```java
import components.simplereader.SimpleReader;

/**
 * Program to convert an XML file stored with RSS feeds (version 2.0) feed from
 * a given URL into an HTML index file, then process all RSS feeds into their
 * own HTML files that are accessible from the index file.
 *
 * @author Isaac Frank
 *
 */
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
     * @ensures out.content = #out.content * [the HTML "opening" tags]
     */
    private static void outputHeader(XMLTree channel, SimpleWriter out) {
        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";

        // Getting indexes of title, link, and description
        int titleChildIndex = getChildElement(channel, "title");
        int linkChildIndex = getChildElement(channel, "link");
        int descriptionChildIndex = getChildElement(channel, "description");

        // Html opening tags and printing title
        out.print("<html> <head> <title>");
        String title = "Empty Title";
        if (channel.child(titleChildIndex).numberOfChildren() > 0) {
            title = channel.child(titleChildIndex).child(0).label();
        }
        out.print(title);

        // Html closing tags
        out.println("</title> </head> <body>");

        // Html header and page title w/ link
        out.print("<h1>");
        out.print(
                "<a href = \"" + channel.child(linkChildIndex).child(0).label()
                        + "\">" + title + "</a>");
        out.println("</h1>");

        // Html paragraph w/ channel description
        out.println("<p>");
        String description = "No description";
        if (channel.child(descriptionChildIndex).numberOfChildren() > 0) {
            description = channel.child(descriptionChildIndex).child(0).label();
        }
        out.println(description);
        out.println("</p>");

        // Table Headers
        out.println("<table border = \"" + 1 + "\">");
        out.println("<tr>");
        out.println("<th>" + "Date" + "</th>");
        out.println("<th>" + "Source" + "</th>");
        out.println("<th>" + "News" + "</th>");
        out.println("</tr>");
    }

    /**
     * Outputs the "closing" tags in the generated HTML file. These are the
     * expected elements generated by this method:
     *
     * </table>
     * </body> </html>
     *
     * @param out
     *            the output stream
     * @updates out.contents
     * @requires out.is_open
     * @ensures out.content = #out.content * [the HTML "closing" tags]
     */
    private static void outputFooter(SimpleWriter out) {
        assert out != null : "Violation of: out is not null";
        assert out.isOpen() : "Violation of: out.is_open";

        out.println("</table>");
        out.println("</body> </html>");
    }

    /**
```

```java
     * Finds the first occurrence of the given tag among the children of the
     * given {@code XMLTree} and return its index; returns -1 if not found.
     *
     * @param xml
     *            the {@code XMLTree} to search
     * @param tag
     *            the tag to look for
     * @return the index of the first child of type tag of the {@code XMLTree}
     *         or -1 if not found
     * @requires [the label of the root of xml is a tag]
     * @ensures <pre>
     * getChildElement =
     *  [the index of the first child of type tag of the {@code XMLTree} or
     *   -1 if not found]
     * </pre>
     */
    private static int getChildElement(XMLTree xml, String tag) {
        assert xml != null : "Violation of: xml is not null";
        assert tag != null : "Violation of: tag is not null";
        assert xml.isTag() : "Violation of: the label root of xml is a tag";

        int index = -1;

        // Iterates through children until a tag is found or all children are searched
        int i = 0;
        while (i < xml.numberOfChildren() && index < 0) {
            if (xml.child(i).isTag() && xml.child(i).label().equals(tag)) {
                index = i;
            }
            i++;
        }

        return index;
    }

    /**
     * Processes one news item and outputs one table row. The row
```

```java
contains three
     * elements: the publication date, the source, and the title (or
     * description) of the item.
     *
     * @param item
     *            the news item
     * @param out
     *            the output stream
     * @updates out.content
     * @requires [the label of the root of item is an <item> tag] and
     *           out.is_open
     * @ensures <pre>
     * out.content = #out.content *
     *   [an HTML table row with publication date, source, and title of news item]
     * </pre>
     */
    private static void processItem(XMLTree item, SimpleWriter out) {
        assert item != null : "Violation of: item is not null";
        assert out != null : "Violation of: out is not null";
        assert item.isTag() && item.label().equals("item") : ""
                + "Violation of: the label root of item is an <item> tag";
        assert out.isOpen() : "Violation of: out.is_open";

        // Finding indexes
        int titleChildIndex = getChildElement(item, "title");
        int linkChildIndex = getChildElement(item, "link");
        int descriptionChildIndex = getChildElement(item, "description");
        int pubDateChildIndex = getChildElement(item, "pubDate");
        int sourceChildIndex = getChildElement(item, "source");

        out.println("<tr>");

        // Printing pubDate table cell
        String pubDate = "No date available";
        if (pubDateChildIndex != -1) {
            pubDate = item.child(pubDateChildIndex).child(0).label();
        }
        out.println("<td>" + pubDate + "</td>");

        // Printing source table cell
        String source = "No source available";
        if (sourceChildIndex != -1) {
            XMLTree src = item.child(sourceChildIndex);
            if (src.numberOfChildren() > 0) {
                String srcAttributeVal = src.attributeValue("url");
                source = "<a href = \"" + srcAttributeVal;
                source += "\">" + src.child(0).label() + "</a>";
            }
        }
        out.println("<td>" + source + "</td>");

        // Printing title table cell, checking if description and link are needed
        String titleOrDsc = "No title available";
        String link = "";
        if (titleChildIndex != -1
                && item.child(titleChildIndex).numberOfChildren() > 0) {
            titleOrDsc = item.child(titleChildIndex).child(0).label();
        } else if (descriptionChildIndex != -1
                && item.child(descriptionChildIndex).numberOfChildren() > 0) {
            titleOrDsc = item.child(descriptionChildIndex).child(0).label();
        }
        if (linkChildIndex != -1) {
            link = item.child(linkChildIndex).child(0).label();
        }

        out.print("<td><a href = \"" + link + "\">" + titleOrDsc + "</a></td>");

        // Ending the row
        out.println("</tr>");
    }

    /**
     * Processes one XML RSS (version 2.0) feed from a given URL converting it
     * into the corresponding HTML output file.
     *
     * @param url
     *            the URL of the RSS feed
     * @param file
     *            the name of the HTML output file
     * @param out
     *            the output stream to report progress or errors
     * @updates out.content
     * @requires out.is_open
     * @ensures <pre>
     * [reads RSS feed from url, saves HTML document with table of news items
     *   to file, appends to out.content any needed messages]
     * </pre>
     */
    private static void processFeed(String url, String file, SimpleWriter out) {
        SimpleWriter outFile = new SimpleWriter1L(file);

        // Checking if xml is RSS 2.0
        XMLTree xml = new XMLTree1(url);
        if (xml.isTag()) {
            if (xml.hasAttribute("version")) {
                if (xml.attributeValue("version").equals("2.0")) {
                    XMLTree channel = xml.child(0);
                    outputHeader(channel, outFile);

                    // Iterating through all "item" children of channel
                    int i = 0;
                    while (i < channel.numberOfChildren()) {
                        if (channel.child(i).label().equals("item")) {
                            processItem(channel.child(i), outFile);
                        }
                        i++;
                    }

                    outputFooter(outFile);
                }
            }
        } else {
            out.println("This is not an RSS 2.0 file");
        }
        outFile.close();
    }

    /**
     * Main method.
     *
     * @param args
     *            the command line arguments; unused here
     *
     *            Processes XML document with RSS feeds stored inside and
     *            outputs them to an index HTML file
     */
    public static void main(String[] args) {
        SimpleReader in = new SimpleReader1L();
        SimpleWriter out = new SimpleWriter1L();
        SimpleWriter outFile = new SimpleWriter1L("index.html");

        // User input
        out.print("Enter a URL for an XML Document of RSS feeds: ");
        String url = in.nextLine();

        XMLTree feeds = new XMLTree1(url);

        // Outputting HTML headers and starting an unordered list
        outFile.print("<html> <head> <title>");
        outFile.print(feeds.attributeValue("title"));
        outFile.println("</title> </head>");
        outFile.println("<h1>" + feeds.attributeValue("title") + "</h1>");
        outFile.println("<ul>");

        // Iterating through all RSS feeds stored in XMLTree feeds
        int i = 0;
        int numChildren = feeds.numberOfChildren();
        while (i < numChildren) {
            // Creating a new list element with a link to each processed RSS feed
            outFile.print("<li>");
            XMLTree feed = feeds.child(i);
            outFile.print("<a href = \"" +
feed.attributeValue("file"));
            outFile.print("\">" + feed.attributeValue("name") +
"</a>");
            processFeed(feed.attributeValue("url"),
feed.attributeValue("file"),
                    out);
            outFile.println("</li>");
            i++;
        }

        // Outputting HTML footers and closing unordered list
        outFile.println("</ul>");
        outFile.println("</body>");
        outFile.println("</html>");

        // Closing output streams
        in.close();
        out.close();
        outFile.close();

    }
}

```