ECE250 – Project 0
Playlist of Songs Design Document

Nandita Lohani, UW UserID: nlohani

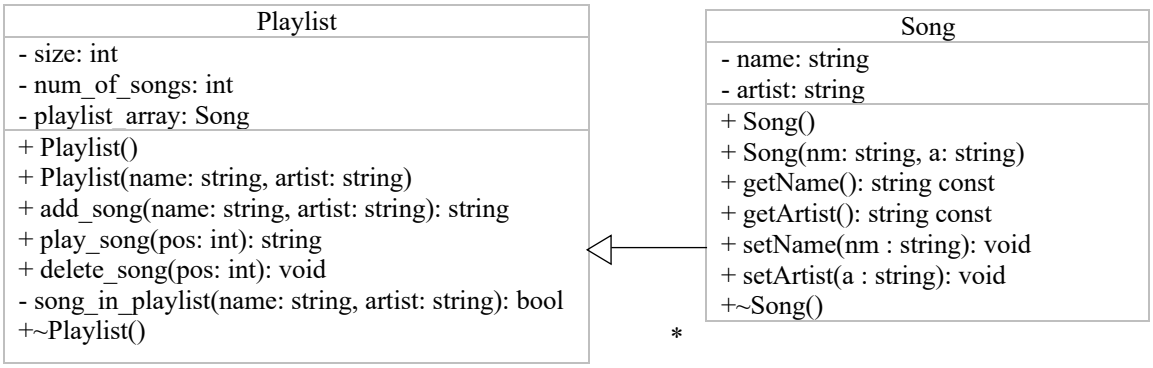
September 27, 2022

1. Overview of classes

Class	Song	Playlist
Description	The purpose of the Song class is to represent a song as a name and an artist. It provides getter and setter operations for a song such as getting the name and artist and setting the name and artist.	The Playlist class represents a playlist of songs as a dynamic array and provides operations such as creating a new playlist, adding a new song, playing and/or deleting an existing song.
Member Variables	name – name of the song (string) artist – artist of the song (string)	size – size of the playlist array (int) num_of_songs – tracks the number of songs in the playlist (int) *playlist_array – array that stores songs using objects from the Song class
Member Function	setName – passes a string parameter and sets it to name setArtist - passes a string parameter and sets it to artist getName – returns the name (string) getArtist – returns the artist (string)	in_playlist – private, helper function to check if a song already exists in the playlist (bool) add_song – passes two string parameters (name and artist) and sets them to the current Song object within the playlist, calls the private in_playlist function to avoid repeated entries play_song - passes an integer as the song position within the array and plays the entry in that position delete_song - passes an integer as the song position within the array, deletes the entry in that position and moves all entries after the deleted one by 1

The Playlist class creates a dynamic array of objects from the Song class through the playlist_array pointer variable. Each entry in the dynamic array consists of a name and artist from the Song class which are accessible using the getter functions and can be modified using the setter functions from the Song class.

2. UML class diagram



3. Details on design decisions

All constructors in the design include two constructors to make the program more flexible by allowing the user to call it in two ways as one constructor can assign default values to the variables and the second one only accepts user-defined values. The Playlist and the Song class both consist of a constructor that initializes the variables in the class with default values. The second constructor for the Song class assigns the name and artist of the song using the setter functions of the Song class. The second constructor for the Playlist class takes an input parameter and assigns it to the variable for the size of the array, sets the number of songs in the array to 0 and creates a dynamic playlist array of objects from the Song class.

The destructors for each class were designed to deallocate memory and to clean up for each class objects. Since the Playlist class dynamically allocates memory space for the playlist array, the destructor manually deletes the array using the delete function. The Song class only contains primitive variable data types and does not have any base classes, so a default destructor is sufficient destroys the data of all objects once there is no more use for them.

Since the getName and getArtist functions in the Song class only retrieve the values of the private member variable, they should not modify any data in the process. Thus, they are constant functions in the design.

4. Test cases

The testing strategy for the design was to test each section of the program before testing the program altogether. Testing and extracting the proper commands from the command line was done with various print statements within the driver file. The test for the functions and operations were tested as follows:

Test Areas	Test Cases
Creation	• Create the playlist array with user given size
add_song	• Add a song (name and artist) into the playlist • Attempt to add a restricted song into the playlist • Attempt to add more entries than the size of the array • Attempt to add a song that is already in the playlist (also checks if in_playlist function works)
play_song	• Play the song at a position the playlist • Attempt to play a song outside of the array size • Attempt to play a song that does not exist (position in the array has no entry)
delete_song	• Delete the song at a position and move the entries after the position by 1 • Attempt to delete a song that is outside of the array size • Attempt to delete a song at a position that is empty in the playlist • Play a song at the previously deleted position to check if deletion was successful (the song at that position should be replaced by song at the position + 1)
Other	• Create a playlist of size 0 and attempt to add songs • Create a playlist of a large size and add many songs to ensure program does not crash • Each function delivers the correct outputs for each case (i.e. success, can not play 0, etc.)

In addition to the tests, there were additional tests performed to ensure all functions work altogether. For example, a test case file was made with all the commands and each scenario.