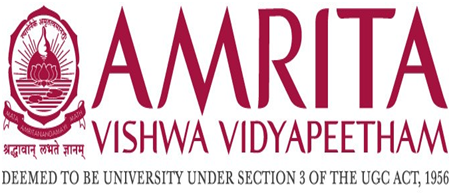
**MUSIC PLAYER IN JAVA**

REPORT SUBMITTED BY

|  |  |
| --- | --- |
| **BARANIDHARAN SELVARAJ****[CB.SC.U4AIE23015]** | **KEERTHIVASAN S V****[CB.SC.U4AIE23037]** |
| **MOPURU SAI BAVESH REDDY****[CB.SC.U4AIE23044]** | **SUTHEKSHAN KARUPPUSAMI****[CB.SC.U4AIE23067]** |

**AS A PART OF SUB JECT**

**22AIE111 OBJECT ORIENTED PROGRAMMING IN JAVA**



**CENTRE FOR COMPUTATIONAL ENGINEERING AND NETWORKING**

**AMRITA SCHOOLOF ARTIFICIAL INTELLIGENCE**

**AMRITA VISHWA VIDYAPEETHAM**

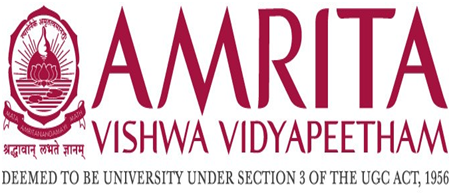
COIMBATORE-641112(INDIA)

**JUNE -2024**

**AMRITA SCHOOLOF ARTIFICIAL INTELLIGENCE**

**AMRITA VISHWA VIDYAPEETHAM**

**COIMBATORE-641112**



**BONAFIDE CERTIFICATE**

This is to certify that the report entitled “\_MUSIC PLAYER IN JAVA**\_**” submitted by \_\_\_\_\_\_**GROUP-17**\_\_\_\_\_\_, for the award of the Degree of Bachelor of Technology in the “CSE(AI) ” is a bonafide record of the work carried out by her under our guidance and supervision at Amrita School of Artificial Intelligence, Coimbatore.

**Mr.Vipin das**

Project Guide

**Dr.K.P.Soman**

Professor and Head CEN

Submitted for the university examination held on 03-06-2024

**AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE**

**AMRITA VISHWA VIDYAPEETHAM**

COIMBATORE - 641 112

**DECLARATION**

We hereby declare that this thesis entitled “MUSIC PLAYER IN JAVA”, is the record of the original work done by me under the guidance of Mr.Vipin Das, Centre for Computational Engineering and Networking, Amrita School of Artificial Intelligence, Coimbatore. To the best of my knowledge this work has not formed the basis for the award of any degree/diploma/ associate ship/fellowship/or a similar award to any candidate in any University.

**Signature of the Faculty**

**Names Roll Numbers**

**BARANIDHARAN SELVARAJ CB.SC.U4AIE22015**

**KEERTHIVASAN S V CB.SC.U4AIE22037**

**MOPURU SAI BAVESH REDDY CB.SC.U4AIE22044**

**SUTHEKSHAN KARUPPUSAMI CB.SC.U4AIE22067**

**Place: Ettimadai**

**Date:03-06-2024**

**Contents**

Acknowledgement……………………………………………………………………………………………………5

1.**MODULE-1: APP………………………………………………………………………………………………………………….…....6**

1.1.**MODULE DESCRIPTION………………………………………………………………………………………………………….…6**

1.2. **MODULE IMPLEMENTATION…………………………………………………………………………………………….…....6**

1.3. **RELAVANCE OF MODULE……………………………………………………………………………………………….….……6**

1.4.**UML DIAGRAM OF THE MODULE…………………………………………………………………………………..………..7**

**1.5.APPLICATION OUTPUT………………………………………………………………………………………………….…........8**

2. **MODULE-2: MUSIC PLAYER GUI………………………………………………………………………………………….……..8**

2.1. **MODULE DESCRIPTION……………………………………………………………………………………………….……...8**

2.2. **MODULE IMPLEMENTATION………………………………………………………………………………………….……8**

2.3. **RELAVANCE OF MODULE…………………………………………………………………………………….……………..9**

2.4. **UML DIAGRAM OF THE MODULE………………………………………………………………………….…………….10**

**2.5.APPLICATION OUTPUT………………………………………………………………………………………….…………….10**

3. **MODULE-3: MUSIC PLAYER…………………………………………………………………………………….………………...11**

3.1. **MODULE DESCRIPTION……………………………………………………………………………………………………….11**

3.2. **MODULE IMPLEMENTATION………………………………………………………………………………………….……11**

3.3. **RELAVANCE OF MODULE…………………………………………………………………………………….……………..12**

3.4. **UML DIAGRAM OF THE MODULE…………………………………………………………………….………..………..13**

4. **MODULE-4: SONG……………………………………………………………………………………………………………..………14**

4.1. **MODULE DESCRIPTION………………………………………………………………………………………..……………..14**

4.2. **MODULE IMPLEMENTATION………………………………………………………………………………………..……..14**

4.3. **RELAVANCE OF MODULE…………………………………………………………………………………………..…….….15**

4.4. **UML DIAGRAM OF THE MODULE………………………………………………………………………….……….…....15**

5. **MODULE-5: MUSIC PLAYER DIALOG……………………………………………………………………………….……..…..16**

5.1. **MODULE DESCRIPTION…………………………………………………………………………………….……………….…16**

5.2. **MODULE IMPLEMENTATION…………………………………………………………………….……………………….…16**

5.3. **RELAVANCE OF MODULE……………………………………………………………………………………………………..17**

5.4. **UML DIAGRAM OF THE MODULE…………………………………………………………………………………...…….17**

**5.5.APPLICATION OUTPUT……………………………………………………………………………………………………….….18**

**6. CONCLUSION………………………………………………………………………………………………………………………….…..19**

# **Acknowledgement**

We would like to express our special thanks of gratitude to our teacher (Mr.Vipin Das Sir), who gave us the golden opportunity to do this wonderful project on the topic (MUSIC PLAYER IN JAVA), which also helped us in doing a lot of Research and we came to know about so many new things. We are thankful for the opportunity given. We would also like to thank our group members, as without their cooperation, we would not have been able to complete the project within the prescribed time.

1. **MODULE-1: App.java**

1.1. **MODULE DESCRIPTION**

The "App" module serves as the main entry point for a Java Swing-based music player application. It initializes the graphical user interface (GUI) and handles the creation and visibility of the main application window. Additionally, it demonstrates the use of the ‘Song’ class to manage and retrieve metadata from an MP3 file.

1.2. **MODULE IMPLEMENTATION**

The App class is implemented as follows:

Main Method: The ‘main’ method utilizes ‘SwingUtilities.invokeLater’ to ensure that the GUI updates are performed on the Event Dispatch Thread (EDT), which is the proper way to create and manipulate Swing components.

Runnable Implementation: Within the ‘invokeLater’ method, a new ‘Runnable’ is instantiated and its ‘run’ method is overridden. This method creates and displays an instance of ‘MusicPlayerGUI’.

Song Class Usage: The ‘Song’ class is instantiated with the path to an MP3 file. The title and artist of the song are then retrieved and printed to the console using the ‘getSongTitle()’ and ‘getSongArtist()’ methods.

1.3. **RELAVANCE OF MODULE**

The App module is crucial for the following reasons:

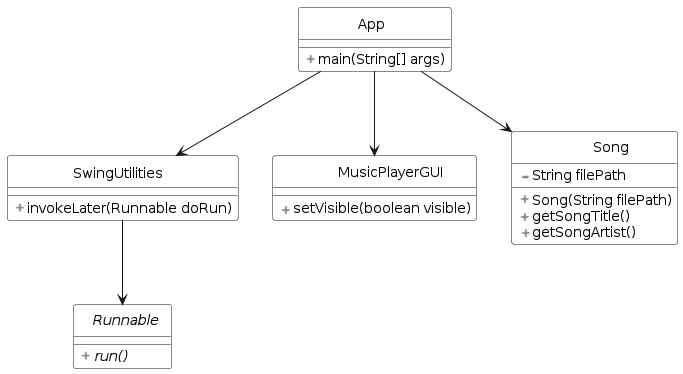
**Initialization**: It serves as the starting point of the application, ensuring that the GUI is correctly initialized and displayed.

**Thread Safety**: By using ‘SwingUtilities.invokeLater’, it guarantees that all GUI-related operations are performed in a thread-safe manner, avoiding potential concurrency issues.

**Demonstration**: It provides a basic demonstration of how the ‘Song’ class can be used to handle MP3 files, showcasing the integration of different components of the application.

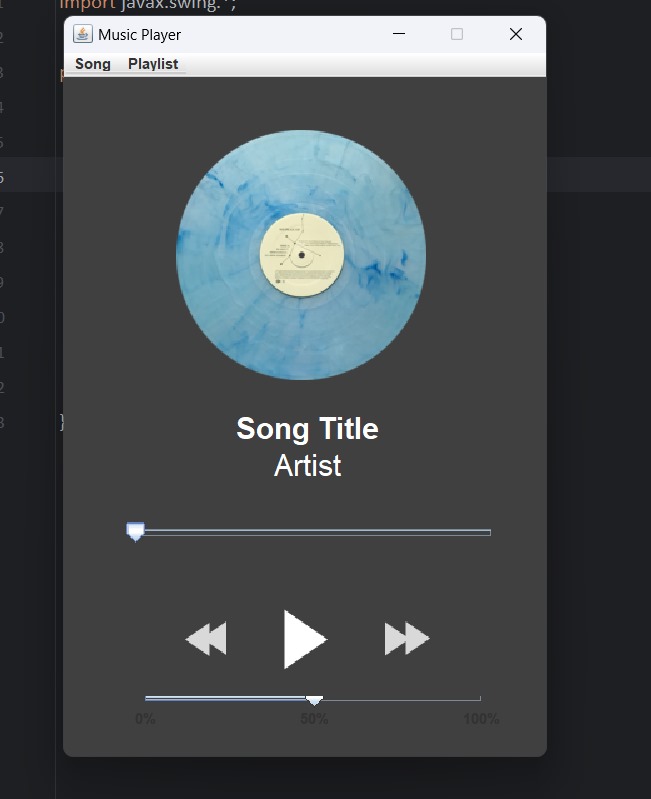
**User Interaction**: It sets up the environment where users can interact with the music player, making it the cornerstone of the user experience.

1.4. **UML DIAGRAM OF THE MODULE**

****

**Figure 1: UML Diagram for App.java**

**1.5 Application OUTPUT**

****

**Figure 1.1: Music Player GUI**

2. **MODULE-2: MusicPlayerGUI.java**

2.1. **MODULE DESCRIPTION**

The "MusicPlayer" module serves as the core functionality provider for the music player application. It is responsible for managing playback, controlling the playlist, and interacting with the GUI to reflect the current state of the player. The module uses the JLayer library to handle MP3 playback and provides methods for loading songs, playing, pausing, stopping, and navigating through a playlist.

2.2. **MODULE IMPLEMENTATION**

The‘MusicPlayer‘ class is implemented as follows:

**Fields**:

‘playSignal’: A synchronization object to manage playback state changes.

‘musicPlayerGUI’: A reference to the GUI for updating its state.

‘CurrentSong’*:* Stores the current song being played.

‘playlist’: A list to manage a collection of songs.

‘currentPlaylistIndex’: Tracks the current song's index in the playlist.

‘advancedPlayer’: An instance of ‘AdvancedPlayer’ from the JLayer library to handle MP3 playback.

‘isPaused’, ‘songFinished’, ‘pressedNext’, ‘pressedPrev’: Flags to manage the playback state.

‘currentFrame’,’currentTimeInMilli’: Track playback position for pausing and resuming.

**Constructor**:

Initializes the ‘musicPlayerGUI’ and sets up the initial state.

**Methods**:

‘getCurrentSong()’: Returns the current song.

‘setCurrentFrame(int frame)’,’setCurrentTimeInMilli(int timeInMilli)’: Setters for updating playback state.

‘loadSong(Song song)’: Loads and plays a single song.

‘loadPlaylist(File playlistFile)’: Loads a playlist from a file and plays the first song.

‘pauseSong()’: Pauses the current song.

‘stopSong()’: Stops the current song.

‘nextSong()’,’prevSong()’: Navigate to the next or previous song in the playlist.

‘playCurrentSong()’: Plays the current song, handling both fresh starts and resumes.

‘startMusicThread()’: Starts a new thread to handle music playback.

‘startPlaybackSliderThread()’: Starts a new thread to update the playback slider.

‘playbackStarted(PlaybackEventevt)’: Callback for when playback starts.

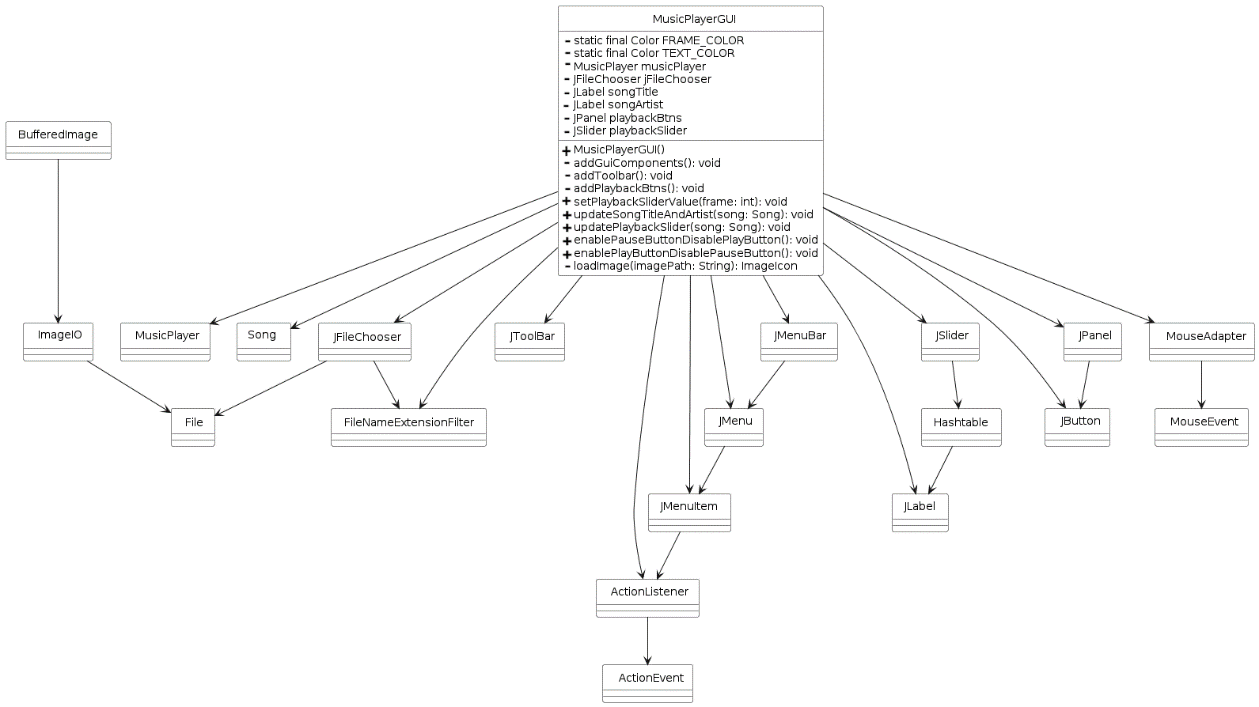
‘playbackFinished(PlaybackEventevt)’: Callback for when playback finishes or the player stops.

2.3. **RELAVANCE OF MODULE**

The ‘ MusicPlayer ‘ module is integral to the functionality and user experience of the music player application:

* **Playback Control**: It manages all aspects of audio playback, including starting, pausing, stopping, and resuming music.
* **Playlist Management**: It handles the loading and navigation of playlists, allowing users to enjoy multiple songs seamlessly.
* **GUI Interaction**: It updates the GUI to reflect the current state of playback, enhancing the user experience by providing visual feedback.
* **Thread Management**: It ensures smooth playback and GUI updates by using separate threads for music playback and slider updates.
* **Event Handling**: It listens for playback events to update the application state and manage transitions between songs effectively.

2.4. **UML DIAGRAM OF THE MODULE**

****

**Figure 2: UML Diagram for MusicPlayerGUI.java**

**2.5 Application OUTPUT**

****

**FIGURE 1.2**

3. **MODULE-3: MUSIC PLAYER**

3.1. **MODULE DESCRIPTION**

The "MusicPlayer" module in this music player application is designed to provide the core functionality required for audio playback. It interacts with the user interface, manages a playlist, and handles song playback, including starting, pausing, stopping, and transitioning between songs. The module utilizes the JLayer library for decoding and playing MP3 files and ensures smooth user interaction by synchronizing playback controls and updating the GUI in real-time.

3.2. **MODULE IMPLEMENTATION**

The ‘MusicPlayer’ class implements the main functionality of the music player, including loading songs and playlists, managing playback state, and handling playback events. The implementation details are as follows:

Fields:

The MusicPlayer class has the following main components:

* Instance variables:
  + musicPlayerGUI: An instance of the MusicPlayerGUI class, which is used to update the GUI components.
  + currentSong: An instance of the Song class, representing the currently playing song.
  + playlist: An ArrayList of Song objects, representing the current playlist.
  + currentPlaylistIndex: An integer variable that keeps track of the index of the currently playing song in the playlist.
  + advancedPlayer: An instance of the AdvancedPlayer class from the JLayer library, which is used to play the audio files.
  + isPaused: A boolean flag indicating whether the music playback is currently paused.
  + songFinished: A boolean flag indicating whether the current song has finished playing.
  + pressedNext: A boolean flag indicating whether the user has requested to play the next song in the playlist.
  + pressedPrev: A boolean flag indicating whether the user has requested to play the previous song in the playlist.
  + currentFrame: An integer variable that keeps track of the current frame position in the audio file.
  + currentTimeInMilli: An integer variable that keeps track of the current time in milliseconds since the song started playing.
  + volume: A float variable that represents the current volume level.
* Constructor: MusicPlayer(MusicPlayerGUImusicPlayerGUI): Initializes the MusicPlayer instance with a reference to the MusicPlayerGUI instance.
* Methods for controlling playback:
  + loadSong(Song song): Loads a single song for playback, setting the currentSong and updating the GUI components.
  + loadPlaylist(File playlistFile): Loads a playlist from a text file, creating a list of Song objects and updating the playlist and currentPlaylistIndex variables.
  + pauseSong(): Pauses the currently playing song by setting the isPaused flag and stopping the advancedPlayer.
  + stopSong(): Stops the currently playing song by stopping the advancedPlayer.
  + nextSong(): Advances to the next song in the playlist by incrementing the currentPlaylistIndex and updating the currentSong.
  + prevSong(): Goes back to the previous song in the playlist by decrementing the currentPlaylistIndex and updating the currentSong.
  + playCurrentSong (): Starts or resumes playing the current song by creating a new advancedPlayer instance and calling its play() method.
* Helper methods:
  + startMusicThread(): Starts a new thread that handles the music playback, either by resuming from the current frame or starting from the beginning.
  + startPlaybackSliderThread(): Starts a new thread that updates the playback slider in the GUI based on the current playback position.
  + setVolume(float volume): Sets the volume level for the audio playback by adjusting the FloatControl of the clip instance.
  + getVolume(): Returns the current volume level.
* Implementations of PlaybackListener interface methods:
  + playbackStarted(PlaybackEvent evt): This method is called when the audio playback starts. It resets the songFinished, pressedNext, and pressedPrev flags.
  + playbackFinished(PlaybackEvent evt): This method is called when the audio playback finishes or the player is closed. It updates the currentFrame if the song is paused and handles the behavior when the song finishes, such as playing the next song in the playlist or updating the GUI

3.3. **RELAVANCE OF MODULE**

The ‘MusicPlayer’ module is crucial to the overall functionality and user experience of the music player application. Its relevance can be highlighted in the following points:

* **Core Functionality**: The module provides the essential functions needed for playing audio files, including loading, playing, pausing, and stopping songs.
* **Playlist Management**: It supports loading and managing playlists, enabling users to play multiple songs seamlessly.
* **User Interaction**: By interacting with the GUI, it ensures that the user interface is updated in real-time, reflecting the current playback state and enhancing the user experience.
* **Thread Management**: The use of separate threads for playback and GUI updates ensures smooth and responsive operation, even during intensive tasks such as real-time slider updates.
* **Event Handling**: The module's ability to handle playback events allows it to manage transitions between songs and update the application state appropriately, ensuring a smooth listening experience.

3.4. **UML DIAGRAM OF THE MODULE**

****

**Figure-3: UML Diagram for MusicPlayer.java**

4. **MODULE-4: SONG**

4.1. **MODULE DESCRIPTION**

The 'Song’ module is designed to encapsulate and manage the details of an individual song within the music player application. It extracts and stores metadata such as the title, artist, length, and file path of the MP3 file. Additionally, it calculates and provides the frame rate per millisecond, which is essential for accurate playback control and GUI updates. This module leverages the Mp3File class from the mp3agic library and the AudioFile class from the jaudiotagger library to read and process the song's metadata.

4.2. **MODULE IMPLEMENTATION**

The `Song` class provides a structured way to handle MP3 file information and metadata. The

implementation details are as follows:

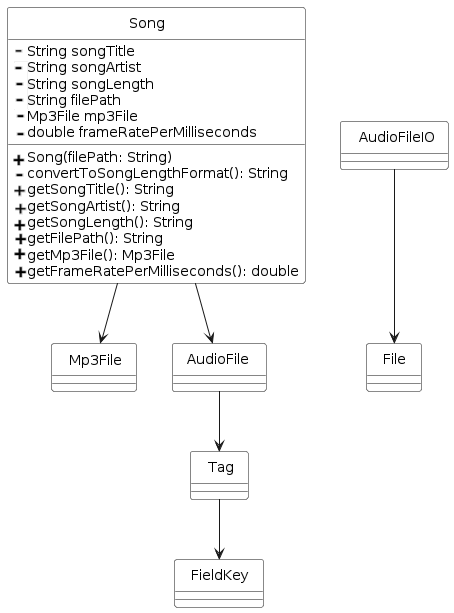
* **Fields**:
* ‘playSignal’: A synchronization object to manage playback state changes.
* ‘musicPlayerGUI’: A reference to the GUI for updating its state.
* ‘currentSong’: Stores the current song being played.
* ‘playlist’: A list to manage a collection of songs.
* ‘currentPlaylistIndex:’ Tracks the current song's index in the playlist.
* ‘advancedPlayer’: An instance of ‘AdvancedPlayer’ from the JLayer library to handle MP3 playback.
* ‘isPaused’,’songFinished’,’pressedNext’,’pressedPrev’: Flags to manage the playback state.
* ‘currentFrame’, ‘currentTimeInMilli’: Track playback position for pausing and resuming.
* **Constructor**:
* Initializes the ‘musicPlayerGUI’ and sets up the initial state.
* **Methods**:
* ‘getCurrentSong()’: Returns the current song.
* ‘setCurrentFrame(int frame)’,’ setCurrentTimeInMilli(int timeInMilli)’: Setters for updating playback state.
* ‘loadSong(Song song)’: Loads and plays a single song.
* ‘loadPlaylist(File playlistFile)’: Loads a playlist from a file and plays the first song.
* ‘pauseSong()’: Pauses the current song.
* ‘stopSong()’: Stops the current song.
* ‘nextSong(), prevSong()’: Navigate to the next or previous song in the playlist.
* ‘playCurrentSong()’: Plays the current song, handling both fresh starts and resumes.
* ‘startMusicThread()’: Starts a new thread to handle music playback.
* ‘startPlaybackSliderThread()’: Starts a new thread to update the playback slider.
* ‘playbackStarted(PlaybackEventevt)’: Callback for when playback starts.
* ‘playbackFinished(PlaybackEventevt)’: Callback for when playback finishes or the player stops.

4.3. **RELAVANCE OF MODULE**

The ‘Song’ module is a fundamental part of the music player application, with several key roles:

* + **Metadata Management**: It provides a way to extract and manage song metadata, enhancing the user experience by displaying accurate and relevant information about each song.
  + **Playback Control**: The frame rate per millisecond calculation is crucial for precise playback control and synchronization with the GUI, ensuring that the playback slider and other visual elements accurately reflect the current playback state.
  + **Integration**: By encapsulating all relevant song information in a single class, it simplifies integration with other modules like MusicPlayer and MusicPlayerGUI, enabling them to function cohesively.
  + **Error Handling**: The module includes error handling to manage scenarios where metadata might not be available, ensuring the application remains robust and user-friendly.

4.4. **UML DIAGRAM OF THE MODULE**

****

**Figure-4: UML Diagram for Song.java**

5. **MODULE-5: MUSIC PLAYER DIALOG**

5.1. **MODULE DESCRIPTION**

The Music Player Dialog module, implemented as the ‘MusicPlaylistDialog’ class, is a GUI component for a music player application. This module allows users to create and manage playlists by adding songs and saving them to a file. The dialog is designed to be user-friendly, with an interface that integrates seamlessly with the main music player application (‘MusicPlayerGUI’). It provides functionality for selecting music files, displaying the selected files within the dialog, and saving the playlist to a text file for later use.

5.2. **MODULE IMPLEMENTATION**

#### **Class Structure**

* **MusicPlaylistDialog**: Extends ‘JDialog‘and contains all the necessary components and logic for creating a playlist.
* **Attributes**:
  + ‘musicPlayerGUI’: Reference to the main GUI application, ensuring the dialog is modal and centered relative to the main window.
  + ‘songPaths’: List to store the paths of selected songs.

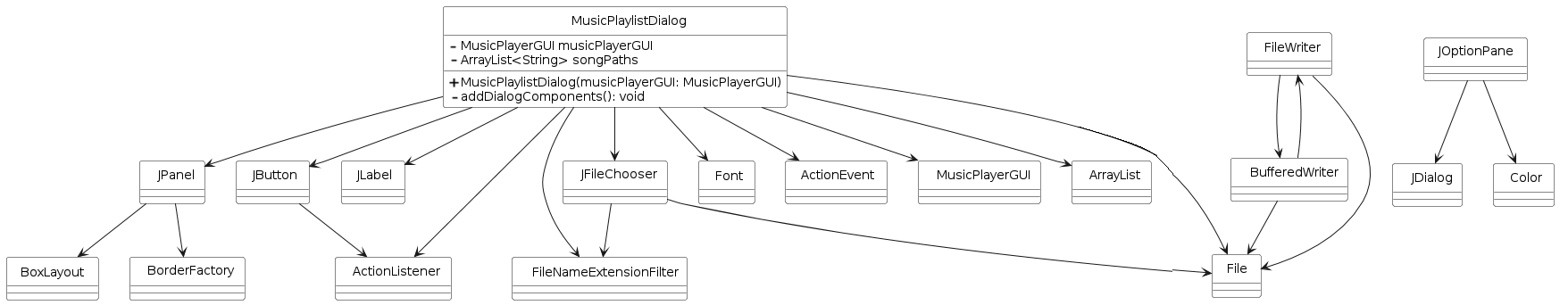
#### **Key Methods and Components**

* **Constructor**:
  + Configures the dialog with a title, size, background color, layout, and modal property.
  + Calls ‘addDialogComponents()’ to add the interactive elements.
* **addDialogComponents()**:
  + **Song Container**: A ‘JPanel‘with a vertical ‘BoxLayout’ to display selected song paths.
  + **Add Song Button**:
    - Opens a ‘JFileChooser’ to allow users to select MP3 files.
    - Adds the selected file path to the ‘songPaths’ list and displays it in the song container.
  + **Save Playlist Button**:
    - Opens a ‘JFileChooser’ to select the location and name for saving the playlist.
    - Ensures the file has a ‘.txt’ extension.
    - Writes the paths from ‘songPaths’ to the file, each on a new line.
    - Displays a success message and closes the dialog upon successful save.

5.3. **RELAVANCE OF MODULE**

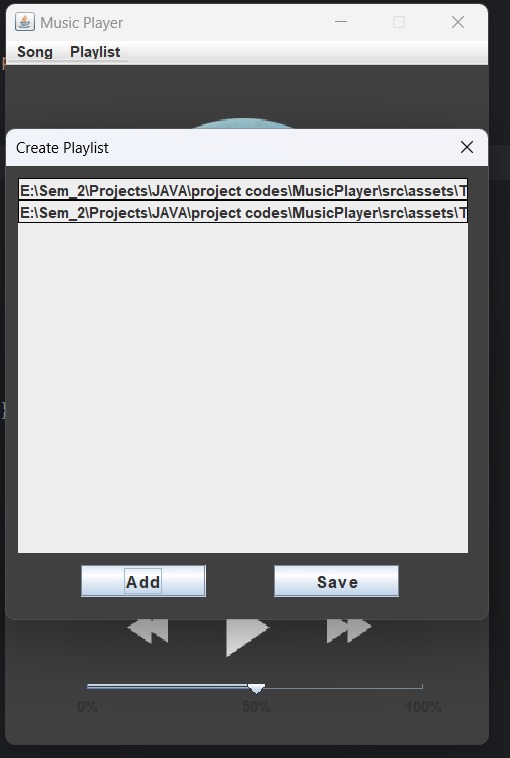
The Music Player Dialog module is crucial for enhancing the functionality and user experience of the music player application. By providing an intuitive interface for creating playlists, it allows users to manage their music collections more effectively. The ability to save playlists ensures that users can easily access and enjoy their curated music selections without having to recreate them each time they use the application. This module not only adds convenience but also enriches the overall usability and appeal of the music player software.

5.4. **UML DIAGRAM OF THE MODULE**

****

**Figure-5: UML Diagram for MusicPlayerDialog.java**

**5.5 Application OUTPUT**

****

**Figure 1.3**

**6.CONCLUSION**

The music player application developed in this project provides a comprehensive solution for users to enjoy their music collection. It offers a user-friendly graphical interface that allows users to load and play individual songs or create and manage playlists. The application leverages various Java libraries, such as JLayer and jaudiotagger, to handle audio playback and metadata extraction effectively.

The modular design of the application separates concerns into different classes, making it easier to maintain and extend the codebase. The Song class encapsulates the properties and metadata of a song file, while the MusicPlayer class manages the core functionality of playback, including loading songs, controlling playback, and navigating through playlists. The MusicPlayerGUI class provides the visual components and user interface, allowing users to interact with the application seamlessly.

The MusicPlaylistDialog class enhances the user experience by enabling the creation and management of playlists, a common feature in music player applications. Users can add songs to a playlist and save it as a text file for future use.