Student ID goes here

Applications Programming: Coursework 2

Introduction

This report outlines the design, programming, and testing process for the creation of a Music Application and Playlist Playing System (MAPPS). This program was coded in Java using the NetBeans IDE, and is designed to run on any system which has the capability to run JAR files.

Included in this report is an analysis of the design process including UML diagrams, a description of the implementation of the project, testing and debugging, and an evaluation. At the end of the report is a comprehensive user manual.

Object Oriented Analysis

The purpose of this system is to play individual tracks from either an album or a playlist. The MP3 files are stored in a data file, outside of the application. The application merely points to this folder.

The class requirements for this project were specified in two parts, the first laying out the basic classes required and the second requiring a GUI. Because of this structure, the design process has closely matched the modified waterfall model.

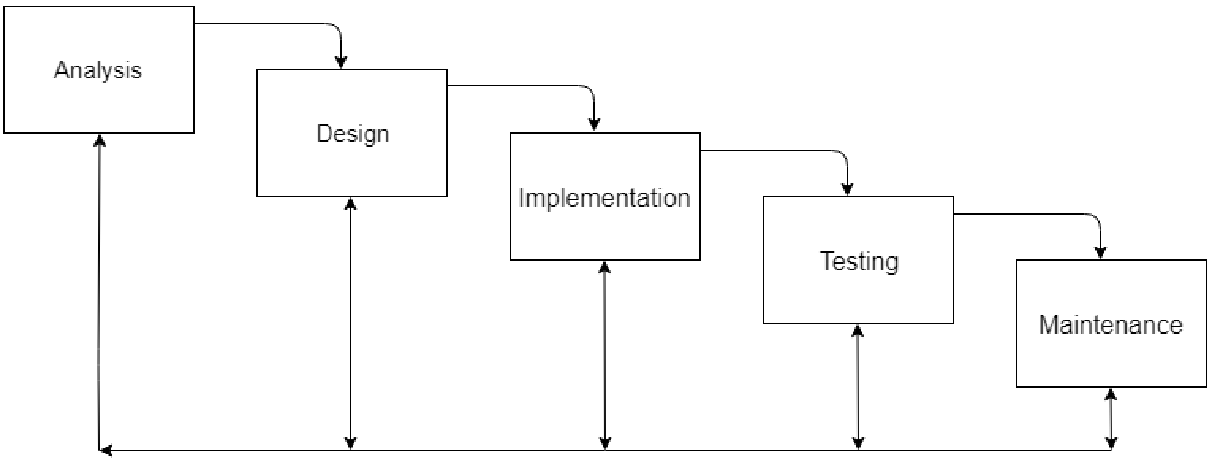


Figure 1: Modified Waterfall Model

The rationale behind this methodology is that, whilst a programmer attempts to work from analysis through to maintenance in order, each step can be returned to at any point when further requirements are discovered. In this case, the original requirements were to create an application which could read in an album collection, sort and print it, and perform various functions including calculating the total duration of all the albums, finding the longest albums and longest tracks. This level of functionality was designed, implemented, and tested.

Before beginning the second set of requirements, there was first some editing of the first version of the system based on feedback. In particular, edits were made to ensure all methods were contained in the correct class, and avoiding some security flaws regarding exposing ArrayLists.

Before describing the objects in detail, it is first important to consider use cases for this system in order to ensure that the objects meet the requirements of the users. In this case, there is only one real class of user: an actor who wants to use the system for the purposes of listening to music. This first use case diagram shows the three main activities a user may want to complete using the system.

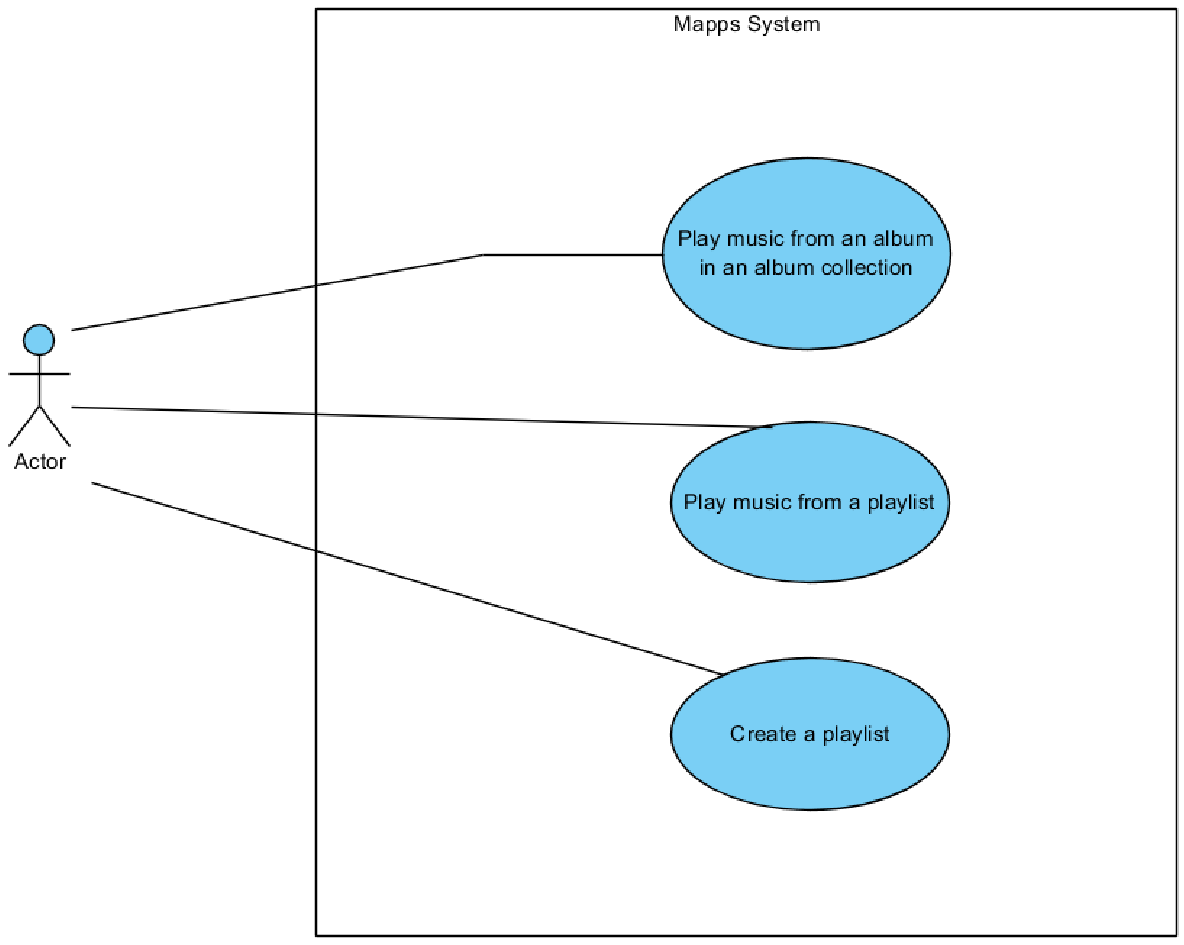

Mapps System
Play music from an album in an album collection
Actor
Play music from a playlist
Create a playlist

Figure 2: Basic Use Case Diagram

This shows the three major uses an actor may have for the Mapps system; however, it is only useful to show the basic requirements. Below is a more detailed version, showing some of how the system communicates with the system itself.

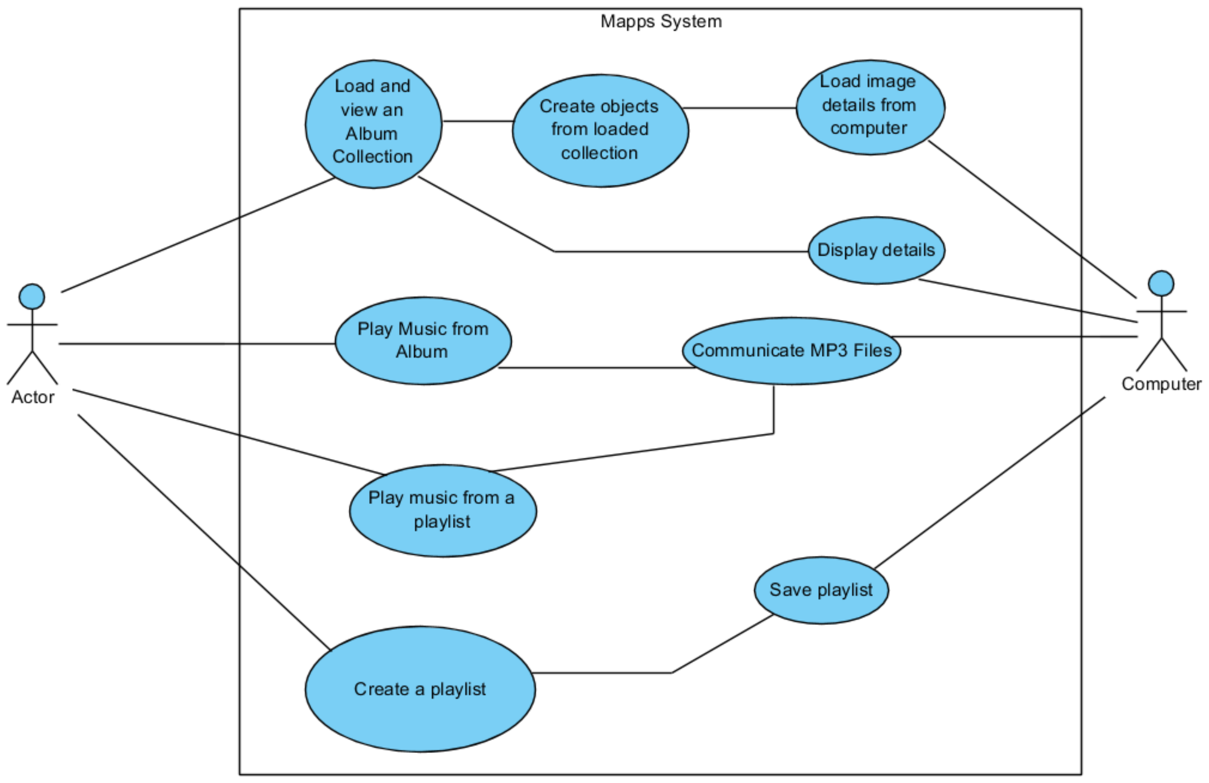

Mapps System
Load and view an Album Collection
Create objects from loaded collection
Load image details from computer
Display details
Play Music from Album
Communicate MP3 Files
Play music from a playlist
Save playlist
Create a playlist
Actor
Computer

Figure 3: More Complex Use Case Diagram

There are six main classes within this system: Duration, Track, Album, AlbumCollection, PlaylistTrack, Playlist, MappsGui, and MusicPlayer. The first 6 of these are referred to in this report as ‘base classes’, classes which were created under the first specification for this application and form the basis of the object parts necessary to make up the system. MappsGui holds the GUI information, and MusicPlayer is a short class mainly taken from Wenjia Wang’s edits of http://introcs.cs.princeton.edu/java/faq/mp3/mp3.html (How to Play an MP3 File in Java, accessed 16/1/19) to allow for the playing of music.

The first object needed is an AlbumCollection. This is a collection of Album objects, meaning an Album object is also needed. AlbumCollections need to be able to be added to and removed from, and as a result an AlbumCollection includes a List of Album objects.

Albums are what music is typically stored in. They first and foremost store tracks, in a LinkedList of Track objects. An Album also needs to be identifiable, so Strings of albumName and artistName are also required as variables within an Album object.

Track objects represent the songs themselves. As a result, they firstly need a String trackName. Tracks also contain a Duration, which exist as their own type of object, rather than merely storing it as a String, as calculations may need to be done on it. Track objects don’t hold a reference to the Album to which they belong, as it can be assumed that, if they do belong to an Album, they must be stored within the Album’s LinkedList.

Duration objects contain three integers: one for each of hours, minutes, and seconds.

PlaylistTracks exist separately to Tracks, as rather than belonging to an Album, they belong to a Playlist. As a result, they need a reference to the Album to which they belong. Consequently, PlaylistTracks inherit all the features of a Track object, and also contain an Album to which they belong.

The final of these base classes is Playlist. Playlists are simply a collection of PlaylistTracks, stored in a LinkedList to allow the addition and removal of PlaylistTracks.

The final two classes are MusicPlayer, and MappsGui. MusicPlayer is an object which allows the playing of MP3 tracks. Meanwhile, MappsGui is the link between the user and the system, using a GUI.

These classes all interact with each other in a number of ways. Below are two class diagrams, both showing the relationship between these classes. The first of these is a more basic version, which clearly shows multiplicity constraints, whilst the second is significantly more complex as it also includes all methods which the classes contain. As a result of the amount of information contained in the second diagram, it is advised a reader views this report using the PDF version, in order to zoom in on key aspects.

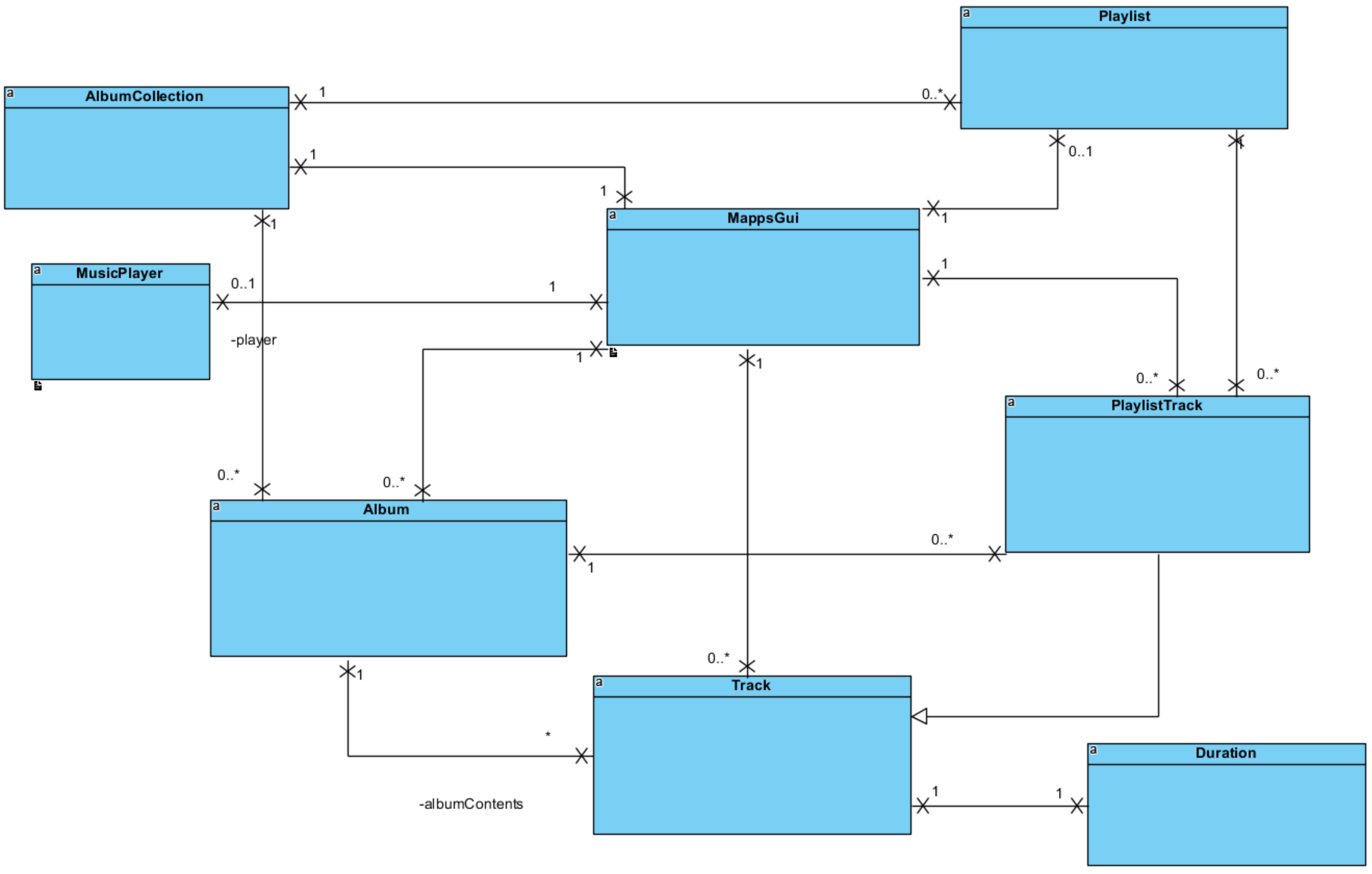


Figure 4: Simple Class Diagram

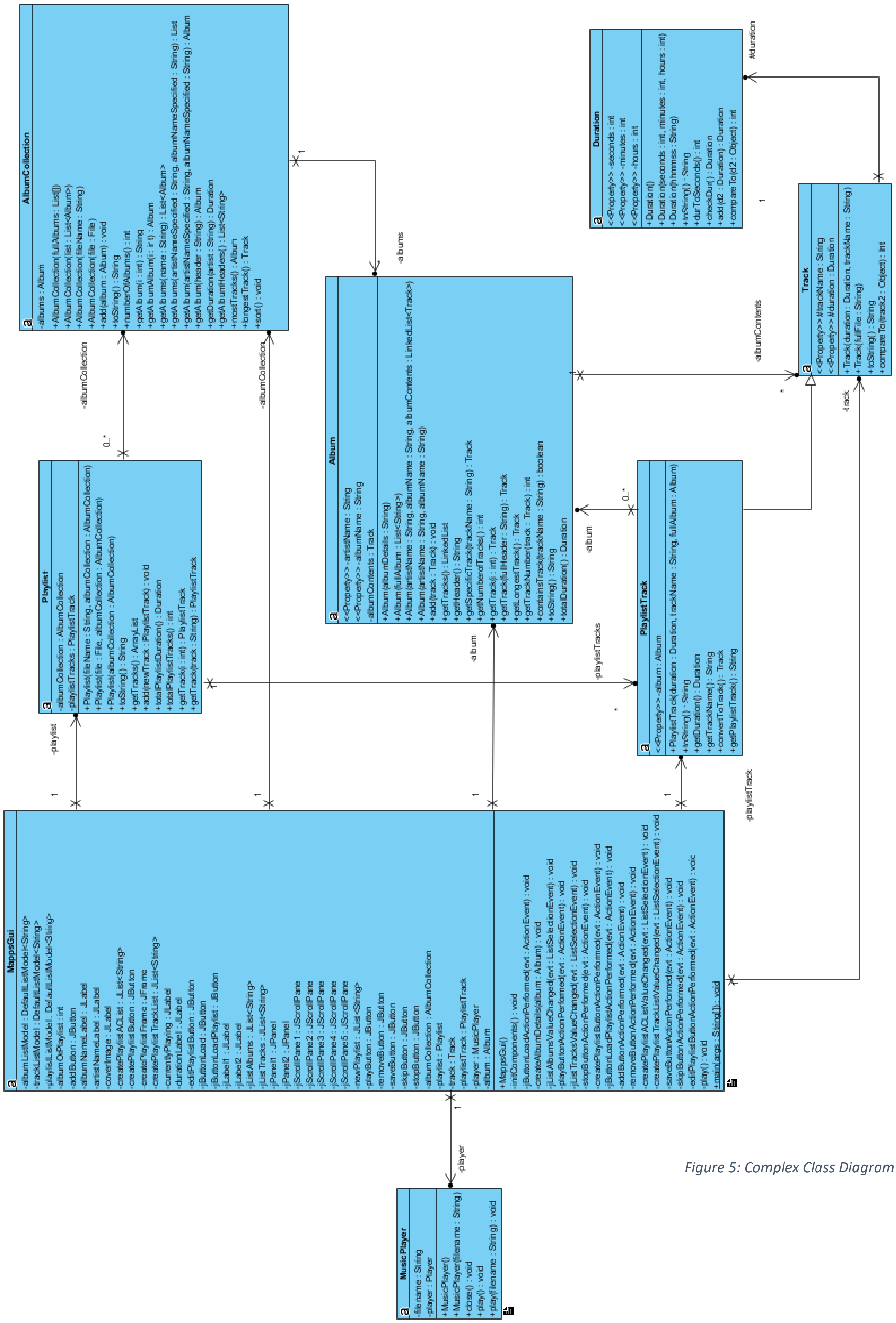

AlbumCollection
-albums : Album
+AlbumCollection(fullAlbums : List[])
+AlbumCollection(list : List<Album>)
+AlbumCollection(fileName : String)
+AlbumCollection(file : File)
+add(album : Album) : void
+toString() : String
+numberOfAlbums() : int
+getAlbum(i : int) : String
+getAlbumAlbum(i : int) : Album
+getAlbums(name : String) : List<Album>
+getAlbums(artistName : String, albumNameSpecified : String) : List
+getAlbum(artistName : String, albumNameSpecified : String) : Album
+getAlbum(header : String) : Album
+getDuration(artist : String) : Duration
+getAlbumHeaders() : List<String>
+mostTracks() : Album
+longestTrack() : Track
+sort() : void
Duration
<<Property>> -seconds : int
<<Property>> -minutes : int
<<Property>> -hours : int
+Duration()
+Duration(seconds : int, minutes : int, hours : int)
+Duration(hhmmss : String)
+toString() : String
+durToSeconds() : int
+checkDur() : Duration
+add(d2 : Duration) : Duration
+compareTo(d2 : Object) : int
Album
<<Property>> -artistName : String
<<Property>> -albumName : String
-albumContents : Track
+Album(albumDetails : String)
+Album(fullAlbum : List<String>)
+Album(artistName : String, albumName : String, albumContents : LinkedList<Track>)
+Album(artistName : String, albumName : String)
+add(track : Track) : void
+getTracks() : LinkedList
+getHeader() : String
+getSpecificTrack(trackName : String) : Track
+getNumberOfTracks() : int
+getTrack(i : int) : Track
+getTrack(fullHeader : String) : Track
+getLongestTrack() : Track
+getTrackNumber(track : Track) : int
+containsTrack(trackName : String) : boolean
+toString() : String
+totalDuration() : Duration
Track
<<Property>> #trackName : String
<<Property>> #duration : Duration
+Track(duration : Duration, trackName : String)
+Track(fullFile : String)
+toString() : String
+compareTo(track2 : Object) : int
Playlist
-albumCollection : AlbumCollection
-playlistTracks : PlaylistTrack
+Playlist(fileName : String, albumCollection : AlbumCollection)
+Playlist(file : File, albumCollection : AlbumCollection)
+Playlist(albumCollection : AlbumCollection)
+toString() : String
+getTracks() : ArrayList
+add(newTrack : PlaylistTrack) : void
+totalPlaylistDuration() : Duration
+totalPlaylistTracks() : int
+getTrack(i : int) : PlaylistTrack
+getTrack(track : String) : PlaylistTrack
PlaylistTrack
<<Property>> -album : Album
+PlaylistTrack(duration : Duration, trackName : String, fullAlbum : Album)
+toString() : String
+getDuration() : Duration
+getTrackName() : String
+convertToTrack() : Track
+getPlaylistTrack() : String
MappsGui
-albumListModel : DefaultListModel<String>
-trackListModel : DefaultListModel<String>
-playlistListModel : DefaultListModel<String>
-albumOrPlaylist : int
-addButton : JButton
-albumNameLabel : JLabel
-artistNameLabel : JLabel
-coverImage : JLabel
-createPlaylistAOList : JList<String>
-createPlaylistButton : JButton
-createPlaylistFrame : JFrame
-createPlaylistTrackList : JList<String>
-currentlyPlaying : JLabel
-durationLabel : JLabel
-editPlaylistButton : JButton
-jButtonLoad : JButton
-jButtonLoadPlaylist : JButton
-jLabel1 : JLabel
-jLabel2 : JLabel
-jListAlbums : JList<String>
-jListTracks : JList<String>
-jPanel1 : JPanel
-jPanel2 : JPanel
-jScrollPane1 : JScrollPane
-jScrollPane2 : JScrollPane
-jScrollPane3 : JScrollPane
-jScrollPane4 : JScrollPane
-jScrollPane5 : JScrollPane
-newPlaylist : JList<String>
-playButton : JButton
-removeButton : JButton
-saveButton : JButton
-skipButton : JButton
-stopButton : JButton
-albumCollection : AlbumCollection
-playlist : Playlist
-track : Track
-playlistTrack : PlaylistTrack
-player : MusicPlayer
-album : Album
+MappsGui()
-initComponents() : void
-jButtonLoadActionPerformed(evt : ActionEvent) : void
-createAlbumDetails(album : Album) : void
-jListAlbumsValueChanged(evt : ListSelectionEvent) : void
-playButtonActionPerformed(evt : ActionEvent) : void
-jListTracksValueChanged(evt : ListSelectionEvent) : void
-stopButtonActionPerformed(evt : ActionEvent) : void
-createPlaylistButtonActionPerformed(evt : ActionEvent) : void
-jButtonLoadPlaylistActionPerformed(evt : ActionEvent) : void
-addButtonActionPerformed(evt : ActionEvent) : void
-removeButtonActionPerformed(evt : ActionEvent) : void
-createPlaylistAOListValueChanged(evt : ListSelectionEvent) : void
-createPlaylistTrackListValueChanged(evt : ListSelectionEvent) : void
-saveButtonActionPerformed(evt : ActionEvent) : void
-skipButtonActionPerformed(evt : ActionEvent) : void
-editPlaylistButtonActionPerformed(evt : ActionEvent) : void
-play() : void
+main(args : String[]) : void
MusicPlayer
-filename : String
-player : Player
+MusicPlayer()
+MusicPlayer(filename : String)
+close() : void
+play() : void
+play(filename : String) : void
-albumCollection
-albumCollection
-albums
-albumContents
#duration
-track
-album
-album
-playlistTracks
-playlistTrack
-playlist
-player
1
0..*
*
1
*
1
1
1
1
1
1
0..*
*
1

Figure 5: Complex Class Diagram

These two diagrams show how the classes all very closely interrelate. Because AlbumCollections contain Albums which contain Tracks which contain a Duration, the relationships can become exceptionally complex.

In both diagrams, no aggregation or composition is present. Whilst it could be said that some of the elements belong in an aggregate relationship (for example, Tracks are part of an Album object), they also have independent existence. In order to allow the code to have more applications in the future, these have been left as not aggregate.

Object Oriented Design

When designing the objects for this system, again some of the work was completed by the specification itself. The main design choices needed were regarding the user interactions with the system itself.

Below is a Sequence Diagram for a user playing a track using the system. This diagram shows the stages the system goes through to play a track, excluding accessing the data folder to access the MP3 files to play, which is excluded for simplicity.

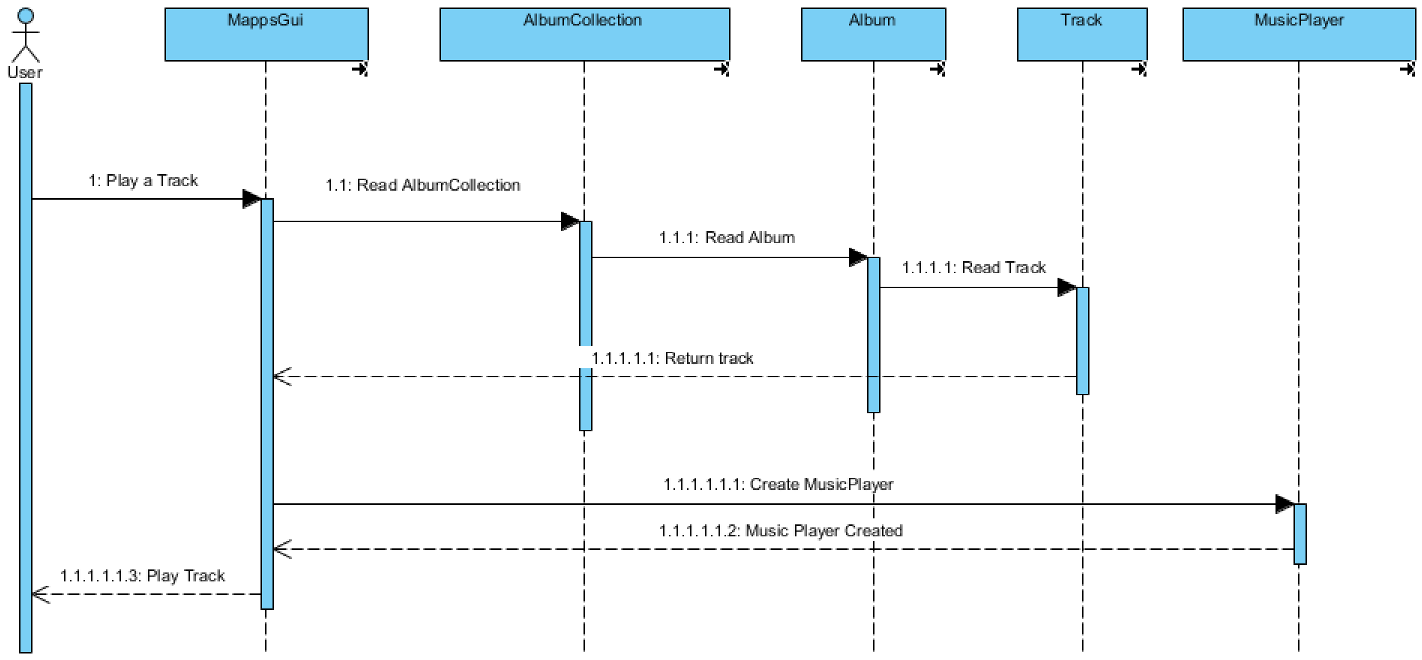


Figure 6: Sequence Diagram

The basic design of the classes includes default constructors for each class, as well as ‘Get’ methods for each component of each class, as well as aspects such as the number of elements in the various lists.

During the design phase, low-fi designs were drawn up of the GUI aspects of this project, as seen below. Some edits were made to these during the implementation of the project, due to constraints in both ability and time. In particular, the selected album no longer is in a different colour, and the cover image of an album is not contained in the AlbumCollection list. The Album details are also displayed to the right of the track list, as when implementing this looked best.

These decisions to change these things don't discount the value of the design changes, instead they were merely an evolution as the programme progressed.

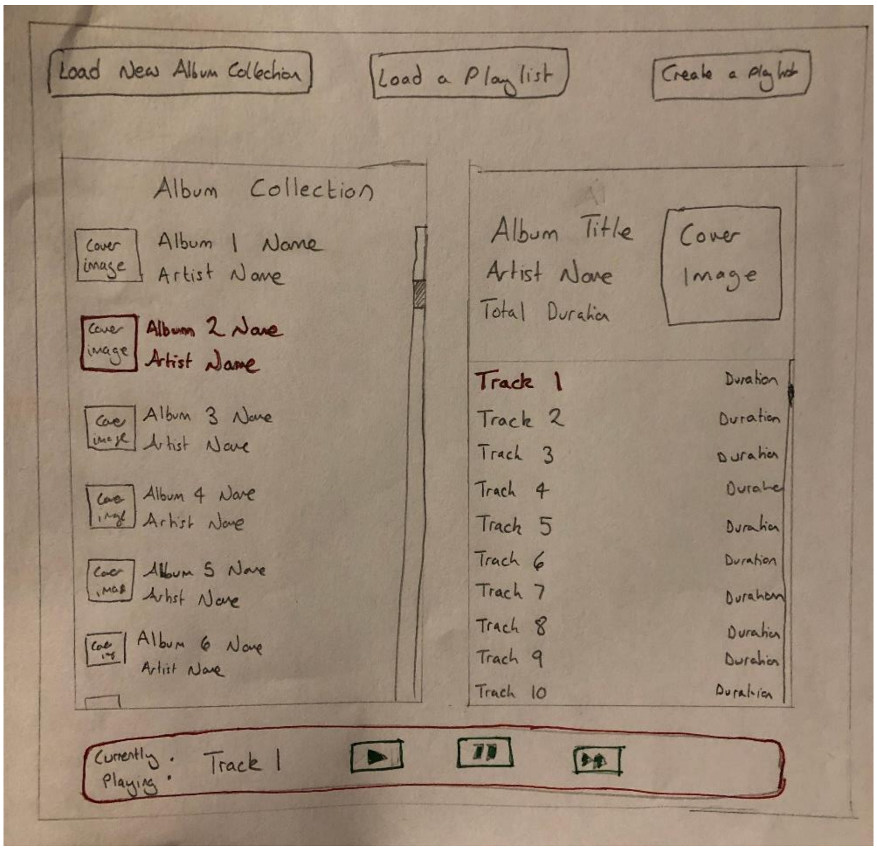


Figure 7: Low-Fi Design

Despite these edits; however, the core of the design with load buttons at the top of the screen, two lists to display albums and tracks, with the play controls at the bottom, has remained the same.

The choice to use Lists (in this case JLists) was deliberate as they can be easily added to or removed from, compared to multiple buttons which would include a large amount more coding.

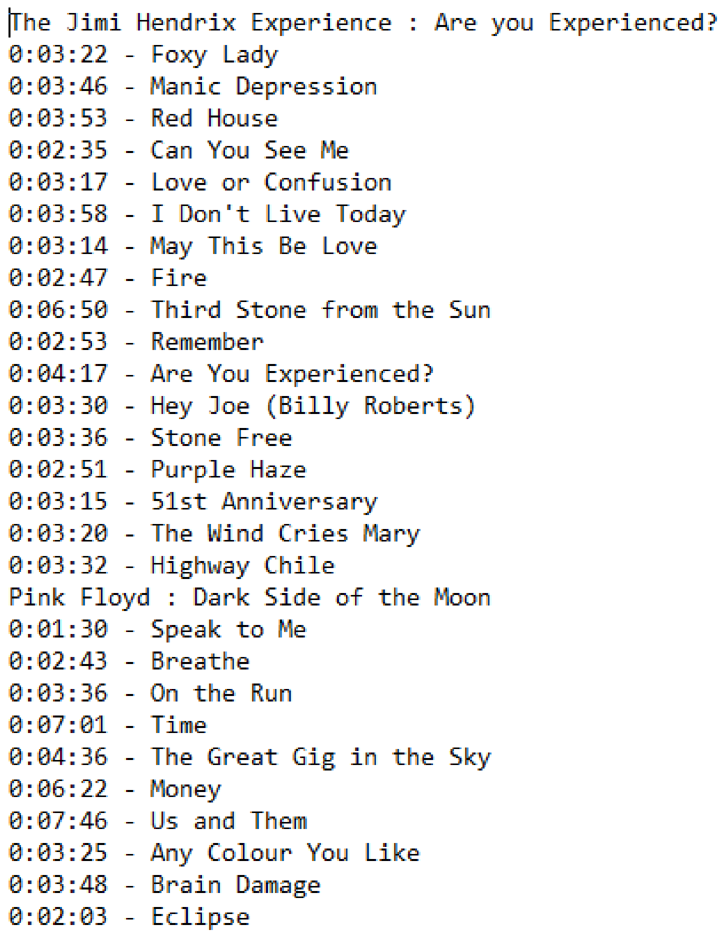
Whilst not demonstrated in this design, the intention was to create playlists in a new frame, in order to keep the design cleaner.

Implementation

In order to implement this system, the IDE NetBeans was used. As discussed above, the base classes created first, in approximately that order. This meant when an element needed to reference a previous class it was already created with most methods needed (although some methods were added afterwards).

When implementing the user-facing element of the application, including the GUI, a different tactic was followed. In this instance, each user case was created, implemented, and tested before moving on to the next aspect. This was to ensure that all aspects of functionality were achieved, and also to provide convenient rollback points should be needed. This was also enforced using Git version control.

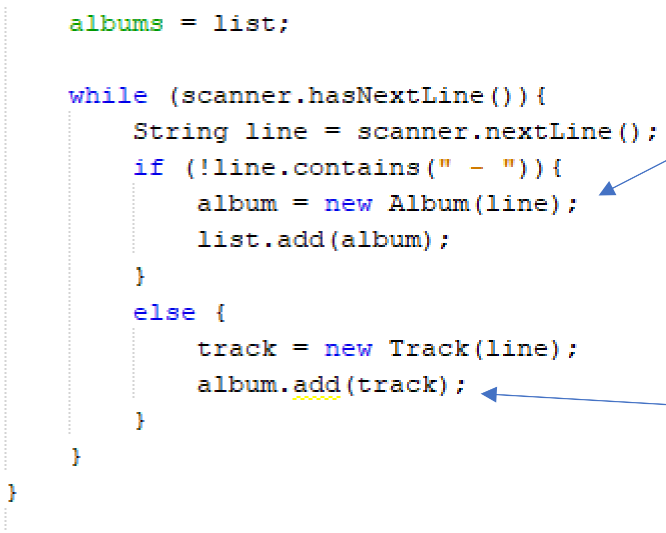
In order to access the tracks, they need to be loaded into the application from a text file. The text file must be in a specific format, with an artist name and album title first, separated by a colon, then each track on a new line, with the duration in the hh:mm:ss format, a dash, then the track title. An example of this is below.



```
The Jimi Hendrix Experience : Are you Experienced?
0:03:22 - Foxy Lady
0:03:46 - Manic Depression
0:03:53 - Red House
0:02:35 - Can You See Me
0:03:17 - Love or Confusion
0:03:58 - I Don't Live Today
0:03:14 - May This Be Love
0:02:47 - Fire
0:06:50 - Third Stone from the Sun
0:02:53 - Remember
0:04:17 - Are You Experienced?
0:03:30 - Hey Joe (Billy Roberts)
0:03:36 - Stone Free
0:02:51 - Purple Haze
0:03:15 - 51st Anniversary
0:03:20 - The Wind Cries Mary
0:03:32 - Highway Chile
Pink Floyd : Dark Side of the Moon
0:01:30 - Speak to Me
0:02:43 - Breathe
0:03:36 - On the Run
0:07:01 - Time
0:04:36 - The Great Gig in the Sky
0:06:22 - Money
0:07:46 - Us and Them
0:03:25 - Any Colour You Like
0:03:48 - Brain Damage
0:02:03 - Eclipse
```

Figure 8: Text File Example

When the application loads a text file in the correct format, it creates an AlbumCollection object, albums for each album in the list, tracks in those albums, and durations for each track. To do this, the text file is first split by album, by splitting based on whether a line contains a dash or not.

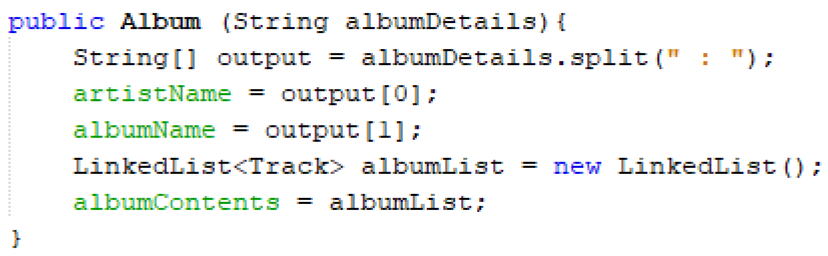


```
    albums = list;

    while (scanner.hasNextLine()){
        String line = scanner.nextLine();
        if (!line.contains(" - ")){
            album = new Album(line);
            list.add(album);
        }
        else {
            track = new Track(line);
            album.add(track);
        }
    }
}
```

If the line from the text file doesn't contain a dash, then it is the start of a new album. The album is created and added to 'list', which is the AlbumCollection's list of albums.

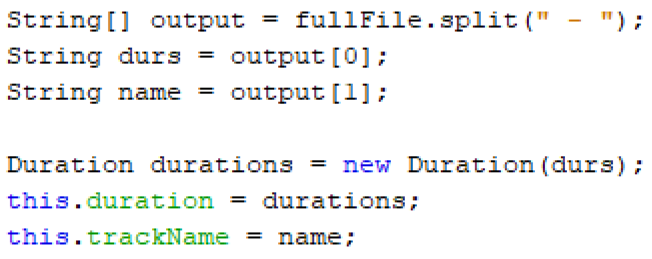
If the line doesn't contain a dash, it must be a track for whichever album was most recently created. The track is created, and added to the album.

The album is created by splitting the line on the colon (" : "). It is separated into the artist name and album title, and then an empty list is created for the tracks to be added into. In a different scenario, this code could be reused with the tracks being passed in simultaneously, as a constructor for this scenario does exist.



```
public Album (String albumDetails){
    String[] output = albumDetails.split(" : ");
    artistName = output[0];
    albumName = output[1];
    LinkedList<Track> albumList = new LinkedList();
    albumContents = albumList;
}
```

The creation of the tracks again involves splitting the line of the track. In this case, the line is split on the dash (" – "), to separate the track name (which is simply added into the track) and the duration, which is created as a duration object then passed back to the track.



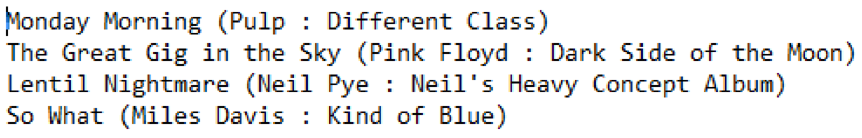
```
String[] output = fullFile.split(" - ");
String durs = output[0];
String name = output[1];

Duration durations = new Duration(durs);
this.duration = durations;
this.trackName = name;
```

The created AlbumCollection is then displayed in a JList in the GUI. From here albums, then tracks, can be selected and played. There is more discussion of the design aspects in the Design section of this report.

Each of these base classes also contain a large number of methods. In particular, because there is a large amount of iterating over objects when creating AlbumCollections, Albums, and Playlists, there are always methods to see how many elements (for example Tracks in an Album) exist. There are also methods to get each part of an object, and manipulation methods including adding to collections.

This system was also designed to be able to create and play playlists. Playlists are made up of PlaylistTrack objects, rather than Track objects, as they also need to contain a pointer to the album from where the track originates. Instead, PlaylistTrack extends Track, inheriting the need for a Duration and a track name.

Like with an AlbumCollection, a playlist needs to be loaded in from a text file in a specific format, with the track name first, then the album and artist in brackets.



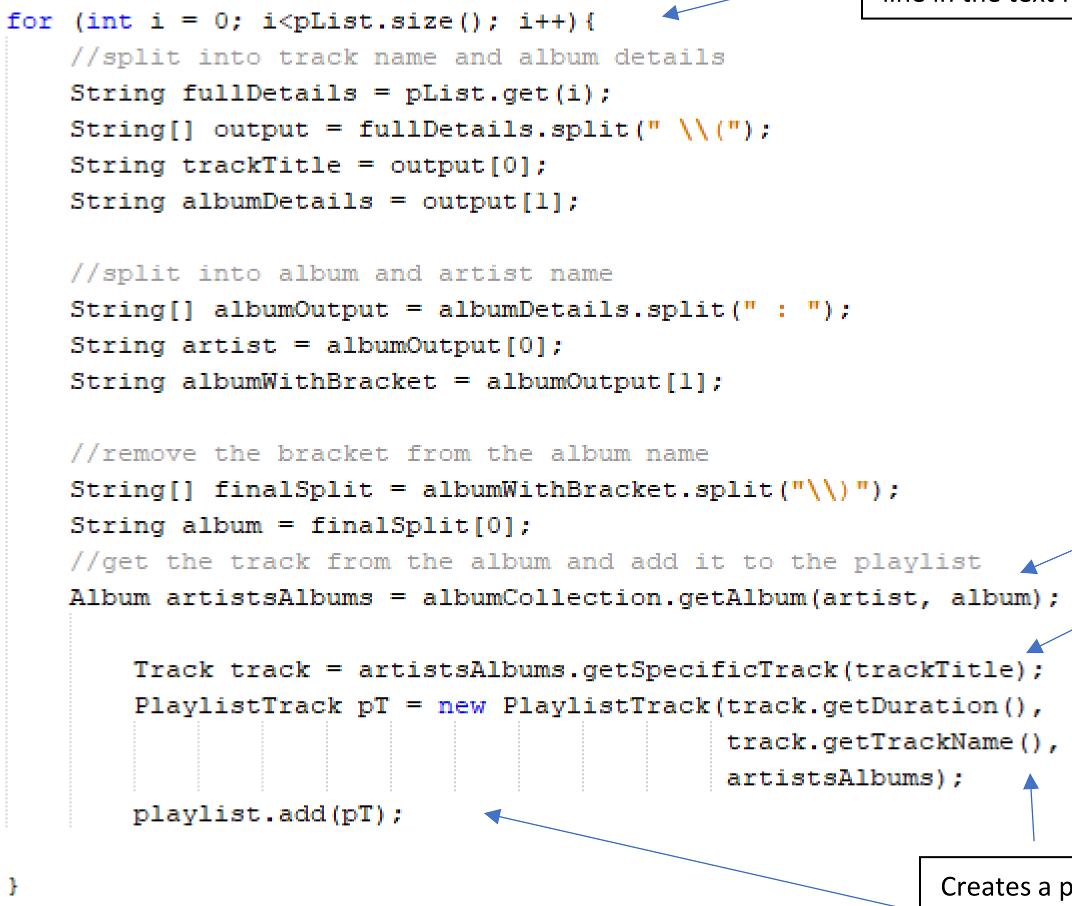
```
Monday Morning (Pulp : Different Class)
The Great Gig in the Sky (Pink Floyd : Dark Side of the Moon)
Lentil Nightmare (Neil Pye : Neil's Heavy Concept Album)
So What (Miles Davis : Kind of Blue)
```

Figure 9: Playlist File Example

A playlist can then be created from this file combined with an AlbumCollection which includes all these tracks. Like with the creation of albums and tracks, each line is split to create the individual elements of the PlaylistTrack: duration, title, and album.

Once the track name and album details have been extracted from the text file, the duration information is taken from the album by finding it from the AlbumCollection. This is done through the method AlbumCollection method getAlbum(artistName, albumName), which scans each Album in the AlbumCollection to find one with the same artist and album names as specified. This Album is then passed back to the Playlist constructor, which calls the getSpecificTrack(trackName) method to retrieve the track needed. From here, all the elements for a PlaylistTrack are present and it can be added to the Playlist. The process is then repeated for each line in the file, until the playlist is created.

For each track in the playlist (each line in the text file), this is repeated.



```
for (int i = 0; i<pList.size(); i++){
    //split into track name and album details
    String fullDetails = pList.get(i);
    String[] output = fullDetails.split(" \\(");
    String trackTitle = output[0];
    String albumDetails = output[1];

    //split into album and artist name
    String[] albumOutput = albumDetails.split(" : ");
    String artist = albumOutput[0];
    String albumWithBracket = albumOutput[1];

    //remove the bracket from the album name
    String[] finalSplit = albumWithBracket.split("\\)");
    String album = finalSplit[0];
    //get the track from the album and add it to the playlist
    Album artistsAlbums = albumCollection.getAlbum(artist, album);

        Track track = artistsAlbums.getSpecificTrack(trackTitle);
        PlaylistTrack pT = new PlaylistTrack(track.getDuration(),
                                             track.getTrackName(),
                                             artistsAlbums);
        playlist.add(pT);

}
```

Finds the Album from the AlbumCollection, and the Track from the Album

Creates a playlist track, and then adds it to the playlist

With regards to creating the GUI, the actual creation of the design was done using the NetBeans design tool, although a first rough draft was done by coding the design manually. This approach allowed me to know how the parts of the GUI would properly interact, rather than allowing NetBeans to create code for me; however, using the design tool to create the final product created a more elegantly designed project than manually coding it would have produced.

The majority of object creation and computation occurs within either the initialisation of objects within the GUI, or within ‘action listener’ event (for example button clicks or list selection changes). Because the majority of the work is done through the calling of methods from other classes, this only sacrifices a small amount of future flexibility with the code.

The one exception to this rule is the Play() method, which gets a selected track from an AlbumCollection (or Playlist), via its album, then creates a MusicPlayer and plays it. This was put in its own method as the ‘skip’ button also needed to play a track. As a result, a new method was created to avoid duplicating a large amount of code.

Testing

The testing for this project involved two main steps: Validation and Verification. Validation is checking that the application being created matches the specification given, whilst verification checks that the application is being created correctly, so that it works.

Validation

Validation testing during this project occurred at two key points, after the original GUI designs were created and again at the end of the project to check the application met all specifications. This was mainly completed by taking the specification and extracting the key requirements:

- Load an Album Collection
- Choose an album from that collection and display information about it including name, artist, duration, and cover image.
- Display the tracks from a selected album.
- Allow a selected track to be played.
- Stop a played track.
- Load a playlist.
- Play a track selected from a playlist.
- Create a playlist.
- Edit a previously created playlist.

From this, the low-fi designs were compared with this to see if they met would allow for all these elements to be met. This process allowed a good sense of how to code the project, as it had a clear end goal.

This process was then repeated when the application was completed, this time comparing the codes' methods as well as the design to ensure each aspect of the specification had been completed and, at least in theory, worked. This was done without executing the code, and before final verification testing, so as to not become distracted by any bugs which could be encountered.

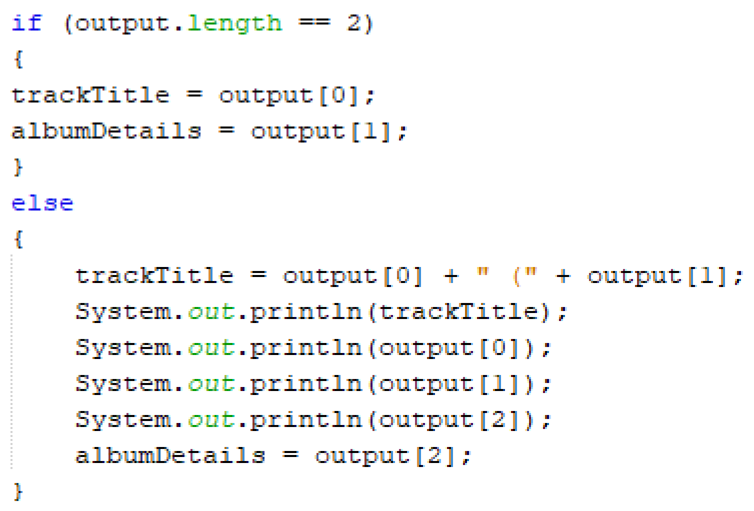
Verification

Verification testing also took place in numerous places. The first stage occurred whilst creating the first part of the specification (the creation of the base classes). At the end of each class, a main method 'test harness' was created and each method tested thoroughly before moving to the next class. These test harnesses have since been deleted from the code in order to make it neater.

The code was also continually ran whilst building the GUI and music player, in order to ensure not too much work was completed before discovering an application-stopping bug. This process was informal, with the code simply being ran every so often, particularly when 'action listener' parts had been created (for example, to check that a clicking a button did as expected).

Once the application had been completed, more formal testing took place. The first step of this was attempting to ‘break’ the code, trying to create as many scenarios as possible to find bugs in the code. These bugs were then examined using the debugger tool, and fixed.

One issue that occurred was that, when loading a playlist, the original creation method could not cope with any track which contained brackets in the track name, as the loader split the track name on the brackets. This led to a “Array Index Out of Bounds” exception, as the part of the track name which was in the brackets would not include a colon, which was expected in order to create the PlaylistTrack. This issue was solved by checking the number of elements within the split line: if it is more than 2, then it is assumed that outputs [0] and [1] need to be joined with the addition of “ (“ in order to form the track title. This method does not currently allow for track titles with multiple sets of brackets; however it is assumed that this is unlikely to form much of a problem. This fix can also be easily replicated for any number of elements, should the need arise.


```
if (output.length == 2)
{
trackTitle = output[0];
albumDetails = output[1];
}
else
{
    trackTitle = output[0] + " (" + output[1];
    System.out.println(trackTitle);
    System.out.println(output[0]);
    System.out.println(output[1]);
    System.out.println(output[2]);
    albumDetails = output[2];
}
```

Another issue that was discovered was that, if a user pressed the play button whilst a different track was already playing, the newly selected track wouldn’t play. This was fixed by closing the player at the beginning of the play() method in the MappsGui class.

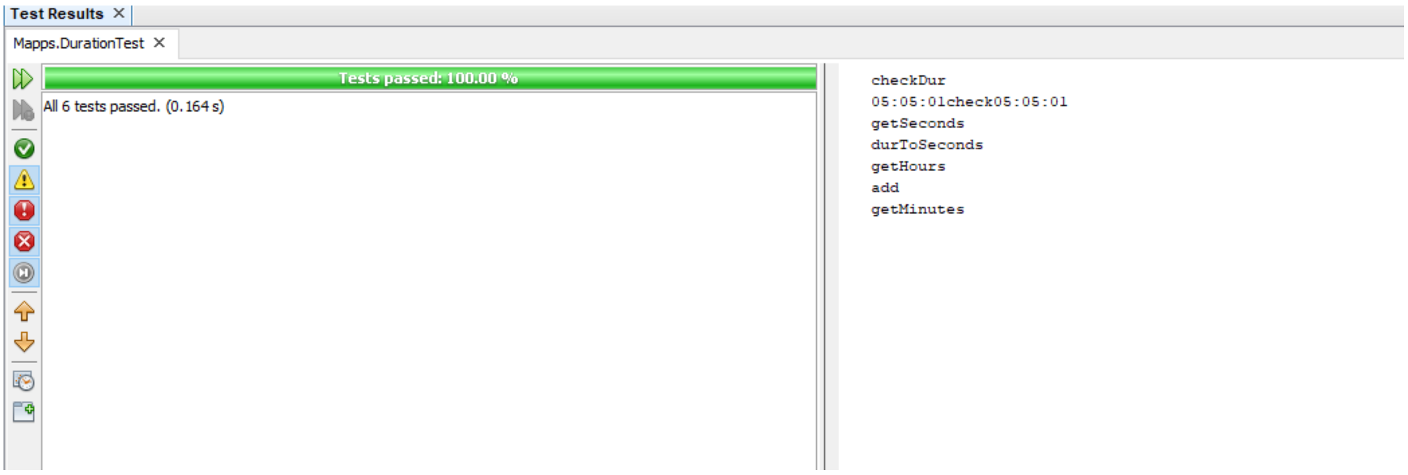
Another set of bugs encountered was when accessing and creating the cover images for the albums. Firstly, because the file names don’t include special characters, these need to be removed. This led to excess white space, which meant that image files couldn’t be found. There were two solutions to this: firstly, calling trim() on the final string removed all white space at the end of the String. This worked for issues where the removed word or symbol came at the beginning or end of the String, however it was not useful where a double space had occurred when removing a special character (such as ‘&’) from the middle of a string. This was fixed by running a second replace characters, this time replacing a double space with a single space. Another issue that occurred in this method was that there were two different naming conventions for where an artist name contained multiple words, as some merely removed all spaces whilst others initialised the artist name. As a result, these two strategies would always override the others. This was fixed by adding a new try-catch clause within the catch clause itself, which attempts to create the image based on the second naming convention. This method of fixing the issue is slightly ‘clunky’, as it deals with individual circumstances rather than creating catch-all rules.

In order to fully test the user experience, a test script was devised at the time of design. When the project was completed, this was completed to ensure the user could do everything they needed to, without bugs. The application passed, as seen below.

Test Case	Expected Result	Pass/Fail
Load an AlbumCollection	AlbumCollection loads and displays in the first list	Pass
Select an album	Album tracks displayed in second list, details display to right (title, artist, duration), cover image displays if exists	Pass
Select a track	Track is selected	Pass
Play a selected track	Track plays, file name outputted to the log	Pass
Play a track which does not have a corresponding MP3 file	An error window should display	Pass
Load a playlist	Playlist should display in the second list	Pass
Select a track	Track is selected	Pass
Play a selected track	Track plays, file name outputted to the log	Pass
Load a playlist without an album collection	Error message displays	Pass
Create a playlist button pressed	New frame opens with three lists, 1 showing the loaded album collection, the others blank. A save button at the bottom	Pass
Create playlist button without an album collection	Error message displays	Pass
Make a playlist	Selecting an album displays tracks in second list, these can be selected	Pass
Playlist add button pressed	Selected track added to third list	Pass
Playlist remove button pressed	Selected track removed from list	Pass
Save button pressed	Save button dialogue opens, allows saving of playlist	Pass
Edit playlist button pressed	Same frame as create playlist opens, but with third list filled with current playlist	Pass
Edit playlist button pressed without a playlist loaded	Error message displays	Pass
Tracks added/removed from playlist to be edited	Works correctly	Pass
Newly edited playlist can be saved	Saves successfully	Pass

Figure 10: Test Script

Finally, all the base classes were subjected to JUnit testing, to ensure the methods worked as expected. These too passed.



Tests passed: 100.00 %

All 6 tests passed. (0.164 s)

```
checkDur
05:05:01check05:05:01
getSeconds
durToSeconds
getHours
add
getMinutes
```

Evaluation

This program has many drawbacks in its current form. Firstly, the loading of images into the system is badly put together. Because cover images could be named in many different ways, a number of different character replacements were needed in order to ensure an image was loaded correctly. In a future version of this software, this would be a more streamlined process, perhaps with a stipulation that a single naming convention be followed.

This issue of code catering to individual circumstances can also be found in the handling of brackets in PlaylistTrack names (as discussed in the Testing section of this report). This limits the reusability of the code.

Secondly, because a lot of code is contained within an action listener event, it isn't particularly easily reused by future developers. This is counterbalanced, however, by the majority of methods being contained within the base classes. As a result, if the GUI were to be redesigned, or a new GUI created, a large amount of these methods could be simply transplanted.

Summary

In conclusion, this application meets all the criteria set out in the specification, and also exists in a form easily accessible to users.

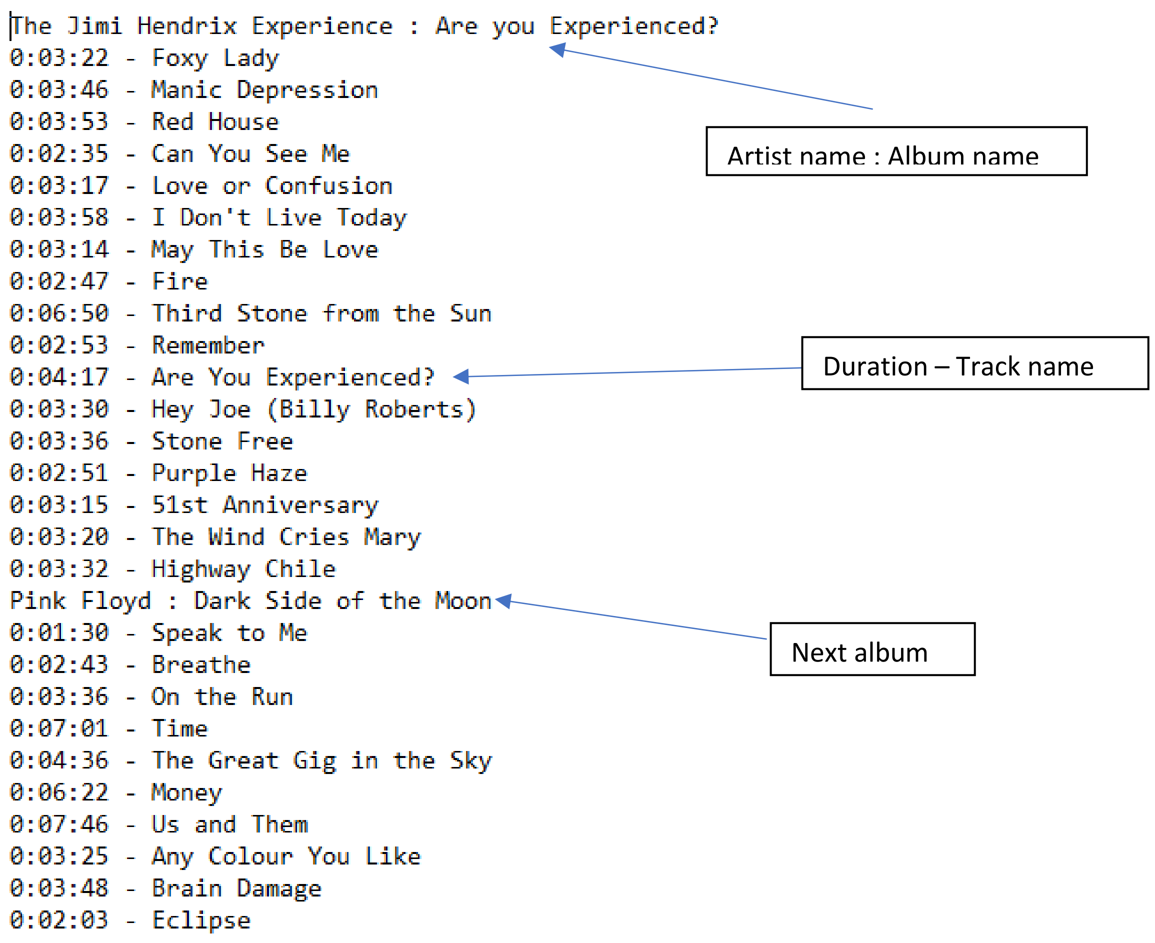
Despite this, the project still has some drawbacks. Parts of the code are very specific to their application at that point. Furthermore, some of the means of achieving the aims may not be the most efficient or be the best practice. This is due to inexperienced coding, and also a lack of time.

User Manual

1) Setup
   There is some setup involved in running tracks using the Mapps system. Firstly, all audio and image files must be saved in the same place as the application, in a folder called 'data'.
   There are also some naming conventions which must be followed. Track names must have any special characters (for example '&') removed from the title and artist names. & symbols should be replaced with underscores, whilst other characters siuch as apostrophes should simply be ignored. The track names must also have all spaces replaced with underscores. Finally, the full file name must begin with the artist name, followed by the track number, then the name.
   For example: Lovira_-_01_-_ All_things_considered.mp3
   Mr__Mrs_Smith_-_06_-_Kitchen_sink.mp3

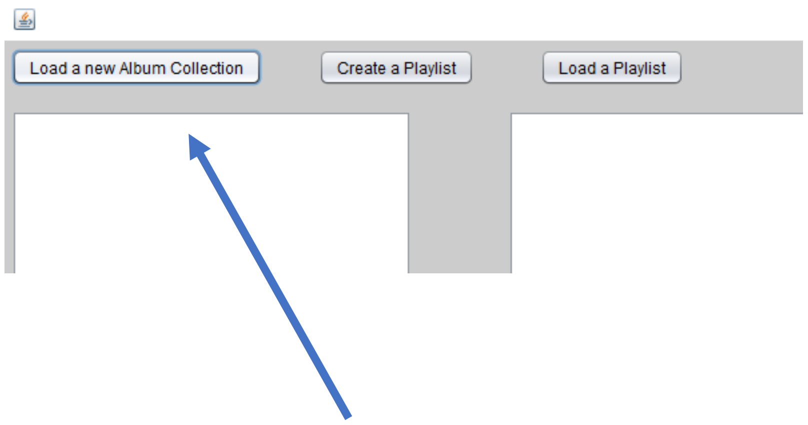
   The cover image files also must follow some naming conventions. Spaces should be removed, as should all special characters should be removed. If there are multiple words in the artist name, these can either left with all spaces removed or initialised. The artist name and title should be separated by an underscore.
   For example: PF_Darksideofthemoon
   Pulp_Differentclass

Album Collections to be played need to be stored in the format shown below. This should be saved as a text file anywhere on your computer.

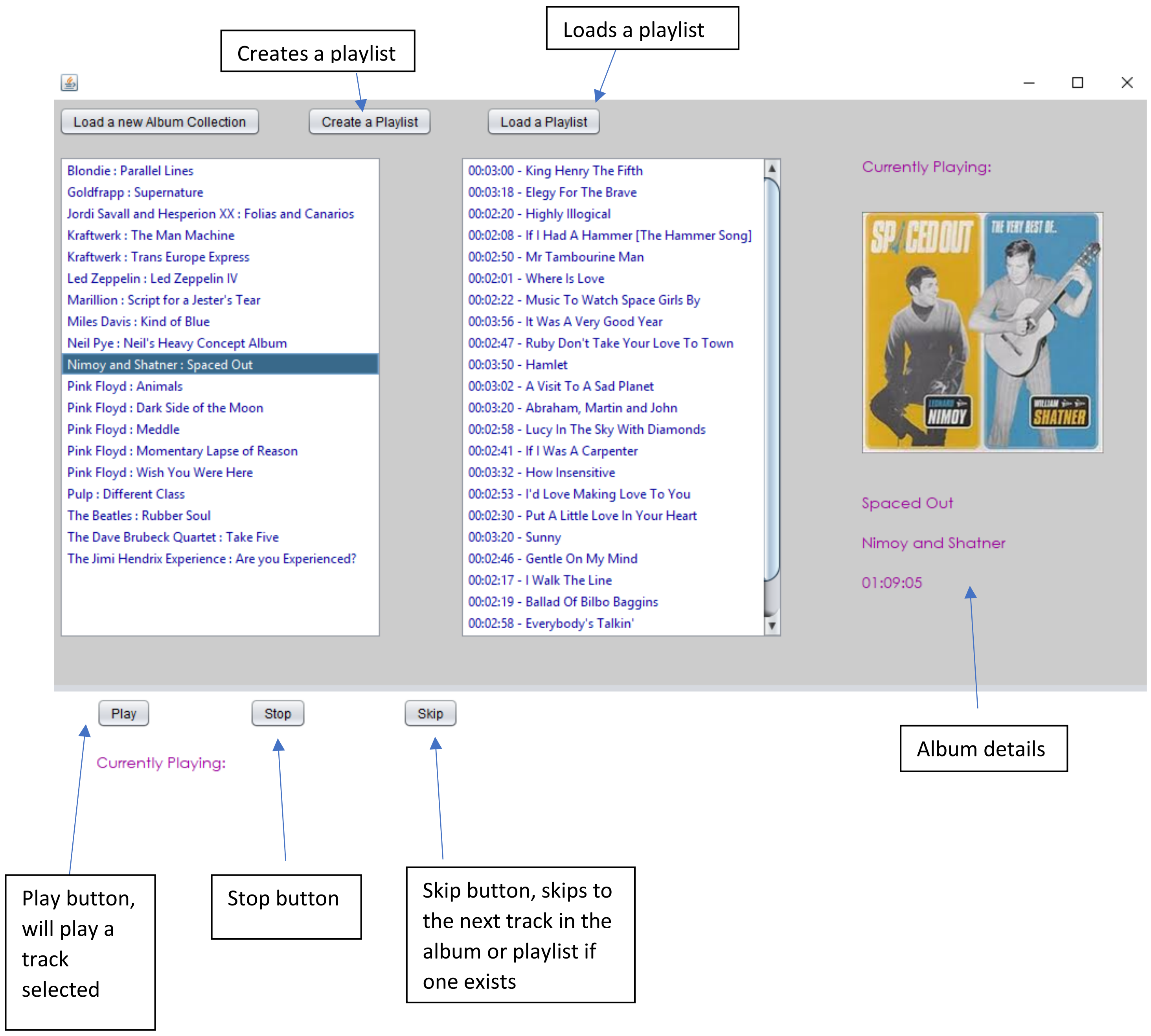


```
The Jimi Hendrix Experience : Are you Experienced?
0:03:22 - Foxy Lady
0:03:46 - Manic Depression
0:03:53 - Red House
0:02:35 - Can You See Me
0:03:17 - Love or Confusion
0:03:58 - I Don't Live Today
0:03:14 - May This Be Love
0:02:47 - Fire
0:06:50 - Third Stone from the Sun
0:02:53 - Remember
0:04:17 - Are You Experienced?
0:03:30 - Hey Joe (Billy Roberts)
0:03:36 - Stone Free
0:02:51 - Purple Haze
0:03:15 - 51st Anniversary
0:03:20 - The Wind Cries Mary
0:03:32 - Highway Chile
Pink Floyd : Dark Side of the Moon
0:01:30 - Speak to Me
0:02:43 - Breathe
0:03:36 - On the Run
0:07:01 - Time
0:04:36 - The Great Gig in the Sky
0:06:22 - Money
0:07:46 - Us and Them
0:03:25 - Any Colour You Like
0:03:48 - Brain Damage
0:02:03 - Eclipse
```

2) Playing from an album
Upon opening the application, you should see this screen:
The first thing you should do is load an album collection text file using the ‘Load a new Album Collection’ button.



This will display a list of albums which you can select from. When selecting an album, the tracks are then displayed. Selecting a track, then the play button, will play a track.


Creates a playlist
Loads a playlist
Load a new Album Collection
Create a Playlist
Load a Playlist
Currently Playing:
Play
Stop
Skip
Currently Playing:
Album details
Play button, will play a track selected
Stop button
Skip button, skips to the next track in the album or playlist if one exists

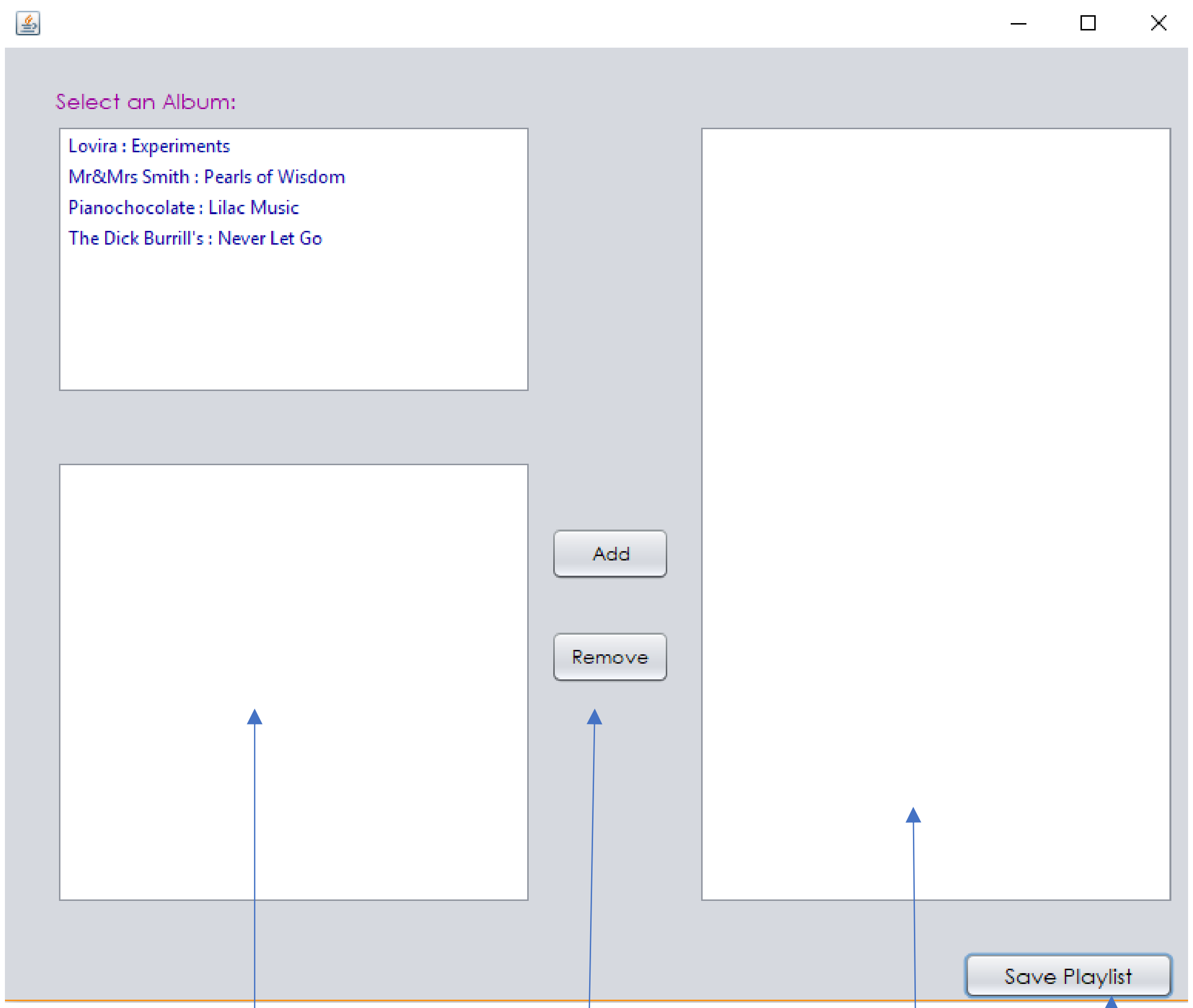
Blondie : Parallel Lines
Goldfrapp : Supernature
Jordi Savall and Hesperion XX : Folias and Canarios
Kraftwerk : The Man Machine
Kraftwerk : Trans Europe Express
Led Zeppelin : Led Zeppelin IV
Marillion : Script for a Jester's Tear
Miles Davis : Kind of Blue
Neil Pye : Neil's Heavy Concept Album
Nimoy and Shatner : Spaced Out
Pink Floyd : Animals
Pink Floyd : Dark Side of the Moon
Pink Floyd : Meddle
Pink Floyd : Momentary Lapse of Reason
Pink Floyd : Wish You Were Here
Pulp : Different Class
The Beatles : Rubber Soul
The Dave Brubeck Quartet : Take Five
The Jimi Hendrix Experience : Are you Experienced?

00:03:00 - King Henry The Fifth
00:03:18 - Elegy For The Brave
00:02:20 - Highly Illogical
00:02:08 - If I Had A Hammer [The Hammer Song]
00:02:50 - Mr Tambourine Man
00:02:01 - Where Is Love
00:02:22 - Music To Watch Space Girls By
00:03:56 - It Was A Very Good Year
00:02:47 - Ruby Don't Take Your Love To Town
00:03:50 - Hamlet
00:03:02 - A Visit To A Sad Planet
00:03:20 - Abraham, Martin and John
00:02:58 - Lucy In The Sky With Diamonds
00:02:41 - If I Was A Carpenter
00:03:32 - How Insensitive
00:02:53 - I'd Love Making Love To You
00:02:30 - Put A Little Love In Your Heart
00:03:20 - Sunny
00:02:46 - Gentle On My Mind
00:02:17 - I Walk The Line
00:02:19 - Ballad Of Bilbo Baggins
00:02:58 - Everybody's Talkin'

Spaced Out
Nimoy and Shatner
01:09:05

A playlist can be loaded from the load playlist button, and played from in the exact same way as an album.

3) Creating a playlist
Once an album collection has been loaded, a playlist can be created by pressing the create playlist button. Tracks can be selected from any of the albums in the collection and added to the playlist. This can then be saved to anywhere on your computer, and can later be loaded into the Mapps system.


Select an Album:
Lovira : Experiments
Mr&Mrs Smith : Pearls of Wisdom
Pianochocolate : Lilac Music
The Dick Burrill's : Never Let Go
Add
Remove
Save Playlist

Here tracks from a selected album will display

The add button adds a track selected from the list on the left into the playlist, whilst the remove button removes a track from the playlist

Tracks in the playlist

Opens a window to save the playlist

References:

Wayne, K. (2005), How to Play an MP3 File in Java, as edited by Wang, W (2014), accessed 16/1/19