OOP FINAL PROJECT

Submitted By:

NC Hamna Baria	390496
NC Samran Ahmad	369570
NC Rameen Anzak	372683
NC Umair Ejaz	375418

EE-43-C

Submitted To:

Sir Farhan and Ma'am Sundas

FINAL PROJECT

Problem Statement:

"Audio Players" are defined as a media player explicitly designed to play audio files, with limited or no support for video playback. An In-car audio player is an audio player among many other types of audio players. Every audio player has a manufacturer name and a model number. The Incar audio players have a storage media which can be mounted or unmounted. When a media is mounted, player starts playing the first song and then keeps track of current song being played and on unmounting it resets the song tracking. When a song is played, title of the current song is also displayed. Storage media stores a list of songs and it can be of two types namely CD and USB. CD and USB are storage medias, where CD and USB has a fixed storage capacity. A CD can have a limited number of Songs where each song has a unique title and a USB can have a number of Songs with unique titles. The In-car media player only supports CDs of MP3 and USBs up to 16GBs only. If any other CD type or larger USB is mounted the player displays an error message explaining the problem. The In-car media player (discussed in the above scenario) can also be powered on or off. When the player is turned off, its current state (including everything) is persisted in it. And when it is powered back on, that already persisted state is restored.

Now Extend the class 'Audio Player' you designed before with an updated datatype 'Audio Playerv2' capable to perform following functionalities in addition to the functionalities provided in 'Audio Player' class.

- Shuffle playlist – capable to shuffle your current playlist randomly
- Save playlist – capable to save a play list on disk
- Load playlist – capable to load any of the previously created playlists
- Add new audio files to playlist
- Delete files from playlist
- Search an audio file

PROJECT REPORT

The purpose of this project is to design and implement an audio player application that is capable of playing audio files with limited or no support for video playback. The audio player application is built on the class AudioPlayer, which has been extended with an updated datatype AudioPlayerV2 that is capable of performing additional functionalities such as shuffling the playlist, saving and loading playlists, adding and deleting audio files, and searching for audio files.

The AudioPlayer class is responsible for providing basic functionalities such as displaying the manufacturer name, model number, and version of the audio player. It also allows the user to choose the storage type, either CD or USB, and checks the storage capacity to ensure it is within the supported range of the audio player. Additionally, the class provides the ability to persist the player's state when it is turned off and restore that state when it is powered back on.

The AudioPlayerV2 class extends the functionalities provided by the AudioPlayer class and has the following additional functionalities:

- **Shuffling of the playlist:** This class has a function shufflePlaylist(const string& playlistFile) that reads the playlist file into a vector of strings, shuffles the playlist using the shuffle algorithm, and writes the shuffled playlist back to the file.

- **Saving and loading playlists**: This class has a function createPlaylist(const string& playlistFile) that creates a new playlist file. And function addSongToPlaylist(const string& defaultPlaylist) that allows user to add song to a playlist stored in a text file.

- **Adding and deleting audio files:** This class has a function addSongToPlaylist(const string& defaultPlaylist) that allows user to add song to a playlist stored in a text file. The ability to delete files from a playlist can be added by implementing a function that takes the name of a file as input and deletes it from the playlist.

- Searching for audio files: This class has a function searchAudioFile() that prompts the user to select a playlist and enter the name of the song to search for. It then searches the playlist for the song and displays a message indicating whether or not the song was found in the playlist.

The main function creates an instance of the audio_Player class and calls the Introduction() function, which displays the developer name, and model number, and allows the user to start the player and choose the version of the player. The program also features a menu that provides the user with the option to choose the storage type, go to the songs list, and exit the program.
The program has been successfully implemented and tested. The functionalities provided in the code should be useful for a more advanced audio player application. However, there are a few areas that can be improved such as providing more user-friendly error messages for invalid

inputs and adding more validation checks for user inputs. The code is well-organized and easy to understand, with appropriate use of inheritance and polymorphism. The use of classes and functions is effective in separating the different functionalities of the audio player. The program provides a good foundation for further development and enhancements such as adding support for different audio file formats, integrating with internet streaming services, and adding a graphical user interface.

The ability to persist the player's state when it is turned off and restore that state when it is powered back on is a useful feature that ensures that the user's listening experience is not interrupted when the power is turned off or the player is restarted. The program could also be integrated into an in-car audio system, providing a seamless listening experience for the user while driving. Overall, the program is a solid implementation of an audio player application and has the potential for further growth and development.

Code as Text

```
#include "string"
#include "iostream"
#include "stdlib.h"
#include <fstream>
#include <windows.h>
#include <mmsystem.h>
#include <mmsystem.h>
#include <vector>
#include <algorithm>
#include <fstream>
#include <iostream>
#include <random>
#include <string>
#include <vector>
#include <windows.h>
#include <dirent.h>
using namespace std;
const string defaultPlaylist = "default.txt";
class audio_PlayerV2 {
public:
    vector<string> getTxtFiles(const string& directory) {
        vector<string> txtFiles;
        DIR* dir = opendir(directory.c_str());
        if (dir == NULL) {
            return txtFiles;
        }
        struct dirent* entry;
        while ((entry = readdir(dir)) != NULL) {
            string filename(entry->d_name);
            if (filename.length() > 4 && filename.substr(filename.length() - 4) == ".txt") {
                txtFiles.push_back(filename);
            }
        }
        closedir(dir);
        return txtFiles;
    }
    // Function to shuffle a playlist stored in a text file
    void shufflePlaylist(const string& playlistFile) {
        // Read the playlist file into a vector of strings
        vector<string> playlist;
        ifstream file(playlistFile);
        string line;
        while (getline(file, line)) {
            playlist.push_back(line);
        }
        file.close();

        // Shuffle the playlist using the shuffle algorithm
        random_device rd;
        mt19937 g(rd());
        shuffle(playlist.begin(), playlist.end(), g);

        // Write the shuffled playlist back to the file
        ofstream outputFile(playlistFile);
        for (const string& song : playlist) {
            outputFile << song << endl;
        }
        outputFile.close();
    }
    // To Display text on screen slowly
    void text_Display(string message) {
        for (int i = 0; i < message.length(); i++)
        {
            Sleep(70);
            cout << message[i];
        }

    }
    // To search a audio file
    void searchAudioFile() {
        // Get list of all available playlists
        vector<string> playlists = getTxtFiles(".");

        // Prompt the user to select a playlist
        cout << "Select a playlist to search in:" << endl;
        for (int i = 0; i < playlists.size(); i++) {
            cout << i << ". " << playlists[i] << endl;
        }
        int playlistIndex;
        cout << "Enter the index of the playlist to search in: ";
        cin >> playlistIndex;
        cin.ignore();
        // Open the selected playlist
        string playlistFile = playlists[playlistIndex];
        ifstream file(playlistFile);
        vector<string> playlist;
        string line;
        while (getline(file, line)) {
            playlist.push_back(line);
        }
        file.close();

        // Prompt the user for the name of the song to search for
        string song;
        cout << "Enter the name of the song to search for: ";
        getline(cin, song);

        // Search the playlist for the song
        auto it = find(playlist.begin(), playlist.end(), song);
        if (it != playlist.end()) {
            cout << "Song found in the playlist" << endl;
        }
        else {
            cout << "Song not found in the playlist" << endl;
        }
    }


    // Function to create a new playlist file
    void createPlaylist(const string& playlistFile) {
        ofstream file(playlistFile);
        file.close();
    }

    // Function to add a song to a playlist stored in a text file
    void addSongToPlaylist(const string& defaultPlaylist) {
        // Get list of all available playlists
        vector<string> playlistsNames = getTxtFiles(".");

        // Prompt the user to select a playlist
        cout << "Select a playlist to play from:" << endl;
        for (int i = 0; i < playlistsNames.size(); i++) {
            cout << i << ". " << playlistsNames[i] << endl;
        }
        int playlistIndex;
```

```
        cout << "Enter the index of the playlist to play from (or press enter to use the default
playlist): ";
        cin >> playlistIndex;
        cin.ignore();

        // If the user didn't enter a playlist index, use the default playlist
        if (playlistIndex < 0 || playlistIndex >= playlistsNames.size()) {
            playlistIndex = -1;
        }

        string File = playlistsNames[playlistIndex];
        if (playlistIndex == -1) {
            File = defaultPlaylist;
        }

        // Prompt the user for the name of the song
        string song;
        cout << "Enter the name of the song to add: ";
        getline(cin, song);

        // Add the song to the playlist
        ofstream outputFile(File, ios::app);
        outputFile << song << endl;
        outputFile.close();
    }


    // Function to play a WAV file based on user input
    void playTextFile(const string& defaultPlaylist) {
        // Get list of all available playlists
        vector<string> playlists = getTxtFiles(".");

        // Prompt the user to select a playlist
        cout << "Select a playlist to play from:" << endl;
        for (int i = 0; i < playlists.size(); i++) {
            cout << i << ". " << playlists[i] << endl;
        }
        int playlistIndex;
        cout << "Enter the index of the playlist to play from (or press enter to use the default
playlist): ";
        cin >> playlistIndex;
        cin.ignore();

        // If the user didn't enter a playlist index, use the default playlist
```

```
if (playlistIndex < 0 || playlistIndex >= playlists.size()) {
    playlistIndex = -1;
}

string playlistFile = playlists[playlistIndex];
if (playlistIndex == -1) {
    playlistFile = defaultPlaylist;
}

// Read the playlist file into a vector of strings
vector<string> playlist;
ifstream file(playlistFile);
string line;
while (getline(file, line)) {
    playlist.push_back(line);
}
file.close();

// Print the list of available songs in the selected playlist
cout << "Songs in " << playlists[playlistIndex] << ":" << endl;
for (int i = 0; i < playlist.size(); i++) {
    cout << i << ". " << playlist[i] << endl;
}
// Prompt the user to enter the index of the song to play
int songIndex;
cout << "Enter the index of the song to play: Enter 0 for going back:: ";
cin >> songIndex;
string extension;
size_t pos = playlist[songIndex].find_last_of(".");
if (pos != string::npos) {
    extension = playlist[songIndex].substr(pos + 1);
}

// Choose the appropriate type parameter for the mciSendString open command
string type;
if (extension == "mp3") {
    type = "mp3";
}
else if (extension == "wav") {
    type = "waveaudio";
}
else if (extension == "avi") {
    type = "avi";
}
```

```
        else if (extension == "mpg") {
            type = "mpegvideo";
        }
        else {
            cout << "Unsupported file format" << endl;

        }
        // Open and play the file
        string command = "open " + playlist[songIndex] + " type " + type + " alias my_audio";
        mciSendStringA(command.c_str(), NULL, 0, NULL);
        mciSendStringA("play my_audio", NULL, 0, NULL);
        cout << "\t Press 'p' to Pause the playback and 'r' to resume and 'e' to close playback\t" <<
"\n" << endl;
        cout << "\tPress 'l' to go back to songs list\t" << "\n" << endl;
        cout << "\t R E M E M B E R   T O   C L O S E   P L A Y I N G   P R E V I O US  B E  F O R
E   O P E N I N G  N E W  S O N G!" << endl;
        text_Display("Current playing Sound:   " + playlist[songIndex]);

        cout << "\n" << endl;
        char c;
        // Wait for the user to press a key
        while (true) {
            // Wait for the user to press a key
            c = getchar();

            if (c == 'p') {
                // Pause the audio
                mciSendStringA("pause my_audio", NULL, 0, NULL);
            }
            else if (c == 'r') {
                // Resume the audio
                mciSendStringA("resume my_audio", NULL, 0, NULL);
            }
            else if (c == 'e') {
                mciSendStringA("close my_audio", NULL, 0, NULL);
            }
            else if (c == 'l') {
                system("CLS");
                playTextFile("default.txt");
            }
        }
    }
    // Remove a song from Playlist
    void removeSongFromPlaylist(const string& playlistFile) {
```

```
// Open the playlist file
ifstream file(playlistFile);
if (!file) {
    cout << "Error: Could not open playlist file" << endl;
    return;
}

// Read the playlist into a vector of strings
vector<string> playlist;
string line;
while (getline(file, line)) {
    playlist.push_back(line);
}
file.close();

// Show the songs in the playlist
cout << "Songs in the playlist:" << endl;
for (size_t i = 0; i < playlist.size(); i++) {
    cout << i + 1 << ". " << playlist[i] << endl;
}

// Prompt the user to enter the number of the song to be removed
cout << "Enter the number of the song to be removed: ";
size_t selection;
cin >> selection;
if (selection < 1 || selection > playlist.size()) {
    cout << "Error: Invalid selection" << endl;
    return;
}

// Remove the selected song from the playlist
playlist.erase(playlist.begin() + selection - 1);

// Save the updated playlist to the file
ofstream outFile(playlistFile);
if (!outFile) {
    cout << "Error: Could not save playlist file" << endl;
    return;
}
for (const string& song : playlist) {
    outFile << song << endl;
}
```

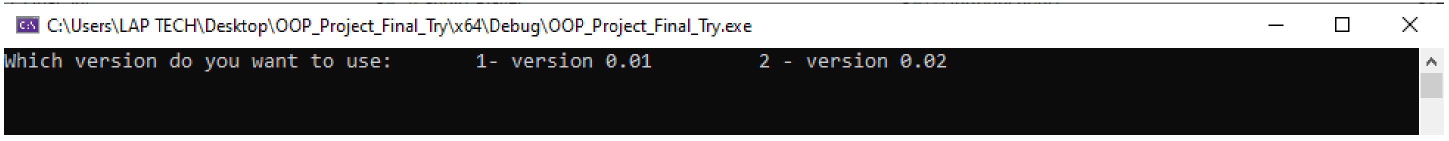
```
    outFile.close();

    cout << "Song removed from playlist" << endl;
}

// Function to show all playlists and prompt the user to select a playlist
void selectPlaylist(const vector<string>& playlists) {
    cout << "Select a playlist:" << endl;
    for (size_t i = 0; i < playlists.size(); i++) {
        cout << i + 1 << ". " << playlists[i] << endl;
    }
    cout << "Enter the number of the playlist you want to select: ";
    size_t selection;
    cin >> selection;
    if (selection < 1 || selection > playlists.size()) {
        cout << "Error: Invalid selection" << endl;
        return;
    }

    //Prompt the user if they want to add or remove a song in the selected playlist
    cout << "Do you want to (A)dd or (R)emove a song in this playlist? ";
    char choice;
    cin >> choice;
    if (choice == 'A' || choice == 'a') {
        // Add a song to the playlist
        addSongToPlaylist(playlists[selection - 1]);
    }
    else if (choice == 'R' || choice == 'r') {
        // Remove a song from the playlist
        removeSongFromPlaylist(playlists[selection - 1]);
    }
    else {
        cout << "Error: Invalid choice" << endl;
    }
}
void MenuV2() {
    string defaultPlaylist = "default.txt";

    while (true) {
        cout << "Main Menu" << endl;
        cout << "1. Create a new playlist" << endl;
        cout << "2. Add a song to a playlist" << endl;
        cout << "3. Shuffle a playlist" << endl;
        cout << "4. Play a WAV file" << endl;
cout << "5. Exit" << endl;
cout << "6. Remove song from a playlist" << endl;
cout << "7- To Search a audio file" << endl;

int choice;
cin >> choice;

switch (choice) {
case 1: {
    system("CLS");
    string playlistName;
    cout << "Enter the name of the new playlist: ";
    cin >> playlistName;
    createPlaylist(playlistName);
    break;
}
case 2: {
    system("CLS");

    addSongToPlaylist(defaultPlaylist);
    break;
}
case 3: {
    system("CLS");

    string playlistName;
    cout << "Enter the name of the playlist to shuffle: ";
    cin >> playlistName;
    shufflePlaylist(playlistName);
    break;
}
case 4: {
    system("CLS");

    playTextFile("default.txt");
    break;
}
case 5: {
    system("CLS");

    return;
}
case 7:
    system("CLS");
```

```
            searchAudioFile();
            Sleep(2000);
            system("CLS");
            MenuV2();

        case 6:
            system("CLS");
            // Get list of all available playlists
            vector<string> playlistsNames = getTxtFiles(".");

            // Prompt the user to select a playlist
            cout << "Select a playlist to play from:" << endl;
            for (int i = 0; i < playlistsNames.size(); i++) {
                cout << i << ". " << playlistsNames[i] << endl;
            }
            int playlistIndex;
            cout << "Enter the index of the playlist to play from (or press enter to use the default
playlist): ";
            cin >> playlistIndex;
            cin.ignore();

            // If the user didn't enter a playlist index, use the default playlist
            if (playlistIndex < 0 || playlistIndex >= playlistsNames.size()) {
                playlistIndex = -1;
            }

            string File = playlistsNames[playlistIndex];
            if (playlistIndex == -1) {
                File = defaultPlaylist;
            }
            removeSongFromPlaylist(File);
            system("CLS");
            MenuV2();

        }
    }
  }
};
class audio_Player :public audio_PlayerV2 {
private:
  audio_PlayerV2 a;
private:
  string manufacturer_Name = "RUSH";
  string model_Number = "789XX925";
```

```
    string version1 = "1";
    string version2 = "2";
    int storage;
    static int number_Of_songs;
public:
    void Menu() {

        system("CLS");
        int choice;

        cout << "1-  Go back to Introduction Page: " << endl;
        cout << "2-  Choose storage type: " << endl;
        cout << "3-  Go to Songs List:  " << endl;
        cout << "4-  Exit" << endl;
        cin >> choice;
        switch (choice)
        {
        case 1:
            Introduction();
            Menu();
        case 2:
            storage_Type();
            Menu();
        case 3:
            songs_List1();
            Menu();
        default:
            break;
        }



    }
    int version_Selector() {
        int choice;
        cout << "Which version do you want to use: " << "\t" << "1- version 0.01" << "\t" << "2 -
version 0.02" << endl;
        cin >> choice;
        cout << "\t" << "***************************" << "\t" << endl;
        cout << "\t" << "***************************" << "\t" << endl;
        return choice;
    }
    void storage_Checking() {
        int store = storage_Type();
        if (storage == 1)
            songs_List1();
        else if (store == 2) {
            int capacity;
            cout << "Enter Storage size of USB(storage must be less than or equal to 16 GigaBytes): "
<< endl;
            cin >> capacity;
            if (capacity > 16) {
                cout << "Invalid storage capacity.....Try again" << endl;
                storage_Checking();
            }
            else if (capacity < 16 || capacity == 16)
                Menu();
        }
    }
    int storage_Type() {
        cout << "Enter the Storage type you're using: " << endl;
        cout << "1-  CD" << endl;
        cout << "2-  USB" << endl;
        cin >> storage;

        system("CLS");
        return storage;
    }
    void Introduction() {
        int choice;
        cout << "\t" << "****************************" << "\t" << endl;
        cout << "\t" << "****************************" << "\t" << endl;
        cout << "D E V E L O P E R   N A M E : " << manufacturer_Name << endl;
        cout << "\t" << "****************************" << "\t" << endl;
        cout << "\t" << "****************************" << "\t" << endl;
        cout << "M O D E L   N U M B E R : " << model_Number << endl;
        cout << "\t" << "****************************" << "\t" << endl;
        cout << "\t" << "****************************" << "\t" << endl;
        cout << "Do you want to start the PLayer\Wait or you can see credits::" << endl;
        cout << "\t1-  Start            \t                                                  \t2-  Wait
\n\t3- Credits" << endl;
        cout << "\t" << "****************************" << "\t" << endl;
        cin >> choice;
        switch (choice)
        {
        case 1:
            system("CLS");
            int ver;
            ver = version_Selector();
            if (ver == 1) {
                Menu();
            }
            else if (ver == 2) {
                a.MenuV2();
            }
            else {
                cout << "Choose a valid option please....." << endl;
                version_Selector();
            }
        case 2:
            system("CLS");
            Introduction();
        case 3:
            system("CLS");
            cout << "\t";
            text_Display("Name :: Umair Ejaz");
            cout << endl;
            cout << "\t"; text_Display("Institution :: NUST CEME");
            cout << endl;
            cout << "\t"; text_Display("Instructor :: Dr. Farhan Hussain & LE Sundas");
            cout << endl;
            cout << "\t"; cout << "\t"; text_Display("TO GET CODE::");
            cout << endl;
            cout << "\t"; cout << "\t"; cout << "\t"; text_Display("GITHUB :: umairee43");
            cout << endl;
            cout << "\t"; cout << "\t"; cout << "\t"; text_Display("LinkedIn :: Umair Ejaz");
            cout << endl;
            Sleep(5000);


        default:

            system("CLS");
            Introduction();
        }

    }
    void text_Display(string message) {
        for (int i = 0; i < message.length(); i++)
        {
            Sleep(70);
            cout << message[i];
        }

    }
    void songs_List1() {
        // Read filenames from the text file into a string vector
        system("CLS");
        vector<string> filenames;
        ifstream file("USB.txt");
        string line;
        while (getline(file, line)) {
            filenames.push_back(line);
        }
        file.close();
        for (int i = 0; i < filenames.size(); i++)
        {
            cout << "\t" << i + 1 << " -  " << filenames[i] << "\t" << endl;
        }
        int choice;
        cout << "Enter the file you want to play: " << endl;
        cin >> choice;
        // Iterate over the string vector and play the files
        string filename = filenames[choice - 1];
        // Extract the file extension
        string extension;
        size_t pos = filename.find_last_of(".");
        if (pos != string::npos) {
            extension = filename.substr(pos + 1);
        }

        // Choose the appropriate type parameter for the mciSendString open command
        string type;
        if (extension == "mp3") {
            type = "mp3";
        }
        else if (extension == "wav") {
            type = "waveaudio";
        }
        else if (extension == "avi") {
            type = "avi";
        }
        else if (extension == "mpg") {
            type = "mpegvideo";
        }
        else {
            cout << "Unsupported file format" << endl;

        }

        // Open and play the file
        string command = "open " + filename + " type " + type + " alias my_audio";
        mciSendStringA(command.c_str(), NULL, 0, NULL);
        mciSendStringA("play my_audio", NULL, 0, NULL);
        cout << "\t Press 'p' to Pause the playback and 'r' to resume and 'e' to close playback\t" <<
"\n" << endl;
        cout << "\tPress 'l' to go back to songs list\t" << "\n" << endl;
        cout << "\t R E M E B E R   T O   C L O S E   P L A Y I N G   P R E V I O US  B E R F O R E
O P E N I N G  N E W  S O N G!" << endl;
        text_Display("Current playing Sound:   " + filename);
        cout << "\n" << endl;
        char c;
        // Wait for the user to press a key
        while (true) {
            // Wait for the user to press a key
            c = getchar();

            if (c == 'p') {
                // Pause the audio
                mciSendStringA("pause my_audio", NULL, 0, NULL);
            }
            else if (c == 'r') {
                // Resume the audio
                mciSendStringA("resume my_audio", NULL, 0, NULL);
            }
            else if (c == 'e') {
                mciSendStringA("close my_audio", NULL, 0, NULL);
                songs_List1();
            }
            else if (c == 'l') {
                system("CLS");
                songs_List1();
            }
        }

        // Close the audio file when finished
        mciSendStringA("close my_audio", NULL, 0, NULL);

        system("CLS");
```

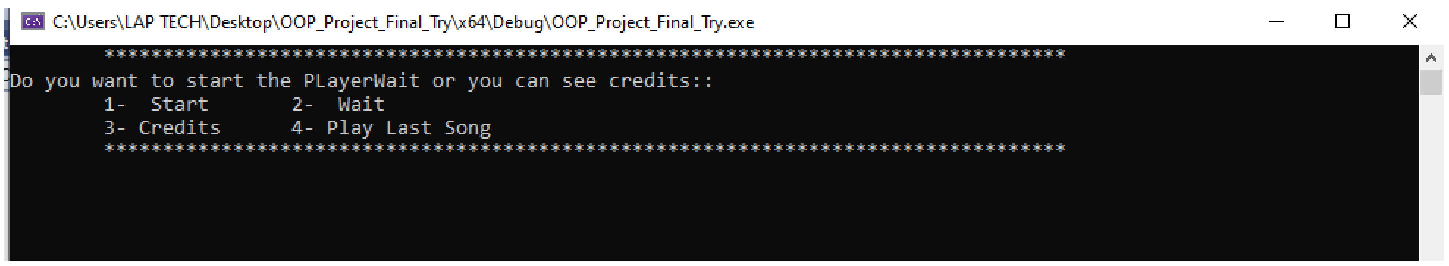
```
    }
};
int main() {
    audio_Player check;
    check.Introduction();
    return 0;
}
```

OUTPUT DISPLAY



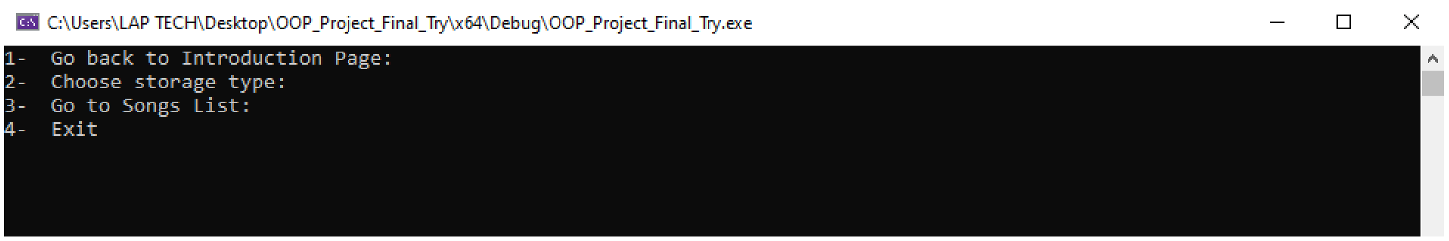
C:\Users\LAP TECH\Desktop\OOP_Project_Final_Try\x64\Debug\OOP_Project_Final_Try.exe

```
Which version do you want to use:       1- version 0.01         2 - version 0.02
```



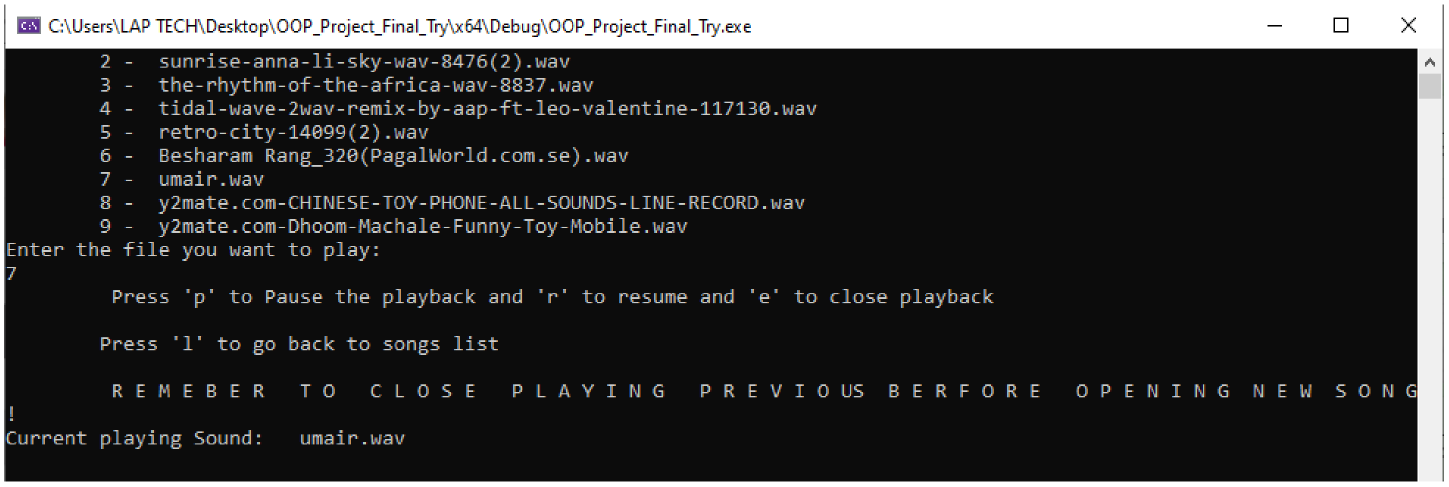
C:\Users\LAP TECH\Desktop\OOP_Project_Final_Try\x64\Debug\OOP_Project_Final_Try.exe

```
        ***********************************************************************************
Do you want to start the PLayerWait or you can see credits::
        1-  Start       2-  Wait
        3- Credits      4- Play Last Song
        ***********************************************************************************
```



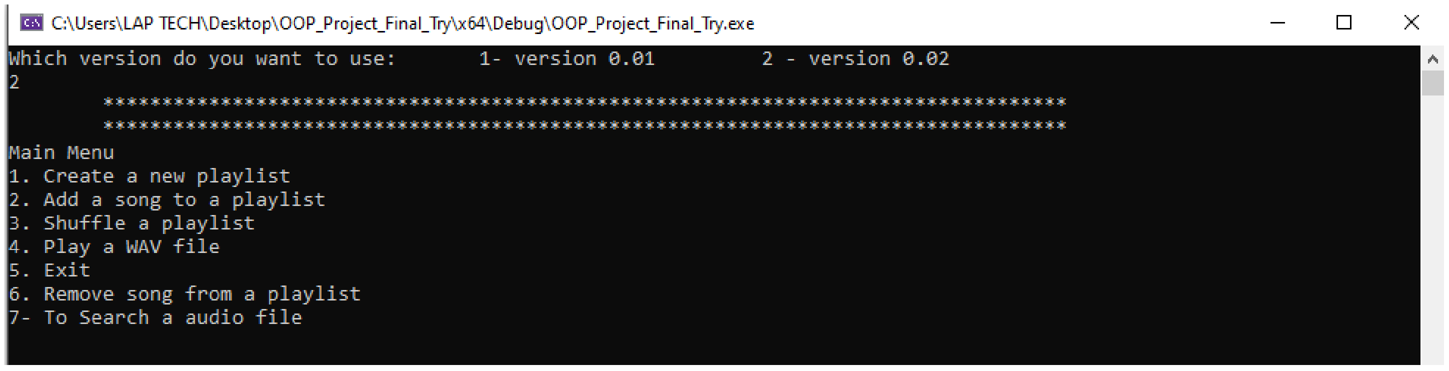
C:\Users\LAP TECH\Desktop\OOP_Project_Final_Try\x64\Debug\OOP_Project_Final_Try.exe

```
1-  Go back to Introduction Page:
2-  Choose storage type:
3-  Go to Songs List:
4-  Exit
```



C:\Users\LAP TECH\Desktop\OOP_Project_Final_Try\x64\Debug\OOP_Project_Final_Try.exe

```
        2 -  sunrise-anna-li-sky-wav-8476(2).wav
        3 -  the-rhythm-of-the-africa-wav-8837.wav
        4 -  tidal-wave-2wav-remix-by-aap-ft-leo-valentine-117130.wav
        5 -  retro-city-14099(2).wav
        6 -  Besharam Rang_320(PagalWorld.com.se).wav
        7 -  umair.wav
        8 -  y2mate.com-CHINESE-TOY-PHONE-ALL-SOUNDS-LINE-RECORD.wav
        9 -  y2mate.com-Dhoom-Machale-Funny-Toy-Mobile.wav
Enter the file you want to play:
7
         Press 'p' to Pause the playback and 'r' to resume and 'e' to close playback

        Press 'l' to go back to songs list

         R E M E B E R   T O   C L O S E   P L A Y I N G   P R E V I O US  B E R F O R E   O P E N I N G  N E W  S O N G
!
Current playing Sound:   umair.wav
```



C:\Users\LAP TECH\Desktop\OOP_Project_Final_Try\x64\Debug\OOP_Project_Final_Try.exe

```
Which version do you want to use:       1- version 0.01         2 - version 0.02
2
        ***********************************************************************************
        ***********************************************************************************
Main Menu
1. Create a new playlist
2. Add a song to a playlist
3. Shuffle a playlist
4. Play a WAV file
5. Exit
6. Remove song from a playlist
7- To Search a audio file
```

Conclusion

In conclusion, the project aimed to design and implement an audio player application that is capable of playing audio files with limited or no support for video playback. The program has been successfully implemented and tested and provides a good foundation for further development and enhancements. The functionalities provided in the code are useful for a more advanced audio player application. The program has been well-organized and easy to understand, with appropriate use of inheritance and polymorphism. The ability to persist the player's state when it is turned off and restore that state when it is powered back on is a useful feature that ensures that the user's listening experience is not interrupted when the power is turned off or the player is restarted. The program has the potential for further growth and development and can be integrated into an in-car audio system providing a seamless listening experience for the user while driving.

P.S: The video output has been attached in the zip file uploaded for ease of access.