Music Manager Brief & Summary

Eoin Fitzsimons – X23151374

National College of Ireland

BSH Computing

# Brief

SUBMISSION DETAILS:

Submit a zipped NetBeans project on Moodle,

Submit a problem-solving Document in Word.

The document is to have a font size of 12 pt. Include student name, student ID, and course name at the top of the page.

It should have a summary of your design outlining the ADTs, interfaces, classes, and your method of managing the songs (max 500 words).

Include the link to your public GitHub repo.

Copy and paste your class diagram to the document.

Late submissions will not be penalised if the student applied for an extension through NCI360 and it was approved.

Students may be asked to attend a viva examination to assess their understanding of the work submitted.

TURNITIN: All report submissions will be electronically screened for evidence of academic misconduct (i.e., plagiarism and collusion)

## Design and implement an application that uses the abstract datatypes you have learnt in class with interfaces and a swing GUI using NetBeans.

Design an application to allow a user to manage their music.

They need to be able to create a playlist of liked songs. These songs will then be used to populate at least 2 playlists organised by criteria, such as genre.

The user needs to be able to add a song from the liked playlist to one of the two genre playlists.

The system's limitation is that only the last song added to the liked playlist can be moved to the appropriate genre-based playlist.

Users can add, search, delete and move songs around on their playlists.

The user needs to be able to see a printed list of whichever playlists they wish.

Show how many songs are in each list.

The user needs to be given the option to create a playlist that can be set to repeat.

# Class Diagram

A screenshot of a computer

Description automatically generated

# Summary

**Abstract Data Types (ADTs)**: The application uses two ADTs, a Stack and a Double Linked List.

* **Stack**: The Stack is a linear data structure that follows the Last-In-First-Out (LIFO) principle. It’s used for managing the “liked” songs in the application. The last song added to the “liked” stack is the first one to be moved to a playlist, adhering to the system’s limitation.
* **Double Linked List**: The Double Linked List is a complex linear data structure consisting of nodes linked in both directions. This allows for efficient alterations from any position within the list.

**Interfaces**: The application defines two interfaces, Stack and Double Linked List, for the ADTs. These interfaces establish a contract for what the classes can do, ensuring consistency and predictability.

* **Stack Interface**: This interface defines the methods that any class implementing a stack data structure must have. These methods include operations like push (add an element) and pop (remove an element).
* **LinkedList Interface**: This interface defines the methods for classes that use a linked list data structure. These methods include operations like add, remove, and search.

**Classes**: The application includes several classes, each with its responsibilities and roles in the system.

* **DLList**: This class implements a Double Linked List. It provides methods for adding, removing, and searching songs in a playlist.
* **LinkedListInterface**: This class defines the methods for using a linked list, providing a blueprint for any class that needs to use a linked list data structure.
* **MusicManagerApp**: This is the main class that runs the application. It creates instances of other classes and calls their methods as needed.
* **MusicManagerGUI**: This class handles the graphical user interface of the application. It creates and manages GUI elements like buttons and text fields and defines how they behave.
* **MyStack**: This class implements a Stack, providing methods for adding, removing, and viewing songs in the “liked” songs stack.
* **Node**: This class represents a node in the Double Linked List. Each node contains some data (a song) and references to the next and/or previous nodes.
* **StackInterface**: This class defines the methods for using a stack.

**Song Management**: The application provides several features for managing songs and playlists.

* **Adding Songs**: Users can add songs to the “liked” stack. When a song is added, it’s converted into title case for uniformity across all songs in the application.
* **Creating Playlists**: Once a song is in the “liked” songs stack, it can be moved to one of two playlists, organised based on certain criteria such as genre.
* **Modifying Playlists**: Users can modify their playlists by removing or swapping songs. The removal operation is case-insensitive, providing flexibility in song management.
* **Displaying Song Lists**: Users can view a printed list of any playlist they wish. The list is dynamically updated whenever a song is added or removed from a playlist.
* **Repeating Playlists**: The application includes a repeat feature that continuously simulates playing songs from a selected playlist. Activates through a menu item and a keyboard shortcut.

# Extras

[EoinFitzsimons/MusicManager2: For NCI DSA CA (github.com)](https://github.com/EoinFitzsimons/MusicManager2)