

```
1 import components.simplereader.SimpleReader;
2 import components.simplereader.SimpleReader1L;
3 import components.simplewriter.SimpleWriter;
4 import components.simplewriter.SimpleWriter1L;
5 import components.xmltree.XMLTree;
6 import components.xmltree.XMLTree1;
7
8 /**
9  * Program to convert an XML RSS (version 2.0) feed from a
10  * given URL into the
11  * corresponding HTML output file.
12  *
13  * @author Ansh Pachauri
14  */
15 public final class RSSAggregator {
16
17     /**
18      * Private constructor so this utility class cannot be
19      * instantiated.
20      */
21     private RSSAggregator() {
22
23     }
24
25     /**
26      * Outputs the "opening" tags in the generated HTML file.
27      * These are the
28      * expected elements generated by this method:
29      *
30      * <html> <head> <title>the channel tag title as the page
31      * title</title>
32      * </head> <body>
33      * <ul>
34      * The unordered list
35      *
36      * @param feeds
37      * the channel element XMLTree
38      * @param out
```

```
36     *           the output stream
37     * @updates out.content
38     * @requires [the root of channel is a <channel> tag] and
    out.is_open
39     * @ensures out.content = #out.content * [the HTML
    "opening" tags]
40     */
41     private static void indexOutputHeader(XMLTree feeds,
    SimpleWriter out) {
42
43         String title = feeds.attributeValue("title");
44
45         out.println("<html>");
46         out.println("<head>");
47         out.println("<title>" + title + "</title>");
48         out.println("<h1>" + title + "</h1>");
49         out.println("</head>");
50         out.println("<body>");
51         out.println("<ul>");
52     }
53
54     /**
55     * Outputs the "closing" tags in the generated HTML file.
    These are the
56     * expected elements generated by this method:
57     *
58     * </ul>
59     * </body> </html>
60     *
61     * @param out
        the output stream
62     * @updates out.contents
63     * @requires out.is_open
64     * @ensures out.content = #out.content * [the HTML
    "closing" tags]
65     */
66
67     private static void indexOutputFooter(SimpleWriter out) {
68
69         out.println("</ul>");
```

```
70         out.println("</body>");
71         out.println("</html>");
72     }
73
74     /**
75      * Processes one XML RSS (version 2.0) feed from a given
76      * URL converting it
77      * into the corresponding HTML output file.
78      * @param url
79      *     the URL of the RSS feed
80      * @param file
81      *     the name of the HTML output file
82      * @param out
83      *     the output stream to report progress or
84      *     errors
85      * @updates out.content
86      * @requires out.is_open
87      * @ensures <pre>
88      * [reads RSS feed from url, saves HTML document with table
89      * of news items
90      * to file, appends to out.content any needed messages]
91      * </pre>
92      */
93     private static void processFeed(String url, String file,
94                                     SimpleWriter out) {
95         XMLTree xml = new XMLTree1(url);
96         SimpleWriter fileOutName = new SimpleWriter1L(file);
97
98         outputHeader(xml.child(0), fileOutName);
99         int i = 0;
100         while (xml.child(0).numberOfChildren() > i) {
101             if (xml.child(0).child(i).label().equals("item")) {
102                 processItem(xml.child(0).child(i),
103                             fileOutName);
104             }
105             i++;
106         }
107         outputFooter(fileOutName);
108     }
```

```
104     }
105
106     /**
107      * Outputs the "opening" tags in the generated HTML file.
108      * These are the
109      * expected elements generated by this method:
110      * <html> <head> <title>the channel tag title as the page
111      * title</title>
112      * </head> <body>
113      * <h1>the page title inside a link to the <channel> link</
114      * h1>
115      * <p>
116      * the channel description
117      * </p>
118      * <table border="1">
119      * <tr>
120      * <th>Date</th>
121      * <th>Source</th>
122      * <th>News</th>
123      * </tr>
124      *
125      * @param channel
126      *         the channel element XMLTree
127      * @param out
128      *         the output stream
129      * @updates out.content
130      * @requires [the root of channel is a <channel> tag] and
131      * out.is_open
132      * @ensures out.content = #out.content * [the HTML
133      * "opening" tags]
134      */
135     private static void outputHeader(XMLTree channel,
136                                     SimpleWriter out) {
137         /*
138          * If title has child then the text will be assigned to
139          * String title,
140          * otherwise, the String title will output "Empty
141          * Title".
142          */
143     }
```

```
135         */
136
137         String title;
138         if (getChildElement(channel, "title") >= 0) {
139             if (channel.child(getChildElement(channel,
140 "title")))
141                 .numberOfChildren() >= 1) {
142                 title = channel.child(getChildElement(channel,
143 "title"))
144                     .child(0).label();
145             } else {
146                 title = "Empty Title";
147             }
148         } else if (getChildElement(channel, "description") >=
149 0) {
150             if (channel.child(getChildElement(channel,
151 "description")))
152                 .numberOfChildren() >= 1) {
153                 title = channel.child(getChildElement(channel,
154 "description"))
155                     .label();
156             } else {
157                 title = "Empty Title";
158             }
159         } else {
160             title = "Empty Title";
161         }
162     }
163
164     /*
165      * If description has child then the text will be
166      assigned to String
167      * desc, otherwise, the String desc will output "No
168      description".
169      */
170     String desc = "";
171     if (channel.child(getChildElement(channel,
172 "description")))
173         .numberOfChildren() >= 0) {
```

```
166         if (channel.child(getChildElement(channel,
167             "description"))
168                 .numberOfChildren() >= 1) {
169             desc = channel.child(getChildElement(channel,
170                 "description"))
171                 .child(0).label();
172         } else {
173             desc = "No Description.";
174         }
175         // header
176         out.println("<html>");
177         out.println("<head>");
178         out.println("<title>" + title + "</title>");
179         out.println("</head>");
180         out.println("<body>");
181         out.println(
182             "<h1><a href=\"\"
183             +
184             channel.child(getChildElement(channel, "link"))
185                 .child(0).label()
186                 + "\">" + title + "</a></h1>");
187         out.println("<p>" + desc + "</p>");
188         out.println("<table border=\"1\">");
189         out.println("<tr>");
190         out.println("<th>Date</th>");
191         out.println("<th>Source</th>");
192         out.println("<th>News</th>");
193         out.println("</tr>");
194     }
195
196     /**
197     * Outputs the "closing" tags in the generated HTML file.
198     * These are the
199     * expected elements generated by this method:
200     * </table>
```

```
201     * </body> </html>
202     *
203     * @param out
204     *         the output stream
205     * @updates out.contents
206     * @requires out.is_open
207     * @ensures out.content = #out.content * [the HTML
    "closing" tags]
208     */
209     private static void outputFooter(SimpleWriter out) {
210
211         out.println("</table>");
212         out.println("</body>");
213         out.println("</html>");
214     }
215
216     /**
217     * Finds the first occurrence of the given tag among the
    children of the
218     * given {@code XMLTree} and return its index; returns -1
    if not found.
219     *
220     * @param xml
221     *         the {@code XMLTree} to search
222     * @param tag
223     *         the tag to look for
224     * @return the index of the first child of type tag of the
    {@code XMLTree}
225     *         or -1 if not found
226     * @requires [the label of the root of xml is a tag]
227     * @ensures <pre>
228     *         getChildElement =
229     *         [the index of the first child of type tag of the {@code
    XMLTree} or
230     *         -1 if not found]
231     * </pre>
232     */
233     private static int getChildElement(XMLTree xml, String tag)
    {
```

```
234
235     int index = -1;
236     int i = 0;
237     while (i < xml.numberOfChildren() && index == -1) {
238
239         if (xml.child(i).label().equals(tag)) {
240             index = i;
241         }
242         i++;
243     }
244     return index;
245 }
246
247 /**
248  * Processes one news item and outputs one table row. The
row contains three
249  * elements: the publication date, the source, and the
title (or
250  * description) of the item.
251  *
252  * @param item
253  *     the news item
254  * @param out
255  *     the output stream
256  * @updates out.content
257  * @requires [the label of the root of item is an <item>
tag] and
258  *     out.is_open
259  * @ensures <pre>
260  * out.content = #out.content *
261  *     [an HTML table row with publication date, source, and
title of news item]
262  * </pre>
263  */
264 private static void processItem(XMLTree item, SimpleWriter
out) {
265     //table start
266     out.println("<tr>");
267     //assigns pubDate with the date then prints the row
```



```
    item
268        String pubDate;
269        if (getChildElement(item, "pubDate") >= 0) {
270            if (item.child(getChildElement(item, "pubDate"))
271                .numberOfChildren() > 0) {
272                pubDate = item.child(getChildElement(item,
273                    "pubDate")).child(0)
274                    .label();
275            } else {
276                pubDate = "No date available";
277            }
278        } else {
279            pubDate = "No date available";
280        }
281        out.println("<th>" + pubDate + "</th>");
282        //assigns source with the source link then prints the
    row item
283        String source;
284        String sourceLink = "";
285        int i = 0;
286        if (getChildElement(item, "source") >= 0) {
287            if (item.child(getChildElement(item, "source"))
288                .numberOfChildren() > 0) {
289                source = item.child(getChildElement(item,
290                    "source")).child(0)
291                    .label();
292                if (item.child(getChildElement(item, "source"))
293                    .hasAttribute("url")) {
294                    sourceLink =
295                        item.child(getChildElement(item, "source"))
296                            .attributeValue("url");
297                    i = 1;
298                } else {
299                    i = 0;
300                }
301            } else {
302                source = "No source available";
```

```
302         }
303
304     } else {
305         source = "No source available";
306     }
307
308     if (i == 1) {
309         out.println(
310             "<th><a href=\"\" + sourceLink + "\">" +
source + "</th>");
311     } else {
312         out.println("<th>" + source + "</th>");
313     }
314     //assigns title with the title of the article then
prints the row item
315     String title = "No title available";
316     if (getChildElement(item, "title") >= 0) {
317         if (item.child(getChildElement(item, "title"))
318             .numberOfChildren() > 0) {
319             title = item.child(getChildElement(item,
"title")).child(0)
320                 .label();
321         }
322
323     } else if (getChildElement(item, "description") >= 0) {
324         if (item.child(getChildElement(item,
"description"))
325             .numberOfChildren() > 0) {
326             title = item.child(getChildElement(item,
"description"))
327                 .child(0).label();
328         }
329     }
330
331     String link = "";
332     if (getChildElement(item, "link") >= 0) {
333         if (item.child(getChildElement(item, "link"))
334             .numberOfChildren() >= 1) {
335             link = item.child(getChildElement(item,
```

```
        "link")).child(0)
336                .label();
337        }
338    }
339
340    out.println("<th><a href=\"\" + link + \"\">\" + title +
    "</th>");
341
342    out.println("</tr>");
343
344    }
345
346    /**
347     * Main method.
348     *
349     * @param args
350     *     the command line arguments; unused here
351     */
352    public static void main(String[] args) {
353        SimpleReader in = new SimpleReader1L();
354        SimpleWriter out = new SimpleWriter1L();
355
356        out.print(
357            "Enter the name of XML file containing URLs for
    RSS v2.0 feeds: ");
358        String url = in.nextLine();
359        XMLTree xml = new XMLTree1(url);
360
361        out.print("Enter output file name: ");
362        String outFile = in.nextLine();
363
364        SimpleWriter fileOut = new SimpleWriter1L(outFile);
365
366        indexOutputHeader(xml, fileOut);
367
368        int i = 0;
369        while (xml.numberOfChildren() > i) {
370            if (xml.child(i).label().equals("feed")) {
371                processFeed(xml.child(i).attributeValue("url"),
```

```
372             xml.child(i).attributeValue("file"),
        fileOut);
373             fileOut.println("<li><a href=\"\"
374             + xml.child(i).attributeValue("file") +
        "\">\"
375             + xml.child(i).attributeValue("name") +
        "</a></li>");
376
377         }
378
379         i++;
380     }
381
382     indexOutputFooter(fileOut);
383
384     in.close();
385     out.close();
386 }
387
388 }
389
```