Object – Oriented Design with UML and Java

Part XV: XML

Version 2011

Markup Languages

- Languages with tags the say something about the data.
- Example: Hypertext Markup Language (HTML)

```
<html>
<head><title>CSCI 4448 - General Information</title></head>
<body text="#000000" bgcolor="#FFFFFF" link="#0000FF" vlink="#3366FF"
alink="#33CCFF">
<b><i><font face="Veranda, Helvetica, sans-serif"><font color="#006600"><font
size=+2>Object Oriented Programming &amp;amp; Design
<br><b><i>University of Colorado at Boulder</font></font></font></i></b>
</body>
</html>
```

XML

eXtensible Markup Language

- Structured data in a text file
- XML looks a bit like HTML but isn't HTML
- XML is text, but isn't meant to be read
- XML is new, but not that new
- XML is not a really a markup language itself, but a meta-language for defining markup languages
  - HTML can be defined using XML
  - Groups defining standard domain-specific XML *dialects*
  - Home-grown XML dialects are common for single applications, too

So What's the Big Deal ?

XML is ubiquitous; you have to use it in the real world.

- XML is self-descriptive
- XML is platform and language neutral
- XML is license-free
- XML is widely supported
- XML is a great way for applications to communicate with each other, albeit verbosely

Self-Descriptive Data

- Consider data used in a pizza business to describe a pizza:
  - Style
  - Toppings
  - Size
  - Price

What do you think the following data record should mean?

- If the data came from a database, we need the database schema, and probably the database engine itself, to interpret the data.
- If the data came from a file, we need to write code to interpret the data.
- Adding or removing fields causes major problems.

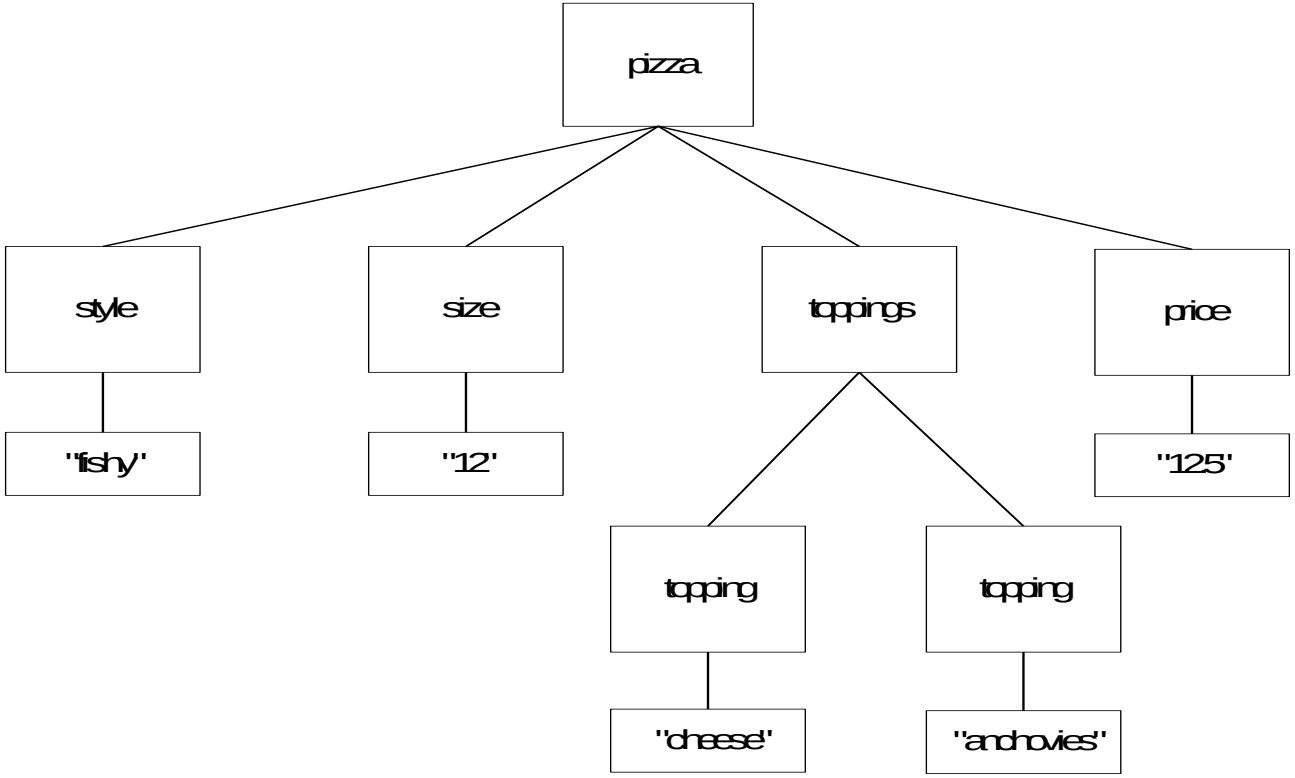
```
"fishy", 12, "cheese", "anchovies", 12.5
```

Self-Descriptive Data (cont.)

- Each datum is tagged with a descriptor that tells us about its semantics
- Possible XML representation for a pizza:

```
<pizza style="fishy">
<toppings>
   <topping>cheese</topping>
   <topping>anchovies</topping>
</toppings>
<size>12</size>
<price>12.5</price>
</pizza>
```

Self-Descriptive Data (cont.)



XML is Platform & Language Neutral

- Things are usually sent in a character format:
  - Usually ASCII Strings
  - Could be Unicode, although this is less common
- Documents are human-readable

But,

- This format can be inefficient (wasteful of bytes)
- Documents can get hard to read
- Writing documents can be error-prone
- This format can be awkward (eg: multiple “name spaces”)
- Binary data can be encoded, but programs at both ends of the conversation must understand the encoding

Special Characters in an XML File

Character Desired	Special String
&	&amp;
<	&lt;
>	&gt;
’	&apos;
”	&quote;

Components of an XML-based Application

- XML Document
  - This is the file that holds the XML text data
  - Domain-specific languages usually have name-spaces
  - Modern databases offer support using XQuery
- XML Parser
  - Invoked by an application program
  - Munches on the XML document and produces a run-time representation of the document that the application can use
- Application program
  - Creates internal structure of objects from the output of the parser
  - Creates new elements for the XML document

Parsers

- **SAX parsers**
  - Munches on the XML document and produces an event for each element
  - Fast and memory-efficient
  - Good for applications that process documents continually
  - Good for applications that are interested only in portions of large documents
- **DOM parsers**
  - Munches on the XML document and produces a tree structure
  - Good for applications that use XML documents for configuration
  - Good for applications that create or modify documents
- The leading parser is the **Xerces** parser from Apache
  - Available for Java, C++ and Perl
  - **`http://xml.apache.org`**

Using an XML Document

The straight-forward approach:

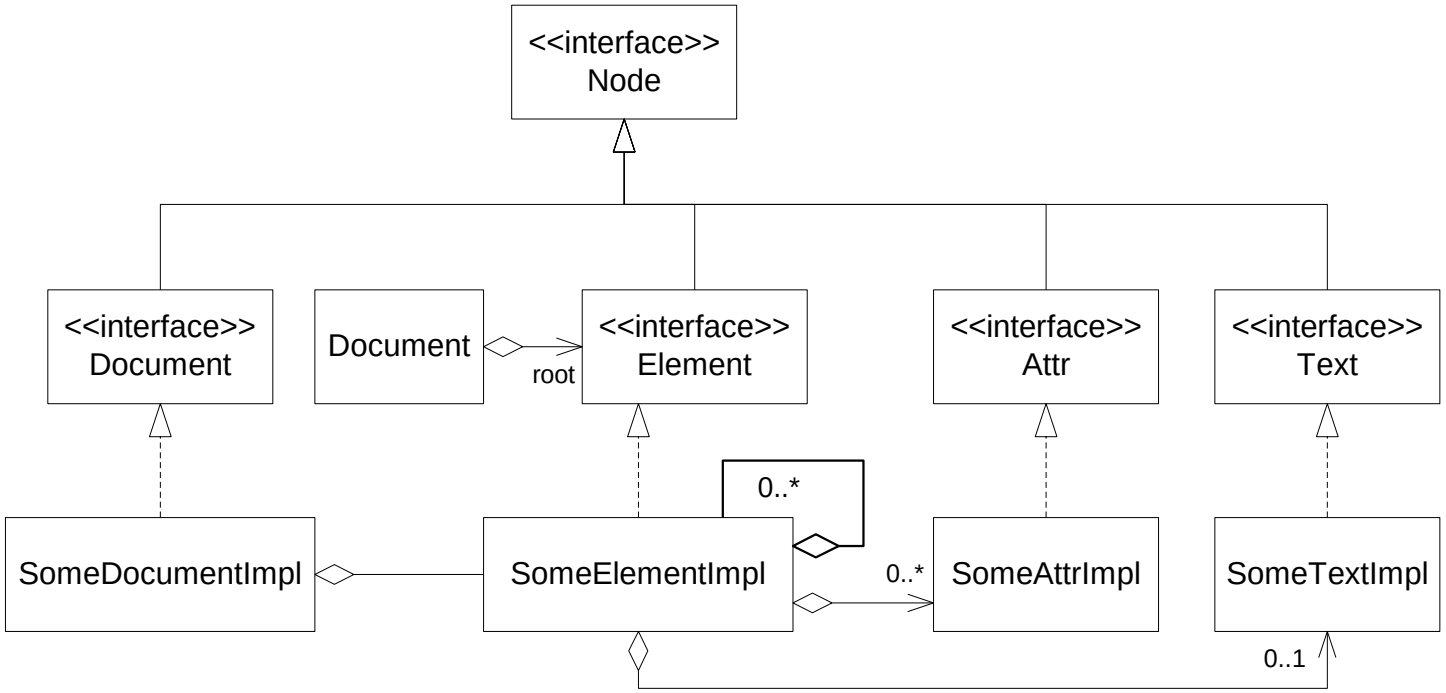
- Instantiate a parser
- Set the parser's features (if you don't want the defaults)
- Ask the parser to parse the file
- Ask the parser to create a document
- Walk around the Document's tree, creating instances of your application classes that correspond to the elements in the tree.

Tools exist to automate some or all of this (e.g.: JiBX)

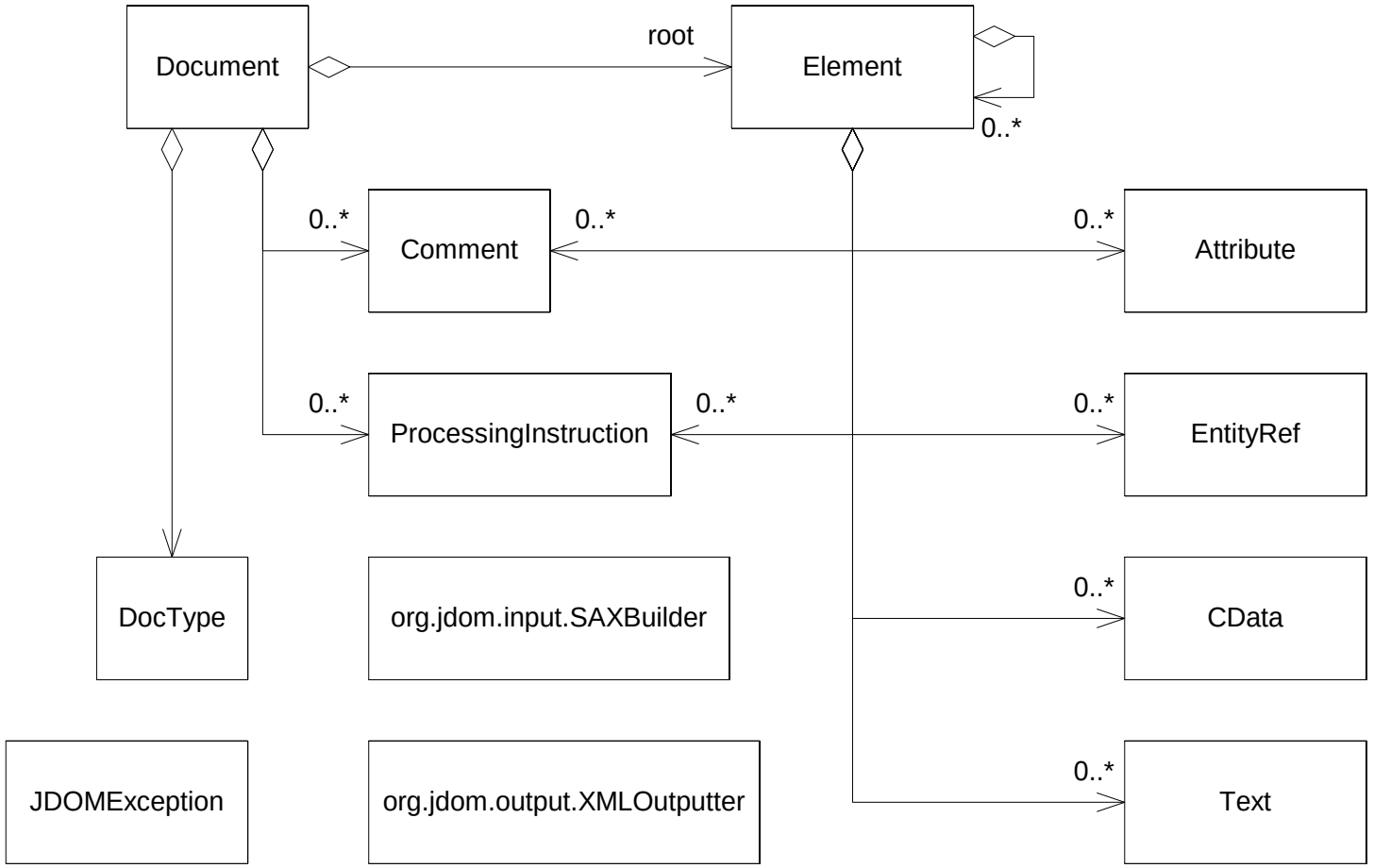
Using an XML Document (cont.)

- All values that you get from the DOM are Strings
  - You must write code to covert to primitive values.
  - This is considered by many to be a weakness of XML in its current incarnation.
- Every character in the XML file gets parsed, including all white space and non-printable characters.
  - Always call trim() on the string values you get back from the DOM.

DOM Class Diagram



JDOM Class Diagram



Document Object Model (DOM)

- A tree that corresponds to the XML document
- An API for walking the tree and manipulating it
- DOM predates JDOM, and is widely used
- Most Java programmers prefer JDOM over DOM
- We will focus on JDOM, a parser specifically for Java.
  - JDOM may use Xerces, or any other commercial XML parser
  - **`http://www.jdom.org`**

JDOM

JDOM is a convenient API for manipulating XML, designed with Java in mind, offering these improvements over DOM:

- Use Strings instead of having to use the old Text class.
- Use Java Collection classes such as List.
- Say goodbye to the old NodeList and NamedNodeMap classes.
- Say goodbye to the ubiquitous abstract class Node. Now if you want an Attribute, you get an Attribute, without having to downcast from Node.
- You can say `new Element("foo");` without having to use factories.
- JDOM provides convenient wrapper classes for parsing and for outputting XML files, such as :

  `DOMBuilder` & `SAXBuilder`

  `DOMOutputter` & `SAXOutputter` & `XMLOutputter`

JDOM - a future Java standard?

JDOM was accepted as JSR-102!

- A JSR is how formal Java specifications are defined.
- JDOM 1.1 available as of September 2008.

JDOM is a good example of iterative design; DOM was found to be cumbersome, and there is now a better way.

JDOM classes in a nutshell

- The class Document represents an XML document, and is a container for all the other stuff.
- Element as you expect, represents an XML element.
- Attribute as you expect, represents an XML attribute.
- Comment as you expect, represents an XML comment.
- Text is a class that you will rarely use, because JDOM provides a String interface where needed, for convenience.
- CData represents unparsed Character Data from an XML file's CDATA declaration.
- DocType represents an XML document's DOCTYPE declaration.
- EntityRef represents an XML entity reference.
- ProcessingInstructions are generally considered part of an XML document's header rather than content, per se.

SAXBuilder

- SAXBuilder uses any SAX parser, quickly building the Document in memory from a variety of XML input sources. DOMBuilder is an alternative that builds an `org.jdom.Document` from an `org.w3c.dom.Document`.
- SAXBuilder is very fast and easy to use.
- The `build()` method can take `Files, InputStreams, Readers, Strings & URLs` as input.
- To run this code, you must have `jdom.jar` and `xerces.jar` in your classpath (both available on the course web site):

```
import org.jdom.input.SAXBuilder;
import org.jdom.*;

SAXBuilder builder = new SAXBuilder();
Document doc = builder.build( file );
Element root = doc.getRootElement();
```

XMLOutputter

- The `output()` method can take `OutputStreams & Writers` as output destinations. There is also a set of `outputString()` methods.

```
import org.jdom.output.XMLOutputter;
import java.io.FileOutputStream;

XMLOutputter out = new XMLOutputter();
out.output( doc, new FileOutputStream( "doc.xml" ) );
```

Manipulating the Tree Structure

```
Element root = new Element( “sticksgame” );
Document doc = new Document( root );
root.addAttribute( new Attribute( “key”, “value” ) );
root.addContent( players[ 0 ].toXML() );

. . .

List pList = root.getChildren( “player” );
Element p0 = (Element) pList.get( 0 );
Attribute nameAttr = p0.getAttribute( "name" );
String p0Name = nameAttr.getValue();
```

Example: Saving a Sticks Game

```
<?xml version="1.0" encoding="UTF-8"?>
<sticksgame className="oop.sticks.SticksGame">
  <player className="oop.sticks.HumanPlayer"
          name="&lt;&lt; dave &gt;&gt;" />
  <player className="oop.sticks.ComputerPlayer"
          name="&lt;&lt; CP #1 MiniMax depth=5 &gt;&gt;"
          depth="5"/>
  <move className="oop.sticks.Move" row="3" numSticks="3" />
  <move className="oop.sticks.Move" row="4" numSticks="4" />
</sticksgame>
```

- Notice the "`<<`" instead of "<<".
- Notice that each Element has a **`className`** Attribute.
- Complete code for The Sticks Game can be found on-line; look for sticksgame.jar. To build & run the code, you will also have to have jdom.jar.

Saving a Sticks Game (cont.)

How can we design simple and reusable code to read and write the save game XML file?

- There are other ways to encode the state of the game as XML; for example, instead of having an ordered list of moves, it can have a dump of the board. The advantage of the ordered list of moves is that it facilitates having an undo feature.
- Let's exploit the fact that each Element has a **`className`** Attribute.
- Define a new interface, XMLizable, to read and write XML.
- Based on the given XML, these classes must implement the XMLizable interface:

`oop.sticks.HumanPlayer`

`oop.sticks.ComputerPlayer`

`oop.sticks.Move`

`oop.sticks.SticksGame`

- Design a ***Factory*** for creating XMLizable things. The Factory will assume that all XML Elements have a className Attribute, and that all such classes are XMLizable.

An XML Factory

```
package oop.xml;

import java.io.*;
import org.jdom.*;
import org.jdom.input.SAXBuilder;
import org.jdom.output.XMLOutputter;

public interface XMLizable {
  // All XMLizable things must have a className XML attribute
  // and a default (no-args) constructor.
  public static final String CLASS_NAME = "className";

  public Element toXML();
  public void    initFromXML( Element ele ) throws Exception;
}
```

An XML Factory (cont.)

```
public class XMLFactory {
  private XMLFactory() {}
  public static XMLizable readFile( String fileName ) {
    XMLizable rootObject = null;
    try {
      SAXBuilder builder = new SAXBuilder(); // no validation
      Document doc = builder.build( new FileInputStream( fileName ) );
      Element rootElement = doc.getRootElement();
      rootObject = makeObject( rootElement );
    }
    catch( Throwable t ) {
      System.out.println( "XMLFactory Can't read file : " + t );
      return null;
    }
    return rootObject;
  }
```

An XML Factory (cont.)

```
public static XMLizable makeObject( Element config )
throws Exception
{
  XMLizable domainObject = null;
  Attribute classAttr = config.getAttribute( XMLizable.CLASS_NAME );
  String className = classAttr.getValue();

  // The class must have a default (no-args) constructor.
  domainObject = (XMLizable)
    Class.forName( className ).newInstance();

  domainObject.initFromXML( config );
  return domainObject;
}
```

An XML Factory (cont.)

```
public static void writeFile( String fileName, XMLizable root ) {
  try {
    Element rootElement = root.toXML();
    Document doc = new Document( rootElement );
    XMLOutputter out = new XMLOutputter();
    out.setIndent( true );
    out.setNewlines( true );
    out.output( doc, new FileOutputStream( fileName ) );
  }
  catch( Throwable t )
  {
    System.out.println( "XMLFactory Can't write file : " + t );
} }
```

Back to the Sticks Game ...

```
// From class oop.sticks.ComputerPlayer
  public Element toXML() {
    Element player = super.toXML(); // calls Player's toXML()
    Attribute depthAttr = new Attribute( "depth", "" + searchDepth );
    player.addAttribute( depthAttr );
    return player;
  }

  public void initFromXML( Element config )
  throws Exception {
    super.initFromXML( config ); // calls Player's initFromXML()
    Attribute depthAttr = config.getAttribute( "depth" );
    String depthString = depthAttr.getValue();
    searchDepth = Integer.valueOf( depthString.trim() ).intValue();
  }
```

The Sticks Game XML mapping

```
// From class oop.sticks.SticksGame
 public Element toXML() {
    Element root = new Element( ”sticksgame" );
    Attribute classAttr = new Attribute( CLASS_NAME,
                                         getClass().getName() );
    root.addAttribute( classAttr );
    root.addContent( players[ 0 ].toXML() );
    root.addContent( players[ 1 ].toXML() );
    Vector moves = layout.getMoves();
    Iterator it = moves.iterator();
    while( it.hasNext() ) {
      Move move = (Move)it.next();
      root.addContent( move.toXML() );
    }
    return root;
  }
```

More XML Technologies

- XSL
  - A way of specifying transformations from one XML structure to another.
- XML:DB
  - Note that database vendors such as Oracle use their own XML-Types
  - Use XPATH to read information from an XML document (even from a database)
- DTD
  - Document Type Definition. Describe the structure of XML documents.
  - Schema (XSD) is newer and better...
- XSD
  - A standard for using an XML document to describe the legal structure of some other document (used to validate the document).
- SOAP
  - Simple Object Access Protocol. A standard way of sending XML service requests and responses using HTTP.
- WSDL
  - Web Services Description Language. An XML dialect for platform and language independent descriptions of programmatic services.

Validating XML (code example)

```
// Validate XML with an XSD
File xmlPath = new File( "c:/TEST.xml" );
File xsdPath = new File( "c:/TEST.xsd" );
URL url = xmlPath.toURL();
DocumentBuilderFactory parserFactory =
  DocumentBuilderFactory.newInstance();
parserFactory.setNamespaceAware(true);
DocumentBuilder parser = parserFactory.newDocumentBuilder();
org.w3c.dom.Document document = parser.parse(xmlPath);
DOMSource domSource = new DOMSource(document);

System.out.println("XML file loaded, but not validated.");
```

Validating XML - continued

```
org.w3c.dom.Element e = document.getDocumentElement();
System.out.println( "Root node = " + e.getNodeName());
SchemaFactory factory = SchemaFactory.newInstance(
  XMLConstants.W3C_XML_SCHEMA_NS_URI );

Source schemaFile = new StreamSource(xsdPath);
Schema schema = factory.newSchema(schemaFile);
System.out.println("XSD file loaded.");

Validator validator = schema.newValidator();
validator.validate( domSource );
System.out.println("XML file is VALID !!!");
```