CS 101 - Algorithms & Programming I

Fall 2022 - Lab 7

Due: Week of November 28, 2022

*Remember the **honor code** for your programming assignments.*
*For all labs, your solutions must conform to the CS101 style **guidelines**!*
All data and results should be stored in variables (or constants where appropriate) with meaningful names.

The objective of this lab is to learn how to define custom classes and use them. Remember that analyzing your problems and designing them on a piece of paper *before* starting implementation/coding is always a best practice. Specifically for this lab, you are to organize your data and methods working on them as classes.

0. Setup Workspace

Start VSC and open the previously created workspace named `labs_ws`. Now, under the `labs` folder, create a new folder named `lab7`.

In this lab, you are to create Java classes/files (under `labs/lab7` folder) as described below. We expect you to submit a total of 3 files, **without compressing them**, including: the `Track` class, a revised version of the `Track` class and a new class required by the revision (only 2 for Thursday's section as they do not need to revise the `Track` class). Do ***not* upload other/previous lab solutions** in your submission. Outputs of sample runs are shown as brown.

Your code must match precisely with the names and types of variables and interface of the methods specified as well as aligning nicely with the sample runs (both code and output).

1. Music Tracks

A music production company wants to build a new music streaming service called Hi-Music World (HMW) specifically for high fidelity sound. The collection in HMW is going to consist of songs or tracks recorded in high fidelity. For this task, HMW CEO Loudmann selected you as an ambitious programmer. You will start this task as a part of this lab assignment as specified below.

Create a new/empty file to define a class named `Track`. Your implementation will provide a simplified definition for a music track, which has the following data members:

Static Data Members:

- `count`: Counts the number of tracks *actively* available by HMW.
- `allCount`: Number of tracks registered to HMW from the beginning. (**Hint:** You can use this to assign a unique id to a track)

Instance Data Members:

- `ID`: An identification number that is unique for each track.
- `title`: Title/name of the track.
- `album`: The positive integer ID of the album it belongs to. If it's a single, then this value will be 0.
- `artist`: The positive integer ID of the singer or a music group.
- `releaseDate`: The release date of the track as a string in format "DD/MM/YYYY".

- `duration`: Total duration of the track in seconds (maximum value may be assumed as several thousands of seconds).
- `genre`: Genre of the track kept as a string of two characters as follows: “UN”: Unspecified (default unless set to some other value), “PO”: Popular, “RO”: Rock, “JA”: Jazz, and “OT”: Other.
- `isActive`: Data that holds whether the track is still available to HMW customers.

Methods:

- Constructor:
    - Takes the title, album, artist, release date and duration of the track and initializes the object accordingly,
    - Assigns a unique ID to it, and
    - Registers the track to the system (which modifies the counts accordingly).

- Accessor methods:
    - You need to implement methods to access the data variables of the track.

- Setter methods:
    - None of the instance data members specified during construction should be allowed to change; hence no setters for them. The setter for the genre needs to make sure the genre text is valid and set it to Unspecified, in case it isn’t.
    - `register()`: Registers a track to the system and modifies `count`, `allCount` accordingly.
    - `unregister()`: Removes a registered track from the system. Notice that you will need to modify `isActive` and `count` to simulate this.

- Service Methods:
    - `isBefore(Track anotherTrack)`: Often the system will need to compare tracks by their release dates. This method takes another instance of a `Track` class and returns true if the release date of the current player (`this.releaseDate`) is “smaller” (i.e., lexicographically before) than the release date of the specified input track. Otherwise, this method returns false. (**Hint:** You can use the `String.compareTo` method for this purpose; however you need to shuffle the release date as in this time format: “YYYY/MM/DD”). A sample run for this method is provided below.

Sample run:

```
Track t1 = new Track("Love over gold", 123, 234, "12/08/1982", (short)2473);
t1.setGenre("RO");
Track t2 = new Track("Gozleri aska gulen", 0, 98, "18/04/2018", (short)257);
t2.setGenre("PO");
Track t3 = new Track("Down to the waterline", 123, 234, "09/06/1978", (short)235);

System.out.println("Is t1 before t2? " + t1.isBefore(t2));
System.out.println("Is t2 before t3? " + t2.isBefore(t3));

Is t1 before t2? true
Is t2 before t3? false
```

2. Additional Functionality

Now that the basic functionality of the `Track` class is complete, you are going to add some additional functionality to your implementation. Once a track is registered, it will get played by the users increasing the play count of the track. To implement this, you will include some more data members to your class and provide service methods accordingly.

Furthermore, sometimes external software might keep track of the play count. In such cases, there will not be individual calls to the play method that increases the play method one by one but a “batch” one to increase it by the specified amount.

Hence, in your class definition you need to add the following members and functionality:

Instance Data Members:

- `playCount:` The number of times this track has been played by users of the system. This could be a rather large number.

Service Methods:

- `play():` Called each time a user plays this track. It increments the count by one. Should print out an error message in case it gets called on an inactive track as it’s an indication of a bug.
- `playBatch(int countToIncrement):` Called by an external software to notify a batch usage. The play count is incremented by the given count. It’s OK to call this on an inactive track since the usage might of the past before the track got inactive.
- `toString():` Returns a string representation that contains all the information stored about a track in a nicely formatted manner.

Sample run (assuming three instances of `Track` was already constructed as in part 1):

```
t1.play();
t3.play();
t3.play();
t1.playBatch(100);
t3.playBatch(10);
t1.play();
t2.playBatch(50);
t1.unregister();
t1.play();
System.out.println(t1.toString());
System.out.println(t2.toString());
System.out.println(t3.toString());

play() called on inactive track!
*************************************
Inactive track 1 among 3 tracks
Title:   Love over gold
Album:   123
Artist:  234
Release: 12/08/1982
Length:  41 min and 13 sec
Genre:   Rock
```

```
Played:  102
*************************************

*************************************
Active track 2 among 2 active tracks
Title:   Gozleri aska gulen
Album:   0
Artist:  98
Release: 18/04/2018
Length:  4 min and 17 sec
Genre:   Popular
Played:  50
*************************************

*************************************
Active track 3 among 2 active tracks
Title:   Down to the waterline
Album:   123
Artist:  234
Release: 09/06/1978
Length:  3 min and 55 sec
Genre:   Unspecified
Played:  12
*************************************
```

To test your class, you should use the code segments provided in sample runs. Keep in mind that your implementation should be compatible with the provided interfaces. For the desired outputs, you need to stay consistent with the details described by this document.