

```
1 import components.simplereader.SimpleReader;
2
3 /**
4  * Program to convert an XML RSS (version 2.0) feed from a given URL into the
5  * corresponding HTML output file.
6  *
7  * @author Vaishnavi Kasabwala
8  */
9 public final class RSSAggregator {
10
11     /**
12      * Private constructor so this utility class cannot be instantiated.
13      */
14     private RSSAggregator() {
15
16     }
17
18     /**
19      * Outputs the "opening" tags in the generated HTML file. These are the
20      * expected elements generated by this method:
21      *
22      * <html> <head> <title>the channel tag title as the page title</title>
23      * </head> <body>
24      * <h1>the page title inside a link to the <channel> link</h1>
25      * <p>
26      * the channel description
27      * </p>
28      * <table border="1">
29      * <tr>
30      * <th>Date</th>
31      * <th>Source</th>
32      * <th>News</th>
33      * </tr>
34      *
35      * @param channel
36      *         the channel element XMLTree
37      * @param out
38      *         the output stream
39      * @updates out.content
40      * @requires [the root of channel is a <channel> tag] and out.is_open
41      * @ensures out.content = #out.content * [the HTML "opening" tags]
42      */
43     private static void outputHeader(XMLTree channel, SimpleWriter out) {
44         assert channel != null : "Violation of: channel is not null";
45         assert out != null : "Violation of: out is not null";
46         assert channel.isTag() && channel.label().equals("channel") : ""
47             + "Violation of: the label root of channel is a <channel> tag";
48         assert out.isOpen() : "Violation of: out.is_open";
49
50         // title
51         int titleNum = getChildElement(channel, "title");
52         XMLTree title = channel.child(titleNum);
53
54         out.print("<html> <head> <title>");
55         if (title.numberOfChildren() > 0) {
56             out.print(title.child(0).label());
57         }
58     }
59 }
```

```

63         out.println("</title>");
64
65         out.println("</head> <body>");
66
67         //link
68         int linkNum = getChildElement(channel, "link");
69         XMLTree link = channel.child(linkNum);
70
71         out.print("<h1>");
72
73         out.print("<a href=\"\" + link.child(0).label() + \"\">");
74         if (title.numberOfChildren() > 0) {
75             out.print(title.child(0).label());
76         }
77         out.println("</a></h1>");
78
79         //description
80         int descriptionNum = getChildElement(channel, "description");
81         XMLTree description = channel.child(descriptionNum);
82
83         out.print("<p>");
84         if (description.numberOfChildren() > 0) {
85             out.print(description.child(0).label());
86         }
87         out.println("</p>");
88
89         out.println("<table border=\"1\">");
90         out.println("<tr><th>Date</th><th>Source</th><th>News</th></tr>");
91     }
92
93     /**
94      * Outputs the "closing" tags in the generated HTML file. These are the
95      * expected elements generated by this method:
96      *
97      * </table>
98      * </body> </html>
99      *
100     * @param out
101     *         the output stream
102     * @updates out.contents
103     * @requires out.is_open
104     * @ensures out.content = #out.content * [the HTML "closing" tags]
105     */
106     private static void outputFooter(SimpleWriter out) {
107         assert out != null : "Violation of: out is not null";
108         assert out.isOpen() : "Violation of: out.is_open";
109
110         out.println("</table>");
111         out.println("</body>");
112         out.println("</html>");
113     }
114
115     /**
116      * Finds the first occurrence of the given tag among the children of the
117      * given {@code XMLTree} and return its index; returns -1 if not found.
118      *
119      * @param xml

```

```

120     *           the {@code XMLTree} to search
121     * @param tag
122     *           the tag to look for
123     * @return the index of the first child of type tag of the {@code XMLTree}
124     *         or -1 if not found
125     * @requires [the label of the root of xml is a tag]
126     * @ensures <pre>
127     * getChildElement =
128     * [the index of the first child of type tag of the {@code XMLTree} or
129     * -1 if not found]
130     * </pre>
131     */
132     private static int getChildElement(XMLTree xml, String tag) {
133         assert xml != null : "Violation of: xml is not null";
134         assert tag != null : "Violation of: tag is not null";
135         assert xml.isTag() : "Violation of: the label root of xml is a tag";
136
137         int n = xml.numberOfChildren();
138         int index = -1;
139         int i = 0;
140
141         while (i < n && index == -1) {
142             if (xml.child(i).label().equals(tag)) {
143                 index = i;
144             }
145             i++;
146         }
147         return index;
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         int indexPubDate = getChildElement(item, "pubDate");
175         int indexSource = getChildElement(item, "source");
176         int indexTitle = getChildElement(item, "title");

```

```

177     int indexDescription = getChildElement(item, "description");
178     int indexLink = getChildElement(item, "link");
179
180     out.println("<tr>");
181
182     //publication date (if pubDate exists, it is required to have a child)
183     if (indexPubDate >= 0) {
184         out.println("<td>" + item.child(indexPubDate).child(0).label()
185             + "</td>");
186     } else {
187         out.println("<td>No date available</td>");
188     }
189
190     //source
191     if (indexSource >= 0
192         && item.child(indexSource).numberOfChildren() > 0) {
193         XMLTree source = item.child(indexSource);
194         out.print("<td>");
195         out.print("<a href=\"" + source.attributeValue("url") + "\">");
196         out.println(source.child(0).label() + "</a></td>");
197     } else {
198         out.println("<td>No source available</td>");
199     }
200
201     // link (if link exists, it is required to have a child)
202
203     if (indexTitle >= 0 && item.child(indexTitle).numberOfChildren() > 0) {
204         XMLTree title = item.child(indexTitle);
205         out.print("<td>");
206         if (indexLink >= 0) {
207             XMLTree link = item.child(indexLink);
208             out.print("<a href=\"" + link.child(0).label() + "\">");
209         }
210         out.println(title.child(0).label() + "</a></td>");
211     } else if (indexDescription >= 0
212         && item.child(indexDescription).numberOfChildren() > 0) {
213         XMLTree description = item.child(indexDescription);
214         out.print("<td>");
215         if (indexLink >= 0) {
216             XMLTree link = item.child(indexLink);
217             out.print("<a href=\"" + link.child(0).label() + "\">");
218         }
219         out.println(description.child(0).label() + "</a></td>");
220     } else {
221         out.print("<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 table 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 xml = new XMLTree1(url);
246         SimpleWriter fileOut = new SimpleWriter1L(file);
247
248         outputHeader(xml.child(0), fileOut);
249         outputFooter(fileOut);
250     }
251
252     /**
253     * Main method.
254     *
255     * @param args
256     *           the command line arguments; unused here
257     */
258     public static void main(String[] args) {
259         SimpleReader in = new SimpleReader1L();
260         SimpleWriter out = new SimpleWriter1L();
261
262         /*
263         * Input the source URL. https://news.yahoo.com/rss/.
264         */
265         out.print(
266             "Enter an XML file containing a list of URLs for RSS v2.0 feeds: ");
267         String url = in.nextLine();
268         XMLTree xml = new XMLTree1(url);
269
270         /*
271         * Asks user for the name of an output file including the .html
272         * extension.
273         */
274         out.print("Please enter the name of an output file: ");
275         String outFile = in.nextLine();
276         SimpleWriter file = new SimpleWriter1L(outFile);
277
278         String attribute = "";
279         if (xml.label().equals("rss")) {
280             attribute = xml.attributeValue("version");
281         }
282
283         XMLTree channel = xml.child(0);
284         if (attribute.equals("2.0")) {
285             outputHeader(channel, file);
286             // item tag and its children
287             for (int i = 0; i < channel.numberOfChildren(); i++) {
288                 if (channel.child(i).label().equals("item")) {
289                     XMLTree item = channel.child(i);
290                     processItem(item, file);

```

```
291         }
292     }
293
294     outputFooter(file);
295 }
296
297 xml.display();
298
299 in.close();
300 out.close();
301 }
302
303 }
```