```java
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;
import components.xmltree.XMLTree;
import components.xmltree.XMLTree1;

/**
 * Program to convert an XML RSS (version 2.0) feed from a
given URL into the
 * corresponding HTML output file.
 *
 * @author Jeng Zhuang
 *
 */
public final class RSSReader {

    /**
     * Private constructor so this utility class cannot be
instantiated.
     */
    private RSSReader() {
    }

    /**
     * Outputs the "opening" tags in the generated HTML
file. These are the
     * expected elements generated by this method:
     *
     * <html> <head> <title>the channel tag title as the
page title</title>
     * </head> <body>
     * <h1>the page title inside a link to the <channel>
link</h1>
     * <p>
     * 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         // Extract channel title
56         int titleIndex = getChildElement(channel,
  "title");
57         String titleText = "Empty Title";
58         if (titleIndex != -1) {
59             XMLTree titleTag = channel.child(titleIndex);
```

```
60              if (titleTag.numberOfChildren() > 0) {
61                  // Get the text inside the <title> tag
62                  titleText = titleTag.child(0).label();
63
64              }
65          }
66
67          // Extract channel link
68          int linkIndex = getChildElement(channel, "link");
69          XMLTree linkTag = channel.child(linkIndex);
70          // Get the text inside the <link> tag
71          String link = linkTag.child(0).label();
72
73          // Extract channel description
74          int descIndex = getChildElement(channel,
    "description");
75          String descText = "No description";
76          if (descIndex != -1) {
77              XMLTree descTag = channel.child(descIndex);
78              if (descTag.numberOfChildren() > 0) {
79                  // Get the text inside the <description>
    tag
80                  descText = descTag.child(0).label();
81              }
82          }
83
84          // Output HTML header
85          out.println("<html>");
86          out.println("<head>");
87          out.println("<title>" + titleText + "</title>");
88          out.println("</head>");
89          out.println("<body>");
90          // Add the channel title as a link
91          out.println("<h1><a href=\"" + link + "\">" +
    titleText + "</a></h1>");
92          // Add the channel description
```

```
 93         out.println("<p>" + descText + "</p>");
 94         // Start the table
 95         out.println("<table border=\"1\">");
 96         out.println("<tr>");
 97         // Add table headers
 98         out.println("<th>Date</th>");
 99         out.println("<th>Source</th>");
100         out.println("<th>News</th>");
101         out.println("</tr>");
102     }
103
104     /**
105      * Outputs the "closing" tags in the generated HTML
    file. These are the
106      * expected elements generated by this method:
107      *
108      * </table>
109      * </body> </html>
110      *
111      * @param out
112      *            the output stream
113      * @updates out.contents
114      * @requires out.is_open
115      * @ensures out.content = #out.content * [the HTML
    "closing" tags]
116      */
117     private static void outputFooter(SimpleWriter out) {
118         assert out != null : "Violation of: out is not
    null";
119         assert out.isOpen() : "Violation of: out.is_open";
120
121         out.println("</table>"); // Close the table
122         out.println("</body>"); // Close the body
123         out.println("</html>"); // Close the HTML document
124     }
125
```

```
126     /**
127      * Finds the first occurrence of the given tag among
    the children of the
128      * given {@code XMLTree} and return its index; returns
    -1 if not found.
129      *
130      * @param xml
131      *            the {@code XMLTree} to search
132      * @param tag
133      *            the tag to look for
134      * @return the index of the first child of type tag of
    the {@code XMLTree}
135      *         or -1 if not found
136      * @requires [the label of the root of xml is a tag]
137      * @ensures <pre>
138      * getChildElement =
139      *  [the index of the first child of type tag of the
    {@code XMLTree} or
140      *   -1 if not found]
141      * </pre> not found]
142      */
143     private static int getChildElement(XMLTree xml, String
    tag) {
144         assert xml != null : "Violation of: xml is not
    null";
145         assert tag != null : "Violation of: tag is not
    null";
146         assert xml.isTag() : "Violation of: the label root
    of xml is a tag";
147
148         int index = -1; // Initialize with -1
149         int i = 0;
150         while (i < xml.numberOfChildren() && index == -1)
    {
151             // Continue until a match is found or all
    children are checked
```

```
152             XMLTree child = xml.child(i);
153             if (child.isTag() &&
    child.label().equals(tag)) {
154                 index = i; // Store the index of the first
    matching child
155             }
156             i++;
157         }
158         return index; // Return the result after the loop
159     }
160
161     /**
162      * Processes one news item and outputs one table row.
    The row contains three
163      * elements: the publication date, the source, and the
    title (or
164      * description) of the item.
165      *
166      * @param item
167      *            the news item
168      * @param out
169      *            the output stream
170      * @updates out.content
171      * @requires [the label of the root of item is an
    <item> tag] and
172      *           out.is_open
173      * @ensures <pre>
174      * out.content = #out.content *
175      *   [an HTML table row with publication date, source,
    and title of news item]
176      * </pre>
177      */
178     private static void processItem(XMLTree item,
    SimpleWriter out) {
179         assert item != null : "Violation of: item is not
    null";
```

```
180         assert out != null : "Violation of: out is not
    null";
181         assert item.isTag() && item.label().equals("item")
182                 : "" + "Violation of: the label root of
    item is an <item> tag";
183         assert out.isOpen() : "Violation of: out.is_open";
184
185         // Process publication date
186         String date = "No date available";
187         int pubDateIndex = getChildElement(item,
    "pubDate");
188         if (pubDateIndex != -1) {
189             XMLTree pubDateTag = item.child(pubDateIndex);
190             // Get the text inside the <pubDate> tag
191             date = pubDateTag.child(0).label();
192         }
193
194         // Process source
195         String source = "No source available";
196         int sourceIndex = getChildElement(item, "source");
197         if (sourceIndex != -1) {
198             XMLTree sourceTag = item.child(sourceIndex);
199             // Get the URL attribute
200             String url = sourceTag.attributeValue("url");
201             String text = "";
202             if (sourceTag.numberOfChildren() > 0) {
203                 // Get the text inside the <source> tag
204                 text = sourceTag.child(0).label();
205             } else {
206                 text = url; // Use the URL as the text if
    no text is available
207             }
208             source = "<a href=\"" + url + "\">" + text +
    "</a>"; // Create a hyperlink
209         }
210
```

```
211         // Process news text and link
212         String newsText = "";
213         int titleIndex = getChildElement(item, "title");
214         if (titleIndex != -1) {
215             XMLTree titleTag = item.child(titleIndex);
216             if (titleTag.numberOfChildren() > 0) {
217                 newsText = titleTag.child(0).label();
218             }
219         }
220 
221         if (newsText.isEmpty()) {
222             int descIndex = getChildElement(item,
    "description");
223             if (descIndex != -1) {
224                 XMLTree descTag = item.child(descIndex);
225                 if (descTag.numberOfChildren() > 0) {
226                     // Get the text inside the <title> tag
227                     newsText = descTag.child(0).label();
228                 }
229             }
230         }
231 
232         if (newsText.isEmpty()) {
233             newsText = "No title available";
234             // Default text if no title or description is
    found
235         }
236 
237         int linkIndex = getChildElement(item, "link");
238         if (linkIndex != -1) {
239             XMLTree linkTag = item.child(linkIndex);
240             // Get the text inside the <link> tag
241             String link = linkTag.child(0).label();
242             // Create a hyperlink
243             newsText = "<a href=\"" + link + "\">" +
    newsText + "</a>";
```

```
244         }
245 
246         // Output table row
247         out.println("<tr>");
248         out.println("<td>" + date + "</td>"); // Add the
    publication date
249         out.println("<td>" + source + "</td>"); // Add the
    source
250         out.println("<td>" + newsText + "</td>"); // Add
    the news title or description
251         out.println("</tr>");
252     }
253 
254     /**
255      * Main method.
256      *
257      * @param args
258      *             the command line arguments; unused here
259      */
260     public static void main(String[] args) {
261         SimpleReader in = new SimpleReader1L();
262         SimpleWriter out = new SimpleWriter1L();
263 
264         // Prompt for RSS feed URL
265         out.print("Enter the URL of an RSS 2.0 feed: ");
266         String url = in.nextLine();
267 
268         // Read XML from URL
269         XMLTree xml = new XMLTree1(url);
270 
271         // Validate RSS 2.0
272         if (!xml.label().equals("rss") || !
    xml.hasAttribute("version")
273                 || !
    xml.attributeValue("version").equals("2.0")) {
274             out.println("Error: Invalid RSS 2.0 feed.");
```

```
275             in.close();
276             out.close();
277             return;
278         }
279 
280         // Get channel element
281         XMLTree channel = xml.child(0);
282 
283         // Prompt for output file
284         out.print("Enter the output HTML file name: ");
285         String outputFile = in.nextLine();
286         SimpleWriter fileOut = new
    SimpleWriter1L(outputFile);
287 
288         // Generate HTML
289         outputHeader(channel, fileOut);
290 
291         // Process each item in the channel
292         for (int i = 0; i < channel.numberOfChildren(); i+
    +) {
293             XMLTree child = channel.child(i);
294             if (child.isTag() &&
    child.label().equals("item")) {
295                 // Process each <item> tag
296                 processItem(child, fileOut);
297             }
298         }
299 
300         outputFooter(fileOut);
301 
302         // Close resources
303         in.close();
304         out.close();
305         fileOut.close();
306     }
307 
```

```
308 }
309
```