```java
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;
import components.xmltree.XMLTree;
import components.xmltree.XMLTree1;

/**
 * Does pretty much what the RSSReader did except worse.
 *
 * @author Gage Farmer
 *
 */
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>
     * <ul>
     * <li>item 1</li>
     * </ul>
     *
     * @param xml
     *            the channel element XMLTree
     * @param out
     *            the output stream
     * @param writer
     *            it writes lol
     * @throws IOException
     * @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 xml, SimpleWriter out,
            FileWriter writer) throws IOException {
        assert xml != null : "Violation of: channel is not null";
        assert out != null : "Violation of: out is not null";
        assert out.isOpen() : "Violation of: out.is_open";

        writer.write("<html> <head> <title>" + xml.attributeValue("title")
                + "</title>" + "\n");
        writer.write("</head> <body>" + "\n");
        writer.write("<h1>" + xml.attributeValue("title") + "</h1>" + "\n");
        writer.write("<ul>");

```

```
 60     }
 61
 62     /**
 63      * Outputs the "opening" tags in the generated HTML file. These are the
 64      * expected elements generated by this method:
 65      *
 66      * <html> <head> <title>the channel tag title as the page title</title>
 67      * </head> <body>
 68      * <h1>the page title inside a link to the <channel> link</h1>
 69      * <p>
 70      * the channel description
 71      * </p>
 72      * <table border="1">
 73      * <tr>
 74      * <th>Date</th>
 75      * <th>Source</th>
 76      * <th>News</th>
 77      * </tr>
 78      *
 79      * @param xml
 80      *            the channel element XMLTree
 81      * @param out
 82      *            the output stream
 83      * @param writer
 84      *            it writes lol
 85      * @throws IOException
 86      * @updates out.content
 87      * @requires [the root of channel is a <channel> tag] and out.is_open
 88      * @ensures out.content = #out.content * [the HTML "opening" tags]
 89      */
 90     private static void outputSubHeader(XMLTree xml, SimpleWriter out,
 91             FileWriter writer) throws IOException {
 92
 93         writer.write("<html> <head> <title>"
 94                 + xml.child(getChildElement(xml, "title")).child(0) + "</title>"
 95                 + "\n");
 96
 97         writer.write("</head> <body>" + "\n");
 98
 99         writer.write("<h1> <a href=\""
100                 + xml.child(getChildElement(xml, "link")).child(0) + "\">"
101                 + xml.child(getChildElement(xml, "title")).child(0)
102                 + "</h1></a>\n");
103
104         writer.write(
105                 "<p>" + xml.child(getChildElement(xml, "description")).child(0)
106                         + "</p>" + "\n");
107
108         writer.write("<table border=\"1\">" + "<tr>" + "\n");
109         writer.write("<th> Date </th>" + "\n");
110         writer.write("<th> Source </th>" + "\n");
111         writer.write("<th> News </th>" + "\n");
112         writer.write("</tr>" + "\n");
113     }
114
115     /**
116      * Outputs the "closing" tags in the generated HTML file. These are the
117      * expected elements generated by this method:
118      *
```

```
119      * </table>
120      * </body> </html>
121      *
122      * @param out
123      *            the output stream
124      * @param writer
125      *            it writes lol
126      * @throws IOException
127      * @updates out.contents
128      * @requires out.is_open
129      * @ensures out.content = #out.content * [the HTML "closing" tags]
130      */
131     private static void outputFooter(SimpleWriter out, FileWriter writer)
132             throws IOException {
133         assert out != null : "Violation of: out is not null";
134         assert out.isOpen() : "Violation of: out.is_open";
135         writer.write("</ul>" + "\n");
136         writer.write("</body> </html>" + "\n");
137
138     }
139
140     /**
141      * Finds the first occurrence of the given tag among the children of the
142      * given {@code XMLTree} and return its index; returns -1 if not found.
143      *
144      * @param xml
145      *            the {@code XMLTree} to search
146      * @param tag
147      *            the tag to look for
148      * @return the index of the first child of type tag of the {@code XMLTree}
149      *         or -1 if not found
150      * @requires [the label of the root of xml is a tag]
151      * @ensures <pre>
152      * getChildElement =
153      *  [the index of the first child of type tag of the {@code XMLTree} or
154      *   -1 if not found]
155      * </pre>
156      */
157     private static int getChildElement(XMLTree xml, String tag) {
158         assert xml != null : "Violation of: xml is not null";
159         assert tag != null : "Violation of: tag is not null";
160         assert xml.isTag() : "Violation of: the label root of xml is a tag";
161
162         int i = 0;
163
164         while (i < xml.numberOfChildren() && xml.child(i).label() != tag) {
165             i++;
166         }
167
168         return i;
169     }
170
171     /**
172      * Processes one news item and outputs one table row. The row contains three
173      * elements: the publication date, the source, and the title (or
174      * description) of the item.
175      *
176      * @param item
177      *            the news item
```

```
178      * @param out
179      *            the output stream
180      * @param writer
181      *            it writes
182      * @throws IOException
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             FileWriter writer) throws IOException {
193         assert item != null : "Violation of: item is not null";
194         assert out != null : "Violation of: out is not null";
195         assert out.isOpen() : "Violation of: out.is_open";
196
197         writer.write("<li> <a href=\"" + item.attributeValue("file") + "\">"
198                 + item.attributeValue("name") + "</a></li>\n");
199
200     }
201
202     /**
203      * Processes one news item and outputs one table row. The row contains three
204      * elements: the publication date, the source, and the title (or
205      * description) of the item.
206      *
207      * @param item
208      *            the news item
209      * @param out
210      *            the output stream
211      * @throws IOException
212      * @updates out.content
213      * @requires [the label of the root of item is an <item> tag] and
214      *           out.is_open
215      * @ensures <pre>
216      * out.content = #out.content *
217      *   [an HTML table row with publication date, source, and title of news item]
218      * </pre>
219      */
220     private static void processSubItem(XMLTree item, SimpleWriter out,
221             FileWriter writer) throws IOException {
222         assert item != null : "Violation of: item is not null";
223         assert out != null : "Violation of: out is not null";
224         assert item.isTag() && item.label().equals("item") : ""
225                 + "Violation of: the label root of item is an <item> tag";
226         assert out.isOpen() : "Violation of: out.is_open";
227
228         writer.write("<tr><td>"
229                 + item.child(getChildElement(item, "pubDate")).child(0)
230                 + "</td>" + "\n");
231         if (item.child(getChildElement(item, "source")) != null) {
232             writer.write("<td><a href=\""
233                     + item.child(getChildElement(item, "source"))
234                             .attributeValue("url")
235                     + "\">"
236                     + item.child(getChildElement(item, "source")).child(0)
```

```
237                     + "</td>" + "\n");
238         } else {
239             writer.write("<td><a href=\"" + "No Source Available" + "\">"
240                     + "Link" + "</td>" + "\n");
241         }
242
243         if (item.child(getChildElement(item, "link")) != null
244                 && item.child(getChildElement(item, "title")) != null) {
245             writer.write("<td><a href=\""
246                     + item.child(getChildElement(item, "link")).child(0) + "\">"
247                     + item.child(getChildElement(item, "title")).child(0)
248                     + "</td>" + "\n");
249         } else {
250             writer.write("<td><a href=\""
251                     + item.child(getChildElement(item, "link")).child(0) + "\">"
252                     + "Link" + "</td>" + "\n");
253         }
254
255         System.out.println("Item Processed");
256
257     }
258
259     /**
260      * Processes one XML RSS (version 2.0) feed from a given URL converting it
261      * into the corresponding HTML output file.
262      *
263      * @param url
264      *            the URL of the RSS feed
265      * @param filep
266      *            the name of the HTML output file
267      * @param out
268      *            the output stream to report progress or errors
269      * @param sub
270      *            true if the url is part of a larger tree
271      * @throws IOException
272      * @updates out.content
273      * @requires out.is_open
274      * @ensures <pre>
275      * [reads RSS feed from url, saves HTML document with table of news items
276      *   to file, appends to out.content any needed messages]
277      * </pre>
278      */
279     private static void processFeed(String url, String filep, SimpleWriter out,
280             boolean sub) throws IOException {
281         File file = new File(filep);
282         FileWriter writer = new FileWriter(filep);
283         XMLTree xml = new XMLTree1(url);
284
285         if (sub) {
286             outputSubHeader(xml.child(0), out, writer);
287             for (int i = 0; i < xml.child(0).numberOfChildren(); i++) {
288                 if (xml.child(0).child(i).label() == "item") {
289                     System.out.println("Processing item " + i);
290                     processSubItem(xml.child(0).child(i), out, writer);
291                 }
292             }
293         } else {
294             outputHeader(xml, out, writer);
295             for (int i = 0; i < xml.numberOfChildren(); i++) {
```

```
296                 if (xml.child(i).label() == "feed") {
297                     processItem(xml.child(i), out, writer);
298                     processFeed(xml.child(i).attributeValue("url"),
299                             xml.child(i).attributeValue("file"), out, true);
300                 }
301             }
302         }
303 
304         outputFooter(out, writer);
305 
306         writer.close();
307     }
308 
309     /**
310      * Main method.
311      *
312      * @param args
313      *            the command line arguments; unused here
314      * @throws IOException
315      */
316     public static void main(String[] args) throws IOException {
317         SimpleReader in = new SimpleReader1L();
318         SimpleWriter out = new SimpleWriter1L();
319 
320         System.out.print("Enter an RSS feed URL: ");
321         String input = in.nextLine();
322 
323         // start of the process recursion loop
324         processFeed(input, "index.html", out, false);
325 
326         in.close();
327         out.close();
328     }
329 
330 }
331 
```