CPS209 Assignment 2:

Extended MyAudio App

Due Date: Friday April 14 11:59pm

Late Policy: 10% off per day, not accepted more than 3 days late.

You are to extend the **MyAudio** application program from assignment 1. This programming assignment will increase your knowledge of Java Collections, File I/O and Exceptions, objects and classes, inheritance, and interfaces. **You must do this assignment alone – no groups. Do not attempt to find source code on the web for this assignment. It will not help you and you risk extremely serious consequences. Your program will be checked for plagiarism.** Begin designing and programming early! This assignment is worth 10 percent of your mark. **If there is some part of the assignment you do not understand, please email your professor as soon as possible and they will clarify the issue.**

NOTE: some parts of the assignment are using java concepts (specifically Maps) that we have not yet studied in the lectures. You may want to wait until this material is covered in the lectures before attempting this part of the assignment.

Program Functionality Requirements:

We will add some advanced features to assignment 1. **Below is a description of the new functionality for your assignment and some new methods and modifications to the existing functionality as well as modifications to the user interface.** NOTE: **You are encouraged to** extend and modify your own assignment 1 (A1) code and keep the classes essentially as they were except for changes and additions specified below. You are permitted to add other instance variables and other methods if you think they are necessary.

IMPORTANT!!: We will be releasing a solution to assignment 1 on April 1 or thereabouts (once everyone has submitted assignment 1). You are permitted to use this solution as the basis for your assignment 2 if you choose, or you may build upon your own solution to assignment.

1. **MyAudioUI:** Add 5 new actions (commands): **SEARCH, SEARCHA, SEARCHG, DOWNLOADA, DOWNLOADG** and 1 modified action **DOWNLOAD**. The new actions take the following arguments:
    - **SEARCH:** Add an action called SEARCH to myAudioUI as well as necessary code to class AudioContentStore that searches the **store** for an audio content with the specified **title**. That is, the user types SEARCH then is prompted to enter a title string. If the audio content with this title is found in the store then print the index of this content and the info for this content. A separate part of this new functionality is to use a **Map** in class AudioContentStore that maps a title (string) to an integer value. The integer value represents an index into the contents array list. See Requirement 3 below for details on the Map part of the assignment. The SEARCH functionality can be done without a Map but you will not receive full marks for this part.

- **SEARCHA:** Add an action called SEARCHA to myAudioUI as well as necessary code to class AudioContentStore that searches the store for an audio content with the specified artist name. That is, the user types SEARCHA then is prompted to enter an artist string. If the audio content with this artist name is found in the store then print the indices and info of all audio content with this artist (use author string for audio books). A separate part of this new functionality is to use a **Map** in class AudioContentStore that maps an artist string to an ***array list of integer***. The integers in the array list represent indices into the contents array list. See Requirement 3 below for details on the Map part of the assignment. The SEARCHA functionality can be done without a Map but you will not receive marks for the Map part.

- **SEARCHG:** Add an action called SEARCHG to myAudioUI as well as necessary code to class AudioContentStore that searches the store for a song with the specified genre ("POP" "ROCK" etc). That is, the user types SEARCHG then is prompted to enter a genre string. If the song with this genre is found in the store then print the indices and info of all songs with this genre. A separate part of this new functionality is to use a **Map** in class AudioContentStore that maps a genre string to an ***array list of integer***. The integers in the array list represent indices into the contents array list. See Requirement 3 below for details on the Map part of the assignment. The SEARCHG functionality can be done without a Map but you will not receive marks for the Map part.

- **DOWNLOAD:** Modify the download action so that it takes two store indices instead of one store index as parameters – i.e. a *fromIndex* and a *toIndex*. That is, you should now be able to download a range of songs/books etc. from the store (e.g. from song 2 to song 6, inclusive). If some of the songs are already in the library, then and error message for each of these songs should be printed (see the video)

- **DOWNLOADA:** Create a new download action that takes an artist string as parameter and downloads all audio content with this artist name (use author for audio books) from the store. Make use of the artist map for full marks in the Map part.

- **DOWNLOADG:** Create a new download action that takes a genre string as parameter and downloads all songs in this genre from the store. Make use of the genre map.

2. **Library:** the following is a list of additions and changes required for the *Library* class:

   a. Replace the use of the error message variable ***errMsg*** and the boolean return types for the methods in class Library with exceptions. That is, if an exception occurs (for example, in method **playSong(int index)** a song is not

found in the library)then **throw a new exception**. Create custom exception classes (for example: class **AudioContentNotFoundException**) by extending class **RuntimeException** (see the Exception lecture slides for examples of this). Place these new exception classes in the Library.java file at the bottom of the file, outside of the Library class. Note: this means return for the public methods in class Library can now be changed to **void. Update the code accordingly in class MyAudioUI.**

b. Catch the thrown exceptions in class **MyAudioUI**. It might be convenient to place the **catch**() {…} code block(s) together in one part of the file and use a single **try** {…} that surrounds the if statements inside the while loop – it is your design choice. The catch() block(s) should print any message contained in the exception object to the screen and then continue the loop waiting for the next user input. Note: there may be some places where you need a separate try{} catch{} to surround a specific action (e.g. download)

c. **File I/O**: replace the code in the constructor method of class AudioContentStore (that currently creates a fixed set of AudioContent objects) with code that reads the audio content information from a file called **store.txt**. Create appropriate AudioContent objects from the file information and add them to the contents array list. You may use the posted file store.txt. This file has a specific fixed format for each audio content. A song has the following format: The first line is the keyword SONG. The 2^{nd} – 8^{th} lines contain the strings id, title, year, length, artist, composer, genre. The 9^{th} line contains the number of lines of lyrics. Then the actual lyrics lines follow. An audio book has the following format: The first line is the keyword AUDIOBOOK. The 2^{nd} – 7^{th} lines contain the strings id, title, year, length, author, narrator. The 8^{th} line contains the number of chapters. This is followed by the chapter titles. After the chapter titles is a line containing the number of lines of a chapter, followed by the chapter lines. This is repeated for each chapter. See the posted store.txt to help you understand the format above. The Podcast class is not included in assignment 2.

Note: we suggest you create a private method in AudioContentStore that reads the store.txt file and returns an array list of AudioContent objects. Inside the AudioContentStore constructor method, call this private method, surrounded with try{…} catch(IOException e) {…}. If an IOException occurs in the private method, print out the IOException message and exit program using System.exit(1).

3. **Map**: Use a map to create a fast search capability that allows you to search the store for audio content titles. Given a title string, search the store to find a song/audiobook by that title. That is, the map in the AudioContentStore maps a title string (the key) to an integer index value. This integer represents the index for the song/book in the contents array list. Create another map that uses the artist (string) as a key and maps to an integer array list(rather than a single integer). The integers in the array list represent indices into the contents array list. Use artist for song and author for audio book as the key. Create a third map that uses the genre (string) of a song as a key and maps to an integer array list(rather than a single integer). The integers in the array list represent indices into the contents array list. You must build the three maps in the constructor method of class AudioContentStore after the contents array list has been filled up from the store.txt file.

4. **BONUS:** Create a new action **SEARCHP**. This action supports the ability to search (and print) all audio content that partially matches a target string. The target string could appear anywhere in the audio content (in the title, artist, lyrics, chapter etc etc). For example, the target string could be "Ale" and the search would match all the songs by Alessia Cara. If the target string is "Ishmael" then the search would match the book Moby Dick.

Grading (Out of 10 marks)

Note: Highest Grade Possible: 11 out of 10

File I/O and exception handling for reading the store.txt file and creating AudioContent objects and storing in contents array list	2.5 marks
Good use of customized exceptions instead of **errMsg** variable. Exceptions thrown and caught properly. Removal of boolean return types in methods of class Library	2 marks
New actions DOWNLOADA, DOWNLOADG and updated DOWNLOAD action to support index range of audio content to download	1 mark for DOWNLOADA and DOWNLOADG 0.5 marks for updated DOWNLOAD
A - SEARCH action for title B - Correct use of a map to support fast search (and building of Map)	1 mark 1 mark
A - SEARCHA action for artist B - Correct use of a map to support fast search (and building of Map)	0.5 mark 0.5 mark
A - SEARCHG action for genre B - Correct use of a map to support fast search (and building of Map)	0.5 mark 0.5 mark
Bonus SEARCHP for audio content matching a partial string	1 mark
Penalty for insufficient comments or bad program structure or bad use of exceptions, maps etc.	Up to -1 marks

Submitting Your Assignment: READ CAREFULLY!!

- Use D2L to submit your assignment.
- Inside each of your ".java" files have your name and student id as comments at the top of the file. Make sure your files can be compiled as is without any further modification!!
- Include a README.txt file that describes what aspects of your program works and what doesn't work (or partially works). If your program does not compile, state this!! You will still get part marks. The TA will use this file as a guide and make fewer marking mistakes.

- **Do not include ".class" files!!!!**
- **Do not use any Package keyword in your java files!!!**
- Once everything is ready, submit it.