```java
import java.io.*;
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;
import components.xmltree.XMLTree;
import components.xmltree.XMLTree1;


/**
 * Program to convert an XML RSS (version 2.0) feed from a given URL into the
 * corresponding HTML output file.
 *
 * @author Put your name here
 *
 */
public final class RSSReader {

    /**
     * Private constructor so this utility class cannot be instantiated.
     */
    private RSSReader() {
    }

    /**
     * Outputs the "opening" tags in the generated HTML file. These are the
     * expected elements generated by this method:
     *
     * <html>
     * <head>
     * <title>the channel tag title as the page title</title>
     * </head>
     * <body>
     *  <h1>the page title inside a link to the <channel> link</h1>
     *  <p>the channel description</p>
     *  <table border="1">
     *   <tr>
     *    <th>Date</th>
     *    <th>Source</th>
     *    <th>News</th>
     *   </tr>
     *
     * @param channel
     *            the channel element XMLTree
     * @param out
     *            the output stream
     * @updates out.content
     * @requires [the root of channel is a <channel> tag] and out.is_open
     * @ensures out.content = #out.content * [the HTML "opening" tags]
     */
    private static void outputHeader(XMLTree channel, SimpleWriter out) {
        assert channel != null : "Violation of: channel is not null";
        assert out != null : "Violation of: out is not null";
        assert channel.isTag() && channel.label().equals("channel") : ""
                + "Violation of: the label root of channel is a <channel> tag";
        assert out.isOpen() : "Violation of: out.is_open";
        out.print("<html>\n\t"//using lots of \t in here, looks messy but .html file looks
```

```
   good
58                 + "<head>\n\t\t"
59                 + "<title>");
60         String title;// checks for title
61         if (getChildElement(channel, "title") == -1) {
62             title = "Empty Title";
63         } else {
64             title = channel.child(getChildElement(channel, "title")).child(0).label();
65         }
66         out.print(title);
67         out.print("</title>\n\t"
68                 + "</head>\n\t<body>"
69                 + "\n\t\t"
70                 + "<h1><a href=\"");
71         String link = channel.child(getChildElement(channel, "link")).child(0).label();
72         out.print(link);//print the link
73         out.print("\">");
74         out.print(title);
75         out.print("</a></h>\n\t\t"
76                 + "<p>");
77         String description; //check for a description
78         if (getChildElement(channel, "description") == -1) {
79         description = "No description available";
80         } else {
81         description = channel.child(getChildElement(channel, "description")).child(0).label();
82         }
83         out.print(description);
84         out.print("</p>\n\t\t"
85                 + "<table border=\"1\">\n\t\t\t"
86                 + "<tr>\n\t\t\t\t"
87                 + "<th>Date</th>"
88                 + "\n\t\t\t\t"
89                 + "<th>Source</th>\n\t\t\t\t"
90                 + "<th>News</th>\n\t\t\t"
91                 + "</tr>");
92     }
93
94     /**
95      * Outputs the "closing" tags in the generated HTML file.  These are the
96      * expected elements generated by this method:
97      *
98      *  </table>
99      * </body>
100     * </html>
101     *
102     * @param out
103     *            the output stream
104     * @updates out.contents
105     * @requires out.is_open
106     * @ensures out.content = #out.content * [the HTML "closing" tags]
107     */
108    private static void outputFooter(SimpleWriter out) {
109        assert out != null : "Violation of: out is not null";
110        assert out.isOpen() : "Violation of: out.is_open";
111        out.print("\n\t\t"
112                + "</table>\n\t"
113                + "</body>\n"
```

```
114                 + "</html>");
115     }
116
117     /**
118      * Finds the first occurrence of the given tag among the children of the
119      * given {@code XMLTree} and return its index; returns -1 if not found.
120      *
121      * @param xml
122      *            the {@code XMLTree} to search
123      * @param tag
124      *            the tag to look for
125      * @return the index of the first child of type tag of the {@code XMLTree}
126      *         or -1 if not found
127      * @requires [the label of the root of xml is a tag]
128      * @ensures <pre>
129      * getChildElement =
130      *  [the index of the first child of type tag of the {@code XMLTree} or
131      *   -1 if not found]
132      * </pre>
133      */
134     private static int getChildElement(XMLTree xml, String tag) {
135         assert xml != null : "Violation of: xml is not null";
136         assert tag != null : "Violation of: tag is not null";
137         assert xml.isTag() : "Violation of: the label root of xml is a tag";
138         int i = 0;
139         int index = -1; //checks for index of a tag, outputs -1 if not there
140         while (i<xml.numberOfChildren()) {
141             if (xml.child(i).label().equals(tag)) {
142                 index = i;
143             }
144             i++; //forgetting this led to about 15 minutes of debug tracing just wanted to say
145         }
146         return index;
147     }
148
149     /**
150      * Processes one news item and outputs one table row. The row contains three
151      * elements: the publication date, the source, and the title (or
152      * description) of the item.
153      *
154      * @param item
155      *            the news item
156      * @param out
157      *            the output stream
158      * @updates out.content
159      * @requires
160      * [the label of the root of item is an <item> tag] and out.is_open
161      * @ensures <pre>
162      * out.content = #out.content *
163      *   [an HTML table row with publication date, source, and title of news item]
164      * </pre>
165      */
166     private static void processItem(XMLTree item, SimpleWriter out) {
167         assert item != null : "Violation of: item is not null";
168         assert out != null : "Violation of: out is not null";
169         assert item.isTag() && item.label().equals("item") : ""
170                 + "Violation of: the label root of item is an <item> tag";
```

```
171         assert out.isOpen() : "Violation of: out.is_open";
172         out.print("\n\t\t\t<tr>\n\t\t\t\t<td>");
173         String date; //check for the date
174         if (getChildElement(item, "pubDate") == -1) {
175             date = "No date available";
176             } else {
177             date = item.child(getChildElement(item, "pubDate")).child(0).label();
178             }
179         out.print(date);
180         out.print("</td>\n\t\t\t\t<td>");
181 
182         String source, slink;//check for source
183         if (getChildElement(item, "source") == -1) {
184             source = "No source available";
185             out.print(source);
186             out.print("</td>\n\t\t\t\t<td>");
187             } else {
188             source = item.child(getChildElement(item, "source")).child(0).label();
189             slink = item.child(getChildElement(item, "source")).attributeValue("url");
190             out.print("<a href=\"");
191             out.print(slink);
192             out.print("\">");
193             out.print(source);
194             out.print("</a></td>\n\t\t\t\t<td>");
195             }
196 
197 
198         String news, nlink = "";
199         //independent check for title
200         if (getChildElement(item, "title") != -1) {
201             news = item.child(getChildElement(item, "title")).child(0).label();
202         } else {
203             news = "No title available";
204         }
205         //independent check for link to article
206         if (getChildElement(item, "link") != -1) {
207             nlink = item.child(getChildElement(item, "link")).child(0).label();
208         }
209         //case for if link exists or not, omits the a href if there is no link
210         if (nlink.equals("")) {
211             out.print(news);
212             out.print("</td>\n\t\t\t"
213                     + "</tr>");
214         } else {
215             out.print("<a href=\"");
216             out.print(nlink);
217             out.print("\">");
218             out.print(news);
219             out.print("</a></td>\n\t\t\t"
220                     + "</tr>");
221         }
222 
223     }
224 
225 
226     /**
227      * Main method.
```

```
228      *
229      * @param args
230      *            the command line arguments; unused here
231      * @throws IOException
232      */
233     public static void main(String[] args) throws IOException {
234         SimpleReader in = new SimpleReader1L();
235         SimpleWriter out = new SimpleWriter1L();
236         //I set the directory to the F: drive because it was easier to find
237         SimpleWriter output = new SimpleWriter1L("F:/output.html");
238         out.println("Input a link: ");
239         String link = in.nextLine();
240
241         XMLTree xml = new XMLTree1(link);
242         XMLTree channel = xml.child(0);
243         //makes sure that xml root is a tag as per instructions
244         if (xml.isTag()) {
245             //call to each method
246             outputHeader(channel, output);
247             //calls once for every child item in channel
248             for (int i = 0; i<channel.numberOfChildren(); i++) {
249                 if (channel.child(i).label().equals("item")) {
250                     processItem(channel.child(i), output);
251                 }
252             }
253             outputFooter(output);
254         } else {
255             out.print("root is not a tag");
256         }
257         in.close();
258         out.close();
259     }
260 }
```