Analysis of Song Database

Design Decisions and Reflections

by: Alina Grigorovitch

Introduction to Programming Using Java, section 82

Program Design and Analysis

Design and Implementation

My program was constructed of a SongDatabase class, a SongDatabaseBuilder class, a SongDB.txt file, and an optional Song class. SongDatabase extended the JFrame class and consisted of a main method and a constructor method that created the GUI song database. The main method handled file input in a try..catch block that caught FileNotFound and IOException exceptions. Within main, for each line read, if the line was not null, the String was converted into a collection ArrayList<String>. Each ArrayList<String> was then added to another ArrayList containing ArrayList<String> objects. This ArrayList<ArrayList<String>> collection was then passed to the SongDatabaseBuilder method called in the SongDatabase constructor method.

The SongDatabaseBuilder class handled the song data that was passed in an ArrayList of ArrayLists. It set the initially empty ArrayList<ArrayList<String>> 'songs' to the collection passed from the SongDatabase class. Int variables were assigned at the beginning for each index in the ArrayList<String> corresponding to the type of data it contained (for example, _id = 0; _song = 1). These variable names, rather than numbers, were then used throughout the program to access this information. This was done with the database file in mind and was built in to allow for easy to make changes to the program in case the database file should change. Another int, index, was set to 0 and updated to the ArrayList index value of the song selected in the JcomboBox. This allowed the other data to be accessed from the objects using the 'index' value. Finally, an empty ArrayList<String> songToAdd was created for the accept actionEvent (described below).

Next, the GUI Panel start method created the initial GUI state consisting of three panels: one for Jlabels, one for the JComboBox and JtextFields, aligned left and right, respectively, and one for the Jbuttons, aligned center. The Labels described the ComoBox and TextFields. The ComboBox used the setSelectedItem(songs.get(index).get(_song)) method, which obtained the ArrayList<String> object at the index value using get, and then obtained the String data object from the called ArrayList<String> by calling the get method again. Since this returns a String, this call format was used to obtain any piece of information from the double ArrayList 'songs' and populate all of the display fields in the second panel. Finally, the start method set all display information using the song at index value 0, no TextFields editable, and all Buttons enabled.

Next, SongDatabaseBuilder overrode the actionPerformed method for the ComboBox and TextFields. The code straightforwardly implemented the Project 3 specifications using the setEnabled and setEditable methods. Of note were the actions for delete, accept, and exit. Delete straightforwardly removed the entry by setting index with the ComboBox's getSelectedItem method, and by calling songs.remove(index). The most complicated implementation occurred when the source was the accept button. Temporary Strings were created to store each data String (newID, newSong, and so forth). Checks on all editable fields were performed. The regex import was used to check the price and ID fields by calling a separate regexChecker method that took the regex String and the getSelectedItem string, then returned true if there was a match. Other fields were checked for null input. If the input String was null, the corresponding Labels changed with instructions to add text and select accept again. As a precaution, the newID etc Strings were set to “N/A” in the else block in case the user failed to comply. If all checks passed, the temporary Strings populated songToAdd by calling the Arrays.toList method and passing the

Strings obtained from the editable fields. Then, the song was added using the updateArray method which called the songs.add() method. Finally, the exit action event called the outputDB method which, in a try..catch block, created a new File and PrintWriter to write to the File specified at the same filepath as the input file (to override the original input database). A new PrintWriter called the outputDB method, and the double ArrayList looped through using a for..each loop, creating a String out of the Strings in each ArrayList<String> collections, the using the PrintWriter to print a new line of that String to the output file. This could have been done more elegantly using a nested for..each loop, and would have been a necessary design implementation for a database file that contained many more than 6 fields.

Alternatives and Additional Ideas

The major alternative design idea was to create Song objects from each ArrayList<String> passed by the input, store these objects in a new ArrayList<Song> collection, and access information on each song via its variables (Strings id, song, artist, album, description, price). The ArrayList<ArrayList<String>> design seemed appealing because it bypassed the use of classes, and technically the program was already handling String objects, which is why an additional class seemed unnecessary. In hindsight, I think this might have been the better method because, while bulkier and requiring more storage, it is easier for another programmer to read and manipulate.

Another design enhancement would have implemented a reset() method that would return the GUI to its original state. This would have involved less coding in the ActionEvent checker for the cancel and accept buttons.

In hindsight, I did not need to create temporary String variables in the accept button checker method, and could have instead called from the JTextFields if those entries passed the checks implemented beforehand.

Finally, a further enhancement could have been added to the file output. Since an IO error could have occurred at any point, it was unsafe to write to a database file named the same as the input file. Instead, the program should have written to a temporary file, and if no IO errors occurred, that temporary file should have been destroyed and copied to file by the same name as the input file and replaced that.

What I Learned

From a technical aspect, I learned how to integrate many different pieces to build a larger program: Swing(GUI), Regex, try..catch, and I/O. This project focused much more on those aspects, which are universal to all programming languages regardless of whether they are OOP or not.

References

1. “Java: The Complete Reference 9th Edition.” Schildt, Herbert. Oracle Press, 2014 McGraw Hill.
2. Stackoverflow.com