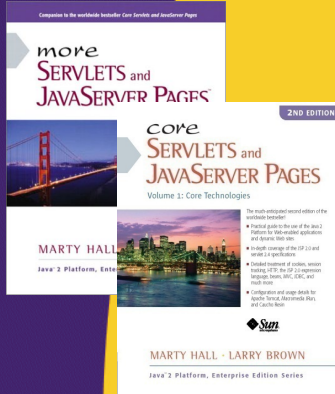




# Creating and Parsing XML Files with DOM

Originals of Slides and Source Code for Examples:  
<http://courses.coreservlets.com/Course-Materials/java.html>

**Customized Java EE Training:** <http://courses.coreservlets.com/>  
Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.



**For live Java EE training, please see training courses at <http://courses.coreservlets.com/>.**

**JSF 2, PrimeFaces, Servlets, JSP, Ajax (with jQuery), GWT, Android development, Java 6 and 7 programming, SOAP-based and RESTful Web Services, Spring, Hibernate/JPA, XML, Hadoop, and customized combinations of topics.**



**Taught by the author of *Core Servlets and JSP*, *More Servlets and JSP*, and this tutorial. Available at public venues, or customized versions can be held on-site at your organization. Contact [hall@coreservlets.com](mailto:hall@coreservlets.com) for details.**

# Agenda

- **Options for input files**
- **XML overview**
- **Comparing XML with HTML**
- **Parsing an XML document**
  - Creating a DocumentFactory and Document
- **Extracting data from parsed Document**
  - Known structure, attribute values only
  - Known structure, attribute values and body content
  - Unknown structure
- **Specifying grammars with DTDs**

4

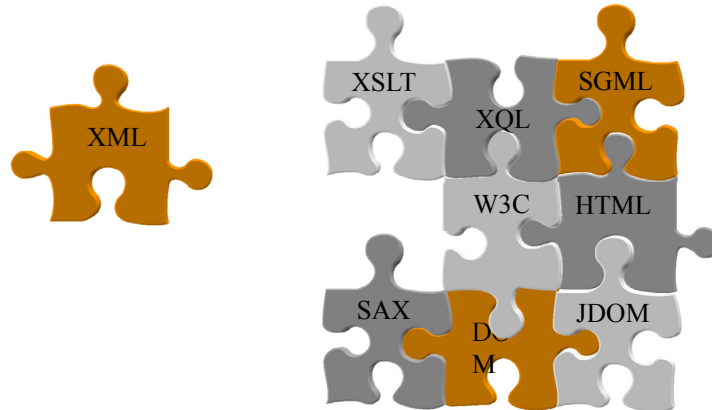
# Options for Data Input Files

- **Properties files**
  - Simple names and values
    - Pros: human editable, very easy to create, one line to create from Java code, one line to read in
    - Cons: strings only, no structure, no validation
- **Regular files**
  - File with Scanner, String.split, StringTokenizer, regular expressions
    - Pros: human editable, CPU and memory efficiency, arbitrary data types, arbitrary structure
    - Cons: lots of programming, inflexible and hard to maintain, no validation
- **Serialized data structures**
  - File object with ObjectInputStream/ObjectOutputStream
    - Pros: very CPU and memory efficient, very flexible, extremely low programmer effort
    - Cons: program-created data only: not human editable
- **XML Files (topic of this section)**
  - Pros: human editable, structured representation, validation
  - Cons: inefficient file usage, moderate programmer effort

5

# XML Overview

- When people refer to XML, they typically are referring to XML and related technologies



6

# XML Resources

- **Java API Docs**
  - <http://java.sun.com/j2se/1.5.0/docs/api/>
  - <http://java.sun.com/j2se/1.5.0/docs/api/org/w3c/dom/Node.html>
  - <http://java.sun.com/j2se/1.5.0/docs/api/org/w3c/dom/Element.html>
- **XML 1.0 Specification**
  - <http://www.w3.org/TR/REC-xml>
- **WWW consortium's Home Page on XML**
  - <http://www.w3.org/XML/>
- **Sun Page on XML and Java**
  - <http://java.sun.com/xml/>
- **O'Reilly XML Resource Center**
  - <http://www.xml.com/>

7

## XML Overview

- **EX**tensible **M**arkup **L**anguage (XML) is a meta-language that describes the content of the document (self-describing data)
  - Java = Portable Programs
  - XML = Portable Data
- **XML does not specify the tag set or grammar of the language**
  - Tag Set – markup tags that have meaning to a language processor
  - Grammar – rules that define correct usage of a language's tags

8

## Applications of XML

- **Configuration files**
  - Used extensively in J2EE architectures
- **Media for data interchange**
  - A better alternative to proprietary data formats
- **B2B transactions on the Web**
  - Electronic business orders (ebXML)
  - Financial Exchange (IFX)
  - Messaging exchange (SOAP)

9

## HTML vs. XML

- **HTML is a specific language**
  - XML is a family of languages
- **HTML has container tags and standalone tags**
  - `<H1>...</H1>`
  - `<BR>` `<IMG SRC="...">` `<HR>`
  - XML only has container tags
    - `<blah>...</blah>` or `<blah/>`
- **HTML tag and element names are not case sensitive**
  - `<H1 ALIGN="...">` or `<h1 aLiGn="...">`
  - XML tag and element names are always case sensitive
    - Uppercase, lowercase, or mixed case are all legal
      - `<someTag/>` not the same as `<sometag/>`

10

## HTML vs. XML (Continued)

- **HTML clients tolerate syntax errors**
  - XML parsers insist on *well-formed* XML (legal low-level syntax)
    - Illegal: `<foo<` `<foo><bar>...</foo></bar>` `<blah att="huh">`
  - Parsers may or may not insist on *valid* documents (documents that follow grammar rules as given in a DTD or Schema)
    - Rules might specify `<bar>` can only appear inside `<foo>...</foo>`
- **HTML does not require a single root element**
  - `<HEAD>...</HEAD><BODY>...</BODY>` is legal
  - XML requires one main element enclosing entire document
    - `<?xml version="1.0"..."?><mainElement>...</mainElement>`
- **HTML does not require quotes around attribute values**
  - `<H1 ALIGN=CENTER>...</H1>`
  - XML requires single or double quotes
    - `<someTag someAttribute="some value"/>`
    - `<foo att1="bar" att2='baz'>...</foo>`

11

## Simple XML Example (Optional Parts in Italics)

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE ...>
<mainElement someAttribute="something">
  <subElement1 blah="some value">
    <subSubElement>foo</subSubElement>
    <subSubElement>bar</subSubElement>
  </subElement1>
  <subElement2>
    Yadda yadda yadda
  </subElement2>
  ...
</mainElement>
```

12

## Parsing XML: DOM and SAX

- **DOM (Document Object Model)**
  - Parses entire document
  - Represents result as a tree
  - Lets you search tree
  - Lets you modify tree
  - Good for reading data/configuration files
- **SAX**
  - Parses until you tell it to stop
  - Fires event handlers for each:
    - Start tag
    - Tag body
    - End tag
  - Low level
  - Good for very large documents, especially if you only care about very small portions of the document

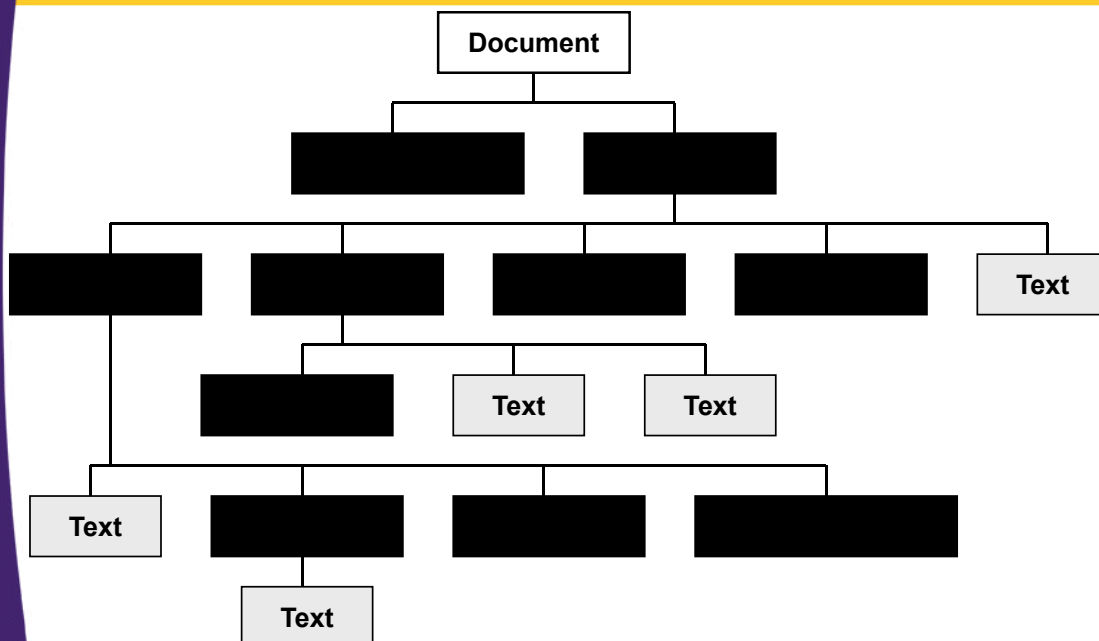
13

# Document Object Model (DOM)

- **DOM supports navigating and modifying XML documents**
  - Hierarchical tree representation of document
    - Tree follows standard API
    - Creating tree is vendor specific
- **DOM is a language-neutral specification**
  - Bindings exists for Java, C++, CORBA, JavaScript, C#
    - Bad news: methods do not precisely follow usual Java naming convention
    - Good news: you can switch to other languages (e.g., JavaScript and Ajax) with minimal learning curve
- **Official Website for DOM**
  - <http://www.w3c.org/DOM/>

14

# DOM Tree



15

## Steps to Using DOM: Creating a Parsed Document

### 1. Import XML-related packages

```
import org.w3c.dom.*;  
import javax.xml.parsers.*;  
import java.io.*;
```

### 2. Create a DocumentBuilder

```
DocumentBuilderFactory factory =  
    DocumentBuilderFactory.newInstance();  
DocumentBuilder builder =  
    factory.newDocumentBuilder();
```

### 3. Create a Document from a file or stream

```
Document document =  
    builder.parse(new File(file));
```

16

## Steps to Using DOM: Extracting Data from Parsed Document

### 4. Extract the root element

```
Element root =  
    document.getDocumentElement();
```

### 5. Examine attributes

- `getAttribute("attributeName")` returns specific attribute
- `getAttributes()` returns a Map (table) of names/values

### 6. Examine sub-elements

- `getElementsByTagName("subelementName")` returns a list of subelements of specified name
- `getChildNodes()` returns a list of all child nodes
  - Both methods return data structure containing Node entries, not Element.
    - » Node is parent interface of Element
    - » Results of `getElementsByTagName` can be typecast to Element
    - » Results of `getChildNodes` are Node entries of various types (see later slides)

17



## Example: Extracting Simple Top-Level Information

- Look at root element only
- Assume attribute names known in advance

```
<?xml version="1.0" encoding="utf-8"?>
<company name="... Applied Physics Laboratory"
        shortName="JHU/APL"
        mission="Enhancing national security ...">
  <head name="Richard Roca" phone="410-778-1234"/>
  <department name="Air and Missile Defense"
            mission="Enhance the operational ...">
    <group name="A1E" numStaff="20"/>
    <group name="A1F" numStaff="15"/>
    <group name="A1G" numStaff="25"/>
  </department>
  <department name="Research and Technology..."
            mission="Assure that ...">
    <group name="RSI" numStaff="15"/>
    <group name="RSS" numStaff="10"/>
  </department>
</company>
```

18

## Example: Source Code

```
import org.w3c.dom.*;
import javax.xml.parsers.*;
import java.io.*;

public class DomTest1 {
  public static void main(String[] args) {
    String file = "test1.xml";
    if (args.length > 0) {
      file = args[0];
    }
  }
}
```

19

## Example: Source Code (Continued)

```
try {
    DocumentBuilderFactory factory =
        DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document document = builder.parse(new File(file));
    Element root = document.getDocumentElement();
    System.out.println(root.getTagName());
    System.out.printf("  name: %s%n",
        root.getAttribute("name"));
    System.out.printf("  short name: %s%n",
        root.getAttribute("shortName"));
    System.out.printf("  mission: %s%n",
        root.getAttribute("mission"));
} catch (Exception e) {
    e.printStackTrace();
}
}
```

20

## Example: Results (Indentation of Long Lines Added)

```
> java DomTest1 test1.xml
company
  name: The Johns Hopkins University
        Applied Physics Laboratory
  short name: JHU/APL
  mission: Enhancing national security
            through science and technology
```

21

## Examining Children with `getElementsByTagName`

- **Supply tag name as argument**
  - Tag names are case sensitive
    - `someElement.getElementsByTagName("blah")` not same as `someElement.getElementsByTagName("Blah")`
  - "\*" allowed as argument to match all child tags
  - Returns a `NodeList`
- **NodeList has two methods**
  - `getLength`: number of child elements
  - `item(index)`: element at given index
    - `item` returns `Node` (parent interface of `Element`), so you should do a typecast on return value

22

## Example 2: Examining Child Elements

- Use `getElementsByTagName`
- Final values are in attributes, not tag bodies

```
<?xml version="1.0" encoding="utf-8"?>
<company name="... Applied Physics Laboratory"
  shortName="JHU/APL"
  mission="Enhancing national security ...">
  <head name="Richard Roca" phone="410-778-1234"/>
  <department name="Air and Missile Defense"
    mission="Enhance the operational ...">
    <group name="A1E" numStaff="20"/>
    <group name="A1F" numStaff="15"/>
    <group name="A1G" numStaff="25"/>
  </department>
  <department name="Research and Technology..."
    mission="Assure that ...">
    <group name="RSI" numStaff="15"/>
    <group name="RSS" numStaff="10"/>
  </department>
</company>
```

23

## Example 2: Code

- Initial code same as last example

```
try {
    DocumentBuilderFactory factory =
        DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document document = builder.parse(new File(file));
    Element root = document.getDocumentElement();
    System.out.println(root.getTagName());
    System.out.printf("  name: %s%n",
        root.getAttribute("name"));
    System.out.printf("  short name: %s%n",
        root.getAttribute("shortName"));
    System.out.printf("  mission: %s%n",
        root.getAttribute("mission"));
}
```

24

## Example 2: Code (Continued)

```
NodeList departments =
    root.getElementsByTagName("department");
for(int i=0; i<departments.getLength(); i++) {
    Element department = (Element)departments.item(i);
    System.out.println(department.getTagName());
    System.out.printf("  name: %s%n",
        department.getAttribute("name"));
    System.out.printf("  mission: %s%n",
        department.getAttribute("mission"));
    System.out.printf("  staff: %s%n",
        countStaff(department));
}
} catch (Exception e) {
    e.printStackTrace();
}
}
```

25

## Example 2: Code (Continued)

```
public static int countStaff(Element department) {
    int departmentStaff = 0;
    NodeList groups =
        department.getElementsByTagName("group");
    for(int i=0; i<groups.getLength(); i++) {
        Element group = (Element)groups.item(i);
        int groupStaff =
            toInt(group.getAttribute("numStaff"));
        departmentStaff = departmentStaff + groupStaff;
    }
    return(departmentStaff);
}
```

26

## Example 2: Results

```
> java DomTest2 test2.xml
company
  name: The ... Applied Physics Laboratory
  short name: JHU/APL
  mission: Enhancing national security through ...
department
  name: Air and Missile Defense
  mission: Enhance the operational capabilities ...
  staff: 60
department
  name: Research and Technology Development Center
  mission: Assure that JHU/APL has ...
  staff: 25
```

27

# Extracting Body Content

- **Simple to get data out of attributes**
  - `<foo someAttribute="some data"/>`
- **Harder to get data out of tag body**
  - `<foo>some data</foo>`
- **Extracting tag body**
  - Normalize tree to merge text nodes
    - `rootElement.normalize()`
  - Get text from node (really from child #text node)
    - String body =  
`someElement.getFirstChild().getNodeValue();`
- **Assumptions**
  - Tags have body content or subelements, but not both
  - You know which elements have body content

28

# Example 3: Printing Lab Descriptions

```
<?xml version="1.0" encoding="utf-8"?>
<company name="... Applied Physics Laboratory"
  shortName="JHU/APL"
  mission="Enhancing national security through science
and technology">
  <head name="Richard Roca" phone="410-778-1234"/>
  <department name="Air and Missile Defense"
    mission="Enhance ...">
    <group name="A1E" numStaff="20"/>
    <group name="A1F" numStaff="15"/>
    <group name="A1G" numStaff="25"/>
    <lab name="Ship Self-Defense System (SSDS) Laboratory">
      This closed facility supports SSDS Mk 1 and Mk 2 Combat
      Systems used for ...
    </lab>
    <lab name="System Concept ... (SCOPE) Laboratory">
      The SCOPE Laboratory is a closed area in which ...
    </lab>
  </department>
  <department name="Research and Technology..." ...>
    ...
  </department>
</company>
```

29

## Example 3: Code

```
...
System.out.println("APL Labs by Department");
Element root = document.getDocumentElement();
// Combine text nodes from multiple lines and
// eliminate empty text nodes
root.normalize();
NodeList departments =
    root.getElementsByTagName("department");
for(int i=0; i<departments.getLength(); i++) {
    Element department = (Element)departments.item(i);
    NodeList labs =
        department.getElementsByTagName("lab");
    for(int j=0; j<labs.getLength(); j++) {
        Element lab = (Element)labs.item(j);
        System.out.printf("  Lab name: %s\n",
            lab.getAttribute("name"));
        String labDescription =
            lab.getFirstChild().getNodeValue();
        System.out.println(labDescription);
    }
}
```

30

## Example 3: Results

```
> java DomTest3 test2.xml
APL Labs by Department
```

```
Lab name: Ship Self-Defense System (SSDS) Laboratory
This closed facility supports SSDS Mk 1 and Mk 2 Combat
Systems used for air defense on several non-Aegis class
ships. The SSDS Laboratory is a simulated shipboard
environment, with sensor simulation and hardware and
software in the loop, which allows new technology or
concepts to be tested. Its uses include data reduction
and analysis of at-sea tests and analysis of Trouble
Reports from the SSDS fleet, along with development
and testing of fixes.
```

```
Lab name: System Concept and Performance Evaluation (SCOPE) Laboratory
The SCOPE Laboratory is a closed area in which system concepts
and designs for Ballistic Missile Defense are developed and
analyzed. Standard Missile-3 discrimination has been a
primary focus. The APL Defended Area Model (ADAM), the
APL Concept Engagement Tool (ACENT), the Ballistic Missile
Localization and Selection Tool (BLAST), and the APL
Area/Theater Engagement Missile/Ship Simulation (ARTEMIS)
and its versions were developed and are hosted in SCOPE.
```

```
Lab name: Phased Array Radar Systems Engineering (PARSE) Facility
The PARSE Laboratory is a secure computer facility used ...
```

31

# Dealing with Unknown DOM Structures

- **Sometimes you need to explore trees**
  - Looking for node or content of a certain name or value
- **Tools**
  - Node.getChildNodes
    - Returns a NodeList of child nodes
      - getLength returns int
      - item(index) returns Node.
        - » But "Node" encompasses a lot. Body content, CDATA sections, and real elements are all considered nodes (#text nodes, #cdata-section nodes, and Element nodes)
  - Node.getNodeName
    - For Element nodes, this is the tag name
      - For other nodes, "name" is #text, #cdata-section, etc.
  - Node.getNodeType
    - Compare to Node.TEXT\_NODE, etc.
  - Node.getNodeValue
    - null for most things except text nodes and CDATA sections
  - Node.getAttributes
    - Map of attributes of the node (null for non-Element nodes)

32

# Example 4: Tracing an Arbitrary Document

```
try {
    DocumentBuilderFactory factory =
        DocumentBuilderFactory.newInstance();
    //factory.setValidating(true);
    //factory.setNamespaceAware(true);
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document document = builder.parse(new File(file));
    Element root = document.getDocumentElement();
    // Combine text nodes from multiple lines and
    // eliminate empty text nodes
    root.normalize();
    printNode(root, 0);
} catch (Exception e) {
    e.printStackTrace();
}
}
```

33



## Example 4: Tracing an Arbitrary Document (Continued)

```
// Node is the parent class of Element

public static void printNode(Node node, int depth) {
    String prefix = padChars(2*depth, " ");
    if (node.getNodeType() == Node.TEXT_NODE) {
        System.out.printf("%s%n", prefix, node.getNodeValue());
    } else {
        NamedNodeMap attributes = node.getAttributes();
        if ((attributes == null) || (attributes.getLength() == 0)) {
            System.out.printf("%s%n",
                prefix, node.getNodeName());
        } else {
            System.out.printf("%s [%", // No newline
                prefix, node.getNodeName());
            printAttributes(attributes);
            System.out.println(" ]");
        }
    }
    NodeList children = node.getChildNodes();
    for(int i=0; i<children.getLength(); i++) {
        Node childNode = children.item(i);
        printNode(childNode, depth+1);
    }
}
```

34

## Example 4: Tracing an Arbitrary Document (Continued)

```
private static void printAttributes(NamedNodeMap attributes) {
    for(int i=0; i<attributes.getLength(); i++) {
        Node attribute = attributes.item(i);
        System.out.printf(" %s=\"%s\"", attribute.getNodeName(),
            attribute.getNodeValue());
    }
}
```

35

## Example 4: Tracing an Arbitrary Document (Indented Results)

```
> java DomTest4 test2.xml
company [ mission="Enhancing national security through science ..."
          name="The Johns Hopkins University Applied Physics Laboratory"
          shortName="JHU/APL" ]

head [ name="Richard Roca" phone="410-778-1234" ]

department [ mission="Enhance the operational capabilities of ..."
              name="Air and Missile Defense" ]

group [ name="A1E" numStaff="20" ]

group [ name="A1F" numStaff="15" ]

group [ name="A1G" numStaff="25" ]

lab [ name="Ship Self-Defense System (SSDS) Laboratory" ]

This closed facility supports SSDS Mk 1 and Mk 2 Combat
Systems used for air defense on several non-Aegis class
ships. The SSDS Laboratory is a simulated shipboard
environment, with sensor simulation and hardware and
```

36

## Specifying Grammar: Document Type Definition (DTD)

- **Defines Structure of the Document**
  - Allowable tags and their attributes
  - Constraints on attribute values
  - Tag nesting order
  - Number of occurrences of tags
  - Entity definitions

37

# DTD Example

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!ELEMENT perennials (daylily)*>
<!ELEMENT daylily (cultivar, award*, bloom, cost)+>
<!ATTLIST daylily
    status (in-stock | limited | sold-out) #REQUIRED>
<!ELEMENT cultivar (#PCDATA)>
<!ELEMENT award (name, year)>
<!ELEMENT name (#PCDATA)>
<!ATTLIST name note CDATA #IMPLIED>
<!ELEMENT year (#PCDATA)>
<!ELEMENT bloom (#PCDATA)>
<!ATTLIST bloom code (E | EM | M | ML | L | E-L) #REQUIRED>
<!ELEMENT cost (#PCDATA)>
<!ATTLIST cost discount CDATA #IMPLIED>
<!ATTLIST cost currency (US | UK | CAN) "US">
```

38

# Defining Elements

- **<!ELEMENT name definition/type>**

```
<!ELEMENT daylily (cultivar, award*, bloom, cost)+>
<!ELEMENT cultivar (#PCDATA)>
<!ELEMENT id (#PCDATA | catalog_id)>
```

- **Types**

- ANY Any well-formed XML data
- EMPTY Element cannot contain any text or child elements
- PCDATA Character data only (should not contain markup)
- elements List of legal child elements (no character data)
- mixed May contain character data and/or child elements (cannot constrain order and number of child elements)

39

## Defining Elements, cont.

- **Cardinality**

- [none] Default (one and only one instance)
- ? 0, 1
- \* 0, 1, ..., N
- + 1, 2, ..., N

- **List Operators**

- , Sequence (in order)
- | Choice (one of several)

40

## Grouping Elements

- **Set of elements can be grouped within parentheses**

- (Elem1?, Elem2?)+
  - Elem1 can occur 0 or 1 times followed by 0 or 1 occurrences of Elem2
  - The group (sequence) must occur 1 or more times

- **OR**

- ((Elem1, Elem2) | Elem3)\*
  - Either the group of Elem1, Elem2 is present (in order) or Elem3 is present, 0 or more times

41

# Element Example

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE Person [
  <!ELEMENT Person ( (Mr|Ms|Miss)?, FirstName,
    MiddleName*, LastName, (Jr|Sr)? )>
  <!ELEMENT FirstName (#PCDATA)>
  <!ELEMENT MiddleName (#PCDATA)>
  <!ELEMENT LastName (#PCDATA)>
  <!ELEMENT Mr EMPTY>
  <!ELEMENT Ms EMPTY>
  ...
  <!ELEMENT Sr EMPTY>
]>
<Person>
  <Mr/>
  <FirstName>Lawrence</FirstName>
  <LastName>Brown</LastName>
</Person>
```

42

# Defining Attributes

- **<!ATTLIST element attrName type modifier>**

- **Examples**

```
<!ELEMENT Customer (#PCDATA )>
<!ATTLIST Customer id CDATA #IMPLIED>

<!ELEMENT Product (#PCDATA )>
<!ATTLIST Product
  cost CDATA #FIXED "200"
  id CDATA #REQUIRED>
```

43

# Attribute Types

- **CDATA**
  - Essentially anything; simply unparsed data  
`<!ATTLIST Customer id CDATA #IMPLIED>`
- **Enumeration**
  - attribute (value1|value2|value3) [Modifier]
- **Eight other attribute types**
  - ID, IDREF, NMTOKEN, NMTOKENS, ENTITY, ENTITIES, NOTATION

44

# Attribute Modifiers

- **#IMPLIED**
  - Attribute is not required  
`<!ATTLIST cost discount CDATA #IMPLIED>`
- **#REQUIRED**
  - Attribute must be present  
`<!ATTLIST account balance CDATA #REQUIRED>`
- **#FIXED "value"**
  - Attribute is present and always has this value  
`<!ATTLIST interpreter language CDATA #FIXED "EN">`
- **Default value (applies to enumeration)**
  - `<!ATTLIST car color (red | white | blue) "white" >`

45

# Document Entities

- `<!ENTITY name "replacement">`
- **Entities refer to a data item, typically text**
  - General entity references start with `&` and end with `;`
  - The entity reference is replaced by its true value when parsed
  - The characters `<` `>` `&` `"` `'` require entity references to avoid conflicts with the XML application (parser)  
`&lt;` `&gt;` `&amp;` `&quot;` `&apos;`
- Example

```
<?xml version="1.0" standalone="yes" ?>
<!DOCTYPE book [
<!ELEMENT book (title)>
<!ELEMENT title (#PCDATA)>
<!ENTITY COPYRIGHT "2005, Prentice Hall">
]>
<book>
  <title>Core Servlets &amp; JSP, &COPYRIGHT;</title>
</book>
```

46

# Document Entities (Continued)

- **CDATA (character data) is not parsed**
  - `<![CDATA[...]]>`
  - Entities and special tags are allowed

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<server>
  <port status="accept">
    <![CDATA[8001 <= port < 9000]]>
  </port>
</server>
```

47

## Limitations of DTDs

- **DTD itself is not in XML format**
  - More work for parsers
- **Does not express data types**
  - Weak data typing
- **Limited expressibility**
  - Cannot express full regular expressions, much less context-free languages
- **No namespace support**
  - <foo:bar/>
- **Document can override external DTD definitions**
- **XML Schema are more powerful and flexible**
  - DTDs still dominate in industry, however

48

## Summary

- **XML documents**
  - XML prolog then main root element
  - Case sensitive tag and attribute names
  - Closing tag (or embedded /) required
- **Parsing**

```
DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();
DocumentBuilder builder =
    factory.newDocumentBuilder();
Document document = builder.parse(new File(file));
Element root = document.getDocumentElement();
```
- **Extracting data**
  - `getAttribute`, `getElementsByTagName`, `getFirstChild`,  
`getChildNodes`, `getNodeName`, `getNodeValue`

49





# Questions?

JSF 2, PrimeFaces, Java 7, Ajax, jQuery, Hadoop, RESTful Web Services, Android, Spring, Hibernate, Servlets, JSP, GWT, and other Java EE training.

**Customized Java EE Training: <http://courses.coreservlets.com/>**

Java, JSF 2, PrimeFaces, Servlets, JSP, Ajax, jQuery, Spring, Hibernate, RESTful Web Services, Hadoop, Android.  
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.