COMP1161

LAB2

PREAMBLE

This lab aims to reinforce the concepts of writing a simple Java class that uses aggregation. The specific skills to be covered are:

1. Using Aggregation
2. Implementing logic using selection and iteration
3. Using an ArrayList.

THE PROBLEM

The **Ministry** of Vybsie_Culcha intends to implement a grant for singers in order to offset difficulties experienced in the pandemic. The grant is to be funded from a fixed pool of funds (so the first set of singers that exhaust the funds are the only ones that are able to receive the grant). You are contracted to write a program that models the expected payouts. In order to reduce the possibility of fraud however, the ministry has decided to implement a (controversial) strategy that requires entertainers to prove they are the song owners by recording it at a **Studio**, even if already released. The **Ministry** maintains a list of **Studio**s, and code for the **Ministry** and **Studio** classes have already been fully implemented. The starting code also includes calls to a template of a **Singer** class that interacts with a **Song** class. Your task is to execute eight(8) tasks that complete the **Singer** and **Song** classes.

The HackerRank link to this exercise is at https://www.hackerrank.com/comp1161-lab2-23

MARKING CRITERIA (PLEASE REVIEW BEFORE STARTING)

Criterion	**Mark(s)**
Accessors and mutators for Singer and Song	1
Explanation of the importance of the toString method	1
Correct implementation of logic to sum items in the song ArrayList	1
Explanation of your logic for tryToRegisterSong()	2

NOTE ::: No part of Ministry or Studio should be modified.

Note that where convenient, and allowed, you may declare private (support) methods.

LAB EXERCISES

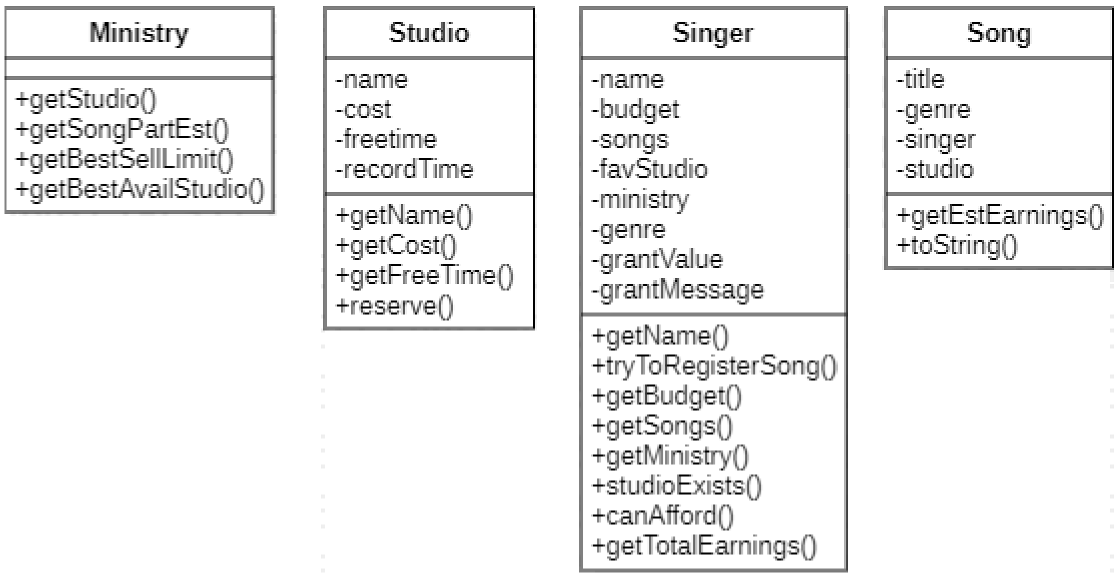

Figure 1:Simplified UML Class Diagram for the exercise

Figure 1 depicts the classes to which you will interface while completing the exercise.

1. In class **Singer**,:
   a. Complete the first constructor that accepts name, genre, budget, as well as a reference to the Ministry(incoming arguments include n for the name, g for genre, and b for budget).
   b. Implement the second constructor so that all instance variables are initialized.
2. In class **Song**, write the calculation in method getEstEarnings of class Song to evaluate estimated earnings. Note that while in the real world, other methods would be used to estimate earnings, in this simplified model, the estimated earnings will be the length of the title, multiplied by the a value extracted by the ministry using the method getSongPartEs*t(hint.. the singer keeps a reference to the ministry. You can observe how the reference is used in method getClaimableEarnings)*.
3. In class **Singer**, correct the method sumEstEarnings to evaluate the sum of estimates earnings for the singer.
4. Record a song in a studio and try to register the song
   a. In class **Song** write a mutator named setStudio(Studio studio) –method for the song that accepts a **Studio,** and sets the associated studio on the on the song to the referenced Studio.
   b. In class **Singer**, method tryToRegisterSong, if the singer has a preferred studio, set studio to the preferred studio (use method setStudio()), then add the song to the list of registered songs.
5. In clas**s Singer**, method tryToRegisterSong, update logic to check if artist can afford studio before setting the studio on the song.
6. In class **Singer**, method tryToRegisterSong, update logic to allow ministry to suggest a studio by calling the method getBestAvailableStudio(int budget, Studio preferred) from Ministry, which returns a suggested studio. If the method returns a studio (returned value is not null) then set the studio for the song to the returned studio if the singer is able to afford it. You can then check if the song has been connected with a studio, and if it is, add the song to the list of registered songs.
7. In class **Singer**, method tryToRegisterSong , ensure singer's budget is reduced by the cost of the studio.
8. In class **Singer**, method tryToRegisterSong , call the **Studio**'s reserve method when adding a song to the list of registered songs. Also, check to see if a studio is available (using the isAvailable() method in the studio class) before assigning the song to the studio.