CS193j, Stanford
Winter, 2001-02

Handout #30
Nick Parlante

HW4 - XEdit

This is a relatively small homework to give you a little practice with Java's XML parsing features. HW4 is due midnight ending Thu Mar 14th. For this assignment, you will build a tool that does search/replace edits on XML files. Programs like perl, awk are very popular for this sort of thing with straight text files, and we will extend the idea to a program which understands the structure of XML. Of course, we will rely on Java's XML libraries to do the work of parsing and writing the XML itself.

Suppose we have the following XML file...

```
<?xml version="1.0" encoding="UTF-8"?>
<products>
  <product>
    <title>Widget</title>
    <id>10</id>
    <color>red</color>
    <description>This finely crafted red widget, with over 10 jewels,
    will be a welcome addition to any widget collection.</description>
  </product>
  <product>
    <title>Box Of Peeps</title>
    <id>11</id>
    <color>yellow</color>
  </product>
  <product>
    <title>USB Cable</title>
    <id>12</id>
    <color>green</color>
    <description>A lovely USB cable (includes a free Widget)</description>
    <bundle><id>10</id></bundle>
  </product>
</products>
```

XEdit uses command line arguments to specify search/replace operations on an XML file. The program reads the XML file specified in the last argument, processes it, and writes the new form of the XML to standard output.

Usage: java XEdit -s search -r replace -t tag file.xml

-s search = the string to search for in text between tags (these are of type TEXT_NODE in the DOM tree). The search is case-sensitive.

-r replace = the replacement text for the search string. Defaults to the empty string "" if not specified, in which case the search text is effectively deleted.

-t tag = a constraint to only do the search/replace when nested inside the given tag. If not specified, the search/replace happens in all the text nodes in the document.

So for example "-s 10 -r 11 -t id products.xml" changes the two occurrences of "<id>10</id>" to "<id>11</id>" -- one in Widget and one in the USB Cable <bundle> section. Note that the 10 in the description is not changed. This is the advantage of XML -- the data is structured, so we can change the id 10's without disturbing the other 10's.

The search/replace may be nested a few levels below the tag, so "-s red -r rouge -t product products.xml" changes the two occurrences of "red" to "rouge" in the Widget. The commands "-s finely -t description products.xml" or "-s finely products.xml" will delete the one occurrence of the word "finely".

The starter file contains the routine code to do the I/O, read and write the XML and parse the command line arguments, so you can concentrate on traversing the DOM to perform the search/replace.

- See the JAXP docs (linked off the course page). In particular, most of our operations can be done off the Node class: getNodeType(), getNodeName(), getChildNodes(), getNodeValue(), setNodeValue().

- For nodes of type TEXT_NODE, the program should use getNodeValue()/setNodeValue() to get and set the text. Use the String class for the search/replace itself. Rather than change the text, you construct a new string with the replacement text spliced in.

- Otherwise, for nodes of type ELEMENT_NODE, the algorithm should recur over the child nodes. The algorithm will need to check if the getNodeName() of the node is equal to the -t tag to control if search/replace is active or not. We recommend using a boolean replaceActive parameter to communicate if search/replace is active from one call to the next.

- The starter code reads the command line arguments and sets up ivars "search" "replace" and "tag" for you -- the recursion can just look at these ivars.

Logistics

In later versions of Java, the XML libraries will be a standard part of the Java environment, but for new we have to add the libraries manually. We'll use the Sun/Apache XML libraries which are in the files /usr/class/cs108/jar/jaxp.jar and /usr/class/cs108/jar/crimson.jar. On Unix, add the following definition to your .cshrc file which adds the .jar files to your CLASSPATH. The "." at the end of the CLASSPATH means to also search the current directory, which is what you want so it can find your classes. The setenv line is one long line of text, and it ends in a dot. The colon (:) separates the entries in the CLASSPATH.

```
setenv CLASSPATH /usr/class/cs108/jar/jaxp.jar:/usr/class/cs108/jar/crimson.jar:.
```

Use the "source" command to read your .csrhc file, and then you can check your work with the "echo" command...

```
elaine17:~> source .cshrc
elaine17:~> echo $CLASSPATH
/usr/class/cs108/jar/jaxp.jar:/usr/class/cs108/jar/crimson.jar:.
```

For Metrowerks or other environments, copy the .jar files to your working directory and add them to your project. You also need to make the .jar files available at runtime. Probably the easiest way to do this is to bring up the project inspector window, and select the "merge into output" option for the two .jar files.