Application Development with XML and Java

Lecture 7

XML Parsing

JDOM

JDOM Overview

- JDOM: Java Package for easily reading and building XML documents.
- Created by two programmers: Brett McLaughlin and Jason Hunter.
- Started as an open source project in 2000.
- Started because of programmers' general frustration with the regular DOM API Package.

- From the JDOM Web site:

"Build a better mousetrap and the world will beat a path to your door."

-- Emerson

- JDOM Web Site: http://www.jdom.org
- Official JDOM FAQ: http://www.jdom.org/docs/faq.html

JDOM Philosophy

- JDOM should be straightforward for Java programmers.
- JDOM should support easy and efficient document modification.
- JDOM should hide the complexities of XML wherever possible, while remaining true to the XML specification.
- JDOM should integrate with DOM and SAX.
- JDOM should be lightweight and fast.
- JDOM should solve 80% (or more) of Java/XML problems with 20% (or less) of the effort

JDOM and Sun

- Sun maintains a process called, "Java Community Process"
- This enables groups of developers to suggest new API for the core Java platform.
- To be considered, a new API is given a Java Specification Request (JSR.)
- JSRs are then reviewed, and are either accepted into the Java Platform or rejected.
- JDOM has been under consideration as JSR 102.
- Recently, JSR 102 was approved for eventual inclusion into the core Java platform.
- It is currently unclear which package JDOM will be included in: the standard kit or the enterprise kit.
- Either way, JDOM will probably become an official part of Java.
- JDOM is fast becoming a standard, and has received widespread support.

JDOM v. DOM

- DOM:
  - The main issue with the DOM is that it was defined to be language independent.
  - This is generally a good thing, because you can therefore use the DOM in multiple languages, e.g. C, C++, Perl, Java, etc.
  - However, it also presents several problems, including:
    - DOM is not optimized for Java, and
    - DOM API is very large
- JDOM
  - JDOM was designed for Java optimization.
  - Makes it much easier for regular Java programmers.
  - JDOM API is much smaller and more manageable than the DOM API.

JDOM v. DOM

- JDOM
  - JDOM was designed for Java optimization.
  - Makes it much easier for regular Java programmers.
  - JDOM API is much smaller and more manageable than the DOM API.

JDOM API Overview

- JDOM has four main packages:
  - org.jdom: classes that represent an XML document and its parts.
  - org.jdom.input: classes for reading a document into memory.
  - org.jdom.output: classes for writing a document to a stream or file.
  - org.jdom.adapters: classes for hooking up to DOM implementations (we are skipping this package.)

JDOM Parsers

- JDOM is not actually an XML parser.
- Rather, JDOM is an interface for manipulating/creating XML documents.
- To work, JDOM needs an XML parser to do the actual parsing.
- JDOM works with lots of different parsers:
  - Oracle XML Parser
  - Sun XML Parser
  - etc., etc.
- By default, JDOM will use the Xerces XML Parser.
- Xerces is automatically bundled with JDOM, so it works right out of the box.
- All the examples in class assume the default Xerces parser.

Basic Example

- jdom1.java: works with local files
  - Takes an XML file name from the command line.
  - Displays the specified XML file.
- To read in an XML file, use the org.jdom.input package.
- There are two main options:
  - SAXBuilder: uses a SAX parser to build the document (faster option, recommended.)
  - DOMBuilder: uses a DOM parser to build the document (slower option, not recommended.)

Using the SAXBuilder

- To read in a local file via the SAXBuilder:

```
SAXBuilder builder = new
  SAXBuilder();
Document doc=builder.build(new
  File(fileName));
```

- Once you have a builder object, call the `build()` method, and specify a File object.
- The `build()` method will return a `Document` object.
- The `Document` object encapsulates all data regarding your XML document.

XML Output: org.jdom.output

- To output an XML document, use the org.jdom.output package.
- The main class here is the **`XMLOutputter` class**.
- `XMLOutputter` is used to easily output any XML document to the screen or a specific file.

XMLOutputter

- To output an XML document to the screen:

```
XMLOutputter out = new
  XMLOutputter ();
out.output (doc,
  System.out);
```

- Here, **you are specifying an XML `Document` object, and an output stream.**

JDOM1 Example

- To use JDOM1, specify an XML file on the command line. For example:

```
java jdom1 document.xml
```

- Program output:

```
JDOM1 Example
Downloading file:  document.xml
<?xml version="1.0"
  encoding="UTF-8"?>
<!DOCTYPE DOCUMENT SYSTEM
  "simple.dtd">
<DOCUMENT trackNum="1234"
  secLevel="unclassified">
<TITLE>Sample Document</TITLE>
<AUTHOR>
<LASTNAME>Kosmopoulos</LASTNAME>
<COMPANY>City</COMPANY></AUTHOR>
<SUMMARY>This is element text and
  an entity follows:This is a
  very simple sample
 document.
</SUMMARY>
</DOCUMENT>
```



```
import avail.*;
import org.jdom.*;
import org.jdom.input.SAXBuilder;
import
org.jdom.output.XMLOutputter;

/**
 * Basic JDOM Example
 * Outputs any local XML file
specified on the command line
 * Example usage:
 * java jdom1 document.xml
 */

public class jdom1 {

  // Download and Output XML File
  public void process (String
fileName) {
    try {
      //  Use SAXBuilder
      SAXBuilder builder = new
SAXBuilder();
      Document doc =
builder.build(new File(fileName));
```



```
      //  Use XMLOutputter
      XMLOutputter out = new
XMLOutputter ();
      out.output (doc, System.out);
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public static void main (String[]
args) {
    System.out.println ("JDOM1
Example");
    System.out.println
("Downloading file:  "+args[0]);
    jdom1 app = new jdom1();
    app.process(args[0]);
  }

}
```


JDOMException

- In the event of a well-formedness error, JDOM will throw a JDOMException.
- For example, let us parse the XML document on the next slide…

```
<?xml version="1.0" encoding="UTF-
8"?>
<!DOCTYPE DOCUMENT SYSTEM
"simple.dtd">
<DOCUMENT trackNum="1234">
<TITLE>Sample Document</TITLE>
<AUTHOR>
<FIRSTNAME>Kosmas</FIRSTNAME>
<LASTNAME>Kosmopoulos</LASTNAME>
<COMPANY>City</COMPANY></AUTHOR>
<SUMMARY>This is element text and
an entity follows:
This is a very simple sample
document.
</SUMMARY>
```

End </DOCUMENT> tag is missing. Hence, document is not well-formed.

Well-formedness Errors

- jdom1 will output the following error:

```
JDOM1 Example
Downloading file:  document2.xml
org.jdom.JDOMException: Error on
   line 10 of document
   file:/home1/e/eqc3844/xml/document
   2.xml: The element type "DOCUMENT"
   must be terminated by the matching
   end -tag "</DOCUMENT>".
         at
   org.jdom.input.SAXBuilder.build(SA
   XBuilder.java:296)
         at
   org.jdom.input.SAXBuilder.build(SA
   XBuilder.java:617)
         at
   org.jdom.input.SAXBuilder.build(SA
   XBuilder.java:599)
         at
   jdom1.process(jdom1.java:20)
         at jdom1.main(jdom1.java:34)
```


Validation

- By default, validation is turned off.
- Hence, by default, errors in validity are completely ignored.
- To turn validation on, pass true to the Builder constructor. For example:

```
SAXBuilder builder = new SAXBuilder
   (true);
```

- With validation turned on, validation errors are reported as JDOMExceptions.

JDOM2

- JDOM2 turns validation on.
- Errors in well-formedness are reported as JDOMExceptions.
- Errors in validity are also reported as JDOMExceptions.
- If no errors occur, program outputs:

```
Document is well-formed
Document is valid
```

```
import avail.*;
import org.jdom.*;
import org.jdom.input.SAXBuilder;
import
org.jdom.output.XMLOutputter;

/**
 * Basic JDOM Example
 * Outputs any local XML file
specified on the command line
 * Performs XML Validation
 * Example usage:
 * java jdom2 document.xml
 */

public class jdom2 {

  // Download and Output XML File
  public void process (String
fileName) {
```



```
try {
    // Use SAXBuilder
    // turn XML validation on
    SAXBuilder builder = new SAXBuilder (true);
    Document doc = builder.build(new
File(fileName));
    // If we get here without any exceptions,
    // the document is both well-formed and valid
    System.out.println ("Document is well-
formed");
    System.out.println ("Document is valid");
   } catch (JDOMException e) {
    System.out.println ("JDOM Exception:
"+e.getMessage());
   }
 }

 public static void main (String[] args) {
   System.out.println ("JDOM2 Example with
Validation");
   System.out.println ("Downloading file:
"+args[0]);
   jdom2 app = new jdom2();
   app.process(args[0]);
 }
}
```


JDOM Tree

- Every XML document will have a root element.
- For example, in the following XML document:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE WEATHER SYSTEM
   "Weather.dtd">
<WEATHER>
   <CITY NAME="Hong Kong">
     <HI>87</HI>
     <LOW>78</LOW>
   </CITY>
</WEATHER>
```

- WEATHER is the root element.

Working with the Root Node

- To get the root Element, use the following code:

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

- Once you have the root node, you can walk through the XML tree.

Element Data

- Given any XML element, you can retrieve the element name or the embedded text.
- For example:
  - element.getName(): Returns name of the element.
  - element.getText(): Returns embedded text.

Getting Element Children

- You can also query an Element for a list of children.
- For example, given the following XML:

```
<CITY NAME="Hong Kong">
  <HI>87</HI>
  <LOW>78</LOW>
</CITY>
```

- The CITY element has two children: HI and LOW.

Getting Element Children

- To get a List of children, call the getChildren() method.
- Example:

```
List kids = element.getChildren();
Iterator iterator = kids.iterator();
while (iterator.hasNext()) {
 Element kid = (Element) iterator.next();
 processElement (kid);
}
```

- getChildren() returns a Java List object.
- You can then iterate through the list of children.

Example: jdom3.java

- jdom3.java downloads an RSS News File containing top news stories.
- Once downloaded, the program "walks" the entire XML tree, and simply prints out the names of all elements within the document.
- It also keeps a running count of the number of elements.

Sample Source File

```
<?xml version="1.0" encoding="iso-8859-
  1"?>
<!DOCTYPE moreovernews SYSTEM
  "http://p.moreover.com/xml_dtds/moreo
  vernews.dtd">
   <moreovernews>
      <article id="_28421464">

  <url>http://c.moreover.com/click/here
  .pl?x28421450</url>
        <headline_text>NIAID
  Establishes Functional Genomic
  Research Center</headline_text>
        <source>UniSci</source>
        <media_type>text</media_type>
        <cluster>moreover...</cluster>
        <tagline> </tagline>

  <document_url>http://unisci.com/</doc
  ument_url>
        <harvest_time>Dec  1 2001
  10:59AM</harvest_time>
        <access_registration>
  </access_registration>
        <access_status>
  </access_status>
      </article>
```

Sample Output

moreovernews
article
url
headline_text
source
media_type
cluster
tagline
document_url
harvest_time
access_registration
access_status
article
url
…

The program simply outputs all elements within the XML document.


```
import avail.*;
import java.net.*;
import java.util.List;
import java.util.Iterator;
import org.jdom.*;
import org.jdom.input.SAXBuilder;
import org.jdom.output.XMLOutputter;

/**
 * Basic JDOM Example
 * Illustrates how to traverse a JDOM
tree
 * Example usage:
 * java jdom4
 */
public class jdom4 {
  private int numElements = 0;

  // Download and Output XML File
  public void process (String url)
throws MalformedURLException {
```



```
try {
      //  Use SAXBuilder
      SAXBuilder builder = new
SAXBuilder();
      Document doc = builder.build(new
URL(url));
      Element root =
doc.getRootElement();
      processElement (root);
      System.out.println ("Total
Number of Elements Processed:  "
        +numElements);
    } catch (JDOMException e) {
      System.out.println ("JDOM
Exception:  "+e.getMessage());
    }
  }
```



```
 //  Recursive Function to Process
Elements
  //  Prints the Element Name and keeps
a running count
  //  out total number of elements.
  private void processElement (Element
element) {
    numElements++;
    String elementName =
element.getName();
    System.out.println (elementName);
    List kids = element.getChildren();
    Iterator iterator =
kids.iterator();
    while (iterator.hasNext()) {
      Element kid = (Element)
iterator.next();
      processElement (kid); //
Recursive
    }
  }
```

```
  public static void main (String[]
args) throws Exception {
    System.out.println ("JDOM3
Example");
    jdom3 app = new jdom3();

app.process("http://p.moreover.com/cgi
-local/page?"+
      "c=Top%20stories&o=xml");
  }

}
```