XML programming in Java technology, Part 3

Presented by developerWorks, your source for great tutorials

ibm.com/developerWorks

Table of contents

If you're viewing this document online, you can click any of the topics below to link directly to that section.

Section 1. Introduction

About this tutorial

In an *earlier tutorial*, I showed you the basics of XML parsing in the Java language. I covered the major APIs (DOM, SAX, and JDOM), and went through a number of examples that demonstrated the basic tasks common to most XML applications. The *second tutorial in the series* covered parser features, namespaces, and XML validation. This final tutorial looks at more difficult things that I didn't cover before, such as:

- Building XML structures without an XML document
- Converting between one API and another (SAX events to DOM trees, for example)
- Manipulating tree structures

Programming interfaces

As in the previous tutorials, I cover these APIs:

- The Document Object Model (DOM), Levels 1, 2, and 3
- The Simple API for XML (SAX), Version 2.0
- JDOM

Although many of the sample programs I discuss here use JAXP (the Java API for XML parsing), I won't discuss JAXP specifically in this tutorial.

About the examples

Most of the examples here work with the Shakespearean sonnet that appeared in the previous tutorials. The structure of this sonnet is:

```
<sonnet>
  <author>
    <lastName>
    <firstName>
    <nationality>
```

```
      <yearOfBirth>
      <yearOfDeath>
    </author>
    <lines>
      [14 <line> elements]
    </lines>
  </sonnet>
```

I'll use this sample document throughout this tutorial. Links to the complete set of sample files are shown below:

- sonnet.xml on page 39
- sonnet.dtd on page 39
- ParseString.java on page 40
- DomBuilder.java on page 41
- JdomBuilder.java on page 46
- CsvToSax.java on page 48
- test.csv on page 52
- SaxToDom.java on page 53
- DomToSax.java on page 58
- DomSorter.java on page 63
- DomAttributes.java on page 65
- JdomSorter.java on page 67
- sonnetSorter.xsl on page 68
- DomThree.java on page 69
- DomTreePrinter.java on page 70
- DomFour.java on page 72
- SaxKiller.java on page 74

As an alternative, you can download x-java3_codefiles.zip to view these files in a text editor.

In addition to the sonnet, you'll also learn how to parse files of comma-separated values and text strings, including several approaches to converting that information into XML or XML data structures.

Setting up your machine

You'll need to set up a few things on your machine before you can run the examples. (I'm assuming that you know how to compile and run a Java program, and that you know how to set your `CLASSPATH` variable.)

1. First, visit *the home page of the Xerces XML parser* (http://xml.apache.org/xerces2-j/) at the Apache XML Project (http://xml.apache.org/xerces2-j/). You can also go directly to the *download page* (http://xml.apache.org/xerces2-j/download.cgi) (http://xml.apache.org/xerces2-j/download.cgi).
2. Unzip the file that you downloaded from Apache. This creates a directory named `xerces-2_5_0` or something similar, depending on the release level of the parser. The JAR files you need (`xercesImpl.jar` and `xml-apis.jar`) should be in the Xerces root directory.
3. Visit *the JDOM project's Web site* (http://jdom.org/) and download the latest version of JDOM (http://jdom.org/).
4. Unzip the file you unloaded from JDOM. This creates a directory named `jdom-b9` or something similar. The JAR file you need (`jdom.jar`) should be in the `build` directory.
5. Finally, download the zip file of examples for this tutorial, x-java3_codefiles.zip, and unzip the file.
6. Add the current directory (`.`), `xercesImpl.jar`, `xml-apis.jar`, and `jdom.jar` to your `CLASSPATH`.

About the author

In a multicenter, double-blind clinical test, *Doug Tidwell* (mailto:dtidwell@us.ibm.com?cc=dwxed@us.ibm.com) was shown to provide significant relief from seasonal allergy symptoms caused by programming with XML and Java technologies.

Also available is slow-acting Doug Tidwell (Doug Tidwell SA), which delivers a consistent dose of medication for up to 24 hours. Side effects of Doug Tidwell were generally mild and included dizziness, moderate to severe nausea, numbness in the extremities, and, in rare cases, paralysis and death.

Ask your doctor if Doug Tidwell is right for you.

For further details, consult *Doug's blog*.

Section 2. Building XML structures from scratch

Parsing a string

Sometimes you might want to parse an XML string. Typically, a parser works with an XML document stored in a file. If another component sends you a string containing an XML document, you don't want to write the string out to a file and then read the file back in and parse it. What you want to do instead is invoke the parser against the string itself.

The trick is to convert the Java `String` into an `org.xml.sax.InputSource`. The parser -- whether it's DOM-based or SAX-based (whether it uses JAXP's `DocumentBuilder` or `SAXParser`) -- can take the `InputSource` and parse it just like any other markup. To convert the `String`, the code is:

```
ParseString ps = new ParseString();
String markup = new String("<html><body><h1>" +
                           "This XML document was a " +
                           "<b>string!</b>" +
                           "</h1></body></html>");
InputSource iSrc = new InputSource(new StringReader(markup));
ps.parseAndPrint(iSrc);
```

Here, you create the `InputSource` from a `StringReader`, which you created from the `String`. Within the `parseAndPrint` method, the code looks very similar to the parsing samples in the previous tutorials:

```
public void parseAndPrint(InputSource xmlSource)
{
  Document doc = null;

  try
  {
    DocumentBuilderFactory dbf =
      DocumentBuilderFactory.newInstance();
    DocumentBuilder db = dbf.newDocumentBuilder();
    doc = db.parse(xmlSource);
    if (doc != null)
      DomTreePrinter.printNode(doc);
  }
```

The only change is that this method takes `InputSource` as its input an instead of a URL. To print the string, you use the `com.ibm.dw.xmlprogjava.DomTreePrinter` class. The complete source code is in the files ParseString.java on page 40 andDomTreePrinter.java on page 70.

Building a DOM tree from scratch

In the earlier DOM applications, you got a DOM tree from the parser after it parsed an XML file. Sometimes, you might want to create a DOM tree without an XML source file. For example, you might need to convert the results of an SQL query into a DOM tree, and then use a library of XML tools against the DOM tree.

The `DomBuilder` application does this. Although all of the nodes it builds are hard-wired into the application, you can easily add your own code to generate the nodes you want.

As you'd expect, you need to start by asking the factory object to create a `DocumentBuilder`:

```
try
{
  DocumentBuilderFactory dbf =
    DocumentBuilderFactory.newInstance();
  DocumentBuilder docBuilder = dbf.newDocumentBuilder();
    Document doc = docBuilder.getDOMImplementation().
    createDocument("", "sonnet", null);
  . . .
```

Start by creating a new `DocumentBuilderFactory` and a new `DocumentBuilder` as before. Next, call `DocumentBuilder.getDOMImplementation()` to get an instance of something that implements the `DOMImplementation` interface. Use that object's `createDocument` method to get a new `Document` object. (Note: `DOMImplementation` is part of the Document Object Model, not JAXP.)

In this example, the three arguments to the `createDocument` method specify that your new `Document` doesn't have a namespace, the root element name is `sonnet`, and the `Document` doesn't have a `DOCTYPE`.

Using the **`DocumentElement`**

Now that you've created your `Document`, you need to add things to it. The code sample below starts by getting the `DocumentElement` from the `Document` object. The difference between the two is subtle but important: The `Document` object is the entire structure that represents the parsed version of the XML document; the `DocumentElement` is the root element that contains all of the

XML markup. (Comments can appear outside the root element of an XML document; those comments would be in the `Document` object, but not the `DocumentElement`.) In the sample XML file, the `<sonnet>` element contains the rest of the document.

```
. . .
Document doc = docBuilder.getDOMImplementation().
  createDocument("", "sonnet", null);

Element root = doc.getDocumentElement();
root.setAttribute("type", "Shakespearean");

Element author = doc.createElement("author");

Element lastName = doc.createElement("lastName");
lastName.appendChild(doc.createTextNode("Shakespeare"));
author.appendChild(lastName);
. . .
Element yearOfDeath = doc.createElement("yearOfDeath");
yearOfDeath.appendChild(doc.createTextNode("1616"));
author.appendChild(yearOfDeath);

root.appendChild(author);
. . .
```

In this listing, you set the `type` attribute of the root element (`<sonnet>`), then you create the `<author>` element. **Throughout the code, the `Document` object is used as a factory to create new `Nodes`.** Your code also has to create the hierarchy of the document. To build the `<author>` element, you create the `<author>` element itself, then you create the other elements contained in `<author>` (`<lastName>`, `<firstName>`, and so forth). As you create the child elements of `<author>`, you append them to the `<author>` element. When the `<author>` element is complete, you append it to its parent, the `<sonnet>` element.

Finally, notice the awkwardness of adding text to an element. To create this markup:

```
<yearOfDeath>1616</yearOfDeath>
```

You have to create the `<yearOfDeath>` element, then create a text node containing the text `1616`, then append the text node to the `<yearOfDeath>` element, then you can append the `<yearOfDeath>` element to the `<author>` element. To add a text node to an element, there is no method like `Element.setText()` as you might expect. Peculiarities like this are what led to the creation of JDOM; I'll show you how to build an XML document with JDOM in just a moment.

Running **`DomBuilder`**

To run this application, simply type `java DomBuilder` at the command line. You should see output like this:

```
C:\adv-xml-prog>java DomBuilder
<?xml version="1.0" ?>
<sonnet type="Shakespearean"><author><lastName>Shakespeare
</last-name><firstName>William</firstName><nationality>Bri
tish</nationality><yearOfBirth>1564</yearOfBirth><yearOfDe
ath>1616</year-of-death></author><title>Sonnet 130</title>
<lines><line>My mistress' eyes are nothing like the sun,</
line><line>Coral is far more red than her lips red.</line>
<line>If snow be white, why then her breasts are dun,</lin
e><line>If hairs be wires, black wires grow on her head.<l
```

You don't have to include an XML file name, because the code builds the DOM tree from scratch (in fact, that's the point of this example). As the output shows, you don't have any whitespace nodes in your DOM tree because you haven't gone to the trouble of putting them into the tree.

Later, in Generating SAX events from comma-separated values on page 9, I'll show you how to generate SAX events from a variety of sources. For now, you can look at DomBuilder.java on page 41 for the complete source listing of `DomBuilder`.

Building a JDOM **`Document`** from scratch

Building a JDOM `Document` is, as you'd expect, much easier than the DOM version of the task. As you may recall from JDOM's goals, creating the `Document` works much the same as creating a Java object. Here's how the code starts:

```
public void buildDocument()
{
  Element root = new Element("sonnet");
  root.setAttribute("type", "Shakespearean");

  Vector author = new Vector();
  author.add(new Element("lastName").addContent("Shakespeare"));
  author.add(new Element("firstName").addContent("William"));
  author.add(new Element("nationality").addContent("British"));
  author.add(new Element("yearOfBirth").addContent("1564"));
  author.add(new Element("yearOfDeath").addContent("1616"));
```

```
    root.addContent(new Element("author").setContent(author));

    root.addContent(new Element("title").addContent("Sonnet 130"));
```

Notice how much simpler this code is. For example, when adding text to a node using the DOM, you had to create a text node (using the document node as an element factory), then make the text node a child of an element, then make that element the child of another element, and so on. With JDOM, you can create a `Vector` of elements, then use the `setContent` method to add everything in the `Vector` as a child of some other element.

Once you've set the content of your `root` element, you can create a JDOM `Document` object with it:

```
  Vector lines = new Vector();

  lines.add(new Element("line").
    addContent("My mistress' eyes are nothing like the sun,"));
  . . .
  lines.add(new Element("line").
    addContent("As any she belied with false compare."));

  root.addContent(new Element("lines").setContent(lines));

  Document doc = new Document(root,
                      new DocType("sonnet", "sonnet.dtd"));

  try
  {
    XMLOutputter xo = new XMLOutputter("  ", true);
    xo.output(doc, System.out);
  }
```

As in the JDOM applications from the previous tutorials, you use an `XMLOutputter` to write the document to the console. Notice that JDOM lets you create a `DOCTYPE` declaration when you create the `Document` object.

For the complete source code, see JdomBuilder.java on page 46.

Generating SAX events from comma-separated values

The final SAX-generating example illustrates how to generate SAX events from a non-XML data source. This technique is extremely useful. Any code that processes data can fire SAX events from that data, allowing a SAX parser to treat that data as an XML data source. The next section, Converting from one API to another on page 14, contains an example called`SaxToDom` that converts

SAX events into DOM objects; combining the two techniques gives you an extremely flexible way to process many different kinds of data.

In this example, the data source used is a file of comma-separated values, also known as a CSV file. (Code that accesses a database through a JDBC driver would also be a good example, but the database connection is more complicated.) You'll use a Java `StreamTokenizer` to parse the files, then you'll generate the appropriate XML from the tokens found in the data.

Here are a few lines of the sample file, `test.csv`:

```
"000010","CHRISTINE","I","HAAS","A00","3978",19650101
"000020","MICHAEL","L","THOMPSON","B01","3476",19731010
"000030","SALLY","A","KWAN","C01","4738",19750405
"000050","JOHN","B","GEYER","E01","6789",19490817
```

This listing was generated from a SQL query, with each line in the file containing seven data fields. Use the following XML element names to wrap this data:

```
static String tagNames[] = {"employeeNumber",
                            "firstName",
                            "middleInitial",
                            "lastName",
                            "deptNo",
                            "extension",
                            "dateOfBirth"};
```

CSV to SAX, continued

To start, set up the `StreamTokenizer`:

```
BufferedReader br = new BufferedReader(new FileReader(uri));
StreamTokenizer st = new StreamTokenizer(br);

st.eolIsSignificant(true);
st.whitespaceChars(',', ',');
st.quoteChar('"');
```

This defines the properties of the `StreamTokenizer`. Now define three character arrays that you'll use to pretty-print the XML output:

```
char [] lineBreak = new String("\n").toCharArray();
char [] singleIndent = new String("  ").toCharArray();
char [] doubleIndent = new String("    ").toCharArray();
```

As you parse the items in the CSV file, go through the following steps:

1. Fire the `startDocument` event.
2. Fire the `startElement` event for the `<employees>` element.
3. For each row, do the following:
   A. Fire the `startElement` event for the `<employee>` element.
   B. Loop through all of the tokens in this row. You'll use your static array of tag names to generate the XML elements; for example, the first element will be `<employeeNumber>`, the second will be `<firstName>`, and so forth. For each token, fire `startElement` for the element, fire `characters` for the text, then fire `endElement`.
   C. Fire the `endElement` event for the `<employee>` element.
4. Fire the `endElement` event for the `<employees>` element.
5. Fire the `endDocument` event.

Firing the SAX events

First of all, fire the `startDocument` event. You know you're going to wrap the entire XML document in an `<employees>` element, so you can go ahead and fire `startElement`. After processing the entire document, you'll fire the `endElement` event for the `<employees>` element. Here's the code:

```
dh.startDocument();
dh.startElement(null, null, "employees", null);
dh.ignorableWhitespace(lineBreak, 0, lineBreak.length);
```

(Notice that the `ignorableWhitespace` event is fired to add a line break to the output.)

At this point, you should set up nested `while` loops to process the rows of the file. The outer `while` loop executes until the `StreamTokenizer` returns a type equal to the end-of-file marker (`StreamTokenizer.TT_EOF`). For each iteration through the loop, call `startElement` for the `<employee>` element, process all of the items in the current line of the source file, then call `endElement` for `<employee>`. The inner loop processes each line until `StreamTokenizer` finds either the end-of-line marker (`TT_EOL`) or the end-of-file marker (`TT_EOF`). Here's the first section of the code:

```
st.nextToken();
while (st.ttype != StreamTokenizer.TT_EOF)
```

```
{
  dh.ignorableWhitespace(singleIndent, 0, singleIndent.length);
  dh.startElement(null, null, "employee", null);
  dh.ignorableWhitespace(lineBreak, 0, lineBreak.length);

  int i = 0;
  while (st.ttype != StreamTokenizer.TT_EOL &&
         st.ttype != StreamTokenizer.TT_EOF)
  {
```

Notice that you're using the variable `i` to count how many elements the tokenizer has found. Use this to retrieve the element name from the array discussed earlier.

The **`StreamTokenizer`** class

Before I continue, a few words about how the `StreamTokenizer` class works. The first time you use the class, you need to call the `nextToken` method. That method tells the tokenizer to find the next token in the file (in this case, the first one). At that point, you can get that token from the tokenizer object by using the `nval` field for numeric values and by using the `sval` field for string values. You also use the `ttype` field to determine the token's type. The code for converting a numeric token to a series of SAX events is:

```
if (st.ttype == StreamTokenizer.TT_NUMBER)
{
  char [] chars = BigInteger.valueOf((long)st.nval).
                  toString().toCharArray();
  dh.ignorableWhitespace(doubleIndent, 0, doubleIndent.length);
  dh.startElement(null, null, tagNames[i], null);
  dh.characters(chars, 0, chars.length);
  dh.endElement(null, null, tagNames[i]);
  dh.ignorableWhitespace(lineBreak, 0, lineBreak.length);
}
```

Java's `BigInteger` class is used here to handle the very large values that can appear in comma-separated files. (For example, dates from relational databases are often encoded as eight-digit numbers.) Convert the numeric value to a character array, then invoke the appropriate SAX events.

At the end of the inner loop, increment the counter (`i`) and call the `StreamTokenizer.nextToken` method to advance the tokenizer. When the inner loop ends, invoke `endElement` for the `<employee>` element. When the outer loop ends, invoke `endElement` for the `<employees>` element, followed by the `endDocument` event:

```
          st.nextToken();
          i++;
        }

        dh.ignorableWhitespace(singleIndent, 0, singleIndent.length);
        dh.endElement(null, null, "employee");
        st.nextToken();
        dh.ignorableWhitespace(lineBreak, 0, lineBreak.length);
      }

      dh.endElement(null, null, "employees");
      dh.ignorableWhitespace(lineBreak, 0, lineBreak.length);
      dh.endDocument();
    }
```

Using this technique, you've now converted a comma-separated data stream into a series of SAX events that represent that data as if it were an XML document. You can use this approach for any kind of structured or semi-structured data.

You can see the complete source code in CsvToSax.java on page 48. The sample data file is in test.csv on page 52.

Section 3. Converting from one API to another

Converting SAX events to DOM trees

Typically an XML application uses DOM or SAX, however sometimes you might want to use *both* interfaces together. For example, suppose you have a reporting system that generates invoices for 10,000 customers. Those invoices are created as a single XML file containing 10,000 `<invoice>` elements. To process each `<invoice>`, you want to use a DOM tree. Unfortunately, you don't have a machine with enough memory to create a DOM tree with potentially millions of objects representing those 10,000 invoices.

For this example, you could use a hybrid approach. To parse the XML file with a SAX parser, use all of the SAX events for a given `<invoice>` to build a DOM tree. When you get a `startElement` event for an `<invoice>` element, you create a new DOM tree. As your code receives SAX events, you add the appropriate `Node`s to the DOM tree. When you get an `endElement` event for the `<invoice>`, pass the DOM tree to your invoice-processing routine. When you've processed the current `<invoice>`, you can delete the DOM tree and start over with a new set of SAX events.

Mapping SAX events to DOM objects

To convert SAX events into a DOM tree, consider the most common SAX events:

`startDocument`
It's reasonable to think you'd use this event to ask your `DocumentBuilder` object to create a `Document`. Unfortunately, `startDocument` doesn't tell you the *name* of the root element, so you have to do that in the `startElement` handler.

`startElement`
You have to handle two cases of `startElement`:

- If this is the first `startElement` event (in other words, it's the root element), use your `DocumentBuilder` object to create a new `Document`. The information in the `startElement` event tells you the name of the root element, among other things, so you'll use that information to set the root element's name.
- If this isn't the first `startElement` event, use the `Document` object to create a new `Element`. Any attributes contained in the `startElement` event are added to the new `Element`. When you're finished, put the new `Element` on a stack.

`characters`
: For the `characters` event, create a new `Text` node and append it as a child of the node on the top of your stack.

`ignorableWhitespace`
: If you want to include whitespace in the DOM tree, create a new `Text` node that contains the whitespace. As with ordinary character events, you can append it to the node on the top of the stack.

`endElement`
: The `endElement` means the parser has found the end of an element. That means you need to pop the complete element off the stack, then add it as a child of the element that's now at the top of the stack. (To avoid an `EmptyStackException`, make sure that a parent element is on the stack.)

`endDocument`
: Ignore this event. After you process the final `endElement` event, the stack will contain a single item, the root element of the document.

Next, I'll show you the event handlers in `SaxToDom` and demonstrate how you can create DOM objects as SAX events arrive.

Using a stack

I mentioned earlier that SAX events are stateless. A given `characters` event merely tells you that the parser found some characters in the document; it doesn't tell you anything about the element that contains those characters. If you need that information (and you often do), you have to keep track of it yourself.

As you get events from the SAX parser, you can convert each event into the appropriate type of DOM `Node`. Once you've created the `Node`, you need to know its parent. The most efficient way to do this is with a stack -- specifically `java.util.Stack`. Process the different DOM `Node` types as follows:

° For an `Element` node, when you create it, put it on the top of the stack. When you have the complete element (when you receive the `endElement` event), pop the element off the stack and append it to the element now on the top of the stack.
° For a `Text` node, create it and append it as a child of the element on top of the stack. You create `Text` nodes in response to both `characters` and `ignorableWhitespace` events.

In both cases, the `peek()` method of the `java.util.Stack` class lets you append the text node without having to pop the element off the stack first.

In your code, you're ignoring other node types such as `ProcessingInstruction` and `Comment`. If you were creating those node types, you would simply append them to the element on the top of the stack. (If you implement this, be aware that comments and processing instructions can occur outside the root element.)

The **`startElement`** event handler

To handle the `startElement` event, you usually create a new DOM `Element` and put it on the stack. However, this situation has an extra complication: For the *first* `startElement` event, you need to create a new DOM `Document` object. All of the other DOM objects you create will be descendants of the `Document` object. Here's the code:

```
if (firstElementNotFoundYet)
{
  root = docBuilder.getDOMImplementation().
    createDocument(namespaceURI, rawName, null);
  Element docElement = root.getDocumentElement();
  if (attrs != null)
  {
    int len = attrs.getLength();
    for (int i = 0; i < len; i++)
      docElement.setAttribute(attrs.getQName(i),
                              attrs.getValue(i));
  }

  elementStack = new Stack();
  elementStack.push(docElement);
  firstElementNotFoundYet = false;
}
else
{
  Element currentElement = root.createElement(rawName);
  if (attrs != null)
  {
    int len = attrs.getLength();
    for (int i = 0; i < len; i++)
      currentElement.setAttribute(attrs.getQName(i),
                                  attrs.getValue(i));
  }
  elementStack.push(currentElement);
}
```

The method calls here -- such as `createDocument()`, `getDocumentElement()`, and `createElement()` -- are the same ones you used in `DomBuilder`. The main difference in the processing here is the use of the `elementStack` to keep track of the latest `Element` you've created.

The **`characters`** and **`ignorableWhitespace`** event handlers

You can handle these events by creating a new text node and adding it to the `Element` on top of the stack. You can use the `Stack.peek()` method to access the top item on the stack without actually removing it. These two event handlers are coded as:

```
public void ignorableWhitespace(char ch[], int start, int length)
{
  characters(ch, start, length);
}

public void characters(char ch[], int start, int length)
{
  ((Element) elementStack.peek()).
    appendChild(root.createTextNode(new String(ch, start, length)));
}
```

The `ignorableWhitespace` event handler merely calls the `characters` event handler. In the `characters` event handler, you can use the `peek()` method to access the item on the top of the stack. Notice that you have to cast the item to an `Element`; the `peek()` method returns a Java `Object`. You can then create a new text node and append it to the `Element` on the top of the stack.

The **`endElement`** event handler

To handle the `endElement` event, you need to remove the item on the top of the stack and append it to the item that was previously beneath it. The one exception to this is the `endElement` event at the end of the document; for that event, the stack will have only one item. Here's how the code looks:

```
public void endElement(String namespaceURI, String localName,
                       String rawName)
{
  if (elementStack.size() > 1)
  {
    Element currentElement = (Element) elementStack.pop();
    ((Element) elementStack.peek()).appendChild(currentElement);
  }
}
```

If the stack has more than one item on it, you can pop the current element off the stack, then append it to the item that's now on top of the stack.

When you receive the final `endElement` event, a single root element is on top of the stack. You can then pop the root element off the stack and process it.

Wrapping it all up

All that's left to do now is to create the parser object that will parse the file and build the DOM tree from the SAX events. The source of the `parseAndPrint` method is:

```
Document root = null;   // global variable
. . .
public void parseAndPrint(String uri)
{
  try
  {
    dbf = DocumentBuilderFactory.newInstance();
    docBuilder = dbf.newDocumentBuilder();

    SAXParserFactory spf = SAXParserFactory.newInstance();
    SAXParser sp = spf.newSAXParser();
    sp.parse(uri, this);

    if (root != null)
      printDomTree(root);
  }
  . . .
```

Here, you create a `DocumentBuilder`, which is used to create DOM objects. You also create a `SAXParser` object to parse the XML file and generate SAX events.

When you run the sample, you will see:

```
C:\adv-xml-prog>java SaxToDom sonnet.xml
<?xml version="1.0"?>
<!DOCTYPE sonnet SYSTEM "sonnet.dtd">
<sonnet type="Shakespearean">
  <author>
    <lastName>Shakespeare</lastName>
    <firstName>William</firstName>
    <nationality>British</nationality>
    <yearOfBirth>1564</yearOfBirth>
    <yearOfDeath>1616</yearOfDeath>
  </author>
  <title>Sonnet 130</title>
  <lines>
    <line>My mistress' eyes are nothing like the sun,</line>
    . . .
```

For a complete listing of the source code, see SaxToDom.java on page 53.

Generating SAX events from a DOM tree

You've now seen how to create a DOM tree from SAX events. Next I'll show you how to generate SAX events from a DOM tree. I'm showing you this for the sake of completeness; I'm not sure why anybody would need to do this. (If you think of a use for this technique, please *let me know* (mailto:dtidwell@us.ibm.com) .)

First, take a look at the mapping between nodes in a DOM tree and SAX events:

`DOCUMENT_NODE`
: Call `startDocument()`, then process everything in the document node, then call `endDocument()`. To process everything in the document, use the recursive technique that was used throughout the DOM examples.

`ELEMENT_NODE`
: Now take a look at the element to gather any attributes it has; you'll need those when you call `startElement()`. After you call `startElement()`, you'll process all of the element's children, then call `endElement()`.

`TEXT_NODE`
: For this node type, you simply create a `char` array that contains the node's value.

The other major task is that your code has to implement the `DefaultHandler` interface. That interface defines the SAX event handlers. As your code traverses the DOM tree, you'll create SAX events and send them to yourself.

Next, I'll show you the code.

Creating SAX events

I've defined how you're going to map DOM nodes to the various SAX event types, so now it's time to take a look at the code. The steps you'll go through are:

1. Create a DOM parser.
2. Parse the file to create a DOM tree.
3. Walk through the DOM tree, converting the DOM nodes to the appropriate SAX events.
4. As the SAX events are fired, use the SAX-printing routines you developed earlier to print the SAX events. As the SAX events are fired, use the SAX-printing routines from *the first tutorial in this series*. See the source code in DomToSax.java on page 58 for the details.

The code for steps 1 and 2 is:

```
try
{
  dbf = DocumentBuilderFactory.newInstance();
  db = dbf.newDocumentBuilder();
  doc = db.parse(uri);
  if (doc != null)
    generateSAXEvents(doc, this);
}
```

Next, I'll go through the three node types handled in the `DomToSax` class.

Creating SAX events, continued

First, take a look at the `DOCUMENT_NODE` handler:

```
case Node.DOCUMENT_NODE:
{
  dh.startDocument();
  generateSAXEvents(((Document)node).getDocumentElement(), dh);
  dh.endDocument();
  break;
}
```

Fire the `startDocument` event, call the routine recursively to process the document element, then fire the `endDocument` event.

An `ELEMENT_NODE` is handled similarly to a `DOCUMENT_NODE`, with the exception that you need to process the attributes of the DOM element before you can fire the `startElement` event. `startElement` requires that you pass an object that implements the `Attributes` interface, along with the element name. Here's the code:

```
case Node.ELEMENT_NODE:
{
  AttributesImpl saxAttrs = new AttributesImpl();
  if (node.hasAttributes())
  {
    NamedNodeMap attrs = node.getAttributes();
    for (int i = 0; i < attrs.getLength(); i++)
      saxAttrs.addAttribute(null, null,
                            attrs.item(i).getNodeName(),
                            null, attrs.item(i).getNodeValue());
  }

  dh.startElement(null, null, node.getNodeName(), saxAttrs);

  if (node.hasChildNodes())
  {
    NodeList children = node.getChildNodes();
    for (int i = 0; i < children.getLength(); i++)
      generateSAXEvents(children.item(i), dh);
  }

  dh.endElement(null, null, node.getNodeName());
  break;
}
```

Similar to the way you handled the document node, you now fire `startElement`, invoke the routine recursively, then fire the `endElement` event.

The final (and simplest) case handles a `TEXT_NODE`. The `getNodeValue` method of a text node returns a Java `String`; convert that to an array of characters (`char`), then fire the `characters` event:

```
case Node.TEXT_NODE:
{
  char[] chars = node.getNodeValue().toCharArray();
  dh.characters(chars, 0, chars.length);
  break;
}
```

Notice that you don't have to handle ignorable whitespace differently because DOM doesn't distinguish between whitespace and other text nodes. If you wanted to process them separately, you have more work to do.

Running this code with the command `java DomToSax sonnet.xml` returns the same results you'd expect, based on the earlier examples. The complete source code is in DomToSax.java on page 58.

Section 4. Manipulating tree structures

Manipulating a DOM tree

The Document Object Model provides a number of methods for adding, moving, and deleting nodes in a DOM tree. To illustrate how this works, I'll show you an application that sorts the 14 `<line>` elements. In sorting these elements, you'll need to move nodes from one place to another in the DOM tree.

Because a sonnet has only 14 lines, you should use a bubble sort to put them in order. While this code works perfectly for the sample document, I'll show you some shortcuts to take along the way. (As an exercise, feel free to make this code more robust.)

For starters, you need to get all of the `<line>` elements in the DOM tree. Fortunately, the `Document` and `Element` interfaces contain the `getElementsByTagName` method. Given a tag name, this method returns a `NodeList` with all the elements with that tag name. The body of the code is:

```
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
dbf.setIgnoringElementContentWhitespace(true);
DocumentBuilder db = dbf.newDocumentBuilder();

doc = db.parse(uri);
if (doc != null)
{
  NodeList theLines = doc.getDocumentElement().
                      getElementsByTagName("line");
  sortLines(theLines);
  DomTreePrinter.printNode(doc);
}
```

The first few lines are the standard DOM parsing code you've used before. Next, `getElementsByTagName` is used to get a `NodeList` of the `<line>` elements; pass that to the `sortLines` method. After the lines of the sonnet have been sorted, you use the `DomTreePrinter` class to print out the updated DOM tree.

Notice that you don't pass the DOM tree to the `sortLines` method. As long as your `Document` object hasn't been garbage-collected, you can start with any of the nodes in the `NodeList` and find that node's parent, that node's siblings, that node's children, and so forth. That means you can start with a node, look at its next sibling (the next `<line>` element), and compare the text of those elements. If you need to swap the two, you can use DOM functions to tell the parent of those nodes to move one node in front of another.

...or is the DOM tree manipulating you?

Now things get more difficult. Conceptually, the bubble sort isn't complicated; the problem is getting the text of a given node. You might think that the `getNodeValue` method would do what you want, but that's not the case. According to the DOM standard, the value of an `Element` node is null. To get the text of a given `<line>`, you need to get all of its `Text` node children. When you call `getNodeValue` with a `Text` node, what you get back is the text you're looking for.

To make the code more readable, use the `getTextFromLine` method to extract the text you need:

```
public String getTextFromLine(Node lineElement)
{
  StringBuffer returnString = new StringBuffer();

  if (lineElement.getNodeName().equals("line"))
  {
    NodeList kids = lineElement.getChildNodes();
    if (kids != null)
      if (kids.item(0).getNodeType() == Node.TEXT_NODE)
        returnString.append(kids.item(0).getNodeValue());
  }
  else
    returnString.setLength(0);

  return new String(returnString);
}
```

Actually, this code is cheating, because you only look at the first child node. That works for the sample document, but you would have to do more work for more sophisticated documents. (It might be legal to have a <b> element inside a `<line>`, for example.) This code uses the `getNodeName` method to make sure this is the right kind of element, then it gets that node's children, then it makes sure the first child of the node is a text node. Assuming all of those things are true, the method returns the text of the element.

Sorting nodes

Your final task is to actually sort the nodes. Use the `String.compareTo` function to find out which of two lines appears first in sorted order. If you need to swap the two nodes, use the DOM `insertBefore` method. This method inserts one node in front of another; best of all, if the node already exists in the DOM tree, it is moved to the new location. Here's the code:

```
public void sortLines(NodeList theLines)
{
  if (theLines != null)
  {
    int len = theLines.getLength();
    for (int i = 0; i < len; i++)
      for (int j = 0; j < (len - 1 - i); j++)
        if (getTextFromLine(theLines.item(j)).
            compareTo(getTextFromLine(theLines.item(j+1)))
            > 0)
          theLines.item(j).getParentNode().
            insertBefore(theLines.item(j+1),
            theLines.item(j));
  }
}
```

Although this code looks confusing, it's really not that bad. The `for` loops handle the bubble sort. The `if` statement compares the text of this line and the text of the next line; if you need to swap them, you can use `insertBefore`. Notice that `getParentNode` is used to get the parent of a node. Once you have the parent, you tell the parent to move the next line before the current node.

You can see the complete source code in DomSorter.java on page 63.

Working with attributes

Manipulating the attributes of nodes in a DOM tree is very similar to the other functions I've covered here. Several DOM methods work with attributes:

`Node.getAttributes()`
: If this `Node` is an `Element`, this method returns a `NamedNodeMap` of the element's attributes. If the `Node` is anything else, the method returns `null`.

`Element.getAttribute(String name)`
: Returns the string value of the named attribute.

`Element.getAttributeNode(String name)`

`Element.getAttributeNodeNS(String namespaceURI, String name)`
: These methods return an object that has the specified name (and namespace, if specified) and implement the `Attr` interface.

`Element.hasAttribute(String name)`

`Element.hasAttributeNS(String name)`
: These methods return true if the element has an attribute with the specified name (and namespace, if specified).

`Element.removeAttribute(String name)`

`Element.removeAttributeNS(String namespaceURI, String name)`
: These methods remove the attribute with the given name (and namespace, if specified).

`Element.removeAttributeNode(Attr oldAttr)`
: This one is confusing: The argument to the method is an object that implements the `Attr` interface. You want the `Element` to remove the attribute that matches this object. The method returns the object (the `Attr`) that was removed. To complicate things further, if the attribute has a default value (in other words, if the attribute has a value whether the XML document specifies it or not), the deleted attribute is replaced with a new `Attr` that has the default value.

`Element.setAttribute(String name, String value)`

`Element.setAttributeNS(String namespaceURI, String name, String value)`
: These methods add a new attribute with the specified name and value (and namespace, if specified).

`Element.setAttributeNode(Attr newAttribute)`

`Element.setAttributeNodeNS(Attr newAttribute)`
: These methods add the `Attr` object passed in as an argument. If the new attribute replaces an existing attribute with the same name (and namespace, if specified), these methods return the replaced object; otherwise, these methods return `null`.

The code sample DomAttributes.java on page 65 parses an XML file, then it uses the `Element.setAttribute` method to add an attribute to every `Element` node in the DOM tree. Its last task is to use the `DomTreePrinter` class to print out the modified DOM tree.

Manipulating a JDOM tree

Now that you've manipulated a DOM tree, I'll show you how to do the same thing with JDOM. As you'll see shortly, JDOM offers several convenient methods that simplify the task, particularly when compared with the DOM version.

The code begins by scanning the command line, and then calling the `parseAndSortLines()` method to parse and process the sonnet:

```
public static void main(String[] argv)
{
  if (argv.length == 0 ||
      (argv.length == 1 &&argv[0].equals("-help")))
  // print message and exit

  JdomSorter js = new JdomSorter();
  js.parseAndSortLines(argv[0]);
}

public void parseAndSortLines(String uri)
{
  try
  {
    SAXBuilder sb = new SAXBuilder();
    Document doc = sb.build(new File(uri));
    sortLines(doc);
```

The `sortLines()` method is where most of the actual work takes place. Pass your entire document to this method; its first task is to get all of the `<line>` elements. When you worked with the DOM, you used `getElementsByTagName()` to find all of the elements you wanted. Once you had those elements, you could then use the parent element (accessible through `getParentNode()`) to move the `<line>`s around as needed.

More JDOM manipulations

The JDOM equivalents of `getElementsByTagName()` are `getChild()` and `getChildren()`. The main conceptual difference between JDOM and DOM is that *JDOM only works with the children of an element, not its descendants*. In other words, you can't start at the root of the document and ask for all of the `<line>` elements in the document; you have to find the `<lines>` element and ask for all of its `<line>` children.

Because you know the structure of your XML document, you can get the `<lines>` element pretty quickly. Here's how to do it:

```
public void sortLines(Document sonnet)
{
  Element linesElement = sonnet.getRootElement().
    getChild("lines");
  List lines = linesElement.getChildren("line");
```

Given a `Document` object, ask for the root element (`<sonnet>`), then ask for its child named `<lines>`. From there, ask for all of the `<line>` children of that element. If you didn't know the exact structure of the document, you would have to use the `getChildren()` method to get all of an element's children, select the appropriate child element, then use `getChildren()` until you found the element you wanted.

Notice that JDOM returns a `List`, part of the Java Collections API. One of the many great things about the `List` interface is that any changes you make to the `List` are reflected in the underlying data structure.

In JDOM, the bubble sort routine is:

```
for (int i = 0; i < 14; i++)
  for (int j = 0; j < (14 - 1 - i); j++)
    if (((Element)lines.get(j)).getText().
        compareTo(((Element)lines.get(j+1)).getText())
        > 0)
      lines.add(j, lines.remove(j+1));
```

A couple of things are worth mentioning here. First of all, notice that JDOM provides you with a `getText()` method to get the text of a given element. That means you don't have to write a utility routine to get the text children of an element as you did with DOM. Next, notice that you have to cast items in the `List` to be `Elements`. When you need to swap two adjacent lines, you use the `add()` and `remove()` methods together. Remove the line at position `j+1`, then insert it at position `j`.

One final point: Because the `List` interface lets you modify the underlying data structure directly, your `sortLines()` method doesn't return anything. Changes you make to the `<lines>` element are reflected in the `Document` object itself.

Outputting the results

Now that you've sorted the lines of the sonnet, your final task is to write it out. You'll use an `XMLOutputter` along with several of its features that I haven't mentioned before:

```
sortLines(doc);
XMLOutputter xo = new XMLOutputter();
xo.setTrimAllWhite(true);
xo.setIndent("  ");
xo.setNewlines(true);
xo.output(doc, System.out);
```

Tell the outputter to remove all extraneous whitespace with the `setTrimAllWhite()` method, then use `setIndent()` and `setNewlines()` to set up pretty-printing of your XML document. The results look like this:

```
C:\adv-xml-prog>java JdomSorter sonnet.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE sonnet SYSTEM "sonnet.dtd">
<sonnet type="Shakespearean">
  <author>
. . .
  <lines>
    <line>And in some perfumes is there more delight</line>
    <line>And yet, by Heaven, I think my love as rare</line>
    <line>As any she belied with false compare.</line>
    <line>But no such roses see I in her cheeks.</line>
    <line>Coral is far more red than her lips red.</line>
. . .
```

The complete source code is in JdomSorter.java on page 67.

A final word about tree manipulation

Although it's beyond the scope of this tutorial, be aware that the simplest way to sort the lines of a sonnet is with an XSLT stylesheet. XSLT provides the wonderful `<xsl:sort>` element that does what you've done the hard way in the examples here.

Here's the bulk of the stylesheet:

```
<xsl:template match="lines">
  <lines>
    <xsl:for-each select="line">
      <xsl:sort/>
      <xsl:copy>
        <xsl:apply-templates select="*|@*|text()"/>
      </xsl:copy>
    </xsl:for-each>
  </lines>
</xsl:template>
```

Again, I won't explain this stylesheet in any detail, but the template above does all the work of sorting for you. When you find a `<lines>` element, you output a new `<lines>` element, sort all the `<line>` elements inside it, then copy them to the result document.

You can see the complete stylesheet in sonnetSorter.xsl on page 68.

Section 5. Advanced DOM features

Serializing a DOM tree

So far, you've used the `printDomTree` in the DOM examples. Now I'll show you two other ways to print out (or **serialize**) a DOM tree. The first approach is to simply move `printDomTree` to a separate class so you don't have to include this method in the Java source code of every DOM application you create.

The second approach is to use the `DOMSerializer` class. This class is part of DOM Level 3, but at this point it hasn't been added to the factory classes of JAXP. (`DOMSerializer` may or may not be added to JAXP in the future, by the way.)

For the first approach, simply create a package named `com.ibm.dw.xmlprogjava` and a class within that package named `DomTreePrinter`. That class contains a single `public static` named `printNode`. Here's how the code looks:

```
package com.ibm.dw.xmlprogjava;
. . .

public class DomTreePrinter
{
  /** Prints the specified node, recursively. */
  public static void printNode(Node node)
  {
    int type = node.getNodeType();
    switch (type)
    {
      // print the document element
      case Node.DOCUMENT_NODE:
        {
          System.out.println("<?xml version=\"1.0\" ?>");
          printNode(((Document)node).getDocumentElement());
          break;
        }
```

The only change was to rename the method `printNode` instead of `printDomTree`. For the complete source code, see DomThree.java on page 69 and DomTreePrinter.java on page 70.

You'll see how to create a `DOMSerializer` next.

Using an **`LSSerializer`**

The second way to write the DOM tree as an XML document is to use the `LSSerializer` interface. This is part of the `org.w3c.dom.ls` package, which is part of the DOM Level 3 Load and Save specification. Support for this as-yet-unfinished standard in Xerces is likely to change, but the code that works as of July 2004 is:

```
import org.apache.xerces.dom.DOMOutputImpl;
import org.w3c.dom.Document;
import org.w3c.dom.bootstrap.DOMImplementationRegistry;
import org.w3c.dom.ls.DOMImplementationLS;
import org.w3c.dom.ls.LSOutput;
import org.w3c.dom.ls.LSParser;
import org.w3c.dom.ls.LSSerializer;

. . .

public class DomFour
{
  public void parseAndPrint(String uri)
  {
    Document doc = null;

    try
    {
      System.setProperty(DOMImplementationRegistry.PROPERTY,
        "org.apache.xerces.dom.DOMXSImplementationSourceImpl");
      DOMImplementationRegistry direg =
        DOMImplementationRegistry.newInstance();
      DOMImplementationLS dils =
        (DOMImplementationLS) direg.getDOMImplementation("LS");
      LSParser lsp = dils.createLSParser
        (DOMImplementationLS.MODE_SYNCHRONOUS, null);

      doc = lsp.parseURI(uri);

      LSSerializer domWriter = dils.createLSSerializer();
      LSOutput lso = new DOMOutputImpl();
      lso.setByteStream(System.out);
      domWriter.write(doc, lso);
    }
    catch (Exception e)
    {
      System.err.println("Sorry, an error occurred: " + e);
    }
  }
```

This code uses several classes specific to the Xerces implementation of the DOM Level 3 interfaces. As of July 2004, this code only compiles and runs with a special build of the Xerces parser. As the DOM Level 3 standard progresses and the Xerces implementation becomes more mature, this code will almost certainly change. In the code above, the classes `DOMXSImplementationSourceImpl` and `DOMOutputImpl` are specific to the Xerces parser. For the complete (and refreshingly short) source code, see DomFour.java on page 72.

Other DOM functions

Although you don't need them for the sonnet-sorting example, several other DOM methods are useful when manipulating DOM trees. The most commonly-used methods are:

`appendChild(Node newChild)`
: Appends the node `newChild` as the last child of the parent node.

`removeChild(Node oldChild)`
: Removes the node `oldChild` from the parent node.

`replaceChild(Node newChild, Node oldChild)`
: Replaces `oldChild` with `newChild`. (**Note:** Both `newChild` and `oldChild` must have been created by the same `DocumentBuilder`.)

Using a different DOM parser

Throughout the samples in this tutorial, you've used the Xerces parser. Technically, though, almost all of the samples could use any other JAXP-compliant parser without any changes. (The one exception is the `DOMSerializer` sample, which uses DOM Level 3 classes not yet standardized by JAXP.) One goal of JAXP is to allow you to change parsers without making any changes to your source code. JAXP accomplishes this by loading a particular parser at runtime.

At runtime, JAXP determines the name of the class that implements the `DocumentBuilderFactory` interface. JAXP looks for the class name in four places (in this order):

1. The value of the `javax.xml.parsers.DocumentBuilderFactory` property
2. The value of the `javax.xml.parsers.DocumentBuilderFactory` property in the `jre/lib/jaxp.properties` file
3. In examining all of the JAR files in the CLASSPATH, the first value found in a `META-INF/services/javax.xml.parsers.DocumentBuilderFactory`

file

4. The default `DocumentBuilderFactory` for the Java platform (in JDK 1.4, the default parser is `org.apache.crimson.jaxp.DocumentBuilderFactoryImpl`)

You can specify the `javax.xml.parsers.DocumentBuilderFactory` property in two ways. The first is by using the `-D` parameter from the command line:

```
java -Djavax.xml.parsers.DocumentBuilderFactory=[DBF class] . . .
```

The second is by adding the following code to your Java source before you create the `DocumentBuilderFactory`:

```
System.setProperty("javax.xml.parsers.DocumentBuilderFactory",
         "org.apache.xerces.jaxp.DocumentBuilderFactoryImpl");
DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
```

Section 6. Advanced SAX features

Killing a SAX parser

When I first showed you a SAX parser, I noted that one of the advantages of using SAX is that you get SAX events as the parser reads the XML file. When using a DOM parser, you can only see a given element after the entire document has been processed; with SAX, you see that element as soon as the parser does.

You can take this a step further by stopping the SAX parser when you find what you're looking for. I'll show you how to create a fatal error that kills the parser, which means the SAX parser won't even read the entire XML file.

In your parser-killing application, you'll look for the fourth `<line>` element in your sonnet. When you find it, you'll create a `SAXParseException`, then call the `fatalError` method. I'll show you that code next.

The search is on

Because SAX events are stateless, your code has to keep track of all the events that you've seen. If you want to find the fourth `<line>` element, you know it starts with the fourth `startElement` event for which the `rawName` is `line`. All of the `characters` and `ignorableWhitespace` events that your parser gets while it's in the fourth `<line>` element are part of the element's text, and when it finds the fourth `endElement` with a `rawName` of `line`, you know you have everything you're looking for. At that point, create your exception and call `fatalError`.

First of all, use these three variables to manage state information:

```
int lineCount = 0;
boolean inFourthLine = false;
StringBuffer fourthLine = new StringBuffer();
```

Use `lineCount` to count how many `<line>` elements you've seen so far, set the flag `inFourthLine` when you see the `startElement` event for the fourth `<line>` element, and use `fourthLine` to store the text of the `<line>` element itself.

The `startElement` method looks like :

```
public void startElement(String namespaceURI, String localName,
                         String rawName, Attributes attrs)
{
  if (rawName.equals("line") && ++lineCount == 4)
      inFourthLine = true;
}
```

I'll show you the other event handlers next.

Handling other events

Now take a brief look at the other event handlers that the parser killer uses. First are the `characters` and `ignorableWhitespace` handlers -- which merely check the `inFourthLine` flag and store the text if the parser is in the fourth `<line>`:

```
public void characters(char ch[], int start, int length)
{
  if (inFourthLine)
    fourthLine.append(new String(ch, start, length));
}

public void ignorableWhitespace(char ch[], int start, int length)
{
  if (inFourthLine)
    characters(ch, start, length);
}
```

Now for the `endElement` handler. When your parser reaches the end of the fourth `<line>`, print the text of the line, then create the exception to kill the parser:

```
public void endElement(String namespaceURI, String localName,
                       String rawName)
  throws SAXException
{
  if (rawName.equals("line") && inFourthLine)
  {
    System.out.println("\nThe text of the fourth line is: \n");
    System.out.println("\t" + fourthLine);

    SAXParseException spe =
      new SAXParseException("Found the fourth <line>, " +
                            "so we killed the parser!",
                            new LocatorImpl());
    fatalError(spe);
  }
}
```

If you run `SaxKiller`, you should see something like this:

```
C:\xml-prog-java>java SaxKiller sonnet.xml

The text of the fourth line is:

        If hairs be wires, black wires grow on her head.
```

The exception isn't written out because you `catch` it in the `main` method:

```
SaxKiller s1 = new SaxKiller();
try
{
  s1.parseURI(argv[0]);
}
// We're expecting an exception, so we ignore
// anything that happens...
catch (Exception e) { }
```

See SaxKiller.java on page 74 for the complete listing.

Using a different SAX parser

As you saw earlier, JAXP allows you to specify a different DOM parser at runtime; it also allows you to specify a `SAXParserFactory` implementation at runtime. At runtime, JAXP determines the name of the class that implements the `SAXParserFactory` interface. In order, JAXP looks for the class name in these four places:

1. The value of the `javax.xml.parsers.SAXParserFactory` property
2. The value of the `javax.xml.parsers.SAXParserFactory` property in the `jre/lib/jaxp.properties` file
3. Looking through all of the JAR files in the CLASSPATH, the first value found in a `META-INF/services/javax.xml.parsers.SAXParserFactory` file
4. The default `SAXParserFactory` for the Java platform (in JDK 1.4, the default parser is `org.apache.crimson.jaxp.SAXParserFactory`)

You can specify the `javax.xml.parsers.SAXParserFactory` property in two ways. The first is by using the `-D` parameter from the command line:

```
java -Djavax.xml.parsers.SAXParserFactory=[SPF class name] . . .
```

The second is by adding the following code to your Java source before you create the `SAXParserFactory`:

```
System.setProperty("javax.xml.parsers.SAXParserFactory",
         "org.apache.xerces.jaxp.SAXParserFactoryImpl");
SAXParserFactory spf = SAXParserFactory.newInstance();
```

This code tells JAXP to use the Xerces parser from the Apache Software Foundation.

Section 7. Summary and references

Summary

In this final installment of the XML programming in Java technology tutorial series, I covered the more esoteric details of the DOM, SAX, and JDOM APIs. At this point, you should know just about everything a parser can do. As you build your own XML applications, I hope these methods and techniques make your life easier.

Resources

For the complete examples, download x-java3_codefiles.zip.

Review the previous tutorials in this series:

° "*XML programming in Java technology, Part 1*" covers the basics of manipulating XML documents using Java technology, and looks at the common APIs for XML (*developerWorks*, January 2004).

° "*XML programming in Java technology, Part 2*" shows you how to do tasks such as generate XML data structures, validate XML documents, work with namespaces, and interface XML parsers with non-XML data sources (*developerWorks*, July 2004).

Visit the *DOM Technical Reports page* (http://www.w3.org/DOM/DOMTR) at the W3C for links to all things DOM-related. To view the individual specs, visit:

° *Document Object Model Level 1* (http://www.w3.org/TR/2000/WD-DOM-Level-1-20000929)

° *DOM Level 2 Core*

° *DOM Level 3 Core*

Read about *SAX Version 2.0* (http://sax.sourceforge.net/) .

Learn all about JDOM at the *JDOM project's home page* (http://www.jdom.org/) .

If you want a refresher on the fundamentals of XML itself, read Doug's popular "*Introduction to XML*" tutorial (*developerWorks*, August 2002).

Browse a wide range of excellent titles at the *developerWorks* *Developer*

Bookstore (http://devworks.krcinfo.com/) , including *these books on XML and Java technology*.

Find more resources related to the technologies discussed here on the *developerWorks XML* (http://www.ibm.com/developerworks/xml/) and *Java technology* (http://www.ibm.com/developerworks/java/) zones.

Finally, find out how you can become an *IBM Certified Developer in XML and related technologies* (http://www.ibm.com/certify/certs/adcdxmlrt.shtml) .

Feedback

Please send us your feedback on this tutorial. We look forward to hearing from you! Additionally, you are welcome to contact the author, Doug Tidwell at *dtidwell@us.ibm.com*.

Colophon

This tutorial was written entirely in XML, using the developerWorks Toot-O-Matic tutorial generator. The open source Toot-O-Matic tool is an XSLT stylesheet and several XSLT extension functions that convert an XML file into a number of HTML pages, a zip file, JPEG heading graphics, and two PDF files. Our ability to generate multiple text and binary formats from a single source file illustrates the power and flexibility of XML. (It also saves our production team a great deal of time and effort.)

For more information about the Toot-O-Matic, visit www-106.ibm.com/developerworks/xml/library/x-toot/ .

Appendix: Source code

sonnet.xml

```
<?xml version="1.0"?>
<!DOCTYPE sonnet SYSTEM "sonnet.dtd">
<sonnet type="Shakespearean">
  <author>
    <lastName>Shakespeare</lastName>
    <firstName>William</firstName>
    <nationality>British</nationality>
    <yearOfBirth>1564</yearOfBirth>
    <yearOfDeath>1616</yearOfDeath>
  </author>
  <title>Sonnet 130</title>
  <lines>
    <line>My mistress' eyes are nothing like the sun,</line>
    <line>Coral is far more red than her lips red.</line>
    <line>If snow be white, why then her breasts are dun,</line>
    <line>If hairs be wires, black wires grow on her head.</line>
    <line>I have seen roses damasked, red and white,</line>
    <line>But no such roses see I in her cheeks.</line>
    <line>And in some perfumes is there more delight</line>
    <line>Than in the breath that from my mistress reeks.</line>
    <line>I love to hear her speak, yet well I know</line>
    <line>That music hath a far more pleasing sound.</line>
    <line>I grant I never saw a goddess go,</line>
    <line>My mistress when she walks, treads on the ground.</line>
    <line>And yet, by Heaven, I think my love as rare</line>
    <line>As any she belied with false compare.</line>
  </lines>
</sonnet>
```

sonnet.dtd

```
<!-- sonnet.dtd                                        -->

<!-- sonnet is the root of the document                -->
<!ELEMENT sonnet  (author,title?,lines)>
<!-- the default sonnet type is "Shakespearean"        -->
<!ATTLIST sonnet  type (Shakespearean | Petrarchan)
                       "Shakespearean">

<!-- author contains information about the author      -->
<!ELEMENT author  (lastName,firstName,nationality,
                   yearOfBirth?,yearOfDeath?)>

<!-- lastName, firstName, nationality, yearOfBirth,
     and yearOfDeath are all elements inside author. -->

<!ELEMENT lastName (#PCDATA)>
<!ELEMENT firstName (#PCDATA)>
<!ELEMENT nationality (#PCDATA)>
<!ELEMENT yearOfBirth (#PCDATA)>
<!ELEMENT yearOfDeath (#PCDATA)>
```

```
<!-- The title of the sonnet                           -->
<!ELEMENT title (#PCDATA)>

<!-- The lines element contains the 14 lines of the
     sonnet.                                           -->
<!ELEMENT lines (line,line,line,line,
                 line,line,line,line,
                 line,line,line,line,
                 line,line)>

<!ELEMENT line (#PCDATA)>
```

ParseString.java

```
/*
 * (C) Copyright IBM Corp. 2004.  All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantibility and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

import com.ibm.dw.xmlprogjava.DomTreePrinter;
import java.io.StringReader;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.xml.sax.InputSource;

/**
 * A sample DOM application.  This sample creates a DOM tree from
 * a String.
 */

public class ParseString
{
  public void parseAndPrint(InputSource xmlSource)
  {
    Document doc = null;

    try
    {
      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
      DocumentBuilder db = dbf.newDocumentBuilder();
      doc = db.parse(xmlSource);
      if (doc != null)
        DomTreePrinter.printNode(doc);
    }
    catch (Exception e)
    {
      System.err.println("Sorry, an error occurred: " + e);
    }
  }

  /** Main program entry point. */
  public static void main(String argv[])
  {
    if (argv.length == 1 && argv[0].equals("-help"))
    {
      System.out.println("\nUsage:  java ParseString");
      System.out.println("\nParses an XML document that's " +
                         "contained in a string.");
      System.exit(1);
    }

    ParseString ps = new ParseString();
    String markup = new String("<html><body><h1>" +
                               "This XML document was a <b>string!</b>" +
                               "</h1></body></html>");
    InputSource iSrc = new InputSource(new StringReader(markup));
    ps.parseAndPrint(iSrc);
  }
}
```

DomBuilder.java

```
/*
 * (C) Copyright IBM Corp. 2004.  All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantibility and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

/**
 * A sample DOM application.  This program illustrates how to
 * create a DOM tree without an XML source.
 */
```

```
public class DomBuilder
{
  /** Prints the specified node, recursively. */
  public void printDomTree(Node node)
  {
    int type = node.getNodeType();
    switch (type)
    {
      // print the document element
      case Node.DOCUMENT_NODE:
        {
          System.out.println("<?xml version=\"1.0\" ?>");
          printDomTree(((Document)node).getDocumentElement());
          break;
        }

         // print element and any attributes
      case Node.ELEMENT_NODE:
        {
          System.out.print("<");
          System.out.print(node.getNodeName());

          if (node.hasAttributes())
          {
            NamedNodeMap attrs = node.getAttributes();
            for (int i = 0; i < attrs.getLength(); i++)
              printDomTree(attrs.item(i));
          }
          System.out.print(">");

          if (node.hasChildNodes())
          {
            NodeList children = node.getChildNodes();
            for (int i = 0; i < children.getLength(); i++)
              printDomTree(children.item(i));
          }

          break;
        }

        // Print attribute nodes
      case Node.ATTRIBUTE_NODE:
        {
          System.out.print(" " + node.getNodeName() + "=\"");
          if (node.hasChildNodes())
          {
            NodeList children = node.getChildNodes();
            for (int i = 0; i < children.getLength(); i++)
              printDomTree(children.item(i));
          }
          System.out.print("\"");
          break;
        }

        // handle entity reference nodes
      case Node.ENTITY_REFERENCE_NODE:
        {
          System.out.print("&");
          System.out.print(node.getNodeName());
          System.out.print(";");
          break;
        }

        // print cdata sections
      case Node.CDATA_SECTION_NODE:
        {
          System.out.print("<![CDATA[");
          System.out.print(node.getNodeValue());
          System.out.print("]]>");
          break;
        }

        // print text
      case Node.TEXT_NODE:
        {
          System.out.print(node.getNodeValue());
          break;
        }

        // print comment
      case Node.COMMENT_NODE:
        {
          System.out.print("<!--");
          System.out.print(node.getNodeValue());
          System.out.print("-->");
          break;
        }

        // print processing instruction
      case Node.PROCESSING_INSTRUCTION_NODE:
        {
          System.out.print("<?");
          System.out.print(node.getNodeName());
          String data = node.getNodeValue();
          {
            System.out.print(" ");
            System.out.print(data);
          }
          System.out.print("?>");
          break;
        }
    }

    if (type == Node.ELEMENT_NODE)
    {
      System.out.print("</");
      System.out.print(node.getNodeName());
      System.out.print('>');
    }
  } // printDomTree(Node)

  /** Main program entry point. */
  public static void main(String argv[])
  {
    if (argv.length == 1 && argv[0].equals("-help"))
    {
      System.out.println("\nUsage:  java DomBuilder");
      System.out.println("\nBuilds a DOM tree with DOM method calls, " +
                         "then prints it.");
      System.exit(1);
    }

    try
    {
      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
      DocumentBuilder docBuilder = dbf.newDocumentBuilder();
      Document doc = docBuilder.getDOMImplementation().
          createDocument("", "sonnet", null);

      Element root = doc.getDocumentElement();
      root.setAttribute("type", "Shakespearean");

      Element author = doc.createElement("author");

      Element lastName = doc.createElement("last-name");
      lastName.appendChild(doc.createTextNode("Shakespeare"));
      author.appendChild(lastName);

      Element firstName = doc.createElement("first-name");
      firstName.appendChild(doc.createTextNode("William"));
      author.appendChild(firstName);

      Element nationality = doc.createElement("nationality");
      nationality.appendChild(doc.createTextNode("British"));
      author.appendChild(nationality);

      Element yearOfBirth = doc.createElement("year-of-birth");
      yearOfBirth.appendChild(doc.createTextNode("1564"));
      author.appendChild(yearOfBirth);

      Element yearOfDeath = doc.createElement("year-of-death");
      yearOfDeath.appendChild(doc.createTextNode("1616"));
      author.appendChild(yearOfDeath);

      root.appendChild(author);

      Element title = doc.createElement("title");
      title.appendChild(doc.createTextNode("Sonnet 130"));
      root.appendChild(title);

      Element lines = doc.createElement("lines");

      Element line01 = doc.createElement("line");
      line01.appendChild(doc.createTextNode
             ("My mistress' eyes are nothing like the sun,"));
      lines.appendChild(line01);

      Element line02 = doc.createElement("line");
      line02.appendChild(doc.createTextNode
             ("Coral is far more red than her lips red."));
      lines.appendChild(line02);

      Element line03 = doc.createElement("line");
      line03.appendChild(doc.createTextNode
             ("If snow be white, why then her breasts are dun,"));
      lines.appendChild(line03);

      Element line04 = doc.createElement("line");
      line04.appendChild(doc.createTextNode
             ("If hairs be wires, black wires grow on her head."));
      lines.appendChild(line04);

      Element line05 = doc.createElement("line");
      line05.appendChild(doc.createTextNode
             ("I have seen roses damasked, red and white,"));
      lines.appendChild(line05);

      Element line06 = doc.createElement("line");
      line06.appendChild(doc.createTextNode
             ("But no such roses see I in her cheeks."));
      lines.appendChild(line06);

      Element line07 = doc.createElement("line");
      line07.appendChild(doc.createTextNode
             ("And in some perfumes is there more delight"));
      lines.appendChild(line07);

      Element line08 = doc.createElement("line");
      line08.appendChild(doc.createTextNode
             ("Than in the breath that from my mistress reeks."));
      lines.appendChild(line08);

      Element line09 = doc.createElement("line");
      line09.appendChild(doc.createTextNode
             ("I love to hear her speak, yet well I know"));
      lines.appendChild(line09);

      Element line10 = doc.createElement("line");
      line10.appendChild(doc.createTextNode
             ("That music hath a far more pleasing sound."));
      lines.appendChild(line10);

      Element line11 = doc.createElement("line");
      line11.appendChild(doc.createTextNode
             ("I grant I never saw a goddess go,"));
      lines.appendChild(line11);

      Element line12 = doc.createElement("line");
      line12.appendChild(doc.createTextNode
             ("My mistress when she walks, treads on the ground."));
      lines.appendChild(line12);

      Element line13 = doc.createElement("line");
      line13.appendChild(doc.createTextNode
             ("And yet, by Heaven, I think my love as rare"));
      lines.appendChild(line13);

      Element line14 = doc.createElement("line");
      line14.appendChild(doc.createTextNode
             ("As any she belied with false compare."));
      lines.appendChild(line14);
```

```
      root.appendChild(lines);

      DomBuilder db = new DomBuilder();
      db.printDomTree(doc);
    }
    catch (Exception e)
    {
      System.err.println(e);
    }
  }
}
```

JdomBuilder.java

```
/*
 * (C) Copyright IBM Corp. 2004.  All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantibility and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

import java.io.IOException;
import java.util.Vector;
import org.jdom.DocType;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.output.XMLOutputter;

public class JdomBuilder
{
  public static void main(String[] argv)
  {
    if (argv.length == 1 && argv[0].equals("-help"))
    {
      System.out.println("\nUsage:  java JdomBuilder");
      System.out.println("\nUses the JDOM API to create a Document " +
                         "object without an XML source file.");
      System.exit(1);
    }

    JdomBuilder jb = new JdomBuilder();
    jb.buildDocument();
  }

  public void buildDocument()
  {
    Element root = new Element("sonnet");
    root.setAttribute("type", "Shakespearean");

    Vector author = new Vector();
    author.add(new Element("last-name").addContent("Shakespeare"));
    author.add(new Element("first-name").addContent("William"));
    author.add(new Element("nationality").addContent("British"));
    author.add(new Element("year-of-birth").addContent("1564"));
    author.add(new Element("year-of-death").addContent("1616"));
    root.addContent(new Element("author").setContent(author));

    root.addContent(new Element("title").addContent("Sonnet 130"));

    Vector lines = new Vector();

    lines.add(new Element("line").
      addContent("My mistress' eyes are nothing like the sun,"));
    lines.add(new Element("line").
      addContent("Coral is far more red than her lips red."));
    lines.add(new Element("line").
      addContent("If snow be white, why then her breasts are dun,"));
    lines.add(new Element("line").
      addContent("If hairs be wires, black wires grow on her head."));
    lines.add(new Element("line").
      addContent("I have seen roses damasked, red and white,"));
    lines.add(new Element("line").
      addContent("But no such roses see I in her cheeks."));
    lines.add(new Element("line").
      addContent("And in some perfumes is there more delight"));
    lines.add(new Element("line").
      addContent("Than in the breath that from my mistress reeks."));
    lines.add(new Element("line").
      addContent("I love to hear her speak, yet well I know"));
    lines.add(new Element("line").
      addContent("That music hath a far more pleasing sound."));
    lines.add(new Element("line").
      addContent("I grant I never saw a goddess go,"));
    lines.add(new Element("line").
      addContent("My mistress when she walks, treads on the ground."));
    lines.add(new Element("line").
      addContent("And yet, by Heaven, I think my love as rare"));
    lines.add(new Element("line").
      addContent("As any she belied with false compare."));
    root.addContent(new Element("lines").setContent(lines));

    Document doc = new Document(root, new DocType("sonnet", "sonnet.dtd"));

    try
    {
      XMLOutputter xo = new XMLOutputter("  ", true);
      xo.output(doc, System.out);
    }
    catch (IOException ioe)
    {
      System.err.println("IO Exception: " + ioe);
    }
  }
}
```

CsvToSax.java

```
/*
 * (C) Copyright IBM Corp. 2004.  All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantibility and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.StreamTokenizer;
import java.math.BigInteger;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.helpers.DefaultHandler;

public class CsvToSax
  extends DefaultHandler
{
  static String tagNames[] = {"employeeNumber",
                              "firstName",
                              "middleInitial",
                              "lastName",
                              "deptNo",
                              "extension",
                              "dateOfBirth"};

  public void scanCsvFile(String uri, DefaultHandler dh)
      throws java.io.FileNotFoundException,
             java.io.IOException,
             org.xml.sax.SAXException
  {
    BufferedReader br = new BufferedReader(new FileReader(uri));
    StreamTokenizer st = new StreamTokenizer(br);

    // We want to read the file one line at a time, so end-ofline matters
    st.eolIsSignificant(true);
    // The delimiter between fields is a comma, not a space
    st.whitespaceChars(',', ',');
    // All strings are in double quotes
    st.quoteChar('"');

    char [] lineBreak = new String("\n").toCharArray();
    char [] singleIndent = new String("  ").toCharArray();
    char [] doubleIndent = new String("    ").toCharArray();
```

```
dh.startDocument();
dh.startElement(null, null, "employees", null);
dh.ignorableWhitespace(lineBreak, 0, lineBreak.length);

// Get the first token, then check its type
st.nextToken();
while (st.ttype != StreamTokenizer.TT_EOF)
{
  // We're not at EOF, so start a row
  dh.ignorableWhitespace(singleIndent, 0, singleIndent.length);
  dh.startElement(null, null, "employee", null);
  dh.ignorableWhitespace(lineBreak, 0, lineBreak.length);

  int i = 0;
  while (st.ttype != StreamTokenizer.TT_EOL &&
         st.ttype != StreamTokenizer.TT_EOF)
  {
    // We use the BigInteger class here to write long numbers out
    // Without this, the date fields (which are written something
    // like (19991013) get converted to scientific notation....
    if (st.ttype == StreamTokenizer.TT_NUMBER)
    {
      char [] chars = BigInteger.valueOf((long)st.nval).
          toString().toCharArray();
      dh.ignorableWhitespace(doubleIndent, 0, doubleIndent.length);
      dh.startElement(null, null, tagNames[i], null);
      dh.characters(chars, 0, chars.length);
      dh.endElement(null, null, tagNames[i]);
      dh.ignorableWhitespace(lineBreak, 0, lineBreak.length);
    }
    else if (st.ttype != StreamTokenizer.TT_EOL &&
             st.ttype != StreamTokenizer.TT_EOF)
    {
      // For reasons that escape me, if the token is "+", it
      // is interpreted as NULL.
      if (st.sval != null)
      {
        char [] chars = st.sval.trim().toCharArray();
        dh.ignorableWhitespace(doubleIndent, 0, doubleIndent.length);
        dh.startElement(null, null, tagNames[i], null);
        dh.characters(chars, 0, chars.length);
        dh.endElement(null, null, tagNames[i]);
        dh.ignorableWhitespace(lineBreak, 0, lineBreak.length);
      }
    }
    st.nextToken();
    i++;
  }

  // We've hit either the end of the line or the end of the file,
  // so close the row.
  dh.ignorableWhitespace(singleIndent, 0, singleIndent.length);
  dh.endElement(null, null, "employee");
  st.nextToken();
  dh.ignorableWhitespace(lineBreak, 0, lineBreak.length);
}
```

```
    // Now we're at the end of the file, so close the XML document,
    dh.endElement(null, null, "employees");
    dh.ignorableWhitespace(lineBreak, 0, lineBreak.length);
    dh.endDocument();
  }

  /** Start document. */
  public void startDocument()
  {
    System.out.println("<?xml version=\"1.0\"?>");
  } // startDocument()

  /** Start element. */
  public void startElement(String namespaceURI, String localName,
                           String rawName, Attributes attrs)
  {
    System.out.print("<");
    System.out.print(rawName);
    if (attrs != null)
    {
      int len = attrs.getLength();
      for (int i = 0; i < len; i++)
      {
        System.out.print(" ");
        System.out.print(attrs.getQName(i));
        System.out.print("=\"");
        System.out.print(attrs.getValue(i));
        System.out.print("\"");
      }
    }
    System.out.print(">");
  } // startElement(String,AttributeList)

  /** Characters. */
  public void characters(char ch[], int start, int length)
  {
    System.out.print(new String(ch, start, length));
  } // characters(char[],int,int);

  /** Ignorable whitespace. */
  public void ignorableWhitespace(char ch[], int start, int length)
  {
    characters(ch, start, length);
  } // ignorableWhitespace(char[],int,int);

  /** End element. */
  public void endElement(String namespaceURI, String localName,
                         String rawName)
  {
    System.out.print("</");
    System.out.print(rawName);
    System.out.print(">");
  } // endElement(String)

  /** End document. */
  public void endDocument()
  {
    // No need to do anything.
  } // endDocument()

  /** Processing instruction. */
  public void processingInstruction(String target, String data)
  {
    System.out.print("<?");
    System.out.print(target);
    if (data != null && data.length() > 0)
    {
      System.out.print(' ');
      System.out.print(data);
    }
    System.out.print("?>");

  } // processingInstruction(String,String)

  //
  // ErrorHandler methods
  //

  /** Warning. */
  public void warning(SAXParseException ex)
  {
    System.err.println("[Warning] "+
                       getLocationString(ex)+": "+
                       ex.getMessage());
  }

  /** Error. */
  public void error(SAXParseException ex)
  {
    System.err.println("[Error] "+
                       getLocationString(ex)+": "+
                       ex.getMessage());
  }

  /** Fatal error. */
  public void fatalError(SAXParseException ex)
    throws SAXException
  {
    System.err.println("[Fatal Error] "+
                       getLocationString(ex)+": "+
                       ex.getMessage());
    throw ex;
  }

  /** Returns a string of the location. */
  private String getLocationString(SAXParseException ex)
  {
    StringBuffer str = new StringBuffer();

    String systemId = ex.getSystemId();
    if (systemId != null)
    {
      int index = systemId.lastIndexOf('/');
      if (index != -1)
        systemId = systemId.substring(index + 1);
      str.append(systemId);
    }
    str.append(':');
    str.append(ex.getLineNumber());
    str.append(':');
    str.append(ex.getColumnNumber());

    return str.toString();
  } // getLocationString(SAXParseException):String

  /** Main program entry point. */
  public static void main(String argv[])
  {
    if (argv.length == 0 ||
        (argv.length == 1 && argv[0].equals("-help")))
    {
      System.out.println("\nUsage:  java CsvToSax uri");
      System.out.println("   where uri is the URI of your ");
      System.out.println("     comma-separated values document.");
      System.out.println("   Sample:  java CsvToSax sonnet.xml");
      System.out.println("\nEchoes SAX events back to the console.");
      System.exit(1);
    }

    CsvToSax c2s = new CsvToSax();
    try
    {
      c2s.scanCsvFile(argv[0], c2s);
    }
    catch (FileNotFoundException fnfe)
    {
      System.err.println("Error - File " + argv[0] + " not found!");
    }
    catch (SAXException se)
    {
      System.err.println("SAX Exception: " + se);
    }
    catch (IOException ioe)
    {
      System.err.println("IO Exception: " + ioe);
    }
  } // main(String[])
}
```

test.csv

```
"000010","CHRISTINE","I","HAAS","A00","3978",19650101
"000020","MICHAEL","L","THOMPSON","B01","3476",19731010
"000030","SALLY","A","KWAN","C01","4738",19750405
"000050","JOHN","B","GEYER","E01","6789",19490817
"000060","IRVING","F","STERN","D11","6423",19730914
"000070","EVA","D","PULASKI","D21","7831",19800930
"000090","EILEEN","W","HENDERSON","E11","5498",19700815
"000100","THEODORE","Q","SPENSER","E21","0972",19800619
"000110","VINCENZO","G","LUCCHESSI","A00","3490",19580516
"000120","SEAN"," ","O'CONNELL","A00","2167",19631205
```

```
"000130","DOLORES","M","QUINTANA","C01","4578",19710728
"000140","HEATHER","A","NICHOLLS","C01","1793",19761215
"000150","BRUCE"," ","ADAMSON","D11","4510",19720212
"000160","ELIZABETH","R","PIANKA","D11","3782",19771011
"000170","MASATOSHI","J","YOSHIMURA","D11","2890",19780915
"000180","MARILYN","S","SCOUTTEN","D11","1682",19730707
"000190","JAMES","H","WALKER","D11","2986",19740726
"000200","DAVID"," ","BROWN","D11","4501",19660303
"000210","WILLIAM","T","JONES","D11","0942",19790411
"000220","JENNIFER","K","LUTZ","D11","0672",19680829
"000230","JAMES","J","JEFFERSON","D21","2094",19661121
"000240","SALVATORE","M","MARINO","D21","3780",19791205
"000250","DANIEL","S","SMITH","D21","0961",19691030
"000260","SYBIL","P","JOHNSON","D21","8953",19750911
"000270","MARIA","L","PEREZ","D21","9001",19800930
"000280","ETHEL","R","SCHNEIDER","E11","8997",19670324
"000290","JOHN","R","PARKER","E11","4502",19800530
"000300","PHILIP","X","SMITH","E11","2095",19720619
"000310","MAUDE","F","SETRIGHT","E11","3332",19640912
"000320","RAMLAL","V","MEHTA","E21","9990",19650707
"000330","WING"," ","LEE","E21","2103",19760223
"000340","JASON","R","GOUNOT","E21","5698",19470505
```

SaxToDom.java

```
/*
 * (C) Copyright IBM Corp. 2004.  All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantibility and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

import java.util.Stack;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.helpers.DefaultHandler;
```

```
/**
 * A SAX application that builds a DOM tree from the SAX events,
 * then traverses the tree backwards.
 */

public class SaxToDom
  extends DefaultHandler
{
  DocumentBuilderFactory dbf = null;
  DocumentBuilder docBuilder = null;
  Document root = null;
  boolean firstElementNotFoundYet = true;
  Stack elementStack = null;

  /** Prints the specified node, recursively. */
  public void printDOMTree(Node node)
  {
    int type = node.getNodeType();
    switch (type)
    {
      // print the document element
      case Node.DOCUMENT_NODE:
        {
          System.out.println("<?xml version=\"1.0\" ?>");
          printDOMTree(((Document)node).getDocumentElement());
          break;
        }

        // print element with attributes
      case Node.ELEMENT_NODE:
        {
          System.out.print("<");
          System.out.print(node.getNodeName());

          if (node.hasAttributes())
          {
            NamedNodeMap attrs = node.getAttributes();
            for (int i = 0; i < attrs.getLength(); i++)
              printDOMTree(attrs.item(i));
          }
          System.out.print(">");

          if (node.hasChildNodes())
          {
            NodeList children = node.getChildNodes();
            for (int i = 0; i < children.getLength(); i++)
              printDOMTree(children.item(i));
          }

          break;
        }

      // Print attribute nodes
      case Node.ATTRIBUTE_NODE:
        {
          System.out.print(" " + node.getNodeName() + "=\"");
          if (node.hasChildNodes())
          {
```

```
          NodeList children = node.getChildNodes();
          for (int i = 0; i < children.getLength(); i++)
            printDOMTree(children.item(i));
        }
        System.out.print("\"");
        break;
      }

      // handle entity reference nodes
    case Node.ENTITY_REFERENCE_NODE:
      {
        System.out.print("&");
        System.out.print(node.getNodeName());
        System.out.print(";");
        break;
      }

      // print cdata sections
    case Node.CDATA_SECTION_NODE:
      {
        System.out.print("<![CDATA[");
        System.out.print(node.getNodeValue());
        System.out.print("]]>");
        break;
      }

      // print text
    case Node.TEXT_NODE:
      {
        System.out.print(node.getNodeValue());
        break;
      }

      // print processing instruction
    case Node.PROCESSING_INSTRUCTION_NODE:
      {
        System.out.print("<?");
        System.out.print(node.getNodeName());
        String data = node.getNodeValue();
        {
          System.out.print(" ");
          System.out.print(data);
        }
        System.out.print("?>");
        break;
      }
  }

  if (type == Node.ELEMENT_NODE)
  {
    System.out.print("</");
    System.out.print(node.getNodeName());
    System.out.print('>');
  }
} // printDOMTree(Node)

public void parseAndPrint(String uri)
{
  try
  {
    dbf = DocumentBuilderFactory.newInstance();
    docBuilder = dbf.newDocumentBuilder();

    SAXParserFactory spf = SAXParserFactory.newInstance();
    SAXParser sp = spf.newSAXParser();
    sp.parse(uri, this);

    if (root != null)
      printDOMTree(root);
  }
  catch (ParserConfigurationException pce)
  {
    System.err.println("Parser configuration error : " + pce);
  }
  catch (Exception e)
  {
    System.err.println(e);
  }
}

/** Start document. */
public void startDocument()
{
  // If we knew the name of the root element, we could create
  // the Document object here.  We don't know the name of the
  // root, so we'll create it in startElement(), using a flag
  // to see if this is our first (aka root) element.
}

/** Start element. */
public void startElement(String namespaceURI, String localName,
                         String rawName, Attributes attrs)
{
  if (firstElementNotFoundYet)
  {
    root = docBuilder.getDOMImplementation().
      createDocument(namespaceURI, rawName, null);
    Element docElement = root.getDocumentElement();
    if (attrs != null)
    {
      int len = attrs.getLength();
      for (int i = 0; i < len; i++)
        docElement.setAttribute(attrs.getQName(i), attrs.getValue(i));
    }

    elementStack = new Stack();
    elementStack.push(docElement);
    firstElementNotFoundYet = false;
  }
  else
  {
    Element currentElement = root.createElement(rawName);
    if (attrs != null)
    {
      int len = attrs.getLength();
      for (int i = 0; i < len; i++)
        currentElement.setAttribute(attrs.getQName(i), attrs.getValue(i));
    }
    elementStack.push(currentElement);
  }
} // startElement

/** Ignorable whitespace. */
public void ignorableWhitespace(char ch[], int start, int length)
{
  characters(ch, start, length);
}

/** Characters. */
public void characters(char ch[], int start, int length)
{
  ((Element) elementStack.peek()).
    appendChild(root.createTextNode(new String(ch, start, length)));
} // characters(char[],int,int);

/** End element. */
public void endElement(String namespaceURI, String localName,
                       String rawName)
{
  if (elementStack.size() > 1)
  {
    Element currentElement = (Element) elementStack.pop();
    ((Element) elementStack.peek()).appendChild(currentElement);
  }
} // endElement(String)

/** End document. */
public void endDocument()
{
  // Do nothing...
} // endDocument()

/** Processing instruction. */
public void processingInstruction(String target, String data)
{
  ((Element) elementStack.peek()).
    appendChild(root.createProcessingInstruction(target, data));
} // processingInstruction(String,String)

//
// ErrorHandler methods
//

/** Warning. */
public void warning(SAXParseException ex)
{
  System.err.println("[Warning] "+
                     getLocationString(ex)+": "+
                     ex.getMessage());
}

/** Error. */
public void error(SAXParseException ex)
{
  System.err.println("[Error] "+
                     getLocationString(ex)+": "+
                     ex.getMessage());
}

/** Fatal error. */
public void fatalError(SAXParseException ex)
  throws SAXException
{
  System.err.println("[Fatal Error] "+
                     getLocationString(ex)+": "+
                     ex.getMessage());
  throw ex;
}

/** Returns a string of the location. */
private String getLocationString(SAXParseException ex)
{
  StringBuffer str = new StringBuffer();

  String systemId = ex.getSystemId();
  if (systemId != null)
  {
    int index = systemId.lastIndexOf('/');
    if (index != -1)
        systemId = systemId.substring(index + 1);
    str.append(systemId);
  }
  str.append(':');
  str.append(ex.getLineNumber());
  str.append(':');
  str.append(ex.getColumnNumber());

  return str.toString();
} // getLocationString(SAXParseException):String

/** Main program entry point. */
public static void main(String argv[])
{
  if (argv.length == 0 ||
        (argv.length == 1 && argv[0].equals("-help")))
  {
    System.out.println("\nUsage:  java SaxToDom uri");
    System.out.println("   where uri is the URI of your XML document.");
    System.out.println("   Sample:  java SaxToDom sonnet.xml");
    System.out.println("\nEchoes SAX events back to the console.");
    System.exit(1);
  }

  SaxToDom s2d = new SaxToDom();
  s2d.parseAndPrint(argv[0]);
} // main(String[])
}
```

DomToSax.java

```
/*
 * (C) Copyright IBM Corp. 2004.  All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantibility and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.helpers.AttributesImpl;
import org.xml.sax.helpers.DefaultHandler;

public class DomToSax
  extends DefaultHandler
{
  DocumentBuilderFactory dbf = null;
  DocumentBuilder db = null;

  public void parseAndPrint(String uri)
      throws SAXException
  {
    Document doc = null;

    try
    {
      dbf = DocumentBuilderFactory.newInstance();
      db = dbf.newDocumentBuilder();
      doc = db.parse(uri);
      if (doc != null)
        generateSAXEvents(doc, this);
    }
    catch (Exception e)
    {
      System.err.println("Sorry, an error occurred: " + e);
    }
  }

  /** Generates SAX events from the DOM tree. */
  public void generateSAXEvents(Node node, DefaultHandler dh)
      throws SAXException
  {
    int type = node.getNodeType();
    switch (type)
    {
      // The document node corresponds to the startDocument() event
      case Node.DOCUMENT_NODE:
        {
          dh.startDocument();
          generateSAXEvents(((Document)node).getDocumentElement(), dh);
          dh.endDocument();
          break;
        }

        // print element with attributes
      case Node.ELEMENT_NODE:
        {
          AttributesImpl saxAttrs = new AttributesImpl();
          if (node.hasAttributes())
          {
            NamedNodeMap attrs = node.getAttributes();
            for (int i = 0; i < attrs.getLength(); i++)
              saxAttrs.addAttribute(null, null,
                                    attrs.item(i).getNodeName(),
                                    null, attrs.item(i).getNodeValue());
          }

          dh.startElement(null, null, node.getNodeName(), saxAttrs);

          if (node.hasChildNodes())
          {
            NodeList children = node.getChildNodes();
            for (int i = 0; i < children.getLength(); i++)
              generateSAXEvents(children.item(i), dh);
          }

          dh.endElement(null, null, node.getNodeName());
          break;
        }

        // print text
      case Node.TEXT_NODE:
        {
          char[] chars = node.getNodeValue().toCharArray();
          dh.characters(chars, 0, chars.length);
          break;
        }

        // print processing instruction
      case Node.PROCESSING_INSTRUCTION_NODE:
        {
          dh.processingInstruction(node.getNodeName(),
                                   node.getNodeValue());
          break;
        }
    }
  } // generateSAXEvents(Node, ContentHandler)

  /** Start document. */
  public void startDocument()
  {
    System.out.println("<?xml version=\"1.0\"?>");
  } // startDocument()

  /** Start element. */
  public void startElement(String namespaceURI, String localName,
                           String rawName, Attributes attrs)
  {
    System.out.print("<");
    System.out.print(rawName);
    if (attrs != null)
    {
      int len = attrs.getLength();
      for (int i = 0; i < len; i++)
      {
        System.out.print(" ");
        System.out.print(attrs.getQName(i));
        System.out.print("=\"");
        System.out.print(attrs.getValue(i));
        System.out.print("\"");
      }
    }
    System.out.print(">");
  } // startElement(String,AttributeList)

  /** Characters. */
  public void characters(char ch[], int start, int length)
  {
    System.out.print(new String(ch, start, length));
  } // characters(char[],int,int);

  /** Ignorable whitespace. */
  public void ignorableWhitespace(char ch[], int start, int length)
  {
    characters(ch, start, length);
  } // ignorableWhitespace(char[],int,int);

  /** End element. */
  public void endElement(String namespaceURI, String localName,
                         String rawName)
  {
    System.out.print("</");
    System.out.print(rawName);
    System.out.print(">");
  } // endElement(String)

  /** End document. */
  public void endDocument()
  {
    // No need to do anything.
  } // endDocument()

  /** Processing instruction. */
  public void processingInstruction(String target, String data)
  {
    System.out.print("<?");
    System.out.print(target);
    if (data != null && data.length() > 0)
    {
      System.out.print(' ');
      System.out.print(data);
    }
    System.out.print("?>");

  } // processingInstruction(String,String)

  //
  // ErrorHandler methods
  //

  /** Warning. */
  public void warning(SAXParseException ex)
  {
    System.err.println("[Warning] "+
                       getLocationString(ex)+": "+
                       ex.getMessage());
  }

  /** Error. */
  public void error(SAXParseException ex)
  {
    System.err.println("[Error] "+
                       getLocationString(ex)+": "+
                       ex.getMessage());
  }

  /** Fatal error. */
  public void fatalError(SAXParseException ex)
    throws SAXException
  {
    System.err.println("[Fatal Error] "+
                       getLocationString(ex)+": "+
                       ex.getMessage());
    throw ex;
  }

  /** Returns a string of the location. */
  private String getLocationString(SAXParseException ex)
  {
    StringBuffer str = new StringBuffer();

    String systemId = ex.getSystemId();
    if (systemId != null)
    {
      int index = systemId.lastIndexOf('/');
      if (index != -1)
        systemId = systemId.substring(index + 1);
      str.append(systemId);
    }
    str.append(':');
    str.append(ex.getLineNumber());
    str.append(':');
    str.append(ex.getColumnNumber());

    return str.toString();
  } // getLocationString(SAXParseException):String
  /** Main program entry point. */
  public static void main(String argv[])
  {
    if (argv.length == 0 ||
        (argv.length == 1 && argv[0].equals("-help")))
    {
      System.out.println("\nUsage:  java DomToSax uri");
      System.out.println("   where uri is the URI of your XML document.");
      System.out.println("   Sample:  java DomToSax sonnet.xml");
      System.out.println("\nEchoes SAX events back to the console.");
      System.exit(1);
    }

    DomToSax d2s = new DomToSax();
    try
    {
      d2s.parseAndPrint(argv[0]);
    }
    catch (SAXException se) { }
  } // main(String[])
}
```

DomSorter.java

```
/*
 * (C) Copyright IBM Corp. 2004.  All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantibility and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

import com.ibm.dw.xmlprogjava.DomTreePrinter;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

/**
 * A sample DOM application.  This application reads an XML document,
 * builds a DOM tree, then sorts the nodes in the DOM tree.
 */

public class DomSorter
{
```

```
public void parseAndSortLines(String uri)
{
  Document doc = null;

  try
  {
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
    dbf.setIgnoringElementContentWhitespace(true);
    DocumentBuilder db = dbf.newDocumentBuilder();
    doc = db.parse(uri);
    if (doc != null)
    {
      NodeList theLines = doc.getDocumentElement().
                          getElementsByTagName("line");
      sortLines(theLines);
      DomTreePrinter.printNode(doc);
    }
  }
  catch (Exception e)
  {
    System.err.println("Sorry, an error occurred: " + e);
  }
}

public String getTextFromLine(Node lineElement)
{
  StringBuffer returnString = new StringBuffer();

  if (lineElement.getNodeName().equals("line"))
  {
    NodeList kids = lineElement.getChildNodes();
    if (kids != null)
      if (kids.item(0).getNodeType() == Node.TEXT_NODE)
        returnString.append(kids.item(0).getNodeValue());
  }
  else
    returnString.setLength(0);

  return new String(returnString);
}

/** Sorts the <line> elements in the file.
    It uses a bubble sort algorithm, since a
    sonnet only has 14 lines.                **/
public void sortLines(NodeList theLines)
{
  if (theLines != null)
  {
    int len = theLines.getLength();
    for (int i = 0; i < len; i++)
      for (int j = 0; j < (len - 1 - i); j++)
        if (getTextFromLine(theLines.item(j)).
            compareTo(getTextFromLine(theLines.item(j+1)))
            > 0)
          theLines.item(j).getParentNode().
            insertBefore(theLines.item(j+1),
                         theLines.item(j));
  }
  }

  /** Main program entry point. */
  public static void main(String argv[])
  {
    if (argv.length == 0 ||
        (argv.length == 1 && argv[0].equals("-help")))
    {
      System.out.println("\nUsage:  java DomSorter uri");
      System.out.println("   where uri is the URI of the XML " +
                         "document you want to sort.");
      System.out.println("   Sample:  java DomSorter sonnet.xml");
      System.out.println("   Note: Your XML document must " +
                         "use the sonnet DTD.");
      System.out.println("\nSorts the lines of an XML sonnet, then " +
                         "writes the DOM tree to the console.");
      System.exit(1);
    }

    DomSorter ds = new DomSorter();
    ds.parseAndSortLines(argv[0]);
  }
}
```

DomAttributes.java

```
/*
 * (C) Copyright IBM Corp. 2004.  All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantibility and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

import com.ibm.dw.xmlprogjava.DomTreePrinter;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;
import org.w3c.dom.Element;

/**
 * A sample DOM application.  This application reads an XML document,
 * builds a DOM tree, adds an attribute to each element in the DOM
 * tree, then prints the DOM tree.
 */
```

```
public class DomAttributes
{
  public void parseAndProcessAttributes(String uri)
  {
    Document doc = null;

    try
    {
      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
      dbf.setIgnoringElementContentWhitespace(true);
      DocumentBuilder db = dbf.newDocumentBuilder();
      doc = db.parse(uri);
      if (doc != null)
      {
        addAttributes(doc);
        DomTreePrinter.printNode(doc);
      }
    }
    catch (Exception e)
    {
      System.err.println("Sorry, an error occurred: " + e);
    }
  }

  public void addAttributes(Node node)
  {
    int type = node.getNodeType();
    switch (type)
    {
      // print the document element
      case Node.DOCUMENT_NODE:
      {
        addAttributes(((Document)node).getDocumentElement());
        break;
      }

      case Node.ELEMENT_NODE:
      {
        Element el = (Element)node;
        el.setAttribute("size", "12");
        if (node.hasChildNodes())
        {
          NodeList children = node.getChildNodes();
          for (int i = 0; i < children.getLength(); i++)
            addAttributes(children.item(i));
        }
        break;
      }
    }
  }

  /** Main program entry point. */
  public static void main(String argv[])
  {
    if (argv.length == 0 ||
        (argv.length == 1 && argv[0].equals("-help")))
    {
      System.out.println("\nUsage:  java DomAttributes uri");
      System.out.println("   where uri is the URI of the XML " +
                         "document you want to sort.");
      System.out.println("   Sample:  java DomAttributes sonnet.xml");
      System.out.println("\nAdds an attribute to all of the elements " +
                         "in the DOM tree, then writes the DOm tree " +
                         "to the console.");
      System.exit(1);
    }

    DomAttributes da = new DomAttributes();
    da.parseAndProcessAttributes(argv[0]);
  }
}
```

JdomSorter.java

```
/*
 * (C) Copyright IBM Corp. 2004.  All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantibility and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

import java.io.File;
import java.io.IOException;
import java.util.List;
import java.util.ListIterator;
import org.jdom.DocType;
import org.jdom.Document;
import org.jdom.Element;
import org.jdom.input.SAXBuilder;
import org.jdom.output.XMLOutputter;

public class JdomSorter
{
  public static void main(String[] argv)
  {
    if (argv.length == 0 ||
        (argv.length == 1 && argv[0].equals("-help")))
    {
      System.out.println("\nUsage:  java JdomSorter uri");
      System.out.println("   where uri is the URI of your XML sonnet.");
      System.out.println("   Sample:  java JdomSorter sonnet.xml");
      System.out.println("\nUses the JDOM API to parse an XML sonnet, " +
                         "sorts the lines of the sonnet,");
      System.out.println("then writes the sorted sonnet " +
                         "back to the console.");
      System.exit(1);
    }

    JdomSorter js = new JdomSorter();
    js.parseAndSortLines(argv[0]);
  }

  public void parseAndSortLines(String uri)
  {
    try
    {
      SAXBuilder sb = new SAXBuilder();
      Document doc = sb.build(new File(uri));
      sortLines(doc);
      XMLOutputter xo = new XMLOutputter();
      xo.setTrimAllWhite(true);
      xo.setIndent("  ");
      xo.setNewlines(true);
      xo.output(doc, System.out);
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
  }

  public void sortLines(Document sonnet)
  {
    Element linesElement = sonnet.getRootElement().
      getChild("lines");
    List lines = linesElement.getChildren("line");
    for (int i = 0; i < 14; i++)
      for (int j = 0; j < (14 - 1 - i); j++)
        if (((Element)lines.get(j)).getText().
             compareTo(((Element)lines.get(j+1)).getText())
             > 0)
          lines.add(j, lines.remove(j+1));
  }
}
```

sonnetSorter.xsl

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

  <xsl:template match="lines">
    <lines>
      <xsl:for-each select="line">
        <xsl:sort/>
        <xsl:copy>
          <xsl:apply-templates select="*|@*|text()"/>
        </xsl:copy>
      </xsl:for-each>
    </lines>
  </xsl:template>

  <xsl:template match="*|@*|text()">
```

```
    <xsl:copy>
      <xsl:apply-templates select="*|@*|text()"/>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

DomThree.java

```
/*
 * (C) Copyright IBM Corp. 2004.  All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantibility and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

import com.ibm.dw.xmlprogjava.DomTreePrinter;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;

/**
 * This code uses a separate DomTreePrinter class to
 * write a DOM tree to standard output.
 */

public class DomThree
{
  public void parseAndPrint(String uri)
  {
    Document doc = null;

    try
    {
      DocumentBuilderFactory dbf =
        DocumentBuilderFactory.newInstance();
      DocumentBuilder db = dbf.newDocumentBuilder();
      doc = db.parse(uri);
      if (doc != null)
        DomTreePrinter.printNode(doc);
    }
    catch (Exception e)
    {
      System.err.println("Sorry, an error occurred: " + e);
    }
  }

  /** Main program entry point. */
  public static void main(String argv[])
  {
    if (argv.length == 0 ||
        (argv.length == 1 && argv[0].equals("-help")))
    {
      System.out.println("\nUsage:  java DomThree uri");
      System.out.println("   where uri is the URI of the XML " +
                         "document you want to print.");
      System.out.println("   Sample:  java DomThree sonnet.xml");
      System.out.println("\nParses an XML document, then writes " +
                         "the DOM tree to the console.");
      System.exit(1);
    }

    DomThree d3 = new DomThree();
    d3.parseAndPrint(argv[0]);
  }
}
```

DomTreePrinter.java

```
/*
 * (C) Copyright IBM Corp. 2004.  All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantibility and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

package com.ibm.dw.xmlprogjava;

//import javax.xml.parsers.DocumentBuilder;
//import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.NamedNodeMap;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

/**
 * A DOM traversal program.  Given a DOM Node, the one static method
 * of this program prints the contents of the Node as XML.
 */

public class DomTreePrinter
{
  /** Prints the specified node, recursively. */
  public static void printNode(Node node)
  {
    int type = node.getNodeType();
    switch (type)
    {
      // print the document element
      case Node.DOCUMENT_NODE:
        {
          System.out.println("<?xml version=\"1.0\" ?>");
          printNode(((Document)node).getDocumentElement());
          break;
        }

         // print element and any attributes
      case Node.ELEMENT_NODE:
        {
          System.out.print("<");
          System.out.print(node.getNodeName());

          if (node.hasAttributes())
          {
            NamedNodeMap attrs = node.getAttributes();
            for (int i = 0; i < attrs.getLength(); i++)
              printNode(attrs.item(i));
          }
          System.out.print(">");

          if (node.hasChildNodes())
          {
            NodeList children = node.getChildNodes();
            for (int i = 0; i < children.getLength(); i++)
              printNode(children.item(i));
          }

          break;
        }

      // Print attribute nodes
      case Node.ATTRIBUTE_NODE:
        {
          System.out.print(" " + node.getNodeName() + "=\"");
          if (node.hasChildNodes())
          {
            NodeList children = node.getChildNodes();
            for (int i = 0; i < children.getLength(); i++)
              printNode(children.item(i));
          }
          System.out.print("\"");
          break;
        }

        // handle entity reference nodes
      case Node.ENTITY_REFERENCE_NODE:
        {
          System.out.print("&");
          System.out.print(node.getNodeName());
          System.out.print(";");
          break;
        }
        // print cdata sections
      case Node.CDATA_SECTION_NODE:
        {
          System.out.print("<![CDATA[");
          System.out.print(node.getNodeValue());
          System.out.print("]]>");
          break;
        }

        // print text
      case Node.TEXT_NODE:
        {
          System.out.print(node.getNodeValue());
          break;
        }

      case Node.COMMENT_NODE:
        {
          System.out.print("<!--");
          System.out.print(node.getNodeValue());
          System.out.print("-->");
          break;
        }

        // print processing instruction
      case Node.PROCESSING_INSTRUCTION_NODE:
        {
          System.out.print("<?");
          System.out.print(node.getNodeName());
          String data = node.getNodeValue();
          {
            System.out.print(" ");
            System.out.print(data);
          }
          System.out.print("?>");
          break;
        }
    }

    if (type == Node.ELEMENT_NODE)
    {
      System.out.print("</");
      System.out.print(node.getNodeName());
      System.out.print('>');
    }
  } // printNode(Node)
}
```

DomFour.java

```
/*
 * (C) Copyright IBM Corp. 2004.  All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantibility and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

import org.apache.xerces.dom.DOMOutputImpl;
import org.w3c.dom.Document;
import org.w3c.dom.bootstrap.DOMImplementationRegistry;
import org.w3c.dom.ls.DOMImplementationLS;
import org.w3c.dom.ls.LSOutput;
import org.w3c.dom.ls.LSParser;
import org.w3c.dom.ls.LSSerializer;

/**
 * This code uses the DOM Level 3 LS classes to serialize
 * an XML file to standard output.
 */

public class DomFour
{
  public void parseAndPrint(String uri)
  {
    Document doc = null;

    try
    {
      System.setProperty(DOMImplementationRegistry.PROPERTY,
        "org.apache.xerces.dom.DOMXSImplementationSourceImpl");
      DOMImplementationRegistry direg =
        DOMImplementationRegistry.newInstance();
      DOMImplementationLS dils =
        (DOMImplementationLS) direg.getDOMImplementation("LS");
      LSParser lsp = dils.createLSParser
        (DOMImplementationLS.MODE_SYNCHRONOUS, null);

      doc = lsp.parseURI(uri);

      LSSerializer domWriter = dils.createLSSerializer();
      LSOutput lso = new DOMOutputImpl();
      lso.setByteStream(System.out);
      domWriter.write(doc, lso);
    }
    catch (Exception e)
    {
      System.err.println("Sorry, an error occurred: " + e);
    }
  }

  /** Main program entry point. */
  public static void main(String argv[])
  {
    if (argv.length == 0 ||
        (argv.length == 1 && argv[0].equals("-help")))
    {
      System.out.println("\nUsage:  java DomFour uri");
      System.out.println("   where uri is the URI of the XML " +
                         "document you want to print.");
      System.out.println("   Sample:  java DomFour sonnet.xml");
      System.out.println("\nParses an XML document, then writes " +
                         "the DOM tree to the console.");
      System.exit(1);
    }

    DomFour d4 = new DomFour();
    d4.parseAndPrint(argv[0]);
  }
}
```

SaxKiller.java

```
/*
 * (C) Copyright IBM Corp. 2004.  All rights reserved.
 *
 * US Government Users Restricted Rights Use, duplication or
 * disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
 *
 * The program is provided "as is" without any warranty express or
 * implied, including the warranty of non-infringement and the implied
 * warranties of merchantibility and fitness for a particular purpose.
 * IBM will not be liable for any damages suffered by you as a result
 * of using the Program. In no event will IBM be liable for any
 * special, indirect or consequential damages or lost profits even if
 * IBM has been advised of the possibility of their occurrence. IBM
 * will not be liable for any third party claims against you.
 */

import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;
import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.SAXParseException;
import org.xml.sax.helpers.DefaultHandler;
import org.xml.sax.helpers.LocatorImpl;

/**
 * A sample SAX application.  This code illustrates how to kill
 * the SAX parser when you've found what you needed from your
 * document.  In this case, we kill the parser after the fourth
 * <line> element is found.
 */

public class SaxKiller
  extends DefaultHandler
{
  int lineCount = 0;
  boolean inFourthLine = false;
  StringBuffer fourthLine = new StringBuffer();
  boolean notSuccessful = true;

  public boolean isNotSuccessful()
  {
    return notSuccessful;
  }

  public void parseURI(String uri)
      throws Exception
  {
    SAXParserFactory spf = SAXParserFactory.newInstance();
    SAXParser sp = spf.newSAXParser();
    sp.parse(uri, this);
  }

  /** Start element. */
  public void startElement(String namespaceURI, String localName,
                           String rawName, Attributes attrs)
  {
    if (rawName.equals("line") && ++lineCount == 4)
        inFourthLine = true;
  } // startElement(String,AttributeList)

  /** Characters. */
  public void characters(char ch[], int start, int length)
  {
    if (inFourthLine)
      fourthLine.append(new String(ch, start, length));
  } // characters(char[],int,int);

  /** End element. */
  public void endElement(String namespaceURI, String localName,
                         String rawName)
    throws SAXException
  {
    if (rawName.equals("line") && inFourthLine)
    {
      System.out.println("\nThe text of the fourth line is: \n");
      System.out.println("\t" + fourthLine);

      notSuccessful = false;
      SAXParseException spe =
        new SAXParseException("Found the fourth <line>, " +
                              "so we killed the parser!",
                              new LocatorImpl());
      fatalError(spe);
    }
  } // endElement(String)

  //
  // ErrorHandler methods
  //

  /** Warning. */
  public void warning(SAXParseException ex)
  {
    System.err.println("\n\n[Warning] "+
                       getLocationString(ex)+": "+
                       ex.getMessage());
  }
  /** Error. */
  public void error(SAXParseException ex)
  {
    System.err.println("\n\n[Error] "+
                       getLocationString(ex)+": "+
                       ex.getMessage());
  }

  /** Fatal error. */
  public void fatalError(SAXParseException ex)
      throws SAXParseException
  {
    throw ex;
  }

  /** Returns a string of the location. */
  private String getLocationString(SAXParseException ex)
  {
    StringBuffer str = new StringBuffer();

    String systemId = ex.getSystemId();
    if (systemId != null)
    {
      int index = systemId.lastIndexOf('/');
      if (index != -1)
        systemId = systemId.substring(index + 1);
      str.append(systemId);
    }
    str.append(':');
    str.append(ex.getLineNumber());
    str.append(':');
    str.append(ex.getColumnNumber());

    return str.toString();
  } // getLocationString(SAXParseException):String

  /** Main program entry point. */
  public static void main(String argv[])
  {
    if (argv.length == 0 ||
        (argv.length == 1 && argv[0].equals("-help")))
    {
      System.out.println("\nUsage:  java SaxKiller uri");
      System.out.println("   where uri is the URI of your XML document.");
      System.out.println("   Sample:  java SaxKiller sonnet.xml");
      System.out.println("\nIllustrates how to kill a SAX parser by " +
                         "throwing an exception.");
      System.exit(1);
    }

    SaxKiller s1 = new SaxKiller();
    try
    {
      s1.parseURI(argv[0]);
    }
    // We're expecting an exception, so we ignore anything that happens...
    catch (Exception e)
    {
      if (s1.isNotSuccessful())
        System.err.println(e);
    }
  } // main(String[])
}
```