DELFT UNIVERSITY OF TECHNOLOGY
Faculty of Electrical Engineering, Mathematics and Computer Science

TI1206 Object-Oriented Programming (computer exam)

January 11th, 2017, 18:30-21:30 (total duration: 3 hours)

Exam created by dr. A. Zaidman and checked by dr. G. Gousios

You can make use of the following during this exam:

- Java in Two Semesters (Charatan & Kans)
- The slides from the lectures, either printed on paper or available online through Blackboard + notes
- The Java API document (javadoc) that is available through Blackboard (Blackboard → TI1206 → Assignments → Java 1.8 API)

This exam contains **1 assignment (10 points)** (total exam: 5 pages).
→ the detailed scoring roster is on page 5

Log into the computer with the following:
Login: ewi_ti1200
Paswoord: Welkom01

HINT 1:	**Read the entire assignment and only then start implementing**
HINT 2:	**Look at the last page to get an overview of how your score is built up for this exam.**
HINT 3:	**It might be that jUnit 4.0 is not in your Eclipse build path by default. If you add a unit test case through the "New" jUnit wizard, then Eclipse will notice that jUnit is not in the build pat hand Eclipse will directly you tot he build path menu. Go to the "Libraries" tab, click "Add Library" and Eclipse will suggest to add jUnit.**

A few hints:

- Your program must **compile** (fail to compile == fail this exam)
- When your program is finished, use the 7Zip program to zip your files. Specifically, make a **zip file of your src folder** (the folder that contains your .java files) when your assignment is ready. Give the zip file the following name <studentnumber>.zip, so for example 12121212.zip. Please also put the inputfile into this zip. The class files are not necessary.
- The **workspace directory of Eclipse** is located on the H: drive. Use the 7Zip program to go to the H: drive to zip your assignment (**do not** try to zip your files through Windows Explorer, this might not work).

- 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.
- **Upload** the zip file containing your solution through Blackboard → TI1206 → Assignments → Exam January 9th, 2017 2nd, 2016. At that location, you will also find the inputfile that you should use for the exam assignment.
- All software present on the computer can be used.
- The **network has been disabled** so that you can only access Blackboard.
- **Mobile phones** remain in your backpack or coat and will not appear on the table. You should switch off your phone.
- If you make an attempt at fraud or commit **fraud**, you will be punished.

Spotify is looking into extending its free offering with downloadable music. Spotify intends to develop a new app for Android and iOs and has asked you to develop a prototype in Java, which can later be extended to a full-fledged Android app. Obviously, this product offering should be different from the paid offer to download music, that is why Spotify is exploring songs that you can listen to once, after which the song will be deleted. Advertisements are also a big part of Spotify's business model, so you will have to alternate songs and adds during play.

The prototype application that you should develop reads in a manifest file that contains the albums, the songs in the albums and the adds that are downloaded for offline consumption. An example manifest file is shown below:

```
CDS
CD U2, Songs of Innocence, 2014
SONG 1, The Miracle, 4:15
SONG 2, Every Breaking Wave, 4:12
SONG 3, California, 4:00
SONG 4, Song for Someone, 3:47
CD Coldplay, Parachutes, 2000
SONG 1, Don't Panic, 2:17
SONG 2, Shiver, 5:00
SONG 3, Spies, 5:19
SONG 4, Sparks, 3:47
SONG 5, Yellow, 4:27
ADDS
ADD ING Bank, 0:20
ADD Bol.com, 0:15
ADD Albert Heijn, 0:30
ADD Specsavers, 0:15
ADD Kruidvat, 0:10
ADD MediaMarkt.nl, 0:20
ADD KPN, 0:30
ADD TU Delft, 0:15
ADD ABN Amro, 0:30
ADD Rabobank, 0:15
```

The implicit meaning of the file being that 4 songs from the CD "Songs of Innocence" from U2 and released in 2014 are available on the app (i.e., have been downloaded and are ready to play). A second album containing another 5 songs is also downloaded, as are a bunch of adds (adds *always* appear after the CDs/songs). A song has a track number, a title and a length (minutes:seconds). An add has an advertiser (i.e. the company) and a length (minutes:seconds).

The complete file **spotify.txt** is available through Blackboard.

Spotify asks you to design and implement a program that:

- **Reads in** the file spotify.txt
- **Outputs** the playlist to the screen (see details below)
- **Offer** a shuffle mode (see details below)
- Allows to **add** a new CD containing songs (with all details such as year of release, length of songs, etc.
- 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 look like:

```
Please make your choice:
   1 - Show the current playlist
   2 – Add a new CD including songs
   3 – Play
   4 – Shuffle
   5 – Stop the program
```

Option 1
The playlist is shown on screen in the following format:

```
Album: U2’s Songs of Innocence
     Track The Miracle (4:15)
Next add: ING Bank (0:20)
Album: U2’s Songs of Innocence
     Track Every Breaking Wave (4:12)
Next add: Bol.com (0:15)
…
```

If you look closely, you see that the default playlist is a “merge” from the songs (in the same order that you read them from file) and the adds (also in order).

Hint: when reading in create two datastructures, one for the songs, one for the adds. Once everything has been read in, create a new datastructure that merges songs and adds. To perform this merge, iterate over all songs and after each song you add an advertisement. You stop when the songs are exhausted. If the advertisements should be exhausted before all your songs are exhausted, start from the first advertisement again.

Option 2
Through questions you ask the user to fill in all the necessary data for the CD and the songs in the CD. To ease reading in, you may ask the user to specify the number of songs he/she is going to enter.

After adding the CD+songs, add the songs to the playlist and introduce adds in between the songs (for ease you can start at the first add again).

Option 3
When the user selects play, the first song in the playlist is shown on screen, similarly to in Option 1. Also, the add that immediately follows the song is shown on screen (again similarly to Option 1).
Subsequently, the song and the add are removed from the playlist. Also remove the song from the datastructure containing the songs.

Option 4
Instead of taking the first song in the playlist, you need to **shuffle** the playlist. You should do this by using appropriate functionality in the **Collections** class. After shuffling the playlist, you take the two entries at the top and display those (similar to Option 3).
Subsequently, the top 2 entries are removed the playlist. Also remove the song from the datastructure containing the songs.

Hint 1: using any other way for shuffling will not give you points.
Hint 2: it might be that after shuffling songs and adds do not follow a nice pattern anymore (which is OK!)

Option 3&4
Both for option 3 and 4 there is one more step to take: all songs that have not been played yet should be written back to file. In addition, all adds, whether played or not should also be written back to file again.
Similarly to the original file, the CDs/songs should go first. Then the adds (mind you, all adds should be written back to file, also those that have been played).

Option 5
The application stops.

Some important things to consider for this assignment:
- Think about the usefulness of applying **inheritance**.
- When writing to the file spotify.txt, the old version of the file should be overwritten.
- The **filename spotify.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)

Other things to consider:
- 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. Also provide javadoc comments.

Furthermore, shuffling can be computationally intensive, certainly if your music catalogue is big. That is why you should also try to have the shuffling done with a **thread** so that while the catalogue is being shuffled, other stuff can still be done. Make sure new CDs/songs cannot be added during shuffling.

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 "Sorting to be implemented" to the screen.

Overview of the composition of your grade (total: 10)

- 2.5 for compilation; if it doesn't compile → final score = 1
- 1 for a well thought-out application of inheritance
    - Other criteria: are you using inheritance and polymorphism in the right way, is the functionality correctly distributed over the inheritance hierarchy
- 0.4 for correctly implementing equals() methods in all classes except for the class containing the main()
- 0.7 for implementing reading in a file (functioning code that leads to exception still gives part of the score)
- 0.7 for writing to a file (functioning code that leads to exception still gives part of the score)
- 0.5 for nicely styled code. Aspects being considered:
    - Length and complexity of methods
    - Length of parameter lists
    - Well-chosen identifier names
    - Whitespace, indentation, …
    - Javadoc
- 0.5 for adding CDs/songs
- 0.5 for a well-working textual interface (including option 1, which prints the playlist to screen)
- 0.4 for not hardcoding the filename
- 1.1 for jUnit tests
    - 0.5 for testing the class Song (depending on how well you test, you get a score between 0.0 and 0.5)
    - 0.6 for testing all other classes (except the class that contains main(), you also 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…)
- 1 for implementing threads
    - 0.5 for implementing the thread corrrectly
    - 0.5 for synchronization
- 0.7 for shuffling

There is a 1 point deduction if you do not work in the default package. So work in the default package!