```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 Yiming Cheng
 *
 */
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";
        //find the content of the title
        String name = " ";
        if (getChildElement(channel, "title") <= -1
                || channel.child(getChildElement(channel, "title"))
                        .numberOfChildren() == 0) {
            name = "No Title";
        } else {
            name = channel.child(getChildElement(channel, "title")).child(0)
                    .label();
        }
        //find the description that is provided
        String description = " ";
        if (getChildElement(channel, "description") <= -1
                || channel.child(getChildElement(channel, "description"))
```

```
68                          .numberOfChildren() == 0) {
69              description = "No Description";
70          } else {
71              description = channel.child(getChildElement(channel, "description"))
72                      .child(0).label();
73          }
74          //print the information that the page needs
75          out.println("<html>");
76          out.println("<head>");
77          out.println("<title>" + name + "</title>");
78          out.println("</head>");
79          out.print("<body>");
80          out.println("<h1>" + channel.child(getChildElement(channel, "link"))
81                  .child(0).label() + "</h1>");
82          out.println("<p>");
83          out.println(description);
84          out.println("</p>");
85          out.println(" <table border=\"1\">");
86          out.println("<tr>");
87          out.println("<th>Date</th>");
88          out.println("<th>Source</th>");
89          out.println("<th>News</th>");
90          out.println("</tr>");
91
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         out.println("</table>");
112         out.println("</body>");
113         out.println("</html>");
114     }
115
116     /**
117      * Finds the first occurrence of the given tag among the children of the
118      * given {@code XMLTree} and return its index; returns -1 if not found.
119      *
120      * @param xml
121      *            the {@code XMLTree} to search
122      * @param tag
123      *            the tag to look for
124      * @return the index of the first child of type tag of the {@code XMLTree}
125      *         or -1 if not found
126      * @requires [the label of the root of xml is a tag]
127      * @ensures <pre>
128      * getChildElement =
129      *  [the index of the first child of type tag of the {@code XMLTree} or
```

```
130      *   -1 if not found]
131      * </pre>
132      */
133     private static int getChildElement(XMLTree xml, String tag) {
134         assert xml != null : "Violation of: xml is not null";
135         assert tag != null : "Violation of: tag is not null";
136         assert xml.isTag() : "Violation of: the label root of xml is a tag";
137
138         //compare the information
139
140         int index = -1;
141         int number = xml.numberOfChildren();
142         int i = 0;
143         while (index == -1 && number > i) {
144             if (xml.child(i).isTag()) {
145                 if (xml.child(i).label().equals(tag)) {
146                     index = i;
147                 }
148             }
149             i++;
150         }
151         return index;
152     }
153
154     /**
155      * Processes one news item and outputs one table row. The row contains three
156      * elements: the publication date, the source, and the title (or
157      * description) of the item.
158      *
159      * @param item
160      *            the news item
161      * @param out
162      *            the output stream
163      * @updates out.content
164      * @requires [the label of the root of item is an <item> tag] and
165      *           out.is_open
166      * @ensures <pre>
167      * out.content = #out.content *
168      *   [an HTML table row with publication date, source, and title of news item]
169      * </pre>
170      */
171     private static void processItem(XMLTree item, SimpleWriter out) {
172         assert item != null : "Violation of: item is not null";
173         assert out != null : "Violation of: out is not null";
174         assert item.isTag() && item.label().equals("item") : ""
175                 + "Violation of: the label root of item is an <item> tag";
176         assert out.isOpen() : "Violation of: out.is_open";
177
178         //find the information about the chart
179         out.println("<tr>");
180         String publicationDate = " ";
181         //compare the date
182         if (getChildElement(item, "pubDate") != -1) {
183             publicationDate = item.child(getChildElement(item, "pubDate"))
184                     .child(0).label();
185         } else {
186             publicationDate = "No data available";
187         }
188         out.println("<td>" + publicationDate + "</td >");
189         String sourceOfLink = " ";
190         String URL = "";
191         //find the URL that the chart would be contained
```

```
192         if (getChildElement(item, "source") != -1) {
193             sourceOfLink = item.child(getChildElement(item, "source")).child(0)
194                     .label();
195             URL = item.child(getChildElement(item, "source"))
196                     .attributeValue("url");
197             out.println("<th><a href = \"" + sourceOfLink + "\">" + URL + "</a>"
198                     + "</th>");
199         } else {
200             sourceOfLink = "No source available";
201             out.println("<td>" + sourceOfLink + "</td>");
202 
203         }
204         String newsTitle = " ";
205         // provide the title of the RSS
206         if (getChildElement(item, "title") > -1
207                 && item.child(getChildElement(item, "title"))
208                         .numberOfChildren() != 0) {
209             newsTitle = item.child(getChildElement(item, "title")).child(0)
210                     .label();
211         } else if (getChildElement(item, "descrption") > -1
212                 && item.child(getChildElement(item, "description"))
213                         .numberOfChildren() != 0) {
214             newsTitle = item.child(getChildElement(item, "description"))
215                     .child(0).label();
216         } else {
217             newsTitle = "No title available";
218         }
219         String newsLink = "";
220         if (getChildElement(item, "link") > -1) {
221             newsLink = item.child(getChildElement(item, "link")).child(0)
222                     .label();
223             out.println("<td><a href=\"" + newsLink + "\">" + newsTitle + "</a>"
224                     + "</td>");
225 
226         } else {
227             out.println("<td>" + newsTitle + "</td>");
228         }
229         out.println("</tr>");
230 
231     }
232 
233     /**
234      * Main method.
235      *
236      * @param args
237      *            the command line arguments; unused here
238      */
239     public static void main(String[] args) {
240         SimpleReader in = new SimpleReader1L();
241         SimpleWriter out = new SimpleWriter1L();
242         out.print("Please type a URL about RSS 2.0");
243         String input = in.nextLine();
244         XMLTree xml = new XMLTree1(input);
245 
246         /*
247          * TODO: fill in body
248          */
249         //prompt the user to type a valid xml address
250         while (!xml.label().equals("rss")
251                 && xml.attributeValue("xml") == "2.0") {
252             out.print("Please type a URL about RSS 2.0");
253             input = in.nextLine();
```

```
254             xml = new XMLTree1(input);
255         }
256         out.print("Please enter the file that you want to output");
257         //prompt the user to type the file that they want to output
258         String fileName = in.nextLine();
259         SimpleWriter file = new SimpleWriter1L(fileName);
260         outputHeader(xml.child(0), file);
261         int a = 0;
262         //the code would run the different items from the address
263         while (a < xml.child(0).numberOfChildren()) {
264             if (xml.child(0).child(a).label().equals("item")) {
265                 processItem(xml.child(0).child(a), file);
266             }
267             a++;
268         }
269         outputFooter(file);
270         in.close();
271         out.close();
272     }
273
274 }
```