CS1003 – Programming with Data

Practical 2: Semi-structured data processing

Deadline: 26 February 2021, 9pm
Credits: 30% of coursework mark
MMS is the definitive source for deadline and credit details

You are expected to have read and understood all the information in this specification and any accompanying documents at least a week before the deadline. You must contact the lecturer regarding any queries well in advance of the deadline.

This practical involves retrieving semi structured data from an online data source and using basic text processing techniques to perform the specified task. You will need to decompose the problem into a number of methods as appropriate and classes if necessary. You will also need to test your solution carefully and write a report.

Synopsis

The task is to write a Java program to perform a word frequency analysis on data returned from OMDb's online JSON API. OMDb (The Open Movie Database) is a website that contains various kinds of information about movies. Its API allows searching for movies by movie title, by type, by year of release, etc. It also allows retrieving more detailed information about a particular movie when the movie's ID is provided.

Your program will run a search using this API to retrieve a list of movie IDs. Then, it will retrieve the "plot" text for each movie by making further requests (one request per movie, using the movie ID). Finally, it will print a histogram of the most frequent 10 words together with the number of occurrences. Keep reading for more details!

Web APIs and the OMDb API

Typically, web APIs like the one we use in this practical are structured to have a base address, followed by a number of parameters. Each parameter is written as a key value pair, separated by an equals sign (key=value). The parameters are separated using an ampersand sign (&), and the parameter list is separated from the base address using a question mark (?). Here is an example from the OMDb API:

```
https://www.omdbapi.com/?r=json&s=mary+poppins
```

The base address in this example is **`https://www.omdbapi.com`**, the first parameter is called **`r`** and its value is **`"json"`**, the second parameter is called **`s`** and its value is **`"mary poppins"`**. Notice how the space characters are replaced by a plus sign. This request will instruct the API to search for movies that have the words "mary poppins" in their title and return the result as a JSON document.

The full API specification for the OMDb API can be found on the OMDb website: https://www.omdbapi.com

In this practical we will make 2 kinds of requests:

- Search movies by title:
  - **`https://www.omdbapi.com/?apikey=XXX&r=json&s=TITLE`**
- Request by movie ID:
  - **`https://www.omdbapi.com/?apikey=XXX&r=json&i=ID`**

The underlined parts of these queries will change depending on the query you want to make within your program. The following section contains more information about the API Key component.

Getting an OMDb API Key

In order to limit the number of API queries, OMDb uses a key based system. This is very common with free APIs. There are 2 levels of API Keys, one paid and one free. The free tier gives us 1000 queries per day, which should be more than enough for this practical.

1. Go to http://www.omdbapi.com/apikey.aspx, select the free option, enter your email address.
2. You will receive an email from them. Click on the second link to activate your API key.
3. Click on the first link in the email to check if the API is working for you.
4. Here is another link to check with, just replace XXX with your API key and you should see the search results for "Mary Poppins"

   `https://www.omdbapi.com/?s=Mary+Poppins&apikey=XXX`

Use the API Key you receive during development and testing. Leave the key hard coded in your program so your marker can test your program too.

Reading from a local cache

The daily query limit of OMDb can be limiting during development. In order to allow you to test your program without downloading data from OBDb repeatedly, we provide a cache of results to all of the queries stacscheck will run. Each result is stored in a JSON file on StudRes. See the following directory:

https://studres.cs.st-andrews.ac.uk/CS1003/Practicals/P2/Tests/cachedir

You can open these files in your web browser and explore the structure of the JSON files.

The files in the local cache follow a strict naming convention. This is achieved by removing the apikey argument from the request, as follows:

- For searching movies by title:
  - **`https://www.omdbapi.com/?r=json&s=TITLE`**
- For requesting by movie ID:
  - **`https://www.omdbapi.com/?r=json&i=ID`**

Once you construct this request, use the following code snippet to encode the special characters in it and add a file extension. This will allow the string to be used as a valid filename.

```
String filepath = URLEncoder.encode(filepath, "UTF-8") + ".json"
```

Code organisation

- Create a directory called CS1003-P2 and a file called CS1003P2.java inside of this directory.
- You may create more classes for organising your code. These should be stored in the same directory.
- The code will be compiled by running **`javac *.java`** in the CS1003-P2 directory.

Task description

Your program should take a search term as an input in the form of a command line argument. Using this search term, construct a request string to search movies by title[1]. Each result contains an **`imdbID`** field, which is a unique identifier for that movie. Next, make one request per movie and extract the Plot field from the responses. Concatenate all plot values into a single string and count the number of occurrences of each word in this string. Print the most frequent 10 words together with the number of occurrences. The frequent words are printed in decreasing order of the word count, ties broken using dictionary ordering on the word. On each line print a tab character, followed by the count, followed by another tab character, and finally the word itself.

Before printing the histogram of most frequent words, we print the title and the plot of each movie as well. See the sample output section for the exact formatting expected.

In the following we make some simplifying assumptions about how to process this data and give some hints, so make sure to read the whole practical specification before starting to work on your solution.

Cleaning textual data

Textual data you extract from the "Plot" field needs to be cleaned before counting words in it. There are many ways of cleaning/normalising textual data, we chose a very straightforward definition in this practical.

- Remove all non-alphanumeric characters by replacing them with a space character.
  ```
  String text = …;
  text = text.replaceAll("[^a-zA-Z0-9]", " ");
  ```
- Convert the comment text to lowercase.
  ```
  text = text.toLowerCase()
  ```
- Split text on whitespace (space, tab and new line characters) to get an array of words.
  ```
  text = text.split("[ \t\n\r]")
  ```

Filtering stop words

Some words appear very frequently in common textual data, these are called stop words.

We will filter these words so we can focus on more interesting words! We provide a file called **`stopwords`**[2] on StudRes and as part of the stacscheck tests. This file has a word per line. In addition to filtering stop words, filter any single character words as well. The file path will be provided as a command line argument.

Command line arguments

Your program (CS1003P2) should take 3 command line arguments.

1. Source. Accepted values are "cache" or "online"
2. The path to the stop word file
3. The search term

[1] This request will return the first 10 results. More results can be retrieved by adding a page parameter, but we will only use the first page in this practical.

[2] Taken from http://gist.github.com/sebleier/554280

Hints on downloading

Once you construct a request as a String, create a URL object by passing the String to the URL's constructor. Call openConnection to create a URLConnection. A URLConnection is like a FileConnection, but with a URL instead. Once you open the connection, you can read from the input stream in the same way you normally read from standard input, for example using a Scanner.

```
URL url = new URL(urlString);
URLConnection connection = url.openConnection();
// You can now read from connection.getInputStream()
```

Hints on JSON parsing

We make two JSON parsing libraries available, you can choose which one to use: **`javax.json-1.0.jar`**, **`json-20190722.jar`**. The first one of these libraries provides a streaming API and the second provides an object mode API. We have used the first one during our exercises and had examples in our lectures and the second one is documented here: http://stleary.github.io/JSON-java/index.html.

The jar files can be found in the **`/cs/studres/CS1003/Practicals/P2/Tests`** directory. Both of these will be added to the classpath by stacscheck during automated testing, you may want to your add your chosen library to the classpath during development too.

Explore the structure of the JSON served by OMDb carefully. For example, the top level object returned as a result of the search request contains a Search field which is an array of objects. Each entry in this array contains data about an individual movie. Individual movie entries contain an **`imdbID`** field that is the unique identifier for the movie. Using this ID, you should construct a request for the individual movie. Individual movies contain a "Plot" field which is typically a short String.

Testing

This assignment makes use of the School's automated checker stacscheck. You should therefore ensure that your program can be tested using the auto-checker. It should help you see how well your program performs on the tests we have made public and will hopefully give you an insight into any issues prior to submission. The automated checking system is simple to run from the command line in your CS1003-P2 directory:

```
stacscheck /cs/studres/CS1003/Practicals/P2/Tests
```

You can run subsets of the test suite independently as well. Try the following commands:

```
stacscheck /cs/studres/CS1003/Practicals/P2/Tests/args
stacscheck /cs/studres/CS1003/Practicals/P2/Tests/queries/source_cache
stacscheck /cs/studres/CS1003/Practicals/P2/Tests/queries/source_online
```

The first one only checks command line argument handling behaviour. The second one reads from the StudRes cache and only the third one actually downloads data from OBDb API. In the unlikely event that you run out of your 1000 query per day daily allowance, only the **`source_online`** tests will fail.

Make sure to type the command exactly - occasionally copying and pasting from the PDF specification will not work correctly. If you are struggling to get it working, ask a demonstrator.
Use a lab computer for running stacscheck for consistency between the development environments.

The automated checking system will only check the basic operation of your program. It is up to you to provide evidence that you have thoroughly tested your program.

Sample output

```
> java CS1003P2 cache stopwords "star wars"
Movie 1
Title: Star Wars: Episode IV - A New Hope
Plot: Luke Skywalker joins forces with a Jedi Knight, a cocky pilot, a Wookiee and two droids
to save the galaxy from the Empire's world-destroying battle station, while also attempting to
rescue Princess Leia from the mysterious Darth Vader.

Movie 2
Title: Star Wars: Episode V - The Empire Strikes Back
Plot: After the Rebels are brutally overpowered by the Empire on the ice planet Hoth, Luke
Skywalker begins Jedi training with Yoda, while his friends are pursued by Darth Vader and a
bounty hunter named Boba Fett all over the galaxy.

Movie 3
Title: Star Wars: Episode VI - Return of the Jedi
Plot: After a daring mission to rescue Han Solo from Jabba the Hutt, the Rebels dispatch to
Endor to destroy the second Death Star. Meanwhile, Luke struggles to help Darth Vader back from
the dark side without falling into the Emperor's trap.

Movie 4
Title: Star Wars: Episode VII - The Force Awakens
Plot: As a new threat to the galaxy rises, Rey, a desert scavenger, and Finn, an ex-
stormtrooper, must join Han Solo and Chewbacca to search for the one hope of restoring peace.

Movie 5
Title: Star Wars: Episode I - The Phantom Menace
Plot: Two Jedi escape a hostile blockade to find allies and come across a young boy who may
bring balance to the Force, but the long dormant Sith resurface to claim their original glory.

Movie 6
Title: Star Wars: Episode III - Revenge of the Sith
Plot: Three years into the Clone Wars, the Jedi rescue Palpatine from Count Dooku. As Obi-Wan
pursues a new threat, Anakin acts as a double agent between the Jedi Council and Palpatine and
is lured into a sinister plan to rule the galaxy.

Movie 7
Title: Star Wars: Episode II - Attack of the Clones
Plot: Ten years after initially meeting, Anakin Skywalker shares a forbidden romance with Padmé
Amidala, while Obi-Wan Kenobi investigates an assassination attempt on the senator and
discovers a secret clone army crafted for the Jedi.

Movie 8
Title: Star Wars: Episode VIII - The Last Jedi
Plot: Rey develops her newly discovered abilities with the guidance of Luke Skywalker, who is
unsettled by the strength of her powers. Meanwhile, the Resistance prepares for battle with the
First Order.

Movie 9
Title: Rogue One: A Star Wars Story
Plot: The daughter of an Imperial scientist joins the Rebel Alliance in a risky move to steal
the plans for the Death Star.

Movie 10
Title: Star Wars: Episode IX - The Rise of Skywalker
Plot: The surviving members of the resistance face the First Order once again, and the
legendary conflict between the Jedi and the Sith reaches its peak bringing the Skywalker saga
to its end.
```

```
Most frequent words in the plot fields:

       7      jedi
       5      skywalker
       4      galaxy
       4      luke
       3      darth
       3      rescue
       3      vader
       2      anakin
       2      battle
       2      clone
```

Submission

Your report must be structured as follows (a report template is provided on StudRes):

- Overview: Give a short overview of the practical: what were you asked to do, and what did you achieve? Clearly list which parts you have completed, and to what extent.
- Design: Describe the design of your program. Justify the decisions you made. In particular, describe the classes you chose, the methods they contain, a brief explanation of why you designed your solution in the way that you did, and any interesting features of your Java implementation.
- Testing: Describe how you tested your program. In particular, describe how you designed different tests. Your report should include the output from a number of test runs to demonstrate that your program satisfies the specification. Please note that simply reporting the result of stacscheck is not enough; you should do further testing and explain in the report how you convinced yourself that your program works correctly.
- Evaluation: Evaluate the success of your program against what you were asked to do.
- Conclusion: Conclude by summarising what you achieved, what you found difficult, and what you would like to do given more time.

Don't forget to add a header including your matriculation number, the name of your tutor and the date.

Upload

Package up your CS1003-P2 folder and a PDF copy of your report into a zip file as in P1, and submit it using MMS, in the slot for Practical P2. After doing this, it is important to verify that you have uploaded your submission correctly by downloading it from MMS. You can then run stacscheck directly on your zip file to make sure that your code still passes stacscheck. For example, if your file is called 190000000-P2.zip, save it to your Downloads directory and run:

```
cd ~/Downloads
stacscheck --archive 190000000-P2.zip /cs/studres/CS1003/Practicals/P2/Tests
```

We covered using **scp** for copying files from your local computer to the lab computers and back. Please seek help immediately if you are not comfortable with doing this.

Marking Rubric

1-6	Very little evidence of work, software which does not compile or run, or crashes before doing any useful work. You should seek help from your tutor immediately.
7-10	An acceptable attempt to complete the main task with serious problems such as not compiling or crashing often during execution.
11-13	A competent attempt to complete the main task. Serious weaknesses such as using wrong data types, poor code design, weak testing, or a weak report riddled with mistakes.
14-16	A good attempt to complete the main task together with good code design, testing and a report.
17-18	Evidence of an excellent submission with no serious defects, good testing, accompanied by an excellent report.
19-20	An exceptional submission. A correct implementation of the main task, extensive testing, accompanied by an excellent report. In addition, it goes beyond the basic specification in a way that demonstrates use of concepts covered in class and other concepts discovered through self-learning.

See also the standard mark descriptors in the School Student Handbook:
http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors

Lateness penalty

The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof):
http://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:
https://www.st-andrews.ac.uk/students/rules/academicpractice

Going Further

Change **`r=json`** to **`r=xml`** to retrieve the results as XML instead of JSON!