CS 1331 Homework 4

Due Thursday September 20th, 2012 8:00PM

Introduction

This homework will cover Class Design and General OOP: constructors, getters, setters, and visibility modifiers. Please read all the directions before starting.

Do not forget about javadocing. All methods in all .java files should be javadoc'd. **Please follow the javadoc guidelines outlined in Homework 3.**

To recap:

You must have **class javadocs**:

description, @author, and @version tags

You must have **method javadocs:**

description, @param, and @return tags

- `@author` For indicating who wrote the class
- `@version` Shows the version number for the class, and possibly the date
- `@param paramName` For explaining what a parameter means in a method or constructor. Note the parameter name is repeated in the javadoc
- `@return` For explaining what a method returns

5.0 Overview

In this assignment, you will be creating a Music Collection. A music collection will contain several Albums. Each Album will have an associated favorite track.

You may use additional methods to those listed here.

5.1 Song Class

Create a class called `Song` that will represent a individual song. 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.
    a. `public Song(String title, String artist, String genre)`
        ◦ Assigns the instance variables
    b. `public Song(String title, String artist)`
        ◦ Sets a default value of “unknown” for genre
3. Getters: Provide getters for the following variable using **standard getter notation**
    a. title
        a.i. example: `public String getTitle() {...}`
    b. artist
    c. genre
4. Setters: Provide setters for the following variables using **standard setter notation.** Once again, all parameter names must match instance variable names for the entire assignment.
    a. genre – example: `public void setGenre(String genre) {...}`
5. `toString()`: A toString method that **returns** (not prints) a String comprised of the artist, title, and genre.

5.2 Album

Create a class called `Album` that represents an album in the music collection. Each album will be associated with a favorite track. All parameter names must match instance variable names.

1. **Private i**nstance 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 `favoriteTrack` for favorite track on the album.
    e. An int `trackNumber` for track number of the favorite track.
2. A **static** variable `numAlbums` that keeps track of the number of Albums created

3. Constructors: You will have multiple constructors.
    a. `public Album(String title, Song favoriteTrack, int trackNumber) {...}`
        - Assigns the instance variables from the parameters.
        - Assigns artist using information from the track.
        - Assigns the genre using information from the track
        - Increments the static variable numAlbums.
    b. `public Album(String title, Song favoriteTrack) {...}`
        - Sets a default value of 1 for the trackNumber.
4. Getters: Provide getters for the following variable using **standard getter notation**
    a. title
    b. favoriteTrack
    c. trackNumber
5. Setters: Provide setters for the following variables using **standard setter notation.** Once again, all parameter names must match instance variable names.
    a. genre – example: `public void setGenre(String genre) {…}`
        - When you update the genre of the album, the genre must be updated on the associated favorite track
6. `toString()`: A toString method that **returns** (not prints) a String comprised of the album title, artist, and genre.

5.3 MusicCollection

Create a class called `MusicCollection` that represents a Music Collection. Each collection will have **at least 3 albums**. You will the print out the collection and allow the user to pick an album to perform certain actions 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. Create at least **3** S**ongs**, using at least one of each version of the Song constructors.
2. Create an **Album** for **each** of the songs, using at least one of each of the different Album constructors.
3. Have a main loop that performs the following:
    1. Prints out a numbered ordering of all the album titles **only**. You must use the albums getter to accomplish this.
    2. Allows you to select which Album you would like to use, and then call the `albumOptions` method on that album.
    3. The loop should terminate if the user enters 0 for the album selection.
    4. Continues to loop otherwise.

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

```
public static void albumOptions(Album variableName) {…}
```

As you can see, the method takes in an Album object, which you may name as you please. The method must provide the following functionality:

1. Print out the Album information, using the Album's toString method. (Remember, toString itself does not print anything).
2. Prompt and then allow the user to pick from the following options:
    1. *Get Favorite Track:* prints out the song information (using the song's toString) of the favoriteTrack of the album, including the track number.
    2. *Change Genre:* Allow the user to change the genre of the Album. This must update the genre of the associated song.
    3. *Return:* Allow the user to return to the main loop.

To see some example output of what your program might look like, go to the next page.

Don't forget to use shadowing (local variable names matching instance variable names) throughout the assignment. Sometimes, after using Scanner.nextInt() a newline character is left over in the buffer in Java 6, and you must call Scanner.nextLine() to clear it before you can read in another string.

```
Music Collection:
[1] Flux for Life
[2] How I Got Over
[3] Safe in the Steep Cliffs
Select an Album (0 to quit): 1

"Flux for Life" by Mimosa: dub
Album options:
[1] Get Favorite Track
[2] Change genre
[0] return
Selection: 1

Track No. 1: Mimosa - Dead Like Me, Genre: dub

Album options:
[1] Get Favorite Track
[2] Change genre
[0] return
Selection: 2

New genre: dubstep
Album options:
[1] Get Favorite Track
[2] Change genre
[0] return
Selection: 1

Track No. 1: Mimosa - Dead Like Me, Genre: dubstep

Album options:
[1] Get Favorite Track
[2] Change genre
[0] return
Selection: 0

Music Collection:
[1] Flux for Life
[2] How I Got Over
[3] Safe in the Steep Cliffs
Select an Album (0 to quit): 2

"How I Got Over" by The Roots: hip/hop
Album options:
[1] Get Favorite Track
```

```
[3] Safe in the Steep Cliffs
Select an Album (0 to quit): 2

"How I Got Over" by The Roots: hip/hop
Album options:
[1] Get Favorite Track
[2] Change genre
[0] return
Selection: 1

Track No. 8: The Roots - The Day, Genre: hip/hop

Album options:
[1] Get Favorite Track
[2] Change genre
[0] return
Selection: 0

Music Collection:
[1] Flux for Life
[2] How I Got Over
[3] Safe in the Steep Cliffs
Select an Album (0 to quit): 1

"Flux for Life" by Mimosa: dubstep
Album options:
[1] Get Favorite Track
[2] Change genre
[0] return
Selection: 0

Music Collection:
[1] Flux for Life
[2] How I Got Over
[3] Safe in the Steep Cliffs
Select an Album (0 to quit): 0



------------------
(program exited with code: 0)
Press return to continue
```

Turn-in Procedure

Turn in the following files on T-Square. When you're ready, double-check that you have *submitted* and not just saved as draft.

- Song.java
- Album.java
- MusicCollection.java
- Any other files needed to run your program

All .java files should have a descriptive javadoc comment.

Don't forget your collaboration statement. You should include a statement with every homework you submit, even if you worked alone.

Verify the Success of Your HW Turn-In

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.**
    (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.

**Note: Missing files will not be given any credit, and non-compiling homework solutions will receive few to zero points. Also recall that late homework (past the grace period of 2 am) will not be accepted regardless of excuse. Treat the due date with respect. The real due date and time is 8 pm Thursday.