XML for Java Developers

Tuyen Nguyen, PSE
Team Lead

Warm up - Introductions

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

Course Objectives

- To gain a better understanding of XML, DTD, XSL
- Practice creating XML documents with a text editor
- Learn JAXP and JAXB
- To extract information using JAXP- DOM and SAX
- To bind Java object and XML using JAXB

Course Outline

- Module 1: Understanding XML and SAX
  - XML documents
  - Data constraints with DTD
  - Workshop
  - Q & A
  - XML Processing using SAX (Simple API for XML)
  - Workshop
  - Q & A
- Module 2: DOM and JAXB – Java Architecture for XML Binding
  - XML Processing using DOM (Document Object Model)
  - XML Schema Definition
  - Mapping XML into Java object with JAXB
  - Others library: XStream
  - Q & A

Course Audience and Prerequisite

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

Assessment Disciplines

- Class Participation: 60%
- Assignment: 40%
- Passing Scores: 70%

Course Duration- 6 hours

- Duration: 3 hours / module
- Break time: 15 minutes / module
- Total module: 2

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
- Xstream home page. http://xstream.codehaus.org/

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

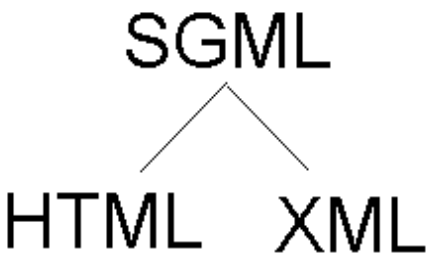
Set Up Environment

- To complete the course, your PC must have:
  - JDK 5+
  - Eclipse IDE
  - Network connection

Module 1: Understanding XML and SAX

What is XML?

- *XML- eXtensible Markup Language*
- 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>Chính trị - Xã hội</title>
    <description>Chính trị - 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;Thac
      <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
```

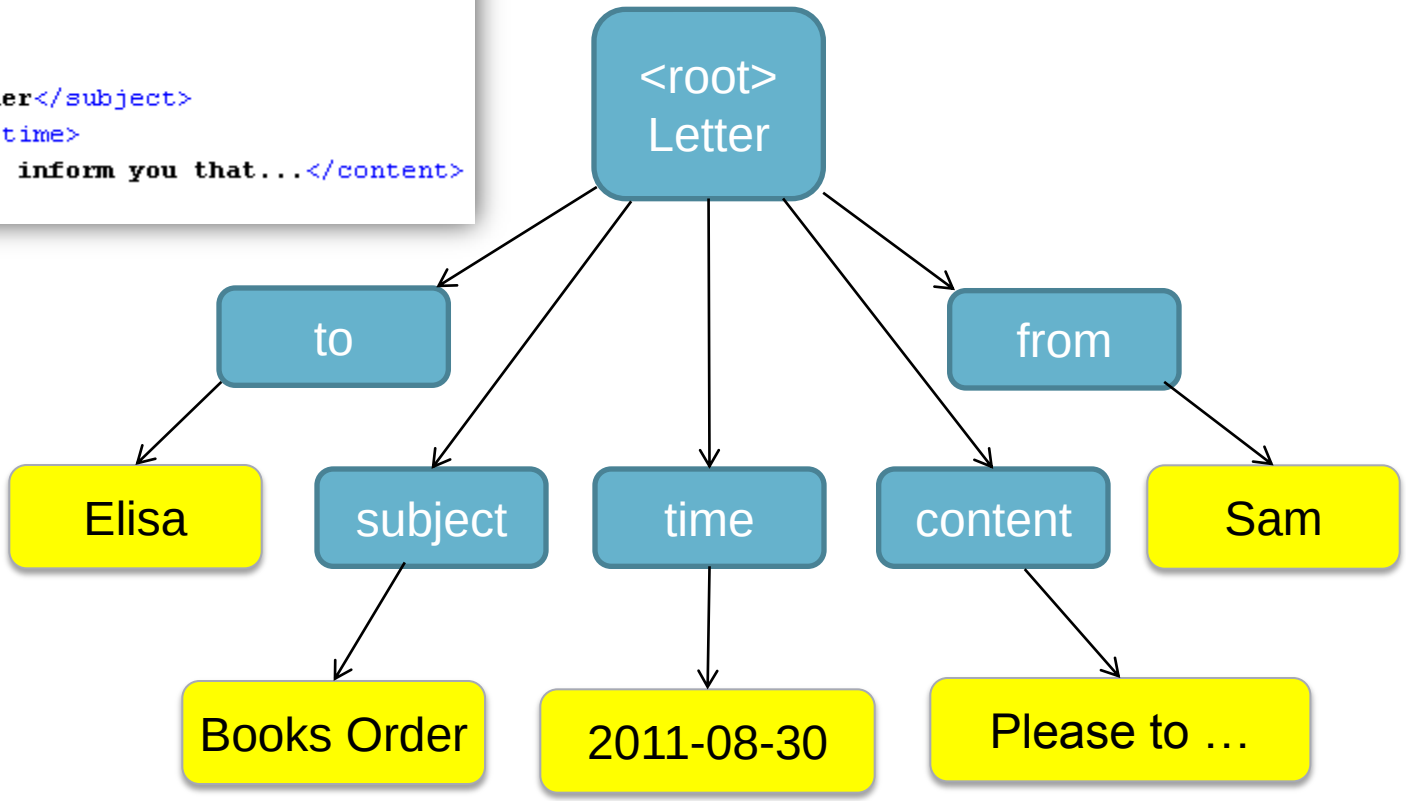
- XML Can be Used to Exchange Data (Web Services, RSS…)
- XML Can be Used to Store Data (configuration files of Struts, Spring Framework)
- XML Can be Used to Create New Languages (Ant build files, XML Schema…)

XML Document is a tree

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

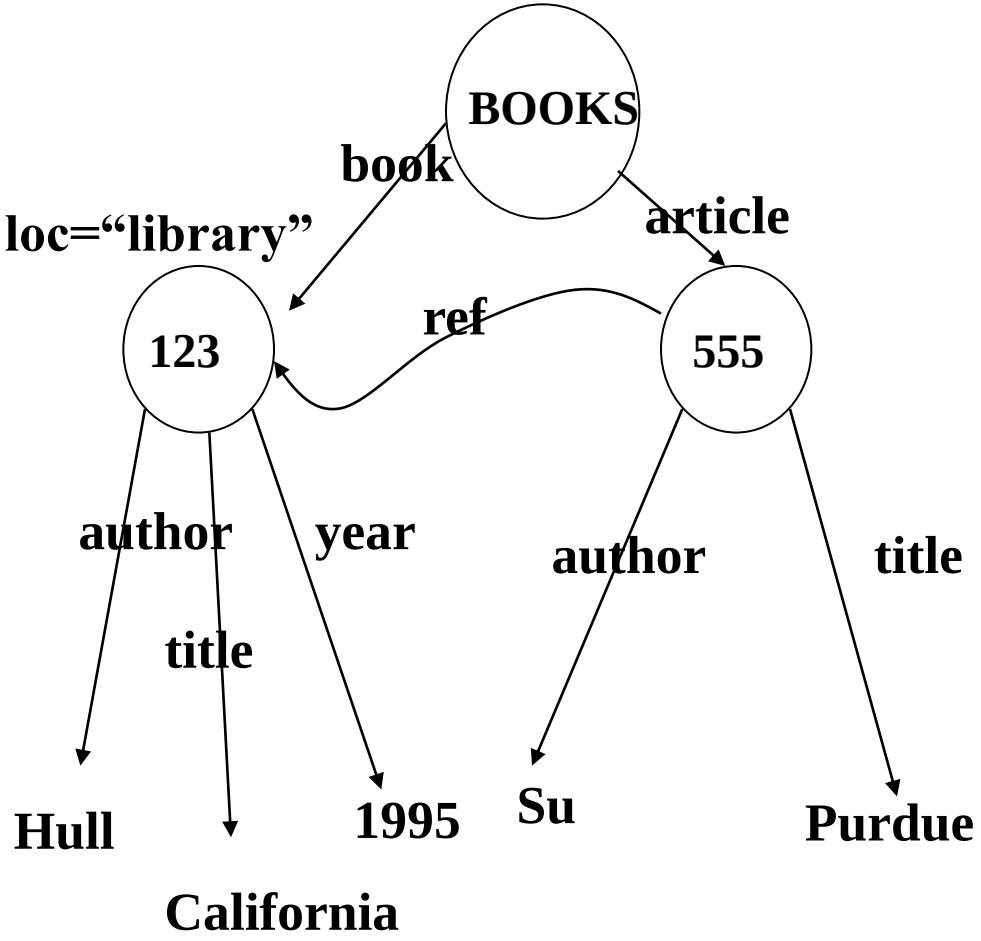
```
<?xml version = "1.0" ?>

<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: <…®

  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

sample5.xml

```
<?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

sample6.xml

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

<message>          start tag
    <important-flag />          empty-element tag
    <text>Where are you?</text>
</message>          end tag
```

Note: Tags are case sensitive

- 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)

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>
```

<	<	less than
>	>	greater than
&	&	ampersand
'	'	apostrophe
"	"	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 < (<) > (>) & (&) ' (‘) " (“)

sample10.xml

```
<?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

sample12.xml

```
<?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

sample9.xml | sample8.xml

```
<?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

Java API for XML Processing (JAXP)

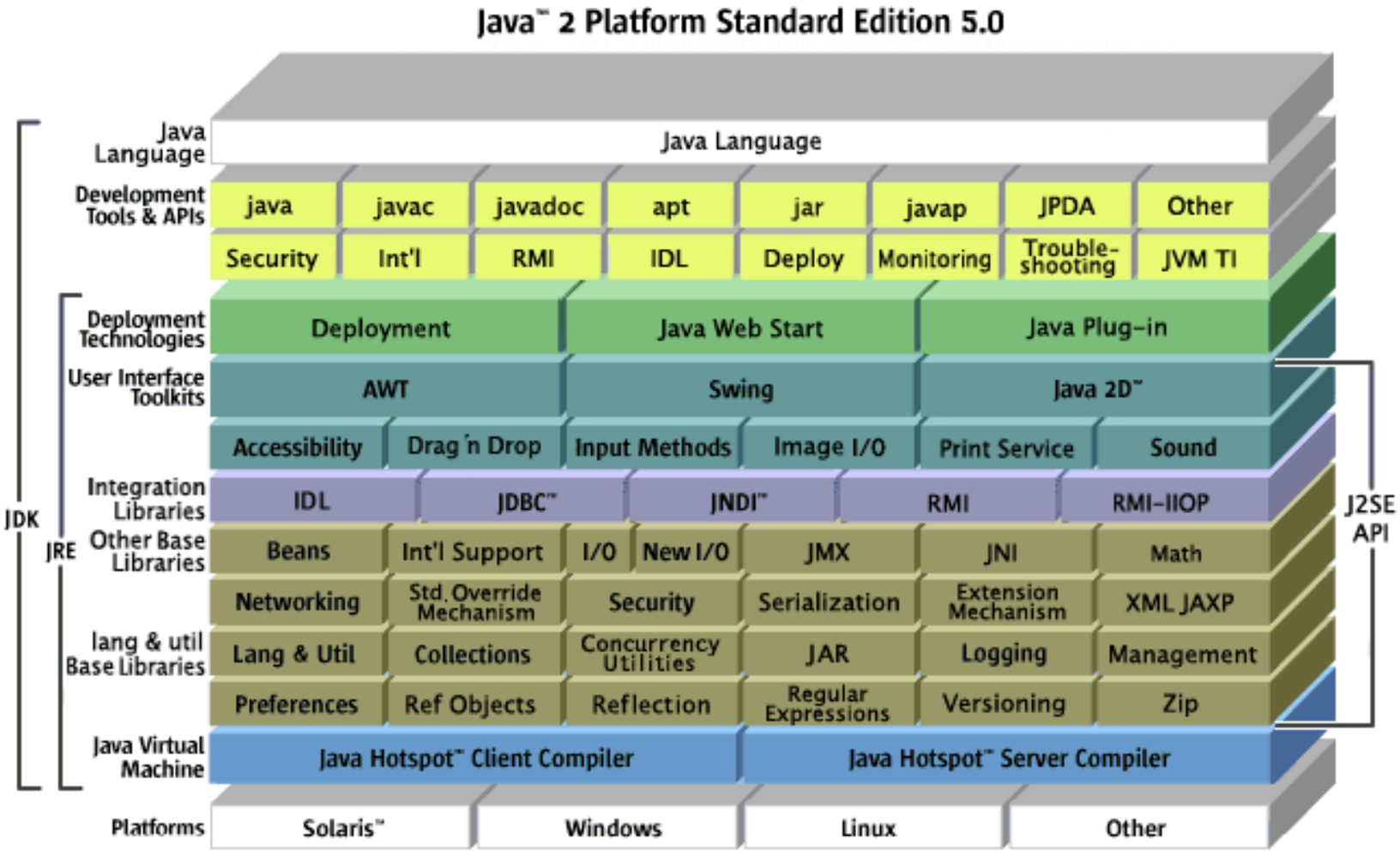
JAXP Overview

- The Java API for XML Processing (JAXP) is used to
  - Parse XML documents as a stream of events (SAX) or to build an object representation of it (DOM)
  - Convert XML documents to other XML documents or to other formats (XSLT)

  Note: We will focus on parsing XML using SAX and DOM methods in this course (Other methods like StAX, JDOM… is not mentioned)

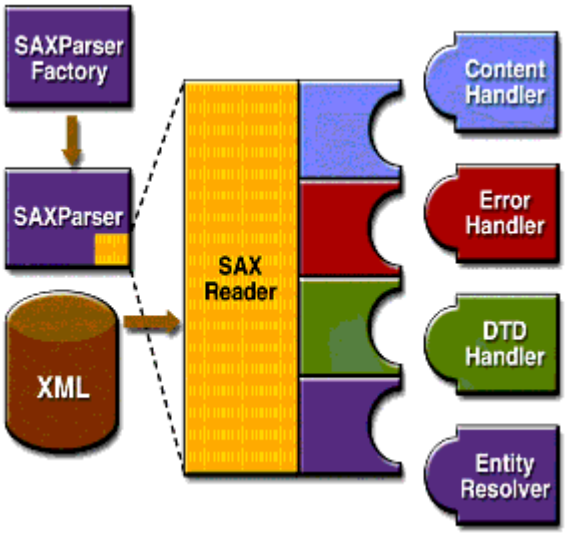
- J2SDK 1.4 includes JAXP 1.1, JDK 5 includes JAXP 1.3, JDK 6 includes JAXP 1.4 (JAXP 1.3 with StAX)

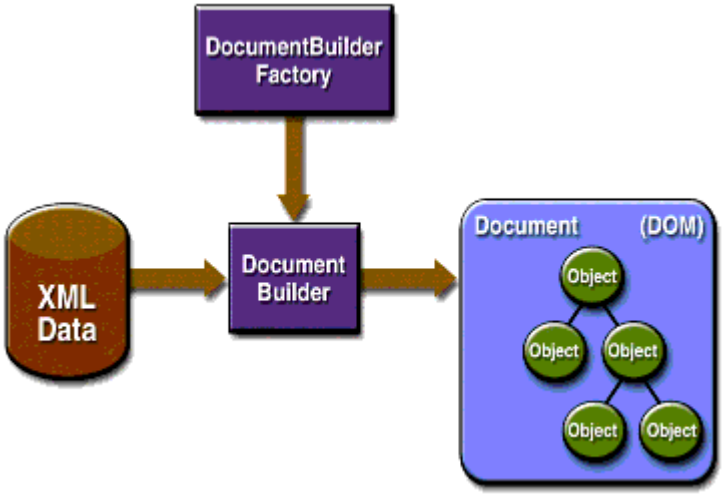
What is JAXP? (cont.)



Parsing XML Documents

- Simple API for XML (SAX)
- Document Object Model (DOM)





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

http://download.oracle.com/javase/1.4.2/docs/api/org/xml/sax/helpers/DefaultHandler.html

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 version = "1.0" ?>
<Employee>
<Emp_Id> E-001 </Emp_Id>
<Emp_Name> Vinod </Emp_Name>
<Emp_E-mail> Vinod1@yahoo.com </Emp_E-mail>
</Employee>
```

Console | Problems | Javadoc | Declaration | Properties

```
<terminated> TestXmlSAX [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe
Start Element: Employee
Start Element: Emp_Id
Content:  E-001
End Element: Emp_Id
Start Element: Emp_Name
Content:  Vinod
End Element: Emp_Name
Start Element: Emp_E-mail
Content:  Vinod1@yahoo.com
End Element: Emp_E-mail
End Element: Employee
```

Exercise 1: You are given a XML file about the company information. Use SAX parser to output the Addresses and Employees information

- Input: the XML file
- Output: the result on the console screen
- Time: 30’
- Send the output along with your source code to tnguyen256@csc.com or save it to \\qc-training\Freshers\XML\Exercise1
  - Print your full screen containing the result
  - Hint
- Deadline: Jul 12 2013
- Your name should be in the email subject or the folder name