example, the following is not allowed:
  <**title**>Professor<name>Doe</**title**></name>
- Element tag name and content must meet certain restrictions.
- Although not required, tag names should be self-describing and depict the true meaning of the content embedded in them.
- Tag names need to be "declared" or "defined" in a definition language, e.g., in XML Schema (later lecture).

Attributes

- Attribute values must always be in quotes. Both single and double quotes are valid. Double quotes are most common though.
- Attributes are always contained within the opening tag of an element:

```
<course level = “senior” required = “true”>
        Service-Oriented Computing
</course>
```

- Attributes within an element must have unique names: The same attribute name can appear once only within an element.

```
<course level = “senior” level = “graduate”> … <course>
        is not acceptable
```

- Attributes are more complex to manipulate by program code than elements, as they are not a part of the tree elements

Special XML Empty Element Tag Convention

- An empty element is an element without content or child elements:

  <myElement></myElement>

  which can be represented with a single tag of the form

  <myElement />

- Why do we need empty elements? 

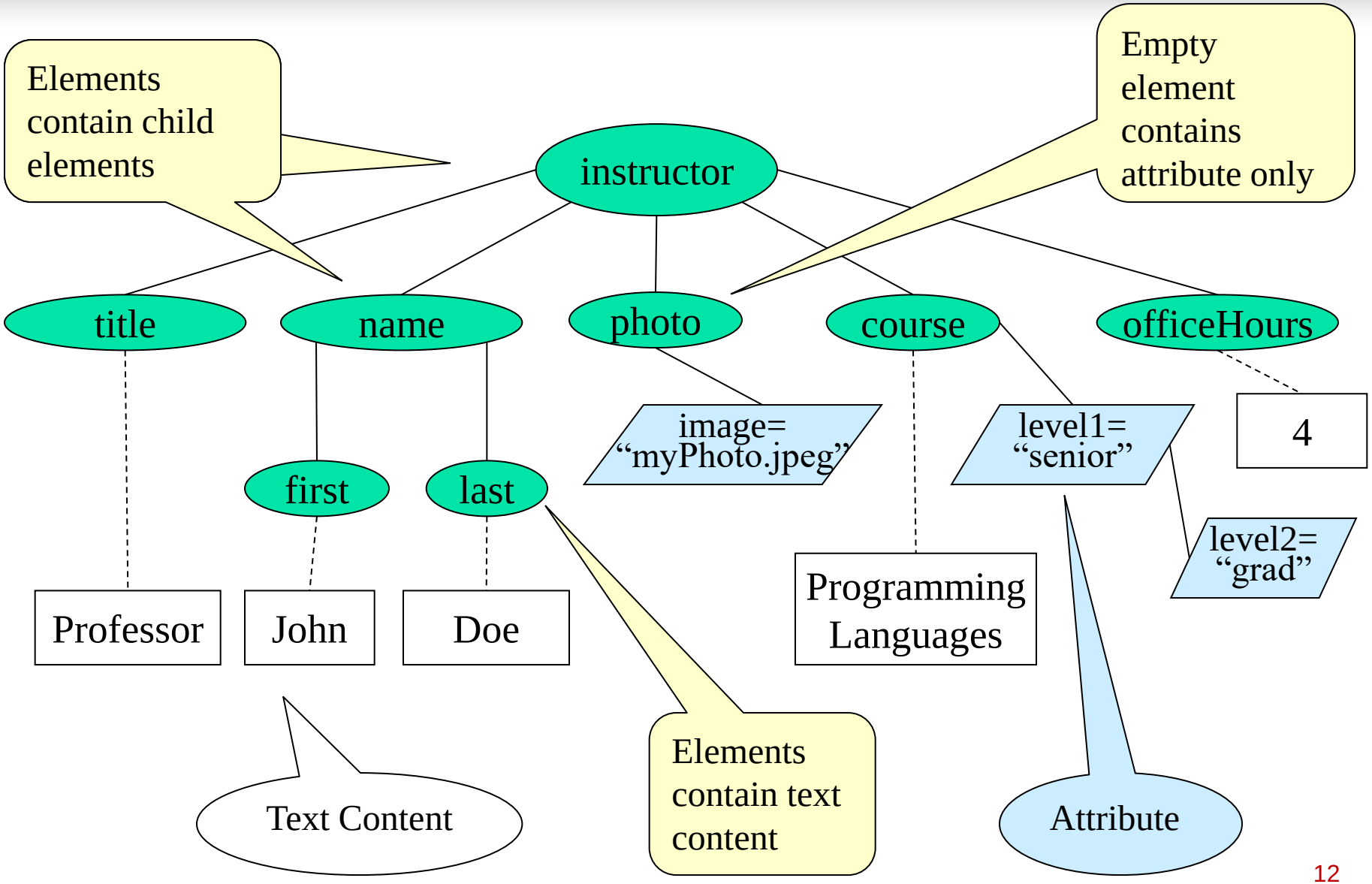
  It can still have attributes:

  <photo image =“myPhoto.jpeg” />

  which is a short hand notation of

  <photo image =“myPhoto.jpeg”> </photo>

XML Document Can be Visually Represented as a Rooted Tree



Representing Special Characters in Text Content

- Five characters are reserved for markup purpose. The parsers need to differentiate them from the text content characters.

character	Entity name	Meaning
<	lt	less than
>	gt	greater than
&	amp	ampersand
'	apos	apostrophe, e.g., computer's keyboard
"	quot	quotation

Two ways to differentiate them:

- Use **Entity Reference** for these characters

  For example, to represent 0 < x < 100, you can use:
  <Range> 0 < x < 100</Range>

Some textboxes do not take characters < and >, because the data will be stored in XML

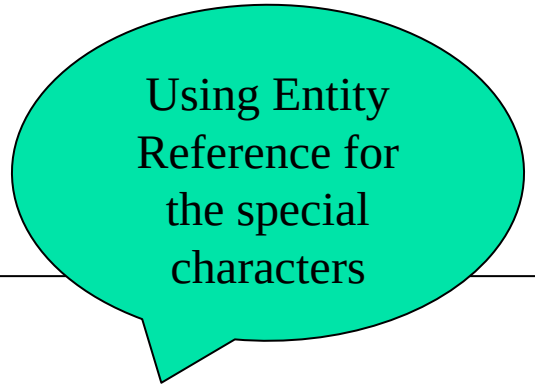
- Use **Character Reference** (ASCII code) for these characters

  For example,
  <Range> 0 < x < 100</Range>

Character and Parsed Character Data

If a piece of text has many markup characters, the reference method makes the text not readable.

```
<myCode>
    function AND(a, b) {
        if (a == “true” && b == “true”) then
            { return “true” }
        else
        { return “false” }
    }
</myCode>
```




```
<myCode>
    function AND(a, b) {
        if (a == &quot;true&quot; &amp;&amp; b == &quot;true&quot;) then
            { return &quot;true&quot; }
        else
        { return &quot;false&quot; }
    }
</myCode>
```

Using *Character* and *Parsed Character Data*

XML parsers differentiate two types of textual data:

- **CDATA -** Character Data: Data that appears between the pair: <![CDATA[and]]> tags
- **PCDATA -** Parsed Character Data: Any data that are not in CDATA tags.
- Using CDATA for text with many markup characters is more readable:

```
<myCode>
    <![CDATA[
        function AND(a, b) {
            if (a == “true” && b == “true”) then
                { return “true” }
            else
                { return “false” }
        }
    ]]>
</myCode>
```

Representing Special Characters in CDATA

- XML parsers will ignore (not do syntax checking) on CDATA but parse PCDATA — that is, interpret it as a part of the markup language.
- The practical implication is that for data between the tags <![CDATA[and]]>, we do not have to conform to the syntax of XML.
- Outside the tags <![CDATA[and]]>, we must conform to the syntax of XML.

Whitespace in XML

- XML defines four characters to be whitespace

Character	Unicode Value
tab	#x9
newline	#xA
carriage return	#xD
space	#x20

- Why would whitespace be handled differently from other characters? 

Collapsing (normalizing) or Preserving

- Normalization: Remove additional whitespaces
- Preserving: Keep all whitespaces

To be discussed later

- Declaration in XML Document Type Definition (DTD)

```
<!DOCTYPE instructor [
<!ATTRIBUTE code xml:space #FIXED “preserve”>
<!EMEMENT name (first, middle?, last)>
<!EMEMENT first (#PCDATA)>
<!EMEMENT last (#PCDATA)>
<!EMEMENT course (#PCDATA)>
<!EMEMENT officeHours (#PCDATA)>
]>
```

If use “default” Whitespaces will be collapsed!

Comments

```
<?xml version="1.0" encoding="UTF-8">
<myCode>
    <!-- This function performs logic AND operation.
         It takes two boolean variables as input
    -->
    function AND(a, b) {
        if (a == &quot;true&quot; &amp;&amp; b ==
&quot;true&quot;) then
            { return &quot;true&quot; }
        else
        { return &quot;false&quot; }
    }
</myCode>
```

Namespaces

- Namespaces provide a mechanism for qualifying (scoping) element and attribute names to avoid naming collisions.
- In C++, we can use the namespace prefixes (scope resolution operator) to qualify its elements so that the elements won't clash if used in the same document with other elements having the same names but different definitions.

```
<?xml version="1.0"?>
<cse:Courses
    xmlns:cse="http://scai.asu.edu/courses/cse.php"
    xmlns:asu="http://www.asu.edu/it/tempe/classrooms/"
    <cse:Course>
        <cse:Name>Distributed Software Development</cse:Name>
        <cse:Code >CSE445</cse:Code>
        <cse:Level>Senior</cse:Level>
        <asu:Room asu:Image="layout210.jpeg" >BYAC210</asu:Room>
        <cse:Cap>40</cse:Cap>
    </cse:Course>
```

XML namespace

An Element Qualifier does not apply to Attribute

```
...
<cse:Course>
    <cse:Name>Introduction to Programming Languages</cse:Name>
    <cse:Code >CSE240</cse:Code>
    <cse:Level>Sophomore</cse:Level>
    <asu:Room asu:Image="layout110.jpeg">BYAC110</asu:Room>
    <cse:Cap>82</cse:Cap>
</cse:Course>
</cse:Courses>
```

Note, an element's attributes are not automatically scoped to a namespace. In the following example, the Image attribute doesn't belong to a namespace:

```
<asu:Room Image="layout110.jpeg"></asu:Room>
```

However, you can use namespace prefixes to join attributes to namespaces:

```
<asu:Room asu:Image="layout110.jpeg"></asu:Room>
```


Ira A. Fulton Schools of
Engineering
Arizona State University

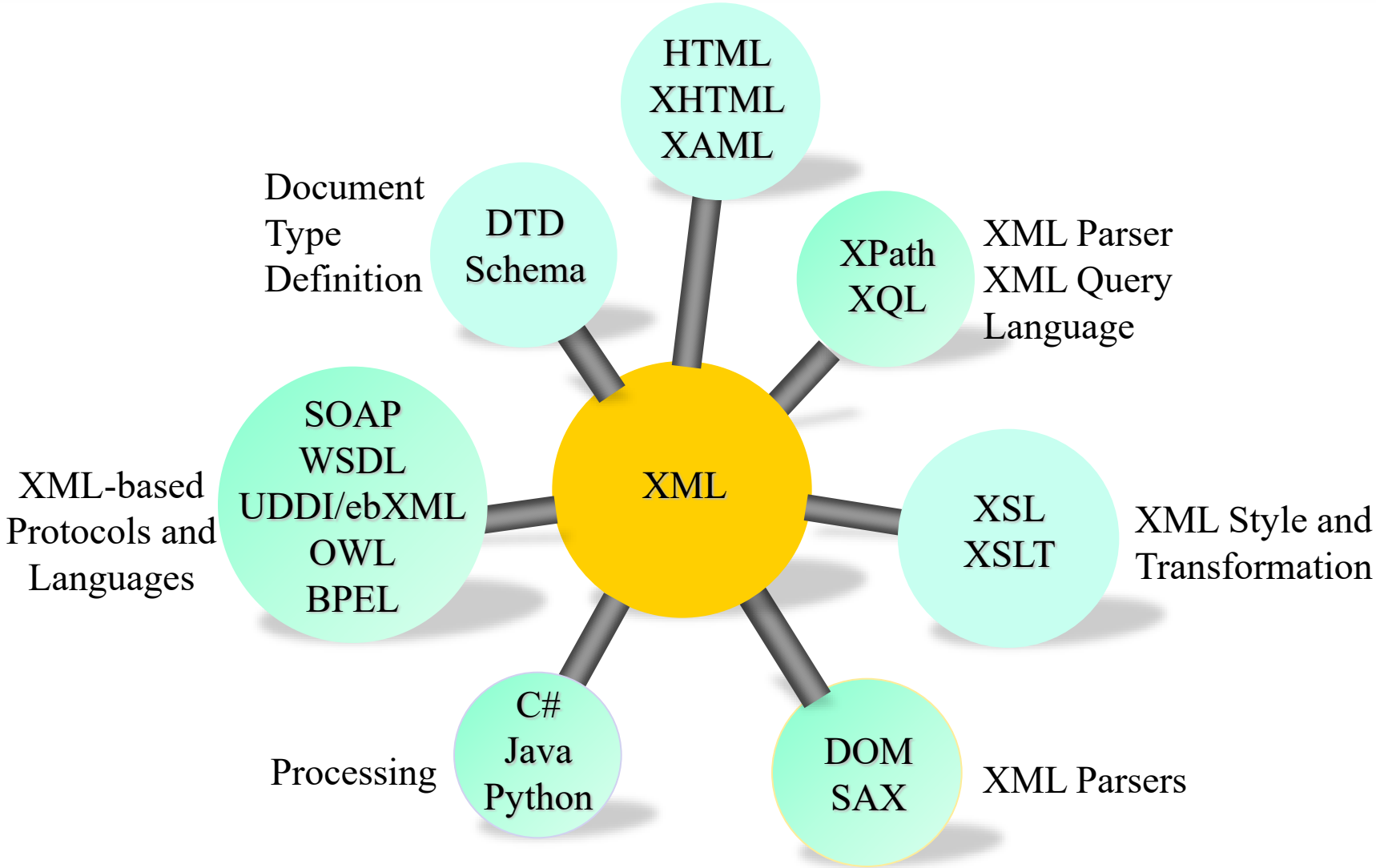
M8 L2
XML Processing and DOM Model

Ira A. Fulton Schools of Engineering
Arizona State University

Lecture Outline

- XML and Related Technologies
- XML Processing Overview and Models
- DOM Model

XML Related Technologies



XML Parsers

There are three types of parsers for extracting data from XML

- DOM: Document Object Model
  - Standard: www.w3.org/TR/DOM-level-2-core
  - Read the entire document into the memory for random access
    -- problem: if the doc is huge.
  - Python DOM: https://docs.python.org/3/library/xml.dom.html
  - Java DOM Parser: http://www.jdom.org/
  - MSXML: free MS parser that supports DOM, as well as SAX
- SAX: Simple (Stream) API for XML
  - www.saxproject.org and https://docs.python.org/3/library/xml.sax.html
  - Not a w3 standard
  - Event-based API, read the document in a stream
  - Developed by the Java community, and supported in all environments
- XPath

XML Classes in .Net

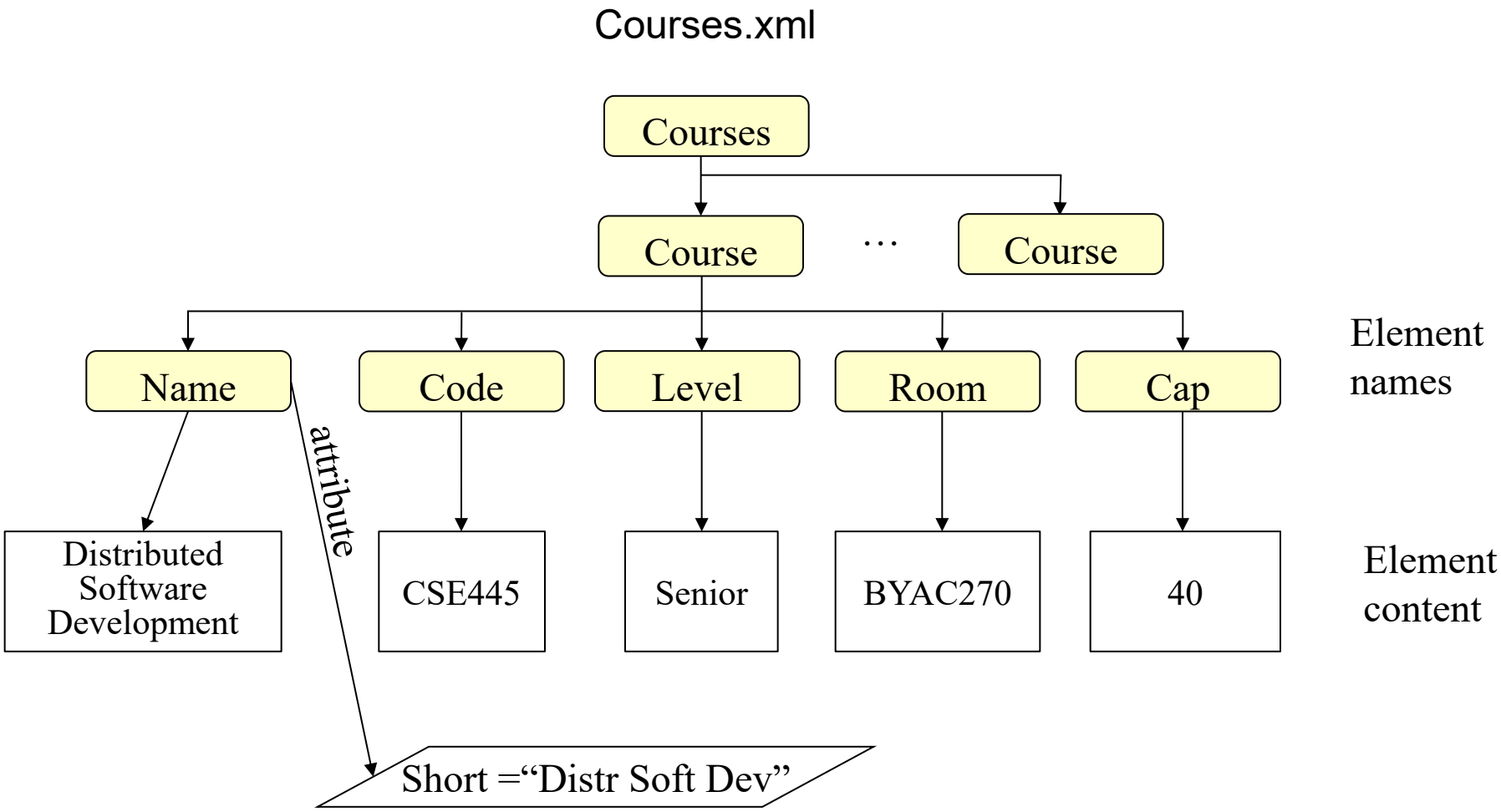
- The .Net FCL's *System.Xml* namespace offers a variety of classes for reading and writing XML documents.
- The *XmlDocument* class is similar to MSXML, but it is simpler to use.
- If you prefer a stream-based approach (SAX), you can use *XmlTextReader* or the schema-aware *XmlvalidatingReader*, instead.
- A complementary class named *XmlTextWriter* allows you to create XML documents from scratch.

The *XmlDocument* Class in .Net

- ***XmlNode*** class is the base class in .Net's implementation of the DOM.
- Each node of a DOM tree is an instance of *XmlNode* class, which exposes methods and properties for navigating a DOM tree, e.g., reading and writing node content, adding and removing nodes.
- *XmlDocument* derives (inherits) from *XmlNode* and adds methods and properties of its own, supporting loading and saving of documents, the creation of new nodes, and other operations.
- *XmlDocument* provides a programmatic interface to XML documents that complies with the DOM specification.
- It represents a document as an upside-down rooted tree of **nodes**, with the root element, or document element, at the top.

DOM Representation of a Simple XML Doc

http://venus.sod.asu.edu/WSRepository/xml/Courses.xml



Online XML Viewer: http://codebeautify.org/xmlviewer

XML VIEWER

XML Input sample

```
1  <Courses xmlns="http://venus.eas.asu.edu/WsRepository/xml"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi
   :schemaLocation="http://venus.eas.asu.edu/WsRepository/xml
   Courses.xsd">
2
3  <Course>
4  <Code Undergrad="CSE445" Graduate="CSE598"/>
5  <Name>Distributed Software Development</Name>
6  <Instructor>
7  <Name>
8  <First>Yinong</First>
9  <Last>Chen</Last>
10 </Name>
11 <Contact Office="BYENG414">
12 <Phone>480-965 2769</Phone>
13 </Contact>
14 </Instructor>
15 <Room>BYAC210</Room>
16 <cap>57</cap>
17 </Course>
18 <Course>
19 <Code Undergrad="CSE360"/>
20 <Name>Software Engineering</Name>
21 <Instructor>
22 <Name>
23 <First>James</First>
24 <Last>Collofello</Last>
25 </Name>
26 <Contact>
27 <Phone>480-965 0000</Phone>
28 </Contact>
29 </Instructor>
30 <Room>BYAC270</Room>
31 <cap>65</cap>
32 </Course>
33 </Courses>
```

Load Url
Browse
Tree View
Beautify / Format
Minify

XML To JSON
Export To CSV
Download

- courses ..
  - @xmlns: http://venus.eas.asu.edu/WsRepository/xml
  - @xmlns:xsi: http://www.w3.org/2001/XMLSchema-instance
  - @xsi:schemaLocation: http://venus.eas.asu.edu/WsReposito Courses.xsd
  - course ..
    - code
      - @Undergrad: CSE445
      - @Graduate: CSE598
    - name Distributed Software Development
    - instructor ..
      - name ..
        - first Yinong
        - last Chen
      - contact ..
        - @Office: BYENG414
        - phone 480-965 2769
    - room BYAC210
    - cap 57
  - course ..
    - code
      - @Undergrad: CSE360
    - name Software Engineering
    - instructor ..
      - name ..
        - first James
        - last Collofello

Using *XmlDocument* to load a Doc

- The following statements create an *XmlDocument* object and initialize it with the contents of Courses.xml:

```
XmlDocument xd = new XmlDocument ();      //create an object
xd.Load ("Courses.xml");                  // call Load method
```

- The method *Load* parsers the file "Courses.xml" and builds an in-memory tree representation.
- It throws an *XmlException* if the document isn't well-formed -- Doing the validation while reading.

Load a Doc from a Remote Site

- The following statements create an *XmlDocument* object and initialize it with the contents of Courses.xml:

Code embedded in Web page

```
<%@ Page language=C# debug="true" %>
<%@ Import Namespace="System.Xml" %>
<SCRIPT runat="server">
void Page_Load(Object Sender, EventArgs e)
{
  XmlDocument xd = new XmlDocument();
  xd.Load("http://venus.sod.asu.edu/WSRepository/xml/Courses.xml");
  this.Label.Text = xd.FirstChild.Name;
}
</SCRIPT>
```

Reading XML Doc and Write to Screen

```
void OutputNode (XmlNode node)        // recursive
{
    If (node == null) exit;
    Console.WriteLine ("Type={0}\tName={1}\tValue={2}",
                node.NodeType, node.Name, node.Value);
    if (node.HasChildNodes)
    {
        XmlNodeList children = node.ChildNodes;
        foreach (XmlNode child in children)
            OutputNode (child);
    }
}
```

Reading XML Doc and Write to Screen (Python)

```
def OutputNode(node):
    if node == None:
        return
    print("Type = {0}\tName = {1}\tValue =
{2}".format(node.nodeType, node.nodeName,
node.nodeValue))
    if node.hasChildNodes():
        children = node.childNodes
        for child in children:
            OutputNode(child)   // Is it tail-recursive
```

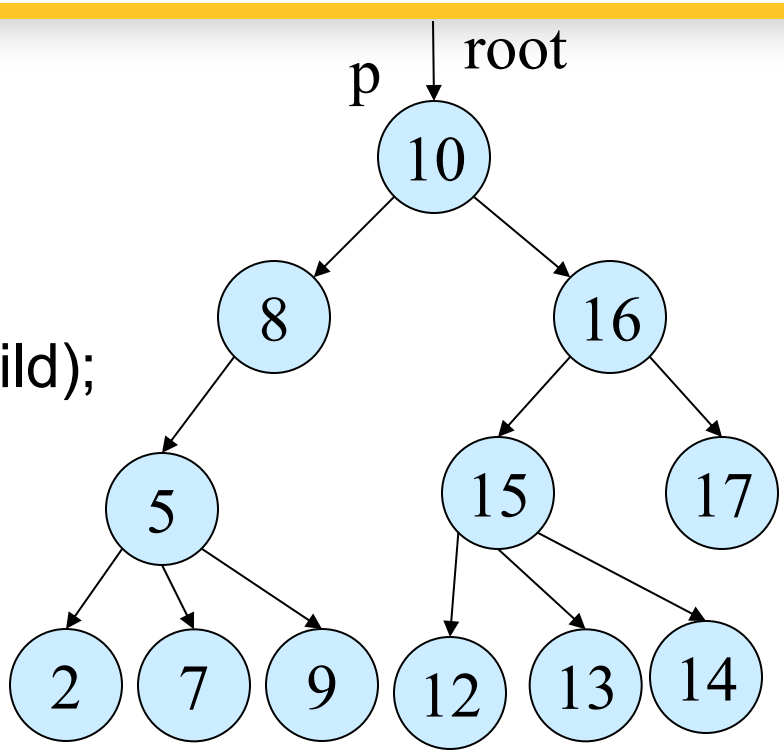
```
                    // What traversing order is used
```

Pre-Order Tree Traversing Algorithms

```
preorderTraverse(p)
if p ≠ null then
    print(p.data);
    for each child node
        preorderTraverse(p.nextChild);
```

```
inorderTraverse(p)
if p ≠ 0 then
    inorderTraverse(p.left);
    print(p.data);
    inorderTraverse(p.right);
```

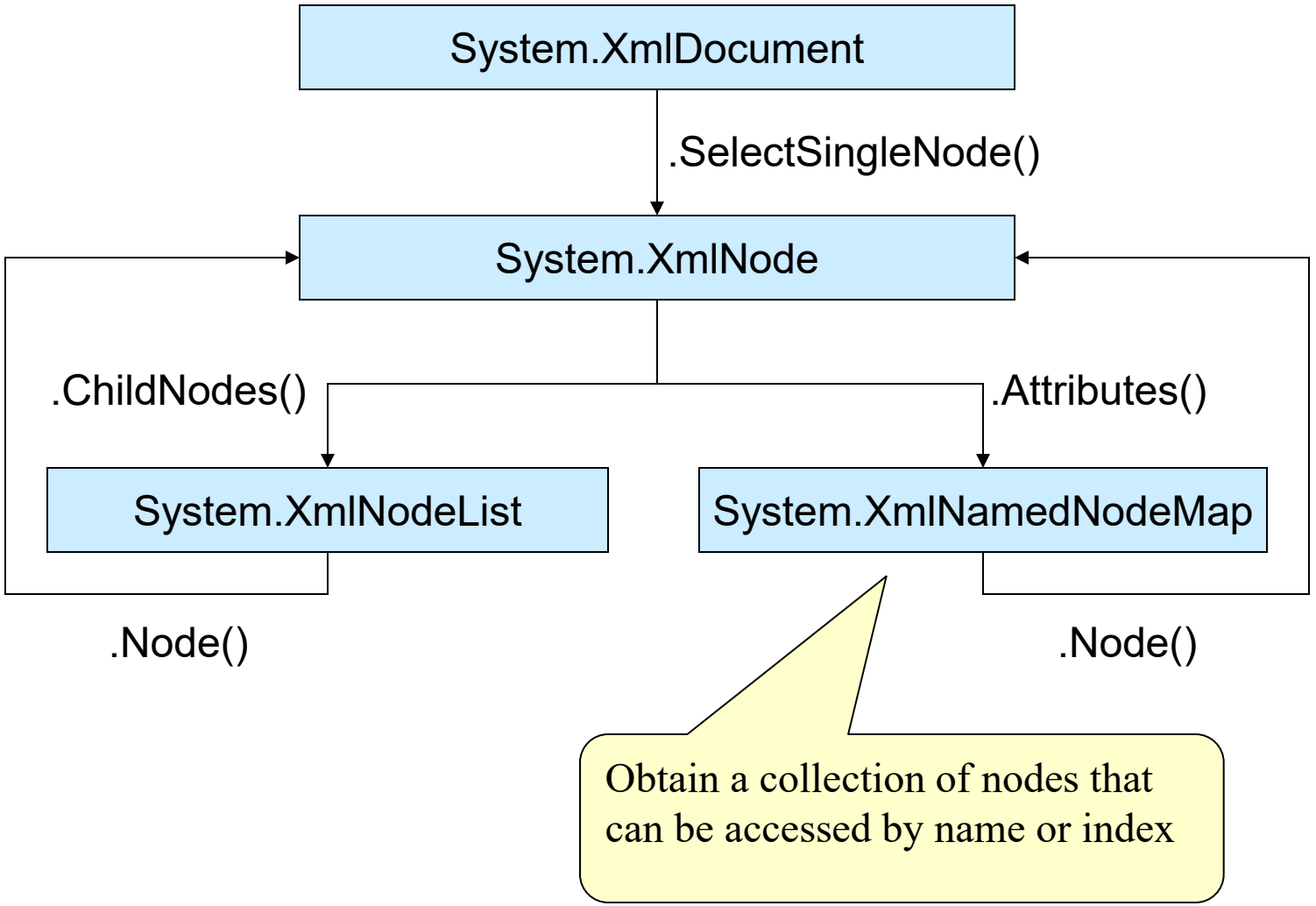
```
postorderTraverse(p)
if p ≠ 0 then
    for each child node
        postorderTraverse(p.nextChild);
        print(p.data);
```



.Net Classes Defined in System.Xml Namespace

XmlAttribute	Represents an attribute, with methods dealing with attributes.
XmlDocument	Represents an XML document, dealing with the overall doc.
XmlDocumentType	Represents the document type declaration.
XmlElement	Represents an element.
XmlLinkedNode	Deals with nodes immediately preceding or following this node.
XmlNamedNodeMap	Represents a collection of nodes that can be accessed by name or index.
XmlNode	Represents a single node in the XML document.
XmlNodeList	Represents an ordered collection of nodes.
XmlReader	Represents a reader that provides fast, non-cached, forward-only access to XML data.

Typical Use of Classes and Methods in System.Xml Namespace



Methods in XmlDocument Class

Attributes	Gets an XmlAttributeCollection containing the attributes of this node (inherited from XmlNode)
BaseURI	Overridden. Gets the base URI of the current node.
ChildNodes	Gets all the child nodes of the node (inherited from XmlNode)
DocumentElement	Gets the root XmlElement for the document.
DocumentType	Gets the node containing the DOCTYPE declaration.
FirstChild	Gets the first child of the node (inherited from XmlNode)
HasChildNodes	Gets a value indicating whether this node has any child nodes (inherited from XmlNode)
IsReadOnly	Overridden. Gets a value indicating whether the current node is read-only.
Item	Overloaded. Gets the specified child element (inherited from XmlNode)

Member Functions in XmlDocument Class (contd.)

LastChild	Gets the last child of the node (inherited from XmlNode)
NextSibling	Gets the node immediately following this node (inherited from XmlNode)
NodeType	Overridden. Gets the type of the current node.
ParentNode	Overridden. Gets the parent node of this node (for nodes that can have parents).
Prefix	Gets or sets the namespace prefix of this node (inherited from XmlNode)
PreserveWhitespace	Gets or sets a value indicating whether to preserve white space in element content.
PreviousSibling	Gets the node immediately preceding this node (inherited from XmlNode)
Schemas	Gets or sets the XmlSchemaSet object associated with this XmlDocument.
Value	Gets or sets the value of the node (inherited from XmlNode)

XmlNodeType Example

XmlNodeType	Example
Attribute	<Course image="My445.jpeg">
CDATA	<![CDATA["This is character data"]]>
Comment	<!-- This is a comment -->
Document	<Courses>
Document Type	<!DOCTYPE Courses SYSTEM "Courses.dtd">
Element	*<Course>*
Entity	<!ENTITY filename "Courses.xml">
EntityReference	<
Notation	<!NOTATION GIF89a SYSTEM "gif">
ProcessingInstruction	<?xml-stylesheet type="text/xsl" href="Courses.xsl"?>
Text	<Code>CSE445</Code>
Whitespace	<Name/>\r\n<Name/>
XmlDeclaration	<?xml version="1.0"?>


Ira A. Fulton Schools of
Engineering
Arizona State University

M8 L3
XML Reader and Writer

Ira A. Fulton Schools of Engineering
Arizona State University

Lecture Outline

- DOM Review
- SAX Model
- XML Reader,
- XML Writer

DOM Review

```
XmlDocument docRef= new XmlDocument(); // document object
docRef.Load("http://venus.sod.asu.edu/WSRepository/xml/Courses.xml");
XmlNode nodeRef = docRef.DocumentRef;  // node object
OutputNode(nodeRef);   // Call the recursive method to traverse tree
```

```
static void OutputNode (XmlNode node)  {     // recursive method
        If (node == null) exit;
        Console.WriteLine ("Type={0}\tName={1}\tValue={2}",
                        node.NodeType, node.Name, node.Value);
        if (node.HasChildNodes)  {
            XmlNodeList children = node.ChildNodes;
            foreach (XmlNode child in children)
                OutputNode (child);
    }    }
```

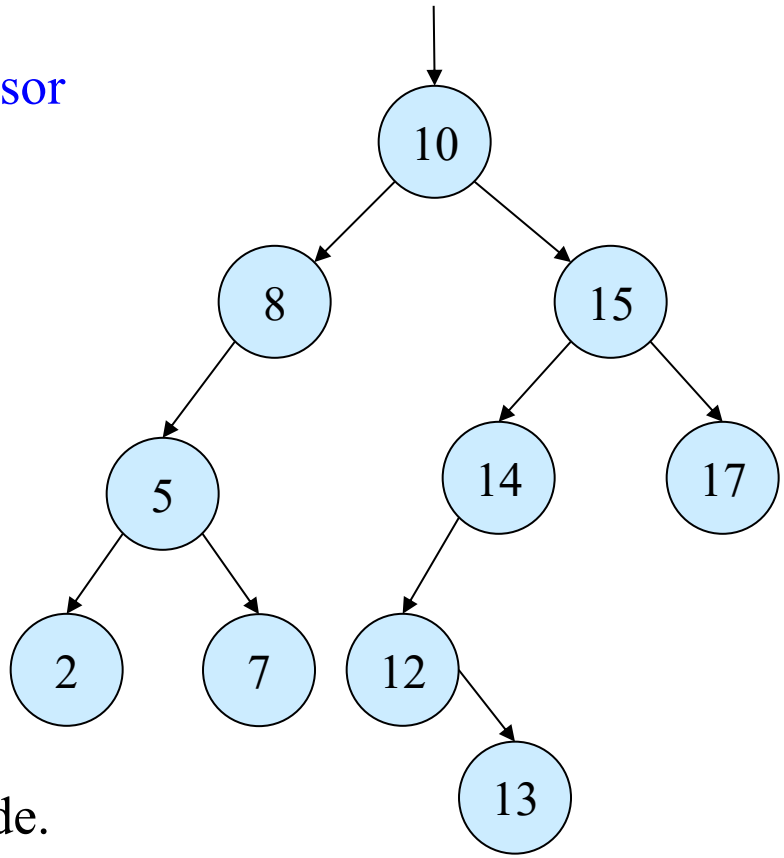
See text section 4.2.1 for full working code

Using SAX Model: *XmlTextReader* Class

- *XmlDocument* allows you to move backwards, forwards, and sideways, and make changes in a in-memory document tree.
- If you want simply to read XML and are less interested in the structure than its contents, you can use *XmlTextReader* class.
- *XmlTextReader,* also belongs to the *System.Xml* namespace, provides a fast, forward-only, read-only interface to XML documents.
- It is stream-based, just like reading a text file. It is more memory-efficient than *XmlDocument*, because it does not read the entire document into memory at one time, … but may be less time efficient, as it does not allow you to define algorithm or path.
- It is easier than *XmlDocument* to read through a document searching for particular elements, attributes, or other content items. The recursive method is built-in!

Using *XmlTextReader* is Simple

- The basic idea is to create an *XmlTextReader* object from a file, a URL, or other data source;
- Call *XmlTextReader.Read( )* repeatedly until you find the content you're looking for or reach the end of the document;
- Each call to *Read( )* advances an imaginary cursor to the next node in the document (normally, in pre-order traversing);
- *XmlTextReader* properties, such as *NodeType, Name, Value,* and *AttributeCount,* expose information about the current node;
- Methods, such as *GetAttribute, MoveToFirstAttribute,* and *MoveToNextAttribute,* let you access the attributes, if any, attached to the current node.



Some Member Functions in XmlTextReader Class

AttributeCount	Overridden. Gets the number of attributes on the current node.
Depth	Overridden. Gets the depth of the current node to the root in the XML document.
EOF	Overridden. Gets a value indicating whether the reader is positioned at the end of the stream.
HasAttributes	Gets a value indicating whether the current node has any attributes (inherited from XmlReader)
HasValue	Overridden. Gets a value indicating whether the current node can have a Value other than String.Empty.
IsEmptyElement	Overridden. Gets a value indicating whether the current node is an empty element (for example, <MyElement/>).
Name	Overridden. Gets the qualified name of the current node.
Value	Overridden. Gets the text value of the current node.
ValueType	Gets The Common Language Runtime (CLR) type for the current node (inherited from XmlReader)

Read the XML Doc like Reading a Text File

```
XmlTextReader reader = null;
try {
    reader = new XmlTextReader ("Courses.xml");
    reader.WhitespaceHandling = WhitespaceHandling.None;
    while (reader.Read ()) {
        Console.WriteLine ("Type={0}\tName={1}\tValue={2}",
        reader.NodeType, reader.Name, reader.Value);
    }
}
finally {
    if (reader != null)
        reader.Close();
}
```

Local file is used. You can use a remote file, e.g., http://venus.sod.asu.edu/WSRepository/xml/Courses.xml

Also Print the Attributes

```
try {
    reader = new XmlTextReader ("Courses.xml");
    while (reader.Read()) {
        Console.WriteLine ("Type={0}\tName={1}\tValue={2}",
            reader.NodeType, reader.Name, reader.Value);
        If (reader.AttributeCount > 0) {
            while (reader.MoveToNextAttribute()) {
                Console.WriteLine ("Type={0}\tName={1}\tValue={2}",`
                    reader.NodeType, reader.Name, reader.Value);
            }
        }
    }
}
finally {
    if (reader != null)
    reader.Close ();
}
```

?

?

What does this line of code do?

(A) Create an unshaped object
(B) Create a tree and load the elements of the file into the tree;
(C) Create an object and shape the object according to the node of the file "Courses.xml";
(D) Load the element at the cursor into an object

Extracting a Particular Attribute Value (e.g. Image)

```
try {
     XmlTextReader reader = new XmlTextReader ("Courses.xml");
    reader.Whitespacehandling = WhitespaceHandling.None;
    while (reader.Read()) {
        if (reader.NodeType == XmlNodeType.Element &&
                                reader.Name == "Course" &&
                                reader.AttributeCount > 0) {
            while (reader.MoveToNextAttribute ())
            {
                if (reader.Name == "Image") {
                    Console.WriteLine (reader.Value);
                    break;
                }
            }
        }
    }
}
finally {
    if (reader != null)  reader.Close ();
}
```

Iteration pattern

See text section 4.2.2 for full working code

The *XmlTextWriter* Class

- The *XmlDocument* class is more frequently used for reading and modifying existing XML documents.
- *XmlTextWriter* is designed for creating an XML file from scratch, and it features an assortment of *Write* methods that generate various types of XML nodes, including
  - Document
  - elements,
  - attributes,
  - comments,
  - and more.

WriteStartElement

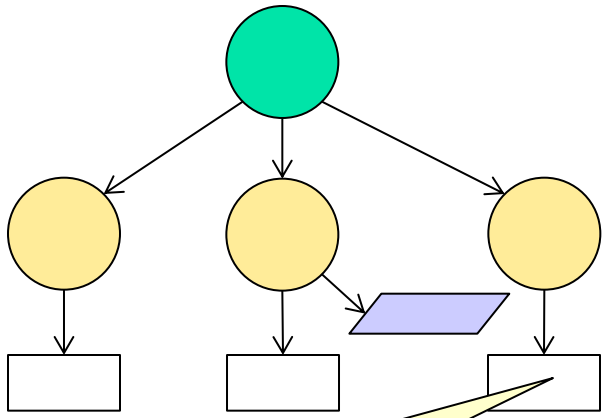
WriteElementString

WriteElementString

WriteAttributeString

WriteElementString

WriteEndElement



What do you do if you want to create a child element here, instead of text content?

WriteStartElement

Example: Creating an XML Doc

```
try {
    writer = new XmlTextWriter (“Courses.xml”,
    System.Text.Encoding.Unicode);
    writer.Formatting = Formatting.Indented;
    writer.WriteStartDocument ();
    writer.WriteStartElement(“Courses”);
    writer.WriteStartElement(“Course”);
    writer.WriteElementString(“Name”, “SOC”);
    writer.WriteElementString(“Code”, “CSE445”);
    writer.WriteElementString(“Level”, “Senior”);
    writer.WriteStartElement("Room");
    writer.WriteAttributeString("Image","layout150.jpeg");
    writer.WriteString("BYAC150");
    writer.WriteEndElement();
    writer.WriteElementString(“Cap”, “40”);
    writer.WriteEndElement();
    writer.WriteEndElement();
    writer.WriteEndDocument();
}
finally {  if (writer != null) writer.Close (); }
```

```
<?xml version=“1.0”
                encoding “utf-16”?>
<Courses>
  <Course >
   <Name>SOC</Name>
   <Code >CSE445</Code>
   <Level>Senior</Level>
   <Room Image=“layout150.jpeg”>
                BYAC150
   </Room>
   <Cap>40</Cap>
  </Course>
</Courses>
```

See text section 4.2.3 for full working code

Methods in the *XmlTextWriter* Class

public methodClose	Overridden. Closes this stream and the underlying stream.
public methodEquals (inherited from Object)	Overloaded. Determines whether two Object instances are equal.
public methodFlush	Overridden. Flushes whatever is in the buffer to the underlying streams.
public methodGetType (inherited from Object)	Gets the Type of the current instance.
public methodWriteAttributes (inherited from XmlWriter)	When overridden in a derived class, writes out all the attributes found at the current position in the XmlReader.
public methodWriteAttributeString (inherited from XmlWriter)	Overloaded. When overridden in a derived class, writes an attribute with the specified value.
public methodWriteCData	Overridden. Writes out a <![CDATA[...]]> block containing the specified text.
public methodWriteCharEntity	Overridden. Forces the generation of a character entity for the specified Unicode character value.
public methodWriteChars	Overridden. Writes text one buffer at a time.
public methodWriteEndAttribute	Overridden. Closes the previous WriteStartAttribute call.
public methodWriteEndDocument	Overridden. Closes any open elements or attributes and puts the writer back in the Start state.
public methodWriteEndElement	Overridden. Closes one element and pops corresponding namespace scope.
public methodWriteEntityRef	Overridden. Writes out an entity reference as &name;.
public methodWriteFullEndElement	Overridden. Closes one element and pops the corresponding namespace scope.
public methodWriteStartAttribute	Overloaded. Overridden. Writes the start of an attribute.
public methodWriteStartDocument	Overloaded. Overridden. Writes the XML declaration with the version "1.0".
public methodWriteStartElement	Overloaded. Overridden. Writes the specified start tag.
public methodWriteString	Overridden. Writes the given text content.
public methodWriteWhitespace	Overridden. Writes out the given white space.

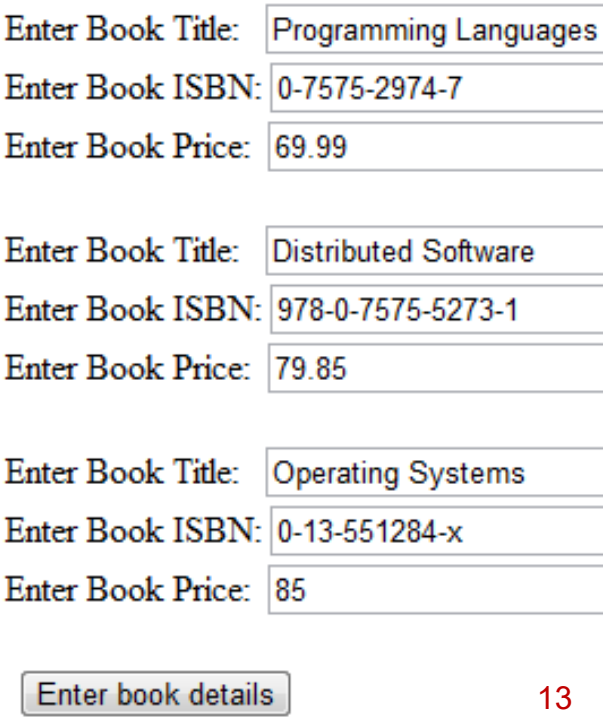
Example: Writing web data into XML File

```
using System.Xml;
using System.IO;
public partial class Seller : System.Web.UI.Page  {
    protected void Page_Load(object sender, EventArgs e) { }
    protected void btnEnterBook_Click(object sender, EventArgs e)  {
        string title1 = txtTitle1.Text;
        string isbn1 = txtIsbn1.Text;
        string sPrice1 = txtPrice1.Text;

        string title2 = txtTitle2.Text;
        string isbn2 = txtIsbn2.Text;
        string sPrice2 = txtPrice2.Text;

        string title3 = txtTitle3.Text;
        string isbn3 = txtIsbn3.Text;
        string sPrice3 = txtPrice3.Text;
```

Taking data from text boxes



Code Behind the Data Enter (Writer) Page

```
string fLocation = Path.Combine(Request.PhysicalApplicationPath,
      @"App_Data\Book.xml");   // or: HttpRuntime.AppDomainAppPath
if (File.Exists(fLocation)) {
      File.Delete(fLocation);
}
FileStream fState = null;
try {
fState = new FileStream(fLocation, FileMode.CreateNew);
XmlTextWriter writer = new XmlTextWriter(fState,
      System.Text.Encoding.Unicode);
writer.Formatting = Formatting.Indented;
writer.WriteStartDocument();
writer.WriteStartElement("Books");
writer.WriteStartElement("Book");
writer.WriteElementString("Title", title1);
writer.WriteElementString("Isbn", isbn1);
writer.WriteElementString("Price", sPrice1);
writer.WriteEndElement();
```

In this example, delete the file if it exists, and create a new file. Not the best way to do it!

Open text file

Open XML file

Other modes include OpenOrCreate, Append, …

Code Behind the Data Enter (Writer) Page

```
    writer.WriteStartElement("Book");
    writer.WriteElementString("Title", title2);
    writer.WriteElementString("Isbn", isbn2);
    writer.WriteElementString("Price", sPrice2);
    writer.WriteEndElement();
    writer.WriteStartElement("Book");
    writer.WriteElementString("Title", title3);
    writer.WriteElementString("Isbn", isbn3);
    writer.WriteElementString("Price", sPrice3);
    writer.WriteEndElement();
    writer.WriteEndElement();
    writer.WriteEndDocument();
    writer.Close();
    fState.Close();
}
```

XMLWriter continues to write

```
<?xml version="1.0" encoding="utf-16"?>
<Books>
  <Book>
    <Title>Programming Languages</Title>
    <Isbn>0-7575-2974-7</Isbn>
    <Price>69.99</Price>
  </Book>
  <Book>
    <Title>Distributed Software</Title>
    <Isbn>978-0-7575-5273-1</Isbn>
    <Price>79.85</Price>
  </Book>
  <Book>
    <Title>Operating Systems</Title>
    <Isbn>0-13-551284-x</Isbn>
    <Price>85</Price>
  </Book>
</Books>
```

It is necessary to close the XMLWriter and to close the file stream connection. You cannot open the file if any one is not closed.

Code Behind the Data Enter (Writer) Page

```
    finally  {
          // writer.Close();
          // fState.Close();
    }

    Response.Redirect("Default.aspx");
}
    protected void txtIsbn_TextChanged(object sender, EventArgs e)
    {
          // can write an event handler to do something as user types
    }
}
```

You could put them here, in case the session crashes, it allows other sessions to access the file

Making text suggestion/recommendation here

Another way of creating an XML File: Combining XmlDocument Class and XMLWriter

- Saves all the children of the **XmlDocument** node through a specified XmlWriter in the following steps:
- Load an XML file into memory as a tree:

```
XmlDocument xd = new XmlDocument();
xd.Load("Courses.xml");
```

- Modify the tree, using tree operations:
  - insertion(), deletion(), balancing(), traversing(), etc.
- Write the modified tree back using

```
XmlTextWriter writer = new XmlTextWriter(Courses.xml);
writer.Formatting = Formatting.Indented;
xd.WriteContentTo(writer);
```


Ira A. Fulton Schools of
Engineering
Arizona State University

M8 L4
XPath:
XML Path Language

Ira A. Fulton Schools of Engineering
Arizona State University

Lecture Outline

- Path Concepts and Basics
- XPath Classes
- XQuery

XML Path vs File System Path

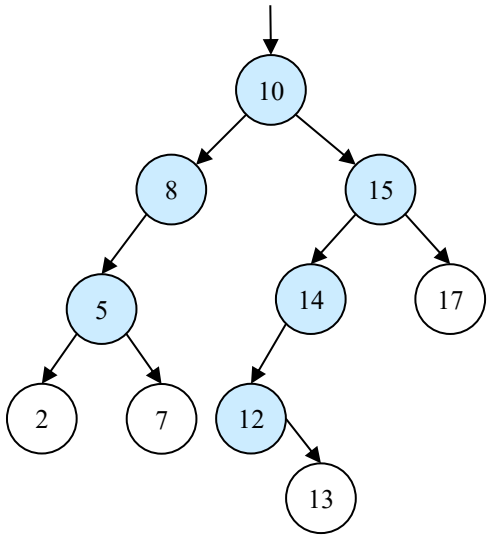
- XPath, short for *XML Path Language,* is a language for accessing parts of an XML document.
- In a file system, a path

  \Courses\CSE445\Assignments

  identifies the Assignments subdirectory of the directory's CSE445, which is a subdirectory of the root directory Courses.
- In an XML document, /Courses/Course identifies all elements named *Course* that are children of the root element *Courses*.
- "/Courses/Course" is an XPath expression.
- XPath expressions are fully described in the XPath specification found at *http://www.w3.org/TR/xpath*.

XPath Basics

- **Nodes**: In XPath, a document is represented as a tree of nodes. A node can be an element, an attribute, or content which is text, and thus, we have
  - Element nodes
  - Attribute nodes
  - Text nodes, the content text of an element
- An XPath **expression** is a path from one part of an XML document to another part of the document;
- An expression consists of a sequence of nodes, **functions**, function return **values**, and **variables** (which hold values of nodes and functions).
- The value at the termination point of an expression is the value of expression.
- The value of an expression can be one of the four XPath data types:
  - Node-set
  - Boolean
  - Number: including both integer and floating-point numbers
  - String of Unicode characters



XPath Expressions

- The most common type of expression is the *location path.*
- /Courses/Course

  which evaluates to all Course elements that are children of a root element named *Courses.*
- /Courses/Course/@Image

  which evaluates to the attribute (not element) named *Image* that belongs to *Course* elements, which in turn are children of the root element *Courses*
- The double-slash expression evaluates to all *Course* elements anywhere in the document:

  //Course
- The // prefix is extremely useful for locating elements in a document regardless of where they are positioned, e.g., //Deanslist
- XPath also supports wildcards.

  /Courses/* Select all child elements of the element named Courses;

  /Courses /Course/@* Select all attributes belong to Course.
- Use "." and ".." for current and parent directory, respectively, just like DOS / UNIX.

XPath Functions (Incomplete List)

- Node-set Functions
  - *number count(node-set);* // Return the number of items in the node-set
  - *node-set id(arg);* // Select elements (node set) by the unique ID of arg.
  - *number position();* // Returns the index of the node within the parent.
- String Functions
  - *number string-length(string);* // Return string length
  - *string concat(string, string, string*);* // Returns the concatenation of multiple strings.
  - *string substring(string, number, number?);* // Returns substring starting at the position specified in 2nd argument and the length specified in the 3rd argument. If 3rd argument not specified, it returns the substring to the end.
- Boolean Functions
  - *boolean(arg);* // convert arg to boolean, e.g., If arg is a non-empty node-set, it is converted to true, otherwise, to false.
  - *not(bool arg);* // Returns true if argument is false, otherwise false.

Apply XPath for Extracting Data

- Expressions are the building blocks of XPath programs.
- XPath can be used to extract data from an XML file.
- Used in this way, XPath becomes a query language — the XML equivalent of SQL.
- The W3C has developed an official XML query language called XQuery (http://www.w3.org/TR/xquery/)
- XPath is a part of the more powerful XQuery language.

XPath Classes Using Framework Class Library

- *System.Xml.XPath* namespace contains classes for putting XPath to work in managed applications:
  - *XPathDocument* creates an XPath document from an XML documents; which is based on DOM model: The entire XML file is loaded into memory for queries. XQuery does not load the entire document.
  - *XPathNavigator* provides a mechanism for performing XPath queries;
  - *XPathNodeIterator* represents node sets generated by XPath queries and lets you iterate over them
- XML Database implement XPath in DB, instead of in memory.

XPathDocument Class

- Step 1 in performing XPath queries on XML documents is to create an *XPathDocument* wrapping the XML document itself;

  XPathDocument dx = new XPathDocument ("Courses.xml");

  http://venus.sod.asu.edu/WSRepository/xml/Courses.xml

- Step 2 is to create an *XPathNavigator* object:

  XPathNavigator nav = dx.CreateNavigator ();

  which returns a set of nodes.

- Step 3 will select nodes meeting given criteria.

XPathNodeIterator Class

- Step 4 is to execute the query using different methods:
  - *Evaluate* executes any XPath expression. It returns a generic *Object* that can be a string, a number, a bool, or an *XPathNodeIterator,* depending on the expression and the type of data that it returns.
  - *Select* works with expressions that return node sets. It always returns an *XPathNodeIterator* representing an XPath node set. The following statement uses *Select* to create a node set representing all nodes that match the expression "//Course":

```
XPathNodeIterator iterator = nav.Select ("//Course");
Console.WriteLine("Select finds {0}, nodes", iterator.Count);
```

  - *Current* property exposes an *XPathNavigator* object that represents the current node
  - A family of Move methods that you can call to move any direction: up, down, or sideways.

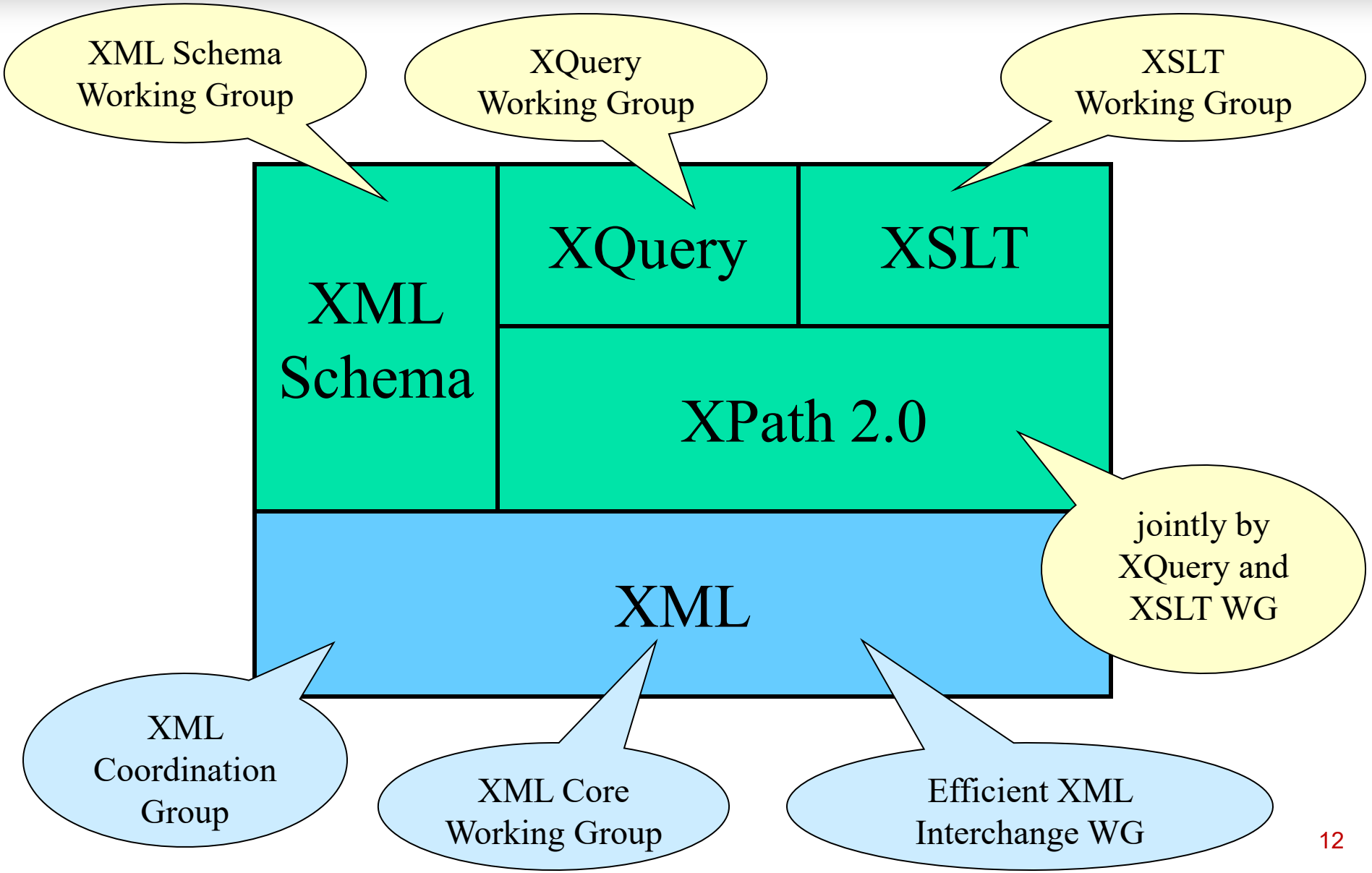
XPathDemo.cs

- XPathDocument is based on DOM model.
- Can it be based on SAX model?

```
class MyApp {
    static void Main () {
        XPathDocument dx = new XPathDocument ("Courses.xml");
        XPathNavigator nav = dx.CreateNavigator ();
        XPathNodeIterator iterator = nav.Select ("/Courses/Course");
        while (iterator.MoveNext ()) {
            XPathNodeIterator it = iterator.Current.Select ("Name");
            it.MoveNext ();
            string courseName = it.Current.Value;
            it = iterator.Current.Select ("Code");
            it.MoveNext ();
            string courseCode = it.Current.Value;
            Console.WriteLine ("{0} {1}", courseName, courseCode);
        }
    }
}
```

See text section 4.3 for full working code

XQuery Standard is based on Other XML Standards



Programming the Queries in XQuery

```
<html>
{
    let $d := doc("Courses.xml")/Courses
    for $b in $d/Course
        where $b/Level > 200
        order by $b/Code
        return <Course>
                    {$b/Name, $b/Level, $b/Cap}
               </Course>
}
</html>
```

SQL syntax

- from
- where
- orderby
- select

Oracle XQuery: Retrieve Info

```
SELECT xmlquery(
         '<POSummary lineItemCount="{count($XML/PurchaseOrder/LineItems/LineItem)}">
         {
            $XML/PurchaseOrder/User,
            $XML/PurchaseOrder/Requestor,
            $XML/PurchaseOrder/LineItems/LineItem[2]
         }
         </POSummary>'
         passing object_value AS "XML"
         returning content
       ).getclobval() initial_state
FROM   PURCHASEORDER
WHERE  xmlExists(
         '$XML/PurchaseOrder[Reference=$REF]'
          passing object_value AS "XML",
                  'ACABRIO-100PDT' AS "REF"
       )
/
```

Oracle: Deleting a node using XQuery update

```
UPDATE PURCHASEORDER
SET    object_value = XMLQuery(
       'copy $NEWXML := $XML modify (
           delete nodes $NEWXML/PurchaseOrder/LineItems/LineItem[@ItemNumber=$ITEMNO]
          )
        return $NEWXML'
       passing object_value as "XML", 2 as ITEMNO
       returning CONTENT
     )
WHERE xmlExists(
       '$XML/PurchaseOrder[Reference=$REF]'
        passing object_value as "XML",
                'ACABRIO-100PDT' as "REF"
     )
/
```


Ira A. Fulton Schools of
Engineering
Arizona State University

M8 L5
XML Processing in Java

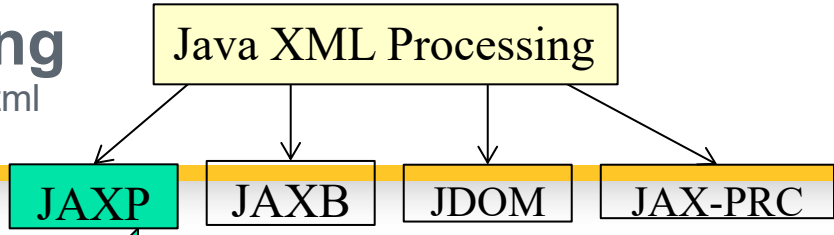
Ira A. Fulton Schools of Engineering
Arizona State University

Lecture Outline

- Java Packages for XML Processing
- Applying Design Patters
- Java SAX Reader and Event Handlers
- Java DOM Builder and Event Handlers

Java Packages for XML Processing

https://www.oracle.com/java/technologies/jaxp-introduction.html



- **JAXP: Java API for XML Processing**

  provides a common (vendor-independent) interface for creating and using the standard SAX, DOM, and XSLT APIs

- **JAXB: Java Architecture for XML Binding**

  defines a mechanism for creating Java objects as XML (*marshalling*: use in other applications), and for creating Java objects from such structures (*unmarshalling*: use those objects in your own application).

- **JDOM: Java DOM**

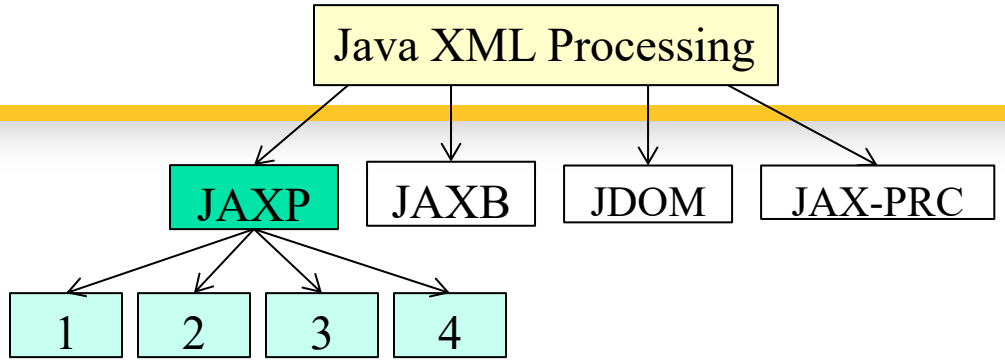
  creates a tree of *objects* from an XML document.

- **JAX-RPC: API for XML-based Remote Process Calls**

  provides a mechanism for publishing available services in an external registry, and for consulting the registry to find those services.

Remotable and non-remotable

JAXP Packages



There are four packages:

- **javax.xml.parsers**

  Provide common interfaces for different vendors' SAX and DOM parsers.

- **org.w3c.dom**

  Defines the Document class (a DOM), as well as classes for all of the components of a DOM.

- **org.xml.sax**

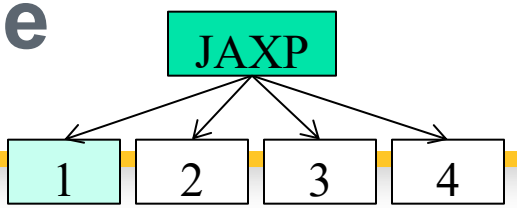
  Defines the basic SAX APIs.

- **javax.xml.transform**

  Defines the XSLT APIs that let you transform XML into other forms, e.g., HTML table, PDF. (XSLT in Lecture 17)

JAXP javax.xml.parsers Package

http://java.sun.com/j2se/1.5.0/docs/api/



It provides four patterns/classes for processing of XML documents

- SAXParserFactory

  A design pattern

  Defines a factory API that enables applications to configure and obtain a SAX-based parser to parse XML documents.

- SAXParser

  A class in the design pattern

  Defines the API that wraps an XMLReader implementation class.

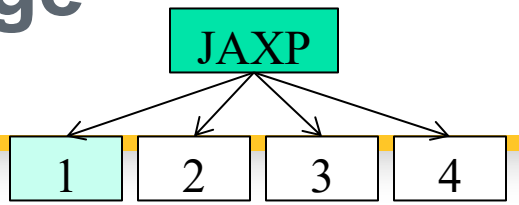
- DocumentBuilderFactory

  Defines a factory API that enables applications to obtain a parser that produces DOM object trees from XML documents.

- DocumentBuilder

  Defines the API to obtain DOM Document instances from an XML document.

Use JAXP javax.xml.parsers Package Example



A specific combination of factory, interface, class, and methods

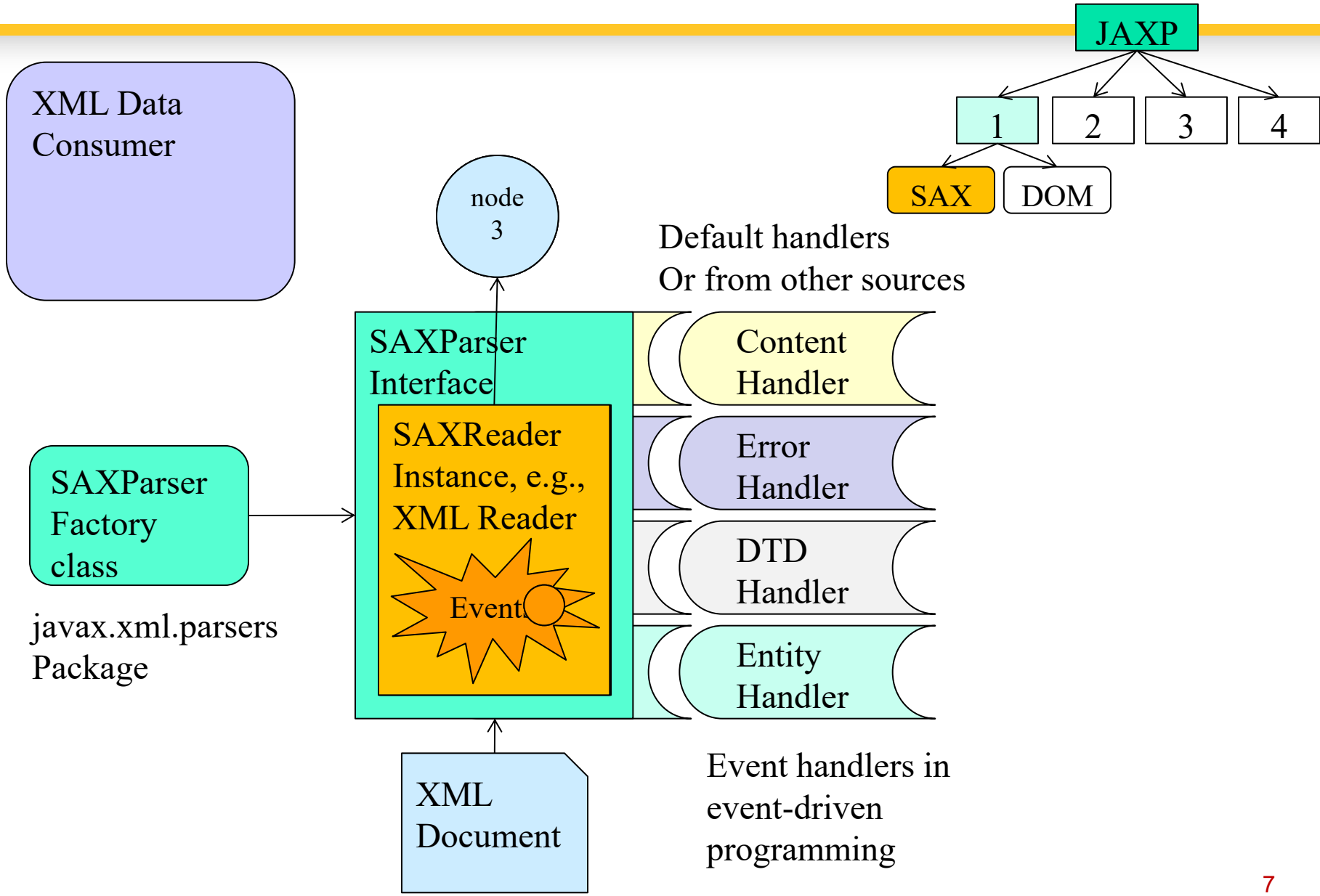
- Use SAXParserFactory class to create an instance (object)

  It is an interface, which defines several kinds of parser() methods without the implementations
- SAXReader: This is a specific SAXParser, which wraps a XMLReader. SAXReader carries on the conversation with the SAX event handlers.
- Handlers

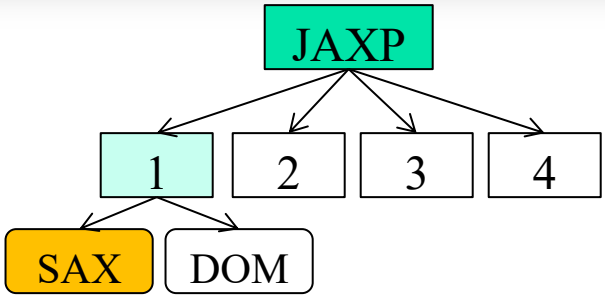
  You can pass a set of handlers to the interface to provide the implementations.
- DefaultHandlers

  implement the ContentHandler, ErrorHandler, DTDHandler, and EntityResolver interfaces

Use the Classes to Process XML Document



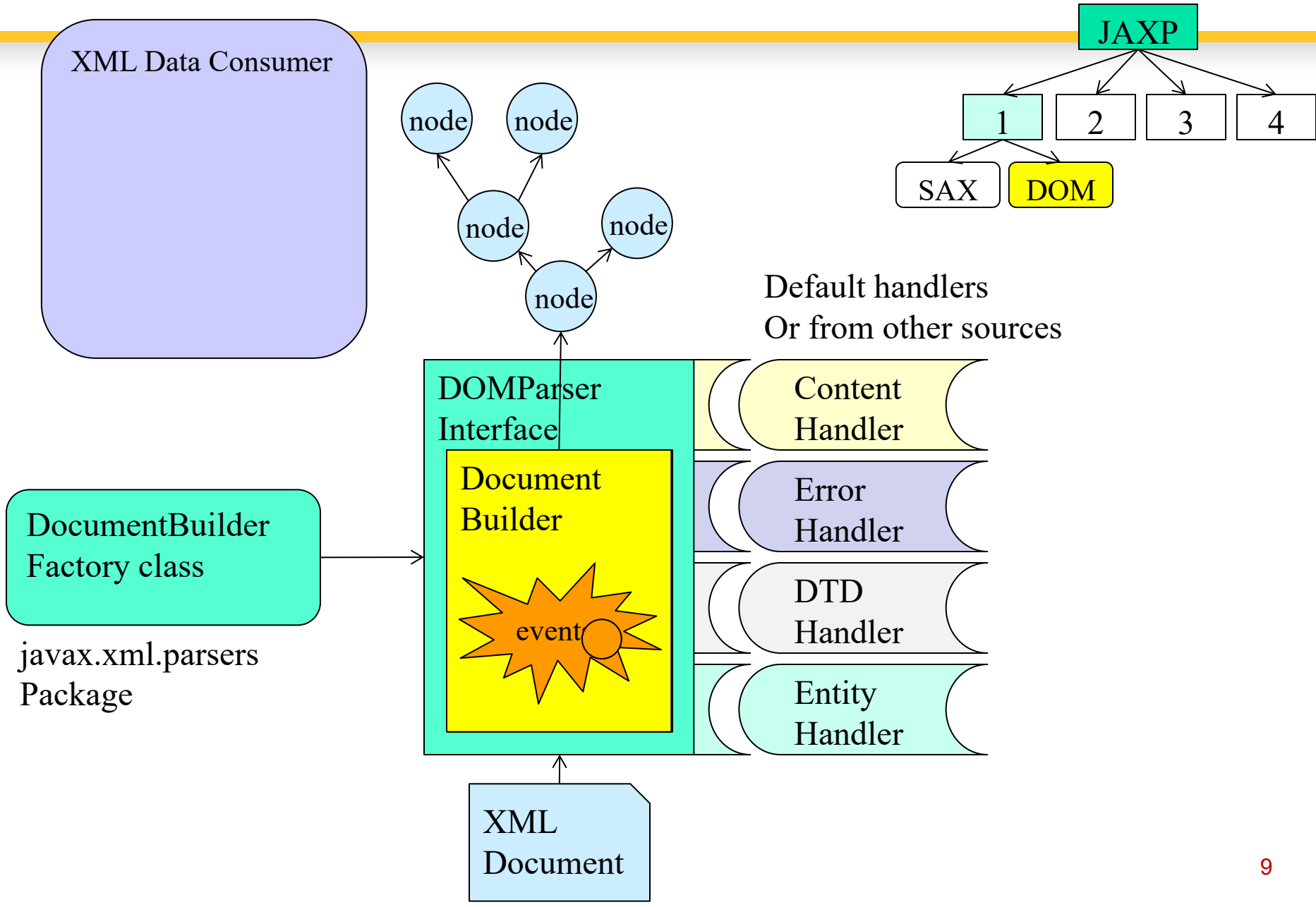
XML SAX Reader: org.xml.sax (Java)




```
import java.io.FileReader;
import org.xml.sax.XMLReader;
import org.xml.sax.InputSource;
import org.xml.sax.helpers.XMLReaderFactory;
import org.xml.sax.helpers.DefaultHandler;
public class MySAXApp extends DefaultHandler  {
      public MySAXApp () { super(); }// constructor
      public static void main (String args[]) throws Exception  {    // command inputs
            XMLReader xr = XMLReaderFactory.createXMLReader();
            MySAXApp handler = new MySAXApp();
            xr.setContentHandler(handler);       // add content hander
            xr.setErrorHandler(handler);          // add error hander
            // Parse each file provided via the command line input
            for (int i = 0; i < args.length; i++) {          // get the number of file names
                  FileReader r = new FileReader(args[i]);  // Open a file reader object
                  xr.parse(new InputSource(r));          // parse the file opened
      }     }     }
```

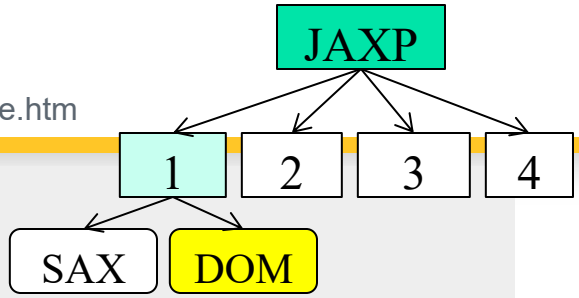
No need to call handlers. Called when an event occurs.

DocumentBuilderFactory & DocumentBuilder



Example

https://www.tutorialspoint.com/java/xml/javax_xml_parsers_documentbuilder_inputsource.htm




```
public class DocumentBuilderDemo {
      public static void main(String[] args) {
            // create a new DocumentBuilderFactory
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            try { // use the factory to create a documentbuilder
                  DocumentBuilder builder = factory.newDocumentBuilder();
                  // create a new document from input source
                  FileInputStream fis = new FileInputStream("Courses.xml");
                  InputSource is = new InputSource(fis);
                  Document doc = builder.parse(is);  // Create an in-memory tree
                  Element element = doc.getDocumentElement(); // get the first element
                  NodeList nodes = element.getChildNodes(); // get all child nodes
                  // print the text content of each child
                  for (int i = 0; i < nodes.getLength(); i++) {
                        System.out.println("" + nodes.item(i).getTextContent());
                  }
            }
            catch (Exception ex) {
                  ex.printStackTrace();
}     }     }
```

There is little difference between C# code and Java code


Ira A. Fulton Schools of
Engineering
Arizona State University