Assignment 1

Song Library Design & Implementation with GUI

Posted Tue Feb 7
Due Tue Feb 21 by 11 PM in Canvas
Worth 40 points (4% of course grade.)

You will work **in pairs** on this assignment. Read the DCS Academic Integrity Policy for Programmming Assignments - you are responsible for this.

You may use ANY IDE (Eclipse, IntelliJ, NetBeans, VSCode, whatever) for your projects.

Make sure your project is NOT modular: if you are using Eclipse, when you create the project choose **Don't create** when the create module-info.java dialog pops up.

Here are some useful sites for Java FX documentation:

- Java FX Documentation Project
- Oracle documentation on Java FX. See in particular the sections on Work with UI Controls and Work with Layouts.
- Java FX API

Java/Java FX Version: You must use Java 11 or higher. Any version of Java FX is fine.

Read the FAQs!!! at the end of this page.

Requirements

Design and implement an application with a graphical user interface to manage a library of songs.

A song is uniquely identified by a combination of name and artist (case INsensitive, i.e. upper or lower case are the same).
This means either name or artist may be duplicated, but the combination of name and artist may not.

Your application should have a SINGLE WINDOW with three functions:

1. **Song list display**, with the ability to select ONE song from the list.

   The list will display the name and artist ONLY for each song, in **alphabetical order of names** (and **within duplicate names, by alphabetical order of artists**). Unless the list is empty, one song is always pre-selected, and its details shown - see the following item.

2. **Song detail**, with name, artist, album, and year, of the song that is selected in the song list interface

3. **Song Add/Delete/Edit**, for adding a new song, deleting a selected song, and editing a selected song:

    - **Add**: When a new song is added, the song name and artist should be entered at the very least. Year and album are optional. If the name and artist are the same as an existing song, the add should not be allowed - a message should be shown in a pop-up dialog, or by some other means within the main application window.

      The newly added song should automatically be placed in the correct position in the alphabetical order in the list. Also, it should be automatically selected, replacing the previously selected song, and its details should be shown.

    - **Edit**: ANY of the fields can be changed. But neither of the song name or artist field can be empty. Again, if name and artist conflict with those of an existing song, the edit should NOT be allowed - a message should be shown in a pop-up dialog, or by some other means within the main application window.

    - **Delete**: Only the selected song in the list can be deleted. After deletion, the next song in the list should be selected, and the details displayed. If there is no next song, the previous song should be selected, and if there is no previous song either, then the list is empty and the detail info is all blanks.

    For any of the add/delete/edit actions, your application should ask the user to confirm or cancel when they trigger the action (click a button) to go through with the add/delete/edit. This allows them to back out at the last minute if they change their mind.

Note: If you use the song detail display area for add/delete/edit as well (instead of two separate areas), then make sure the interface is not confusing to the user, particularly if the add or edit is disallowed or aborted by the user.

When your program is started, it should show the current list of songs in the library, in the song list display, with the first song selected by default. (The first time the program is run, there should be nothing in the display, since there won't be any songs in the library.)

The song library data should **persist** across different sessions of your program. This means any update the user makes to the songs in a run session should carry over into the next session when the user runs the app again.

To implement persistance, you can save the song list in a file in whatever TEXT (human readable) format you like, and then read it into your program when it starts up. This means you may **not** serialize the data, even if you know how to do it, because serialized data is not human readable. However, you are free to use JSON, but make sure you tell your grader which library you used (if any) to read and write JSON.

Your application is NOT required to play a song when it is selected. In other words, there is no requirement to have audio playing functionality, and there will not be any extra credit if you choose to implement this functionality.

The application window need NOT be resizable. When the window is closed, the application should be terminated.

Most of the aspects of design (layouts and widgets) and implementation (event handling) you need for this assignment have been covered in class. If there is any aspect you want to add that has not been covered in class, you are expected to discover these for yourself. You may use any code posted on the course Canvas site in your projects without attribution.

NOTE:

- You MUST use Java FX for all UI-related code. (No Swing stuff.) Any version of Java FX is fine.
- In your app, there should be only ONE window for all user interaction.
- The layout for this single application window should be in fxml ONLY. In other words, none of the UI elements should be created using Java code.
- You may use pop-ups ONLY to show an error message, or to ask for confirm/cancel of an action, but NOT for anything else - all inputs and edits must be done in the single main window.
  For the pop-up dialogs, you will NOT be using fxml since you can work with the standard dialog UIs in the JavaFX framework.

  Note: Switching scenes within the main window is fine, since no new windows are popped up. Also, partitioning the main window into various sub-areas is also fine for the same reason.

Grading

GUI Component	Points
Song list+selection	5
Details display	3
Add/Delete/Edit	25
Persistence	7

For the first three items, the grade will depend on correctness and completeness of functionality, as well as **effective/appropriate** use of JavaFX widgets. For the last part (persistence), grading is on correctness, and on ensuring this is transparent to the user, i.e. the user should not have to be concerned with where and how the song list is stored between sessions.

Effective/appropriate means the UI should be clear, the user should know exactly what to expect when events are triggered, the user should know exactly what to do to achieve their objective. If you need to explain to someone how to use the interface, then it's not a good enough interface.

IMPORTANT: Your UI should not use widgets that are an overkill for the task at hand because if a complex widget is used for a simple task, it will confuse the user. So, for instance, do NOT use a table view at all for ANY aspect of the UI, it will be a major overkill since the requirements do not ask you to dynamically rearrange the display on user request (the kind of thing the table view is used for). If you do so, you will lose credit - see the **Point Deductions** section at the end of this document. Same applies to tree view.

Finally, your UI is not required to be aesthetically pleasing (no points for this because it's highly subjective), but you are most welcome to dazzle us with your creativity (just for the heck of it!) as long as it does not get in the way of clean functionality.

You may use any of the classes in the standard Java SDK, but **no external third-party libraries EXCEPT if you want to use JSON to store and retrieve song data.** If you elect to use JSON, you will need to use a JSON library (jar)- inform your grader so that they know to get the library when they test your app.

Alternatively, you may use the CSV format to store song data, but you do not need an external library to process CSV - see the FAQs (last question)

Submission Instructions

Make sure your project is NOT modular: if you are using Eclipse, when you create the project choose **Don't create** when the create module-info.java dialog pops up.

You may implement the application in as many files as you want, but you MUST call your main class `SongLib` (this is the class with the `main` method.)

Also, you MUST organize your classes into packages. There is no requirement for how many packages you need to use, as long as there is at least one. In other words, NONE of your classes must be directly under `src` (which basically amounts to a default package) - this is bad practice (and is discouraged by most IDEs) because it severely limits the way in which you can control access to members of a class.

Make sure to write your names at the top of each of the Java source files in your project.

Zip your project source and data files into `songlib.zip`. Make sure to preserve the project directory hierarchy so we can build your project. Do NOT include binary (.class) files in the zipped collection.

Since you are only submitting Java source and data files, not binaries, it does not matter what IDE you use (Eclipse, IntelliJ, NetBeans, VSCode) to build your application. Your grader will be able to build the application in their IDE out of your source and data files.

Submit your `songlib.zip` file to Canvas. Just ONE submission per team, please!

Point Deductions

Be aware that implementing design and functionality as specified, and submitting ("releasing") your implementation as required, are both important aspects of software development in the real world. The point deductions listed here are intended to alert you to this.

- 5 points, if the application window does not show up at all, but we can fix the code to make it show. So make sure it actually shows when you run the application.
- 5 points, if the application window cannot be closed, or it closes but does not terminate the application

- 5 points, if you do not implement your main method in a class named `SongLib` (see Submission Instructions)
- 5 points, if you have any code in the default package (directly under `src`) (see Submission Instructions)
- Up to 5 points, if you do not write your name at the top of all the files you submit.
- 10 points, if the displayed list of songs is not in alphabetical order as specified in the **Song list display** item.
- 10 points, if you did not use FXML for the UI design of the main window. (This does not apply to the pop-up dialogs to report input errors, or for confirming/canceling any of the add/edit/delete actions - for these you will use the standard dialog UIs provided by Java FX.)
- 10 points, if any window other than the application frame pops up at any time (except for error dialogs for incorrect add and edit, if you choose to use dialogs for these, or dialogs for confirm/cancel of any action).
- 10 points, if you use a Table View or Tree View anywhere.
- Points will be deducted for each test case that cannot be tried because either your data file or your UI does not allow for testing it. The exact number of points will depend the point value of the test case(s).
- **No credit**, if you used any external libraries--you are only allowed to use the standard Java SDK, and optionally JSON (with notification to grader).
- **No credit**, if your program does not compile, or cannot be run. This is a course in software development, there is NO excuse to submit something that does not compile or does not run.

FAQs

- **Q**. Can I use SceneBuilder to make my fxml?
  A. Yes.
- **Q**. Can we switch scenes in the main window? Or would that still be considered a popup/multiple windows?
  A. You can switch scenes, you are not popping up an independent window by doing this.
- **Q**. What characters are permitted for song name, artist, and album?
  A. Any printable character except '|' (vertical bar)
- **Q**. If user enters leading/trailing spaces for name, artist, or album, should we get rid of them?
  A. Yes.
- **Q**. Are we required to show lowercase versions of name and artist when displaying the song info?

A. When you show the info (list or details), it should be exactly as the user entered (so they are not surprised). But when you sort in your code, it should be case insensitive.

- **Q**. For the year, do we have to take into account the BC/AD stuff?
  A. No, the only thing you need to check is that it is a positive integer.

- **Q**. Are we allowed to place the text file with the song list directly under the project folder? If not, is there a particular place that we should put it in?
  A. It should be under the project folder, but not under the code folders. So either directly under the project folder, or you can make a data folder under project, and then put the text file there. (Grading wise, we don't have any requirement as such, so what I am saying here is just best practice.)

- **Q**. For submission, do we need a data file with songs in it?
  A. No. The first time we run your program, there shouldn't be any songs in the library, so the display should not show anything.

- **Q**. Do JSON files count as human readable?
  A. Yes. "JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write." (json.org) Make sure to tell your grader what library you are using to read and write JSON.

- **Q**. If we use CSV file format for storing the data, are we allowed to use external libraries.
  A. To use CSV you don't need external libraries, but if you do want to use externals, make sure to inform your grader so they know what library to use when they test your application. (WARNING! A song name, artist, or album could well have a comma in it, so using comma as a separator between fields is a bad idea.)
