Music Collection

1 Introduction

In this assignment, you will be simulating a music collection system. You'll be writing a Song class and an Album class to contain them, and then you'll be writing a driver to pull it all together in an interactive manner.

2 Problem Description

You are a music collector who exclusively collects vinyl (because the quality of non-physical media is atrocious, of course). However, you are having a difficult time remembering all of your music and all of your favorite songs for each album. So, in order to make things faster, you've decided to use your Object-Oriented Programming knowledge to list all your music. To accomplish this, you'll need to write a few classes: `Song.java`, `Album.java`, and `MusicCollectionDriver.java`. Each of these classes will represent a single Song, an Album of songs, and the Driver which utilizes them.

3 Solution Description

3.1 Song class

Create a class called `Song` that will represent an individual song. For every method, all parameter names must match instance variable names.

1. **Private** instance variables:
   (a) A String `title` representing the title of the song.
   (b) A String `artist` for the song artist.
   (c) A String `genre` for the song genre.
2. Constructors. You will have multiple constructors which utilize constructor chaining.
   (a) `public Song(String title, String artist, String genre)`
   (b) `public Song(String title, String artist)`
      i. Utilizes constructor chaining and sets a default value of "unknown" for the genre.

3. Getters. Provide getters for the following variables using **standard getter notation**:
    (a) `title`
    (b) `artist`
    (c) `genre`
4. Setters. Provide setters for the following variable using **standard setter notation**. Again, note that all parameter names must match instance variable names for the entire assignment.
    (a) `genre` — for example — `public void setGenre(String genre)`
5. `toString()`: A toString() method that **returns** a String comprised of the artist, title, and genre.

3.2 Album

Create a class called Album that represents an album in your music collection. An album can have any number of Song objects stored in an array, and has an associated favorite track.

1. **Private** instance variables:
    (a) A String `title` representing the title of the album.
    (b) A String `artist` for the album artist.
    (c) A String `genre` for the album genre.
    (d) A Song[] `songs` for the tracks of the album.
    (e) A Song `favoriteTrack` which corresponds to the favorite track. (By default, this will be the first track in the album).
2. Constructor which initializes instance variables of Album.
    (a) `public Album(String title, Song firstTrack)`
        i. Assigns artist and genre using information from the track.
3. Getters. Provide getters for the following variables using **standard getter notation**:
    (a) `title`
    (b) `favoriteTrack`
4. Setters. Provide setters for the following variable using **standard setter notation**. Again, note that all parameter names must match instanve variable names for the entire assignment.
    (a) `genre` — for example — `public void setGenre(String genre).` When updating the genre for an album, all the tracks in the album should have their genre updated as well.
5. `public void addSong(Song s, boolean isFavorite)`

   (a) If the `songs` array has space remaining, put the provided song into the array.
   (b) If it doesn't, double the size of the backing array and copy the elements over.
   (c) If the provided song is the favorite song on the album, update the corresponding instance variable.

6. `toString()`: A toString() method that **returns** a String comprised of the title, artist, and genre. Should also include the `toString()`s of each song in the album.

3.3 MusicCollectionDriver

Create a class called `MusicCollectionDriver` that represents your entire music collection. Each collection starts out with at least 3 albums, each of which has at least 1 song in them. You can set these to albums and songs of your choice - feel free to include your favorite music! You will then print out the collection and allow the user to pick an album to perform a certain action on.
Your music collection should have a single **static Scanner** object that you use throughout the class. Your `main` method should perform the following:

1. Instantiate at least 3 Songs, using at least one of each version of the Song constructors.
2. Instantiate an Album for each of the songs.
3. Write a loop that performs the following:
   (a) Prints out a numbered ordering of the album titles only. You must use the album's methods to accomplish this.
   (b) Allows the user to select an Album, then call the `albumOptions` method on that album.
   (c) The loop should terminate if the user enters -1 for the album selection, otherwise it should continue looping.

You will also have a static (why must this be static?) `albumOptions` method. The header will look as follows:

```
public static void albumOptions(Album albumName)
```

As you can see, this method will take in an Album object. The method should provide the following functionality:

1. Print out the album information using the `toString()` method.
2. Prompt the user for the following actions:
   (a) *Get Favorite Track*: Print out the information of the Album's favorite track.
   (b) *Change genre*: Allow the user to change the genre of the Album.
   (c) *Exit*: Allow the user to return to the main loop.

Running the program should look something like this:

> Note: `$` is the command prompt on Unix. On Windows it will look something like `C:\>`.

```
$ java MusicCollectionDriver

Music Collection:
[0]: Moving Pictures
[1]: Demon Days
[2]: Cross
Select an album (or -1 to quit): 0

Moving Pictures, Tracks:
Red Barchetta by Rush (Progressive Rock)
[0]: Get favorite track
[1]: Change genre
[2]: Add song
Your choice (or -1 to quit): 2

Song name?
YYZ

Is this your favorite song on the album? (true/false)
false

Moving Pictures, Tracks:
Red Barchetta by Rush (Progressive Rock)
YYZ by Rush (Progressive Rock)
[0]: Get favorite track
[1]: Change genre
[2]: Add song
Your choice (or -1 to quit): 0

Favorite Song on Moving Pictures:
Red Barchetta By Rush (Progressive Rock)

Moving Pictures, Tracks:
Red Barchetta by Rush (Progressive Rock)
YYZ by Rush (Progressive Rock)
[0]: Get favorite track
[1]: Change genre
[2]: Add song
Your choice (or -1 to quit): -1

Music Collection:
[0]: Moving Pictures
[1]: Demon Days
[2]: Cross
Select an album (or -1 to quit): 2

Cross, Tracks:
Waters of Nazareth by Justice (Dubstep)
[0]: Get favorite track
[1]: Change genre
[2]: Add song
Your choice (or -1 to quit): 1

New genre:
Electronica

Cross, Tracks:
Waters of Nazareth by Justice (Electronica)
[0]: Get favorite track
[1]: Change genre
[2]: Add song
```

```
Your choice (or -1 to quit): -1

Music Collection:
[0]: Moving Pictures
[1]: Demon Days
[2]: Cross
Select an album (or -1 to quit): -1
```

4 Checkstyle

Review the CS 1331 Code Conventions and download the Checkstyle jar file and the configuration file to the directory that contains your Java source files. Run Checkstyle on your code like this (in the directory containing all your Java source files):

```
$ java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java
Starting audit...
Audit done.
```

The message above means there were no Checkstyle errors. You can easily count the errors by piping the output of Checkstyle through `grep`.

```
$ java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java | grep -cEv "(Starting
    audit...|Audit done)"
0
$
```

The `-c` option tells `grep` to count matching lines instead of printing them, `E` means use `egrep` (extended `grep`) syntax, and `v` means invert the match. Here we use an inverted match to discard the two non-error lines of Checkstyle's output. If you have `egrep` you could leave off the `-E` and just use `egrep`.

```
$ java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java | egrep -cv "(Starting
    audit...|Audit done)"
0
$
```

Alternatively, if you are Windows, you can use this command to count the number of style errors by piping output through the `findstr` program.

```
C:\textbackslash > java -jar checkstyle-5.6-all.jar -c cs1331-checkstyle.xml *.java |findstr /v
    "Starting audit..." | findstr /v "Audit done" | find /c /v "!!!!!"
```

The Java source files we provide contain no Checkstyle errors. You are responsible for any Checkstyle errors you introduce when modifying these files. For this assignment, there will be a maximum of 20 points lost due to Checkstyle errors. In future homeworks we will be increasing this cap, so get into the habit of fixing these style errors early!

5 Turn-in Procedure

Submit all of the Java source files you created to T-Square Do not submit any compiled bytecode (`.class` files), the Checkstyle jar file, or the `cs1331-checkstyle.xml` file. When you're ready, double-check that you have submitted and not just saved a draft.

6 Verify the Success of Your Submission to T-Square

Practice safe submission! Verify that your HW files were truly submitted correctly, the upload was successful, and that the files compile and run. It is solely your responsibility to turn in your homework and practice this safe submission safeguard.

1. After uploading the files to T-Square you should receive an email from T-Square listing the names of the files that were uploaded and received. If you do not get the confirmation email almost immediately, something is wrong with your HW submission and/or your email. Even receiving the email does not guarantee that you turned in exactly what you intended.
2. After submitting the files to T-Square, return to the Assignment menu option and this homework. It should show the submitted files.
3. Download copies of your submitted files from the T-Square Assignment page placing them in a new folder.
4. Recompile and test those exact files.
5. This helps guard against a few things.
   (a) It helps insure that you turn in the correct files.
   (b) It helps you realize if you omit a file or files.[1] (If you do discover that you omitted a file, submit all of your files again, not just the missing one.)
   (c) Helps find last minute causes of files not compiling and/or running.

[1]Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework will not be accepted regardless of excuse. Treat the due date with respect. The real due date is midnight Wednesday. Do not wait until the last minute!