Running Head: DESIGN AND ANALYSIS OF SONGS DATABASE GUI

Design and Analysis Songs Database GUI Title Page

Code Summary and Reflection

Brian Loughran

Johns Hopkins University

General Design

The major data structure used in this program is the Song class. Within this class I store values which populate the fields of the GUI, including the title, item code, description, artist, album and price. This class is also used to throw exceptions for incorrect input. Displaying information about these exceptions is done with EventHandlers, in a hidden Label (errorBar), which is positioned below the exit button. The database of songs is an ArrayList of data type Song. ArrayList was used because of its flexibility in list length and the ability to easily loop through the database.

A few variables that need their state maintained and accessible in many functions and handlers were defined in the superclass SongsDB. This includes the Songs ArrayList, the mode flag (used to determine if the user is doing add or edit functionality), and all the fields and buttons in the GUI. This allows for accessing and editing their values at any time in the code (helpful if you want to change/access their state from multiple different methods/handlers).

In the start() method, fields and buttons are formatted, and event handlers are defined for the Title combo box as well as the add, edit, delete, cancel, add and exit buttons. The handlers are defined as inner classes to the start() method. Handlers were not created for the item code, description, artist, album or price fields. This is because changing their state does not affect the state of the Songs ArrayList and does not affect any of the other fields in the GUI. A user can “finalize” their input to these fields by clicking any of add, edit, delete, accept or cancel. The Title field needed a handler because changing the selection in the combo box does change the value of other fields in the GUI.

Alternative Approaches

One approach considered was to use List rather than ArrayList to store the list of array type Song. This would have a slight advantage of being able to access the List by index, rather than using the .get() method in ArrayList. But using List would also require creating a new List every time the length of the songs database changes (whenever a user adds or deletes a song). This is because List must have a specified length. The benefits of having a data structure which could easily change length with the .add() and .remove() methods outweighed the benefit of accessing items directly by index, as using the .get() method of ArrayList is not too much of a hassle.

Another alternate approach considered was to display a message box on user input data verification errors. I instead chose to use a hidden text field which would act like an error bar. I prefer the error bar, but both provide the same general functionality. Message boxes seem to be “going out of style”. They are sometimes described as “show stoppers”, as they halt progress of the program and can interrupt the user’s train of thought, forcing the user to deal with the dialog box. The error bar option accomplishes the same task of notifying the user of an error, but has the added advantage of being less invasive than a message box.

Learning and Looking Back

From this project I learned a lot about formatting complex GUI’s, and how to coordinate their state with the underlying data. In this project I had to make sure the GUI properly displayed and updated data which was of the form ArrayList<Object>. I also got practice in using GridMap, and learned that items in the map can be centered and can span multiple indices.

If I were to start this project again, I may consider using a Map rather than an ArrayList. The map could use item code as the key, and the rest of the song object for the value. The specifications specifically state that the item code must be unique, and some logic had to be added to make sure to throw an error with the ArrayList that I was using. I could get this same logic, essentially “for free”, by using the Map data structure, and display an informative error message in the error bar based on the error thrown by the Map.

References

https://stackoverflow.com/questions/2275004/in-java-how-do-i-check-if-a-string-contains-a-substring-ignoring-case