```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 Bashir Ali
 *
 */
public final class RSSAggregator {

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

    /**
     * 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";

        out.println("<html>");
        out.println("<head>");
        out.print("<title>");

        /**
         * creating tree with title tag as root node. If text child of title
         * exists, then print title.
         */
```

```
63          XMLTree title = channel.child(getChildElement(channel, "title"));
64          String titleLabel = "";
65          if (title.numberOfChildren() > 0) {
66              XMLTree titleTextChild = title.child(0);
67              titleLabel = titleTextChild.label();
68              out.print(titleLabel);
69          }
70          out.println("</title>");
71
72          out.println("</head>");
73          out.println("<body>");
74          out.print("<h1>");
75
76          // creating tree with link tag as root node and printing link
77          XMLTree link = channel.child(getChildElement(channel, "link"));
78          XMLTree linkTextChild = link.child(0);
79          String linkLabel = linkTextChild.label();
80
81          /**
82           * if there is a text child of title, print title with the link
83           * attached. if there is no text child of title, print the link as the
84           * header with no title.
85           */
86          if (!titleLabel.equals("")) {
87              out.print("<a href = " + "\"" + linkLabel + "\"" + "> " + titleLabel
88                      + "</a>");
89          } else {
90              out.print("<a href = " + "\"" + linkLabel + "\"" + "> " + linkLabel
91                      + "</a>");
92          }
93          out.println("</h1>");
94          out.println("<p>");
95
96          /**
97           * creating tree with description tag as root node. If text child of
98           * title exists, then print title
99           */
100         XMLTree description = channel
101                 .child(getChildElement(channel, "description"));
102         String descriptionLabel = "";
103         if (description.numberOfChildren() > 0) {
104             XMLTree descriptionTextChild = description.child(0);
105             descriptionLabel = descriptionTextChild.label();
106             out.println(descriptionLabel);
107         } else {
108             out.println("No description available.");
109         }
110
111         out.println("</p>");
112         out.println("<table border=\"1\">");
113         out.println("<tr>");
114         out.println("<th>Date</th>");
115         out.println("<th>Source</th>");
116         out.println("<th>News</th>");
117         out.println("</tr>");
118     }
119
```

```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("</table>");
        out.print("</body>");
        out.print("<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 index = -1;

        for (int i = 0; i < xml.numberOfChildren(); i++) {
            if (xml.child(i).label().equals(tag)) {
                index = i;
            }
        }
        return index;
    }

    /**
     * Processes one news item and outputs one table row. The row contains three
     * elements: the publication date, the source, and the title (or
```

```
177      * description) of the item.
178      *
179      * @param item
180      *            the news item
181      * @param out
182      *            the output stream
183      * @updates out.content
184      * @requires [the label of the root of item is an <item> tag] and
185      *           out.is_open
186      * @ensures <pre>
187      * out.content = #out.content *
188      *   [an HTML table row with publication date, source, and title of news item]
189      * </pre>
190      */
191     private static void processItem(XMLTree item, SimpleWriter out) {
192         assert item != null : "Violation of: item is not null";
193         assert out != null : "Violation of: out is not null";
194         assert item.isTag() && item.label().equals("item") : ""
195                 + "Violation of: the label root of item is an <item> tag";
196         assert out.isOpen() : "Violation of: out.is_open";
197
198         out.println("<tr>");
199
200         /**
201          * Check if pubDate exists, if it does, then print the pubDate text
202          * child else, print no pubDate available
203          */
204         int pubDateIndex = getChildElement(item, "pubDate");
205         String pubDate = "<td>No publishing date available</td>";
206         if (pubDateIndex != -1) {
207             pubDate = "<td>" + item.child(pubDateIndex).child(0).label()
208                     + "</td>";
209         }
210         out.println(pubDate);
211
212         /**
213          * Check if source exists, if it does, create url string if source has
214          * children, print hyperlink with source else, just print link else,
215          * print no source avail.
216          */
217         int sourceIndex = getChildElement(item, "source");
218         String source = "<td>No source available</td>";
219         if (sourceIndex != -1) {
220             String url = item.child(sourceIndex).attributeValue("url");
221             if (item.child(sourceIndex).numberOfChildren() > 0) {
222                 String titleLabel = item.child(sourceIndex).child(0).label();
223                 out.println("<td>" + "<a href = " + "\"" + url + "\"" + "> "
224                         + titleLabel + "</a>" + "</td>");
225             } else {
226                 out.println("<td>" + "<a href = " + "\"" + url + "\"" + "> "
227                         + url + "</a>" + "</td>");
228             }
229
230         } else {
231             out.println(source);
232         }
233
```

```
234         /**
235          * Check if title or description exist and create a string if link
236          * exists, print string with hyperlink else, print string
237          */
238
239         int titleIndex = getChildElement(item, "title");
240         int descriptionIndex = getChildElement(item, "description");
241         int linkIndex = getChildElement(item, "link");
242
243         String news = "";
244
245         if (titleIndex != -1 && item.child(titleIndex).numberOfChildren() > 0) {
246             news = item.child(titleIndex).child(0).label();
247         } else if (descriptionIndex != -1
248                 && item.child(descriptionIndex).numberOfChildren() > 0) {
249             news = item.child(descriptionIndex).child(0).label();
250         }
251
252         if (linkIndex != -1) {
253             String link = item.child(linkIndex).child(0).label();
254             out.println("<td>" + "<a href=\"" + link + "\">" + news + "</a>"
255                     + "</td>");
256         } else {
257             out.println("<td>" + news + "</td>");
258         }
259
260         out.println("</tr>");
261
262     }
263
264     /**
265      * Processes one XML RSS (version 2.0) feed from a given URL converting it
266      * into the corresponding HTML output file.
267      *
268      * @param url
269      *            the URL of the RSS feed
270      * @param file
271      *            the name of the HTML output file
272      * @param out
273      *            the output stream to report progress or errors
274      * @updates out.content
275      * @requires out.is_open
276      * @ensures <pre>
277      * [reads RSS feed from url, saves HTML document with table of news items
278      *   to file, appends to out.content any needed messages]
279      * </pre>
280      */
281     private static void processFeed(String url, String file, SimpleWriter out) {
282         XMLTree rss = new XMLTree1(url);
283         SimpleWriter fileOut2 = new SimpleWriter1L(file);
284
285         String version = "";
286         if (rss.hasAttribute("version")) {
287             version = rss.attributeValue("version");
288         }
289
290         if (rss.isTag() && version.equals("2.0")) {
```

```
291             XMLTree channel = rss.child(0);
292             outputHeader(channel, fileOut2);
293             int i = 0;
294             while (i < channel.numberOfChildren()) {
295                 if (channel.child(i).label().equals("item")) {
296                     processItem(channel.child(i), fileOut2);
297                 }
298 
299                 i++;
300             }
301             outputFooter(fileOut2);
302         }
303         fileOut2.close();
304 
305     }
306 
307     /**
308      * Main method.
309      *
310      * @param args
311      *            the command line arguments; unused here
312      */
313     public static void main(String[] args) {
314         SimpleReader in = new SimpleReader1L();
315         SimpleWriter out = new SimpleWriter1L();
316 
317         out.print("Enter the name of an XML file: ");
318         String xmlFile = in.nextLine();
319         XMLTree xml = new XMLTree1(xmlFile);
320 
321         out.print("Enter the name of an HTML file: ");
322         String html = in.nextLine();
323         if (!html.endsWith(".html")) {
324             html = html + ".html";
325         }
326 
327         SimpleWriter fileOut = new SimpleWriter1L(html);
328 
329         fileOut.println("<html>");
330         fileOut.println("<head>");
331         fileOut.print("<title>");
332         fileOut.print(xml.attributeValue("title"));
333         fileOut.println("</title>");
334         fileOut.println("</head>");
335         fileOut.println("<body>");
336         fileOut.print("<h2>");
337         fileOut.print(xml.attributeValue("title"));
338         fileOut.print("</h2>");
339         fileOut.println("<ul>");
340 
341         /**
342          * for loop goes through all children of input xml file and creates and
343          * prints an html page. html page is linked on new home page
344          */
345         for (int i = 0; i < xml.numberOfChildren(); i++) {
346             fileOut.print("<li>");
347             String url = xml.child(i).attributeValue("url");
```

```
348             String file = xml.child(i).attributeValue("file");
349             String name = xml.child(i).attributeValue("name");
350             processFeed(url, file, out);
351             fileOut.print(
352                     "<a href = " + "\"" + file + "\"" + "> " + name + "</a>");
353             fileOut.println("</li>");
354         }
355 
356         fileOut.println("</ul>");
357         fileOut.println("</body>");
358         fileOut.println("</html>");
359 
360         in.close();
361         out.close();
362         fileOut.close();
363     }
364 
365 }
366 
```