INTRODUCTION

In your role as a software developer, you have been contracted by QUB's Music Department to create a Java application capable of managing digital music (or sound) files for a Jukebox system. Specifically, you are required to build and test a prototype system that meets the deliverables outlined in the sections below.

GENERAL INSTRUCTIONS - CODE IMPLEMENTATION

1. Create a new Java project named 'Assessment2'.
2. Add three packages to 'Assessment2' named part01, part02 and part03.
3. Implement your code as described below:
    - 'part01' should contain all code associated with the core requirements.
    - 'part02' should contain the code related to implementation of additional features only, the only exception being the console application which you should copy from part01 to part02. Use import statements as required to access the definitions from part01.
    - 'part03' should contain any code relating to testing

SUBMISSION DATE/TIME – FRIDAY 26TH MARCH 2021 BY 5 PM.

PART 1: CORE FUNCTIONALITY (50 MARKS AVAILABLE)

To demonstrate that the proposed Java application is feasible, the following features must be implemented.

1.1 Implementation of the object classes, enumerator and interface shown in Figure 1 below. (You may leave the ***Jukebox*** class to Part 2.)

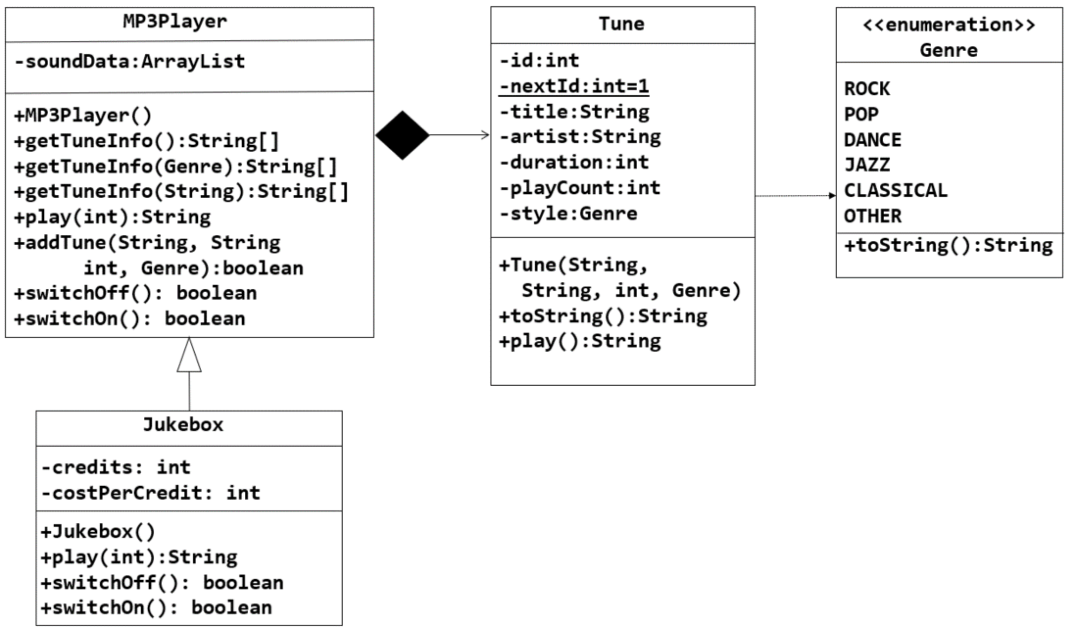

Figure 1: A (partial) UML Class Diagram for QUB Music Management Software.

Further details on the role and behaviour of the above classes follows – please read the entire document before starting. Canvas will also contain a list of frequently asked questions (FAQ) – a useful resource for clarification purposes.

Interfaces for ***MP3Player*** and ***Tune*** are available on Canvas to help you structure these classes. These will be discussed in class and must be used.

1.2 A console (menu-based) application to manage (through an ***MP3Player*** instance) the interaction with ***Tune*** instances within the system. At this stage of development, there is no need to consider the ***Jukebox*** class. The following menu options should be provided:

- ***Select from Full List***: the user should be able to view summary details of ***all Tune instances*** in the system (listed one after another and ordered by title) and select one to be played (by id). If a selected ***Tune*** can be played, the application should report the details of the Tune with a message to say that it has been played (or an error)
- ***Select Tune by Artist***: the user should be able to view summary details of Tune instances (ordered by title) for a ***specified artist*** and select one (by id) to be played. If a selected ***Tune*** can be played, the application should report the details of the Tune with a message to say that it has been played (or an error).
- ***Select Tune by Genre***: the user should be able to view summary details of Tune instances (ordered by title) for a ***specified genre*** and select one (by id) to be played. If a selected ***Tune*** can be played, the application should report the details of the Tune with a message to say that it has been played (or an error).
- ***Add New Tune***: the user should be able to add a new tune to the ***MP3Player*** instance. The same tune (as specified by title, artist, genre and duration) can't be added more than once.
- ***Display the Top 10***: user should be able to view a list of the top 10 tunes (names and number of plays) in order of number of plays.
- ***Switch Off***: the user should be able to turn the MP3Player off.
- ***Switch On***: the user should be able to turn the MP3Player on.
- ***Exit***: the user should be able to exit the system.

Notes:

A. The console application should create a ***MP3Player*** instance.

B. The console application should create and add a number (at least 5) of tunes to the ***MP3Player*** instance for testing purposes.

C. When adding a new tune, the user should be prompted for the title, artist, duration, style (Genre).

D. When selecting from a full list of tunes, the user should be able to view the details of all available tunes within the ***MP3Player*** and be prompted to pick one for play using its (displayed) tune id.

E. When selecting a tune by artist, the user should be able to select an artist from a list and then able to view the details of all available tunes for that selection and be prompted to pick one for play using its (displayed) tune id.

F. When selecting a tune by genre, the user should be able to select a genre from a list and then able to view the details of all available tunes for that selection and be prompted to pick one for play using its (displayed) tune id.

G. When switching an ***MP3Player*** off, this can only be done if it is already on. When switching an ***MP3Player*** on, this can only be done if it is already off.

H. For enumeration ***Genre***, the ***toString***() method should return a String corresponding to the current value:

Value	***String***
ROCK	"Rock and Roll"
POP	"Easy Listening Pop"
DANCE	"Techno Dance"
JAZZ	"Smooth Jazz"
CLASSICAL	"Classical"
OTHER	"Unknown Genre"

I. You should make use of the ***Menu*** class defined in Menu.java (from the assessment specification on Canvas).

J. While implementing, you will find opportunities to use ***searching and sorting algorithms***. You must base any solution for these algorithms on the ***material presented in the lectures***. (i.e. you must implement yourself)

K. You ***may not*** add additional public methods to ***MP3Player.*** For class ***Tune***, you may add getters/setters as appropriate. You may add private attributes and methods to all classes, however, these must be justified (use comments).

Marks awarded will be based on the functionality and quality of the application.

PART 2: ADDITIONAL FEATURES (30 MARKS AVAILABLE)

Further to the features outlined above, QUB's Music Department is interested in evaluating any additional functionality that may be of benefit to the application. Accordingly, within the time and resource constraints of the project, you have been asked to investigate and (where possible) implement additional functionality:

Adding the Jukebox class

1. Copy the application and ***Menu*** classes from part01 to part02 – no other files should be copied. Use import statements to reference definitions in part01.
2. For the console application, assign the ***MP3Player*** reference to an Instance of ***Jukebox***.
3. Create a new class, ***Jukebox*** which extends the behaviour of ***MP3Player***. Implement as per the UML specification.
   - Add a default constructor – should set **credits** and **costPerCredit** to 0.
   - Add a method called ***insertCoin*** to the definition of ***Jukebox*** – this method should take a single integer parameter representing the value of a coin (in pence). Only 10p, 20p, 50p, 1 pound and 2 pound coins are allowed. If ***costPerCredit*** is at 0, this means that it's free to play a tune. The credits attribute should be modified only if ***costPerCredit*** is above 0.
   - Add a mutator for ***costPerCredit*** to the definition of ***Jukebox***.
   - Override the ***play*** method, which should return the same message as ***play*** in the superclass if there are enough ***credits*** in a ***Jukebox*** (1 credit per play if ***costPerCredit*** is above 0 and free play otherwise) and a suitable error message if ***play*** is not possible (due to lack of credits).
   - Override the ***switchOff*** and ***switchOn*** methods – ***switchOff*** (if allowed) should store the state of a ***Jukebox*** to a CSV file; ***switchOn*** (if allowed) should restore the state of a ***Jukebox*** from a CSV file
4. Update the console application to allow the user to purchase credits for a ***Jukebox*** instance.

Notes

1. A sample data file (CSV format) should be provided as part of your submission – avoid using absolute file paths.
2. Each piece of additional functionality in line with those described above will be awarded marks based on the quality of the functionality provided.
3. Use of serialization for storing/retrieving data is not permitted – data must be written to and read from a CSV file, using the techniques described in the practical classes.

To help QUB review any additional features implemented, include a short Microsoft Word document with your submission that provides a title and short description (max 2-3 sentences) for each feature.

NOTE: Part 1, Part 2 and Part 3 should be submitted as separate packages as described in the general instructions.

PART 3: TESTING THE APPLICATION (20 MARKS)

As part of QUBs quality assurance procedure, you are required to submit evidence that the application has been thoroughly tested. This should take the form of a completed testing document. The scope of this testing should cover all areas of core functionality outlined in **Part 1** of this assignment specification. Testing documentation must be submitted using the template provided by QUB, available on Canvas.

CLASS INFORMATION

MP3Player Attributes and Methods:

soundData – list of ***Tune*** objects managed by an ***MP3Player***

getTuneInfo() – should return and array of Strings where each String contains full details of a Tune, ordered by Title and covering ***all available*** Tune objects. The method should return ***null*** if no Tune object are available

getTuneInfo(String artist) – should return an array (as above) for Tune objects by a specific ***artist***, specified by the String parameter. Again, the data should be ordered by Title. The method should return ***null*** if no Tune objects are available.

getTuneInfo(Genre gen) – should return an array (as above) for Tune objects for a specific ***genre***, specified by the String parameter. Again, the data should be ordered by Title. The method should return ***null*** if no Tune objects are available.

play(int tuneId) – using the integer parameter (representing the ***id*** of a Tune) should identify a Tune and 'play' it (call the play method for a tune). The method should return ***null*** If it does not exist or the result of calling the play method for a Tune if it does exist.

addTune(String title, String artist, int duration, Genre gen) – using the integer parameter (representing the ***id*** of a Tune) this method should create and add a new Tune object to the ***soundData*** ArrayList if it's not already present. Two Tune objects are the same if the have the same title, artist, duration and genre.

switchOff – this method will request that an MP3Player be turned off (if not already off). An MP3Player which is off will not permit the above behaviour. Should return a boolean.

switchOn – this method will request that an MP3Player be turned on (if it's not already on). Should return a boolean.

Tune Attributes and Methods:

id – a unique Tune identifier

nextId – next usable Tune identifier

title – Tune title

artist – Tune artist

duration – duration (in seconds) of the Tune

style – track genre

constructor – parameters: title, artist, duration, style in that order

play – this method should return a String in the following format:

"Now Playing … Four Seasons Winter, by Vivaldi"

where "Four Seasons Winter" is the title of a Tune by artist "Vivaldi"

toString()- returns a formatted String including all details of a Tune in ***single line***.

Jukebox Attributes and Methods:

credits – number of credits in a Jukebox – one credit for 1 play

costPerCredit – cost (in pence) of a single credit – a zero value will indicate that Tues are free to play

play(int tuneId) – a Tune can be played if there are enough credits in the Jukebox or costPerCredit is 0 (i.e. free play). Should return a String (or null) in line with the play method in MP3Player

switchOff – will turn the Jukebox off (if not already off), saving its state to a CSV file

switchOn – will turn the Jukebox on (if not already on), restoring its state from a CSV file

Submission Instructions

1. Create a new folder to hold all your assignment files for submission. The folder should be suitably named and include your student number.
2. Copy and paste the ‘part01’, ‘part02’ and ‘part03’ (for testing) folders from your java project into the folder created in step 1. ENSURE that the ‘part01’, ‘part02’ and ‘part03’ folders contain all the corresponding .java files.
3. Add your testing spreadsheet to the folder created in step 1. Make sure to include your test code (for Part 3)
4. Include any data files created for Part 2.
5. Add the Microsoft Word document describing any additional features to the folder created in step 1.
6. Verify that you have all the required files included in the folder created in step 1.
7. Compress the folder to a zip file and upload it to the assignment location in the CSC1029 module on Canvas by **5.00pm** on **Friday 26th March 2021.**

NOTE: Please check that you have included the original development Java (.java) files. If you submit .class files these will NOT be marked. These submissions will be date-stamped and in accordance with University regulations, late submissions will be penalised. **Failure to follow the submission instructions may result in a reduced or zero mark for part or all of the assignment!**