Module 2:
DOM – Document Object Model
JAXB – Java Architecture for XML Binding

Tuyen Nguyen, PSE
Team Lead

Module 2 Agenda

- Document Object Model (DOM)
- XML Schema Definition (XSD)
- Java Architecture for XML Binding (JAXB)
- Introduction XStream

Document Object Model

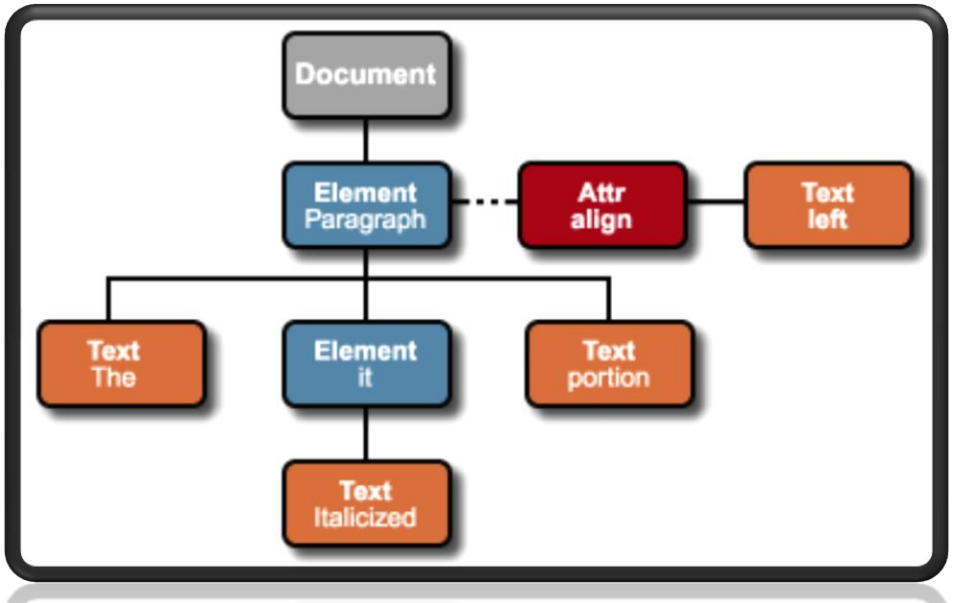
- A W3C standard for platform- and language-neutral dynamic access and update of the content, structure, and style of XML documents
- Is implemented in a wide variety of languages, including Java, JavaScript, C++, dotNet, …
- Presents an XML document as a tree-structure (a node tree), with the elements, 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.

dom1.xml

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



Document Object Model (cont.)

- DOM vs. JDOM:
  - DOM is designed for XML documents (mixed content), JDOM is for XML data
  - In JDOM, after you navigate to an element that contains text, invoking a text() returns its content
  - When processing a DOM, you must inspect the list of subelements to "put together" the text of the node
  - JDOM is an effort to adapt the DOM API for Java, providing a more natural and easy-to-use interface. It's a Java 3rd party library

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();

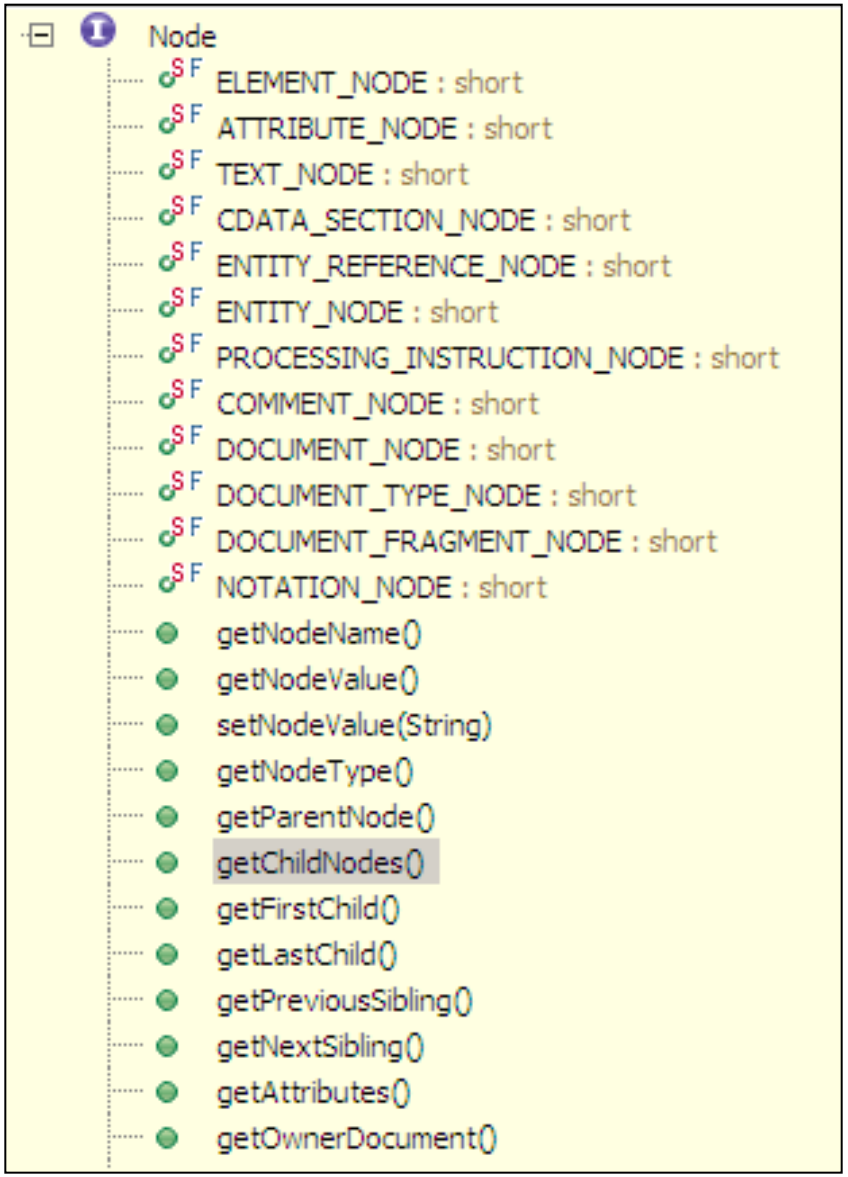
- Examine various properties of the node

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()



Traversing the DOM (cont.)

- Example: Using getElementsByTagName()

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

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

```
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
    }
}
```

XPath

- XPath is a language for addressing, searching, and matching pieces of the document. A W3C Recommendation
- XPath 1.0 is available from JAXP 1.3 (JDK 5)
- Useful article: “The Java XPath API”. http://www.ibm.com/developerworks/library/x-javaxpathapi.html
- Example: XPath for a document using namespaces

```
<?xml version="1.0"?>
<Albums xmlns="urn:yahoo:srchmm">
    <Album>
        <Title>Like a Prayer</Title>
    </Album>
    <Album>
        <Title>Express Yourself</Title>
    </Album>
</Albums>
```

XPath (cont.)

```
public static class MyNamespaceContext implements NamespaceContext {
    public String getNamespaceURI(String prefix) {
        if (prefix == null) throw new NullPointerException("Null prefix");
        else if ("pre".equals(prefix)) return "urn:yahoo:srchmm";
        else if ("xml".equals(prefix)) return XMLConstants.XML_NS_URI;
        return XMLConstants.NULL_NS_URI;
    }
    // This method isn't necessary for XPath processing.
    public String getPrefix(String uri) {
        throw new UnsupportedOperationException();
    }
    // This method isn't necessary for XPath processing either.
    public Iterator getPrefixes(String uri) {
        throw new UnsupportedOperationException();
    }
}

public static void main(String[] args) throws Exception {

    DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
    factory.setNamespaceAware(true);
    DocumentBuilder builder = factory.newDocumentBuilder();
    Document document = builder.parse(new File(args[0]));

    XPath xpath = XPathFactory.newInstance().newXPath();
    xpath.setNamespaceContext(new MyNamespaceContext());
    String expression = "/pre:Albums/pre:Album/pre:Title";
    NodeList titleNodes = (NodeList) xpath.evaluate(expression, document, XPathConstants.NODESET);
    for (int i = 0; i < titleNodes.getLength(); i++) {
        System.out.println(titleNodes.item(i).getTextContent());
    }
}
```

Exercise 2: You are given a XML file about the company information. Print out the addresses of the company and all employees' information using DOM.

- Input: the XML file
- Output: the result on the console screen
- Time: 30'
- Send your results to tnguyen256@csc.com or save it to : \\qc-training\Freshers\XML\Exercise2
- Your name should be in the email subject or the folder name
- Deadline: Jul 12 2013
- Hints

Note: the same requirement as Exercise 2, but by using DOM

Points to Remember

- Two ways to parse XML documents:
  - Simple API for XML (SAX): fast and efficient, event-driven model
  - Document Object Model (DOM): presents an XML document as a tree-structure (a node tree). DOM consumes more memory than SAX

Feature	SAX	DOM
Ease of Use	Medium	High
XPath Capability	Not supported	Supported
CPU and Memory Efficiency	Good	Varies
Read XML	Supported	Supported
Write XML	Not supported	Supported
Create, Read, Update, Delete	Not supported	Supported

Q&A

Contact: Tuyen Nguyen

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

XML Schema Definition

- XML Schema Definition (or XSD) describes the structure of an XML document
- XML Schemas are the successors of DTDs with more powerful features:
  - written in XML
  - support data types
  - put restrictions on text content
  - and more control over structure and content of xml

Referring to a schema

- To refer to a DTD in an XML document, the reference goes *before* the root element:
  - <?xml version="1.0"?>
    <!DOCTYPE rootElement SYSTEM "*url*">
    <rootElement> ... </rootElement>
- To refer to an XML Schema in an XML document, the reference goes *in* the root element:
  - <?xml version="1.0"?>
    <rootElement
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    (The XML Schema Instance reference is required)
    xsi:noNamespaceSchemaLocation="*url*.xsd">
    (This is the location of the XML file)
    ...
    </rootElement>

The XSD document

- Since the XSD is written in XML, it starts like this:
  - <?xml version="1.0"?>
    <xs:schema
    xmlns:xs="http://www.w3.rg/2001/XMLSchema">
- The file extension is .xsd
- The root element is <schema>

“Simple” and “Complex” elements

- A “simple” element is one that contains text and nothing else
  - A simple element cannot have attributes
  - A simple element cannot contain other elements
  - A simple element cannot be empty
  - However, the text can be of many different types, and may have various restrictions applied to it
- **If an element isn’t simple, it’s “complex”**
  - A complex element may have attributes
  - A complex element may be empty, or it may contain text, other elements, or both text and other elements

Defining a simple element

- A simple element is defined as
  <xs:element name="*name*" type="*type*" />
  where:
  - *name* is the name of the element
  - the most common values for *type* are

    xs:boolean xs:integer
    xs:date xs:string
    xs:decimal xs:time
- Other attributes a simple element may have:
  - default="*default value*" *if no other value is specified*
  - fixed="*value*" *no other value may be specified*

Defining an attribute

- Attributes themselves are always declared as simple types
- An attribute is defined as
  `<xs:attribute name="name" type="type" />`
  where:
  - *name* and *type* are the same as for xs:element
- Other attributes a simple element may have:
  - default="*default value*" *if no other value is specified*
  - fixed="*value*" *no other value may be specified*
  - use="optional" *the attribute is not required (default)*
  - use="required" *the attribute must be present*

Restriction

- The general form for putting a restriction on a text value is:
  - <xs:element name="*name*"> *(or* xs:attribute*)*
    <xs:restriction base="*type*">
    ... the restrictions ...
    </xs:restriction>
    </xs:element>
- For example:
  - <xs:element name="age">
    <xs:restriction base="xs:integer">
    <xs:minInclusive value="0">
    <xs:maxInclusive value="140">
    </xs:restriction>
    </xs:element>

Restrictions on numbers

- minInclusive -- number must be ≥ the given value
- minExclusive -- number must be > the given value
- maxInclusive -- number must be ≤ the given value
- maxExclusive -- number must be < the given value
- totalDigits -- number must have exactly value digits
- fractionDigits -- number must have no more than value digits after the decimal point

Restrictions on strings

- length -- contain exactly value characters
- minLength -- contain at least value characters
- maxLength -- contain no more than value characters
- Pattern -- the value is a regular expression
- whiteSpace -- tells what to do with whitespace
  - value="preserve" : Keep all whitespace
  - value="replace" : Change all whitespace characters to spaces
  - value="collapse" : Remove leading and trailing whitespace, and replace all sequences of whitespace with a single space

Enumeration

- An enumeration restricts the value to be one of a fixed set of values
- Example:

```
<xs:element name="season">
    <xs:simpleType>
        <xs:restriction  base="xs:string">
            <xs:enumeration value="Spring"/>
            <xs:enumeration value="Summer"/>
            <xs:enumeration value="Autumn"/>
            <xs:enumeration value="Fall"/>
            <xs:enumeration value="Winter"/>
        </xs:restriction>
    </xs:simpleType>
</xs:element>
```

Complex elements

- A complex element is defined as

```
<xs:element   name="name">
    <xs:complexType>
        ... information about the complex type...
    </xs:complexType>
</xs:element>
```

- Example:

```
<xs:element   name="person">
    <xs:complexType>
        <xs:sequence>
            <xs:element  name="firstName"  type="xs:string" />
            <xs:element  name="lastName"  type="xs:string" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

- <xs:sequence> says that elements must occur in this order
- Remember that attributes are always simple types

xs:sequence and xs:all

- xs:sequence defines that elements must occur in the order (see the previous example)
- xs:all allows elements to appear in any order
-
```
<xs:element   name="person">
    <xs:complexType>
        <xs:all>
            <xs:element  name="firstName"  type="xs:string" />
            <xs:element  name="lastName"  type="xs:string" />
        </xs:all>
    </xs:complexType>
 </xs:element>
```
- You can use minOccurs="n" and maxOccurs="n" to specify how many times an element may occur (default value is 1)
  - In xs:all, n may only be 0 or 1

Referencing

- Once you have defined an element or attribute (with name="…"), you can refer to it with ref="…"
- Example:
  -
```
<xs:element   name="person">
    <xs:complexType>
        <xs:all>
            <xs:element  name="firstName"  type="xs:string" />
            <xs:element  name="lastName"  type="xs:string" />
        </xs:all>
    </xs:complexType>
</xs:element>
```
  -
```
<xs:element  name="student"  ref="person“>
```

For more XSD References

- For more XSD References, please go to:
  - http://www.w3.org/XML/Schema
  - http://www.w3schools.com/schema/schema_elements_ref.asp

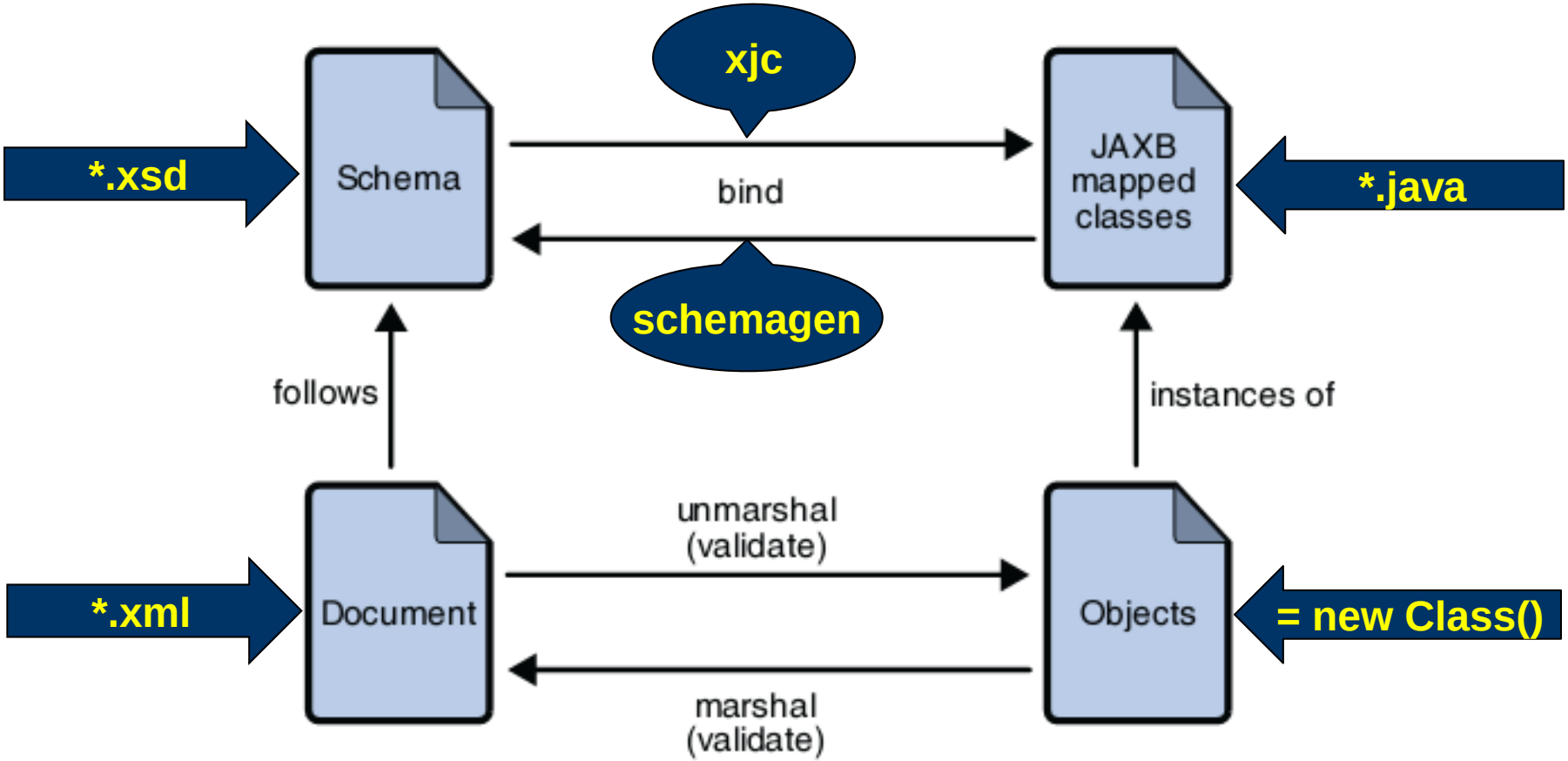
Lesson 3 Agenda

- XML Schema Definition (XSD)
- Java Architecture for XML Binding (JAXB)
- Introduction XStream

Java Architecture for XML Binding

- Java Architecture for XML Binding, or JAXB for short, provides a fast and convenient way to bind between XML schemas and Java representations
- JAXB's goal:
  - Easy to use, don't require SAX/DOM knowledge and just require a minimal XML knowledge
  - Can customize mapping between XML and Java
  - Follow standard design and naming conventions in generated Java
  - Deliver core functionality ASAP
  - Marshalling objects to XML and unmarshalling back to objects results in equivalent objects

Java Architecture for XML Binding



Java Architecture for XML Binding

- The JAXB *xjc* schema binding compiler transforms, or binds, a source XML schema to aset of JAXB content classes in the Java programming language

xjc [-d path …] schema

- The JAXB Schema Generator, *schemagen*, creates a schema file for each name space referenced in your Java classes

schemagen [-d path][java-source-iles]

Steps to generate Java classes from XSD

- Generating Java file from XSD file
  - Windows: %JAXB_HOME%\bin\xjc po.xsd
  - UNIX: %JAXB_HOME%/bin/xjc.sh po.xsd

```
<xsd:complexType name="USAddress">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="street" type="xsd:string"/>
    <xsd:element name="city" type="xsd:string"/>
    <xsd:element name="state" type="xsd:string"/>
    <xsd:element name="zip" type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
</xsd:complexType>
```

Steps to generate Java classes from XSD

```
<xsd:complexType name="USAddress">
  <xsd:sequence>
    <xsd:element name="name" type="xsd:string"/>
    <xsd:element name="street" type="xsd:string"/>
    <xsd:element name="city" type="xsd:string"/>
    <xsd:element name="state" type="xsd:string"/>
    <xsd:element name="zip" type="xsd:decimal"/>
  </xsd:sequence>
  <xsd:attribute name="country" type="xsd:NMTOKEN" fixed="US"/>
</xsd:complexType>
```

```
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(name = "USAddress", propOrder = {
    "name", "street", "city", "state", "zip"
})
public class USAddress {
    @XmlElement(required = true)
    protected String name;
    @XmlElement(required = true)
    protected String street;
    @XmlElement(required = true)
    protected String city;
    @XmlElement(required = true)
    protected String state;
    @XmlElement(required = true)
    protected BigDecimal zip;
    @XmlAttribute(name = "country")
    @XmlJavaTypeAdapter(CollapsedStringAdapter.class)
    @XmlSchemaType(name = "NMTOKEN")
    protected String country;
```

Steps to Unmarshalling XML to Java

- Create an Unmarshaller object

JAXBContext jc = JAXBContext.newInstance(“<package>");

Unmarshaller u = jc.createUnmarshaller();

- Map the Xml document to a Tree java object by calling

JAXBElement be = (JAXBElement)
***u.unmarshal(<xml_input>)**;*

<RootType root> = (<RootType>) be.getValue();

Steps to Marshalling Java to XML

- Create an Marshaller object

JAXBContext jc = JAXBContext.newInstance(“<package>");

Marshaller m = jc.createMarshaller();

- Map the Java object back to Xml document by calling

JAXBElement poe = (JAXBElement)
u.marshal(<rootElement>, <xml_input>);

An Example

```
try {
    JAXBContext jc = JAXBContext.newInstance("generated");
    Unmarshaller u = jc.createUnmarshaller();
    JAXBElement poe = (JAXBElement) u.unmarshal(new FileInputStream("D:\\po.xml"));

    PurchaseOrderType po = (PurchaseOrderType) poe.getValue();

    // change the billto address
    USAddress address = po.getBillTo();
    address.setName("John Bob");
    address.setStreet("242 Main Street");
    address.setCity("Beverly Hills");
    address.setState("CA");
    address.setZip(new BigDecimal("90210"));

    // create a Marshaller and marshal to a file
    Marshaller m = jc.createMarshaller();
    m.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT, Boolean.TRUE);
    m.marshal(poe, new FileOutputStream("D:\\po.xml"));
} catch (JAXBException je) {
    je.printStackTrace();
} catch (IOException ioe) {
    ioe.printStackTrace();
}
```

Exercise 3: Given an XML document and its XSD file, read and add/remove some entries in the XML file using JAXB

- Input: an XML file and a XSD file
- Output: the modified content of the XML file
- Time: 30’
- Send your result to tnguyen256@csc.com or save it to \\qc-training\Freshers\XML\Exercise3
- Your name should be in the email subject or the folder name
- Deadline: Jul 12 2013

Lesson 3 Agenda

- XML Schema Definition (XSD)
- Java Architecture for XML Binding (JAXB)
- Introduction XStream

Introduction XStream

- XStream is a simple library to serialize objects to XML and back again
- Features:
  - Very ease of use
  - No mappings required
  - High performance
  - Clear error messages

  For more details: http://xstream.codehaus.org/

- Aug 11, 2011: XStream 1.4.1 was released

Two minutes tutorial

- Create Java object that will be serialized

```
public class PhoneNumber {
    private int code;
    private String number;

    // ... constructors and methods
```

```
public class Person {
    private String firstname;
    private String lastname;
    private PhoneNumber phone;
    private PhoneNumber fax;

    // ... constructors and methods
```

Two minutes tutorial (cont.)

- Step to Serialize java object to XML

```
XStream xstream = new XStream();

Person joe = new Person("Joe", "Walnes");
joe.setPhone(new PhoneNumber(123, "1234-456"));
joe.setFax(new PhoneNumber(123, "9999-999"));

// serialize the object to xml
String xml = xstream.toXML(joe);
System.out.println(xml);
```

```
<com.csc.tly6.xml.Person>
  <firstname>Joe</firstname>
  <lastname>Walnes</lastname>
  <phone>
    <code>123</code>
    <number>1234-456</number>
  </phone>
  <fax>
    <code>123</code>
    <number>9999-999</number>
  </fax>
</com.csc.tly6.xml.Person>
```

- Step to De-serialize java object to XML

```
//de-serialize the xml back to java object
Person newJoe = (Person)xstream.fromXML(xml);
```

Points to Remember

- XML Schema XSD describes the structure of an XML document
- JAXP
- XStream is a simple java library to serialize objects to XML and back again

Q&A

Contact: TuyenNguyen

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