Lab 4: XML SAX and DOM Practice

Due date: Monday, February 2, beginning of the lab period.

Lab Assignment

The purpose of this assignment is to learn XML DOM and SAX APIs and use them to traverse, analyze and extract information from XML documents.

What to look for

Programs that work with XML data, do not directly access XML files stored on disk. Instead, XML data is read once, parsed into an internal data structure and this data structure is accessed and traversed by the program.

The internal data structure is known as DOM, Document Object Model, or DOM Tree. The API is known as DOM API.

DOM API is a World Wide Web Consortium standard. The portion of the DOM API that is of interest to us includes main interfaces (in Java sense of this word) for representing XML Nodes: elements, text, attributes, etc., as well as interfaces for representing *lists of XML nodes* and *collections of attributes*.

The key DOM API functionality can be broken into two categories:

- **DOM Tree Manipulation.** This category includes methods for creation and modification of the DOM tree. Creation of individual XML nodes, their insertion into DOM trees, deletion of XML nodes from DOM trees, and changes in the node content can be done via DOM API methods.
- **DOM Tree Traversal.** This category includes methods for traversal of DOM trees. Given a node in a DOM tree, DOM API allows to retrieve information about (and "move to") its parent, its children, its siblings.

Stage 2 of the course project will involve implementing an API of XML access methods. Some (and possibly all) of the methods in this API will come from the DOM API. It is therefore important that you have first-hand knowledge of DOM API and the work of individual tree manipulation and tree traversal methods.

The Tasks

This is a team lab: each project team needs to submit only one assignment.

Your task is to write a number of Java programs which accept as input an XML file and produce output as specified below.

XMLElmentHistogram

The first program you shall implement is XMLElementHistogram. This program accepts as input an XML file, parses it into a DOM tree, and outputs the list of all XML element names found in the XML file. For each XML element, the program prints the number of times it is found in the XML document. The XML elements are outputed in the descending order of the number of occurrences.

For example, consider the following XML file `test1.xml`.

```
<root>
  <a>1</a>
  <b><a><b>3</b><b>4</b></a><c>1</c></b>
  <b><c>6</c></b>
</root>
```

Here is the expected output of the XMLElementHistogram program:

```
> java XMLElementHistogram test1.xml

<b>   : 4
<a>   : 2
<c>   : 2
<root>: 1
```

The above is the minimal required output. You can improve it to actually display a histogram of element occurrences, for example as follows:

```
<b>   : **** 4
<a>   : **   2
<c>   : **   2
<root>: *    1
```

Notice that input to this program may be a **very large** XML file, so when implementing a histogram for such a file, one "*" on the histogram should represent more than one XML element occurrence.

DFSOrder

The second program is called DFSOrder. This program shall accept as input an XML file name, and produce, as output the list of all XML elements in this file, together with their pre-order and post-order index numbers (see the XML index structures project handout for more information on these numbers).

For example, consider the following XML file `test2.xml`

```
<a>
   <b>
      <c>1</c>
      <c>2</c>
   </b>
   <c>3</c>
 </a>
```

The minimal output of the program will be of the form:

```
> java DFSOrder test2.xml

Element     Preorder    Postorder
---------------------------------
<a>            1          10
<b>            2          7
<c>            3          4
<c>            5          6
<c>            8          9
---------------------------------
```

More involved output may employ visual cues (e.g., indentation) to represent the structure of the XML document.

This program will be tested for correctness only on relatively small XML documents.

XMLStructureIndex

The third program is called XMLStructureIndex. This program shall accept as input an XML file name and produce, as output the list of XML element nodes with the following information associated with each of them:

- Unique Id. Feel free to number the nodes in the order of traversal you perform.
- Parent. The id of the parent element.
- Layer. The layer (length of a path from the root) of the element.
- Sibling Rank. The position of the element in the list of children of its parent.
- Leaf or not? Information on whether the element is a leaf element of not a leaf element.

For example, for file `test2.xml` (see above), the minimal program output will be:

```
Element     Id   Parent   Layer  Sibling Rank   Leaf?
---------------------------------------------------------
<a>           1    No       0        0 (or 1)      No
<b>           2    1        1        1             No
<c>           3    2        2        1             Yes
<c>           4    2        2        2             Yes
<c>           5    1        1        2             Yes
---------------------------------------------------------
```

This program will be tested for correctness only on relatively small XML documents.

XMLAttributeExtract

The fourth program will take as input the name of an XML file, and will do the following. First, the XML file shall be parsed into a DOM tree. The DOM tree shall be searched for all XML elements that have attributes associated with them. The program then, shall output the following information:

- For each XML element with at least one attribute, list its name and its DFS number (i.e., the order in which this element is accessed during a DFS traverals of the XML elements of the DOM tree).
- For each attribute of such an XML element, list its name, and its assigned value. Attributes should be listed under their element names. (see sample output below).

Consider the following XML file `test3.xml`.

```
<root id="1">
  <a id="2" l="4">
    <b>1</b>
    <c id="4">2</c>
  </a>
  <b l = "16">
    <a>1</a>
  </b>
</root>
```

The output can look as follows:

```
> java XMLAttributeExtract test3.xml

Node: <root>, Id: 1

    Attributes:  Name   Value
                 -------------
                  id      1

Node: <a>, Id: 2

    Attributes:  Name   Value
                 -------------
                  id      2
                  l       4

Node: <c>, Id: 4

    Attributes:  Name   Value
                 -------------
                  id      4

Node: <b>, Id: 5

    Attributes:  Name   Value
                 -------------
                  l      16
```

XMLContentExtract

The fifth, and final program is called `XMLContentExtract`. It takes as input the name of an XML file. As output, it produces the string, containing, in order of occurrence in the XML file, the full content of the XML document.

The content of individual XML elements shall be separated by "|" markers (see output below). It is the responsibility of your program to format the output "nicely" to ensure that it fits the screen (read: you are responsible for inserting line breaks in somewhat reasonable places).

Consider the following XML file `test4.xml`.

```
<root>
  <s>A quick <color>brown</color>
     fox <verb>jumps</verb> over a
    <lazy> lazy</lazy> dog. </s>
  <author>unknown</author>
</root>
```

A possible output may look as follows:

```
> java XMLContentExtract test4.xml

 | A quick | brown | fox | jumps | over a |
lazy | dog. | unknown |
```

Testing

Your programs will be tested for correctness and scalability.

Correctness testing. Correctness testing will be conducted typically using small or medium-sized XML documents, for which it is relatively straightforward to establish correct output. Most of the files will come out of the Lab 2 submissions made by teams. Feel free to use all the files in your testing. The links to the group pages with XML files can be found at

http://wiki.csc.calpoly.edu/csc468-winter2009/

For programs where it explicitly is indicated, only small-sized XML documents will be used for correctness testing.

Scalability testing. Scalability testing will be conducted to ensure that your program can handle reasonably large input data. A number of publically available XML datasets will be used for scalability testing. Links to these files will be made available on the course web page before the due date for the assignment. We will not be verifying correctness of your program's work on large-scale XML documents (this might prove hard to do). We will make sure that the program produces output in reasonable time, does not choke, dump core, or exhibit any other abnormal behavior.

Parting Notes

- **Ignore** all DTD/XML Schema information in the XML files. That is, use non-validating mode for your parsing.

- **DOM vs. SAX**. SAX, Simple API for XML is an event-based XML parsing API. It is used by the DOM implementation to actuall read the data from the XML file. SAX API is also available to you directly. Some programs do not require the use of DOM API for achieving their goal: SAX is sufficient. (the main difference is that DOM creates an in-memory data structure which can be traversed multiple times. SAX sees events of three types: start tag, content and end tag, and passed these events to the calling program, which then handles them. However, once an event is generated, it is gone from memory, so SAX does not use an internal data structure to represent XML). If you determine that there is no need to use DOM, you are allowed to build your program using SAX API only.
- Note that a few of the programs you have to write have to collect the data which is needed for creation of the native XML index structures for the project. Feel free to draw relevant conclusions from this observation.

Submission Instructions

Your files should be submitted as a single archive (either a zip or a gzipped tar). Name your file `<group>-lab04.ext` where `.ext` is the appropriate extension and `<group>` is the name of the group (`Money, Pointers, Food, Group-Name`). Create a `README` file, containing the list of group members, and any instructions necessary for compiling and running the program.

As usual, my preference is to run the programs from command line, NOT from inside Eclipse or other development environment.

Submit via `handin` using the following command:

```
handin dekhtyar lab04 <archive-file> README
```