CSE1100: Object Oriented Programming

T. Overklift / A. Zaidman

Date: 04-11-2020 Duration: 3 hours Pages: 5

Allowed resources

The following resources can be used during this exam:

- The slides, videos, assignments, tutorials and book pertaining to CSE1100.
- The Java API documentation.
- Existing internet resources (**if you copy a solution from the internet, copy the URL and put the URL in the Javadoc**)
- Any IDE that you like (e.g., IntelliJ), including its features to generate code like getters, equals methods, etc.

Disallowed resources

- Any information that you obtain from a person, either directly or indirectly.
- Actively asking / answering questions on online platforms.

→ Attempts at fraud will be reported to the Board of Examiners

About the exam

- Read the entire assignment and only then start implementing.
- You can find a scoring roster on the final page of this document.
- Hand in your program **before 12:00** (when working with regular time).
  - **Upload** a zip file containing your solution through WebLab.
  - Create your zip file **five minutes ahead of the deadline.**
  - Make sure it includes at least your source files (.java).
  - Name the zipfile **[studentnumber].zip**, e.g., 1234567.zip You can find your student number on your student card (7 digits; below your picture).
- Please, do not use specific packages but rather use the **default package** (this makes correcting the exam that much easier for us). if you do use a specific package, you will lose 1 point.
- Your project (sources and tests) **must compile** to get a passing grade. Sometimes your IDE will allow you to "proceed" even when some classes fail to compile. We will still see this as a compilation failure. Ensure that ALL classes you hand in compile.

Setting up a your project in IntelliJ

- Once IntelliJ has started go to File > New > Project
- Select an appropriate Java version (we recommend 11 or higher), and click next twice.
- Type a project name (the exact name doesn't matter).
- Click finish.
- Don't use packages other than the "default package".

Deezer music streaming

Deezer is looking into extending its free plans with streamable music. Deezer intends to develop a new app for Android and has asked you to develop a prototype in Java, which can later be extended to a full-fledged Android app for users and a backend for the Deezer employees. Deezer's business model resembles the freemium model, meaning that a free version is available but more limited than the (premium) paying version. Also, commercials are interleaved with the music in the free version.

You need to create a **prototype with a textual interface**. Instead of actually playing the music, your prototype needs to be able to create and print playlists to screen, as well as some other features.

The prototype application that you should develop reads in a manifest file that contains the albums, the songs in the albums and the ads. An example manifest file is shown below. The complete file **deezer.txt** is available through Weblab.

```
ALBUMS
ALBUM U2; Songs of Innocence; 2014
ARTISTS Larry Mullen Jr.; The Edge; Bono; Adam Clayton
TRACK 1; The Miracle; 4:15
TRACK 2; Every Breaking Wave; 4:12
TRACK 3; California; 4:00
TRACK 4; Song for Someone; 3:47
ALBUM Billie Eilish; When We All Fall Asleep Where Do We Go?; 2018
ARTISTS Billie Eilish; Finneas O'Connell
TRACK 1; Bad Guy; 3:14
TRACK 2; When the Party's Over; 3:16
ADS
AD ING Bank; 0:20; 0.05 euros
AD Bol.com; 0:15; 0.10 euros
AD Albert Heijn; 0:30; 0.20 euros
AD Specsavers; 0:15; 0.05 euros
AD Kruidvat; 0:10; 0.07 euros
```

The implicit meaning of the file being that there are 3 albums containing a varying number of songs. Each of the albums also has a list of artists that contributed to it. Each song of an album has a track number, a title, and length (minutes:seconds). Ads always appear after the albums near the bottom of the file. An advertisement has an advertiser (i.e., the company), a length (minutes:seconds), and a commercial price.

Deezer asks you to design and implement a program that:

- **Reads in** a file as specified by the user, for example the file deezer.txt. Reading in the file happens at start-up of the program.
- **Outputs** the album catalogue & playlists to the screen (see details below). The order of the playlist is dependent on the advanced options (see next bullet point).
- Offer advanced modes to **create randomized playlists and interleave ads**.
- Allows to **queue** a song to the front of a playlist.
- Allows to **add** a new ad.
- Allows to **write to file** all song information (preserving the file format!).
- Write an **`equals()`** method for each class (except for the class that contains the `main()` method)

To enable user interaction, please provide a **command line interface** with `System.out.*`. This interface should have the following menu:

```
   Please make your choice:
      1 – Print all albums & songs
      2 – Add a new advertisement
      3 – Enable shuffling of songs
      4 – Add song to front of playlist
      5 – Write to file
      6 – Calculate commercial value
      7 – Stop the program
```

Option 1: Print all albums & songs
This option prints all albums with all songs that are currently in the system in a neat format.
An example print is shown below. Please note: your format doesn't have to be exactly the same.

```
      Album: U2's Songs of Innocence
      Artists: Larry Mullen Jr., The Edge, Bono & Adam Clayton
            Track: The Miracle (4:15)
            Track: Every Breaking Wave (4:12)
      …
```

Option 2: Add a new advertisement
Through questions you ask the user to fill in all the necessary data for an advertisement (advertising company, duration & price) and add the new advertisement to the list of ads currently in the system.

Option 3: Create a randomized playlist (& print it)
If the user selects this option, a new random playlist is created. Implement this by creating a list that interleaves 10 songs & 10 ads that are randomly chosen from all songs and ads in the collection. This should result in a playlist that starts with a song, followed by and ad, followed by a song, then an ad, and so on. Duplicate songs and ads in the playlist are allowed.
An example of the printed playlist is shown below. Please note: your format doesn't have to be exactly the same.

```
      Track: The Miracle (4:15)
      AD: Bol.com (0:15)
      Track: Every Breaking Wave (4:12)
      AD: ABN Amro (0:30)
      …
```

Furthermore, creating this playlist can be computationally intensive, certainly if your music catalogue is big. That is why you should also try to create the playlist with a **thread** so that while the playlist is being created, other stuff can still be done. Make sure songs cannot be added to the playlist (option 4) while it is being created.
Hint: *Think carefully about how you can arrange **albums, songs** & **ads** in one or multiple datastructure(s), this choice can make large a difference when trying to create and shuffle a playlist!*
Hint: *If you cannot make sense of a random, implement option 3 while using all songs / ads sequentially as you've read them in.*
Hint: *Even if you are not able to implement the shuffling in the short period of time that you have, you can still implement the multi-threading. In that case, have a separate thread print* "`shuffling to be implemented`" *to the screen.*

Option 4: Add song to front of playlist (& print it)
Allow the user to provide the title of a song. If a track with this title exists in your catalogue, add this song to the beginning of the current playlist, and print the resulting playlist. You may assume song titles are unique. Remember to also add a (random) advertisement after the song! The format of the playlist you print should be similar to the one in the option 3.

If the song title the user provides does not match any song in your catalogue, inform the user. You do not need to change the playlist in this case.

Option 5: Write to file
Overwrite the existing file (that the user has specified at the start-up of the program) with the information that is currently in the system in exactly the same file format as the input. The information that you write to file should come from the data structure that you have created.

Option 6: Calculate commercial value
Using **functional programming**, determine the *commercial value* of the last playlist that was created (using option 3 or 4), by adding the individual commercial prices of the ads present in the current playlist. Print the price to screen, e.g., “3.10 euros”. If the playlist is empty (or doesn’t exist) simply print 0.00 euros.

Option 7: Quit application
The application stops.

Some important things to consider for this assignment:

- The program *and* the tests should **compile**
- Think about the usefulness of applying **inheritance and/or interfaces**.
- The **filename deezer.txt 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 “command line input”)
- Write unit tests (look at how your score for this exam is built up at the very end of this document)
- 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. Also provide javadoc comments.

Handing in your work

To hand in your work you must create a **ZIP** (so not .7z or anything else) file containing all your **.java**-files, you may also .zip your entire project folder.

Important: Rename the ZIP file to your student number, e.g. “`4567890.zip`”.

Double-check the correctness of the number using your campus card.

Finally, submit your .zip archive to Weblab!

Grade composition

2 points **Compilation**
- If your solution does not compile → final score = 1

1 point **Inheritance**
- Proper use of inheritance. Additionally there should be a good division of logic between classes & interfaces, as well as the proper use of (non-)access modifiers.

0.3 points **`equals()` implementation**
- Correct implementation of `equals()` in all classes that are part of your data model.

0.6 points **File reading**
- Being able to read the user-specified files, and parsing the information into Objects. *A partially functioning reader may still give some points.*

0.6 points **File writing**
- Being able to write the user-specified files, and parsing the information into Objects. *A partially functioning writer may still give some points.*

1 point **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 textual interface (including option 1, which prints the playlist to screen). *A partially functioning interface may still give some points.*

0.3 points **Adding new advertisement**
- For adding new ad by allowing the user to enter the details (via option 2).

1.5 points **JUnit tests**
- *0.5 points* for testing the class Song (depending on how well you test, you get a score between 0.0 and 0.5)
- *0.6 points* for testing all other classes & methods (except the `main()` method, and methods that depend on external resources such as files, you get a score between 0.0 and 0.6 depending on how well you test). Do not use files in your tests! (although you can create a String with (part of) the content of a file to test reading in…).
- *0.4 points* for testing with randomness, consider using a *seed (Don't forget to mention where you found the information – the URL!)*

0.5 points **Threads** for implementing threads
- *0.3 points* for implementing the thread correctly.
- *0.2 points* for correct use of synchronization.

0.7 point **Playlists (with randomness)**
- Implementing playlist shuffling with randomness (options 3).
- *0.4 points* for a working implementation.
- *0.3 points* for using proper randomness.

0.5 point **Queue a song to front of playlist**
- Find a song provided by the user in your catalogue and add it to the front playlist.

0.5 point **Functional Cost Calculation**
- Working option 6 which uses functional programming
- *0.2 points* maximum when not using functional programming.

There is a 1 point penalty if you do not work in the default package.
There is a 0.5 point penalty if you hardcode the filename.