

```

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 Michael Gorman
8  *
9  */
10 public final class RSSReader {
11
12     /**
13      * Private constructor so this utility class cannot be instantiated.
14      */
15     private RSSReader() {
16
17     }
18
19     /**
20      * Outputs the "opening" tags in the generated HTML file. These are the
21      * expected elements generated by this method:
22      *
23      * <html> <head> <title>the channel tag title as the page title</title>
24      * </head> <body>
25      * <h1>the page title inside a link to the <channel> link</h1>
26      * <p>
27      * the channel description
28      * </p>
29      * <table border="1">
30      * <tr>
31      * <th>Date</th>
32      * <th>Source</th>
33      * <th>News</th>
34      * </tr>
35      *
36      * @param channel
37      *         the channel element XMLTree
38      * @param out
39      *         the output stream
40      * @updates out.content
41      * @requires [the root of channel is a <channel> tag] and out.is_open
42      * @ensures out.content = #out.content * [the HTML "opening" tags]
43      */
44     private static void outputHeader(XMLTree channel, SimpleWriter out) {
45         assert channel != null : "Violation of: channel is not null";
46         assert out != null : "Violation of: out is not null";
47         assert channel.isTag() && channel.label().equals("channel") : ""
48             + "Violation of: the label root of channel is a <channel> tag";
49         assert out.isOpen() : "Violation of: out.is_open";
50
51         //initialize strings as desired values
52         String title = "Empty title";
53         String link = null;
54         String description = "No description";
55
56         //get indexes of each child element
57         int titleInd = getChildElement(channel, "title");
58         int linkInd = getChildElement(channel, "link");
59         int descInd = getChildElement(channel, "description");
60
61
62
63
64

```

```

65      /*
66      * the following if statements check if title and description have
67      * children and if so then they redefine the strings as the labels of
68      * their respective children
69      */
70      if (channel.child(titleInd).numberOfChildren() > 0) {
71          title = channel.child(titleInd).child(0).label();
72      }
73
74      if (channel.child(descInd).numberOfChildren() > 0) {
75          description = channel.child(descInd).child(0).label();
76      }
77
78      //no if statement necessary because link is mandatory under channel
79      link = channel.child(linkInd).child(0).label();
80
81      //print statements to html file
82      out.println("<html>");
83      out.println("<head>");
84      out.println("<title>" + title + "</title>");
85      out.println("</head>");
86      out.println("<body>");
87      out.println("<h1><a href=\"" + link + "\">" + title + "</a></h1>");
88      out.println("<p>" + description + "</p>");
89      out.println("<table border=\"" + link + "\">");
90
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      //closing statements
111      out.println("</table>");
112      out.print("</body> </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          //initialize index
137          int index = -1;
138          //store total num of children for xml tree
139          int childrenNum = xml.numberOfChildren();
140          /*
141           * while loop that will sift through all children of xml and if the
142           * label of that child matches string tag then the index of that child
143           * is stored in index
144           */
145          int i = 0;
146          while (index == -1 && i < childrenNum) {
147              if (xml.child(i).label().equals(tag)) {
148                  index = i;
149              }
150              i++;
151          }
152          //return index
153          return index;
154      }
155
156      /**
157       * Processes one news item and outputs one table row. The row contains three
158       * elements: the publication date, the source, and the title (or
159       * description) of the item.
160       *
161       * @param item
162       *         the news item
163       * @param out
164       *         the output stream
165       * @updates out.content
166       * @requires [the label of the root of item is an <item> tag] and
167       *         out.is_open
168       * @ensures <pre>
169       * out.content = #out.content *
170       * [an HTML table row with publication date, source, and title of news item]
171       * </pre>
172       */
173      private static void processItem(XMLTree item, SimpleWriter out) {
174          assert item != null : "Violation of: item is not null";
175          assert out != null : "Violation of: out is not null";
176          assert item.isTag() && item.label().equals("item") : ""
177              + "Violation of: the label root of item is an <item> tag";
178          assert out.isOpen() : "Violation of: out.is_open";
179
180          //initialize strings to default values
181          String pubDate = "No date available";
182          String source = "No source available";

```

```

183     String titleDescription = "No title available";
184     /*
185      * source url and link must be initialized as empty which is important
186      * later in if statements
187      */
188     String sourceURL = "";
189     String link = "";
190
191     //String description = "No description available";
192     //get indexes which each element occurs
193     int pubDateInd = getChildElement(item, "pubDate");
194     int sourceInd = getChildElement(item, "source");
195     int titleInd = getChildElement(item, "title");
196     int urlInd = getChildElement(item, "link");
197     int descInd = getChildElement(item, "description");
198     //if source exists
199     if (sourceInd != -1) {
200         source = item.child(sourceInd).child(0).label();
201         sourceURL = item.child(sourceInd).attributeValue("url");
202     }
203     /*
204      * if else statement that checks if title has a value and if not checks
205      * for a description and if not then it stays as no title available
206      */
207     if (titleInd != -1) {
208         if (item.child(titleInd).numberOfChildren() > 0) {
209             titleDescription = item.child(titleInd).child(0).label();
210         }
211     } else if (descInd != -1) {
212         if (item.child(descInd).numberOfChildren() > 0) {
213             titleDescription = item.child(descInd).child(0).label();
214         }
215     }
216
217     if (pubDateInd != -1) {
218         pubDate = item.child(pubDateInd).child(0).label();
219     }
220
221     if (urlInd != -1) {
222         link = item.child(urlInd).child(0).label();
223     }
224     /*
225      * if else if else statement that prints each item to a row and checks
226      * for what links are provided and changes print statements accordingly
227      */
228     out.println("<tr>");
229     if (!sourceURL.isEmpty() && !link.isEmpty()) {
230         out.println("<td>" + pubDate + "</td>");
231         out.println("<td>");
232         out.println("<a href=\"\" + sourceURL + \"\">" + source + "</a>");
233         out.println("</td>");
234         out.println("<td>");
235         out.println(
236             "<a href=\"\" + link + \"\">" + titleDescription + "</a>");
237         out.println("</td>");
238     } else if (!sourceURL.isEmpty()) {
239         out.println("<td>" + pubDate + "</td>");
240         out.println("<td>");
241         out.println("<a href=\"\" + sourceURL + \"\">" + source + "</a>");

```

```

242         out.println("</td>");
243         out.println("<td>");
244     } else {
245         out.println("<td>" + pubDate + "</td>");
246         out.println("<td>" + source + "</td>");
247         out.println("<td>" + titleDescription + "</td>");
248     }
249     out.println("</tr>");
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     //prompt user for RSS URL
263     out.print("Enter an RSS URL: ");
264     String url = in.nextLine();
265
266     //prompt user for html file name to write to
267     out.print("Enter an html file name: ");
268     String htmlFile = in.nextLine();
269     //initialize new output stream that writes to file
270     SimpleWriter fileOut = new SimpleWriter1L(htmlFile);
271
272     //initialize xml tree off inputed url
273     XMLTree rssWeb = new XMLTree1(url);
274
275     /*
276      * if the root is a tag and has version attribute of value 2.0 then
277      * proceed to write to html file as desired output headers first and if
278      * the tag label is "item" then process the item and add it to the html
279      * file if RSS file is not in proper format print error statement
280      */
281     if (rssWeb.isTag() && rssWeb.hasAttribute("version")
282         && rssWeb.attributeValue("version").equals("2.0")) {
283         XMLTree channel = rssWeb.child(0);
284         outputHeader(channel, fileOut);
285         for (int i = 0; i < channel.numberOfChildren(); i++) {
286             if (channel.child(i).label().equals("item")) {
287                 processItem(channel.child(i), fileOut);
288             }
289         }
290         outputFooter(fileOut);
291     } else {
292         out.println("Error with RSS file.");
293     }
294
295     //close input and output streams
296     fileOut.close();
297     in.close();
298     out.close();
299 }
300 }

```