# Objective

* Learn to use the [Java Collections Framework](http://java.sun.com/j2se/1.5.0/docs/guide/collections/index.html) & Java Input Output
* Implement a system according to specifications that are provided by someone else.
* Learn to implement Object oriented analysis and design

# Instructions

1. You are not required to provide the GUI for the system rather you are providing menu based interface for music library app on console.
2. Sample code is provided to handle MP3 playback and you need to integrate MP3 playback with your music library App. Therefore, you will also not be considering the library that handles the playback of your MP3s.
3. You are also provided with a sample text file “Music.txt” which will serve as music library database.

# Functional Requirements – (from SRS/RSD)

The music library app will run on a single computer and will be accessible to a single user. There will only be a single Library of music.

All Songs contain a title, a path to the associated MP3 file on the disk (e.g., "C:\My MP3s\Ode to Joy.mp3"), and a track number. Songs are uniquely identifiable by MP3 file path; no two Songs can exist that reference the same MP3 file. Songs should be associated with Artists, Albums and the Library.

Artists have a name. No distinction will be made between individuals and groups (e.g., "Michael Jackson" and "The Jackson 5" are both valid Artists). Artists are uniquely identifiable by name. No two artists can have the same name.

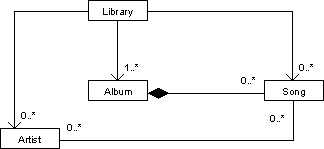
Albums are a collection of Songs, ordered by track number (where each song on an album has a different track number) – and will store the album title. In addition, there will be at least one album in the system at all times (a "default" album) and every song must be associated with a single album. Any song for which the user has not entered an album will belong to the default album. Albums are uniquely identified by name.

There will be a single Library which contains collections of Albums, Songs, and Artists. A careful design of the associations between these four classes will be required in order to meet the following functional requirements. Your program must allow the user to:

|  |  |
| --- | --- |
| RequirmentNo | Description |
| R1 | View a list of all the songs in the library and see information about each song, including the album on which it appears and the artist(s) who  performed it. |
| R1.1 | View a list of all the artists in the library. |
| R1.2 | View a list of all the albums in the library. |
| R3 | Edit any information in the library. This includes changing the title of  songs and albums and changing the names of artists. |
| R4 | View a list of all the songs by a particular artist. |
| R5 | View a list of all the songs on a particular album. |
| R6 | Add songs to and remove them from the library. |
| R7 | Add artists to and remove them from songs. |
| R8 | Add songs to and remove them from albums. |

# Design

The following class diagram gives an overview of the music library app:



* Each instance of the Song class will correspond to a unique file on disk and will store the following pieces of information:
  + The path to the file on disk
  + The title of the song
  + The duration of the song
  + The track number
* Each instance of the Artist class will store the name of a person (eg. "Lucinda Williams") or group (eg. "Ben Harper & the Innocent Criminals", "Radiohead") that performs Songs. There will be no distinction between groups and individual people.
* Each instance of the Album class will represent an album - a collection of Songs ordered by track number (where each song on an album has a different track number) - and will store the album title. In addition, there will be at least one album in the system at all times (a "default" album) and every song will be associated with exactly one album as shown above. Any song for which the user has not entered an album will belong to the default album.
* There will be a single instance of the Library class in the system which will maintain collections of the other three classes and provide methods to create and destroy them that ensure consistency between the library and the songs, artists, and albums.
* Each class will also implement the associations depicted above.

# Code

In addition to everything stated above, you should keep the following in mind while writing code:

* When editing information in the library or adding songs, users of your program will specify Artists and Albums by entering their names/titles.
* We assume the path to songs will not change, that is, if the user renamed or moved a file they would need to remove the corresponding Song from the Library and create a new Song.
* The library will not be allowed to contain two Albums with the same title, two Artists with the same name, or two Songs representing the same file.
* Any method that returns a collection of Albums should return them in alphabetical order by title.
* Any method that returns a collection of Artists should return them in alphabetical order by name, with one exception: invoking getArtists() on a Song should return them in the order they were added to the Song.
* The order in which Songs are returned is irrelevant (since it is not clear by which attribute the user is most likely to want their songs ordered - we can leave this decision up to the user interface) with the exception that the getSongs() method of the Album class must return its Songs ordered by track number.

ATTENTION: Be Brave, and dare it by yourself. Don’t Cheat from a friend or internet. This will lead toward severe penalties for sure.

Happy Coding.