```
 1: import java.io.*;
 2:
 3: import java.text.DecimalFormat;
 4:
 5: import javax.xml.parsers.DocumentBuilder;
 6: import javax.xml.parsers.DocumentBuilderFactory;
 7: import javax.xml.parsers.ParserConfigurationException;
 8:
 9: import org.apache.xml.serialize.OutputFormat;
10: import org.apache.xml.serialize.XMLSerializer;
11:
12: import org.w3c.dom.Document;
13: import org.w3c.dom.Element;
14: import org.w3c.dom.Node;
15: import org.w3c.dom.NodeList;
16: import org.xml.sax.SAXException;
17:
18:
19: /**
20:  * Lecture 3's demonstration of
21:  * javax.xml.parsers.DocumentBuilder.
22:  *
23:  * @author  Computer Science E-259
24:  **/
25: public class DocumentBuilderDemo
26: {
27:     /**
28:      * Main driver.  Expects one command-line argument:
29:      * the name of the file to parse.
30:      *
31:      * @param argv [0] - filename
32:      */
33:     public static void main(String [] argv)
34:     {
35:         // ensure proper usage
36:         if (argv.length != 1)
37:         {
38:             System.out.println("Usage: java DocumentBuilderDemo filename");
39:             System.exit(1);
40:         }
41:
42:         // grab filename
43:         String input = argv[0];
44:
45:         // instantiate a DOM parser
46:         DocumentBuilder parser;
47:         try
48:         {
49:             DocumentBuilderFactory factory
50:              = DocumentBuilderFactory.newInstance();
51:             parser = factory.newDocumentBuilder();
52:         }
53:         catch (ParserConfigurationException e)
54:         {
55:             e.printStackTrace();
56:             return;
57:         }
58:
59:         // parse the document and grab the Document node
60:         Document doc;
61:         try
62:         {
63:             doc = parser.parse(input);
64:         }
65:         catch (SAXException e)
66:         {
67:             e.printStackTrace();
68:             return;
69:         }
70:         catch (IOException e)
71:         {
72:             System.err.println("Error reading file.\n");
73:             e.printStackTrace();
74:             return;
75:         }
76:
77:         // Intantiate DOMDemo object
78:         DocumentBuilderDemo demo = new DocumentBuilderDemo();
79:
80:         // calculate sale prices
81:         demo.slashPrices(doc.getDocumentElement());
82:
83:         // serialize the modified document out to System.out
84:         XMLSerializer serializer =
85:          new XMLSerializer(System.out,
86:                            new OutputFormat("XML", "UTF-8", true));
87:
88:         try
89:         {
90:             serializer.serialize(doc);
91:         }
92:         catch (IOException e)
93:         {
94:             System.err.println("Error reading file.\n");
95:             e.printStackTrace();
96:         }
97:     }
98:
99:
100:     /**
101:      * To every price element, encountered recursively, add a sale-price
102:      * attribute that reflects the price less a 20% discount.
103:      *
104:      * @param elt   element whose price is to be slashed
105:      */
106:     void slashPrices(Element elt)
107:     {
108:         // get element's children, if any
109:         NodeList children = elt.getChildNodes();
110:
111:         // iterate through children
112:         for (int i = 0; i < children.getLength(); i++)
113:         {
114:                         // get current child
115:             Node n = children.item(i);
116:
117:             // we're only interested in Element children
118:             if (n.getNodeType() != Node.ELEMENT_NODE)
119:                 continue;
120:
121:             // if this child is a price element, slash it!
122:             if (n.getNodeName().equals("price"))
123:             {
124:                 double price = 0.80 *
```

```
125:                   new Double(n.getFirstChild().getNodeValue()).doubleValue();
126:                  DecimalFormat df = new DecimalFormat(".00");
127:                 ((Element) n).setAttribute("sale-price", df.format(price));
128:             }
129:
130:             // recurse on child
131:             slashPrices((Element) n);
132:         }
133:     }
134: }
```

```
 1: import cscie259.project1.mf.*;
 2:
 3: import java.io.*;
 4: import java.util.Iterator;
 5: import java.util.List;
 6:
 7:
 8: /**
 9:  * Lecture 3's demonstration of Project 1's DOMBuilder.
10:  *
11:  * @author  Computer Science E-259
12:  **/
13:
14: public class DOMBuilderDemo
15: {
16:     // counters
17:     private int texts_      = 0;
18:     private int elements_   = 0;
19:
20:
21:     /**
22:      * Main driver.  Expects one command-line argument:
23:      * the name of the file to parse.
24:      *
25:      * @param argv [0] - filename
26:      */
27:
28:     public static void main(String [] argv)
29:     {
30:         // grab filename
31:         String input = argv[0];
32:
33:         try
34:         {
35:             // instantiate My First XML Parser and a DOMBuilder
36:             XMLParser p = new XMLParser();
37:             DOMBuilder db = new DOMBuilder();
38:
39:             // parse the document
40:             p.parse(input, db);
41:
42:             // grab the Document node
43:             Document doc = db.getDocument();
44:
45:             // instantiate an object of this class
46:             DOMBuilderDemo demo = new DOMBuilderDemo();
47:
48:             // count its Element and Text nodes
49:             try
50:             {
51:                 demo.visit(doc.getDocumentElement());
52:             }
53:             catch (Exception e)
54:             {
55:                 e.printStackTrace();
56:             }
57:
58:             // report results
59:             System.out.println("There were " + demo.elements_ +
60:                                " elements.");
61:             System.out.println("There were " + demo.texts_ +
62:                                " text nodes.");
63:         }
64:         catch (Exception e)
65:         {
66:             e.printStackTrace();
67:         }
68:     }
69:
70:
71:     /**
72:      * Recursively count Element and Text nodes.
73:      *
74:      * @param n node to examine and then recurse on
75:      */
76:
77:     public void visit(Node n)
78:     {
79:         switch (n.getNodeType())
80:         {
81:             // if this is a Text node, record such
82:             case Node.TEXT_NODE:
83:                 texts_++;
84:                 break;
85:
86:             // if this is an Element node, record such, and then
87:             // recurse on any children
88:             case Node.ELEMENT_NODE:
89:                 elements_++;
90:                 List children = n.getChildNodes();
91:                 Iterator it = children.iterator();
92:                 while (it.hasNext())
93:                     visit((Node) it.next());
94:
95:             break;
96:         }
97:     }
98: }
```

```
<items>
    <item>
        <name>Widget</name>
        <price>5.25</price>
    </item>
    <item>
        <name>Big Widget</name>
        <item>
            <name>Component 1</name>
            <price>345.00</price>
        </item>
        <item>
            <name>Component 2</name>
            <price>123.10</price>
        </item>
    </item>
</items>
```