```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 Vaishnavi Kasabwala
 *
 */
public final class RSSAggregator2 {

    /**
     * Private constructor so this utility class cannot be instantiated.
     */
    private RSSAggregator2() {
    }

    /**
     * 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 feeds, SimpleWriter out) {
        assert feeds != null : "Violation of: channel is not null";
        assert out != null : "Violation of: out is not null";
        assert feeds.isTag() && feeds.label().equals("feeds") : ""
                + "Violation of: the label root of channel is a <channel> tag";
        assert out.isOpen() : "Violation of: out.is_open";

        out.println("<html>");
        out.println("<head>");
        out.println("<title>" + feeds.attributeValue("title") + "</title>");
        out.println("<body>");
        out.println("<h2>" + feeds.attributeValue("title") + "</h2>");
        out.println("<ul>");
    }

```

```java
    /**
     * Outputs the "closing" tags in the generated HTML file. These are the
     * expected elements generated by this method:
     *
     * </table>
     * </body> </html>
     *
     * @param out
     *            the output stream
     * @updates out.contents
     * @requires out.is_open
     * @ensures out.content = #out.content * [the HTML "closing" tags]
     */
    private static void outputFooter(SimpleWriter out) {
        assert out != null : "Violation of: out is not null";
        assert out.isOpen() : "Violation of: out.is_open";

        out.println("</ul>");
        out.println("</body>");
        out.println("</html>");
    }

    /**
     * Finds the first occurrence of the given tag among the children of the
     * given {@code XMLTree} and return its index; returns -1 if not found.
     *
     * @param xml
     *            the {@code XMLTree} to search
     * @param tag
     *            the tag to look for
     * @return the index of the first child of type tag of the {@code XMLTree}
     *         or -1 if not found
     * @requires [the label of the root of xml is a tag]
     * @ensures <pre>
     * getChildElement =
     *  [the index of the first child of type tag of the {@code XMLTree} or
     *   -1 if not found]
     * </pre>
     */
    private static int getChildElement(XMLTree xml, String tag) {
        assert xml != null : "Violation of: xml is not null";
        assert tag != null : "Violation of: tag is not null";
        assert xml.isTag() : "Violation of: the label root of xml is a tag";

        int n = xml.numberOfChildren();
        int index = -1;
        int i = 0;

        while (i < n && index == -1) {
            if (xml.child(i).label().equals(tag)) {
                index = i;
            }
            i++;
        }
        return index;
    }

```

```
120     /**
121      * Processes one news item and outputs one table row. The row contains three
122      * elements: the publication date, the source, and the title (or
123      * description) of the item.
124      *
125      * @param item
126      *            the news item
127      * @param out
128      *            the output stream
129      * @updates out.content
130      * @requires [the label of the root of item is an <item> tag] and
131      *           out.is_open
132      * @ensures <pre>
133      * out.content = #out.content *
134      *   [an HTML table row with publication date, source, and title of news item]
135      * </pre>
136      */
137     private static void processItem(XMLTree item, SimpleWriter out) {
138         assert item != null : "Violation of: item is not null";
139         assert out != null : "Violation of: out is not null";
140         assert item.isTag() && item.label().equals("item") : ""
141                 + "Violation of: the label root of item is an <item> tag";
142         assert out.isOpen() : "Violation of: out.is_open";
143
144         int indexPubDate = getChildElement(item, "pubDate");
145         int indexSource = getChildElement(item, "source");
146         int indexTitle = getChildElement(item, "title");
147         int indexDescription = getChildElement(item, "description");
148         int indexLink = getChildElement(item, "link");
149
150         out.println("<tr>");
151
152         //publication date (if pubDate exists, it is required to have a child)
153         if (indexPubDate >= 0) {
154             out.println("<td>" + item.child(indexPubDate).child(0).label()
155                     + "</td>");
156         } else {
157             out.println("<td>No date available</td>");
158         }
159
160         //source
161         if (indexSource >= 0
162                 && item.child(indexSource).numberOfChildren() > 0) {
163             XMLTree source = item.child(indexSource);
164             out.print("<td>");
165             out.print("<a href=\"" + source.attributeValue("url") + "\">");
166             out.println(source.child(0).label() + "</a></td>");
167         } else {
168             out.println("<td>No source available</td>");
169         }
170
171         // link (if link exists, it is required to have a child)
172
173         if (indexTitle >= 0 && item.child(indexTitle).numberOfChildren() > 0) {
174             XMLTree title = item.child(indexTitle);
175             out.print("<td>");
176             if (indexLink >= 0) {
```

```
177                 XMLTree link = item.child(indexLink);
178                 out.print("<a href=\"" + link.child(0).label() + "\">");
179             }
180             out.println(title.child(0).label() + "</a></td>");
181         } else if (indexDescription >= 0
182                 && item.child(indexDescription).numberOfChildren() > 0) {
183             XMLTree description = item.child(indexDescription);
184             out.print("<td>");
185             if (indexLink >= 0) {
186                 XMLTree link = item.child(indexLink);
187                 out.print("<a href=\"" + link.child(0).label() + "\">");
188             }
189             out.println(description.child(0).label() + "</a></td>");
190         } else {
191             out.print("<td>No title available</td>");
192         }
193
194         out.println("</tr>");
195     }
196
197     /**
198      * Processes one XML RSS (version 2.0) feed from a given URL converting it
199      * into the corresponding HTML output file.
200      *
201      * @param url
202      *            the URL of the RSS feed
203      * @param file
204      *            the name of the HTML output file
205      * @param out
206      *            the output stream to report progress or errors
207      * @updates out.content
208      * @requires out.is_open
209      * @ensures <pre>
210      * [reads RSS feed from url, saves HTML document with table of news items
211      *   to file, appends to out.content any needed messages]
212      * </pre>
213      */
214     private static void processFeed(String url, String file, SimpleWriter out) {
215         out.println("<li><a href=\"" + url + "\">" + file + "</a></li>");
216     }
217
218     /**
219      * Main method.
220      *
221      * @param args
222      *            the command line arguments; unused here
223      */
224     public static void main(String[] args) {
225         SimpleReader in = new SimpleReader1L();
226         SimpleWriter out = new SimpleWriter1L();
227
228         /*
229          * Input the source URL.
230          * http://web.cse.ohio-state.edu/software/2221/web-sw1/assignments/
231          * projects/rss-aggregator/feeds.xml
232          */
233         out.print(
```

```
234                 "Enter an XML file containing a list of URLs for RSS v2.0 feeds: ");
235         String url = in.nextLine();
236         XMLTree xml = new XMLTree1(url);
237 
238         /*
239          * Asks user for the name of an output file including the .html
240          * extension.
241          */
242         out.print("Please enter the name of an output file: ");
243         String outFile = in.nextLine();
244         SimpleWriter file = new SimpleWriter1L(outFile);
245 
246         String attribute = "";
247         if (xml.label().equals("rss")) {
248             attribute = xml.attributeValue("version");
249         }
250 
251         XMLTree feeds = xml.child(0);
252         if (attribute.equals("2.0")) {
253             outputHeader(feeds, file);
254             // item tag and its children
255             for (int i = 0; i < feeds.numberOfChildren(); i++) {
256                 if (feeds.child(i).label().equals("feed")) {
257                     XMLTree feed = feeds.child(i);
258                     processItem(feed, file);
259                 }
260             }
261 
262             outputFooter(file);
263         }
264 
265         xml.display();
266 
267         in.close();
268         out.close();
269     }
270 
271 }
```