Objective

The goal of this exercise is to design and implement a database for managing a music player application. The exercise will incorporate **inheritance**, **polymorphism**, and **templates** to provide a flexible and efficient solution.

Tasks

You are tasked with building a database to manage music player entities such as **Songs**, **Albums**, and **Playlists**. Each of these entities will have unique attributes and behaviors, but they also share some common functionality.

1. (1 point) **Create a base class `MusicEntity`** with:
    - Common attributes (e.g., `id`, `name`).
    - Methods for common actions such as `print_details()` and `get_duration()`.

2. (2 points) **Create the following derived classes from `MusicEntity`**:
    - **`Song`**:
        - Attributes: artist, duration.
        - Methods: Override `print_details()` and `get_duration()` to display song-specific details.
    - **`Album`**:
        - Attributes: List of songs.
        - Methods: Add methods to add/remove songs and calculate the total duration of the album.
    - **`Playlist`**:
        - Attributes: List of songs and/or albums (polymorphically).
        - Methods: Add methods to shuffle songs (also within stored albums), add/remove items, and calculate the total duration of the playlist.

3. (3 points) **Implement a generic container class `Database<T>` using templates**, representing a collection of `MusicEntity` objects.
    - This class should:
        - Allow storing, adding, removing, and retrieving items of type `T`.
        - Support searching for items by a given criterion (e.g., by name or ID).
        - Be compatible with all derived classes of `MusicEntity`.
        - Provide a specialization for the type `Playlist` to allow song shuffling.

4. (2 points) **Create and test a menu-driven command-line interface** to interact with a *database of playlists*.
    - Options should include:
        - Add a song, album, or playlist.
        - View details of all entities.
        - Search for an entity by name or ID.
        - Remove an entity by ID.
        - Shuffle a given playlist or list its contents.

5. (2 points) **Configuration and compilation**
    1. Develop a CMake script for easy compilation of the C++ library.
    2. Provide clear instructions on compiling the library.

6. (5 points) **Python bindings using pybind11**:
    1. Bind the C++ functions, classes and their methods to Python, properly handling exceptions.
    2. Ensure the Python interface is user-friendly and adheres to Python conventions.
    3. Write a Python script to demonstrate the usage of the classes.

7. (Optional) **Add functionality** to:
    1. Sort songs or albums by a specific property (e.g., name or duration).
    2. Write and read a database to/from file.
    3. Manage metadata like user-defined tags or playback statistics for songs.

Evaluation criteria:

- **Code organization and correctness**.
- Correct use of **inheritance** and **polymorphism** to model the problem.
- Effective use of **templates** for flexibility and type safety.
- Proper **memory management** and **exception handling**.
- **Intuitive user interface** with clear instructions on how to interact with the music database.
- Seamless **integration between C++ and Python**.