REPORT

SWING-BASED MUSIC PLAYER IMPLEMENTATION IN JAVA

I. INTRODUCTION

The Java Swing-Based Music Player is an interactive graphical user interface (GUI) application developed using Java Swing libraries. Its primary function is to serve as a platform for playing music files in the WAV format. This project aims to provide a user-friendly environment for managing playlists, controlling playback, and offering essential music player functionalities.

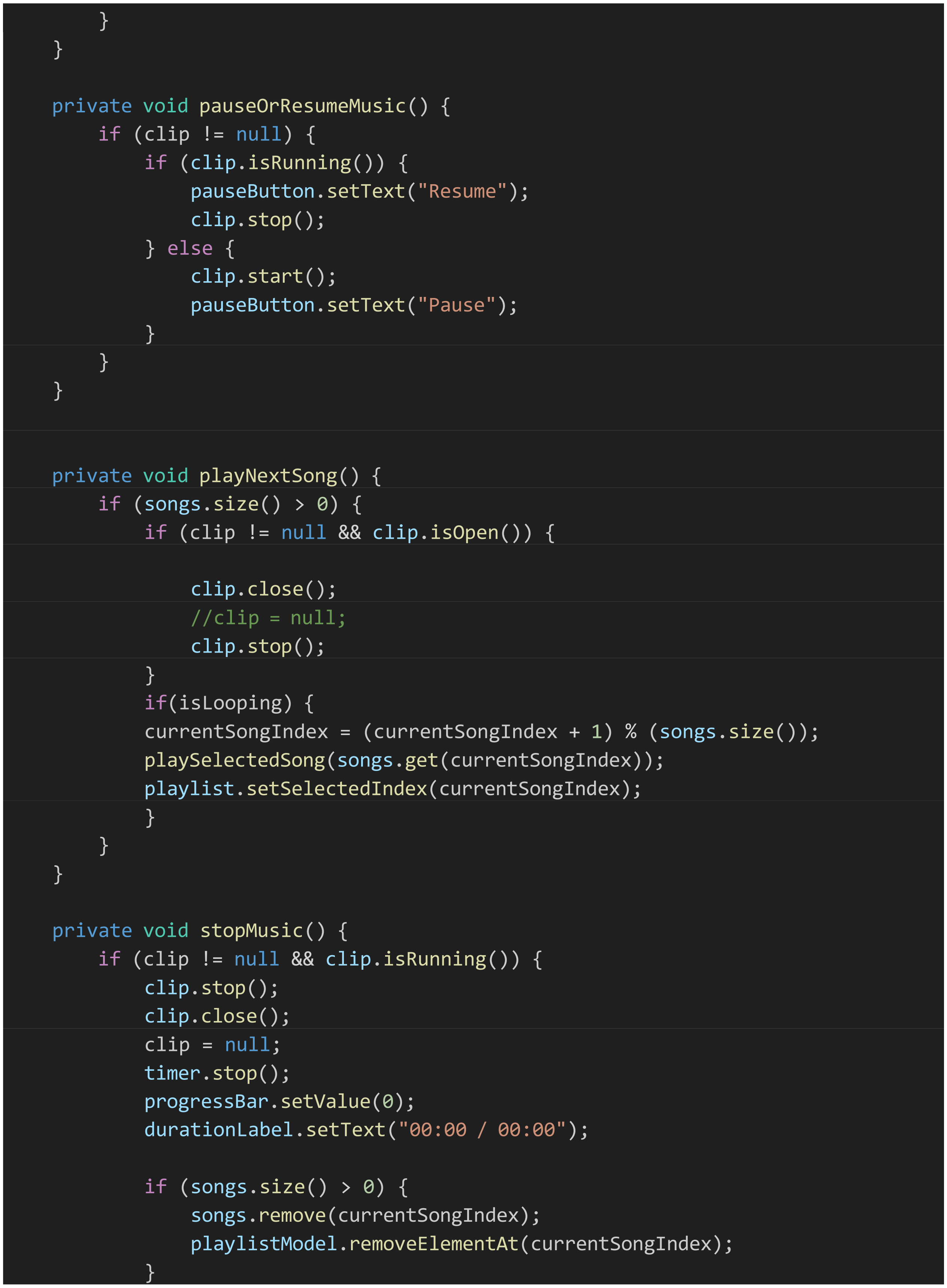
The core objectives of this project encompass the following key points:

- Implementing a GUI-based music player using Java Swing components.
- Enabling users to add WAV format music files to a playlist.
- Facilitating control over playback (play, pause, next track) and loop functionalities.
- Displaying real-time progress information during music playback.

II. PROGRAM

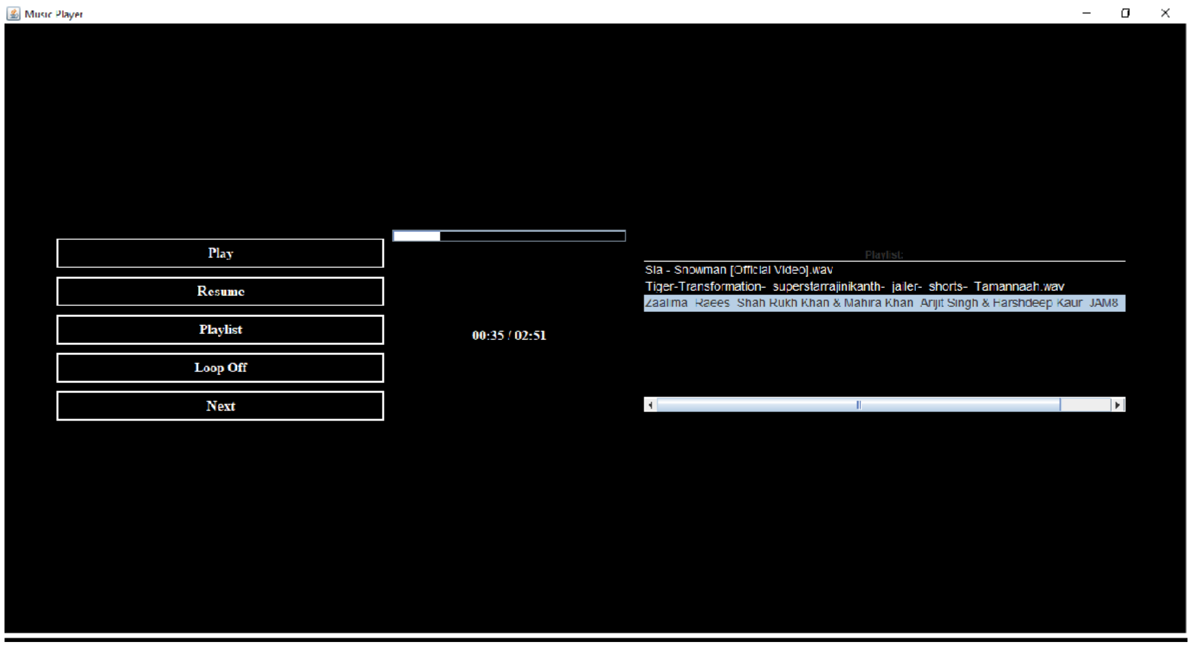
```java
import javax.sound.sampled.*;
import javax.swing.*;
import javax.swing.border.*;
import javax.swing.filechooser.FileNameExtensionFilter;
import javax.swing.plaf.basic.BasicButtonUI;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.ArrayList;
import java.util.List;

public class JavaGUIMusicPlayerJFrame extends JFrame implements ActionListener {

    private JButton playButton;
    private JButton pauseButton;
    private JButton playlistButton;
    private JButton loopButton;
    private JButton nextButton;
    private boolean isPaused;
    private boolean isLooping = false;
    private JFileChooser fileChooser;
    private Clip clip;
    private Timer timer;
    private JProgressBar progressBar;
    private JLabel durationLabel;
    private JList<String> playlist;
    private DefaultListModel<String> playlistModel;
    private List<File> songs;
    private int currentSongIndex = 0;


    public JavaGUIMusicPlayerJFrame() {
        super("Music Player");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());
        getContentPane().setBackground(Color.BLACK);

        Font blockFont = new Font("Times New Roman", Font.BOLD, 16);
        Font buttonFont = new Font("Times New Roman", Font.PLAIN, 14);

        JPanel mainPanel = new JPanel(new GridBagLayout());
        mainPanel.setBorder(new EmptyBorder(20, 50, 20, 50));
        mainPanel.setBackground(Color.BLACK);

        GridBagConstraints gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.gridy = 0;
        gbc.gridwidth = 1;
        gbc.gridheight = 2;
        gbc.weightx = 0.7;
        gbc.fill = GridBagConstraints.BOTH;

        JPanel buttonPanel = new JPanel();
        buttonPanel.setLayout(new GridLayout(5, 1, 10, 10));
        buttonPanel.setBorder(new EmptyBorder(10, 10, 10, 10));
        buttonPanel.setBackground(Color.BLACK);

        playButton = createStyledButton("Play", buttonFont);
        pauseButton = createStyledButton("Pause", buttonFont);
        playlistButton = createStyledButton("Playlist", buttonFont);
        loopButton = createStyledButton("Loop Off", buttonFont);
        nextButton = createStyledButton("Next", buttonFont);

        playButton.addActionListener(this);
        pauseButton.addActionListener(this);
        playlistButton.addActionListener(this);
        loopButton.addActionListener(this);
        nextButton.addActionListener(this);

        playButton.setFont(blockFont);
        pauseButton.setFont(blockFont);
        playlistButton.setFont(blockFont);
        loopButton.setFont(blockFont);
        nextButton.setFont(blockFont);

        buttonPanel.add(playButton);
        buttonPanel.add(pauseButton);
        buttonPanel.add(playlistButton);
        buttonPanel.add(loopButton);
        buttonPanel.add(nextButton);

        mainPanel.add(buttonPanel, gbc);

        progressBar = new JProgressBar();
        progressBar.setBackground(Color.BLACK);
        progressBar.setForeground(Color.WHITE);
        progressBar.setValue(0);

        durationLabel = new JLabel("00:00 / 00:00", SwingConstants.CENTER);
        durationLabel.setForeground(Color.WHITE);
        durationLabel.setFont(blockFont);

        gbc.gridx = 1;
        gbc.gridy = 0;
        gbc.weightx = 0.3;
        gbc.gridheight = 1;
        mainPanel.add(progressBar, gbc);

        gbc.gridx = 1;
        gbc.gridy = 1;
        gbc.weightx = 0.3;
        gbc.fill = GridBagConstraints.BOTH;
        mainPanel.add(durationLabel, gbc);

        playlistModel = new DefaultListModel<>();
        playlist = new JList<>(playlistModel);
        playlist.setFont(new Font("Arial", Font.PLAIN, 14));
        playlist.setForeground(Color.WHITE);
        playlist.setBackground(Color.BLACK);
        playlist.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
        playlist.setBorder(BorderFactory.createLineBorder(Color.WHITE));

        JScrollPane playlistScrollPane = new JScrollPane(playlist);
        playlistScrollPane.setBorder(BorderFactory.createEmptyBorder());
        playlistScrollPane.setPreferredSize(new Dimension(150, 150));

        JPanel playlistPanel = new JPanel(new BorderLayout());
        playlistPanel.setBackground(Color.BLACK);
```

```java
        playlistPanel.setBorder(new EmptyBorder(20, 20, 20, 20));
        playlistPanel.add(new JLabel("Playlist:", SwingConstants.CENTER),
BorderLayout.NORTH);
        playlistPanel.add(playlistScrollPane, BorderLayout.CENTER);

        gbc.gridx = 2;
        gbc.gridy = 0;
        gbc.gridheight = 2;
        gbc.weightx = 1.0;
        gbc.fill = GridBagConstraints.BOTH;
        mainPanel.add(playlistPanel, gbc);

        add(mainPanel, BorderLayout.CENTER);

        fileChooser = new JFileChooser(".");
        fileChooser.setFileFilter(new FileNameExtensionFilter("WAV Files",
"wav"));

        songs = new ArrayList<>();

        setSize(600, 300);
        setLocationRelativeTo(null);
        setVisible(true);

        timer = new Timer(100, e -> {
            if (clip != null && clip.isRunning()) {
                long time = clip.getMicrosecondPosition() / 1_000_000;
                long duration = clip.getMicrosecondLength() / 1_000_000;

                String timeStr = String.format("%02d:%02d / %02d:%02d",
                        time / 60, time % 60, duration / 60, duration % 60);
                durationLabel.setText(timeStr);

                int progress = (int) (((double) clip.getMicrosecondPosition() /
clip.getMicrosecondLength()) * 100);
                progressBar.setValue(progress);
            }
        });
        timer.setInitialDelay(0);
    }

    private JButton createStyledButton(String label, Font font) {
        JButton button = new JButton(label);
        button.setFont(font);
        button.setForeground(Color.WHITE);
        button.setBackground(Color.BLACK);
        button.setBorder(new CompoundBorder(new LineBorder(Color.WHITE, 2), new
EmptyBorder(5, 15, 5, 15)));
        button.setUI(new BasicButtonUI() {
            @Override
            protected void paintButtonPressed(Graphics g, AbstractButton b) {
                Graphics2D g2 = (Graphics2D) g.create();
                g2.setColor(Color.WHITE);
                g2.fillRect(0, 0, b.getWidth(), b.getHeight());
                g2.dispose();
            }
        });
        button.setFocusPainted(false);
        return button;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == playButton) {
            if (songs.size() > 0) {
                pauseButton.setText("Resume");
                playSelectedSong(songs.get(currentSongIndex));
            } else {
                selectSong();
            }

            if(pauseButton.getText() == "Resume") {
                pauseButton.setText("Pause");
            }
        } else if (e.getSource() == pauseButton) {
            pauseOrResumeMusic();
        } else if (e.getSource() == playlistButton) {
            selectSong();
        } else if (e.getSource() == loopButton) {
            toggleLoop();
        } else if (e.getSource() == nextButton) {
            playNextSong();
        }
    }

    private void selectSong() {
        int result = fileChooser.showOpenDialog(this);
        if (result == JFileChooser.APPROVE_OPTION) {
            File selectedFile = fileChooser.getSelectedFile();
            playlistModel.addElement(selectedFile.getName());
            songs.add(selectedFile);
        }
    }

    private void playSelectedSong(File selectedFile) {
        try {
            if (clip != null && clip.isRunning()) {
                clip.stop();
            }

            AudioInputStream audioIn =
AudioSystem.getAudioInputStream(selectedFile);

            clip = AudioSystem.getClip();
            clip.open(audioIn);
            clip.start();

            if (!timer.isRunning()) {
                timer.start();
            }

            clip.addLineListener(event -> {
                if (event.getType() == LineEvent.Type.STOP) {
                    if (!isLooping && pauseButton.getText()!="Resume") {
                        stopMusic();
                        //if (currentSongIndex < songs.size() - 1) {
                            //currentSongIndex++;
                            currentSongIndex = (currentSongIndex + 1) %
(songs.size());
                            playSelectedSong(songs.get(currentSongIndex));
                            playlist.setSelectedIndex(currentSongIndex);
                        //}
                    }
                    else if (pauseButton.getText() == "Resume") {
                        stopMusic();
                    }
                    else {
                        clip.setMicrosecondPosition(0);
                        clip.start();
                    }
                }
            });

        } catch (Exception ex) {
            ex.printStackTrace();
```



```java
        }
    }

    private void pauseOrResumeMusic() {
        if (clip != null) {
            if (clip.isRunning()) {
                pauseButton.setText("Resume");
                clip.stop();
            } else {
                clip.start();
                pauseButton.setText("Pause");
            }
        }
    }


    private void playNextSong() {
        if (songs.size() > 0) {
            if (clip != null && clip.isOpen()) {

                clip.close();
                //clip = null;
                clip.stop();
            }
            if(isLooping) {
            currentSongIndex = (currentSongIndex + 1) % (songs.size());
            playSelectedSong(songs.get(currentSongIndex));
            playlist.setSelectedIndex(currentSongIndex);
            }
        }
    }

    private void stopMusic() {
        if (clip != null && clip.isRunning()) {
            clip.stop();
            clip.close();
            clip = null;
            timer.stop();
            progressBar.setValue(0);
            durationLabel.setText("00:00 / 00:00");

            if (songs.size() > 0) {
                songs.remove(currentSongIndex);
                playlistModel.removeElementAt(currentSongIndex);
            }
        }
    }

    private void toggleLoop() {
        isLooping = !isLooping;
        if (isLooping) {
            loopButton.setText("Loop On");
        } else {
            loopButton.setText("Loop Off");
        }
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new JavaGUIMusicPlayerJFrame());
    }
}
```

III. <u>OUTPUT</u>



```
00:35 / 02:51
```

```
Sia - Snowman [Official Video].wav
Tiger-Transformation-  superstarrajinikanth-  jailer-  shorts-  Tamannaah.wav
Zaalima  Raees  Shah Rukh Khan & Mahira Khan  Arijit Singh & Harshdeep Kaur  JAM8
```

IV. FUNCTIONALITIES AND FEATURES

Playback Control:

- Play/Pause: Enables users to start or pause music playback.
- Next Song: Allows users to skip to the next track in the playlist.
- Looping: Provides an option to toggle continuous looping of the currently playing track.
- Real-time Progress Display: Presents the progress of the current track and its duration.

Playlist Management:

- File Selection: Users can add WAV format music files to the playlist through the file chooser.
- Playlist Display: Shows the list of added songs for selection and playback.
- Dynamic Updating: Updates playlist dynamically upon adding or removing songs.

V. METHODOLOGY

The development of the Swing-Based Music Player involved a systematic approach that encompassed various stages and methodologies to achieve a functional and user-friendly application. The methodology employed during the project implementation can be outlined as follows:

1. Requirement Analysis and Planning
    - Identification of Project Goals: Defined the primary objectives of the music player application, including playback control, playlist management, and user interface design.
    - User Story Creation: Developed user stories to comprehend user interactions and functionalities expected from the application.
    - Initial Planning: Outlined the project roadmap, identified necessary tools, and delineated the scope of functionalities to be included.

2. Design Phase
    - UI/UX Design: Utilized paper prototypes and wireframes to sketch the initial user interface design, incorporating essential components like buttons, progress bars, playlist display, and navigation controls.
    - Architecture Planning: Defined the software architecture, emphasizing modularity and scalability using Java Swing components.
    - Component Allocation: Distributed responsibilities among different classes and components for improved code organization and maintainability.

3. Implementation
    - Java Swing and Audio Handling: Leveraged Java Swing libraries for GUI creation and the Java Sound API for audio file handling and playback functionalities.
    - Code Development: Wrote code modules to implement various features, including play, pause, next track, playlist management, and real-time progress tracking.
    - Error Handling and Testing: Implemented exception handling and conducted iterative testing to identify and rectify errors in audio handling, UI components, and functionalities.

4. Iterative Development and Testing
    - Incremental Development: Adopted an iterative approach, building functionalities incrementally and performing continuous testing after each implementation phase.
    - User Testing: Conducted user testing sessions to gather feedback and iteratively refine the application based on user suggestions.

5. Documentation and Report Writing
    - Code Documentation: Generated code comments and documentation for better code understanding and maintainability.

      Report Compilation: Documented the project's progress, methodologies, and implementation details for a comprehensive report presentation.

6. Conclusion and Future Steps

- Finalization: Completed the Swing-Based Music Player application based on the outlined requirements and functionalities.
- Future Scope: Outlined potential enhancements and additional features for future iterations of the application

VI. REFERENCES

1. Java Documentation : Information and documentation about Java Swing libraries and Java Sound API were referred to extensively during the development of the project.
   https://www.baeldung.com/java-play-sound

2. Official Oracle Documentation : Oracle's official Java documentation provided insights into Java's audio handling and Swing GUI components.
   https://docs.oracle.com/javase/tutorial/uiswing/

3. Stack Overflow : Various threads on Stack Overflow provided troubleshooting solutions and guidance during the implementation of specific functionalities.
   https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/debugging-a-stack-overflow

4. Programming Books:

Programming with Java : A Primer – E. Balagurusamy

5. Online Tutorials or Websites:

https://youtu.be/Q6yl-7ayn1w?si=w7j3PYKvieR9gB_i

VII. CONCLUSION

In conclusion, this project demonstrates a robust implementation of a music player using Java Swing, emphasizing functionality and ease of use. The application meets the objectives set forth, providing a solid foundation for future enhancements and improvements.