Report: Music Album and Playlist Playing System

100263728

January 15, 2019

1 Introduction

The Sound Company is a newly established company striving to launch an innovative music player that is able to play audio files. The company wants to compete with the current market and is looking to create an original design that has great usability. The aim is to provide people with a music player that can play a range of album collections and playlists. The music player will also have the option for users to create their own playlists from their available tracks.

The target audience of the Sound Company is people of all ages but with a specific focus on ages 16-30. The music player is also angled towards those with a particular interest in music and would be appealing to this target market as it enables users to play their own collections of music and create their own original playlists. The company wants to make sure that they uphold their outstanding reputation, therefore the product produced must be highly reliable, functional, easy to navigate and aesthetically appealing to the target market. The music industry is a very popular area which is why the company wishes to enter this market with a product that could generate a large profit. If the music player is successful there would be consideration for further development, in order to help increase profit margins. Something that could be explored in the future is development of the music player to produce a premium music player, this could be something that the sound company could charge customers for.

1.1 The Problem

There are already companies such as Spotify that provide a similar service, they offer usage for free and customers pay a small fee for premium usage. Premium includes perks that remove adverts between songs and allow users to download and play their music without streaming. This threatens the Sound Company's opportunity for success as customers may already be using Spotify's music player and may be satisfied with this. Therefore, it is important to consider that people may not be willing to try another music player. This factor is an important consideration for the Sounds Company, as a lack of ability to

acquire customers could lead to failure of the launched product.

1.2 The Strategy

It is evident that Spotify has many advantages and has obtained a large customer base, however it is important to highlight that with their free subscription there are major negatives that may deter users away from the music player. These include the fact that you are unable to select the song that you want to listen to on certain devices which means that you have to listen to albums and playlists on shuffle, this may not be appealing to users. Another aspect is the use of adverts having to listen to multiple adverts before being able to listen to the music selected is off putting, as people use the music player to listen to music not adverts.

The Sound Company can make sure that the music player they design enables users to select the music they want to listen to without having to wait for it to shuffle to the song they want to play. Also adverts will not be incorporated into the music player, therefore music will not be disrupted and customers will be able to use the music player as they intended. By improving services provided, The Sound Company may be able to attract some of Spotify's users and encourage them to change the music player they use. This would be a very positive outcome for the company.

1.3 The Dilemma

To create a low cost music player that can provide all of the functionality that is desired could take longer than the expected deadline. However, it is possible for a decent product that could be launched to be produced in this time frame. This can be achieved by outlining which aspects are definitely required and what is not necessary but would be nice to implement during the design process. Furthermore, it is important to consider the improvements that may be made by Spotify in the development timeframe and whether this could make the product being developed have less of an impact when released on the current market.

1.4 The Plan

The Sound Company have decided that this product should be announced imminently so that advertising of the new product can be launched as soon as possible. Other company's designs will be analysed in order to inspire the design of The Sound Company's music player. In the design process the requirements of the music player will be outlined and extra features will be implemented if time constraints allow. Implementation of the designs will be executed with the necessary adaptations required to meet the projects targets. A

high level of testing will be used in order to ensure that the program produced is bug free and suitable for release to end users.

2 Analysis of Music Player Design

Several companies have developed highly functional music players that are available for public use, these have both their advantages and disadvantages when it comes to both design and usability. Analysis of a variety of music players will help to aid the design of this music player, selecting the positive aspects from competitors and also learning from the not so good features in order to make a competitive music player.

2.1 Spotify

Companies such as Spotify offer an appealing user interface with high functionality. Although this is a very popular music player for the current market it is important to highlight that if people are unfamiliar with the interface it can be difficult to navigate, as you have to launch the player in order to view the available music. This is a design aspect that could be improved in order to make it more user friendly and is something to consider when designing the user interface. The images of playlists are displayed nicely once the player is launched, they all have a title which gives an indication of what type of music the playlist contains. Ensuring that the description of the music offered is clear and concise is an important aspect to consider when designing, as being able to clearly distinguish between things makes the music player easier to use. Once you have clicked into a playlist or album all of the tracks are displayed clearly with both their track titles and durations. There is the option to click on a track to play the music or a button that you can click to play the whole playlist. This is designed very well as all information is clearly displayed in a simple manner and it functions well. Being able to see the total duration of the playlist could be added as this may be a useful piece of information for users.

A make a new playlist option is offered in the sidebar on Spotify, it is a good location to store the option as it is clear to users. There is also an option to name the playlist which is easy to use and adds a personal touch. Allowing users to freely name their playlists means that playlists can be distinguished by the users input. When making a playlist Spotify gives a list of recommended songs to add, this is a nice feature as it may aid the user in selecting songs that they enjoy and may even introduce them to new music that they haven't seen before. Due to the number of songs available on Spotify when adding songs you have to search for the song, this may be the most efficient way of making a playlist but it may not be overly appealing to users as a lot of searching is required. Spotify is very aesthetically pleasing, the simple colour scheme of black, green and white works well. The colours are contrasting and easy to read which are key requirements for a successful design.

2.2 Windows Media Player

Another well known music player is Windows media player which offers an extremely simple user interface. The program is easy to use as all items are clearly labelled and the play bar does exactly what would be expected. In terms of being aesthetically pleasing it could do with a bit of improvement in order to appeal to the target market. Nevertheless, it is important to highlight that the Windows media player differs from some of the other music players discussed, as there is no streaming service and therefore hard copies of files are needed. This aspect of the music player being designed is very similar to the design of this music player and can be learnt from. Another advantage of the Windows media player is that it can be used for not only music but can also play mp4 files. This increases the functionality of the player and increases the target audience due to its versatility. Making a playlist in the music player is very simple, after clicking on playlist you are given the option to 'click here' to make a playlist which you are then able to name. A simple drag and drop is used in order to add any tracks that are stored in your library. This music player keeps things simple whilst maintaining high functionality.

2.3 ITunes

ITunes has a neatly labelled sidebar with its options clearly outlined. Once you have selected one of the options (for example Albums) it displays the albums artwork with the artist name and the album name. Although this looks aesthetically appealing you are unable to see the tracks without clicking on an album. This means that if you want to view another album you then have to go back to click onto something else. The colour scheme is very simple using just black, white and grey which all work well contrasting against each other. ITunes has its own function for making playlists called Genius which allows you to select a song and it will mix songs that it thinks goes well together to generate a playlist. This is a useful feature for those that don't want to make a playlist and want the program to do the majority of the work. Making a playlist can be done by clicking the further options and adding a track to that playlist. This could be more user friendly as having to expand options to enable the addition of a track to a playlist seems more complex than is necessary. Having a recently added section is a nice touch, as users are likely to want to listen to their most recently downloaded tracks.

2.4 Sound Cloud

One particularly appealing aspect about using Soundcloud is that albums and playlists are shown under different categories for example "chill" or "workout", this means that users can select music based on how they are feeling or what they are doing. I believe that

this is appealing to the target market as music has already been sorted for them and they don't have to do much manual searching, allowing them to be less decisive when deciding what to listen to. Having the play button over the top of the image displayed is also a nice touch as the desired music can be played with minimal effort. On the other hand, the option to view the songs on the album or playlist upfront may be a better option as it would gives the user visibility of the tracks that would be played upon click of the play button. If the user wants to view the tracks, they would have to click into each individual album or playlist to view them and then click back in order to select something else; the design could be altered slightly to improve the usability of the player. For example, the option to display all of the albums or playlists and upon clicking on them they show the information and album cover based on what was selected on the same page. Once you have clicked into a particular album or playlist on Soundcloud it gives you the option to "Add to playlist" and "Add to Next up" these are both really useful features. The use of "Add to playlist" means that if the user is listening to a track on a playlist or album and if they really enjoy the track they are able to use this button to add it one of the playlists they have made for themselves. The "Add to Next up" is also a great feature as this allows the user to be able to queue up the music that they want to listen to next. This means that the user may not have to interact with the music player as often as it is now set up to do what the user requires. Soundcloud also offers the option to share the music you have been listening to, this feature is something that may be appealing to the target audience. Soundcloud shows the track information and the appropriate image of what is playing alongside the audio bar, this is very useful for users as it enables them to see what they are listening to without having to try and figure out what they had selected to listen to.

2.5 Key Features That Will Inspire the Design

Many positives have been discussed when evaluating competitor's music players, these aspects have all been taken on board and will be thought about in depth when designing the user interface for The Sound Company. It is also important to consider some of the poorer aspects of the designs, thinking about how these could be adapted in order to improve usability and ultimately make the music player appealing to the target market. The aspects that particularly stood out from the music players reviewed was Spotify's use of contrasting colours, this is an aspect that is really important when considering the overall design of the player. Spotify's simple way of displaying all of a playlist's or album collection's information was also something that was really appealing in terms of design. A user being able to view all of the information they need in one place is a massive positive and definitely something I will take in to consideration upon designing the music player. Windows media player utilises a clearly labelled play bar which does exactly what would be expected, I feel this is really important making the design simple but highly functional. The playing of audio files will be inspired by this utilising simple logos with known functions is something that will be incorporated in to the design. The simplicity of the Windows media player is something that will also inspire the design. Soundcloud

displays the information next to what is playing, this is a very useful feature that I think would be worth integrating in to the design of this music player as it improves the usability of the program. One of the really appealing aspects of the ITunes design is the use of a clearly labelled sidebar that provides the functionality of the music player. This is another aspect that could work really well if incorporated into the Graphical User Interface (GUI) design.

3 Software Design

3.1 Music Player Requirements

This music play will be able to play a variety of tracks, specifically it must be able to do the following:

- Load Album Collections
- Display album information
- Display album image
- Play any of the tracks displayed on an album
- Load Playlists
- Display playlist information
- Play tracks provided on the playlist
- Make new playlist
- Name new playlist
- Add and remove tracks from playlist
- Save playlist
- Stop, play, pause and resume tracks

Additional potential features:

- Information of the track playing with a display image
- Loop and shuffle track buttons
- Next and previous track
- Add album to playlist
- Duration bar

3.2 Music Player Use Cases

This document outlines how the requirements of the project into a set of use cases. Deliverables include:

- Actors: Identifying the actors for the music player
- Use cases both primary and secondary: This part describes the interaction that takes place between the actors and the music player. Although useful it is important to emphasise that use case do not detail how the project will be completed.

Actors: There is one main role played by users for the design of this program.

- User: This actor views both the albums and playlists information. It also interacts with the system in order to play and stop tracks found on both albums and playlists. The actor can also create their own playlist and save this under a name of their choice.

The actor for the use cases has been defined, moving forwards the next step is to utilise the previously outlined requirements in order to describe the interaction between the actor and the system.

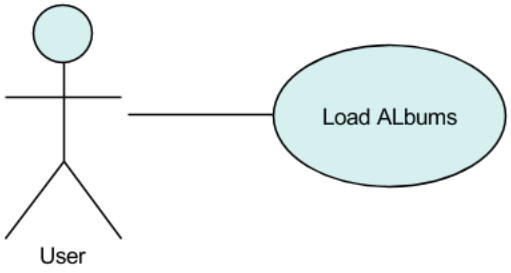
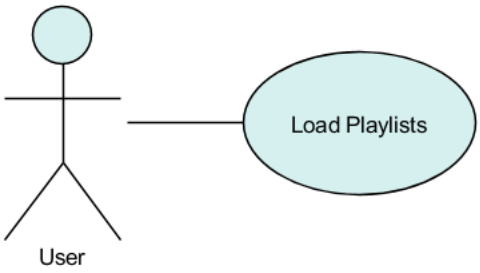


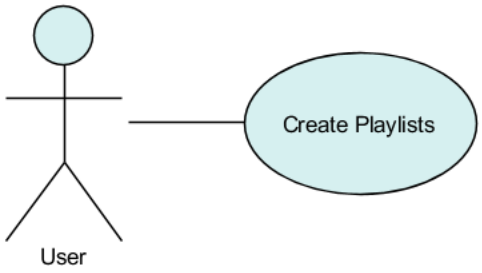
Figure 1: Load Album Collections

- Use case 1: Load the album collection information. The user clicks to load this information with which a file chooser allows them to select their required file. The system will then show the titles of all the albums and the names of the artists. Upon selecting an album the cover image of the album if available and its total duration will appear and also the list of the albums tracks with their durations will be populated.



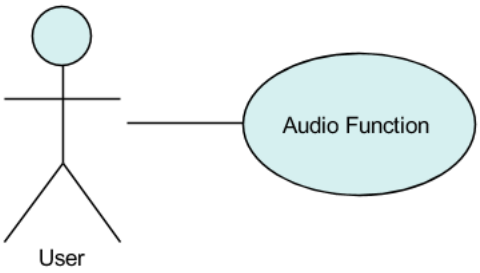
- Use case 2: Load the playlist information. The total duration of the playlist is displayed with the playlist name. The tracks on the playlist and their durations are also displayed. When a track is selected information about the album on which the selected track is stored is presented to the user.

Figure 2: Load Playlist



- Use case 3: Making a playlist using the information provided from the album collection and playlist files. The system will give the option of naming the playlist, adding tracks by selected from the album collections and playlists. Also the option to add albums and removing tracks are available. Once complete the user will be able to save their playlist and play it.

Figure 3: Create Playlist



- Use case 4: Use of audio function, when a track is selected it will automatically play. The user will have the option to pause and resume the track, stop the track.

Figure 4: Audio Function

The diagram below gives an overall representation of the user cases discussed, it shows where certain aspects are extended and how these aspect are related to each other.

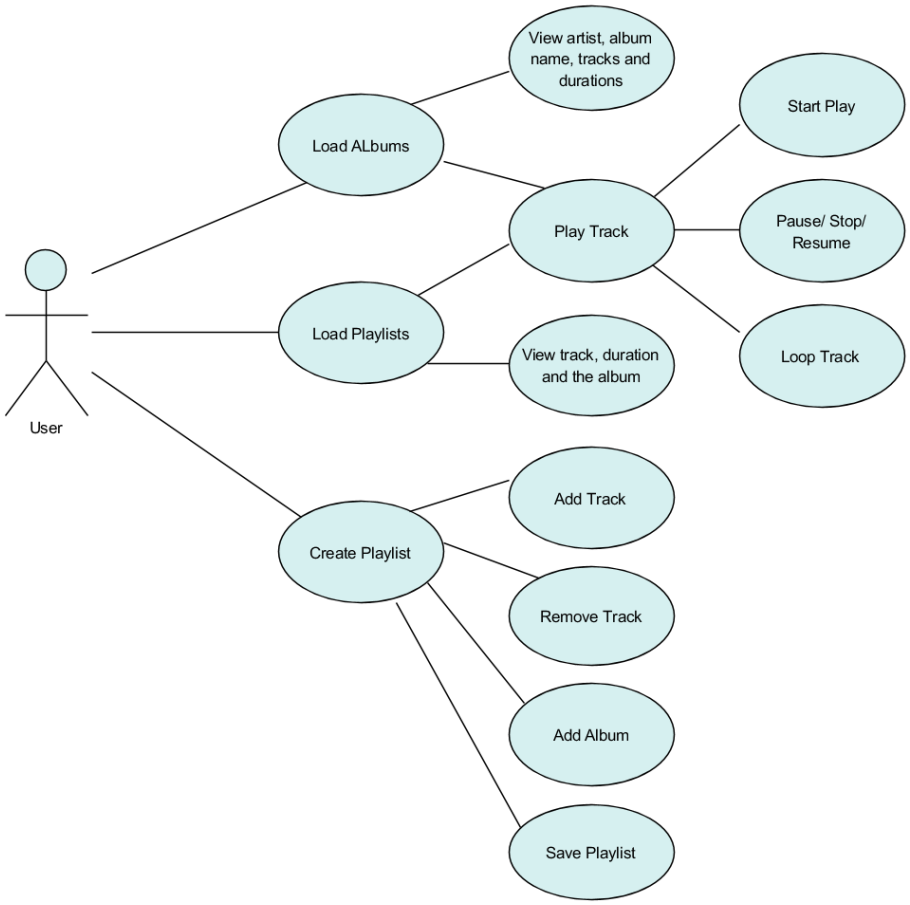


Figure 5: This diagram represents all of the use cases discussed and how they interact with each other

3.3 Unified Modelling Language (UML) Class Diagram

One of the primary goals for this project is to create a well structured architecture that enables users to listen to both their album collections and playlists and have the option of making new playlists. The music player program is comprised of multiple classes these include:

1. Duration, this is a class that is used to store the duration of the Tracks and Albums. The class is made up of three integer variables that represent the number of hours, minutes and seconds. The class contains methods to add durations.

2. Track: this class stores the details of an album track. It contains member variables which store both the title of the track as a String and the Duration of the track.

3. Album class is used to store the details of the album, the instance variables store the name of the artist and title of the album as String objects and also a collection of Track objects, representing the contents of the album.

4. AlbumCollection stores all of the details of a collection of music albums. The class has a member variable which is a collection of Album objects.

5. PlaylistTrack is a subclass of Track that extends the class and references the Album on which the track occurs.

6. Playlist represents a play list of tracks from an album collection. It has two member variables: a reference to the AlbumCollection from which the tracks are drawn and a collection of PlaylistTrack objects which represents the tracks.

7. MP3Player is used in order to read the audio file and utilises functions such as play, pause, stop and resume.

8. MusicPlayer is the class that holds the main JFrame for the program it contains GUI components methods that create interaction with components. This is the main class and it compiles functionality from other classes.

9. MakePlaylist is comprised of the second JFrame which is used in order to make playlists in the program. It uses a file writer to create text files in the specified folder path.

Displayed is a UML class diagram outlining the whole systems requirements based on the classes discussed above. See Appendix 9.1 for a more detailed UML class diagram.

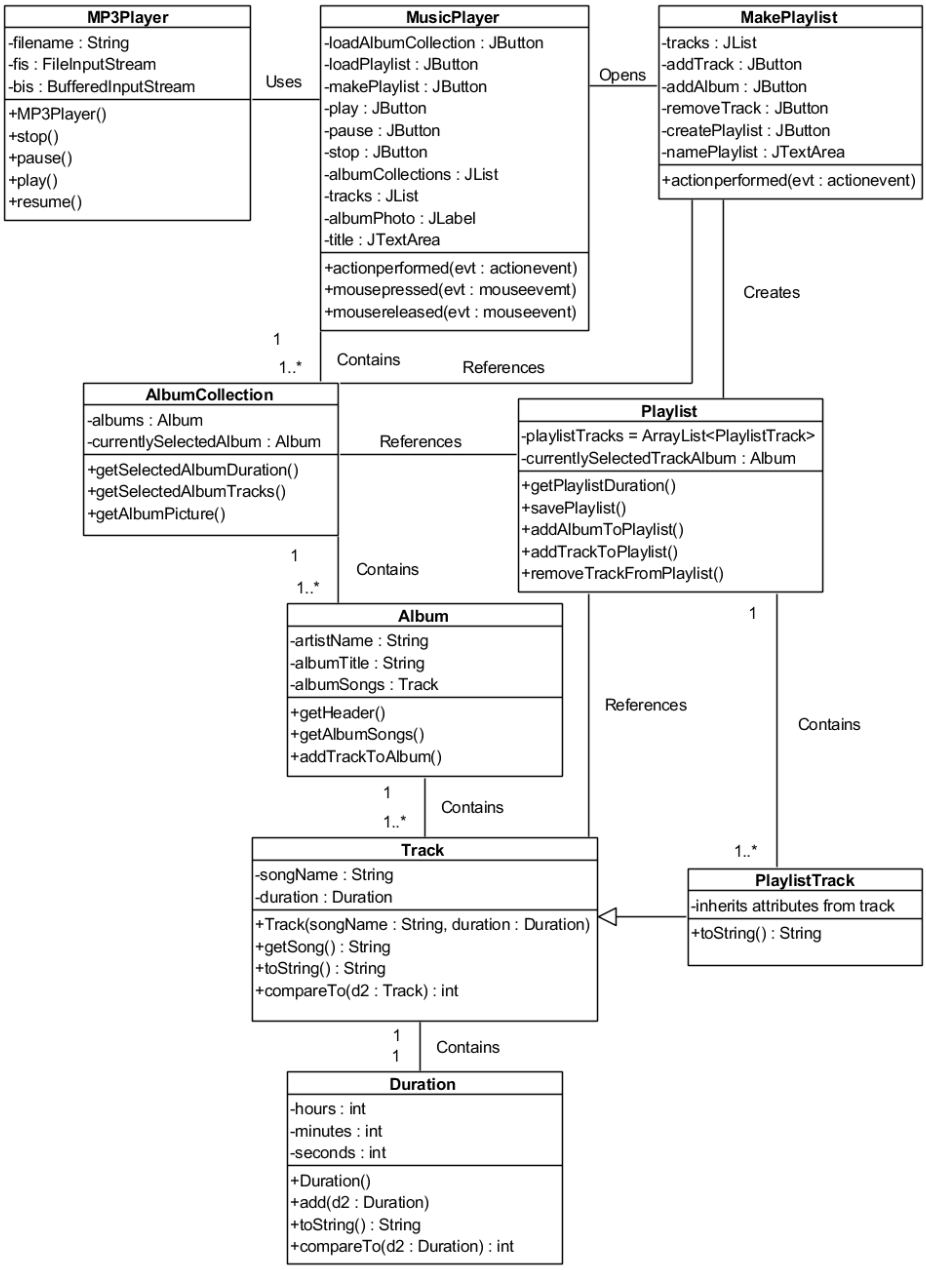


Figure 6: UML class diagram design of the music player

The diagram demonstrates the relationships between the classes and also defines some of the important attributes and methods that each class may require, in order to provide the required functionality of the program. A track has duration and is contained in an album which can also be located in an album collection. Loading of an album collection is one of the key features of the music player. The MusicPlayer has a load albums button that once clicked opens a file chooser that enables the user to select the album collection file

that they wish to display. A function then gets the selected file and populates an album array with all albums and tracks and also populates the user interface list with the album information from the text file. When one of the album titles is clicked a method gets the selected albums tracks and displays them in the tracks list.

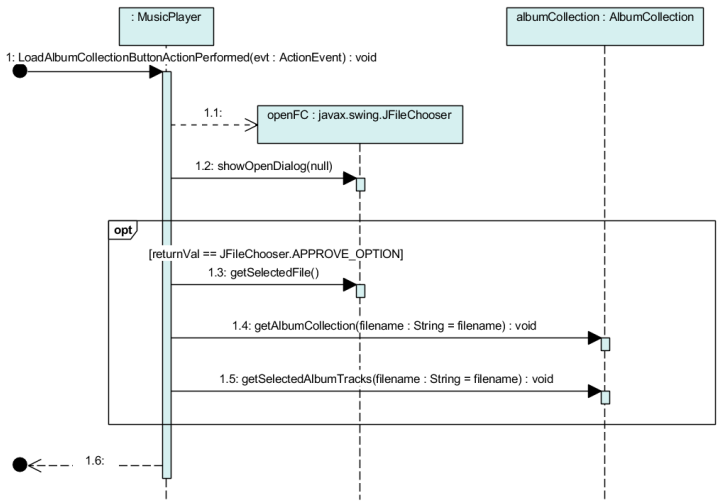


Figure 7: Sequence diagram showing the events for loading an album collection

This sequence of events is really important in the design implementation because if a step is not accounted for the program will not be able to function as expected. Furthermore, the user interface will be unable to display the required information and therefore the program will be unable to progress any further as no audio tracks would be able to be played.

Similarly to the album collection a playlist has both tracks and their durations and references the album collection in order to locate which album a track is on. The MusicPlayer is also crucial in the creation of new playlists as it provides the button that opens the MakePlaylist function. This aspect of the music player requires the information from both the album collection and playlist files and is therefore seen as a later aspect of the design process. A button to create the playlist would be clicked on the original frame that would cause a new frame to open. Once the playlist has been named the user would be able to save the playlist.

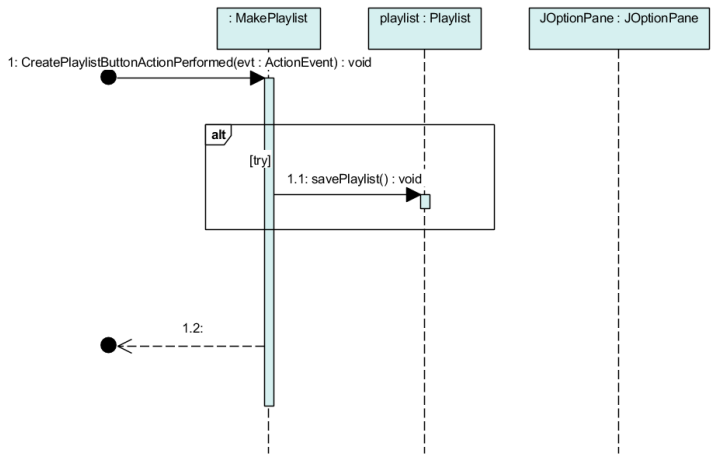


Figure 8: Sequence diagram showing the events for creating a playlist

After re-evaluation of the sequence diagram it was considered that some key design aspects may be absent. Although the user may know that they have clicked save on the playlist, with no acknowledgement of this actually saving confusion could occur. Two extra features that could be useful to implement include: when the save button is clicked the frame is automatically disposed of, this would make it clearer to users that their action of clicking the button has had an effect. The second feature could be a notification that the playlist has been saved, this could be useful as the user may click the save button multiple times which may lead to issues arising within the program. A revision of this sequence diagram is displayed below with the revised steps.

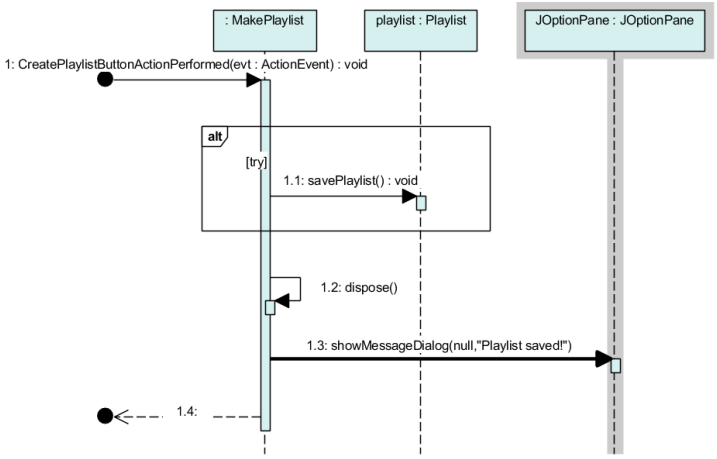


Figure 9: Revised sequence diagram showing the events for creating a playlist

The MusicPlayer class utilises MP3Player in order to play the relevant audio track selected. The track that the user wants to play needs to be clicked in order to implement the action.

The track will be present in either a playlist or an album and therefore there are two paths for this action to take. These sequence paths are displayed in the diagram below.

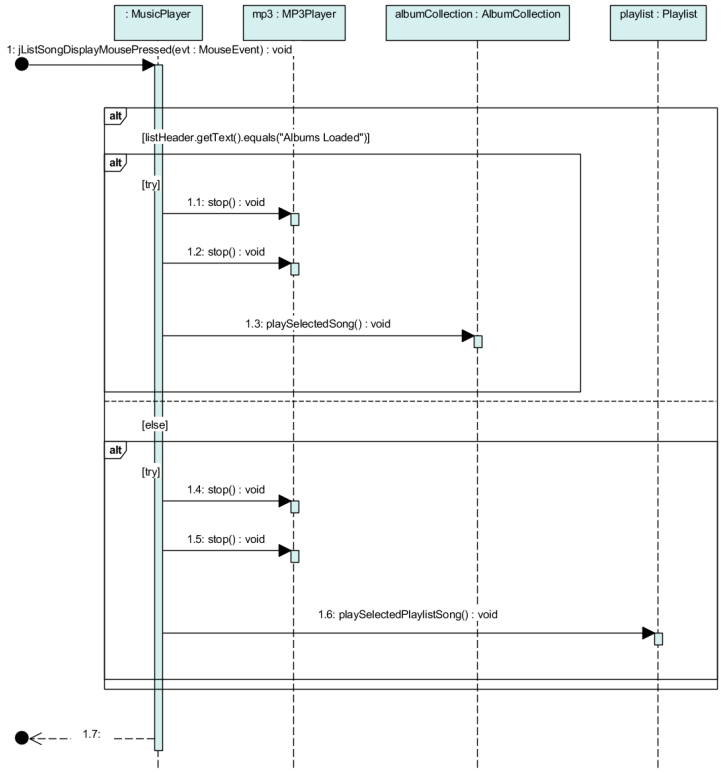


Figure 10: Sequence diagram showing the events for playing a track

A song is selected from the list of tracks displayed, the method then needs to define if the track is from either an album collection or a playlist. It could do this by checking the raw text of the header to see if it equals albums loaded, if it does not equal this the assumption would be that it was a playlist. To ensure there is no overlap from tracks when they are playing two stop methods, one for playlist one for album would need to be implemented. Then both the playlist and album collection path can play the selected song.

It is key to investigate whether the software is too complex. When thinking about this UML design it was taken into consideration that the software is likely to need to change over time. Although this design is not the simplest it does allow for changes to be made to what the class does without having to alter large amounts.

4 The GUI Design

The layout of the GUI has been based on the market research of other music players and the UML diagram. Highlighting positive features of other music players has inspired the design of the GUI for this project. The design of the GUI needs to be both aesthetically and ergonomically pleasing for the target audience. Production of a music player that is simple and easy to use was one of the main targets of this project. The music player needs to be able to show the list of albums or playlists and when selected to display further information of the selected item, enabling users to be able to play a selected track.

4.1 Load Album Collection and Load Playlist Design

Ideally the load playlist and load album collection would have their own buttons but would work in a similar manner using the same JFrame layout. Once the button has been click a file opener should pop up so that the user can select their file. The empty box located on the left of the screen would then should all of the album or playlist information stored in the text file. The user would then have the option to select the album that they would like to view more information on. This would then be able to auto populate the boxes to the right including the image in the square box, the album information and duration to the right of the image and lastly the tracks with their durations below this.

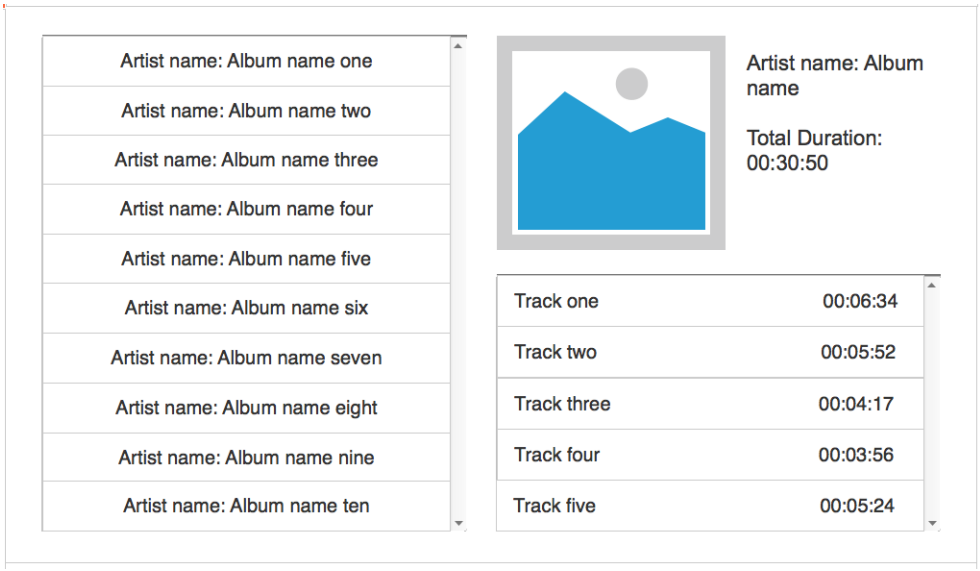


Figure 11: Design for loading of album collections

For the playlist the layout will be similar to that of the album collection. When a playlist is loaded in the left hand box the playlist name will be displayed with the playlists total duration. The tracks that are stored on the playlist will be auto populated in the bottom

right box, with their durations which are retrieved from the album collection information. Once a track is clicked on, the top left box above the track box will display the album cover for that track if one is available and the top right will show the tracks album information.

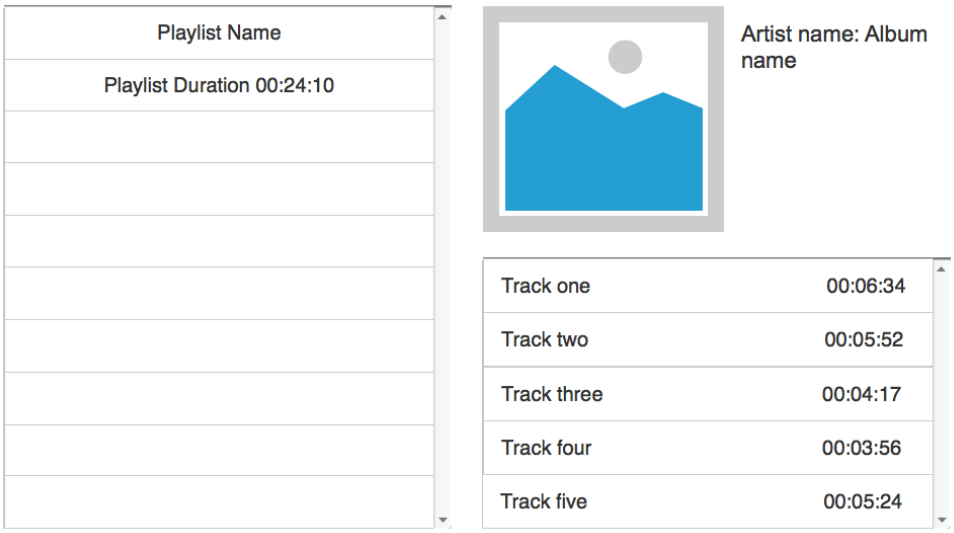


Figure 12: Design for loading of playlist

4.2 Load Album Collection and Load Playlist GUI Components

The GUI components that would work best for this design layout would be the use of buttons for the loading of both the playlists and albums. Use of a file chooser so that users have the option to choose what they would like to load. JLists would be the best option for the box shown on the far left that allows the loading of album collections and playlists. This would work best as it enables the lines from the text file to be separated so that they are considered their own items. This means that it would be possible to add the functionality of clicking on individual items, helping to show specific information based on the item selected. The use of a JList would also be considered the best method for the box displaying the tracks, as these need to be individual items that can be played when selected. A JLabel may be considered the best GUI component to display the images of the album collections as it has the capability of displaying images and also the options for resizing. A JTextArea would be suitable for displaying the extra information displayed next to the image showing the album information and its duration.

4.3 Create New Playlist Design

- This aspect of the music player needs to allow users to be able to create their own playlist from the tracks available from both the album collections and the playlists. This will have a simple design that allows users to select tracks from both album collections and playlists and add them to their own playlist. The create new playlist will overlap the main music player keeping the tracks in view of the user, which will be achieved by making this aspect overlie the sidebar of the music player.

- The main part of this frame will be a box which will display the tracks that the user has added. The user should be able to name their playlist by typing in their desired name in to a box situated above the main box, with a create playlist button located at the bottom of the make playlist panel for when the playlist is complete. They should also be able to add and remove tracks or albums from their custom built playlist. These buttons will be positioned at the bottom of the frame and will be clearly labelled.

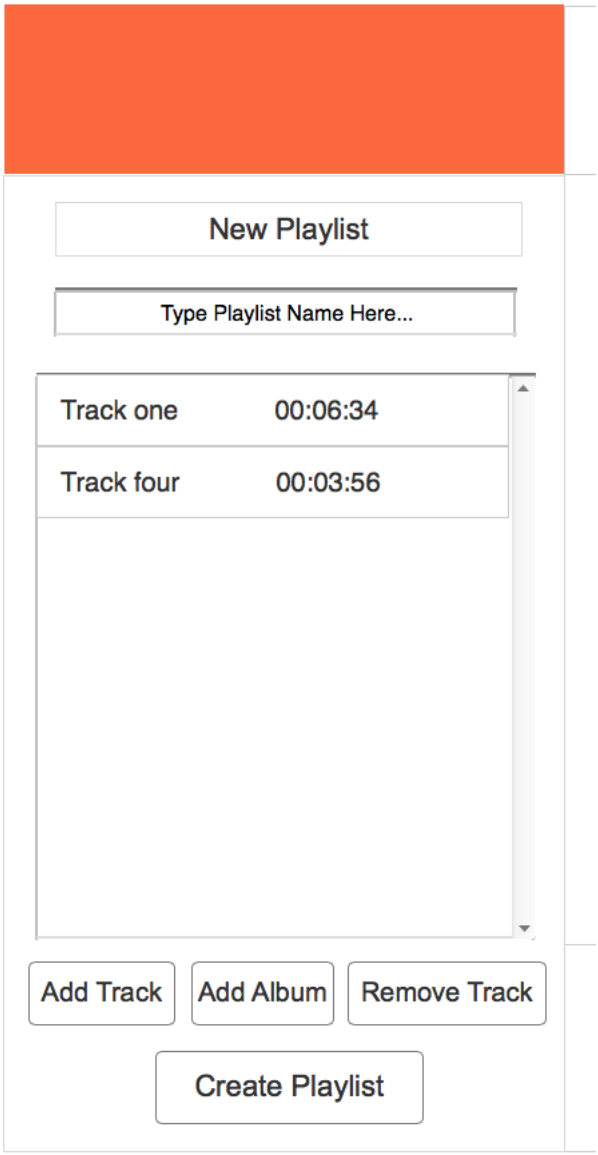


Figure 13: Design for creating playlist

4.4 Create New Playlist GUI Components

This will be created using a second JFrame that overlies the first, it is important that this frame always lies over the top of the main JFrame. Again the use of a JList will be used to display the individual tracks added to the playlist. There will be a text field present

in which the user can manually type the name that they wish to call their playlist file. Buttons will be implemented in order to save the playlist, add and remove tracks and albums and clear the playlist.

4.5 Audio Function

This is the aspect of the music player that provides the functionality for playing the music. This will be actioned by clicking on one of the tracks displayed. The design includes play, pause, stop, loop and shuffle buttons to provide the functionality needed for a music player. The design also displays the duration bar below the buttons so the user is able to see how far along their track is. The track playing with the album information or playlist information is to be displayed to the left of the buttons along with the image of the album collection if one is present. This is a simple design that encompasses the main functions required by the target audience.

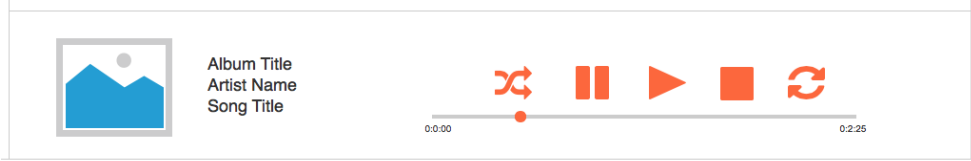


Figure 14: Design for audio function

When the loop button is on it will present the image as green to make it clear to the user that they have chosen to repeat the track selected.

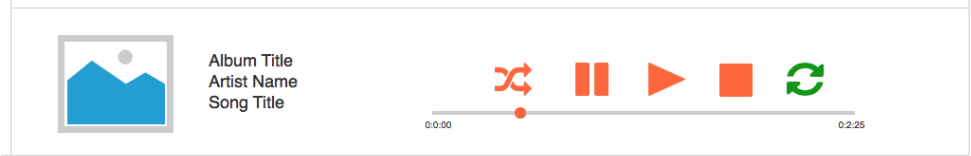


Figure 15: Design for audio function loop

4.6 Audio Function GUI Components

The main components required for this design would be the use of buttons, all of the functions would require their own individual button. The logo for each button would be displayed in the button keeping the interface nice and simple. The album cover image would again be best displayed in a JLabel and resized to the appropriate size. All text that is used to describe what is playing could be stored in a JTextArea.

4.7 Overall Design

In the first design of the GUI the buttons to load albums, load playlist and make playlist were situated at the top of the page. However, with more thought about the layout of the music player it made more sense for the buttons to be populated as a side bar. All buttons are neatly organised in a column and it is clear as to which button to click based on the function you require. This also means that the music player application fits the screen better as a landscape design andso when it is resized items tend to fit the screen better.

The colours selected for this deign are black, white, grey and a touch of orange. These colours all work well as they are contrasting and eye-catching. The text is easy to read and the icons coloured orange stand out against their background, making it clear as to what they are to be used for. Although the colour scheme may be considered simple the colours work well together and are appropriate considering the target audience. All text is coloured black and works well against its contrasting background. The font and its size selected for the whole program design is simple and easy to read. The title of the program is in a fancier font in order to make the program more appealing. Designs of the overall look of each aspect of the music player have also been included.

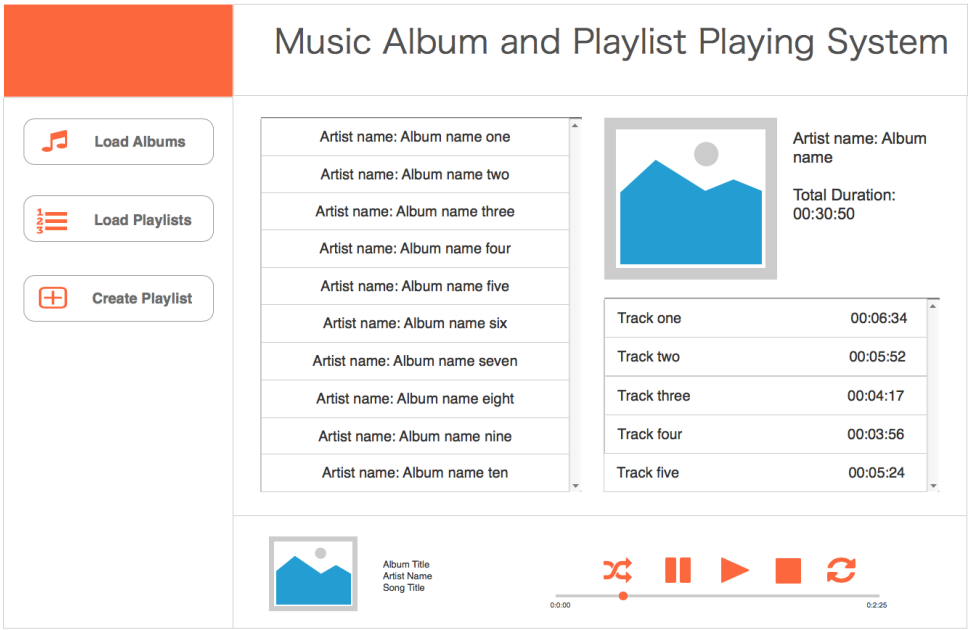


Figure 16: Overall design for loading of album collections

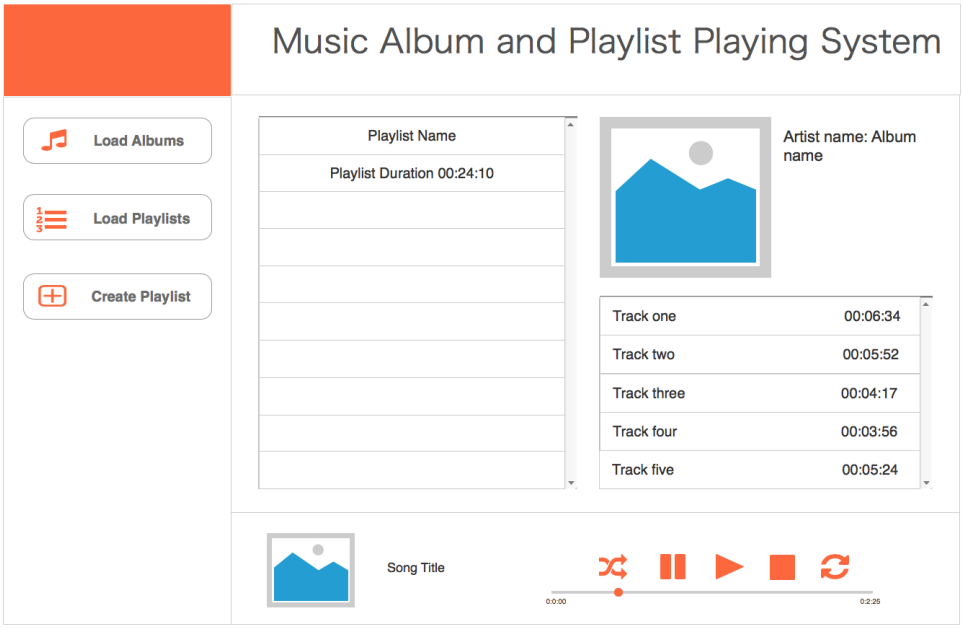


Figure 17: Overall design for loading of playlists

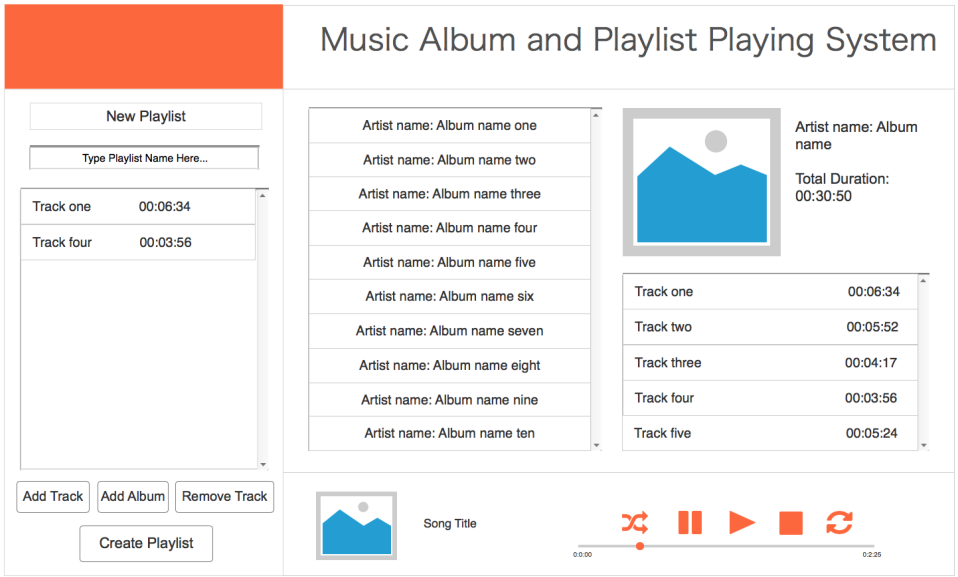


Figure 18: Overall design for creating new playlists

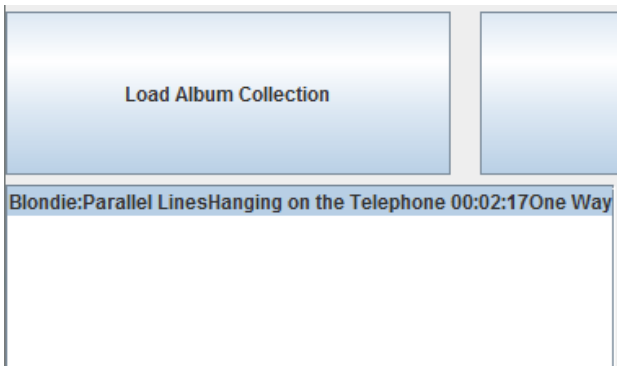
5 GUI Implementation

5.1 Load Album Collection Implementation

A JList was selected to display all of the albums with both their title and the artist name in the selected text file. A JList was chosen for the reason that it enables all items to be split so that they are able to be individual selected. First attempt at code for the load album collection is shown below, initially all albums were displayed on one line of the JList.

```
JFileChooser openFC = new JFileChooser("C:\\Users\\Holly Birch\\Documents\\NetBeansProjects\\AlbumInfoProgram\\text files");
int returnVal = openFC.showOpenDialog(null);
if(returnVal == JFileChooser.APPROVE_OPTION){
    File txtfile = openFC.getSelectedFile();
    String filename = txtfile + "";

albumCollection.getAlbumCollection(filename);
albumCollection.sortAlbums();

//int val = 200;
//for(int i=0;i<val;i++){
String fileread = albumCollection + "";
listModel.addElement(fileread);
//}
jList1.setModel(listModel);

}
```

The commented out for-loop just repeated the same action however on 200 lines of code



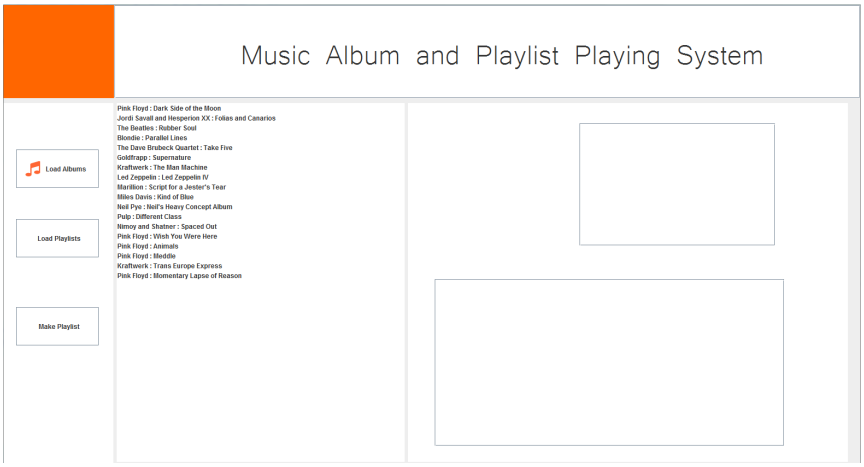
Several problems were identified with the initial code. Firstly, the method was stored in the MusicPlayer class, the code is for AlbumCollection and therefore should be stored in that class. Secondly having everything contained in a string on one line, after researching how JLists work it was evident that each element of a JList equates to a new line. In order to rectify these issues the method was moved to the AlbumCollection class and a for loop was added to read through the filename and create albums. Tracks are added but only album titles are added to the listModel.

```
public void getAlbumTitles(String filename) {

    try {
        //Reading in the file
        File albumFile = new File(filename);
        BufferedReader in
                = new BufferedReader(new FileReader(albumFile));
        String currentLine;

        Album currentAlbum = null;
        //Loop to read through the lines in the text file
        //Checks if char 1 an 4 are: if yes adds track to album
        //If not makes new album

        while ((currentLine = in.readLine()) != null) {

            if (currentAlbum == null) {
                currentAlbum = Album.makeNewAlbum(currentLine);
            } else if (AlbumCollection.
                    isStringTrackInformation(currentLine)) {
        Track currentTrack = new Track(currentLine);
                currentAlbum.addTrackToAlbum(currentTrack);
            } else {
                this.addAlbum(currentAlbum);
                currentAlbum = Album.makeNewAlbum(currentLine);
                listModel.addElement(currentLine);

            }
        }

        MusicPlayer.jListTxtFileDisplay.setModel(listModel);

        in.close();

    } catch (IOException e) {
        e.printStackTrace();

    }
}
```

Filename object is created in MusicPlayer class and method called:

```
private void LoadAlbumCollectionButtonActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser openFC = new JFileChooser("C:\\Users\\Holly Birch\\Documents\\NetBeansProjects\\AlbumInfoProgram\\text files");
    int returnVal = openFC.showOpenDialog(null);
    if(returnVal == JFileChooser.APPROVE_OPTION){
        File txtfile = openFC.getSelectedFile();
        String filename = txtfile + "";

    albumCollection.getAlbumTitles(filename);

    }
```

When this is called it gives the following output:

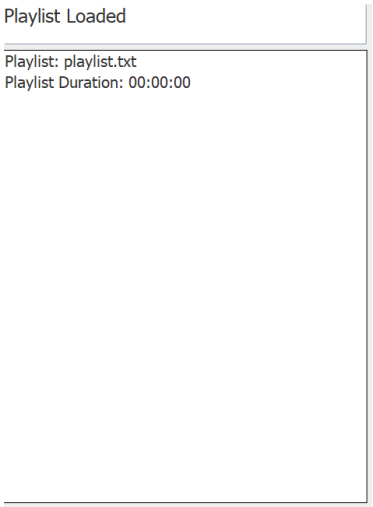


Throughout the implementation of the load album collection aspects had to be altered slightly in order to ensure the functionality of the music player. Some adaptions included adding the shown listHeader so that it was possible to identify when either an album collection was loaded or a playlist was loaded.

This aspect was added to loadAlbumCollection code to ensure that this box was populated correctly:

```
//Checking the listHeader textbox and setting it to "Albums Loaded"
try{
    if(MusicPlayer.listHeader.getText().equals("Playlists Loaded")){
        MusicPlayer.listHeader.setText("Albums Loaded");
    }

    else if(MusicPlayer.listHeader != null){
        MusicPlayer.listHeader.setText("Albums Loaded");
    }

    else{}
}catch (Exception e){
}

try {
    //Reading in the file
```

This was then tested and also works when the playlist is loaded first – changes text to album loaded



When loading the same album collection twice, tracks were being duplicated. In order to fix this issue I decided to change the code for this method. The method name was changed to getAlbumCollection and the following code was added to check the "albums" (this) array for the album on the current line of the bufferedreader. If the album was not matched, the code could proceed. This also prevented tracks being added as the albums were not created.

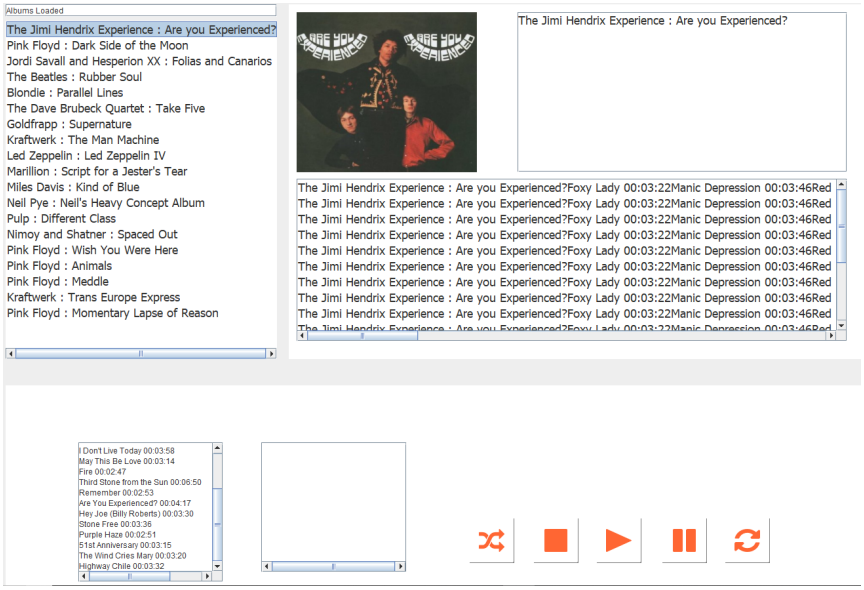
```
} else {
    //Added this code to check the albumCollection first to ensure
    //that we are not getting duplicate albums added to the array
    Album matchedAlbum = this.findAlbum(currentAlbum.getArtistName(), currentAlbum.getAlbumTitle());
    if(matchedAlbum == null) {
        //Also populating a listModel with album titles which
        //is to be displayed on jListTxtFileDisplay
        listModel.addElement(currentAlbum.getArtistName() + " : " + currentAlbum.getAlbumTitle());
        this.addAlbum(currentAlbum);
    }
```

5.2 Load Album Tracks Implementation

Use of another JList was chosen to display album tracks and also for the album info area which displays the album name, artist name and the total duration of the album.

On click of the JList that displays the list of the album collection I wanted to display the corresponding tracks of that album. Shown is the initial attempt at code which returned the selected album JList element with square brackets around it, I reformatted this to remove the brackets. I used JTextArea 4 (bottom left box) to check my returned album string code.

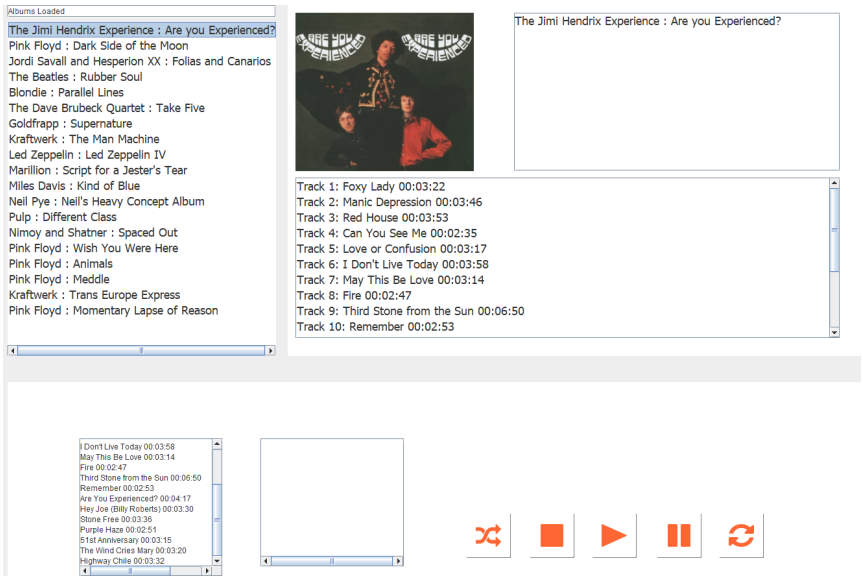
```
public void getAlbumTracks(String filename){

    String selectedJList = MusicPlayer.jListTxtFileDisplay.getSelectedValue() + "";
    selectedJList = selectedJList.replaceAll("\\[", "").replaceAll("\\]","");
    int i=0;
    if(listModel2 != null);

    try{
        listModel2.clear();



        MusicPlayer.AlbumInfoBox.setText(selectedJList);//+albumDuration);
        System.out.println("set Text Area 1 to " + selectedJList);

    }catch(Exception e){}

    String albumString = getselectedAlbum() + "";

    MusicPlayer.jTextArea4.setText(albumString);

    for (Album album : albums){
        String selectedAlbum = selectedJList;


            for (Track track : album.getAlbumsongs()){

                if(album.getHeader().equals(selectedAlbum)){

                listModel2.addElement(albumString);
            }

        }

    MusicPlayer.jListSongDisplay.setModel(listModel2);

    }
}
```



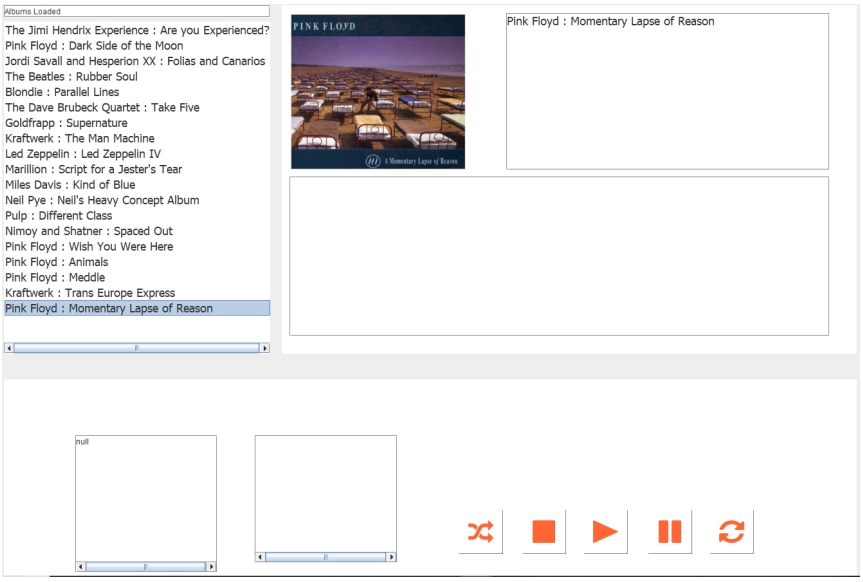
However, this string could only be added to one element of the albumSongsJList which meant that all of the songs on the album were displayed as one element on one line. After a rethink I attempted to loop through all albums until the album header matched the selected album. If they matched, I would loop through all of the tracks on that album and add them as elements to the JListSongs model.

```
public void getAlbumTracks(String filename){

    String selectedJList = MusicPlayer.jListTxtFileDisplay.getSelectedValue() + "";
    selectedJList = selectedJList.replaceAll("\\[", "").replaceAll("\\]","");
    int i=0;
    if(listModel2 != null);

    try{
        listModel2.clear();



        MusicPlayer.AlbumInfoBox.setText(selectedJList);//+albumDuration);
        System.out.println("set Text Area 1 to " + selectedJList);

    }catch(Exception e){}

    String albumString = getselectedAlbum() + "";

    MusicPlayer.jTextArea4.setText(albumString);

    for (Album album : albums){
        String selectedAlbum = selectedJList;


            for (Track track : album.getAlbumsongs()){

                if(album.getHeader().equals(selectedAlbum)){

                i = i + 1;
                System.out.println("getHeader: " + album.getHeader());
                System.out.println("selectedAlbum: " + selectedAlbum);
                //album.getHeader() isn't picking up the last album for some reason

                String albumString2 = "Track " + i +": " + track;
                listModel2.addElement(albumString2);

            }

        }

    MusicPlayer.jListSongDisplay.setModel(listModel2);

    }
}
```

This was successful and returned the desired effect:



However, it didn't appear to return the last album's tracks as shown:



I utilised System.out.println to test what was being returned and print this information to the console log. This revealed that the getHeader() method was not registering the last album. The fixing of the LoadAlbumCollection caused the getHeader() method to read in the last album. Comments and a definition for a class variable (currentlySelectedAlbum) were added to the method.

```
public void getSelectedAlbumTracks(String filename){
    //Method to get tracks of selected album in jListTxtFileDisplay
    String selectedJList = MusicPlayer.jListTxtFileDisplay.getSelectedValue() + "";
    selectedJList = selectedJList.replaceAll("\\[", "").replaceAll("\\]","");
    int i = 0;
    //Setting the AlbumInfoBox to the selected jList value
    try{
        listModel2.clear();
        listmodel2tracks = new ArrayList<>();
        MusicPlayer.AlbumInfoBox.setText(selectedJList);
        System.out.println("set Text Area 1 to " + selectedJList);
    }catch(Exception e){
    }
    //Loop through album collection searching for selected jList value.
    //When found, loop through album songs and add tracks as elements to
    //listModel2. track objects also added to "listmodel2tracks" array to
    //aid with onclick mp3 playing.
    for (Album album : albums){
        String selectedAlbum = selectedJList;
        if(album.getHeader().equals(selectedAlbum)){
            //Setting album to the currently selected album
            currentlySelectedAlbum = album;
            for (Track track : album.getAlbumsongs()){
                i = i + 1;
                String albumString2 = "Track " + i +": " + track;
                listModel2.addElement(albumString2);
                listmodel2tracks.add(track);
            }

        }
        //Displaying populated listModel2
        MusicPlayer.jListSongDisplay.setModel(listModel2);
    }
}
```

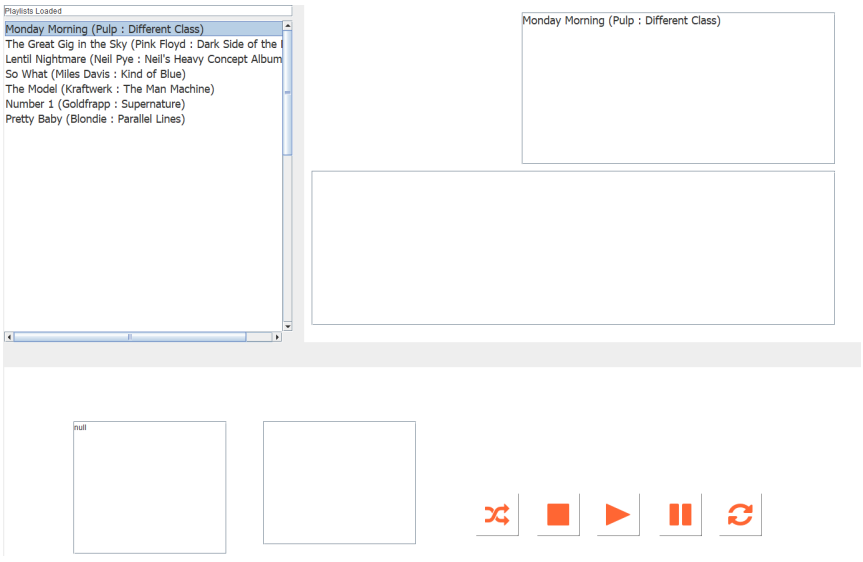
Also added was a function to add tracks to an array called listmodel2tracks – these two additions aided me in playing the mp3 for the selected song.

5.3 Load Playlist Implementation

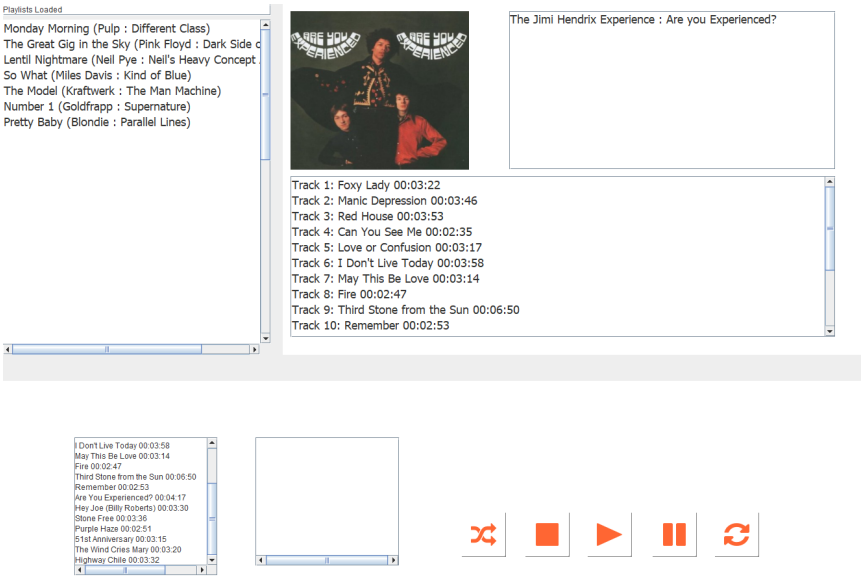
When initially implementing code for this section I hardcoded the file path and forced a loop to get some functionality:

```
private void LoadPlaylistButtonActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser openFC = new JFileChooser("C:\\Users\\Holly Birch\\Documents\\NetBeansProjects\\AlbumInfoProgram\\text files\\Playlists");
    int returnVal = openFC.showOpenDialog(null);
    if(returnVal == JFileChooser.APPROVE_OPTION){
        File txtfile = openFC.getSelectedFile();
        String filename = txtfile + "";
    loadedAlbum = openFC + "";



        BufferedReader br = null;

    //this isn't working very well - not using the filereader in AlbumCollection
    try{
        br = new BufferedReader(new FileReader(filename));
        int val = 500;
      for (int i=-1; i<val; i++){


            String ss = br.readLine();
            listModel.addElement(ss);
        }
        jListTxtFileDisplay.setModel(listModel);


    }   catch (FileNotFoundException ex) {

    }   catch (IOException ex) {

        }

        }
    MusicPlayer.listHeader.setText("Playlists Loaded");

}
```

This appeared to work and read in the playlist as expected:



However I then realised that if Album tracks are loaded first, these stayed in the box:



Once other functionality was fixed, I reviewed the code in order to implement the use of a filename which was defined in the MusicPlayer class. This utilises a bufferedreader that loops through the selected file and added each line as a track to currentPlaylist. Following this, all playlist tracks were added to the JListSongDisplay model as correctly formatted elements.

```
public void getPlaylist(String filename, String displayedFilename) throws FileNotFoundException, IOException{


            BufferedReader br = new BufferedReader(new FileReader(filename));
            String currentLine;
            Playlist currentPlaylist = new Playlist(this);
          try{
              listModel2.clear();

            while ((currentLine = br.readLine()) != null) {
                    currentPlaylist.addTrackToPlaylist(currentLine);
            }
            for(PlaylistTrack track : currentPlaylist.playlistTracks){

                listModel2.addElement(track.getSong() + " (" + track.album.getArtistName() + " : " + track.album.getAlbumTitle() + ") " + track.getDuration());

            }



        jListSongDisplay.setModel(listModel2);


          }catch(Exception e)()
    MusicPlayer.listHeader.setText("Playlist Loaded");
    MusicPlayer.AlbumInfoBox.setText("Playlist: " + displayedFilename + "\n" + currentPlaylist.getPlaylistDuration());
}
```

```
private void LoadPlaylistButtonActionPerformed(java.awt.event.ActionEvent evt) {
    albumCollection.listClear();
    JFileChooser openFC = new JFileChooser("C:\\Users\\Holly Birch\\Documents\\NetBeansProjects\\AlbumInfoProgram\\text files\\Playlists");
    int returnVal = openFC.showOpenDialog(null);
    if(returnVal == JFileChooser.APPROVE_OPTION){
        File txtfile = openFC.getSelectedFile();
        String filename = txtfile + "";
        String [] filenamesplit = filename.split("Playlists");
        String displayedFilename = filenamesplit[1].replace("\\","") +"";
    loadedAlbum = openFC + "";
        try {
            albumCollection.getPlaylist(filename, displayedFilename);
        } catch (IOException ex) {

        }


        }

}
```

Creating the method in this way also allowed me access to the getPlaylistDuration method

which could also be displayed in the albuminfoBox (top right):

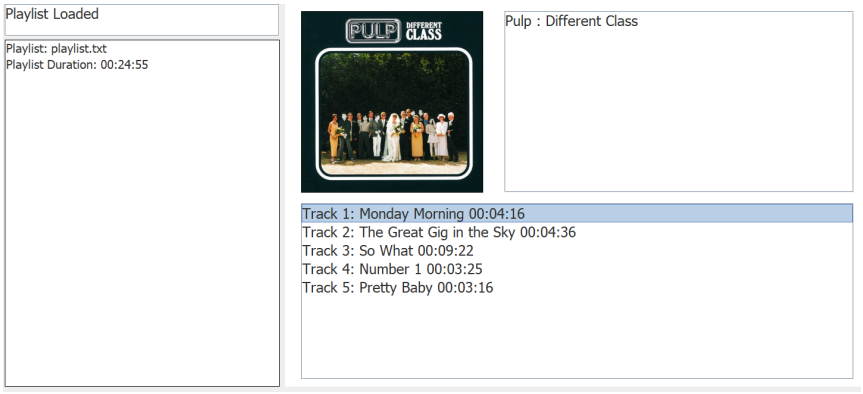
Having this method implemented now meant that I needed album collections loaded prior to loading a playlist as I was no longer simply printing the lines of the playlist file. The final iteration of code is shown below:

```
public void getPlaylist(String filename, String displayedFilename) throws FileNotFoundException, IOException{
    //Reading in the file
    BufferedReader br = new BufferedReader(new FileReader(filename));
    String currentLine;
    //Declaring Playlist variable
    Playlist currentPlaylist = new Playlist(this.albumCollection);
        try{
            //Clearing listModels that are to be populated
            listModel2.clear();
            listmodel2tracks.clear();

            int i = 0;
            //All lines are playlist tracks so:
            while ((currentLine = br.readLine()) != null) {
                currentPlaylist.addTrackToPlaylist(currentLine);
            }
            for(PlaylistTrack track : currentPlaylist.playlistTracks){
                //Adding playlist tracks to the listModel displayed
                //in jListSongDisplay
                i = i + 1;
                listModel2.addElement("Track " + i + ": " + track.getSong() + " " + track.getDuration());
            }

            jListSongDisplay.setModel(listModel2);
        }catch(Exception e){
        }

    //Setting UI display to have relevent playlist information
    MusicPlayer.listHeader.setText("Playlist Loaded");
    MusicPlayer.AlbumInfoBox.setText("Playlist: " + displayedFilename + "\n" + currentPlaylist.getPlaylistDuration());
    playlistInfoModel.clear();
    playlistInfoModel.addElement("Playlist: " + displayedFilename);
    playlistInfoModel.addElement("Playlist Duration: " + currentPlaylist.getPlaylistDuration());
    MusicPlayer.jListTxtFileDisplay.setModel(playlistInfoModel);
    MusicPlayer.AlbumInfoBox.setText("Playlist: " + displayedFilename + "\n" + currentPlaylist.getPlaylistDuration());
}
```

This code moved playlist information into the left side JList instead of the album info box. This allowed another method to be implemented which retrieves the album title and header of the selected playlist track and puts this information into the album info box.

```
public String displayPlaylistTrackArtistandAlbum(){
    //Method used when playlist is loaded.
    //Displays the artist and album of the selected playlist track.
    String selectedSongx = jListSongDisplay.getSelectedValue();
    String [] split1 = selectedSongx.split(": ");
    String [] split2 = split1[1].split(" 00:");
    String selectedSong = split2[0]+"";

    for(Album album : albumCollection.albums){
        for(Track track : album.getAlbumsongs()){
            if(track.getSong().equals(selectedSong)){
                MusicPlayer.AlbumInfoBox.setText(album.getHeader());
            }
        }
    }
    return null;
}
```

This had the added bonus of causing the album picture to show as well:



5.4 Make Playlist Implementation

A new Jframe was created for the creation of new playlists as this seemed like the most feasible option for this project. One of the main issues that was first encountered when making this aspect of the GUI was that when exiting the new make playlist window the whole music player was closing. In order to fix this problem I changed MakePlaylist JFrame properties DefaultCloseOperation from EXIT_ON_CLOSE to DISPOSE. This was successful, the fixing of this issue helped to improve the functionality of the music player. I had an idea to implement an 'add album' button, I thought of getting the listModel populated with tracks for the selected album. However I made an error when writing the code and received the following message: ListModel¡String¿ cannot be converted to DefaultListModel<String>. I had to put (DefaultListModel<String>) before my .getModel method to rectify this. The code below was implemented to ensure that I was pulling right components on click.

```
public void addToPlaylist(){

    String selectedTrack = MusicPlayer.jListSongDisplay.getSelectedValue();
    DefaultListModel<String> selectedAlbum = (DefaultListModel<String>) MusicPlayer.jListSongDisplay.getModel();

    System.out.println(selectedTrack);
    System.out.println(selectedAlbum);

}
```

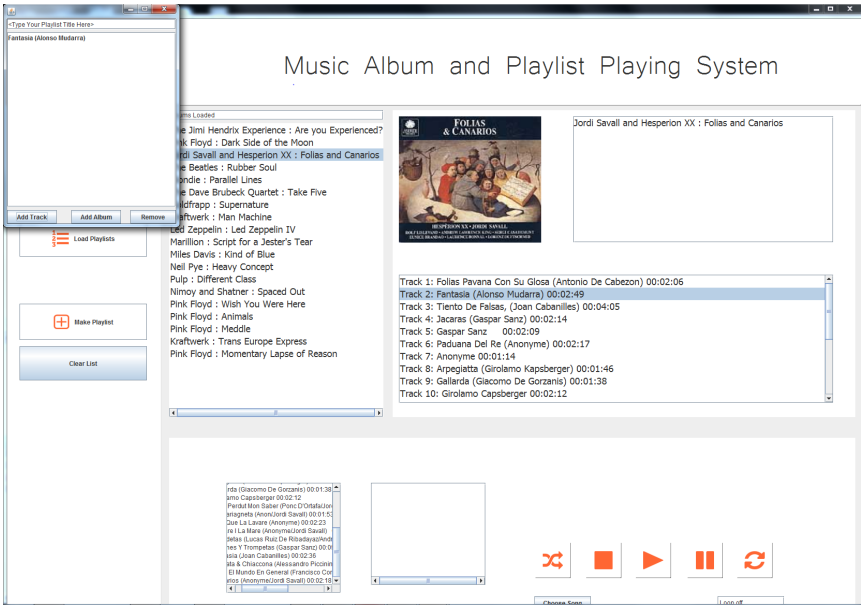
```
Track 1: Foxy Lady 00:03:22
[Track 1: Foxy Lady 00:03:22, Track 2: Manic Depression 00:03:46, Track 3: Red House 00:03:53, Track 4: Can You See Me 00:02:55, Track 5: Love or Confusion 00:03:17, Track 6: I Don't Live Today 00:03:58, Track 7: May This Be Love 00:03:14, Track 8: Fire 00:02:47, T
```

I decided that the best approach was to have two separate buttons with two separate methods: add track or add album to playlist:

5.4.1 Add Track

The code written is a split function which separates the Track name from the duration and track number. Also shown below is the testing of this code:

```
public void addTrackToPlaylist(){

    String selectedTrack = MusicPlayer.jListSongDisplay.getSelectedValue();
    String [] selectedTrackSplit = selectedTrack.split (": ");
    String stringTrackSplit = selectedTrackSplit[1]+"";
        String [] selectedTrackSplit2 = stringTrackSplit.split("00:");


            playlistModel.addElement(selectedTrackSplit2[0]);

    MakePlaylist.NewPlaylistTracks.setModel(playlistModel);


}


public void addAlbumToPlaylist(){

    DefaultListModel<String> selectedAlbum = (DefaultListModel<String>) MusicPlayer.jListSongDisplay.getModel();

    System.out.println(selectedAlbum);

}
```



It was noted that the playlist text format was: ¡track name¿ (¡artist¿ : ¡album¿). I was aware that ¡artist¿ : ¡album¿ was already being pulled when Album Collection was loaded. Furthermore, the correct artist and album has to be selected in order to add a track. Therefore, using a getElement method on the artist and album list allowed me to write the correct playlist format.

```
public void addTrackToPlaylist(){

    String selectedTrack = MusicPlayer.jListSongDisplay.getSelectedValue();
    String selectedArtistAndAlbum = MusicPlayer.jListTxtFileDisplay.getSelectedValue();

    String [] selectedTrackSplit = selectedTrack.split (": ");
        String stringTrackSplit = selectedTrackSplit[1]+"";
            String [] selectedTrackSplit2 = stringTrackSplit.split("00:");



    String finalElement = selectedTrackSplit2[0] + " (" + selectedArtistAndAlbum + ")";
            playlistModel.addElement(finalElement);
            MakePlaylist.NewPlaylistTracks.setModel(playlistModel);
}
```

Further addition to “add track” button: playlist tracks are a different format to album tracks. To help identify if an album collection or playlist is loaded an identifier method was added.

```
public static boolean isAlbumLoaded(String line) {
    if (line.equals("Albums Loaded")){
        return true;
    }
    return false;
}
```

If an album is loaded, the split function will run, else the track will just be taken as a playlist track (correct format)

```
public void addTrackToPlaylist(){

    String selectedTrack = MusicPlayer.jListSongDisplay.getSelectedValue();
    String selectedPlaylistTrack = MusicPlayer.jListSongDisplay.getSelectedValue();
    String header = MusicPlayer.listHeader.getText();
    String [] selectedTrackSplit = selectedTrack.split (": ");
        String stringTrackSplit = selectedTrackSplit[1]+"";
            String [] selectedTrackSplit2 = stringTrackSplit.split("00:");

    String selectedArtistAndAlbum = MusicPlayer.jListTxtFileDisplay.getSelectedValue();
    if(AlbumCollection.isAlbumLoaded(header)){
    String finalElement = selectedTrackSplit2[0] + " (" + selectedArtistAndAlbum + ")";
            playlistModel.addElement(finalElement);
    }
    else{
            playlistModel.addElement(selectedPlaylistTrack);
    }
    MakePlaylist.NewPlaylistTracks.setModel(playlistModel);


}
```

Further work on the MusicPlayer meant that the playlist tracks and album tracks were the same format. Therefore, I removed the identifier and corrected to a consistent method and added comments for clarity.

```
public void addTrackToPlaylist(){

    //Method for the "add" button in the MakePlaylist window.
    //The code below takes the required elements from the UI and puts them
    //together into the given playlist format.
    String selectedAlbumTrack = MusicPlayer.jListSongDisplay.getSelectedValue();
        String [] selectedTrackSplit = selectedAlbumTrack.split (": ");
            String stringTrackSplit = selectedTrackSplit[1]+"";
                String [] selectedTrackSplit2 = stringTrackSplit.split("00:");

    String selectedArtistAndAlbum = MusicPlayer.AlbumInfoBox.getText();
        String [] selectedArtistAndAlbumSplit = selectedArtistAndAlbum.split("[\\r\\n]");

    String finalElement = selectedTrackSplit2[0] + "(" + selectedArtistAndAlbumSplit[0] + ")";

    //Every time this method is called (every click of the button) the
    //finalElement is added to the playlistModel.
    playlistModel.addElement(finalElement);

    MakePlaylist.NewPlaylistTracks.setModel(playlistModel);
}
```

5.4.2 Add Album

Intention of the method is to add the entire album to a playlist. Utilised the fact that I knew a selected album would populate the JListSongDisplay. Therefore, this method gets the JListSongDisplay model and loops through it to put all elements into correct playlist format. It then adds these formatted elements to the MakePlaylist jList. Testing showed the adequate working of this method as displayed.

```java
public void addAlbumToPlaylist(){

    //Method for the "add album" button in the MakePlaylist window.
    //Takes advantage of the fact that the jListSongDisplay model is
    //populated when an album is clicked.

    //Creating a model called selectedAlbum and populating it with the
    //selected album's songs.
    DefaultListModel<String> selectedAlbum = (DefaultListModel<String>) MusicPlayer.jListSongDisplay.getModel();
    String selectedArtistAndAlbum = MusicPlayer.jListTxtFileDisplay.getSelectedValue();

    //Looping through the selectedAlbum model, reformatting the values to
    //the playlist format and adding the reformatted values as elements to
    //the playlistModel
    for(int i = 0; i< selectedAlbum.getSize();i++){

        String trackName = selectedAlbum.getElementAt(i);
        String [] trackNameSplit = trackName.split(": ");
        String trackName2 = trackNameSplit[1];
        String [] trackNameSplit2 = trackName2.split(" 00");
        String trackNameFinal = trackNameSplit2[0] + " (" + selectedArtistAndAlbum + ")";
        playlistModel.addElement(trackNameFinal);
        System.out.println("ADDALBUMTOPLAYLIST TEST: " + trackNameFinal);
    }
    MakePlaylist.NewPlaylistTracks.setModel(playlistModel);
    }
```

```
Selected Album filename:Beatles_RubberSoul.jpg
ADDALBUMTOPLAYLIST TEST: Drive My Car (The Beatles : Rubber Soul)
ADDALBUMTOPLAYLIST TEST: Norwegian Wood (This Bird Has Flown) (The Beatles : Rubber Soul)
ADDALBUMTOPLAYLIST TEST: You Won't See Me (The Beatles : Rubber Soul)
ADDALBUMTOPLAYLIST TEST: Nowhere Man (The Beatles : Rubber Soul)
ADDALBUMTOPLAYLIST TEST: Think for Yourself (The Beatles : Rubber Soul)
ADDALBUMTOPLAYLIST TEST: The Word (The Beatles : Rubber Soul)
ADDALBUMTOPLAYLIST TEST: Michelle (The Beatles : Rubber Soul)
ADDALBUMTOPLAYLIST TEST: What Goes On (The Beatles : Rubber Soul)
ADDALBUMTOPLAYLIST TEST: Girl (The Beatles : Rubber Soul)
ADDALBUMTOPLAYLIST TEST: I'm Looking Through You (The Beatles : Rubber Soul)
ADDALBUMTOPLAYLIST TEST: In My Life (The Beatles : Rubber Soul)
ADDALBUMTOPLAYLIST TEST: Wait (The Beatles : Rubber Soul)
ADDALBUMTOPLAYLIST TEST: If I Needed Someone (The Beatles : Rubber Soul)
ADDALBUMTOPLAYLIST TEST: Run for Your Life (The Beatles : Rubber Soul)
```

When evaluating the design of this button a potential issue was highlight which was if a user clicks add album when a playlist loaded it will not add anything as the playlist is not an album. In the future implementation of even an error message to say that it is "unable to add this as it is a playlist" may increase the usability of the program.

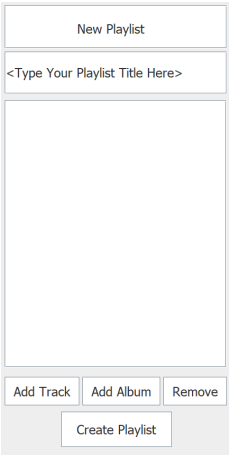
5.4.3 Remove Track

```java
public void removeTrackFromPlaylist(){

    //Method for the "remove" button in the MakePlaylist window.
    //Takes selected track's index and removes this index number from the
    //model.
    DefaultListModel model = (DefaultListModel) MakePlaylist.NewPlaylistTracks.getModel();
    int selectedTrackIndex = MakePlaylist.NewPlaylistTracks.getSelectedIndex();
    if (selectedTrackIndex != -1) {
        model.remove(selectedTrackIndex);
    }
}
```

A new model was created using the current JList model in the MakeNewPlaylist frame. The new model is edited by taking the currently selected element's index of the original model and removed this index from the new model. Testing of this button was carried out in GUI.

5.4.4 Save Playlist

Changed the look of the GUI to add a create playlist button. There is also a prompt to enter playlist title. This is to be used in the savePlaylist() code.



This method creates a new file using the playlist name typed by the user. It then loops through the playlist jList model and converts each element to another line of String using the toString method.

```
public void savePlaylist() throws IOException{

    //Method to save the playlist that has been created to
    //"text files/Playlists/"

    //There is a prompt to type filename into jTextField1
    String filename = MakePlaylist.jTextField1.getText();
    //This is used as the new playlist filename
    File newPlaylist = new File("text files/Playlists/" + filename + ".txt");
    ListModel model = playlistModel;

    PrintWriter pw = new PrintWriter(new OutputStreamWriter(new FileOutputStream(newPlaylist)));

    //Printing the playlistModel to the text file as String
    try{
        int len = model.getSize();
        for (int i = 0; i < len; i++){
            pw.println(model.getElementAt(i).toString());
        }
    }finally{
        pw.close();
    }
}
```

Used GUI to test and IOException was thrown when no playlist title existed. Decided to surround method with a try-catch in order to catch the exception.

```
try {
    playlist.savePlaylist();
}catch (IOException ex) {
}
```

Another thing to consider was the fact that there was no confirmation that a playlist was saved. The window also stayed open so you could click save multiple times. I decided to add a pop-up to confirm playlist was saved and disposed the MakePlaylist window.

```
try {
    playlist.savePlaylist();
}catch (IOException ex) {
}
this.dispose();
JOptionPane.showMessageDialog(null,"Playlist saved!");
```

5.5 Audio Function Implementation

The MP3Player class was created to store the play, pause and stop methods. Fileinputstream and bufferedinputstream defined as class variables.

```
public class MP3Player {

    private String filename;
    private Player player;

    FileInputStream fis;
    BufferedInputStream bis;
```

5.5.1 Play Method

This method was created to use a given filename as the fileinputstream and to define the total length of a track. I also added an output to the catch to highlight the issues when trying to load a specific mp3.

```
public void play(String filename) {
    try {
        fis = new FileInputStream(filename);
        bis = new BufferedInputStream(fis);
        player = new Player(bis);
        trackTotalLength = fis.available();
        location = filename + "";
    }

    catch (FileNotFoundException | JavaLayerException e) {
        System.out.println("Problem in playing: " + filename);
        System.out.println(e);
    }

    catch (IOException ex) {
    }
```

5.5.2 Pause Method

Pause method to stop the player and define a pausepoint.

```
public void pause() {
    if (player != null)

        {
            try{

                pausePoint = fis.available();
                player.close();

                }
            catch(IOException e){
                                }
        }
}
```

subsubsectionResume method Using the total track length defined in the play method and the pause point defined in the pause method, can “start” a player at the total length minus the pause point to resume from where it left off.

```
public void resume() {
    try {
        fis = new FileInputStream(location);
        fis.skip(trackTotalLength - pausePoint); //moved this above bis (track was playing a snippet of beginning, then continuing to skip)
        bis = new BufferedInputStream(fis);
        player = new Player(bis);

        }

    catch (FileNotFoundException | JavaLayerException e) {
        System.out.println("Problem in resuming song");
        System.out.println(e);
        }

    catch (IOException ex) {
    }

    // creata a thread to play music in background
        new Thread() {

        @Override
        public void run() {
            try { player.play(); }
            catch (JavaLayerException e) { System.out.println(e); }
        }
        }.start();
}
```

5.5.3 Stop Method

Stops the current player and sets track length and pause point to 0.

```
public void stop() {
    if (player != null)
        player.close();

        pausePoint = 0;
        trackTotalLength = 0;
        MusicPlayer.SongNameDisplay.setText("");
    }
```

All of the above methods were assigned buttons on the GUI.

5.5.4 Choose song button

Wrote code in the MusicPlayer to add a choose song button and hardcoded it into the directory of one of the provided albums for testing. The original method did not include the mp3.stop() on line 520, this caused multiple players to play over one another. Adding this line effectively cleared the player before starting a new file input stream.

```
private void ChooseSongButtonActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser openFC = new JFileChooser("C:\\Users\\Holly Birch\\Documents\\NetBeansProjects\\AlbumInfoProgram\\mp3 files\\Pianochocolate_LilacMusic(1)");
    int returnVal = openFC.showOpenDialog(null);

    if(returnVal == JFileChooser.APPROVE_OPTION){
        mp3.stop();

        File mp3file = openFC.getSelectedFile();
        String song = mp3file + "";

        String name = openFC.getSelectedFile().getName();
        SongNameDisplay.setText(name);

        mp3.play(song);
    }
}
```

After this, moved on to selecting a song from the Song JList to play a specific MP3 file. In order to do this I needed to build a file path that would match the provided mp3 file exactly.

5.5.5 Playing an album song on click

General file path: mp3 files/<folderArtist>_<folderAlbum>/<fileArtist>_-_<fileTrackNumber>_-_<fileSong>.mp3

```
//FILETRACKNUMBER: Number determined by using index of selected
//value in jList
int listModelIndex = MusicPlayer.jListSongDisplay.getSelectedIndex();
int fileTrackNumber = listModelIndex+1;

//FILESONG: using index of selected value in jList to retrieve
//song name from the listmodel2tracks array.
Track selectedTrack = listmodel2tracks.get(listModelIndex);
String fileSong = selectedTrack.getSong();

//FILEARTIST: currentlySelectedAlbum defined in getAlbumTracks
//method. This is used to obtain artist name. Formatted
//appropriately
String fileArtist = currentlySelectedAlbum.getArtistName();
fileArtist = fileArtist.replaceAll("&","__");
fileArtist = fileArtist.replaceAll("'","");

//Defining the file name:
String fileName = fileArtist + " - " + String.format("%02d", fileTrackNumber) + " - " + fileSong + ".mp3";
fileName = fileName.replace(" ", "_");

//FOLDERARTIST: using currentlySelectedAlbum to obtain artist name.
//Format is different to FILEARTIST.
String folderArtist = currentlySelectedAlbum.getArtistName();
folderArtist = folderArtist.replaceAll("&","");
folderArtist = folderArtist.replaceAll("'","");
folderArtist = folderArtist.replaceAll(" ","");

//FOLDERALBUM: using currentlySelectedAlbum to obtain album title.
//Formatted to remove special characters.
String folderAlbum = currentlySelectedAlbum.getAlbumTitle().replaceAll("[^A-Za-z]", "");
folderAlbum = folderAlbum.replaceAll("&","");
folderAlbum = folderAlbum.replaceAll("'","");
```

Took the approach of digesting the file path into 5 components:

- Folder artist
- Folder album
- File artist
- File track number
- File song

All components were available to me as explained in the comments of the image. Tested the outputs using a clear System.out.println that clearly marked every component of the file path.
Encountered a few problems with the file paths and changed Pianochoclate_LilacMusic to Pianochoclate_Lilac and DickBurril_NeverLetGo changed to TheDickBurrills_NeverLetGo.

```
//Defining the folder name:
String folderName = folderArtist+"_"+folderAlbum;

//Defining the final path:
String path = "mp3 files/"+folderName+"/"+fileName;

//Testing
System.out.println("====ALBUM MODEL====");
System.out.println(listmodel2tracks);
System.out.println("======================");

System.out.println("====ALBUM SONG TESTING====");
System.out.println("FILEARTIST: " + fileArtist);
System.out.println("FILESONG: " + fileSong);
System.out.println("FOLDERARTIST: " + folderArtist);
System.out.println("FOLDERALBUM: " + folderAlbum);
System.out.println("TRACKNUMBER: " + fileTrackNumber);
System.out.println("=========================");
System.out.println("====ALBUM SONG FILE PATH====");
System.out.println(path);
System.out.println("============================");
```

Once I was happy that this would apply to all of the provided mp3 file material, I added the mp3.play() method and allowed it to run the constructed file path. The final method was tested in the GUI and I ensured this was working by listening for the correct song.

```
//Playing the path
mp3.play(path);
//Set display to show which song is playing
MusicPlayer.SongNameDisplay.setText("Now Playing: " + selectedTrack.getSong());
```

5.5.6 Playing a playlist song on click

Could not implement the 'play album song on click' method to playlist tracks. Firstly, the method for obtaining the track number in the other method used the selected element's index in the JListSongDisplay – fine for an album – but a playlist could potentially have songs added to it in a random order. As the playlist tracks already populated the album info box, I felt that it would be easier to obtain the artist and album components from the UI instead. The file track number was the difficult part – I needed to loop through the album collection and search the album headers until they matched a formatted "artist : album" value. I would then set the 'currentlySelectedAlbum' class variable to this album and populate an arraylist with it's tracks. Finally, I returned the index of the selected track from this arraylist (giving me the track number).

```
//FILESONG:
String [] playlistFileSong = MusicPlayer.jListSongDisplay.getSelectedValue().split(": ");
String [] playlistFileSong2 = playlistFileSong[1].split(" 00:");
String fileSong = playlistFileSong2[0];
//FILEARTIST:
String [] playlistFileAlbumAndArtist = MusicPlayer.AlbumInfoBox.getText().split(" : ");
String fileArtist = playlistFileAlbumAndArtist[0].replaceAll("&","__");
fileArtist = fileArtist.replaceAll("'","");
//FOLDERALBUM:
String folderAlbum = playlistFileAlbumAndArtist[1];
folderAlbum = folderAlbum.replaceAll(" ","");
//FOLDERARTIST:
String folderArtist = playlistFileAlbumAndArtist[0].replaceAll("&", "");
folderArtist = folderArtist.replaceAll(" ","");
folderArtist = folderArtist.replaceAll("'","");
//FILETRACKNUMBER:
playlistmodel2tracks.clear();
for (Album album : albumCollection.albums){
    if(album.getHeader().equals(playlistFileAlbumAndArtist[0] + " : " + playlistFileAlbumAndArtist[1])){
        currentlySelectedAlbum = album;
        for (Track track : album.getAlbumsongs()){
            playlistmodel2tracks.add(track.getSong());

        }
    }
}
int playlistFileTrackNumber = playlistmodel2tracks.indexOf(playlistFileSong2[0])+1;
```

Testing was carried out in the same manner as the play album song on click method:

```
//Defining folder name:
String folderName = folderArtist+"_"+folderAlbum;

//Defining file name:
String fileName = fileArtist + " - " + String.format("%02d", playlistFileTrackNumber) + " - " + fileSong + ".mp3";
fileName = fileName.replace(" ", "_");

//Defining the final path:
String path = "mp3 files/"+folderName+"/"+fileName;

//Testing
System.out.println("====PLAYLIST MODEL====");
System.out.println(playlistmodel2tracks);
System.out.println("======================");

System.out.println("====PLAYLIST SONG TESTING====");
System.out.println("FILEARTIST: " + fileArtist);
System.out.println("FILESONG: " + fileSong);
System.out.println("FOLDERARTIST: " + folderArtist);
System.out.println("FOLDERALBUM: " + folderAlbum);
System.out.println("TRACKNUMBER: " + playlistFileTrackNumber);
System.out.println("============================");
System.out.println("====PLAYLIST SONG FILE PATH====");
System.out.println(path);
System.out.println("==============================");
```

```
======================
====PLAYLIST SONG TESTING====
FILEARTIST: Lovira
FILESONG: Maio Maduro Maio
FOLDERARTIST: Lovira
FOLDERALBUM: Experiments
TRACKNUMBER: 3
============================
====PLAYLIST SONG FILE PATH====
mp3 files/Lovira_Experiments/Lovira_-_03_-_Maio_Maduro_Maio.mp3
==============================
```

6 Testing

6.1 Testing Plans

A high level of software quality is a very important aspect, this means ensuring the needs of the user are met by conforming to the requirements of the program. Another important part is testing the program to ensure that the program is free from deficiencies. In this project the software quality will be assessed by over-viewing the maintainability, flexibility, portability, re-usability, reliability, correctness and usability. Many fault handling strategies have been implemented throughout the project including the use of fault avoidance. When carrying out the analysis it was considered aspects which don't work as well and therefore these were ruled out of being included in the project. Throughout the design process it was considered where faults may occur and the design was logically structured in order to aim to avoid potential issues. Also throughout implementation the methodologies were constantly reviewed to consider where issues may arise. Testing will include fault detection execution, debugging of the program. Using testing strategies will help to both validate the program; ensuring that we are building the correct product and also verify that we are building the product correctly.

The objectives of the testing will be to ensure the program can load album collections correctly, also load playlists correctly, play and stop songs from these sources and make playlists from these sources. Within these functions will be various methods that will require testing.

6.1.1 Plan of Features to be Tested for Each Class

Duration:

- Ensure time is of format hh:mm:ss
- Make sure one duration can be added to another

Track:

- Make sure track returns both song name and duration
- Ensure correct duration returned with associated track

PlaylistTrack:

- Check that it is inheriting information from its superclass Track and can return song name and duration and album information

Album:

- Check to see that an album has a title and an artist

- Make sure an album contains the right tracks
- Make sure the total duration of the album can be calculated by adding the track durations

AlbumCollection:

- Return a collection of albums
- Search the collection for a specific album and return it's title, artist. Also return an album's songs and their durations.

Playlist:

- Return a specified set of playlist tracks and their albums and durations

MP3Player:

- Create a player from a specified file path
- Stop the currently running player
- Pause the currently running player
- Resume the currently running player

MusicPlayer:

- Check all buttons work on click
- Check files are loaded into correct location line by line
- Check the jLabel work to adequately resize the image to the box
- Check pulling all correct information from other classes
- Check program functioning as a whole

MakePlaylist:

- Check all buttons work on click
- Check tracks added to playlist in correct format
- Check writes to a correctly named txt file and populates the file correctly
- Check that the file can be reopened

The testing will be judged on whether it passes the task or fails it. An expected output will be defined if this is not met then the test will have failed, if it does meet it it will be considered a pass. All of this information will be stored in a test log. Reasons for failure will be stated, if a test does fail the code will be adapted in order to try and change this outcome to a pass, so that full functionality of the music player can be achieved.
After evaluating testing approaches a Bottom-up testing plan was selected due to it being useful for integrating object-oriented systems. This approach allows us to test classes from the bottom checking that all aspects that feed into the class above are working correctly before proceeding.

Whilst carrying out the testing it will be consider throughout whether or not the passing of each feature is detrimental to the programs function. This will be evaluated in depth if problems occur to decide on whether suspension of testing should take place in order to prioritise more important tasks. Resumption of testing can be recommenced when appropriate in the development process. Netbeans will be required throughout the testing stage with access to a computer that is able to use this software.

6.2 Test Cases and Test Logs

Test cases were put together in order to create the structure of the test logs. These show the full testing of each class that has been implemented. They outline the name of the class and the action taken to test the items specified in the plan. They action taken and the expected output is the same as the information input if it has passed if not the information that should have been output is logged in the reason for failure section. All test logs can be viewed below.

TEST LOG: Duration			
Purpose: To test the Duration class			
Run Number 1	Date 18/12/2018		
Action	Expected Output	Pass/Fail	Reason for failure
Create duration d1 (”02/34/50”)	02:34:50	P	
Create duration d2 and (d1.add(d2))	05:09:40	P	

Table 1:
Duration Test Log

Code implemented to test the duration class both format and adding of durations:

```
public static void main(String[] args) {
    Duration d1 = new Duration ("02/34/50");
    System.out.println("Duration Format" + d1);
    Duration d2 = new Duration ("02/34/50");
    d1.add(d2);

    System.out.println("Addition of Durations" + d1);
}

}
```

Expected output from code in the console log:

```
run:
Duration Format02:34:50
Addition of Durations05:09:40
BUILD SUCCESSFUL (total time: 0 seconds)
```

TEST LOG: Track			
Purpose: To test the Track class			
Run Number 1	Date 18/12/2018		
Action	Expected Output	Pass/Fail	Reason for failure
Create new duration object d1 (”02:03:04”) and track object t1 (”SongName” , d1)	SongName 02:03:04	P	
Check duration assigned (d1) matches out print from track duration	02:03:04 and SongName 02:03:04	P	

Table 2:
Track Test Log

Code to make sure that the track returns both song name and duration:

```
public static void main(String[] args) {
    Duration d1 = new Duration("02:03:04");
    Track t1 = new Track("SongName", d1);
    System.out.println("Printing a Track and it's Duration: " + t1);
}
```

Expected console log both the song name and duration are printed:

```
run:
Printing a Track and it's Duration: SongName 02:03:04
BUILD SUCCESSFUL (total time: 0 seconds)
```

Code to ensure that the correct duration is returned with it's associated track:

```
public static void main(String[] args) {
    Duration d1 = new Duration("02:03:04");
    Track t1 = new Track("SongName", d1);
    System.out.println(d1);
    System.out.println("Printing a Track and it's Duration: " + t1);
}
```

Expected console log is returned showing that the correct duration value is assigned to the track:

```
run:
02:03:04
Printing a Track and it's Duration: SongName 02:03:04
BUILD SUCCESSFUL (total time: 0 seconds)
```

TEST LOG: Album			
Purpose: To test the Album class			
Run Number 1	Date 18/12/2018		
Action	Expected Output	Pass/Fail	Reason for failure
Create new album called a1 containing tracks t1 and t2, which contain duration d1 and d2. Call addTrackToAlbum()	AlbumTitle : Artist SongName 02:03:04 SongName2 00:01:02	P	

Table 3:
Album Test Log

Code implemented to test return of album title, artist, tracks on the album and their durations:

```
public static void main(String[] args) {
    Duration d1 = new Duration("02:03:04");
    Duration d2 = new Duration("00:01:02");
    Track t1 = new Track("SongName", d1);
    Track t2 = new Track("SongName2", d2);
    Album a1 = new Album("AlbumTitle", "Artist");
    a1.addTrackToAlbum(t1);
    a1.addTrackToAlbum(t2);


    System.out.println("Printing an Album and it's Tracks with their durations: \n" + a1);
}
```

Console log shows that the input gives the expected output and therefore passes testing:

```
Printing an Album and it's Tracks with their durations:
Artist : AlbumTitle
SongName 02:03:04
SongName2 00:01:02
BUILD SUCCESSFUL (total time: 0 seconds)
```

TEST LOG: AlbumCollection			
Purpose: To test the AlbumCollection class			
Run Number 1	Date 18/12/2018		
Action	Expected Output	Pass/Fail	Reason for failure
Create new album collection called ac1 with two albums a1 and a2 each containing two tracks t1, t2 and t3, t4	Artist : AlbumTitle SongName 02:03:04 SongName2 00:01:02 Artist2 : AlbumTitle2 SongName3 00:03:04 SongName3 00:03:45	P	
Return selected album. Created new album called acfind and called findAlbum() method on ac1 to define acfind	AlbumCollection for AlbumTitle2 Artist2 : AlbumTitle2 SongName3 00:03:04 SongName3 00:03:45	P	

Table 4:
AlbumCollection Test Log

Code implemented to test return of album collections including album title, artist, tracks on the album and their durations:

```
public static void main(String[] args) {
    Duration d1 = new Duration("02:03:04");
    Duration d2 = new Duration("00:01:02");
    Duration d3 = new Duration("00:03:04");
    Duration d4 = new Duration("00:03:45");

    Track t1 = new Track("SongName", d1);
    Track t2 = new Track("SongName2", d2);
    Track t3 = new Track("SongName3", d3);
    Track t4 = new Track("SongName3", d4);

    Album a1 = new Album("AlbumTitle", "Artist");
    a1.addTrackToAlbum(t1);
    a1.addTrackToAlbum(t2);
    Album a2 = new Album("AlbumTitle2", "Artist2");
    a2.addTrackToAlbum(t3);
    a2.addTrackToAlbum(t4);

    AlbumCollection ac1 = new AlbumCollection();
    ac1.addAlbum(a1);
    ac1.addAlbum(a2);

    System.out.println("Printing an AlbumCollection and it's Albums with their Tracks and their durations: \n" + ac1);

}
```

Console log shows the expected output is returned:

```
Printing an AlbumCollection and it's Albums with their Tracks and their durations:
Artist : AlbumTitle
SongName 02:03:04
SongName2 00:01:02
Artist2 : AlbumTitle2
SongName3 00:03:04
SongName3 00:03:45
BUILD SUCCESSFUL (total time: 0 seconds)
```

Code implemented to test returning a selected album and the expected information:

```
public static void main(String[] args) {
    Duration d1 = new Duration("02:03:04");
    Duration d2 = new Duration("00:01:02");
    Duration d3 = new Duration("00:03:04");
    Duration d4 = new Duration("00:03:45");

    Track t1 = new Track("SongName", d1);
    Track t2 = new Track("SongName2", d2);
    Track t3 = new Track("SongName3", d3);
    Track t4 = new Track("SongName3", d4);

    Album a1 = new Album("AlbumTitle", "Artist");
    a1.addTrackToAlbum(t1);
    a1.addTrackToAlbum(t2);
    Album a2 = new Album("AlbumTitle2", "Artist2");
    a2.addTrackToAlbum(t3);
    a2.addTrackToAlbum(t4);

    AlbumCollection ac1 = new AlbumCollection();
    ac1.addAlbum(a1);
    ac1.addAlbum(a2);

    Album acfind = ac1.findAlbum("Artist2", "AlbumTitle2");

    System.out.println("Searching AlbumCollection for AlbumTitle2 and displaying it's Tracks and their durations: \n" + acfind);

}
```

Console log shows the expected return of the album chosen including album title, artist name, song names and durations:

```
run:
Searching AlbumCollection for AlbumTitle2 and displaying it's Tracks and their durations:
Artist2 : AlbumTitle2
SongName3 00:03:04
SongName3 00:03:45
BUILD SUCCESSFUL (total time: 0 seconds)
```

TEST LOG: PlaylistTrack			
Purpose: To test the PlaylistTrack class			
Run Number 1	Date 18/12/2018		
Action	Expected Output	Pass/Fail	Reason for failure
Created track t1 and added to album a1 created playlist track p1 using t1.getSong() and t1.getDuration() and a1	SongName : Artist : AlbumTitle	P	

Table 5:
PlaylistTrack Test Log

Code used to get the album, track and duration information:

```
public static void main(String[] args) {
    Duration d1 = new Duration("02:03:04");
    Duration d2 = new Duration("00:01:02");
    Duration d3 = new Duration("00:03:04");
    Duration d4 = new Duration("00:03:45");

    Track t1 = new Track("SongName", d1);
    Track t2 = new Track("SongName2", d2);
    Track t3 = new Track("SongName3", d3);
    Track t4 = new Track("SongName3", d4);

    Album a1 = new Album("AlbumTitle", "Artist");
    a1.addTrackToAlbum(t1);
    a1.addTrackToAlbum(t2);
    Album a2 = new Album("AlbumTitle2", "Artist2");
    a2.addTrackToAlbum(t3);
    a2.addTrackToAlbum(t4);

    AlbumCollection ac1 = new AlbumCollection();
    ac1.addAlbum(a1);
    ac1.addAlbum(a2);

    PlaylistTrack p1 = new PlaylistTrack(t1.getSong(),t1.getDuration(),a1);

    System.out.println("Printing new playlist track made from t1: \n" + p1);

}
```

Console log showing the expected results and passing testing:

```
run:
Printing new playlist track made from t1:
SongName : Artist : AlbumTitle
BUILD SUCCESSFUL (total time: 0 seconds)
```

TEST LOG: Playlist			
Purpose: To test the Playlist class			
Run Number 1	Date 18/12/2018		
Action	Expected Output	Pass/Fail	Reason for failure
Created playlist called playlist, using ac. Created two playlist tracks p1(t1, a1) and p2(t3, a2). Added p1 and p2 to playlist.	SongName : Artist : AlbumTitle SongName3 : Artist2 : AlbumTitle2	P	

Table 6:
Playlist Test Log

Code used to create playlist and add songs:

```
public static void main(String[] args) {
    Duration d1 = new Duration("02:03:04");
    Duration d2 = new Duration("00:01:02");
    Duration d3 = new Duration("00:03:04");
    Duration d4 = new Duration("00:03:45");

    Track t1 = new Track("SongName", d1);
    Track t2 = new Track("SongName2", d2);
    Track t3 = new Track("SongName3", d3);
    Track t4 = new Track("SongName3", d4);

    Album a1 = new Album("AlbumTitle", "Artist");
    a1.addTrackToAlbum(t1);
    a1.addTrackToAlbum(t2);
    Album a2 = new Album("AlbumTitle2", "Artist2");
    a2.addTrackToAlbum(t3);
    a2.addTrackToAlbum(t4);

    AlbumCollection ac1 = new AlbumCollection();
    ac1.addAlbum(a1);
    ac1.addAlbum(a2);

    PlaylistTrack p1 = new PlaylistTrack(t1.getSong(),t1.getDuration(),a1);
    PlaylistTrack p2 = new PlaylistTrack(t3.getSong(),t3.getDuration(),a2);

    Playlist playlist = new Playlist(ac1);
    playlist.addPlaylistTrack(p1);
    playlist.addPlaylistTrack(p2);

    System.out.println("Printing playlist containing two tracks: \n" + playlist);

}
```

Console log showing the songs addded to the created playlist:

```
run:
Printing playlist containing two tracks:
SongName : Artist : AlbumTitle
SongName3 : Artist2 : AlbumTitle2

BUILD SUCCESSFUL (total time: 0 seconds)
```

TEST LOG: MP3Player			
Purpose: To test the MP3Player class			
Run Number 1	Date 22/12/2018		
Action	Expected Output	Pass/Fail	Reason for failure
Create String filepath to Pianochocolate : Lilac Music 0:04:58 - More Than the Ocean. MP3.play(filepath)	Can hear the selected track playing	P	
Run MP3.play(filepath), TimeUnit.seconds.sleep(10) then run MP3.stop()	Can hear selected track playing for ten seconds, then should stop	P	
Run MP3.play(filepath), TimeUnit.seconds.sleep(10) then run MP3.pause() TimeUnit.seconds.sleep(10) Run MP3.resume	Can hear selected track playing for ten seconds then pauses for ten seconds and resumes from the paused point	P	

Table 7:
MP3Player Test Log

Code showing the hard coding of a filepath to play an MP3 track:

```
public static void main(String[] args) {
    String filepath = "C:\\Users\\Holly Birch\\Documents\\NetBeansProjects\\AlbumInfoProgram\\mp3 files\\Pianochocolate_LilacMusic(1)\\Pianochocolate_-_01_-_More_Than_the_Ocean.mp3";
    MP3Player mp3 = new MP3Player();
    mp3.play(filepath);

}
```

Code showing playing and then stopping after ten seconds has passed:

```
public static void main(String[] args) {
    String filepath = "C:\\Users\\Holly Birch\\Documents\\NetBeansProjects\\AlbumInfoProgram\\mp3 files\\Pianochocolate_LilacMusic(1)\\Pianochocolate_-_01_-_More_Than_the_Ocean.mp3";
    MP3Player mp3 = new MP3Player();
    mp3.play(filepath);
    try {
        TimeUnit.SECONDS.sleep(10);
    } catch (InterruptedException ex) {
        System.out.println("Error with waiting");
    }
    try{mp3.stop();
    }catch(Exception e){
    }

    }
```

Code demonstrating the music player pausing for ten seconds and then resuming at the pause point:

```
public static void main(String[] args) {
    String filepath = "C:\\Users\\Holly Birch\\Documents\\NetBeansProjects\\AlbumInfoProgram\\mp3 files\\Pianochocolate_LilacMusic(1)\\Pianochocolate_-_01_-_More_Than_the_Ocean.mp3";
    MP3Player mp3 = new MP3Player();
    mp3.play(filepath);
    try {
        TimeUnit.SECONDS.sleep(10);
    } catch (InterruptedException ex) {
        System.out.println("Error with waiting");
    }
    try{mp3.pause();
    }catch(Exception e){
    }
    try {
        TimeUnit.SECONDS.sleep(10);
    } catch (InterruptedException ex) {
        System.out.println("Error with waiting");
    }
    try{mp3.resume();
    }catch(Exception e){
    }
    }
```

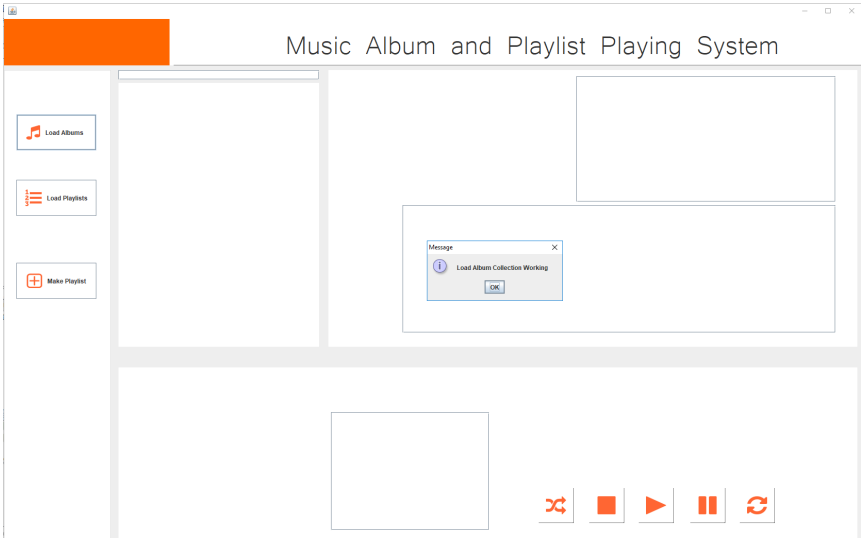
TEST LOG: MusicPlayer			
Purpose: To test the MusicPlayer class			
Run Number 1	Date 05/01/2019		
Action	Expected Output	Pass/Fail	Reason for failure
Create pop up to confirm when each button in the JFrame is clicked	Pop up is shown everytime a button is clicked	P	
Load AlbumCollections from albums.txt. Select an album from the JList	Album Collections populated and selected album's track information displayed correctly	P	
Get JLabel dimensions and set image to fit the dimensions	Image fits JLabel	P	

Table 8:
MusicPlayer Test Log

This code was implemented to give a response upon click of the Load Albums button to show that the event was being registered:

```
    private void LoadAlbumCollectionButtonActionPerformed(java.awt.event.ActionEvent evt) {
        JOptionPane.showMessageDialog(null,"Load Album Collection Working");
    }
```

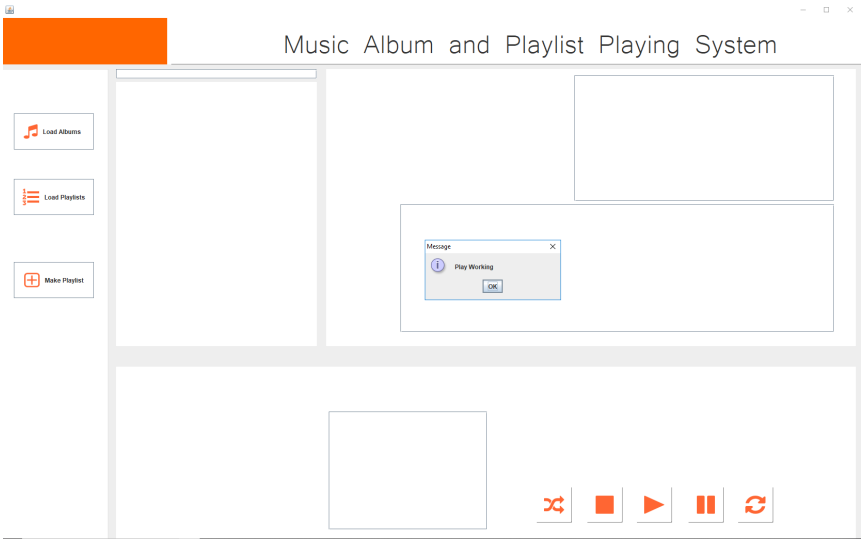
Output showing that a response is registered when the button is clicked:



This code also is added to show that the play button is functioning as expected:

```
private void PlayButtonActionPerformed(java.awt.event.ActionEvent evt) {
    JOptionPane.showMessageDialog(null,"Play Working");
}
```

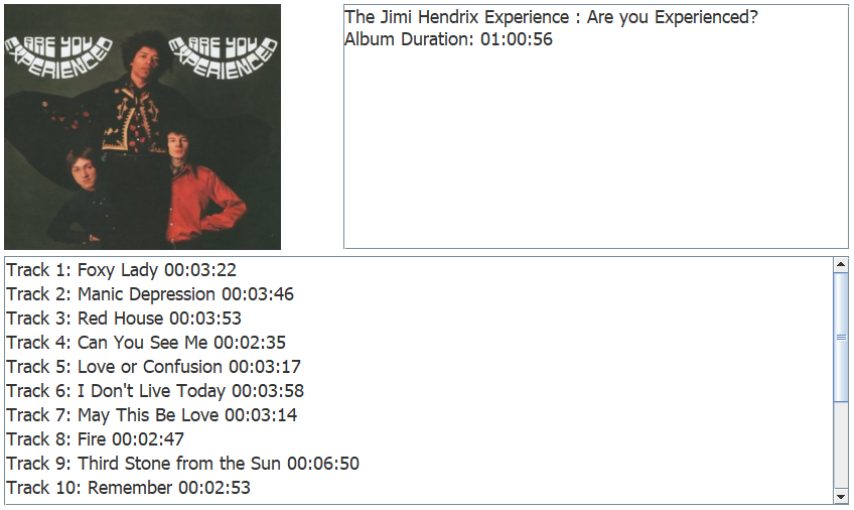
Expected output, pop up shows that the button clicked has been registered:



The Jimmy Hendrix Are You Experienced image has dimensions of 425 x 425 whereas the JLabel dimensions are 300 x 300. The code shows how the implementation of resizing the image to fit the JLabel was executed.

```
ImageIcon imgPath = new javax.swing.ImageIcon(javax.swing.JLabel.class.getResource("/albuminfoprogram/newpackage/"+returnedAlbum));
Image img1 = imgPath.getImage();
Image img2 = img1.getScaledInstance(MusicPlayer.jLabel1.getWidth(), MusicPlayer.jLabel1.getHeight(), Image.SCALE_SMOOTH);
ImageIcon scaledImage = new ImageIcon (img2);
MusicPlayer.jLabel1.setIcon(scaledImage);//new javax.swing.ImageIcon(javax.swing.JLabel.class.getResource("/albuminfoprogram/newpackage/"+returnedAlbum))); // NOI18N
MusicPlayer.jLabel2.setIcon(scaledImage);
```

This image shows that the image has resized to fill the whole JLabel as expected so therefore the output required is shown:

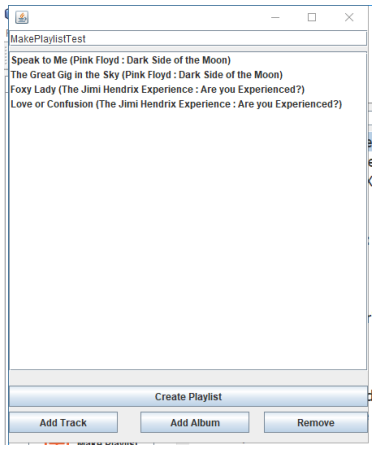


TEST LOG: MakePlaylist			
Purpose: To test the MakePlaylist class			
Run Number 1	Date 08/01/2019		
Action	Expected Output	Pass/Fail	Reason for failure
Clicking of add track	Track is added to the MakePlaylist JList	P	
Add four tracks to playlist and name it makeplaylisttest and click create playlist	Text file called makeplaylisttest.txt produced with tracks and album title and artist of four tracks	P	
Open text file and play track	Text file opens containing information and can play a track	F	Album collection not loaded first. When album collection loaded the playlist works

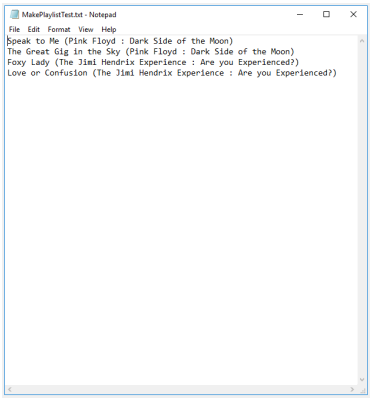
Table 9:
MakePlaylist Test Log

Evidence of clicking add track and the tracks moving accross into the make playlist JFrame:

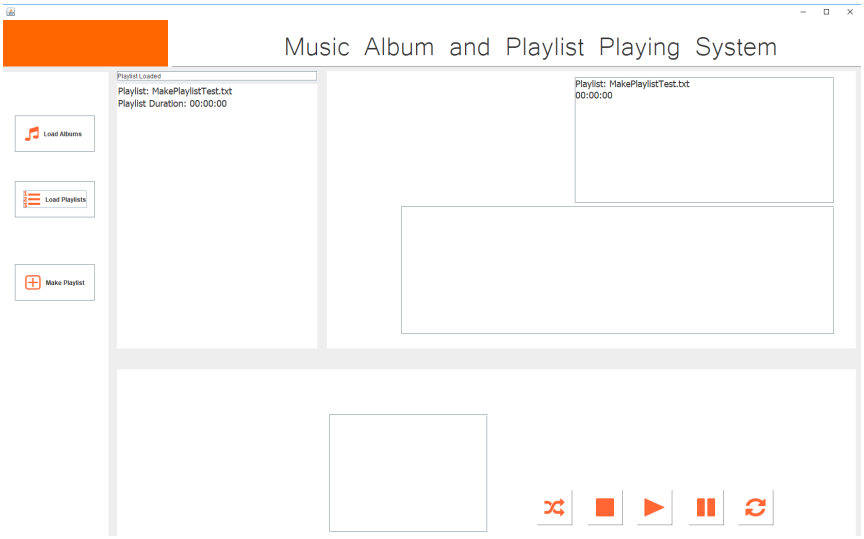
Showing the tracks that were added to the playlist and name that was given:



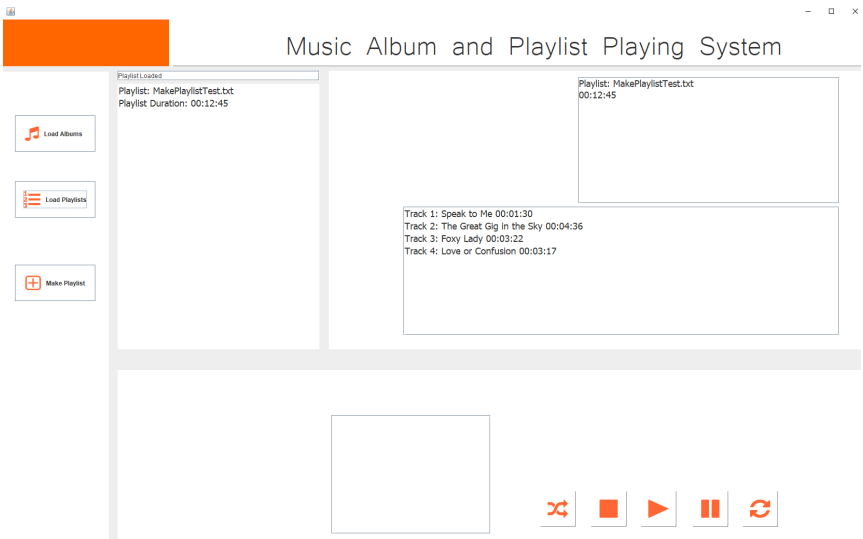
Evidence that the playlist that was created generated and saved a text file :



The image displaying the outcome of loading a playlist when the relevant album collection has not been loaded. This aspect of loading the playlist failed testing however understanding of why this happened has been recorded.



This is an image displaying the outcome of the working playlist after the relevant album collection has been loaded.



Through copious testing throughout the implementation of the code it is possible to confirm exactly which aspects of the program are working. Production of the test logs for each class has been essential in the tracking of the program testing. The main issue highlighted from the program testing was that the relevant album collection needed to be loaded before playing of a chosen playlist. This is due to the information being stored in the album collection file and used for the playlists. Although it was not possible ot fix this issue

due to time constraints it is something that has been acknowledged and could be improve in future developments. The program has also been launched on various workspaces by different users, using just the JAR file and following the user manaual provided in Appendix 9.2. All attempts to use the music player have been successful with positive feedback and therefore I am happy to conclude that the program performs correctly.

7 Evaluation and Discussion

Overall this project has been successful with the production of a functioning music player that meets user requirements. The music player enables users to be able to play the mp3 of their available album collections and playlists viewing the artist name, album name, track information, duration and total duration with the appropriate cover image. It also allows users to create and name their own playlists with both the option to add and remove tracks. A thorough approach was taken when designing the program and has developed along with the project.

The music player is aesthetically appealing due to its simple contrasting design. Use of logos with a splash of colour for the buttons works well and their functionality is clear to the end users. The layout design of the music player works well in terms of functionality as it works in a logical order for example once the user has clicked on an album the population of the albums information is displayed, this is the order of events that you would expect to occur. Although the design works well in terms of functionality and appealing to the target market it is important to consider areas for improvement. The main issue highlighted with the music player is the fact users have to load album collections prior to the playlists. If the album collection is not loaded before the playlist the playlist would be unable to load due to the album collection files containing further information that is required such as the duration of the tracks. A warning was implemented to instruct the user to load the album collection before the playlist if no album collection is loaded in the background. This bug is not completely fixed and the program would have improved functionality if the player was able to find this information from the stored files or a slight improvement would be for the program to be able to check exactly if the album collections information had been loaded for the songs contained in the playlist the user was trying to load.

Another area to consider if future development was to take place is the option to allowing the user to load all of their playlists into the left hand JList so that they could go through and view each one individually the same way that they are able to with the album collections. However, in terms of functionality all of the information about the playlist that is required for the user is displayed and accessible to the user.

Furthermore, consideration of aspects such as having plus signs next to the tracks to add them to the playlist instead of an add button may be appealing to the younger target market also once the track has been added putting the option of a minus button next to

the track instead of clicking remove may be more appealing to the eye. Implementation of a play logo next to the track so that when a user hovers over the track it is visible may also be something that looks more appealing to users. This is also a similar scenario when considering click of the play button, something other companies that have produced music players have is the change of this button to a pause button. This ultimately makes the audio bar look a bit tidier as not so many buttons are present at first sight of the music player. All of these aspects are ideas as to how the future design aspects could be improved to make the music player a more innovative and appealing program.

The GUI design was constructed using two JFrames however when evaluating this design aspect, it was considered that the program may have improved functionality if everything was to work within one JFrame. This is because the use of two JFrames means that one overlaps the other. As tracks are being clicked from the main JFrame it was important that the make playlist JFrame always stayed on top of the other frame otherwise the user would have to keep clicking open the other frame. Although this is an acceptable solution and works well in terms of functionality if time permitting combining all aspects may be the best option in terms of maximising functionality.

Due to time limitations I was unable to implement the shuffle button. This would have been an extra function for the music player that could be implemented in the future based on the user's requirements. A random generator could be implement to generate a number which could be used to coincide with a track number enabling a random track to be played from either an album collection or a playlist. Other aspects that would have been a plus to implement is the duration sliding bar shown in the design concepts. This was initially put into the design concept as it is something that could be considered very useful in terms of allowing users to see how far through a song they are. This information could help to aid their decision on whether they would like to continue listening to the song for example because they are nearly at the end or whether they want to change the track because there is still two minutes remaining. Although this would have been a nice function to implement it was not considered a necessary aspect for the target audience. Future developments of the music player could include features such as this that could help to enhance the users experience of using the music player. Next and previous track are also extra functions that enable users to quickly skip over tracks. This is a good function to have as it means that a user does not have to spend lots of time interaction with the music player if they do not want to, however this feature was also seen as something that could be a future improvement. Lastly one other feature that could be something desired by the target audience is the ability to change the volume level in the program. This was something not implemented initially as it is possible to alter the volume level using the functions on your computer. However, having the option to do everything within the program would be advantageous which is why this is also another function that should definitely be considered for future development of this music player.

Evaluating the design produced has enabled an in depth analysis of the final product

to take place. Many points have been discussed in terms of improvements, these are not considered limitations of the product merely possible future improvements. Critical analysis of all aspects enabled development of design concepts in order to progress the product further. If this project was to be reviewed aspects that have been discussed would be put forward and analysed further to determine their level of requirement for the music player.

8 Summary

To conclude this is considered a very successful project, the music player produced has been well designed and meets all of the music players requirements that were outlined at the start of the project. Additional to The Sound Companies expectations it also fulfils some of the additional requirements that were to be considered if time constraints allowed. A main contributor to the success of the project is the upfront in-depth design process that was used to identify the path the project took. The program was adapted throughout the development with the target market in mind. Most issues that were highlighted as potential downfalls of the program were eliminated and the few that remained were discussed in the discussion. One of the main issues was certainly having to load an album collection prior to the use of a playlist however this could be altered if future improvements were to be made. Thorough testing of the program has been undertaken throughout the development of the music player with logs kept to track the progress. More importantly testing of the final JAR file has been the final evidence that proves that a fully functional music player has been produced. The release of the music player should be a successful launch that is able to compete with the likes of companies such as Spotify.

9 Appendix

9.1 UML Class Diagram with Further Detail

Figure 102: Described UML Diagram

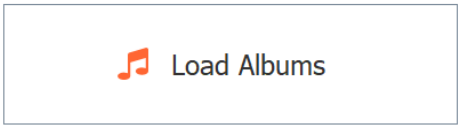
9.2 User Manual

Create a folder named MusicPlayer and store it on the C drive. In this folder place another folder called album images where all of the album images should be stored. Another folder called mp3 files that has all of the audio folders stored inside. Lastly a folder called text files that has two folder insider one labelled AlbumCollections which stores all of the album collection text files and the other called Playlists which stores all of the playlist text files, the AlbumImageList text file should be stored in the main level of this folder.

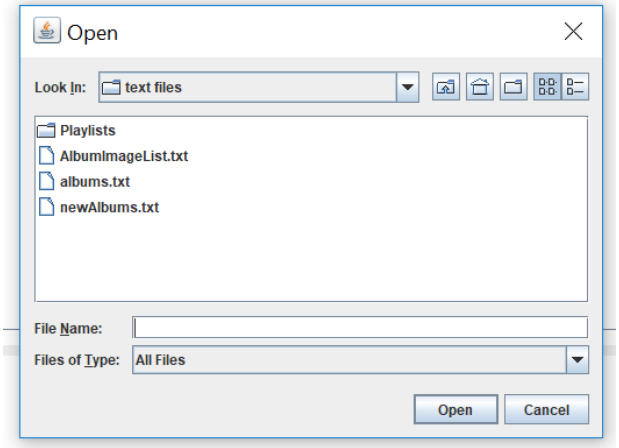
Unzip the MusicPlayer folder and launch the JAR file by double clicking on it.

9.2.1 Load Album Collections Instructions

1. Launch the JAR file provided
2. Click Load Albums using the button shown below

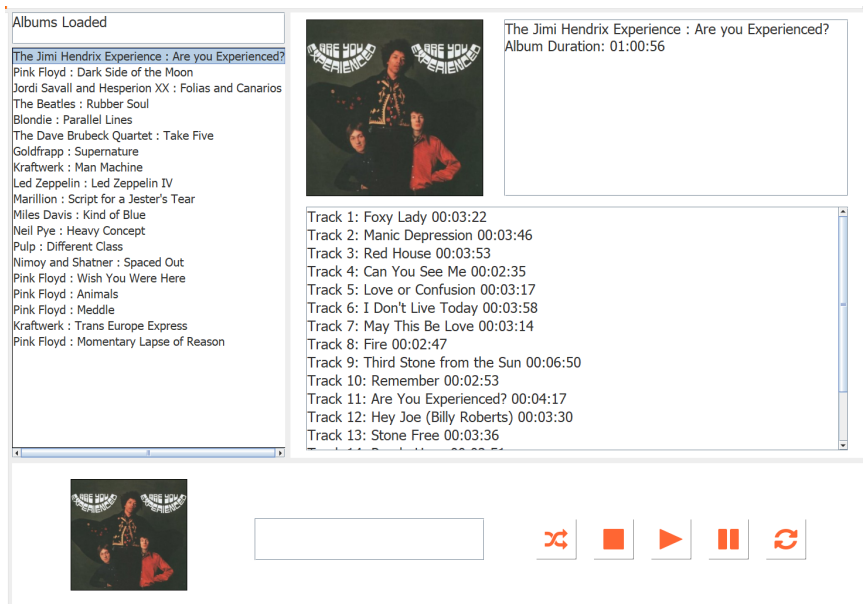


3. Select the file that you would like to load using the file chooser shown below



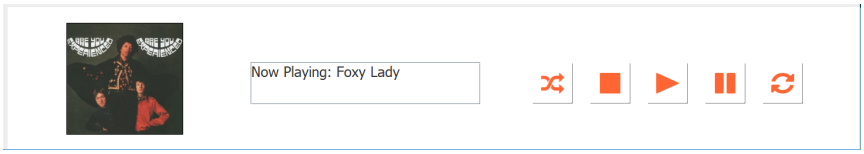
4. Once opened the album titles and artist names should populate in the box located to the left

5. Choose the album that you would like to listen to and select this, the relevant information should populate on the right



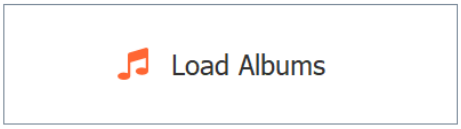
6. Select the track you would like to play

7. Use the audio functions to play, pause, stop and loop your track

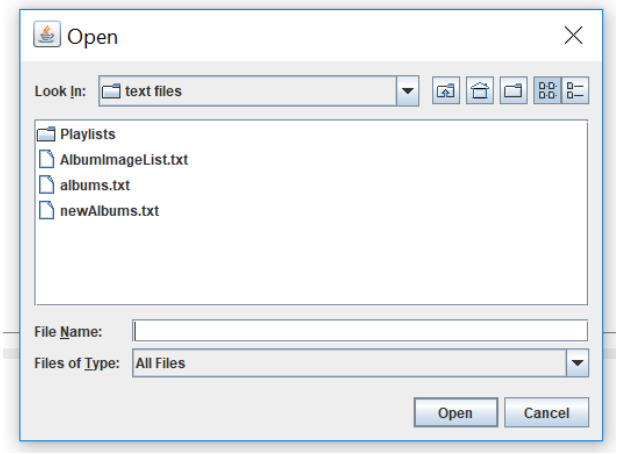


9.2.2 Play Playlist Instructions

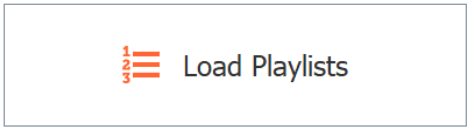
1. Launch the JAR file provided
2. Click Load Albums using the button shown below



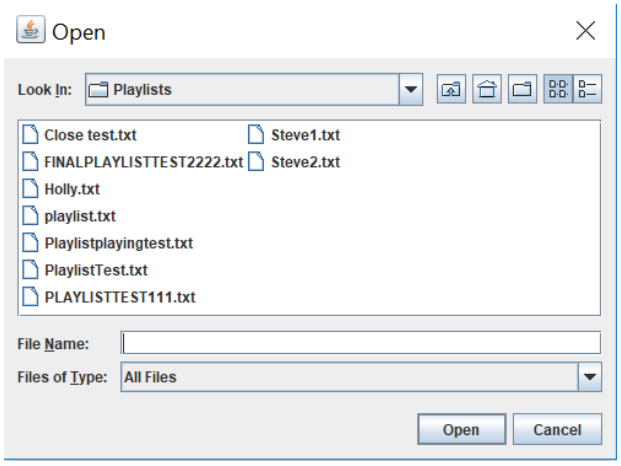
3. Select the file to load the albums that are in the playlist



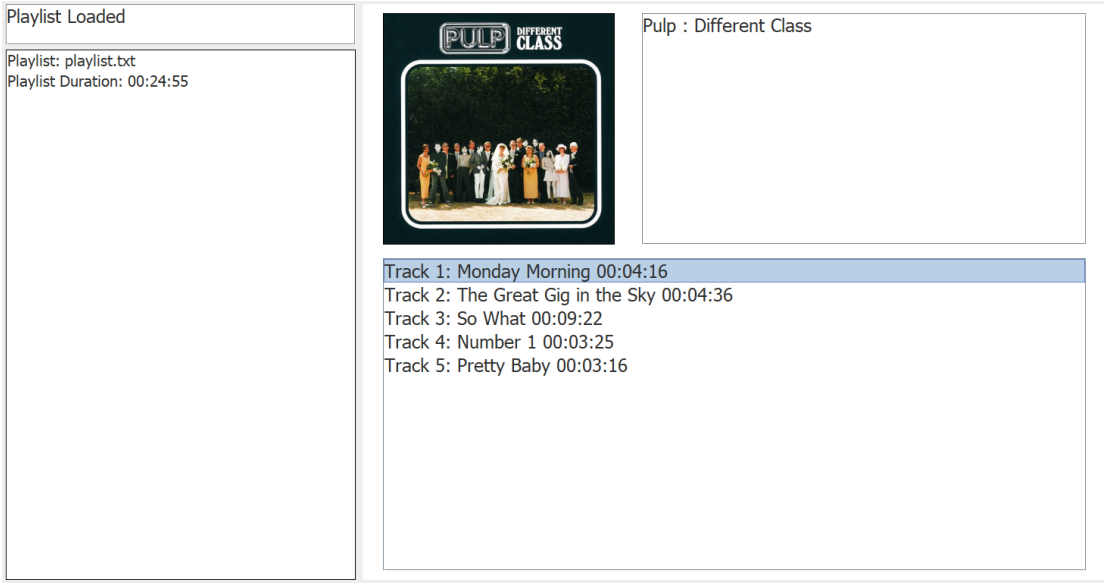
4. Click Load Playlist using the button shown below



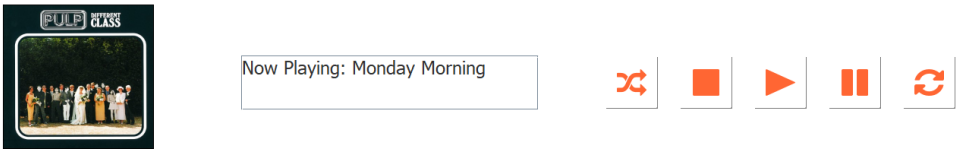
5. Select the file to load the playlist that you would like to listen too



6. Track information will be displayed select the track that you would like to play



7. Use the audio functions to play, pause, stop and loop your track

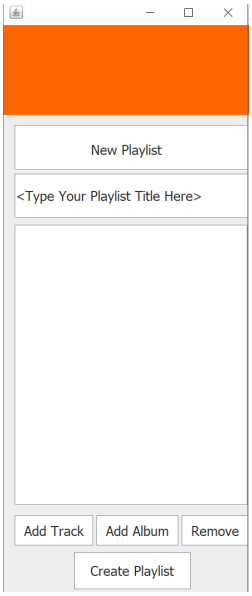


9.2.3 Create Playlist Instructions

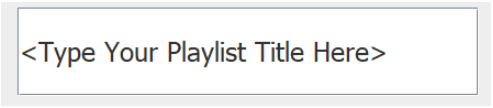
1. Launch the JAR file provided

2. After loading any of the album collections or playlists that you would like to use to create your playlist
3. Click Make Playlist

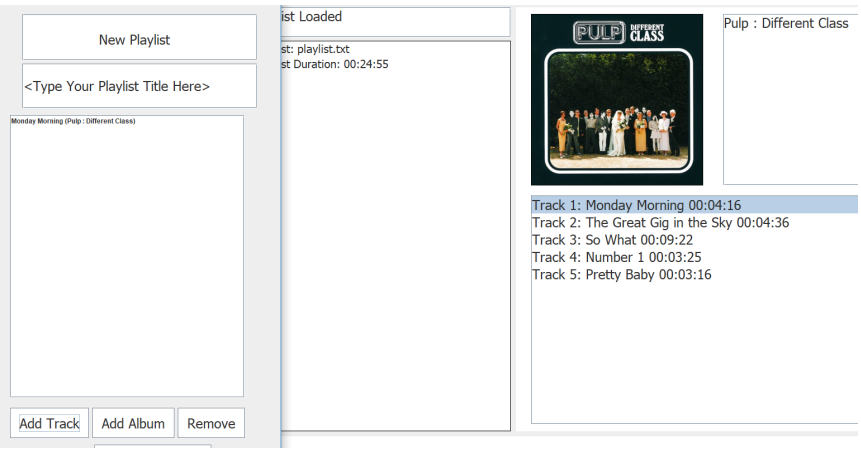
4. a new panel will open



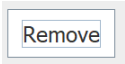
5. Type in the name that you would like to call your playlist into the box shown below



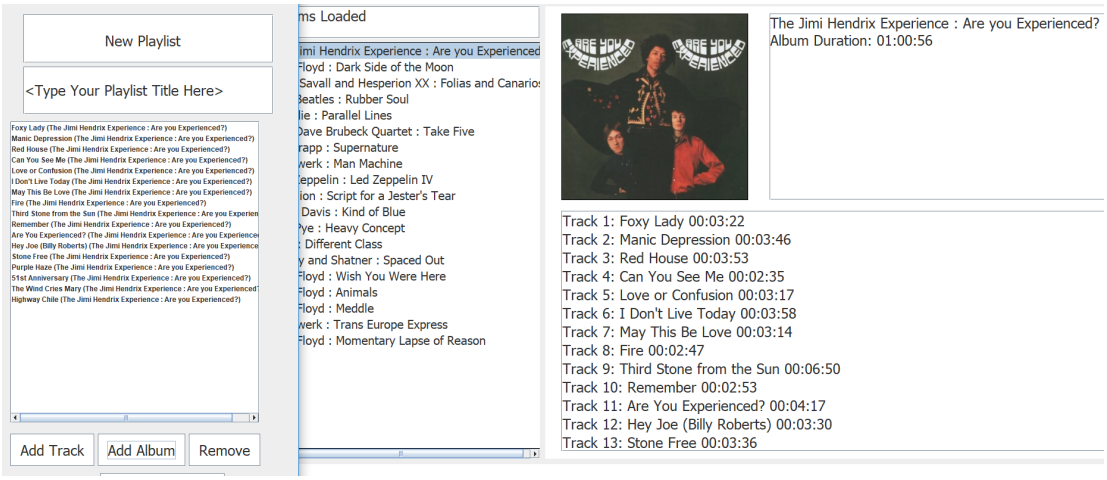
6. Click on the tracks you would like to add to the playlist shown on the main panel and use the add button to move the tracks over to your new playlist



7. If you wish to remove a track, select the track and click remove



8. If you want to add a whole album, select the album and click add album



9. Once you have finished adding all of the tracks you require click the create playlist button and this will generate your playlist