```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 Shyam Sai Bethina
 *
 */
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
```

```
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         int linkIndex = getChildElement(channel, "link");
56         String link = channel.child(linkIndex).child(0).label();
57
58         String title = "Empty Title";
59         int titleIndex = getChildElement(channel, "title");
60         if (channel.child(titleIndex).numberOfChildren() > 0) {
61             title = channel.child(titleIndex).child(0).label();
62         }
63
64         String description = "No description";
65         int descriptionIndex = getChildElement(channel,
    "description");
66         if (channel.child(descriptionIndex).numberOfChildren() >
    0) {
67             description =
    channel.child(descriptionIndex).child(0).label();
68         }
69
70         out.println("<html>");
71         out.println("<head>");
72         out.println("<title>" + title + "</title>");
73         out.println("</head>");
74         out.println("<body>");
75         out.println(" <h1><a href='" + link + "'>" + title + "</
```

```
   a></h1>");
 76         out.println(" <p>" + description + "</p>");
 77         out.println(" <table border=1>");
 78         out.println("  <tr>");
 79         out.println("   <th>Date</th>");
 80         out.println("   <th>Source</th>");
 81         out.println("   <th>News</th>");
 82         out.print("  </tr>");
 83
 84     }
 85
 86     /**
 87      * Outputs the "closing" tags in the generated HTML file.
   These are the
 88      * expected elements generated by this method:
 89      *
 90      * </table>
 91      * </body> </html>
 92      *
 93      * @param out
 94      *            the output stream
 95      * @updates out.contents
 96      * @requires out.is_open
 97      * @ensures out.content = #out.content * [the HTML "closing"
   tags]
 98      */
 99     private static void outputFooter(SimpleWriter out) {
100         assert out != null : "Violation of: out is not null";
101         assert out.isOpen() : "Violation of: out.is_open";
102
103         out.println(" </table>");
104         out.println("</body>");
105         out.print("</html>");
106     }
107
108     /**
109      * Finds the first occurrence of the given tag among the
   children of the
110      * given {@code XMLTree} and return its index; returns -1 if
   not found.
111      *
112      * @param xml
113      *            the {@code XMLTree} to search
114      * @param tag
```

```
115     *            the tag to look for
116     * @return the index of the first child of type tag of the
    {@code XMLTree}
117     *         or -1 if not found
118     * @requires [the label of the root of xml is a tag]
119     * @ensures <pre>
120     * getChildElement =
121     *  [the index of the first child of type tag of the {@code
    XMLTree} or
122     *   -1 if not found]
123     * </pre>
124     */
125    private static int getChildElement(XMLTree xml, String tag) {
126        assert xml != null : "Violation of: xml is not null";
127        assert tag != null : "Violation of: tag is not null";
128        assert xml.isTag() : "Violation of: the label root of xml
    is a tag";
129
130        int index = -1;
131        boolean found = false;
132        for (int i = 0; i < xml.numberOfChildren() && !found; i++)
    {
133            if (tag.equals(xml.child(i).label())) {
134                index = i;
135                found = true;
136            }
137        }
138
139        return index;
140    }
141
142    /**
143     * Processes one news item and outputs one table row. The row
    contains three
144     * elements: the publication date, the source, and the title
    (or
145     * description) of the item.
146     *
147     * @param item
148     *            the news item
149     * @param out
150     *            the output stream
151     * @updates out.content
152     * @requires [the label of the root of item is an <item> tag]
```

```
    and
153      *           out.is_open
154      * @ensures <pre>
155      * out.content = #out.content *
156      *   [an HTML table row with publication date, source, and
    title of news item]
157      * </pre>
158      */
159     private static void processItem(XMLTree item, SimpleWriter
    out) {
160         assert item != null : "Violation of: item is not null";
161         assert out != null : "Violation of: out is not null";
162         assert item.isTag() && item.label().equals("item") : ""
163                 + "Violation of: the label root of item is an
    <item> tag";
164         assert out.isOpen() : "Violation of: out.is_open";
165
166         if (item.numberOfChildren() > 0) {
167             String pubDate = "No date available";
168             String titleOrDescr = "No title available";
169             String source = "No source available.";
170             String link = "";
171             String url = "";
172
173             int indexForTitle;
174             if (getChildElement(item, "title") != -1) {
175                 indexForTitle = getChildElement(item, "title");
176             } else {
177                 indexForTitle = getChildElement(item,
    "description");
178             }
179
180             if (item.child(indexForTitle).numberOfChildren() > 0)
    {
181                 titleOrDescr =
    item.child(indexForTitle).child(0).label();
182             }
183
184             int indexForDate;
185             if (getChildElement(item, "pubDate") != -1) {
186                 indexForDate = getChildElement(item, "pubDate");
187                 if (item.child(indexForDate).numberOfChildren() >
    0) {
188                     pubDate =
```

```
    item.child(indexForDate).child(0).label();
189                 }
190             }
191 
192             int indexForSource;
193             if (getChildElement(item, "source") != -1) {
194                 indexForSource = getChildElement(item, "source");
195                 if (item.child(indexForSource).numberOfChildren()
    > 0) {
196                     source =
    item.child(indexForSource).child(0).label();
197                     url =
    item.child(indexForSource).attributeValue("url");
198                 }
199             }
200 
201             int indexForLink;
202             if (getChildElement(item, "link") != -1) {
203                 indexForLink = getChildElement(item, "link");
204                 if (item.child(indexForLink).numberOfChildren() >
    0) {
205                     link =
    item.child(indexForLink).child(0).label();
206                 }
207             }
208 
209             out.println("  <tr>");
210             out.println("   <th>" + pubDate + "</th>");
211             out.println("   <th><a href='" + url + "'>" + source +
    "</a></th>");
212             out.println("   <th><a href='" + link + "'>" +
    titleOrDescr
213                     + "</a></th>");
214             out.print("  </tr>");
215 
216         }
217     }
218 
219     /**
220      * Main method.
221      *
222      * @param args
223      *            the command line arguments; unused here
224      */
```

```
225     public static void main(String[] args) {
226         SimpleReader in = new SimpleReader1L();
227         SimpleWriter out = new SimpleWriter1L();
228 
229         out.println("Input URL of an RSS 2.0 feed: ");
230         String userUrl = in.nextLine();
231         XMLTree tree = new XMLTree1(userUrl);
232         out.println("Input name of HTML file: ");
233         String userHTML = in.nextLine();
234         SimpleWriter outHTML = new SimpleWriter1L(userHTML +
    ".html");
235 
236         if (tree.label().equals("rss")
237                 && tree.attributeValue("version").equals("2.0")) {
238             XMLTree channel = tree.child(0);
239             outputHeader(channel, outHTML);
240             for (int i = 0; i < channel.numberOfChildren(); i++) {
241                 if (channel.child(i).label().equals("item")) {
242                     processItem(channel.child(i), outHTML);
243                 }
244             }
245         } else {
246             out.println("Sorry, not RSS or version 2.0");
247         }
248 
249         outputFooter(outHTML);
250 
251         in.close();
252         out.close();
253         outHTML.close();
254     }
255 
256 }
```