

```
1 import components.simplereader.SimpleReader;
2
3 /**
4  * CSE 2221 Project #4. Program to convert an XML RSS (version 2.0) feed from a
5  * given URL into the corresponding HTML output file.
6  *
7  * @author Faye Leigh
8  */
9 public final class RSSReader {
10
11     /**
12      * Private constructor so this utility class cannot be instantiated.
13      */
14     private RSSReader() {
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         /*
51          * Get indices of title, link, and description tags
52          */
53         int titleIndex = getChildElement(channel, "title"),
54             linkIndex = getChildElement(channel, "link"),
55             descriptionIndex = getChildElement(channel, "description");
56
57         /*
58          * Get title, link, and description
59          */
60         String title, link, description;
61         if (channel.child(titleIndex).numberOfChildren() > 0) {
62             title = channel.child(titleIndex).child(0).label();
63         } else {
64             title = "No title";
65         }
66     }
67 }
```

```

70     }
71     if (channel.child(descriptionIndex).numberOfChildren() > 0) {
72         description = channel.child(descriptionIndex).child(0).label();
73     } else {
74         description = "No description";
75     }
76     link = channel.child(linkIndex).child(0).label();
77
78     /*
79     * Start printing to HTML file
80     */
81     out.println("<html><head><title>" + title + "</title></head><body>");
82     out.println("<h1>");
83     out.println("<a href=" + link + ">" + title + "</a>");
84     out.println("</h1>");
85     out.println("<p>");
86     out.println(description);
87     out.println("</p>");
88     out.println("<table border=\"1\">");
89     out.println("<tr>");
90     out.println("<th>Date</th>");
91     out.println("<th>Source</th>");
92     out.println("<th>News</th>");
93     out.println("</tr>");
94 }
95
96 /**
97  * Outputs the "closing" tags in the generated HTML file. These are the
98  * expected elements generated by this method:
99  *
100  * </table>
101  * </body> </html>
102  *
103  * @param out
104  *         the output stream
105  * @updates out.contents
106  * @requires out.is_open
107  * @ensures out.content = #out.content * [the HTML "closing" tags]
108  */
109 private static void outputFooter(SimpleWriter out) {
110     assert out != null : "Violation of: out is not null";
111     assert out.isOpen() : "Violation of: out.is_open";
112
113     out.println("</table>");
114     out.println("</body></html>");
115 }
116
117 /**
118  * Finds the first occurrence of the given tag among the children of the
119  * given {@code XMLTree} and return its index; returns -1 if not found.
120  *
121  * @param xml
122  *         the {@code XMLTree} to search
123  * @param tag
124  *         the tag to look for
125  * @return the index of the first child of type tag of the {@code XMLTree}
126  *         or -1 if not found
127  * @requires [the label of the root of xml is a tag]
128  * @ensures <pre>
129  *         getChildElement =
130  *         [the index of the first child of type tag of the {@code XMLTree} or
131  *         -1 if not found]
132  * </pre>
133  */
134 private static int getChildElement(XMLTree xml, String tag) {

```

```

135     assert xml != null : "Violation of: xml is not null";
136     assert tag != null : "Violation of: tag is not null";
137     assert xml.isTag() : "Violation of: the label root of xml is a tag";
138
139     int index = -1, i = 0;
140     boolean childFound = false;
141
142     /*
143     * Checks each child of xml and stops when matching tag label is found
144     */
145     while (i < xml.numberOfChildren() && !childFound) {
146         if (xml.child(i).isTag() && xml.child(i).label() == tag) {
147             index = i;
148             childFound = true;
149         }
150         i++;
151     }
152     return index;
153 }
154
155 * Processes one news item and outputs one table row. The row contains three
172 private static void processItem(XMLTree item, SimpleWriter out) {
173     assert item != null : "Violation of: item is not null";
174     assert out != null : "Violation of: out is not null";
175     assert item.isTag() && item.label().equals("item") : ""
176         + "Violation of: the label root of item is an <item> tag";
177     assert out.isOpen() : "Violation of: out.is_open";
178
179     int titleIndex = getChildElement(item, "title"),
180         linkIndex = getChildElement(item, "link"),
181         dateIndex = getChildElement(item, "pubDate"),
182         sourceIndex = getChildElement(item, "source"),
183         descriptionIndex = getChildElement(item, "description");
184
185     out.println("<tr>"); //Start of table row
186
187     /*
188     * Checks that there is a publication date tag and that it is not empty.
189     * Adds publication date to table if available
190     */
191     if (dateIndex > -1 && item.child(dateIndex).numberOfChildren() > 0) {
192         out.println(
193             "<td>" + item.child(dateIndex).child(0).label() + "</td>");
194     } else {
195         out.println("<td>No date available</td>");
196     }
197
198     /*
199     * Checks that there is a source tag and that it has a text child. If it
200     * has both, adds the text to table with hyperlink to url. If only url
201     * exists, adds url to table
202     */
203     if (sourceIndex > -1) {
204         if (item.child(sourceIndex).numberOfChildren() > 0) {
205             out.println("<td><a href=\"\"\"
206                 + item.child(sourceIndex).attributeValue("url") + "\">\"
207                 + item.child(sourceIndex).child(0).label()
208                 + "</a></td>");
209         } else {
210             out.println(
211                 "<td>\"\"\" + item.child(sourceIndex).attributeValue("url")
212                 + "\"</td>");
213         }
214     } else {

```

```

215         out.println("<td>No source available</td>");
216     }
217
218     /*
219     * Checks that there is a title and that it is not empty.
220     */
221     if (titleIndex > -1 && item.child(titleIndex).numberOfChildren() > 0) {
222
223         /*
224         * If link is available, adds title to table with hyperlink.
225         * Otherwise, adds title to table with no hyperlink
226         */
227         if (linkIndex > -1
228             && item.child(linkIndex).numberOfChildren() > 0) {
229             out.println("<td><a href=\""
230                 + item.child(linkIndex).child(0).label() + "\">"
231                 + item.child(titleIndex).child(0).label()
232                 + "</a></td>");
233         } else {
234             out.println("<td>" + item.child(titleIndex).child(0).label()
235                 + "</td>");
236         }
237
238         /*
239         * If no title, checks that there is a description and that it is
240         * not empty
241         */
242     } else if (descriptionIndex > -1
243         && item.child(descriptionIndex).numberOfChildren() > 0) {
244         /*
245         * If link is available, adds description to table with hyperlink.
246         * Otherwise, adds description to table with no hyperlink
247         */
248         if (linkIndex > -1
249             && item.child(linkIndex).numberOfChildren() > 0) {
250             out.println("<td><a href=\""
251                 + item.child(linkIndex).child(0).label() + "\">"
252                 + item.child(descriptionIndex).child(0).label()
253                 + "</a></td>");
254         } else {
255             out.println(
256                 "<td>" + item.child(descriptionIndex).child(0).label()
257                 + "</td>");
258         }
259
260         /*
261         * If no title or description and link is available, adds hyperlink
262         * to table. If none are available, no hyperlink is added
263         */
264     } else {
265         if (linkIndex > -1
266             && item.child(linkIndex).numberOfChildren() > 0) {
267             out.println("<td><a href=\""
268                 + item.child(linkIndex).child(0).label()
269                 + "\">No title available</a></td>");
270         } else {
271             out.println("<td>No title available</td>");
272         }
273     }
274 }
275
276 /**
277  * Main method.
278  *
279  * @param args

```

```
280      *           the command line arguments; unused here
281      */
282      public static void main(String[] args) {
283          SimpleReader in = new SimpleReader1L();
284          SimpleWriter out = new SimpleWriter1L();
285
286          out.println("Please enter the URL of an RSS 2.0 feed");
287          String url = in.nextLine();
288          out.println("Please enter a name for the output file (no extension)");
289          String outputFileName = in.nextLine() + ".html";
290
291          XMLTree xml = new XMLTree1(url);
292          SimpleWriter html = new SimpleWriter1L(outputFileName);
293          int channelIndex = getChildElement(xml, "channel");
294
295          outputHeader(xml.child(0), html);
296
297          /*
298           * Process each item tag in channel
299           */
300          for (int i = 0; i < xml.child(channelIndex).numberOfChildren(); i++) {
301              if (xml.child(channelIndex).child(i).label() == "item") {
302                  processItem(xml.child(channelIndex).child(i), html);
303              }
304          }
305
306          outputFooter(html);
307
308          html.close();
309          in.close();
310          out.close();
311      }
312  }
313 }
314
```