```xml
<?xml version="1.0" encoding="utf-8"?>

<!-- ========================================================================
     Buildfile for Project 1, version 8.0
     Computer Science E-259

     This buildfile "shipped" in the root of the following hierarchy.

     project1-8.0/
       docs/
         cscie259/
           project1/
             mf/
       meta/
       samples/
         xml/
       src/
         cscie259/
           project1/
             mf/

     To compile the code explored in questions 11 through 21,
     execute ‘ant compile-Tester‘ from within project1-8.0/.

     To compile the code explored in question 22, execute
     ‘ant compile-AttributeConverter‘ from within project1-8.0/.

     To compile both simultaneously, execute ‘ant compile‘ or ‘ant‘
     from within project1-8.0/.

     To generate Javadoc for your code (in project1-8.0/docs/),
     execute ‘ant javadoc‘ from within project1-8.0/.

     To publish Javadoc for your code at
     http://www.people.fas.harvard.edu/˜username/cscie259/javadoc/project1-8.0/,
     where username is your FAS username, execute ‘ant publish-javadoc‘
     from within project1-8.0/.

     To delete your build/ directory along with its contents,
     execute ‘ant clean‘ from within project1-8.0/.
======================================================================== -->

<project name="project1" default="compile" basedir=".">

    <description>Project 1</description>

    <!-- set global properties for this build -->
    <property name="build" location="build"/>
    <property name="docs" location="docs"/>
    <property name="meta" location="meta"/>
    <property name="src" location="src"/>

    <!-- init -->
    <target name="init">

        <!-- set the standard DSTAMP, TSTAMP, and TODAY properties -->
        <!-- according to the default formats                     -->
        <tstamp/>

        <!-- Create the build directory structure used by compile -->
        <mkdir dir="${build}"/>

    </target>

    <!-- compile-Tester -->
    <target name="compile-Tester"
            depends="init"
            description="compile cscie259.project1.mf.*">
        <javac srcdir="${src}"
               debug="true"
               destdir="${build}"
               fork="true"
               includes="cscie259/project1/mf/*.java"
               listfiles="true"/>
    </target>

    <!-- compile-AttributeConverter -->
    <target name="compile-AttributeConverter"
            depends="init"
            description="compile cscie259.project1.AttributeConverter">
        <javac srcdir="${src}"
               debug="true"
               destdir="${build}"
               fork="true"
               includes="cscie259/project1/*.java"
               listfiles="true"/>
    </target>

    <!-- compile -->
    <target name="compile"
            depends="compile-Tester,compile-AttributeConverter"
            description="compile everything"/>

    <!-- javadoc -->
    <target name="javadoc"
            description="generate Javadoc">
        <delete includeemptydirs="true">
            <fileset dir="${docs}" includes="**/*"/>
        </delete>
        <javadoc packagenames="cscie259.project1.*"
                 author="true"
                 destdir="${docs}"
                 header="Project 1"
                 nodeprecated="true"
                 protected="true"
                 sourcepath="${src}"
                 version="true"
                 windowtitle="Project 1"/>
    </target>

    <!-- publish-javadoc -->
    <target name="publish-javadoc"
            depends="javadoc"
            description="publish Javadoc">
        <copy todir="${user.home}/public_html/cscie259/javadoc/project1-8.0">
             <fileset dir="${docs}"/>
        </copy>
        <chmod dir="${user.home}/public_html/cscie259"
               includes="**/*"
               parallel="false"
               perm="a+rX"
               type="both"/>
    </target>

    <!-- clean -->
    <target name="clean"
            description="remove build directory">
        <delete dir="${build}"/>
    </target>

</project>
```

```java
package cscie259.project1;

import org.apache.xml.serialize.OutputFormat;
import org.apache.xml.serialize.XMLSerializer;


/**
 * A program for converting elements’ attributes to child elements.
 *
 * You MAY modify this file.
 *
 * @author  Computer Science E-259
 * @version 8.0
 *
 * @author  YOUR NAME GOES HERE
 **/
public class AttributeConverter
{
    /**
     * Main entry point to program.
     *
     * @param argv [0] - filename
     */
    public static void main(String[] argv)
    {
        // grab filename from command line
        if (argv.length != 1)
        {
            System.err.println(
                "usage: java " + "cscie259.project1.AttributeConverter "
                + "filename");
            System.exit(1);
        }
        String filename = argv[0];

        // create a serializer with which to pretty print our output
        XMLSerializer serializer = new XMLSerializer(
                System.out,
                new OutputFormat("XML", "UTF-8", true));

        // TODO
    }
}
```

```java
package cscie259.project1.mf;


/**
 * A simplified version of org.xml.sax.helpers.AttributesImpl.
 *
 * An Attributes object stores zero or more attributes.
 *
 * You MAY modify this file to whatever extent you see fit, provided you do
 * not change the declarations of addAttribute, getLength, getName, or
 * getValue.
 *
 * @author  Computer Science E-259
 * @version 8.0
 *
 * @author  YOUR NAME GOES HERE
 **/
public class Attributes
{
    /**
     * Adds a new attribute (i.e., name/value pair) to the collection.
     *
     * @param name  new attribute's name
     * @param value new attribute's value
     */
    public void addAttribute(String name, String value)
    {
        // TODO
        return;
    }


    /**
     * Return the number of attributes in the list.
     *
     * @return the number of attributes in the list
     */
    public int getLength()
    {
        // TODO
        return 0;
    }


    /**
     * Return an attribute's name by index.
     *
     * @param index the attribute's index (zero-based).
     *
     * @return the attribute's name if available else null if the
     * attribute's name is not available or the index is out of range
     */
    public String getName(int index)
    {
        // TODO
        return null;
    }


    /**
     * Return an attribute's value by index.
     *
     * @param index the attribute's index (zero-based)
     *
     * @return the attribute's value or null if the index is out of range
     */
    public String getValue(int index)
    {
        // TODO
        return null;
    }
}
```

```java
package cscie259.project1.mf;


/**
 * A simplified, non-interface version of org.w3c.dom.Attr.
 *
 * You MAY modify this file to whatever extent you see fit,
 * provided you retain the current declarations and definitions of,
 * at least, getNodeType and appendChild.
 *
 * @author  Computer Science E-259
 * @version 8.0
 *
 * @author  YOUR NAME GOES HERE
 **/
public class Attr extends Node
{
    /**
     * Sets node’s name and value.
     *
     * @param name  name for new attribute
     * @param value value for new attribute
     */
    Attr(String name, String value)
    {
        setNodeName(name);
        setNodeValue(value);
    }


    /**
     * Returns code (Node.ATTRIBUTE_NODE) signifying this node’s type.
     *
     * @return Node.ATTRIBUTE_NODE
     */
    public int getNodeType()
    {
        return Node.ATTRIBUTE_NODE;
    }


    /**
     * Throws a RuntimeException, since attributes cannot have children.
     *
     * @param newChild node to be added as a child of this node
     */
    public void appendChild(Node newChild)
    {
        throw new RuntimeException("Error: attributes cannot have children");
    }
}
```

```java
package cscie259.project1.mf;


/**
 * A simplified version of org.xml.sax.ContentHandler.
 *
 * Essentially, any class that implements this interface
 * can "handle the content" encountered by an XML parser.
 *
 * You MAY NOT modify this file.
 *
 * @author  Computer Science E-259
 * @version 8.0
 **/
public interface ContentHandler
{
    /**
     * Should be called immediately after a chunk of
     * character data is parsed.
     *
     * @param content parsed character data
     */
    void characters(String content);


    /**
     * Should be called immediately after an XML document is parsed.
     */
    void endDocument();


    /**
     * Should be called immediately after an end tag is parsed.
     *
     * @param name closed element’s name
     */
    void endElement(String name);


    /**
     * Should be called immediately before an XML document is parsed.
     */
    void startDocument();


    /**
     * Should be called immediately after a start tag is parsed.
     *
     * @param name opened element’s name.
     * @param atts list of the opened element’s attributes
     */
    void startElement(String name, Attributes atts);
}
```

```java
package cscie259.project1.mf;


/**
 * A simplified version of org.xml.sax.helpers.DefaultHandler.
 *
 * Essentially, this class does nothing with the content
 * encountered by an XML parser.  It exists to facilitate
 * the development of more useful ContentHandlers by providing,
 * quite simply, a default implementation of ContentHandler’s
 * methods.
 *
 * You MAY NOT modify this file.
 *
 * @author  Computer Science E-259
 * @version 8.0
 **/
public class DefaultHandler implements ContentHandler, ErrorHandler
{
    /**
     * Should be called immediately after a chunk of
     * character data is parsed.
     *
     * @param content   parsed character data
     */
    public void characters(String content) {}


    /**
     * Should be called immediately before an XML document is parsed.
     */
    public void startDocument() {}


    /**
     * Should be called immediately after a start tag is parsed.
     *
     * @param name  The opened element’s name.
     * @param atts  A list of the opened element’s attributes.
     */
    public void startElement(String name, Attributes atts) {}


    /**
     * Should be called immediately after an XML document is parsed.
     */
    public void endDocument() {}


    /**
     * Should be called immediately after an end tag is parsed.
     *
     * @param name closed element’s name
     */
    public void endElement(String name) {}


    /**
     * Should be called immediately after a parsing error is encountered.
     *
     * @param exception exception related to the error
     */
    public void fatalError(Exception exception) {}
}
```

```java
package cscie259.project1.mf;

import java.util.List;


/**
 * A simplified, non-interface version of org.w3c.dom.Document.
 *
 * An object of this class represents a DOM’s topmost node.
 *
 * You MAY NOT modify this file.
 *
 * @author  Computer Science E-259
 * @version 8.0
 **/
public class Document extends Node
{
    /**
     * Returns code (Node.DOCUMENT_NODE) signifying this node’s type.
     *
     * @return Node.DOCUMENT_NODE
     */
    public int getNodeType()
    {
        return Node.DOCUMENT_NODE;
    }


    /**
     * Returns child node that is the root element of the document.
     *
     * @return child node that is the root element of the document
     */
    public Element getDocumentElement()
    {
        // storage for node
        Element elt;

        // attempt to retrieve root element
        List children = getChildNodes();
        elt = (children != null) ? (Element) children.get(0) : null;

        // return node, if any
        return elt;
    }
}
```

```java
package cscie259.project1.mf;


/**
 * A simplified version of org.apache.xml.utils.DOMBuilder.
 *
 * A DOMBuilder is a ContentHandler that builds a DOM out of
 * SAX events.
 *
 * You MAY modify this file to whatever extent you see fit.
 * However, you MUST complete the implementation
 * of getDocument so that it returns a node of type
 * DOCUMENT_NODE whose descendants represent the contents
 * encountered by the XML parser.  Those descendants should be of
 * type ELEMENT_NODE, ATTRIBUTE_NODE, and/or TEXT_NODE.  And, clearly,
 * you MUST augment this class’s implementation so that it actually
 * handles SAX events and builds a DOM.
 *
 * @author  Computer Science E-259
 * @version 8.0
 *
 * @author  YOUR NAME GOES HERE
 **/
public class DOMBuilder extends DefaultHandler
{
    /**
     * The DOM’s topmost node.
     */
    private Document doc_;


    /**
     * Returns document’s topmost node (i.e., its sole Document node).
     *
     * @return document’s topmost node
     */
    public Document getDocument()
    {
        return doc_;
    }


    // TODO
}
```

```
package cscie259.project1.mf;

import java.util.Iterator;


/**
 * A class whose sole purpose in life is to walk your DOM.
 *
 * You MAY modify this file so that it handles your
 * implementation of attributes.
 *
 * @author  Computer Science E-259
 * @version 8.0
 *
 * @author  YOUR NAME GOES HERE
 **/
public abstract class DOMWalker
{
    /**
     * Initiates a walk on given document, passing SAX events to handler.
     *
     * @param doc     document’s topmost node
     * @param handler DefaultHandler for SAX events
     */
    public static void walk(Document doc, DefaultHandler handler)
    {
        handler.startDocument();
        visit(doc.getDocumentElement(), handler);
        handler.endDocument();
    }


    /**
     * Recursively visits each node in the DOM, passing SAX events to handler.
     * You MUST complete the implementation of this method so that it
     * handles your implementation of attributes.
     *
     * @param cur     node currently being visited
     * @param handler DefaultHandler for SAX events
     */
    private static void visit(Node cur, DefaultHandler handler)
    {
        switch (cur.getNodeType())
        {
            case Node.TEXT_NODE:
                handler.characters(cur.getNodeValue());

                break;

            case Node.ELEMENT_NODE:

                // PROVIDE SUPPORT FOR YOUR IMPLEMENTATION OF
                // ATTRIBUTES BELOW; IN OTHER WORDS, REPLACE
                // null BELOW WITH A REFERENCE TO AN Attributes OBJECT
                // STORING THE CURRENT ELEMENT’S COLLECTION
                // OF ATTRIBUTES
                handler.startElement(cur.getNodeName(), null);

                Iterator iter = cur.getChildNodes().iterator();

                while (iter.hasNext())
                    visit((Node) iter.next(), handler);

                handler.endElement(cur.getNodeName());

                break;

            default:
                throw new RuntimeException(
                    "Type " + cur.getNodeType() + " not handled");
        }
    }
}
```

```java
package cscie259.project1.mf;


/**
 * A simplified, non-interface version of org.w3c.dom.Element.
 *
 * You MAY modify this file, provided you do not change the
 * declarations or definitions of the constructor and getNodeType.
 *
 * @author  Computer Science E-259
 * @version 8.0
 *
 * @author  YOUR NAME GOES HERE
 **/
public class Element extends Node
{
    /**
     * Sets node’s name.
     *
     * @param   name    name for new element
     */
    public Element(String name)
    {
        setNodeName(name);
    }


    /**
     * Returns code (Node.ELEMENT_NODE) signifying this node’s type.
     *
     * @return Node.ELEMENT_NODE
     */
    public int getNodeType()
    {
        return Node.ELEMENT_NODE;
    }
}
```

```java
package cscie259.project1.mf;


/**
 * A simplified version of org.xml.sax.ErrorHandler.
 *
 * Essentially, any class that implements this interface
 * can "handle the errors" encountered by an XML parser.
 *
 * You MAY NOT modify this file.
 *
 * @author  Computer Science E-259
 * @version 8.0
 **/
public interface ErrorHandler
{
    /**
     * Should be called immediately after a parsing error is encountered.
     *
     * @param exception exception related to the error
     */
    void fatalError(Exception exception);
}
```

```java
package cscie259.project1.mf;

import java.util.LinkedList;
import java.util.List;


/**
 * A simplified version of org.w3c.dom.Node.
 *
 * You MAY NOT modify this file.
 *
 * @author  Computer Science E-259
 * @version 8.0
 **/
public abstract class Node
{
    /**
     * Child elements are to be stored in a List.
     */
    private List<Node> children_ = new LinkedList<Node>();


    /**
     * An Attr or Element node’s name.
     */
    private String name_ = null;


    /**
     * A reference to this node’s parent, if any.
     */
    private Node parent_ = null;


    /**
     * An Attr or Text node’s value.
     */
    private String value_ = null;


    /**
     * Short code identifying the type of a Document node.
     */
    public static final int DOCUMENT_NODE = 0;


    /**
     * Short code identifying the type of an Element node.
     */
    public static final int ELEMENT_NODE = 1;


    /**
     * Short code identifying the type of a Attr node.
     */
    public static final int ATTRIBUTE_NODE = 2;


    /**
     * Short code identifying the type of a Text node.
     */
    public static final int TEXT_NODE = 3;


    /**
     * Appends new child to node.
     *
     * @param newChild  child to be added
     */
    public void appendChild(Node newChild)
    {
        newChild.parent_ = this;
        children_.add(newChild);
    }


    /**
     * Gets node’s children.
     *
     * @return  List of node’s children.
     */
    public List getChildNodes()
    {
        return children_;
    }


    /**
     * Gets node’s name.
     *
     * @return  node’s name.
     */
    public String getNodeName()
    {
        return name_;
    }


    /**
     * Returns code signifying this node’s type.
     *
     * @return  node’s type
     */
    public abstract int getNodeType();


    /**
     * Gets node’s value.
     *
     * @return  node’s value.
     */
    public String getNodeValue()
    {
        return value_;
    }


    /**
     * Returns node’s parent.
     *
     * @return  node’s parent
     */
    public Node getParentNode()
    {
        return parent_;
    }


    /**
     * Sets node’s name.
     *
     * @param   name    node’s name
     */
    public void setNodeName(String name)
    {
        name_ = name;
    }


    /**
     * Sets node’s name.
     *
     * @param   value    node’s value
     */
    public void setNodeValue(String value)
    {
        value_ = value;
    }
}
```

```java
package cscie259.project1.mf;

import java.io.BufferedWriter;
import java.io.OutputStreamWriter;


/**
 * A driver for testing your code.
 *
 * You MAY modify this file to whatever extent you see fit.
 *
 * @author  Computer Science E-259
 * @version 8.0
 *
 * @author  YOUR NAME GOES HERE
 **/
public class Tester
{
    /**
     * Default constructor is private so that this utility
     * class cannot be instantiated.
     */
    private Tester() {}


    /**
     * Main driver.  Expects two command-line arguments: the name of the file
     * to be parsed, followed by a test number.  Valid test numbers, at
     * present, are 1 and 2.  1 invokes testing of XMLParser (and
     * related classes) and XMLSerializer.  2 invokes testing of
     * XMLParser (and related classes), DOMBuilder,
     * DOMWalker, and XMLSerializer.
     *
     * @param argv [0] - filename, [1] - testnumber
     */
    public static void main(String[] argv)
    {
        // enforce proper usage
        if (argv.length < 2)
        {
            System.out.println(
                "usage: java cscie259.project1.mf.Tester "
                + "filename testnumber");

            return;
        }

        // execute requested test case
        switch (Integer.parseInt(argv[1]))
        {
            // Test XMLParser (and related classes) and
            // XMLSerializer
            case 1:

                // instantiate a parser
                XMLParser p1 = new XMLParser();

                // instantiate a BufferedWriter for System.out
                BufferedWriter bw1 = new BufferedWriter(
                        new OutputStreamWriter(System.out));

                // by default, don’t ask XMLSerializer to pretty-print;
                // rely on input file’s own whitespace, if any
                XMLSerializer s1 = new XMLSerializer(bw1, false);

                // try to parse the file, serializing in the process!
                p1.parse(argv[0], s1);

                break;

            // Test XMLParser (and related classes), DOMBuilder,
            // DOMWalker, and XMLSerializer
            case 2:

                // instantiate a parser
                XMLParser p2 = new XMLParser();

                // instantiate a DOMBuilder
                DOMBuilder db = new DOMBuilder();

                // try to parse the file, building a DOM in the process!
                p2.parse(argv[0], db);

                // grab the DOM’s topmost node
                Document doc = db.getDocument();

                // instantiate a BufferedWriter for System.out
                BufferedWriter bw2 = new BufferedWriter(
                        new OutputStreamWriter(System.out));

                // by default, don’t ask XMLSerializer to pretty-print;
                // rely on input file’s own whitespace, if any
                XMLSerializer s2 = new XMLSerializer(bw2, false);

                // walk the DOM, serializing in the process!
                DOMWalker.walk(doc, s2);

                break;

            default:
                System.out.println("Error: testnumber must be 1 or 2");
        }
    }
}
```

```java
package cscie259.project1.mf;


/**
 * A simplified, non-interface version of org.w3c.dom.Text.
 *
 * You MAY NOT modify this file.
 *
 * @author  Computer Science E-259
 * @version 8.0
 **/
public class Text extends Node
{
    /**
     * Sets node’s value.
     *
     * @param value value for new text node
     */
    public Text(String value)
    {
        setNodeValue(value);
    }


    /**
     * Throws a RuntimeException, since text nodes cannot have children.
     *
     * @param newChild node to be added as a child of this node
     */
    public void appendChild(Node newChild)
    {
        throw new RuntimeException("Error: text nodes cannot have children");
    }


    /**
     * Returns code (Node.TEXT) signifying this node’s type.
     *
     * @return Node.TEXT
     */
    public int getNodeType()
    {
        return Node.TEXT_NODE;
    }
}
```

```java
package cscie259.project1.mf;

import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;


/**
 * A simplified XML parser.  In essence, this class supports a subset
 * of the functionality collectively offered by javax.xml.parsers.SAXParser
 * and javax.xml.parsers.DocumentBuilder.
 *
 * You MAY modify this file.
 *
 * @author  Computer Science E-259
 * @version 8.0
 *
 * @author  YOUR NAME GOES HERE
 **/
public class XMLParser
{
    /**
     * Storage for input file’s contents.
     */
    private String data_;


    /**
     * A reference to the currently registered DefaultHandler.
     */
    private DefaultHandler handler_;


    /**
     * Index of our current location in input file’s contents.
     */
    private int index_ = 0;


    /**
     * Returns true if the next characters in the stream are the beginning
     * of an element’s end tag.
     *
     * @return true iff next characters in the stream are the beginning
     * of an element’s end tag
     */
    protected boolean isEndTag()
    {
        return (data_.charAt(index_) == ’<’)
        && (data_.charAt(index_ + 1) == ’/’);
    }


    /**
     * Returns true if the next character in the stream is the beginning
     * of an element’s start tag.
     *
     * @return true iff next character in the stream is the beginning
     * of an element’s start tag
     */
    protected boolean isStartTag()
    {
        return data_.charAt(index_) == ’<’;
    }


    /**
     * Parses the specified file, if possible, passing SAX events
     * to given handler.
     *
     * @param filename name of file whose contents are to be parsed
     * @param handler  DefaultHandler for SAX events
     */
    public void parse(String filename, DefaultHandler handler)
    {
        // initialize to clean up from any previous parse
        data_ = "";
        index_ = 0;
        handler_ = handler;

        // attempt to open file and read contents into local storage
        try
        {
            File f = new File(filename);
            int filesize = (int) f.length();
            byte[] filebytes = new byte[filesize];
            DataInputStream in = new DataInputStream(new FileInputStream(f));
            in.readFully(filebytes);
            in.close();
            data_ = new String(filebytes);
        }
        catch (IOException E)
        {
            handler_.fatalError(new Exception("Error: could not read file"));
            return;
        }

        // parse the document; hopefully there’s a root element!
        handler_.startDocument();
        readElement();
        handler_.endDocument();
    }


    /**
     * Parses an element and its content.
     */
    protected void readElement()
    {
        if (!isStartTag())
        {
            handler_.fatalError(new RuntimeException("Error: expecting " +
                                                      "start of element"));
            return;
        }

        // parse end tag
        String name = readStartTag();

        // keep reading in more elements and text until an end tag
        // is encountered
        while (!isEndTag())
        {
            if (isStartTag())
                readElement();
            else
                readText();
        }

        // parse end tag, ensuring it matches most current start tag
        readEndTag(name);
    }


    /**
     * Parses an end tag, ensuring its name matches currently opened
     * element's name.
     *
     * @param checkName currently opened element's name with which
     * end tag should be compared
     */
    protected void readEndTag(String checkName)
    {
        // start name from scratch
        String name = "";

        // read starting <
        index_++;

        // read /
        index_++;

        // read name
        while (data_.charAt(index_) != '>')
        {
            name += data_.charAt(index_);
            index_++;
        }

        // read ending >
        index_++;

        // ensure content is well-formed
        if (!checkName.equals(name))
        {
            handler_.fatalError(new RuntimeException("Error: expecting " +
                                                     "closing tag for " +
                                                     checkName));
            return;
        }

        // pass this SAX event to handler
        handler_.endElement(name);
    }


    /**
     * Parses a start tag, returning opened element's name.
     *
     * @return name of element
     */
    protected String readStartTag()
    {
        // start name from scratch
        String name = "";

        // Read starting <
        index_++;

        // Read name
        while (data_.charAt(index_) != '>')
        {
            name += data_.charAt(index_);
            index_++;
        }

        // Read ending >
        index_++;

        // pass this SAX event to handler;
        // you MUST replace null below with a reference to
        // this element's Attributes object
        handler_.startElement(name, null);

        // return this element's name, for later comparision
        // with an end tag
        return name;
    }


    /**
     * Parses character data.
     */
    protected void readText()
    {
        // start character data from scratch
        String content = "";

        // accumulate characters until next tag
        while (data_.charAt(index_) != '<')
        {
            content += data_.charAt(index_);
            index_++;
        }

        // pass this SAX event to handler
        handler_.characters(content);
    }
}
```

```java
package cscie259.project1.mf;

import java.io.BufferedWriter;
import java.io.IOException;


/**
 * A ContentHandler for SAX events that serializes (to an output stream)
 * the events back into XML.  Essentially, a simplified version of
 * org.apache.xml.serialize.XMLSerializer.
 *
 * You MAY modify this file to provide support for the
 * serialization of attributes.
 *
 * @author  Computer Science E-259
 * @version 8.0
 *
 * @author  YOUR NAME GOES HERE
 **/
public class XMLSerializer extends DefaultHandler
{
    /**
     * The output stream to which we are serializing.
     */
    private BufferedWriter out_;


    /**
     * A flag signifying whether output should be indented (i.e.,
     * pretty-printed).
     */
    private boolean prettyPrint_;


    /**
     * The current level of indentation, if applicable.
     */
    private int indentLevel_;


    /**
     * Configures XMLSerializer with given BufferedWriter and for
     * pretty-printedness, if applicable.
     *
     * @param writer      BufferedWriter for serialization
     * @param prettyPrint flag indicating whether to pretty-print
     */
    public XMLSerializer(BufferedWriter writer, boolean prettyPrint)
    {
        out_ = writer;
        prettyPrint_ = prettyPrint;
        indentLevel_ = 0;
    }


    /**
     * Prints out character data, pretty-printed if applicable.
     *
     * @param content  character data
     */
    public void characters(String content)
    {
        try
        {
            // pretty-print if applicable
            if (prettyPrint_)
                indent();

            // write character data
            out_.write(content, 0, content.length());

            // pretty-print if applicable
            if (prettyPrint_)
                out_.newLine();
        }
        catch (IOException E)
        {
            throw new RuntimeException("Error: I/O error " + E.getMessage());
        }
    }


    /**
     * Closes the output stream.
     */
    public void endDocument()
    {
        try
        {
            out_.close();
        }
        catch (IOException E)
        {
            throw new RuntimeException("Error: I/O error " + E.getMessage());
        }
    }


    /**
     * Prints out the end element tag, pretty-printed if applicable, and
     * updates the current level of indentation.
     *
     * @param name name of element
     */
    public void endElement(String name)
    {
        try
        {
            // pretty-print if applicable
            if (prettyPrint_)
            {
                indentLevel_--;
                indent();
            }

            // write </
            out_.write('<');
            out_.write('/');

            // write element's name
            out_.write(name, 0, name.length());

            // write >
            out_.write('>');

            // pretty-print if applicable
            if (prettyPrint_)
                out_.newLine();
        }
        catch (IOException E)
        {
            throw new RuntimeException("Error: I/O error " + E.getMessage());
        }
    }


    /**
     * Prints the number of indents currently appropriate.
     *
     * @throws IOException run-time failure of writing operation
     */
    private void indent() throws IOException
    {
        for (int i = 0; i < indentLevel_; i++)
        {
            out_.write("    ");
        }
    }


    /**
     * Prints out the start element tag, pretty-printed if applicable, and
     * updates the current level of indentation.
     *
     * @param name name of element
     * @param atts element’s collection of attributes
     */
    public void startElement(String name, Attributes atts)
    {
        try
        {
            // pretty-print if applicable
            if (prettyPrint_)
                indent();

            // write <
            out_.write(’<’);

            // write element’s name
            out_.write(name, 0, name.length());

            // write element’s attributes, if any
            // TODO

            // write >
            out_.write(’>’);

            // pretty-print if applicable
            if (prettyPrint_)
            {
                out_.newLine();
                indentLevel_++;
            }
        }
        catch (IOException E)
        {
            throw new RuntimeException("Error: I/O error " + E.getMessage());
        }
    }
}
```