Mini Project #3 Reflection

**GENERAL PROGRAM DESIGN**

**Classes.** The program follows a simple design with only two classes. The first class is the SongDetails class which takes input from a file and creates SongDetails elements which are later added to a TreeMap for storage. SongDetails uses two constructors. The first, used to read in files, collects array elements. The second, collecting six Strings is used when editing entries. This constructor converts the final String, the price, into a variable of type double. This is done to allow a future ability to collect payment, as well as verification that the entry is numeric (discussed below). SongDatabase also defines get and set methods for all text fields used in the program and overrides the toString() method with a custom printout.

**Methods.** The SongDatabase class handles most of the action in this program through the main(), readDBFile(), start(), returnToViewMode(), and writeOut() methods. readDBFile() is called from main() which attempts to read in data from a file at a location specified by runtime parameters. It uses a BufferedReader to loop through the lines of a file, separating elements based on commas, assigning the parsed data to elements in a newSong array, which is then added to the songs TreeMap, where the item code is the key. If no file is found, a FileNotFoundException is caught and the user is asked if they want to create one.

Control returns to main() where launch() is called which initiates the start() method. Start creates and initiates all text field labels, text fields, the comboBox (songListComboBox), and buttons and sets their initial state based on whether the requested song database is empty or contains data. An action listener listens to the songListComboBox and when new selections are made, it retrieves the respective data from the songs TreeMap and displays the data in the text fields. All buttons utilize anonymous inner classes to listen for changes. Add, edit, and delete buttons, when there is change, respond by adjusting the property isEditable() and/or isDisable(), and updating the statusLabel. The cancel button calls returnToViewMode() which sets the correct state of the fields and buttons. The accept button listens for clicks and responds in three different ways depending on whether it was called from Add, Edit, or Delete mode. Finally, the exit button invokes the writeOut() method which uses a BufferedWriter() to loop through songs.entrySet() and write the data to a file at the location specified by the runtime parameters. writeOut() catches IO Exceptions by updating the errorLabel.

Because anonymous inner classes are used to handle the ActionEvents and ChangeEvents, there was no need for additional classes beyond the two created.

**ALTERNATIVE APPROACHES**

I considered creating separate classes for each of the button listeners, however felt that for the limited work they would be doing, separating the action from the button would create more disconnected code. Most buttons have limited dutie,s with the exception of acceptButton and exitButton. I realized shortly after beginning to build the program that the acceptButton would actually be responsible for adding, editing, and deleting entries. Perhaps a better approach would have been to create a separate class to handle acceptButton’s responsibility. I did, however, simplify the code by recognizing that acceptButton was returning the program to the same state after every accepted action. At that point I removed the redundant code and formed the returnToViewMode() method.

Originally, I also had another class that was responsible for reading in or creating the new song database file. I realized after implementing that class that it mainly consisted of the method, and that it was not necessary to be in a class of its own.

One aspect I considered in the planning phase was the type of data structure that would be most appropriate. I settled on the TreeMap due to its ability to link key/value pairs, ability to add and remove efficiently from anywhere within its storage, not limited to top and bottom, and restriction against adding two items with the same key. I liked the last feature because it would allow the database to prevent adding the same song twice (based on its item code). This took a bit of extra work to implement, however, as the program is better suited to using the song title as the key, because this is what the combo box list is selecting. This, of course, would not be a good design, as there are many different songs with the same name. In the future, I would make two enhancements to this program. First, I would organize the loadedSongs ListView alphabetically using a comparator. Secondly, I would make the buttons responsive to pressing the Enter key from the keyboard.

**Lessons Learned**

This program helped me become more comfortable using generic classes, like the TreeMap<String, SongDetails> that holds the database information. I also spent a lot of time trying to get the graphics correctly loaded. Initial versions used GridPanes and StackPanes. I think I originally misunderstood the StackPane, and spent quite a bit of time thinking that my action listeners on my buttons and text fields were not working, only to find out that in fact they were working, they were just layered below the pane holding the labels, and therefore were not responding to clicks. I also now feel more comfortable using action and change listeners to monitor action in a program.