```java
import components.simplereader.SimpleReader;

/**
 * Program to convert a set of XML RSS (version 2.0) feed from a given URL into
 * the corresponding HTML output file. There will be an index page with links to
 * all the single RSS feeds.
 *
 * @author Robert Frenken
 *
 */
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";

        // know from earlier that this is a valid rss, so all these elements are
        // a part of the rss

        int titleIndex = getChildElement(channel, "title");
        int linkIndex = getChildElement(channel, "link");
        String title = "";
        String link = "";
        if (channel.child(titleIndex).numberOfChildren() > 0) {
            title = channel.child(titleIndex).child(0).label();
        }
        if (channel.child(linkIndex).numberOfChildren() > 0) {
            link = channel.child(linkIndex).child(0).label();
        }

        out.print("<html xmlns=\"http://www.w3.org/1999/xhtml\">\r\n"
                + "<head>\r\n"
                + "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=ISO-8859-1\" />\r\n"
                + "<title>" + title + "</title>\r\n" + "\r\n" + "</head>\r\n"
                + "\r\n" + "\r\n" + "<body>");

        out.print("<h1>\r\n" + "<a href= \"" + link + "\">" + title + "</a>\r\n"
                + "</h1>\r\n");

        int descriptionIndex = getChildElement(channel, "description");
        if (channel.child(descriptionIndex).numberOfChildren() > 0) {
            String description = channel.child(descriptionIndex).child(0)
                    .label();
            out.print("<p>" + description + "</p>\r\n");
        } else {
            out.print("<p>" + "No description available" + "</p>\r\n");
        }

        out.print("<table border = \"1\">\r\n" + "<tr>\r\n"
                + "<th>Date</th>\r\n" + "<th>Source</th>\r\n"
                + "<th>News</th>\r\n" + "</tr>\r\n");

    }

    /**
     * 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.print("</table>\r\n" + "</body>\r\n" + "</html>");
    }

    /**
     * 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 counter = 0;
        int elementIndex = -1;
        while (counter < xml.numberOfChildren() && elementIndex != counter) {
            if (xml.child(counter).isTag()
                    && tag.equals(xml.child(counter).label())) {
                elementIndex = counter;
            }
            counter++;
        }

        return elementIndex;
    }

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

        out.print("<tr>\r\n");

        int pubDateIndex = getChildElement(item, "pubDate");
        // pubdate column, either there or not
        if (pubDateIndex != -1) {
            String pubDate = item.child(pubDateIndex).child(0).toString();
            out.print("<td>" + pubDate + "</td>\r");
        } else {
            out.print("<td>No date available</td>");
        }

        int sourceIndex = getChildElement(item, "source");
        // source column, either there with url, or not
        if (sourceIndex != -1) {
            if (item.child(sourceIndex).numberOfChildren() > 0) {
                String source = item.child(sourceIndex).child(0).label();
                String sourceURL = item.child(sourceIndex)
                        .attributeValue("url");
                out.println("<td>\r\n" + "<a href= \"" + sourceURL + "\">"
                        + source + "</a>\r\n" + "</td>\r\n");
            }

        } else {
            out.print("<td>No source available</td>");
        }

        // title column, with either a title with or without a link, or a description
        // with or without a link, or "no title available" with or without a link

        int titleIndex = getChildElement(item, "title");
        int descriptionIndex = getChildElement(item, "description");
        int linkIndex = getChildElement(item, "link");
        if (titleIndex != -1 && item.child(titleIndex).numberOfChildren() > 0) {

            String title = item.child(titleIndex).child(0).label();

            // checks if a link tag exists in the particular item

            if (linkIndex != -1) {
                String link = item.child(linkIndex).child(0).label();
                out.println("<td>\r\n" + "<a href= \"" + link + "\">" + title
                        + "</a>\r\n" + "</td>\r\n");
            } else {
                out.println("<td>\r\n" + title + "</td>\r\n");
            }

        } else if (descriptionIndex != -1) {
            if (item.child(descriptionIndex).numberOfChildren() > 0) {
                String description = item.child(descriptionIndex).child(0)
                        .label();
                // checks if a link tag exists in the particular item
                if (getChildElement(item, "link") != 1) {
                    String link = item.child(linkIndex).child(0).label();
                    out.println("<td>\r\n" + "<a href= \"" + link + "\">"
                            + description + "</a>\r\n" + "</td>\r\n");
                } else {
                    out.println("<td>\r\n" + description + "</td>\r\n");
                }
            }

        } else {
            if (linkIndex != -1) {
                String link = item.child(linkIndex).child(0).label();
                out.println("<td>\r\n" + "<a href= \"" + link + "\">"
                        + "No title available" + "</a>\r\n" + "</td>\r\n");
            } else {
                out.print("<td>No title available</td>");

            }
        }
        out.print("</tr>\r\n");
    }

    /**
     * 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) {
        XMLTree xml = new XMLTree1(url);
        SimpleWriter outHTML = new SimpleWriter1L(file);
        outputHeader(xml.child(0), outHTML);
        // create channel xml similar to lab
        XMLTree channel = xml.child(0);

        // go through all child tags of channel, looking for item tags
        for (int i = 0; i < channel.numberOfChildren(); i++) {

            if (channel.child(i).isTag()
                    && channel.child(i).label().equals("item")) {

                processItem(channel.child(i), outHTML);
            }
        }
        outputFooter(outHTML);

        outHTML.close();
    }

    /**
     * Main method.
     *
     * @param args
     *            the command line arguments; unused here
     */
    public static void main(String[] args) {
        SimpleReader in = new SimpleReader1L();
        SimpleWriter out = new SimpleWriter1L();

        out.print(
                "Enter the URL XML containing a list of RSS 2.0 news feeds: ");
        String urlFeed = in.nextLine();

        XMLTree xmlFeed = new XMLTree1(urlFeed);

        // makes sure the top tag is correct
        while (!xmlFeed.hasAttribute("title")
                || !xmlFeed.label().equals("feeds")) {
            out.println("This is not a valid xml with a list of RSS feeds");
            out.print(
                    "Enter the URL XML containing a list of RSS 2.0 news feeds: ");
            urlFeed = in.nextLine();
            xmlFeed = new XMLTree1(urlFeed);
        }

        // ask user to output to an html file
        out.print("Enter an html file to publish the list of RSS feeds: ");
        String htmlFile = in.nextLine();
        SimpleWriter outHTML = new SimpleWriter1L(htmlFile);

        // create header for HTML page
        outHTML.print("<html>\r\n" + "<head>\r\n" + "<title>\r\n"
                + xmlFeed.attributeValue("title"));
        outHTML.print("</title>\r\n" + "</head>\r\n");
        outHTML.print("<body>\r\n" + "<h2>" + xmlFeed.attributeValue("title")
                + "</h2>\r\n");
        outHTML.print("<ul>\r\n");
        // create body for HTML page
        for (int i = 0; i < xmlFeed.numberOfChildren(); i++) {
            outHTML.print("<li>\r\n");
            outHTML.print("<a href = " + "\""
                    + xmlFeed.child(i).attributeValue("file") + "\"" + ">"
                    + xmlFeed.child(i).attributeValue("name") + "</a>\r\n");
            outHTML.print("</li>\r\n");
        }
        // create footer for HTML page
        outHTML.print("</ul>\r\n" + "</body>\r\n" + "</html>");

        // iterate through the feeds to determine if each feed is a valid RSS feed
        for (int i = 0; i < xmlFeed.numberOfChildren(); i++) {
            String url = xmlFeed.child(i).attributeValue("url");
            // Puts url into xml tree object
            XMLTree xml = new XMLTree1(url);

            /*
             * Makes sure XML is an rss Checks if it has an attribute version,
             * that it's value is "2.0", and if the label name is "rss".
             */
            while (!xml.hasAttribute("version")
                    || !(xml.attributeValue("version").equals("2.0"))
                    || !(xml.label().equals("rss"))) {

```

```
                out.println("This is a valid XML, but not an RSS.");
                out.print("Enter the URL of an RSS 2.0 news feed: ");
                url = in.nextLine();
                xml = new XMLTree1(url);
            }
        }

        for (int i = 0; i < xmlFeed.numberOfChildren(); i++) {
            processFeed(xmlFeed.child(i).attributeValue("url"),
                    xmlFeed.child(i).attributeValue("file"), out);
            out.println(xmlFeed.child(i).attributeValue("file"));
        }

        in.close();

        out.close();
        outHTML.close();
    }

}
```