# BandCamp

## Vision Statement

An innovative, data driven approach to sharing and finding music.

## Summary

BandCamp is a system aimed at musicians who want to share their music with the world, as well as allowing music aficionados search for new music and interact with its creator.

The BandCamp system will allow a user to register an account to become a member. As a member they can upload songs to their account so that other members can listen to them. They can also listen to music uploaded by other members.

Songs can be searched for by their title, the member who uploaded it, by genre or by all three.

## GitHub Repositories

Code: <https://github.com/JamesDaniel/OOAD_BandCamp_Code>

Doc: <https://github.com/JamesDaniel/OOAD_BandCamp>

## Use Cases

**Bandcamp Use Case 1.1: Create Account**

The user creates a new account by visiting the website, clicking on the Sign up button and providing their information.

### **Bandcamp Use Case 1.2 : Log on**

The user logs onto their account by providing their username and password. If the user has forgotten either there is a process which they can go through to set a new one.

### **Bandcamp Use Case 1.3** Log off

The user logs off by clicking on the Log Off link.

### **Bandcamp Use Case 1.4** Upload Song

Primary Actor: The user – wants to upload a song to the application so it can be shared and listened to by other users.

Preconditions: The user must have an account and must be logged into the account.

Post conditions: The user’s song has been uploaded, the user has received a notification on their screen and has been redirected to the song’s page.

Main Success Scenario:

1. User selects “Upload Song” from their home page.
2. User is bought to a page where they can input their song details.
3. User provides a name for their song and an upload link. User may optionally provide a tag for the song.
4. User clicks the “Upload” button.
5. Viewer sees a loading screen informing them their song is being uploaded. Shortly after a message box is displayed that their song has uploaded.
6. User is bought to the page for the song that they have uploaded.

Alternative Flows:

Incorrect link/no data provided

1. User provides an incorrect link or does not provide the required song name in step 3.
2. User clicks on the upload button.
3. A message is displayed to the user that they cannot continue until they have corrected the mistake.

### **Bandcamp Use Case 1.5** Listen to Song

The user wants to listen to a song. The user has already logged on and the song already exists in the database.

1. The user clicks on search and types in the name of the song they want to listen to. A list of all songs containing the words typed into the search bar will be returned. If there are no songs which contain these words the returned search will be empty and a message will be displayed that no songs match the search criteria. Otherwise the user clicks on the song they want to listen to and are bought to that songs page. The user can click on the play button to begin playing it or the stop button to stop the song. At any stage while the song is playing the user can click on a link to go to a different page. This will cause the song to stop playing.
2. The user clicks on search and types in the tags for the song they want to listen to. A list of all songs containing these tags will be returned. If there are no songs which contain all these tags the returned search will be empty and a message will be displayed that no songs match the search criteria. Otherwise the user clicks on the song they want to listen to and are bought to that songs page. The user can click on the play button to begin playing it or the stop button to stop the song. At any stage while the song is playing the user can click on a link to go to a different page. This will cause the song to stop playing.
3. The user clicks on the page of the user who uploaded the song they want to listen to. They click on "Uploaded Songs". This will display all songs that this user has uploaded. They click on the song they want to listen to and are bought to that songs page. The user can click on the play button to begin playing it or the stop button to stop the song. At any stage while the song is playing the user can click on a link to go to a different page. This will cause the song to stop playing.

**Bandcamp Use Case 1.6 Tag a song**

The user assigns a tags to a song from a list of available tags. The user adds all the mandatory fields for the song. The system verifies if the user has logged into the system, records the users tag and song details with the song he/she wants to upload and informs the user with all the details associated with the song on successful upload.

### **Bandcamp Use Case 1.7** Make a comment

An user wants to make a comment on a song/event. The user has already logged into the system, the song/event already exists in the the database, selected a song/event, and the user has the read & write rights to the song/event.

**Possible scenarios:**

1. The user searches a song by name they want to comment. A list of all the songs of this title will be returned. The user clicks on the one they want to comment to and are brought to that songs page.
2. The user searches a song by a tag name they want to comment. A list of all the songs containing these tags will be returned. The user clicks on the song they want to comment to and are bought to that songs page. The user enters comment and clicks on the comment button to submit it.
3. The user clicks on the page of the user who uploaded the song they want to comment to. He/She clicks on "Uploaded Songs" and click on the song that they want to comment to.

**For each scenarios:**

The user enters comment and clicks on the comment button to submit it. A user can "hashtag" another user by writing '@' followed by an user name in the comment. The system records the users comment, displays it on the UI and notifies the song uploader. If another user were tagged, they are also notified.

**The things that could go wrong are:**

1. The song the user searches for does not exists, in which case the user exits;
2. No songs are returned with the tags that the user has included, in which case the user exits;
3. There is an error in writing to the database, in which case the system informs the user with this, with a notice that to contact Technical Support.

**Bandcamp Use Case 1.8: Member follows another Member**

Follow Member

**Primary Actor:** Member

**Level:** User Goal

**Main Success Scenario**

1. Member visits the page of a member whom they are not yet following.
2. Member clicks “follow” button.

### **Bandcamp Use Case 1.9:** Member gets notice of viral song (Casual)

Viral Song Notice (Casual Version)

There are three ratings a member can give any song, “Yah”,“Blah”, or “Nah”. Yah corresponds with a positive review, “Blah” means it is neither good nor bad, and “Nah” indicates a member does not like the song. If a song gets over 200 “Yah’s” in one day or 1000 in a week, it is considered to be viral.

The song will keep the viral status for one week. A member has a feed on their home page. When a member uses the system, any songs that are viral will appear in their feed one time.

**Bandcamp Use Case 1.10 Populate Member Home Page with Top 20 Tags**

Top 20 Tags

**Primary Actor:** Member, System

**Level:** Subfunction

**Precondition:** Member is logged in. Member has rated multiple songs

**Main Success Scenario**

Profile owner has given a “Yah” rating to many songs.

1. Member visits a profile page (their own or someone else’s)
2. The system counts up the total for every tag that is used in songs that profile owner has rated.
3. The top 20 most popular tags are displayed beside that profile owner’s name and profile image.

**Alternate Scenario**

Profile owner has not rated any songs

1. The system finds no song ratings given by the profile owner.
2. Profile name and Image is displayed but no song tags are displayed on the profile page.

**Alternate Scenario**

Profile has more than one, less than twenty song tags

1. The system finds less than 20 song tags associated with songs the profile owner has given a “Yah” rating to.
2. Profile name and Image is displayed with only the tags available.

**Post condition**

As the member gives more “Yah” ratings to more songs, the top 20 tags are continually recalculated.

### **Bandcamp Use Case 1.11** Create Account(Brief)

**Actors:** Registering User, Account System

**Description:**

When a registering user opens the system they will submit their details, accept terms and conditions, and click the ‘Sign Up’ button.

### **Bandcamp Use Case 1.12:** Search For all Songs with Matching Title or Matching Artist or Genre

**Actors:** Member

**Scope:** The overall search process utilised by a member

**Level:** User goal

**Precondition:** Member is logged into system

**Post-condition:** Returns matching results or notifies member none were found

**Main Success Scenario:  
Steps:**

1. Member types song title or Artist name or genre into search function.
2. Member submits search.
3. System looks for songs matching criteria.
4. All matching songs, their artists and genres are displayed to member.

**Alternative Scenarios:  
Steps:**

* 4a1. No match is found
* 4a2. Message informing user no match was found is displayed.

**Priority:** High

**Frequency of Use:** High

**Open Issues:**

Textual Analysis

* Artist (this is a role a member can perform)
* Title - This should have been included in first iteration. It was decided this would not be made a class but rather an attribute of song.
* Song
* Member
* Title
* Genre
* Name - This should have been included in first iteration. It was decided this would not be made a class but rather an attribute of member

### **Bandcamp Use Case 1.13:** Create An Event(Fully Dressed)

**Actors:** Registered User, Event System

**Goal in Context:** Registered user creates an event through the system.

**Scope:** A music event system

**Level:** User goal

**Stakeholders and Interests:**

Registered User - Wants to create a music event as an event page with the system.

Registered Users - Wants to have access to music event pages.

**Pre Conditions:** Database is accessible

**Post Conditions:** None

**Steps:**

1. **Registered User:** Clicks the ‘Create an Event’ button.
2. **Registered User:** Inputs event location details.
3. **Event System:** Validates user input.
4. **Event System:** Prompts user to confirm event creation.
5. **Registered User:** Confirms event creation.
6. **Event System:** Creates event preview for the registered user to submit.
7. **Registered User:** User visually checks the preview and submits the event creation.
8. **Event System:** Creates event in system.
9. **Event System:** Adds a record of the event into the database.
10. **Event System:** Redirects registered user to the new event page they created.
11. **Event System:** Shows a popup dialog suggesting the registered user adds event members and/or songs.

## Reason for choosing First Iteration Use Case

Search function lies at the core of our system. Therefore, we thought it offered the most value to the end user. As our iterations show, there is a natural evolution that occurs with this functionality.

## Iteration Plan

In the first iteration, we will search by song title and return the first song if found.

In the second iteration, we will expand the search to include song artist.

In the final iteration, we will expand the search to include genres. All the songs found will be returned.

## First Iteration

### [**Bandcamp Use Case 1.13: Search For Music**](http://bit.ly/1J9i1QL)

**Actors:** Member

**Scope:** The overall search process utilised by a member

**Level:** User goal

**Precondition:** Member is logged into system

**Post-condition:** Returns matching result or notifies member none were found

**Main Success Scenario:  
Steps:**

1. Member types song title into search function.
2. Member submits search.
3. System looks for matching song title.
4. First matching song details are displayed to member.

**Alternative Scenarios:  
Steps:**

* 4a1. No match is found
* 4a2. Message informing user no match was found is displayed.

**Priority:** High

**Frequency of Use:** High

### Textual Analysis

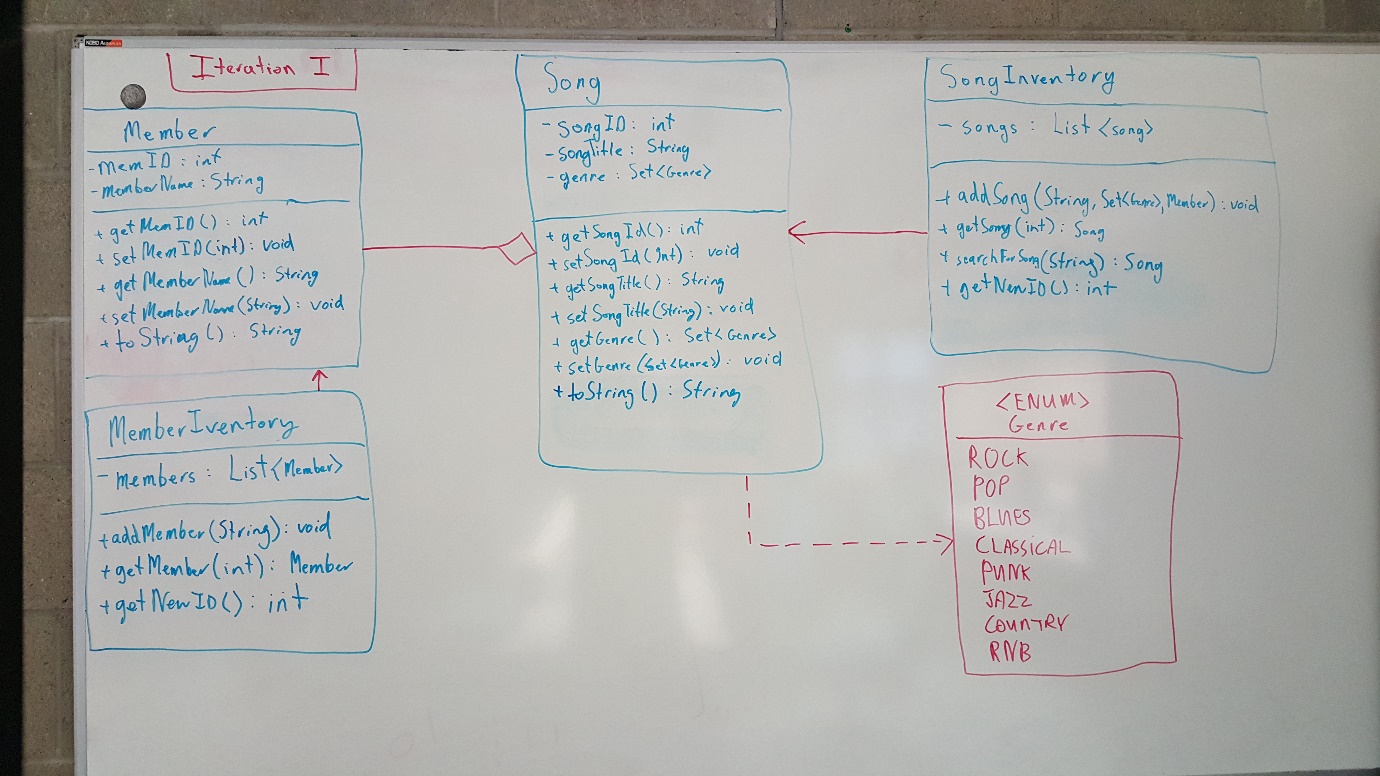
Nouns:

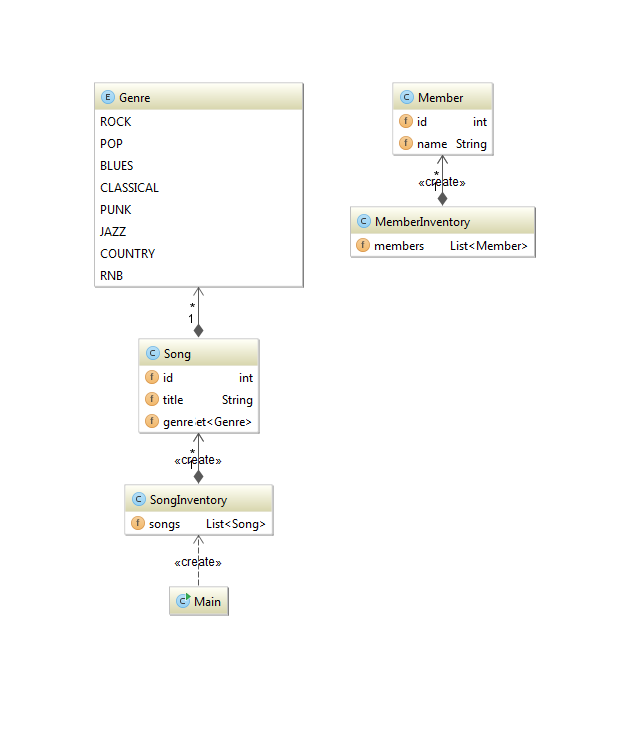
* Member (Person) – made into a class
* Song (Thing) – made into a class

Verbs:

* Search – made into method

### Domain Model





### Link to Code:

http://bit.ly/1J9i1QL

## Second Iteration

### [**Bandcamp Use Case 1.13: Search For Song by Matching Title or Artist**](http://bit.ly/1O2UG4x)

**Actors:** Member

**Scope:** The overall search process utilised by a member

**Level:** User goal

**Precondition:** Member is logged into system

**Post-condition:** Returns matching result or notifies member none were found

**Main Success Scenario:  
Steps:**

1. Member types song title or artist name into search function.
2. Member submits search.
3. System looks for matching song titles.
4. First matching song details are displayed to member.

**Alternative Scenarios:  
Steps:**

* 4a1. No match is found
* 4a2. Message informing user no match was found is displayed.

**Priority:** High

**Frequency of Use:** High

### Textual Analysis

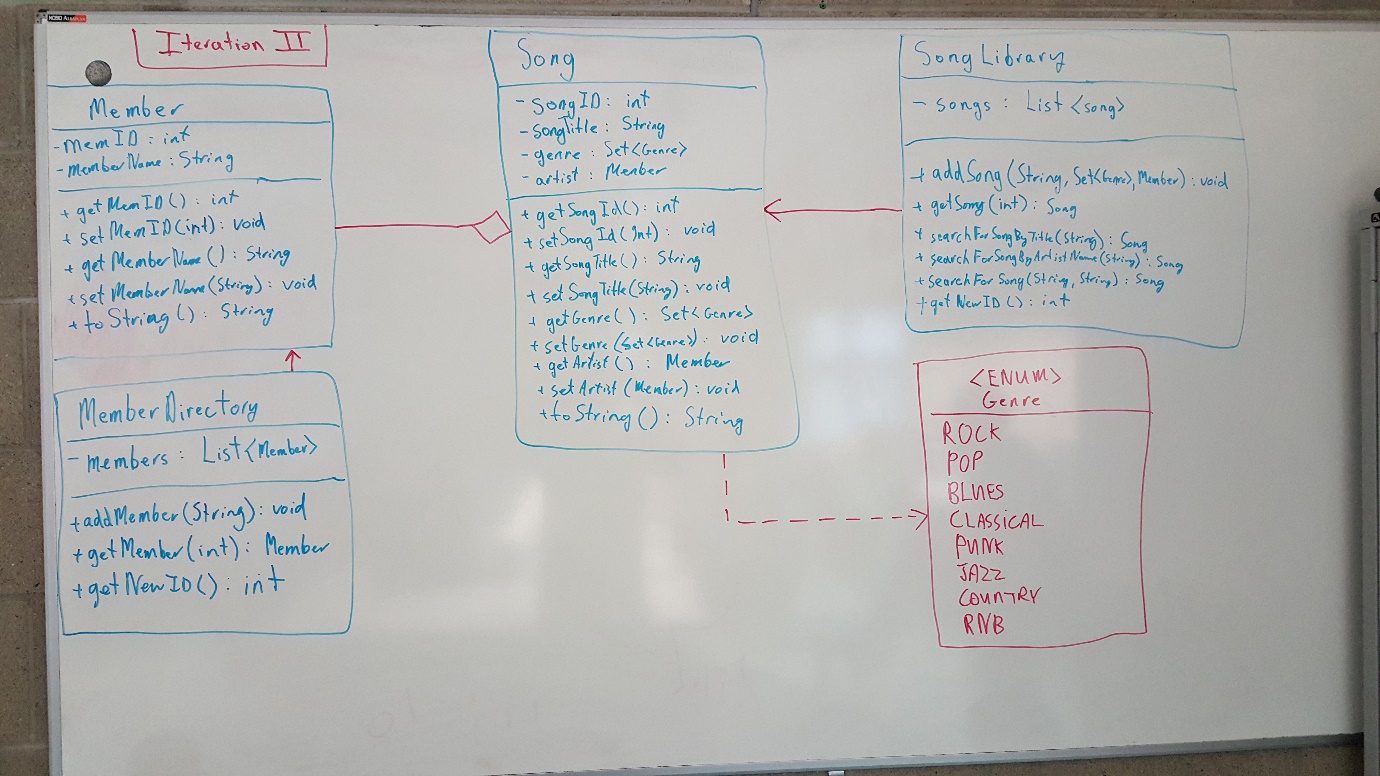
Nouns:

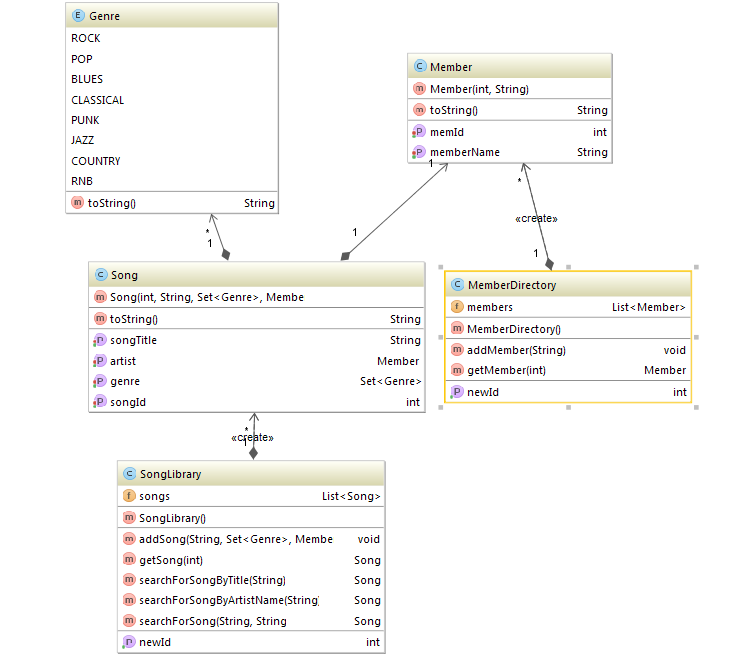
* Member (Person) – made into a class
* Song (Thing) – made into a class
* Title (Thing) – instead of making a class, it is an attribute of Song
* Artist (Person) – role Member performs

Verbs:

* Search – made into method

### Domain Model





### Link to Code:

<http://bit.ly/1O2UG4x>

## Third Iteration

### [**Bandcamp Use Case 1.13: Search For all Songs with Matching Title or Matching Artist or Genre**](http://bit.ly/1QqxS5g)

**Actors:** Member

**Scope:** The overall search process utilised by a member

**Level:** User goal

**Precondition:** Member is logged into system

**Post-condition:** Returns matching results or notifies member ‘song not found’

**Main Success Scenario:  
Steps:**

1. Member types song title or Artist name or genre into search function.
2. Member submits search.
3. System looks for songs matching criteria.
4. All matching songs, their artists and genres are displayed to member.

**Alternative Scenarios:  
Steps:**

* 4a1. No match is found
* 4a2. Message informing user no match was found is displayed.

**Priority:** High

**Frequency of Use:** High

### Textual Analysis

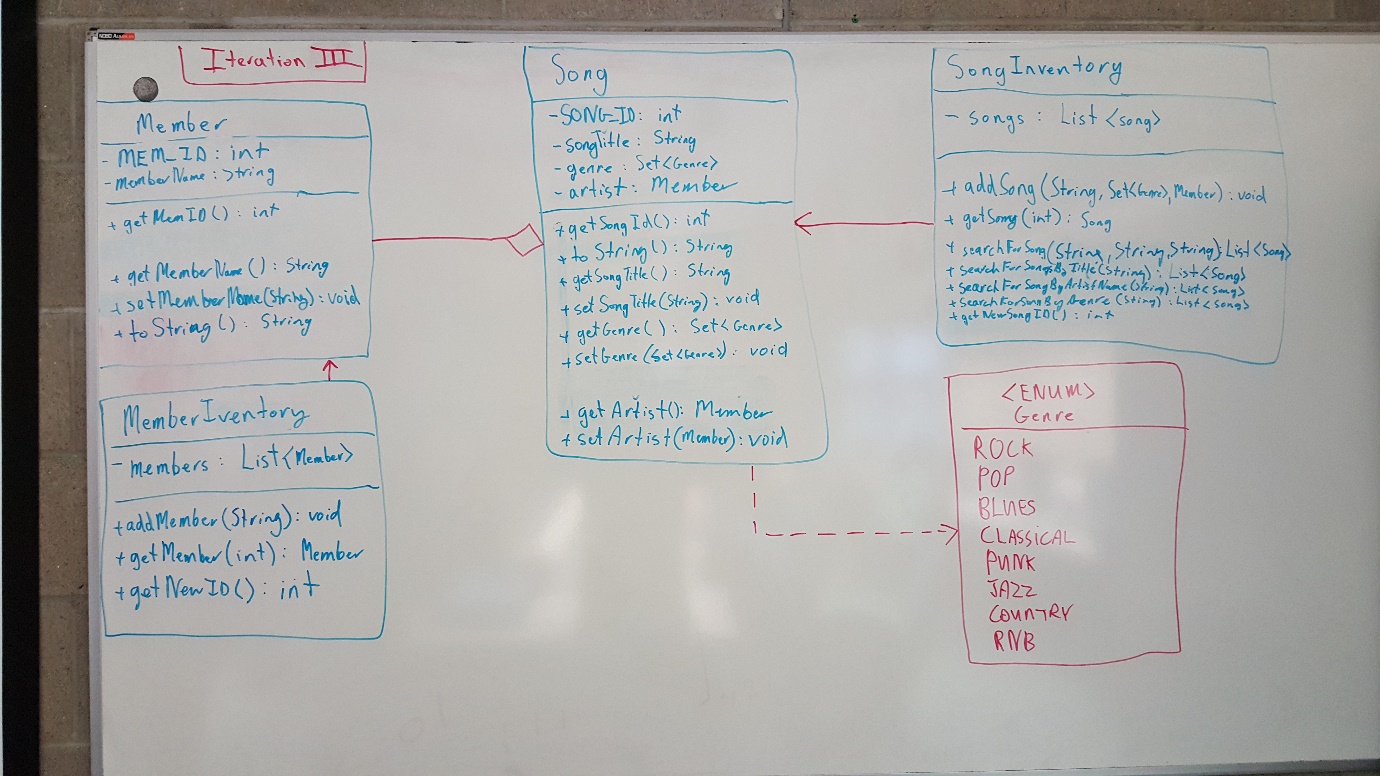
Nouns:

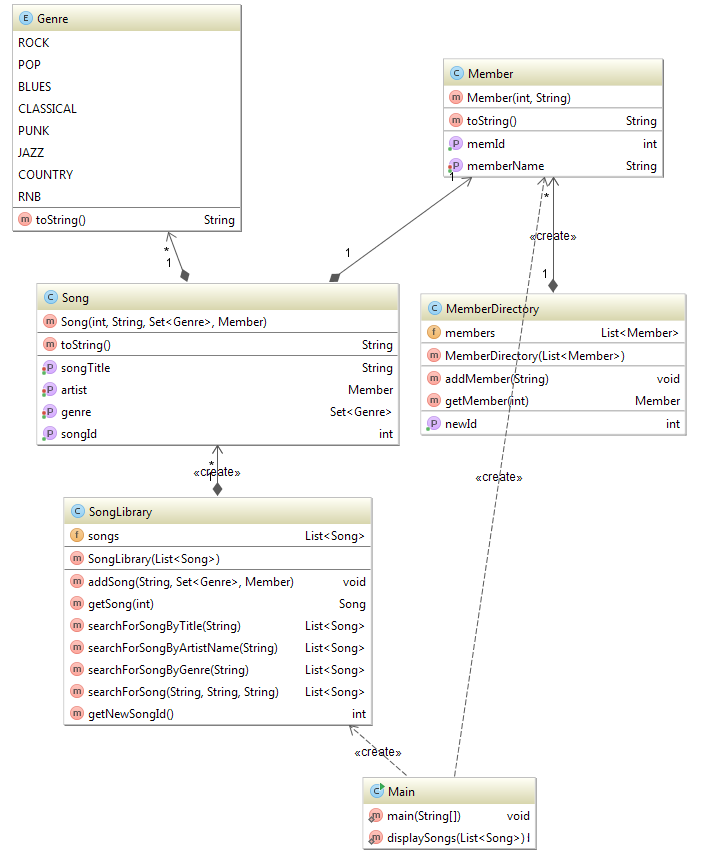
* Member (Person) – made into a class
* Song (Thing) – made into a class
* Title (Thing) – instead of making a class, it is an attribute of Song
* Artist (Person) – role Member performs
* Genre (Thing) – we created an Enum to keep genre type value safe

Verbs:

* Search – made into method

### Domain Model





### Link to Code:

<http://bit.ly/1QqxS5g>

## Conclusion

Due to the iterative nature of the project we felt the work could not be assigned to individual members of the team. In all cases, we worked in pairs.

By doing all the use cases at the start, it became apparent, as to which use cases might lend themselves better to the iterative process. It was also then clear what uses cases would be more suited to be either brief, casual or fully dressed.

We found the textual analysis very beneficial for developing a starting point.

Creating the UML Diagrams on the whiteboard first gave us a platform to debate and discuss important design decisions before starting to code. We feel this not only led to implementing better code but also sped up the development process.

Using GitHub was a key-part of our development plan. It is as integral to our development as our IDE choice.