XML for Java ME Developers

Tuyen Nguyen, PSE

Team Lead

Warm up - Introductions

- Introduce yourself:
  - Your name
  - Your previous project
  - Your experience with Java ME, XML

Course Objectives

- To gain a better understanding of XML, DTD, XSD
- XML Programming Using SAX
- XML Programming Using DOM
- XML Programming Using StAX
- Excercises

Course Outline

- An overview of XML, DTD, XSD
- XML Programming Using SAX
- XML Programming Using DOM
- XML Programming Using StAX
- Excercises

Course Audience and Prerequisite

- The course is for Java ME (Micro Edition) developers who wants to learn about XML and extract XML information
- The following are prerequisites to this course:
  - “Java ME Fundamentals” course

Assessment Disciplines

- Class Participation: 60%
- Assignment (3 Excercises): 40%
- Passing Scores: 70%

Course Duration - 3 hours

- Duration: 3 hours
- Break time: 15 minutes / module
- Total module: 1

Further References

- “The J2EE(TM) 1.6 Tutorial”, Oracle. http://download.oracle.com/javaee/6/tutorial/doc/
- “Extensible Markup Language (XML)”, W3C. http://www.w3.org/XML/
- JAXP project home page. https://jaxp.dev.java.net/
- W3Schools home page. http://www.w3schools.com
- JSR 280 API. http://docs.oracle.com/javame/config/cldc/opt-pkgs/api/xml/jsr280/

Course Administration

- In order to complete the course, you must:
  - Sign in the Class Attendance List
  - Participate in the course
  - Complete your assignments
  - Provide your feedback in the course evaluation

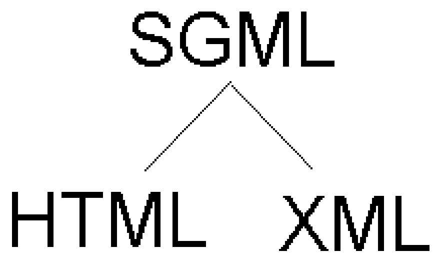
Set Up Environment

- To complete the course, your PC must have:
  - JDK 5+
  - Eclipse Juno IDE or NetBeans version 7.3
  - Java ME SDK 3.2
  - Network connection

An introduction to XML

What is XML?

- *XML- e**X**tensible **M**arkup **L**anguage*
- Markup language for documents containing structured information
- Designed to transport and store data
- Bridge for data exchange on the Web
- Based on Standard Generalized Markup Language (SGML)



Comparisons

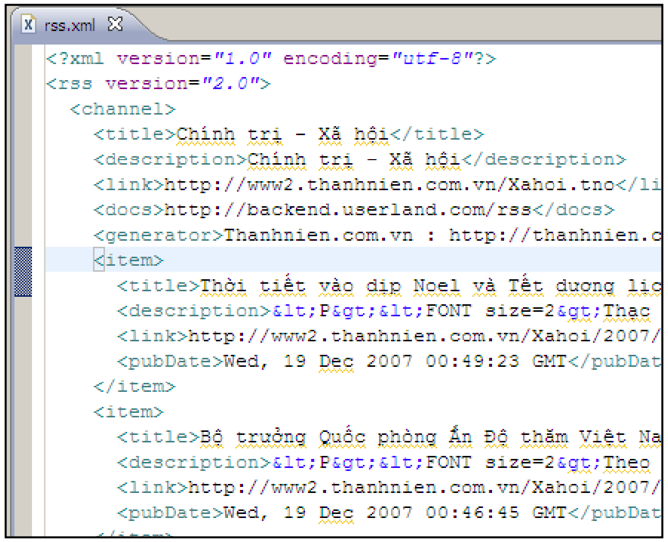
XML

- Extensible set of tags
- Content orientated
- Standard Data infrastructure
- Allows multiple output forms

HTML

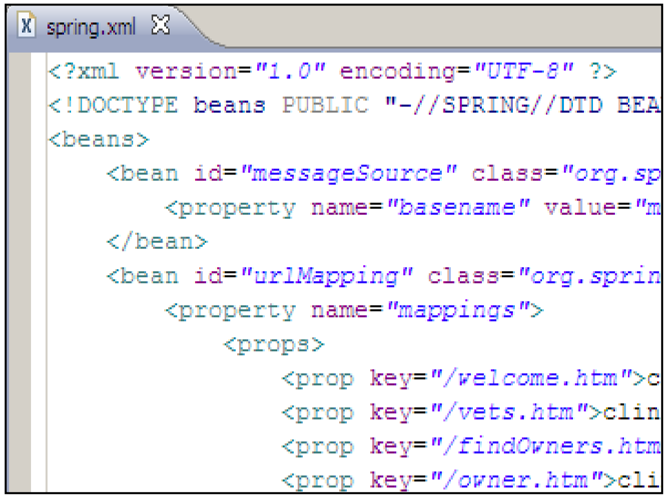
- Fixed set of tags
- Presentation oriented
- No data validation capabilities
- Single presentation

How can XML be Used?



rss.xml

```
<?xml version="1.0" encoding="utf-8"?>
<rss version="2.0">
  <channel>
    <title>Chinh tri - Xã hội</title>
    <description>Chinh tri - Xã hội</description>
    <link>http://www2.thanhnien.com.vn/Xahoi.tno</li
    <docs>http://backend.userland.com/rss</docs>
    <generator>Thanhnien.com.vn : http://thanhnien.c
    <item>
      <title>Thời tiết vào dịp Noel và Tết dương lic
      <description>&lt;P&gt;&lt;FONT size=2&gt;Thạc
      <link>http://www2.thanhnien.com.vn/Xahoi/2007/
      <pubDate>Wed, 19 Dec 2007 00:49:23 GMT</pubDat
    </item>
    <item>
      <title>Bộ trưởng Quốc phòng Ấn Độ thăm Việt Na
      <description>&lt;P&gt;&lt;FONT size=2&gt;Theo
      <link>http://www2.thanhnien.com.vn/Xahoi/2007/
      <pubDate>Wed, 19 Dec 2007 00:46:45 GMT</pubDat
```



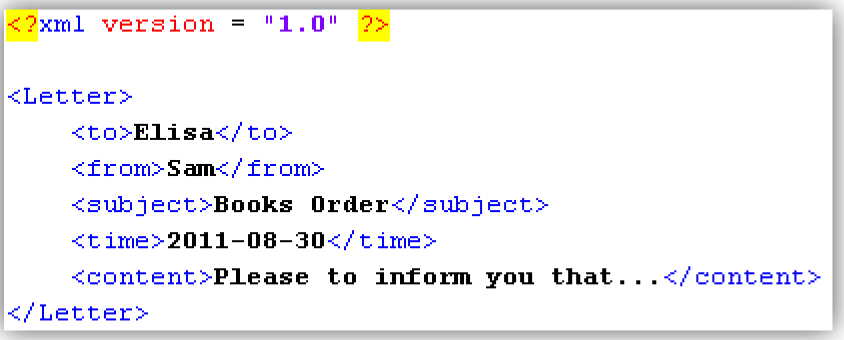
spring.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEA
<beans>
    <bean id="messageSource" class="org.sp
        <property name="basename" value="m
    </bean>
    <bean id="urlMapping" class="org.sprin
        <property name="mappings">
            <props>
                <prop key="/welcome.htm">c
                <prop key="/vets.htm">clin
                <prop key="/findOwners.htm
                <prop key="/owner.htm">cli
```

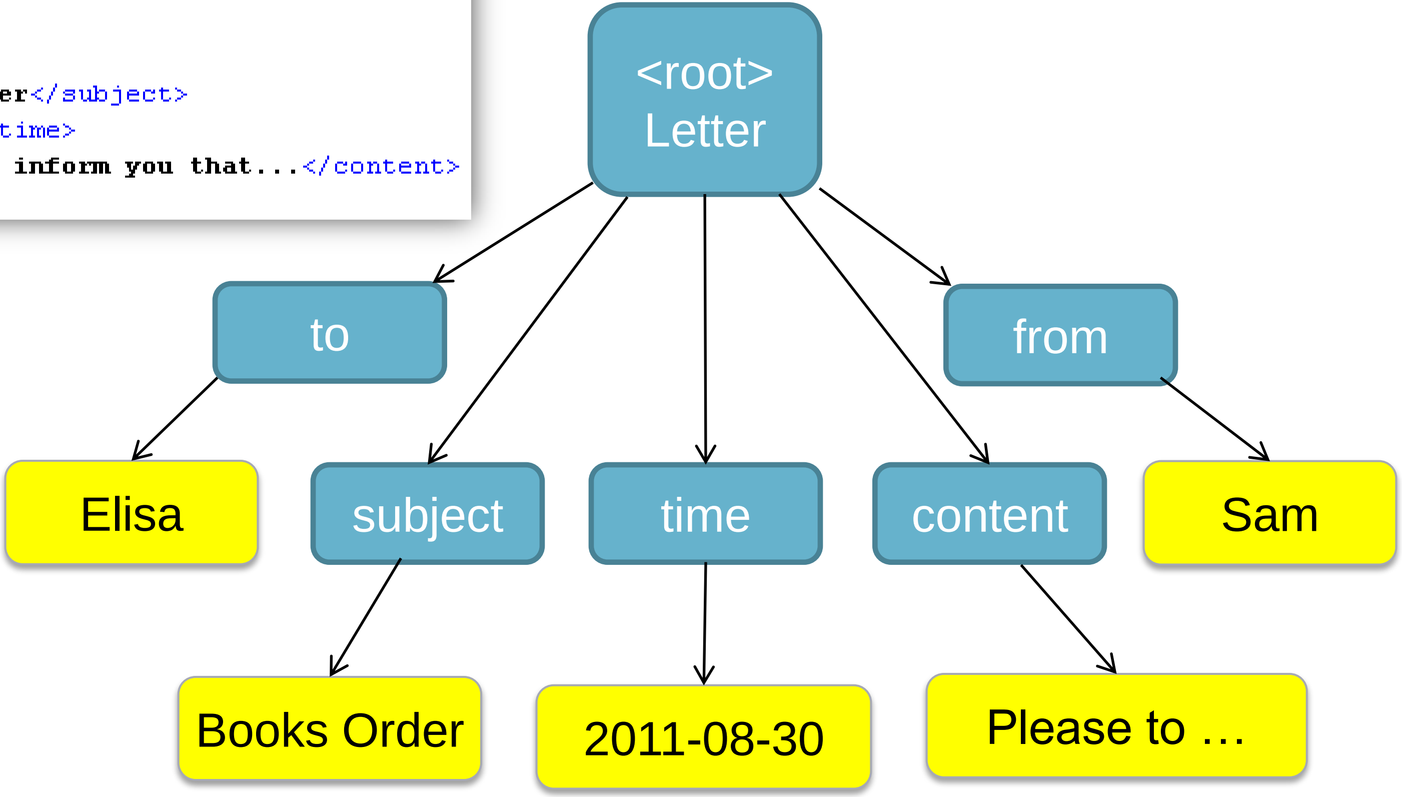
- XML Can be Used to Exchange Data (Web Services, RSS…)
- XML Can be Used to Store Data (configuration files of Struts, Spring Framework)
- XML Can be Used to Create New Languages (Ant build files, XML Schema…)

XML Document is a tree

- XML documents form a tree structure that starts at "the root" and branches to "the leaves"

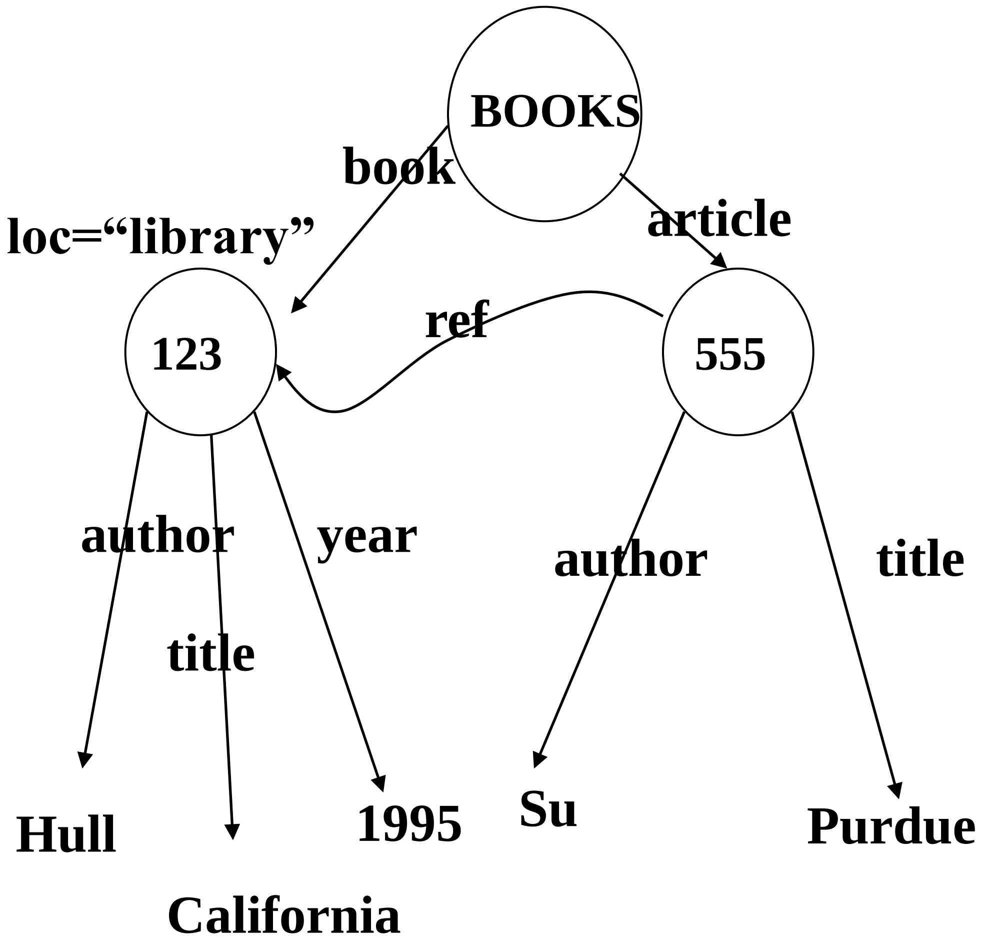


```
<?xml version = "1.0" ?>

<Letter>
    <to>Elisa</to>
    <from>Sam</from>
    <subject>Books Order</subject>
    <time>2011-08-30</time>
    <content>Please to inform you that...</content>
</Letter>
```



XML Document: Example

```
<BOOKS>
<book id="123" loc="library">
   <author>Hull</author>
   <title>California</title>
   <year> 1995 </year>
</book>
<article id="555" ref="123">
   <author>Su</author>
   <title> Purdue</title>
</article>
</BOOKS>
```



What are the parts?

- Header stuff- The XML declaration

  <?xml version="1.0" standalone="yes"?>

- The DOCTYPE

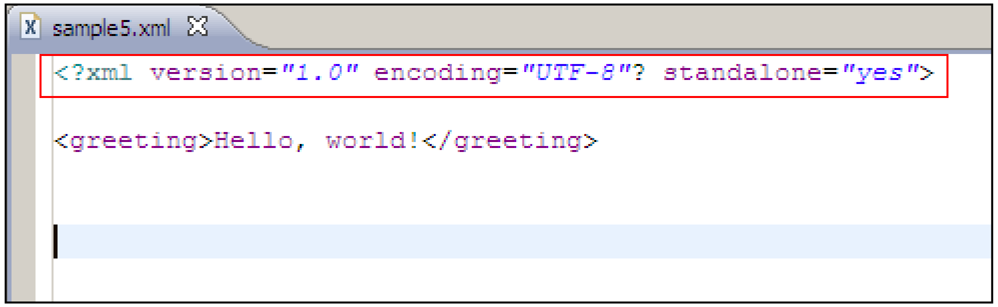
  <!DOCTYPE catalog SYSTEM "http://www.xyz.com/DTDs/catalog.dtd">

- Main document stuff

Elements:	<book>...</book>
Attributes:	<article id="123">…</article>
Text or other content:	Tools, computer
Entity references:	&lt;…&#174;
Comments	<!-- Prepared by... -->

XML Declarations

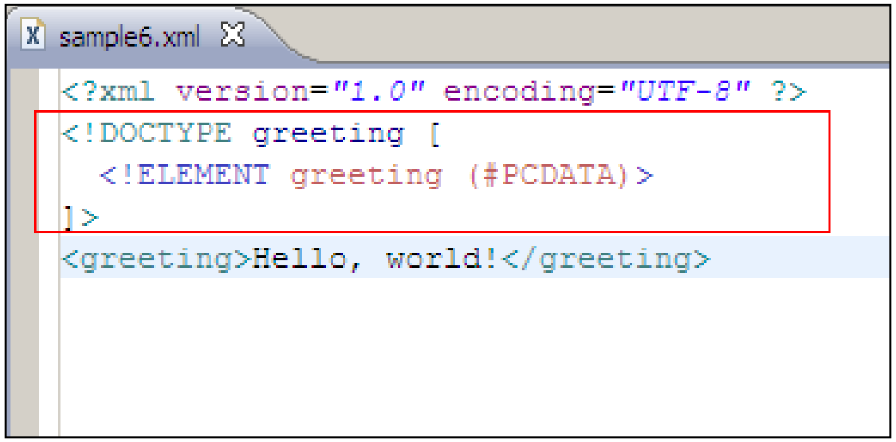
- XML documents SHOULD begin with an XML declaration which specifies the version of XML being used
  - version (required): Identifies the version of the XML markup language used in the data
  - encoding: Identifies the character set used to encode the data
- If the XML declaration is included, it must be at the first position of the first line in the XML document



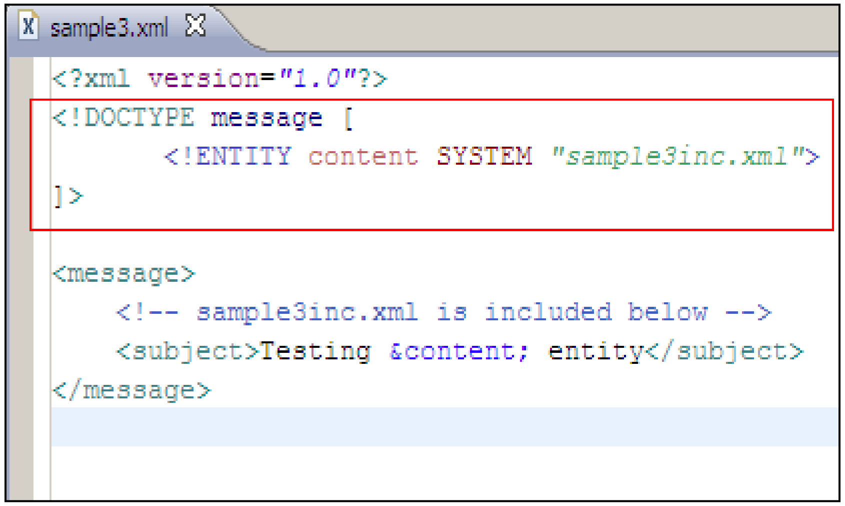
```
<?xml version="1.0" encoding="UTF-8"? standalone="yes">

<greeting>Hello, world!</greeting>
```

Document Type Declaration

- Purpose:
  - To define constraints on the XML logical structure (DTD)
  - To support the use of predefined storage units (character references, entity references)

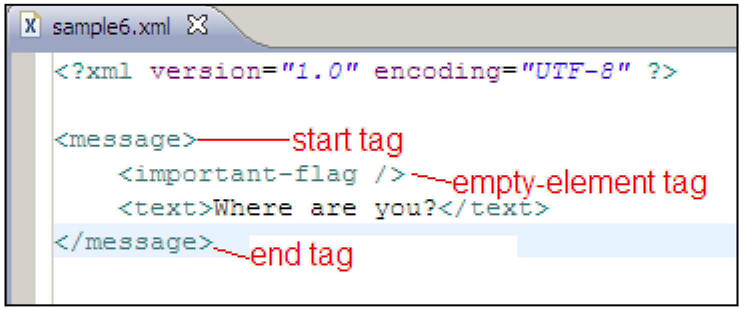

sample6.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE greeting [
  <!ELEMENT greeting (#PCDATA)>
]>
<greeting>Hello, world!</greeting>
```


sample3.xml

```
<?xml version="1.0"?>
<!DOCTYPE message [
      <!ENTITY content SYSTEM "sample3inc.xml">
]>

<message>
    <!-- sample3inc.xml is included below -->
    <subject>Testing &content; entity</subject>
</message>
```

Start-Tags, End-Tags, and Empty-Element Tags

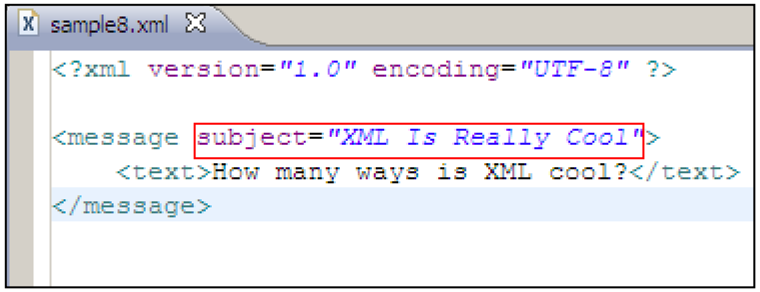


```
<?xml version="1.0" encoding="UTF-8" ?>

<message>
    <important-flag />
    <text>Where are you?</text>
</message>
```

- Beginning of every non-empty XML element is marked by a start-tag
- End of every element that begins with a start-tag MUST be marked by an end-tag
- Text between start-tag and end-tag is called element content
- Empty-element tags may be used for any element which has no content
- There is exactly one root element, which contains other elements (and other things)

Note: Tags are case sensitive

Start-Tags, End-Tags, and Empty-Element Tags (cont.)

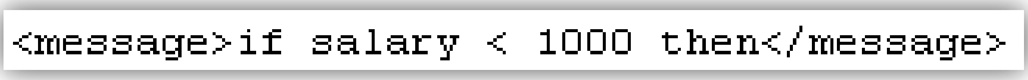

sample8.xml

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

<message subject="XML Is Really Cool">
    <text>How many ways is XML cool?</text>
</message>
```

- XML element can have attributes in the start tag (or empty-element tag)
- Attributes are used to provide additional information about elements
- Attribute values must always be enclosed in ‘single’ or ”double” quotes

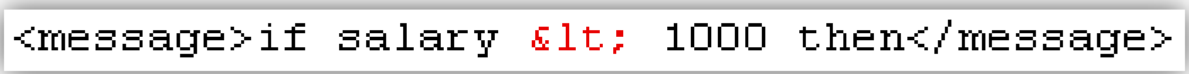
Character References

- Issue: Some characters have a special meaning in XML

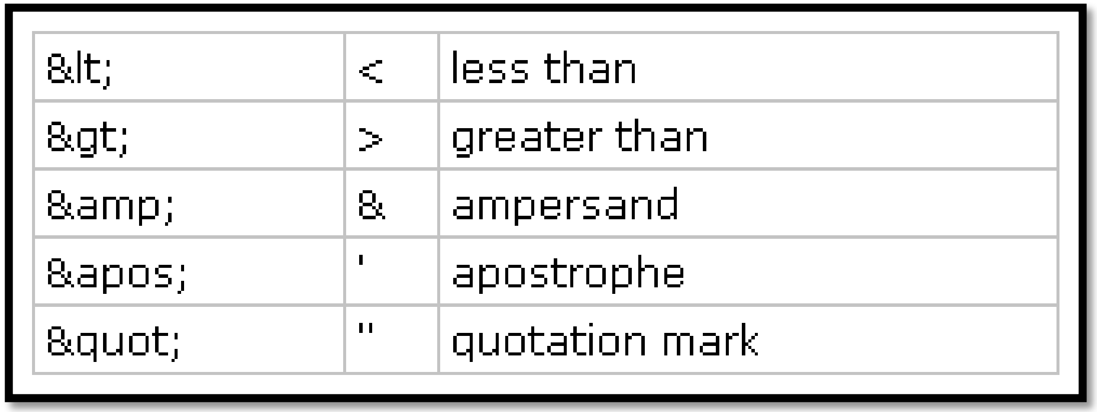


```
<message>if salary < 1000 then</message>
```

- Solution: Use the predefined entity references



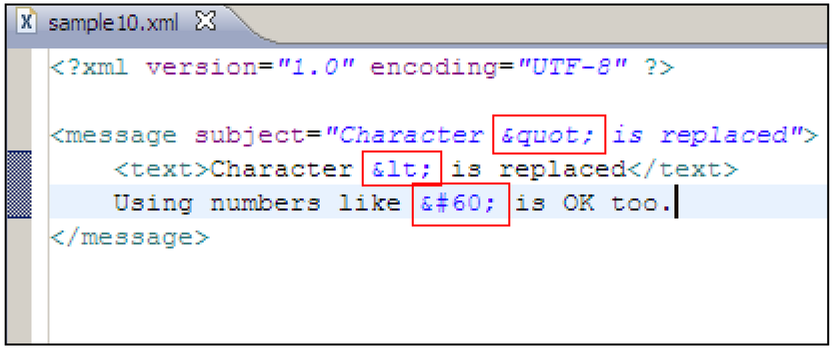
```
<message>if salary &lt; 1000 then</message>
```



&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

Character References (cont.)

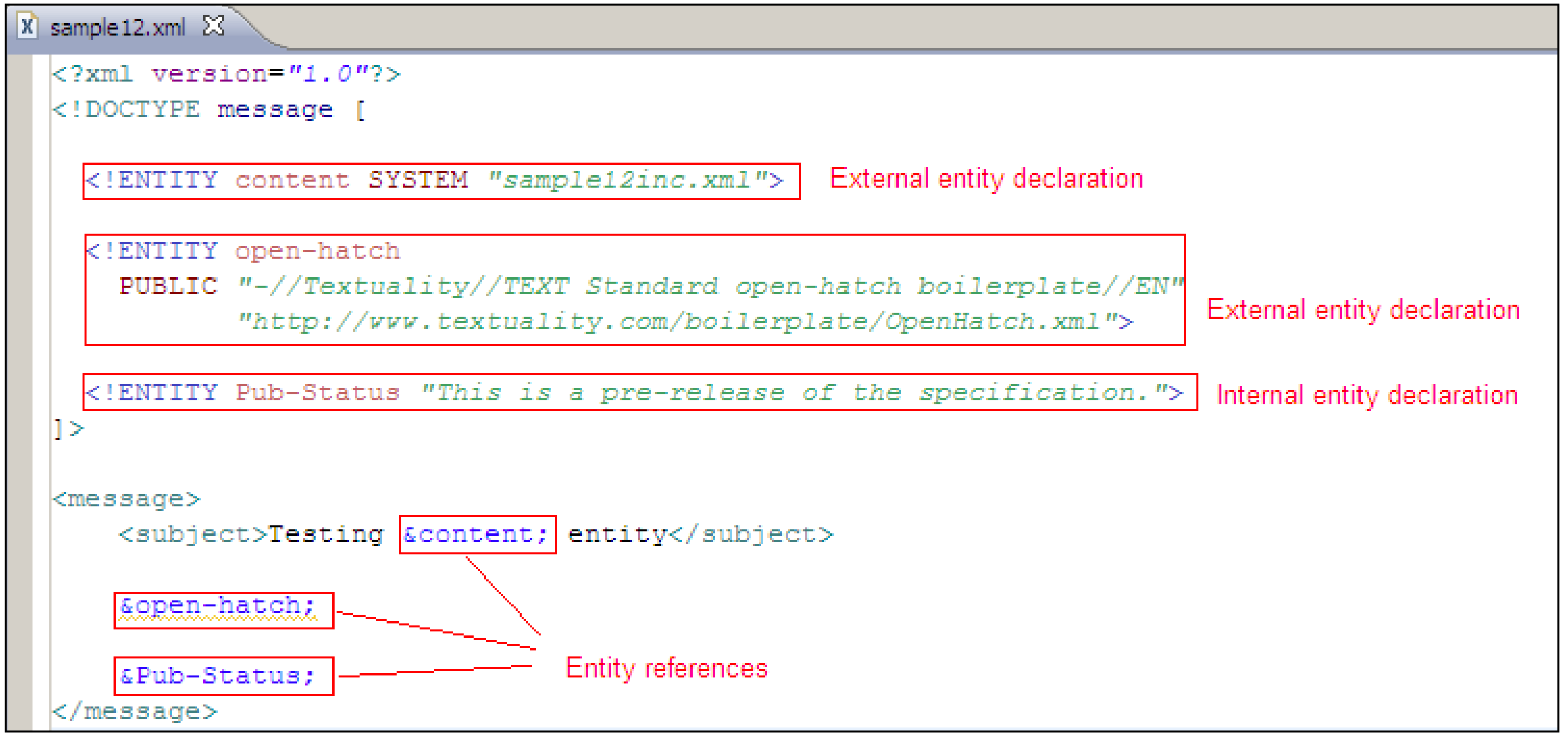
- To refer to a specific character in the ISO/IEC 10646 character set (for example one not directly accessible from available input devices)
- Formats:
  - &# number ;
  - &#x hex number ;
  - Or using predefined references &lt; (<) &gt; (>) &amp; (&) &apos; (') &quot; (")



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

<message subject="Character &quot; is replaced">
    <text>Character &lt; is replaced</text>
    Using numbers like &#60; is OK too.
</message>
```

Entity References

- Refer to the content of a named entity
- Entity Declarations: Internal, External
- Note: Often used for including other XML (fragment) files



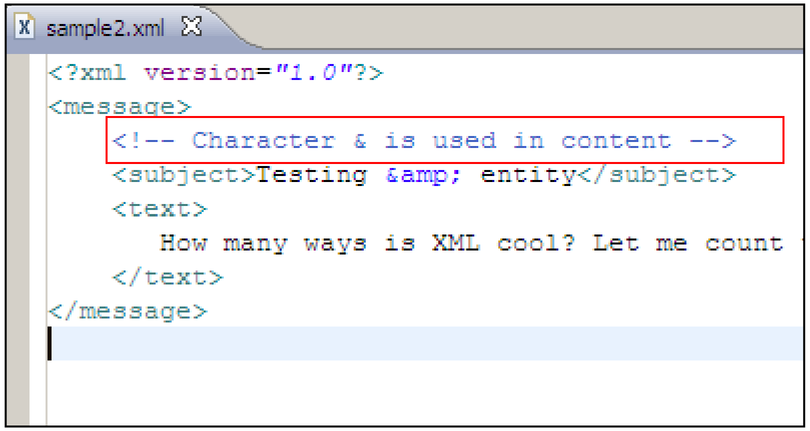
```
<?xml version="1.0"?>
<!DOCTYPE message [

  <!ENTITY content SYSTEM "sample12inc.xml">

  <!ENTITY open-hatch
    PUBLIC "-//Textuality//TEXT Standard open-hatch boilerplate//EN"
           "http://www.textuality.com/boilerplate/OpenHatch.xml">

  <!ENTITY Pub-Status "This is a pre-release of the specification.">
]>

<message>
    <subject>Testing &content; entity</subject>

    &open-hatch;

    &Pub-Status;
</message>
```

External entity declaration
External entity declaration
Internal entity declaration
Entity references

XML Comments

- The syntax for writing comments in XML:

<!-- This is a comment -->

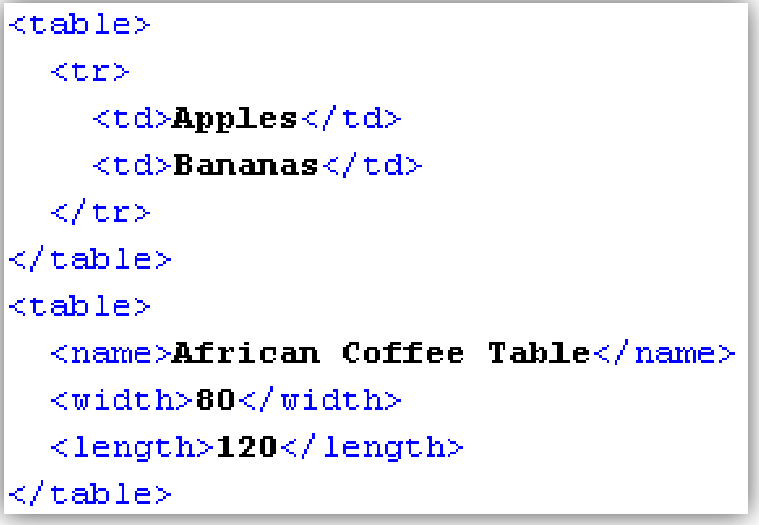


sample2.xml

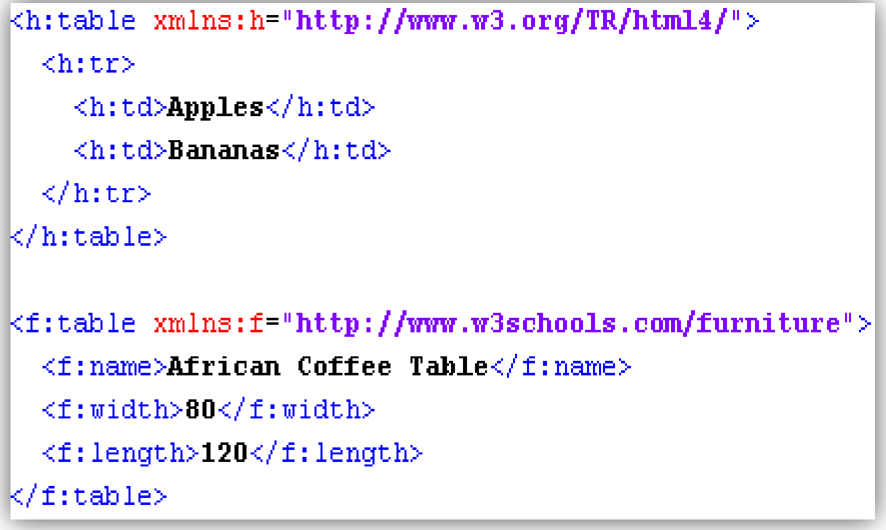
```
<?xml version="1.0"?>
<message>
    <!-- Character & is used in content -->
    <subject>Testing &amp; entity</subject>
    <text>
       How many ways is XML cool? Let me count
    </text>
</message>
```

Namespaces

- XML Namespaces provide a method to avoid element name conflicts



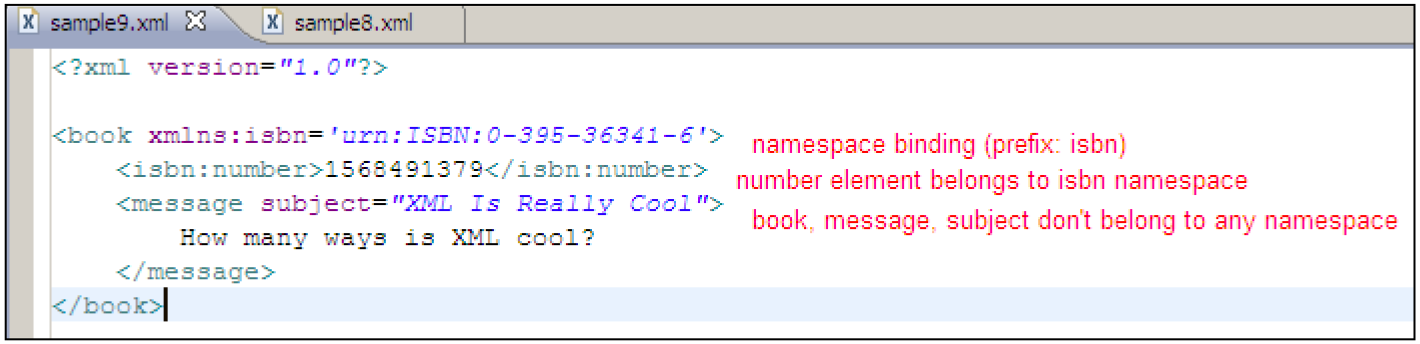
```
<table>
  <tr>
    <td>Apples</td>
    <td>Bananas</td>
  </tr>
</table>
<table>
  <name>African Coffee Table</name>
  <width>80</width>
  <length>120</length>
</table>
```



```
<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="http://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>
```

Namespaces (cont.)

- A namespace binding is declared using an attribute which its name must either be **xmlns** or begin **xmlns:**
- If an element type or attribute name is not specifically declared to be in an XML namespace and there is no default namespace then that name is not in any XML namespace

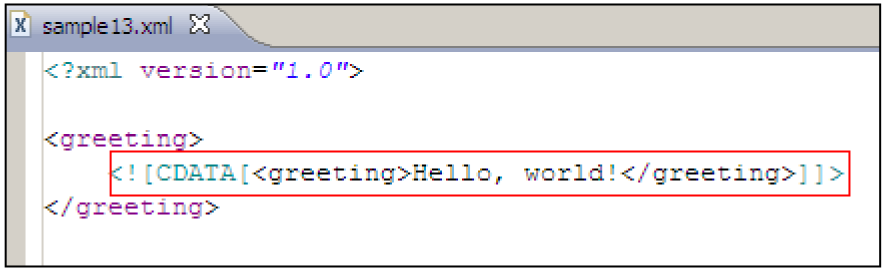


```
<?xml version="1.0"?>

<book xmlns:isbn='urn:ISBN:0-395-36341-6'>
    <isbn:number>1568491379</isbn:number>
    <message subject="XML Is Really Cool">
        How many ways is XML cool?
    </message>
</book>
```

namespace binding (prefix: isbn)
number element belongs to isbn namespace
book, message, subject don't belong to any namespace

CDATA Sections

- CDATA sections are used to escape blocks of text containing special characters (like <, >, &, ..)
- CDATA sections begin with the string "**<![CDATA[**" and end with the string "**]]>**"
- CDATA sections may occur anywhere character data may occur



sample13.xml

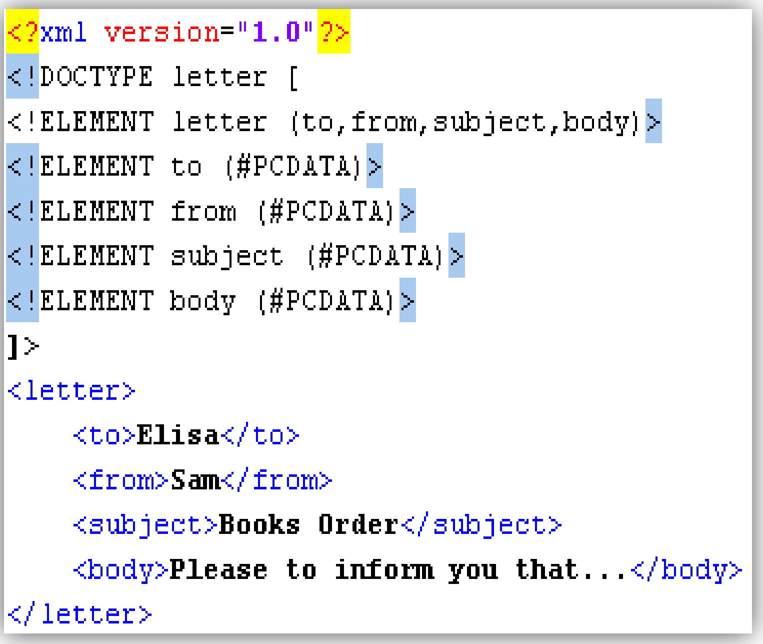
```
<?xml version="1.0">

<greeting>
    <![CDATA[<greeting>Hello, world!</greeting>]]>
</greeting>
```

Points to Remember

- XML is used for data exchange/storage
- One XML document has only one root element
- Elements have attributes and can contain other elements
- Namespace used for qualifying element names and attribute names
- References used for replacing values to placeholders (can be used for including fragment files)
- Escaping characters by references/CDATA sections

DTD- Document Type Definition

- It defines the document structure with a list of legal elements and attributes
- A DTD is a set of rules that allow us to specify our own set of elements and attributes.
- A DTD is grammar to indicate what tags are legal in XML documents
- XML Document is valid if it has an attached DTD and document is structured according to rules defined in DTD



```
<?xml version="1.0"?>
<!DOCTYPE letter [
<!ELEMENT letter (to,from,subject,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<letter>
    <to>Elisa</to>
    <from>Sam</from>
    <subject>Books Order</subject>
    <body>Please to inform you that...</body>
</letter>
```

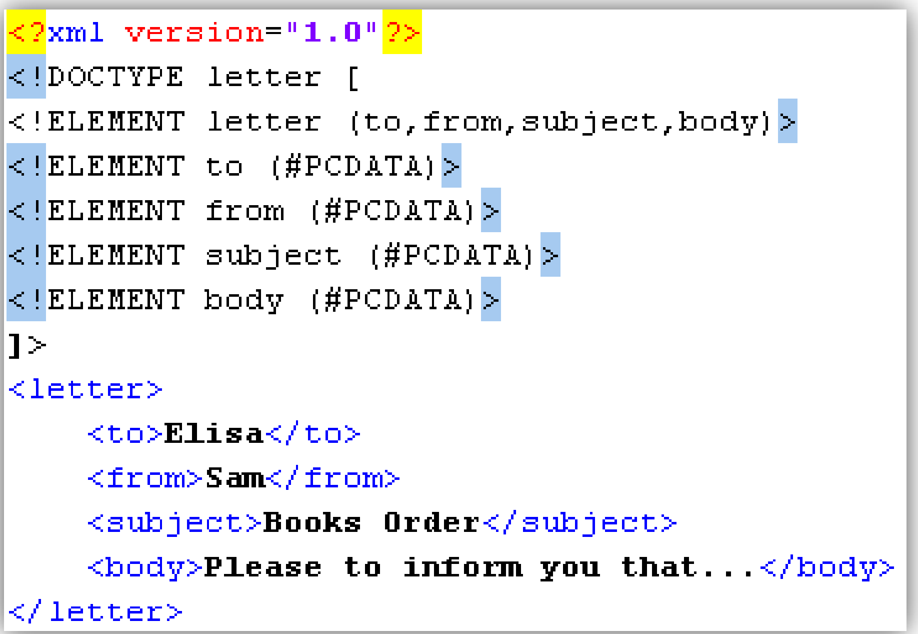
Why use DTD?

- With a DTD, each of your XML files can carry a description of its own format
- With a DTD, independent groups of people can agree to use a standard DTD for interchanging data
- Your application can use a standard DTD to verify that the data you receive from the outside world is valid
- You can also use a DTD to verify your own data

DTD Declaration

- A DTD can be declared inline inside an XML document, or as an external reference
- Internal DTD Declaration:

<!DOCTYPE root-element [element-declarations]>

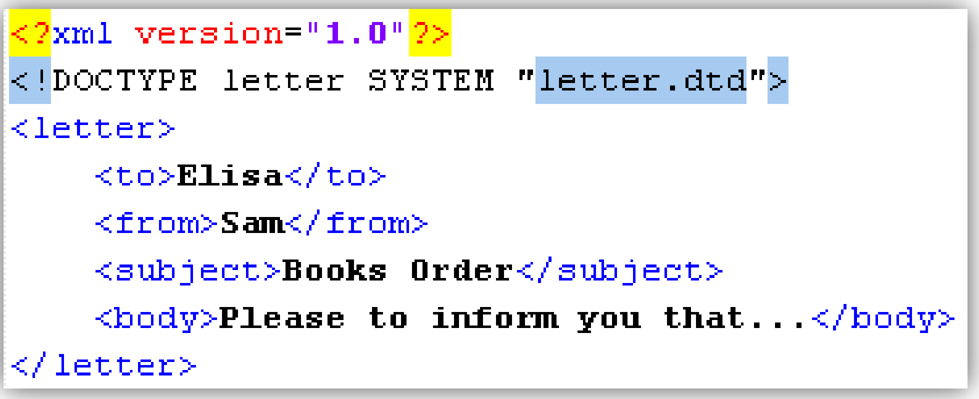


```
<?xml version="1.0"?>
<!DOCTYPE letter [
<!ELEMENT letter (to,from,subject,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<letter>
    <to>Elisa</to>
    <from>Sam</from>
    <subject>Books Order</subject>
    <body>Please to inform you that...</body>
</letter>
```

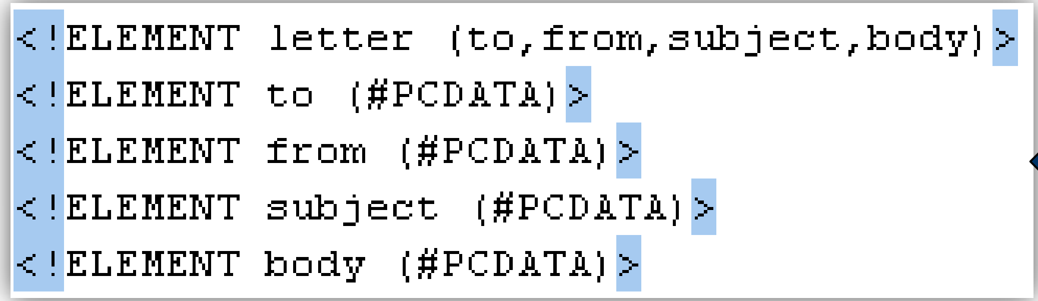
DTD Declaration (cont.)

- External DTD Declaration:

  <!DOCTYPE root-element SYSTEM "filename">



```
<?xml version="1.0"?>
<!DOCTYPE letter SYSTEM "letter.dtd">
<letter>
    <to>Elisa</to>
    <from>Sam</from>
    <subject>Books Order</subject>
    <body>Please to inform you that...</body>
</letter>
```



```
<!ELEMENT letter (to,from,subject,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT subject (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

letter.dtd

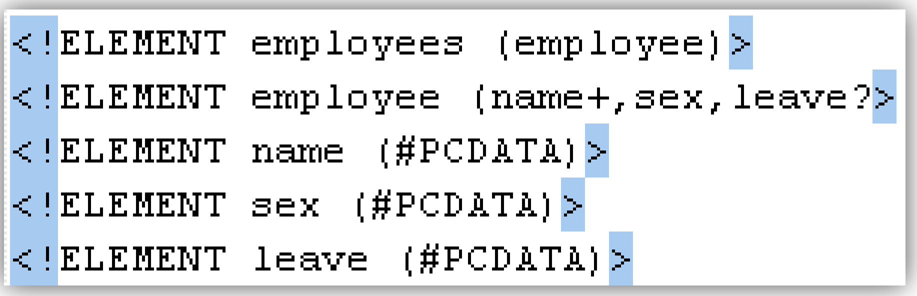
DTD Elements

- In a DTD, elements are declared with an ELEMENT declaration

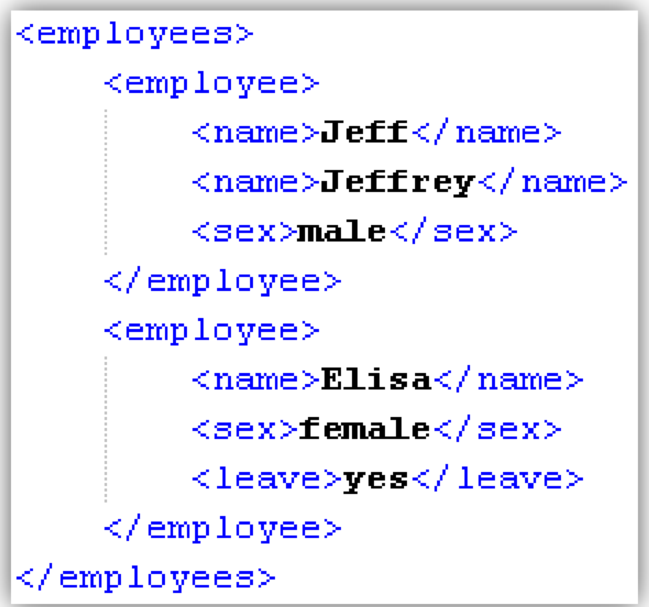
  <!ELEMENT element-name (element-content)>

- Syntax:
  - <!ELEMENT element-name EMPTY>
  - <!ELEMENT element-name ANY>
  - <!ELEMENT element-name (child1,child2,...)>
  - <!ELEMENT element-name (child1|child2)>
  - <!ELEMENT element-name (#PCDATA)>
  - <!ELEMENT element-name (child-name+)>
  - <!ELEMENT element-name (child-name*)>
  - <!ELEMENT element-name (child-name?)>

DTD Elements (cont.)



```
<!ELEMENT employees (employee)>
<!ELEMENT employee (name+,sex,leave?>
<!ELEMENT name (#PCDATA)>
<!ELEMENT sex (#PCDATA)>
<!ELEMENT leave (#PCDATA)>
```



```
<employees>
    <employee>
        <name>Jeff</name>
        <name>Jeffrey</name>
        <sex>male</sex>
    </employee>
    <employee>
        <name>Elisa</name>
        <sex>female</sex>
        <leave>yes</leave>
    </employee>
</employees>
```

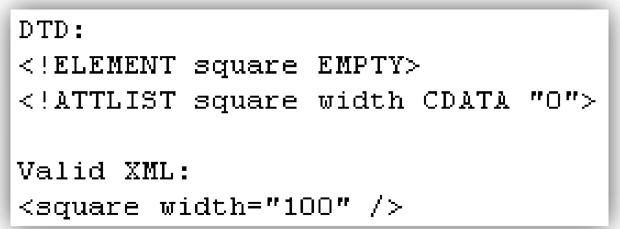
DTD Attributes

- In a DTD, attributes are declared with an ATTLIST declaration

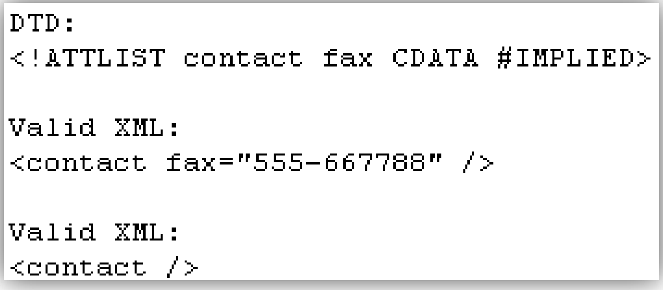
  <!ATTLIST element-name attribute-name attribute-type attribute-value>

- Syntax:
  - <!ATTLIST element-name attribute-name attribute-type (value | #IMPLIED | #REQUIRED | #FIXED value)>
  - <!ATTLIST element-name attribute-name (en1|en2|..) default-value>

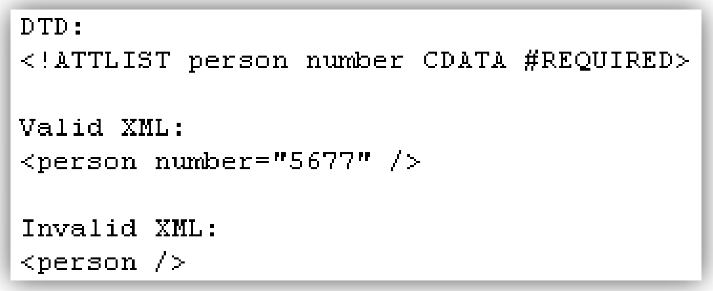
DTD Attributes (cont.)



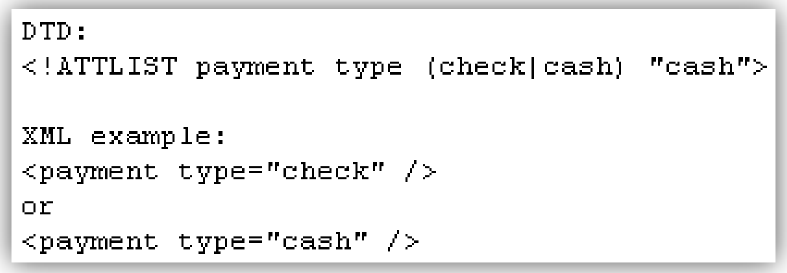
```
DTD:
<!ELEMENT square EMPTY>
<!ATTLIST square width CDATA "0">

Valid XML:
<square width="100" />
```



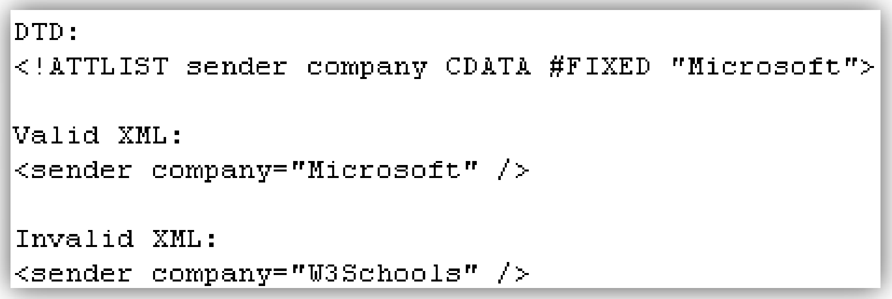
```
DTD:
<!ATTLIST contact fax CDATA #IMPLIED>

Valid XML:
<contact fax="555-667788" />

Valid XML:
<contact />
```



```
DTD:
<!ATTLIST person number CDATA #REQUIRED>

Valid XML:
<person number="5677" />

Invalid XML:
<person />
```



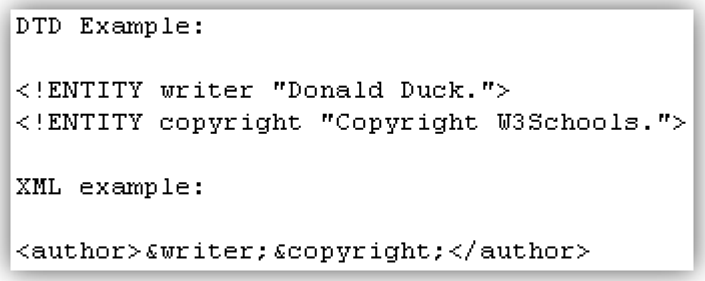
```
DTD:
<!ATTLIST payment type (check|cash) "cash">

XML example:
<payment type="check" />
or
<payment type="cash" />
```



```
DTD:
<!ATTLIST sender company CDATA #FIXED "Microsoft">

Valid XML:
<sender company="Microsoft" />

Invalid XML:
<sender company="W3Schools" />
```

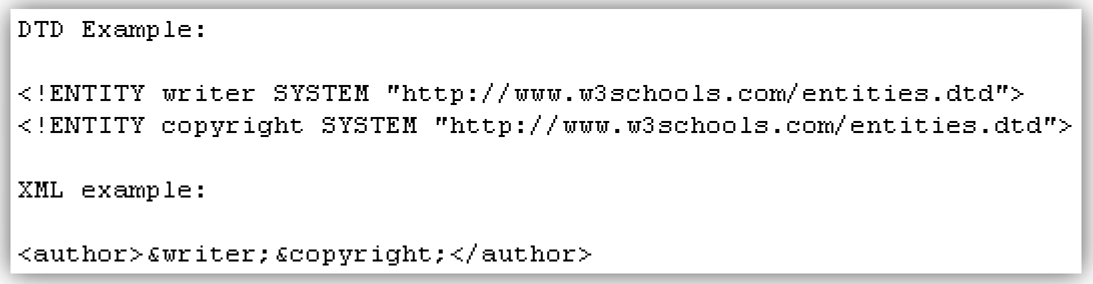
DTD Entities

- Entities are variables used to define shortcuts to standard text or special characters
- Entities can be declared internal or external:
  - <!ENTITY entity-name "entity-value">



```
DTD Example:

<!ENTITY writer "Donald Duck.">
<!ENTITY copyright "Copyright W3Schools.">

XML example:

<author>&writer;&copyright;</author>
```

  - <!ENTITY entity-name SYSTEM "URI/URL">



```
DTD Example:

<!ENTITY writer SYSTEM "http://www.w3schools.com/entities.dtd">
<!ENTITY copyright SYSTEM "http://www.w3schools.com/entities.dtd">

XML example:

<author>&writer;&copyright;</author>
```

DTD Summary

- DTD is used to describe the structure of an XML document.
- DTD can be declared inside your XML document, or as an external reference.

Q&A

Contact: Tuyen Nguyen

Mobile +84 983 830 860 | tnguyen256@csc.com

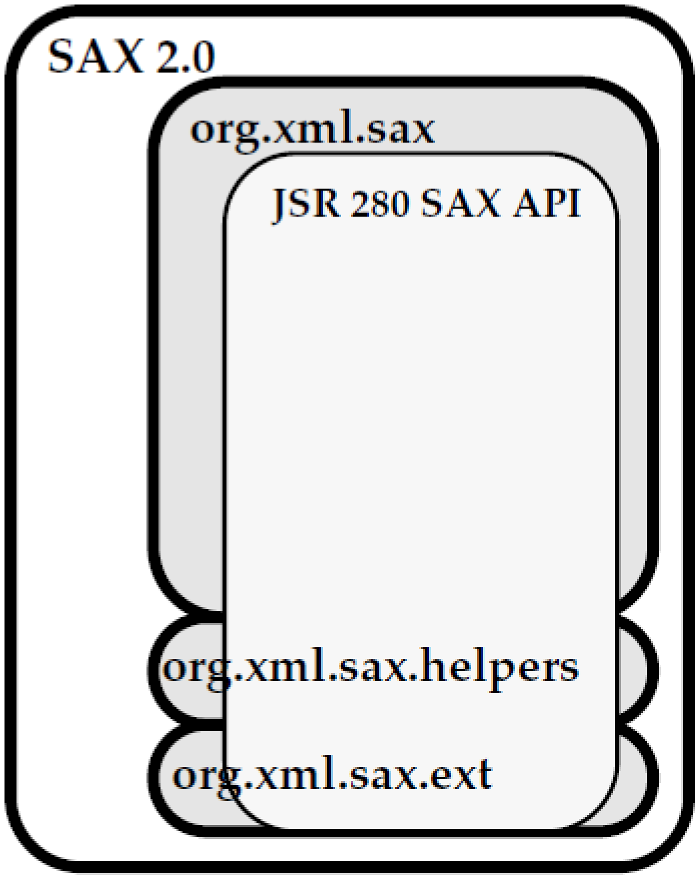
JSR 280 Introduction

JSR 280 Introduction

- A specification for the XML API for the JavaTM Micro Edition (JavaTM ME) Platform
- Optional package for CLDC 1.1
- API reference: http://docs.oracle.com/javame/config/cldc/opt-pkgs/api/xml/jsr280/

XML Programming Using SAX

JSR 280 SAX API



Graphical Representation of JSR 280 SAX Subset

Simple API for XML

- SAX is a event driven API
  - No class models the XML document itself
  - feeds content to the application through a callback interface/methods
- SAX is fast and efficient, it requires much less memory than DOM, because SAX does not construct an internal representation (tree structure) of the XML data, as a DOM does
- SAX is the real choice for truly huge XML documents

Steps to writing SAX Handlers

- Create a parser instance:

SAXParserFactory factory = SAXParserFactory.newInstance();

SAXParser parser = factory.newSAXParser();

  – setNamespaceAware
  – setVadidating (check data based on DTD)

- Implement the EntityResolver, DTDHandler, ContentHandler, ErrorHandler interfaces (or extend DefaultHandler class) to handle events

Steps to writing SAX Handlers (cont.)

- Invoke the parser with the designated content

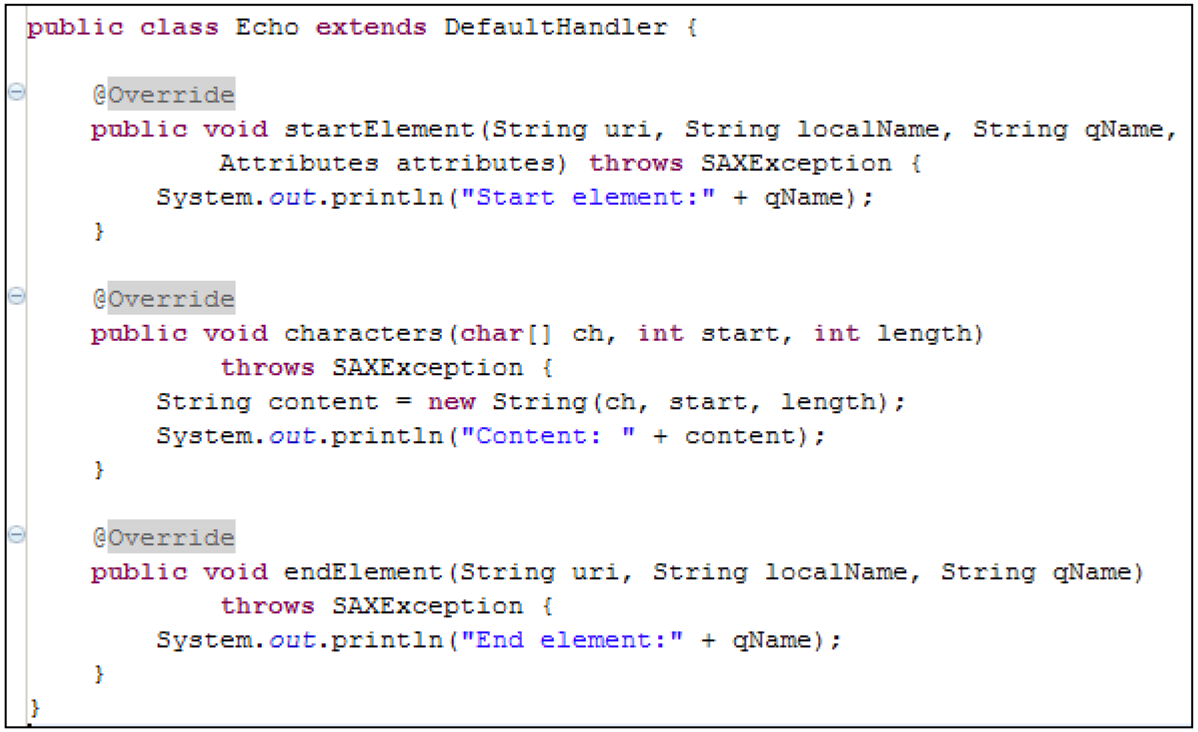
handlerparser.parse(xmlSource, handler);

  - xmlSource: from URI, file, InputSource
  - Handler: the event implemented class

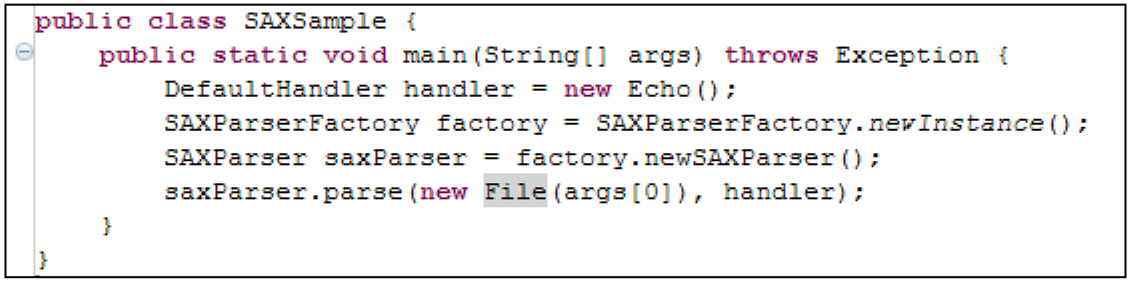
Some usually events

- void startDocument()
  - Receive notification of the beginning of the document
- void endDocument()
  - Receive notification of the end of the document
- void startElement(String uri, String localName, String qName, Attributes attributes)
  - Receive notification of the start of an element
- void endElement(String uri, String localName, String qName)
  - Receive notification of the end of an element
- void error(SAXParseException e)
  - Receive notification of a recoverable parser error

Example

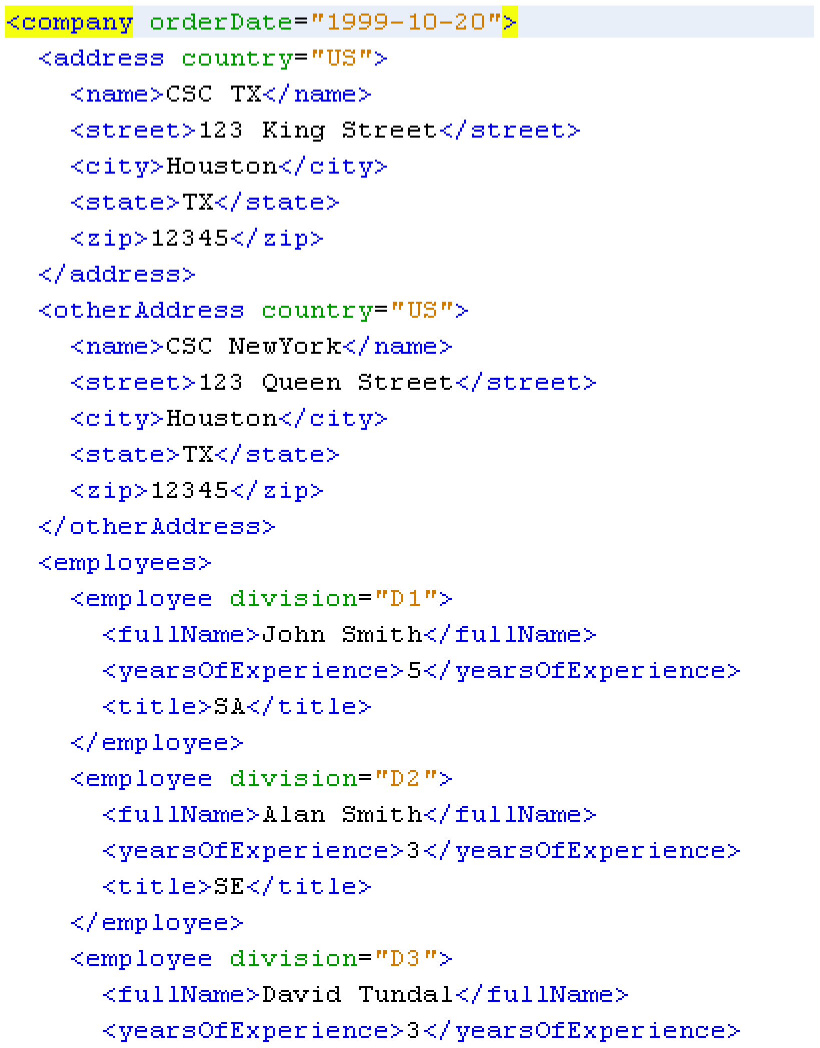


```java
public class Echo extends DefaultHandler {

    @Override
    public void startElement(String uri, String localName, String qName,
            Attributes attributes) throws SAXException {
        System.out.println("Start element:" + qName);
    }

    @Override
    public void characters(char[] ch, int start, int length)
            throws SAXException {
        String content = new String(ch, start, length);
        System.out.println("Content: " + content);
    }

    @Override
    public void endElement(String uri, String localName, String qName)
            throws SAXException {
        System.out.println("End element:" + qName);
    }
}
```



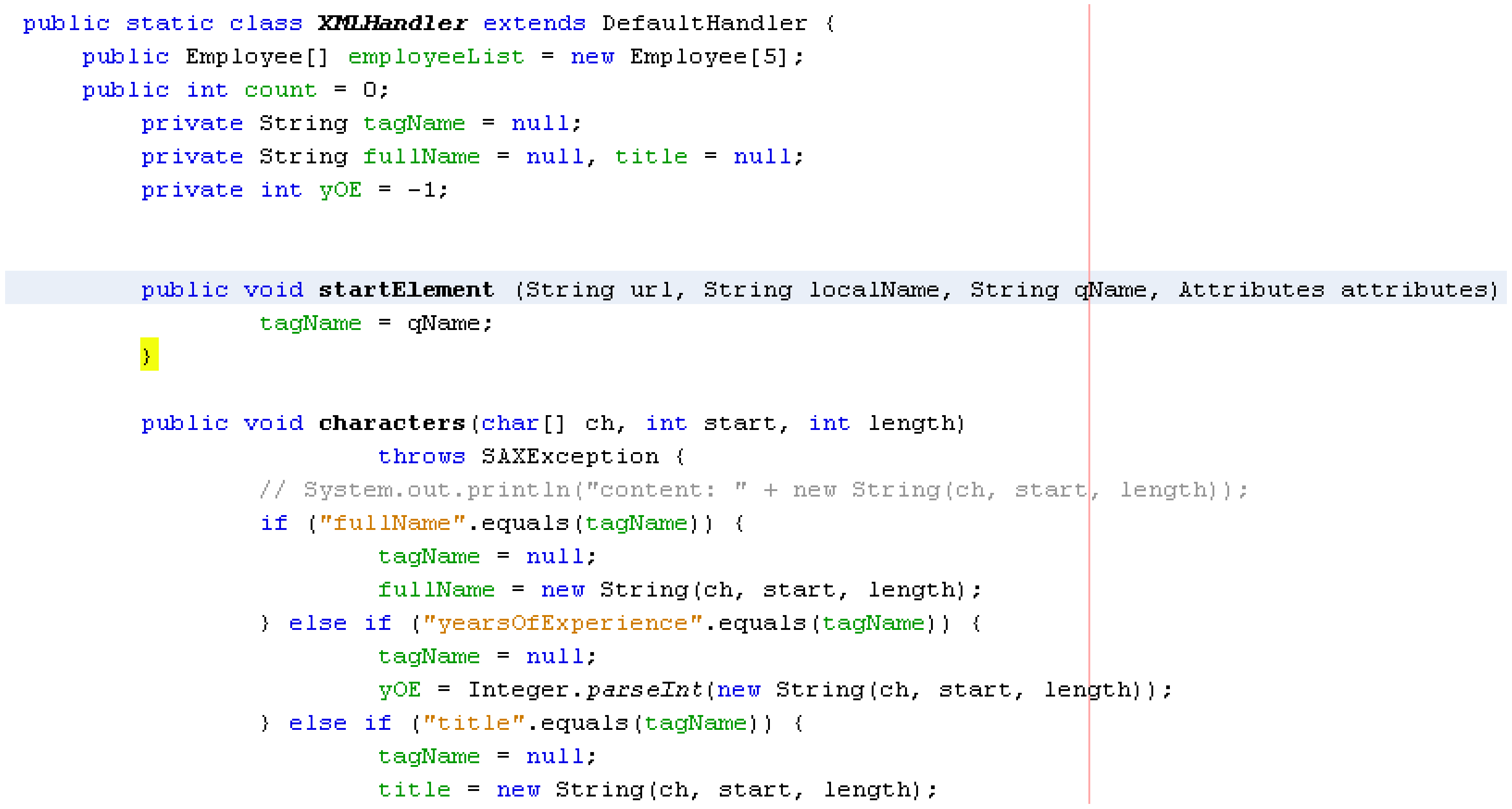
```java
public class SAXSample {
    public static void main(String[] args) throws Exception {
        DefaultHandler handler = new Echo();
        SAXParserFactory factory = SAXParserFactory.newInstance();
        SAXParser saxParser = factory.newSAXParser();
        saxParser.parse(new File(args[0]), handler);
    }
}
```

Example (cont.)



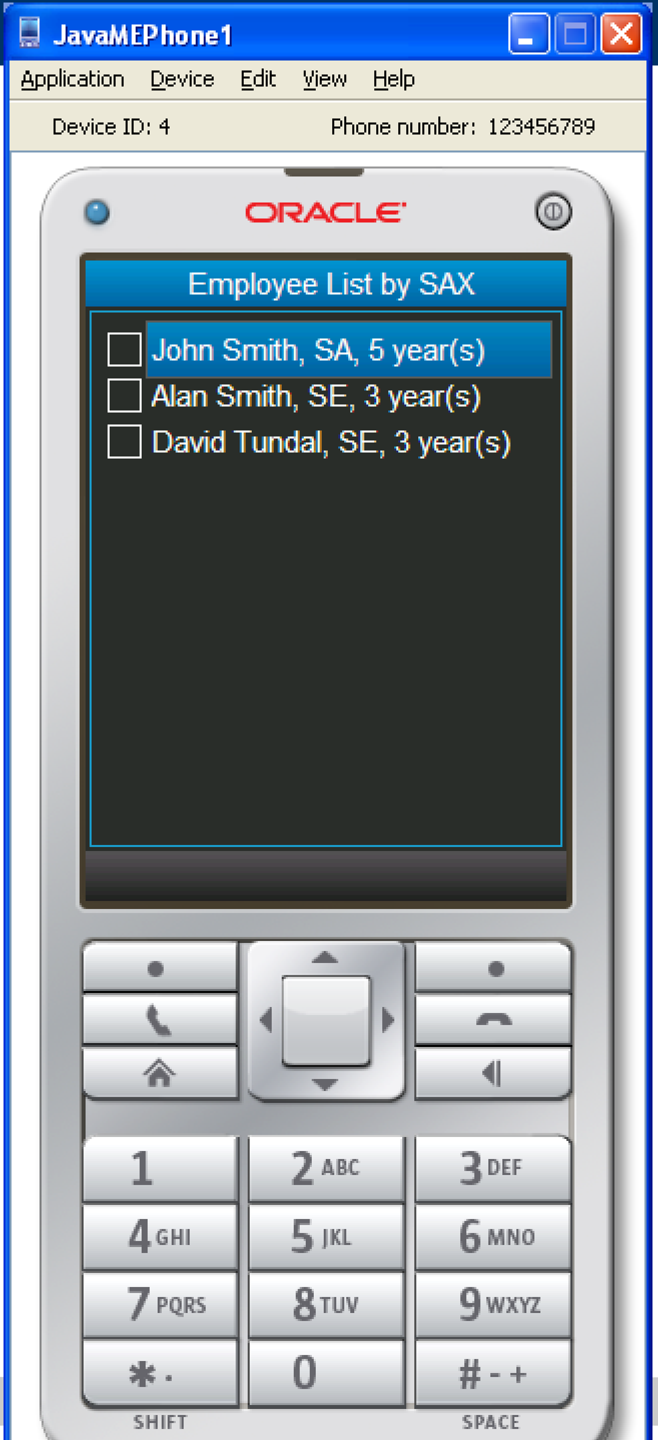
```xml
<company orderDate="1999-10-20">
  <address country="US">
    <name>CSC TX</name>
    <street>123 King Street</street>
    <city>Houston</city>
    <state>TX</state>
    <zip>12345</zip>
  </address>
  <otherAddress country="US">
    <name>CSC NewYork</name>
    <street>123 Queen Street</street>
    <city>Houston</city>
    <state>TX</state>
    <zip>12345</zip>
  </otherAddress>
  <employees>
    <employee division="D1">
      <fullName>John Smith</fullName>
      <yearsOfExperience>5</yearsOfExperience>
      <title>SA</title>
    </employee>
    <employee division="D2">
      <fullName>Alan Smith</fullName>
      <yearsOfExperience>3</yearsOfExperience>
      <title>SE</title>
    </employee>
    <employee division="D3">
      <fullName>David Tundal</fullName>
      <yearsOfExperience>3</yearsOfExperience>
```

Example (cont.)



```
public static class XMLHandler extends DefaultHandler {
    public Employee[] employeeList = new Employee[5];
    public int count = 0;
        private String tagName = null;
        private String fullName = null, title = null;
        private int yOE = -1;

        public void startElement (String url, String localName, String qName, Attributes attributes)
                tagName = qName;
        }

        public void characters(char[] ch, int start, int length)
                        throws SAXException {
                // System.out.println("content: " + new String(ch, start, length));
                if ("fullName".equals(tagName)) {
                        tagName = null;
                        fullName = new String(ch, start, length);
                } else if ("yearsOfExperience".equals(tagName)) {
                        tagName = null;
                        yOE = Integer.parseInt(new String(ch, start, length));
                } else if ("title".equals(tagName)) {
                        tagName = null;
                        title = new String(ch, start, length);
```

Example (cont.)



Employee List by SAX

- John Smith, SA, 5 year(s)
- Alan Smith, SE, 3 year(s)
- David Tundal, SE, 3 year(s)

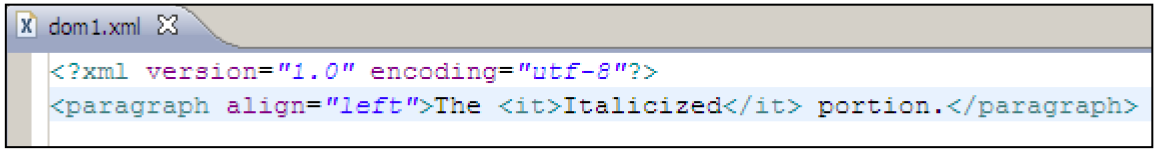
XML Programming Using DOM

Document Object Model

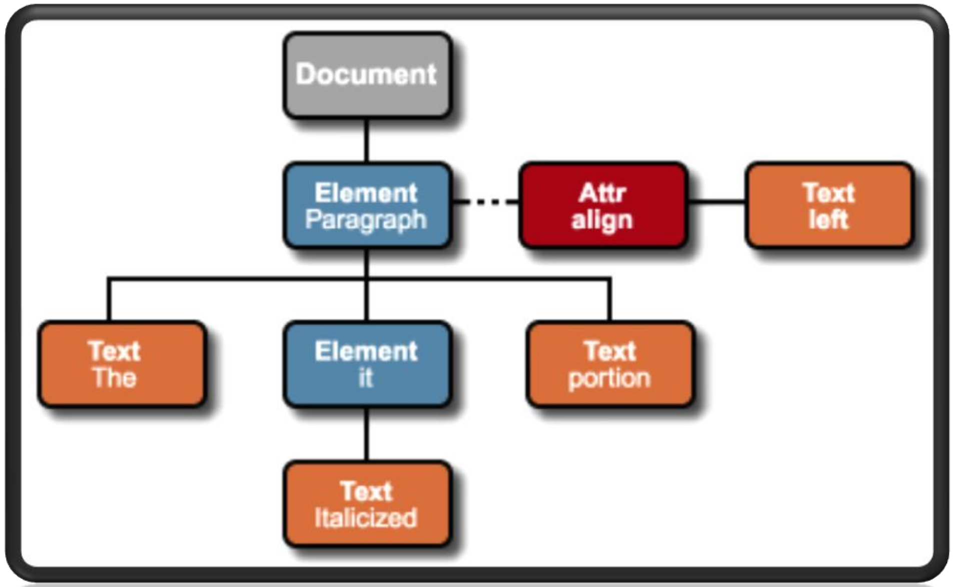
- A W3C standard for platform- and language-neutral dynamic access and update of the content, structure, and style of XML documents
- Is implemented in a wide variety of languages, including Java, JavaScript, C++, dotNet, …
- Presents an XML document as a tree-structure (a node tree), with the elements, attributes, text, … defined as nodes.
  - random access to widely separated parts of the original document
  - memory intensive compared to SAX

Document Object Model (cont.)

- An example
  - Document, Element, Text, and Attr pieces are Nodes
  - The Text nodes are independent nodes, not values of Element nodes.



```
<?xml version="1.0" encoding="utf-8"?>
<paragraph align="left">The <it>Italicized</it> portion.</paragraph>
```



Steps to writing DOM

- Create a JAXP document builder:

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

  - setNamespaceAware
  - setVadidating (check data based on DTD)

- Invoke the parser to create a Document representing an XML parse document

```
Document document = builder.parse(someInputStream);
```

Steps to writing DOM (cont.)

- Normalize the tree

document.getDocumentElement().normalize();

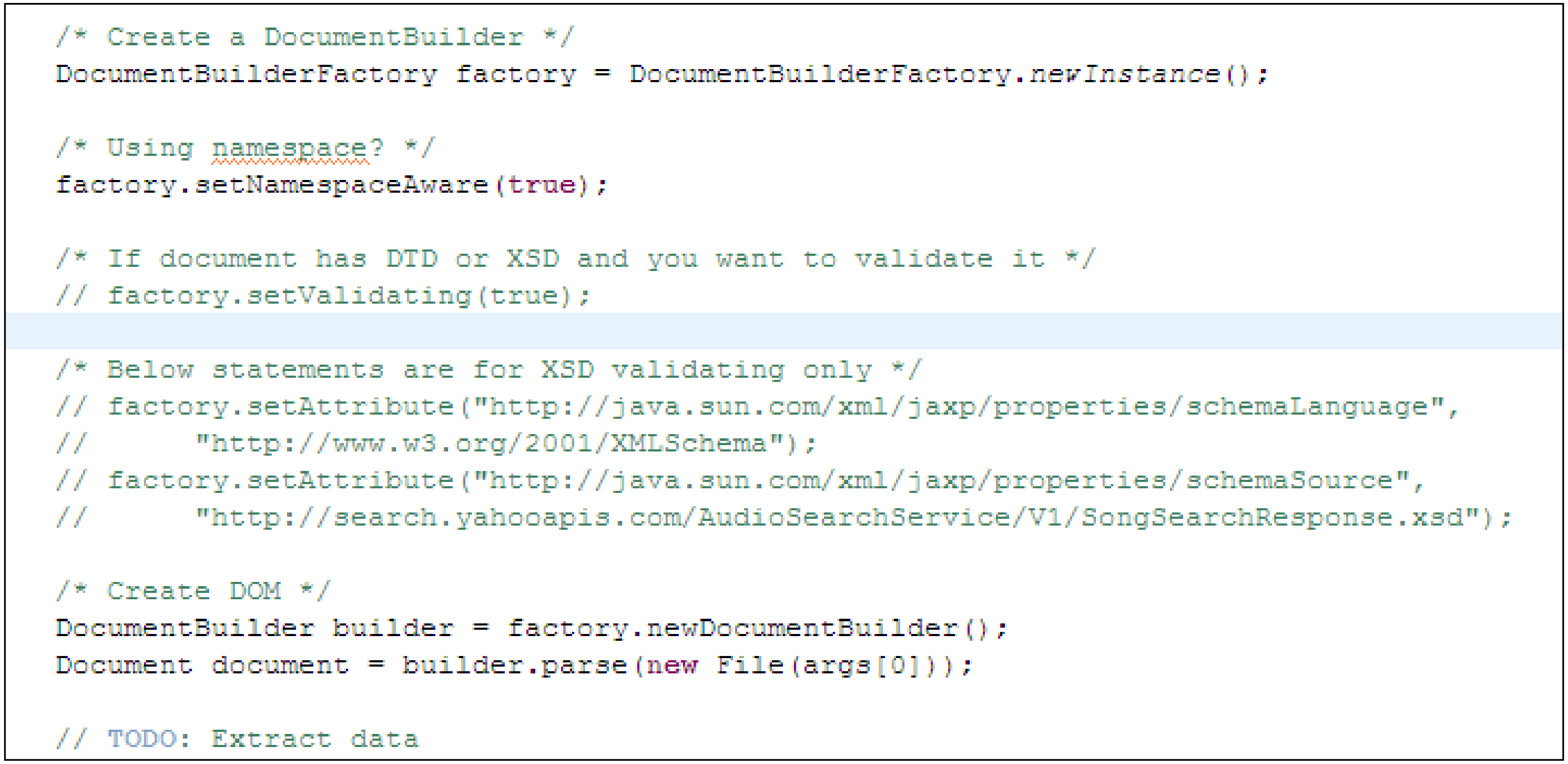
  – This means to combine textual nodes that were on multiple lines and to eliminate empty textual nodes

- Obtain the root node of the tree

Element rootElement = document.getDocumentElement();

- Examine various properties of the node

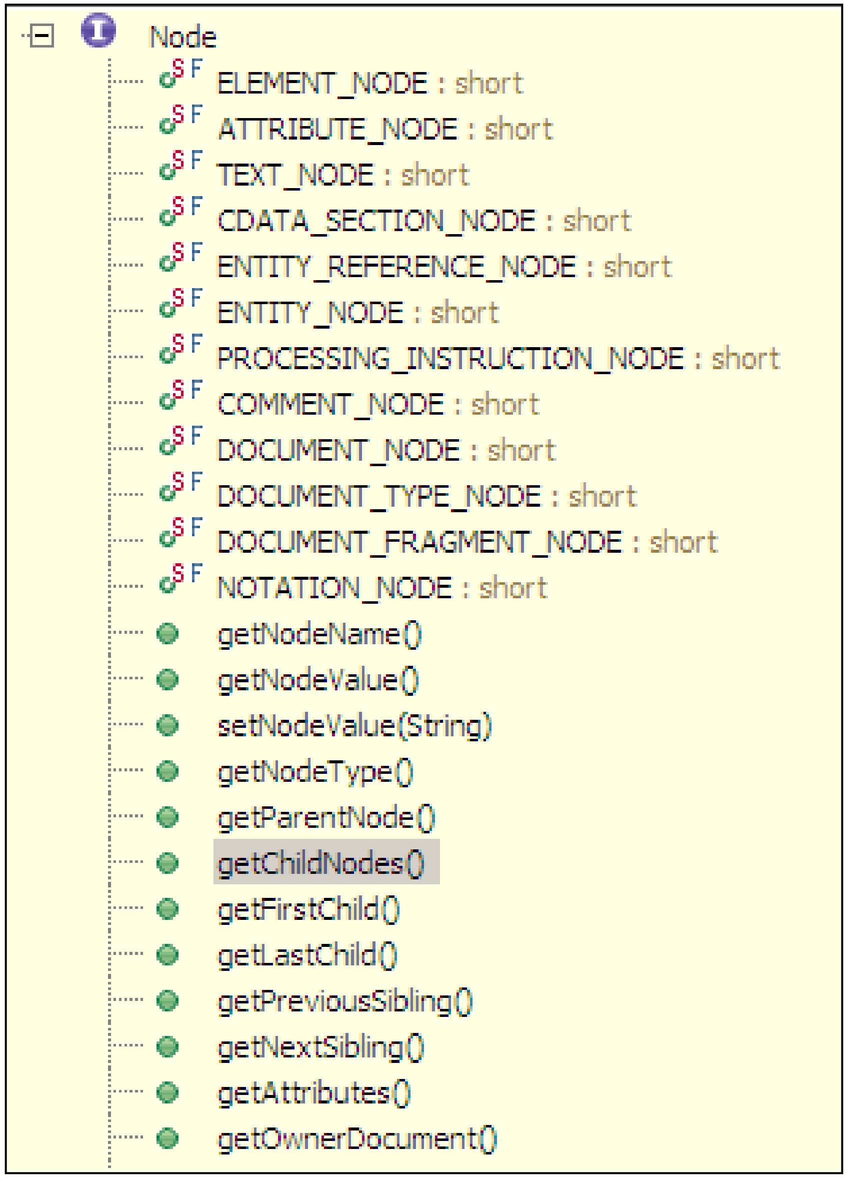
Building the DOM



```
/* Create a DocumentBuilder */
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();

/* Using namespace? */
factory.setNamespaceAware(true);

/* If document has DTD or XSD and you want to validate it */
// factory.setValidating(true);

/* Below statements are for XSD validating only */
// factory.setAttribute("http://java.sun.com/xml/jaxp/properties/schemaLanguage",
//       "http://www.w3.org/2001/XMLSchema");
// factory.setAttribute("http://java.sun.com/xml/jaxp/properties/schemaSource",
//       "http://search.yahooapis.com/AudioSearchService/V1/SongSearchResponse.xsd");

/* Create DOM */
DocumentBuilder builder = factory.newDocumentBuilder();
Document document = builder.parse(new File(args[0]));

// TODO: Extract data
```

Traversing the DOM

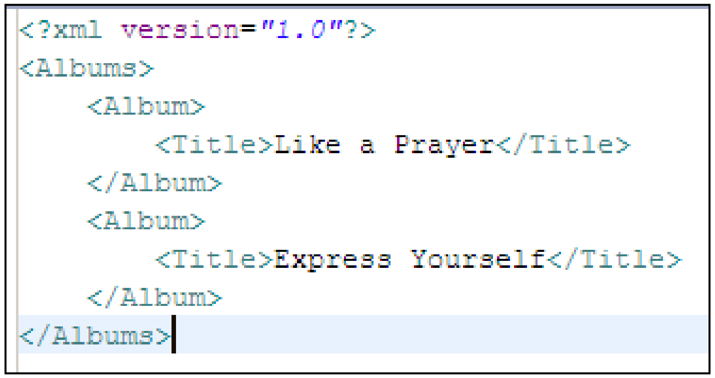
- Use methods:
  - getOwnerDocument()
  - getParentNode()
  - getChildNodes()
  - getFirstChild()
  - getLastChild()
  - getPreviousSibling()
  - getNextSibling()
  - getAttributes()
  - Element interface only:
  - getElementsByTagName ()
  - getElementsByTagNameNS()



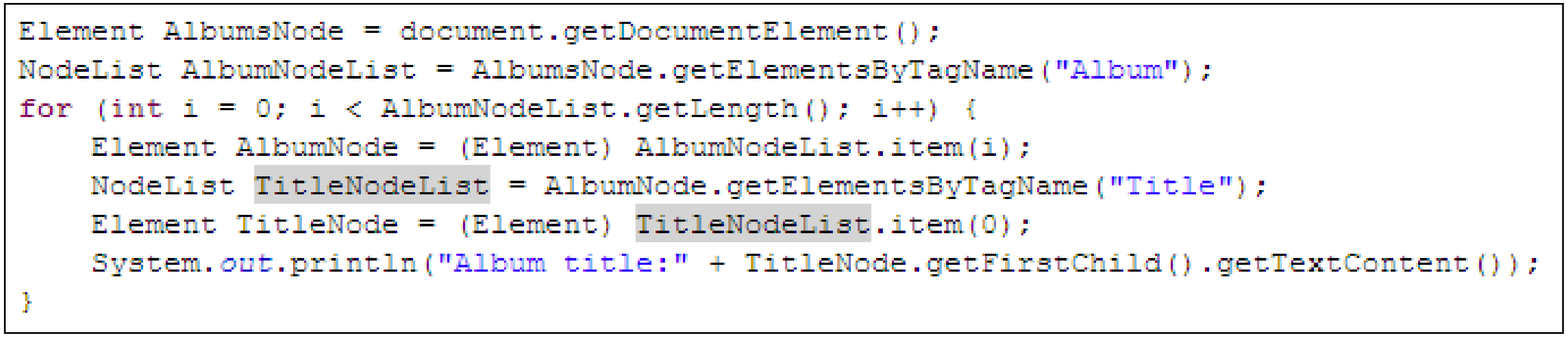
```
Node
  S F ELEMENT_NODE : short
  S F ATTRIBUTE_NODE : short
  S F TEXT_NODE : short
  S F CDATA_SECTION_NODE : short
  S F ENTITY_REFERENCE_NODE : short
  S F ENTITY_NODE : short
  S F PROCESSING_INSTRUCTION_NODE : short
  S F COMMENT_NODE : short
  S F DOCUMENT_NODE : short
  S F DOCUMENT_TYPE_NODE : short
  S F DOCUMENT_FRAGMENT_NODE : short
  S F NOTATION_NODE : short
  getNodeName()
  getNodeValue()
  setNodeValue(String)
  getNodeType()
  getParentNode()
  getChildNodes()
  getFirstChild()
  getLastChild()
  getPreviousSibling()
  getNextSibling()
  getAttributes()
  getOwnerDocument()
```

Traversing the DOM (cont.)

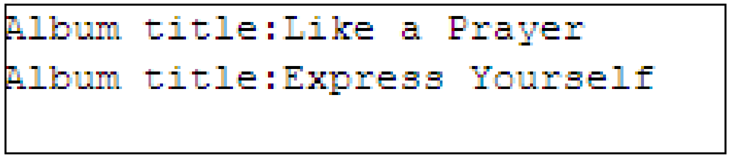
- Example: Using getElementsByTagName()



```xml
<?xml version="1.0"?>
<Albums>
    <Album>
        <Title>Like a Prayer</Title>
    </Album>
    <Album>
        <Title>Express Yourself</Title>
    </Album>
</Albums>
```



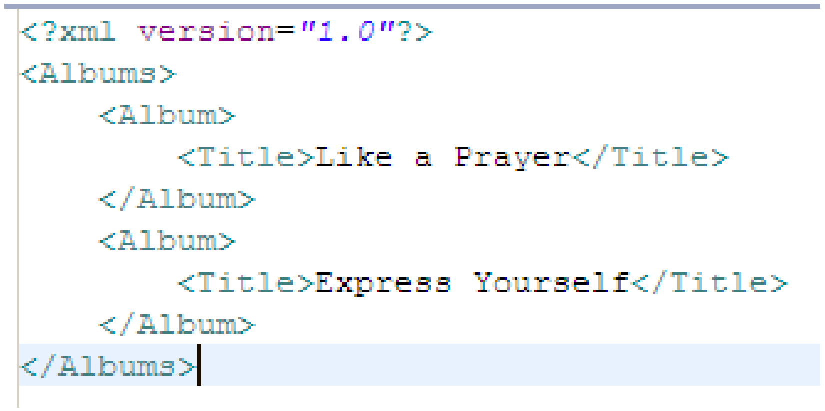
```java
Element AlbumsNode = document.getDocumentElement();
NodeList AlbumNodeList = AlbumsNode.getElementsByTagName("Album");
for (int i = 0; i < AlbumNodeList.getLength(); i++) {
    Element AlbumNode = (Element) AlbumNodeList.item(i);
    NodeList TitleNodeList = AlbumNode.getElementsByTagName("Title");
    Element TitleNode = (Element) TitleNodeList.item(0);
    System.out.println("Album title:" + TitleNode.getFirstChild().getTextContent());
}
```



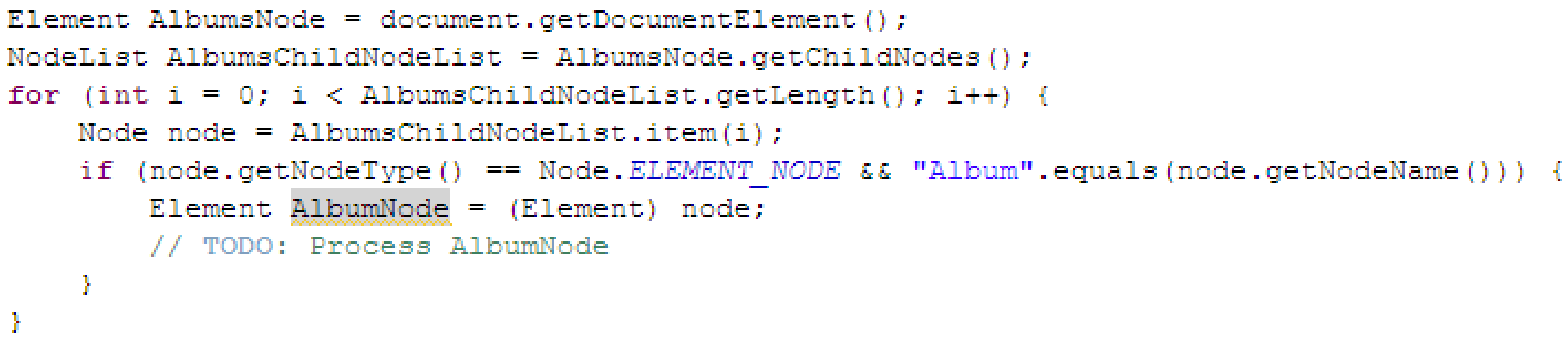
```
Album title:Like a Prayer
Album title:Express Yourself
```

Traversing the DOM (cont.)

- Example: Using getChildNodes(), getNodeType() and getNodeName()

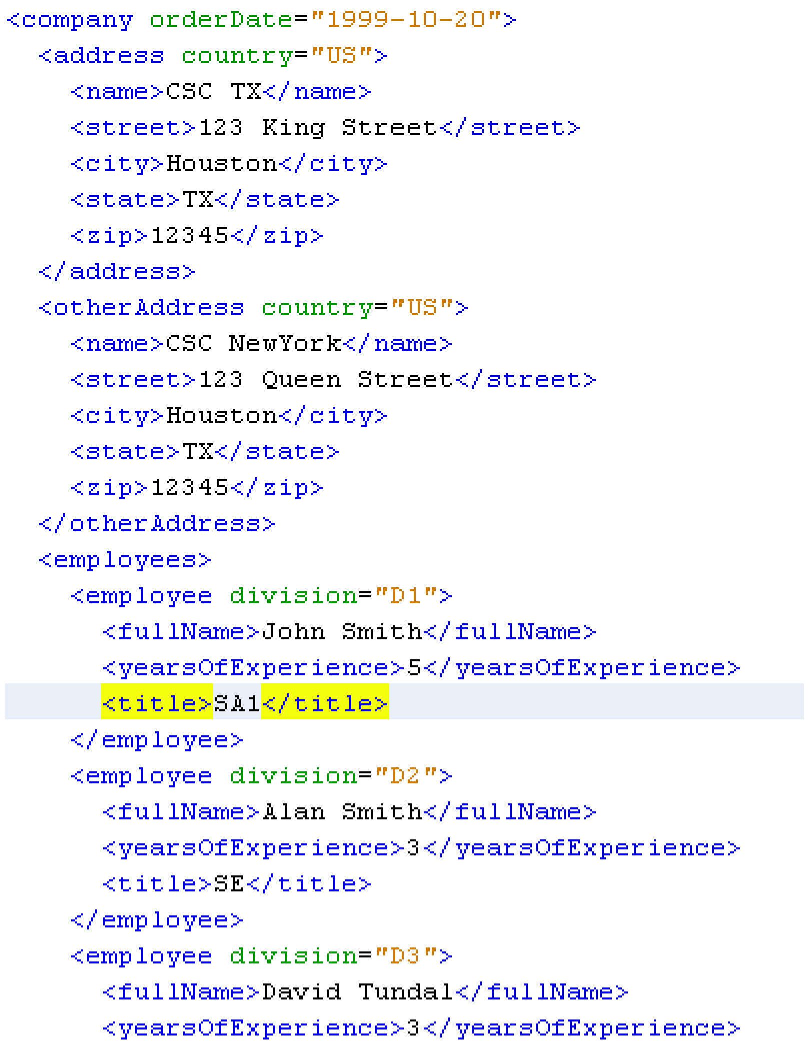


```xml
<?xml version="1.0"?>
<Albums>
    <Album>
        <Title>Like a Prayer</Title>
    </Album>
    <Album>
        <Title>Express Yourself</Title>
    </Album>
</Albums>
```



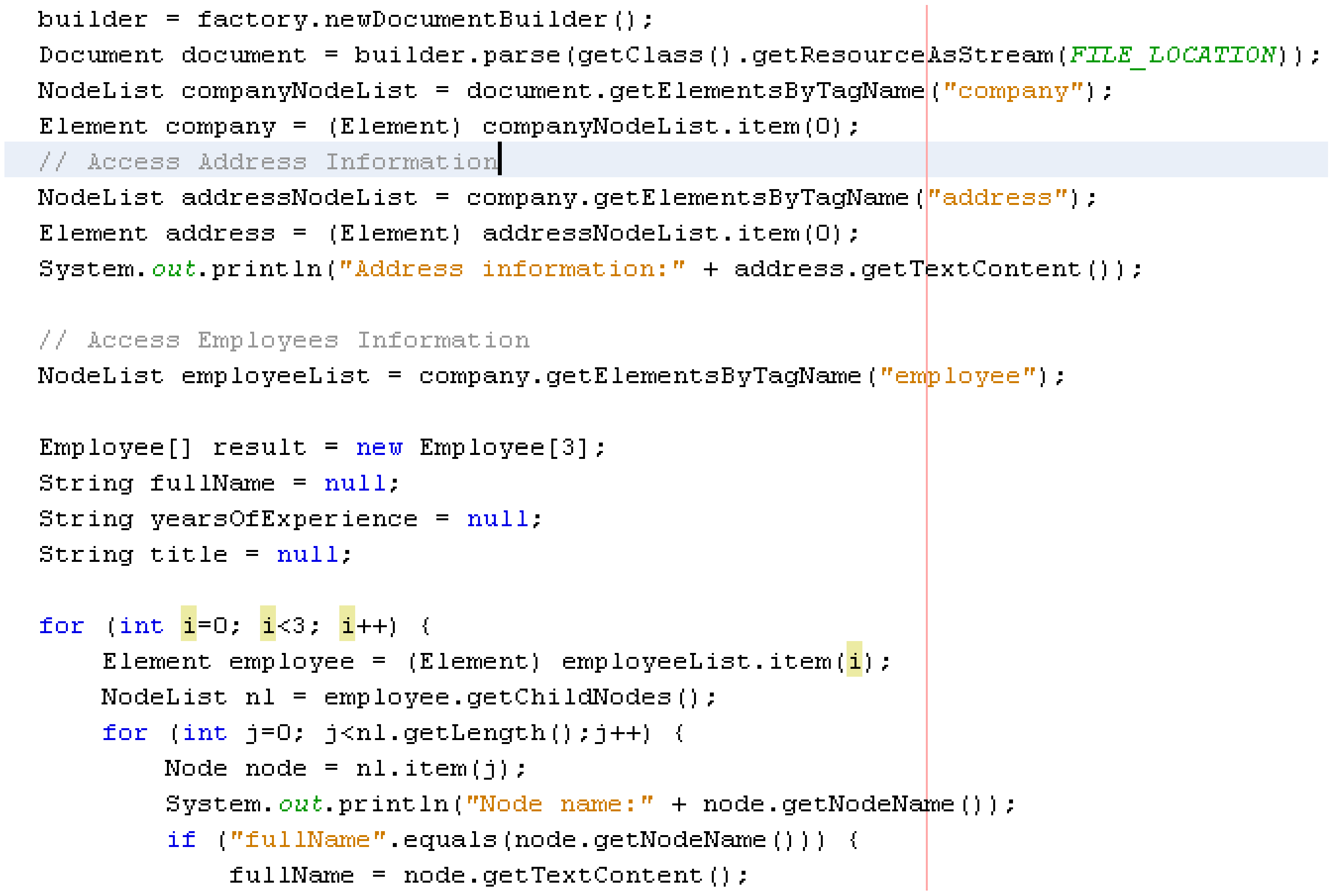
```java
Element AlbumsNode = document.getDocumentElement();
NodeList AlbumsChildNodeList = AlbumsNode.getChildNodes();
for (int i = 0; i < AlbumsChildNodeList.getLength(); i++) {
    Node node = AlbumsChildNodeList.item(i);
    if (node.getNodeType() == Node.ELEMENT_NODE && "Album".equals(node.getNodeName())) {
        Element AlbumNode = (Element) node;
        // TODO: Process AlbumNode
    }
}
```

Example



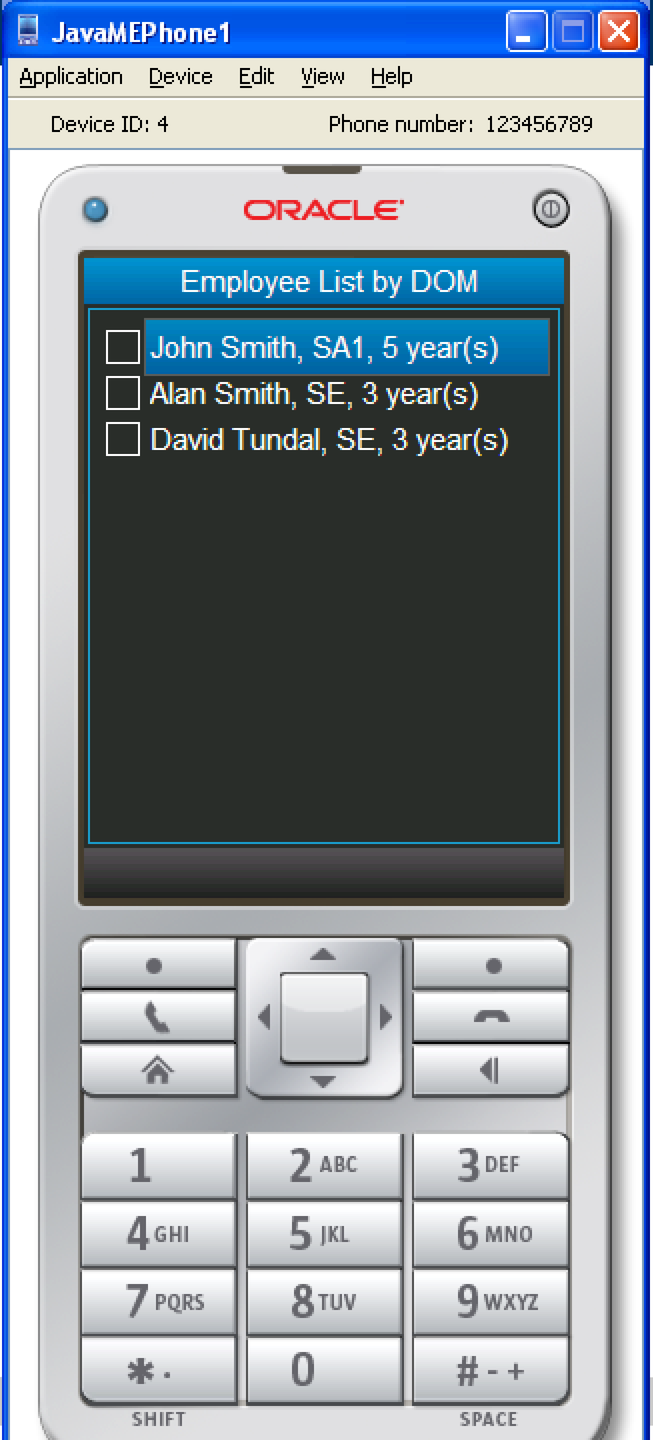
```xml
<company orderDate="1999-10-20">
  <address country="US">
    <name>CSC TX</name>
    <street>123 King Street</street>
    <city>Houston</city>
    <state>TX</state>
    <zip>12345</zip>
  </address>
  <otherAddress country="US">
    <name>CSC NewYork</name>
    <street>123 Queen Street</street>
    <city>Houston</city>
    <state>TX</state>
    <zip>12345</zip>
  </otherAddress>
  <employees>
    <employee division="D1">
      <fullName>John Smith</fullName>
      <yearsOfExperience>5</yearsOfExperience>
      <title>SA1</title>
    </employee>
    <employee division="D2">
      <fullName>Alan Smith</fullName>
      <yearsOfExperience>3</yearsOfExperience>
      <title>SE</title>
    </employee>
    <employee division="D3">
      <fullName>David Tundal</fullName>
      <yearsOfExperience>3</yearsOfExperience>
```

Example (cont.)



```java
builder = factory.newDocumentBuilder();
Document document = builder.parse(getClass().getResourceAsStream(FILE_LOCATION));
NodeList companyNodeList = document.getElementsByTagName("company");
Element company = (Element) companyNodeList.item(0);
// Access Address Information
NodeList addressNodeList = company.getElementsByTagName("address");
Element address = (Element) addressNodeList.item(0);
System.out.println("Address information:" + address.getTextContent());

// Access Employees Information
NodeList employeeList = company.getElementsByTagName("employee");

Employee[] result = new Employee[3];
String fullName = null;
String yearsOfExperience = null;
String title = null;

for (int i=0; i<3; i++) {
    Element employee = (Element) employeeList.item(i);
    NodeList nl = employee.getChildNodes();
    for (int j=0; j<nl.getLength();j++) {
        Node node = nl.item(j);
        System.out.println("Node name:" + node.getNodeName());
        if ("fullName".equals(node.getNodeName())) {
            fullName = node.getTextContent();
```

Example (cont.)



Device ID: 4 Phone number: 123456789

Employee List by DOM

- John Smith, SA1, 5 year(s)
- Alan Smith, SE, 3 year(s)
- David Tundal, SE, 3 year(s)

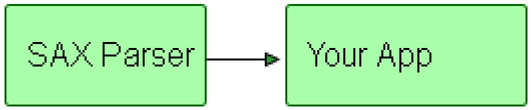
XML Programming Using StAX

StAX

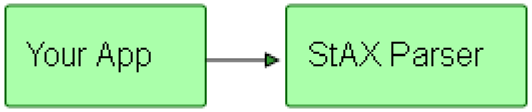
- A popular programming model for sequential XML processing
- It’s like SAX API which use event model, with some main differences:
  - StAX is a "pull" API. SAX is a "push" API.
  - StAX can do both XML reading and writing. SAX can only do XML reading.

StAX - "Pull" vs. "Push" Style API

- SAX is a push style API.
- The SAX parser iterates through the XML and calls methods on the handler object provided by you.

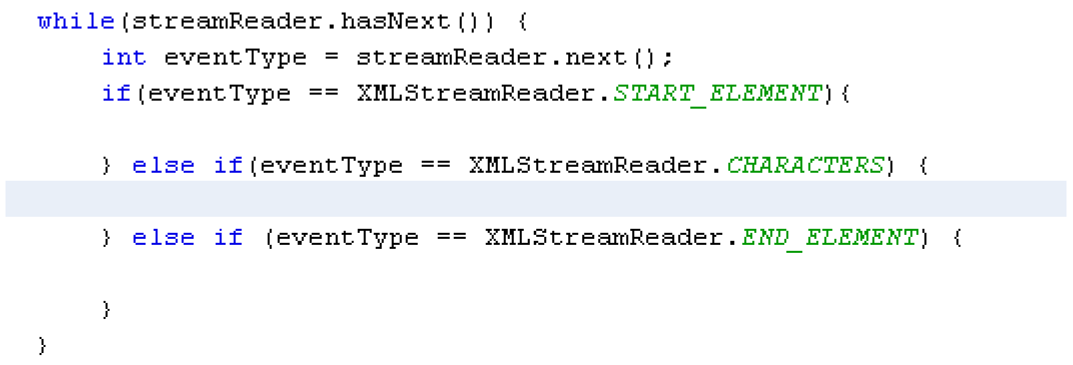


- StAX is a pull style API.
- Control your code to move the StAX parser from item to item in the XML file yourself.



Steps to writing StAX

- Create the input factory: XMLInputFactory inputFactory = XMLInputFactory.newInstance();
- Get Input Stream from file system: InputStream in = getClass().getResourceAsStream(FILE_LOCATION);
- Create XMLEventReader object: XMLStreamReader streamReader = inputFactory.createXMLStreamReader(in);
- Navigate through the list of event:



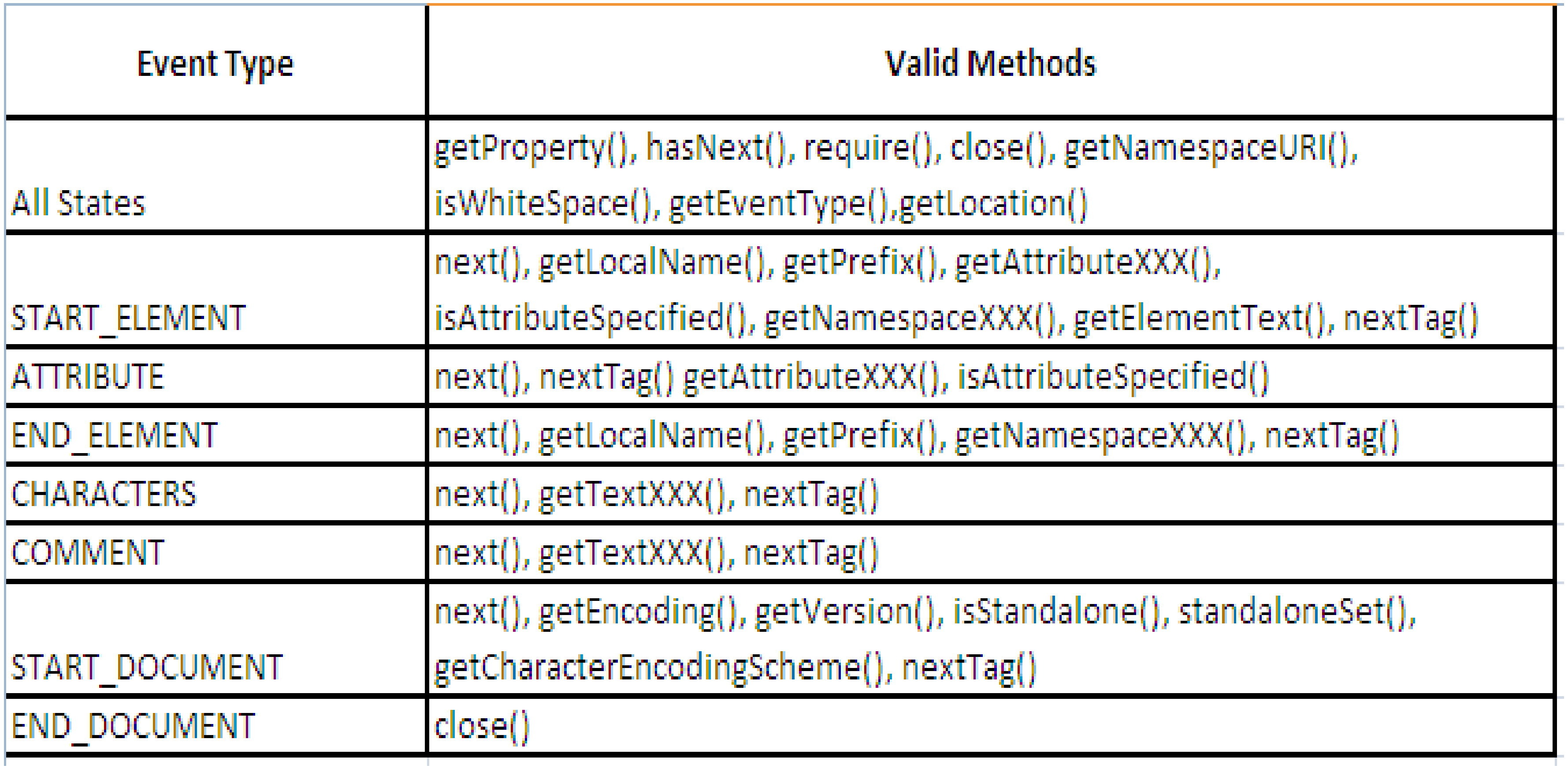
```
while(streamReader.hasNext()) {
    int eventType = streamReader.next();
    if(eventType == XMLStreamReader.START_ELEMENT){

    } else if(eventType == XMLStreamReader.CHARACTERS) {

    } else if (eventType == XMLStreamReader.END_ELEMENT) {

    }
}
```

List of StAX events

- START_ELEMENT: Indicates an event is a start element
- END_ELEMENT: Indicates an event is an end element
- CHARACTERS: Indicates an event is characters
- COMMENT: Indicates an event is a comment
- START_DOCUMENT: Indicates an event is a start document
- END_DOCUMENT: Indicates an event is an end document
- ENTITY_REFERENCE: Indicates an event is an entity reference
- ATTRIBUTE: Indicates an event is an attribute

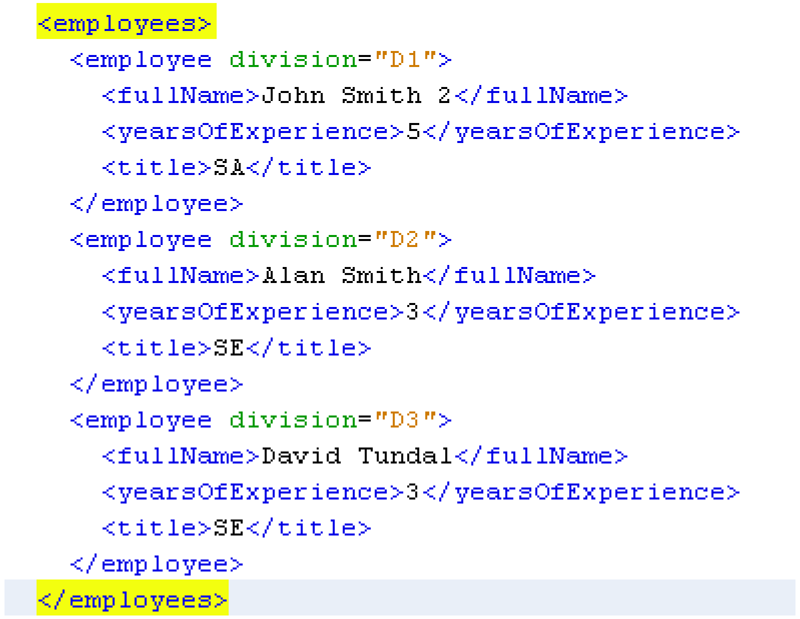
- And more ...

StAX methods



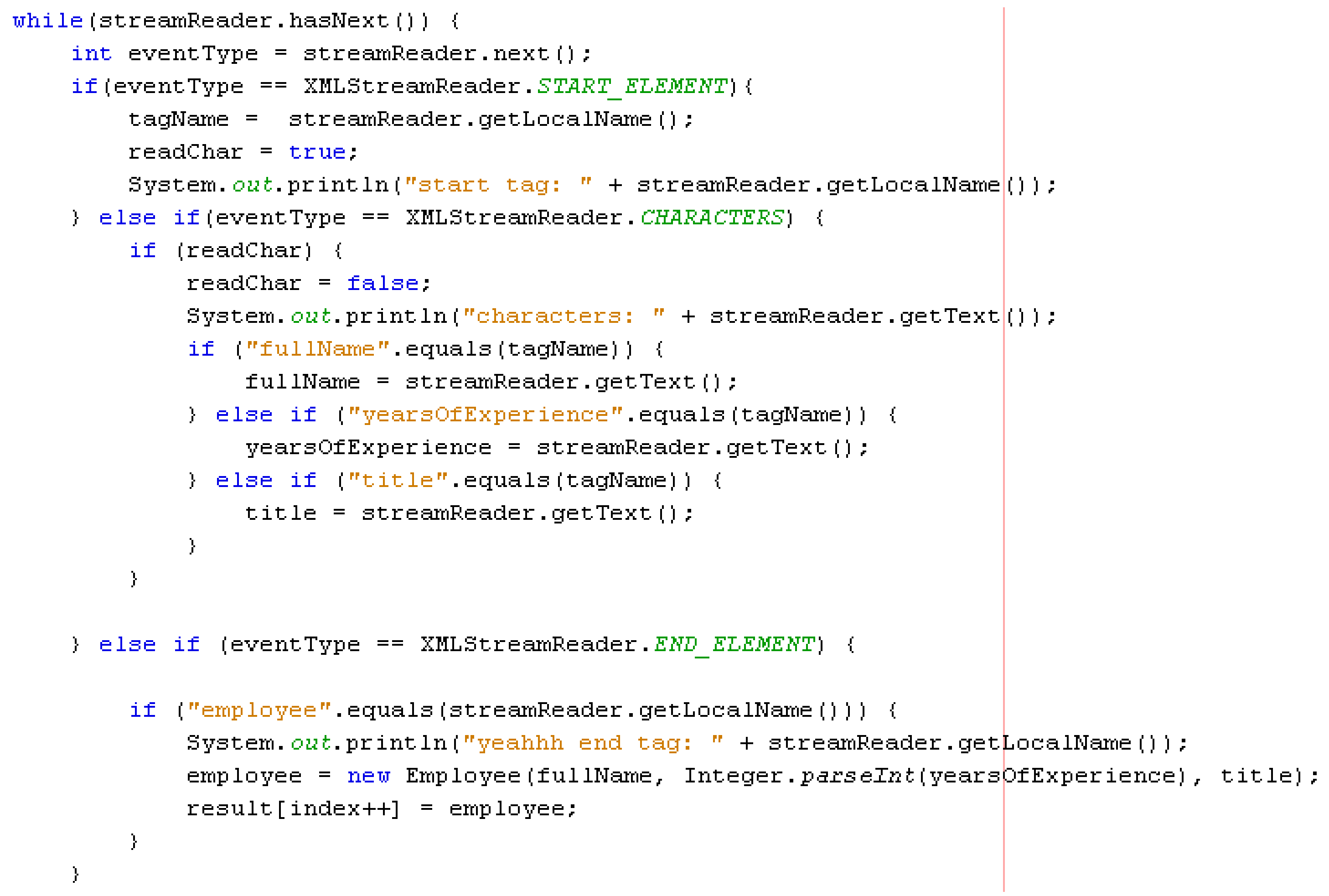
Event Type	Valid Methods
All States	getProperty(), hasNext(), require(), close(), getNamespaceURI(), isWhiteSpace(), getEventType(),getLocation()
START_ELEMENT	next(), getLocalName(), getPrefix(), getAttributeXXX(), isAttributeSpecified(), getNamespaceXXX(), getElementText(), nextTag()
ATTRIBUTE	next(), nextTag() getAttributeXXX(), isAttributeSpecified()
END_ELEMENT	next(), getLocalName(), getPrefix(), getNamespaceXXX(), nextTag()
CHARACTERS	next(), getTextXXX(), nextTag()
COMMENT	next(), getTextXXX(), nextTag()
START_DOCUMENT	next(), getEncoding(), getVersion(), isStandalone(), standaloneSet(), getCharacterEncodingScheme(), nextTag()
END_DOCUMENT	close()

Example



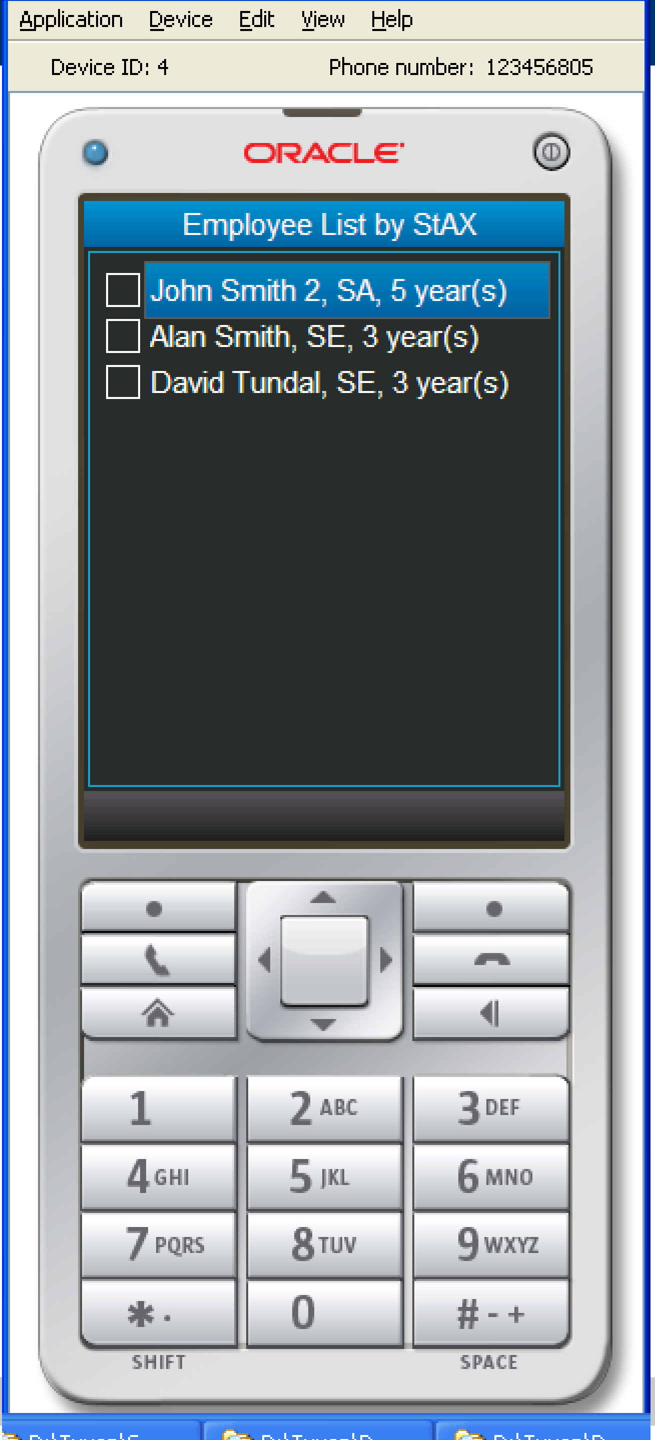
```
<employees>
  <employee division="D1">
    <fullName>John Smith 2</fullName>
    <yearsOfExperience>5</yearsOfExperience>
    <title>SA</title>
  </employee>
  <employee division="D2">
    <fullName>Alan Smith</fullName>
    <yearsOfExperience>3</yearsOfExperience>
    <title>SE</title>
  </employee>
  <employee division="D3">
    <fullName>David Tundal</fullName>
    <yearsOfExperience>3</yearsOfExperience>
    <title>SE</title>
  </employee>
</employees>
```

Example (cont.)



```
while(streamReader.hasNext()) {
    int eventType = streamReader.next();
    if(eventType == XMLStreamReader.START_ELEMENT){
        tagName =  streamReader.getLocalName();
        readChar = true;
        System.out.println("start tag: " + streamReader.getLocalName());
    } else if(eventType == XMLStreamReader.CHARACTERS) {
        if (readChar) {
            readChar = false;
            System.out.println("characters: " + streamReader.getText());
            if ("fullName".equals(tagName)) {
                fullName = streamReader.getText();
            } else if ("yearsOfExperience".equals(tagName)) {
                yearsOfExperience = streamReader.getText();
            } else if ("title".equals(tagName)) {
                title = streamReader.getText();
            }
        }

    } else if (eventType == XMLStreamReader.END_ELEMENT) {

        if ("employee".equals(streamReader.getLocalName())) {
            System.out.println("yeahhh end tag: " + streamReader.getLocalName());
            employee = new Employee(fullName, Integer.parseInt(yearsOfExperience), title);
            result[index++] = employee;
        }
    }
```

Example (cont.)



Employee List by StAX

- John Smith 2, SA, 5 year(s)
- Alan Smith, SE, 3 year(s)
- David Tundal, SE, 3 year(s)

Excercises

Exercise 1

- Install Java ME SDK 3.2 on Eclipse Juno/Net Beans 7.3
- Given Employee_1.xml, please parse and show it on GUI using SAX
- Send the source code and screen shot to tnguyen256@csc.com

Exercise 2

- Install Java ME SDK 3.2 on Eclipse Juno/Net Beans 7.3
- Given Employee_2.xml, please parse and show it on GUI using DOM
- Send the source code and screen shot to tnguyen256@csc.com

Exercise 3

- Install Java ME SDK 3.2 on Eclipse Juno/Net Beans 7.3
- Given Employee_3.xml, please parse and show it on GUI using StAX
- Send the source code and screen shot to tnguyen256@csc.com

Other third-party libraries

Other third-party libraries

- kXML
- Home page: http://kxml.sourceforge.net/

Q&A

Contact: Tuyen Nguyen

Mobile +84 983 830 860 | tnguyen256@csc.com