Muze Music Library System v1

Design Documentation  
Prepared by Team 05:

* Jarred Moyer <[jam4936@rit.edu](mailto:jam4936@rit.edu)>
* Cameron Riu <[cmr6689@rit.edu](mailto:cmr6689@rit.edu)>
* Shane Burke <[sdb5978@rit.edu](mailto:sdb5978@rit.edu)>
* Fahd Masood <[fxm1492@rit.edu](mailto:fxm1492@rit.edu)>
* Ryan Borger <[rlb8800@rit.edu](mailto:rlb8800@rit.edu)>

[**Frequently Used Terms**](#_heading=h.stdkdww994n1) **2**

[**Summary**](#_heading=h.b1nthpa3jm37) **2**

[**Domain Model**](#_heading=h.1fob9te) **3**

[**System Architecture**](#_heading=h.3znysh7) **4**

[**Subsystems**](#_heading=h.2et92p0) **6**

Searching 6

User Input 8

Database Parsing 10

[**Status of the Implementation**](#_heading=h.1t3h5sf) **12**

[**Appendix**](#_heading=h.4d34og8) **13**

# 

# 

# Frequently Used Terms

|  |  |
| --- | --- |
| Term | Definition |
| MMLS | Muze Music Library System |
| Library | User defined collection of releases, artists, and songs |
| Database | Program defined list of all possible songs, artists, and releases |

# 

# 

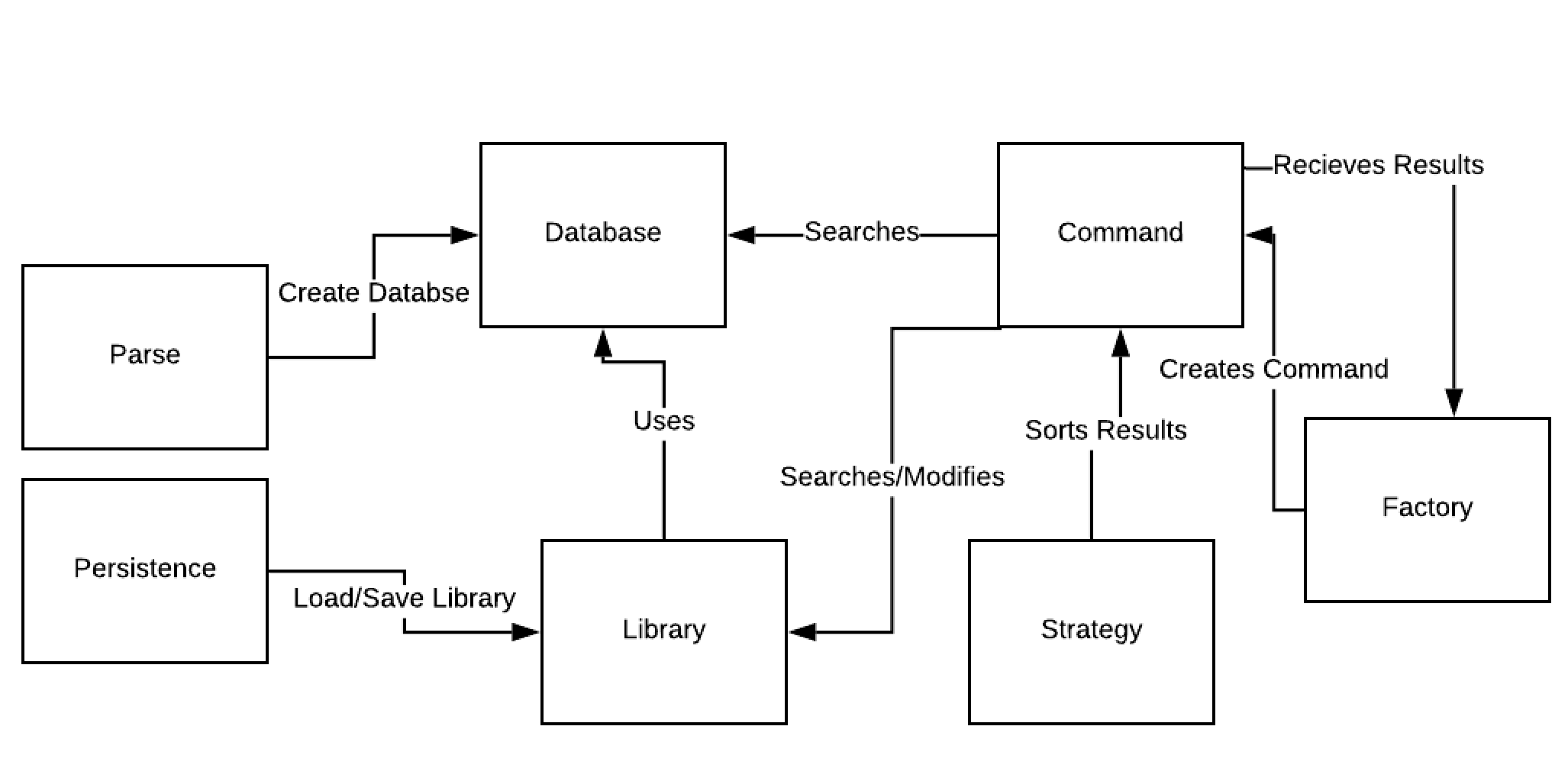
# Summary

The MMLS aims to provide music aficionados a way to track the songs, releases, and artists in their collections. Users can add songs and releases to their personal library and can optionally specify their acquisition date. Users add songs from a database by searching with keywords, id numbers, release dates, and other tags. Users can also filter their personal library using similar methodologies. Users also have the ability to explore artists and their releases using an in-application search feature. Finally, if a user removes an item from their collection, the system can remove the item from the system.

# Domain Model

# 

# System Architecture



Parse: The responsibility of Parse is to be able to parse the CSV files and create a database from the data contained in the files.

Persistence: The responsibility of Perseverance is to handle the state of the user’s Library. As media is added or removed from the Library these changes need to be saved. By saving the state of the library this allows for retrieval of media after the Library has undergone modification.

Database: The responsibility of the Database is to contain all of the information from the CSV files. The Database will contain all Artists, Songs, and Releases that the user will be able to search the Database for.

Command: The responsibility of the command strategy is to allow users to undo commands issued when querying, as well as to streamline the handling of the different forms of requests from the command line. The song and release are able to be added and removed from a users library and the help response will display a list of possible commands to the user. Users can also search the database or their library. This strategy will be beneficial as there are multiple requests that only certain classes will be affected by. The command pattern uses the Command Interface. This interface executes the commands.

Library: The responsibility of the Library is to contain all of the user’s media. The user can add Artists, Songs, and Releases to their personal Library.

Strategy: The responsibility of the Strategy pattern is sorting the results. Results can be sorted Alphabetically, by Acquisition Date, by Release Date, or by Rating depending on what the user is searching for. Included in the Strategy pattern is the Comparator Interface. This interface allows each sorting component to use the compare() method to be able to sort the results.

Factory: The Request class may represent multiple types of user requests. By creating a RequestFactory for the desired type of Request, polymorphism may be used to create the appropriate Request object through a factory method belonging to the RequestFactory interface. This way, the creation of the Request object is segregated from other behavior. Included in the Factory pattern is the Factory Interface.

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# Subsystems

This section provides detailed design for specific subsystems described in the system architecture.

## Searching

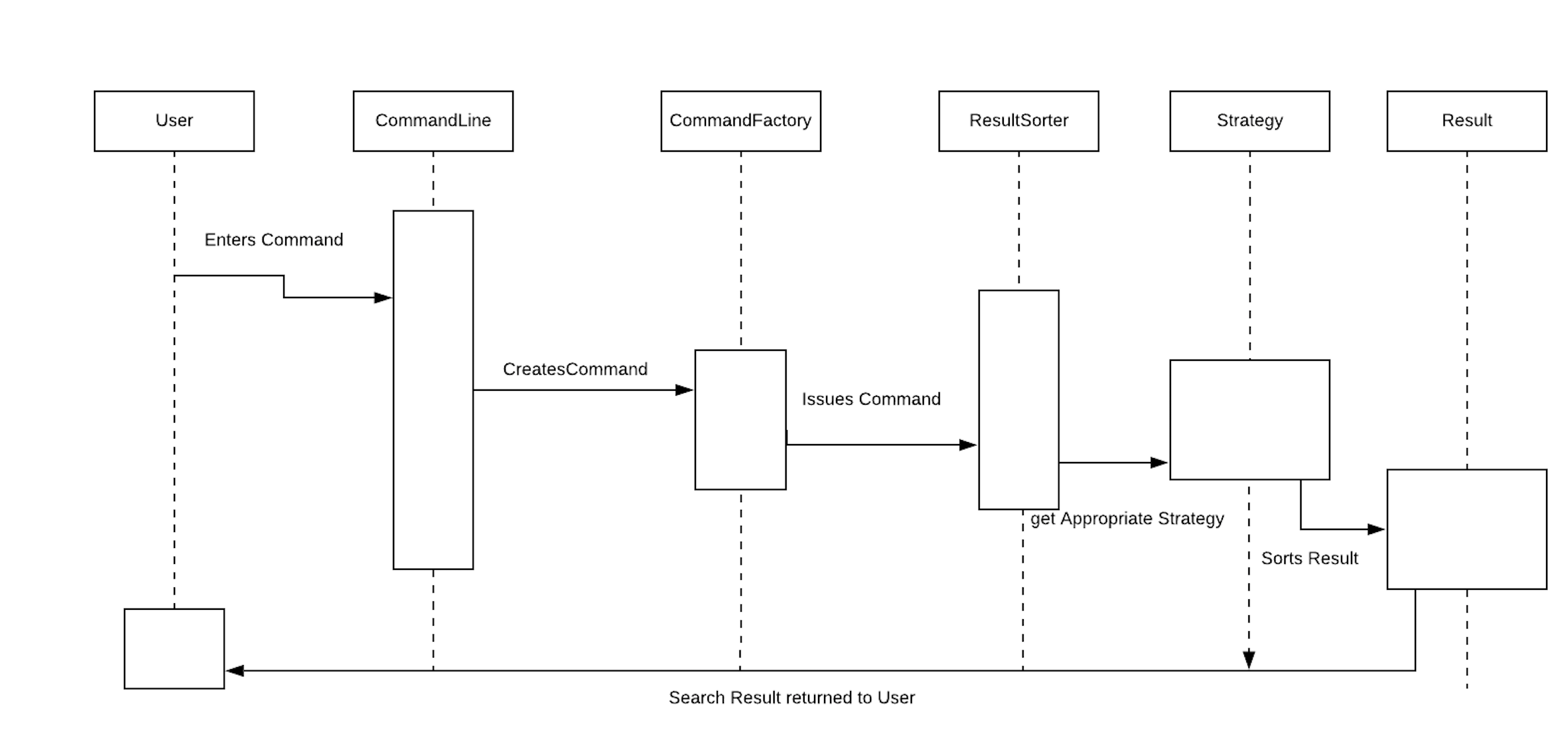
The CommandLine creates an instance of CommandFactory. When the user enters a line of text, it is passed to the CommandFactory’s *createCommand* method, wherein a command is created and returned. The Command is then returned to and executed by the CommandLine.

Depending on the type of search, the Library or Database is supplied to the SearchCommand object. The appropriate item list (releases, artists, or songs from the database/library) is obtained. Filters are successively applied to the list, narrowing down the results until the remaining items (if any) match the given parameters. Filters use generic types to ensure reusability.

If an optional parameter is supplied, then the corresponding filter object is created. Given the current list of results and the filter string, a new list is returned containing matching results. Once the results are finalized, a ResultSorter object is created to sort the results. The ResultSorter object generates the appropriate sorting strategy object based on the strategy specified by the user. The results are sorted, and the CommandFactory is notified. Upon notification, the Command also pushes the list of search results to the CommandFactory.

Sequence Diagrams:

Searching: This sequence diagram shows how user input is handled in the command line and processed.



|  |  |  |  |
| --- | --- | --- | --- |
| **Name: Command** | | | **GoF pattern: Command** |
| **Participants** | | | |
| **Class** | **Role in GoF pattern** | **Participant's contribution in the context of the application** | |
| **CommandLine** | **Invoker** | After receiving the Command from the CommandFactory, the CommandLine executes the returned command. If no valid command was returned, then an error message is printed. In the case of the “back” command, an empty command with no operations is executed. | |
| **Library** | **Receiver** | For AddCommand and RemoveCommand, operations are performed on the items in the Library. These commands do not notify the CommandFactory, but a message is printed to standard output indicating that the library operation was successful. | |
| **CommandFactory** | **Receiver** | The CommandFactory is the receiver of all search commands. When a search has been completed, the CommandFactory is notified and “pushed” the results. These results are used in further operations by the CommandFactory (or other commands). | |
| **Command** | **Command** | This interface defines the minimum behavior a command must possess: an *executeCommand()* method. All command classes implement this interface. | |
| **SearchCommand** | **ConcreteCommand** | This is an abstract class representing any command that searches either the library or database. It contains a reference to the CommandFactory, which is the receiver of these commands. | |
| **LibraryCommand** | **ConcreteCommand** | This abstract class represents any command that requires the Library and/or Database as well as the list of search results. The receiver of these commands is the Library. | |
| **ListCommand** | **ConcreteCommand** | This command responds to a user query for all artists currently in the library. Its receiver is the CommandFactory; the results of the query are pushed to the CommandFactory so they can be printed. | |
| **Deviations from the standard pattern:** Some command classes are abstract. For example, the abstract class SearchCommand implements Command, but the concrete implementation of *executeCommand()* does not exist in this class. | | | |
| **Requirements being covered:** Searching, user requests, library operations | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name:** Strategy | | | **GoF pattern:**Strategy |
| **Participants** | | | |
| **Class** | **Role in GoF pattern** | Participant's contribution in the context of the application | |
| ResultSorter | Context | The ResultSorter delegates sorting implementation to the Comparator Interface. | |
| Comparator | Strategy | Comparator declares an interface common to all the sorting strategies | |
| SortAlphabetically | ConcreteStrategy A | Implements the algorithm using the Comparator interface to sort songs, releases, and artists alphabetically. | |
| SortByAcquisitionDate | ConcreteStrategy B | Implements the algorithm using the Comparator interface to sort releases and songs by acquisition date. | |
| SortByRating | ConcreteStrategy C | Implements the algorithm using the Comparator interface to sort releases, artists and songs by rating. | |
| SortByReleaseDate | ConcreteStrategy D | Implements the algorithm using the Comparator interface to sort releases, artists and songs by release date. | |
| **Deviations from the standard pattern:**  ResultSorter is a class instead of an interface. It contains case statements, which determine which sorting technique to implement. | | | |
| **Requirements being covered:** Allows the user to sort provided data in multiple ways. | | | |

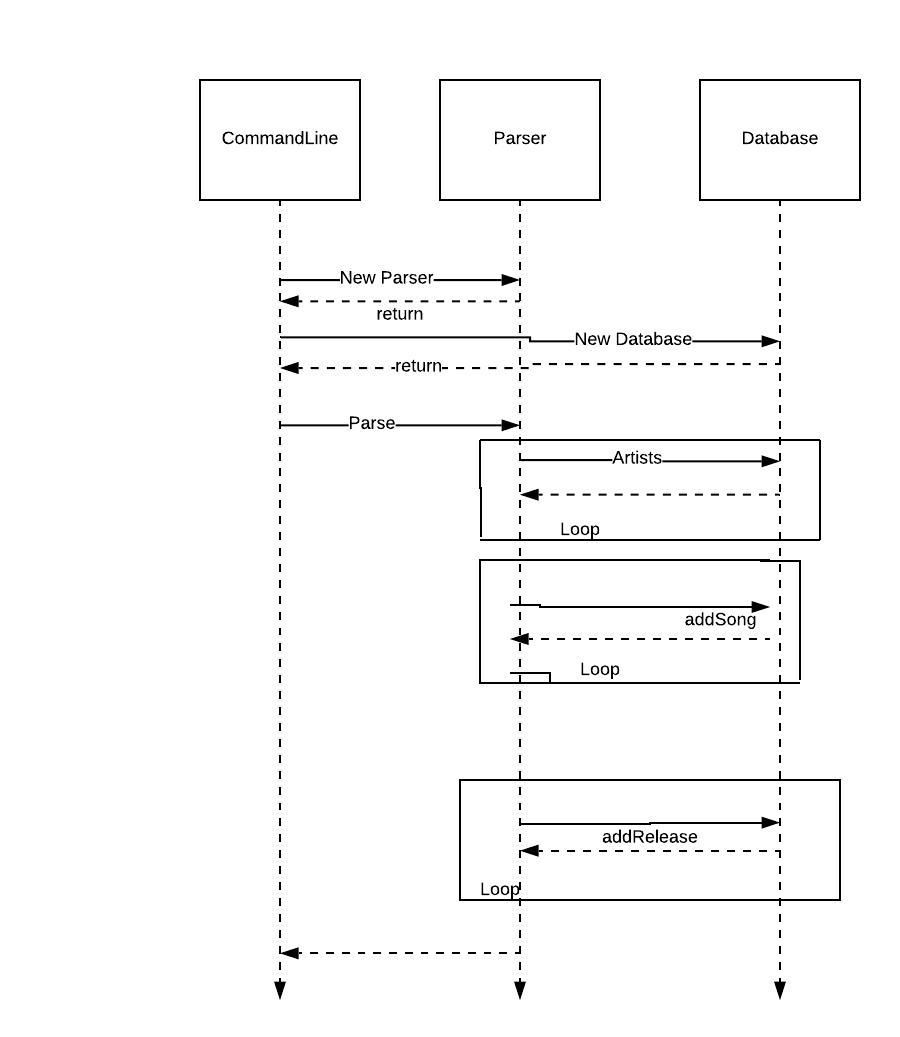
## User Input

* The user can input commands into the command line which are then differentiated by the command factory and returned back to the command line class which executes the proper command.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name: Factory** | | | **GoF pattern: Factory Method** |
| **Participants** | | | |
| **Class** | **Role in GoF pattern** | **Participant's contribution in the context of the application** | |
| **CommandLine** | **Client** | When a user makes requests to the command line the command creates a command factory and executes the returned command. | |
| **Factory** | **Creator** | The factory interface provides the create command method that is used by the concrete creator. | |
| **CommandFactory** | **ConcreteCreator** | The command factory uses the user input to create the needed command based on the Matcher. | |
| **Deviations from the standard pattern: Only has one concrete factory method** | | | |
| **Requirements being covered: User input** | | | |

|  |  |  |  |
| --- | --- | --- | --- |
| **Name: Command** | | | **GoF pattern: Command** |
| **Participants** | | | |
| **Class** | **Role in GoF pattern** | **Participant's contribution in the context of the application** | |
| **CommandLine** | **Invoker** | After receiving the Command from the CommandFactory, the CommandLine executes the returned command. If no valid command was returned, then an error message is printed. In the case of the “back” command, an empty command with no operations is executed. | |
| **Library** | **Receiver** | For AddCommand and RemoveCommand, operations are performed on the items in the Library. These commands do not notify the CommandFactory, but a message is printed to standard output indicating that the library operation was successful. | |
| **CommandFactory** | **Receiver** | The CommandFactory is the receiver of all search commands. When a search has been completed, the CommandFactory is notified and “pushed” the results. These results are used in further operations by the CommandFactory (or other commands). | |
| **Command** | **Command** | This interface defines the minimum behavior a command must possess: an *executeCommand()* method. All command classes implement this interface. | |
| **SearchCommand** | **ConcreteCommand** | This is an abstract class representing any command that searches either the library or database. It contains a reference to the CommandFactory, which is the receiver of these commands. | |
| **LibraryCommand** | **ConcreteCommand** | This abstract class represents any command that requires the Library and/or Database as well as the list of search results. The receiver of these commands is the Library. | |
| **ListCommand** | **ConcreteCommand** | This command responds to a user query for all artists currently in the library. Its receiver is the CommandFactory; the results of the query are pushed to the CommandFactory so they can be printed. | |
| **Deviations from the standard pattern:** Some command classes are abstract. For example, the abstract class SearchCommand implements Command, but the concrete implementation of *executeCommand()* does not exist in this class. | | | |
| **Requirements being covered:** Searching, user requests, library operations | | | |

## Database Parsing



# Status of the Implementation

Finished, needs unit testing and refactoring.

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# Appendix

This section provides fine-grained design details for all of the classes in your design. You will capture this information using the CRC (Class-Responsibilities-Collaborators) card format below.

|  |  |
| --- | --- |
| **Class:** Database |  |
| **Responsibilities:** This class contains the collections of songs, artists, and releases that the user and the music library has access to. The collections are stored as HashMaps where the key is the GUID of the respective entity and the value is that entities object. Contains getters and setters for each. |  |
| **Collaborators:** ... |  |
| **Uses:** Song, Artist, Release | **Used by:** Parser, Song, Release, CommandFactory, LibraryCommand, AddCommand, RemoveCommand, RateCommand, ExploreCommand, DatabaseSearchCommand, DatabaseSearchSongCommand, DatabaseSearchReleaseCommand |
| **Author:** Jarred Moyer, Shane Burke |  |

|  |  |
| --- | --- |
| **Class:** Parser |  |
| **Responsibilities:** This class uses the provided CSV files to parse each line correctly into the database class. Each line in the file is converted to its respective object (Song, Artist, Release). The files are parsed at every program start. |  |
| **Collaborators:** ... |  |
| **Uses:** Database, Artist, Song, Release | **Used by:** Command Line |
| **Author:** Jarred Moyer, Cameron Riu |  |

|  |  |
| --- | --- |
| **Class:** Item |  |
| **Responsibilities:** This abstract class has the primary responsibility of providing the getters and setters for its subclasses which are Song, Artist, and Release. |  |
| **Collaborators:** ... |  |
| **Uses:** ... | **Used by:** Song, Artist, Release, LibrarySearchSongCommand, LibrarySearchReleaseCommand, LibrarySearchArtistCommand, DatabaseSearchSongCommand, DatabaseSearchReleaseCommand |
| **Author:** Shane Burke, Ryan Borger, Jarred Moyer |  |

|  |  |
| --- | --- |
| **Class:** Song |  |
| **Responsibilities:** This class holds the data for a song from one line of the song CSV file. It contains the artist of the song, the duration, and the GUID of the song. |  |
| **Collaborators:** ... |  |
| **Uses:** Artist, Database, Item | **Used by:** Database, Library, Parser, Release, AddCommand, RemoveCommand, RateCommand, ExploreCommand, LibrarySearchSongCommand, DatabaseSearchSongCommand |
| **Author:** Jarred Moyer, Shane Burke, Cameron Riu |  |

|  |  |
| --- | --- |
| **Class:** Artist |  |
| **Responsibilities:** This class holds the data for an artist from one line of the artist CSV file. It contains the name, the GUID, and the type if applicable. |  |
| **Collaborators:** ... |  |
| **Uses:** Item | **Used by:** Database, Library, Parser, Song, Release, ExploreCommand, LibrarySearchArtistCommand |
| **Author:** Jarred Moyer, Shane Burke, Cameron Riu |  |

|  |  |
| --- | --- |
| **Class:** Release |  |
| **Responsibilities:** This class holds the data for an artist from one line of the Release CSV file.It contains the GUID, the duration, the artist GUID, the artist, the title, the issue date, the medium, a list of songs that are the track list, and the GUID’s for that song list. |  |
| **Collaborators:** ... |  |
| **Uses:** Item, Song, Artist, Database | **Used by:** Database, Library, Parser, AddCommand, Remove Command, ExploreCommand, LibrarySearchSongCommand, LibrarySearchReleaseCommand, DatabaseSearchReleaseCommand |
| **Author:** Jarred Moyer, Ryan Borger, Cameron Riu, Shane Burke |  |

|  |  |
| --- | --- |
| **Class:** Library |  |
| **Responsibilities:** This class holds the users personal database data. Songs, releases, and artists are organized into hashmaps and are populated and their acquisition date stored. Also handles initiating save requests to persist help. |  |
| **Collaborators:** ... |  |
| **Uses:** Song, Release, Artist, PersistHelp, Date | **Used by:** CommandFactory, LibraryCommand, AddCommand, RemoveCommand, RateCommand, ExploreCommand, LibrarySearchCommand, LibrarySearchSongCommand, LibrarySearchArtistCommand, LibrarySearchReleaseCommand |
| **Author:** Jarred Moyer, Shane Burke |  |

|  |  |
| --- | --- |
| **Class:** PersistHelp |  |
| **Responsibilities:** Handles the loading and saving required to ensure the persistence of the library object. |  |
| **Collaborators:** ... |  |
| **Uses:** Library | **Used by:** CommandLine, Library |
| **Author:** Jarred Moyer |  |

|  |  |
| --- | --- |
| **Class:** CommandLine |  |
| **Responsibilities:** The command line is the front end of the program. It calls the database parsing and then when the user inputs a command then creates the command factory to execute the command. |  |
| **Collaborators:** ... |  |
| **Uses:** Database, CommandFactory | **Used by:** ... |
| **Author:** Jarred Moyer, Shane Burke, Cameron Riu |  |

|  |  |
| --- | --- |
| **Class:** ResultSorter |  |
| **Responsibilities:** Dictates which sorting algorithm to use depending on what the user specifies in the command line. |  |
| **Collaborators:** ... |  |
| **Uses:** Comparator | **Used by:** LibrarySearchArtistCommand  LibrarySearchCommand  LibrarySearchReleaseCommand  LibrarySearchSongCommand |
| **Author:** Shane Burke |  |

|  |  |
| --- | --- |
| **Class:** SortAlphabetically |  |
| **Responsibilities:** Comparator that takes in items and compares their names for sorting purposes. Results will be presented in alphabetical order. |  |
| **Collaborators:** ... |  |
| **Uses:** Comparator | **Used by:** ResultSorter |
| **Author:** Ryan Borger |  |

|  |  |
| --- | --- |
| **Class:** SortByAcquisitionDate |  |
| **Responsibilities:** Comparator that takes in items and compares their acquisition date for sorting purposes. Results will be presented by the most recent acquisition date. |  |
| **Collaborators:** ... |  |
| **Uses:** Comparator | **Used by:** ResultSorter |
| **Author:** Ryan Borger |  |

|  |  |
| --- | --- |
| **Class:** SortByRating |  |
| **Responsibilities:** Comparator that takes in items and compares their rating for sorting purposes. Results will be presented by highest to lowest rating. |  |
| **Collaborators:** ... |  |
| **Uses:** Comparator | **Used by:** ResultSorter |
| **Author:** Ryan Borger, Shane Burke |  |

|  |  |
| --- | --- |
| **Class:** SortByReleaseDate |  |
| **Responsibilities:** Comparator that compares two item’s release date for sorting purposes. Results will be presented by the most recent release date. |  |
| **Collaborators:** ... |  |
| **Uses:** Comparator | **Used by:** ResultSorter |
| **Author:** Ryan Borger |  |

|  |  |
| --- | --- |
| **Class:** CommandLine |  |
| **Responsibilities:** ... |  |
| **Collaborators:** ... |  |
| **Uses:** ... | **Used by:** ... |
| **Author:** ... |  |

|  |  |
| --- | --- |
| **Class:** Command |  |
| **Responsibilities:** This interface is used by all ConcreteCommands and provides a single method to execute commands. |  |
| **Collaborators:** ... |  |
| **Uses:** ... | **Used by:** CommandFactory, LibraryCommand, HelpCommand, AddCommand, RemoveCommand, RateCommand, ExploreCommand, Search Command |
| **Author:** Shane Burke |  |

|  |  |
| --- | --- |
| **Class:** Factory |  |
| **Responsibilities:** This interface is used by the factory class to allow for the creation of a command based upon a user request. |  |
| **Collaborators:** ... |  |
| **Uses:** ... | **Used by:** CommandFactory |
| **Author:** Shane Burke, Cameron Riu |  |

|  |  |
| --- | --- |
| **Class:** CommandFactory |  |
| **Responsibilities:** This class acts as the ConcreteCreator in the Factory Method pattern. It is responsible for taking in a user request and determining which command to create based on the arguments of the request. It uses search results from a search made by the user to modify the library and/or act on results shown in the command line. |  |
| **Collaborators:** ... |  |
| **Uses:** Factory, Library, Database, Command, DatabaseSearchSongCommand, LibrarySearchArtistCommand, AddCommand, RateCommand, RemoveCommand, ExploreCommand, HelpCommand | **Used by:** CommandLine, SearchCommand, LibrarySearchSongCommand, LibrarySearchArtistCommand, LibrarySearchReleaseCommand, DatabaseSearchCommand, DatabaseSearchSongCommand, DatabaseSearchReleaseCommand |
| **Author:** Shane Burke, Cameron Riu |  |

|  |  |
| --- | --- |
| **Class:** LibraryCommand |  |
| **Responsibilities:** This abstract class provides the constructor for all commands that act within or on the library by providing the global library and database, the matcher used in the CommandFactory, and the search results from a user search. |  |
| **Collaborators:** ... |  |
| **Uses:** Library, Database, Command | **Used by:** AddCommand, RemoveCommand, RateCommand, ExploreCommand |
| **Author:** Shane Burke, Cameron Riu |  |

|  |  |
| --- | --- |
| **Class:** AddCommand |  |
| **Responsibilities:** This class is responsible for adding a song or a release to the library based on the search results and the number input by the user in the request. |  |
| **Collaborators:** ... |  |
| **Uses:** Library, Database, LibraryCommand, Command, Song, Release | **Used by:** CommandFactory |
| **Author:** Cameron Riu, Shane Burke, Ryan Borger |  |

|  |  |
| --- | --- |
| **Class:** RemoveCommand |  |
| **Responsibilities:** This class is responsible for removing songs or releases from the library based on a search result list that the user can choose from. |  |
| **Collaborators:** ... |  |
| **Uses:** Library, Database, LibraryCommand, Command, Song, Release | **Used by:** CommandFactory |
| **Author:** Cameron Riu, Shane Burke |  |

|  |  |
| --- | --- |
| **Class:** RateCommand |  |
| **Responsibilities:** This class is responsible for adding a rating to a song in the library by using the provided search results and the rating given by the user. |  |
| **Collaborators:** ... |  |
| **Uses:** Library, Database, LibraryCommand, Command, Song | **Used by:** CommandFactory |
| **Author:** Cameron Riu, Shane Burke |  |

|  |  |
| --- | --- |
| **Class:** ExploreCommand |  |
| **Responsibilities:** This class is responsible for letting the user explore an artist by displaying the songs and the releases by that artist. The user can also explore the releases of the artist which will list the track names. |  |
| **Collaborators:** ... |  |
| **Uses:** Library, Database, LibraryCommand, Command, Song, Artist, Release | **Used by:** CommandFactory |
| **Author:** Cameron Riu, Shane Burke |  |

|  |  |
| --- | --- |
| **Class:** HelpCommand |  |
| **Responsibilities:** This class’ only responsibility is to print out the list of possible commands and how to use them. |  |
| **Collaborators:** ... |  |
| **Uses:** Command | **Used by:** CommandFactory |
| **Author:** Cameron Riu, Shane Burke |  |

|  |  |
| --- | --- |
| **Class:** SearchCommand |  |
| **Responsibilities:** The responsibility of the abstract search command class is to provide the constructor that gives the subclasses the matcher and CommandFactory objects. |  |
| **Collaborators:** ... |  |
| **Uses:** Command, CommandFactory | **Used by:** CommandFactory |
| **Author:** Shane Burke, Cameron Riu |  |

|  |  |
| --- | --- |
| **Class:** LibrarySearchCommand |  |
| **Responsibilities:** The responsibility of this abstract class is to also give the global library to the subclasses extending the functionality of the SearchCommand class. |  |
| **Collaborators:** ... |  |
| **Uses:** SearchCommand, Library | **Used by:** CommandFactory |
| **Author:** Shane Burke, Cameron Riu |  |

|  |  |
| --- | --- |
| **Class:** LibrarySearchSongCommand |  |
| **Responsibilities:** This class allows the user to search for songs given the parameters of the search and filters the results based on those parameters. |  |
| **Collaborators:** ... |  |
| **Uses:** LibrarySearchCommand, Library, CommandFactory, Song, Release, Item | **Used by:** CommandFactory |
| **Author:** Shane Burke |  |

|  |  |
| --- | --- |
| **Class:** LibrarySearchArtistCommand |  |
| **Responsibilities:** This class allows the user to search for artists given the parameters of the search and filters the results based on those parameters. |  |
| **Collaborators:** ... |  |
| **Uses:** LibrarySearchCommand, Library, CommandFactory, Artist, Item | **Used by:** CommandFactory |
| **Author:** Shane Burke |  |

|  |  |
| --- | --- |
| **Class:** LibrarySearchReleaseCommand |  |
| **Responsibilities:** This class allows the user to search for releases given the parameters of the search and filters the results based on those parameters. |  |
| **Collaborators:** ... |  |
| **Uses:** LibrarySearchCommand, Library, CommandFactory, Release, Item | **Used by:** CommandFactory |
| **Author:** Shane Burke |  |

|  |  |
| --- | --- |
| **Class:** DatabaseSearchCommand |  |
| **Responsibilities:** The responsibility of this abstract class is to also give the global database to the subclasses extending the functionality of the SearchCommand class. |  |
| **Collaborators:** ... |  |
| **Uses:** Database, CommandFactory | **Used by:** CommandFactory |
| **Author:** Shane Burke, Cameron Riu |  |

|  |  |
| --- | --- |
| **Class:** DatabaseSearchSongCommand |  |
| **Responsibilities:** This class allows the user to search for songs given the parameters of the search and filters the results based on those parameters. |  |
| **Collaborators:** ... |  |
| **Uses:** Database, CommandFactory, Song, Item | **Used by:** CommandFactory |
| **Author:** Shane Burke |  |

|  |  |
| --- | --- |
| **Class:** DatabaseSearchReleaseCommand |  |
| **Responsibilities:** This class allows the user to search for releases given the parameters of the search and filters the results based on those parameters. |  |
| **Collaborators:** ... |  |
| **Uses:** Database, CommandFactory, Release, Item | **Used by:** CommandFactory |
| **Author:** Shane Burke |  |