Curtin University — Computing Discipline

Object Oriented Program Design (COMP1001)

Semester 2, 2018

Assignment

Due: Stage 1: The week starting Monday October 1, at the start of your practical
Stage 2: The week starting Monday October 15, at the start of your practical
Stage 3: The week starting Monday October 29, at the start of your practical

Weight: 20% of the unit mark.

1 Introduction

This assignment is submitted in three stages (see the end of the specification for details on each stage). Each stage will take significant time to implement. You cannot finish any stage in a single night or weekend! You will need to plan your time carefully and work on each bit as the topic is covered.

Your task is to create a music media library. The library will be able to perform the following:

- Store details of music stored on Records, Cassettes, and Digitally;
- Read the library details from an input file;
- Add songs to the library based on user input;
- Calculate the total play time of all songs stored in the library;
- Find a specific song (based on first occurrence name matches) and print the play instructions for that song.
- Allow the user to select a media type and option, and then print number of songs and total run time for that type:
  - Records, and the record play speed.
  - Digital, and the file type.
  - Cassette, and the tape side.
- Save the library to a file.

2 Task Details

The final functionality of the music library is as follows:

2.1 Data Validation:

The details of all music must conform to the following specifications.

- All songs have the following pieces of information:
  Name – must not be null or an empty string.
  Artist – must not be null or an empty string.
  Duration – must be a real number between 0.0 and 9.59 (inclusive). This must be a valid `<MMM>.<SS>` time, where MMM is the minutes, and SS is a 2 digit seconds, if the seconds are more than 2 digits, they should be truncated to 2 digits. ie: 9.87 is invalid, 9.599 should be truncated to 9.59 which is valid.
- Records have the following additional information:
  Track Number – Must be an integer between 0 and 20 (inclusive).
  Play Speed – Must be one of the following strings, casing does not matter:
  - "33 1/3 RPM"
  - "45 RPM"
  - "78 RPM"
- Digital tracks have the following additional information:
  Type – Must be one of the following strings, casing does not matter:
  - "wav"
  - "mp3"
  - "acc"
- Cassettes have the following additional information:
  Track Number – Must be an integer between 0 and 20 (inclusive).
  Start Time – Must be a real number between 0.0 and 160.0 (inclusive). This must be a valid `<MMM>.<SS>` time, with the same format and truncation rules as duration. ie: 5.89 is invalid.
  Side – Must be a character, A, or B, lowercase, or uppercase.

2.2 Add Music:

Add songs to the library based on user input, the same validation as the previous section applies.

2.3 Total Duration:

Calculate the total play time of all songs stored in the library;

2.4 Play Instructions

The user should be able to see the play instructions for each song. An example for each of the 3 music types is shows below.

```
Zero to Hero is track number 3 and the record is be played at 33 1/3 rpm.
Make a Man Out of You requires the wav codec to play.
Be Out Guest is track number 2, it starts at 2.30 minutes, on side A.
```

2.5 Search:

Find a specific song (based on first occurrence name matches) and print the play instructions for that song. This is a simple linear search, loop through each array, and print the instructions for the first (*only* the first) match. Matches should ignore case.

2.5.1 Search & Duration:

Allow the user to select a media type and option, and then print the number of songs and total run time for that type:

- Records, and the record play speed.
- Digital, and the file type.
- Cassette, and the tape side.

2.6 Input & Output

Input and output should be done using the techniques from the lectures and practical sessions.

2.6.1 Input File

You will be required to load the music library from a csv file (the name of which is supplied by the user). For each music type the csv file will conform to the following format:

Record – `R,<Name>,<Artist>,<Duration>,<Track Number>,<Play Speed>`
Digital – `D,<Name>,<Artist>,<Duration>,<File Type>`
Cassette – `C,<Name>,<Artist>,<Duration>,<Track Number>,<Start Time>,<Side>`

The first letter of each row must be R, D, or C (in any case), this letter indicates the type of music the row is for. Below is an example input:

```
R,Zero to Hero,The Muses,4.37,3,33 1/3 RPM
D,Make a Man Out of You,Mulan and Chang,3.57,WAV
C,Be Our Guest,The Dishes,2.57,2,2.3,a
```

Any malformed rows should be discarded when reading the file and an appropriate message displayed to the user informing them of such, the remainder of the file should continue to be read. The file itself may also be empty, your algorithm should check for this and print an appropriate error message.

2.6.2 Output File

Your program must be able to save the current music library, to a file that conforms to the same format as the input file. That is, the generated output file must be able to be used on the next run of the program to "restore" the library to the same state. All real numbers should be formatted to 2 decimal places.

2.6.3 User Input

All user input must be validated where appropriate.

2.7 User Output

All output to the user should be formatted in a pleasant, easy to read manner. This include prompts for input. All real numbers should be formatted to 2 decimal places.

2.8 Error Handling

Under no circumstance should the user be aware of exception handling, this includes having unhandled exceptions, and the user seeing any form of a stack trace.

All error messages displayed to the user should be meaningful to a non-programmer.

3 Documentation

You must thoroughly document your code using block comments (`/*...*/`) and in-line comments (`//...`). For each Class, you must place a comment at the top of the file containing your name, the file name, the purpose of the class and any modification history. For each method you write you must place a comment immediately above it explaining its purpose, it's import and export values, and how it works. Other comments must be included as necessary.

4 Submission Requirements

Electronic copies of all assignment documents must be kept in your Curtin user accounts, and not edited after the due date as the assignment will be tested in your tutorial sessions. These should be in a directory called `OOPDAssignmentStage<X>`, located immediately under your home directory. Failure to do this will result in 0 marks being allocated to the testing portion of the marking.

Submit each stage of your assignment electronically, via Blackboard (lms.curtin.edu.au), before the deadline.

Each stage should be submitted as a single .tar.gz file containing:

A declaration of originality – whether scanned or photographed or filled-in electronically, but in any case *complete*.

Your implementation – including all your pseudo code, java code, test files, readme, etc …
Do not submit your .class files.

Your report – a single PDF report, as outlined for each stage. The specified length of the report is a guideline! If your report fails to address any of the specified topics you will be penalised (regardless of length). **Your report must also be submitted to your tutor, in hard copy, within the first 5 minutes of your practical.**

Note: do not use .zip, .zipx, .rar, .7z, etc, they will be taken as **non-submissions** and instantly receive zero (0) for the whole assignment.

You are responsible for ensuring that your submission is correct and not corrupted. You may make multiple submissions, but only your latest submission will be marked. You should download your submission, extract the contents and ensure it still compiles/runs. If your submission is corrupted you will receive a mark of **zero (0)** with no chance of resubmission.

Please note that assessments submitted in the Discipline of Computing must comply with the Discipline of Computing Coding Standard, which can also be found on Blackboard. All code must be fully documented.

5 Extensions & Late Assessments

No extensions will be granted. If exceptional circumstances prevent you from submitting an assignment on time, contact the Unit Coordinator ASAP with relevant documentation. After the due date and time is too late.

Late submissions will incur the penalties laid out in the Unit Outline. Anything after the due date is considered a late submission, even if only a minute late.

6 Demonstration

You will be required to demonstrate each stage of your assignment in your registered practical session.If you cannot attend your registered practical session you must make arrangements with the lecturer by the date indicated below.

Failure to demonstrate any stage of the assignment during your registered practical will result in a mark of ZERO (0) for the entire assignment. During each demonstration you will be required to answer questions about your design. If you cannot answer the questions satisfactorily, you will receive a mark of zero for that stage.

Stage	Week	Date (starting)	Alt. arrangement deadline
Stage 1	10	01/10/2018	24/09/2018
Stage 2	12	15/10/2018	08/10/2018
Stage 3	14	29/10/2018	22/10/2018

7 Staged Submission and Marking

You must submit each stage. Certain aspects will only be marked in the associated stage.

Stage 1 (30 marks) – the user interface, file input, data validation.

Stage 2 (45 marks) – file output, play time(total and custom), printing the play instructions, adding songs, and basic object orientation.

Stage 3(25 marks) – correctly using a superclass, some of the marks will overlap with previous functionality, these marks are for the OO side of it, not for implementing the functionality itself.

The pseudocode algorithm and Java implementation will both be marked. Your java code will be assessed by its suitability as a valid implementation of your supplied pseudocode. The demonstration requirements for each submission are described above. Marks will be allocated for the appropriate choices of classes, the overall functionality, and adhering to best programming practices as discussed in the lectures and practicals.

Signoffs:

You cannot pass the assignment without attending your weekly tutorial and attempting your signoff questions. This assignment will be weight by the average of your signoff marks according to the table below:

Once your mark has been calculated, the practical signoff result will be applied as follows:

Your total practical signoffs will be awarded a mark out of 10, which will have the following effect (multiplier) on the assignment mark.

Total	Effect
8 - 10	100%
7 - 7.9	90%
6 - 6.9	80%
5 - 5.9	70%
3 - 4.9	50%
0 - 2.9	30%

8 Academic Misconduct – Plagiarism and Collusion

You must read the AcademicIntegrityInCoding pdf and abide by it or you will risk academic misconduct

This is an assessable task, and as such there are strict rules. You must not ask for or accept help from *anyone* else on completing the tasks. You must not show your work to another student enrolled in this unit who might gain unfair advantage from it.

These things are considered **plagiarism** or **collusion**.

Staff can provide assistance in helping you understand the unit material in general, but nobody is allowed to help you solve the specific problems posed in this document. The purpose of the assignment is for *you* to solve them *on your own*.

Please see Curtin's Academic Integrity website for information on academic misconduct (which includes plagiarism and collusion).

The unit coordinator may require you to provide an oral justification of, or to answer questions about, any piece of written work submitted in this unit. Your response(s) may be referred to as evidence in an Academic Misconduct inquiry. In addition, your assignment submission may be analysed by Turnitin and/or other systems to detect plagiarism and/or collusion.

Submission Stage 1 - Menu, Input, and Validation

The first submission requires you complete the following:

Menu looping: All menu/submenu options must be present, for options that will not be implemented until stage 2 a message in the following format should be printed.
``<Selected Option> is not yet implemented''

File Input: Your program must read and validate the contents of specified file.
The details of *valid* lines in the file should be passed to the relevant stub methods, as described in section below.
This stage of the assignment requires that at most 10 of each music type be accepted.

Data Validation: All data input to your program (by the user or a file) must be validated. All validation should be in appropriate submodules.

OutputFile: The output file mechanics should be set up in this stage. As the program does not store any data at this point, the following should be printed, on individual lines.

```
Data storage not yet implemented.
This is a test file.
To ensure the output is correctly printed.
```

Note: you should use 3 separate print statements to produce the output in order to ensure your file output is functioning correctly. If you combine them into one statement you may be unaware of some issues, and have to spend more time in stage 2 fixing the issue.

8.1 Stub Code

The following methods are available in the `MusicStub` class. After reading each entry from the input file, you should pass the valid details to the following methods. If any details are invalid errors messages will be printed to the terminal.

These stub method calls will be replaced with object creation in Stage 2.

```
/**************************************************************************
* SUBMODULE: addRecord                                                    *
* IMPORTS: name (String), artist (String), duration (Real),               *
*          trackNumber (Integer), playSpeed (String)                      *
* EXPORT: none                                                            *
* PURPOSE: A stub for the functionality of adding a new record.           *
* BEHAVIOUR: If provided valid details the method will print a            *
*            success message.                                             *
*            If any import is invalid the method will print  an error     *
*            message.                                                     *
*            If too many records have been added, an error message        *
*            will be printed.                                             *
**************************************************************************/
```

```
/*************************************************************************
* SUBMODULE: addDigital                                                  *
* IMPORTS: name (String), artist (String), duration (Real),              *
*          type (String)                                                 *
* EXPORT: none                                                           *
* PURPOSE: A stub for the functionality of adding a digital track        *
* BEHAVIOUR: If provided valid details the method will print a           *
*            success message.                                            *
*            If any import is invalid the method will print  an error    *
*            message.                                                    *
*            If too many digital tracks have been added, an error        *
*            message will be printed.                                    *
*************************************************************************/

/*************************************************************************
* SUBMODULE: addCassette                                                 *
* IMPORTS: name (String), artist (String), duration (Real),              *
*          trackNumber (Integer), start (Real), side (Character)         *
* EXPORT: none                                                           *
* PURPOSE: A stub for the functionality of adding a new cassette.        *
* BEHAVIOUR: If provided valid details the method will print a           *
*            success message.                                            *
*            If any import is invalid the method will print  an          *
*            error message.                                              *
*            If too many records have been added, an error message       *
*            will be printed.                                            *
*************************************************************************/
```

The test harness for these methods is available in `MusicStubTest.java`

8.2 Java File Structure

Your code for this stage should be broken into the following files.

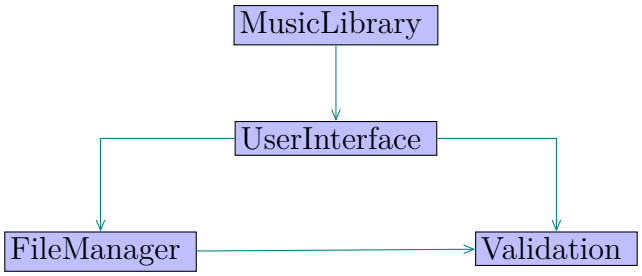
MusicLibrary – where the main method is located. Your main should only have the high-level last resort exception handling, and should call the menu from the User-Interface class.

UserInterface – This class should have all the user I/O. Including validation for said I/O.

FileManager - This class should have all the file I/O methods.

Validation – All the validation needed for records, cassettes, and digital music. In the second submission these submodules will move to different locations. The methods are in a separate class as they will be required from several locations.

The class structure should look similar to the image below. You have a lot of leeway with the exact methods involved. The diagram is just showing which classes will need to call methods in which other classes.



8.3 Report

Your submission must include a half page report. In the report you must discuss the following:

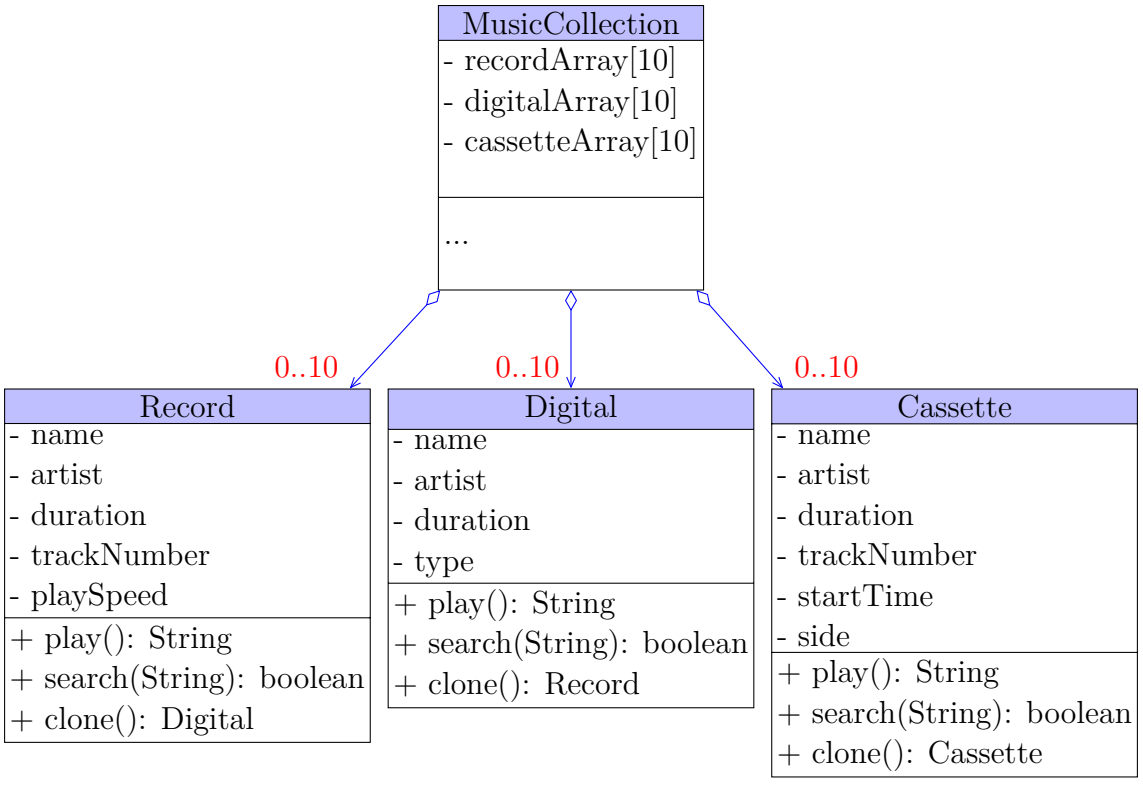
- Justify your menu implementation decisions.
- Justify your approach to data validation.
- Discuss any challenges you had in your implementation/design.

Submission Stage 2 - Objects and Association

This stage of the assignment requires you add basic object orientation. This means:

- All functionality should now be implemented.
- You should have 3 arrays, one of each object type. The max number of each object type should be 10.
- You will need to introduce some form of *controller class* that manages the functionality. Both the controller and user interface class should have no static methods ie: you must create objects of them.
- The storage facility (arrays) should only be constructed once. It must initially be constructed based on file input ie: you cannot perform any operations on the library until it is constructed. Subsequent attempts to construct the library should fail and an appropriate error message output to the user ie: you cannot read another file.
- To have the search method take a String, you will need to treat the cassette side search criteria as a string (this will be vital for the next stage).

The basic UML for the container classes is shown below. You will require all the accessors and mutators as emphasised in the lectures and practicals (these are omitted below for simplicity). The methods in MusicCollection have also been left for you to decide.



8.4 Report

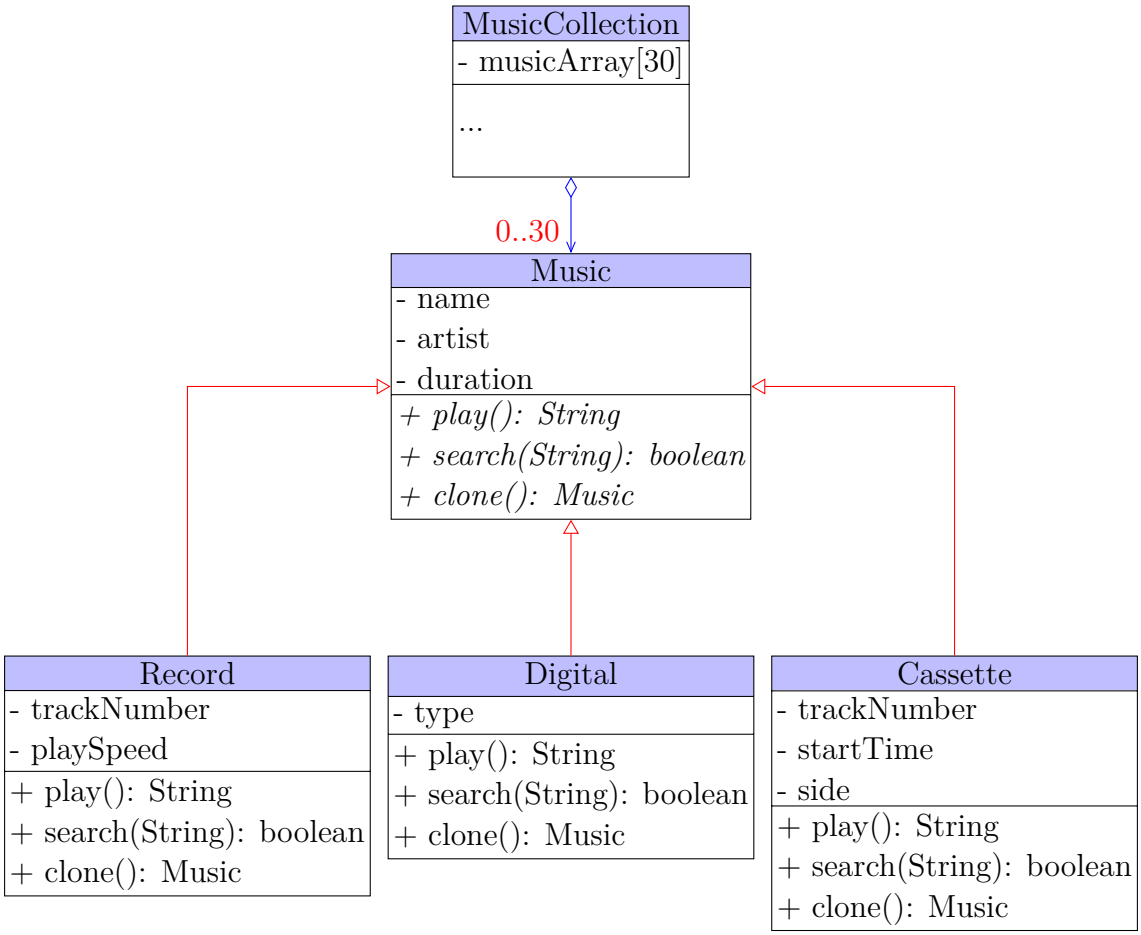
Your submission must include a half page report. In the report you must discuss the following:

- Justify your design decisions in regards to what functionality was placed in the container classes, and what functionality was placed elsewhere.
- Discuss what issues you had to over come when using an array of objects.
- Discuss any difficulties with managing 3 separate arrays,and how it would be simpler to have a single array.

Submission Stage 3 - Inheritance

This stage of the assignment introduces inheritance to the design. This means:

- You should have a single array of the superclass. The max number of objects stored in this array should be 30, the objects may be of any type in any order.
- You should only need to refer to the exact subclass when *initially* constructing an object.
- You will need to make good use of abstract methods (in the diagram below, an abstract method is represented with italic text) .



8.5 Report

Your submission must include a half page report. In the report you must discuss the following:

- Discuss how the addition of inheritance simplified your design, especially in regards to the array/s.
- Discuss any complications inheritance introduced in your design.
- Discuss and justify any down casting present in our code.

End of Assignment