CS1003 Week 9 Practical: Working with Online Data

This practical is worth **20%** of the overall coursework mark. It is due on **Thursday 12th April 2018 (week 9) at 21:00**. As for every practical, you should arrive in the lab having prepared in advance, by studying this specification and reviewing relevant course material.

Necessary Skills and Competencies

- Implementing robust command line argument parsing
- Implementing a file-based cache
- Choosing appropriate classes and methods from an XML API
- Working with an XML web-based API
- Identifying and dealing with possible error conditions
- Testing and debugging
- Writing clear, tidy, consistent and understandable code

Requirements

The task is to write a Java program to interact with the search API of DBLP. DBLP is a service which provides bibliographic information on major computer science publications. See http://dblp.org for more information.

DBLP provides a Search API which allows us to search for publications, authors, and venues. The API supports both XML and JSON as a communication format for the search results. In the core part of this practical we will use the XML format. See the following page for more information about the DBLP Search API.

http://dblp.uni-trier.de/faq/How+to+use+the+dblp+search+API.html

Your program will take a search query as input from the user on the command line, together with a choice of author/publication/venue. It will then execute a query using the DBLP Search API to receive a search result, encoded in XML. You will also need to maintain the search results in a *cache directory*. This cache directory will allow you to save the results of previously executed searches. Consequently, you will be able to avoid making an identical query to the server more than once.

Specifically, your tasks are the following.

- Accept the following 3 values on the command line interface.
  - `--search X`, where `X` is one of "author", "publication", or "venue"
  - `--query X`, where `X` is the query string
  - `--cache X`, where `X` is the path of your cache directory
- These 3 values may appear in any order on the command line.
- Produce appropriate error messages when the command line arguments are invalid.
- For all 3 kinds of search queries:
  (Consult the DBLP Search API page (linked above) for reference.)
  - Set the format to XML
  - Allow up to 40 hits to be returned
  - Set the maximum number of completion terms to 0
- For venue searches, only print the name of the venue for each search result. You should be able to make a single API call, and the response will contain all the necessary information.
- For publication searches, print the title of the publication together with the number of authors. This information is available in the XML response. Similar to venue searches, publication searches can be implemented by a single API call.

- For author searches, print the name of the author together with the total number of publications by this author and the total number of co-authors they have. In order to be able to calculate these numbers, you will need to make a second API call for each author. Make use of the “url” node inside the “hit” node for making the second API call. When you append the “.xml” file extension to this URL and load it from the server, you will see that it contains further information about the given author.
- Save the XML responses under the specified cache directory after every API call. Use the `URLEncoder.encode` method to convert the URL into a valid filename.
  - For example, the URL http://dblp.org/search/author/api?q=ozgur%20akgun (which can be used to search for an author with name “ozgur akgun”) is converted to the following String using `URLEncoder.encode`:
    http%3A%2F%2Fdblp.org%2Fsearch%2Fauthor%2Fapi%3Fq%3Dozgur%2520akgun
    This can now be used as the filename in the cache.
- Before making a request to the server, check the cache. If there is a saved response in the cache, use that instead of making a new request. (In a real application there would be strategies for *invalidating* the cache – that is deciding when to clear parts of the cache. But we will not worry about cache-invalidation for this practical.)

Running your program

The following is an example command which should search for the author Jack Cole.

```
java W09Practical --cache myCacheDirectory --search author --query "jack cole"
```

The output for this command should look like the following.

```
Jack Cole - 6 publications with 18 co-authors.
D. Jackson Coleman - 1 publications with 5 co-authors.
G. Cole Jackson - 1 publications with 4 co-authors.
```

And, the XML response is the following.

```
<?xml version="1.0" encoding="UTF-8"?>
<result>
<query id="20768">jack* cole*</query>
<status code="200">OK</status>
<time unit="msecs">0.20</time>
<completions total="2" computed="2" sent="0">
</completions>
<hits total="3" computed="3" sent="3" first="0">
<hit score="2" id="337075">
<info><author>Jack Cole</author><url>http://dblp.org/pid/14/891</url></info>
<url>URL#337075</url>
</hit>
<hit score="2" id="337309">
<info><author>D. Jackson Coleman</author><url>http://dblp.org/pid/129/0680</url></info>
<url>URL#337309</url>
</hit>
<hit score="2" id="774760">
<info><author>G. Cole Jackson</author><url>http://dblp.org/pid/182/3136</url></info>
<url>URL#774760</url>
</hit>
</hits>
</result>
```

Notice the lack of information about number of publications, and the number of co-authors. This data is available through the URLs listed in the XML response. By adding the .xml extension to them you can get them in a machine-readable format.

Try visiting these URLs in your browser to get a better idea of what the file contents look like.

Your program should deal gracefully with possible errors such as the cache directory being unavailable, or the files containing data in an unexpected format. The source code for your program should follow common style guidelines, including:

- formatting code neatly
- consistency in name formats for methods, fields, variables
- minimising code duplication
- avoiding long methods
- using comments and informative method/variable names to make the code clear to the reader

Running the Automated Checker

As usual, please use the automated checker to help test your attempt. The checker is invoked from the command line on the Linux Lab Machines in your W09-Practical folder:

```
stacscheck /cs/studres/CS1003/Practicals/W09/Tests
```

If all goes well, you should see something similar to below.

```
Testing CS1003 Week 9 Practical
- Looking for submission in a directory called 'src': found in current directory
* BUILD TEST - public/build : pass
* TEST - public/_malformed/1/test : pass
* TEST - public/_malformed/2/test : pass
* TEST - public/_malformed/3/test : pass
* TEST - public/_malformed/4/test : pass
* TEST - public/_malformed/5/test : pass
* INFO - public/_style/infoCheckStyle : pass
* TEST - public/author/jackcole/test : pass
* TEST - public/author/johnrsmith/test : pass
* TEST - public/author/johnsmith/test : pass
* TEST - public/publication/database/test : pass
* TEST - public/publication/databaset/test : pass
* TEST - public/publication/mariadb/test : pass
* TEST - public/publication/semi-structured/test : pass
* TEST - public/venue/distributeddb/test : pass
* TEST - public/venue/logic/test : pass
* TEST - public/venue/math/test : pass
* TEST - public/venue/parallelmath/test : pass
18 out of 18 tests passed
```

The Cache Directory

Your program should run correctly with an empty cache directory. In a scenario like this, it will start creating files as it makes web requests and it will reuse the cached files when necessary.

DBLP is a live website. The data they serve may change at any time. In order to provide you with consistent automated checking, we provide a cache directory that was created by running the stacscheck tests. This cache directory should contain all the necessary responses. By using this cache directory, you will be able to compare the operation of your program against the expected behaviour.

Do not forget to test with an empty cache directory as well! Your program will be tested with and without an existing cache directory.

For stacscheck testing, place the cache directory next to your src directory.

Testing

You are encouraged to create and document your own tests to test how your program deals gracefully with possible errors such as the input file being unavailable, or the file containing data in an unexpected format. To prove that you have tested the above scenarios, paste terminal output from each of your tests into your report in the Testing section. Be clear what each test demonstrates.

You might want to write simpler tests than those we provide in the Automated Checker, especially to test meaningful subsets of functionality before you finish the whole program.

Deliverables

Hand in via MMS to the Week 9 Practical slot, as single .zip file containing your entire W05-Practical directory which should contain:

- All your Java source files as specified above
- A brief report containing the usual sections for *Overview*, *Design & Implementation*, *Testing*, *Evaluation* (against requirements), *Conclusions* (your achievements, difficulties and solutions, and what you would have done given more time). You can use any software you like to write your report, but your submitted version must be in PDF format.
- Do not submit the contents of your cache directory!

Extensions

If you wish to experiment further, you could try any or all of the following:

- DBLP provides a JSON format for its search API as well. Instead of XML, ask the API to provide the data in JSON format and process this instead.
- Adapt your program to process data from a different online API, line Wikipedia or DuckDuckGo.
- Instead of the text format, output your result as an HTML document with clickable links.
- Make your program interactive in some way. For example, present a list of authors and let the user select one author and navigate to their publications, from there on to other authors, and so on.

You are free to implement your own extensions. However, clearly explain these in your report. If the extensions change the basic functionality, submit them in a separate "`extension`" folder inside your ZIP file.

If you use any additional libraries to implement your extensions, ensure you provide these with your submission with clear instructions to the marker on how to resolve these dependencies and run your program.

Marking

This submission will be tested by the School's automated checker `stacscheck` so make sure that your software can run the provided tests by running the following command on a lab machine:

```
stacscheck /cs/studres/CS1003/Practicals/W09/Tests
```

Passing or failing automated tests does not automatically affect your mark, but make sure `stacscheck` can run because it is an important tool and it helps the markers provide fairer and more detailed feedback.

Your submission will be marked using the standard mark descriptors in the School Student Handbook:

https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors

However, more specifically, for this practical:

- 1-6: Very little evidence of work, software does not compile or run, or crashes before doing any useful work. You should seek help from your tutor.
- 7-10: A decent attempt which gets part of the way toward a solution but has serious problems such as not compiling (at the low end) or lacking in robustness and maybe sometimes crashing during execution (at the high end).
- 11-13: A solution which is mostly correct and goes some way towards fulfilling basic requirements but has issues such as not always printing out the correct values.
- 14-15: A correct solution accompanied by a good report, but which can be improved in terms of code quality, for example: poor method/OO structure, lack of comments, or lack of reuse.
- 16-17: A very good, correct solution to the main problem containing clear and well-structured code achieving all required basic functionality, demonstrating very good design and code quality, good method and class decomposition, elegant implementation of methods with reuse where appropriate, and accompanied by a well-written report.
- 18-19: As above, but implementing at least one or more extensions, accompanied by an excellent report. It is key that you focus on getting a very good solution to the basic requirements prior to touching any of the extension activities. A weak solution with an attempt at extensions is still a weak solution.
- 20: As above, but with multiple extensions, outstanding code decomposition and design, and accompanied by an exceptional report.

Lateness

The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):

https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties

Good Academic Practice

The University policy on Good Academic Practice applies:

www.st-andrews.ac.uk/students/rules/academicpractice/