COMP8714 : Advanced Software Development Practices
Programming Assignment

YouTube Trender

Submission Date

5pm, Thursday, Week 10

Important Information

You do **NOT** need a YouTube/Google account to complete and achieve a high mark for this assignment. You will only be required to read YouTube video data from a file.

You can do this assignment in pairs. You must let me know who you are pairing with and only **one submission is required**.

The assignment is worth 40% of the overall topic mark. The 40% is broken down into three phases as discussed below.

Specification

The assignment is to design, implement, test, and document a command line application to analyse the results of a YouTube search using the YouTube Data API: e.g. "Trending Topics" of YouTube videos.

YouTube is a global video-sharing website headquartered in San Bruno, California, United States. The site allows users to upload, view, rate, share, and comment on videos.[1] Available content includes video clips, TV clips, music videos, movie trailers, and other content such as video blogging, short original videos, and educational videos. User can also add a title and description to the videos and by examining the contents of and detecting which words appear frequently across titles and descriptions we can detect "trending topics." See below for more information on YouTube and the YouTube Data API that generate the list of videos used in this assignment:

YouTube: https://www.youtube.com/yt/about/en-GB/
Wikipedia on Youtube: https://en.wikipedia.org/wiki/YouTube
YouTube Data API: https://developers.google.com/youtube/v3/getting-started
YouTube Data API for listing videos: https://developers.google.com/youtube/v3/docs/videos/list

It is your job to develop a command line application that can detect the YouTube Trending Topics.

[1] https://en.wikipedia.org/wiki/YouTube

Assignment Phases Overview

The assignment is broken down into three phases:

1. Parse a YouTube video data string in into a YouTube video object **[40%]**
2. Sort Video objects by different features (e.g. title, channel title, views, date, etc.) **[20%]**
3. Index the list of videos for word usage, aka "Trending Topics" **[20%]**

You must also submit a document that describes what you have done, how it relates to the theory sections of the topic and any extra features that you have built for your application. You will also include the final revision number for your code in the repository. This document will be submitted via Canvas.

4. Application description document **[20%]**

At each phase you should extend at the command line interface to display the results of your improvement and allow the user to interact with the underlying data structures and algorithms.

Each phase will have marks allocated as follows:

- 70% Correctness
- 15% In Code Documentation
- 15% Demonstration and Testing Code

Your submission will be the contents of your repository directory for the assignment and a PDF of your Application Description document submitted via Canvas

In Code Documentation

Your code should be documented. That is, it should contain Javadoc comments that describe methods, classes, instance and class variables, etc., which you have written.

Use Javadoc (see http://docs.oracle.com/javase/8/docs/technotes/guides/javadoc/index.html).

Phase 1 Detailed Description [up to 40%]

The goal of phase 1 is to parse a YouTube data string in into a Video object. You will need to at least:

- Create a Java class called `YouTubeVideo` to represent the contents of a video (minimum of):
  - channelId, channelTitle, publishedAt (date), title, description, viewCount
- Create a Java class called `YouTubeDataParser` that can take a file name and extract a list of videos from the given JSON file
- Create a Java class called `YouTubeDataParserException` that extends `Exception` to indicate a parsing error. The `YouTubeDataParser` should throw this exception when it encounters an error during the parsing of a JSON file (e.g. the file not found, malformed JSON file)
- Document the code using Javadoc.

Phase 2 Detailed Description [up to 20%]

Phase 2 is to sort Videos by different features of your newly created `YouTubeVideo` object (e.g. Channel, Date, etc.). You will need to at least be able to sort videos based on

- the channel title,
- the published at date,
- the number of views,
- something to do with the description of the video (e.g. its length).

You can use the static methods of the utility class `Collections` to sort your videos. Add Javadoc to your code to document the added functionality.

Phase 3 Detailed Description [up to 20%]

The task of phase 3 is to implement a trending topic analyser for the list of videos. This amounts to indexing the word usage across the **title** and **description** of the videos. A simple way to do this is to count the number of times a single word appears in the list of videos. Figure 1 lists a pseudo code example for indexing the words used in videos.

```
SomeCollection words (this could be a set, list, map, etc)

for each video
   for each word in video(title and description)

       if (word is in words)
           increment count for word in words
       else
           add word to words and set count to 1
       end

       associate video with word

   end
end
```

Figure 1. Pseudo code for indexing the words in videos

NOTE: in this example the "word" is not cleaned in anyway (e.g. all punctuation symbols are included in the "word", "twinkle," is NOT the same as "twinkle") and it counts multiple occurrences of a word in a title and description. **Your indexer should meet this specification**.

Once indexed, you need to be able to

- quickly find a word
- find the count associated with a word,
- find all the videos that use that word
- find the word that is used the most
- create a list of words sorted by their counts

You will need to use classes from the Collections framework such as `Lists`, `Maps` and `Sets` to implement this algorithm. You may need to create a new class to store the information about a word, e.g. the word itself, the count, the videos associated with it.

Testing your Application

It is expected that you will thoroughly **test** your application by developing code that **demonstrates** how the classes you have created work. The tests will test the classes you build to represent the videos and other pieces of data (e.g. words in the index, comparators, lists) and the functionality associated with them (sorting, find most used word, getting the title of a video).

You are provided with a set of JSON files as part of the skeleton code in the "data" directory.

Your assignment will be partially assessed as to how well you demonstrate that it works.

You have been supplied with several .json files as part of the skeleton code that contain different examples of the type of data returned from YouTube Data API query. The table below describes each file and the number of videos contained within them

File name	Description	No. Videos
youtubedata.json	A single video items in video query (mock data)	1
youtubedata_15_50.json	50 video items from a query from category 15	50
youtubedata_loremipsum.json	10 mock video items with useful descriptions and titles	10
youtudedata_indextest.json	A single video item for testing indexing	1
youtubedata_singleitem.json	A single video item not in a list	1
youtubedata_malformed.json	A single video items in video query (mock data), but with an error	1

The file "youtubedata_malformed.json" has one video item in a list but contains an error. This file is useful for test the detection of errors in your parser. For example, it may throw an exception indicating a problem with the provided .json file.

The file "youtubedata_singleitem.json" contains that data for a single video but not in a list. This is suitable for testing a class that represent video item data. The data is a JSONObject (enclosed between "{" and "}").

The "youtudedata_indextest.json" contains a fake title and description for testing your indexing. There should be 1 "ONE", 2 "TWO"'s, 3 "THREE"'s, four "FOUR"'s, and five "FIVE"'s.

Along with the .json files are corresponding "_info.txt" files available on FLO. These files contain information about the video contained in the .json file and can be used to provide info for your

testing (e.g. the number of videos). That is, given the .json file you should expect your program to be able to provide the information contained in the "_info.txt" file.

The sorting and word counts in the "_info.txt" files closely follow the algorithm in the assignment specification. **Your implementation must conform that what has been specified.**

Note that:

- The word count includes repeats of the word in the text of one message
    - e.g. text:"aaa aaa" => count for "aaa" is 2

I will inspect the code for correctness, but unit tests will give me more insight to how your code works and make it **easier to mark**.

Application description document [up to 20%]

You should write a brief document to describe the features of your application. It is to be submitted through Canvas as a PDF file. You can include text from this document and but it is not a copy of your Javadoc.

This document can be thought of as an informal user guide and can include screen shots of the code and its output. You will describe the demonstration and test code. It should also describe what you have done and how it relates to the theory sections of the topic and any extra features that you have built for your application. Further, you will include a reflective section on what difficulties, or not, you faced during the completion of the assignment and how you chose the data structures you used in solving the problem. You will also include in the document the final revision number for your code in the repository.

If it is not submitted, then you will receive 0 marks for this component.

Minimal Class Diagram Example

The UML diagrams[2] in Figure 2 show a minimal set of classes that you may implement to complete the assignment. They are by no means prescriptive and are shown as guide to what you might need to do to complete the assignment.

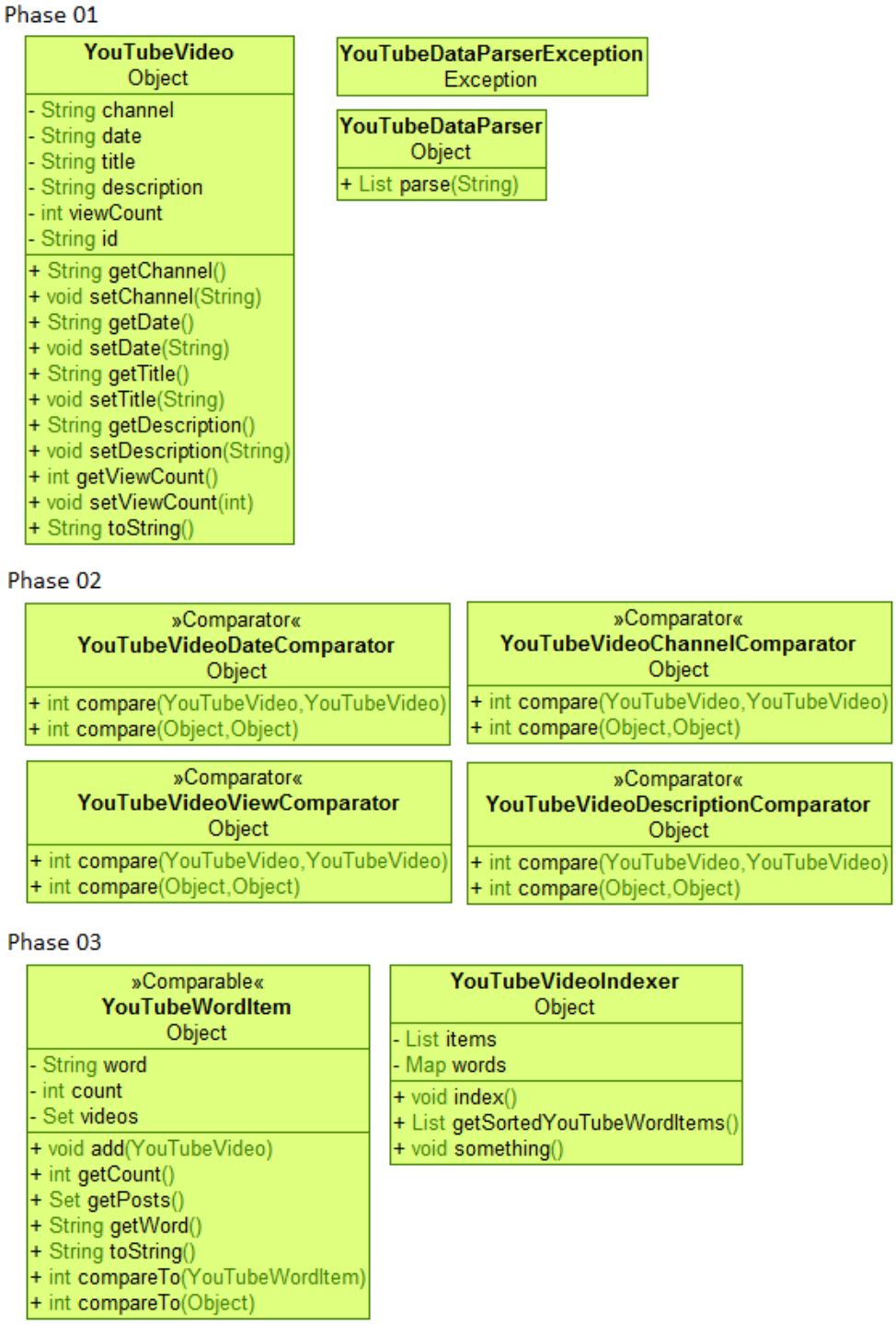

Figure 2. Minimal Class Diagram Example

[2] Class diagrams generated using JUG: http://sourceforge.net/projects/jug/

HOW TO

How To: Get Started

A skeleton IntelliJ project is provided to get you started. This is available in the repository for the topic. Once downloaded, open the project in IntelliJ and browse through the provided files. You are provided with the sample JSON files and the associated _info.txt files (in the data dir), the JSONP library (.jar file in the lib dir), and a single class called `YouTubeTrender`. This is your starting point and contains a sample test method you can use as a guide.

How To: Get YouTube Data

This is an FYI, you do NOT need to get your own YouTube data

For this assignment, we will use the result of a YouTube Data API query over videos as our YouTube video data string. The YouTube data API requires OAuth authentication for any queries over the videos and as such requires a developer to have a Google account and the necessary OAuth token.

For this assignment **you do not need a Google account** or be able to query the YouTube servers to complete the assignment to a high level (e.g. get an HD). You are provided with sample query results.

The data provided to you comes from a query of videos from the most popular videos from the category of 15 which is "Pets & Animals" with 50 videos retrieved, for example

```
https://www.googleapis.com/youtube/v3/videos?part=snippet%2Cstatistics&chart=mostPopular&maxResults=50&regionCode=AU&videoCategoryId=15&key={YOUR_API_KEY}
```

Entering this into a web browser will result in an error as it is not OAuth authenticated. However, you can use the YouTube Data "try it" facility to test it out.

https://developers.google.com/apis-explorer/#p/youtube/v3/youtube.videos.list

Properly executed, the data returned is in a format friendly to our application which can easily inspect and parse the return of a search and can be saved to a file. This format is known as the JSON format. Details will be given below on how to inspect and parse a JSON formatted data.

To make development and testing easier, you have been provided with some .json files that contain the results of queries. This means you can test your application without a network connection or a Google Account.

NOTE: you do not need to get data from the YouTube servers to complete the assignment or achieve a HD

How To: Parse JSON data

To inspect or parse the data contained within the result of a YouTube data query we need to be able to parse the JSON data object. JSON[3], or Java Script Object Notation, is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. JSON is built on two structures

- A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
    - Denoted by "{" and "}", e.g. {"name":"value","name2":"value2"}
- An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.
    - Denoted by "[" and "]", e.g. [{"name":"value","name2":"value2"}, {"name":"value","name2":"value2"}]

We could attempt to parse the JSON format ourselves, but there exist numerous APIs for doing this (there are 20 listed on the JSON website just for Java). You have been provided with the JSONP library file (javax.json-1.0.4.jar) as part of your skeleton code (in the lib directory) for parsing JSON data. We will be using the object model part of the library and documentation for JSONP can be found at:

https://javaee.github.io/jsonp/

Javadoc API documentation for this can be found at:

https://static.javadoc.io/javax.json/javax.json-api/1.1.2/index.html?overview-summary.html

There are a number of interfaces and classes in the API but you will be mainly interested in

- **JsonObject** (extends java.util.Map)
    - string and value pairs contained between "{" and "}"
- **JsonArray** (extends java.util.List)
    - order list of values between "[" and "]"

as these are the main structures contained in the Reddit Things Listing.

The JSONP API has a number of methods for returning the correct type, e.g `snippet.get("title")` returns the id field as String, but if the value is empty this can cause an exception. This means that to use the API correctly you need to sometimes check what the type of the contents is first, that is

```
JsonValue.ValueType value = snippet.get("title");
if (value.getValueType() == JsonValue.ValueType.NULL)
{
    // deal with null value
}
```

[3] http://www.json.org/

So, what does a YouTube Data query result in JSON format look like? Figure 4 is an example that has been truncated to one search result for clarity. Figure 3 lists a portion of basic Java code to extract the videos from data. That fields of the video are listed here: https://developers.google.com/youtube/v3/docs/videos

```
public static void test1() throws FileNotFoundException {

    System.out.println("Performing Test 1");
    String filename = "data/youtubedata_15_50.json";
    int expectedSize = 50;

    System.out.println("Testing the file: " + filename);
    System.out.println("Expecting size of: " + expectedSize);

    // Read data
    JsonReader jsonReader = Json.createReader(new FileInputStream(filename));
    JsonObject jobj = jsonReader.readObject();

    // read the values of the item field
    JsonArray items = jobj.getJsonArray("items");

    System.out.println("Size of input: " + items.size());
    System.out.println("Sucess: " + (expectedSize == items.size()));

    for (int i = 0; i < items.size(); i++) {
        JsonObject video = items.getJsonObject(i);
        JsonObject snippet = video.getJsonObject("snippet");
        System.out.println("Video " + i + ": " + snippet.getString("title"));
    }

}
```

Figure 3. Basic code to read video from a YouTube query

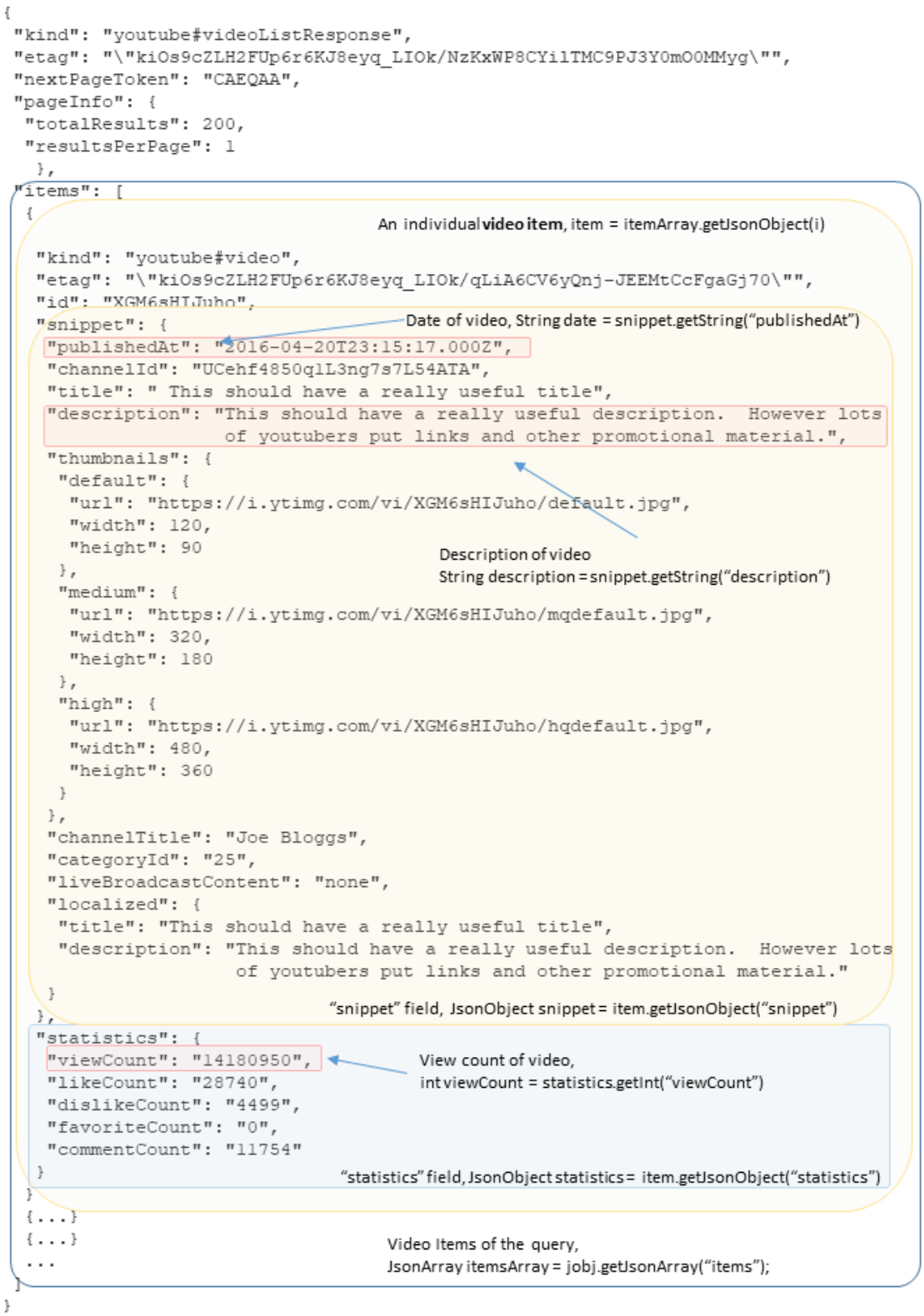

Figure 4. Structure of a YouTube Data JSON Video Search Query Result. "..." indicates truncated data

