eXtensible Markup Language (XML)

Sara Sprenkle

August 1, 2006

Announcements

- Assignment 6 due Thursday
- Project 2 due next Wednesday

- Quiz
- TA Evaluation

Using the Synchronized Keyword

- Can use synchronized keyword inside of methods

```
synchronized {
// atomic operation
}
```

  - Implies synchronization on `this` object
  - Same rules as synchronized method access
    - Only one thread in synchronized code at a time

Using the Synchronized Keyword

- Alternatively, can use to synchronize access to an object

```
synchronized (sharedObject) {
   // atomic operation
}
```

  - Same rules as synchronized method access
- Equivalent code blocks:

```
synchronized (this) {
   // atomic operation
}
```

```
synchronized {
   // atomic operation
}
```

What is XML?

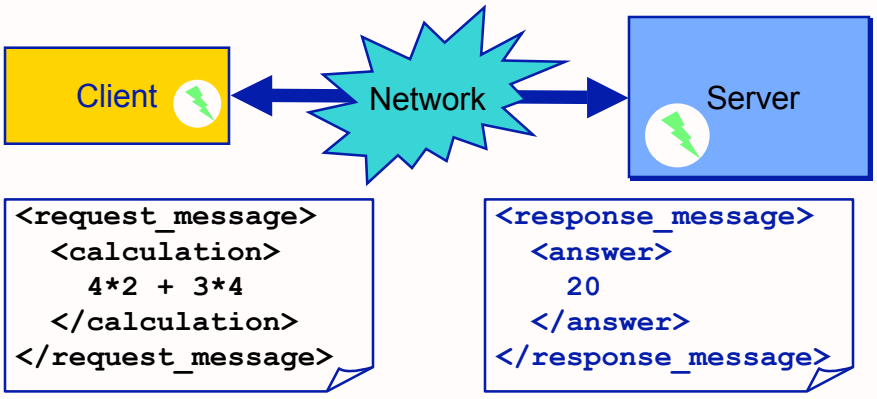
- eXtensible Markup Language
- Describes data in a textual, hierarchical (tree-based) structure
  - Structure, store, and transmit information
  - Portable: preserves meaning and structure across machines/platforms
- Doesn't *do* anything itself
  - Other applications use XML to share information

What is XML?

- Simplified subset of Standard Generalized Markup Language (SGML)
- Defines other languages, such as RSS
  - Get updates on your favorite web sites
- Like well-structured HTML
- Different focus from HTML
  - XML: focuses on describing information
  - HTML: focuses on displaying information, how data looks

One Use of XML

- Transmit messages across network



Java and XML

- Java class libraries to read, parse, validate, generate, and transform XML data

XML Documents

- All text
  - makes it relatively easy for humans (and programs) to create, examine, and debug them
- Syntax rules
  - Very strict
  - Very simple
  - Make document easy to parse
- Case-sensitive
- The ML of “Markup Language”
  - Tags, enclosed in angle brackets, < >

XML Tags

- Identify data using tags
  - called elements

```
<name>
```

- Place content between tags:

```
<name>Jack Sparrow</name>
```

- Tags can have attributes, like this:

```
<name language="English">Jack Sparrow</name>
```

- Tags can contain child elements
  -

```
<name><first>Jack</first><last>Sparrow</last></name>
```

Examples of XML Syntax

- Elements/Tags:

```
<date>22 May 2006</date>
<yesterday/>
<date><month>10</month><day>22</day></date>
```

"Empty" tag

- Attributes:

```
<time zone="-2">5:57:39</time>
<time gmt='yes'>7:57:39</time>
```

Either "" or ''

- Comments:

```
<!-- XML data comment -->
```

- Entity References — Need to be declared previously

```
&otherLoc;  &copyr;  &LegalDisclaimer;
```

- Document type reference — Describes doc's structure

```
<!DOCTYPE MAIL SYSTEM "email.dtd">
```


Well-Formedness

- All documents must be **well-formed**
  - Each tag must have a corresponding end tag

```
<name>Jack Sparrow</name>
```

Start Tag for a **Name** piece of data

Corresponding End tag

  - Or, the tag must be self-contained

```
<name value="Jack Sparrow"/>
```

XML Syntax Example

```
<?xml version="1.0"?>
<!DOCTYPE mail SYSTEM "email.dtd">
<mail>
   <message>
        <to>cisc370@udel.edu</to>
        <from>sprenks@udel.edu</from>
        <body type="text/plain">Assignment due</body>
        <unread/>
   </message>
   <message importance="high">
        <to>badStudent@udel.edu</to>
        <from>sprenks@udel.edu</from>
        <subject>CISC370</subject>
        <body>Please see me for extra help</body>
   </message>
</mail>
```

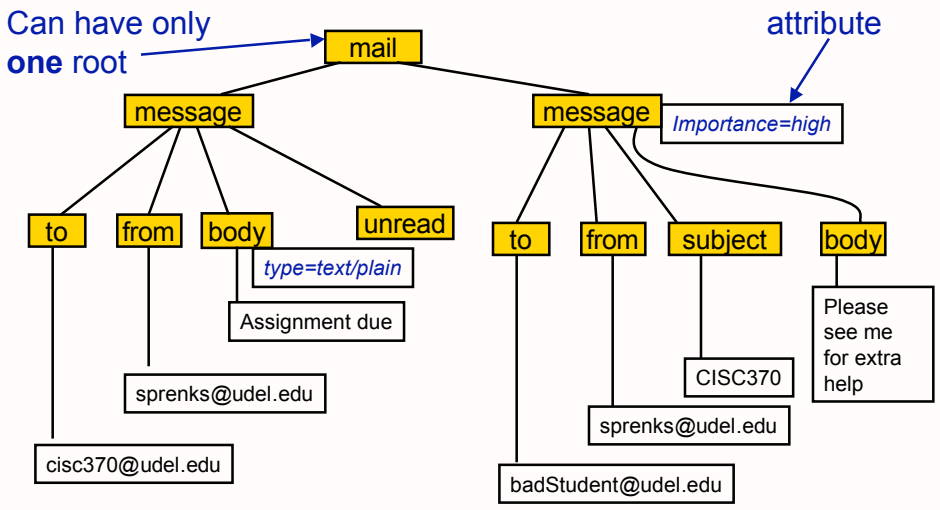
Prolog/Declaration

DOCTYPE reference

root

Viewing in Eclipse

XML Syntax Example



XML Syntax: Escape Codes

- < less than sign <
- > greater than sign >
- & ampersand &
- " quote "
- ' apostrophe ’

XML Design Considerations

- When to use attributes and when to use child elements?
- Elements
  - Best for “things”, particularly those that have properties
  - Easier to extend later
- Attributes
  - Best for properties, like modifiers or units

XML Structure Specifications

- XML documents that do not have some kind of specification can contain **any** tags
  - makes standardized communication very hard!
- Most XML documents obey a particular set of rules as to their structure
  - Rules define what tags, attributes, and entities may appear in the document
- XML documents can use namespaces, which are separate scopes within which tags are defined
  - similar to namespaces in C++

Specifying Valid XML Tags

- Document Type Definition (DTD)
  - Written in SGML (which has a very confusing syntax!)
  - DTD approach is not used very much anymore
- Schema
  - Current approach
  - Written in XML
    - Makes it much easier to define the structure of an XML document

XML Programming

- Generally, programs (and programmers) deal with XML in one of three ways
  - **Parsing:** XML document as program input
    - Given an XML document, extract the stored data and process it programmatically
  - **Generation:** XML doc as program output
    - Given some data, generate an XML representation of it
  - **Transformation:** XML doc as both program input and output
    - Given an XML document, transform it into some other kind of document

Java XML Programming

- Java APIs for XML Processing (JAXP)
  - included with Java 1.4 and later
  - APIs make programmatically working with XML under Java very easy and straightforward
  - Sun's collection of XML APIs
- Other packages available, which all work a little differently
  - Just a library of classes and methods
  - www.jdom.org

Fundamental Models for XML Parsing

- **SAX - Simple API for XML**
  - Serial, sequential access to XML tags and content
  - event-oriented callback model
  - fast and low overhead
  - difficult to use for transforms
- **DOM - Document Object Model for XML**
  - Tree-based access to entire XML document
  - data traversal model
  - keeps entire document in memory
  - easy to use for transforms

The SAX Programming Model

- Serial Access with the Simple API for XML
  - Requires more programming
  - Event-driven
  - Harder to visualize
- A SAX Parser chops an XML document into a sequence of events
  - Event: encountering an element of the XML document
  - Parser delivers XML events and errors, if any occur, to application by calling methods on handler objects (provided by program)

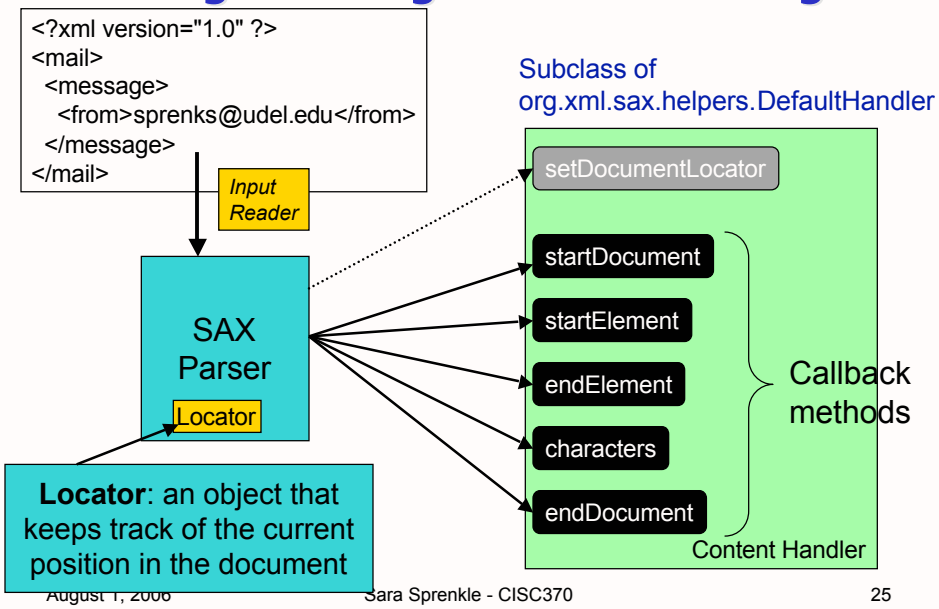
The SAX Programming Model

- Handler objects must implement particular interfaces:
  - `org.xml.sax.ContentHandler`
    for receiving events about document contents and tags
  - `org.xml.sax.ErrorHandler`
    for receiving errors and warnings that occur during parsing
  - `org.xml.sax.DTDHandler`
  - `org.xml.sax.EntityResolver`

SAX Parsers

- Most SAX parsers can read XML data from any valid java.io.Reader
  - Why a Reader?
- or directly from a URL (for directly parsing XML that exists on the Internet)
- SAX parsers can be **validating** or non-validating
  - Validation: checking a document for conformance to its specified DTD or schema
  - Most SAX parsers support both operational modes

SAX Programming – Event Passing



SAX Programming – Basics

- To use SAX, you need to import the relevant packages, e.g.,
  - `import org.xml.sax.*;`
  - `import org.xml.sax.helpers.*;`
  - `import javax.xml.parsers.*;`
- To do parsing, you need a SAX XMLReader object

```
SAXParserFactory sp_factory =
  SAXParserFactory.newInstance();
SAXParser sp = sp_factory.newSAXParser();
XMLReader theParser = sp.getXMLReader();
```

SAX Programming – Errors

- SAX Parser can encounter three kinds of errors during parsing:
  - **warning** a non-serious problem occurred
  - **error** a serious but recoverable problem occurred
  - **fatalError** a problem occurred that is so grave that parsing cannot continue.

  Parsing can continue
- Warning and error conditions result from violation of document validation (DTD or schema constraints)
- FatalError conditions are from I/O problems or non-well-formed XML
- Not all SAX parsers treat each condition in the same category
  - Check the docs!
- All SAX parsers can throw SAXExceptions
  - Any SAX handler method (the methods the parser calls on your handler object) can also throw a SAXException

SAX Programming – Example

- Example program consists of two classes:
  - **`SAXExample`**
    - supplies a main() method that creates a SAX parser and invokes it to parse a file specified on the command line
  - **`SAXHandler`**
    - acts as the SAX ContentHandler and ErrorHandler

SAX Programming – Example

```
1     import java.io.*;
2     import org.xml.sax.*;
3     import org.xml.sax.helpers.*;
4     import javax.xml.parsers.*;
5
6     public class SAXExample {
7       public static void main(String [] args) {
8         try {
9           System.out.println("Creating and setting up the SAX parser.");
10          SAXParserFactory sp_factory = SAXParserFactory.newInstance();
11          XMLReader theReader = sp_factory.newSAXParser().getXMLReader();
12          SAXHandler theHandler = new SAXHandler();
13          theReader.setContentHandler(theHandler);
14          theReader.setErrorHandler(theHandler);
15          theReader.setFeature("http://xml.org/sax/features/validation", false);
16          System.out.println("Making InputSource for " + args[0]);
17          FileReader file_in = new FileReader(args[0]);
18          System.out.println("About to parse...");
19          theReader.parse(new InputSource(file_in));
20          System.out.println("...parsing done.");
21        } catch (Exception e) {
22          System.err.println("Error: " + e);  e.printStackTrace();
23        }
24      }
25    }
```

Can call `parse` with other args

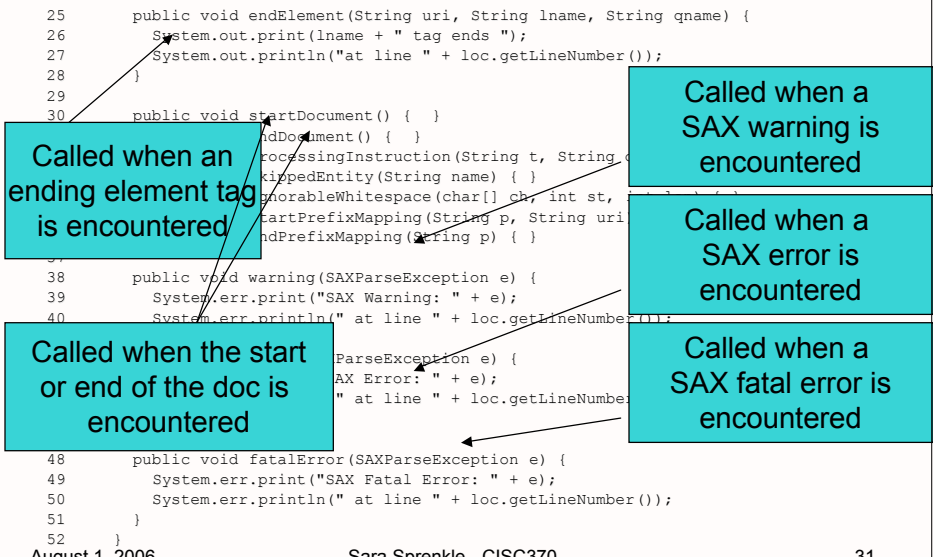
SAX Programming – Example

```
1     import org.xml.sax.*;
2
3     public class SAXHandler implements ContentHandler, ErrorHandler
5     {
6       private Locator loc = null;
7
8       public void setDocumentLocator(Locator l) {
9         loc = l;
10      }
11
        rs(char [] ch, int st, int len) {
        ing(ch, st, len);
        ("Got content string '" + s + "'");
16      }
17
18      public void startElement(String uri, String lname,
19                               String qname, Attributes attrs) {
20        System.out.print(lname + " tag with ");
21        System.out.print(attrs.getLength() + " attrs starts");
22        System.out.println(" at line " + loc.getLineNumber());
23      }
24
```

Called when a set of characters is encountered

Called when a starting element tag is encountered

SAX Programming – Example




```
25      public void endElement(String uri, String lname, String qname) {
26        System.out.print(lname + " tag ends ");
27        System.out.println("at line " + loc.getLineNumber());
28      }
29
30      public void startDocument() {  }
        ndDocument() {  }
        rocessingInstruction(String t, String
        kippedEntity(String name) { }
        gnorableWhitespace(char[] ch, int st,
        tartPrefixMapping(String p, String uri
        ndPrefixMapping(String p) { }
38      public void warning(SAXParseException e) {
39        System.err.print("SAX Warning: " + e);
40        System.err.println(" at line " + loc.getLineNumber());
          ParseException e) {
          AX Error: " + e);
          " at line " + loc.getLineNumber
48      public void fatalError(SAXParseException e) {
49        System.err.print("SAX Fatal Error: " + e);
50        System.err.println(" at line " + loc.getLineNumber());
51      }
52    }
```

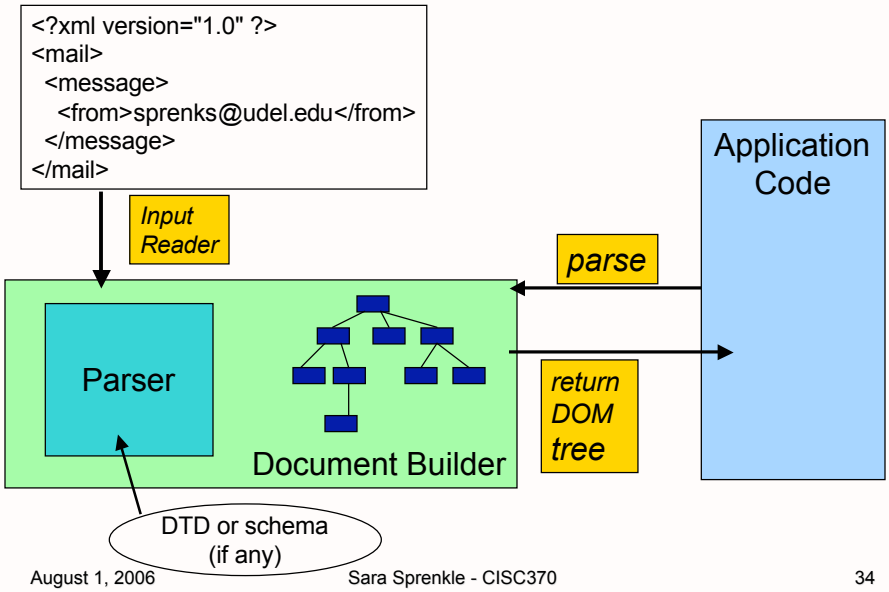

DOM Programming – Overview

- DOM parser builds an in-memory tree representation of the entire XML document
  - reference to tree is returned to your program
- DOM tree is composed of objects, most of which implement the following interfaces from the `org.w3c.dom` package:
  - **`Node`** parent interface of all DOM tree nodes
  - **`Element`** represents a tag in a XML document, may have children
  - **`Document`** represents an entire XML document, always has children
  - **`Text`** contents of an element or an attribute
  - **`Attr`** represents an attribute of an element

DOM Programming – Overview

- Your program can manipulate the DOM tree
  - Traverse, modify, print, apply XSL transforms (XML Stylesheet Language Transforms … more about that in a bit), and many more functions
- Most DOM parsers allow either validation or non-validation modes
  - Validation refers to checking for conformance against the document's specified DTD or schema

DOM Programming – Model



DOM Programming – Creating the DOM Tree

- How does the DOM API generate the tree representing the XML document?
  - It uses a SAX parser!
  - The DOM APIs run a SAX parser to parse the XML document in an event-driven manner, constructing the DOM tree in memory

DOM Programming – Basics

- To create a DOM DocumentBuilder with the JAXP packages

```
DocumentBuilderFactory dbuilder_factory =
  DocumentBuilderFactory.newInstance();
dbuilder_factory.setValidating(true);
DocumentBuilder dbuilder =
  dbuilder_factory.newDocumentBuilder();
```

- Build a document tree from XML data found in a particular file

```
FileReader file_in = new FileReader("thefile.xml");
Document doc = dbuilder.parse(new
  InputSource(file_in));
```

DOM Programming – Example

```
1     import org.w3c.dom.*;
2     import javax.xml.parsers.*;
3     import org.xml.sax.*;
4
5     public class SimpleXML2 {
6       public static void main(String [] args) {
8         try {
9           System.out.println("Creating Document builder.");
10          DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
11          dbf.setValidating(true);
12          DocumentBuilder db = dbf.newDocumentBuilder();
13          System.out.println("Ready to parse!");
14          FileReader file_in = new FileReader("theFile.xml");
15          Document doc = db.parse(new InputSource(file_in));
16          System.out.println("Parsed document, ready to process.");
17          myDOMTreeProcessor proc = new myDOMTreeProcessor();
18          proc.process(doc, System.out);
19        }
20        catch (Exception e) {
21          System.err.println("XML Exception thrown: " + e);
22          e.printStackTrace();
23        }
24      }
25    }
```

If want to validate

DOM Programming – Example

- MyDOMTreeProcessor

Changing the DOM Tree

- After you have the DOM tree, you can:
  - search for particular nodes
  - extract element and text values
  - alter the tree:
    - add elements to the tree or change them
    - add attributes to elements or change their values
    - re-arrange elements and subtrees
    - synthesize entirely new trees or subtrees
- If you create a new tree or subtree, you should apply the normalize() method before processing it further
  - optimizes layout

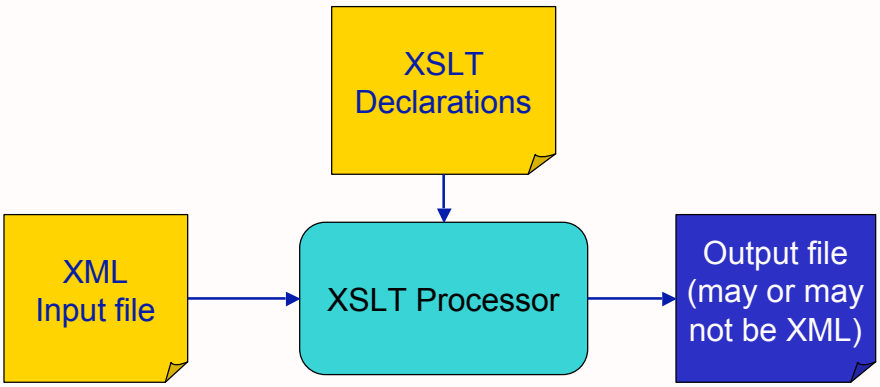
XML Stylesheet Language (XSL)

- Core XML technology for performing transformations to and on XML documents
- Offers extensive template matching and processing instructions for transforming XML data into other XML-like data
  - e.g., XML->XML, XML->HTML, XML->text
- A programming language onto itself
  - Geared towards pattern matching and formatting

XSL Transformations (XSLT)

- An XSL Processor is a system (usually invoked through an object interface) that can apply XSLT transforms to some XML input and generate another document (normally the data in another form)
- Transformation rules are expressed in XSL
- JAXP includes the Java implementation of the Apache Xalan XSLT processor

XSL Transformations (XSLT)



- A concrete example:
  - http://www.cis.udel.edu/~sprenkle/bibtex2html/

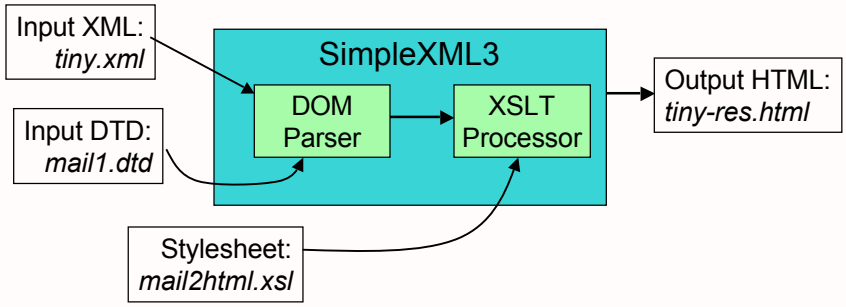
XML Programming - Transformations

- Advantage of the DOM programming model: can be used easily with XSL
- JAXP fully supports XSLT with the package javax.xml.transform and its sub-packages
- The sequence of steps for creating and applying transforms with XSLT and Xalan is
  - Obtain or create a DOM tree, create a Source from it
  - Create a TransformerFactory
  - Create a Templates object based on a particular XSLT document
  - Create a Transformer object from the Templates object
  - Create an output Results object
  - Apply the Transformer to the Source and the Result

XSLT Programming – Example

- This program applies a supplied XSL file to an XML file, and produces an output file.

```
% java SimpleXML3 tiny.xml mail2html.xsl tiny-res.html
```



XSLT Programming – Example

```
1     import org.w3c.dom.*;
2     import javax.xml.parsers.*;
3     import javax.xml.transform.*;
4     import org.xml.sax.*;
5
6     public class SimpleXML3 {
7       public static void main(String [] args) {
8         String src_file = args[0];
9         String template_file = args[1];
10        String res_file = args[2];
11        try {
12          System.out.println("Creating Document builder.");
13          DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
14          dbf.setValidating(true);
15          DocumentBuilder db = dbf.newDocumentBuilder();
16          InputSource file_in = new InputSource(src_file);
17          Document doc = db.parse(file_in);
18          System.out.println("Parsed document okay");
19          System.out.println("Using " + template_file + " to make " + res_file);
20          myXSLTProcessor proc = new myXSLTProcessor();
21          proc.process(doc, template_file, res_file);
22        }
23        catch (Exception e) {
24          System.err.println("XML Exception thrown: " + e);
25        }
26      }
27    }
```


XSLT Programming – Example

```
1     import org.w3c.dom.*;
2     import javax.xml.parsers.*;
3     import javax.xml.transform.*;
4     import javax.xml.transform.dom.*;
5     import javax.xml.transform.stream.*;
6
7     public class MyXSLTProcessor {
8       public void process(Document d, String tmpl, String res)
9         throws TransformerException, DOMException
10      {
11        System.out.println("Creating transformer factory.");
12        TransformerFactory tf = TransformerFactory.newInstance();
13
14        System.out.println("Creating sources and results.");
15        DOMSource ds = new DOMSource(d);
16        StreamSource ss = new StreamSource(tmpl);
17        StreamResult sr = new StreamResult(new java.io.File(res));
18
19        System.out.println("Creating template & transformer");
20        Templates tt = tf.newTemplates(ss);
21        Transformer xformer = tt.newTransformer();
22
23        System.out.println("Performing transform to make " + res);
24        xformer.transform(ds, sr);
25      }
26    }
```

Summary

- XML is a technology for storing data in a standard, portable, and text-based way
  - Eases communication between remote programs
  - Basis of SOAP: web service messages
- Java has packages to help your Java programs parse XML, build XML documents, and perform XML transformations
- Two basic models for XML parsing
  - SAX: sequential access, event-driven
  - DOM: in-memory tree structure
- XSLT can be used to transform XML into other XML or other textual formats

XML Limitations

- Verbose descriptions
  - Can be redundant
- Hierarchical rather than relational model
  - Relational: databases
  - Must choose between

```
<movie>                <actor>
  <actor …>              <movie …>
  <actor …>              <movie …>
  …                      …
</movie>               </actor>
<movie>                <actor>
…                      …
```

Project 2: Multithreaded Web Server

- Completed writeup up is on line
- Create multithreaded web server
  - Handle performance issues with threads
  - Read from an XML configuration file
    - Configure your web server
  - JUnit Testing
    - “sufficient testing” -- use your judgement
- Test Plan
- Lots of opportunities for Extra Credit