Library Book Management System

Team B: SWEN-262 Design Project R1

Charles Barber, Nicholas Feldman, Christopher Lim, Anthony Palumbo, Edward Wong

Table of Contents

SUMMARY

REVISION TABLE

Revision Number	Revision Date	Summary	Author
0.1	*02/14/2017*	*Domain Model*	*Anthony Palumbo*
0.2	*02/16/2017*	*Nouns & Verbs, Knowns & Unknowns*	*Charles Barber, Edward Wong*
0.3	*02/21/2017*	*Design Pattern Usage*	*Nicholas Feldman*
1.0	*02/22/2017*	*Initial Creation and Changes*	*Christopher Lim*
1.1	*02/03/2017*	*State vs. Strategy*	*Nicholas Feldman*
1.2	*03/02/2017*	*Design Evaluation*	*Anthony Palumbo*
1.3	*03/03/2017*	*Added UML Class Diagrams*	*Nicholas Feldman*
1.4	*03/10/2017*	*Updated UML Class Diagrams*	*Anthony Palumbo*
1.5	*03/10/2017*	*Added Feature Requirements*	*Christopher Lim*
1.6	*03/11/2017*	*Added Subsystem Design*	*Charles Barber*
1.7	*03/11/2017*	*Added Design Pattern Usage to Subsystems*	*Nicholas Feldman*
1.8	*03/15/2017*	*Updated UML Class Diagrams*	*Anthony Palumbo*
1.9	*03/15/2017*	*Added Architecture Model*	*Christopher Lim*
1.10	*03/15/2017*	*Added CRC Cards*	*Edward Wong*
1.11	*03/17/2017*	*Added Sequence Diagrams*	*Anthony Palumbo, Charles Barber*
1.12	*03/17/2017*	*Updated Domain Model*	*Anthony Palumbo*
1.13	*03/19/2017*	*Completed CRC Cards*	*Anthony Palumbo, Edward Wong*
1.14	*03/20/2017*	*Formatted Document*	*Christopher Lim*

Problem Statement

Design and implement the Library Book Management System (LBMS). The LBMS is Book Worm Library's (BWL) system for providing book information to users, tracking library visitor statistics for a library statistics report, tracking checked out books, and allowing the library inventory to be updated. It is the server-side system that provides an API used by client-side interfaces that BWL employees use.

System Requirements

At a high-level this project will be source controlled on GitHub, implemented in Java as a desktop application. Must be compatible with the standard Java 1.8 SDK installed on the RIT SE lab machines. The system does not require or use any form of external database, persisting only in standard Java constructs. The system will be delivered as an executable jar file and require no network connection to function. A start.bat file will be provided to set any required environment variables, perform any program specific initialization, and execute the program, and will able to be executed from a Windows Command Prompt.

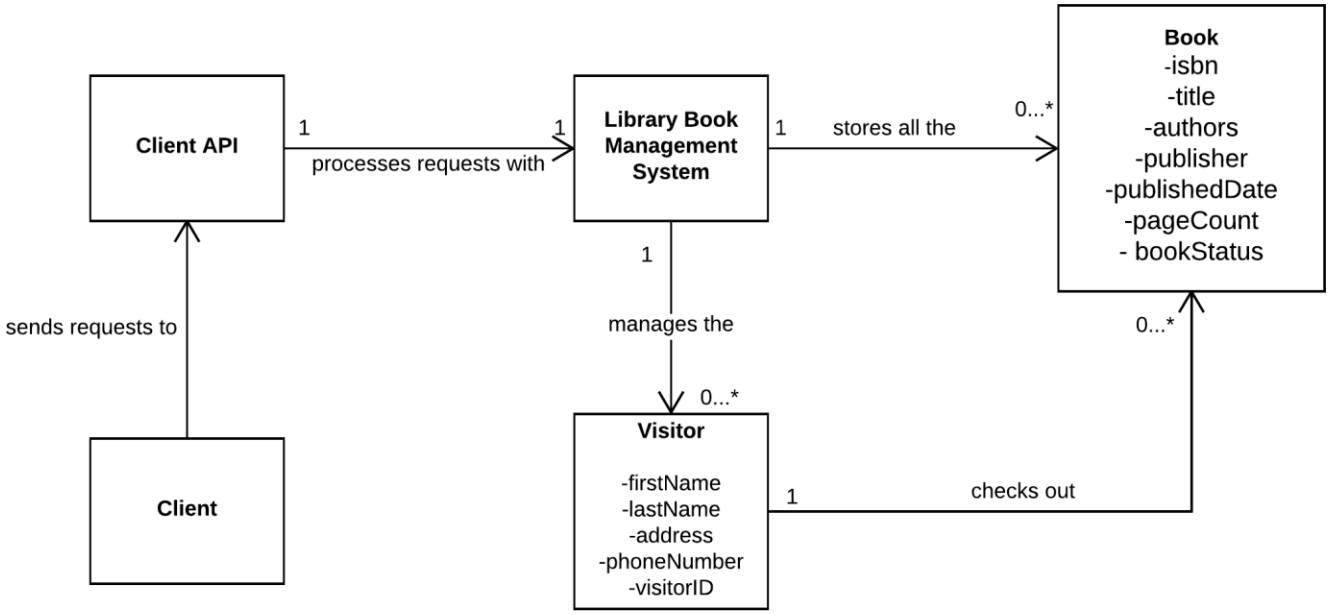
Feature Requirements

No.	*User Story Name*	*Description*
1	API	The LBMS shall be use text-based requests and responses. An LBMS exchange consists of one text string sent by a client followed by one text string sent by the system. The system shall receive requests from a client as text strings. A client shall terminate request strings with a semicolon (;) character. If the exchange is a partial client request, i.e. does not end with a termination character, the system response shall indicate that it received the partial client request and the system shall wait to receive the remainder of the request in the next one or more exchanges. If the exchange completes a client request, the system shall perform the requested operation, and provide a response for the request according to the LBMS server reply format specification.
2	Visitor Registration	The LBMS will require that first time visitors to the library register. The system will store the following information for each visitor: first name, last name, address, and phone number, and a visitor ID (a unique 10-digit ID generated the first time a visitor registers).
3	Visits	The LBMS shall keep track of visits by visitors and the time each visitor spends at the library during each visit. The system shall keep track of the time each visitor spends at the library during each visit. (Information will be used for statistical data in library reports.)
4	Operational Hours	The library opens at 08:00 every day. All visits in progress are automatically ended when the library closes at 19:00 when visitors remaining in the library are asked to leave. Visits do not extend over multiple days.

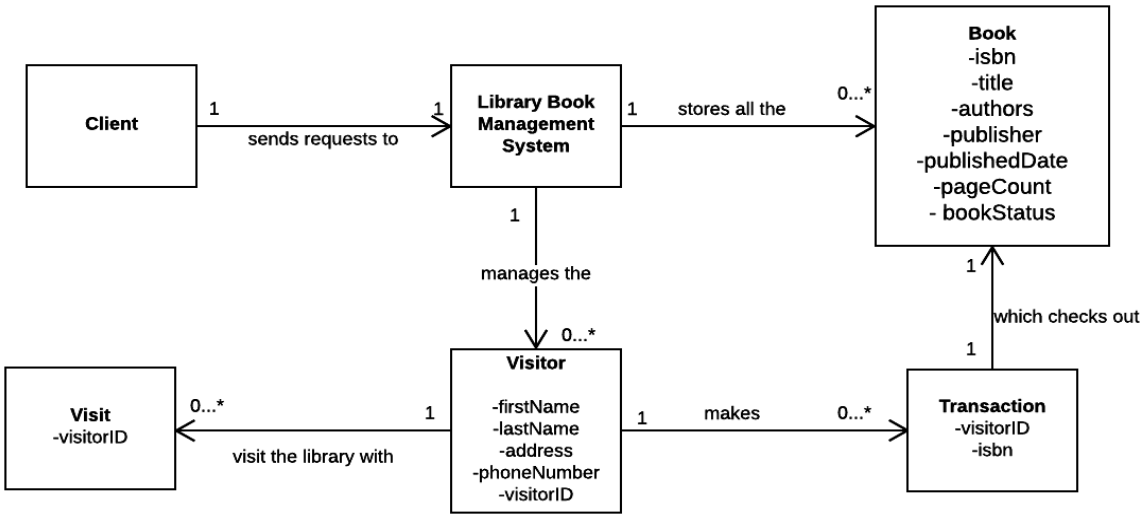
5	Searching	The LBMS shall respond to queries for book information. The system shall store book data for all books currently in BWL's possession. Book data shall consist of: isbn, title, author (can be multiple authors), publisher, published date, page count, number of copies, and number of copies currently checked out. The system shall respond with all information matching the provided search parameters in the order requested in the query. The client can request an ordering by title, publish date, total number of copies, and the number of available copies (i.e. not checked out). The system shall respond with an empty string when there are no books matching the query.
6	Checking out	The LBMS shall track checked out books by visitors. The system shall allow for a maximum of 5 books per visitor to be checked out and each book may be checked out for a maximum of 7 days. The system will store the data of the book check out transaction with the following information: isbn, visitor ID, date checked out, due date.
7	Fines	The LBMS shall apply an initial $10.00 fine to all books 1 day overdue. Subsequently, adding $2.00 for each additional week overdue, with a maximum fine of $30.00.
8	Statistics Report	The LBMS shall respond to queries for an informational report of the library. The system shall respond to a statistical query with the following information about a queried month at the library: i. The number of books currently owned by the library. ii. The number of visitors of the library. iii. The average amount of time spent at the library for a visit. iv. The books purchased for the specified month. v. The amount of money collected through checked out book fines
9	Advance Time	The LBMS shall support a feature to track and advance time. On initial startup, the LBMS system will record the date. The LBMS system shall track the number of days that have passed. The LBMS system shall allow users to move the date forward by a specified number of days. The time of day will remain unaffected. Upon each date change the system will generate a report of any overdue books (checked out by users past the due date).
10	Clean Shutdown	The LBMS system shall provide a mechanism for a "clean" shutdown of the system. The system shall end any visits in progress at system shutdown and persist all data at system shutdown.
11	Startup	The LBMS system shall restore persistent state on startup. Any state previously stored will be restored on startup.

DOMAIN MODEL

ORIGINAL



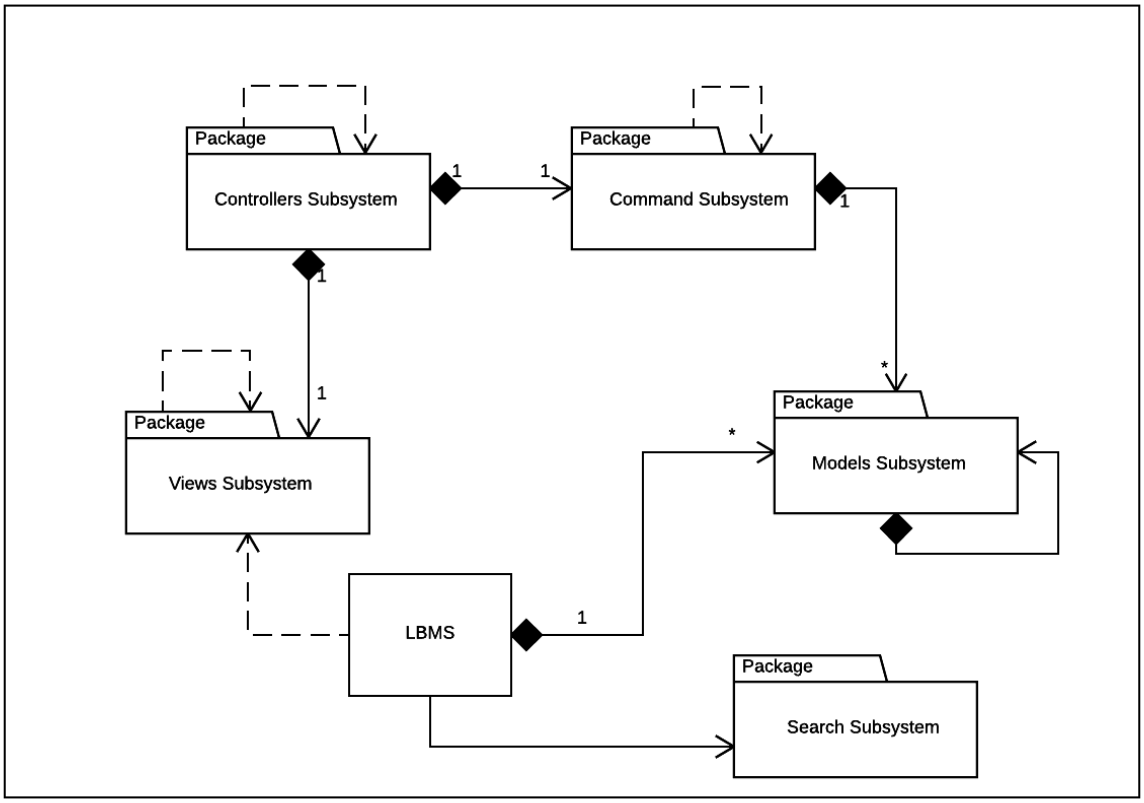
UPDATED



ARCHITECTURAL MODEL

The following model displays the interactions between each subsystem forming our entire system. The entry point into the system is through LBMS, which is dependent on the Views Subsystem as it is displayed to the user as a view. User input is sent to the Controllers Subsystem which interprets and handles the input and converts into a specific command within the Command Subsystem. Also, the Controllers Subsystem will update the view with the appropriate view once the command is executed. The Command Subsystem interacts with the model modifying or accessing the data. Model data are stored in memory within LBMS. This memory can be queried using the Search Subsystem, which is used for things such as a search command.

The advantages of this design are that it separates concerns using a model-view-controller architecture, allowing us to update the view at any time while keeping the interaction with the model and the model itself the same. Also, the design patterns used allow us to reuse many of the implementations such as the searches and models. Additionally, adding additional algorithms or features is incredibly simple and involves minimal adjustment to the rest of the system to achieve.

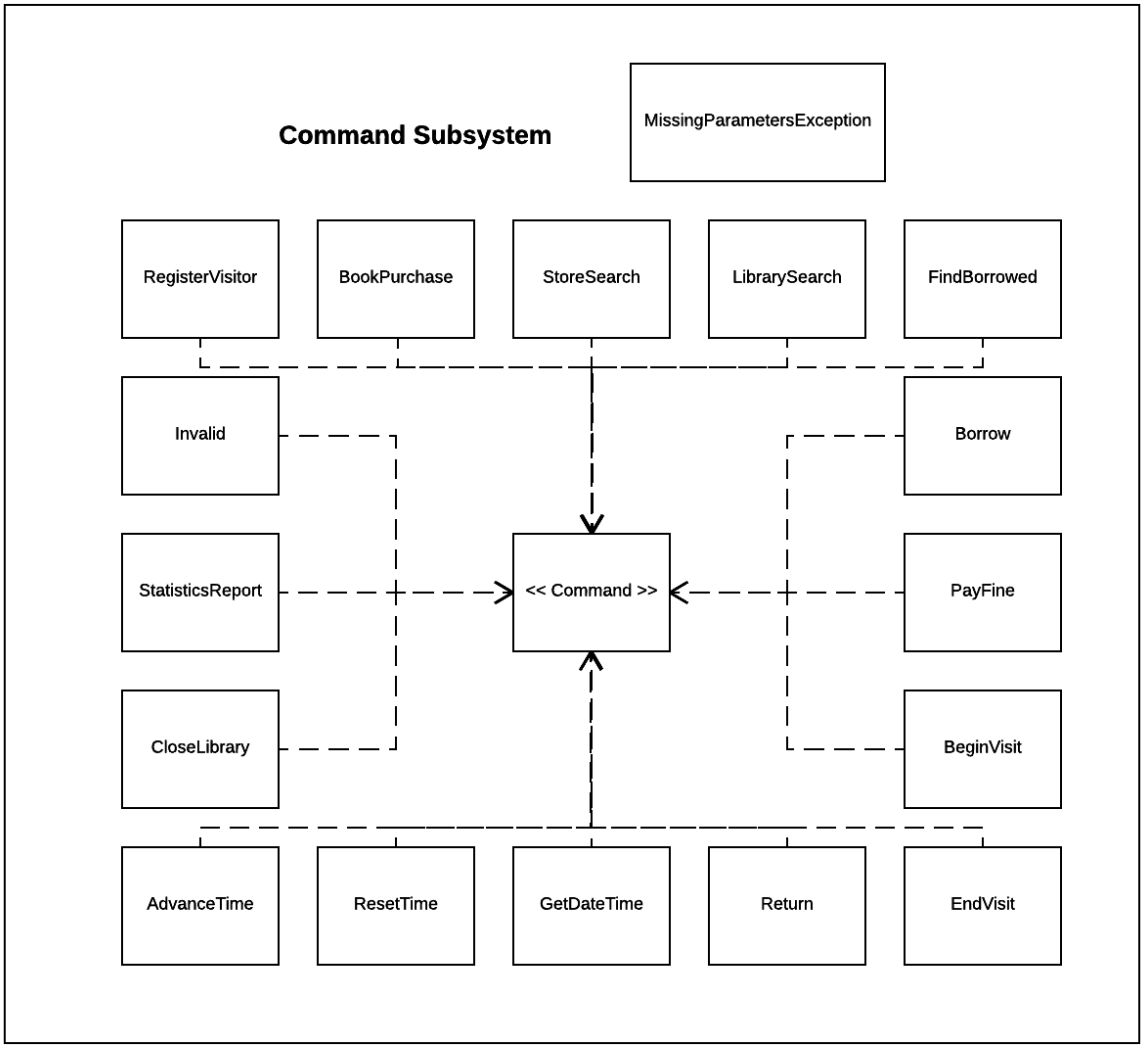


SUBSYSTEM DESIGN

COMMAND SUBSYSTEM

The Command Subsystem is not only an individual subsystem, but is an implementation of the command design pattern. The system receives a request and a Command class is generated based on the specific request. Each class inherits from the interface, Command, and implements an execute method that tells the system to access or modify the data. Through the command a response is returned and handled by the CommandController in the Controllers Subsystem.

We implemented the Command pattern due to its ability to decouple senders and receivers throughout the request execution process. Also, its natural inclination towards handling requests made it simple and straight-forward to implement for our system.



Name: Library Commands		**GoF pattern:** Command
Participants		
Class	**Role in GoF pattern**	**Participant's contribution in the context of the application**
Command	Interface	This interface is the template for the concrete commands. It declares and requires each command to implement the execute() and parseResponse() methods. These methods are common to all commands and do not share a common implementation. Each concrete command initializes by retrieving relevant information from an input string. The parseResponse() method in each concrete command is used to properly format output relevant to the command.
AdvanceTime	ConcreteCommand	This class defines the steps specific to advancing the system time by some number of days and hours.
BeginVisit	ConcreteCommand	This class defines the steps specific to having a visitor begin a visit at the library.
BookPurchase	ConcreteCommand	This class defines the steps specific to purchasing a book from the bookstore to add to the library's collection.
Borrow	ConcreteCommand	This class defines the steps specific to having a visitor borrow an available book from the library's collection.
CloseLibrary	ConcreteCommand	This class defines the steps specific to closing the library at closing time.
EndVisit	ConcreteCommand	This class defines the steps specific to having a visitor end his or her visit to the library.
FindBorrowed	ConcreteCommand	This class defines the steps specific to finding the books currently borrowed from the library by a particular visitor.
GetDateTime	ConcreteCommand	This class defines the steps specific to retrieving the system date and time (which may be different from the current date and time).
Invalid	ConcreteCommand	This class defines the steps specific to informing the user an invalid command string was entered.
LibrarySearch	ConcreteCommand	This class defines the steps specific to searching the library's collection of books for books matching input criteria.
PayFine	ConcreteCommand	This class defines the steps specific to having a visitor who owes overdue book fines pay those fines.

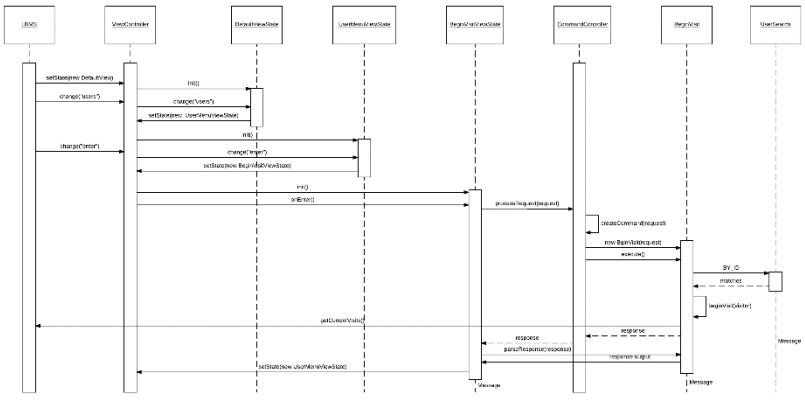
RegisterVisitor	ConcreteCommand	This class defines the steps specific to having a new visitor register with the system.
ResetTime	ConcreteCommand	This class defines the steps specific to resetting the system time to the current date and time.
Return	ConcreteCommand	This class defines the steps specific to having a visitor return a book they have borrowed.
StatisticsReport	ConcreteCommand	This class defines the steps specific to generating a report of the current state of the library.
StoreSearch	ConcreteCommand	This class defines the steps specific to searching the bookstore for books available for purchase by the library which match input criteria.
LBMS	Reciever	This class contains the state of the system and therefore receives the actions executed by the commands.

Deviations from the standard pattern:

- The command interface includes another method besides execute, parseResponse(), which aids in the formatting of relevant output from the commands execution to be displayed in the text-based user interface.

Requirements being covered:

- The application must perform several different actions relating to application content.
- All commands can be treated identically from the outside since they all have a similar pattern of creation and execution. The execution is always handled in a method named execute() and contains all steps required to properly perform the action.

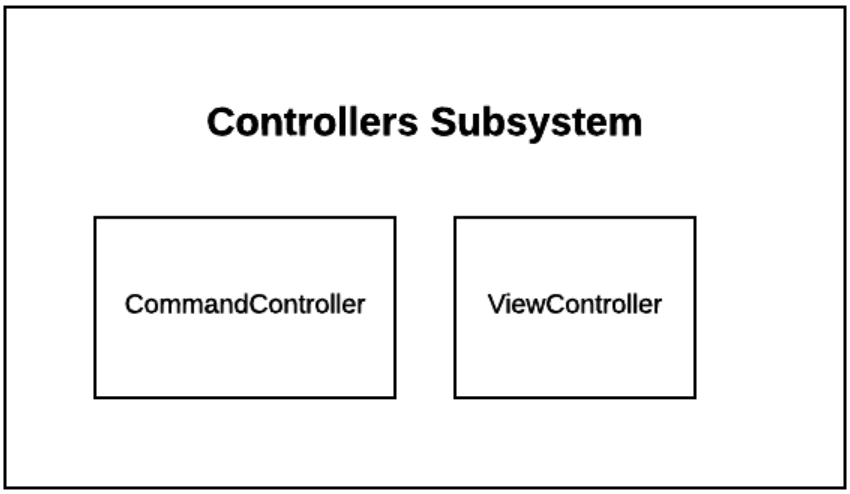


Link: https://www.lucidchart.com/invitations/accept/7cb6370b-6914-46c4-be52-64d4075ae51c

CONTROLLERS SUBSYSTEM

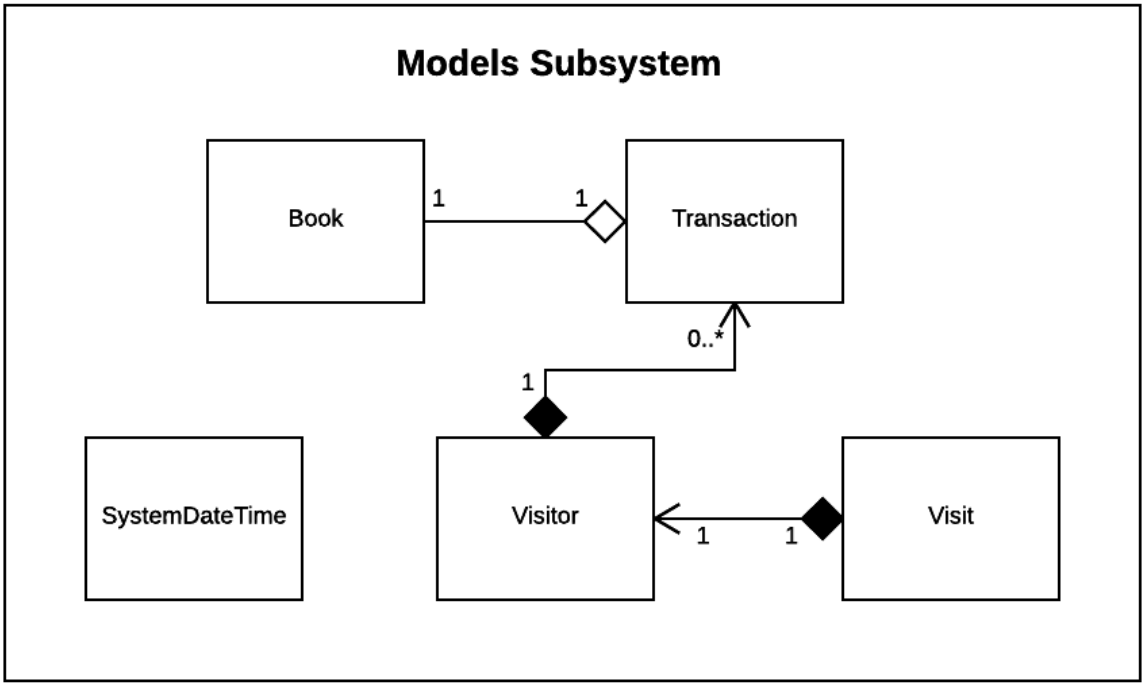
The Controllers Subsystem acts as an element of the MVC architecture. It handles how not only the views interact with the model, but also how the requests interact with the model. CommandController takes in requests and generates specifics commands that execute and return a response. Based on the Command generated the view is updated by the ViewController to match the now updated model and specific response generated.

We chose this architecture due to the ability to decouple the view from the model and the Command processing from the model. Additionally, it will allow the addition of a GUI without modification to the model, for future iterations of the system.



MODELS SUBSYSTEM

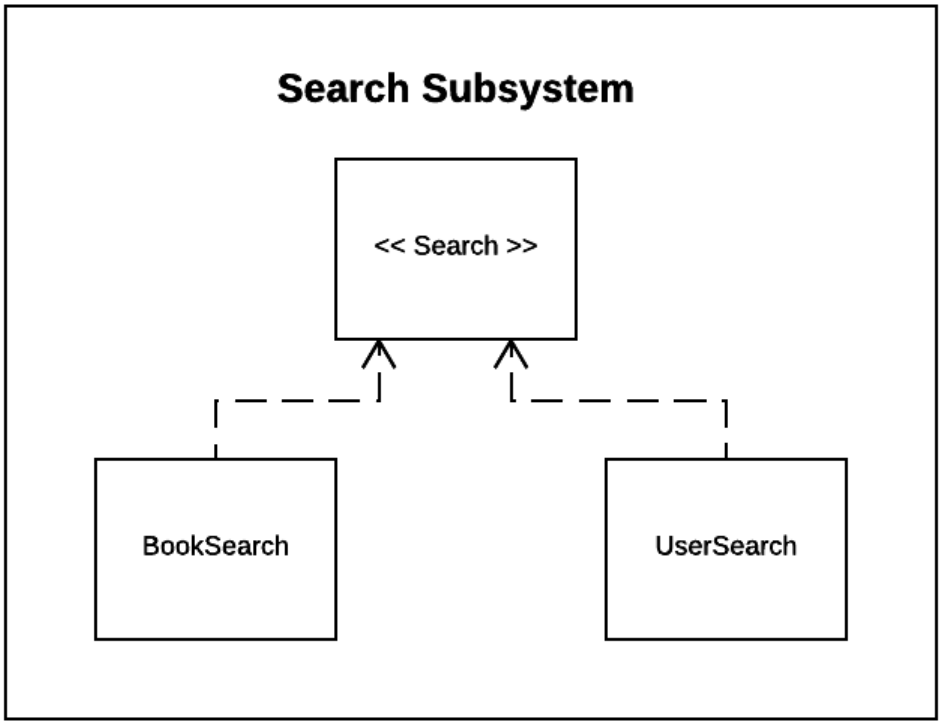
The Models Subsystem is just a grouping of all the models we implemented and their interactions. Each class stores different data required by the system, and all inherit from the Serializable interface, allowing them to persist across startups. One class, SystemDateTime, implements the Singleton pattern, since there will only ever be one instance of SystemDateTime.



Name: System Clock		**GoF pattern: Singleton**
Participants		
Class	**Role in GoF pattern**	**Participant's contribution in the context of the application**
SystemDateTime	Singleton	This class is responsible for keeping track of the system time for the library book management system. It is run on a separate thread to avoid an incorrect time due to processing of the rest of the program. When the system is started a new system date time object is created, after that the instance of the first created one is returned to follow the singleton pattern.
Deviations from the standard pattern: No deviations.		
Requirements being covered: Only one instance of a class can be given at any time, there can only be one clock for the system.		

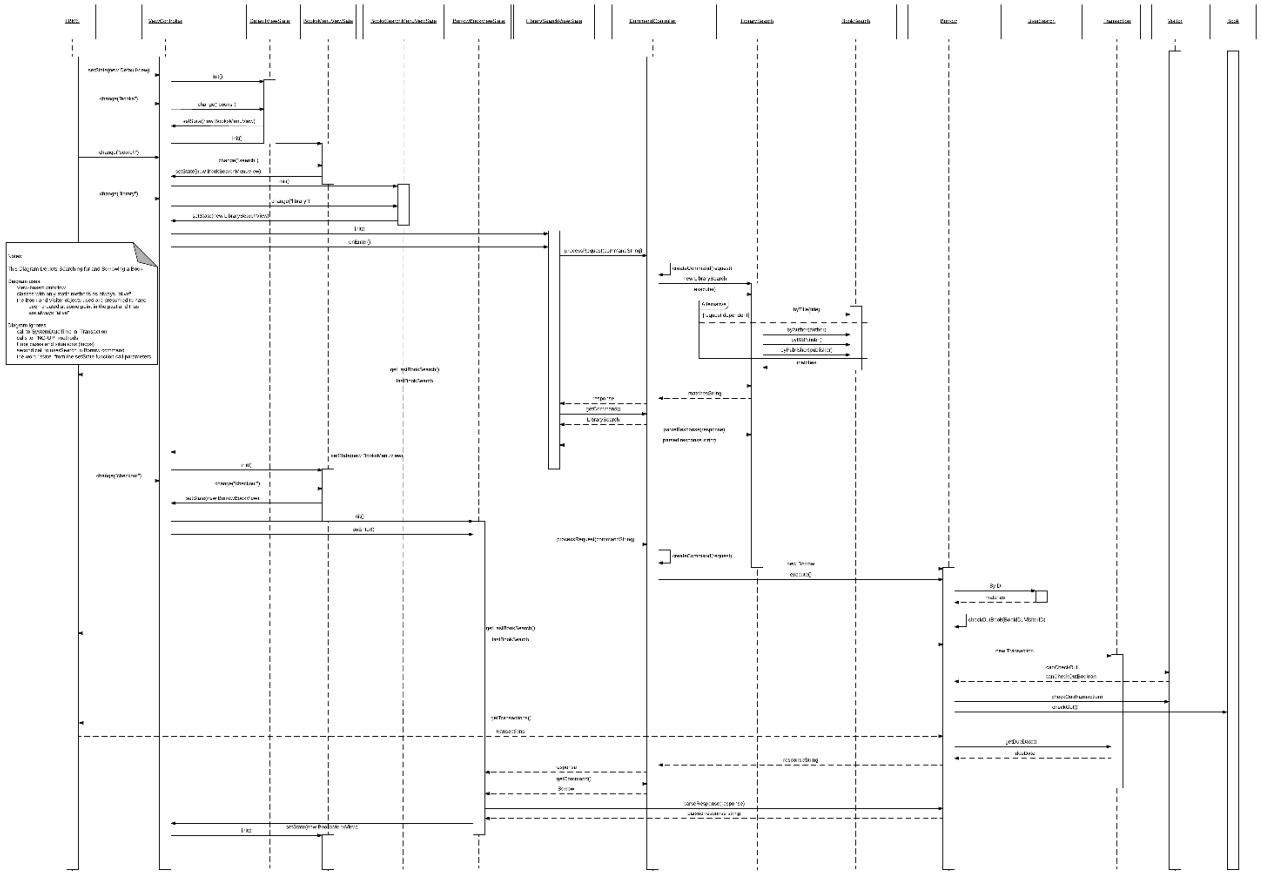
SEARCH SUBSYSTEM

The Search Subsystem implements the Strategy design pattern, inheriting from a single interface, Search, and implementing different algorithms in each subclass. The strategy design patterns allow us to add additional searching algorithms quickly and efficiently, without having to modify the rest of the system.



Name: Search		**GoF pattern: Strategy**
Participants		
Class	**Role in GoF pattern**	**Participant's contribution in the context of the application**
Search	Interface	This interface is used to declare the search() and findFirst() methods to be implemented in the concrete implementations. These methods are common to all searches and contain the ability to find all and find one object that matches given criteria, respectively.
BookSearch	ConcreteStrategy	This class contains the steps specific to searching for a book object in the system.
UserSearch	ConcreteStrategy	This class contains the steps specific to searching for a user object in the system.
Deviations from the standard pattern: • This implementation made use of Java enums to contain different ways of searching for a particular object type within the same class, while keeping the		

search for different types of objects in separate classes.
Requirements being covered: • The application must provide the ability to search for objects in different ways, depending on user input. • Each class implementation provides different ways of searching while representing a specific instance of the general action: search.

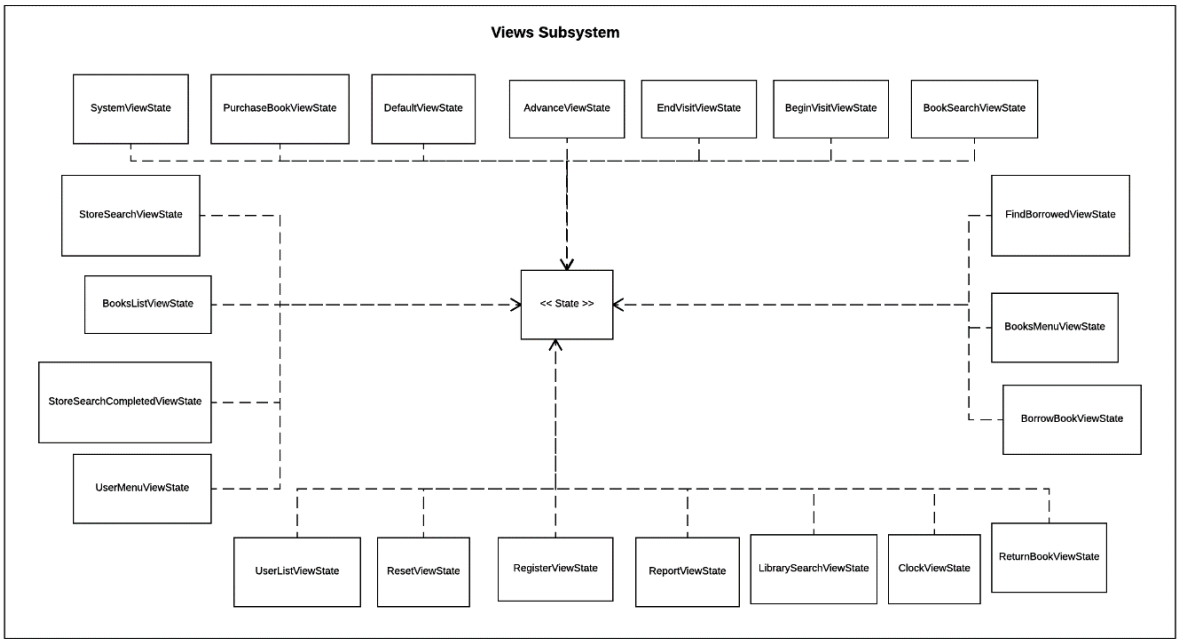


Link: https://www.lucidchart.com/invitations/accept/18444757-7632-4977-873a-c3e3ba822b97

VIEWS SUBSYSTEM

The Views Subsystem plays two crucial roles in our system. It acts as the View element from our MVC architecture, but also implements the State pattern. Each view inherits from an interface, State, and implements its methods. Each state generated is based on a given response from the system and determined by ViewController.

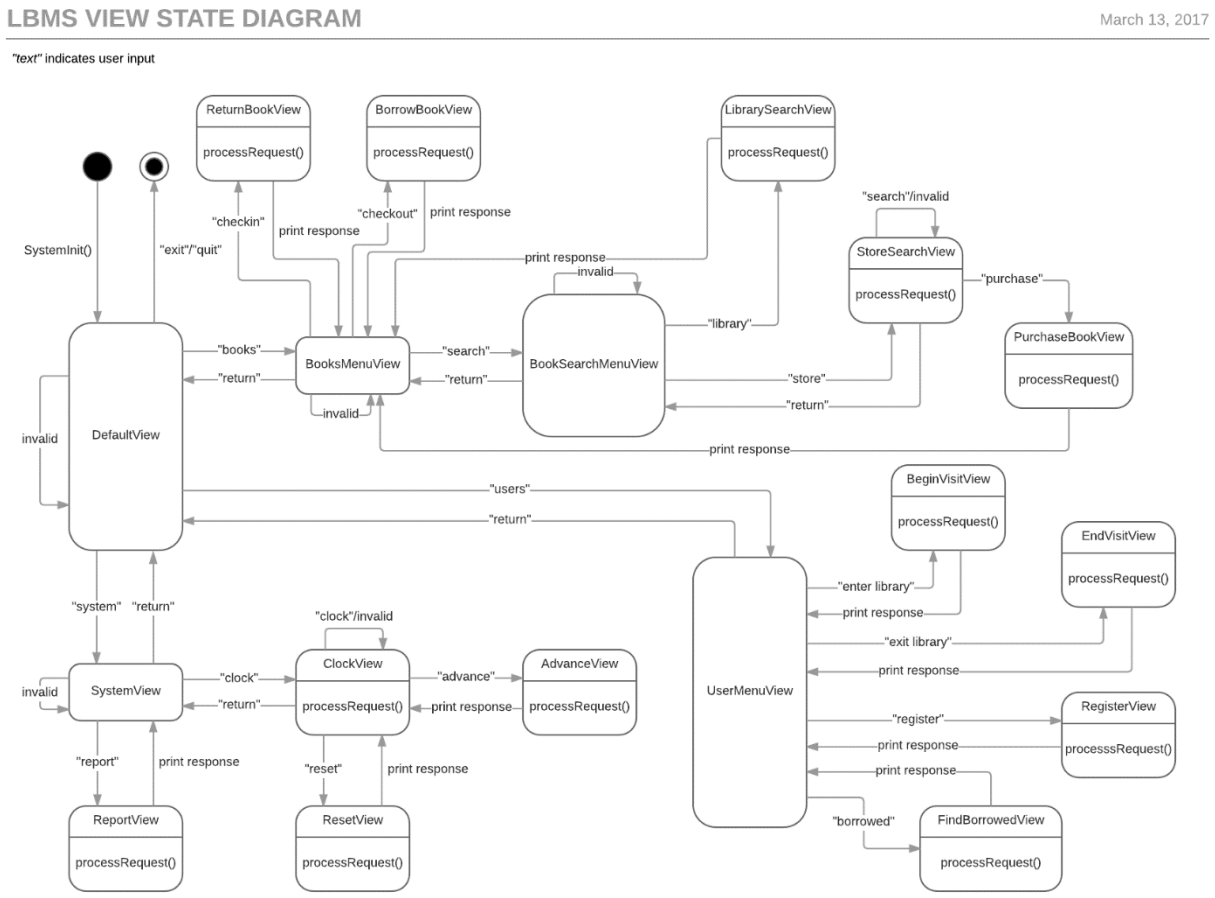
We chose to combine the State design pattern and the view element from MVC to create what we call, ViewState. It was natural to combine these two because that only one view will ever exist at a time. Additionally, we can easily add additional views, without modifying the reset of the system since the Views Subsystem is decoupled with all the models.



Name: Views		**GoF pattern: State**
Participants		
Class	**Role in GoF pattern**	**Participant's contribution in the context of the application**
State	Interface	This is the interface for all the state classes, it requires a state class to have an init(), onEnter(), change(), and it has a default flush method. Although every class may not use these methods, they must have them.
AdvanceViewState	ConcreteState	This class is used for advancing the system clock by days and hours
BeginVisitViewState	ConcreteState	This class is used for starting a visit by a registered visitor that is not already in

		the library.
BookSearchMenuViewState	ConcreteState	This class is used for searching for books from the book store.
BooksListViewState	ConcreteState	This class is used for listing the books that a book search found.
BooksMenuViewState	ConcreteState	This class is used for selecting which action the user wished to take when dealing with the books.
BorrowBookViewState	ConcreteState	This class is used for borrowing a book with a registered visitor that does not have any overdue fines.
ClockViewState	ConcreteState	This class is used for showing the system clock to the user because it may differ from real time if the time simulation is advanced.
DefaultViewState	ConcreteState	This is the default class for a view that is set on the system start.
EndVisitViewState	ConcreteState	This class is used when a registered visitor who enters the library has now exited and their visit must be recorded.
FindBorrowedViewState	ConcreteState	This class finds the borrowed books for a specific visitor so that they can return them and determine if they are eligible to check out another book.
LibrarySearchViewState	ConcreteState	This class is used for searching the books in the library that have been purchased.
PurchaseBookViewState	ConcreteState	This class is used for the library purchasing books from the book store and adding them to their inventories.
RegisterViewState	ConcreteState	This class is used to register visitors in the library.
ReportViewState	ConcreteState	This class is used to generate the report for the library including any necessary system statistics.
ResetViewState	ConcreteState	This class is used to reset the system time, it should only be used in testing, not real world usage of the system.
ReturnBookViewState	ConcreteState	This class is used when a registered visitor wished to return a borrowed book, the visitor must not have any outstanding fines with the library from overdue books before returning one.
StoreSearchViewState	ConcreteState	This class is used to search books from the book store in order for the library to purchase them.

SystemViewState	ConcreteState	This class is used to view information about the library book management system.
UserListViewState	ConcreteState	This class is used to list the registered users so that they can be viewed for system testing.
UserMenuViewState	ConcreteState	This class is used to view possible actions regarding users in the library book management system.
Deviations from the standard pattern: The only deviation from the standard pattern is the changed method names used.		
Requirements being covered: The system only has one given state at a time and the system can only complete certain actions from any given state.		



Link: https://www.lucidchart.com/invitations/accept/c1952095-d04d-4349-b7fd-9182b5be4b12

APPENDIX

MAIN

Class: LBMS	
Responsibilities: The LBMS class is responsible for storing all info related to the library such as the books, visitors, transactions, etc. This class is capable of saving updated data through serialization after shutdown and deserialization during startup. It is also responsible for taking in user input and parsing books.txt to create book objects that are available to buy from the book store. When the LBMS is closed, some commands are not allowed to execute.	
Collaborators	
Uses: CommandController, View Controller, DefaultViewState, Book, SystemDateTime, Visit, Transaction, Visitor	**Used by:** BeginVisit, BookPurchase, Borrow, EndVisit, FindBorrowed, LibrarySearch, RegisterVisitor, Return, StatisticsReport, StoreSearch, Visitor, BookSearch, UserSearch, BooksListViewState, UserListViewState
Author: Team B	

COMMAND

Class: Command	
Responsibilities: All commands used to change the info stored in the library inherit from this interface. Each class inheriting from this interface implements a execute() and parseResponse() method. The execute() method manipulates the data according to the input while parseResponse() produces the proper output to the user in order to present the result in a clear manner.	
Collaborators	
Uses: None	**Used by:** AdvanceTime, BeginVisit, BookPurchase, Borrow, CloseLibrary, EndVisit, FindBorrowed, GetDateTime, Invalid, LibrarySearch, PayFine, RegisterVisitor, ResetTime, Return, StatisticsReport, StoreSearch
Author: Team B	

Class: AdvanceTime	
Responsibilities: The AdvanceTime class manually moves the time forward based on the number of days and/or number of hours inputted by the user. The user can choose from 0-7 days and 0-23 hours to advance the time. If the time was successfully advanced, the output will display a success message. Otherwise, it will output a failure message with why it failed to advance the time.	
Collaborators	
Uses: Command, SystemDateTime	**Used by:** CommandController
Author: Team B	

Class: BeginVisit	
Responsibilities: This class is responsible for adding a visit to the LBMS when given a proper visitorID. If the visitorID given does not exist within the LBMS or the visitor with the visitorID is already in the library, an error message will be displayed to the user. If the visit was successfully started, a success message will be displayed with the date and time the visit started.	
Collaborators	
Uses: Command, LBMS, SystemDateTime, Visit, Visitor, UserSearch, MissingParameterException	**Used by:** CommandController
Author: Team B	

Class: BookPurchase	
Responsibilities: The BookPurchase class allows books from the last store search to be bought for library inventory. Using temporary book ID's given to each book from the last search, the user can choose which books to buy along with the quality. If the user uses an ID that does not apply to any book returned from the last search, a failure message will be output. If the LBMS already has the book in its inventory (a hashmap), the book object will be added to the values of the proper ISBN (the key). Otherwise, a new key with the right ISBN is created along with the book as its value.	
Collaborators	
Uses: Command, LBMS, Book, BookSearch, MissingParametersException	**Used by:** CommandController
Author: Team B	

Class: Borrow	
Responsibilities: The Borrow class enables books returned from the last library book search to be checked out of the library. Checking out books are not permitted if the visitorID provided does not exist, the visitor has an outstanding fine, the visitor already has five books checked out, and/or the bookIDs given do not exist. If the books are borrowed with no errors, a success message will be returned along with a due date, which is a week from the date the was borrowed. Otherwise, an error message is returned stating why the book(s) could not be borrowed.	
Collaborators	
Uses: Command, LBMS, Book, SystemDateTime, Transaction, Visitor, UserSearch, MissingParametersException	**Used by:** CommandController
Author: Team B	

Class: CloseLibrary	
Responsibilities: If the library is closed, some commands such as borrowing books and beginning a visit are not possible so when a user tries to use these commands during closed hours, CloseLibrary takes over and notifies the user that the library is closed and the original command will not work.	
Collaborators	
Uses: Command	**Used by:** CommandController
Author: Team B	

Class: EndVisit	
Responsibilities: EndVisit removes a visitor from the library. It also adds a visit to the total visits that the LBMS records. It won't work if the visitorID does not exist or the visitorID is not in the library. If it successfully ends a visit, the time the visit ends and the duration of the visit is returned to the user.	
Collaborators	
Uses: Command, LBMS, SystemDateTime, Visit, Visitor, UserSearch	**Used by:** CommandController
Author: Team B	

Class: FindBorrowed	
Responsibilities: Given a valid visitorID, this command presents the number of books the visitor with the visitorID has borrowed and which specific books were borrowed. If the visitorID does not exist, then an error message is returned. This class prepares the returned books for the “return” command as they are given a temporary ID.	
Collaborators	
Uses: Command, LBMS, Book, Transaction, Visitor, BookSearch, UserSearch	**Used by:** CommandController
Author: Team B	

Class: GetDateTime	
Responsibilities: GetDateTime simply outputs the current LBMS date and time.	
Collaborators	
Uses: Command, SystemDateTime	**Used by:** CommandController
Author: Team B	

Class: Invalid	
Responsibilities: This class handles inputted “commands” that the LBMS is not supposed to accept. A message stating that the “command” is invalid is output if the user enters a false command.	
Collaborators	
Uses: Command	**Used by:** ControllerCommand
Author: Team B	

Class: LibrarySearch	
Responsibilities: This class is responsible for returning specific books according to user input. Specifications include the isbn, title, authors, publisher, and sort order but the user can omit some fields of the search if they choose to do so by using a "*". The search results can only be ordered by title, publish date, and availability. If the user inputs a different kind of sort method, an error message is output. Each book returned from the search are prepared for borrowing by being given a temporary ID.	
Collaborators	
Uses: Command, LBMS, Book, BookSearch, MissingParametersException	**Used by:** CommandController
Author: Team B	

Class: MissingParametersException	
Responsibilities: This class is an exception class used when a given request is missing required parameters. If the user does not input all needed parameters for a command, the exception comes in and returns a message stating that the request is missing some parameters.	
Collaborators	
Uses: None	**Used by:** BeginVisit, BookPurchase, Borrow, EndVisit, LibrarySearch, RegisterVisitor, StatisticsReport, StoreSearch
Author: Team B	

Class: PayFine	
Responsibilities: PayFine allows visitors' fines to be paid. Given a valid amount and visitorID from the user, the amount will be subtracted from the visitor's total balance and the remaining balance will be returned. If the visitorID given does not exist, an error message will appear. An error message also appears if the entered amount to pay is negative or exceeds the visitor's total balance.	
Collaborators	
Uses: Command, UserSearch	**Used by:** CommandController
Author: Team B	

Class: RegisterVisitor	
Responsibilities: This class is responsible for adding a new visitor to the LBMS. With a name, address, and phone number given, the information is stored in the LBMS (in a hashmap) and the date and time of the register is returned. If the name, address, and phone number of an already registered visitor is inputted again, an error message appears. However, registering a visitor can be successful as long as at least one field is different from all already registered visitors.	
Collaborators	
Uses: Command, LBMS, SystemDateTime, Visitor, UserSearch, MissingParametersException	**Used by:** CommandController
Author: Team B	

Class: ResetTime	
Responsibilities: This class is used for testing purposes. It automatically updates the date and time stored in the LBMS to the current date and time.	
Collaborators	
Uses: Command, SystemDateTime	**Used by:** CommandController
Author: Team B	

Class: Return	
Responsibilities: This class allows checked out books to be returned. Given a valid visitorID and ID returned from the “borrowed” command, the borrowed book will be added back into the LBMS inventory. If the book is returned overdue, a fine will be added to the visitor’s total balance and the IDs of the overdue books are returned. Error messages can occur if the given visitorID or book IDs do not exist. Even if one book ID is not valid, the whole command is cancelled.	
Collaborators	
Uses: Command, LBMS, Book, SystemDateTime, Transaction, Visitor, UserSearch	**Used by:** CommandController
Author: Team B	

Class: StatisticsReport	
Responsibilities: The StatisticsReport class provides different stats on library usage. This includes the total number of books in the library, total number of registered visitors, the average length of a visit, the number of books purchased, the amount of fines collected, and the amount of fines that still need to be collected. If a certain number of days is input, the report only includes the stats covering those number of days. If the number of days is omitted, the report covers all stats recorded since the beginning of the simulation.	
Collaborators	
Uses: Command, LBMS, Book, SystemDateTime, Visit, Visitor, MissingParametersException	**Used by:** CommandController
Author: Team B	

Class: StoreSearch	
Responsibilities: The responsibility of this class is to find books in the book store that fit inputted specifications. These specifications include the title, authors, ISBN, publisher, and sort order. The results can only be sorted by title and publish date (most recent first). If the user tries to sort the results in a different way, an error message is returned stating that the mentioned method of sorting is invalid. Like in LibrarySearch, any field may be omitted by inputting a "*" in its place. The books returned from the search are prepared for purchase from the store by being given a temporary ID.	
Collaborators	
Uses: Command, LBMS, Book, BookSearch, MissingParametersException	**Used by:** CommandController
Author: Team B	

CONTROLLERS

Class: CommandController	
Responsibilities: This class is responsible for the creation and execution of commands. These commands are created based on a given request string. With the first word of the request, CommandController is able to determine which command to create. This class can detect if the request does not have enough inputted parameters for the command and if the request is incomplete (due to the absence of an ending semicolon). From a valid request string, this class creates a response string that describes the results of the command. The majority of the response string is generated in the commands themselves through their execution. This class cleans and completes the response strings and ultimately returns the properly formatted response.	
Collaborators	
Uses: AdvanceTime, BeginVisit, BookPurchase Borrow, CloseLibrary, EndVisit, FindBorrowed, GetDateTime, Invalid, LibrarySearch, PayFine, RegisterVisitor, ResetTime, Return, StatisticsReport, StoreSearch	**Used by:** LBMS, AdvanceViewState, BeginVisitViewState, BorrowBookViewState, ClockViewState, EndVisitViewState, FindBorrowedViewState, LibrarySearchViewState, PurchaseBookViewState, RegisterViewState, ReportViewState, ResetViewState, ReturnBookViewState, StoreSearchViewState
Author: Team B	

Class: ViewController	
Responsibilities: The responsibility of this class is to hold the current state of the system and to change the view/state to the proper view/state based on a command. (It changes the state by calling the change() method of the current state. In this way, the children states handle the change.)	
Collaborators	
Uses: State	**Used by:** LBMS, AdvanceViewState, BeginVisitViewState, BookSearchMenuViewState, BooksListViewState, BooksMenuViewState, BorrowBookViewState, ClockViewState, DefaultViewState, EndVisitViewState, FindBorrowedViewState, LibrarySearchViewState, PurchaseBookViewState, RegisterViewState, ReportViewState, ResetViewState, ReturnBookViewState, StoreSearchViewState, SystemViewState, UserListViewState, UserMenuViewState
Author: Team B	

MODELS

Class: Book	
Responsibilities: This class holds the state and behaviors for a book object. A book has a title, publisher, ISBN, publish date, and a purchase date. The total number of copies of the book that is in library is also tracked along with the number of copies checked out. A book can be purchased, checked out/borrowed, and returned.	
Collaborators	
Uses: None	**Used by:** LBMS, BookPurchase, Borrow, FindBorrowed, LibrarySearch, Return, StatisticsReport, StoreSearch, BooksListViewState
Author: Team B	

Class: SystemDateTime	
Responsibilities: This class is a custom made datetime class for the LBMS. It is used to store the system date and time separate from the real current date and time. Through this class, the system time can be advanced.	
Collaborators	
Uses: None	**Used by:** LBMS, AdvanceTime, BeginVisit, Borrow, EndVisit, GetDateTime, RegisterVisitor, ResetTime, Return, StatisticsReport
Author: Team B	

Class: Transaction	
Responsibilities: This class has the state and behaviors for a transaction object. A transaction holds the ISBN of the book checked out, the visitorID of the visitor who checked out the book, the date of the checkout, and the date the book checked out is due. It is also able to calculate the fines that an overdue book has.	
Collaborators	
Uses: None	**Used by:** LBMS, Borrow, FindBorrowed, Return
Author: Team B	

Class: Visit	
Responsibilities: This class has the state and behaviors of a Visit object. A visit object possesses a visitor, the date and time the visit started, the time the visit ended, and the duration of the visit. It's only behavior is ending a visit where it calculates when the visitor leaves and the duration of visit. It also removes the visitor from the library in the LBMS.	
Collaborators	
Uses: None	**Used by:** LBMS, BeginVisit, EndVisit, StatisticsReport
Author: Team B	

Class: Visitor	
Responsibilities: This class holds the state and behavior of a Visitor object. The state includes the name, address, phone number and visitorID of the visitor. It also tracks what books they have checked out, whether they are currently in the library or not, and their fines. A visitor can only check out books as long as they have less than 5 books checked out and they have no outstanding fines. Of course, the visitor is able to return their borrowed books where if the book is overdue, their total fine increases based on the fine stored in the transaction.	
Collaborators	
Uses: LBMS	**Used by:** LBMS, BeginVisit, Borrow, EndVisit, FindBorrowed, RegisterVisitor, Return, StatisticsReport, UserListViewState
Author: Team B	

SEARCH

Class: Search	
Responsibilities: Search is a generic interface to facilitate the two classes that implement it (BookSearch and UserSearch). Its main method is search() which is supposed to find objects depending on the given criteria.	
Collaborators	
Uses: None	**Used by:** BookSearch, UserSearch
Author: Team B	

Class: BookSearch	
Responsibilities: BookSearch implements the Search interface to find books based on given specifications. It allows books to be searched in different ways such as authors, ISBN, title, and publisher. Two types of searches are implemented in this class: search, which searches for books in the library and searchBookstoBuy which searches for books in the book store.	
Collaborators	
Uses: Search, LBMS, Book	**Used by:** BookPurchase, FindBorrowed, LibrarySearch, StoreSearch
Author: Team B	

Class: UserSearch	
Responsibilities: UserSearch is similar to BookSearch except it searches for visitors instead of books. It can search for visitors by id, name, address, or phone number.	
Collaborators	
Uses: Search, LBMS, Visitor	**Used by:** BeginVisit, Borrow, EndVisit, FindBorrowed, PayFine, RegisterVisitor, Return
Author: Team B	

VIEWS

Class: State	
Responsibilities: All views used to change the info presented in the test-based user interface inherit from this interface. The views in the system represent the states the system can exist in. The system will accept different commands and process those commands differently depending on the current state. Each class inheriting from this interface implements init(), onEnter(), and change() methods. The init() method initializes the view usually by updating the menu options presented to the user. In states where more information is required, the init() method is also used to gather user input. The onEnter() method is used to process a command if one is associated with the state. The change() method is used to change state depending on user input. The changing of states is handled in the individual child states through this method.	
Collaborators	
Uses: None	**Used by:** AdvanceViewState, BeginVisitViewState, BookSearchMenuViewState, BooksListViewState, BooksMenuViewState, BorrowBookViewState, ClockViewState, DefaultViewState, EndVisitViewState, FindBorrowedViewState, LibrarySearchViewState, PurchaseBookViewState, RegisterViewState, ReportViewState, ResetViewState, ReturnBookViewState, StoreSearchViewState, SystemViewState, UserListViewState, UserMenuViewState, ViewController
Author: Team B	

Class: AdvanceViewState	
Responsibilities: This class generates the UI for advancing the time. The user is prompted for the number of days and hours to advance the system clock. This class then creates and executes an AdvanceTime command by sending an appropriate request string to the CommandController.	
Collaborators	
Uses: State, ViewController, CommandController	**Used by:** ClockViewState
Author: Team B	

Class: BeginVisitViewState	
Responsibilities: This class generates the UI for starting a library visit. The user is prompted for the visitorID of the visitor who would like to enter the library. This class then creates and executes a BeginVisit command by sending an appropriate request string to the CommandController.	
Collaborators	
Uses: State, ViewController, CommandController, UserMenuViewState	**Used by:** UserMenuViewState
Author: Team B	

Class: BookSearchMenuViewState	
Responsibilities: This class generates the UI for searching. There are two kinds of searching which can be performed: searching the library and searching the bookstore. A menu is displayed here which indicates these two options to the user. The state of the system is then changed based on user input.	
Collaborators	
Uses: State, ViewContoller, LibrarySearchViewState, StoreSearchViewState, BooksMenuViewState	**Used by:** BooksMenuViewState, StoreSearchViewState
Author: Team B	

Class: BooksListViewState	
Responsibilities: This class is used for testing. It allows the user to list all the books in the library through the UI.	
Collaborators	
Uses: State, ViewController, Book, BooksMenuViewState, LBMS	**Used by:** BooksMenuViewState
Author: Team B	

Class: BooksMenuViewState	
Responsibilities: This class generates the UI which allows the user to navigate to functionality relating to books. A menu is displayed with options to search, checkout, checkin, and list the library books. The state of the system is then changed based on user input.	
Collaborators	
Uses: State, ViewController, BookSearchMenuViewState, BooksListViewState, ReturnBookViewState, DefaultViewState, BorrowBookViewState	**Used by:** BookSearchMenuViewState, DefaultViewState
Author: Team B	

Class: BorrowBookViewState	
Responsibilities: This class generates the UI for borrowing a library book. The user is prompted for the visitorID of the visitor who would like to borrow a book, the ID(s) of the book they are borrowing (from the latest search). This class then creates and executes a Borrow command by sending an appropriate request string to the CommandController.	
Collaborators	
Uses: State, ViewController, CommandController, BooksMenuViewState	**Used by:** BooksMenuViewState
Author: Team B	

Class: ClockViewState	
Responsibilities: This class generates the UI relating to the system clock. In particular, the system date and time is displayed here along with a menu of actions the user can take. These actions include advancing time, resting the clock (for testing purposes), or viewing the current system time. This class creates and executes a GetDateTime command by sending an appropriate request string to the CommandController in order to display the system time. The state of the system is then changed based on user input.	
Collaborators	
Uses: State, ViewController, CommandController, AdvanceViewState, ResetViewState, SystemViewState	**Used by:** AdvanceViewState, ResetViewState, SystemViewState
Author: Team B	

Class: DefaultViewState	
Responsibilities: This class generates the UI for the general system menu. This is the state which the system enters on startup. A welcome message is displayed along with a menu of options which allow the user to navigate to a specific part of the system including the books, users, and system menus. The state of the system is then changed based on user input.	
Collaborators	
Uses: State, ViewController, BooksMenuViewState, UserMenuViewState, SystemViewState	**Used by:** LBMS, BooksMenuViewState, UserMenuViewState, SystemViewState
Author: Team B	

Class: EndVisitViewState	
Responsibilities: This class generates the UI for ending a library visit. The user is prompted for the visitorID of the visitor who would like to leave the library. This class then creates and executes an EndVisit command by sending an appropriate request string to the CommandController.	
Collaborators	
Uses: State, ViewController, CommandController, UserMenuViewState	**Used by:** UserMenuViewState
Author: Team B	

Class: FindBorrowedViewState	
Responsibilities: This class generates the UI for displaying the books a visitor has borrowed. The user is prompted for the visitorID of the visitor whose borrowed books the user would like to see. This class then creates and executes a FindBorrowed command by sending an appropriate request string to the CommandController.	
Collaborators	
Uses: State, ViewController, CommandController, UserMenuViewState	**Used by:** UserMenuViewState
Author: Team B	

Class: LibrarySearchViewState	
Responsibilities: This class generates the UI for searching the libraries collection of books. The user is prompted for the title, author(s), ISBN, and publisher of the book for which they are searching. A title and author is required for the search while the ISBN and publisher are optional. A sort-order is another optional user input which dictates the order in which the resulting books are displayed. This class then creates and executes a LibrarySearch command by sending an appropriate request string to the CommandController.	
Collaborators	
Uses: State, ViewController, CommandController, BooksMenuViewState	**Used by:** BookSearchMenuViewState, BooksMenuViewState
Author: Team B	

Class: PurchaseBookViewState	
Responsibilities: This class generates the UI for purchasing a book for the library. The user is prompted for the quantity of each book to purchase and the ID(s) of the book(s) they would like to buy (ID(s) assigned from last search). This class then creates and executes a BookPurchase command by sending an appropriate request string to the CommandController.	
Collaborators	
Uses: State, ViewController, CommandController, BooksMenuViewState	**Used by:** StoreSearchViewState, BooksMenuViewState
Author: Team B	

Class: RegisterViewState	
Responsibilities: This class generates the UI for registering a new visitor. The user is prompted for the name, address, and phone number of the visitor who would like to register. This class then creates and executes a RegistorVisitor command by sending an appropriate request string to the CommandController.	
Collaborators	
Uses: State, ViewController, CommandController, UserMenuViewState	**Used by:** UserMenuViewState
Author: Team B	

Class: ReportViewState	
Responsibilities: This class generates the UI for displaying a report of the library book management system. The user is prompted for the number of days he or she would like the report to include. This class then creates and executes a StatisticsReport command by sending an appropriate request string to the CommandController.	
Collaborators	
Uses: State, ViewController, CommandController, SystemViewState	**Used by:** SystemViewState
Author: Team B	

Class: ReturnBookViewState	
Responsibilities: This class generates the UI for returning a borrowed book to the library. The user is prompted for the visitorID of the visitor who is returning the book(s) and the ID(s) of the book(s) they would like to return. This class then creates and executes a Return command by sending an appropriate request string to the CommandController.	
Collaborators	
Uses: State, ViewController, CommandController, BooksMenuViewState	**Used by:** BooksMenuViewState
Author: Team B	

Class: StoreSearchViewState	
Responsibilities: This class generates the UI for searching the collection of books available for purchase. The user is prompted for the title, author(s), ISBN, and publisher of the book for which they are searching. A title is required for the search while the author(s), ISBN, and publisher are optional. A sort-order is another optional user input which dictates the order in which the resulting books are displayed. This class then creates and executes a StoreSearch command by sending an appropriate request string to the CommandController. A menu is displayed which allows the user to purchase books returned from the previous search or to search again. The state of the system is then changed based on user input.	
Collaborators	
Uses: State, ViewController, CommandController, PurchaseBookViewState, StoreSearchViewState, BookSearchMenuViewState	**Used by:** BookSearchMenuViewState
Author: Team B	

Class: SystemViewState	
Responsibilities: This class generates the UI which allows the user to navigate to functionality relating to the system in general. A menu is displayed with options to view the system clock or generate a system report. The state of the system is then changed based on user input.	
Collaborators	
Uses: State, ViewController, ClockViewState, ReportViewState, DefaultViewState	**Used by:** DefaultViewState, ReportViewState, ClockViewState
Author: Team B	

Class: UserListViewState	
Responsibilities: This class generates and displays a list of all registered visitors in the system. This class is used for testing.	
Collaborators	
Uses: State, ViewController, LBMS, Visitor, UserMenuViewState	**Used by:** UserMenuViewState
Author: Team B	

Class: UserMenuViewState	
Responsibilities: This class generates the UI which allows the user to navigate to functionality relating to library visitors. A menu is displayed with options to enter the library, exit the library, register a new user, list users, and view borrowed books. The state of the system is then changed based on user input.	
Collaborators	
Uses: State, ViewController, RegistorViewState, UserListViewState, FindBorrowedViewState, DefaultViewState, BeginVisitViewState, EndVisitViewState	**Used by:** DefaultViewState, BeginVisitViewState, EndVisitViewState, RegisterViewState, FindBorrowedViewState
Author: Team B	