Overview

For this project, I needed to create a program to interact with the search API of DBLP, a provider of data on academic computer science publications. My program had to be able to search each of three information sources; publications, authors and venues. The user would input a set of command-line arguments to my program, including a selection of field, the query to be searched, and a directory of caches that could be utilised by my program if applicable. My program would then perform this search, parse the XML document that would be returned by the API and display the results in a pre-defined manner for the user. Additionally, my program would update the cache directory of previous searches, and draw XML files from it instead of using the API if the correct file was available.

Design & Implementation

For this program, I decided to make use of a SAX parser to aid me in parsing the retrieved API documents from DBLP. I began by setting the search URL, and developing the behaviour of the parser. To do this, I created 4 'handlers'. Each instance of a parser uses a handler to determine how it should behave upon parsing certain items, such as the beginning of a new tag, or elements within a tag. A default handler is defined for the SAX parser, and all of my handlers extend this class for ease of use. The changes I made to each handler are discussed below.

dblpAuthorHandler.java

This handler defines the 'first' search needed for authors. Upon beginning a new element (by encountering a tag), this handler checks the name of the element against 'info', 'author' and 'url'. As the SAX parser runs through the document linearly, I would use these checks to set a Boolean in my program to illustrate being 'in' an element. Specifically, both 'url' and 'info' are needed as the 'url' tag is used more than once, and the URL that I needed was always embedded within an 'info' element. I then overrode the 'characters' method, which takes the characters parsed from within an element (not including tags) and places them in a character array. Thus, if the 'author' Boolean is true, my program appends the characters in the element (the author's name) to a StringBuilder. I appended these values as apostrophes would escape the characters method, and appending would 'connect' the Strings split by an apostrophe, such as in names like "O'Hara". When both the 'info' and 'url' elements were active, a string for the URL is also built. Finally, upon ending these three elements, the data within the StringBuilder was input into ArrayLists for storage and later use. Also, the Booleans and StringBuilder were reset for the next element encounter.

dblpAuthorURLHandler.java

This handler defines the 'second' search needed for authors, namely the search for their co-authors. Helpfully, the XML in this case was conducive to a less complex solution as the data was contained within the tags as part of the 'attributes' field. Thus upon encountering a new tag of the correct name, I added the necessary index of 'attributes'. I also implemented a check for when there are no co-authors, as this would mean there was no 'coauthors' tag from which to parse data. Thus if the element had ended without finding any co-authors, a value of 0 was appended to the necessary list.

dblpPublicationHandler.java

Here, the use of 'startElement' is similar to that of the first author handler in that it sets a Boolean for the 'title' tag, then appends the title of the publication (again considering apostrophes) to a StringBuilder, and adds the built String to a list at the end of the element. For the number of authors, only an increment integer was needed, as further data about the authors was not required.

dblpVenueHandler.java

This handler only re-uses concepts used in the other handlers.

I then used two more general classes to bring the program together, as follows:

Search.java

This class is the main location for interacting with the search APIs. Taking the choice made by the user as a parameter for the class, the correct URL beginning is selected, and the final URL is constructed by appending the user's query and some additional parameters to the search defined in the brief. In case the search query has spaces in it, these are replaced by '%' signs in order for the URL to be valid. Then, the URL is matched against filenames in the cache directory to determine whether or not a call to the DBLP server is required. An input stream is then opened from the cache file if it exists, and from the URL if not. Then a parser is created, and a handler is assigned to it depending on the search type. The parser then parses the XML document, and the handler manages the data for later use.

W09Practical.java

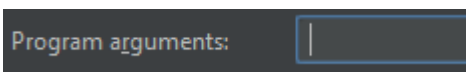
The main function of this class is to ensure that the arguments input by the user are valid before initiating a search using based on those arguments. Firstly, if no arguments are supplied, then a usage statement is printed for the user. If arguments are supplied, then the existence of all three double-hyphenated commands is confirmed and their index in the argument array is noted. If these commands are not present, then the user is informed and the program exits. The 'search' choice is then considered. As this choice has a limited number of valid values, these are exhaustively tested, and if one has not been selected the user is informed and the program exits. An Array Index exception is used in the case that this choice has not been supplied and 'search' was the last command used. In other cases where no choice has been supplied, the program assumes that the next term is the choice (which will often be another command) and the program will exit anyway. The query is tested in a very similar way; however any string will be accepted as there is no limit on the query. An exception is still used if the string is not present. Finally, the cache filepath is parsed and the program checks that it is a valid directory for caches. If not, the user is informed and the program exits. If all of these values are valid, then they are assigned to variables and used later on in the program.
Once all of the argument validation has been performed, the 'Search' class can be instanced and called. The data saved within lists in the relevant handler is then retrieved and printed in the defined format for the user.

Testing

The main testing process was that of the Autochecker. The Autochecker tests for, in order:

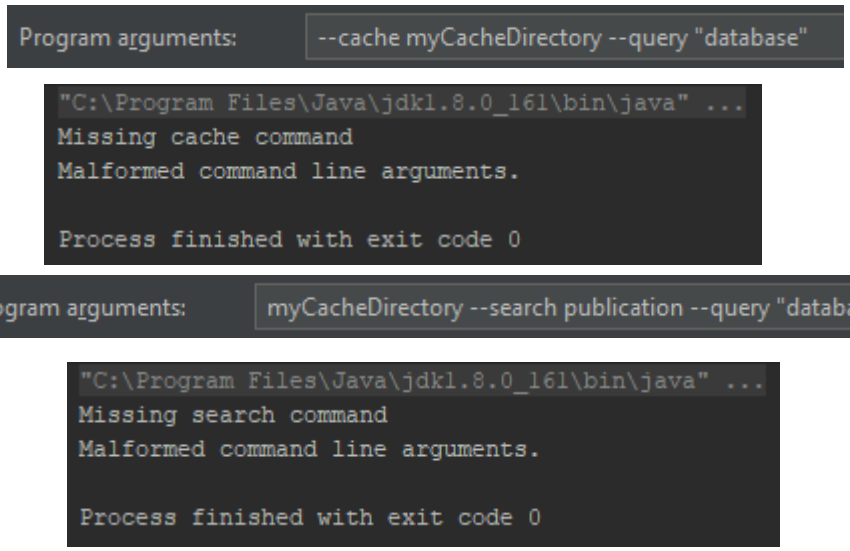
1. Correct responses from my program when arguments are invalid
2. A style check
3. 3 example searches for authors
4. 4 example searches for publications
5. 4 example searches for venues

My program passed all of these tests, with some warnings from the style checker which I chose to ignore.
These tests show that my program compiles correctly, responds with a helpful error statements when invalid arguments are given to the 'main' method, and outputs correct data when in 11 sample searches. I also tried a few test inputs where there were no arguments provided, and where the double-hyphenated commands were not present, the inputs and outputs to these tests are shown below.




```
"C:\Program Files\Java\jdk1.8.0_161\bin\java" ...
Usage: java W09Practical --search [author, publication, venue] --query <query> --cache <cache_filepath>

Process finished with exit code 0
```



Evaluation

Overall, I think that I have successfully created a program that satisfies the brief well. The autochecker has been satisfied, and my program seems to run well. I think that I have shown effective use of the SAX parser and its handlers, and I have managed to mesh this usage well with my prior knowledge. However, I think that my code can be made much cleaner, particularly with regards to the handling of command-line argument validation. I blame this partly on the restrictiveness of the autochecker, which requires that some of the 'commands' be checked before others, even if those commands appear later than other arguments that require validation. However, I also recognise that had I built my program with the autochecker's conditions in mind, I could have created a program which fulfilled it and was also elegant.

Conclusion

In conclusion, I am happy with how the basic version of this program has turned out. I think that I have shown a good understanding of the principles of working with APIs, I/O and data presentation. I am also glad to have learned an entirely new concept in the SAX parser, and been able to implement it myself without outside help. However, I feel that in implementing this 'new' concept, I have let my good practice suffer somewhat and I hope to improve this in my next practical.