|  |
| --- |
| RUNNING HEADER: 605.201 MINI PROJECT #3 Program DESIGN AND ANALYSIS |
| **605.201 MINI PROJECT #3** |
| **Program Design And Analysis** |
|  |
| **Guan Yue Wang** |
| **Johns Hopkins University** |

|  |
| --- |
|  |

General Program Design

This program simulates a Song Database that allows user to show and modify a list of Song objects with external data connection. The whole program is constructed by 3 classes. First of all, Song class basically simulates a new Song data type which contains all characteristics of a song such as item code, description, artist, album, and price. It has the functionality to let user get the set those information. In addition to that, SongDatabaseFrame class constructs the user interface based on an Array List of Song object it intakes. In the interface, Combo box is used to allow user switch between songs, text field is used to display the corresponding details for the selected Song, and label is utilized to show text field label. Furthermore, there are buttons to add, edit, and modify the Song list as well as accept and cancel buttons to confirm changes. These actions take effects in both front end combo box contents as well as back end array list that stores all the Song objects. All the changes made during the process would be output to the external text file upon exit. Last but not the least, the main SongDatabase class reads the external text file and store all the Song details as Song objects in a Array List. This Array List is then used to create the SongDatabaseFrame user interface with all the functions mentioned above connected to this Array List of Songs. If there's no existing text database file found, a FileNotFound exception handler is utilized to create a new text data file with default Song objects displayed in the user interface created.

The major data structures used includes List and Array List. To be specific, List is used to extract Song details in the text data file since all the song information is stored at each line separated by commas. After each song's characteristics are extracted at per line level, these details are used to create Song objects which are eventually put into an Array List. Array List is chosen here because its dynamic array makes one easier to add, modify, and remove Song objects.

Alternative Approaches

One of the alternative approaches I've considered is to use Tree Map to store Song objects with item code as key and other information as mapped value. However, it's rejected due to several reasons. First of all, it's harder to modify and output Song objects in the mapped values compared to elements in Array List. Secondly, requirements have not specified whether there can be duplicate item codes or not whereas Tree Map can only have unique item codes as keys. Therefore, it creates potential risks in the future. In addition to above points, it's more efficient to store everything as Song once per item instead of storing it twice as keys and mapped values. All in all, array list becomes a better choice here resulted from its efficiency and functionality under this circumstance.

Other than the data structure, several different layouts are also tested during the design phase of the user interface such as border layout and box layout. However, flow layout is eventually used as this Song Database's layout. As there are a lot of labels, text fields, and buttons are displayed in one interface, flow layout makes people easier to follow and also costs less time to development compared to other layouts.

Conclusion

By doing this project, I learnt the design of the interface itself can be as time consuming as the development of the backend program logics. All the factors such as panels positions, section layouts, fields set up, selection types, buttons, action performers can all have a significant impact on user experience. Also, it's truly amazing how the object-oriented logics are integrated with GUI design and generates benefits in the aspect of both program development and user experience.