Course: Web Application Development
Lab Instructor: Assoc.Prof.Nguyen Van Sinh
Email: nvsinh@hcmiu.edu.vn

Lab 7 - XML

Content:

- XML
  - Defining XML with DTD
  - Processing XML with DOM and SAX
  - Transforming XML with XSLT
- Practices Exercises

Refer from Session 8: XML and Chapter 23 of the Textbook: Core Web Programming, Second Edition.

Duration: 3 hours

Part 1: Defining XML with DTD

- **What is XML?**
  - *XML* (eXtensible Markup Language) is a meta-language that ***describes*** the content of the document (self-describing meta data). The structure of XML document is defined by DTD or XML Schema.
  - An XML document can be *well-formed* if it follows basic syntax rules.
  - An XML document is *valid* if its structure matches its Document Type Definition (DTD) or its XML Schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<authors>
  <name>
    <firstname>Larry</firstname>
    <lastname>Brown</lastname>
  </name>
  <name>
    <firstname>Marty</firstname>
    <lastname>Hall</lastname>
  </name>
</authors>
```

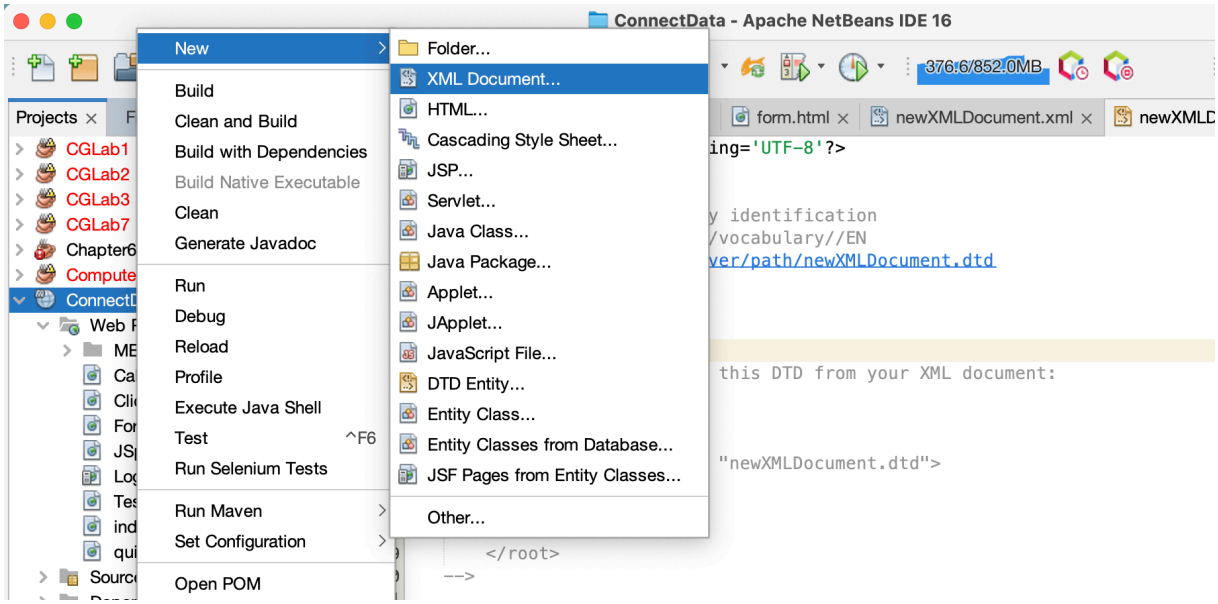
Example: authors.xml

- **What is DTD?**
  - *DTD* (Document Type Definition) defines the grammar/structure of document.
  - DTD is not in XML format. It is defined in a DTD file.
  - Learn more about DTD in ***Chapter 6, XML How to Program***.

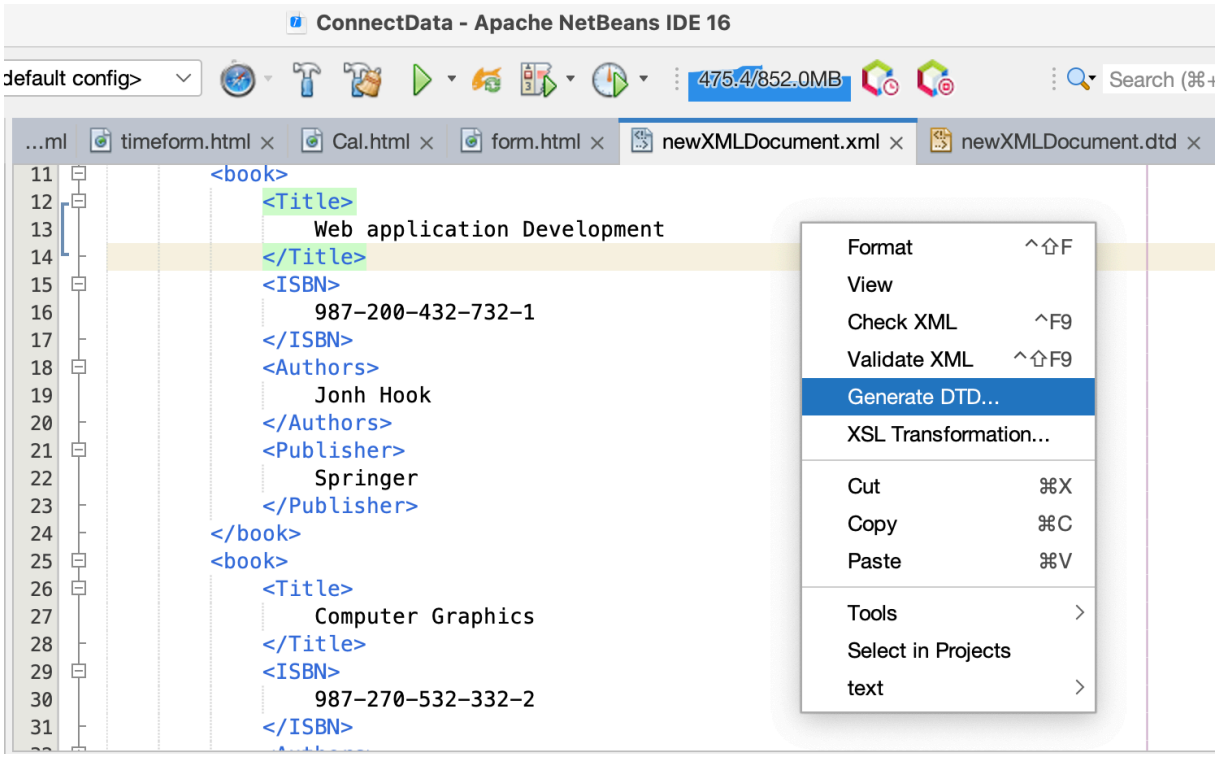
```
<?xml version='1.0' encoding='UTF-8'?>

<!--- Put your DTDDoc comment here. -->
<!ELEMENT authors (name)*>

<!--- Put your DTDDoc comment here. -->
<!ELEMENT name (lastname|firstname)*>

<!--- Put your DTDDoc comment here. -->
<!ELEMENT firstname (#PCDATA)>

<!--- Put your DTDDoc comment here. -->
<!ELEMENT lastname (#PCDATA)>
```

Example: authors.dtd

Implement XML and generate DTD on NetBeans Environment: right click on your project and create new XML file:



After you have XML file, right click on the XML editor -> Generate DTD:



Exercise 1: Create XML and DTD to store the following books data:

ISBN-10	Title	Author	Publisher	Publication date	Price ($)
0470114878	Beginning XML, 4th Edition (Programmer to Programmer)	David Hunter, Jeff Rafter, Joe Fawcett, and Eric van Dist	Wrox	May 21, 2007	**26.39**
0596007647	XML in a Nutshell, Third Edition	Elliotte Rusty Harold and W. Scott Means	O'Reilly Media, Inc.	September 2004	**26.37**
0596004206	Learning XML, Second Edition	Erik Ray	O'Reilly Media, Inc.	September 22, 2003	**26.37**
0130655678	Definitive XML Schema (The Charles F. Goldfarb Definitive XML Series)	Priscilla Walmsley	Prentice Hall PTR	December 17, 2001	**33.38**

Hint: there are two files (book.xml and book.dtd)

Part 2: Processing XML with DOM and SAX (refer chapter 23: core Web Programming)

- **Document Object Model (DOM)**
  - The Document Object Model, DOM for short, is an abstract data structure that represents XML documents as trees made up of nodes stored in memory.
  - The DOM is a standard issued by the W3C. This specification is unrelated to any one programming languages, therefore it is not designed for use with Java

specifically. Instead, various bindings of the DOM specification exist for the various programming languages including Java, JavaScript, C++, Python, and Perl.

- The **Xerces Java Parser 1.4.4** supports the XML 1.0 recommendation and contains advanced parser functionality, such as support for the W3C's XML Schema recommendation version 1.0, DOM Level 2 version 1.0, and SAX Version 2, in addition to supporting the industry-standard DOM Level 1 and SAX version 1 APIs.
- Steps for DOM Parsing:
  - Create a JAXP document builder.
  - Invoke the parser to create a Document representing an XML document.
  - Obtain the root node of the tree
  - Examine and/or modify properties of the node and its children

The following figure is the DOM Model representation of the previous XML file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE HOUSE [
<!ENTITY newroom "project room">
]>
<HOUSE>
    <UPSTAIRS sqfeet="200">
        <ROOM>bedroom</ROOM>
    </UPSTAIRS>
    <?color blue?>
    <DOWNSTAIRS sqfeet="900">
        <ROOM>bathroom</ROOM>
        <ROOM>kitchen</ROOM>
        <ROOM>living <![CDATA[ room ]]></ROOM>
        <ROOM>&newroom;</ROOM>
    </DOWNSTAIRS>
</HOUSE>
```

Mozilla Firefox

File Edit View Go Bookmarks Tools Help

```
-->
- <WebClass>
  - <student>
      <name>Dang Kha Doanh</name>
      <idNum>IT060111</idNum>
      <date-of-birth>15-02-84</date-of-birth>
      <city>Can Tho</city>
    </student>
  - <student>
      <name>Chau Nhuan Phat</name>
      <idNum>IT060112</idNum>
      <date-of-birth>19-12-86</date-of-birth>
      <city>Ha Long</city>
    </student>
  - <student>
      <name>Nguyen Phi Hung</name>
      <idNum>IT060113</idNum>
      <date-of-birth>05-02-83</date-of-birth>
      <city>HCM</city>
    </student>
  - <student>
      <name>Duong Trieu Vi</name>
      <idNum>IT060114</idNum>
      <date-of-birth>05-12-88</date-of-birth>
      <city>Ha Noi</city>
    </student>
  - <student>
      <name>Ngo Minh Phuong</name>
      <idNum>IT060115</idNum>
      <date-of-birth>28-12-84</date-of-birth>
      <city>Da Nang</city>
    </student>
</WebClass>
```

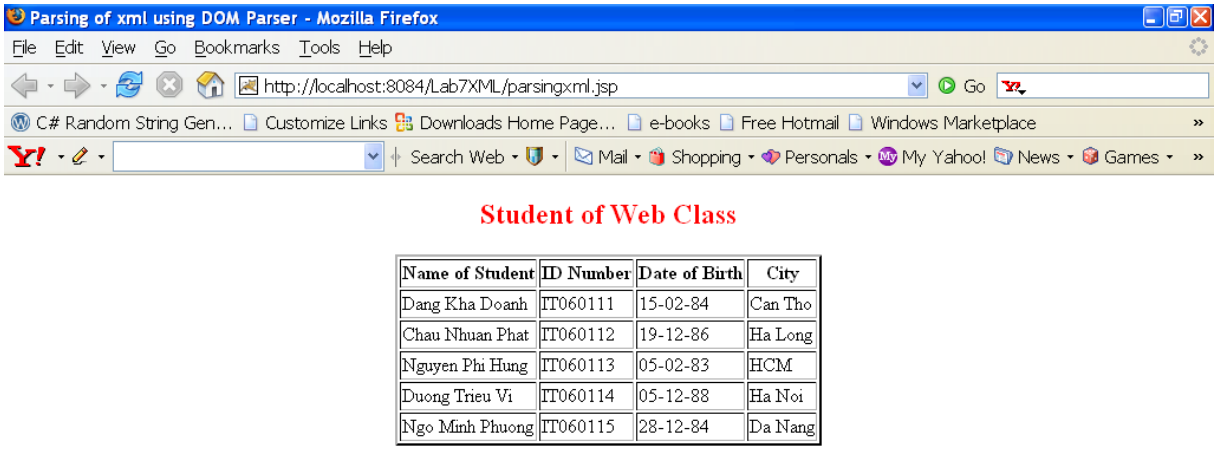
Exercise 2: Create an XML file "WebClass.xml" as below, then create a JSP file "parsingxml.jsp" to get the content of the WebClass.xml.

Copy the following source:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
```

```
<%@page import="org.w3c.dom.*, javax.xml.parsers.*" %>
<%
  DocumentBuilderFactory docFactory = DocumentBuilderFactory.newInstance();
  DocumentBuilder docBuilder = docFactory.newDocumentBuilder();
  Document doc = docBuilder.parse("src//WebClass.xml");
%>
<%!
  public boolean isTextNode(Node n){
    return n.getNodeName().equals("#text");
  }
%>
<html>
  <head><title>Parsing of xml using DOM Parser</title></head>
  <body>
    <h2><font color='red'>Student of Web Class</font></h2>
    <table border="2">
      <tr>
        <th>Name of Student</th>
        <th>ID Number</th>
        <th>Date of Birth</th>
        <th>City</th>
      </tr>
        <%
          Element  element = doc.getDocumentElement();
          NodeList personNodes = element.getChildNodes();
          for (int i=0; i<personNodes.getLength(); i++){
            Node stu = personNodes.item(i);
            if (isTextNode(stu))
              continue;
            NodeList NameDOBCity = stu.getChildNodes();
        %>
      <tr>
        <%
          for (int j=0; j<NameDOBCity.getLength(); j++ ){
            Node node = NameDOBCity.item(j);
            if ( isTextNode(node))
              continue;
        %>
        <td><%= node.getFirstChild().getNodeValue() %></td>
        <%}%>
      </tr>
        <%}%>
    </table>
  </body>
</html>
```

This is the result on browser:



Exercise 3: The same result as Exercise 2, but we use Servlet to parse WebClass.xml

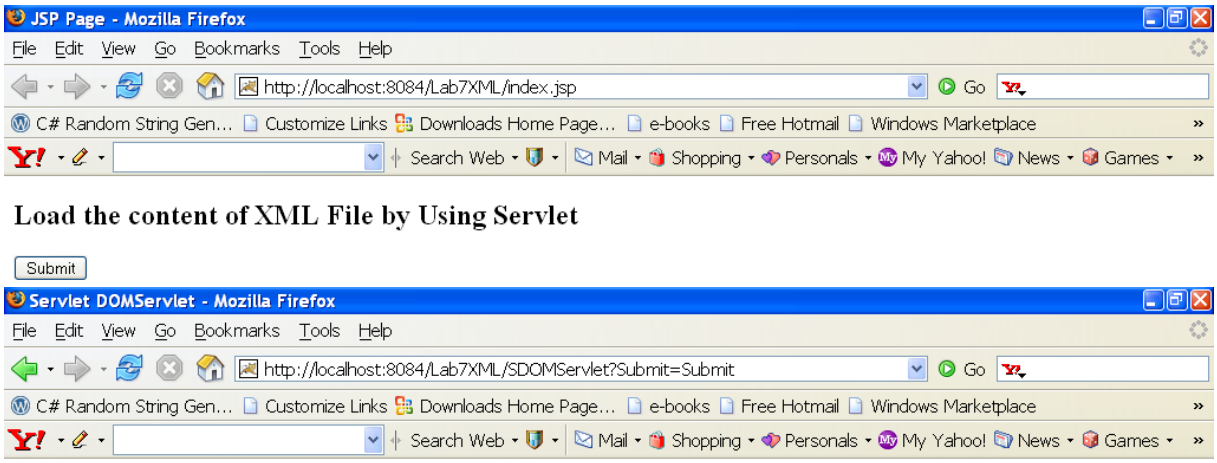
Hint: Create a Servlet "DOMServlet.java", copy the source code below to test:

```
protected void processRequest(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            // TODO output your page here
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet DOMServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1><center>List of Students in Web Class
</center></h1>");

            out.println("<center><table border=1 cellpadding=0
bgcolor=#FFFFFF></center>");
            out.println("<tr><td><b>Name</b></td> <td><b>ID</b></td>
<td><b>DATE</b></td> <td><b>CITY</b></td> </tr>");

            DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();
            // Turn on namespace support
            factory.setNamespaceAware(true);

            // Create a JAXP document builder
            DocumentBuilder parser = factory.newDocumentBuilder();

            // Read the entire document into memory
            Document document = parser.parse("src//WebClass.xml");

            // Obtain the root node of the tree
            Node booklist = document.getDocumentElement();
```

```
            NodeList books = booklist.getChildNodes();

            int nBooks = books.getLength();
            for (int i = 0; i < nBooks; i++) {

                Node book = books.item(i);
                if (book.getNodeType() != Node.TEXT_NODE) {
                    out.println("<tr>");
                    printBook(book, out);
                    out.println("</tr>");
                }

            }

            out.println("</body>");
            out.println("</html>");

        } catch (SAXException ex) {
            Logger.getLogger(DOMServlet.class.getName()).log(Level.SEVERE,
null, ex);
        } catch (ParserConfigurationException ex) {
            Logger.getLogger(DOMServlet.class.getName()).log(Level.SEVERE,
null, ex);
        } finally {
            out.close();
        }
    }

    private void printBook(Node book, PrintWriter out) {
        NamedNodeMap attributes = book.getAttributes();

        if (attributes != null) {
            NodeList childNodes = book.getChildNodes();
            String name = "";
            String id = "";
            String date = "";
            String city = "";
            for (int i = 0; i < childNodes.getLength(); i++) {
                Node child = childNodes.item(i);
                String nodeName = child.getLocalName();
                if (nodeName != null) {
                    if (nodeName.equals("name")) {
                        NodeList children = child.getChildNodes();
                        Node dateNode = children.item(0);
                        if (dateNode.getNodeType() == Node.TEXT_NODE) {
                            name = dateNode.getNodeValue();
                        }
                    } else if (nodeName.equals("idNum")) {
                        NodeList children = child.getChildNodes();
                        Node dateNode = children.item(0);
                        if (dateNode.getNodeType() == Node.TEXT_NODE) {
                            id = dateNode.getNodeValue();
```

```
                    }
                } else if (nodeName.equals("date-of-birth")) {
                    NodeList children = child.getChildNodes();
                    Node priceNode = children.item(0);
                    if (priceNode.getNodeType() == Node.TEXT_NODE) {
                        date = priceNode.getNodeValue();
                    }
                } else if (nodeName.equals("city")) {
                    NodeList children = child.getChildNodes();
                    Node priceNode = children.item(0);
                    if (priceNode.getNodeType() == Node.TEXT_NODE) {
                        city = priceNode.getNodeValue();
                    }
                }
            }
        }
        out.print("<td>" + name + "</td>" + "<td>" + id + "</td>" +
"<td>" + date + "</td>" + "<td>" + city + "</td>");
    }
}
```

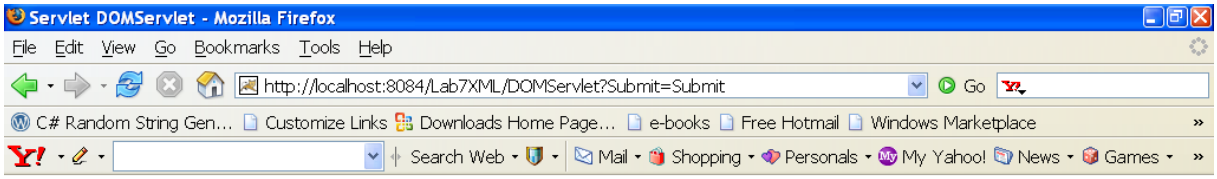
This is the result:



Name	ID	DATE	CITY
Dang Kha Doanh	IT060111	15-02-84	Can Tho
Chau Nhuan Phat	IT060112	19-12-86	Ha Long
Nguyen Phi Hung	IT060113	05-02-83	HCM
Duong Trieu Vi	IT060114	05-12-88	Ha Noi
Ngo Minh Phuong	IT060115	28-12-84	Da Nang

Exercise 4: Base on the source code of **Exercise 3**, you create a Servlet to get the content of book.xml (from **Exercise 1**)

This is the result:



List of Books

ISBN-10	TITLE	AUTHOR	PUBLISHER	DATE	PRICE
0470114878	Beginning XML, 4th Edition (Programmer to Programmer)	David Hunter;Jeff Rafter;Joe Fawcett;Eric van der Vlist;	Wrox	May 21, 2007	26.39
0596007647	XML in a Nutshell, Third Edition	Elliotte Rusty Harold;W. Scott Means;	O'Reilly Media, Inc.	September 2004	26.37
0596004206	Learning XML, Second Edition	Erik Ray;	O'Reilly Media, Inc.	September 22, 2003	26.37
0130655678	Definitive XML Schema (The Charles F. Goldfarb Definitive XML Series)	Priscilla Walmsley;	Prentice Hall PTR	December 17, 2001	33.38

- **Simple API for XML (SAX)**
  - SAX is an alternate method for parsing XML documents that uses an ***event-based model*** - notifications called events are raised as the document is parsed.
  - With this event-based model, no tree structure is created by the SAX-based parser to store the XML document's data - data is passed to the application from the XML document as it is found.
  - The SAX parser invokes certain methods when events occur. The following specifies some methods called upon events occurs (see API for more details)

Method Name	Description
`setDocumentLocator`	Invoked at the beginning of parsing.
`startDocument`	Invoked when the parser encounters the start of an XML document.
`endDocument`	Invoked when the parser encounters the end of an XML document.
`startElement`	Invoked when the start tag of an element is encountered.
`endElement`	Invoked when the end tag of an element is encountered.
`characters`	Invoked when text characters are encountered.
`ignorableWhitespace`	Invoked when whitespace that can be safely ignored is encountered.
`processingInstruction`	Invoked when a processing instruction is encountered.

Given the following XML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<RootElement param="value">
    <FirstElement>
        Some Text
    </FirstElement>
    <SecondElement param2="something">
        Pre-Text <Inline>Inlined text</Inline> Post-text.
    </SecondElement>
</RootElement>
```

This XML document, when passed through a SAX parser, will generate a sequence of **events** like the following:

- XML Processing Instruction, named *xml*, with attributes *version* equal to "1.0" and *encoding* equal to "UTF-8"
- XML Element start, named *RootElement*, with an attribute *param* equal to "value"
- XML Element start, named *FirstElement*
- XML Text node, with data equal to "Some Text" (note: text processing, with regard to spaces, can be changed)
- XML Element end, named *FirstElement*
- XML Element start, named *SecondElement*, with an attribute *param2* equal to "something"
- XML Text node, with data equal to "Pre-Text"
- XML Element start, named *Inline*
- XML Text node, with data equal to "Inlined text"
- XML Element end, named *Inline*
- XML Text node, with data equal to "Post-text."
- XML Element end, named *SecondElement*
- XML Element end, named *RootElement*

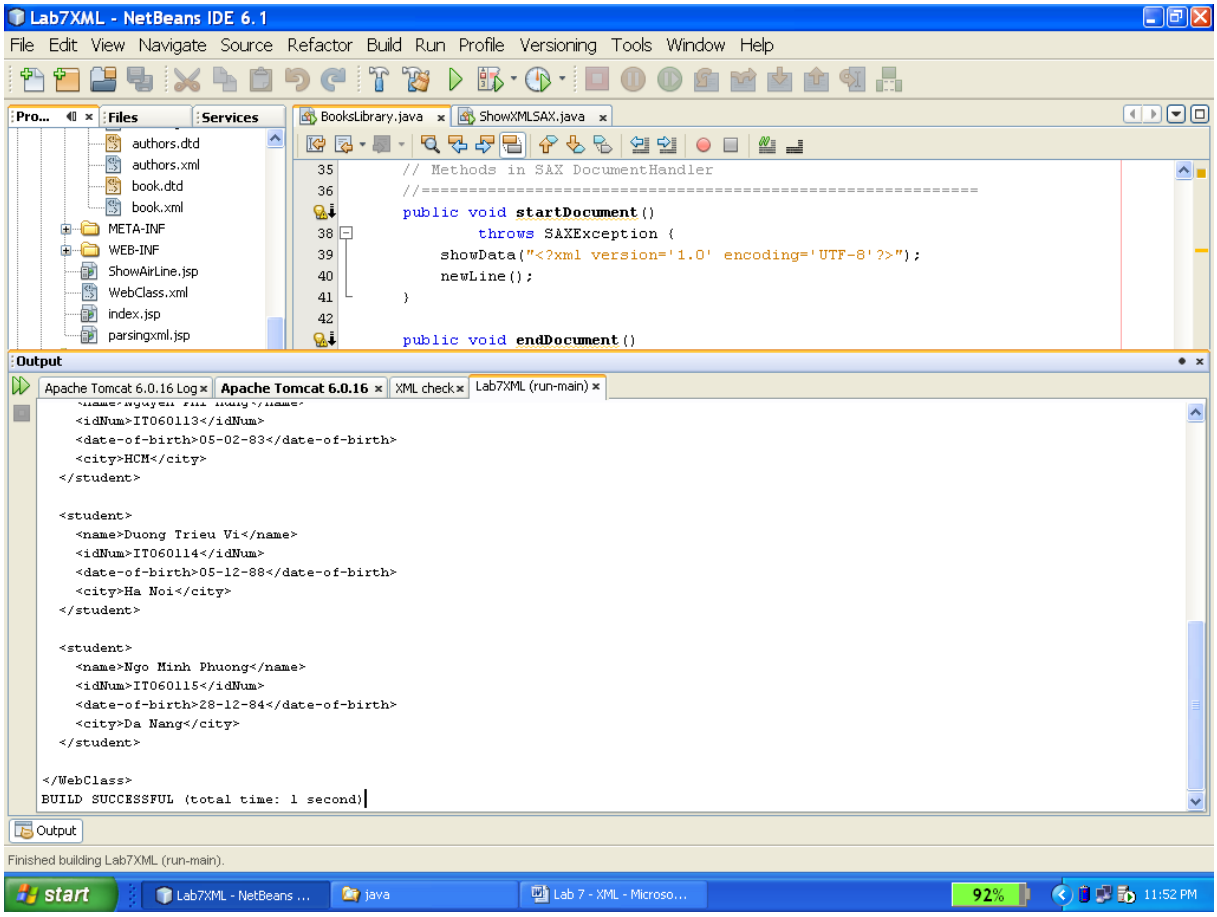
- Steps for SAX Parsing:
  - Tell the system which parser you want to use.
  - Create a parser instance.
  - Create a content handler to respond to ***parsing events***.
  - Invoke the parser with the designated content handler and document.
- **Exercise 5:** copy source below to test how to get XML file using SAX

```
import java.io.*;
import org.xml.sax.*;
import javax.xml.parsers.SAXParserFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;

public class BooksLibrary extends HandlerBase {
    protected static final String XML_FILE_NAME = "src//WebClass.xml";
    public static void main(String argv[]) {
        // Use the default (non-validating) parser
        SAXParserFactory factory = SAXParserFactory.newInstance();
```

```
        try {
            // Set up output stream
            out = new OutputStreamWriter(System.out, "UTF8");
            // Parse the input
            SAXParser saxParser = factory.newSAXParser();
            saxParser.parse(new File(XML_FILE_NAME), new BooksLibrary());
        } catch (Throwable t) {
            t.printStackTrace();
        }
        System.exit(0);
    }
    static private Writer out;

    //============================================================
    // Methods in SAX DocumentHandler
    //============================================================
    public void startDocument()
            throws SAXException {
        showData("<?xml version='1.0' encoding='UTF-8'?>");
        newLine();
    }

    public void endDocument()
            throws SAXException {
        try {
            newLine();
            out.flush();
        } catch (IOException e) {
            throw new SAXException("I/O error", e);
        }
    }

    public void startElement(String name, AttributeList attrs)
            throws SAXException {
        showData("<" + name);
        if (attrs != null) {
            for (int i = 0; i < attrs.getLength(); i++) {
                showData(" ");
                showData(attrs.getName(i) + "=\"" + attrs.getValue(i) +
"\"");
            }
        }
        showData(">");
    }

    public void endElement(String name)
            throws SAXException {
        showData("</" + name + ">");
    }

    public void characters(char buf[], int offset, int len)
            throws SAXException {
```

```
            String s = new String(buf, offset, len);
            showData(s);
        }

        private void showData(String s)
                throws SAXException {
            try {
                out.write(s);
                out.flush();
            } catch (IOException e) {
                throw new SAXException("I/O error", e);
            }
        }

        // Start a new line
        private void newLine()
                throws SAXException {
            String lineEnd = System.getProperty("line.separator");
            try {
                out.write(lineEnd);
            } catch (IOException e) {
                throw new SAXException("I/O error", e);
            }
        }
}
```

This is the result:



Part 3: Transforming XML

- **What is XSLT?**
  - **XSLT** (Extensible Stylesheet Language Transformations) is a language for transforming XML documents into HTML, XML, or other document formats.

  - **XSL** (Extensible Stylesheet Language) is a language for expressing stylesheets, specifies how to transform XML into HTML, XML, or other document formats. Use:
    - XPath to identify parts of an XML document
    - XSLT templates to apply transformations

  - **XPath** is an expression language used by XSLT to:
    - Locate elements and/or attributes within an XML document.
    - Test Boolean conditions
    - Manipulate strings
    - Perform numerical calculations
    - Ex:

    <xsl:template match="/name/first" >
      ...
    </xsl:template>

  - **XSLT Stylesheet Elements**
    - Matching and selection templates:
      - `xsl:template match="xpath"`
        - Defines a template rule for producing output
        - Applied only to nodes which match the pattern
        - Invoked by using <xsl:apply-templates>
      - `xsl:apply-templates`
        - Applies matching templates to the children of the context node
      - `xsl:value-of`
        - Evaluates the expression as a string and sends the result to the output
        - Applied only to the first match
        - ". " selects the text value of the current node

    Ex:

```
<xsl:template match="/">
<html>
<head><title>Ktee Siamese</title></head>
<body>
     <xsl:apply-templates/>
</body>
</html>
</xsl:template>

<xsl:template match="name">
     <h2><xsl:value-of select="."/></h2>
</xsl:template>
```

    - Branching elements
      - `xsl:for-each select="expression"`

- Processes each node selected by the XPath expression
- `xsl:if`
  - Evaluates Select any number of alternatives the expression to a boolean and if true, applies the template body
- `xsl:choose`
  - Select any number of alternatives
- `xsl:output`
  - Controls the format of the stylesheet output
  - Useful attributes:

```
method= "[html|xml|text]"
indent="[yes|no]"
version="version"
doctype-public="specification"
encoding="encoding"
standalone="[yes|no]"
```

- *Steps for Translating a Document*
  - 1. Tell the system which parser to use
  - 2. Establish a factory in which to create transformations
  - 3. Create a transformer for a particular style sheet
  - 4. Invoke the transformer to process the document
- **Exercise 6: (**refer 23.6**)**
  - Write XSL document to transform the **book.xml** XML document to HTML.
  - Use XSLT Example under the Code folder to check above documents.
  - Run XsltExample.java file, the results will be shown like below: (***refer from chapter 23.7 XSLT Example 1: XSLT Document Editor – Textbook Core Web Programming, Second Edition***)

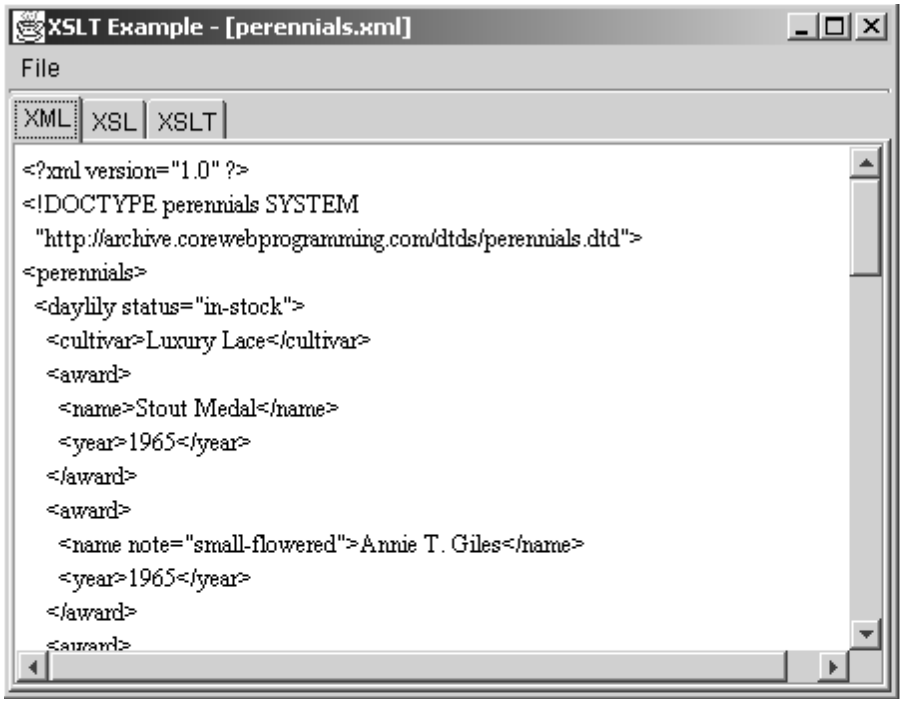


Figure 1: Presentation of XML tabbed pane in XsltExample with perennials.xml loaded

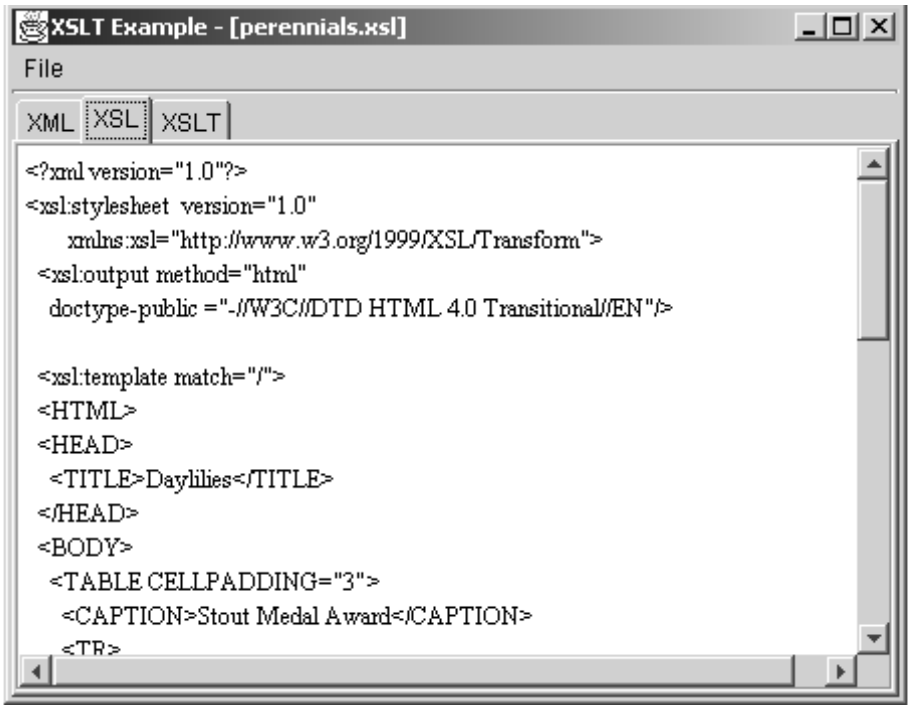


Figure 2: Presentation of XSL tabbed pane in XsltExample with perennials.xsl loaded

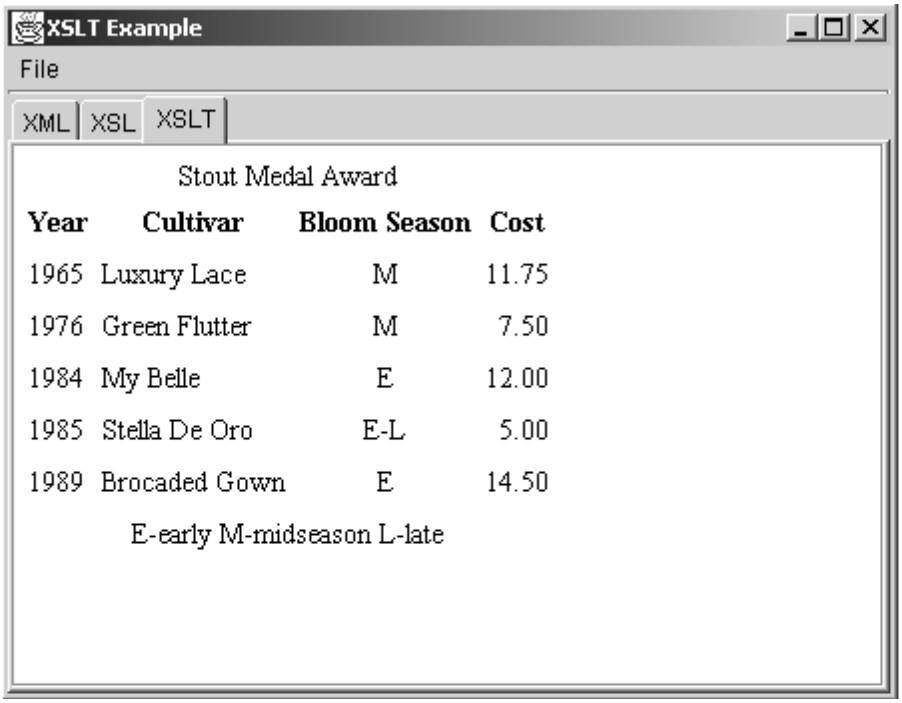


Figure 3: Result of XSLT transformation of perennials.xml and perennials.xsl

Additional exercise (bonus):

Write a JSP file to get information in XML file as Exercise 1, and export the output is as an excel file.