```java
import java.io.File;

/**
 * Program to convert an XML RSS (version 2.0) feed from a given URL into the
 * corresponding HTML output file.
 *
 * @author Gage Farmer
 *
 */
public final class RSSReader {

    /**
     * Private constructor so this utility class cannot be instantiated.
     */
    private RSSReader() {
    }

    /**
     * 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
     * @ensures out.content = #out.content * [the HTML "opening" tags]
     */
    private static void outputHeader(XMLTree channel, SimpleWriter out,
            FileWriter writer) {
        assert channel != null : "Violation of: channel is not null";
        assert out != null : "Violation of: out is not null";
        assert channel.isTag() && channel.label().equals("channel") : ""
                + "Violation of: the label root of channel is a <channel> tag";
        assert out.isOpen() : "Violation of: out.is_open";

        try {

            writer.write("<html> <head> <title>"
                    + channel.child(getChildElement(channel, "title")).child(0)
                    + "</title>" + "\n");
            writer.write("</head> <body>" + "\n");
            writer.write("<h1> <a href=\""
                    + channel.child(getChildElement(channel, "link")).child(0)
                    + "\">"
```

```
 69                     + channel.child(getChildElement(channel, "title")).child(0)
 70                     + "</h1></a>\n");
 71             writer.write("<p>" + channel
 72                     .child(getChildElement(channel, "description")).child(0)
 73                     + "</p>" + "\n");
 74             writer.write("<table border=\"1\">" + "<tr>" + "\n");
 75             writer.write("<th> Date </th>" + "\n");
 76             writer.write("<th> Source </th>" + "\n");
 77             writer.write("<th> News </th>" + "\n");
 78             writer.write("</tr>" + "\n");
 79         } catch (IOException e) {
 80             System.out.println("Error in outputHeader");
 81             e.printStackTrace();
 82         }
 83     }
 84
 85     /**
 86      * Outputs the "closing" tags in the generated HTML file. These are the
 87      * expected elements generated by this method:
 88      *
 89      * </table>
 90      * </body> </html>
 91      *
 92      * @param out
 93      *            the output stream
 94      * @updates out.contents
 95      * @requires out.is_open
 96      * @ensures out.content = #out.content * [the HTML "closing" tags]
 97      */
 98     private static void outputFooter(SimpleWriter out, FileWriter writer) {
 99         assert out != null : "Violation of: out is not null";
100         assert out.isOpen() : "Violation of: out.is_open";
101         try {
102             writer.write("</table>" + "\n");
103             writer.write("</body> </html>" + "\n");
104         } catch (IOException e) {
105             System.out.println("Error in outputFooter");
106             e.printStackTrace();
107         }
108
109     }
110
111     /**
112      * Finds the first occurrence of the given tag among the children of the
113      * given {@code XMLTree} and return its index; returns -1 if not found.
114      *
115      * @param xml
116      *            the {@code XMLTree} to search
117      * @param tag
118      *            the tag to look for
119      * @return the index of the first child of type tag of the {@code XMLTree}
120      *         or -1 if not found
121      * @requires [the label of the root of xml is a tag]
122      * @ensures <pre>
123      * getChildElement =
124      *  [the index of the first child of type tag of the {@code XMLTree} or
125      *   -1 if not found]
126      * </pre>
127      */
```

```
128     private static int getChildElement(XMLTree xml, String tag) {
129         assert xml != null : "Violation of: xml is not null";
130         assert tag != null : "Violation of: tag is not null";
131         assert xml.isTag() : "Violation of: the label root of xml is a tag";
132
133         boolean found = false;
134         XMLTree temp = xml;
135         int i = 0;
136
137         while (!found) {
138             if (xml.child(i).label() == tag) {
139                 found = true;
140             } else {
141                 i++;
142             }
143         }
144
145         return i;
146     }
147
148     /**
149      * Processes one news item and outputs one table row. The row contains three
150      * elements: the publication date, the source, and the title (or
151      * description) of the item.
152      *
153      * @param item
154      *            the news item
155      * @param out
156      *            the output stream
157      * @updates out.content
158      * @requires [the label of the root of item is an <item> tag] and
159      *           out.is_open
160      * @ensures <pre>
161      * out.content = #out.content *
162      *   [an HTML table row with publication date, source, and title of news item]
163      * </pre>
164      */
165     private static void processItem(XMLTree item, SimpleWriter out,
166             FileWriter writer) {
167         assert item != null : "Violation of: item is not null";
168         assert out != null : "Violation of: out is not null";
169         assert item.isTag() && item.label().equals("item") : ""
170                 + "Violation of: the label root of item is an <item> tag";
171         assert out.isOpen() : "Violation of: out.is_open";
172
173         try {
174
175             writer.write("<tr><td>"
176                     + item.child(getChildElement(item, "pubDate")).child(0)
177                     + "</td>" + "\n");
178             writer.write("<td><a href=\""
179                     + item.child(getChildElement(item, "source"))
180                             .attributeValue("url")
181                     + "\">"
182                     + item.child(getChildElement(item, "source")).child(0)
183                     + "</td>" + "\n");
184             writer.write("<td><a href=\""
185                     + item.child(getChildElement(item, "link")).child(0) + "\">"
186                     + item.child(getChildElement(item, "title")).child(0)
```

```
187                     + "</td>" + "\n");
188
189         } catch (IOException e) {
190             System.out.println("Error in processItem");
191             e.printStackTrace();
192         }
193
194     }
195
196     /**
197      * Main method.
198      *
199      * @param args
200      *             the command line arguments; unused here
201      * @throws IOException
202      */
203     public static void main(String[] args) throws IOException {
204         SimpleReader in = new SimpleReader1L();
205         SimpleWriter out = new SimpleWriter1L();
206         File page = new File("page.html");
207         FileWriter writer = new FileWriter("page.html");
208
209         System.out.print("Enter an RSS feed URL: ");
210         String input = in.nextLine();
211         XMLTree channel = new XMLTree1(input);
212         channel = channel.child(0);
213
214         outputHeader(channel, out, writer);
215
216         for (int i = 0; i < channel.numberOfChildren(); i++) {
217             if (channel.child(i).label() == "item") {
218                 processItem(channel.child(i), out, writer);
219             }
220         }
221
222         outputFooter(out, writer);
223
224         writer.close();
225         in.close();
226         out.close();
227     }
228
229 }
```