IFT 2015 Data Structure **Professor François Major**

Deadline

You will have 3 weeks (December 5th, 2021 at 23h59) to submit your results on the studiUm. you can work within a group of 2 but you are allowed to work alone. You must submit with this format: TP2-Matricule1-Matricule2-tar.gz or TP2-Matricule1-Matricule2.zip. (other formats won't be accepted). Give only one Matricule if you are working alone.

Your MuzStream.java and playlist.java are important for your grade.

Grading

10% if your program complies without error

30% if your program runs without error on the input provided file

40% if your program produces the right results on the input provided file

20% for the quality of your code (comments and so on)

A penalty of 10% will be applied to you per day if you submit with a delay and it will be started from 00:00 of December 6th, 2021.

Example

Examples of inputs and outputs will be provided to your shortly

It will be important to respect the output syntax for the correction

Questions

If you have any questions, please post it on the TP2 forum on the studiUm

Description

You are developing a music streaming application, `MuzStream`. One of the features is a play queue (referred to later as the playlist), much like Spotify's, where the songs that have been queued will automatically play. The application must have the following behavior:

- The songs are considered as processes and are being played in an order determined by their priorities; i.e. the song with the highest priority is played out first, then the second highest priority, and so on;
- The priorities will be determined by integer values, where the lower the value the higher the priority;
- Songs with equal priorities are played out in any order;
- The playlist is initially filled with `playlistCapacity` (given to the program through one of its arguments) songs taken sequentially from the listeners' requests, which will be provided in an ASCII file (whose name is also given as an argument);
- When placed in the playlist, the priority of a song is 0;
- If the next song in the `requests` is already in the playlist, its priority increases by one; i.e. the integer value representing its priority is decremented by one;
- Songs are removed from the playlist as they are played. When the size of the playlist reaches `playlistLimit`, an integer given as an argument representing a percentage of the full capacity (e.g. 20 for 20%), it is refilled by songs from the requests file; following the above priority policy;
- The application will stop when the playlist is empty or after the playlist is filled `numberOFillings`, also an integer given as an argument, times;

Each time the playlist is filled, the application releases the TOP-`k`, that is the `k` songs played the most often since the application was launched, from number 1 the song

- that played the most, then the second most, and so on, to the kth; `k` is also given as an argument to the program.
- For each song in the TOP-`k`, you must report the average time interval between two plays of the song.

The program's call with its argument is as the following:

```
java MuzStream requests playlistCapacity playlistLimit numberOfFillings k
```

Input

`requests` (String) → the name of the file that contains the listeners' requests (PS. this String contains the full path of the file). The songs must be read sequentially from this file.

`playlistCapacity` (int) → the maximum number of songs that can be placed in the playlist

`playlistLimit` (int) → the percentage of songs in the playlist, which when reached triggers the refilling

`numberOfFillings` (int) → the number of times the playlist is refilled before the application stops (PS. the playlist will play until it reaches a load of `playlistLimit` percent for the last time after the last refill; which will also generate the last TOP-`k`; the last TOP-`k` is also generated when the playlist becomes empty; this happens when `numberOfFillings` has not been reached and there is no more songs in the playlist)

Listeners' requests file format

Each line in the requests contains the following information:

<artist><tab><title><tab><duration><newline>

where:

<artist> is a String

<title> is a String

<duration> is an int (which gives the duration of the song in seconds)

<tab> is the tabulation character

<newline> is the end-of-line/carriage return character.

Program call example

```
java MuzStream listenersRequest01 4 25 2 3
```

`listenersRequest01`:

ArtistA	Song0	300
ArtistB	Song1	500
ArtistC	Song2	450
ArtistA	Song0	300
ArtistA	Song3	250

Behavior

Initial playlist:

[(-1, (ArtistA, Song0, 300), (0, (ArtistB, Song1, 500), (ArtistC, Song2, 450), (ArtistA, Song3, 250)]

Playing ArtistA Song0 300

Playlist: [(0, (ArtistB, Song1, 500), (ArtistC, Song2, 450), (ArtistA, Song3, 250)]

Playing ArtistB Song1 500

Playlist: [(ArtistC, Song2, 450), (ArtistA, Song3, 250)]

Playing ArtistC Song2 450

Playlist: [(ArtistA, Song3, 250)]

Refill

Playlist: [(ArtistA, Song3, 250)]

Top3:

ArtistA	Song0	950
ArtistB	Song1	450
ArtistC	Song2	0

Playing ArtistA Song3

Playlist: []

Top3:

ArtistA	Song0	1200
ArtistB	Song1	700
ArtistC	Song2	250

N.B. The priority of ArtistA Song0 is -1 in the initial list because it was present the second time it was read from the requests file; it becomes the first song to be played. As each song has played only once in this situation, the average time between two plays becomes the time elapsed since the song has been played the first time (from its end).

Hint

You can use a sortable positional list for the TOP-`k`, similarly to what has already been done in the course. Do not forget to maintain the elapsed time since a song has been played.