MONASH University

MONASH INFORMATION TECHNOLOGY

FIT1051 ASSIGNMENT ONE - TASK 6

Submission deadline: Before the start of Week 10 Applied Class, via Moodle, and to be demonstrated to the tutor during that class

Weight: 2% of your overall unit mark

Academic Integrity:

Please be reminded of the academic integrity standards that are expected of you at Monash. You should work alone and ask the unit staff for help if needed. Do not post your work in public forums or send your work to anyone. Do not copy/paste text or code from other sources and present it as your own — this includes use of generative AI tools. Breaching these academic integrity requirements can incur serious penalties. Please note, as per university requirements, similarity detection will be used across all student submissions. You may be asked for a short interview if there are any concerns regarding similarity or the use of Generative AI.

ASSIGNMENT ONE OVERVIEW:

Each week, you will expand a simple music program into a full playlist manager, learning Java concepts step by step. The activities are designed such that it introduces the fundamental concepts of software development from the perspective of a Java developer.

OVERVIEW OF THE MUSIC PLAYLIST MANAGER:

A Music Playlist Manager (MPM) is a console-based application that lets users create and manage their own music playlists. It simulates a personal music assistant where users can add, remove, and organize songs according to their preferences.

- Creating a Playlist:
  The MPM allows the user to create a new playlist by giving it a name. This playlist acts like a container where songs can be added, removed, or rearranged. Users can have multiple playlists for different moods or occasions.
- Managing Songs:
  The MPM keeps track of songs in each playlist. Each song has details such as the title, artist, genre, and duration. Users can add new songs, delete existing ones, and view all songs in a playlist.
- Playing Music (Simulation):
  The MPM simulates the experience of listening to music. Users can "play" songs in order, shuffle them randomly, or loop through a playlist. Instead of actual audio, the system prints song details to the console when played.
- Searching and Filtering:
  The MPM allows users to search for songs by keywords (e.g., artist name, genre). It can also filter songs based on duration, mood, or popularity, making it easier to find the perfect track.
- Tracking Usage:
  The MPM can record how many times a song has been played, or which songs are the user's favorites. Over time, it can suggest frequently played songs to simulate a personalized recommendation system.
- Expanding the System:
  The MPM is designed to grow over time. Additional features, such as playlist sharing, sorting songs by different criteria, or adding ratings, can be implemented as the system develops.

THIS WEEK'S TASK INSTRUCTIONS / REQUIREMENTS:

This week, the Music Playlist application was refactored to improve loose coupling and high cohesion. While the `MenuController` class still directly uses Playlist and Song objects, this is acceptable because it operates at the UI level, handling data display and user interactions rather than data management.

In the A1Task6.zip:

1. In the Playlist class, implement the `shufflePlay()` method, which should randomly reorder the songs and simulate playing each one by printing in the console. This operation must not change the original order of the playlist.
2. Implement a feature to track the most browsed playlist in the `PlaylistManager` class.
3. And, when the user selects the option to quit the application from the main menu, the program should show a summary of the user's preferences and statistics, including the most browsed playlist and the most played song. To achieve this, implement the `displayUserPreferenceSummary()` method in the `MenuController` class.

FOR HD (*IN ADDITION TO ABOVE LISTED TASKS*)**:**

- Add/Fix the Javadoc comments for class and methods.
- Add a field called mood to the Playlist class. This field should represent the playlist's genre or mood, with predefined options: {Study, Relax, Party, Sad}. Initialize mood in the constructor and provide appropriate getter and setter methods. You may want to use a type that restricts the value to these predefined options. The `displayUserPreference()` method should also include the user's preferred mood in the summary.
- Read and understand all parts of the Music Playlist application. Be prepared to answer questions on sections you may not have directly worked on. It is essential to have a full understanding of how the application functions.
- Modular code that is easy to read and maintain and follows good OOP and programming practices.
- Follows coding guidelines with proper code documentation.

Do's	Don'ts
• ADD CODE TO THE GIVEN PROJECT • UPLOAD YOUR SOLUTION TO MOODLE UNDER THE CORRECT A1 WEEKLY TASK SUBMISSION LINK. • ATTEND THE APPLIED SESSION ON YOUR TASK'S DUE DATE TO GET MARKED. • MUST USE THE PROJECT FILES PROVIDED	• MODIFY ANY EXISTING CODE IN THE TEMPLATE (EXCEPT COMMENTS). • SUBMIT THE WRONG PROJECT ZIP FILE. • MISS THE APPLIED SESSION — MARKING IS DONE IN CLASS ONLY. • USE YOUR OWN PROJECT FILES.