**Software Workshop I**

**Assessment 2 - Mock**

**Marks available: 40**

**Date of assessment: Week 10 Mock**

**Set by: Jacqui Chetty**

**Background:**

You are tasked with completing a menu-driven application in Python using Object-Orientated-Programming. The application focuses on streaming music.

**Instructions:**

* There is a template project that needs to be downloaded from Canvas, found in Week 10 (labs). Make use of this to complete your solution. The template project consists of **musicStreamingTemplate.py**.
* Once you have completed your solution, zip your whole PyCharm project and upload it to Canvas (use the week 5 mock practice upload).
* When coding, you may add more functions, but you **must not remove existing ones**.
* You must not change any existing function header signatures or return types.
* The structure of each class is provided for you in the template as well as below described in the class descriptions.
* Familiarise yourself with the template.

**Assumptions:**

* You do not need to deal with case-sensitivity. Assume all inputs will be in the correct case.
* The system consists of 7 classes: **User**, **FreeUser**, **PremiumUser**, **Song, PlayList, Admin** and **Library**. You do not need to implement any extra classes.

**Task A – Option 1 [5 marks]**

This task will involve writing the functionality to create a user. A user has a userId and a name. Users do not have passwords.

Code for this task should be written in code labelled:

"""

Task A - Insert code here

"""

**Background:**

* You have been given the code to ask the user to enter a userId and their name.
* The first character of the userId must either be a ‘F’ for freeUser or a ‘P’ for preimiumUser.

**Instructions:**

1. If a userId start with ‘F’ then create a FreeUser() object, otherwise ‘P’ for the creation of a PremiumUser() object.
2. Call add\_user() on the Admin class and pass user to this method.
3. Implement the add\_user() in the Admin class where a user is added to the users {}. The users {} contains the userId and the name of the user.

**Task B – Option 2: Add a song to the library [3 marks]**

This task will involve writing the code for the add\_song\_library() method.

Code for this task(s) should be written in code labelled:

"""

Task B - Insert code here

"""

**Background:**

* In main() the prompts are provided for you.

**Instructions:**

1. Create a song object of the Song class and pass the appropriate variables to the constructor of the Song class.
2. Call add\_song\_library() and implement the code by adding the song to the song [].

**Task C – Option 3: Create a Playlist and Add Songs [6 marks]**

This task will involve writing the code to allow users to create a playlist as well as add songs to the playlist.

Code for this task will involve writing code in multiple areas. These areas are all labelled:

"""

Task C - Insert code here

"""

**Background:**

* The code to verify if a user exists is provided for you.
* The prompts to get a playlist\_name is provided for you.
* The code also verifies if the playlist already exists in the Admin class and if it does an appropriate message is displayed. the playlist object is created for the existing playlist name.
* Once a playlist exists a song must be added. The code for iterating through all the added songs is provided for you.
* The prompt to the user asking the song number (listed as part of the iteration) is provided to you.

**Instructions:**

1. If no playlist exists create a playlist object and pass the playlist\_name to the constructor of the Playlist class.
2. Implement the add\_playlist() in the Admin class by adding the playlist to the playlists {}.
3. Now that a playlist exists add a song by verifying the number entered is between 1 and the end of the number of songs in the library.
4. Implement the add\_song() method in the Playlist class.

**Task D – Option 4: Play a song [7 marks]**

This task will involve writing the code to allow users to play a song.

Code for this task will involve writing code in multiple areas.

These areas are all labelled:

"""

Task D - Insert code here

"""

**Background:**

* The code to verify if a user exists is provided for you.
* The code to verify if a playlist exists is provided for you.

**Instructions:**

* + - 1. Iterate through the playlist and list the songs numbering them 1 through x where x is the last song in the playlist.
      2. Prompt the user to enter a song\_choice.
      3. Implement the play\_song method in the User class.
      4. The prompt is provided for you. Add the song to the play\_history [].
      5. Call the increment\_song\_count() method in the Song class to increment the count.

**Task E – Option 5: View the playlist** **[9 marks]**

This task will involve writing the code to view all the songs in the playlist.

Code for this task will involve writing code in multiple areas. These areas are all labelled:

"""

Task E - Insert code here

"""

**Background:**

* The prompt to ask the user to enter a playlist is provided for you.

**Instructions:**

1. Call the view\_playlists() method in the Admin class so that the user can view the playlist names.
2. Implement the view\_playlists() method so that the available playlist names are listed.
3. Use the get() to retrieve the playlist name from playlists in the Admin class.
4. Store the playlist name in a variable called playlist.
5. If the playlist exist call the view\_sings() method and implement the view\_songs() in the Playlist class.
6. If the playlist cannot be found print an appropriate message.

**Task F – Option 6: View user history** **[5 marks]**

This task will involve writing the code to view the songs that the user has been playing.

Code for this task will involve writing code in multiple areas. These areas are all labelled:

"""

Task F - Insert code here

"""

**Background:**

* The prompt to ask the user to enter their uiserID is provided for you.

**Instructions:**

* 1. Use the get() to retrieve the userID from admin.user.
  2. If the user exists call the view\_history() method in the User class and implement the code for this method.
  3. If a use cannot be found print an appropriate message.

**Tasks G and H: Options 7 and 8: Admin view all songs and view all users [5 marks]**

This task will involve writing the code to view the songs in the library as well as view all users.

**Background:**

* The method calls for each option are provided for you.
* A library object has been created in main().

**Instructions:**

* 1. Implement the view\_library()method by printing all the songs that are in the library.
  2. Implement the view\_users() method in the Admin class, making use of the users {} to print each user’s name.

**Class Descriptions:**

**User Class:**

**Instance Attributes:**

* *user\_id*: **str** - The ID of the user.
* *name*: **str** - The name of the user.
* *Is\_premium: bool – set to False for a* ***FreeUser*** *or True if a* ***PremiumUser***.
* *play\_history*: **list** - A list of all the songs played by a user.

**Methods:**

* **play\_song(self, song)**
  1. Append a song title to the play\_history [].
  2. Call increment\_song\_count() in the Song class.
* **View\_history(self)** 
  1. Iterate the play\_history [] and print each song.

**FreeUser** and **PremiumUser** Classes inherits from User – all code is provided to you.

**Song Class:**

**Instance Attributes:**

* *song\_id*: **str** – unique id of a song.
* *title*: **str** – title of a song.
* *title*: **str** – title of a song.
* *artist*: **str** – artist of a song.
* play\_count: **int** – counts the number of times a song is played.

**Methods:**

* **increment\_song\_count(self, vehicle)**
  1. Increments the play\_count by 1.

**Playlists Class:**

**Instance Attributes:**

* *name*: **str** – The name of the song.
* *songs*: **list** – A lsit of all the songs in the playlist.

**Methods:**

* **add\_song(self, song)**
  + 1. Adds a song to the songs [].
* **view\_songs(self)**
  + 1. Lists the songs in the songs [].

**Admin Class:**

**Instance Attributes:**

* *admin\_id*: **str** - The admin ID.
* *name*: **str** - The name of the user.
* *users*: **dict** – The users are stored here.
* *library*: **obj** (**Library** class) – The library object created in main().
* *playlists*: dict – A playlist is added to the dictionary, consisting of a playlist name.

**Methods:**

* **add\_user(self, user)**
  1. A user is added to the users {}.
* **view\_user(self)**
  1. All users are printed.

1. **add\_playlist(self, playlist)**
   1. A playlist is added to the playlists {}.
2. **View\_playlists(self)**
   1. All playlist names are printed.

**Library Class:**

**Instance Attributes:**

* *songs*: **list** – The songs are added to the library.

**Methods:**

* **add\_song\_library(self, song)**

1. Print the song by title and artist if the song exists.
2. If it does not exist add the song to the songs[] and print an appropriate message.

* **view\_library(self)**

1. Print a message if no songs exist in the library.
2. If there are songs print each song by title and artist.

**Note:**

1. Not all methods have been included as part of the question set out above.
2. It is a mock so play around with it until you are feeling comfortable implementing code.
3. I have included more completed code than what would be in a “real” test. This is to support your learning. Take some of the code out, try it on your own.