[CS209A-22Fall] Assignment 1 (100 points)

> Question Design: Yida Tao
>
> Test cases & OJ: Yilun Qiu; Code sample: Ruotong Zou
>
> Git & Code styles: Chi Xu

Deadline: 23:55pm Oct. 21

Grace period: 48 hours. If you missed the deadline, you could still submit within a 48-hour grace period, i.e., before 23:55pm Oct. 23. Yet, you'll get a 40% penalty (i.e., you'll get 60% of your score). Submissions after the grace period will NOT be accepted.

Movie Analysis

In this assignment, you will design a `MovieAnalyzer` class, which could read a movie dataset and perform various useful analyses. You'll have a chance to use techniques such as `Collections`, `Lambda`, and `Streams`, which we've covered in the lectures.

The `MovieAnalyzer` class has one constructor that reads a dataset file from a given path. This class also has 6 other methods that perform data analyses. You'll implement these methods in the `MovieAnalyzer` class. Method details are described below.

0. Reading the dataset

```
public MovieAnalyzer(String dataset_path)
```

The constructor of `MovieAnalyzer` takes the path of the dataset file and reads the data. The dataset is in `csv` format and has the following columns:

- Series_Title - Name of the movie
- Released_Year - Year at which that movie released
- Certificate - Certificate earned by that movie
- Runtime - Total runtime of the movie
- Genre - Genre of the movie
- IMDB_Rating - Rating of the movie at IMDB site
- Overview - mini story/ summary
- Meta_score - Score earned by the movie
- Director - Name of the Director
- Star1,Star2,Star3,Star4 - Name of the Stars
- Noofvotes - Total number of votes
- Gross - Money earned by that movie

Note: For data cells that are empty or ill-formatted, you could simply ignore that entire row when you perform the following analysis.

1. Movie count by year (10 points)

```
public Map<Integer, Integer> getMovieCountByYear()
```

This method returns a `<year, count>` map, where the key is the year while the value is the number of movies released in that year.

The map should be sorted by **descending order of** `year` (i.e., from the latest to the earliest).

2. Movie count by genre (15 points)

```
public Map<String, Integer> getMovieCountByGenre()
```

This method returns a `<genre, count>` map, where the key is the genre while the value is the number of movies in that genre.

The map should be sorted by **descending order of** `count` (i.e., from the most frequent genre to the least frequent genre). If two genres have the same count, then they should be sorted by the alphabetical order of the genre names.

3. Movie count by co-stars (15 points)

```
public Map<List<String>, Integer> getCoStarCount()
```

If two people are the stars for the same movie, then the number of movies that they co-starred increases by 1.

This method returns a `<[star1, star2], count>` map, where the key is a list of names of the stars while the value is the number of movies that they have co-starred in. Note that the length of the key is 2 and the names of the stars should be sorted by alphabetical order in the list.

4. Top movies (20 points)

```
public List<String> getTopMovies(int top_k, String by)
```

This method returns the top K movies (paramter `top_k`) by the given criterion (paramter `by`). Specifically,

- `by="runtime"`: the results should be movies sorted by **descending order** of `runtime` (from the longest movies to the shortest movies) .
- `by="overview"`: the results should be movies sorted by **descending order** of the length of the overview (from movies with the longest overview to movies with the shortest overview).

Note that the results should be a list of movie titles. If two movies have the same runtime or overview length, then they should be sorted by alphabetical order of their titles.

5. Top stars (20 points)

```
public List<String> getTopStars(int top_k, String by)
```

This method returns the top K stars (paramter `top_k`) by the given criterion (paramter `by`). Specifically,

- `by="rating"`: the results should be stars sorted by **descending order of the average** `rating` of the movies that s/he have starred in.
- `by="gross"`: the results should be stars sorted by **descending order of the average** `gross` of the movies that s/he have starred in.

Note that the results should be a list of star names. If two stars have the same average rating or gross, then they should be sorted by the alphabetical order of their names.

6. Search movies (20 points)

```
public List<String> searchMovies(String genre, float min_rating, int max_runtime)
```

This method searches movies based on three criterion:

- `genre`: genre of the movie
- `min_rating`: the rating of the movie should >= `min_rating`
- `max_runtime`: the runtime (min) of the movie should <= `max_runtime`

Note that the results should be a list of movie titles that meet the given criteria, and sorted by alphabetical order of the titles.

Evaluation

- **Code Correctness**: We deploy automatic test cases on the OJ system (https://oj.cse.sustech.edu.cn) to test the correctness of your code. Please submit `MovieAnalyzer.java` to OJ. You could use our sample test cases and sample test data to test your code locally before submitting to OJ.
- **Version Control**: You should use `GitHub` to manage the code changes of your project (see lab 1 for further details of how to use `git`). You should made **at least 2 commits**. Your remote repo on GitHub **should be set to `private` before A1 deadline, so that no one else will see your code.**
- **Coding Style**: You should pay attention to write readable and maintainable code along the way. See lab 1 for how to use `CheckStyle` for that purpose. **After the deadline of A1**, you can set your GitHub repo to `public`, and we'll check whether any of your commits have reduced `CheckStyle` warnings according to `google_checks.xml`.

OJ Tips

- Please specify `StandardCharsets.UTF_8` when you read the `.csv` file.
- Please don't include any Chinese characters in your code comments.