Project 2: Jukebox Hero

CS121 Computer Science I

Overview

Novel Entertainment Systems (NES) is entering a new arena for their entertainment products… Music. NES has been contracted to build a music analytics tool called Jukebox Hero. The customer will provide a music catalog in the form of a plain text file. Each row in the file represents a single song with the following comma separated values (csv): Artist, Album, Title, Duration (in secs).

The Jukebox Hero tool will allow the user to open a music catalog file, print stats about the catalog (number of songs, number of artists, number of albums, total play time), search for songs in the catalog, and display the entire catalog.

Objectives

- Use the ArrayList class to manage both Song and String objects
- Use for-each loops to process data stored in ArrayLists
- Use while loop with keyboard Scanner to process and validate user input selections
- Use while loop with file Scanner to read lines from text file
- Use String Scanner to process fields (tokens) from CSV (Comma Separated Values) formatted data

Getting Started

- Create a new folder named "JukeboxHero"
- Download the project support files and import them into the project (p2-jukeboxhero-support.zip contains the files listed below).
    - Song.java
    - jimmy_buffett-collection.csv
    - music-collection.csv
    - grader.sh
- Create a class named JukeboxHero as a driver for the project that provides the menu structure and console UI (**U**ser **I**nterface).

Project Specification

Quality Requirements (20 points)

1. Project Submission
    - Code submitted to the correct cs121 section and with **only** the required files
2. `README.md`
    - `README.md` file is named properly and has the required sections including your reflection (see Documentation section)
3. Javadoc Comments
    - Project should have proper Javadoc formatted comments (see Documentation section)
4. Coding Style
    - Code should be properly *indented* and have consistent vertical spacing
    - Variable and Method names should follow the *camelCase* naming convention
    - Class names should follow the *TitleCase* naming convention
    - Constants should be in all *CAPITAL_LETTERS*
    - In-line comments to describe what different code sections do

Code Requirements (80 points)

Part 1: Selection Menu Loop

The Selection Menu Loop is the main user interface for the Jukebox Hero project. Each of the successive parts will build out the functionality of the Jukebox Hero tool. When the program starts, it should first display a menu and then enter a while loop to display command prompt similar to the following:

```
*****************************
*       Program Menu        *
*****************************
(L)oad catalog
(S)earch catalog
(A)nalyse catalog
(P)rint catalog
(Q)uit

Please enter a command (press 'm' for Menu):
```

A Scanner attached to `System.in` should be used to read a selection from the user. This Scanner should be created at the start of the program, used for all keyboard input throughout the program, and closed when the program exits. For the sake of clarity, we'll refer to this as a keyboard `Scanner`. A switch statement should be used to process the command selected by the user. If a user enters an invalid command, simply display an invalid selection message in the console and display the command prompt again:

```
Invalid selection!
Please enter a command (press m for Menu):
```

The menu should initially be displayed before the program enters the selection loop. Only when the user presses 'm' should the menu be printed to the console again. When the user presses 'q' the selection loop exits and a goodbye message is displayed before the program quits.

```
Please enter a command (press 'm' for Menu): m

*****************************
*       Program Menu        *
*****************************
(L)oad catalog
(S)earch catalog
(A)nalyse catalog
(P)rint catalog
(Q)uit

Please enter a command (press 'm' for Menu): q
```

```
Goodbye!
```

NOTE: All menu commands should be case insensitive. (IE: 'S' and 's' should both search catalog)

Part 2: Load Catalog and Print Catalog

To implement part 2, a data structure will be necessary to store potentially 100s of Song objects. To accomplish this, use an ArrayList of Song objects to hold all the songs that are read in from the catalog files. Use the following to create a new songList of Song objects:

```
ArrayList<Song> songList = new ArrayList<Song>();
```

In order for the songList to have the necessary scope and data lifetime, it should be declared and instantiated outside (and above) the menu selection while loop.

When the user selects the (L)oad catalog option from the menu, use the keyboard `Scanner` (`System.in`) to prompt the user for the catalog file to load. You should reuse the keyboard `Scanner` created in Part 1 to get the file name.

NOTE: You should have one and only one keyboard Scanner in your application, created at the start of the program and closed at the end. You may see strange results if you attempt to create multiple Scanners connected to System.in and your program will not work with the grader.sh script that will be used to test your code.

Once you have a String containing the catalog filename, create a new File Scanner to open the catalog file and use a while loop to read it line-by-line. The catalog file contains **c**omma **s**eparated/delimited **v**alues (CSV) formatted data with each line representing a single song. The song data on each row is formatted as follows: Artist, Album, Title, Duration (in secs).

This is an example line from a catalog file :
```
Jimmy Buffett,Songs You Know by Heart,Cheeseburger in Paradise,172
```

Be sure to enclose your File Scanner and any code that depends upon it inside of a try/catch block. The program should cleanly handle the situation where a user specifies a filename that does not exist, display a message to the user, and display the selection menu prompt again without modifying the songList.

```
Please enter a command (press 'm' for Menu): l
Load Catalog...
Please enter filename: wrongfilename.csv
```

```
Unable to open file: wrongfilename.csv

Please enter a command (press 'm' for Menu):
```

Each time a new catalog is loaded, it should replace (not append to) the songList. Before loading songs from the catalog file into the songList, use the songLists's `clear()` method to remove the current songs from the list.

For each line in the catalog file, create a new String Scanner to parse out the individual values for artist, album, title and duration. In order to access these fields as individual tokens using the String scanner, the delimiter will need to be changed from the default whitespace character to a comma. Once the new String scanner has been created, use its `useDelimiter()` method to change the delimiter to a comma as shown below:

```
myStringScanner.useDelimiter(“,”);
```

With the delimiter changed to a comma, now each call to the `next()` method will return the fields "artist", "album", "title", and "duration" respectively. You may assume that there are no errors in the csv data and that there will always be these four fields.

Now use the values from these fields to create a new Song object and add it to your ArrayList of Song objects named *songList*.

```
Please enter a command (press 'm' for Menu): l
Load Catalog...
Please enter filename: jimmy_buffett-collection.csv
Successfully loaded 23 songs!
```

When the user selects the (P)rint catalog option from the menu, use a foreach loop to iterate through each element in the songList and use the toString() method, defined in the Song class, to display the song details in the console. The output should first display the number of songs in the songList followed by a divider. Then the list of songs should be displayed.

```
Please enter a command (press 'm' for Menu): p
Song list contains 23 songs...
---------------------------------
Woman Goin' Crazy On Caroli... Jimmy Buffett       Havana Daydreamin'                250
My Head Hurts My Feet Stink... Jimmy Buffett       Havana Daydreamin'                155
The Captain and the Kid        Jimmy Buffett       Havana Daydreamin'                198
Big Rig                        Jimmy Buffett       Havana Daydreamin'                211
Defying Gravity                Jimmy Buffett       Havana Daydreamin'                163
  ...
```

Part 3: Catalog Search

When the user selects (S)earch from the menu, use the keyboard Scanner to prompt the user for their search query. Then use a for loop to walk through each song in the songList and check the **song title** to see if it contains the search term. If it does, add the song object to a second ArrayList<Song> called searchResults.

NOTE: The search should be case insensitive meaning that searching for "cheese" will match a song that contains "Cheese".

HINT: String objects have a contains() method that is extremely handy for checking whether one string is contained within another. When searching for matches, this will be more useful than the equals() method because contains() would find "cheese" in "cheeseburger" while equals() would not.

After checking all the songs in the songList for matches, print the number of songs that matched, then use a for loop to print out all the songs in the searchResults list. The following is an example use case:

```
Please enter a command (press 'm' for Menu): s
Search Catalog...
Please enter the search query: cheese
Found 1 matches
---------------------------------
Cheeseburger in Paradise       Jimmy Buffett       Songs You Know by Heart          172
```

Part 4: Catalog Analysis

The analysis option displays the number of songs, number of unique artists, number of unique albums and the total playtime of the album. The following is an example use case:

```
Please enter a command (press 'm' for Menu): a
Catalog Analysis...
        Number of Artists: 1
        Number of Albums: 2
        Number of Songs: 23
        Catalog Playtime: 4542
```

Use an ArrayList of String objects for both the artistList and the albumList. Only add an artist or album to their respective list if it is not already in the list. *(Do not correct for case in album and artist analysis to get results matching grader script results.)* The Number of Artists and Number of Albums can be determined by checking the number of elements in each list.

HINT: ArrayList objects have a contains() method that is extremely handy for checking whether the ArrayList contains an object that is equal to the specified object. The contains() method walks through the ArrayList and calls the equals() method on each item in the list and checks it against the specified object. It returns true if it finds a match.

Testing

Once you have completed your program, copy your **Song.java**, **JukeboxHero.java** and **README.md** files into a new directory on the onyx server (if you did the project on your computer). Also copy in the **grader.sh** script included in the project support files zip.

- Manually check program output against sample output above (Parts 1-4) for the *jimmy_buffett-collection.csv* catalog
- Manually check program output against the following for the *music-collection.csv* catalog

```
Welcome to Jukebox Hero,
    bringing order to playlists one song at a time...

*****************************
*      Program Menu         *
*****************************
(L)oad catalog
(S)earch catalog
(A)nalyse catalog
(P)rint catalog
(Q)uit

Please enter a command (press 'm' for Menu): l
Load Catalog...
Please enter filename: music-collection.csv
Successfully loaded 1125 songs!

Please enter a command (press 'm' for Menu): s
Search Catalog...
Please enter the search query: cheese
Found 2 matches
---------------------------------
Cheese Cake                    Aerosmith           Night In The Ruts               255
Cheeseburger in Paradise       Jimmy Buffett       Songs You Know by Heart         172

Please enter a command (press 'm' for Menu): a
Catalog Analysis...
        Number of Artists: 9
        Number of Albums: 112
        Number of Songs: 1125
        Catalog Playtime: 284550

Please enter a command (press 'm' for Menu): q
Goodbye!
```

- The program should not crash when invalid input is supplied by the user
    - Part 1: Test that invalid menu selections display warnings to the user and show the prompt again.
    - Part 2: Test that a missing file or empty filename does not cause the program to crash.

    - Part 3: Test that search for a term that does not exist in the catalog, such as "icecream", shows zero matches and does not crash
    - All: Test that operations performed on an empty catalog do not cause the program to crash

- Verify that your program is ready to compile and be tested using the grader.sh script. This will show whether your files are named correctly and take in input as required. This is the same script that will be used for grading, so your program must run with this script!
    - Set permission for the script to be executable with `$ chmod +x grader.sh`
    - Run the script with `$ ./grader.sh`

Extra Credit (5 points)

Once the requirements for this project have been completed and thoroughly tested the extra credit may be attempted. Begin by creating a copy of your **JukeboxHero.java** solution named **JukeboxHeroEC.java**. Both of these files must be submitted to receive credit for this project

The base solution packs all the program logic into the individual cases of the menu switch statement. This approach can make the code more difficult to read as well as lead to coding mistakes because it is difficult to keep track of which code-block is currently active. A better approach is to implement private helper methods. However, because we have not covered how to create our own classes yet, these methods will need to be static methods.

Specification

Create the following private static helper methods to implement the functionality behind each of the menu options and call them from the associated case within the menu switch statement.

- `private static void displayMenu()`
- `private static void loadCatalog(ArrayList<Song> sl, Scanner kbd)`
- `private static void printCatalog(ArrayList<Song> sl)`
- `private static void searchCatalog(ArrayList<Song> sl, Scanner kbd)`
- `private static void analyseCatalog(ArrayList<Song> sl)`

NOTE: You should not create any static variables (aka class variables) to implement this extra credit. All required components (song list and keyboard scanner) should be passed in as parameters to the methods.

Documentation

If you haven't already done so, add **javadoc comments** to each class and all methods.

- Javadoc *class* comments should include both a short description of the class AND and @author YOURNAME tag.
  - The class javadoc comment is placed immediately above the class header.
  - Your class comment must include the @author tag at the end of the comment. This will list you as the author of your software when you create your documentation.
- Javadoc *method* comments should be placed before ALL methods.
  - Methods that are defined by an interface do not require additional comments since they are provided within the interface specification.

Include a plain-text file called **README.md** that describes your program and how to use it. Expected formatting and content are described in README_TEMPLATE.md. See README_EXAMPLE.md for an example.