```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 Joseph Fong
 *
 */
public final class RSSAggreagtor {

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

    /**
     * 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";

        // Creates string for each value to be printed in HTML
        String title = "No title available";
        String description = "No description available";
        String link = "N/A";

        // Creates int for index of each node in channel
        int titleIndex = getChildElement(channel, "title");
        int descriptionIndex = getChildElement(channel, "description");
        int linkIndex = getChildElement(channel, "link");

```

```
65          // Sets string value to children of channels child nodes
66          if (titleIndex != -1
67                  && channel.child(titleIndex).numberOfChildren() > 0) {
68              title = channel.child(titleIndex).child(0).label();
69          }
70          if (descriptionIndex != -1
71                  && channel.child(descriptionIndex).numberOfChildren() > 0) {
72              description = channel.child(descriptionIndex).child(0).label();
73          }
74          link = channel.child(linkIndex).child(0).label();
75
76          // Prints out header
77          out.println("<html>");
78          out.println("<head>");
79          out.println("<title>" + title + "</title>");
80          out.println("</head>");
81          out.println("<body>");
82          out.println("\t<h1><a href=\"" + link + "\">" + title + "</a></h1>");
83          out.println("\t<p>");
84          out.println(description);
85          out.println("\t</p>");
86          out.println("\t<table border=\"1\">");
87          out.println("\t\t<tr>");
88          out.println("\t\t\t<th>Date</th>");
89          out.println("\t\t\t<th>Source</th>");
90          out.println("\t\t\t<th>News</th>");
91          out.println("\t\t</tr>");
92
93      }
94
95      /**
96       * Outputs the "closing" tags in the generated HTML file. These are the
97       * expected elements generated by this method:
98       *
99       * </table>
100      * </body> </html>
101      *
102      * @param out
103      *            the output stream
104      * @updates out.contents
105      * @requires out.is_open
106      * @ensures out.content = #out.content * [the HTML "closing" tags]
107      */
108     private static void outputFooter(SimpleWriter out) {
109         assert out != null : "Violation of: out is not null";
110         assert out.isOpen() : "Violation of: out.is_open";
111
112         out.println("\t</table>");
113         out.println("</body>");
114         out.println("</html>");
115
116     }
117
118     /**
119      * Finds the first occurrence of the given tag among the children of the
120      * given {@code XMLTree} and return its index; returns -1 if not found.
121      *
122      * @param xml
123      *            the {@code XMLTree} to search
```

```
124      * @param tag
125      *            the tag to look for
126      * @return the index of the first child of type tag of the {@code XMLTree}
127      *         or -1 if not found
128      * @requires [the label of the root of xml is a tag]
129      * @ensures <pre>
130      * getChildElement =
131      *  [the index of the first child of type tag of the {@code XMLTree} or
132      *   -1 if not found]
133      * </pre>
134      */
135     private static int getChildElement(XMLTree xml, String tag) {
136         assert xml != null : "Violation of: xml is not null";
137         assert tag != null : "Violation of: tag is not null";
138         assert xml.isTag() : "Violation of: the label root of xml is a tag";
139
140         // creates 2 ints to parse tree nodes
141         int i = 0, childElement = -1;
142         // boolean created to be used to exit loop
143         boolean exit = true;
144
145         // while loop used to search for child element equal to tag
146         while (i < xml.numberOfChildren() && exit) {
147             if (xml.child(i).label().equals(tag)) {
148                 childElement = i;
149                 exit = false;
150             }
151             i++;
152         }
153         // returns correct index
154         return childElement;
155     }
156
157     /**
158      * Processes one news item and outputs one table row. The row contains three
159      * elements: the publication date, the source, and the title (or
160      * description) of the item.
161      *
162      * @param item
163      *            the news item
164      * @param out
165      *            the output stream
166      * @updates out.content
167      * @requires [the label of the root of item is an <item> tag] and
168      *           out.is_open
169      * @ensures <pre>
170      * out.content = #out.content *
171      *   [an HTML table row with publication date, source, and title of news item]
172      * </pre>
173      */
174     private static void processItem(XMLTree item, SimpleWriter out) {
175         assert item != null : "Violation of: item is not null";
176         assert out != null : "Violation of: out is not null";
177         assert item.isTag() && item.label().equals("item") : ""
178                 + "Violation of: the label root of item is an <item> tag";
179         assert out.isOpen() : "Violation of: out.is_open";
180
181         // creates string for each value of item children
182         String pubDate = "No date available";
```

```
183         String source = "No source available";
184         String sourceLink = "N/A";
185         String title = "No title available";
186         String description = "";
187         String link = "N/A";
188 
189         // creates int for index of publication date
190         int pubDateIndex = getChildElement(item, "pubDate");
191 
192         /*
193          * if statements verify that the item has a child and that it also has a
194          * child then assigns value of sting to the child label
195          */
196         if (pubDateIndex != -1
197                 && item.child(pubDateIndex).numberOfChildren() > 0) {
198             pubDate = item.child(pubDateIndex).child(0).label();
199         }
200 
201         int sourceIndex = getChildElement(item, "source");
202 
203         if (sourceIndex != -1
204                 && item.child(sourceIndex).numberOfChildren() > 0) {
205             source = item.child(sourceIndex).child(0).label();
206             sourceLink = item.child(sourceIndex).attributeValue("url");
207         }
208 
209         int titleIndex = getChildElement(item, "title");
210 
211         if (titleIndex != -1 && item.child(titleIndex).numberOfChildren() > 0) {
212             title = item.child(titleIndex).child(0).label();
213         }
214 
215         int descriptionIndex = getChildElement(item, "description");
216 
217         if (descriptionIndex != -1
218                 && item.child(descriptionIndex).numberOfChildren() > 0) {
219             description = item.child(descriptionIndex).child(0).label();
220         }
221 
222         int linkIndex = getChildElement(item, "link");
223 
224         if (linkIndex != -1 && item.child(linkIndex).numberOfChildren() > 0) {
225             link = item.child(linkIndex).child(0).label();
226         }
227 
228         // prints out table
229         out.println("\t\t<tr>");
230         out.println("\t\t\t<td>" + pubDate + "</td>");
231 
232         // if statements used to verify if there is a link
233         if (!sourceLink.equals("N/A")) {
234             out.println("\t\t\t<td><a href=\"" + sourceLink + "\">" + source
235                     + "</a></td>");
236         } else {
237             out.println("\t\t\t<td>" + source + "</td>");
238         }
239 
240         /*
241          * if statement chain used to print out title with link or description
```

```
242          * if there is no title
243          */
244         if (!link.equals("N/A") && !title.equals("No title available")) {
245             out.println("\t\t\t<td><a href=\"" + link + "\">" + title
246                     + "</a></td>");
247         } else if (link.equals("N/A") && !title.equals("No title available")) {
248             out.println("\t\t\t<td>" + title + "</td>");
249         } else if (title.equals("No title available") && !description.equals("")
250                 && !link.equals("N/A")) {
251             out.println("\t\t\t<td><a href=\\\"\"" + link + "\"\\\">\""
252                     + description + "</td>");
253         } else if (title.equals("No title available") && !description.equals("")
254                 && link.equals("N/A")) {
255             out.println("\t\t\t<td>" + description + "</td>");
256         }
257 
258         out.println("\t\t</tr>");
259     }
260 
261     /**
262      * Processes one XML RSS (version 2.0) feed from a given URL converting it
263      * into the corresponding HTML output file.
264      *
265      * @param url
266      *            the URL of the RSS feed
267      * @param file
268      *            the name of the HTML output file
269      * @param out
270      *            the output stream to report progress or errors
271      * @updates out.content
272      * @requires out.is_open
273      * @ensures <pre>
274      * [reads RSS feed from url, saves HTML document with table of news items
275      *   to file, appends to out.content any needed messages]
276      * </pre>
277      */
278     private static void processFeed(String url, String file, SimpleWriter out) {
279 
280         // opens fileWriter and xml tree
281         SimpleWriter fileWriter = new SimpleWriter1L(file);
282         XMLTree xml = new XMLTree1(url);
283 
284         // verifies url is valid rss 2.0
285         if (!xml.label().equals("rss") || !xml.hasAttribute("version")
286                 || !xml.attributeValue("version").equals("2.0")) {
287             out.println("Error: feed is not valid RSS 2.0");
288         } else {
289             // creates channel
290             XMLTree channel = xml.child(0);
291 
292             // prints out header into HTML file
293             outputHeader(channel, fileWriter);
294 
295             // uses for loop + if statement to print out each item
296             for (int i = 0; i < channel.numberOfChildren(); i++) {
297                 XMLTree item = channel.child(i);
298                 if (channel.child(i).label().equals("item")) {
299                     processItem(item, fileWriter);
300                 }
```

```
301             }
302
303             // prints out footer
304             outputFooter(fileWriter);
305         }
306         // closes fileWriter
307         fileWriter.close();
308     }
309
310     /**
311      * Main method. Asks user for valid RSS feed and outputs a HTML file
312      * displaying list of links for included rss 2.0 pages.
313      *
314      * @param args
315      *            the command line arguments; unused here
316      */
317     public static void main(String[] args) {
318         SimpleReader in = new SimpleReader1L();
319         SimpleWriter out = new SimpleWriter1L();
320
321         // asks user for valid rss feed
322         out.println("Please enter a valid index xml url: ");
323         String url = in.nextLine();
324         XMLTree xml = new XMLTree1(url);
325
326         // verifies xml tree is a valid feed
327         while (!xml.label().equals("feeds") || !xml.hasAttribute("title")) {
328             out.println("The url entered is not a valid feed");
329             out.println("Please enter a valid index xml url: ");
330             url = in.nextLine();
331             xml = new XMLTree1(url);
332         }
333
334         // asks user for title of output file
335         out.println("Title for index page file: ");
336         String indexName = in.nextLine();
337         // creates simple writer to write to file
338         SimpleWriter fileWriter = new SimpleWriter1L(indexName);
339
340         // creates string for title of feed
341         String title = xml.attributeValue("title");
342
343         // for loop runs through all children a <feeds> and creates html pages
344         for (int i = 0; i < xml.numberOfChildren(); i++) {
345             String htmlName = xml.child(i).attributeValue("file");
346             String feedURL = xml.child(i).attributeValue("url");
347             processFeed(feedURL, htmlName, out);
348         }
349
350         // output header to html file
351         fileWriter.println("<html>");
352         fileWriter.println("<head>");
353         fileWriter.println("<title>" + title + "</title>");
354         fileWriter.println("</head>");
355         fileWriter.println("<body>");
356         fileWriter.println("<h1>" + title + "</h1>");
357         fileWriter.println("<ul style=\"list-style-type:square\">");
358         // for loop prints out list for all <feeds> children
359         for (int j = 0; j < xml.numberOfChildren(); j++) {
```

```
360             String feedName = xml.child(j).attributeValue("name");
361             String fileName = xml.child(j).attributeValue("file");
362             fileWriter.println("<li><p><a href=\"" + fileName + "\">" + feedName
363                     + "</a></p></li>");
364         }
365         fileWriter.println("</body>");
366         fileWriter.println("</html>");
367 
368         // closes simplewriter/reader
369         in.close();
370         out.close();
371         fileWriter.close();
372     }
373 }
```