```java
import java.io.File;

/**
 * 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
     * @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) {
        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";

        try {
            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>");

        } catch (IOException e) {
            System.out.println("Error in outputHeader");
            e.printStackTrace();
        }
    }

    /**
     * Outputs the "opening" tags in the generated HTML file. These are the
     * expected elements generated by this method:
```

```
69      *
70      * <html> <head> <title>the channel tag title as the page title</title>
71      * </head> <body>
72      * <h1>the page title inside a link to the <channel> link</h1>
73      * <p>
74      * the channel description
75      * </p>
76      * <table border="1">
77      * <tr>
78      * <th>Date</th>
79      * <th>Source</th>
80      * <th>News</th>
81      * </tr>
82      *
83      * @param xml
84      *            the channel element XMLTree
85      * @param out
86      *            the output stream
87      * @param writer
88      *            it writes lol
89      * @updates out.content
90      * @requires [the root of channel is a <channel> tag] and out.is_open
91      * @ensures out.content = #out.content * [the HTML "opening" tags]
92      */
93     private static void outputSubHeader(XMLTree xml, SimpleWriter out,
94             FileWriter writer) {
95
96         try {
97
98             writer.write("<html> <head> <title>"
99                     + xml.child(getChildElement(xml, "title")).child(0)
100                    + "</title>" + "\n");
101
102            writer.write("</head> <body>" + "\n");
103
104            writer.write("<h1> <a href=\""
105                    + xml.child(getChildElement(xml, "link")).child(0) + "\">"
106                    + xml.child(getChildElement(xml, "title")).child(0)
107                    + "</h1></a>\n");
108
109            writer.write("<p>"
110                    + xml.child(getChildElement(xml, "description")).child(0)
111                    + "</p>" + "\n");
112
113            writer.write("<table border=\"1\">" + "<tr>" + "\n");
114            writer.write("<th> Date </th>" + "\n");
115            writer.write("<th> Source </th>" + "\n");
116            writer.write("<th> News </th>" + "\n");
117            writer.write("</tr>" + "\n");
118        } catch (IOException e) {
119            System.out.println("Error in outputHeader");
120            e.printStackTrace();
121        }
122    }
123
125     * Outputs the "closing" tags in the generated HTML file. These are the
139    private static void outputFooter(SimpleWriter out, FileWriter writer) {
140        assert out != null : "Violation of: out is not null";
141        assert out.isOpen() : "Violation of: out.is_open";
```

```
142         try {
143             writer.write("</ul>" + "\n");
144             writer.write("</body> </html>" + "\n");
145         } catch (IOException e) {
146             System.out.println("Error in outputFooter");
147             e.printStackTrace();
148         }
149 
150     }
151 
152     /**
153      * Finds the first occurrence of the given tag among the children of the
154      * given {@code XMLTree} and return its index; returns -1 if not found.
155      *
156      * @param xml
157      *            the {@code XMLTree} to search
158      * @param tag
159      *            the tag to look for
160      * @return the index of the first child of type tag of the {@code XMLTree}
161      *         or -1 if not found
162      * @requires [the label of the root of xml is a tag]
163      * @ensures <pre>
164      * getChildElement =
165      *  [the index of the first child of type tag of the {@code XMLTree} or
166      *   -1 if not found]
167      * </pre>
168      */
169     private static int getChildElement(XMLTree xml, String tag) {
170         assert xml != null : "Violation of: xml is not null";
171         assert tag != null : "Violation of: tag is not null";
172         assert xml.isTag() : "Violation of: the label root of xml is a tag";
173 
174         int i = 0;
175 
176         while (i < xml.numberOfChildren()) {
177             if (xml.child(i).label() == tag) {
178                 break;
179             } else {
180                 i++;
181             }
182         }
183 
184         return i;
185     }
186 
187     /**
188      * Processes one news item and outputs one table row. The row contains three
189      * elements: the publication date, the source, and the title (or
190      * description) of the item.
191      *
192      * @param item
193      *            the news item
194      * @param out
195      *            the output stream
196      * @param writer
197      *            it writes
198      * @updates out.content
199      * @requires [the label of the root of item is an <item> tag] and
200      *           out.is_open
```

```
201      * @ensures <pre>
202      * out.content = #out.content *
203      *   [an HTML table row with publication date, source, and title of news item]
204      * </pre>
205      */
206     private static void processItem(XMLTree item, SimpleWriter out,
207             FileWriter writer) {
208         assert item != null : "Violation of: item is not null";
209         assert out != null : "Violation of: out is not null";
210         assert out.isOpen() : "Violation of: out.is_open";
211
212         try {
213
214             writer.write("<li> <a href=\"" + item.attributeValue("file") + "\">"
215                     + item.attributeValue("name") + "</a></li>\n");
216
217         } catch (IOException e) {
218             System.out.println("Error in processItem");
219             e.printStackTrace();
220         }
221
222     }
223
224     /**
225      * Processes one news item and outputs one table row. The row contains three
226      * elements: the publication date, the source, and the title (or
227      * description) of the item.
228      *
229      * @param item
230      *            the news item
231      * @param out
232      *            the output stream
233      * @updates out.content
234      * @requires [the label of the root of item is an <item> tag] and
235      *           out.is_open
236      * @ensures <pre>
237      * out.content = #out.content *
238      *   [an HTML table row with publication date, source, and title of news item]
239      * </pre>
240      */
241     private static void processSubItem(XMLTree item, SimpleWriter out,
242             FileWriter writer) {
243         assert item != null : "Violation of: item is not null";
244         assert out != null : "Violation of: out is not null";
245         assert item.isTag() && item.label().equals("item") : ""
246                 + "Violation of: the label root of item is an <item> tag";
247         assert out.isOpen() : "Violation of: out.is_open";
248
249         try {
250
251             writer.write("<tr><td>"
252                     + item.child(getChildElement(item, "pubDate")).child(0)
253                     + "</td>" + "\n");
254             try {
255                 writer.write("<td><a href=\""
256                         + item.child(getChildElement(item, "source"))
257                                 .attributeValue("url")
258                         + "\">"
259                         + item.child(getChildElement(item, "source")).child(0)
```

```
260                         + "</td>" + "\n");
261             } catch (AssertionError a) {
262                 writer.write("<td><a href=\"" + "No Source Available" + "\">"
263                         + "Link" + "</td>" + "\n");
264             }
265
266             try {
267                 writer.write("<td><a href=\""
268                         + item.child(getChildElement(item, "link")).child(0)
269                         + "\">"
270                         + item.child(getChildElement(item, "title")).child(0)
271                         + "</td>" + "\n");
272             } catch (AssertionError a) {
273                 writer.write("<td><a href=\""
274                         + item.child(getChildElement(item, "link")).child(0)
275                         + "\">" + "Link" + "</td>" + "\n");
276             }
277
278             System.out.println("Item Processed");
279
280         } catch (IOException e) {
281             System.out.println("Error in processItem");
282             e.printStackTrace();
283         }
284
285     }
286
287     /**
288      * Processes one XML RSS (version 2.0) feed from a given URL converting it
289      * into the corresponding HTML output file.
290      *
291      * @param url
292      *            the URL of the RSS feed
293      * @param filep
294      *            the name of the HTML output file
295      * @param out
296      *            the output stream to report progress or errors
297      * @param sub
298      *            true if the url is part of a larger tree
299      * @throws IOException
300      * @updates out.content
301      * @requires out.is_open
302      * @ensures <pre>
303      * [reads RSS feed from url, saves HTML document with table of news items
304      *   to file, appends to out.content any needed messages]
305      * </pre>
306      */
307     private static void processFeed(String url, String filep, SimpleWriter out,
308             boolean sub) throws IOException {
309         File file = new File(filep);
310         FileWriter writer = new FileWriter(filep);
311         XMLTree xml = new XMLTree1(url);
312
313         if (sub) {
314             outputSubHeader(xml.child(0), out, writer);
315             for (int i = 0; i < xml.child(0).numberOfChildren(); i++) {
316                 if (xml.child(0).child(i).label() == "item") {
317                     System.out.println("Processing item " + i);
318                     processSubItem(xml.child(0).child(i), out, writer);
```

```
319                 }
320             }
321         } else {
322             outputHeader(xml, out, writer);
323             for (int i = 0; i < xml.numberOfChildren(); i++) {
324                 if (xml.child(i).label() == "feed") {
325                     processItem(xml.child(i), out, writer);
326                     processFeed(xml.child(i).attributeValue("url"),
327                             xml.child(i).attributeValue("file"), out, true);
328                 }
329             }
330         }
331 
332         outputFooter(out, writer);
333 
334         writer.close();
335     }
336 
337     /**
338      * Main method.
339      *
340      * @param args
341      *            the command line arguments; unused here
342      * @throws IOException
343      */
344     public static void main(String[] args) throws IOException {
345         SimpleReader in = new SimpleReader1L();
346         SimpleWriter out = new SimpleWriter1L();
347 
348         System.out.print("Enter an RSS feed URL: ");
349         String input = in.nextLine();
350 
351         // start of the process recursion loop
352         processFeed(input, "index.html", out, false);
353 
354         in.close();
355         out.close();
356     }
357 
358 }
```