



Feature Delegations:

 = Chris

 = Zabir

Assignment 1

Song Library Design & Implementation with GUI

Posted Wed Sep 16
Due Fri Oct 2 by 11 PM in Sakai
Worth 40 points (4% of course grade.)

You will work in **pairs** on this assignment. Read the [DCS Academic Integrity Policy for Programming Assignments](#) - you are responsible for this.

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 is a good starting [documentation](#) to explore Java FX. [Work with UI Controls](#) and [Work with Layouts](#) are the most relevant links on that page.

Java Version: Java 11 or higher should be fine.

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).** 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. 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, the user should be able to cancel (or back out of) the action if they change their mind after starting the action.

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. 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.)

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.

NOTE:

- You **MUST use Java FX** for all UI-related code. (No Swing stuff.)
- 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.

Grading

GUI Component	Points
Song list+selection	7
Details display	6
Add/Delete/Edit	20
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 to show song details because the requirements do not ask you to dynamically rearrange the display on user request. If you do so, you will lose credit - see the **Point Deductions** section at the end of this document.

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.

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 design 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.

- **If you are using Eclipse:**

Export your project to a zip file called `songlib.zip` (see how to do this in the Eclipse page, under "Zipping up a Project"). Make sure you have done this correctly by importing it back as a project into Eclipse (see "Importing a Zipped Project into Eclipse" in the Eclipse page). Since you are working pairs, a good test would be for one of you to zip the project, and for the other to import the zipped project into their Eclipse installation.

- **If you are not using Eclipse:**

Zip your project structure into `songlib.zip`, and make sure to inform your grader so that they know not to treat it like an Eclipse project file.

Submit your `songlib.zip` file to Sakai. 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.)
- 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 to show details of items.
- 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 permission from 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.