```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 VishalKumar
 *
 */
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
     * @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";

        // Get positions of child, link description
        int titlePos = getChildElement(channel, "title");
        int linkPos = getChildElement(channel, "link");
        int descPos = getChildElement(channel, "description");

        // print title
        if (channel.child(titlePos).numberOfChildren() > 0) {
            out.println("<html><head><title>"
```

```
63                      + channel.child(titlePos).child(0).label() + "</title>");
64          } else {
65              out.println("<html><head><title>" + "Empty Title" + "</title>");
66          }
67
68          out.println("</head><body>");
69
70          // Print 1st header with hyperLink to page
71          out.print("<h1><a href=\"" + channel.child(linkPos).child 0).label()
72                  + "\">" + channel.child(titlePos).child(0).label()
73                  + "</a> </h1>");
74          out.println("<p>");
75
76          // print description of channel
77          if (channel.child(descPos).numberOfChildren() > 0) {
78              out.println(channel.child(descPos).child(0).label());
79          } else {
80              out.println("No description");
81          }
82
83          // print closing text of the header
84          out.println("</p>");
85          out.println("<table border=\"1\">");
86          out.println("<tr>");
87          out.println("<th>Date</th>");
88          out.println("<th>Source</th>");
89          out.println("<th>News</th>");
90          out.println("</tr>");
91
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> </html>
100      *
101      * @param out
102      *            the output stream
103      * @updates out.contents
104      * @requires out.is_open
105      * @ensures out.content = #out.content * [the HTML "closing" tags]
106      */
107     private static void outputFooter(SimpleWriter out) {
108         assert out != null : "Violation of: out is not null";
109         assert out.isOpen() : "Violation of: out.is_open";
110
111         // print closing text of the html page
112         out.println("</table>");
113         out.print("</body></html>");
114     }
115
116     /**
117      * Finds the first occurrence of the given tag among the children of the
118      * given {@code XMLTree} and return its index; returns -1 if not found.
119      *
```

```
120      * @param xml
121      *            the {@code XMLTree} to search
122      * @param tag
123      *            the tag to look for
124      * @return the index of the first child of type tag of the {@code XMLTree}
125      *         or -1 if not found
126      * @requires [the label of the root of xml is a tag]
127      * @ensures <pre>
128      * getChildElement =
129      *  [the index of the first child of type tag of the {@code XMLTree} or
130      *   -1 if not found]
131      * </pre>
132      */
133     private static int getChildElement(XMLTree xml, String tag) {
134         assert xml != null : "Violation of: xml is not null";
135         assert tag != null : "Violation of: tag is not null";
136         assert xml.isTag() : "Violation of: the label root of xml is a tag";
137
138         int pos = -1;
139         // loop through tree and find first position of desired tag
140         for (int i = 0; i < xml.numberOfChildren() && pos == -1; i++) {
141             String name = xml.child(i).label();
142             if (name.equals(tag)) {
143                 pos = i;
144             }
145         }
146         //return the position
147         return pos;
148     }
149
150     /**
151      * Processes one news item and outputs one table row. The row contains three
152      * elements: the publication date, the source, and the title (or
153      * description) of the item.
154      *
155      * @param item
156      *            the news item
157      * @param out
158      *            the output stream
159      * @updates out.content
160      * @requires [the label of the root of item is an <item> tag] and
161      *           out.is_open
162      * @ensures <pre>
163      * out.content = #out.content *
164      *   [an HTML table row with publication date, source, and title of news item]
165      * </pre>
166      */
167     private static void processItem(XMLTree item, SimpleWriter out) {
168         assert item != null : "Violation of: item is not null";
169         assert out != null : "Violation of: out is not null";
170         assert item.isTag() && item.label().equals("item") : ""
171                 + "Violation of: the label root of item is an <item> tag";
172         assert out.isOpen() : "Violation of: out.is_open";
173
174         // get positions of date, source, title and link
175         int datePos = getChildElement(item, "pubDate");
176         int sourcePos = getChildElement(item, "source");
```

```
177         int titlePos = getChildElement(item, "title");
178         int linkPos = getChildElement(item, "link");
179         int descPos = getChildElement(item, "description");
180 
181         out.print("<tr>");
182         // print the date
183         if (datePos != -1) {
184             out.println(
185                     "<td>" + item.child(datePos).child(0).label() + "</td>");
186         } else {
187             out.println("<td> No date availible </td>");
188         }
189         // print the source
190         if (sourcePos != -1) {
191             out.println("<td><a href=\""
192                     + item.child(sourcePos).attributeValue("url") + "\">"
193                     + item.child(sourcePos).child(0).label() + "</a></td>");
194         } else {
195             out.println("<td> No source availible </td>");
196         }
197         // print the article title or the article description with a hyperlink to the article
198         if (titlePos != -1) {
199             if (item.child(titlePos).numberOfChildren() > 0) {
200                 out.print("<td><a href=\""
201                         + item.child(linkPos).child(0).label() + "\">");
202                 out.print(item.child(titlePos).child(0).label() + "</a></td>");
203             } else if (descPos != -1) {
204                 if (item.child(descPos).numberOfChildren() > 0) {
205                     out.print("<td><a href=\""
206                             + item.child(linkPos).child(0).label() + "\">");
207                     out.print(
208                             item.child(descPos).child(0).label() + "</a></td>");
209 
210                 } else {
211                     out.println("<td> No title or description availible </td>");
212                 }
213             }
214         } else if (descPos != -1) {
215             if (item.child(descPos).numberOfChildren() > 0) {
216                 out.print("<td><a href=\""
217                         + item.child(linkPos).child(0).label() + "\">");
218                 out.print(item.child(descPos).child(0).label() + "</a></td>");
219             } else {
220                 out.println("<td> No description availible </td>");
221             }
222         } else {
223             out.println("<td> No title or description availible </td>");
224         }
225         out.print("</tr>");
226 
227     }
228 
229     /**
230      * Processes one XML RSS (version 2.0) feed from a given URL converting it
231      * into the corresponding HTML output file.
232      *
233      * @param url
```

```
234      *            the URL of the RSS feed
235      * @param file
236      *            the name of the HTML output file
237      * @param out
238      *            the output stream to report progress or errors
239      * @updates out.content
240      * @requires out.is_open
241      * @ensures <pre>
242      * [reads RSS feed from url, saves HTML document with table of news items
243      *   to file, appends to out.content any needed messages]
244      * </pre>
245      */
246     private static void processFeed(String url, String file, SimpleWriter out) {
247         // construct the xml tree objects
248         XMLTree root = new XMLTree1(url);
249         XMLTree channel = root.child(0);
250 
251         //simpleWriter object to write to an html file
252         SimpleWriter write = new SimpleWriter1L(file);
253 
254         // print the header of the html page
255         outputHeader(channel, write);
256 
257         // check to see if the tree is rss 2.0
258         if (root.label().equals("rss")
259                 && root.attributeValue("version").equals("2.0")) {
260             // loop through the channel tree and process each <item> tag
261             int i = 0;
262             while (i < channel.numberOfChildren()) {
263                 if (channel.child(i).label().equals("item")) {
264                     processItem(channel.child(i), write);
265                 }
266                 i++;
267             }
268         } else {
269             out.print("Link not valid RSS 2.0");
270         }
271 
272         // print the footer of the html page
273         outputFooter(write);
274         write.close();
275     }
276 
277     /**
278      * Main method.
279      *
280      * @param args
281      *            the command line arguments; unused here
282      */
283     public static void main(String[] args) {
284         SimpleReader in = new SimpleReader1L();
285         SimpleWriter out = new SimpleWriter1L();
286 
287         // get xml file name
288         out.print(
289                 "Enter the name of an XML file containing a list of URLs for RSSv2.0 feeds:
    ");
```

```
290         String xmlFile = in.nextLine();
291
292         // setup output steam to file
293         out.print(
294                 "Enter the name of an output file. Make sure to include the .html extension at
    the end!: ");
295         String fileName = in.nextLine();
296         SimpleWriter write = new SimpleWriter1L(fileName);
297
298         // print out title of Main feeds page
299         XMLTree feeds = new XMLTree1(xmlFile);
300         if (feeds.hasAttribute("title")) {
301             write.println("<html><head><title>" + feeds.attributeValue("title")
302                     + "</title></head><body><h2>"
303                     + feeds.attributeValue("title") + "</h2>");
304         } else {
305             write.println("<html><head><title>" + "No title availible"
306                     + "</title></head><body><h2>" + "No title avilible"
307                     + "</h2>");
308         }
309
310         // print out bulleted list of rss 2.0 pages
311         String feedUrl = "No url";
312         String feedName = "No name";
313         String feedFile = "No file";
314         write.println("<ul>");
315         int i = 0;
316         while (i < feeds.numberOfChildren()) {
317             XMLTree feed = feeds.child(i);
318
319             if (feed.hasAttribute("name")) {
320                 feedName = feed.attributeValue("name");
321             }
322             if (feed.hasAttribute("file")) {
323                 feedFile = feed.attributeValue("file");
324             }
325             if (feed.hasAttribute("url")) {
326                 feedUrl = feed.attributeValue("url");
327                 processFeed(feedUrl, feedFile, out);
328             }
329             write.println("<li><a href=\"" + feedFile + "\">" + feedName
330                     + "</a></li>");
331             i++;
332         }
333         // print out footer
334         write.println("</ul></body></html>");
335
336         // close input and output streams
337         in.close();
338         out.close();
339         write.close();
340     }
341
342 }
```