Homework #14
due Wednesday, May 15, 10:00 PM

In this assignment you will implement a "web scraper" that pulls UWM schedule information off of web pages and does a simple conflict check on lecture sections.

1 Concerning the `XMLTokenizer`

XML is a widely used as a data format. It has the advantage as being both human and machine readable. It also leaves open the possibility for extension, with backward compatibility. Please consult the "XML Primer" handout about XML syntax. We will use a home-grown XML tokenizer to read XML files[1] The javadoc is on the course web page.

The XML tokenizer breaks up the input into tokens of one of the following forms (`XMLTokenType`):

OPEN `<`*name*

ATTR *name*`="`*text*`"`

CLOSE `>`

ECLOSE `/>`

ETAG `</`*name*`>`

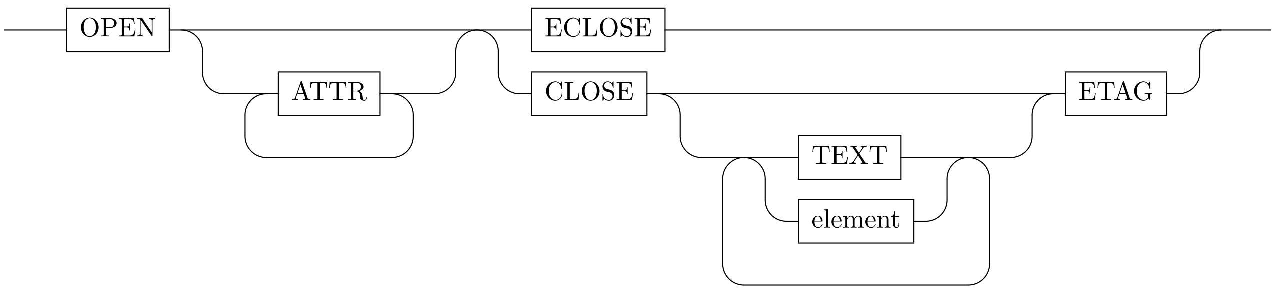
TEXT *text*

ERROR Something went wrong, message is *text*. The error is fatal in that no further tokenizing should be done.

Not all these tokens can occur at any place. Ignoring errors (which can occur anywhere), an `OPEN` is always followed by zero or more `ATTR` tokens and then either a `CLOSE` or `ECLOSE`. The latter three tokens (`ATTR`, `CLOSE` and `ECLOSE`) only occur after `OPEN` in that way. If an element is not closed with `ECLOSE`, it needs to be closed with `ETAG`, in which case the names should match, but the tokenizer doesn't check this property.

The following diagram shows in what order tokens (other than `ERROR`) can appear:

element



This order is implemented (correctly!) by the tokenizer—you don't need to check it. In other words, if you're checking if an `ATTR` tag might appear at the start of an element, you're checking something that will never happen. Don't do that.

[1]This tokenizer is not 100% XML compliant but does pretty well.

To get the next token, use the method `next()`. Some of the tokens come with extra information (*name* or *text*) above. These can be queried by using the `getCurrentName` and/or `getCurrentText` methods of the XML tokenizer. The most recently returned token can be retrieved using `current()`. Alternatively, one can call `saveToken` and then call `next`; the former method ensures that `next()` reuses the current token. The `saveToken` method is very useful if you want to start reading something and then back up (call `saveToken`) and let the same token be read again somewhere else in the code. You can only back up by a single token.

If you want to handle a large number of different token types, you will find your code simpler if you use Java's `switch` keyword. Don't forget to use `break` statements!

Not all HTML is fully compliant XML. In particular, sometimes elements are not properly closed. Also HTML pages can include "script" and "style" elements that are not true XML. These days, however, most web pages are *close* to XML-compliant. You will use the XML tokenizer to read HTML, with "script" and "style" elements handled specially.

2 Concerning the Schedule-Related Classes

We provide a number of ADTs implemented as Java classes that give a simplified model of UWM courses:

Day Day of the week enumeration (single letter).

Days Set of days, e.g. `MW`.

Time Time of day, e.g. `12:45 PM`

Period Period of time for a class (combines "days" and "time").

Course Curricular code and number, and optionally a title, e.g. `COMPSCI-150`.

Kind Kind of section enumeration, e.g. `LAB`.

Section A particular section of a course, e.g. `DIS 603`.

Term Semester term, e.g. `FALL_2019`.

Many of these classes override `equals`, `hashCode` and `toString`, and some further override `fromString`. All Java enumerations provide a `valueOf` method that behaves similarly to our `fromString` methods. You should be familiar with these methods and how to use them. These classes are all provided in the JAR file for the project, together with source code to help you understand the implementation.

3 Concerning the `ImportUWMSchedule` class

One of the classes you will be implementing is `ImportUWMSchedule` which has the task to read courses and sections off HTML pages. The class is pointed to a particular source of HTML and then has a method to read this HTML and create section objects (and courses, periods, times etc.). Essentially it performs the "magical" of converting a web page into Java objects. This class is not amenable to normal unit tests.

We have implemented the whole class for you except the `read` method which does this work. Working with a large messy HTML file can be a little tedious. For that reason, the code is expected to use some helper methods for moving quickly through a HTML looking for particular elements. For this homework, you will implement three helper methods (in class `XMLTokenizerUtil`):

skipUntilOpen Skip tokens until we read an `OPEN` token of the desired form;

skipElement Consume all tokens until the current element is complete.

readTR Consume all tokens until the current table row (`<tr>`) element is read, and then return a list of the text in table data (`<td>`).

All these methods should immediately return when an `ERROR` token is encountered or if the stream of tokens comes to an end.

Starting in Fall 2019, UWM has two dual sections of U/G courses. You should ignore courses whose names end in "G" (e.g. `417G`) or sections including "Graduate" (e.g. `LEC 001 <br/> Graduate`). Otherwise your list of sections will include many duplicates.

There are other complications (some sections have multiple times or multiple instructors or no time periods) that your code will need to deal with. Real-life web pages have many such details that programs need to handle gracefully. On the other hand, you don't need to handle every possible situation that *could* happen. Just make sure your code is clean and easily modifiable. Realistically the web page format changes every few years.

4 Concerning the `CheckSchedule` class

Once the sections are read in, a program can do various analysis tasks. In this homework, you will do only one task: find lecture sections with competing times. Two lecture sections conflict if any time period for one overlaps (use `Period#overlap`) with any period of the other. The method can be implemented with a fairly straightforward quadratic algorithm. It returns a list of pairs (using a home-grown generic immutable `Pair` class).

5 What you need to do

You need to finish the implementation of three classes.

edu.uwm.cs351.util.XMLTokenizerUtil Helper methods for reading HTML.

edu.uwm.cs351.ImportUWMSchedule Read HTML from the web or from a file and find the sections being offered.

edu.uwm.cs351.CheckSchedule Code for checking a list of sections for overlap.

6 Files

The repository includes the following files:

src/TestXMLUtil.java Tests of the utility class including locked tests to check your knowledge of XML and the tokenizer.

src/TestCheckSchedule.java Tests of conflict checking.

src/edu/uwm/cs351/ImportUWMSchedule.java Run this main program to test your import code.

lib/fall2019.html Sample file to test your code, so you don't annoy UWM's web people by constantly accessing the schedule web page. Use Right-click "Open With" the "Text Editor" to view the context, instead of double-clicking, which will open the web page.

lib/expected-fall2019.txt Expected output of running `CheckSchedule` on the sample file.

There are no efficiency or random tests for this assignment. We aren't implementing ADTs and have no invariant tests.