Practical 3 & 4 – YouTube Trender

Submission Due Date

4pm Thursday the 17th of October 2024

Important Information

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

You will do this practical in pairs. You must let me know whom you are pairing with so that you can access a single code repository and so that you can be assessed, because *only one submission is required*.

The practical is worth 30% of the overall topic mark. The 30% is broken down into three phases as discussed below.

The practical sessions in weeks 9, 10 and 11 will be for assistance on the practical.

Specification

The practical is to design, implement, test, and document an 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[1] on videos. 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 practical:

Useful Links

YouTube

Wikipedia on Facebook

YouTube Data API

YouTube Data API for listing videos

[1] Wikipedia

Practical Phases Overview

The practical 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 and any extra features that you have built for your application. This document will be submitted via FLO

4. Application description document [10%]

At each phase you will need to extend the graphical user 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:

- 60% Correctness
- 20% Documentation and Testing
- 20% Graphical User Interface Enhancements

At the end of each phase you should make a copy of your practical for backup and commit it to your repository with the comment "Phase X" where X is the phase number 1, 2, or 3.

- Correct use of the repository [10%]

Your submission will be the contents of your repository on the due date. Code submitted after this date will be penalised at 5% per day late.

Unit Testing

Create unit tests to verify that your code functions correctly for any input that you deem to be necessary for thorough and comprehensive testing. You will be provided with data files and the expected output that can be used in your unit tests.

Documentation

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

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):
  - channel, date, title, description, view count
- Create a Java class called YouTubeDataParser that can take a JSONObject or a String and extract a list of videos from it
- Create a Java class called YouTubeDataParserException that extends Exception to indicate a parsing error
- Connect the classes created to the supplied graphical user interface to load and display videos, as shown in Figure 1
- Create unit tests for phase 1

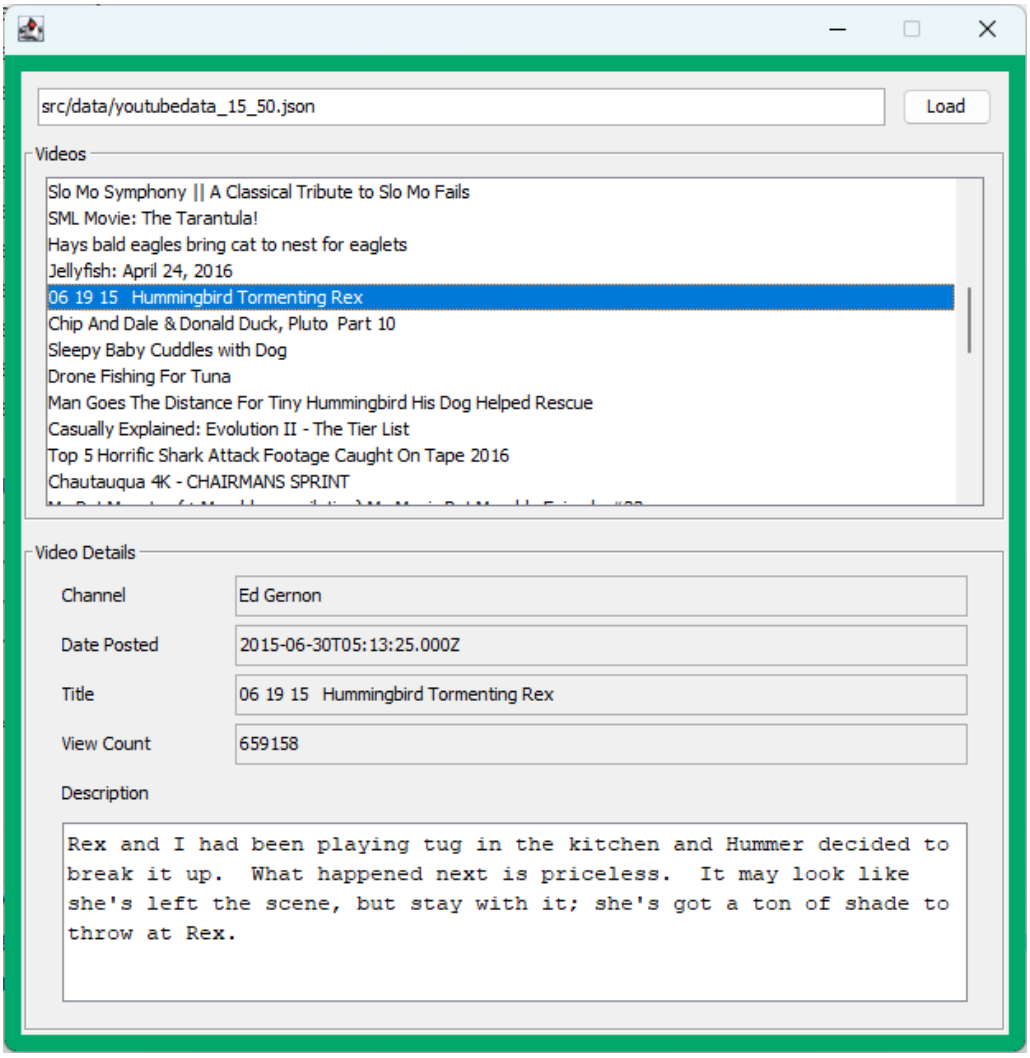


Figure 1: Example of the graphical user interface for Phase 1

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.

Create further unit tests using the supplied data files to verify that your sorting works.

You will need to update the graphical user interface to reflect the changes you have made (i.e. being able to sort videos). Figure 2 is a very basic example of an updated graphical user interface.

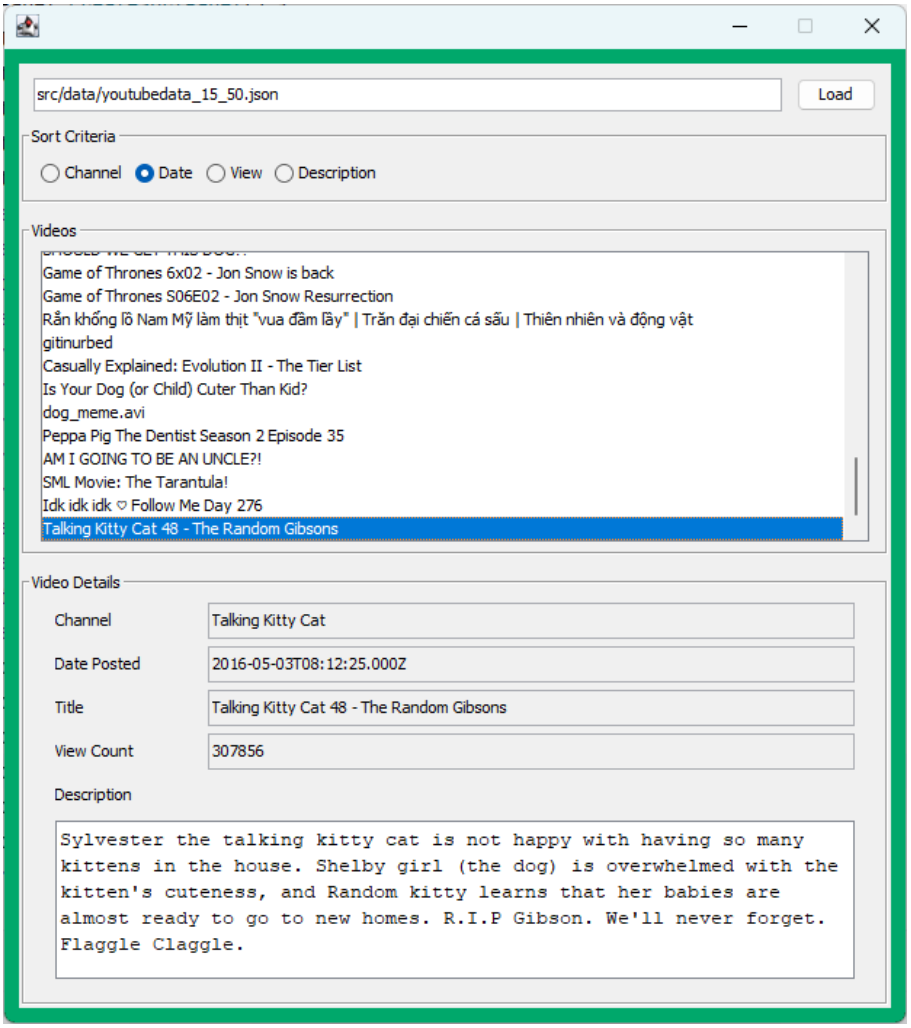


Figure 2: Example of a basic graphical user interface for Phase 2

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. Algorithm 1 lists a pseudo code example for indexing the words used in videos.

```
SomeCollection words (this could be a set, list, map, etc)
foreach post do
    foreach word in video(title and description) do
        if (word is in words) then
            increment count for word in words
        else
            add word to words and set count to 1
        end
        associate video with word
    end
end
```

Algorithm 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 post. Your indexer should meet this specification. If you improve upon it you should still make available this implementation for testing.

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.

Create further unit tests using the supplied data files to verify that your trending topic algorithm works. The supplied data assumes the above algorithm is implemented as specified.

You will need to update the graphical user interface to allow the user to index the videos and display the results (e.g. Figure 3)

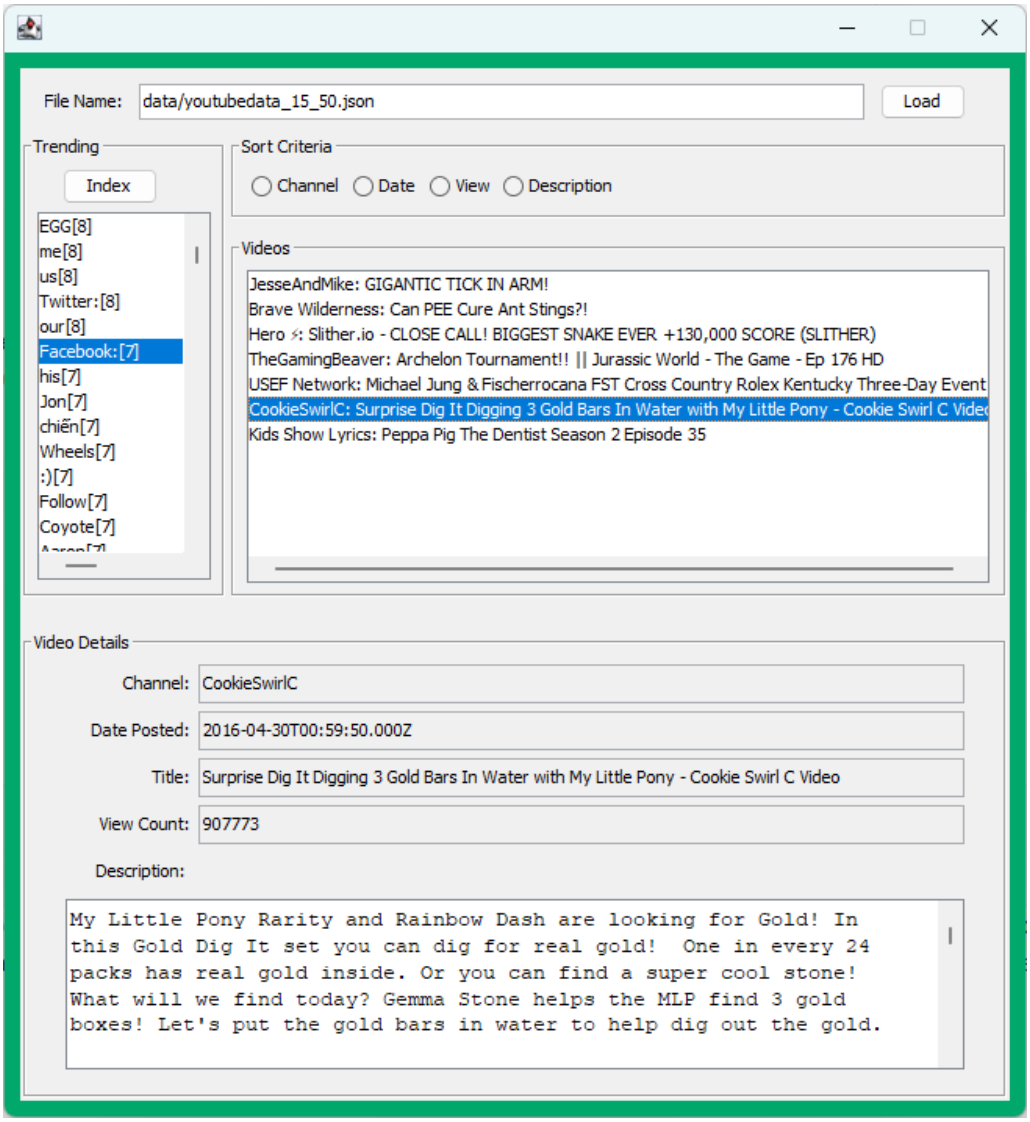


Figure 3: Example of a basic graphical user interface for Phase 3

Testing your Application

It is expected that you will thoroughly test your application using unit testing. The unit tests will not test your graphical user interface, but 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). This will make your underlying classes somewhat separate from your user interface and build a more robust and reusable application (e.g. you could use the classes with another user interface such as a mobile device).

You will be provided with a set of data files and the expected output that you can use in your unit tests.

Your practical will be partially assessed as to whether or not it passes the unit tests that utilise the supplied data files.

Application description document [up to 10%]

You should write a brief document to describe the features of your application. You can include text from this document, but most importantly you need to describe any additional features that you have implemented that might not be obvious. For example, include information on any interesting algorithms you implemented to index the videos.

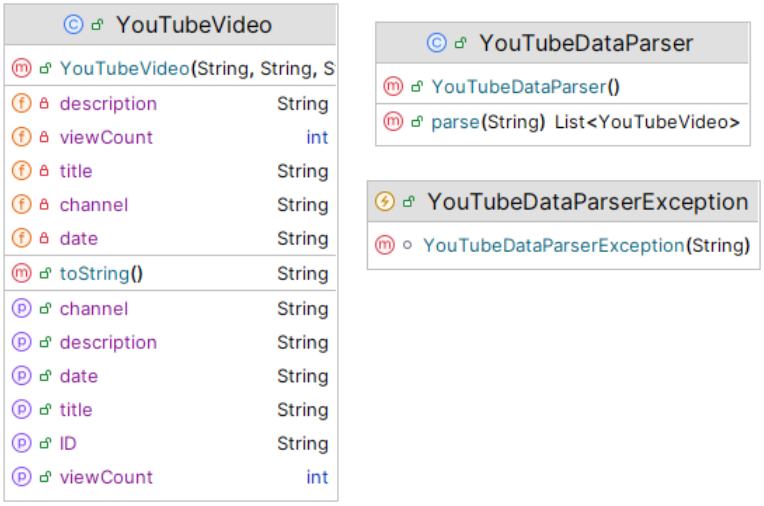
This document can be thought of as an informal user guide.

If this document is not submitted, then you will receive 0 marks for this component and it will be assumed that you have only implemented the basics, that is, it will be difficult to achieve a HD without this document.

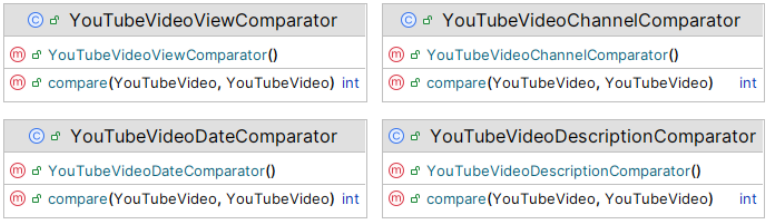
Minimal Class Diagram Example

The UML diagrams in Figures 4a, 4b and 4c show a minimal set of classes that you may implement to complete the practical.

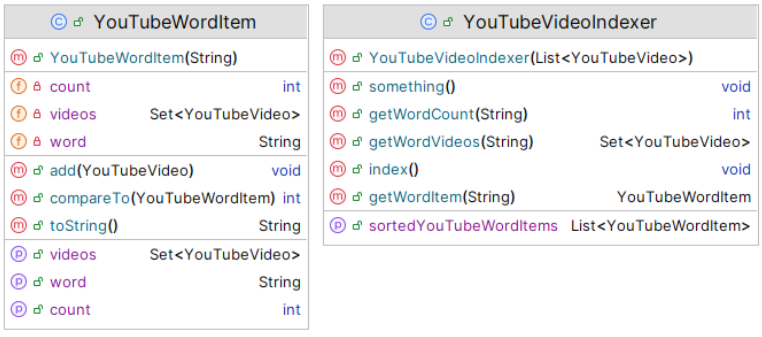
They are by no means prescriptive and are shown as guide to what you might need to do to complete the practical.



(a) Phase 1



(b) Phase 2



(c) Phase 3

Figure 4: Minimal Class Diagram Example

HOW TO

How To: Get YouTube Data

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

For this practical, 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 practical you do not need a Google account or be able to query the YouTube servers to complete the practical 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. Link below

try it

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 post.

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 practical or achieve a HD

So what does a YouTube Data query result in JSON format look like? Figure 5 is an example that has been truncated to one search result for clarity. Figure 6 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

Figure 5: Structure of a YouTube Data Json video search query result. "…" indicates truncated data

```
JsonReader jsonReader;

//read data
jsonReader = Json.createReader(new FileInputStream(fileName));

// create the top-level json object
JsonObject jobj = jsonReader.readObject();


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

System.out.println("number of videos: " + items.size());

for (int i = 0; i < items.size(); i++) {
    JsonObject video = items.getJsonObject(i);
    // get the snippet and print the title
    JsonObject snippet = video.getJsonObject("snippet");
    String title = snippet.getString("title");
    System.out.println("title: " + title);
}
```

Figure 6: Basic code to read posts from a YouTube query

How To: Get a High Distinction

Completing all the tasks outlined in the phases (including passing the unit tests and the minimal graphical user interface) will probably result in a mark in the range CR-DN. To achieve a HD you must undertake the "required core work for the topic at a high level and considerable additional work in wider areas relevant to the topic. " Thus, you need to complete the minimum and also extend beyond that. Areas where this can be achieved in this practical are

- comprehensive unit testing
- better graphical user interface design
- more extensive YouTubeVideo class design (e.g. incorporate other statistics)
- sorting by more features (e.g. on other statistics)
- searching your YouTubeVideo list
- different ways to index the YouTubeVideo (e.g. phrases (¿ 1 word))
- better cleaning of the words used in the index (e.g. make "twinkle" and "twinkle," the same)

These additional features need to be outlined in your Application Description Document so that they can be assessed.

Your practical must still pass the required unit tests.