```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 Vaishnavi Kasabwala
 *
 */
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";

        // title
        int titleNum = getChildElement(channel, "title");
        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          //description
 68          int descriptionNum = getChildElement(channel, "description");
 69          XMLTree description = channel.child(descriptionNum);
 70
 71          out.print("<h1><p>");
 72          if (title.numberOfChildren() > 0) {
 73              out.print(description.child(0).label());
 74          }
 75          out.println("</p></h1>");
 76
 77          out.println("<table border=\"1\">");
 78          out.println(
 79                  "<table> <tr><th>Date</th><th>Source</th><th>News</th> </tr>");
 80      }
 81
 82      /**
 83       * Outputs the "closing" tags in the generated HTML file. These are the
 84       * expected elements generated by this method:
 85       *
 86       * </table>
 87       * </body> </html>
 88       *
 89       * @param out
 90       *            the output stream
 91       * @updates out.contents
 92       * @requires out.is_open
 93       * @ensures out.content = #out.content * [the HTML "closing" tags]
 94       */
 95      private static void outputFooter(SimpleWriter out) {
 96          assert out != null : "Violation of: out is not null";
 97          assert out.isOpen() : "Violation of: out.is_open";
 98
 99          out.println("</table>");
100          out.println("</body>");
101          out.println("</html>");
102      }
103
104      /**
105       * Finds the first occurrence of the given tag among the children of the
106       * given {@code XMLTree} and return its index; returns -1 if not found.
107       *
108       * @param xml
109       *            the {@code XMLTree} to search
110       * @param tag
111       *            the tag to look for
112       * @return the index of the first child of type tag of the {@code XMLTree}
113       *         or -1 if not found
114       * @requires [the label of the root of xml is a tag]
```

```
115      * @ensures <pre>
116      * getChildElement =
117      *  [the index of the first child of type tag of the {@code XMLTree} or
118      *   -1 if not found]
119      * </pre>
120      */
121     private static int getChildElement(XMLTree xml, String tag) {
122         assert xml != null : "Violation of: xml is not null";
123         assert tag != null : "Violation of: tag is not null";
124         assert xml.isTag() : "Violation of: the label root of xml is a tag";
125
126         int i = xml.numberOfChildren();
127         int index = -1;
128
129         while (i > 0 && index < 0) {
130             i--;
131             if (xml.child(i).label().equals(tag)) {
132                 index = i;
133             }
134         }
135         return index;
136
137         /*
138          * int index = -1; int numChildren = xml.numberOfChildren();
139          *
140          * for (int i = 0; xml.isTag() && xml.label().equals(tag) || i <
141          * numChildren; i++) { if (xml.child(i).label().equals(tag)) { index =
142          * i; } } return index;
143          */
144     }
145
146     /**
147      * Processes one news item and outputs one table row. The row contains three
148      * elements: the publication date, the source, and the title (or
149      * description) of the item.
150      *
151      * @param item
152      *            the news item
153      * @param out
154      *            the output stream
155      * @updates out.content
156      * @requires [the label of the root of item is an <item> tag] and
157      *           out.is_open
158      * @ensures <pre>
159      * out.content = #out.content *
160      *   [an HTML table row with publication date, source, and title of news item]
161      * </pre>
162      */
163     private static void processItem(XMLTree item, SimpleWriter out) {
164         assert item != null : "Violation of: item is not null";
165         assert out != null : "Violation of: out is not null";
166         assert item.isTag() && item.label().equals("item") : ""
167                 + "Violation of: the label root of item is an <item> tag";
168         assert out.isOpen() : "Violation of: out.is_open";
169
170         for (int i = 0; i < item.numberOfChildren(); i++) {
171
```

```
172             if (item.child(i).label().equals("pubDate")
173                     || item.child(i).label().equals("source")
174                     || item.child(i).label().equals("description")) {
175                 out.println("<tr>");
176 
177                 //publication date
178                 if (item.child(i).label().equals("pubDate")
179                         && item.child(i).numberOfChildren() > 0) {
180                     out.println(
181                             "<td>" + item.child(i).child(0).label() + "</td>");
182                 }
183 
184                 //source
185                 if (item.child(i).label().equals("source")) {
186                     if (item.child(i).numberOfChildren() > 0) {
187                         out.println("<td><a href=\"url\">"
188                                 + item.child(i).child(0).label() + "</a></td>");
189                     } else {
190                         out.println("No source available");
191                     }
192                 }
193 
194                 // link
195                 if (item.child(i).label().equals("title")
196                         || item.child(i).label().equals("description")) {
197                     if (item.child(i).numberOfChildren() > 0) {
198                         out.println("<td>" + item.child(i).child(0).label()
199                                 + "</td>");
200                     }
201                 }
202 
203                 out.println("</tr>");
204             }
205         }
206 
207     }
208 
209     /**
210      * Main method.
211      *
212      * @param args
213      *            the command line arguments; unused here
214      */
215     public static void main(String[] args) {
216         SimpleReader in = new SimpleReader1L();
217         SimpleWriter out = new SimpleWriter1L();
218 
219         /*
220          * Input the source URL. https://news.yahoo.com/rss/.
221          */
222         out.print("Enter the URL of an RSS 2.0 news feed: ");
223         String url = in.nextLine();
224         XMLTree xml = new XMLTree1(url);
225 
226         /*
227          * Asks user for the name of an output file including the .html
228          * extension.
```

```
229          */
230         out.print(
231                 "Enter the the name of an output file including the \".html\" extension: ");
232         String outFile = in.nextLine();
233         SimpleWriter file = new SimpleWriter1L(outFile);
234
235         String attribute = "";
236         if (xml.label().equals("rss")) {
237             attribute = xml.attributeValue("version");
238         }
239
240         XMLTree channel = xml.child(0);
241         if (attribute.equals("2.0")) {
242             outputHeader(channel, file);
243             // item tag and its children
244             int itemNum = getChildElement(channel, "item");
245             XMLTree item = channel.child(itemNum);
246             processItem(item, file);
247
248             outputFooter(file);
249
250             xml.display();
251         }
252
253         in.close();
254         out.close();
255     }
256
257 }
```