Music Playlist Manager Assignment

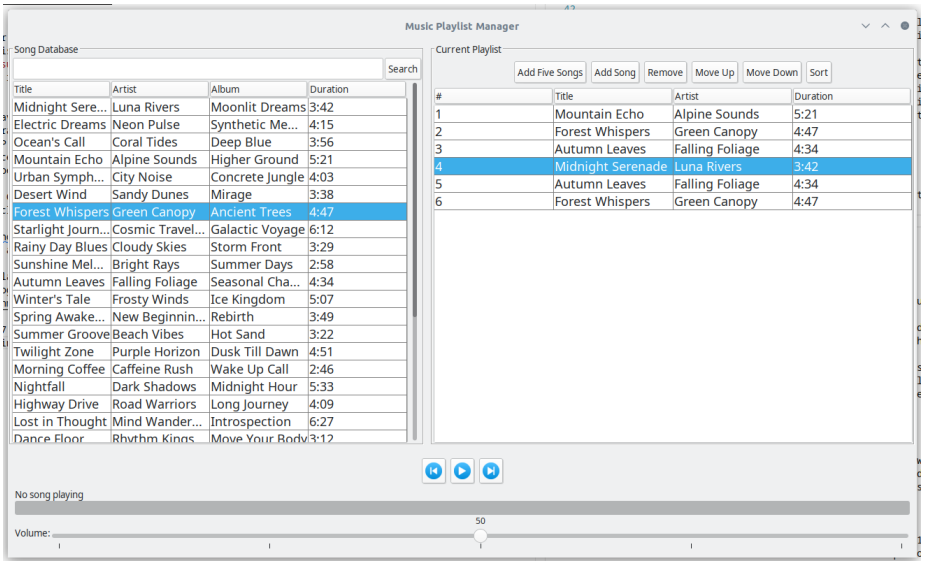

Figure 1: Screenshot of Finished App

Overview

In this assignment, you will implement key components of a Music Playlist Manager application using Java and Swing. This project focuses on applying data structures concepts, particularly linked lists, to create a functional music playlist management system with a graphical user interface.

The application allows users to: - View a database of songs - Create and manage a playlist - Add and remove songs - Reorder songs in the playlist - Sort the playlist using custom data structures

Learning Objectives

- Implement linked list data structures with additional functionality
- Apply sorting algorithms to a linked list implementation
- Understand how MVC (Model-View-Controller) architecture works in a real application
- Practice implementing interfaces (Comparable) in Java
- Work with Java Swing GUI components

Project Structure

The project consists of the following files:

1. **MusicPlaylistManager.java**: Main application class with GUI setup (already implemented)
2. **LinkedList.java**: Generic linked list implementation (already implemented)
3. **BetterLinkedList.java**: Extended linked list with additional functionality (needs implementation)
4. **Song.java**: Class representing a song with metadata (needs to be modified)
5. **PlaylistController.java**: Controller for playlist operations (needs implementation)
6. **SongDatabaseController.java**: Controller for the song database (already implemented)
7. **songs.csv**: CSV file containing the song database (already provided)

Your Tasks

1. Implement the `BetterLinkedList` Methods The `BetterLinkedList.java` file extends the LinkedList class. You should *copy* your own linked list class to this folder and then implement the following methods:

- `swap(int i, int j)`: Swaps the elements at the specified positions
- `swapWithNext(int i)`: Swaps the element at position i with the next element (should be more efficient than swap(i, i+1))
- `sort()`: AFTER #3 below: sorts the elements using bubble sort (elements must implement Comparable)

2. Implement the `PlaylistController` Methods Complete the `PlaylistController.java` file by implementing the following methods:

- `addSong(Song song)`: Adds a song to the end of the playlist
- `removeSong(int index)`: Removes a song at the specified index
- `moveSongUp(int index)`: Moves a song up one position using swapWithNext
- `moveSongDown(int index)`: Moves a song down one position using swapWithNext
- `sortByTitle()`: Sorts the playlist by title using the sort method from BetterLinkedList
- `getSize()`: Returns the number of songs in the playlist
- `getSongAt(int index)`: Gets a song at the specified index
- `clearPlaylist()`: Removes all songs from the playlist

3. Implement the `Song` Class Comparable Interface Modify the `Song.java` file to implement the Comparable interface (We will discuss this in class):

- Make the Song class implement Comparable
- Implement the compareTo method to compare songs by title

4. **Add at least one extension of your choice** Here are some ideas

- Add a button to add 5 random songs to the playlist
- Add a button to remove duplicates from the playlist
- Add mp3 files and enable the transport buttons to actually play them
- Replace the .csv with a folder of .mp3 files and read their ID3 tags
- Remove songs from the database list when they're added to the playlist and move them back when they're removed.
- Add a button to randomize the order of the songs in the playlist
- Enable Sort by any column (right clicking on the Sort button brings up a menu)
- Save and reload playlists that persist across sessions
- Link to an API like SoundCloud or Spotify to load real playlists and play music
- Some similar functionality of your choice that involves working with the data and the UI

Implementation Notes

For `BetterLinkedList.swap(int i, int j)`:

- Ensure both indices are valid before performing the swap
- To swap elements efficiently, swap the data in the nodes rather than reconnecting nodes
- You need to traverse the list to find the nodes at positions i and j
- Handle the case where i equals j (no swap needed)

For `BetterLinkedList.swapWithNext(int i)`:

- This should be more efficient than calling swap(i, i+1) since you only need to traverse the list once
- Make sure the index and index+1 are valid before performing the swap
- Only swap the data, not the node references

For `BetterLinkedList.sort()`:

- Implement a bubble sort algorithm that works with a linked list structure
- Use the compareTo method of the elements (which must implement Comparable)
- Consider edge cases like empty or single-element lists

For `PlaylistController` methods:

- The `updatePlaylistTable()` method is already implemented to update the UI after changes
- Make sure to call this method after any operation that modifies the playlist
- Use the appropriate LinkedList methods to implement the required functionality

- Handle edge cases appropriately (e.g., invalid indices, empty playlist)

Requirements

- Your implementation should correctly handle edge cases (empty lists, invalid indices, etc.)
- Throw appropriate exceptions when indices are out of bounds
- Your solution should not crash when used in the GUI
- Follow the method signatures exactly as specified

Getting Started

1. Download the provided source files and uncompress
2. Set up a Java project in your preferred IDE (VS Code, IntelliJ IDEA, Eclipse if you hate life, etc.)
3. Import the source files into your project
4. Look for the "TODO" comments that indicate where you need to add code
5. Implement the required methods one by one, testing as you go

Testing

Test your implementation thoroughly with various scenarios: - Adding and removing songs at different positions - Moving songs up and down within the playlist - Sorting the playlist - Testing edge cases (empty playlist, invalid indices) - Verifying that the GUI updates correctly after operations

Submission

Submit the following files: - The entire MediaPlayer folder - A brief report (1+ pages) documenting your implementation choices and any challenges faced. Also explain your extension and how you implemented it.

Step-by-Step Implementation Guide

1. **Implement BetterLinkedList.java**:
   - Complete the `swap` method to exchange elements at positions i and j
   - Implement the `swapWithNext` method for efficient adjacent element swapping
   - Implement the `sort` method using bubble sort to arrange elements in ascending order
2. **Implement PlaylistController.java**:
   - Complete `addSong` to add a song to the end of the playlist
   - Implement `removeSong` to remove a song at a specific index
   - Complete `moveSongUp` and `moveSongDown` to reorder songs
   - Implement `sortByTitle` to sort the playlist
   - Add the remaining methods for playlist management
3. **Modify Song.java**:

- Add `implements Comparable<Song>` to the class declaration
- Implement the `compareTo` method to compare songs by title (case insensitive)

4. **Test Your Implementation**:
   - Run the application and verify that all functionality works correctly
   - Test edge cases and make sure your implementation handles them gracefully
5. **Add Your Extension**
   - It should modify the data store in some way
   - It should involve an update to the UI

Deadline

Submit your completed assignment by **March 27 in class**. Penalty for late work at least 50%.

Resources

- Java Comparable Interface: https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html
- Sorting Algorithms Overview: https://www.geeksforgeeks.org/sorting-algorithms/
- Linked List Data Structure: https://www.geeksforgeeks.org/data-structures/linked-list/