Project 1: PlayList Analyzer

CS121 Computer Science I

Overview

In this assignment, you will develop an application that manages a user's songs in a simplified playlist.

Objectives

- Write a class with a main method (PlayList).
- Use an existing, custom class (Song).
- Use classes from the Java standard library.
- Work with numeric data.
- Work with standard input and output.

Getting Started

- Create a new VS Code folder for this assignment and import `Song.java` into your project.
- Create a new class called `PlayList` and add a `main` method. This file should be named `PlayList.java`.
- Read the Song documentation and (if you are feeling adventurous) look through the `Song.java` file to familiarize yourself with what it contains and how it works before writing any code of your own.
- Start implementing your `PlayList` class according to the specifications.
- Test often! Run your program after each task so you can find and fix problems early. It is really hard for anyone to track down problems if the code was not tested along the way.

Development Strategy

Before writing a single line of code, read the specifications listed below and make sure you understand all of the requirements. Work through any necessary calculations/logic on paper or a whiteboard before writing any code.

Project Specification

You will write a class called `PlayList`. It is the driver class, meaning, it will contain the `main` method. In the `main` method, you will gather information about **three songs** from the user by prompting and reading data from the command line. Then, you will store the song information by creating song objects from the given data. You will use the methods available in the `Song` class to extract data from the song objects, calculate statistics for song play times, and print the information to the user. Finally, you will use conditional statements to order songs by play time. The ordered "playlist" will be printed to the user.

Your `PlayList` code should not duplicate any data or functionality that belongs to the `Song` objects. Make sure that you are accessing data using the methods provided by the `Song` class.

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*
    - 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)

1. Read Input and Create Song Objects

IMPORTANT: Do NOT use a loop, ArrayLists, or arrays to read/create your Songs. We will discuss how to do this later, but please don't over complicate things now.

- Create a new `Scanner` object to read from `System.in`.
    - **Important**: You should never create more than one `Scanner` that reads from `System.in` in any program. Creating more than one `Scanner` that reads from `System.in` crashes the script used to test the program. There is only one keyboard; there should only be one Scanner connected to it.

- Prompt the user for the following values and read the user's input using your `Scanner`. Save the input values from the user into temporary variables. You can use the `nextLine()` method of the `Scanner` class to read each line of input.

    - A String for the song *title*.
    - A String for the song *artist*.
    - A String for the song *album.*
    - A String for the song *play time*.

    You should have only one input variable for each input (i.e. only one title variable, one artist variable, etc.). **Reuse** these variables when reading input for all three songs. Once input is complete, these variables should never be used again in your program. All data will be retrieved from Song get methods for all calculations and output.

    A sample input session for one song is shown below. Your program **must** read input in the order below. If it doesn't, it will fail our tests when grading.

```
Enter title: Big Jet Plane
Enter artist: Julia Stone
Enter album: Down the Way
Enter play time (mm:ss): 3:54
```

- Convert the play time to an integer representing the total number of seconds.
    - The play time will be entered in the format mm:ss, where mm is the number of minutes and ss is the number of seconds. You will need to convert this string value to the total number of seconds before you can store it in your

Song object. (Hint 1: use the String class's `indexOf()` method to find the index of the ':' character, then use the `substring()` method to get the minutes and seconds values.) (Hint 2: You can convert a String to an int using the `parseInt` method from the Integer class: `int i = Integer.parseInt(string)`)

- Use the constructor from the Song class to instantiate a new Song object, passing in the variables containing the title, artist, album, and play time (in seconds).
    - It may be useful to print your newly-created Song object to make sure everything looks correct during testing, but be sure to remove this output before submission.
        - To print a song, use the provided `toString` method in Song. For example: System.out.println(song1.toString());
- Follow the same process for two more songs.
    - Because a copy of the values will be stored in your song objects, you should **reuse the same temporary variables** when reading input for the next song.
    - NOTE: Each Song instance you create will store its own data, so you will use the methods of the Song class to get that data when you need it. Don't use the variables that you created for reading input from the user for any other purpose. For example, once you create a song, you may retrieve the title using `song1.getTitle();`

2. Calculate Average Play Time

- Calculate the average play time of the three song objects you created.
    - Use the `getPlayTime()` method to retrieve the play time of each song and calculate the average play time in seconds.
    - Use the `DecimalFormat` formatter to print the average play time formatted to two decimal places, even if they are zeros. Format the output as shown below.

```
Average play time: 260.67
```

3. Find Song With Play Time Closest to 4 Minutes

- Determine which song has play time closest to 4 minutes (240 seconds) and print the title of that song formatted as shown in the output below. In the event of multiple songs equally close to 4 minutes, only output one song title.

```
Song with play time closest to 240 secs is: Big Jet Plane
```

4. Build a Sorted Playlist

- Build a playlist of songs from the shortest to the longest play time and print the songs in order. Make sure you can handle duplicate song lengths.
- To print the formatted song data, use the `toString()` method in the Song class. For example,

```
System.out.println(song1);
```

- Print the sorted playlist as shown in the output below. (Note there are 88 = symbols in the separator.)

```
========================================================================================
Title                          Artist               Album                          Time
========================================================================================
Where you end                  Moby                 Hotel                           198
Big Jet Plane                  Julia Stone          Down the Way                    234
Last Tango in Paris            Gotan Project        Hotel Costes                    350
========================================================================================
```

Testing

Once you have completed your program, copy your **Song.java**, **PlayList.java** and **README** files into a new directory on the onyx server (if you did the project on your computer).

Download the grader.zip archive and unzip into your p1 directory. When you unzip it, the contents of your directory should be as shown below. You can use the "tree" command to list all files in the directory, including sub-directories.

```
[you@onyx p1]$ tree
.
├── grader.sh
├── PlayList.java
├── README
├── Song.java
└── data
    ├── test1
    ├── test1.out
    ├── test2
    ├── test2.out
    ├── test3
    ├── test3.out
    ├── test4
    ├── test4.out
    ├── test5
    ├── test5.out
    ├── test6
    └── test6.out
```

After unzipping the archive, you will need to change the permissions of the grader.sh file using the command,

```
[you@onyx P1]$ chmod +x grader.sh
```

Also correct for potential corrupted characters with the command:

```
[you@onyx P1]$ sed -i -e 's/\r$//' grader.sh
```

To run the grader, use the following command. This is a copy of the SAME grading script we will use when grading your program, so make sure your output matches the expected output for all test cases. You should see output similar to the following when you run the script.

```
[you@onyx P1]$ ./grader.sh

============== Test Results ==============
----------  Program Input (1)   ----------
Last Tango in Paris
Gotan Project
Hotel Costes
05:50
Where you end
Moby
Hotel
3:18
Big Jet Plane
Julia Stone
Down the Way
3:54
----------  Expected Output (1) ----------
Enter title: Enter artist: Enter album: Enter play time (mm:ss): Enter title: Enter artist: Enter
album: Enter play tim
e (mm:ss): Enter title: Enter artist: Enter album: Enter play time (mm:ss):

Average play time for songs: 260.67

Song with play time closest to 240 secs: Big Jet Plane

================================================================================
Title                          Artist              Album                          Time
================================================================================
Where you end                  Moby                Hotel                           198
Big Jet Plane                  Julia Stone         Down the Way                    234
Last Tango in Paris            Gotan Project       Hotel Costes                    350
================================================================================
----------  Your Output (1)     ----------
Enter title:Enter artist:Enter album:Enter play time (mm:ss):05:50
Enter title:Enter artist:Enter album:Enter play time (mm:ss):Enter title:Enter artist:Enter
album:Enter play time (mm:s
s):
```

```
Average playtime: 260.67

Song with play time closest to 240 secs is: Big Jet Plane

================================================================================
Title                          Artist              Album                    Time
================================================================================
Where you end                  Moby                Hotel                     198
Big Jet Plane                  Julia Stone         Down the Way              234
Last Tango in Paris            Gotan Project       Hotel Costes              350
================================================================================
```

Documentation

If you haven't already done so, add **javadoc comments** to your PlayList class.

- Javadoc *class* comments should include both a short description of the class AND an `@author` YOURNAME tag.
    - The class javadoc comment is placed immediately above the class definition
    - 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 public 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.