CS126 Design of Information Structures

Coursework Specification

Term 2, 2022/23

Contents

1 Introduction

A developer of a new Java application has asked for your help in storing a large amount of *film-data*. The application is called **Warwick+**. The purpose of this application is to present data and fun facts about films as well as the cast and crew who worked on these films.

However, for the application to be *practical* all the necessary film-data needs to be stored in a clever way so that they can be accessed *efficiently*. Unfortunately, the developer of **Warwick+** hasn't taken CS126, so they don't really know how to design and implement data structures. With this project, you will have the opportunity to help the developer finalise this application. This task mostly involves designing data structures and embodying them into **Warwick+** so that the different functionalities of the application are efficient.

In order to help you with your task, the developer of **Warwick+** has provided you with a large part of the film-data that are going to be used in the application. These are stored in 3 separate files containing:

- `Movies`: the data about the films such as an *ID* number for each movie, a *title*, and *runtime*.
- `Credits`: the data about who starred in and produced the films.
- `Ratings`: the data about what different users thought about a film (rated out of 5 stars), and when the film was rated.

The developer has left the source code of **Warwick+** for you to finalise. In this code, there are a few Java classes that remain *unfinished*. Specifically, these are the **Movies** class, the **Credits** class, and the **Ratings** class. Furthermore, the developer provided the methods-functions you need to implement for each of these classes, together with their JavaDoc specifications.

Apart from the above, the developer has also implemented the `MyArrayList` data structure into a 4th dataset (called `Keywords`) to show you where to store your data structures and how they can be incorporated into the pre-made classes.

Finally, the developer has left some instructions for you, which include how to build, run and test your code; as well as the file structure of the application (see Section 4).

Therefore, your task is to implement the functions within the `Movies`, `Credits`, and `Ratings` classes through the use of your own data structures.

2 Submission Details

You should submit the following in a **single ZIP file** by **Monday 20th March @ 12 noon** via **Tabula**:

- The `main` directory of the application including **ALL** code located within it. For more information, see Section 4.5.2.
- A 1500-word report discussing the data structure(s) you have implemented for the 3 classes. You are required to justify your choice of the data structure(s) you selected then explain how you implemented it/them.

  The report can be structured however you see fit, but should be saved as a PDF document. References, captions, tables, figures and code listings do not count towards the word limit.

The ZIP file should be uploaded to Tabula by the deadline specified. Instructions for combining all your files together into a ZIP file on the DCS system can be found by running the command `man zip` in the terminal of any DCS machine.

All marking will be done in accordance with the University's marking scheme (more details can be found at `https://warwick.ac.uk/fac/sci/dcs/teaching/handbook/assessment/`). The work you submit should be your own work, and thus should abide by the University's rules on plagiarism. All late submissions and cases of plagiarism will be handled in accordance with the University's regulations (Reg. 36.3 and 11 respectively). Your code will be evaluated on the Department of Computer Science machines. Therefore, and to avoid getting a mark of 0, you should test and verify that your code works on these machines before submitting it to Tabula.

When developing your code, you are **not allowed** to use any data structure that is already implemented within a Java library. This includes the data structures found in the Java `util` package. However, you are allowed to use any Java data structure interface.

It should also be noted that submitting a solution that utilises the `MyArrayList` will not score marks since an example of this solution has been provided for you.

To summarise, you should implement your own data structures within the `Movies`, `Credits`, and `Ratings` classes.

3 Guidance

Firstly, don't panic! Have a read through the documentation provided in Section 4. This explains how to build and run the application. This can be done without writing anything, so make sure you can do that first.

Then have a look at the comments and functions found in the `Movies`, `Credits`, and `Ratings` classes. More information about these classes can be found in Section 4.5.2. Each function you need to implement is preceded by a comment that:

- Describes the purpose of the function.
- Lists each of the parameters for the function (lines starting with `@param`), and what the function should return (lines starting with `@return`).

It is recommended to start coding the `Ratings` class first. This is because it is the smallest and the simplest of the 3 classes you need to implement. When you have completed a function, you can test it using the test suite described in Section 4.4. More details about the location of the code used for testing can be found in Section 4.5.3.

If you have any issues, don't hesitate to contact the module organisers, or the lab tutors! Their contact details can be found on the module's webpage.

4 Warwick+

Warwick+ is a small Java application that pulls in data from a collection of *Comma Separated Value* (CSV) files. It is designed to have a lightweight *user interface* (UI), so that users can inspect and query the data. The application also has a *testing suite* connected to it to ensure all the functions work as expected. The functions called in the **Warwick+** UI are the same as those called in the testing, so if the tests work, the UI will also work.

4.1 Required Software

For the **Warwick+** to compile and run, Java 11 is required. If you are running **Warwick+** on the DCS system, then you don't need to worry about this as it has already been installed for you. However, if you are planning on working on your own machine, then you will need to make sure that you are using Java 11.

Whilst a newer version of Java can be utilised, other parts of the application will also have to be updated and this has not been tested. As such, it is highly recommended you download and use Java 11.

4.2 Building Warwick+

To compile the code, simply run the command shown in the table below in the head directory (the one with `src` directory in it). This will also try and execute the tests.

Linux/DCS System	MacOS	Windows
`./gradlew build`	`./gradlew build`	`./gradlew.bat build`

4.3 Running the Warwick+ Application

To run the application, simply run the command shown in the table below in the head directory (the one with `src` directory in it).

Linux/DCS System	MacOS	Windows
`./gradlew run`	`./gradlew run`	`./gradlew.bat run`

If the code has not been compiled, this command will also compile the code. When this is done, a window will appear with the UI for the application. The terminal will not be able to be used at this time. Instead it will print anything required from the program. To stop the application, simply close the window or press `CTRL` and `C` at the same time in the terminal.

Important Note: The terminal will say that it ran up to 75%, and will stay there for the duration of the program. This is fine, and is expected. All print statements will still go to this terminal, and will appear above the loading bar.

Important Note: The initial UI is dependant on the functions `Credits.getStarsCastID()` and `Credits.getSuperStarCastID()`, and will not load until these functions have returned. As such, the UI may take some time to fully load if these functions have not been implemented efficiently.

4.4 Running the Warwick+ Test Suite

Linux/DCS System	MacOS	Windows
`./gradlew test`	`./gradlew test`	`./gradlew.bat test`

If the code has not been compiled, this command will also compile the code. When ran, this will produce the output from each test function. It will also produce a webpage of the results, which can be found in `build/reports/test/test/index.html`

4.5 Warwick+ File Structure

Every effort has been made to keep the file structure simple and clean, whilst maintaining good coding practices. In the following subsections, a brief description of each of the key directories is given.

4.5.1 `data/`

This directory stores all the data files that are pulled into the application. There are 4 `.csv` files in this directory, 1 for each of the datasets described in Section 1. Each line in these files is a different entry, with values being separated by commas (hence the name Comma Separated Values). You do not need to add, edit or remove anything from this directory for your coursework. More details on how these files are structured can be found in Section 4.6.

4.5.2 `src/main/`

This directory stores all the Java code for the application. As such, there are a number of directories and files in this directory, each of which are required for the application and/or the UI to function. To make things simpler, there are 3 key directories that will be useful for you:

- `java/interfaces/`: Stores the interface classes for the data sets. You do not need to add, edit or remove anything from this directory, but it may be useful to read through.
- `java/stores/`: Stores the classes for the data sets. This is where the `Keywords`, `Movies`, `Credits`, `Ratings` from Section 1 are located, the latter 3 of which are the classes you need to complete. Therefore, you should only need to edit the following files, but it might be worth reading the others:
  - `Movies.java`: Stores and queries all the data about the films.
  - `Credits.java`: Stores and queries all the data about who stared in and worked on the films. The code in this file relies on the Cast and Crew classes which can be found in the `Cast.java` and `Crew.java` files respectively.
  - `Ratings.java`: Stores and queries all the data about the ratings given to films.
- `java/structures/`: Stores the classes for your data structures. As an example, `MyArrayList` from Lab 1 has been provided in there. Any classes you add in here can be accessed by the classes in the stores directory (assuming the classes you add are public). You may add any files you wish to this directory, but `MyArrayList.java` and `IList.java` should *not* be altered or removed, as these are relied on for `Keywords`.

4.5.3 `src/test/`

This directory stores all the code that is related solely to the JUnit tests. As such, there is a Java file for each of the stores you need to implement, and one that manages the tests. You do not need to add, edit or remove anything from this directory for your coursework.

4.6 Data used for Warwick+

All of the available data used by the **Warwick+** application can be found in the `data` directory, as described in Section 4.5.1. Each file in this directory contains a large collection of values, separated by commas (hence the CSV file type). Therefore, each of these can be opened by your favourite spreadsheet program. Most of these values are integers or floating point values, but some are strings. In the cases of strings, double quotation marks (") are used at the beginning and end of the value. Where multiple elements could exist in that value, a JSON object has been used. You do not need to parse these files, **Warwick+** will do that for you in the `LoadData` class. The data generated by the `LoadData` class is passed to the corresponding data store class (`Movies`, `Credits`, `Ratings` and `Keywords`) using the `add` function. The only exception to this is the `Movies` class, more details for this can be found in Section 4.7.1.

To make development easier, we have provided only 1000 films present in the data. This means that there are 1000 entries in the credits data set, and 1000 entries in the keywords data set. However, some films may not have any cast and/or crew (that information may not have been released yet, or it is not known), and some films don't have keywords. In these cases, an empty list of the required classes will be provided to the `add` function. All 1000 films have at least 1 rating.

Important Note: When the code is marked, we will be using a *larger* film data-set. Therefore your implementation should be able to *accommodate larger data sets*, i.e., larger than what you have been provided. In that respect, the data structures you develop should be both memory efficient and time efficient. Every effort has been made to ensure the data we have given you is a fair representation of the larger data set. The problem set is an order of magnitude (10×) larger than the data set provided.

Films		1000
Credits	Film Entries	1000
	Unique Cast	11142
	Unique Crew	9038
Ratings		47343
Keywords	Film Entires	1000
	Unique Keywords	2051

4.7 Key Stats

4.7.1 Movies Metadata

The following is a list all of the data stored about a film using the given name from the CSV file, in the same order they are in the CSV file. Blue fields are the ones that are added through the `add` function in the `Movies` class.

- `adult`: A boolean representing whether the film is an adult film.
- `belongs_to_collection`: A JSON object that stores all the details about the collection a film is part of. This is added to the film using the `addToCollection` function in the `Movies` class. If the film is part of a collection, the collection will contain a collection ID, a collection name, a poster URL related to the collection and a backdrop URL related to the collection.
- `budget`: A long integer that stores the budget of the film in US Dollars. If the budget is not known, then the budget is set to 0. Therefore, this value will always be greater than or equal to 0.
- `genres`: A JSON list that contains all the genres the films is part of. Each genre is represented as a key-value pair, where the key is represented as an ID number, and the value is represented as a string. For ease, **Warwick**+ passes this as an array of `Genre` objects.
- `homepage`: A string representing a URL of the homepage of the film. If the film has no homepage, then this string is left empty.
- `id`: An integer representing the ID of the film. This is used to link this film to other pieces of data in other data sets.
- `imdb_id`: A string representing the unique part of the IMDb URL for a given film. This is added using the `setIMDB` function in the `Movies` class.
- `original_language`: A 2-character string representing the ISO 639 language that the film was originally produced in.
- `original_title`: A string representing the original title of the film. This string may be the same as the `title` field, but this is not always the case.
- `overview`: A string representing an overview of the film.
- `popularity`: A floating point value that represents the relative popularity of the film. This value is always greater than or equal to 0. This data is added by the `setPopularity` function in the `Movies` class.
- `poster_path`: A string representing the unique part of a URL for the film poster. Not all films have a poster available. In these cases, an empty string is given.
- `production_companies`: A JSON list that stores the production countries for a film. Each entry in the JSON list has a key value pair, where the key is the ID of the company, and the value is the name of the company. For ease, **Warwick**+ parses each list element into a `Company` object. This object is then added using the `addProductionCompany` in the `Movies` class.
- `production_countries`: A JSON list that stores the production countries for a film. Each entry in the JSON list has a key value pair, where the key is the ISO 3166 2-character string, and the value is the country name. For ease, **Warwick**+ handles only the key, and uses a function to match this to the country name. This string is added using the `addProductionCountry` in the `Movies` class.
- `release_date`: A long integer representing the number of seconds from 1st January 1970 when the film was released. For ease, **Warwick**+ passes this into a Java Calendar object (more details can be found here: `https://docs.oracle.com/javase/7/docs/api/java/util/Calendar.html`).

- `revenue`: A long integer representing the amount of money made by the film in US Dollars. If the revenue of the film is not known, then the revenue is set to 0. Therefore, this value will always be greater than or equal to 0.
- `runtime`: A floating point value representing the number of minutes the film takes to play. If the runtime is not know, then the runtime is set to 0. Therefore, this value will always be greater than or equal to 0.
- `spoken_languages`: A JSON list that stores all the languages that the film is available in. This is stored as a list of key-value pairs, where the key is the 2-character ISO 639 code, and the value is the language name. For ease, **Warwick+** parses these as an array of keys stored as strings.
- `status`: A string representing the current state of the film.
- `tagline`: A string representing the poster tagline of the film. A film is not guaranteed to have a tagline. In this case, an empty string is presented.
- `title`: A string representing the English title of the film.
- `video`: A boolean representing whether the film is a "direct-to-video" film.
- `vote_average`: A floating point value representing an average score as given by a those on IMDb at the time the data was collected. As such, it is not used in the Review dataset. The score will always be between 0 and 10. This data is added using the `setVote` function in the `Movies` class.
- `vote_count`: An integer representing the number of votes on IMDb at the time the data was collected, to calculate the score for `vote_average`. As such, it is not used in the Review dataset. This will always be greater than or equal to 0. This data is added using the `setVote` function in the `Movies` class.

4.7.2 Credits

The following is a list all of the data stored about the cast and crew of a film using the given name from the CSV file, in the same order they are in the CSV file:

- `cast`: A JSON list that contains all the cast for a particular film. In the JSON list, each cast member has details that relate to there role in the film and themselves. For ease, **Warwick+** passes this into an array of `Cast` objects, with as many fields populated as possible.
- `crew`: A JSON list that contains all the crew for a particular film. In the JSON list, each crew member has details that relate to their role in the film and themselves. For ease, **Warwick+** passes this into an array of `Crew` objects, with as many fields populated as possible.
- `id`: An integer representing the film ID. The values for this directly correlates to the `id` field in the movies data set (see Section 4.7.1).

4.7.3 Ratings

The following is a list all of the data stored about the ratings for a film using the given name from the CSV file, in the same order they are in the CSV file:

- `userId`: An integer representing the user ID. The value of this is greater than 0.
- `movieId`: An integer representing the film ID. The value of this integer directly correlates to the `id` field in the movies data set (see Section 4.7.1).
- `rating`: A floating point value representing the rating between 0 and 5 inclusive.
- `timestamp`: A long integer representing the number of seconds from 1st January 1970 when the rating was made. For ease, **Warwick+** passes this into a Java Calendar object (more details can be found here: `https://docs.oracle.com/javase/7/docs/api/java/util/Calendar.html`).

4.7.4 Keywords

The following is a list all of the data stored about the keywords for a film using the given name from the CSV file, in the same order they appear in the CSV file:

- `id`: An integer representing the film ID. The values for this directly correlates to the `id` field in the movies data set (see Section 4.7.1).
- `keywords`: A JSON list that contains all the keywords relating to a given film. Each keyword is represented as a key-value pair, where the key is represented as an ID number, and the value is represented as a string. For ease, **Warwick**+ passes this into an array of `Keyword` objects.

5 Screenshots

Figure 1: Main Screen (after searching)

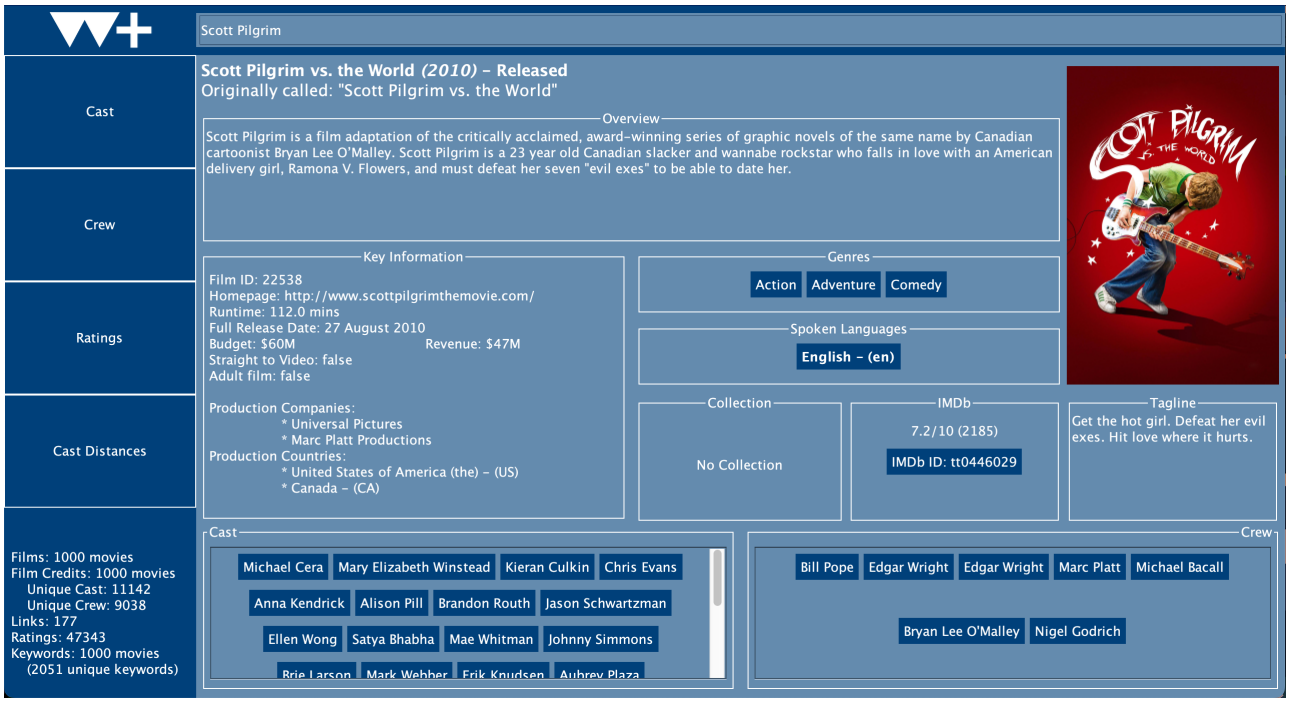

Figure 2: Film Screen

Figure 3: Cast Screen

Figure 4: Crew Screen

Figure 5: Ratings Screen

Figure 6: Cast Distance Screen