

```
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 RSSReader {
16
17     /**
18      * Private constructor so this utility class cannot be instantiated.
19      */
20     private RSSReader() {
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 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         // title
56         int titleNum = getChildElement(channel, "title");
57         XMLTree title = channel.child(titleNum);
```

```

58
59     out.print("<html> <head> <title>");
60     if (title.numberOfChildren() > 0) {
61         out.print(title.child(0).label());
62     }
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     if (link.numberOfChildren() > 0) {
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
80     //description
81     int descriptionNum = getChildElement(channel, "description");
82     XMLTree description = channel.child(descriptionNum);
83
84     out.print("<p>");
85     if (description.numberOfChildren() > 0) {
86         out.print(description.child(0).label());
87     }
88     out.println("</p>");
89
90     out.println("<table border=\"1\">");
91     out.println("<tr><th>Date</th><th>Source</th><th>News</th></tr>");
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     out.println("</table>");
112     out.println("</body>");
113     out.println("</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 n = xml.numberOfChildren();
139     int index = -1;
140     int i = 0;
141
142     while (i < n && index == -1) {
143         if (xml.child(i).label().equals(tag)) {
144             index = i;
145         }
146         i++;
147     }
148     return index;
149 }
150
151 /**
152  * Processes one news item and outputs one table row. The row contains three
153  * elements: the publication date, the source, and the title (or
154  * description) of the item.
155  *
156  * @param item
157  *         the news item
158  * @param out
159  *         the output stream
160  * @updates out.content
161  * @requires [the label of the root of item is an <item> tag] and
162  *         out.is_open
163  * @ensures <pre>
164  * out.content = #out.content *
165  * [an HTML table row with publication date, source, and title of news item]
166  * </pre>
167  */
168 private static void processItem(XMLTree item, SimpleWriter out) {
169     assert item != null : "Violation of: item is not null";
170     assert out != null : "Violation of: out is not null";
171     assert item.isTag() && item.label().equals("item") : ""

```

```
172         + "Violation of: the label root of item is an <item> tag";
173     assert out.isOpen() : "Violation of: out.is_open";
174
175     int indexPubDate = getChildElement(item, "pubDate");
176     int indexSource = getChildElement(item, "source");
177     int indexTitle = getChildElement(item, "title");
178     int indexDescription = getChildElement(item, "description");
179     int indexLink = getChildElement(item, "link");
180
181     out.println("<tr>");
182
183     //publication date (if pubDate exists, it is required to have a child)
184     if (indexPubDate >= 0) {
185         out.println("<td>" + item.child(indexPubDate).child(0).label()
186             + "</td>");
187     } else {
188         out.println("<td>No date available</td>");
189     }
190
191     //source
192     if (indexSource >= 0
193         && item.child(indexSource).numberOfChildren() > 0) {
194         XMLTree source = item.child(indexSource);
195         out.print("<td>");
196         out.print("<a href=\"" + source.attributeValue("url") + "\">");
197         out.println(source.child(0).label() + "</a></td>");
198     } else {
199         out.println("<td>No source available</td>");
200     }
201
202     // link (if link exists, it is required to have a child)
203
204     if (indexTitle >= 0 && item.child(indexTitle).numberOfChildren() > 0) {
205         XMLTree title = item.child(indexTitle);
206         out.print("<td>");
207         if (indexLink >= 0
208             && item.child(indexLink).numberOfChildren() > 0) {
209             XMLTree link = item.child(indexLink);
210             out.print("<a href=\"" + link.child(0).label() + "\">");
211         }
212         out.println(title.child(0).label() + "</a></td>");
213     } else if (item.child(indexDescription).numberOfChildren() > 0) {
214         XMLTree description = item.child(indexDescription);
215         out.print("<td>");
216         if (indexLink >= 0
217             && item.child(indexLink).numberOfChildren() > 0) {
218             XMLTree link = item.child(indexLink);
219             out.print("<a href=\"" + link.child(0).label() + "\">");
220         }
221         out.println(description.child(0).label() + "</a></td>");
222     } else {
223         out.print("<td>No title available</td>");
224     }
225
226     out.println("</tr>");
227 }
228
```

```
229  /**
230  * Main method.
231  *
232  * @param args
233  *      the command line arguments; unused here
234  */
235  public static void main(String[] args) {
236      SimpleReader in = new SimpleReader1L();
237      SimpleWriter out = new SimpleWriter1L();
238
239      /*
240      * Input the source URL. https://news.yahoo.com/rss/.
241      */
242      out.print("Enter the URL of an RSS 2.0 news feed: ");
243      String url = in.nextLine();
244      XMLTree xml = new XMLTree1L(url);
245
246      /*
247      * Asks user for the name of an output file including the .html
248      * extension.
249      */
250      out.print(
251          "Enter the the name of an output file including the \".html\" extension: ");
252      String outFile = in.nextLine();
253      SimpleWriter file = new SimpleWriter1L(outFile);
254
255      String attribute = "";
256      if (xml.label().equals("rss")) {
257          attribute = xml.attributeValue("version");
258      }
259
260      XMLTree channel = xml.child(0);
261      if (attribute.equals("2.0")) {
262          outputHeader(channel, file);
263          // item tag and its children
264          for (int i = 0; i < channel.numberOfChildren(); i++) {
265              if (channel.child(i).label().equals("item")) {
266                  XMLTree item = channel.child(i);
267                  processItem(item, file);
268              }
269          }
270
271          outputFooter(file);
272      }
273
274      xml.display();
275
276      in.close();
277      out.close();
278  }
279
280 }
```