```java
import components.simplereader.SimpleReader;

/**
 * Program to convert an XML RSS (version 2.0) feed from a given URL into the
 * corresponding HTML output file.
 *
 * @author Jonathan Pater
 *
 */
public final class RSSAggregator {

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

    /**
     * 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
```

```
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         int titleIndex = getChildElement(channel, "title");
55         XMLTree title = channel.child(titleIndex);
56         String titleLabel = "Empty Title";
57         if (title.numberOfChildren() > 0) {
58             titleLabel = title.child(0).label();
59         }
60         out.println("<html> <head> <title>" + titleLabel + "</
   title>");
61         out.println("</head> <body>");
62         int linkIndex = getChildElement(channel, "link");
63         XMLTree link = channel.child(linkIndex);
64         String linkContent = link.child(0).label();
65         out.print("<h1>\n<a href=\"" + linkContent + "\">" +
   titleLabel
66                 + "</a>\n</h1>");
67         int descriptionIndex = getChildElement(channel,
   "description");
68         XMLTree description = channel.child(descriptionIndex);
69         String descLabel = "No description";
70         if (description.numberOfChildren() > 0) {
71             descLabel = description.child(0).label();
72         }
73         out.println("<p>\n" + descLabel + "\n</p>");
74         out.println("<table border=\"1\">");
75         out.println(
76                 "<tr>\n<th>Date</th>\n<th>Source</th>\n<th>News</
   th>\n</tr>");
77         for (int i = 0; i < channel.numberOfChildren(); i++) {
78             if (channel.child(i).label().equals("item")) {
79                 XMLTree item = channel.child(i);
80                 processItem(item, out);
81             }
```

```
 82         }
 83
 84     }
 85
 86     /**
 87      * Outputs the "closing" tags in the generated HTML file. These
    are the
 88      * expected elements generated by this method:
 89      *
 90      * </table>
 91      * </body> </html>
 92      *
 93      * @param out
 94      *            the output stream
 95      * @updates out.contents
 96      * @requires out.is_open
 97      * @ensures out.content = #out.content * [the HTML "closing"
    tags]
 98      */
 99     private static void outputFooter(SimpleWriter out) {
100         assert out != null : "Violation of: out is not null";
101         assert out.isOpen() : "Violation of: out.is_open";
102         out.print("</table>\n</body> </html>");
103     }
104
105     /**
106      * Finds the first occurrence of the given tag among the
    children of the
107      * given {@code XMLTree} and return its index; returns -1 if
    not found.
108      *
109      * @param xml
110      *            the {@code XMLTree} to search
111      * @param tag
112      *            the tag to look for
113      * @return the index of the first child of type tag of the
    {@code XMLTree}
114      *         or -1 if not found
115      * @requires [the label of the root of xml is a tag]
116      * @ensures <pre>
117      * getChildElement =
118      *  [the index of the first child of type tag of the {@code
    XMLTree} or
119      *   -1 if not found]
120      * </pre>
```

```
121      */
122     private static int getChildElement(XMLTree xml, String tag) {
123         assert xml != null : "Violation of: xml is not null";
124         assert tag != null : "Violation of: tag is not null";
125         assert xml.isTag() : "Violation of: the label root of xml
    is a tag";
126         int children = xml.numberOfChildren();
127         for (int i = 0; i < children; i++) {
128             String name = xml.child(i).label();
129             if (name.equals(tag)) {
130                 return i;
131             }
132         }
133         return -1;
134     }
135 
136     /**
137      * Processes one news item and outputs one table row. The row
    contains three
138      * elements: the publication date, the source, and the title
    (or
139      * description) of the item.
140      *
141      * @param item
142      *            the news item
143      * @param out
144      *            the output stream
145      * @updates out.content
146      * @requires [the label of the root of item is an <item> tag]
    and
147      *           out.is_open
148      * @ensures <pre>
149      * out.content = #out.content *
150      *   [an HTML table row with publication date, source, and
    title of news item]
151      * </pre>
152      */
153     private static void processItem(XMLTree item, SimpleWriter out)
    {
154         assert item != null : "Violation of: item is not null";
155         assert out != null : "Violation of: out is not null";
156         assert item.isTag() && item.label().equals("item") : ""
157                 + "Violation of: the label root of item is an
    <item> tag";
158         assert out.isOpen() : "Violation of: out.is_open";
```

```
159
160         // IfAv = "if available";
161         int titleIfAv = getChildElement(item, "title");
162         int descIfAv = -1;
163         if (titleIfAv == -1) {
164             descIfAv = getChildElement(item, "description");
165         }
166         int linkIfAv = getChildElement(item, "link");
167         String link2 = "";
168         if (linkIfAv >= 0) {
169             link2 = item.child(linkIfAv).child(0).label();
170         }
171         int sourceIfAv = getChildElement(item, "source");
172         int pubDateIfAv = getChildElement(item, "pubDate");
173         out.println("<tr>");
174         if (pubDateIfAv >= 0) {
175             XMLTree pubDate = item.child(pubDateIfAv);
176             String date = pubDate.child(0).label();
177             out.println("<td>" + date + "</td>");
178         } else {
179             out.println("<td>No date available</td>");
180         }
181         if (sourceIfAv >= 0) {
182             XMLTree source = item.child(sourceIfAv);
183             String link1 = source.attributeValue("url");
184             String source1 = "Name of source not available. "
185                     + "Link is still available here.";
186             int sourceChildren = source.numberOfChildren();
187             // here because the source child does not need to have
    a child node
188             if (sourceChildren > 0) {
189                 source1 = source.child(0).label();
190             }
191             out.println("<td>\n<a href=\"" + link1 + "\">" +
    source1
192                     + "</a>\n</td>");
193         } else {
194             out.println("<td>No source available</td>");
195         }
196         if (titleIfAv >= 0) {
197             XMLTree title = item.child(titleIfAv);
198             String title1 = "No title available";
199             if (title.numberOfChildren() > 0) {
200                 title1 = title.child(0).label();
201             }
```

```
202             if (linkIfAv >= 0) {
203                 out.println("<td>\n<a href=\"" + link2 + "\">" +
    title1
204                         + "</a>\n</td>");
205             } else {
206                 out.println("<td>" + title1 + "</td>");
207             }
208         } else if (descIfAv >= 0) {
209             XMLTree desc = item.child(descIfAv);
210             String desc1 = "No title available";
211             if (desc.numberOfChildren() > 0) {
212                 desc1 = desc.child(0).label();
213             }
214             if (linkIfAv >= 0) {
215                 out.println("<td>\n<a href=\"" + link2 + "\">" +
    desc1
216                         + "</a>\n</td>");
217             } else {
218                 out.println("<td>" + desc1 + "</td>");
219             }
220         } else {
221             out.println("<td>No title available</td>");
222         }
223 
224         out.println("</tr>");
225     }
226 
227     /**
228      * Processes one XML RSS (version 2.0) feed from a given URL
    converting it
229      * into the corresponding HTML output file.
230      *
231      * @param url
232      *            the URL of the RSS feed
233      * @param file
234      *            the name of the HTML output file
235      * @param out
236      *            the output stream to report progress or errors
237      * @updates out.content
238      * @requires out.is_open
239      * @ensures <pre>
240      * [reads RSS feed from url, saves HTML document with tables of
    news items
241      *   to file, appends to out.content any needed messages]
242      * </pre>
```

```
243      */
244     private static void processFeed(String url, String file,
    SimpleWriter out) {
245         XMLTree rssInstance = new XMLTree1(url);
246         XMLTree channel = rssInstance.child(0);
247         outputHeader(channel, out);
248         outputFooter(out);
249     }
250 
251     /**
252      * Main method.
253      *
254      * @param args
255      *            the command line arguments; unused here
256      */
257     public static void main(String[] args) {
258         SimpleReader in = new SimpleReader1L();
259         SimpleWriter out = new SimpleWriter1L();
260         out.print("Insert an XML link/file name for the RSS
    Aggregator: ");
261         String xmlLink = in.nextLine();
262         XMLTree aggregate = new XMLTree1(xmlLink);
263         int aggNumChildren = aggregate.numberOfChildren();
264         out.print("\n\nInsert a name for the outputted HTML file "
265                 + "(including the .html at the end): ");
266         String htmlLink = in.nextLine();
267         SimpleWriter out1 = new SimpleWriter1L(htmlLink);
268         String title1 = aggregate.attributeValue("title");
269         out1.println("<html><head><title>" + title1 + "</title>");
270         out1.println("</head><body>\n<h1>" + title1 + "</h1>");
271         out1.println("<ul>");
272         for (int i = 0; i < aggNumChildren; i++) {
273             XMLTree feed = aggregate.child(i);
274             String feedURL = feed.attributeValue("url");
275             String feedFileName = feed.attributeValue("file");
276             String feedName = feed.attributeValue("name");
277             SimpleWriter out2 = new SimpleWriter1L(feedFileName);
278             processFeed(feedURL, feedFileName, out2);
279             out1.println("<li><a href = \"" + feedFileName + "\">"
    + feedName
280                     + "</a></li>");
281             out2.close();
282         }
283         out1.println("</ul></body></html>");
284         out.println("\nCheck the project folder for the finished
```

```
   HTML files.");
285         in.close();
286         out.close();
287         out1.close();
288 
289     }
290 
291 }
292 
```