The Muze Music Library System RELEASE 2

Design Documentation
Prepared by Team 4:

- Brandon Montijo <bcm8504@rit.se>
- Nikhil Raina <nxr5013@rit.se>
- Chase Marino <cam4555@rit.se>
- Alex Tackett <ajt6798@rit.se>
- Qunzhan Huang<qxh2404@rit.se>

Summary

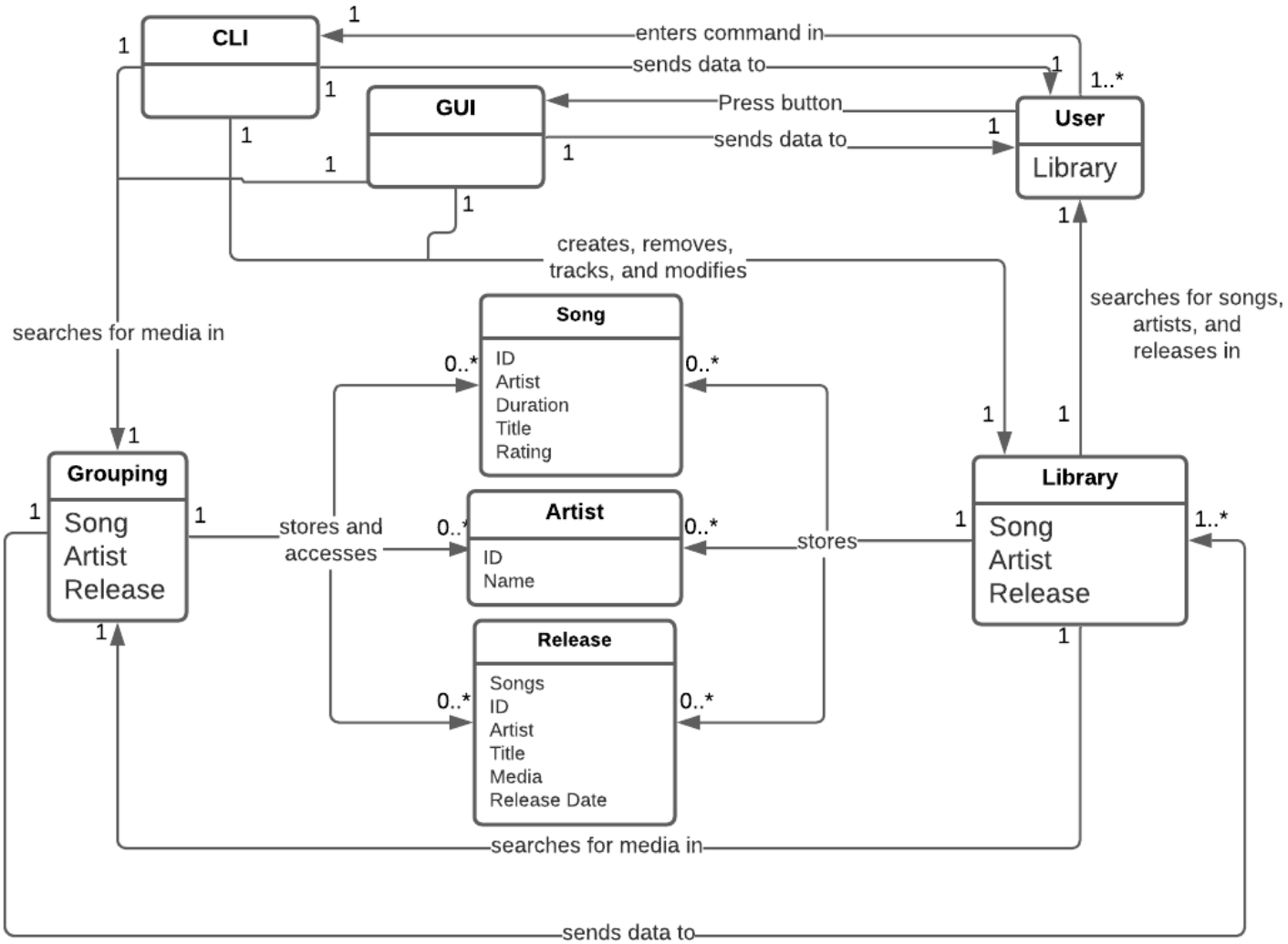
The Muze Music Library System (MMLS) provides an application for music aficionados to track songs in their respective collection. Users can peruse the database via a command line interface for songs by title, artist, or album. They have the ability to create multiple collections in their music library. Each collection allows them to add songs or remove them from there. They can also optionally rate individual songs on a 1-5 star basis. Users have the option to search through the database and library by entering the song title, artist or the album. They can also list their collection by title, artist, or album alphanumerically.

Key system responsibilities are as follows:

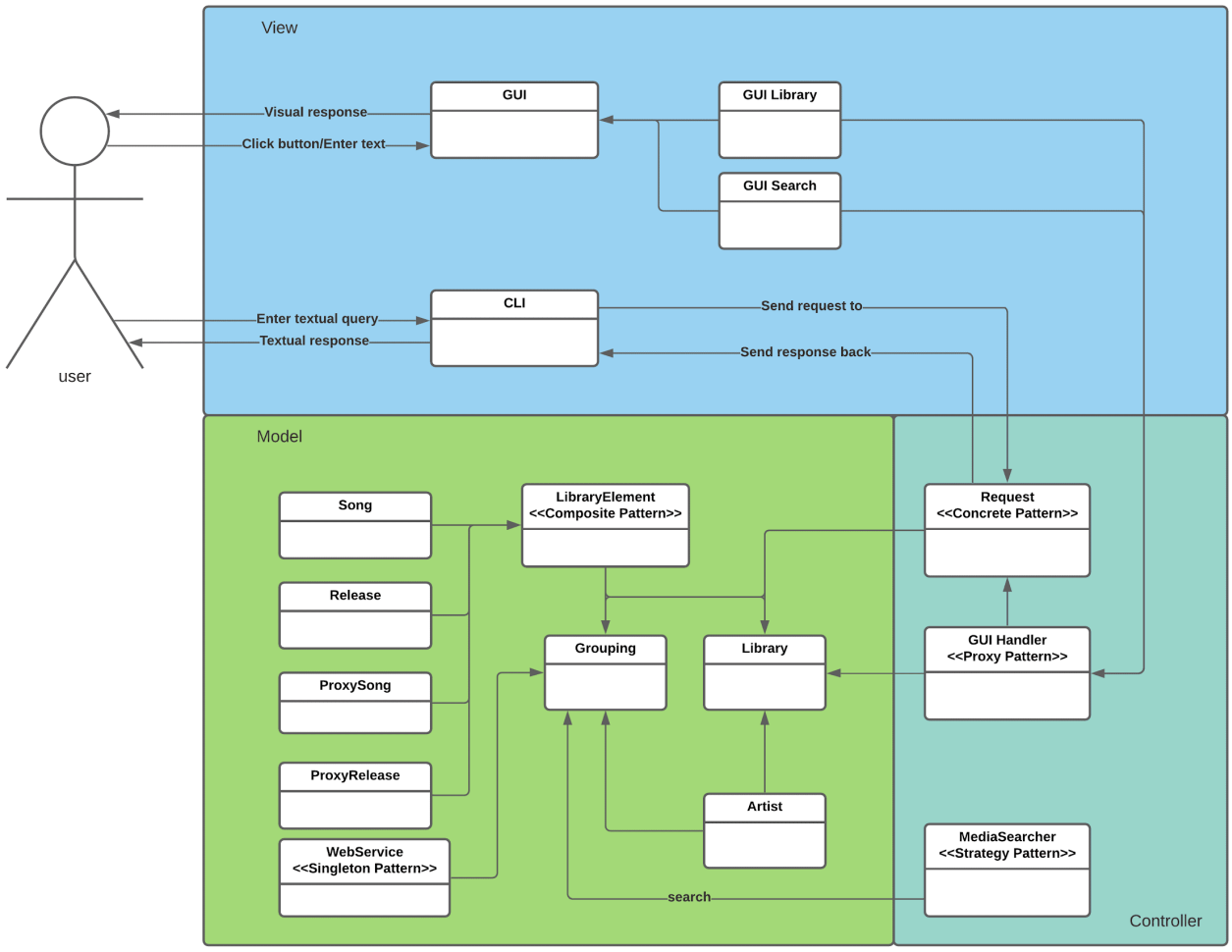
1. The system provides an interface where users can enter basic single-line requests. Requests are handled and provided with a textual response. If needed, users can list all command possibilities using a help request.
2. The system provides a searchable database of all available music passed by the csv files. It responds to queries sent by the user based on what information the user requests for: whether it be artist, song, or release.
3. The system tracks and persists the user's personal library. It allows users to add songs or releases to their collection based on specific queries.

Domain Model

The domain model shown below roughly describes the Muze Music Library System that has been implemented for this project.



System Architecture

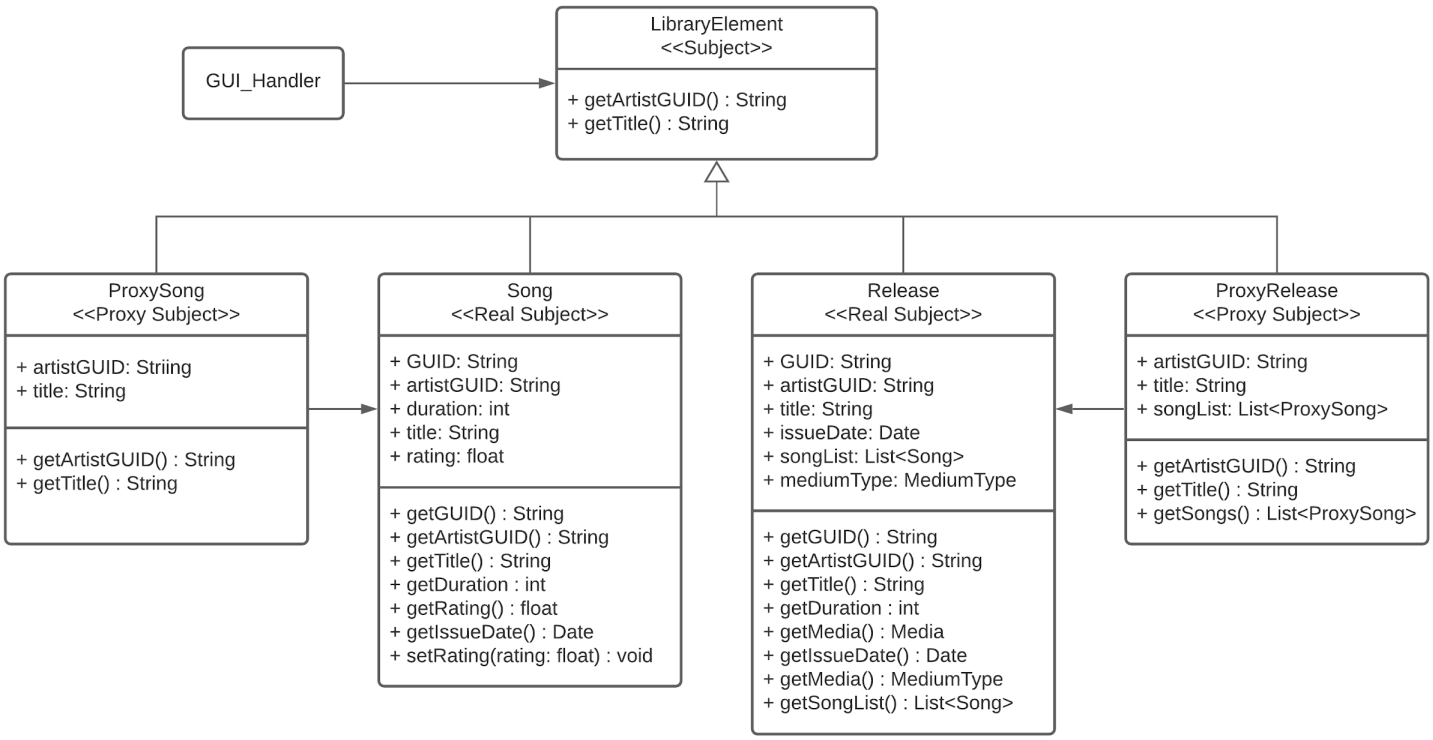


The diagram above shows the system architecture of the MMLS, The highest level design is the Model-View-Controller architecture. This was chosen because of the constant interaction with the user, who may access and manipulate their results from outside the system. The system is text-based so the view will only be printed in the terminal which is processed in CLI . The user interacts primarily with the Command Line Input. Through implementation of the RequestHandler, this handles all of the user input. The controller contains two subsystems, one implements requests following the command pattern and the other following strategy pattern. The model contains data and associated logic, but does not interact with data sent to the user.

Subsystems

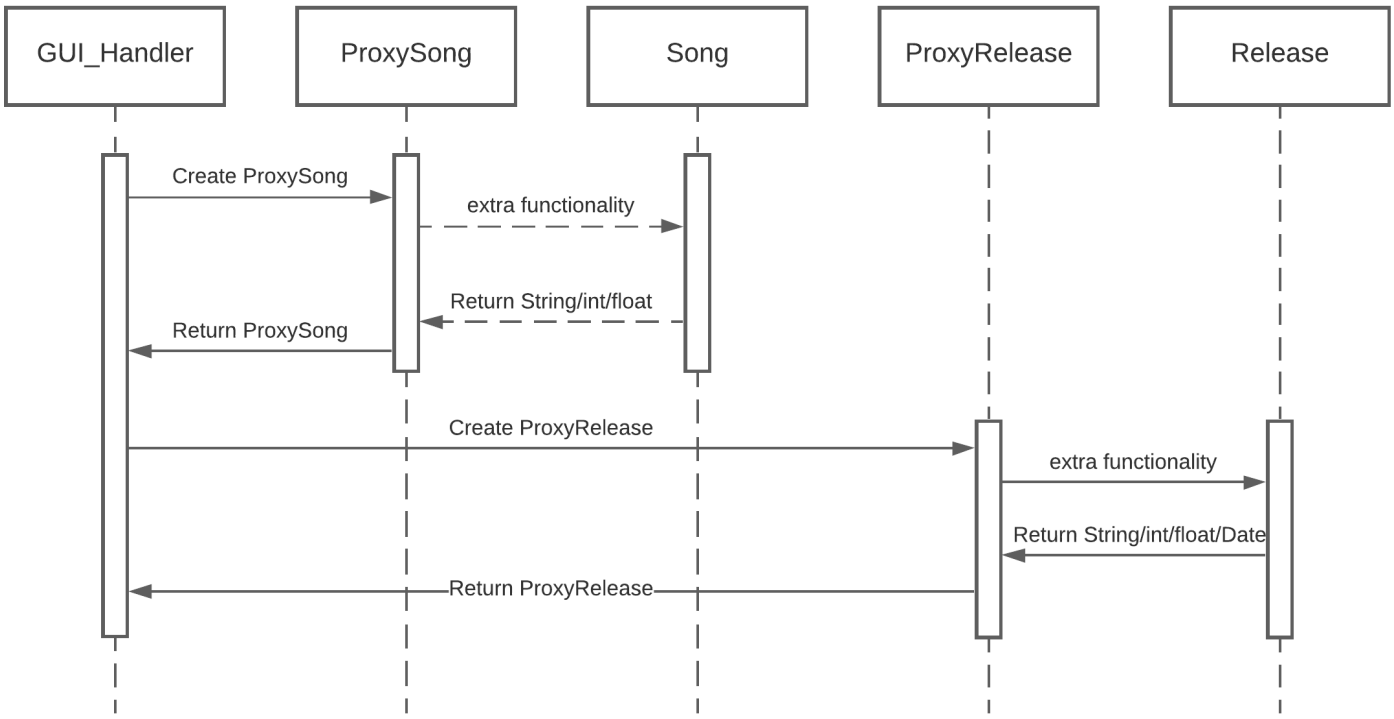
Replacing Expensive Items on GUI with Proxy Pattern

Class Structure Diagram



We decided to implement the proxy pattern, and more specifically a virtual proxy, inside of our GUI side of the application. Since both the Song and Release objects are expensive to create and maintain, i.e. are objects that take up a significant amount of storage space, we decided to create proxies for each. The song proxy class stores only the title and artist guid of a song as opposed to what's shown in the song class. The release proxy class stores only the title, artist GUID, and the song list inside of the release, as opposed to the values shown in the release class. This way, the GUI Library class can easily access the proxy songs and releases to save storage space and streamline the program's execution, increasing efficiency.

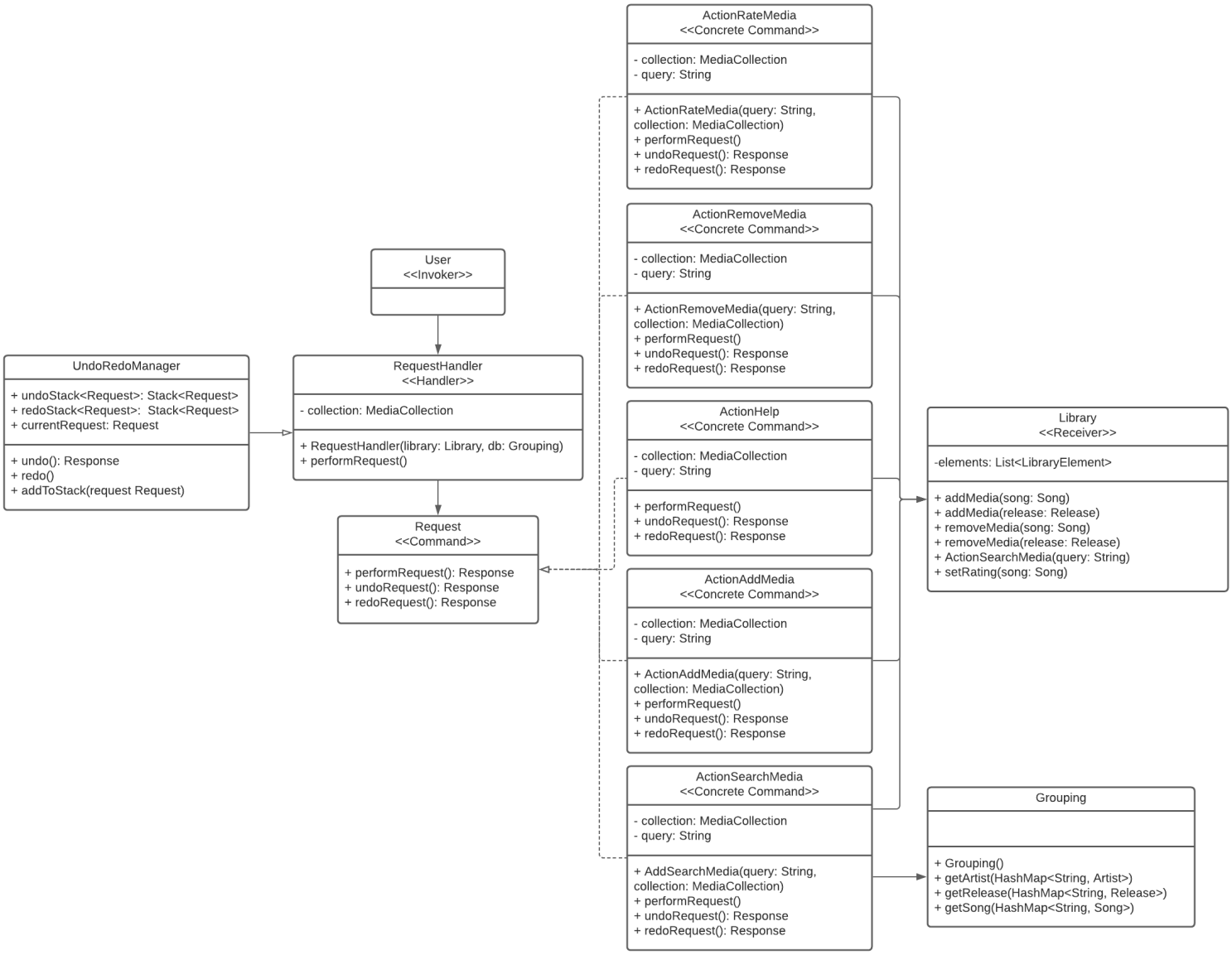
Sequence Diagram



The GUI Handler class creates proxy songs and releases as placeholders based on user requests from either song searches or song additions. If need be, the song or release will extend to the originating “real” class to reach for extra functionality. For example, the proxy song cannot be rated, so it uses the real class to use the functionality rate song. Basically, when the user requests for the actual object by clicking the update button in the library panel, the system creates the real objects for each song or release in a user’s library.

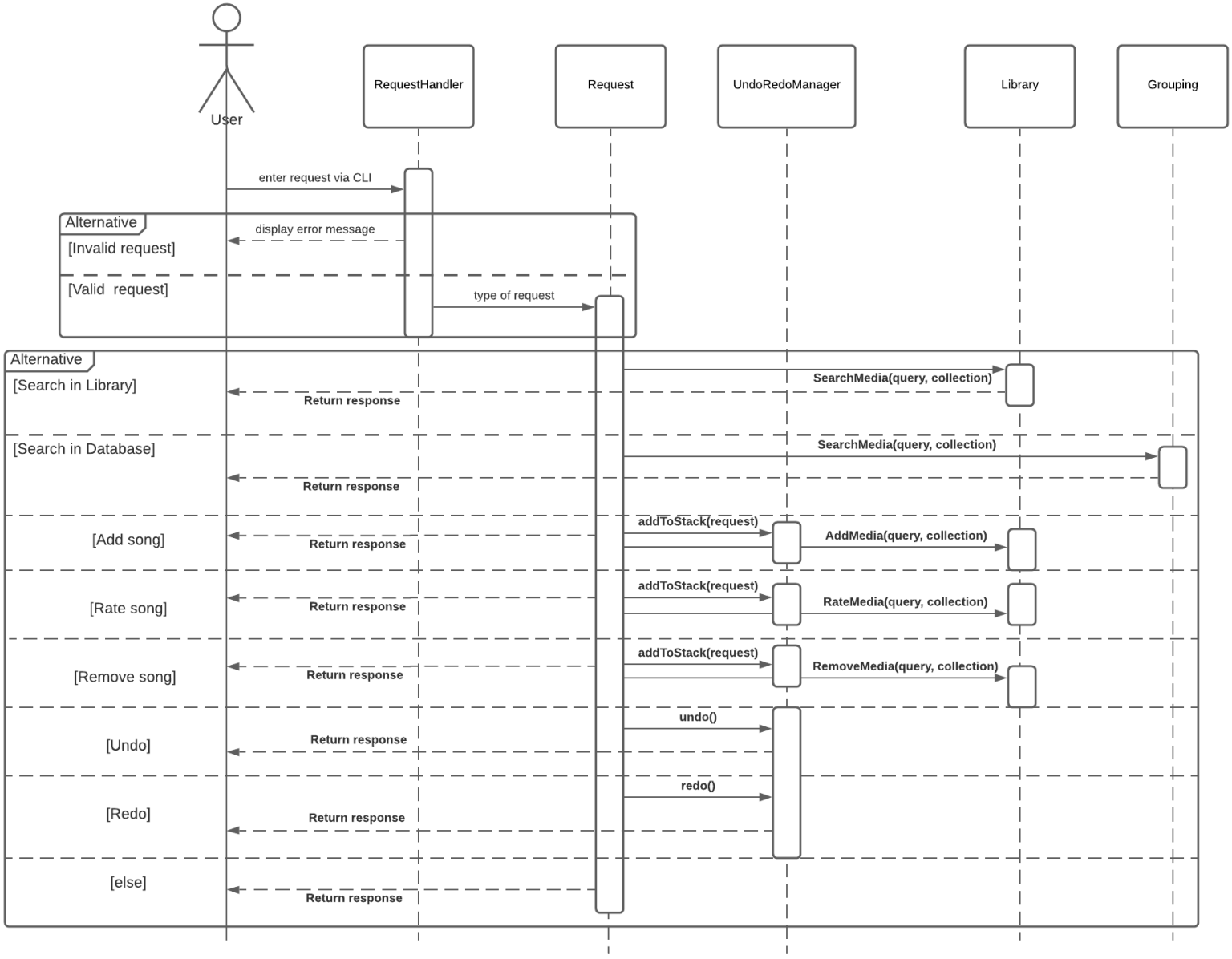
Handling the General UI with the Command Pattern

Class Structure Diagram



We selected to implement the Command design pattern here as it is the best structure available for us that handles requests based on user input. Our request, taken by the CLI, is encapsulated as an object, which is then sent to the RequestHandler. This class delegates it to the proper Action class. From there, the action is executed, and the user is notified of its success or failure through a Response type message displayed in the command line.

Sequence Diagram



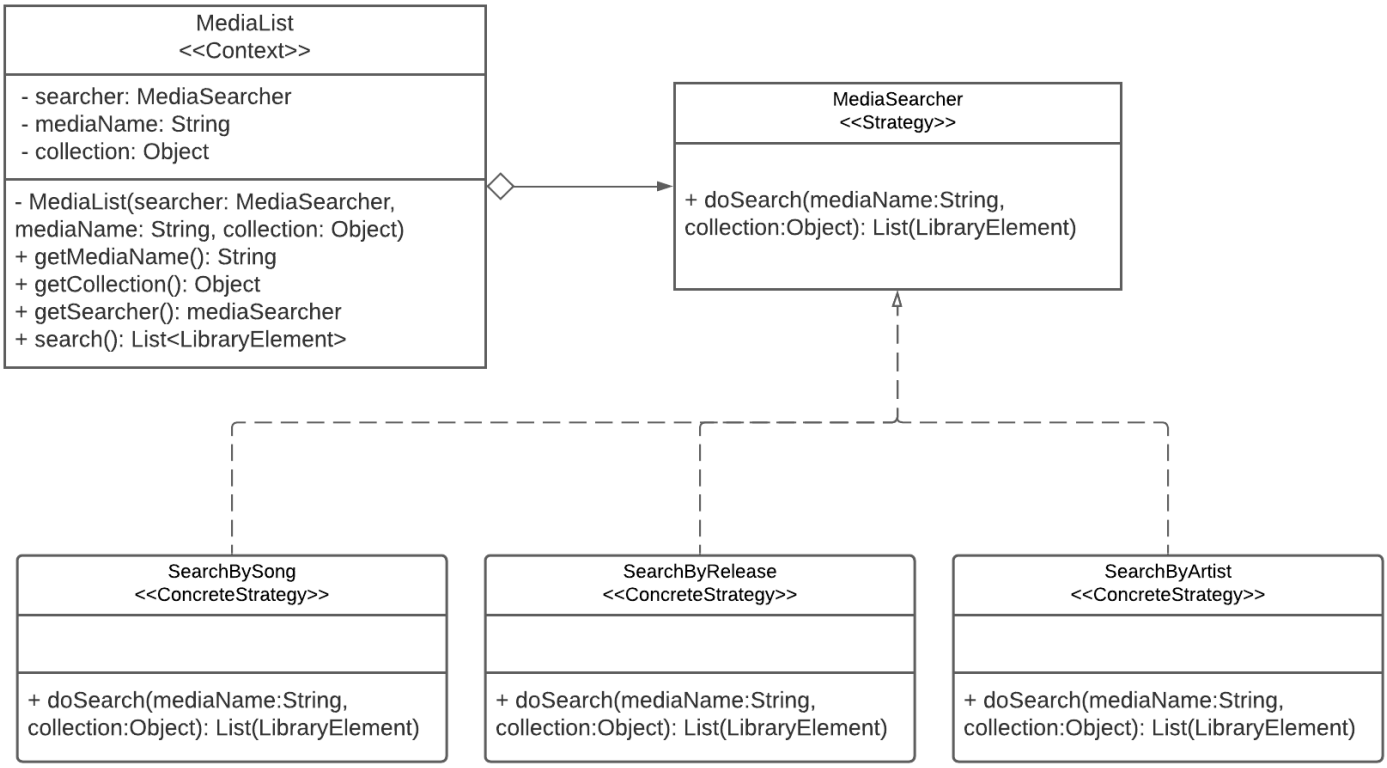
When the user enters a request in the command line interface of CLI, it invokes the method on Request, Request will select the specific command in concrete command. Eventually, the command will be executed in the user Library or the Grouping. Command design pattern was used to create a flexible, changing design to add or remove commands from the user's available requests. Using the Command in this case also preserves the Open/Closed principle by allowing the us to add or remove commands without modifying the code.

Gof Pattern Name: Command		
Participants: RequestHandler, musicCollection, ActionAddMedia, ActionRemoveMedia, ActionRateMedia, ActionHelp, ActionSearchMedia, ActionCreateCollection		
Class	**Role in Pattern**	**Participant's Contribution in the context of the application**
User	Invoker	The User client takes in the commands from CLI, It invokes RequestHandler.
RequestHandler	Handler	The RequestHandler class handles the request from the user, parses the request, holds partial responses, and sends the parsed data to the correct command.
UndoReoManager	Helper	Helper for RequestHandler. Contain two stacks, undo stack and redo stack. Handle the undo and redo request, invoke the undo and redo methods in ActionAddMedia, ActionRemoveMedia and ActionRateMedia
Library	Receiver	The Library is able to receive data and commands that are parsed by RequestHandler, then process the request and update itself based on the given command.
Grouping	Receiver	The Grouping only associates with ActionSearchMedia, user can only do search in the Grouping of MMLS.
Request	Command	Defines the interface for a user action in the MMLS. The request performed method is invoked each time that the command should perform its related task(add, remove, rate...etc).
ActionAddMedia	Concrete Command	A concrete command that performs an add operation, this command allows the user to add media to their collection. Able to perform undo.
ActionRemoveMedia	Concrete Command	A concrete command that performs a remove operation,this command allows the user to remove media to their collection. Able to perform undo.
ActionRateMedia	Concrete Command	A concrete command that performs a rate operation. This command allows the user to rate individual songs.Able to perform undo.
ActionHelp	Concrete Command	A concrete command that performs a help operation. When the command is called, the Instructions about commands will be printed.
ActionSearchMedia	Concrete Command	A concrete command that performs a search operation. this command allow user to search specific object in their

		collection or MMLS
Deviations from the standard pattern: None		
Requirements being covered: R1 functional requirements: ● 1.3. The system shall provide a help request, the response for which will be a list of available requests and how to use them. ● 3.1.The system shall allow the user to add an individual song or a release to their music collection ● 3.2. The user may optionally rate individual songs from 1 to 5 stars. ● 3.3. The system shall allow the user to search their personal library. ● 3.5. The user may remove individual songs or releases from their collection		

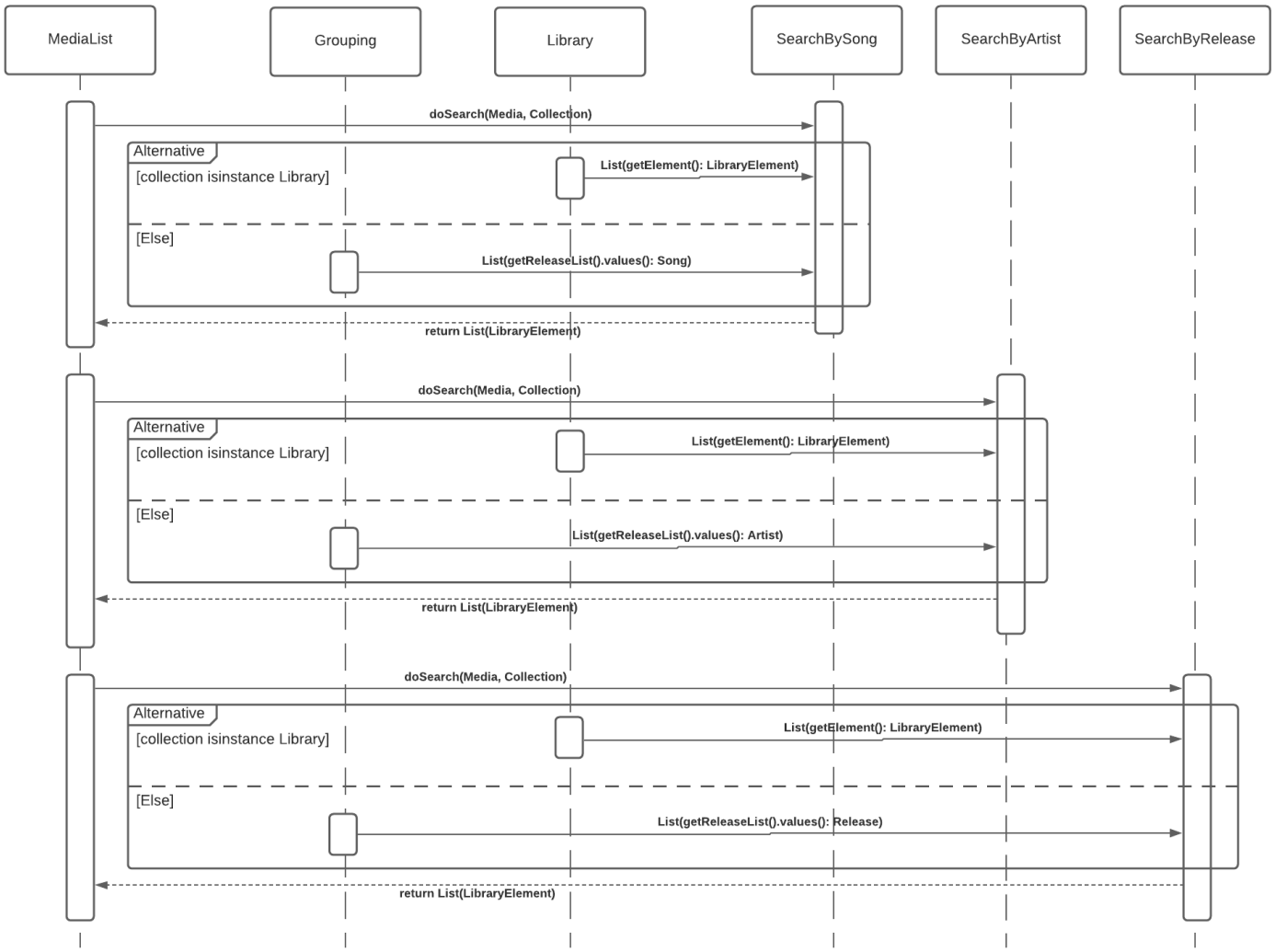
Dynamic search architecture with the Strategy Pattern

Class Structure Diagram



The implementation of our Strategy Pattern revolves around search and exploring the different types of search that the user can perform. As a requirement, the user should be able to search for songs, artists and releases. The three concrete classes of this pattern are the possible ways to accomplish the kind of search that is required. Each class will return a list of Media that will then get displayed to the user. It could be a single song enclosed in a list, a list of songs produced by an artist or a list of songs in a release.

Sequence Diagram



The sequence diagram above illustrates the conceptual behavior of the Strategy Pattern. The overall method calls and interconnections between the interface and its respective concrete classes were minimal. Since there were independent lists already made for the song, releases and the artist, the search concept became easy and straightforward via the Strategy Pattern. The MediaList acted as the context that would pass the ‘instruction’ for the interface to strategise.

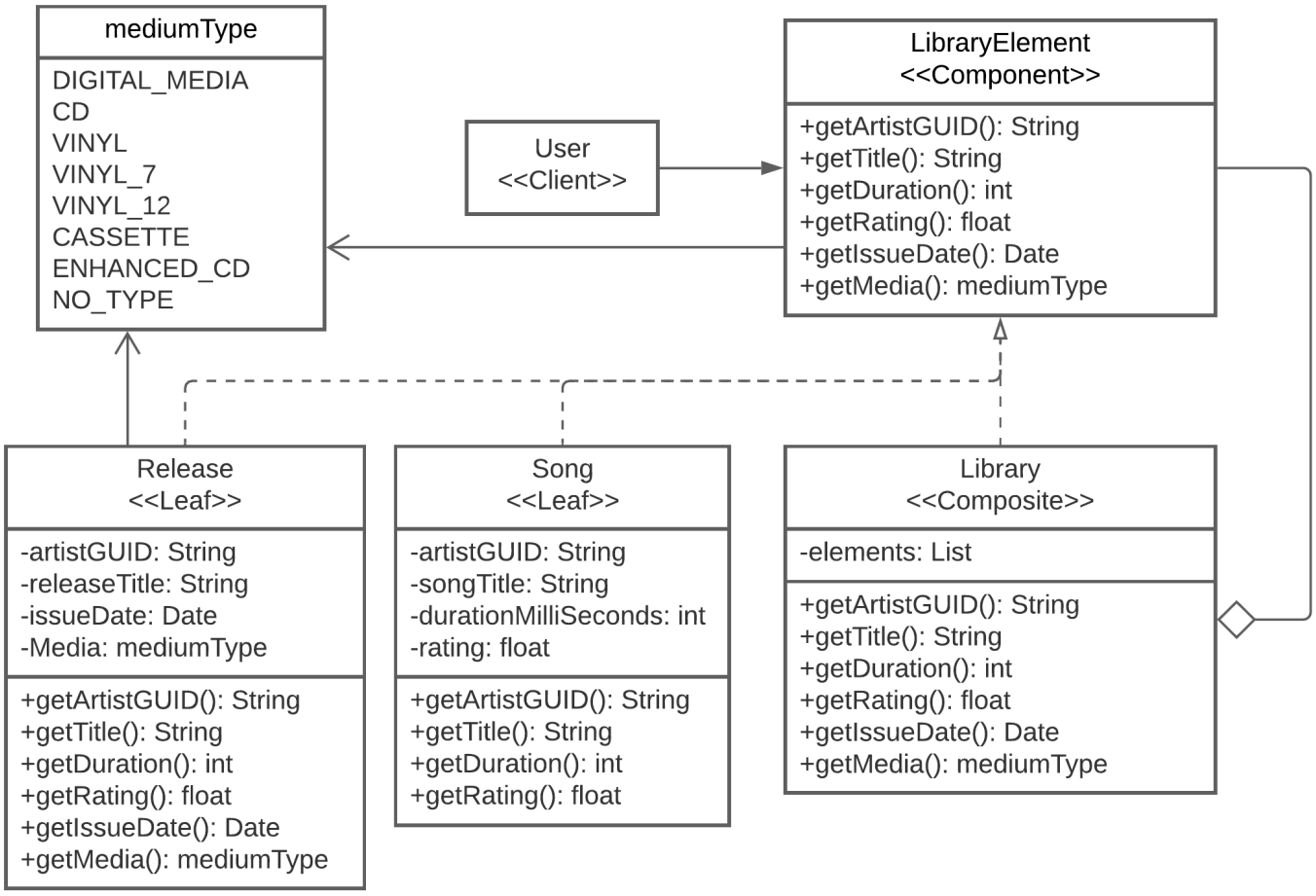
This pattern interacted with the Command pattern wherein the Media that has been added by the user would be sent here to find out what kind of search needs to be done. It would take in the Media entered by the user from the Command Pattern and then pass it to the interface, which implemented our main strategy. This is the mediaList Context that has been mentioned above in the Class structure Diagram. it then passed the search sequence to the interface that would implement the strategy for search. This kind of behavior is possible via abstraction and polymorphism.

After the result from the strategy had been found out, the Main thread would then go down to the respective Concrete Strategy and then execute the Search for the media in the respective domain. The search would result in a list that could either comprise just the song, or a list of songs produced by an artist or a list of songs within a release.

Gof Pattern Name: Strategy		
Participants:		
Class	**Role in Pattern**	**Participant's Contribution in the context of the application**
MediaSearcher	Strategy	This is the interface that will set the type of searching will be needed for the given type of object.
MediaList	Context	This class handles the retrieval of the media from the context and decides what kind of searching strategy needs to be added to it.
SearchBySong	Concrete Command	This handles the search of the entered media in the song category
SearchByRelease	Concrete Command	This handles the search of the entered media in the release category
SearchByArtist	Concrete Command	This handles the search of the entered media in the artist category
Deviations from the standard pattern: None		
Requirements being covered: - Allows users to search for songs, artists and releases in the Grouping - Allows users to search for songs, artists and releases in the specific library mentioned - Allows users to search for songs, artists and releases in the specific collection mentioned		

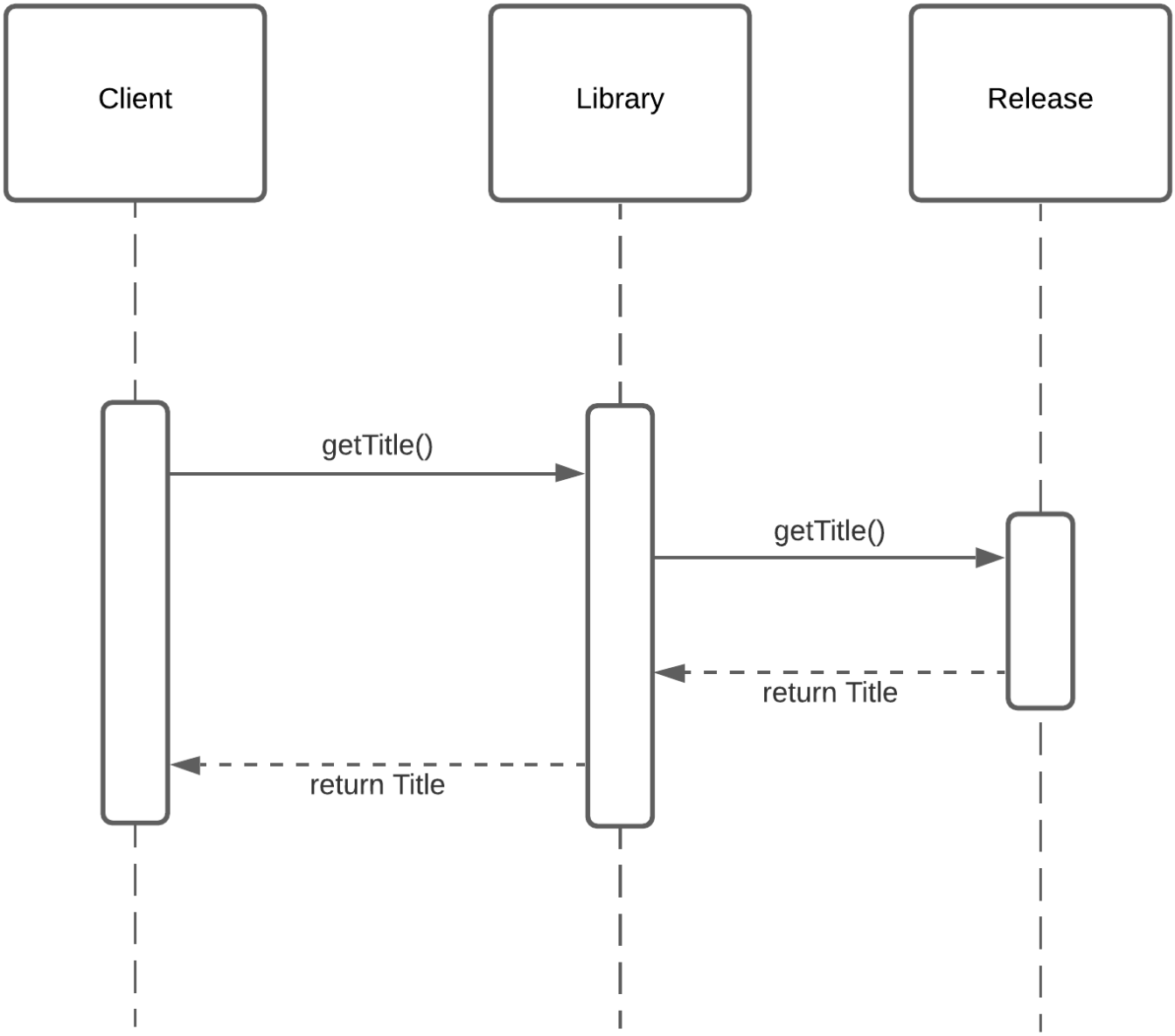
Library and Collections of Media: Composite Pattern

Class Structure Diagram



We chose to use the Composite Pattern, as this pattern is the best structure for dealing with part-whole hierarchies, which is what we have. The libraryElement represents an interface for our two types of data, song, and release, which builds our actual library. This interface helps us create uniformity between our classes. The implementation allows the freedom of easily creating more Leaves in the future if more data types are deemed necessary.

Sequence Diagram

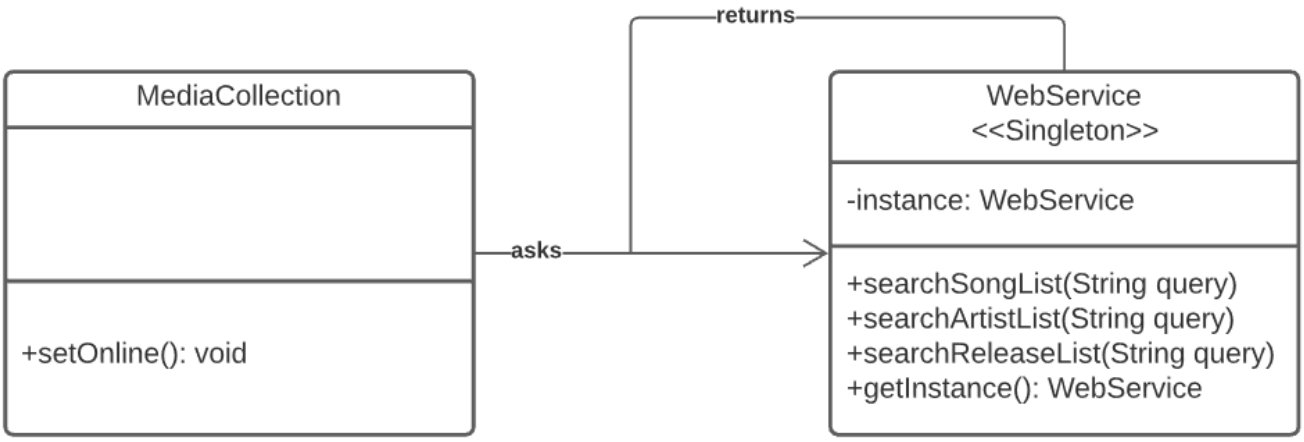


The Client, used to navigate the Library, requests the title of a specific Release. This is sent to the Library, which then implements the request by calling the relevant Release, one of its child elements. The Release fulfills the request by returning its Title. The Library returns this to the Client. The Composite pattern allows any number of child components to be stored within the hierarchy and returned in this fashion; this is the reason why it was chosen for this system. This implementation makes it easy to organize data that other patterns, such as this project's Command pattern implementation, may want to access.

Gof Pattern Name: Composite		
Class	**GoF Participant Name**	**Participant's activity within the pattern in the context of the application.**
User	Client	User interface for browsing the library. It is used to navigate the library. It collects and displays data on the library.
LibraryElement	Component	Defines operations for library elements. This includes getting titles, artists, lengths, ratings, and release dates.
Release	Leaf	Represents a release. Has an artist, rating, release date, title, length, and contains songs.
Song	Leaf	Represents a song. May be contained within a release. Has an artist, release date, length, rating, and title.
Library	Composite	Represents a library. A library may contain any number of songs or releases. It may contain other libraries as a subcollection.
Deviations from the standard pattern: None		
Requirements being covered: **2. The MMLS shall maintain a searchable database of all available music.** **3. The MMLS shall track the user's personal music library.**		

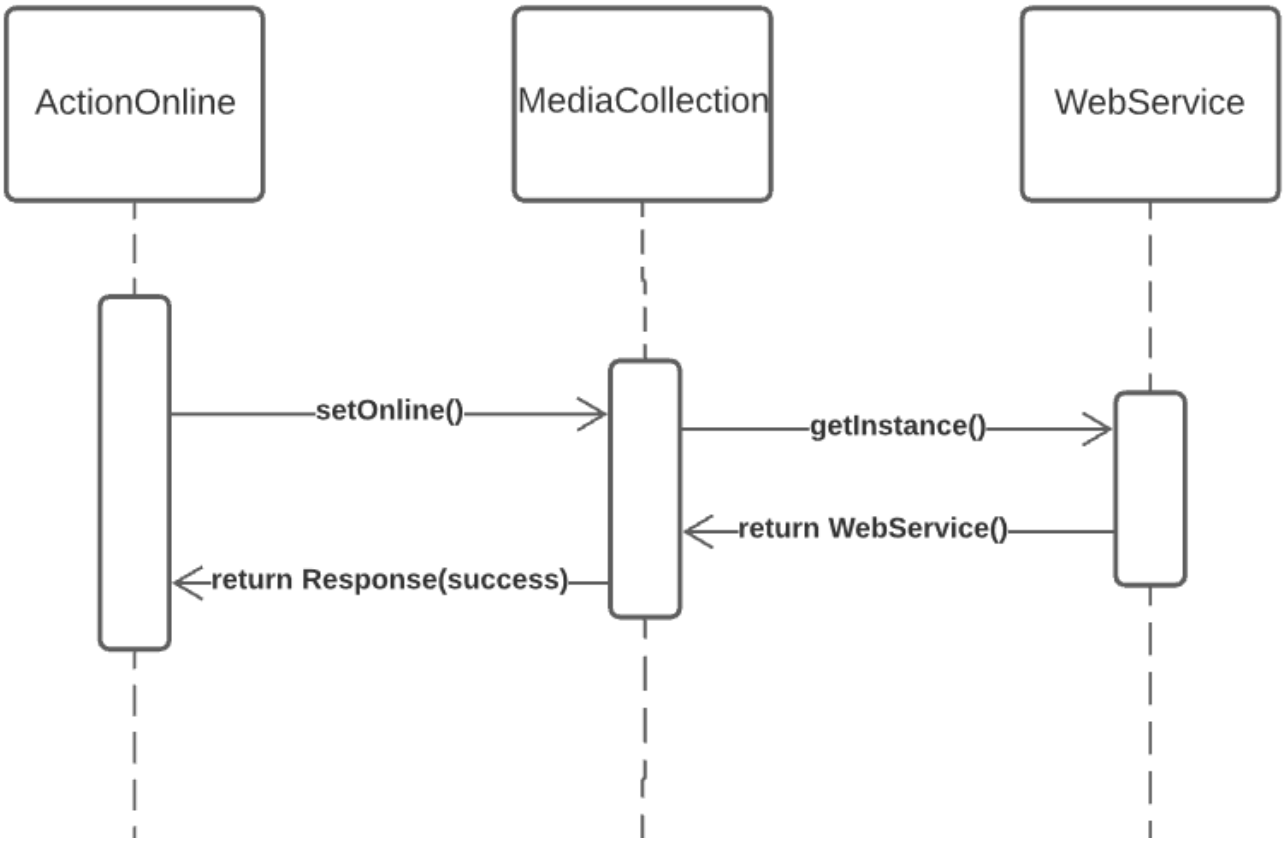
Access the Web Services with the Singleton Pattern

Class Structure Diagram



We chose to use the Singleton Pattern for implementation of WebService, which accesses the MusicBrainz API through JSON queries. The WebService class is used to handle querying MusicBrainz and interpreting the resulting data. The Singleton Pattern ensures that there is only one instance of the object. Since many different classes that aren't necessarily related to the user's media collection could use access to MusicBrainz, and these classes may execute in various sequences determined by user input, it makes sense for a single WebService instance to be easily available to all of these classes.

Sequence Diagram



ActionOnline is a request executed in the CLI or the GUI interface. It calls the setOnline method present in the media collection that has been passed to it. The media collection gets the instance of the WebService and assigns it as the collection's database.

Gof Pattern Name: Singleton		
Class	**GoF Participant Name**	**Participant's activity within the pattern in the context of the application.**
WebService	Singleton	Handles querying data from the MusicBrainz online database. Responsible for converting MusicBrainz objects created from JSON data into a format consistent with the offline database. Has a single instance created on startup that can be accessed from relevant classes.
Deviations from the standard pattern: None		
Requirements being covered:		

Status of the Implementation

For our current implementation, we are able to make all the necessary commands available at our disposal. The user, if followed the way of entering the commands on the CLI, will be able to search within the Grouping class, which can be known as the database for the time being, and in the library for a song and a release. The user will be able to add a song or a release from the database into the Library. The user is able to remove a song or a release from the library. The user is also allowed to remove a song within a release, thereby not needing to remove the entire release itself. The user can ask for help in order to figure out how to write the commands and in what order. The user is allowed to rate a song, but its effect will only occur in the library, provided that song is in the library. We are currently working on an extra feature that will allow the user to create multiple collections within the library. That is an extra effort that hasn't been implemented yet.

The system is now able to retrieve the data of a previous user, provided that the current user was a user who had used the system at some point. The data does get stored in a directory called PersistedData, under the name Libraries.json that holds a list of all the users that have ever used the system in a JSON format.

The largest discrepancy between our domain model and our implementation is the absence of Artist as its own data class. Currently, Artist is treated only as an attribute for songs and releases; this is significantly different from our design, which treats Artist as its own data type. This information is not included in the Artist CSV file provided, despite being in the problem description, so we are unable to implement it at this time. Searching for Artists in the online Musicbrainz database also remains unimplemented due to this problem.

Appendix

Class: csvReader	
Responsibilities: Parses CSV data into different HashMaps that are then allocated in the Grouping. The Grouping class directly calls the public methods of csvReader to access specific data lists so that it can be used as a possible ‘database’ by the system.	
Collaborators:	
Users: Song, Release, Artist	**Used by:** Grouping, Requests
Author: Brandon, Nikhil	

Class: CLI	
Responsibilities: Main class that represents the entire music system. It is the starting point of the system. It takes care of the basic format checking of the command that has been entered via the CLI that gets offered here.	
Collaborators: Library, RequestHandler, Grouping, Library, Response, User	
Users: RequestHandler, Library	**Used by:** ...
Author: Brandon, Nikhil	

Class: RequestHandler	
Responsibilities: Delegates request type to the proper class. It is passed down from the MMLS class and then, depending on the type of command, designates what kind of Concrete Command would take part in the action that the User is trying to perform.	
Collaborators:	
Users: ActionAddMedia, ActionRemoveMedia, ActionRateMedia, ActionHelp, ActionSearchMedia, Request	**Used by:** CLI
Author: Nikhil	

Class: ActionAddMedia	
Responsibilities: Concrete Command that handles requests for adding the media that has been entered by the user. This media is added to the Library provided that it exists within the Grouping list structures.	
Collaborators: Request	
Users: Library, Grouping	**Used by:** RequestHandler
Author: Brandon, Nikhil	

Class: ActionHelp	
Responsibilities: Concrete Command that handles requests for knowing what is the way and the method of entering input to the system. It dictates all the possible ways how a command can be entered, of the type of media and the format in which it is supposed to be entered in.	
Collaborators: Request	
Users:	**Used by:** RequestHandler
Author: Brandon, Nikhil	

Class: ActionRateMedia	
Responsibilities: Concrete Command that handles requests for rating a given media. In this case, it can only be operational with the Song media types. The rating can be done on only the songs that are present within the Library of the user.	
Collaborators: Request	
Users: Library, Grouping	**Used by:** RequestHandler
Author: Brandon, Nikhil	

Class: ActionRemoveMedia	
Responsibilities: Concrete Command that handles requests for removing the media that	

has been entered by the user. This media can be either a song or a release. It also allows removing a specific media within a release so that other songs still remain in the library.	
Collaborators: Request	
Users: Library	**Used by:** RequestHandler
Author: Brandon, Nikhil	

Class: ActionSearchMedia	
Responsibilities: Concrete Command that handles requests for searching the media that has been entered by the user within the Library and the Grouping class. This class acts as a bridge between the command and the strategy pattern. That is, here the context is allowed to instantiate the type of strategy to be used in order to perform the search on the type of media. The search stretches from Artists, Songs and Releases.	
Collaborators: Request	
Users: Grouping, Library	**Used by:** RequestHandler
Author: Nikhil	

Class: ActionShowCollection	
Responsibilities: Concrete Command that handles requests for displaying all the media that is currently in the library of the User. It allows some tab spacing indicating the songs that are within a release and ignores for the ones that are not within a release.	
Collaborators: Request	
Users: Library	**Used by:** RequestHandler
Author: Brandon	

Class: ActionOnline	
Responsibilities: Concrete Command that handles switching the user into online mode.	

This involves setting the user's database to the WebService instance.	
Collaborators: Request	
Users: Library	**Used by:** RequestHandler
Author: Alex	

Class: Grouping	
Responsibilities: Implements the db interface. Takes care of all the data that has been given in the csv files. It does this by taking all the data from the csvParser and then assigns it to specific attributes within this class so that the csvReader only needs to get bothered once, that is when the instance of the Grouping is created. This, in a wholesome level, behaves like a database allowing a storage of different types of data with their data types.	
Collaborators: Response, csvReader	
Users: csvParser	**Used by:** User, MediaCollection, RequestHandler,
Author: Brandon	

Class: Request	
Responsibilities: Interface for requests	
Collaborators: Response	
Users:	**Used by:** RequestHandler, User, ActionAddMedia, ActionHelp, ActionRateMedia, ActionRemoveMedia, ActionShowCollection, ActionSearchMedia
Author: Nikhil	

Class: LibraryElement	
Responsibilities: Interface for representing all types of media. It gets used within the Composite pattern to illustrate a library,	

songs, releases, and artists.	
Collaborators: ...	
Users:	**Used by:** Library, Song, Release, SearchByRelease, SearchBySong, SearchByArtist, ActionRateMedia, ActionRemoveMedia, ActionAddMedia
Author: Brandon/Alex	

Class: Library	
Responsibilities: Receiver for the CLI, which implements the Command pattern. It also serves as a Composite for the Composite pattern. It represents the Library, which contains songs and releases for a user.	
Collaborators: LibraryElement	
Users: ...	**Used by:** CLI, User, MediaCollection, RequestHandler, SearchBySong, SearchByRelease, SearchByArtist
Author: Brandon/Nikhil/Chase	

Class: Song	
Responsibilities: Represents a song. Songs have an ID, an artist represented by their ID, a duration, a song title, and a rating.	
Collaborators: LibraryElement	
Users: ...	**Used by:** ActionAdd/RemoveMedia
Author: Nikhil/Brandon	

Class: Release	
Responsibilities: Represents a release. Releases have an ID, an artist, a title, a release date, and a list of songs.	
Collaborators: LibraryElement, Song	
Users: ...	**Used by:** ActionAddMedia, ActionRemoveMedia

Author: Nikhil/Brandon	

Class: Artist	
Responsibilities: Represents and holds information about an Artist. Artists have a string ID, name, and disambiguation.	
Collaborators: ...	
Users: Media	**Used by:**
Author: Nikhil	

Class: Response	
Responsibilities: Takes in a list and returns a stringified form of it.	
Collaborators: ...	
Users: ...	**Used by:** Action Classes, Grouping, Library
Author: Nikhil	

Class: User	
Responsibilities: Represents and handles a user of the CLI subsystem. The user has their own media collection. They are represented with an integer ID and have a username.	
Collaborators: Library, Grouping	
Users: ...	**Used by:** CLI
Author: Brandon, Qunzhan	

Class: MediaCollection	
Responsibilities: A wrapper for the Grouping and a user's library.	
Collaborators: Library, User	
Users: Library, Grouping	**Used by:**RequestHandler, User, ActionAddMedia, ActionRateMedia, ActionRemoveMedia, ActionShowCollection, ActionSearchMedia
Author: Nikhil	

Class: MediumType	
Responsibilities: Enum representing the different types of mediums that a release can have.	
Collaborators: ...	
Users: ...	**Used by:** Release
Author: Nikhil	

Class: SearchByArtist	
Responsibilities: Concrete Strategy that searches the Grouping or the library by artist. Implements MediaSearcher.	
Collaborators: ...	
Users: Grouping, Library, LibraryElements	**Used by:** MediaList
Author: Nikhil	

Class: SearchByRelease	
Responsibilities: Concrete Strategy that searches the Grouping or the library by release. Implements MediaSearcher.	
Collaborators: ...	
Users: Grouping, Library, LibraryElements	**Used by:** MediaList
Author: Nikhil	

Class: SearchBySong	
Responsibilities: Concrete Strategy that searches the Grouping or the library by song. Implements MediaSearcher.	
Collaborators: ...	
Users: Grouping, Library, LibraryElements	**Used by:** MediaList
Author: Nikhil	

Class: MediaList	

Responsibilities: Strategy Context that dictates what type of search needs to be applied for the respective media entered.	
Collaborators: ...	
Users: MediaSearcher	**Used by:** ActionSearchMedia
Author: Nikhil	

Class: MediaSearcher	
Responsibilities: Interface for search methods. Does a search for the passed string in the passed collection. Where it searches depends on the class that implements it.	
Collaborators: ...	
Users:	**Used by:** SearchBySong, SearchByArtist, SearchByRelease
Author: Nikhil	

Class: ActionCreateCollection	
Responsibilities: Creates a new collection.	
Collaborators: ...	
Users:	**Used by:** RequestHandler
Author: Nikhil	

Class: WebService	
Responsibilities: Implements the db interface. Queries the MusicBrainz API when the user has chosen to access the online database. It is a singleton; it stores its own single instance that can be accessed by other classes. Interprets Json results through the use of the Gson library. Gson creates Song, Artist, and Release objects from reading the Json data into classes representing Json objects, which are then converted to an acceptable format for the offline database. Given a release's ID, populates the release with songs from that release before adding the release to a user's library. This must be done in a separate lookup because MusicBrainz does not send tracklist data in an initial query.	
Collaborators:	
Users:	**Used by:** MediaCollection, ActionAddMedia, SearchByRelease, SearchBySong, SearchByArtist
Author: Alex	

Class: WebArtistCredit	
Responsibilities: Represents an artist as returned by the MusicBrainz Web Service. Used to format data from GSon.	
Collaborators:	
Users:	**Used by:** WebService
Author: Alex	

Class: WebMedium	
Responsibilities: Represents a type of music object as returned by the MusicBrainz Web Service. Used to format data from	

GSon.	
Collaborators:	
Users:	**Used by:** WebService
Author: Alex	

Class: WebRecording	
Responsibilities: Represents a recording as returned by the MusicBrainz Web Service. Used to format data from GSon.	
Collaborators:	
Users:	**Used by:** WebService
Author: Alex	

Class: WebRelease	
Responsibilities: Represents a release as returned by the MusicBrainz Web Service. Used to format data from GSon.	
Collaborators:	
Users:	**Used by:** WebService, WebReleaseList
Author: Alex	

Class: WebReleaseList	
Responsibilities: Represents a list of releases as returned by the MusicBrainz Web Service. Used to format data from GSon.	
Collaborators: WebRelease	
Users:	**Used by:** WebService
Author: Alex	

Class: WebSong	
Responsibilities: Represents a song as returned by the MusicBrainz Web Service. Used to format data from GSon.	
Collaborators:	
Users:	**Used by:** WebService
Author: Alex	

Class: WebTrack	
Responsibilities: Represents a track as returned by the MusicBrainz Web Service. Used to format data from GSon.	
Collaborators:	
Users:	**Used by:** WebService
Author: Alex	

Class: Grouping	
Responsibilities: Groups together media via HashMaps of Artists, releases, and songs	
Collaborators: Artist, Release, Song	
Users:	**Used by:** CLI, GUIStart
Author: Nikhil	

Class: db	
Responsibilities: Interface that defines getters for song, release, and artist lists. Implemented by WebService and Grouping.	
Collaborators:	
Users:	**Used by:** Grouping, WebService, MediaCollection
Author: Alex	

Class: GUI_AddMedia	
Responsibilities: GUI class for adding media functionality. Creates a jPanel such that a user can enter a media name, select media type, and add it to their library.	
Collaborators:	
Users:	**Used by:** GUI_Handler
Author: Brandon	

Class: GUI_Home	
Responsibilities: GUI class for home page. After using the GUI_Start class to sign in, a user is brought to this home page where	
Collaborators:	
Users:	**Used by:** GUI_Handler
Author: Brandon	

Class: GUI_Library	
Responsibilities: GUI class jPanel for library view. This class creates the page where a user can view and update their library by adding, removing, and rating songs via buttons and text boxes.	
Collaborators:	
Users:	**Used by:** GUI_Handler
Author: Brandon	

Class: GUI_RateMedia	
Responsibilities: GUI class jPanel for rating media. This class handles the pop up for rating media. The user can specify the name of the media and then the rating (1-5).	
Collaborators:	

Users:	**Used by:** GUI_Handler
Author: Brandon	

Class: GUI_RemoveMedia	
Responsibilities: GUI class jPanel for removing media functionality. A user can remove media from their library by entering the media name and selecting the media type from the drop down.	
Collaborators:	
Users:	**Used by:** GUI_Handler
Author: Brandon	

Class: GUI_Search	
Responsibilities: GUI class jPanel for users to search. This page allows users to search through the grouping for songs or releases based on entered queries in the text box. After a search, the page is updated with a list of results such that the user can view it.	
Collaborators:	
Users:	**Used by:** GUI_Handler
Author: Brandon	

Class: GUI_Start	
Responsibilities: GUI class jPanel for starting panel, where users log in. After entering a username, the class checks if the user exists in the persisted data and logs into that user, otherwise creates a new user and logs in.	
Collaborators:	
Users:	**Used by:** GUI_Handler
Author: Brandon	

Class: GUI_Handler	

Responsibilities: Handler class that wraps all GUI classes (jPanels) together. This is the class you run in order to start the GUI aspect of the program. It holds the users library page and allows for interaction with the other classes (panels).	
Collaborators:	
Users: GUI_AddMedia, GUI_Home, GUI_Library, GUI_RateMedia, GUI_RemoveMedia, GUI_Search, GUI_Start	**Used by:**
Author: Brandon	

Class: UndoRedoManager	
Responsibilities: Contain two stacks, undo stack and redo stack. handle the undo and redo request, invoke the undo and redo methods in ActionAddMedia, ActionRemoveMedia and ActionRateMedia	
Collaborators:	
Users: ActionAddMedia, ActionRemoveMedia, ActionRateMedia	**Used by:** RequestHandler
Author: Qunzhan Huang	