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

<!-- ======================================================================
     Buildfile for Project 1, version 7.0
     Computer Science E-259

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

     project1-7.0/
       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 project1-7.0/.

     To compile the code explored in question 22, execute
     ‘ant compile-AttributeConverter‘ from within project1-7.0/.

     To compile both simultaneously, execute ‘ant compile‘ or ‘ant‘
     from within project1-7.0/.

     To generate Javadoc for your code (in project1-7.0/docs/),
     execute ‘ant javadoc‘ from within project1-7.0/.

     To publish Javadoc for your code at
     http://www.people.fas.harvard.edu/~username/cscie259/javadoc/project1-7.0/,
     where username is your FAS username, execute ‘ant publish-javadoc‘
     from within project1-7.0/.

     To delete your build/ directory along with its contents,
     execute ‘ant clean‘ from within project1-7.0/.
====================================================================== -->

<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">
        <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="${user.home}/public_html/cscie259/javadoc/project1-7.0">
             <fileset dir="${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">
```

```
125:         <delete dir="${build}"/>
126:     </target>
127:
128: </project>
```

```
 1: package cscie259.project1;
 2:
 3: import org.apache.xml.serialize.OutputFormat;
 4: import org.apache.xml.serialize.XMLSerializer;
 5:
 6:
 7: /**
 8:  * A program for converting elements' attributes to child elements.
 9:  *
10:  * You MAY modify this file.
11:  *
12:  * @author  Computer Science E-259
13:  * @version 7.0
14:  *
15:  * @author  YOUR NAME GOES HERE
16:  **/
17: public abstract class AttributeConverter
18: {
19:     /**
20:      * Main entry point to program.
21:      *
22:      * @param argv [0] - filename
23:      */
24:     public static void main(String[] argv)
25:     {
26:         // grab filename from command line
27:         if (argv.length != 1)
28:         {
29:             System.err.println(
30:                 "usage: java " + "cscie259.project1.AttributeConverter "
31:                 + "filename");
32:             System.exit(1);
33:         }
34:         String filename = argv[0];
35:
36:         // create a serializer with which to pretty print our output
37:         XMLSerializer serializer = new XMLSerializer(
38:                 System.out,
39:                 new OutputFormat("XML", "UTF-8", true));
40:
41:         // TODO
42:     }
43: }
```

```
 1: package cscie259.project1.mf;
 2:
 3:
 4: /**
 5:  * A simplified version of org.xml.sax.helpers.AttributesImpl.
 6:  *
 7:  * An Attributes object stores zero or more attributes.
 8:  *
 9:  * You MAY modify this file to whatever extent you see fit, provided you do
10:  * not change the declarations of addAttribute, getLength, getName, or
11:  * getValue.
12:  *
13:  * @author  Computer Science E-259
14:  * @version 7.0
15:  *
16:  * @author  YOUR NAME GOES HERE
17:  **/
18: public class Attributes
19: {
20:     /**
21:      * Adds a new attribute (i.e., name/value pair) to the collection.
22:      *
23:      * @param name  new attribute's name
24:      * @param value new attribute's value
25:      */
26:     public void addAttribute(String name, String value)
27:     {
28:         // TODO
29:         return;
30:     }
31:
32:
33:     /**
34:      * Return the number of attributes in the list.
35:      *
36:      * @return the number of attributes in the list
37:      */
38:     public int getLength()
39:     {
40:         // TODO
41:         return 0;
42:     }
43:
44:
45:     /**
46:      * Return an attribute's name by index.
47:      *
48:      * @param index the attribute's index (zero-based).
49:      *
50:      * @return the attribute's name if available else null if the
51:      * attribute's name is not available or the index is out of range
52:      */
53:     public String getName(int index)
54:     {
55:         // TODO
56:         return null;
57:     }
58:
59:
60:     /**
61:      * Return an attribute's value by index.
62:      *
63:      * @param index the attribute's index (zero-based)
64:      *
65:      * @return the attribute's value or null if the index is out of range
66:      */
67:     public String getValue(int index)
68:     {
69:         // TODO
70:         return null;
71:     }
72: }
```

```
 1: package cscie259.project1.mf;
 2:
 3:
 4: /**
 5:  * A simplified, non-interface version of org.w3c.dom.Attr.
 6:  *
 7:  * You MAY modify this file to whatever extent you see fit,
 8:  * provided you retain the current declarations and definitions of,
 9:  * at least, getNodeType and appendChild.
10:  *
11:  * @author  Computer Science E-259
12:  * @version 7.0
13:  *
14:  * @author  YOUR NAME GOES HERE
15:  **/
16: public class Attr extends Node
17: {
18:     /**
19:      * Sets node's name and value.
20:      *
21:      * @param name  name for new attribute
22:      * @param value value for new attribute
23:      */
24:     Attr(String name, String value)
25:     {
26:         setNodeName(name);
27:         setNodeValue(value);
28:     }
29:
30:
31:     /**
32:      * Returns code (Node.ATTRIBUTE_NODE) signifying this node's type.
33:      *
34:      * @return Node.ATTRIBUTE_NODE
35:      */
36:     public int getNodeType()
37:     {
38:         return Node.ATTRIBUTE_NODE;
39:     }
40:
41:
42:     /**
43:      * Throws a RuntimeException, since attributes cannot have children.
44:      *
45:      * @param newChild node to be added as a child of this node
46:      */
47:     public void appendChild(Node newChild)
48:     {
49:         throw new RuntimeException("Error: attributes cannot have children");
50:     }
51: }
```

```
 1: package cscie259.project1.mf;
 2:
 3:
 4: /**
 5:  * A simplified version of org.xml.sax.ContentHandler.
 6:  *
 7:  * Essentially, any class that implements this interface
 8:  * can "handle the content" encountered by an XML parser.
 9:  *
10:  * You MAY NOT modify this file.
11:  *
12:  * @author  Computer Science E-259
13:  * @version 7.0
14:  **/
15: public interface ContentHandler
16: {
17:     /**
18:      * Should be called immediately after a chunk of
19:      * character data is parsed.
20:      *
21:      * @param content parsed character data
22:      */
23:     void characters(String content);
24:
25:
26:     /**
27:      * Should be called immediately after an XML document is parsed.
28:      */
29:     void endDocument();
30:
31:
32:     /**
33:      * Should be called immediately after an end tag is parsed.
34:      *
35:      * @param name closed element's name
36:      */
37:     void endElement(String name);
38:
39:
40:     /**
41:      * Should be called immediately before an XML document is parsed.
42:      */
43:     void startDocument();
44:
45:
46:     /**
47:      * Should be called immediately after a start tag is parsed.
48:      *
49:      * @param name opened element's name.
50:      * @param atts list of the opened element's attributes
51:      */
52:     void startElement(String name, Attributes atts);
53: }
```

```
 1: package cscie259.project1.mf;
 2:
 3:
 4: /**
 5:  * A simplified version of org.xml.sax.helpers.DefaultHandler.
 6:  *
 7:  * Essentially, this class does nothing with the content
 8:  * encountered by an XML parser.  It exists to facilitate
 9:  * the development of more useful ContentHandlers by providing,
10:  * quite simply, a default implementation of ContentHandler's
11:  * methods.
12:  *
13:  * You MAY NOT modify this file.
14:  *
15:  * @author  Computer Science E-259
16:  * @version 7.0
17:  **/
18: public class DefaultHandler implements ContentHandler, ErrorHandler
19: {
20:     /**
21:      * Should be called immediately after a chunk of
22:      * character data is parsed.
23:      *
24:      * @param content   parsed character data
25:      */
26:     public void characters(String content) {}
27:
28:
29:     /**
30:      * Should be called immediately before an XML document is parsed.
31:      */
32:     public void startDocument() {}
33:
34:
35:     /**
36:      * Should be called immediately after a start tag is parsed.
37:      *
38:      * @param name  The opened element's name.
39:      * @param atts  A list of the opened element's attributes.
40:      */
41:     public void startElement(String name, Attributes atts) {}
42:
43:
44:     /**
45:      * Should be called immediately after an XML document is parsed.
46:      */
47:     public void endDocument() {}
48:
49:
50:     /**
51:      * Should be called immediately after an end tag is parsed.
52:      *
53:      * @param name closed element's name
54:      */
55:     public void endElement(String name) {}
56:
57:
58:     /**
59:      * Should be called immediately after a parsing error is encountered.
60:      *
61:      * @param exception exception related to the error
62:      */
63:     public void fatalError(Exception exception) {}
64: }
```

```
 1: package cscie259.project1.mf;
 2:
 3: import java.util.List;
 4:
 5:
 6: /**
 7:  * A simplified, non-interface version of org.w3c.dom.Document.
 8:  *
 9:  * An object of this class represents a DOM’s topmost node.
10:  *
11:  * You MAY NOT modify this file.
12:  *
13:  * @author  Computer Science E-259
14:  * @version 7.0
15:  **/
16: public class Document extends Node
17: {
18:     /**
19:      * Returns code (Node.DOCUMENT_NODE) signifying this node’s type.
20:      *
21:      * @return Node.DOCUMENT_NODE
22:      */
23:     public int getNodeType()
24:     {
25:         return Node.DOCUMENT_NODE;
26:     }
27:
28:
29:     /**
30:      * Returns child node that is the root element of the document.
31:      *
32:      * @return child node that is the root element of the document
33:      */
34:     public Element getDocumentElement()
35:     {
36:         // storage for node
37:         Element elt;
38:
39:         // attempt to retrieve root element
40:         List children = getChildNodes();
41:         elt = (children != null) ? (Element) children.get(0) : null;
42:
43:         // return node, if any
44:         return elt;
45:     }
46: }
```

```
 1: package cscie259.project1.mf;
 2:
 3:
 4: /**
 5:  * A simplified version of org.apache.xml.utils.DOMBuilder.
 6:  *
 7:  * A DOMBuilder is a ContentHandler that builds a DOM out of
 8:  * SAX events.
 9:  *
10:  * You MAY modify this file to whatever extent you see fit.
11:  * However, you MUST complete the implementation
12:  * of getDocument so that it returns a node of type
13:  * DOCUMENT_NODE whose descendants represent the contents
14:  * encountered by the XML parser.  Those descendants should be of
15:  * type ELEMENT_NODE, ATTRIBUTE_NODE, and/or TEXT_NODE.  And, clearly,
16:  * you MUST augment this class's implementation so that it actually
17:  * handles SAX events and builds a DOM.
18:  *
19:  * @author  Computer Science E-259
20:  * @version 7.0
21:  *
22:  * @author  YOUR NAME GOES HERE
23:  **/
24: public class DOMBuilder extends DefaultHandler
25: {
26:     /**
27:      * The DOM's topmost node.
28:      */
29:     private Document doc_;
30:
31:
32:     /**
33:      * Returns document's topmost node (i.e., its sole Document node).
34:      *
35:      * @return document's topmost node
36:      */
37:     public Document getDocument()
38:     {
39:         return doc_;
40:     }
41:
42:
43:     // TODO
44: }
```

```
 1: package cscie259.project1.mf;
 2:
 3: import java.util.Iterator;
 4:
 5:
 6: /**
 7:  * A class whose sole purpose in life is to walk your DOM.
 8:  *
 9:  * You MAY modify this file so that it handles your
10:  * implementation of attributes.
11:  *
12:  * @author  Computer Science E-259
13:  * @version 7.0
14:  *
15:  * @author  YOUR NAME GOES HERE
16:  **/
17: public abstract class DOMWalker
18: {
19:     /**
20:      * Initiates a walk on given document, passing SAX events to handler.
21:      *
22:      * @param doc     document's topmost node
23:      * @param handler DefaultHandler for SAX events
24:      */
25:     public static void walk(Document doc, DefaultHandler handler)
26:     {
27:         handler.startDocument();
28:         visit(doc.getDocumentElement(), handler);
29:         handler.endDocument();
30:     }
31:
32:
33:     /**
34:      * Recursively visits each node in the DOM, passing SAX events to handler.
35:      * You MUST complete the implementation of this method so that it
36:      * handles your implementation of attributes.
37:      *
38:      * @param cur     node currently being visited
39:      * @param handler DefaultHandler for SAX events
40:      */
41:     private static void visit(Node cur, DefaultHandler handler)
42:     {
43:         switch (cur.getNodeType())
44:         {
45:             case Node.TEXT_NODE:
46:                 handler.characters(cur.getNodeValue());
47:
48:                 break;
49:
50:             case Node.ELEMENT_NODE:
51:
52:                 // PROVIDE SUPPORT FOR YOUR IMPLEMENTATION OF
53:                 // ATTRIBUTES BELOW; IN OTHER WORDS, REPLACE
54:                 // null BELOW WITH A REFERENCE TO AN Attributes OBJECT
55:                 // STORING THE CURRENT ELEMENT'S COLLECTION
56:                 // OF ATTRIBUTES
57:                 handler.startElement(cur.getNodeName(), null);
58:
59:                 Iterator iter = cur.getChildNodes().iterator();
60:
61:                 while (iter.hasNext())
62:                     visit((Node) iter.next(), handler);
63:
64:                 handler.endElement(cur.getNodeName());
65:
66:                 break;
67:
68:             default:
69:                 throw new RuntimeException(
70:                     "Type " + cur.getNodeType() + " not handled");
71:         }
72:     }
73: }
```

```
 1: package cscie259.project1.mf;
 2:
 3:
 4: /**
 5:  * A simplified, non-interface version of org.w3c.dom.Element.
 6:  *
 7:  * You MAY modify this file, provided you do not change the
 8:  * declarations or definitions of the constructor and getNodeType.
 9:  *
10:  * @author  Computer Science E-259
11:  * @version 7.0
12:  *
13:  * @author  YOUR NAME GOES HERE
14:  **/
15: public class Element extends Node
16: {
17:     /**
18:      * Sets node's name.
19:      *
20:      * @param   name    name for new element
21:      */
22:     public Element(String name)
23:     {
24:         setNodeName(name);
25:     }
26:
27:
28:     /**
29:      * Returns code (Node.ELEMENT_NODE) signifying this node's type.
30:      *
31:      * @return Node.ELEMENT_NODE
32:      */
33:     public int getNodeType()
34:     {
35:         return Node.ELEMENT_NODE;
36:     }
37: }
```

```
 1: package cscie259.project1.mf;
 2:
 3:
 4: /**
 5:  * A simplified version of org.xml.sax.ErrorHandler.
 6:  *
 7:  * Essentially, any class that implements this interface
 8:  * can "handle the errors" encountered by an XML parser.
 9:  *
10:  * You MAY NOT modify this file.
11:  *
12:  * @author  Computer Science E-259
13:  * @version 7.0
14:  **/
15: public interface ErrorHandler
16: {
17:     /**
18:      * Should be called immediately after a parsing error is encountered.
19:      *
20:      * @param exception exception related to the error
21:      */
22:     void fatalError(Exception exception);
23: }
```

```
 1: package cscie259.project1.mf;
 2:
 3: import java.util.LinkedList;
 4: import java.util.List;
 5:
 6:
 7: /**
 8:  * A simplified version of org.w3c.dom.Node.
 9:  *
10:  * You MAY NOT modify this file.
11:  *
12:  * @author  Computer Science E-259
13:  * @version 7.0
14:  **/
15: public abstract class Node
16: {
17:     /**
18:      * Child elements are to be stored in a List.
19:      */
20:     private List<Node> children_ = new LinkedList<Node>();
21:
22:
23:     /**
24:      * An Attr or Element node's name.
25:      */
26:     private String name_ = null;
27:
28:
29:     /**
30:      * A reference to this node's parent, if any.
31:      */
32:     private Node parent_ = null;
33:
34:
35:     /**
36:      * An Attr or Text node's value.
37:      */
38:     private String value_ = null;
39:
40:
41:     /**
42:      * Short code identifying the type of a Document node.
43:      */
44:     public static final int DOCUMENT_NODE = 0;
45:
46:
47:     /**
48:      * Short code identifying the type of an Element node.
49:      */
50:     public static final int ELEMENT_NODE = 1;
51:
52:
53:     /**
54:      * Short code identifying the type of a Attr node.
55:      */
56:     public static final int ATTRIBUTE_NODE = 2;
57:
58:
59:     /**
60:      * Short code identifying the type of a Text node.
61:      */
62:     public static final int TEXT_NODE = 3;
63:
64:
65:     /**
66:      * Appends new child to node.
67:      *
68:      * @param newChild  child to be added
69:      */
70:     public void appendChild(Node newChild)
71:     {
72:         newChild.parent_ = this;
73:         children_.add(newChild);
74:     }
75:
76:
77:     /**
78:      * Gets node's children.
79:      *
80:      * @return  List of node's children.
81:      */
82:     public List getChildNodes()
83:     {
84:         return children_;
85:     }
86:
87:
88:     /**
89:      * Gets node's name.
90:      *
91:      * @return  node's name.
92:      */
93:     public String getNodeName()
94:     {
95:         return name_;
96:     }
97:
98:
99:     /**
100:      * Returns code signifying this node's type.
101:      *
102:      * @return  node's type
103:      */
104:     public abstract int getNodeType();
105:
106:
107:     /**
108:      * Gets node's value.
109:      *
110:      * @return  node's value.
111:      */
112:     public String getNodeValue()
113:     {
114:         return value_;
115:     }
116:
117:
118:     /**
119:      * Returns node's parent.
120:      *
121:      * @return  node's parent
122:      */
123:     public Node getParentNode()
124:     {
```

```
125:         return parent_;
126:     }
127:
128:
129:     /**
130:      * Sets node’s name.
131:      *
132:      * @param   name    node’s name
133:      */
134:     public void setNodeName(String name)
135:     {
136:         name_ = name;
137:     }
138:
139:
140:     /**
141:      * Sets node’s name.
142:      *
143:      * @param   value    node’s value
144:      */
145:     public void setNodeValue(String value)
146:     {
147:         value_ = value;
148:     }
149: }
```

```
 1: package cscie259.project1.mf;
 2:
 3: import java.io.BufferedWriter;
 4: import java.io.OutputStreamWriter;
 5:
 6:
 7: /**
 8:  * A driver for testing your code.
 9:  *
10:  * You MAY modify this file to whatever extent you see fit.
11:  *
12:  * @author  Computer Science E-259
13:  * @version 7.0
14:  *
15:  * @author  YOUR NAME GOES HERE
16:  **/
17: public abstract class Tester
18: {
19:     /**
20:      * Main driver.  Expects two command-line arguments: the name of the file
21:      * to be parsed, followed by a test number.  Valid test numbers, at
22:      * present, are 1 and 2.  1 invokes testing of XMLParser (and
23:      * related classes) and XMLSerializer.  2 invokes testing of
24:      * XMLParser (and related classes), DOMBuilder,
25:      * DOMWalker, and XMLSerializer.
26:      *
27:      * @param argv [0] - filename, [1] - testnumber
28:      */
29:     public static void main(String[] argv)
30:     {
31:         // enforce proper usage
32:         if (argv.length < 2)
33:         {
34:             System.out.println(
35:                 "usage: java cscie259.project1.mf.Tester "
36:                 + "filename testnumber");
37:
38:             return;
39:         }
40:
41:         // execute requested test case
42:         switch (Integer.parseInt(argv[1]))
43:         {
44:             // Test XMLParser (and related classes) and
45:             // XMLSerializer
46:             case 1:
47:
48:                 // instantiate a parser
49:                 XMLParser p1 = new XMLParser();
50:
51:                 // instantiate a BufferedWriter for System.out
52:                 BufferedWriter bw1 = new BufferedWriter(
53:                         new OutputStreamWriter(System.out));
54:
55:                 // by default, don't ask XMLSerializer to pretty-print;
56:                 // rely on input file's own whitespace, if any
57:                 XMLSerializer s1 = new XMLSerializer(bw1, false);
58:
59:                 // try to parse the file, serializing in the process!
60:                 p1.parse(argv[0], s1);
61:
62:                 break;
63:
64:             // Test XMLParser (and related classes), DOMBuilder,
65:             // DOMWalker, and XMLSerializer
66:             case 2:
67:
68:                 // instantiate a parser
69:                 XMLParser p2 = new XMLParser();
70:
71:                 // instantiate a DOMBuilder
72:                 DOMBuilder db = new DOMBuilder();
73:
74:                 // try to parse the file, building a DOM in the process!
75:                 p2.parse(argv[0], db);
76:
77:                 // grab the DOM's topmost node
78:                 Document doc = db.getDocument();
79:
80:                 // instantiate a BufferedWriter for System.out
81:                 BufferedWriter bw2 = new BufferedWriter(
82:                         new OutputStreamWriter(System.out));
83:
84:                 // by default, don't ask XMLSerializer to pretty-print;
85:                 // rely on input file's own whitespace, if any
86:                 XMLSerializer s2 = new XMLSerializer(bw2, false);
87:
88:                 // walk the DOM, serializing in the process!
89:                 DOMWalker.walk(doc, s2);
90:
91:                 break;
92:
93:             default:
94:                 System.out.println("Error: testnumber must be 1 or 2");
95:         }
96:     }
97: }
```

```
 1: package cscie259.project1.mf;
 2:
 3:
 4: /**
 5:  * A simplified, non-interface version of org.w3c.dom.Text.
 6:  *
 7:  * You MAY NOT modify this file.
 8:  *
 9:  * @author  Computer Science E-259
10:  * @version 7.0
11:  **/
12: public class Text extends Node
13: {
14:     /**
15:      * Sets node's value.
16:      *
17:      * @param value value for new text node
18:      */
19:     public Text(String value)
20:     {
21:         setNodeValue(value);
22:     }
23:
24:
25:     /**
26:      * Throws a RuntimeException, since text nodes cannot have children.
27:      *
28:      * @param newChild node to be added as a child of this node
29:      */
30:     public void appendChild(Node newChild)
31:     {
32:         throw new RuntimeException("Error: text nodes cannot have children");
33:     }
34:
35:
36:     /**
37:      * Returns code (Node.TEXT) signifying this node's type.
38:      *
39:      * @return Node.TEXT
40:      */
41:     public int getNodeType()
42:     {
43:         return Node.TEXT_NODE;
44:     }
45: }
```

```
 1: package cscie259.project1.mf;
 2:
 3: import java.io.DataInputStream;
 4: import java.io.File;
 5: import java.io.FileInputStream;
 6: import java.io.IOException;
 7:
 8:
 9: /**
10:  * A simplified XML parser.  In essence, this class supports a subset
11:  * of the functionality collectively offered by javax.xml.parsers.SAXParser
12:  * and javax.xml.parsers.DocumentBuilder.
13:  *
14:  * You MAY modify this file.
15:  *
16:  * @author  Computer Science E-259
17:  * @version 7.0
18:  *
19:  * @author  YOUR NAME GOES HERE
20:  **/
21: public class XMLParser
22: {
23:     /**
24:      * Storage for input file's contents.
25:      */
26:     private String data_;
27:
28:
29:     /**
30:      * A reference to the currently registered DefaultHandler.
31:      */
32:     private DefaultHandler handler_;
33:
34:
35:     /**
36:      * Index of our current location in input file's contents.
37:      */
38:     private int index_ = 0;
39:
40:
41:     /**
42:      * Returns true if the next characters in the stream are the beginning
43:      * of an element's end tag.
44:      *
45:      * @return true iff next characters in the stream are the beginning
46:      * of an element's end tag
47:      */
48:     protected boolean isEndTag()
49:     {
50:         return (data_.charAt(index_) == '<')
51:         && (data_.charAt(index_ + 1) == '/');
52:     }
53:
54:
55:     /**
56:      * Returns true if the next character in the stream is the beginning
57:      * of an element's start tag.
58:      *
59:      * @return true iff next character in the stream is the beginning
60:      * of an element's start tag
61:      */
62:     protected boolean isStartTag()
63:     {
64:         return data_.charAt(index_) == '<';
65:     }
66:
67:
68:     /**
69:      * Parses the specified file, if possible, passing SAX events
70:      * to given handler.
71:      *
72:      * @param filename name of file whose contents are to be parsed
73:      * @param handler  DefaultHandler for SAX events
74:      */
75:     public void parse(String filename, DefaultHandler handler)
76:     {
77:         // initialize to clean up from any previous parse
78:         data_ = "";
79:         index_ = 0;
80:         handler_ = handler;
81:
82:         // attempt to open file and read contents into local storage
83:         try
84:         {
85:             File f = new File(filename);
86:             int filesize = (int) f.length();
87:             byte[] filebytes = new byte[filesize];
88:             DataInputStream in = new DataInputStream(new FileInputStream(f));
89:             in.readFully(filebytes);
90:             in.close();
91:             data_ = new String(filebytes);
92:         }
93:         catch (IOException E)
94:         {
95:             handler_.fatalError(new Exception("Error: could not read file"));
96:             return;
97:         }
98:
99:         // parse the document; hopefully there's a root element!
100:         handler_.startDocument();
101:         readElement();
102:         handler_.endDocument();
103:     }
104:
105:
106:     /**
107:      * Parses an element and its content.
108:      */
109:     protected void readElement()
110:     {
111:         if (!isStartTag())
112:         {
113:             handler_.fatalError(new RuntimeException("Error: expecting " +
114:                                                      "start of element"));
115:             return;
116:         }
117:
118:         // parse end tag
119:         String name = readStartTag();
120:
121:         // keep reading in more elements and text until an end tag
122:         // is encountered
123:         while (!isEndTag())
124:         {
```

```
125:             if (isStartTag())
126:                 readElement();
127:             else
128:                 readText();
129:         }
130:
131:         // parse end tag, ensuring it matches most current start tag
132:         readEndTag(name);
133:     }
134:
135:
136:     /**
137:      * Parses an end tag, ensuring its name matches currently opened
138:      * element's name.
139:      *
140:      * @param checkName currently opened element's name with which
141:      * end tag should be compared
142:      */
143:     protected void readEndTag(String checkName)
144:     {
145:         // start name from scratch
146:         String name = "";
147:
148:         // read starting <
149:         index_++;
150:
151:         // read /
152:         index_++;
153:
154:         // read name
155:         while (data_.charAt(index_) != '>')
156:         {
157:             name += data_.charAt(index_);
158:             index_++;
159:         }
160:
161:         // read ending >
162:         index_++;
163:
164:         // ensure content is well-formed
165:         if (!checkName.equals(name))
166:         {
167:             handler_.fatalError(new RuntimeException("Error: expecting " +
168:                                                      "closing tag for " +
169:                                                      checkName));
170:             return;
171:         }
172:
173:         // pass this SAX event to handler
174:         handler_.endElement(name);
175:     }
176:
177:
178:     /**
179:      * Parses a start tag, returning opened element's name.
180:      *
181:      * @return name of element
182:      */
183:     protected String readStartTag()
184:     {
185:         // start name from scratch
186:         String name = "";
187:
188:         // Read starting <
189:         index_++;
190:
191:         // Read name
192:         while (data_.charAt(index_) != '>')
193:         {
194:             name += data_.charAt(index_);
195:             index_++;
196:         }
197:
198:         // Read ending >
199:         index_++;
200:
201:         // pass this SAX event to handler;
202:         // you MUST replace null below with a reference to
203:         // this element's Attributes object
204:         handler_.startElement(name, null);
205:
206:         // return this element's name, for later comparision
207:         // with an end tag
208:         return name;
209:     }
210:
211:
212:     /**
213:      * Parses character data.
214:      */
215:     protected void readText()
216:     {
217:         // start character data from scratch
218:         String content = "";
219:
220:         // accumulate characters until next tag
221:         while (data_.charAt(index_) != '<')
222:         {
223:             content += data_.charAt(index_);
224:             index_++;
225:         }
226:
227:         // pass this SAX event to handler
228:         handler_.characters(content);
229:     }
230: }
```

```
 1: package cscie259.project1.mf;
 2:
 3: import java.io.BufferedWriter;
 4: import java.io.IOException;
 5:
 6:
 7: /**
 8:  * A ContentHandler for SAX events that serializes (to an output stream)
 9:  * the events back into XML.  Essentially, a simplified version of
10:  * org.apache.xml.serialize.XMLSerializer.
11:  *
12:  * You MAY modify this file to provide support for the
13:  * serialization of attributes.
14:  *
15:  * @author  Computer Science E-259
16:  * @version 7.0
17:  *
18:  * @author  YOUR NAME GOES HERE
19:  **/
20: public class XMLSerializer extends DefaultHandler
21: {
22:     /**
23:      * The output stream to which we are serializing.
24:      */
25:     private BufferedWriter out_;
26:
27:
28:     /**
29:      * A flag signifying whether output should be indented (i.e.,
30:      * pretty-printed).
31:      */
32:     private boolean prettyPrint_;
33:
34:
35:     /**
36:      * The current level of indentation, if applicable.
37:      */
38:     private int indentLevel_;
39:
40:
41:     /**
42:      * Configures XMLSerializer with given BufferedWriter and for
43:      * pretty-printedness, if applicable.
44:      *
45:      * @param writer      BufferedWriter for serialization
46:      * @param prettyPrint flag indicating whether to pretty-print
47:      */
48:     public XMLSerializer(BufferedWriter writer, boolean prettyPrint)
49:     {
50:         out_ = writer;
51:         prettyPrint_ = prettyPrint;
52:         indentLevel_ = 0;
53:     }
54:
55:
56:     /**
57:      * Prints out character data, pretty-printed if applicable.
58:      *
59:      * @param content  character data
60:      */
61:     public void characters(String content)
62:     {
63:         try
64:         {
65:             // pretty-print if applicable
66:             if (prettyPrint_)
67:                 indent();
68:
69:             // write character data
70:             out_.write(content, 0, content.length());
71:
72:             // pretty-print if applicable
73:             if (prettyPrint_)
74:                 out_.newLine();
75:         }
76:         catch (IOException E)
77:         {
78:             throw new RuntimeException("Error: I/O error " + E.getMessage());
79:         }
80:     }
81:
82:
83:     /**
84:      * Closes the output stream.
85:      */
86:     public void endDocument()
87:     {
88:         try
89:         {
90:             out_.close();
91:         }
92:         catch (IOException E)
93:         {
94:             throw new RuntimeException("Error: I/O error " + E.getMessage());
95:         }
96:     }
97:
98:
99:     /**
100:      * Prints out the end element tag, pretty-printed if applicable, and
101:      * updates the current level of indentation.
102:      *
103:      * @param name name of element
104:      */
105:     public void endElement(String name)
106:     {
107:         try
108:         {
109:             // pretty-print if applicable
110:             if (prettyPrint_)
111:             {
112:                 indentLevel_--;
113:                 indent();
114:             }
115:
116:             // write </
117:             out_.write('<');
118:             out_.write('/');
119:
120:             // write element's name
121:             out_.write(name, 0, name.length());
122:
123:             // write >
124:             out_.write('>');
```

```
125:
126:             // pretty-print if applicable
127:             if (prettyPrint_)
128:                 out_.newLine();
129:         }
130:         catch (IOException E)
131:         {
132:             throw new RuntimeException("Error: I/O error " + E.getMessage());
133:         }
134:     }
135:
136:
137:     /**
138:      * Prints the number of indents currently appropriate.
139:      *
140:      * @throws IOException run-time failure of writing operation
141:      */
142:     private void indent() throws IOException
143:     {
144:         for (int i = 0; i < indentLevel_; i++)
145:         {
146:             out_.write("    ");
147:         }
148:     }
149:
150:
151:     /**
152:      * Prints out the start element tag, pretty-printed if applicable, and
153:      * updates the current level of indentation.
154:      *
155:      * @param name name of element
156:      * @param atts element's collection of attributes
157:      */
158:     public void startElement(String name, Attributes atts)
159:     {
160:         try
161:         {
162:             // pretty-print if applicable
163:             if (prettyPrint_)
164:                 indent();
165:
166:             // write <
167:             out_.write('<');
168:
169:             // write element's name
170:             out_.write(name, 0, name.length());
171:
172:             // write element's attributes, if any
173:             // TODO
174:
175:             // write >
176:             out_.write('>');
177:
178:             // pretty-print if applicable
179:             if (prettyPrint_)
180:             {
181:                 out_.newLine();
182:                 indentLevel_++;
183:             }
184:         }
185:         catch (IOException E)
186:         {
187:             throw new RuntimeException("Error: I/O error " + E.getMessage());
188:         }
189:     }
190: }
```