

```
1 import components.simplereader.SimpleReader;
2
3 /**
4  * Program to convert an XML file stored with RSS feeds (version
5  * 2.0) feed from
6  * a given URL into an HTML index file, then process all RSS feeds
7  * into their
8  * own HTML files that are accessible from the index file.
9  *
10 * @author Isaac Frank
11 *
12 */
13 public final class RSSAggregator {
14
15     /**
16      * Private constructor so this utility class cannot be
17      * instantiated.
18      */
19     private RSSAggregator() {}
20
21     /**
22      * Outputs the "opening" tags in the generated HTML file.
23      * These are the
24      * expected elements generated by this method:
25      *
26      * <html> <head> <title>the channel tag title as the page
27      * title</title>
28      * </head> <body>
29      * <h1>the page title inside a link to the <channel> link</h1>
30      * <p>
31      * the channel description
32      * </p>
33      * <table border="1">
34      * <tr>
35      * <th>Date</th>
36      * <th>Source</th>
37      * <th>News</th>
38      * </tr>
39      *
40      * @param channel
41      *         the channel element XMLTree
42      * @param out
43      *         the output stream
44      */
45 }
```

```
45     * @updates out.content
46     * @requires [the root of channel is a <channel> tag] and
    out.is_open
47     * @ensures out.content = #out.content * [the HTML "opening"
    tags]
48     */
49     private static void outputHeader(XMLTree channel, SimpleWriter
    out) {
50         assert channel != null : "Violation of: channel is not
    null";
51         assert out != null : "Violation of: out is not null";
52         assert channel.isTag() &&
    channel.label().equals("channel") : ""
53         + "Violation of: the label root of channel is a
    <channel> tag";
54         assert out.isOpen() : "Violation of: out.is_open";
55
56         // Getting indexes of title, link, and description
57         int titleChildIndex = getChildElement(channel, "title");
58         int linkChildIndex = getChildElement(channel, "link");
59         int descriptionChildIndex = getChildElement(channel,
    "description");
60
61         // Html opening tags and printing title
62         out.print("<html> <head> <title>");
63         String title = "Empty Title";
64         if (channel.child(titleChildIndex).numberOfChildren() > 0)
    {
65             title =
    channel.child(titleChildIndex).child(0).label();
66         }
67         out.print(title);
68
69         // Html closing tags
70         out.println("</title> </head> <body>");
71
72         // Html header and page title w/ link
73         out.print("<h1>");
74         out.print(
75             "<a href = \"" +
    channel.child(linkChildIndex).child(0).label()
76             + "\">" + title + "</a>");
77         out.println("</h1>");
78
```

```
79         // Html paragraph w/ channel description
80         out.println("<p>");
81         String description = "No description";
82         if
83             (channel.child(descriptionChildIndex).numberOfChildren() > 0) {
84             description =
85             channel.child(descriptionChildIndex).child(0).label();
86             out.println(description);
87             out.println("</p>");
88
89             // Table Headers
90             out.println("<table border = \"\" + 1 + \"\">");
91             out.println("<tr>");
92             out.println("<th>" + "Date" + "</th>");
93             out.println("<th>" + "Source" + "</th>");
94             out.println("<th>" + "News" + "</th>");
95             out.println("</tr>");
96         }
97     /**
98      * Outputs the "closing" tags in the generated HTML file.
99      * These are the
100      * expected elements generated by this method:
101      * </table>
102      * </body> </html>
103      *
104      * @param out
105      *         the output stream
106      * @updates out.contents
107      * @requires out.is_open
108      * @ensures out.content = #out.content * [the HTML "closing"
109      * tags]
110      */
111     private static void outputFooter(SimpleWriter out) {
112         assert out != null : "Violation of: out is not null";
113         assert out.isOpen() : "Violation of: out.is_open";
114
115         out.println("</table>");
116         out.println("</body> </html>");
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         int index = -1;
141
142         // Iterates through children until a tag is found or all
    children are searched
143         int i = 0;
144         while (i < xml.numberOfChildren() && index < 0) {
145             if (xml.child(i).isTag() &&
    xml.child(i).label().equals(tag)) {
146                 index = i;
147             }
148             i++;
149         }
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         // Finding indexes
179         int titleChildIndex = getChildElement(item, "title");
180         int linkChildIndex = getChildElement(item, "link");
181         int descriptionChildIndex = getChildElement(item,
"description");
182         int pubDateChildIndex = getChildElement(item, "pubDate");
183         int sourceChildIndex = getChildElement(item, "source");
184
185         out.println("<tr>");
186
187         // Printing pubDate table cell
188         String pubDate = "No date available";
189         if (pubDateChildIndex != -1) {
190             pubDate =
item.child(pubDateChildIndex).child(0).label();
191         }
```

```
192         out.println("<td>" + pubDate + "</td>");
193
194         // Printing source table cell
195         String source = "No source available";
196         if (sourceChildIndex != -1) {
197             XMLTree src = item.child(sourceChildIndex);
198             if (src.numberOfChildren() > 0) {
199                 String srcAttributeVal =
200 src.attributeValue("url");
201                 source = "<a href = \"" + srcAttributeVal;
202                 source += "\">" + src.child(0).label() + "</a>";
203             }
204             out.println("<td>" + source + "</td>");
205
206             // Printing title table cell, checking if description and
207 link are needed
208             String titleOrDsc = "No title available";
209             String link = "";
210             if (titleChildIndex != -1
211                 && item.child(titleChildIndex).numberOfChildren()
212 > 0) {
213                 titleOrDsc =
214 item.child(titleChildIndex).child(0).label();
215             } else if (descriptionChildIndex != -1
216                 &&
217 item.child(descriptionChildIndex).numberOfChildren() > 0) {
218                 titleOrDsc =
219 item.child(descriptionChildIndex).child(0).label();
220             }
221             if (linkChildIndex != -1) {
222                 link = item.child(linkChildIndex).child(0).label();
223             }
224             out.print("<td><a href = \"" + link + "\">" + titleOrDsc +
225 "</a></td>");
226
227             // Ending the row
228             out.println("</tr>");
229         }
230
231 /**
232  * Processes one XML RSS (version 2.0) feed from a given URL
233  * converting it
```

```
228     * into the corresponding HTML output file.
229     *
230     * @param url
231     *         the URL of the RSS feed
232     * @param file
233     *         the name of the HTML output file
234     * @param out
235     *         the output stream to report progress or errors
236     * @updates out.content
237     * @requires out.is_open
238     * @ensures <pre>
239     * [reads RSS feed from url, saves HTML document with table of
news items
240     * to file, appends to out.content any needed messages]
241     * </pre>
242     */
243     private static void processFeed(String url, String file,
SimpleWriter out) {
244         SimpleWriter outFile = new SimpleWriter1L(file);
245
246         // Checking if xml is RSS 2.0
247         XMLTree xml = new XMLTree1(url);
248         if (xml.isTag()) {
249             if (xml.hasAttribute("version")) {
250                 if (xml.attributeValue("version").equals("2.0")) {
251                     XMLTree channel = xml.child(0);
252                     outputHeader(channel, outFile);
253
254                     // Iterating through all "item" children of
channel
255                     int i = 0;
256                     while (i < channel.numberOfChildren()) {
257                         if
(channel.child(i).label().equals("item")) {
258                             processItem(channel.child(i),
outFile);
259                         }
260                         i++;
261                     }
262
263                     outputFooter(outFile);
264                 }
265             }
266             } else {
```

```
267         out.println("This is not an RSS 2.0 file");
268     }
269     outFile.close();
270 }
271
272 /**
273  * Main method.
274  *
275  * @param args
276  *         the command line arguments; unused here
277  *
278  *         Processes XML document with RSS feeds stored
inside and
279  *         outputs them to an index HTML file
280  */
281 public static void main(String[] args) {
282     SimpleReader in = new SimpleReader1L();
283     SimpleWriter out = new SimpleWriter1L();
284     SimpleWriter outFile = new SimpleWriter1L("index.html");
285
286     // User input
287     out.print("Enter a URL for an XML Document of RSS feeds:
");
288     String url = in.nextLine();
289
290     XMLTree feeds = new XMLTree1(url);
291
292     // Outputting HTML headers and starting an unordered list
293     outFile.print("<html> <head> <title>");
294     outFile.print(feeds.attributeValue("title"));
295     outFile.println("</title> </head>");
296     outFile.println("<h1>" + feeds.attributeValue("title") +
"</h1>");
297     outFile.println("<ul>");
298
299     // Iterating through all RSS feeds stored in XMLTree feeds
300     int i = 0;
301     int numChildren = feeds.numberOfChildren();
302     while (i < numChildren) {
303         // Creating a new list element with a link to each
processed RSS feed
304         outFile.print("<li>");
305         XMLTree feed = feeds.child(i);
306         outFile.print("<a href = \"\" +
```



```
        feed.attributeValue("file"));
307         outFile.print("\">" + feed.attributeValue("name") +
        "</a>");
308         processFeed(feed.attributeValue("url"),
        feed.attributeValue("file"),
309                 out);
310         outFile.println("</li>");
311         i++;
312     }
313
314     // Outputting HTML footers and closing unordered list
315     outFile.println("</ul>");
316     outFile.println("</body>");
317     outFile.println("</html>");
318
319     // Closing output streams
320     in.close();
321     out.close();
322     outFile.close();
323
324 }
325 }
326
```