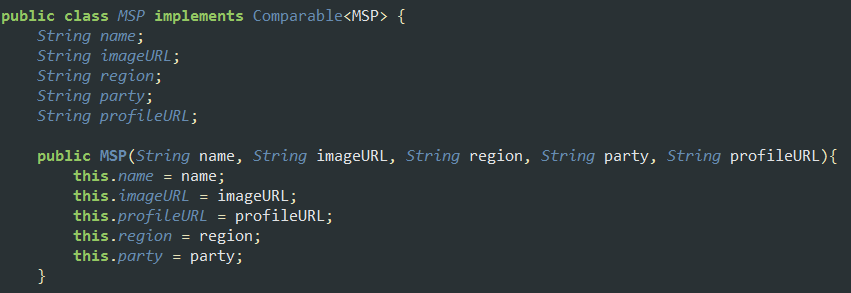
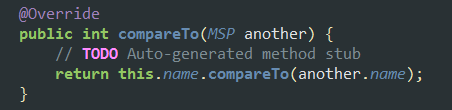
**Dev Diary - Manas Bajaj**

I was responsible for scraping the required data from the Scottish Parliament website and transfer it to a database. This data included information about all the Members of Scottish Parliament (MSPs). I used a HTML parsing library for Java called JSoup to achieve this. I started off by creating a class for MSPs :



This class implements Comparable and thus overrides the following compareTo() method for sorting a list of MSPs by their names:



Moving on, to scrape the above mentioned fields for each MSP, getMSPData() method was used which looked like this:

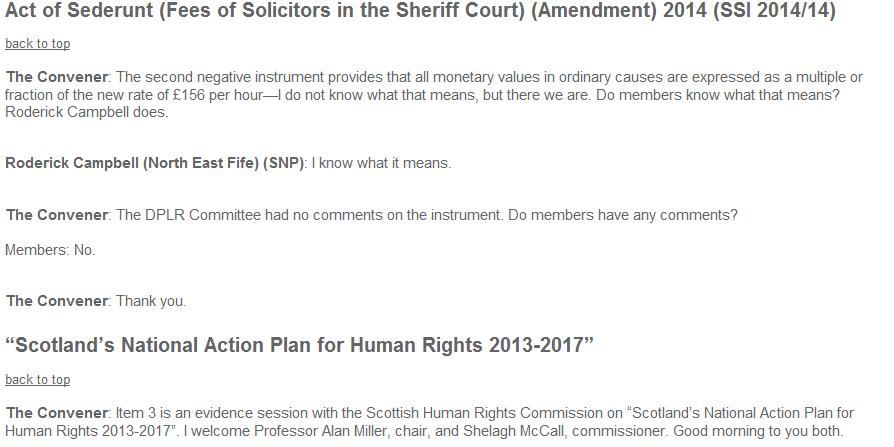


All this data was eventually added to an ArrayList of MSPs:

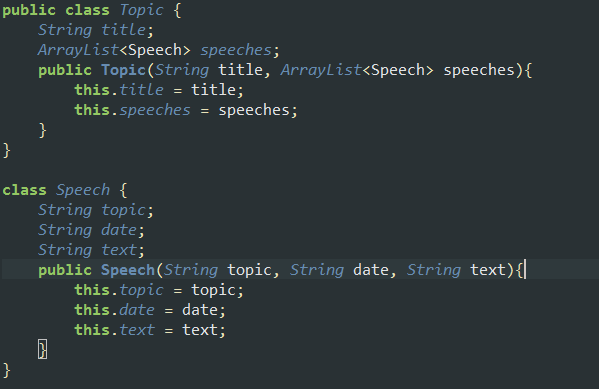


The most challenging part for me was to scrape all the speeches off the webpages that were specific to a particular topic and an MSP. As we were not provided data in a decent data exchange format such as XML or JSON, getting the speech titles seemed impossible at first. Every report has a date and a title and it consists of various speeches on different topics.

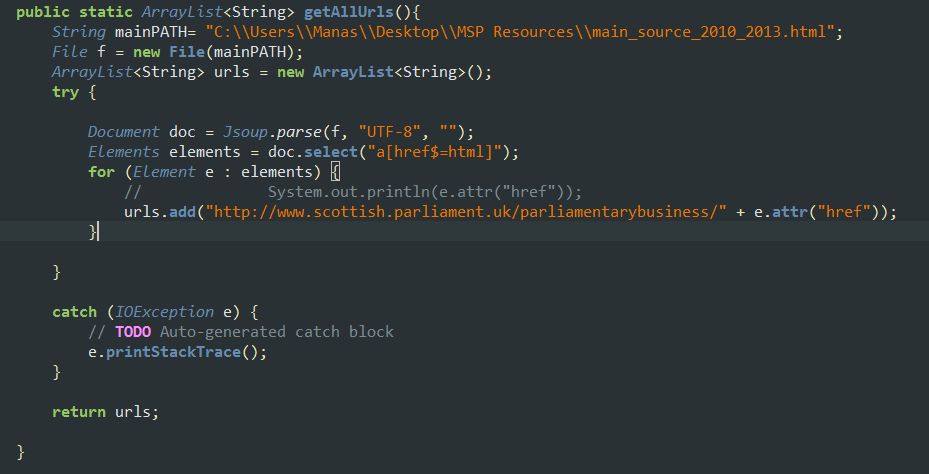
This is how a report looks like:



I started off by creating classes for Speech and Topic:

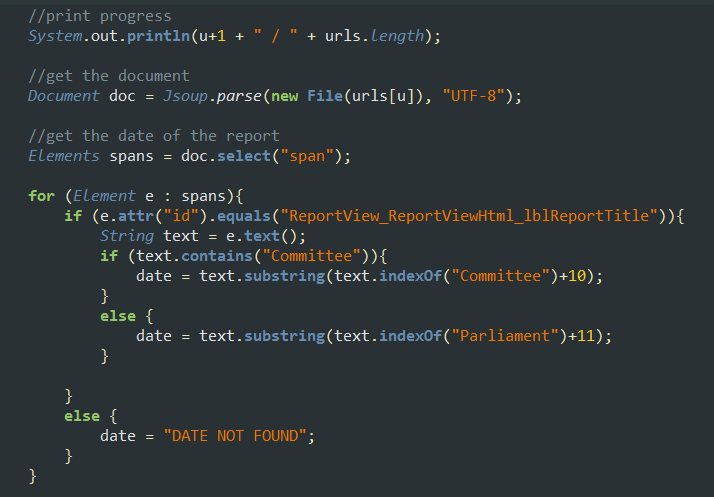


The basic plan was to map every MSP to an ArrayList of speeches they have given in the last 4 years (2010 – 2014). The first task was to get a list of URLs for each of these reports for the given time interval. This was achieved by the following method:



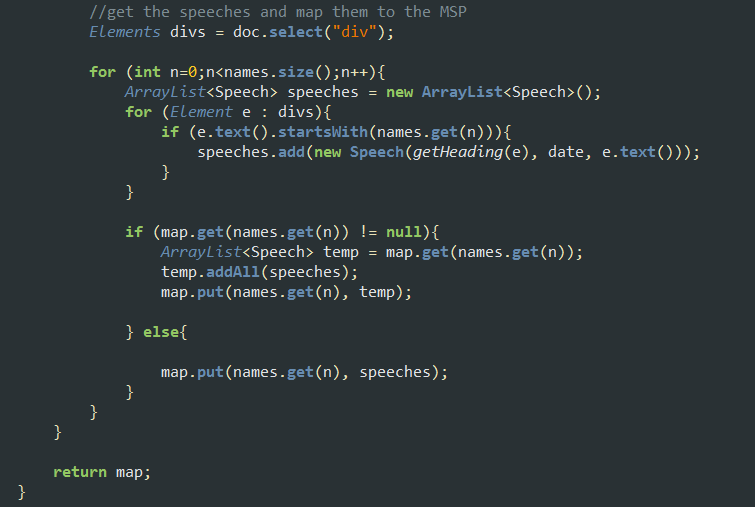
Now, we move onto the main method that did all the parsing:

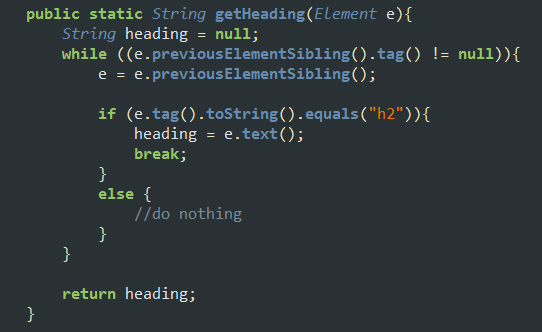
The above for loop basically iterates through the list of URLs that was fetched by getURLS() method previously, and gets all the required data for each speech.



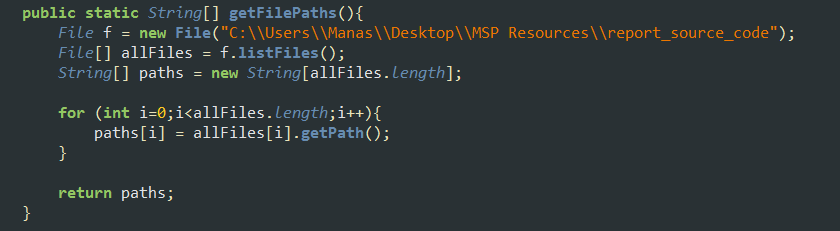
This piece of code prints the progress to the console as the files are being parsed and scrapes the date for each speech.

NOTE: The if else statement is required as there were two types of reports: Committee meetings and Parliament meetings.



And finally, the speeches are scraped and mapped to their MSPs. Also, note that the getHeading() method in the above code is used to get the topic for each speech. It does this by looping through all the previous elements until it finds a heading tag <h2>: 

NOTE: As Java kept throwing a SocketTimeOutException while parsing hundreds of URLs together, I had to first download the source code of all the reports and then parse them. So instead of iterating through the list of URLs, a list of file paths was used, which was fetched by the following method:



In the end, I managed to parse over 50,000 speeches altogether.

**Dev Diary – Chris Swart**

**//Intro song: MAAANASMAAANASMANASMANASMANASMANAS repeat//**

**I had to make an Objective/Subjective filter:**

**This model uses two knowledgebases one uses the other one. The WordNet framework is used within a research papers’ corpus that classifies word senses based whether they are objective or subjective.**

**It took me about 4 days just to understand the whole word sense concept and the logic behind it. Then luckily I could use nltk with the WordNet framework. Word sense disambiguation was a very complex procedure as well and in the end I am using a very simplistic method for deciding which word sense is used of a specified word sense.**

**Originally I wanted to use the distributional distance to wei**

**Teun Kokke, Team Temachmaga**

In our project “MoodSP” we worked with the Scottish parliament data.

We worked in a group of 5 people, of which one was taking care of the data collecting (scraping), two took care of analyzing the data, and two (including me) took care of the platform to display the data. Meaning our work was split into subparts, and we all worked with different programming languages including Java, Python, PHP and SQL.

My main part here was connecting these subparts, i.e. one person would send me the data that was scraped using Java, I caught this data using PHP and forward it to the people that analyzed it (using Python). Then they would return the results to me and I would insert it into the database using SQL after fetching the data. After having the data stored in our database we could visualize this on the web platform.

Image below: File created to access via the python file to insert politician speech data into the database



I also laid the framework and engine to the platform, and wrote the website in HTML, PHP, SQL, CSS MongoDB which were all required in order to come to the result.

Image below: a sample where I load data from the SQL database which can then be printed out in the webpage

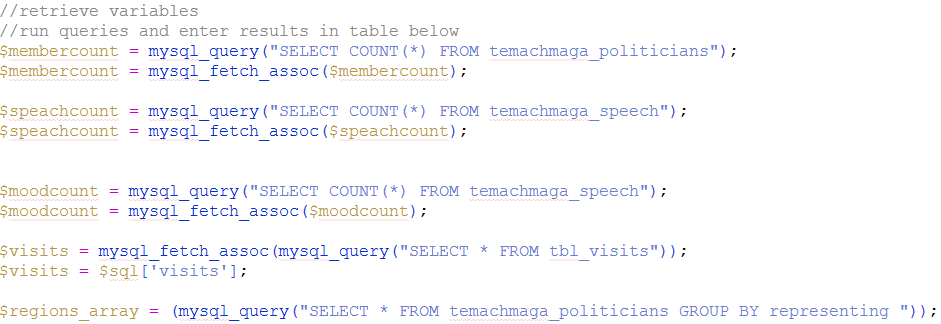


Image below: another sample for loading data from the Mongo Database



Image below: Script for the search function on the main page to search for politicians by name, party or region



Image below: example where the top-10-occuring verbs in each individual speech are displayed, along with their according amount of occurrence

