CSE3OAD/CSE4OAD – Lab 3

Building JavaFX Interfaces for Applications

In this lab, we develop a graphical user interface for an application. The application maintains a collection of music albums. Each music album has: an id, a name, a genre, an attribute to indicate whether or not it is a compilation (a collection of songs from other sources) and the track count.

The following files are provided for this lab:

- HasKey.java. This is an interface, which contains a single method, getkey. This interface is implemented by class MusicAlbum.
- MusicAlbum.java. This class represents a music album. It implements the HasKey interface.
- MusicCatalogDS.java. This is the control class. 11DS" stands for "data source". This class store and manage a collection of MusicAlbums.
- MusicCatalogFXVersion1.class. This is a class file, not a Java file (which is what you have to impelement in Question 1). This class provides the FX interface to interact with the data source controller. You can run this class to have an idea of what is to be expected.
- MusicCatalog.data. This a text file that contain details of a collection of MusicAlbums. When the application loads data, it is data from this file that is loaded.

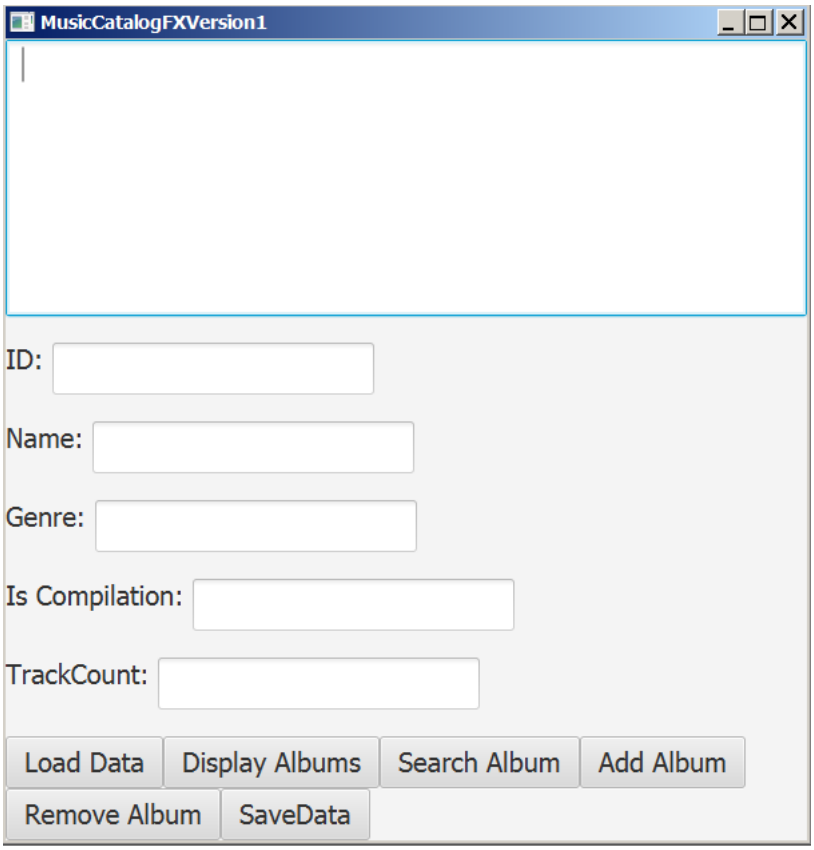
In this lab and the next, we build two versions of the graphical user interface:

- Version 1 (MusicCatalogFXVersion1.java). This version uses a work area that is shared by the operations provided by the application.
- Version 2 (MusicCatalogFXVersion2.java). This version uses the control TableView to present the MusicAlbums in a table with rows and columns. This version will be implemented in the next lab.

Question 1 – Building Version 1 of the Interface

In this question, you will implement a JavaFX class called MusicCatalogFXVersion1.java.

As mentioned earlier, this version is a GUI that uses a work area that is shared by the operations provided by the application. When the application is launched, it presents the screen (stage) shown below.



The screen can be seen as having 3 sections:

- The first section has a text area where we can display a collection of music albums, or just an single music album.
- The second section has 5 text fields, which can be used to perform various operations on the data source.
- The third section has buttons to perform operations: load data, display all music albums, search for an album, add and album, delete an album, and save data.

As mentioned earlier, to see how the application works, you can run the provided MusicCatalogFXVersion1.class.

In implmenting this version 1, if necessary, you can refer to the demo program CatalogFXVersion2 that was provided with Chapter 5 (available on LMS).

Question 2 – Enhancing Version1

If you have nor done so, enhance the version in Question1 by adding a number of Alerts as describe below.

- For the load data operation, an Alert to inform the user that the data has been loaded.
- For the add album operation, add an Alert to ask the user if they want to add the new album or not, and add the album only when the user says "yes".
- Similarly for the remove album operation.
- For the save data operation, add an Alert to inform the user that the data has been saved.

Question 3 – Preparing for the Implementation of Version 2

The aim of this question is to prepare you for the implementation of version 2, which we will implement in the next lab.

a) Make a copy of the demo program CreateTableView.java provided for the lecture, and modify it to present a table view for the music albums.

b) Modify the program to provide a button to sort the rows in the table view by genre (in ascending order), and within genre by name (in ascending order).

   See the demo program CreateCustomizedSorter.java provided for the lecture.

c) Modify the program to add a text field that can be used to filter the rows. The table view should shpw only music albums whose name contains the text in the text field as a substring.

   See the demo program CreateFilter.java provided for the lecture.

■