```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 Jane Weissberg
 *
 */
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
55         // Initialize title to be empty until it is confirmed that
   title has a label
56         String title = "Empty Title";
57         int titleIndex = getChildElement(channel, "title");
58         if (channel.child(titleIndex).numberOfChildren() > 0) {
59             title = channel.child(titleIndex).child(0).label();
60         }
61
62         // Initialize link and set it to the label of the link
   node's child
63         int linkIndex = getChildElement(channel, "link");
64         String link = channel.child(linkIndex).child(0).label();
65
66         /*
67          * Initialize description as no description then change to
   description
68          * if child exists
69          */
70         String description = "No description";
71         int descriptionIndex = getChildElement(channel,
   "description");
72         if (channel.child(descriptionIndex).numberOfChildren() >
   0) {
73             description =
   channel.child(descriptionIndex).child(0).label();
74         }
75
76         // Print html code that will produce title, description,
   and table
```

```
 77         out.println("<html>");
 78         out.println("<head>");
 79         out.println("<title>" + title + "</title>");
 80         out.println("</head>");
 81         out.println("<body>");
 82         out.println("<h1><a href=\"" + link + "\">" + title + "</
    a></h1>");
 83         out.println("<p>" + description + "</p>");
 84         out.println("<table border=\"1\">");
 85         out.println("<tr><th>Date</th><th>Source</th><th>News</
    th></tr>");
 86     }
 87
 88     /**
 89      * Outputs the "closing" tags in the generated HTML file.
    These are the
 90      * expected elements generated by this method:
 91      *
 92      * </table>
 93      * </body> </html>
 94      *
 95      * @param out
 96      *            the output stream
 97      * @updates out.contents
 98      * @requires out.is_open
 99      * @ensures out.content = #out.content * [the HTML "closing"
    tags]
100      */
101     private static void outputFooter(SimpleWriter out) {
102         assert out != null : "Violation of: out is not null";
103         assert out.isOpen() : "Violation of: out.is_open";
104
105         // Print html code that will close each tag
106         out.println("</table>");
107         out.println("</body>");
108         out.println("</html>");
109     }
110
111     /**
112      * Finds the first occurrence of the given tag among the
    children of the
113      * given {@code XMLTree} and return its index; returns -1 if
    not found.
114      *
```

```
115      * @param xml
116      *            the {@code XMLTree} to search
117      * @param tag
118      *            the tag to look for
119      * @return the index of the first child of type tag of the
     {@code XMLTree}
120      *         or -1 if not found
121      * @requires [the label of the root of xml is a tag]
122      * @ensures <pre>
123      * getChildElement =
124      *  [the index of the first child of type tag of the {@code
     XMLTree} or
125      *   -1 if not found]
126      * </pre>
127      */
128     private static int getChildElement(XMLTree xml, String tag) {
129         assert xml != null : "Violation of: xml is not null";
130         assert tag != null : "Violation of: tag is not null";
131         assert xml.isTag() : "Violation of: the label root of xml
     is a tag";
132
133         // Initialize index to -1 (not found)
134         int index = -1;
135         int i = 0;
136         while (i < xml.numberOfChildren() && index == -1) {
137             if (xml.child(i).label().equals(tag)) {
138                 // Set index to the current value if tag is found
139                 index = i;
140             }
141             i++;
142         }
143         // Return the index (either -1 or the found index)
144         return index;
145     }
146
147     /**
148      * Processes one news item and outputs one table row. The row
     contains three
149      * elements: the publication date, the source, and the title
     (or
150      * description) of the item.
151      *
152      * @param item
153      *            the news item
```

```
154      * @param out
155      *            the output stream
156      * @updates out.content
157      * @requires [the label of the root of item is an <item> tag]
    and
158      *           out.is_open
159      * @ensures <pre>
160      * out.content = #out.content *
161      *   [an HTML table row with publication date, source, and
    title of news item]
162      * </pre>
163      */
164     private static void processItem(XMLTree item, SimpleWriter
    out) {
165         assert item != null : "Violation of: item is not null";
166         assert out != null : "Violation of: out is not null";
167         assert item.isTag() && item.label().equals("item")
168                 : "" + "Violation of: the label root of item is an
    <item> tag";
169         assert out.isOpen() : "Violation of: out.is_open";
170
171         // Initialize date to not available and change to date
    label if date exists
172         String date = "No date available";
173         int dateIndex = getChildElement(item, "pubDate");
174         if (dateIndex != -1 &&
    item.child(dateIndex).numberOfChildren() > 0) {
175             date = item.child(dateIndex).child(0).label();
176         }
177
178         // Initialize source to not available and change to source
    label if source exists
179         String source = "No source available";
180         int sourceIndex = getChildElement(item, "source");
181         if (sourceIndex != -1) {
182             XMLTree sourceElement = item.child(sourceIndex);
183             if (sourceElement.hasAttribute("url")) {
184                 String sourceUrl =
    sourceElement.attributeValue("url");
185                 if (sourceElement.numberOfChildren() > 0) {
186                     String sourceLabel =
    sourceElement.child(0).label();
187
188                     source = "<a href=\"" + sourceUrl + "\">" +
```

```
    sourceLabel + "</a>";
189                 }
190             }
191         }
192 
193         // Initialize title to not available and change to title
    label if title exists
194         String news = "No title available";
195         int titleIndex = getChildElement(item, "title");
196         if (titleIndex != -1 &&
    item.child(titleIndex).numberOfChildren() > 0) {
197             news = item.child(titleIndex).child(0).label();
198             // If there is no title child, use description child
199         } else {
200             int descIndex = getChildElement(item, "description");
201             if (descIndex != -1 &&
    item.child(descIndex).numberOfChildren() > 0) {
202                 news = item.child(descIndex).child(0).label();
203             }
204         }
205 
206         /*
207          * Initialize link to empty string and change to label of
    link child if
208          * link exists
209          */
210         String link = "";
211         int linkIndex = getChildElement(item, "link");
212         if (linkIndex != -1) {
213             link = "<a href=\"" +
    item.child(linkIndex).child(0).label() + "\">" + news
214                     + "</a>";
215         } else {
216             link = news;
217         }
218 
219         out.println("<tr><td>" + date + "</td><td>" + source + "</
    td><td>" + link
220                 + "</td></tr>");
221     }
222 
223     /**
224      * Processes one XML RSS (version 2.0) feed from a given URL
    converting it
```

```
225      * into the corresponding HTML output file.
226      *
227      * @param url
228      *            the URL of the RSS feed
229      * @param file
230      *            the name of the HTML output file
231      * @param out
232      *            the output stream to report progress or errors
233      * @updates out.content
234      * @requires out.is_open
235      * @ensures <pre>
236      * [reads RSS feed from url, saves HTML document with table of
    news items
237      *   to file, appends to out.content any needed messages]
238      * </pre>
239      */
240     private static void processFeed(String url, String file,
    SimpleWriter out) {
241         XMLTree xml = new XMLTree1(url);
242 
243         // Ensure XML is the correct version
244         if (xml.label().equals("rss") &&
    xml.hasAttribute("version")
245                 && xml.attributeValue("version").equals("2.0")) {
246 
247             XMLTree channel = xml.child(0);
248 
249             outputHeader(channel, out);
250 
251             for (int i = 0; i < channel.numberOfChildren(); i++) {
252                 if (channel.child(i).isTag()) {
253                     if (channel.child(i).label().equals("item")) {
254                         XMLTree item = channel.child(i);
255                         processItem(item, out);
256                     }
257                 }
258             }
259             outputFooter(out);
260         } else {
261             out.println("This is not an RSS 2.0 file.");
262         }
263     }
264 
265     /**
```

```
266      * Main method.
267      *
268      * @param args
269      *            the command line arguments; unused here
270      */
271     public static void main(String[] args) {
272         SimpleReader in = new SimpleReader1L();
273         SimpleWriter out = new SimpleWriter1L();
274
275         // Prompt user for an RSS 2.0 URL
276         out.print("Enter the URL of an RSS 2.0 news feed: ");
277         String input = in.nextLine();
278
279         // Initialize tree from input
280         XMLTree rss = new XMLTree1(input);
281
282         if (rss.label().equals("feeds") &&
    rss.hasAttribute("title")) {
283             out.print("Enter the name of the output file: ");
284             String fileName = in.nextLine();
285             // Create a new output stream for the file
286             SimpleWriter fileOut = new SimpleWriter1L("data/" +
    fileName);
287
288             String title = rss.attributeValue("title");
289             fileOut.println("<html>");
290             fileOut.println("<head>");
291             fileOut.println("<title>" + title + "</title>");
292             fileOut.println("</head>");
293             fileOut.println("<body>");
294             fileOut.println("<h1 style=\"font-size:150%;\">" +
    title + "</h1>");
295             fileOut.println("<ul>");
296
297             // Process each feed in the input
298             for (int i = 0; i < rss.numberOfChildren(); i++) {
299                 if (rss.child(i).label().equals("feed")) {
300
301                     String url =
    rss.child(i).attributeValue("url");
302                     String news =
    rss.child(i).attributeValue("file");
303                     String name =
    rss.child(i).attributeValue("name");
```

```
304
305                     SimpleWriter newsFeed = new
    SimpleWriter1L("data/" + news);
306
307                     processFeed(url, news, newsFeed);
308
309                     fileOut.println("<li><a href=\"" + news +
    "\">" + name + "</a></li>");
310                     newsFeed.close();
311                 }
312             }
313
314             fileOut.println("</ul>");
315             fileOut.println("</body>");
316             fileOut.println("</html>");
317             fileOut.close();
318         } else {
319             out.println("This file contains no feeds.");
320         }
321
322         in.close();
323         out.close();
324     }
325 }
326
```