\

BINUS UNIVERSITY

BINUS INTERNATIONAL

GuhDify Music

Object Oriented Programming

Final Project Report

Student Information:

Surname: Agastya **Given Name:** Randy **Student ID:** 2602119165

Course Code : COMP6699001 **Course Name :** Object Oriented Programming

Class : L2AC **Lecturer :** Mr. Jude Joseph Lamug Martinez

Type of Assignment: Final Project Report

Submission Pattern

Due Date : 16/06/23 **Submission Date :** 15/06/23

Table of Contents

Project Specifications

Background

For my final project, I was assigned the task to design and implement a Java project that would both stretch my abilities and venture beyond the boundaries of the course material. The ultimate aim of the Term Project is to leverage the knowledge I've acquired about Java Programming and problem-solving to address a unique yet appealing challenge. Consequently, I have chosen to create a music player application in Java, which will use Java Swing to implement its GUI and use several data structures along the way. This report will explain and examine my project.

Brief Description

I made GuhDify Music, a simple but powerful music player that hooks into Spotify's huge collection of songs. Users can easily play, stop, or switch songs, providing a smooth and fun music experience. Plus, they can look back at their song history to see what they've been jamming to lately. This all makes using the app a breeze and keeps users connected to their favorite tunes.

But that's not all. GuhDify Music also includes some cool features that make it extra special. For instance, users can search for songs or artists by typing, but they can also just speak into their device's microphone for hands-free searching. As a beginner developer, I wanted to create an app that's not only useful but also fun and exciting. I believe GuhDify Music hits the spot, combining easy controls with cool features to keep users coming back for more.

Data Structures Used

1. **ArrayList:** ArrayList is a part of Java's standard collections framework. It is a resizable array, meaning its size can be increased or decreased dynamically. In this case, the ArrayList is designed to hold instances of the Track class. ArrayList provides several useful methods for manipulating the data it contains, such as adding, removing, and searching for elements.
2. **List:** A List in Java is an ordered collection, meaning it maintains the insertion order of the elements. The elements can be inserted or accessed by their position in the list.
3. **HashMap:** A HashMap is a part of Java's collections framework that stores elements as key-value pairs. It uses a technique called hashing to store the data.

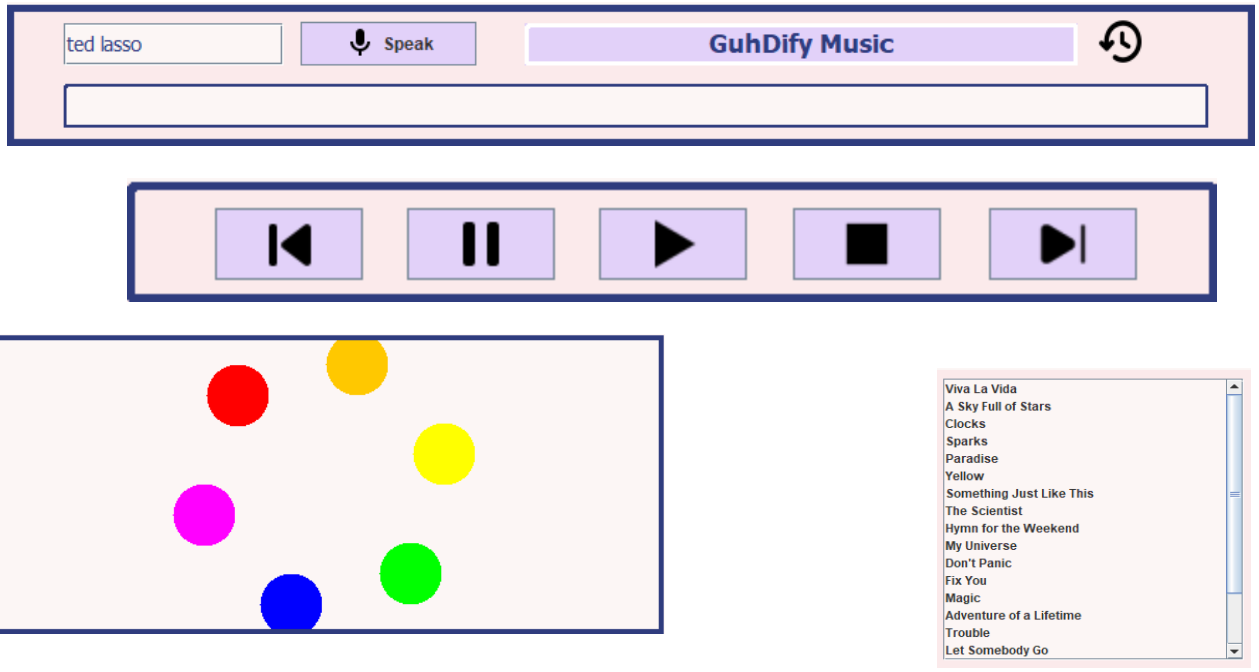
Solution Design

Objective of Design

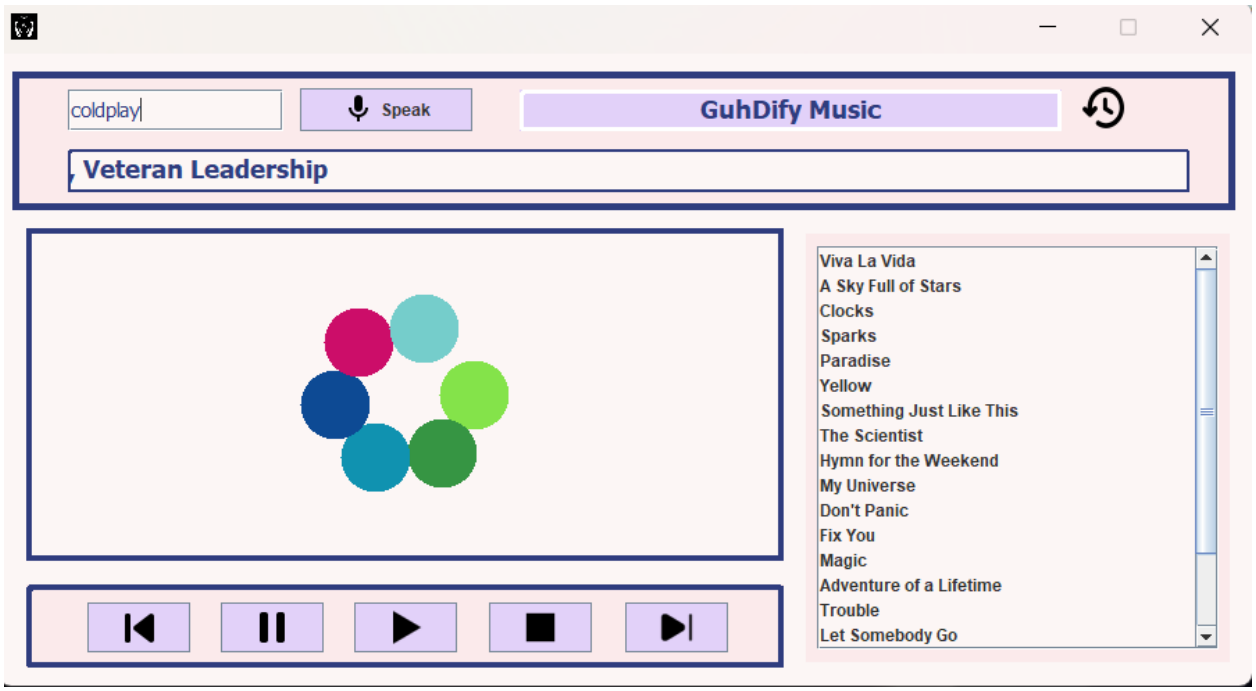
My design goal was to achieve a minimalist aesthetic while ensuring seamless navigation across the application's features. The selected color palette contributes to a visually pleasing user interface, upholds a professional look, and keeps users entertained, ensuring that they don't tire or become disinterested while interacting with the app.

Design

The application's design showcases a versatile demonstration of graphical animations using Java Swing library. It contains several files called JPanel which allows the application to contain different design sections inside of it. For example, in my application, it contains 4 different panels which we can see below:



If we combine all of those panel, the result will be like this:



Color Choice

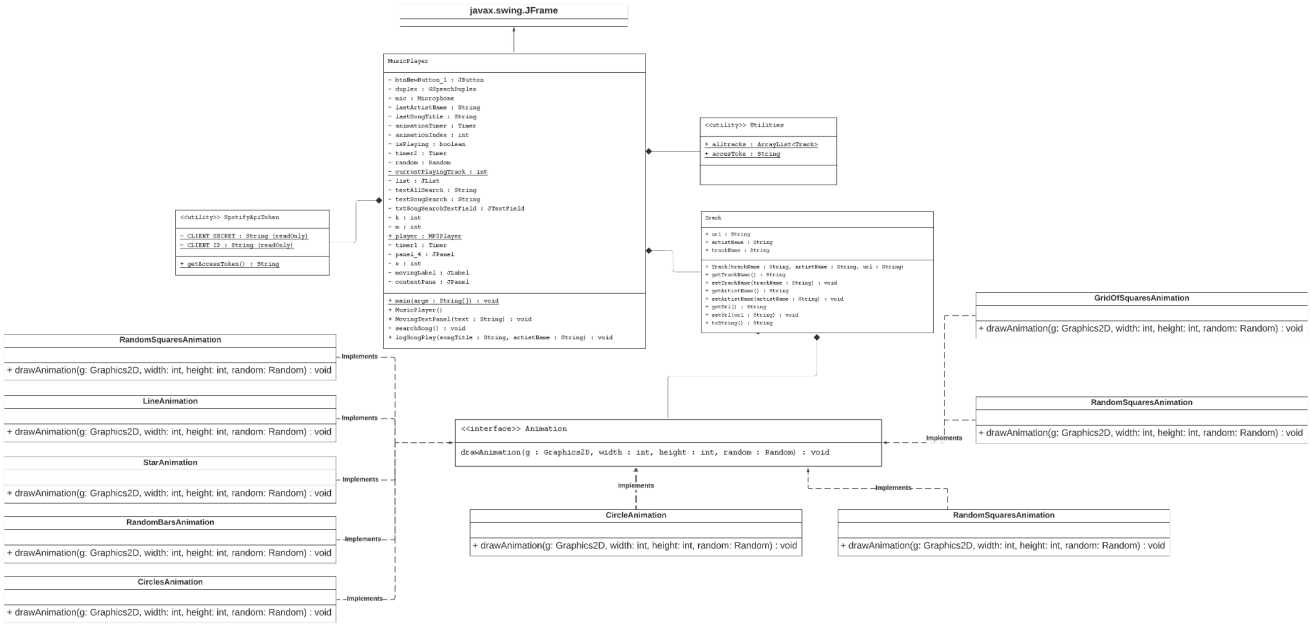
Color is an essential part of the user interface design. It not only contributes to the aesthetic appeal of the application but also plays a crucial role in usability and user experience. In the case of this Music Player Application, three primary colors have been used, namely blue (#2F3C7E), pink (#FBEAEB), and broken white (#FCF6F5).

1. Blue (#2F3C7E): This specific shade of blue serves as a primary color and is primarily used for key elements like borders, text, and icons. It helps in drawing the user's attention to these essential parts of the interface. Blue is often associated with trust, stability, and intelligence, and it gives the application a professional and reliable aura.
2. Pink (#FBEAEB): This soft shade of pink is used as the background color for some key components, such as the panel displaying the current track. Pink is typically associated with softness, sweetness, and love. The choice of this color gives the interface a gentle, warm, and welcoming appeal. It also adds a touch of vibrancy to the interface, making it more visually engaging.
3. Broken White (#FCF6F5): This off-white color, known as broken white, is used as the primary background color for most of the interface. It serves as a neutral canvas that allows the other colors to stand out. This color can provide a calming and soothing effect, adding to the overall comfort of the user experience.

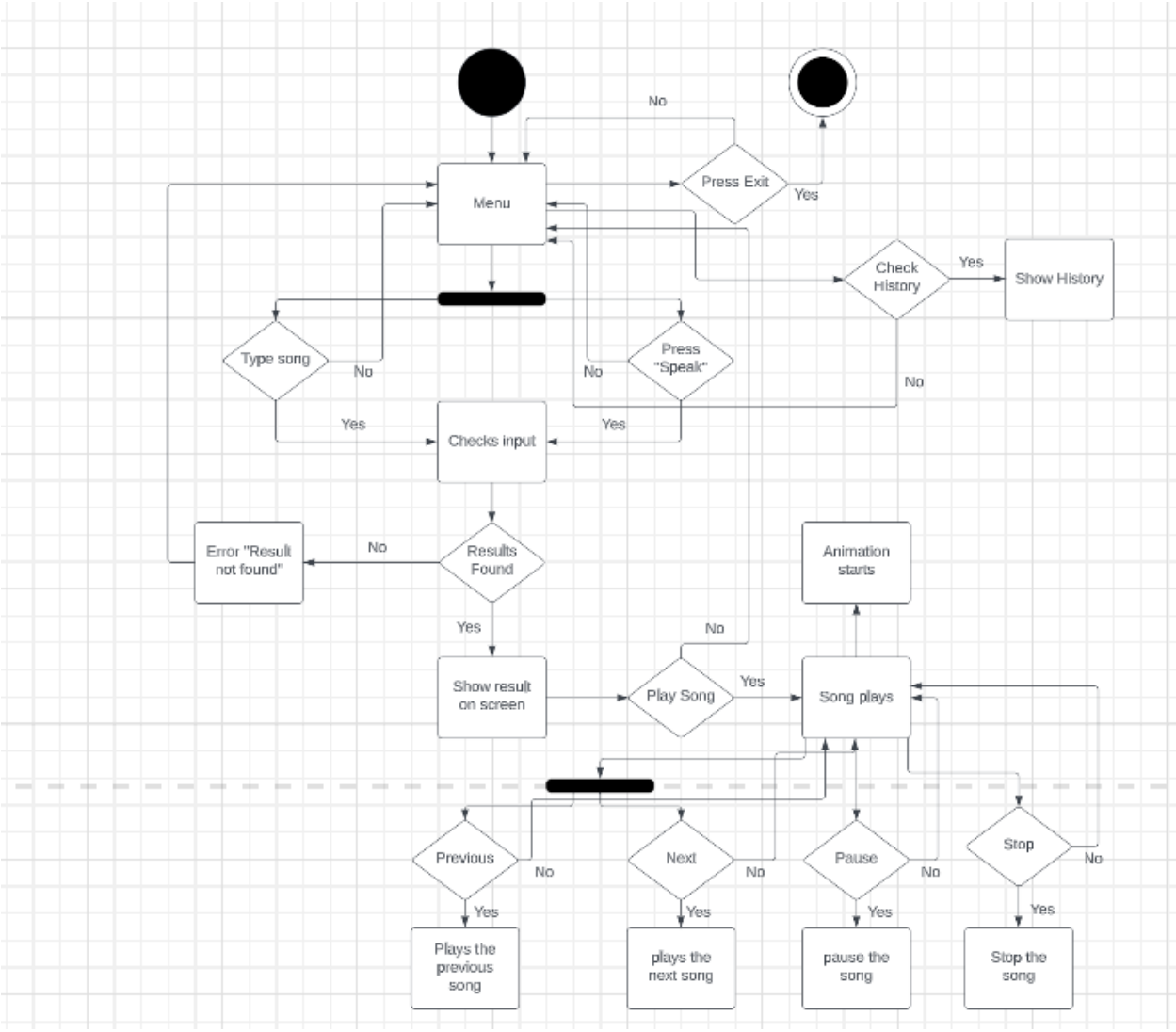
The chosen color palette shows an understanding of color psychology and how colors can impact user perception and experience. The mix of calming blue and broken white, with a splash of vibrant pink, creates a balanced and visually appealing interface. The broken white provides a clean and minimal canvas, the blue offers a professional and trustworthy feel, and the pink adds a touch of warmth and vibrancy.

In terms of contrast, the chosen colors work well together. The contrast between the blue and the broken white or pink is high enough to ensure readability and accessibility. At the same time, the colors are harmonious and pleasing to the eye, adding to the overall aesthetic appeal of the inter

Class Diagram



Flow Chart


No
Menu
Press Exit
Yes
Check History
Yes
Show History
Type song
No
Press "Speak"
No
No
Yes
Checks input
Yes
Error "Result not found"
No
Results Found
Animation starts
Yes
No
Show result on screen
Play Song
Yes
Song plays
Previous
No
Next
No
Pause
No
Stop
No
Yes
Yes
Yes
Yes
Plays the previous song
plays the next song
pause the song
Stop the song

Libraries/Modules

1. **AWT (Abstract Window Toolkit):** This is a fundamental part of Java's standard library used to create window-based applications and applets. It is used to control the graphical aspects of the GuhDify Music Player, such as setting the colors, fonts, and other visual properties. AWT's Timer class is utilized to create the animation timer, which drives the visual transition effects in the user interface.
2. **Swing:** Swing is a GUI (Graphical User Interface) widget toolkit for Java. It's an extension of AWT and provides a more sophisticated set of GUI components than AWT. In the GuhDify Music Player, Swing is used to build the user interface, which includes panels, buttons, labels, text fields, and other GUI components, making it more interactive and user-friendly.
3. **Apache HttpComponents:** This library is used to enable communication with servers over HTTP. It's used to create HTTP clients that send requests to music streaming servers, process responses, and handle entity content such as retrieving song metadata or streaming audio files.
4. **Jackson:** Jackson is a high-performance JSON processor for Java. When your application communicates with servers (for example, Spotify's API), the responses are typically in JSON format. Jackson is used to parse these JSON responses into Java objects, making it easier to handle and manipulate the data.
5. **Java Speech API (darkprograms):** This API is used to enable the voice command feature in the GuhDify Music Player. It supports speech recognition, allowing users to use voice commands to search and control the music playback.
6. **Jaco MP3 Player:** This is a Java library specifically designed to handle MP3 files. Jaco MP3 Player is used in the GuhDify Music Player to play the songs

that users select. It provides an easy-to-use API for controlling the music playback, including play, pause, and stop functions.

7. **JavaFlacEncoder:** This library is used for encoding audio data into FLAC files. When the user gives a voice command, the audio input from the microphone is captured and encoded into FLAC format using JavaFlacEncoder. This encoded data is then sent to the Google Speech API for speech recognition.
8. **java.util.concurrent:** This package is part of Java's standard library and provides utilities for handling and controlling thread execution in concurrent programming. It's used in your application for managing tasks that should run concurrently, ensuring that tasks such as fetching song data, playing music, and user interactions can all happen simultaneously without causing the application to freeze or slow down.

API

Spotify Web API: The Spotify Web API allows developers to interact with the Spotify music service using RESTful web services. It provides several endpoints to fetch data like playlists, albums, tracks, and more. The API also provides functionality to control a user's Spotify player remotely and retrieve data about the user's listening activities.

Google Cloud Speech-to-Text API: This API enables developers to convert audio to text by applying powerful neural network models. The API recognizes over 120 languages and variants to support global user bases. It can process real-time streaming or prerecorded audio, using Google's machine learning technologies.

Essential Algorithms

UI

The main window of the application is created as a JFrame, where various JPanel instances are added to construct different sections of the UI. Each JPanel serves a

specific function, such as housing buttons, labels, or text fields. The properties like color, border, size, and position of these panels are customized according to the design needs.

```
JPanel panel_5 = new JPanel();
panel_5.setBorder(new LineBorder(Color.decode( nm: "#2F3C7E"), thickness: 2, roundedCorners: true)); //border for the moving label area
panel_5.setBackground(Color.decode( nm: "#FCF6F5"));//set bg color for the track rectangle
panel_5.setBounds( x: 40, y: 56, width: 807, height: 31);//set the position and size
panel.add(panel_5);
panel_5.setLayout(null);

movingLabel = new JLabel( text: "Track : "); //label writing inside panel_5
movingLabel.setForeground(Color.decode( nm: "#2F3C7E")); //set fg color
movingLabel.setFont(new Font( name: "Tahoma", Font.BOLD, size: 18));//set font used
movingLabel.setBackground(Color.decode( nm: "#FCF6F5"));//set bg color
movingLabel.setBounds( x: 10, y: 0, width: 803, height: 29);//set position and size
panel_5.add(movingLabel);

JPanel panel_6 = new JPanel(); //panel for title
panel_6.setBorder(new LineBorder(new Color( r: 255, g: 255, b: 255), thickness: 3, roundedCorners: true));//set border
panel_6.setBackground(Color.decode( nm: "#E2D1F9")); // set bg color
panel_6.setBounds( x: 365, y: 13, width: 390, height: 30); //set position and size
panel.add(panel_6);
panel_6.setLayout(null);

JLabel lblNewLabel = new JLabel( text: "GuhDify Music");
lblNewLabel.setForeground(Color.decode( nm: "#2F3C7E"));
lblNewLabel.setBackground(Color.decode( nm: "#2F3C7E"));
lblNewLabel.setFont(new Font( name: "Tahoma", Font.BOLD, size: 18));
lblNewLabel.setBounds( x: 130, y: 0, width: 190, height: 29);
panel_6.add(lblNewLabel); //inserted inside panel_6
```

For the design of the interface, considerable attention is given to customization. The color, border, size, and position of each component are adjusted as per the design needs, creating a visually pleasing interface. The use of images, especially for buttons, enhances the aesthetic appeal and provides a more intuitive interaction for users.

In terms of interactivity, several components respond to user actions. Notably, the search field allows users to enter queries, and the history button fetches and displays recently played songs from a file. These interactive features enrich the user experience, making the application more engaging and user-friendly.

A notable aspect of the layout design is the use of absolute positioning, which provides precise control over the placement of components. While this allows for exact positioning, it can also limit flexibility in terms of dynamic resizing or adaptability to different screen sizes.

Fetch Songs

The primary functionality is encapsulated in the getAccessToken() method of the SpotifyApiToken class. This method communicates with Spotify's server to authenticate the application and obtain an access token, which can be used for subsequent API requests. This process involves several key steps:

1. **Creating an HTTP Connection:** First, the method creates an HttpURLConnection object and opens a connection to the Spotify token endpoint URL

```
URL url = new URL( spec: "https://accounts.spotify.com/api/token");// Create URL object with the Spotify token endpoint URL
HttpURLConnection con = (HttpURLConnection) url.openConnection();// Create a HttpURLConnection object and open the connection
```

2. **Setting Request Properties:** It sets the request method to "POST" and sets the content type to "application/x-www-form-urlencoded", indicating that the data sent in the POST request will be in a specific format.

```
con.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");// Set the request's content type header
```

3. **Forming POST Data:** The method prepares the data to be sent in the POST request. The data includes the grant_type, client_id, and client_secret. The grant type is "client_credentials", indicating that we are using the Client Credentials flow, a type of OAuth 2.0 flow suitable for server-to-server authentication. The client_id and client_secret are unique identifiers provided by Spotify when you register your application.

```
String postData = "grant_type=client_credentials&client_id=" + CLIENT_ID + "&client_secret=" + CLIENT_SECRET;//

con.setDoOutput(true); // Enable output for the connection. This needs to be true because we are using POST met

DataOutputStream out = new DataOutputStream(con.getOutputStream());
out.writeBytes(postData);
out.flush(); // Write the POST data to the output stream
out.close();
```

4. **Sending POST Data:** The method sends the POST data to the server by writing it to the connection's output stream. The method enables output for the connection before sending the data.
5. **Receiving Response**: The method receives the response from the server, which includes an HTTP status code and content. If the status code is 200, the request was successful. Otherwise, an error has occurred.

```
// Read the response from the connection's input stream
BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
String inputLine;
StringBuffer content = new StringBuffer();
while ((inputLine = in.readLine()) != null) {
    content.append(inputLine);
}
in.close();

// Parse the access token from the JSON response
JSONObject json = new JSONObject(content.toString());
String accessToken = json.getString( key: "access_token");

// Store the access token in Utilities class for later use
Utilities.accesToke=accessToken;
System.out.println(accessToken);
return accessToken;
```

6. **Parsing Access Token**: The method reads the server's response from the connection's input stream and parses the access token from the JSON response.

```
// Parse the access token from the JSON response
JSONObject json = new JSONObject(content.toString());
String accessToken = json.getString( key: "access_token");

// Store the access token in Utilities class for later use
Utilities.accesToke=accessToken;
System.out.println(accessToken);
return accessToken;
```

7. **Storing Access Token**: The access token is stored in the Utilities class for later use in fetching song data from the Spotify API.

   In the main function, an instance of the MusicPlayer class is created, which is the main frame of the application. After initializing the MusicPlayer instance, the function calls SpotifyApiToken.getAccessToken(). This function call triggers the aforementioned process of fetching the access token from the Spotify API, which will later be used to fetch song data.

```
SpotifyApiToken.getAccessToken();
```

Fetch Mic Input

The GuhDify Music Player leverages a powerful Speech-to-Text (STT) technology for its voice command feature. It integrates Google's Speech-to-Text API, enabled through the 'GSpeechDuplex' Java library, for robust speech recognition functionality.

The main steps in the algorithm are as follows:

1. **Initial Configuration:** The Microphone object 'mic' and the GSpeechDuplex object 'duplex' are initialized. The language for the STT API is set as English using 'duplex.setLanguage("en")'.

```
duplex.setLanguage("en"); //language used for the speech to text API "en" = English
```

2. **Activating the Microphone:** The application waits for the user to click on the "Speak" button. This action triggers a new thread to begin listening to the microphone input and starts the speech recognition process by calling 'duplex.recognize(mic.getTargetDataLine(), mic.getAudioFormat())'.

```
URL searchUrl = getClass().getResource( name: "/asset/micro1.png");
ImageIcon searchicon = new ImageIcon(searchUrl);
DefaultListModel<String> model = new DefaultListModel<String>();
btnNewButton_1 = new JButton( text: "Speak",searchicon);
 GuhD *
btnNewButton_1.addActionListener(new ActionListener() {
    GuhD *
    public void actionPerformed(ActionEvent e) {

        // Create an executor to schedule tasks
        ScheduledExecutorService executor = Executors.newSingleThreadScheduledExecutor();
        AtomicBoolean songIdentified = new AtomicBoolean( initialValue: false);

        new Thread(() -> {
            try {
                duplex.recognize(mic.getTargetDataLine(), mic.getAudioFormat());

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

3. **Speech Recognition:** The application continues listening until a final response from the Google Speech-to-Text API is received, signaling that a complete command is recognized. The response is checked for being a final response with 'gr.isFinalResponse()'.

```
public void onResponse(GoogleResponse gr) {
    if (gr.isFinalResponse()) {
        songIdentified.set(true);
        duplex.stopSpeechRecognition(); // Stop the recognition here
```

4. **Processing the Response:** If a final response is received, it is stored in 'textSongSearch' and displayed in the search bar. In case the recognized text is of length greater than 2, the application starts the search process with 'searchSong()'. For every track found in the search results, it adds them to the music player playlist and updates the list displaying the song names.

```
textSongSearch = gr.getResponse(); // Get the response text and set it as the song search text
System.out.println("Final : " + textSongSearch);

if (textSongSearch.length() > 2) {
    txtSongSearchText.setText(textSongSearch);  // Make sure to use the same JTextField
    if (textSongSearch.length() < 1) {
        JOptionPane.showMessageDialog( parentComponent: null,  message: "An error has occurred!",  title: "Error", JOptionPane.ERROR_MESSAGE);
    } else {
        searchSong();
    }
```

5. **Timeout Handling:** To ensure the user isn't kept waiting indefinitely if no audio input is detected, a scheduled executor service is utilized. It schedules a task that, after 10 seconds, checks whether a song is identified. If not, the speech recognition is stopped, an error message is displayed, and the 'Speak' button is enabled again.

This algorithm ensures a smooth and efficient voice command feature in the GuhDify Music Player. By fetching the microphone input and using Google's robust Speech-to-Text API, the application provides users a hands-free method to interact with the music player and further enhances the overall user experience.

Search Songs

1. **HTTP Client Creation:** A CloseableHttpClient object is created to execute the HTTP requests.
2. **Creating a URL:** The URL for the Spotify search API is formed by concatenating the base URL, the encoded query (song name), and the search type (in this case, 'track').

```
String query = textSongSearch;// The search query for the song
String type = "track";
String url = "https://api.spotify.com/v1/search?q=" + URLEncoder.encode(query)// Create the URL for the Spotify search API
        + "&type=" + type;
```

3. **Creating an HTTP Get Request:** An HttpGet object is created with the URL, and an authorization header is added, which includes the access token previously obtained from Spotify.

```
HttpGet request = new HttpGet(url);
request.addHeader( name: "Authorization",  value: "Bearer " + Utilities.accesToke);
String trackName, artistName = null, preview_url = null;
if (Utilities.alltracks.size() > 0) { // Clear the list if it is not empty
    Utilities.alltracks.clear();
} else {
    System.out.println("list is empty");
}
```

4. **Sending the Request and Processing the Response:** The HttpGet request is executed using the CloseableHttpClient, and the response is processed. The response entity is extracted and converted into a string, which is a JSON representation of the track information.
5. **Parsing the JSON Response:** The JSON response is parsed using Jackson's ObjectMapper. The "tracks" object, which contains the track items, is retrieved. Each item represents a track and is processed to extract the track's name, artist name, and preview URL. These values are then used to create a new Track object, which is added to the alltracks list in the Utilities class. The index of the first track is also stored in curruntPlayingTrack.

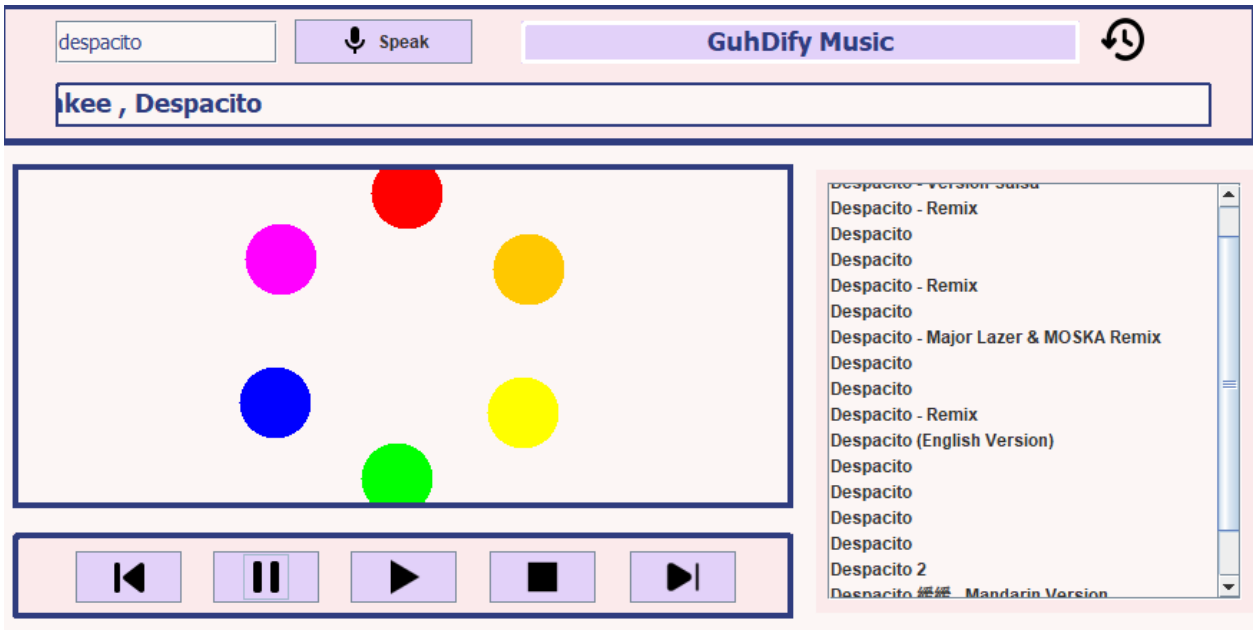
```
JsonNode artistsNode = trackNode.path( s: "artists");
for (JsonNode artistNode : artistsNode) {
    artistName = artistNode.path( s: "name").asText();
    String spotifyUrl = artistNode.path( s: "external_urls").path( s: "spotify").asText();
}

JsonNode preview_urls = trackNode.path( s: "preview_url");
for (JsonNode item : preview_urls) {
    preview_url = item.path( s: "preview_url").asText();
}
```

They are then implemented into the function whereas it will be activated when the 'Enter' key is pressed and the text field is not empty, it triggers the searchSong()

```
public void onResponse(GoogleResponse gr) {
    if (gr.isFinalResponse()) {
        songIdentified.set(true);
        duplex.stopSpeechRecognition(); // Stop the recognition here

        btnNewButton_1.setEnabled(true);
        textSongSearch = gr.getResponse(); // Get the response text and set it as the song search text
        System.out.println("Final : " + textSongSearch);

        if (textSongSearch.length() > 2) {
            txtSongSearchText.setText(textSongSearch);  // Make sure to use the same JTextField
            if (textSongSearch.length() < 1) {
                JOptionPane.showMessageDialog( parentComponent: null,  message: "An error has occurred!",  title: "Error", JOptionPane.ERROR_MESSAGE);
            } else {
                searchSong();
            }
```

It’s also implemented when the speak button is pressed and it detects a sound input, The response from the Google Speech Recognition API is processed and used to set the song search text. This text is then used to trigger the searchSong() function in a similar way to the text field key event listener.

```
public void keyPressed(KeyEvent e) {

    // If the key pressed is the Enter key
    if (e.getKeyCode() == KeyEvent.VK_ENTER) {

        // Get the text in the search text field
        String searchText = txtSongSearchText.getText();

        // If there is any text entered
        if (searchText.length() > 0) {

            // Save the search text
            textSongSearch = searchText;

            // Search for the song based on the text entered
            searchSong();
```



This will be the result whenever the user searches for a song whereas it will be listed on the right side of the panel.

Play Songs

1. Playing a Selected Song:

The user can select a song from a list, and upon selection, an event listener attached to the list item triggers the following sequence of actions:

1.1. The selected song's index is retrieved from the list and its relevant details are displayed in a pop-up message.

1.2. The currently playing song is stopped using the player's stop() method.

1.3. The player's playlist is cleared in preparation for the new song to be added.

1.4. The current track index is updated to the index of the newly selected song.

1.5. The selected song's URL is added to the player's playlist.

1.6. An attempt is made to play the newly added song using the player's play() method.

```
public void actionPerformed(ActionEvent e) {
    try {
        System.out.println("CURRUNT : "+curruntPlayingTrack);

        // Get the song title and artist name of the current track
        String songTitle = Utilities.alltracks.get(curruntPlayingTrack).getTrackName();
        String artistName = Utilities.alltracks.get(curruntPlayingTrack).getArtistName();

        // Update the moving text panel with the current track's artist name and track name
        MovingTextPanel(artistName+" , "+songTitle);
        // Generate a random animation index
        animationIndex = random.nextInt( bound: 10);
        if(timer1.isRunning())
            timer1.restart();
        else
            timer1.start();
        if (!animationTimer.isRunning()) {
            animationTimer.start();
        }
        // Stop the currently playing song
        player.stop();
        // Clear the player's playlist before adding the new track
        player.getPlayList().clear();
        // Add new track to player
        try {
            player.addToPlayList(new URL(Utilities.alltracks.get(curruntPlayingTrack).getUrl()));
        } catch (MalformedURLException ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog( parentComponent: null, message: "Invalid track URL", title: "Error", JOptionPane.ERROR_MESSAGE);
        }
        // Start playing the new song
        player.play();
```

2. Using Play Button:


```
public void actionPerformed(ActionEvent e) {
    try {
        System.out.println("CURRUNT : "+curruntPlayingTrack);

        // Get the song title and artist name of the current track
        String songTitle = Utilities.alltracks.get(curruntPlayingTrack).getTrackName();
        String artistName = Utilities.alltracks.get(curruntPlayingTrack).getArtistName();

        // Update the moving text panel with the current track's artist name and track name
        MovingTextPanel(artistName+" , "+songTitle);
        // Generate a random animation index
        animationIndex = random.nextInt( bound: 10);
        if(timer1.isRunning())
            timer1.restart();
        else
            timer1.start();
        if (!animationTimer.isRunning()) {
            animationTimer.start();
        }
        // Stop the currently playing song
        player.stop();
        // Clear the player's playlist before adding the new track
        player.getPlayList().clear();
        // Add new track to player
        try {
            player.addToPlayList(new URL(Utilities.alltracks.get(curruntPlayingTrack).getUrl()));
        } catch (MalformedURLException ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog( parentComponent: null, message: "Invalid track URL", title: "Error", JOptionPane.ERROR_MESSAGE);
        }
        // Start playing the new song
        player.play();
```

A button with a play icon is attached to an event listener. Upon clicking, the following operations are performed:

2.1. If the current track index is valid, the currently playing song is stopped and the player's playlist is cleared.

2.2. The selected song indicated by the current track index) is added to the player's playlist and the song is played.

2.3. The selected song's details are displayed in a moving text panel and the song play is logged. An error message is shown if no songs are available in the playlist.

3. Playing the Next Song: 

```
public void actionPerformed(ActionEvent e) {
    try {
        if(Utilities.alltracks.size()-1 > curruntPlayingTrack) {
            // Stop the currently playing song
            player.stop();

            // Clear the player's playlist
            player.getPlayList().clear();

            // Update current track index
            curruntPlayingTrack++;

            // Add next track to player
            try {
                player.addToPlayList(new URL(Utilities.alltracks.get(curruntPlayingTrack).getUrl()));
            } catch (MalformedURLException ex) {
                ex.printStackTrace();
                JOptionPane.showMessageDialog( parentComponent: null,  message: "Invalid track URL",  title: "Error", JOptionPane.ERROR_MESSAGE);
            }

            // Start playing the new song
            player.play();
```

A button with a next icon is attached to an event listener. Upon clicking, the following operations are performed:

3.1. If there is a song following the current track in the playlist, the currently playing song is stopped and the player's playlist is cleared.

3.2. The current track index is incremented and the next song is added to the player's playlist. The song is then played.

3.3. The new song's details are displayed in a moving text panel and the song play is logged. An error message is shown if no songs are available in the playlist.

4. Playing the Previous Song: 

```
public void actionPerformed(ActionEvent e) {
    try {
        // If the current playing track index is greater than or equal to 1
        if(curruntPlayingTrack>=1) {
            // Decrement the current track index
            curruntPlayingTrack--;
            // Skip to the previous track in the player
            player.skipBackward();

        }
        System.out.println("CURRUNT : "+curruntPlayingTrack);
        // If the timer2 is running
        if(timer2.isRunning())
            timer2.stop();// Stop the timer2
```

A button with a previous icon is attached to an event listener. Upon clicking, the following operations are performed:

4.1. If there is a song before the current track in the playlist, the currently playing song is stopped and the player's playlist is cleared.

4.2. The current track index is decremented and the previous song is added to the player's playlist. The song is then played.

4.3. The new song's details are displayed in a moving text panel and the song play is logged. An error message is shown if no songs are available in the playlist.

Pause/Stop Songs

1. Pausing a Song: 

```
    GuhD
    public void actionPerformed(ActionEvent e) {
        player.pause(); // Pause the player

        if(timer1.isRunning())
            timer1.stop();

    }
});
```

GuhDify features a button with a pause icon. This button is attached to an event listener which responds to user interaction. When clicked, the following sequence of operations is executed:

1.1. The music player is paused using the pause() method.

1.2. Concurrently, the system checks if there is a running timer, which is likely linked to visual animations synchronized with the song. If the timer is active, it is halted using the stop() method. This ensures the continuity and synchronization between the audio playback and the visual elements, even when the audio is paused.

2. Stopping a Song: 

```
    public void actionPerformed(ActionEvent e) {

        // Stop the player
        player.stop();
        // Stop the animation
        if (animationTimer.isRunning()) {
            animationTimer.stop();
        }
    }
});
```

The program provides a stop button displayed by a stop icon. Similar to the pause button, an event listener is attached to this button. When the button is clicked, the following operations are executed:

2.1. The music player is stopped using the stop() method. This action halts the playback of the current track.

2.2. The system checks if there is a running animation, likely tied to the song's playback. If the animation is active, it is halted using the stop() method. This ensures that visual feedback aligned with the audio playback is discontinued once the audio is stopped.

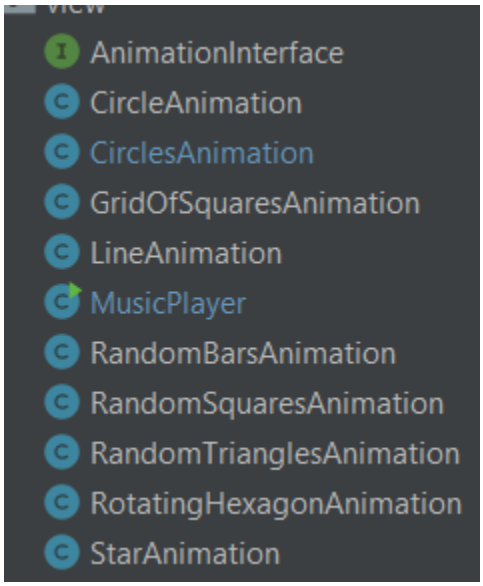
Animation

1. Animation Interface and Implementing Classes

The core of the animation lies in the AnimationInterface. This interface defines a single method, drawAnimation(Graphics2D g, int width, int height, Random random). Any class implementing this interface must provide a definition for this method.

```
void drawAnimation(Graphics2D g, int width, int height, Random random);
```

Various animation classes are provided, such as CircleAnimation, CirclesAnimation, GridOfSquaresAnimation, LineAnimation, RandomBarsAnimation, RandomSquaresAnimation, RandomTrianglesAnimation, RotatingHexagonAnimation, and StarAnimation. Each class contains its unique visualization algorithm which is drawn onto the panel when the drawAnimation method is called.




```
//putting the animation from the animation classes
animations = new HashMap<>();
animations.put(0, new CircleAnimation());
animations.put(1, new LineAnimation());
animations.put(2, new RotatingHexagonAnimation());
animations.put(3, new GridOfSquaresAnimation());
animations.put(4, new RandomBarsAnimation());
animations.put(5, new StarAnimation());
animations.put(6, new CirclesAnimation());
animations.put(7, new RandomSquaresAnimation());
animations.put(8, new RandomTrianglesAnimation());

timer1 = new Timer( delay: 1000 / 30, e -> panel_4.repaint());
animationTimer.start();
```

2. The Music Player Class and Animation Management

In the MusicPlayer class, a HashMap is initialized with an index as the key and corresponding animation as the value. An animationTimer is set to increment the animation index every 10 seconds, effectively switching the animation to be displayed on the screen.

The animationIndex is updated in a circular manner, i.e., after the last animation, the index is reset to the first animation. It ensures a seamless loop of the animations.

In the paintComponent method of the panel, the animation is chosen using the animationIndex, and the drawAnimation method is called to render the current animation.

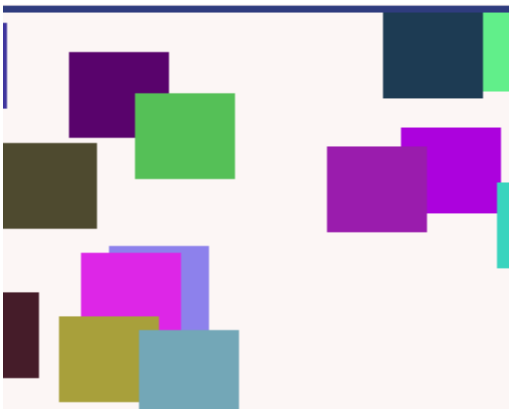
3. Execution Flow

The execution flow of the animation process is as follows:

1. The MusicPlayer class gets initiated. The animation HashMap is filled with

animation objects, each associated with a specific index.
2. The animationTimer starts running, which updates the animationIndex every 10 seconds.
3. The paintComponent method of the JPanel retrieves the current animation using the animationIndex.
4. The drawAnimation method of the current animation gets called. It draws the animation on the panel using the provided Graphics2D object, with width and height parameters and a Random object for randomization elements.
5. The cycle continues with the animationTimer updating the animationIndex, leading to a new animation being drawn every 10 seconds.

This animation system provides an appealing visual element to GuhDify. It adds an interesting dynamic aspect that enriches user experience, enhancing the aesthetic appeal of the music player. Its modularity and use of interface also mean that new animations can easily be added in the future.



Song Logs

The logging algorithm has two core functions:

initialize(): This function reads from the "playedSongs.txt" file and retrieves the last played song and artist details when the application starts. It is important to mention that the song title and artist name are separated by a delimiter " - ".

```
public void initialize() {
    try {
        List<String> allLines = Files.readAllLines(Paths.get( first: "playedSongs.txt"));

        if (!allLines.isEmpty()) {
            String lastLine = allLines.get(allLines.size() - 1);

            // Split the last line into song title and artist name using " - " as the delimiter
            String[] split = lastLine.split( regex: " - ");
            lastSongTitle = split[0].trim(); // Extract the last song title
            lastArtistName = split[1].trim(); // Extract the last artist name
        }
    } catch (IOException e) {
        System.out.println("An error occurred while reading from file.");
        e.printStackTrace();
    }
}
```

logSongPlay(String songTitle, String artistName): This function logs the current song being played, provided it is different from the previously logged song. It writes the song title and the artist's name into the "playedSongs.txt" file, separated by the delimiter " - ", followed by a newline character.

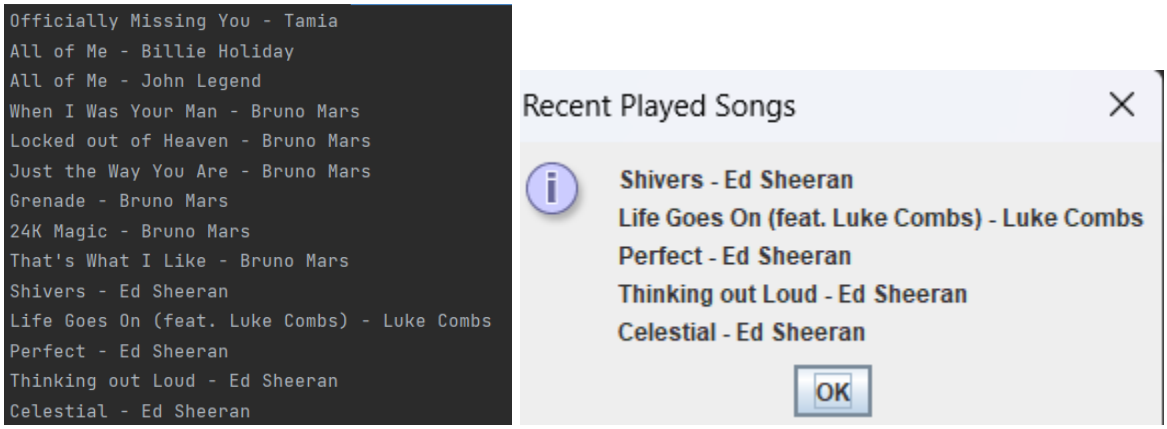
```
public void logSongPlay(String songTitle, String artistName) {
    // Check if the current song and artist are different from the last logged song
    if (!songTitle.trim().equals(lastSongTitle) || !artistName.trim().equals(lastArtistName)) {
        try {
            FileWriter writer = new FileWriter( fileName: "playedSongs.txt", append: true); // Open file in append mode
            writer.write( str: songTitle + " - " + artistName + "\n"); // Write the song and artist to a new line
            writer.close();

            // Update the last song played with the current song and artist
            lastSongTitle = songTitle;
            lastArtistName = artistName;
        } catch (IOException e) {
            System.out.println("An error occurred while writing to file.");
            e.printStackTrace();
        }
    }
}
```

The logging algorithm operates in the following way:

When the MusicPlayer application starts, it invokes the initialize() function to retrieve the last song played.

As songs are played on the application, the logSongPlay(String songTitle, String artistName) function is called. This function first checks if the current song being played is different from the last song logged by comparing the song title and artist name.

If the song or the artist is different, the function opens the "playedSongs.txt" file.



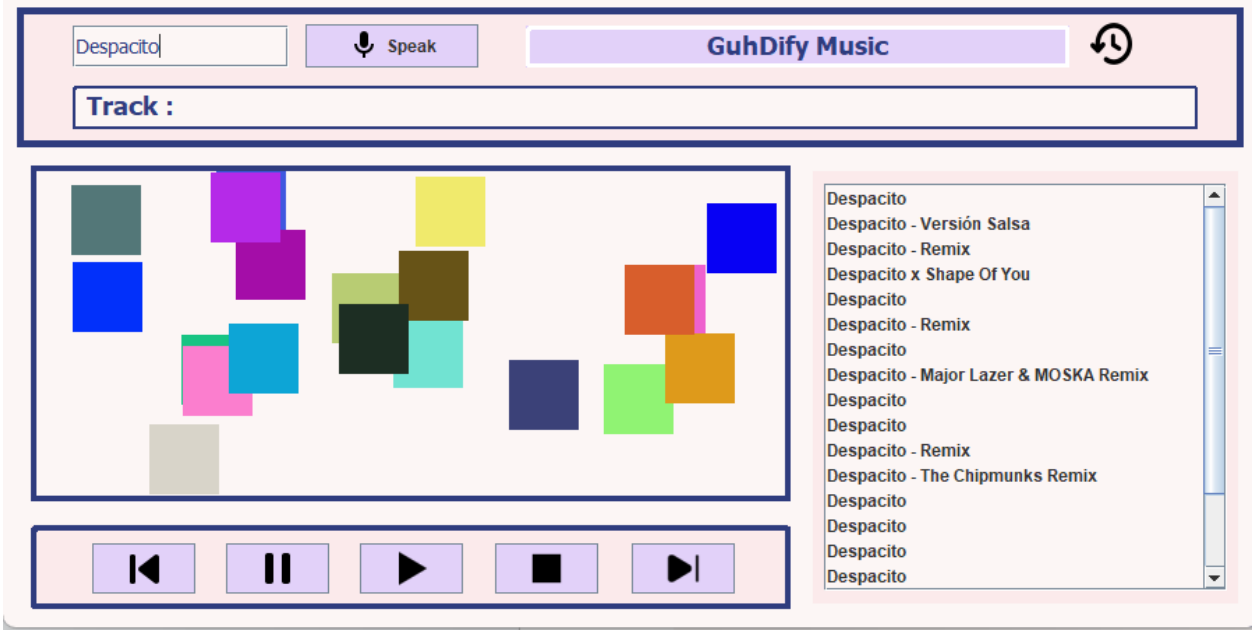
in append mode and writes the song title and artist name on a new line. It then updates the lastSongTitle and lastArtistName with the current song's details.

This function is invoked in several situations:

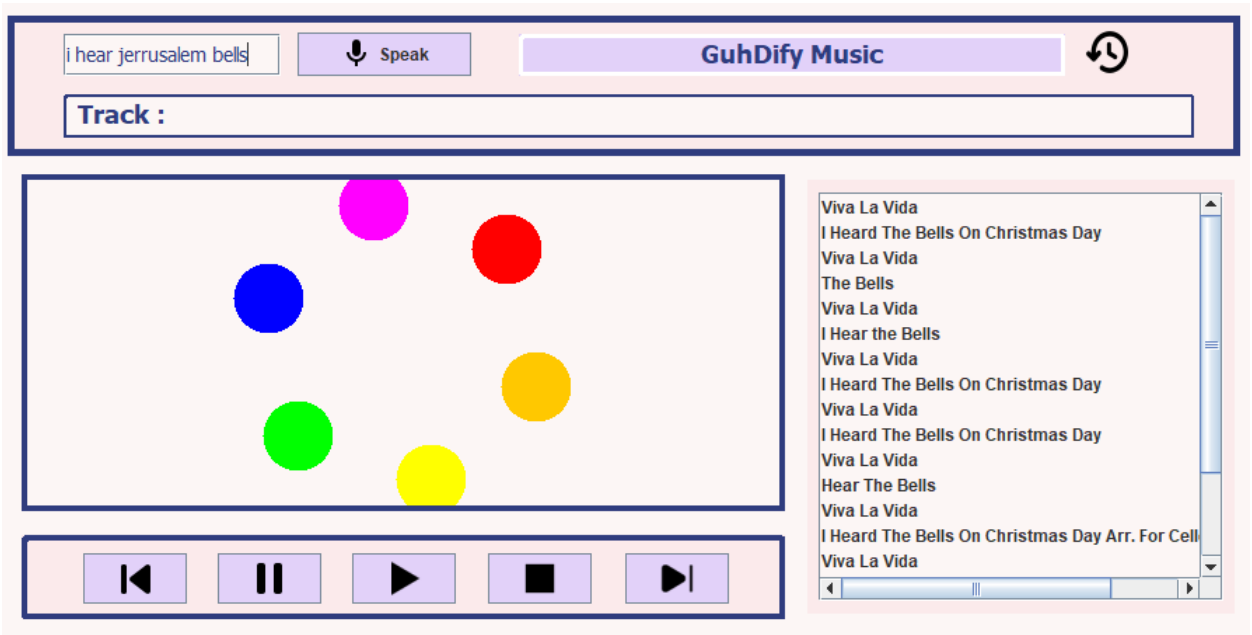
a. When a song is selected and played from the list.

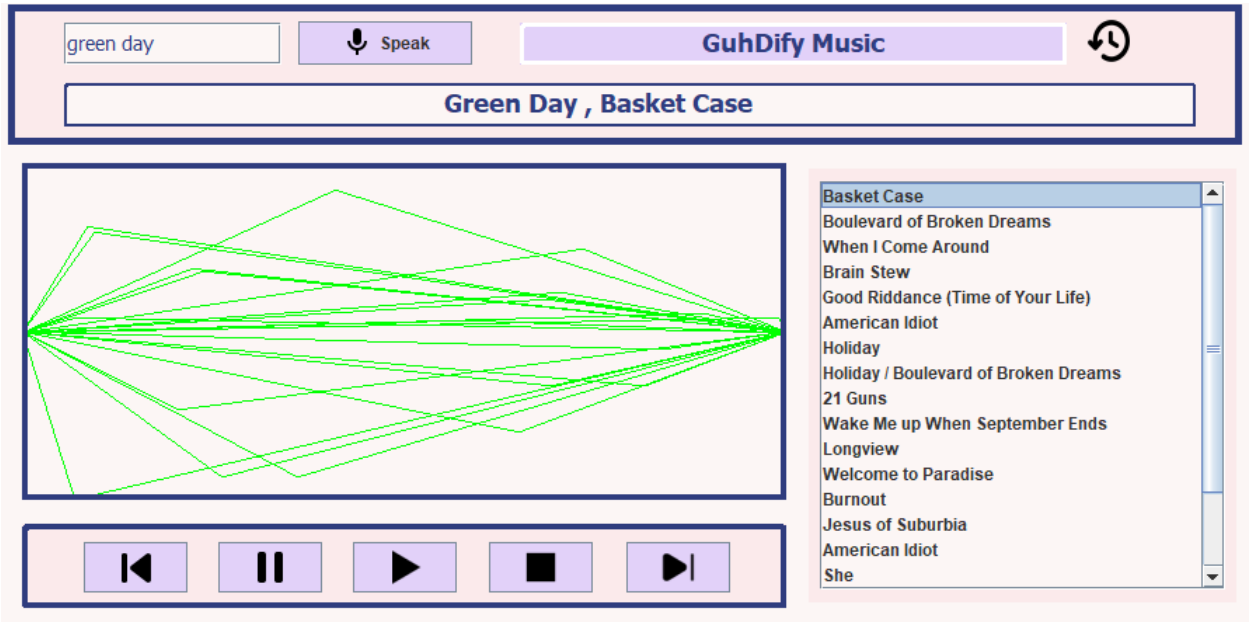
b. When the next or previous button is clicked and the song starts to play.

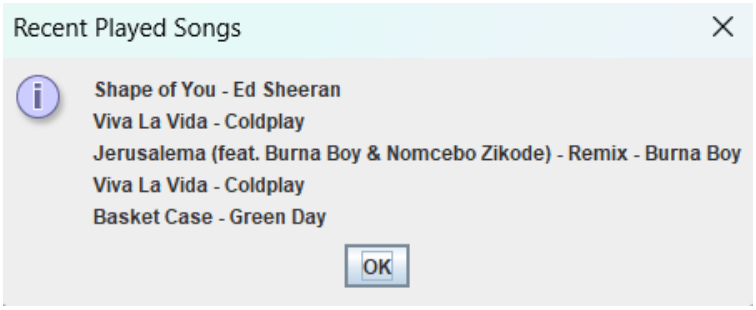
Evidence of a Working Program




Bruno mars
Speak
GuhDify Music
Track :
When I Was Your Man
Locked out of Heaven
Just the Way You Are
Treasure
24K Magic
Leave The Door Open
That's What I Like
Grenade
Leave The Door Open
Finesse
Talking to the Moon
Chunky
Uptown Funk (feat. Bruno Mars)
Gorilla
The Lazy Song
Straight up & Down


i hear jerrusalem bells
Speak
GuhDify Music
Track :
Viva La Vida
I Heard The Bells On Christmas Day
Viva La Vida
The Bells
Viva La Vida
I Hear the Bells
Viva La Vida
I Heard The Bells On Christmas Day
Viva La Vida
I Heard The Bells On Christmas Day
Viva La Vida
Hear The Bells
Viva La Vida
I Heard The Bells On Christmas Day Arr. For Cell
Viva La Vida





Reflection

During this project, I've learned a lot. I've become better at using Java, especially when it comes to building a user interface and handling files. Moreover, I have gained more knowledge on how to implement polymorphism as well as inheritance in Java. I've also learned how to use different types of data structures like ArrayLists, HashMaps, and lists. This has really improved my understanding of how data is stored and accessed in a program. I've also had the chance to work with animations, which was a fun way to make the music player more engaging.

Looking forward, there's definitely more I can do to get better. Even though GuhDify is working well right now, I know it could be more efficient. In the future, I'm going to work on improving the way my program reads from files. Instead of reading the entire file every time, I could use a smarter approach. I could also improve the user interface of my program, perhaps by making it more intuitive or by adding more features. I'm looking forward to learning more and applying what I learn to make my program even better.

Sources

Alex Lee. (2020, February 6). Java GUI Tutorial - Make a GUI in 13 Minutes [Video]. YouTube. https://www.youtube.com/watch?v=5o3fMLPY7qY

Alex Lee. (2020b, August 27). Java Polymorphism Tutorial - Polymorphism Example and Explanation [Video]. YouTube. https://www.youtube.com/watch?v=Ft88V_rDO4I

Bro Code. (2020, September 14). Java GUI: Full Course ☕ (FREE) [Video]. YouTube. https://www.youtube.com/watch?v=Kmgo00avvEw

Bro Code. (2021, March 13). JavaFX mp3 music player 🎵 [Video]. YouTube. https://www.youtube.com/watch?v=-D2OIekCKes

How to extract the data from spotify web API. (n.d.). Stack Overflow. https://stackoverflow.com/questions/57909369/how-to-extract-the-data-from-spotify-web-api

Lkuza. (n.d.). java-speech-api/src/main/java/com/darkprograms/speech/recognizer/GSpeechDuplex.java at master · lkuza2/java-speech-api. GitHub. https://github.com/lkuza2/java-speech-api/blob/master/src/main/java/com/darkprograms/speech/recognizer/GSpeechDuplex.java

Max O'Didily. (2022, November 19). How to Play Music Using Java (Simple) [Video]. YouTube. https://www.youtube.com/watch?v=wJO_cq5XeSA

Web API | Spotify for Developers. (n.d.). https://developer.spotify.com/documentation/web-api