```java
import components.simplereader.SimpleReader;

/**
 * Program to convert an XML RSS (version 2.0) feed from a given URL into the
 * corresponding HTML output file.
 *
 * @author David Park
 *
 */
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
     * </p>
     * <table border="1">
     * <tr>
     * <th>Date</th>
     * <th>Source</th>
     * <th>News</th>
     * </tr>
     *
     * @param channel
     *            the channel element XMLTree
     * @param out
     *            the output stream
     * @updates out.content
     * @requires [the root of channel is a <channel> tag] and out.is_open
     * @ensures out.content = #out.content * [the HTML "opening" tags]
     */
    private static void outputHeader(XMLTree channel, SimpleWriter out) {
        assert channel != null : "Violation of: channel is not null";
        assert out != null : "Violation of: out is not null";
        assert channel.isTag() && channel.label().equals("channel") : ""
                + "Violation of: the label root of channel is a <channel> tag";
        assert out.isOpen() : "Violation of: out.is_open";

        // get the channel title
        int titleIndex = getChildElement(channel, "title");
        String title = "Empty Title"; // set as default
        if (titleIndex >= 0
                && channel.child(titleIndex).numberOfChildren() > 0) {
            title = channel.child(titleIndex).child(0).label();
        }

```

```
63          // extract the channel link
64          int linkIndex = getChildElement(channel, "link");
65          String link = ""; // set as default
66
67          link = channel.child(linkIndex).child(0).label();
68
69          //declare descriptionIndex with index of description
70          int descriptionIndex = getChildElement(channel, "description");
71          String description = "";
72          //check if description exists and has atleast one child
73          if (descriptionIndex >= 0
74                  && channel.child(descriptionIndex).numberOfChildren() > 0) {
75              description = channel.child(descriptionIndex).child(0).label();
76          }
77
78          //print out rss/html info
79          out.println("<html>");
80          out.println("<head>");
81          out.println("<title>" + title + "</title>");
82          out.println("</head>");
83          out.println("<body>");
84          out.println("<h1><a href=\"" + link + "\">" + title + "</a></h1>");
85          out.println("<p>" + description + "</p>");
86          out.println("<table border=\"1\">");
87          out.println("<tr>");
88          out.println("<th>Date</th>");
89          out.println("<th>Source</th>");
90          out.println("<th>News</th>");
91          out.println("</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         // Output the closing HTML tags
112         out.println("</table>");
113         out.println("</body>");
114         out.println("</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         // initialize as if the tag is not found.
140         int result = -1;
141
142         // iterate through children of XMLTree
143         for (int i = 0; i < xml.numberOfChildren() && result == -1; i++) {
144             if (xml.child(i).label().equals(tag)) {
145                 // if child is found with matching tag, update result with its index.
146                 result = i;
147             }
148         }
149         return result;
150     }
151
152     /**
153      * Processes one news item and outputs one table row. The row contains three
154      * elements: the publication date, the source, and the title (or
155      * description) of the item.
156      *
157      * @param item
158      *            the news item
159      * @param out
160      *            the output stream
161      * @updates out.content
162      * @requires [the label of the root of item is an <item> tag] and
163      *           out.is_open
164      * @ensures <pre>
165      * out.content = #out.content *
166      *   [an HTML table row with publication date, source, and title of news item]
167      * </pre>
168      */
169     private static void processItem(XMLTree item, SimpleWriter out) {
170         assert item != null : "Violation of: item is not null";
171         assert out != null : "Violation of: out is not null";
172         assert item.isTag() && item.label().equals("item") : ""
173                 + "Violation of: the label root of item is an <item> tag";
174         assert out.isOpen() : "Violation of: out.is_open";
175
176         // initialize default values
```

```
177         String date = "No Date is Available";
178         String source = "No Source is Available";
179         String title = "No Title or Description Available";
180         String link = ""; // Initialize an empty string for the link
181 
182         // pull date
183         int dateIndex = getChildElement(item, "pubDate");
184         if (dateIndex >= 0) {
185             date = item.child(dateIndex).child(0).label();
186         }
187 
188         // pull source
189         int sourceIndex = getChildElement(item, "source");
190         if (sourceIndex >= 0 && item.child(sourceIndex).hasAttribute("url")
191                 && item.child(sourceIndex).numberOfChildren() > 0) {
192             String sourceUrl = item.child(sourceIndex).attributeValue("url");
193             String sourceText = item.child(sourceIndex).child(0).label();
194             source = "<a href=\"" + sourceUrl + "\">" + sourceText + "</a>";
195             // Wrap source in a link
196         }
197 
198         // pull title
199         int titleIndex = getChildElement(item, "title");
200         int linkIndex = getChildElement(item, "link");
201         if (titleIndex >= 0 && item.child(titleIndex).numberOfChildren() > 0) {
202             title = item.child(titleIndex).child(0).label(); // check if title available
203             if (linkIndex >= 0
204                     && item.child(linkIndex).numberOfChildren() > 0) {
205                 //pull title link if available
206                 link = item.child(linkIndex).child(0).label();
207                 title = "<a href=\"" + link + "\">" + title + "</a>";
208                 // Wrap title in a link
209             }
210         }
211 
212         // pull desc index
213         int descIndex = getChildElement(item, "description");
214         if (descIndex >= 0 && item.child(descIndex).numberOfChildren() > 0) {
215             title = item.child(descIndex).child(0).label();
216         }
217 
218         //output the final values
219         out.println("<tr>");
220         out.println("<td>" + date + "</td>");
221         out.println("<td>" + source + "</td>");
222         out.println("<td>" + title + "</td>");
223         out.println("</tr>");
224 
225     }
226 
227     /**
228      * Main method.
229      *
230      * @param args
231      *            the command line arguments; unused here
232      */
233     public static void main(String[] args) {
```

```
234         SimpleReader in = new SimpleReader1L();
235         SimpleWriter out = new SimpleWriter1L();
236
237         out.println("Enter URL of RSS 2.0 News Feed: "); // ask user for RSS 2.0
238         String url = in.nextLine(); // Store RSS 2.0 as a String to send as argument later
239
240         XMLTree xml = new XMLTree1(url); // send String "url" as argument to XMLTree
241
242         //Check if the given XML Doc is a valid RSS2.0 News Feed
243         if (xml.label().equals("rss") && xml.hasAttribute("version")
244                 && xml.attributeValue("version").equals("2.0")) {
245             out.println("Enter name of output file");
246             String fileName = in.nextLine();
247
248             SimpleWriter fileOut = new SimpleWriter1L(fileName);
249
250             XMLTree channel = xml.child(0);
251             outputHeader(channel, fileOut);
252
253             //iterate through all children of channel
254             for (int i = 0; i < channel.numberOfChildren(); i++) {
255                 //check if label is item
256                 if (channel.child(i).label().equals("item")) {
257                     //process item element with processItem
258                     processItem(channel.child(i), fileOut);
259                 }
260             }
261             outputFooter(fileOut);
262
263             fileOut.close();
264
265         } else {
266             out.println("URL is not a valid RSS 2.0 Feed");
267         }
268         in.close();
269         out.close();
270     }
271 }
272
```