```java
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 Isaac Frank
 *
 */
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
```

```
41      *            the channel element XMLTree
42      * @param out
43      *            the output stream
44      * @updates out.content
45      * @requires [the root of channel is a <channel> tag] and
    out.is_open
46      * @ensures out.content = #out.content * [the HTML "opening"
    tags]
47      */
48     private static void outputHeader(XMLTree channel, SimpleWriter
    out) {
49         assert channel != null : "Violation of: channel is not
    null";
50         assert out != null : "Violation of: out is not null";
51         assert channel.isTag() &&
    channel.label().equals("channel") : ""
52                 + "Violation of: the label root of channel is a
    <channel> tag";
53         assert out.isOpen() : "Violation of: out.is_open";
54
55         // Getting indexes of title, link, and description
56         int titleChildIndex = getChildElement(channel, "title");
57         int linkChildIndex = getChildElement(channel, "link");
58         int descriptionChildIndex = getChildElement(channel,
    "description");
59
60         // Html opening tags and printing title
61         out.print("<html> <head> <title>");
62         String title = "Empty Title";
63         if (channel.child(titleChildIndex).numberOfChildren() > 0)
    {
64             title =
    channel.child(titleChildIndex).child(0).label();
65         }
66         out.print(title);
67
68         // Html closing tags
69         out.println("</title> </head> <body>");
70
71         // Html header and page title w/ link
72         out.print("<h1>");
73         out.print(
74                 "<a href = \"" +
    channel.child(linkChildIndex).child(0).label()
```

```
75                          + "\">" + title + "</a>");
76          out.println("</h1>");
77
78          // Html paragraph w/ channel description
79          out.println("<p>");
80          String description = "No description";
81          if
    (channel.child(descriptionChildIndex).numberOfChildren() > 0) {
82              description =
    channel.child(descriptionChildIndex).child(0).label();
83          }
84          out.println(description);
85          out.println("</p>");
86
87          // Table Headers
88          out.println("<table border = \"" + 1 + "\">");
89          out.println("<tr>");
90          out.println("<th>" + "Date" + "</th>");
91          out.println("<th>" + "Source" + "</th>");
92          out.println("<th>" + "News" + "</th>");
93          out.println("</tr>");
94      }
95
96      /**
97       * Outputs the "closing" tags in the generated HTML file.
    These are the
98       * expected elements generated by this method:
99       *
100      * </table>
101      * </body> </html>
102      *
103      * @param out
104      *            the output stream
105      * @updates out.contents
106      * @requires out.is_open
107      * @ensures out.content = #out.content * [the HTML "closing"
    tags]
108      */
109     private static void outputFooter(SimpleWriter out) {
110         assert out != null : "Violation of: out is not null";
111         assert out.isOpen() : "Violation of: out.is_open";
112
113         out.println("</table>");
114         out.println("</body> </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
139         int index = -1;
140
141         // Iterates through children until a tag is found or all
    children are searched
142         int i = 0;
143         while (i < xml.numberOfChildren() && index < 0) {
144             if (xml.child(i).label().equals(tag)) {
145                 index = i;
146             }
147             i++;
148         }
149
150         return index;
151     }
152
```

```
153     /**
154      * Processes one news item and outputs one table row. The row
    contains three
155      * elements: the publication date, the source, and the title
    (or
156      * description) of the item.
157      *
158      * @param item
159      *            the news item
160      * @param out
161      *            the output stream
162      * @updates out.content
163      * @requires [the label of the root of item is an <item> tag]
    and
164      *           out.is_open
165      * @ensures <pre>
166      * out.content = #out.content *
167      *   [an HTML table row with publication date, source, and
    title of news item]
168      * </pre>
169      */
170     private static void processItem(XMLTree item, SimpleWriter
    out) {
171         assert item != null : "Violation of: item is not null";
172         assert out != null : "Violation of: out is not null";
173         assert item.isTag() && item.label().equals("item") : ""
174                 + "Violation of: the label root of item is an
    <item> tag";
175         assert out.isOpen() : "Violation of: out.is_open";
176
177         // Finding indexes
178         int titleChildIndex = getChildElement(item, "title");
179         int linkChildIndex = getChildElement(item, "link");
180         int descriptionChildIndex = getChildElement(item,
    "description");
181         int pubDateChildIndex = getChildElement(item, "pubDate");
182         int sourceChildIndex = getChildElement(item, "source");
183
184         out.println("<tr>");
185
186         // Printing pubDate table cell
187         String pubDate = "No date available";
188         if (pubDateChildIndex != -1) {
189             pubDate =
```

```
    item.child(pubDateChildIndex).child(0).label();
190         }
191         out.println("<td>" + pubDate + "</td>");
192 
193         // Printing source table cell
194         String source = "No source available";
195         if (sourceChildIndex != -1) {
196             XMLTree src = item.child(sourceChildIndex);
197             if (src.numberOfChildren() > 0) {
198                 String srcAttributeVal =
    src.attributeValue("url");
199                 source = "<a href = \"" + srcAttributeVal;
200                 source += "\">" + src.child(0).label() + "</a>";
201             }
202         }
203         out.println("<td>" + source + "</td>");
204 
205         // Printing title table cell, checking if description and
    link are needed
206         String titleOrDsc = "No title available";
207         String link = "";
208         if (titleChildIndex != -1
209                 && item.child(titleChildIndex).numberOfChildren()
    > 0) {
210             titleOrDsc =
    item.child(titleChildIndex).child(0).label();
211         } else if (descriptionChildIndex != -1
212                 &&
    item.child(descriptionChildIndex).numberOfChildren() > 0) {
213             titleOrDsc =
    item.child(descriptionChildIndex).child(0).label();
214         }
215         if (linkChildIndex != -1) {
216             link = item.child(linkChildIndex).child(0).label();
217         }
218 
219         out.print("<td><a href = \"" + link + "\">" + titleOrDsc +
    "</a></td>");
220 
221         // Ending the row
222         out.println("</tr>");
223     }
224 
225     /**
```

```
226      * Main method.
227      *
228      * @param args
229      *            the command line arguments; unused here
230      */
231     public static void main(String[] args) {
232         SimpleReader in = new SimpleReader1L();
233         SimpleWriter out = new SimpleWriter1L();
234 
235         // User input
236         out.print("Enter a URL for an RSS 2.0 news feed: ");
237         String url = in.nextLine();
238         out.print("Enter the output file name including .html: ");
239         String fileName = in.nextLine();
240 
241         SimpleWriter outFile = new SimpleWriter1L(fileName);
242 
243         // Checking if xml is RSS 2.0
244         XMLTree xml = new XMLTree1(url);
245         if (xml.isTag()) {
246             if (xml.hasAttribute("version")) {
247                 if (xml.attributeValue("version").equals("2.0")) {
248                     XMLTree channel = xml.child(0);
249                     outputHeader(channel, outFile);
250 
251                     // Iterating through all "item" children of
    channel
252                     int i = 0;
253                     while (i < channel.numberOfChildren()) {
254                         if
    (channel.child(i).label().equals("item")) {
255                             processItem(channel.child(i),
    outFile);
256                         }
257                         i++;
258                     }
259 
260                     outputFooter(outFile);
261                 }
262             }
263         } else {
264             out.println("This is not an RSS 2.0 file");
265         }
266 
```

```
267         // Closing output streams
268         in.close();
269         out.close();
270         outFile.close();
271     }
272 }
273
```