Object Oriented Programming

CSE1100 – Resit Programming Exam

A. Zaidman / T. Overklift

Date: 11-01-2024 Duration: 3 hours Pages: 6

Allowed resources

During the exam you are **NOT** allowed to use books or other paper resources. The following resources are available on the exam computer:

- A PDF version of the lecture slides (located on the S-drive of the computer).
- A local (limited) version of Answers EWI.
- The Java API documentation (located on the desktop of the computer).

Set up your computer

1. Log in to windows using:
   Username: `EWI-CSE1100`
   Password: `Good11Luck01!`
2. Once the environment loads you will be asked to provide your **personal NetID and password combination**. Entering these will give you access to a private disk (*P-drive*) where your final exam submission will be stored.
3. Once this is done, please open **My Computer** and navigate to the **Z-drive**, where you will find your exam template. Open the exam template folder, and then open the “OOP Exam Template” IntelliJ file. While IntelliJ starts, you can read the rest of the exam.
4. Once the project has opened you can start implementing your exam in this project! Ensure that you are working in the project on the **Z-drive**, to avoid any issues when handing in!

Rules

- You are not allowed to use the **internet**.
- **Mobile phones / smartwatches** are not allowed. You must turn them off *and* put them away in your coat / backpack.
- Backpacks remain **on the ground**, not on your desk.
- Attempts at **fraud**, or actual acts of fraud, will be punished by the Board of Examiners.

About the exam

- Your project (sources *and* tests) **must compile** to get a passing grade. Sometimes IntelliJ will allow you to run part of your code even when some classes fail to compile. We will still see this as a compilation failure. Ensure that **ALL classes you hand in compile.** If your solution does not compile, you will not get any points.
- **Follow the submission instructions in the last part of the exam carefully**.
- **Stop writing code a few minutes before the end** of the exam so you have time to make sure your submission compiles and to hand in the exam.
- The **grading** is explained on the last page of this exam.

Master Movies

A new startup wants to try their hand at the age-old problem of finding good content to consume at any given time. To this end they want to create an app that can help. Sort through the endless stream of content. To start out with something they can play with they ask you to put together a prototype that can filter through the content so they can test out the combination of different filters. They provide you with the following input:

```
MOVIE Barbie; Adventure, Comedy, Fantasy; PG-13
DURATION 1:54
BOX OFFICE 1442 million
CAST Margot Robbie; Barbie
CAST Ryan Gosling; Ken
SERIES Brooklyn 99; Comedy, Crime; PG-13
DURATION 153 episodes; 0:25
CAST Andy Samberg; Jake Peralta
CAST Melissa Fumero; Amy Santiago
CAST Andre Braugher; Raymond Holt
PODCAST Reply All - Long Distance; Talk-show, Technology; PG
DURATION 0:50
CAST Alex Goldman; themselves
CAST PJ Vogt; themselves
PODCAST Terrible thanks for asking; Talk-show; PG-13
DURATION 0:59; 656 episodes
CAST Nora McInerny; themselves
```

The order of the lines for a single item is fixed. However, **movies**, **series** & **podcasts** may appear in any order in the input file. For each item you will first get the name, then one or more genres, followed by the content rating.
Then, on the next line there is information about the playtime of the item. For movies this consists of just the playtime of the movie, for series this includes information about the number of episodes a series has. For podcasts this depends (on whether the podcast is a one-off or part of a serialisation).
Then, only for movies, there is a line with the box office revenue of the movie.
On the following lines, there will be information about the most relevant cast members of the item (the number of cast members included can vary), consisting of their name followed by the role they play.

The file **items.txt** is available in the template project.

Design and implement a program that:
- **Reads in** the file items.txt and has classes and structures to store and represent the information in the file.
- To enable user interaction, please provide a **command line interface** (reading from `System.in` and writing to `System.out`).

A **Movie** is characterised by:
- The title
- One or more genres
- The content rating *(G, PG, PG-13, R, NC-17; for more information see **option 5**).*
- The duration
- The box office revenue of the movie
- One or more cast members
    - The name of the cast member
    - The role the cast member plays

A **Series** is characterised by:

- The title
- One or more genres
- The content rating *(G, PG, PG-13, R, NC-17; for more information see **option 5**).*
- The duration
- The number of episodes
- One or more cast members
    - The name of the cast member
    - The role the cast member plays

A **Podcast** is characterised by:

- The title
- One or more genres
- The content rating *(G, PG, PG-13, R, NC-17; for more information see **option 5**).*
- The duration
- The number of episodes *(optional, not all podcasts consist of multiple episodes).*
- One or more cast members
    - The name of the cast member
    - The role the cast member plays

Menu and options that should be implemented:

```
Please make your choice:
1 – Show all items that meet all criteria.
2 – Filter items by genre.
3 – Filter items by maximum playtime.
4 – Filter items by minimum playtime.
5 - Filter items by content rating.
6 – Reset all criteria.
7 – Quit the application.
```

Option 1:

Show all the items that meet all the restrictions set by options 2, 3, 4 and 5. Please note that these restrictions can stack on top of each other.
If no restrictions have been set, print all items that have been read from the input file. Make sure this output is in a nicely readable format (please ensure that you do not use the default `toString()` method but create your own to format the content). Ensure that you also print a clear message in case there are no items that meet the criteria / there are no items.

The output could for instance look like (example has been shortened):

```
Movie: Barbie - Genres: Adventure, Comedy, Fantasy. Maturity rating: PG-13
Duration: 1 hour and 54 minutes
Box office revenue: 1442 million
Cast members: Margot Robbie (Barbie), Ryan Gosling (Ken)
Series: Brooklyn Nine-Nine - Gengres: Comedy, Crime. Maturity rating: PC-13
Duration: 25 minutes, 153 episodes.
Cast members: Andy Samberg (Jake Peralta), Melissa Fumero (Amy Santiago),
              Andre Brauger (Raymond Holt)
```

Option 2:

This option should restrict the items that are printed for option 1 based on a genre provided by the user. For example, If the user inputs “crime”, only items that have crime as (one of) their genre(s) should be printed when option 1 is called.

Option 3:

Sometimes we are limited by the amount of time we have available. This option should restrict the items that are printed for option 1 based a **maximum** playtime to help the user choose something suitable. For this option you should ask the user for a maximum playtime (in minutes or hours and minutes) and filter the items based on this maximum.
Please note that for this selection, a user could watch a single episode of a series, so please use the playtime listed in the file for your filtering.

Option 4:

Sometimes we have a bit of free time and want to have a proper binge. This option should restrict the items that are printed for option 1 based a **minimum** playtime to help the user choose something suitable. For this option you should ask the user for a minimum playtime (in minutes or hours and minutes) and filter the items based on this minimum.
Please note that for this selection, a user could watch multiple episodes of a series, so please use the total playtime of series for your filtering.

Option 5:

Not all content is suitable for everyone, therefore we have content ratings. As not everyone knows all the content ratings by heart, please allow the user to choose a content rating by offering a menu with the possibilities, **including the explanations below**, and filter the items that are printed for option 1 based on this choice.
The following content ratings are possible:

G: General Audiences
All ages admitted.

PG: Parental Guidance Suggested
Some material may not be suitable for children.

PG-13: Parents Strongly Cautioned
Some material may be inappropriate for children under 13.

R: Restricted
Under 17 requires accompanying parent or adult guardian.

NC-17: Adults Only
No one 17 and under admitted. Clearly adult. Children are not admitted.

Please note that content ratings have an ordering (from least restrictive to most restrictive)**.** This means that when a user selects the content rating PG-13, we would like to show any items that have that rating or a less restrictive rating (G, PG, PG-13), and filter out any items that have a more restrictive rating (R, NC-17).

Option 6:

This option should reset any restrictions that have been set through options 2, 3, 4 and 5 and restore the original list of items. Since reading from the file is an expensive operation, this option should not read from the file again.

Option 7:

The application stops.

Some important things to consider for this assignment:

- The program should **compile .**
- For a good grade, your program should also work well, without exceptions.
- Take care to have a nice programming style, in other words make use of code indentation, whitespaces, logical identifier names, etc. You can check this with **checkstyle**!
- You should provide Javadoc comments. Proper Javadoc comments include a clarification for each parameter as well as a short description of the method.
- Your program should have an **equals() & hashCode()** method for each class that is used in a non-static way.
- Think about your design: consider using *composition*, *inheritance*, *interfaces*, *enums*, *records*, etc. The textual information should be read into a class structure containing the right attributes to store the data (so storing all information in a single `String` is not allowed, because this would hinder further development).
- Ensure your program is properly **unit tested**. You should aim for **80% line coverage overall (it will likely be lower in the main class of your program), *and* at least the same number of test method as non-trivial methods.** Constructors, getters, and setters are considered trivial methods. *Any method that directly interacts with a file or user input (`System.in`) do not have to be tested.*
- The **filename items.txt or the absolute path should not be hardcoded** in your Java program. Please make sure to let the user provide it when starting the program (either as an explicit question to the user or as a program argument).

Handing in your work

To hand in your work you must create a **ZIP** file containing your project files. To do so, please go to the desktop and click the **"Exam Uploader"**. There you will have to fill in your student number (e.g. 5678901) and click "Upload Exam Data". When this process has completed successfully you can verify whether a zip file with your exam has appeared on the **P-drive**.

Good Luck!

Grade composition

1.4 points	**Compilation** ○ If your solution does not compile your final score = 1.
1.8 points	**Class Design** ○ Proper use of OOP principles. Additionally, there should be a good division of logic between classes & interfaces.
0.3 points	**equals() & hashCode() implementation** ○ Correct implementation of equals() & hashCode() in all classes that are part of your data model (any records are exempt).
1.8 points	**File reading** ○ Being able to read user-specified files, and parsing the information into Objects. *A partially functioning reader may still give some points.*
1.2 points	**Code style** ○ Ensure you have code that is readable. This includes (among others) clear naming, use of whitespaces, length and complexity of methods, Javadoc, etc.
0.5 points	**User Interface** ○ Having a well-working (looping) textual interface (including option 1 which prints the items in the console). *A partially functioning interface may still give some points.*
1.2 points	**JUnit tests** For full points, your project should have above **80% line coverage overall, *and* at least the same number of test method as non-trivial methods.** Constructors, getters, and setters are considered trivial methods. *Any method that directly interacts with a file or user input (`System.in`) does not have to be tested.* You are expected to test the filtering functionality in your program for full points, so take this into account in your design.
1.8 points	**Managing items in the catalogue** ○ *0.3 points* for being able to filter items on genre. ○ *0.3 points* for being able to filter items on a maximum play time. ○ *0.3 points* for being able to filter items on a minimum play time. ○ *0.6 points* for being able to filter items on a content rating properly. ○ *0.3 points* for being able to reset all filters and restore the original catalogue of items.

There is a 0.5 point penalty if you hardcode the filename.
There is a 0.5 point penalty if you do not hand in a proper (zip) archive.