

```

1 import components.simplereader.SimpleReader;
2 import components.simplereader.SimpleReader1L;
3 import components.simplewriter.SimpleWriter;
4 import components.simplewriter.SimpleWriter1L;
5 import components.xmltree.XMLTree;
6 import components.xmltree.XMLTree1;
7
8 /**
9  * Program to convert an XML RSS (version 2.0) feed from a given URL into the
10 * corresponding HTML output file.
11 *
12 * @author Vaishnavi Kasabwala
13 *
14 */
15 public final class RSSAggregator2 {
16
17     /**
18      * Private constructor so this utility class cannot be instantiated.
19      */
20     private RSSAggregator2() {
21     }
22
23     /**
24      * Outputs the "opening" tags in the generated HTML file. These are the
25      * expected elements generated by this method:
26      *
27      * <html> <head> <title>the channel tag title as the page title</title>
28      * </head> <body>
29      * <h1>the page title inside a link to the <channel> link</h1>
30      * <p>
31      * the channel description
32      * </p>
33      * <table border="1">
34      * <tr>
35      * <th>Date</th>
36      * <th>Source</th>
37      * <th>News</th>
38      * </tr>
39      *
40      * @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 feeds, SimpleWriter out) {
49         assert feeds != null : "Violation of: feeds is not null";
50         assert out != null : "Violation of: out is not null";
51         assert feeds.isTag() && feeds.label().equals("feeds") : ""
52             + "Violation of: the label root of feeds is a <feeds> tag";
53         assert out.isOpen() : "Violation of: out.is_open";
54
55         out.println("<html>");
56         out.println("<head>");
57         out.println("<title>" + feeds.attributeValue("title") + "</title>");

```

```

58         out.println("<body>");
59         out.println("<h2>" + feeds.attributeValue("title") + "</h2>");
60         out.println("<ul>");
61     }
62
63     /**
64      * Outputs the "closing" tags in the generated HTML file. These are the
65      * expected elements generated by this method:
66      *
67      * </table>
68      * </body> </html>
69      *
70      * @param out
71      *         the output stream
72      * @updates out.contents
73      * @requires out.is_open
74      * @ensures out.content = #out.content * [the HTML "closing" tags]
75      */
76     private static void outputFooter(SimpleWriter out) {
77         assert out != null : "Violation of: out is not null";
78         assert out.isOpen() : "Violation of: out.is_open";
79
80         out.println("</ul>");
81         out.println("</body>");
82         out.println("</html>");
83     }
84
85     /**
86      * Finds the first occurrence of the given tag among the children of the
87      * given {@code XMLTree} and return its index; returns -1 if not found.
88      *
89      * @param xml
90      *         the {@code XMLTree} to search
91      * @param tag
92      *         the tag to look for
93      * @return the index of the first child of type tag of the {@code XMLTree}
94      *         or -1 if not found
95      * @requires [the label of the root of xml is a tag]
96      * @ensures <pre>
97      * getChildElement =
98      * [the index of the first child of type tag of the {@code XMLTree} or
99      * -1 if not found]
100     * </pre>
101     */
102     private static int getChildElement(XMLTree xml, String tag) {
103         assert xml != null : "Violation of: xml is not null";
104         assert tag != null : "Violation of: tag is not null";
105         assert xml.isTag() : "Violation of: the label root of xml is a tag";
106
107         int n = xml.numberOfChildren();
108         int index = -1;
109         int i = 0;
110
111         while (i < n && index == -1) {
112             if (xml.child(i).label().equals(tag)) {
113                 index = i;
114             }

```

```

115         i++;
116     }
117     return index;
118 }
119
120 /**
121  * Processes one news item and outputs one table row. The row contains three
122  * elements: the publication date, the source, and the title (or
123  * description) of the item.
124  *
125  * @param item
126  *         the news item
127  * @param out
128  *         the output stream
129  * @updates out.content
130  * @requires [the label of the root of item is an <item> tag] and
131  *         out.is_open
132  * @ensures <pre>
133  * out.content = #out.content *
134  * [an HTML table row with publication date, source, and title of news item]
135  * </pre>
136  */
137 private static void processItem(XMLTree item, SimpleWriter out) {
138     assert item != null : "Violation of: item is not null";
139     assert out != null : "Violation of: out is not null";
140     assert item.isTag() && item.label().equals("item") : ""
141         + "Violation of: the label root of item is an <item> tag";
142     assert out.isOpen() : "Violation of: out.is_open";
143
144     int indexPubDate = getChildElement(item, "pubDate");
145     int indexSource = getChildElement(item, "source");
146     int indexTitle = getChildElement(item, "title");
147     int indexDescription = getChildElement(item, "description");
148     int indexLink = getChildElement(item, "link");
149
150     out.println("<tr>");
151
152     //publication date (if pubDate exists, it is required to have a child)
153     if (indexPubDate >= 0) {
154         out.println("<td>" + item.child(indexPubDate).child(0).label()
155             + "</td>");
156     } else {
157         out.println("<td>No date available</td>");
158     }
159
160     //source
161     if (indexSource >= 0
162         && item.child(indexSource).numberOfChildren() > 0) {
163         XMLTree source = item.child(indexSource);
164         out.print("<td>");
165         out.print("<a href=\"\" + source.attributeValue(\"url\") + \">");
166         out.println(source.child(0).label() + "</a></td>");
167     } else {
168         out.println("<td>No source available</td>");
169     }
170
171     // link (if link exists, it is required to have a child)

```

```

172
173     if (indexTitle >= 0 && item.child(indexTitle).numberOfChildren() > 0) {
174         XMLTree title = item.child(indexTitle);
175         out.print("<td>");
176         if (indexLink >= 0) {
177             XMLTree link = item.child(indexLink);
178             out.print("<a href=\"" + link.child(0).label() + "\">");
179         }
180         out.println(title.child(0).label() + "</a></td>");
181     } else if (indexDescription >= 0
182         && item.child(indexDescription).numberOfChildren() > 0) {
183         XMLTree description = item.child(indexDescription);
184         out.print("<td>");
185         if (indexLink >= 0) {
186             XMLTree link = item.child(indexLink);
187             out.print("<a href=\"" + link.child(0).label() + "\">");
188         }
189         out.println(description.child(0).label() + "</a></td>");
190     } else {
191         out.print("<td>No title available</td>");
192     }
193
194     out.println("</tr>");
195 }
196
197 /**
198  * Processes one XML RSS (version 2.0) feed from a given URL converting it
199  * into the corresponding HTML output file.
200  *
201  * @param url
202  *     the URL of the RSS feed
203  * @param file
204  *     the name of the HTML output file
205  * @param out
206  *     the output stream to report progress or errors
207  * @updates out.content
208  * @requires out.is_open
209  * @ensures <pre>
210  * [reads RSS feed from url, saves HTML document with table of news items
211  *  to file, appends to out.content any needed messages]
212  * </pre>
213  */
214 private static void processFeed(String url, String file, SimpleWriter out) {
215     SimpleWriter gen = new SimpleWriter1L(file);
216
217     assert url != null : "Violation of: url is not null";
218     assert gen != null : "Violation of: gen is not null";
219     assert gen.isOpen() : "Violation of: gen.is_open";
220
221     // header
222     // title
223     gen.print("<html> <head> <title>");
224     XMLTree channel = new XMLTree1(url);
225
226     int titleNum = getChildElement(channel, "title");
227     XMLTree title = channel.child(titleNum);
228

```

```

229     gen.print("<html> <head> <title>");
230     if (title.numberOfChildren() > 0) {
231         gen.print(title.child(0).label());
232     }
233     gen.println("DEMO 1 </title>");
234
235     gen.println("</head> <body>");
236
237     //link
238     int linkNum = getChildElement(channel, "link");
239     XMLTree link = channel.child(linkNum);
240
241     gen.print("<h1>");
242
243     gen.print("<a href=\"\" + link.child(0).label() + \"\">");
244     if (title.numberOfChildren() > 0) {
245         gen.print(title.child(0).label());
246     }
247     gen.println("</a></h1>");
248
249     //description
250     int descriptionNum = getChildElement(channel, "description");
251     XMLTree description = channel.child(descriptionNum);
252
253     gen.print("<p>");
254     if (description.numberOfChildren() > 0) {
255         gen.print(description.child(0).label());
256     }
257     gen.println("</p>");
258
259     gen.println("<table border=\"1\">");
260     gen.println("<tr><th>Date</th><th>Source</th><th>News</th></tr>");
261     // process item
262     for (int i = 0; i < channel.numberOfChildren(); i++) {
263         if (channel.child(i).label().equals("item")) {
264             XMLTree item = channel.child(i);
265             processItem(item, gen);
266         }
267     }
268 //footer
269     gen.println("</table>");
270     gen.println("</body>");
271     gen.println("</html>");
272 }
273
274 /**
275  * Main method.
276  *
277  * @param args
278  *     the command line arguments; unused here
279  */
280 public static void main(String[] args) {
281     SimpleReader in = new SimpleReader1L();
282     SimpleWriter out = new SimpleWriter1L();
283
284     /*
285     * Input the source URL.

```

```
286     * http://web.cse.ohio-state.edu/software/2221/web-sw1/assignments/
287     * projects/rss-aggregator/feeds.xml
288     */
289     out.print(
290         "Enter an XML file containing a list of URLs for RSS v2.0 feeds: ");
291     String url = in.nextLine();
292     XMLTree feeds = new XMLTree1(url);
293
294     /*
295     * Asks user for the name of an output file including the .html
296     * extension.
297     */
298     out.print("Please enter the name of an output file: ");
299     String outFile = in.nextLine();
300     SimpleWriter file = new SimpleWriter1L(outFile);
301
302     outputHeader(feeds, file);
303     // item tag and its children
304     for (int i = 0; i < feeds.numberOfChildren(); i++) {
305         if (feeds.child(i).label().equals("feed")) {
306             XMLTree feed = feeds.child(i);
307             String feedUrl = feed.attributeValue("url");
308             String feedFile = feed.attributeValue("file");
309             String feedName = feed.attributeValue("name");
310             file.println("<li><a href=\"\" + feedFile + \"\">\" + feedName
311                 + "</a></li>");
312             processFeed(feedUrl, feedFile, file);
313         }
314     }
315
316     outputFooter(file);
317
318     feeds.display();
319
320     in.close();
321     out.close();
322 }
323
324 }
```