Project 3 - Playlist Manager

# Assignment Title:

Advanced Playlist Manager with Sorting, Search, and Recently Played Queue

# Objective:

This assignment challenges students to implement a playlist manager using both ArrayLists and LinkedLists, allowing them to practice list manipulation, searching, sorting, and managing data with a queue. This will deepen their understanding of different data structures and their trade-offs.

# Assignment Requirements

## Part 1: ArrayList-Based and LinkedList-Based Playlist

1. Playlist Class Implementations: Create *PlaylistArrayList* and *PlaylistLinkedList* classes, each managing a playlist of songs.  
• Define each song with the following attributes:  
 - Title  
 - Artist  
 - Duration (in seconds)

• Implement the following methods for each playlist:  
 - addSong(Song song): Adds a song to the end of the playlist.  
 - insertSong(int index, Song song): Inserts a song at a specified position.  
 - removeSong(int index): Removes a song from a specified position.  
 - getSong(int index): Retrieves the song at a specified position.  
 - shuffle(): Randomly shuffles the playlist. (You may use Java’s random number generator)  
 - printPlaylist(): Displays all songs in the playlist.

2. Sorting and Searching: Implement sorting methods:  
 - sortByTitle(): Sorts songs alphabetically by title.  
 - sortByArtist(): Sorts songs alphabetically by artist.  
 - sortByDuration(): Sorts songs by duration.

• Implement search methods:  
 - searchByTitle(String title): Finds and returns the position of the song with the given title.  
 - searchByArtist(String artist): Finds and returns the position of the first song by the given artist.

3. Testing Requirements: Test each method with sample playlists to ensure functionality, focusing on sorting, searching, and shuffle.

## Part 2: Recently Played Queue

1. Recently Played Queue Class: Implement a RecentlyPlayedQueue class that tracks the last 5 songs that were played.  
• Use a queue structure to manage the 'recently played' list:  
 - addRecentlyPlayed(Song song): Adds a song to the queue. If full, removes the oldest song.  
 - printRecentlyPlayed(): Prints the list of recently played songs in order.

2. Integration with Playlist Manager: Modify the playlist manager to add a song to the recently played queue when played (retrieved with getSong).  
• Ensure that only the last 5 songs are displayed when the recently played queue is printed.

3. Testing Requirements: Test the queue by viewing multiple songs and verifying the correct order of the recently played list.

## Part 3: Playlist Manager Application with Enhanced Features

1. Enhanced User Menu

Build a PlaylistManager class with the following options:  
 - Add Song: Add a new song to the playlist.  
 - Insert Song at Position: Insert a song at a specified index.  
 - Remove Song: Remove a song by index.  
 - View Song: Display a song, adding it to the recently played queue.  
 - Shuffle Playlist: Randomize the song order.  
 - Print Playlist: Display all songs in the current playlist.  
 - Sort Playlist: Sort by title, artist, or duration.  
 - Search for Song: Search by title or artist and display position.  
 - View Recently Played Songs: Display last 5 played songs.  
 - Switch Playlist Type: Toggle between ArrayList and LinkedList-based playlists.

# Deliverables

1. Source Code: Implementations for PlaylistArrayList, PlaylistLinkedList, RecentlyPlayedQueue, and PlaylistManager.

2. Test Cases: Include test cases for each method to verify playlist manipulation, sorting, searching, and queue functionality.

3. Analysis Report: A brief analysis of the time and space complexities of each operation in ArrayLists and LinkedLists.

# Grading Criteria

* Implementation (50%): Correct functionality of PlaylistArrayList, PlaylistLinkedList, RecentlyPlayedQueue, and PlaylistManager.
* Testing and Validation (20%): Effective and thorough test cases for each method.
* Performance Analysis (20%): Insightful analysis of ArrayList vs. LinkedList operations and queue performance.
* Documentation and Code Quality (10%): Clear, organized code with comments explaining key sections.

# Additional Notes

* Collaboration: Students can discuss general ideas but must submit their own code.
* Resources: Refer to course materials or online documentation, but all code must be original.
* Academic Integrity: Ensure the code reflects your own understanding and effort.