```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 Shyam Sai Bethina
 *
 */
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
 46      * @ensures out.content = #out.content * [the HTML "opening"
   tags]
 47      */
 48     private static void outputHeader(XMLTree channel, SimpleWriter
   out) {
 49         assert channel != null : "Violation of: channel is not
   null";
 50         assert out != null : "Violation of: out is not null";
 51         assert channel.isTag() &&
   channel.label().equals("channel") : ""
 52                 + "Violation of: the label root of channel is a
   <channel> tag";
 53         assert out.isOpen() : "Violation of: out.is_open";
 54
 55         int linkIndex = getChildElement(channel, "link");
 56         String link = channel.child(linkIndex).child(0).label();
 57
 58         String title = "Empty Title";
 59
 60         /*
 61          * Checks if title node has children
 62          */
 63         int titleIndex = getChildElement(channel, "title");
 64         if (channel.child(titleIndex).numberOfChildren() > 0) {
 65             title = channel.child(titleIndex).child(0).label();
 66         }
 67
 68         String description = "No description";
 69
 70         /*
 71          * Checks if description node has children
 72          */
 73         int descriptionIndex = getChildElement(channel,
   "description");
 74         if (channel.child(descriptionIndex).numberOfChildren() >
   0) {
 75             description =
   channel.child(descriptionIndex).child(0).label();
 76         }
 77
 78         out.println("<html>");
 79         out.println("<head>");
 80         out.println("<title>" + title + "</title>");
```

```
 81         out.println("</head>");
 82         out.println("<body>");
 83         out.println(" <h1><a href='" + link + "'>" + title + "</
    a></h1>");
 84         out.println(" <p>" + description + "</p>");
 85         out.println(" <table border=1>");
 86         out.println("  <tr>");
 87         out.println("   <th>Date</th>");
 88         out.println("   <th>Source</th>");
 89         out.println("   <th>News</th>");
 90         out.print("  </tr>");
 91 
 92     }
 93 
 94     /**
 95      * Outputs the "closing" tags in the generated HTML file.
    These are the
 96      * expected elements generated by this method:
 97      *
 98      * </table>
 99      * </body> </html>
100      *
101      * @param out
102      *            the output stream
103      * @updates out.contents
104      * @requires out.is_open
105      * @ensures out.content = #out.content * [the HTML "closing"
    tags]
106      */
107     private static void outputFooter(SimpleWriter out) {
108         assert out != null : "Violation of: out is not null";
109         assert out.isOpen() : "Violation of: out.is_open";
110 
111         out.println(" </table>");
112         out.println("</body>");
113         out.print("</html>");
114     }
115 
116     /**
117      * Finds the first occurrence of the given tag among the
    children of the
118      * given {@code XMLTree} and return its index; returns -1 if
    not found.
119      *
```

```
120      * @param xml
121      *            the {@code XMLTree} to search
122      * @param tag
123      *            the tag to look for
124      * @return the index of the first child of type tag of the
    {@code XMLTree}
125      *         or -1 if not found
126      * @requires [the label of the root of xml is a tag]
127      * @ensures <pre>
128      * getChildElement =
129      *  [the index of the first child of type tag of the {@code
    XMLTree} or
130      *   -1 if not found]
131      * </pre>
132      */
133     private static int getChildElement(XMLTree xml, String tag) {
134         assert xml != null : "Violation of: xml is not null";
135         assert tag != null : "Violation of: tag is not null";
136         assert xml.isTag() : "Violation of: the label root of xml
    is a tag";
137
138         int index = -1;
139         boolean found = false;
140
141         /*
142          * Goes through all the children of xml and returns the
    index of when it
143          * finds the first instance with the designated tag, which
    is when found
144          * equals to true.
145          */
146         for (int i = 0; i < xml.numberOfChildren() && !found; i++)
    {
147             if (tag.equals(xml.child(i).label())) {
148                 index = i;
149                 found = true;
150             }
151         }
152
153         return index;
154     }
155
156     /**
157      * Processes one news item and outputs one table row. The row
```

```
    contains three
158      * elements: the publication date, the source, and the title
    (or
159      * description) of the item.
160      *
161      * @param item
162      *            the news item
163      * @param out
164      *            the output stream
165      * @updates out.content
166      * @requires [the label of the root of item is an <item> tag]
    and
167      *           out.is_open
168      * @ensures <pre>
169      * out.content = #out.content *
170      *   [an HTML table row with publication date, source, and
    title of news item]
171      * </pre>
172      */
173     private static void processItem(XMLTree item, SimpleWriter
    out) {
174         assert item != null : "Violation of: item is not null";
175         assert out != null : "Violation of: out is not null";
176         assert item.isTag() && item.label().equals("item") : ""
177                 + "Violation of: the label root of item is an
    <item> tag";
178         assert out.isOpen() : "Violation of: out.is_open";
179
180         String pubDate = "No date available";
181         String titleOrDescr = "No title available";
182         String source = "No source available.";
183         String link = "";
184         String url = "";
185
186         int indexForTitle;
187
188         /*
189          * Checks if title tag exists as a child, gets index of
    description if
190          * title tag does not exist
191          */
192         if (getChildElement(item, "title") != -1) {
193             indexForTitle = getChildElement(item, "title");
194         } else {
```

```
195             indexForTitle = getChildElement(item, "description");
196         }
197 
198         /*
199          * Checks if title or description nodes have children
200          */
201         if (item.child(indexForTitle).numberOfChildren() > 0) {
202             titleOrDescr =
    item.child(indexForTitle).child(0).label();
203         }
204 
205         int indexForDate;
206 
207         /*
208          * Checks if pubDate exists as a child of item node
209          */
210         if (getChildElement(item, "pubDate") != -1) {
211             indexForDate = getChildElement(item, "pubDate");
212             pubDate = item.child(indexForDate).child(0).label();
213 
214         }
215 
216         int indexForSource;
217 
218         /*
219          * Checks if item has "source" child
220          */
221         if (getChildElement(item, "source") != -1) {
222             indexForSource = getChildElement(item, "source");
223             url =
    item.child(indexForSource).attributeValue("url");
224 
225             /*
226              * If "source" node has children, we get the source
    string of the
227              * item
228              */
229             if (item.child(indexForSource).numberOfChildren() > 0)
    {
230                 source =
    item.child(indexForSource).child(0).label();
231             }
232         }
233 
```

```
234         int indexForLink;
235 
236         /*
237          * Checks if "link" exists within the item node
238          */
239         if (getChildElement(item, "link") != -1) {
240             indexForLink = getChildElement(item, "link");
241             link = item.child(indexForLink).child(0).label();
242 
243         }
244 
245         out.println("  <tr>");
246         out.println("   <th>" + pubDate + "</th>");
247 
248         /*
249          * Checks if "url" is empty, and if so, then don't set the
    source name
250          * to a link
251          */
252         if (url.equals("")) {
253             out.println("   <th>" + source + "</th>");
254         } else {
255             out.println("   <th><a href='" + url + "'>" + source +
    "</a></th>");
256         }
257         out.println(
258                 "   <th><a href='" + link + "'>" + titleOrDescr +
    "</a></th>");
259         out.print("  </tr>");
260 
261     }
262 
263     /**
264      * Processes one XML RSS (version 2.0) feed from a given URL
    converting it
265      * into the corresponding HTML output file.
266      *
267      * @param url
268      *            the URL of the RSS feed
269      * @param file
270      *            the name of the HTML output file
271      * @param out
272      *            the output stream to report progress or errors
273      * @updates out.content
```

```
274      * @requires out.is_open
275      * @ensures <pre>
276      * [reads RSS feed from url, saves HTML document with table of
    news items
277      *   to file, appends to out.content any needed messages]
278      * </pre>
279      */
280     private static void processFeed(String url, String file,
    SimpleWriter out) {
281         XMLTree tree = new XMLTree1(url);
282 
283         /*
284          * Creates a writer for each rss feed in the feeds
285          */
286         SimpleWriter outHTML = new SimpleWriter1L(file);
287 
288         /*
289          * Checks if root node is a tag, and if it is, check if
    the label equals
290          * to "rss"
291          */
292         if (tree.isTag() && tree.label().equals("rss")) {
293 
294             /*
295              * Checks if node has attribute "version" and if it
    does, checks if
296              * it has version 2.0
297              */
298             if (tree.hasAttribute("version")
299                     &&
    tree.attributeValue("version").equals("2.0")) {
300                 XMLTree channel = tree.child(0);
301                 outputHeader(channel, outHTML);
302 
303                 /*
304                  * Goes through all the children and outputs the
    processed item
305                  * to the html file
306                  */
307                 for (int i = 0; i < channel.numberOfChildren(); i+
    +) {
308                     if (channel.child(i).label().equals("item")) {
309                         processItem(channel.child(i), outHTML);
310                     }
```

```
311                 }
312             } else {
313                 out.println("Sorry, not RSS or version 2.0");
314             }
315
316         }
317
318         outputFooter(outHTML);
319
320     }
321
322     /**
323      * Main method.
324      *
325      * @param args
326      *            the command line arguments; unused here
327      */
328     public static void main(String[] args) {
329         SimpleReader in = new SimpleReader1L();
330         SimpleWriter out = new SimpleWriter1L();
331
332         /*
333          * Asks for the XML document link
334          */
335         out.println("XML Document of Feeds: ");
336         String answer = in.nextLine();
337         XMLTree tree = new XMLTree1(answer);
338         out.println(tree);
339
340         String titleOfFeeds = tree.attributeValue("title");
341
342         /*
343          * Asks the name of the output file
344          */
345         out.println("Input name of HTML file: ");
346         String userHTML = in.nextLine();
347         SimpleWriter outFile = new SimpleWriter1L(userHTML +
    ".html");
348
349         /*
350          * Did not use outputHeader method because that only works
    for
351          * individual feeds, so wrote the beginning of main HTML
    document in
```

```
352          * this method
353          */
354         outFile.println("<html>");
355         outFile.println("<head>");
356         outFile.println("<title>" + titleOfFeeds + "</title>");
357         outFile.println("</head>");
358         outFile.println("<body>");
359         outFile.println(" <h1>" + titleOfFeeds + "</h1>");
360         outFile.println(" <ul>");
361 
362         /*
363          * Goes through the children of the "feeds" document and
    creates the
364          * HTML document for each feed, and creates an unordered
    list that links
365          * to each feed
366          */
367         for (int i = 0; i < tree.numberOfChildren(); i++) {
368             String name = tree.child(i).attributeValue("name");
369             String url = tree.child(i).attributeValue("url");
370             String file = tree.child(i).attributeValue("file");
371             processFeed(url, file, out);
372             outFile.println(
373                     "  <li><a href='" + file + "'>" + name + "</
    a></li>");
374         }
375 
376         /*
377          * Closing of opened tags
378          */
379         outFile.println(" </ul>");
380         outFile.println("</body>");
381         outFile.println("</html>");
382 
383         in.close();
384         out.close();
385         //outHTML.close();
386     }
387 
388 }
389 
```