CineReviews

Object-Oriented Programming
2nd Project, version 1.4 – 2023-05-28
Contact: carla.ferreira@fct.unl.pt

https://discord.gg/5np2Cee7

Texto adicionado e mensagens de comandos modificadas são apresentados a magenta.

Important remarks

Deadline until 23h59 (Lisbon time) of June 2, 2023.

Team this project is to be MADE BY GROUPS OF 2 STUDENTS.

Deliverables: Submission and acceptance of the source code to **Mooshak** (score > 0). See the course website for further details on how to submit projects to Mooshak.

Recommendations: We value the documentation of your source code, as well as the usage of the best programming style possible and, of course, the correct functioning of the project. Please carefully comment on both interfaces and classes. The documentation in classes can, of course, refer to the one in interfaces, where appropriate. Please comment on the methods explaining what they mean and defining preconditions for their usage. Students may and should discuss any doubts with the teaching team, and with other students, but may not share their code with other colleagues. This project is to be conducted with full respect for the Code of Ethics available on the course website.

1 Development of the application *CineReviews*

1.1 Problem description

The goal of this project is to develop an online review application that collects reviews from movie critics and audiences to determine a movie or series's overall score.

User User accounts can either be a *critic account* or an *audience account*. The reviews made by critics are 5 times more important than the reviews made by the audience. For example, if a reviewer gives 4 points, an audience member gives 5 points, and another audience member gives 4 points, the average review rate would be ((5*4)+(1*5)+(1*4))/(5+1+1). The result should be printed with 1 decimal point. In this example, this would print 4.1. Each registered user is characterised by a username (unique identifier). The application also has *admin* accounts that are responsible for adding information about the shows and artists. Besides the username, admin accounts also have a password and a collection of shows and artists added by each admin.

Artist For each *artist*, the application maintains its name, date and place of birth, and the shows it has participated in as an actor, director, or creator.

Show The application needs to have an extensive library with information about movies and series. The information to be kept is defined next.

- Each movie has the following associated information: title, director, duration (in minutes), age certification, release date, genres, and the top cast. Consider that each movie is uniquely identified by its title.
- Each series has the following associated information: title, creator, number of seasons, age certification, release date, genres, and the top cast. Consider that each series is uniquely identified by its title.

Reviews Each review has the following data: the title of the show, the user who wrote the review, a comment, and the overall classification (excellent, good, average, poor, or terrible). To calculate the average of the reviews assign an integer value to each level of the rating: excellent - 5, good - 4, average - 3, poor - 2, terrible - 1. A review is uniquely identified by the show title and the username of the critic or audience member who wrote the review.

2 Commands

In this section, we present all the commands that the system must be able to interpret and execute. In the following examples, we differentiate `text written by the user` from the feedback written by the program in the console. You may assume that the user will make no mistakes when using the program other than those described in this document. In other words, you only need to take care of the error situations described here, in the exact same order as they are described.

Commands are case-insensitive. For example, the `exit` command may be written using any combination of upper and lowercase characters, such as `EXIT`, `exit`, `Exit`, `exIT`, and so on. In the examples provided in this document, the symbol ↵ denotes a change of line.

If the user introduces an unknown command, the program must write in the console the message Unknown command. Type help to see available commands. For example, the non existing command `someRandomCommand` would have the following effect:

```
someRandomCommand↵
Unknown command. Type help to see available commands.↵
```

If there are additional tokens in the line (e.g. parameter for the command you were trying to write, the program will try to consume them as commands, as well. So, in this example, `someRandom Command` would be interpreted as two unknown commands: `someRandom` and `Command`, leading to two error messages.

```
someRandom Command↵
Unknown command. Type help to see available commands.↵
Unknown command. Type help to see available commands.↵
```

Several commands have arguments. Unless explicitly stated in this document, you may assume that the user will only write arguments of the correct type. However, some of those arguments may have an incorrect value. For that reason, we need to test each argument exactly in the order specified in this document. Arguments will be denoted **`with this style`**, in their description, for easier identification. Also, for any String arguments in the commands,

assume they are case-sensitive. So, "star wars" and "Star Wars" would be two different movie titles, for instance. Not realistic, but it simplifies the implementation.

2.1 `exit` command

Terminates the execution of the program. This command does not require any arguments. The following scenario illustrates its usage.

```
exit↵
Bye!↵
```

This command always succeeds.

2.2 `help` command

Shows the available commands. This command does not require any arguments. The following scenario illustrates its usage.

```
help↵
register - registers a user in the system↵
users - lists all registered users↵
movie - uploads a new movie↵
series - uploads a new series↵
shows - lists all shows↵
artist - adds bio information about an artist↵
credits - lists the bio and credits of an artist↵
review - adds a review to a show↵
reviews - lists the reviews of a show↵
genre - lists shows of given genres↵
released - lists shows released in a given year↵
avoiders - lists artists that have no common projects↵
friends - lists artists that have more projects together↵
help - shows the available commands↵
exit - terminates the execution of the program↵
```

This command always succeeds. When executed, it shows the available commands.

2.3 `register` command

Registers a user in the system. The command receives as arguments the **`user type`**, which can be either **`audience`**, **`critic`**, or **`admin`** followed by the **`username`**. For admin accounts, it is also received the account **`password`**.

```
register critic roger.ebert↵
User roger.ebert was registered as critic.↵
register audience alice↵
User alice was registered as audience.↵
register admin grogu 1234↵
User grogu was registered as admin.↵
```

This command will fail if:

1. The **user type** is unknown, the adequate error message is (Unknown user type!).
2. The user identifier already exists. In this case, the error feedback message is (User <username> already exists!).

```
register moderator peter.bradshaw↵
Unknown user type!↵
register admin roger.ebert password↵
User roger.ebert already exists!↵
```

2.4 `users` command

Lists all registered users. This command always succeeds. If there are no registered users, the output is (No users registered.). Otherwise, it will print the message (All registered users:) in the first line and then print all users alphabetically ordered by identifier. For critic and audience users the format is (User <username> has posted <posts counter> reviews), while for admin the format is (Admin <username> has uploaded <shows counter> shows).

```
users↵
No users registered.↵
... after registering several users...
users↵
All registered users:↵
User alice has posted 10 reviews↵
Admin grogu has uploaded 15 shows↵
User roger.ebert has posted 20 reviews↵
```

2.5 Command `movie`

Uploads a new movie. The command receives as arguments the admin **username** and their **password**, followed by the movie **title**, the **director's name**, the **duration** in minutes, the **age certification**, the **year of release**, an integer number >= 1 of genres (**noGenre**) and the movie **genres**, an integer number >= 1 of cast members (**noCast**) and the **name of each of the cast members**. Any of the artists (director or cast members) that do not yet exist in the application are added with an undefined date and place of birth.

```
movie grogu 1234↵
Dune↵
Denis Villeneuve↵
155↵
12A↵
2019↵
3↵
Action↵
Adventure↵
Drama↵
4↵
Timothée Chalamet↵
Rebecca Ferguson↵
Zendaya↵
Oscar Isaac↵
Movie Dune (2019) was uploaded [5 new artists were created].↵
```

The following errors may occur:

1. If the **username** does not exist or is not an admin user (Admin <username> does not exist!)
2. The **password** is wrong. In this case, the error feedback message is (Invalid authentication!).
3. If the movie **title** already exists in the application either as a movie or series title, the error message is (Show <title> already exists!)

```
movie peter.bradshaw 1234↵
Dune↵
Denis Villeneuve↵
155↵
12A↵
2019↵
1↵
Action↵
1↵
Timothée Chalamet↵
Admin peter.bradshaw does not exist!↵
movie roger.ebert 1234↵
Dune↵
Denis Villeneuve↵
155↵
12A↵
2019↵
1↵
Action↵
1↵
Timothée Chalamet↵
Admin roger.ebert does not exist!↵
```

```
movie grogu 4321↵
Dune↵
Denis Villeneuve↵
155↵
12A↵
2019↵
1↵
Action↵
1↵
Timothée Chalamet↵
Invalid authentication!↵
movie grogu 1234↵
Dune↵
Denis Villeneuve↵
155↵
12A↵
2019↵
1↵
Action↵
1↵
Timothée Chalamet↵
Show Dune already exists!↵
```

2.6 Command **series**

Uploads a new series. The command receives as arguments the admin **username** and their **password**, followed by the series **title**, the name of the series **creator**, the **number of seasons**, the **age certification**, the **year of release**, an integer number >= 1 of genres (**noGenre**) and the series **genres**, an integer number >= 1 of cast members (**noCast**) and the **name of each of the cast members**. Any of the artists (series creator or cast members) that do not yet exist in the application are added with an undefined date and place of birth.

```
series grogu 1234↵
Peaky Blinders↵
Steven Knight↵
6↵
16+↵
2013↵
3↵
Crime↵
Drama↵
Action↵
7↵
Cillian Murphy↵
Paul Anderson↵
Helen McCrory↵
Sam Neill↵
Annabelle Wallis↵
Joe Cole↵
Tom Hardy↵
Series Peaky Blinders (2013) was uploaded [0 new artists were created].↵
```

Commands `series` and `movie` have the same error situations. Check Section 2.5 for the details.

2.7 `shows` command

Lists all shows. This command always succeeds. If there are no shows uploaded in the application, the output is (No shows have been uploaded.). Otherwise, it will print the message (All shows:) in the first line and then print all shows by alphabetical order of their title. For each movie, it will show the title, director, duration, age certification, release year, the main genre (the first one to be given), and the cast. For each series, it will show the title, creator, number of seasons, age certification, release year, the main genre, and the cast. When listing the cast of a movie or series at most 3 members of the cast should be shown, taking into account the insertion order.

```
shows↵
No shows have been uploaded.↵
... after uploading several shows...
shows↵
All shows:↵
Dune; Denis Villeneuve; 155; 12A; 2019; Action; Timothée Chalamet; Rebecca Ferguson; Zendaya↵
IT Crowd; Graham Linehan; 5; 15+; 2016; Comedy; Chris O'Dowd; Richard Ayoade; Katherine Parkinson↵
Peaky Blinders; Steven Knight; 6; 16+; 2013; Crime; Cillian Murphy; Paul Anderson; Helen McCrory↵
```

2.8 Command `artist`

Adds bio information about an artist. The command receives as arguments the artist **`name`**, **`date`** and **`place`** of birth. If the artist does not exist in the application, meaning that the artist has not participated in any movie or series, they are added. Otherwise, if the artist has

participated in some movie or series but still has no bio information defined, this information is added to the artist.

```
artist Tom Hardy↵
15-09-1977↵
London, UK↵
Tom Hardy bio was updated.↵
artist Jane Campion↵
30-04-1954↵
Wellington, New Zealand↵
Jane Campion bio was created.↵
```

The following error may occur:

1. If the artist with **name** already has a bio (Bio of <artist name> is already available!)

```
artist Jane Campion↵
30-12-1990↵
Lisbon, Portugal↵
Bio of Jane Campion is already available!↵
```

2.9 `credits` command

Lists the bio and credits of an artist. This command receives as an argument the artist **name** and presents their bio, if available, and all their credits sorted by release year (more recent first), and then by title. For each credit it will show the title, release year, type of credit (actor, director, creator), and whether is a movie or series.

```
credits Jane Campion↵
30-04-1954↵
Wellington, New Zealand↵
... after uploading several shows of Jane Campion...
credits Jane Campion↵
30-04-1954↵
Wellington, New Zealand↵
The Power of the Dog; 2021; director [movie]↵
Top of the lake; 2013; creator [series]↵
credits Benedict Cumberbatch↵
The Power of the Dog; 2021; actor [movie]↵
Sherlock; 2010; actor [series]↵
artist Benedict Cumberbatch↵
19-07-1976↵
London, UK↵
Benedict Cumberbatch bio was updated.↵
credits Benedict Cumberbatch↵
19-07-1976↵
London, UK↵
The Power of the Dog; 2021; actor [movie]↵
Sherlock; 2010; actor [series]↵
```

This command will fail if:

1. The artist **name** does not exist. In this case, the error feedback message is (No information about <name>!).

```
credits Matthew Goode↵
No information about Matthew Goode!↵
```

2.10 `review` command

Adds a review to a show. The command receives as arguments the **username** of the critic or audience member, the show **title**, a **comment**, and the **classification**, which can be **excellent**, **good**, **average**, **poor**, or **terrible**. In case of success, the review is registered and the current number of reviews for that show is presented.

```
review roger.ebert Peaky Blinders↵
One of the most daft and thrilling hours of the tv week!↵
good↵
Review for Peaky Blinders was registered [12 reviews].↵
review alice Peaky Blinders ↵
Peaky Blinders goes out in predictably thrilling fashion!↵
excellent↵
Review for Peaky Blinders was registered [13 reviews].↵
```

This command will fail if:

1. The **username** does not exist. In this case, the error feedback message is (User <username> does not exist!).
2. The **username** is associated to an admin. In this case, the error feedback message is (Admin <username> cannot review shows!).
3. The **title** is not associated with any show. In this case, the error feedback message is (Show <title> does not exist!).
4. The show with **title** was already reviewed by the **username**. In this case, the error feedback message is (<username> has already reviewed <title>!).

```
review mary Peaky Blinders ↵
Terrible show!↵
poor↵
User mary does not exist!↵
review grogu The Mandalorian ↵
May the Fourth be with you↵
excellent↵
Admin grogu cannot review shows!↵
review roger.ebert The Mandalorian ↵
May the Fourth be with you!↵
excellent↵
Show The Mandalorian does not exist!↵
review roger.ebert Peaky Blinders ↵
One of the most daft hours of the tv week!↵
good↵
roger.ebert has already reviewed Peaky Blinders!↵
```

2.11 `reviews` command

Lists the reviews of a show. This command receives as an argument the show **`title`** and presents all its reviews. If the show has no reviews, the program will write the feedback message (Show <title> has no reviews.). Otherwise, it will print the message (Reviews of <title> [<average of reviews>]:) in the first line and then print the reviews showing first the critics and then the audience members, sorted from the highest to the lowest score and then alphabetically by username. The format is (Review of <username> (<user type>): <comment> [<classification>]).

```
reviews Peaky Blinders↵
Show Peaky Blinders has no reviews.↵
... after receiving two reviews...
reviews Peaky Blinders↵
Reviews of Peaky Blinders [4.5]:↵
Review of stuart.jeffries (critic): Tommy Shelby's back where we want him to be: in all kinds of trouble. [excellent]↵
Review of roger.ebert (critic): One of the most daft and thrilling hours of the tv week! [good]↵
Review of alice (audience): Peaky Blinders goes out in predictably thrilling fashion! [excellent]↵
```

This command will fail if:

1. The **`title`** is not associated with any show. In this case, the error feedback message is (Show <title> does not exist!).

```
reviews The Mandalorian↵
Show The Mandalorian does not exist!↵
```

2.12 `genre` command

Lists shows of given genres. This command receives as argument an integer number >= 1 of genres (**`noGenre`**) and the show **`genres`**. The program will list the shows that cover all

the genres ordered by score (average of reviews), then by release date, and lastly by title. If no show was found with all the genres received as input, the program will write the feedback message (No show was found within the criteria.). Otherwise, it will print the message (Search by genre:). The format for movies is (Movie <title> by <director> released on <release date> [<average reviews>]). The format for series is similar (Series <title> by <creator> released on <release date> [<average reviews>]).

```
genre↵
3↵
Comedy↵
Horror↵
Biography↵
No show was found within the criteria.↵
genre↵
2↵
Drama↵
Action↵
Search by genre:↵
Movie Dune by Denis Villeneuve released on 2019 [4.3] ↵
Series Peaky Blinders by Steven Knight released on 2013 [4.0]↵
```

2.13 `released` command

Lists shows released on a given year. This command receives as argument a (**release date**). The program will list the shows with the same release date ordered by average score, and then by title. If no show was found with that release date the program will write the feedback message (No show was found within the criteria.). Otherwise, it will print the message (Shows released on <release date>:). The format for movies is (Movie <title> by <director> released on <release date> [<average reviews>]). The format for movies is similar (Series <title> by <creator> [<average reviews>]).

```
released↵
1905↵
No show was found within the criteria.↵
released↵
2019↵
Shows released on 2019:↵
Movie Knives Out by Rian Johnson released on 2019 [4.4]↵
Movie Dune by Denis Villeneuve released on 2019 [4.0]↵
Movie Us by Jordan Peele released on 2019 [4.0]↵
```

2.14 `avoiders` command

Artists that have no common projects. This command list the largest group of artists that have never participated in a common movie or series. If no artist exists in the application the feedback message will be (No artists yet!). If all artists have common projects we will write the feedback message (It is a small world!). Otherwise, it will print the message (These <number of artists> artists never worked together:), followed by a list of comma-separated names sorted

alphabetically. In case of a draw, .i.e. there are several groups of artists, these groups should also be sorted alphabetically.

```
avoiders↵
No artists yet!↵
... after uploading a few shows ...
avoiders↵
These 4 artists never worked together:↵
Denis Villeneuve, Jenna Ortega, Jessie Buckley, Jordan Peele↵
... after uploading a few more shows ...
avoiders↵
These 2 artists never worked together:↵
Jenna Ortega, Jessie Buckley↵
... after uploading a few more shows ...
avoiders↵
It is a small world!.↵
```

2.15 `friends` command

Artists that have more projects together. This command lists the pairs of artists that have more projects (movies or series) together. If no artist exists in the application the feedback message will be (No artists yet!). If no collaborations exist between any artists the feedback message will be (No collaborations yet!). Otherwise, it will print the message (These artists have worked on <number of common projects> projects together:), followed by a list of pairs of names that have the highest number of collaborations sorted alphabetically. Also, each pair of names is also sorted alphabetically.

```
friends↵
No artists yet!↵
... after uploading a few shows ...
friends↵
No collaborations yet!↵
friends↵
These artists have worked on 3 projects together:↵
Ethan Hawke and Julie Delpy↵
Ethan Hawke and Richard Linklater↵
Julie Delpy and Richard Linklater↵
Uma Thurman and Quentin Tarantino↵
... after uploading a few more shows ...
friends↵
These artists have worked on 5 projects together:↵
Ethan Hawke and Julie Delpy↵
```

3 Developing this project

Your program should take the best advantage of the elements taught in the Object-Oriented Programming course. You should make this application as **extensible as possible** to make it easier to add, for instance, new kinds of users and shows.

You can start by developing the main user interface of your program, clearly identifying which commands your application should support, their inputs and outputs, and error conditions. Then, you need to identify the entities required for implementing this system. Carefully specify the **interfaces** and **classes** that you will need. You should document their conception and development using a class diagram, as well as document your code adequately, with Javadoc. **Note that in this project you should use the standard Java Collection classes.**

It is a good idea to build a skeleton of your `Main` class, to handle data input and output, supporting the interaction with your program. In an early stage, your program will not really do much. Remember the **stable version rule**: do not try to do everything at the same time. Build your program incrementally, and test the small increments as you build the new functionalities in your new system. If necessary, create small testing programs to test your classes and interfaces.

Have a careful look at the test files, when they become available. You should start with a really bare-bones system with the `help` and `exit` commands, which are good enough for checking whether your commands interpreter is working well, to begin with. Then, address the registration of users and test this command. Then, move on to adding the series and movies, one by one. Check these commands are ok. And so on. Step by step, you will incrementally add functionalities to your program and test them. **Do not try to make all functionalities simultaneously. It is a really bad idea.**

Last, but not least, **do not underestimate the effort for this project**.

4 Submission to Mooshak

To submit your project to Mooshak, please register in the Mooshak contest POO2023-TP2 and follow the instructions that will be made available on the Moodle course website.

4.1 Command syntax

For each command, the program will only produce one output. The error conditions of each command have to be checked in the exact same order as described in this document. If one of those conditions occurs, you do not need to check for the other ones, as you only present the feedback message corresponding to the first failing condition. However, the program does need to consume all the remaining input parameters, even if they are to be discarded.

4.2 Tests

We expect you to create your own tests, based on this specification. Create tests for the "happy day scenario", and also for error situations, so that whenever you add support for a new command, you also create suitable tests for it. So, build your tests as you build your project, incrementally. This mimics what happens with the Mooshak tests. The Mooshak tests verify incrementally the implementation of the commands. They will be made publicly available on May 21, 2023. When the sample test files become available, use them to test what you already have implemented, fix it if necessary, and start submitting your partial project to Mooshak. Do it from the start, even if you just implemented the exit and help commands. By then you will probably have more than those to test, anyway. Good luck!