Library Book Management System

Team B: SWEN-262 Design Project R2

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
2.0	04/03/2017	Fixed Grammatical Issues	Charles Barber
2.1	04/05/2017	Updated Requirements for R2 and Domain Model	Christopher Lim
2.2	04/07/2017	Removed ViewState	Charles Barber
2.3	04/08/2017	Updated UML Class Diagrams	Anthony Palumbo, Nicholas Feldman
2.4	04/08/2017	Updated UML Sequence Diagram	Anthony Palumbo, Charles Barber
2.5	04/15/2017	Added New CRC Cards	Edward Wong

2.6	*04/15/2017*	*Added Description of Library State Implementation*	*Edward Wong*
2.7	*04/16/2017*	*Added Description of Proxy Implementation*	*Nicholas Feldman*
2.8	*04/16/2017*	*Added new design patterns to document*	*Anthony Palumbo*
2.9	*04/17/2017*	*Corrected and Expanded Prose*	*Christopher Lim, Anthony Palumbo*
3.0	*04/18/2017*	*Formatted Document*	*Christopher Lim*
4.0	*04/18/2017*	*Completed Changed Document, all sections affected*	*Anthony Palumbo*

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, in a private repository until after the project is over, and implemented in Java as a desktop application. It will 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 batch file, start.bat, will be provided to set any required environment variables, perform any program specific initialization, and execute the graphical user interface for the program. Another file, startAPI.bat, will be provided to run only the API version of the project that takes in requests and prints out the responses. This version is particularly useful for other developers who may need to run a specific part of the project without the graphical user interface. On a clean exit the program will either produce or update a file named data.ser, this contains all the serialized data for the system and can be deleted to get a clean start of the program.

FEATURE REQUIREMENTS

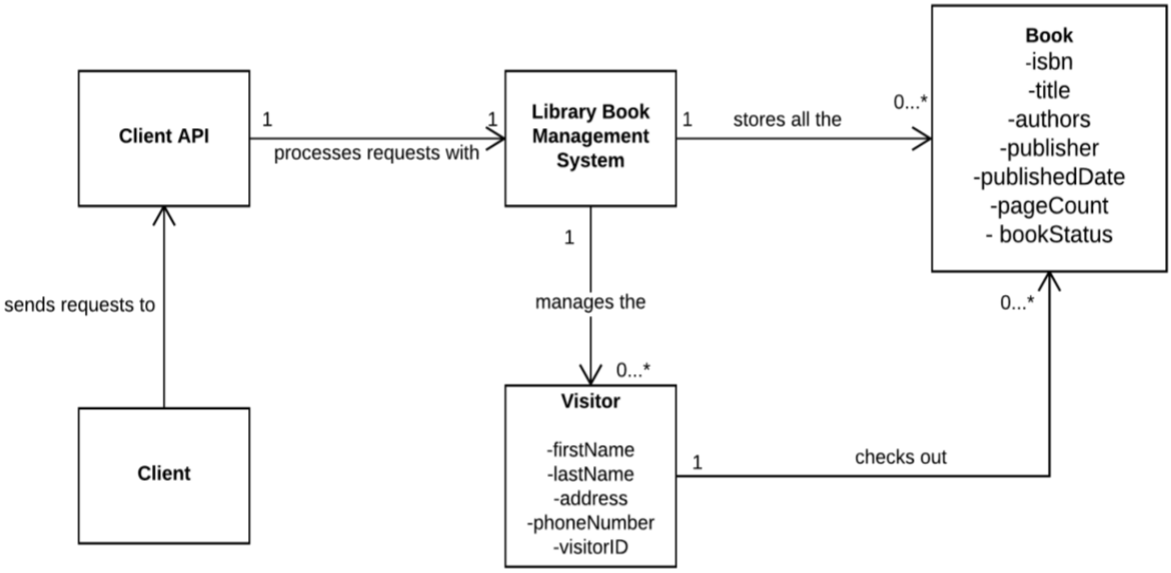
No.	*User Story Name*	*Description*
1	API	The LBMS shall use text-based requests and responses. An LBMS exchange consists of one text string sent by a client followed by one text string sent by the system. The system shall receive requests from a client as text strings. A client shall terminate request 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, phone number, and a visitor ID (a unique 10-digit ID generated upon visitor registration).
3	Visits	The LBMS shall keep track of visits by visitors. 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, and book status (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 each visitor to checkout a maximum of 5 books at a time. 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, $2.00 will be added to the initial fee for each additional week overdue, up to 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 registered at 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 and time. The LBMS system shall track the number of days that have passed while the system is in operation. The LBMS system shall allow users to move the date forward by a specified number of days. The time of day will remain unaffected. The system will also allow for the advancement of the time by a specified number of hours. 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.
12	Concurrent Client Connections	The LBMS shall support operations provided by multiple, concurrent clients. A client will be prompted to log in immediately upon establishing a connection and will be automatically logged out upon disconnecting. Each client connection will be separate in the sense that the library will handle all request and response exchanges for each client, individually. Actions taken by individual clients that change the state of the system will affect other clients (e.g. borrowing books will affect book availability for all clients).
13	Client Accounts	The system shall store client account for visitors and employees. An account represents the username, password, and role for an individual visitor. Each account will have different access permissions depending on the type (either visitor or employee). Employees will have access to the entire system while visitors who are not employees will only be able to begin a visit, end a visit, search the library, and borrow a book in addition to basic system tasks (e.g. connect, log in, log out, etc.).
14	Undo/Redo	The LBMS system shall support the ability to undo and redo actions by any user. The actions that support the undo/redo functionality are the following: Begin Visit, End Visit, Purchase Book, Borrow Book, Return Book, and Pay Fine.

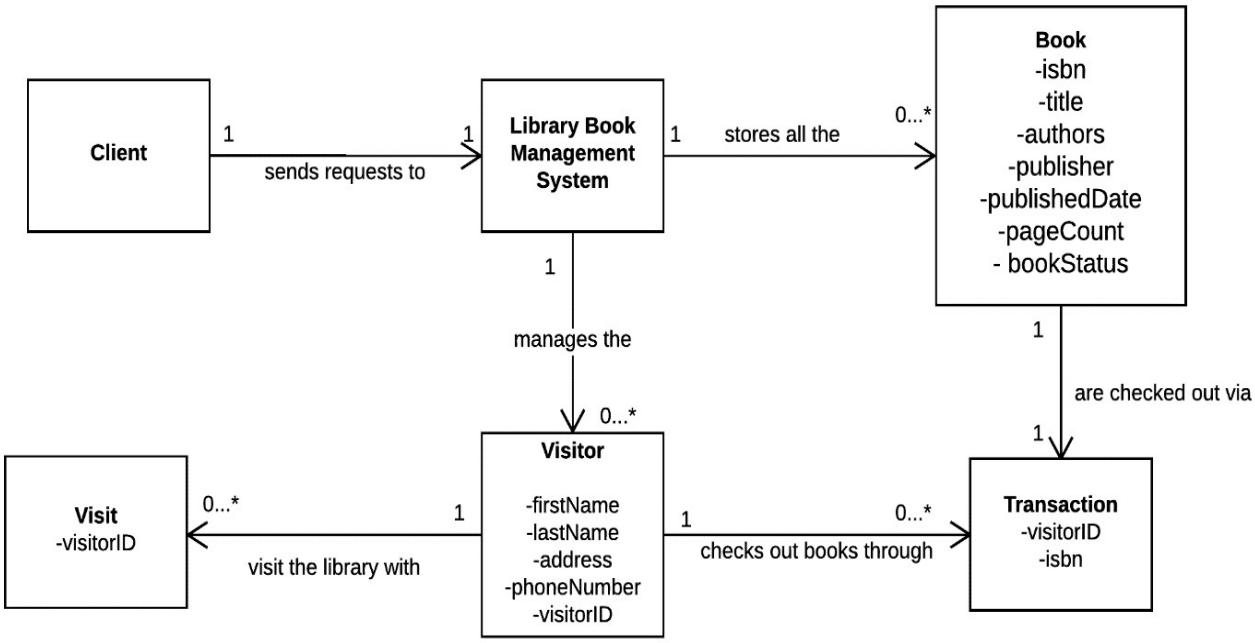
15	Purchase Book Source	The LBMS will now allow employees to choose between a provided books.txt file and Google Books as a source of books available for the library to purchase. In the case of using Google Books as a source of books, only books labeled as for sale in the US will be available for purchase.

DOMAIN MODEL

ORIGINAL



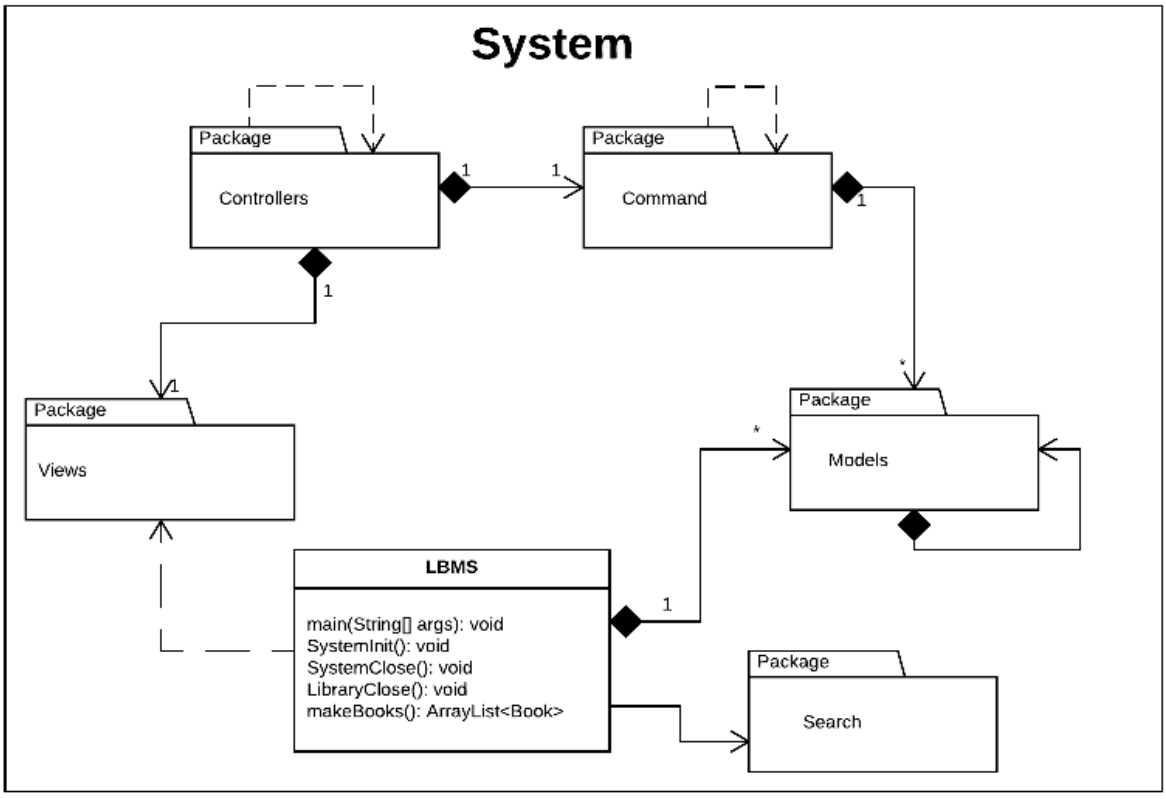
UPDATED



ARCHITECTURAL MODEL

The following model displays the interactions between each main package forming the entire system. We tried to organize the subsystems by package, however each package does not directly correlate to a subsystem. The entry point into the system is through LBMS, which is dependent on the views package as it is displayed to the user as a view. User input is sent to the controllers which interprets and handles the input and converts it into a specific command within the Command Subsystem, contained in the command package. The controllers update the view with the appropriate view once the command is executed. The Command Subsystem interacts with the models modifying and/or accessing the data. Model data are stored in memory within LBMS. This memory can be queried using the Search Subsystem, also contained within the search package, which is used for things such as a search command.

The advantages of this design include separation of concerns using a model-view-controller architecture, allowing the system to update the view at any time while keeping the interaction with the model standardized and encapsulated, and the ability to run two different mode, API and GUI, while reusing much of the code with common functionality. With the structure we have setup, it is easy to add features to the system by following the MVC pattern. The disadvantage of our system is the large number of packages and classes within them we used in order to follow design patterns.

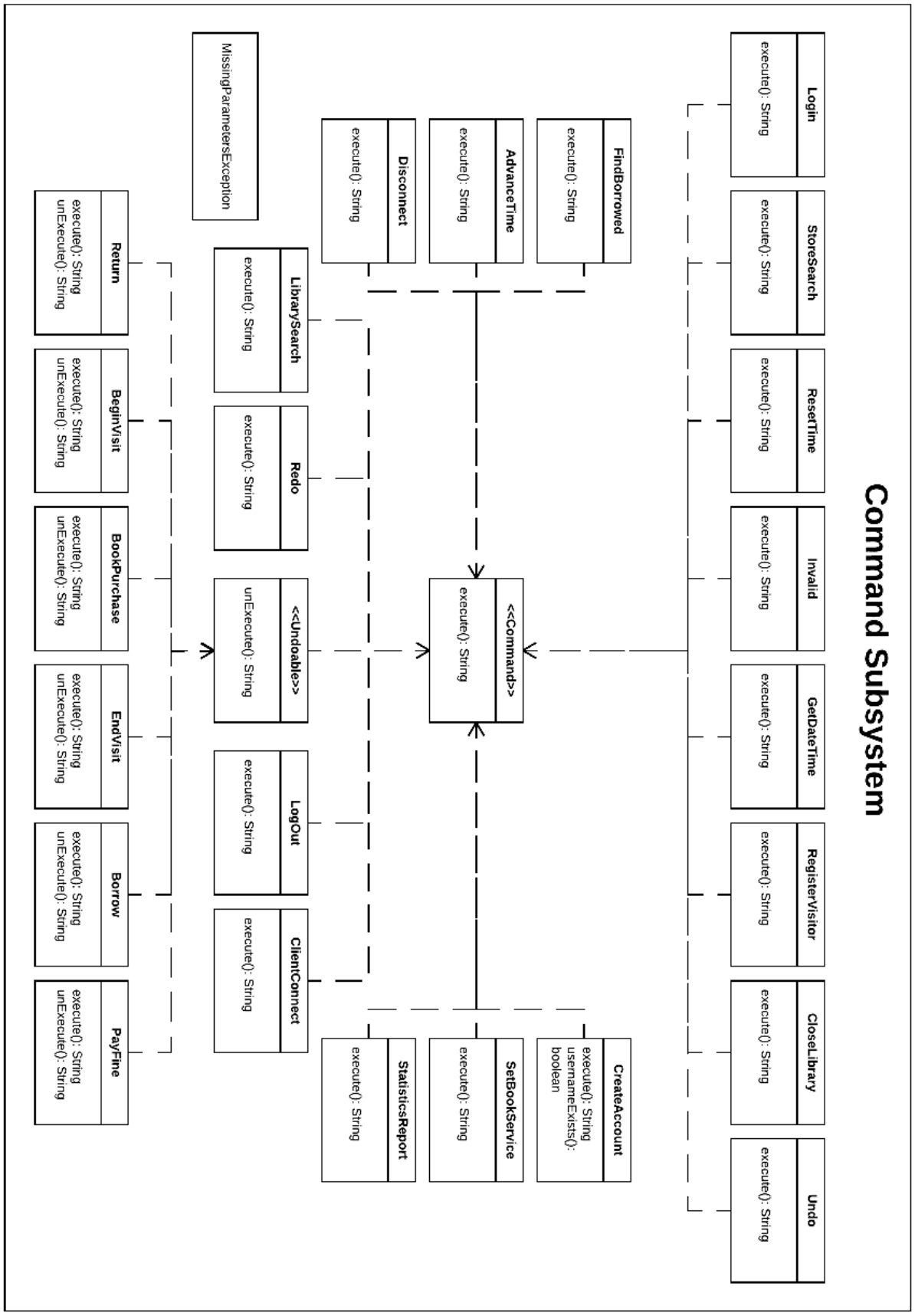


SUBSYSTEM DESIGN

COMMAND SUBSYSTEM

The Command Subsystem implements the command design pattern and is contained to the command package. The main interface, Command, requires that all classes that implement it have an execute() method that returns a String. The way we setup our classes the constructors all take in a request String, and possibly an additional parameter for simplifying the String parsing, and store the data in the command. Then when execute() is called the command interacts with the LBMS class to receive and/or modify its data and generates a response string that follows the request/response format that was given to us for this project. If the original request String is not properly formatted the constructors may throw a MissingParametersException and the CommandController class will handle it. For R2 we added another interface, Undoable, that extends the Command interface and adds a required method called, unExecute(). Commands that implement this interface are able to be “undone” and “redone” by a user of the system, the unExecute() method essentially reverses what was done in execute() and is used when a command is “undone”.

There are several advantages to using this design for commands. First, a new developer can see the direct correlation between each command and the accompanying request requirement from the project website which makes it really simply to add commands to the system. Additionally each command’s implementation is separated into its own class which follows the idea of separation of concerns. The only disadvantage of using this pattern is the requirement that all commands must implement the Command interface, which is not really an issue. Also it passed the responsibility of handling the request formats to the CommandController by throwing a MissingParametersException.


Command Subsystem
Login
execute(): String
StoreSearch
execute(): String
ResetTime
execute(): String
Invalid
execute(): String
GetDateTime
execute(): String
RegisterVisitor
execute(): String
CloseLibrary
execute(): String
Undo
execute(): String
FindBorrowed
execute(): String
AdvanceTime
execute(): String
Disconnect
execute(): String
<<Command>>
execute(): String
CreateAccount
execute(): String
usernameExists(): boolean
SetBookService
execute(): String
StatisticsReport
execute(): String
LibrarySearch
execute(): String
Redo
execute(): String
<<Undoable>>
unExecute(): String
LogOut
execute(): String
ClientConnect
execute(): String
MissingParametersException
Return
execute(): String
unExecute(): String
BeginVisit
execute(): String
unExecute(): String
BookPurchase
execute(): String
unExecute(): String
EndVisit
execute(): String
unExecute(): String
Borrow
execute(): String
unExecute(): String
PayFine
execute(): String
unExecute(): String

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() method. This method is common to all commands and does not share a common implementation. Each concrete command initializes by retrieving relevant information from an input string.
Undoable	Interface	This interface extends the Command interface. It adds functionality to a command by requiring the unExecute() method to reverse the execute() method when it is “undone”.
MissingParameters Exception	N/A	Custom exception used by the constructors of commands.
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.
ClientConnect	ConcreteCommand	This class defines the steps specific to connecting a client instance to the system.
CloseLibrary	ConcreteCommand	This class defines the steps specific to closing the library at closing time.
CreateAccount	ConcreteCommand	This class defines the steps specific to creating an account for a user to log in. Specifically, this process creates username and password credentials for an existing visitor.
Disconnect	ConcreteCommand	This class defines the steps specific to disconnecting a client instance from the system.

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.
LogIn	ConcreteCommand	This class defines the steps specific to logging in a visitor to a client instance.
LogOut	ConcreteCommand	This class defines the steps specific to logging out a visitor to a client instance.
PayFine	ConcreteCommand	This class defines the steps specific to having a visitor who owes overdue book fines pay those fines.
Redo	ConcreteCommand	This class defines the steps specific to redoing an undoable command.
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.
SetBookService	ConcreteCommand	This class defines the steps specific to setting the service that will be providing the responses for the book purchase search (either book.txt [local] or GoogleBooks [google]).
StatisticsReport	ConcreteCommand	This class defines the steps specific to generating a report of the current state of the library.

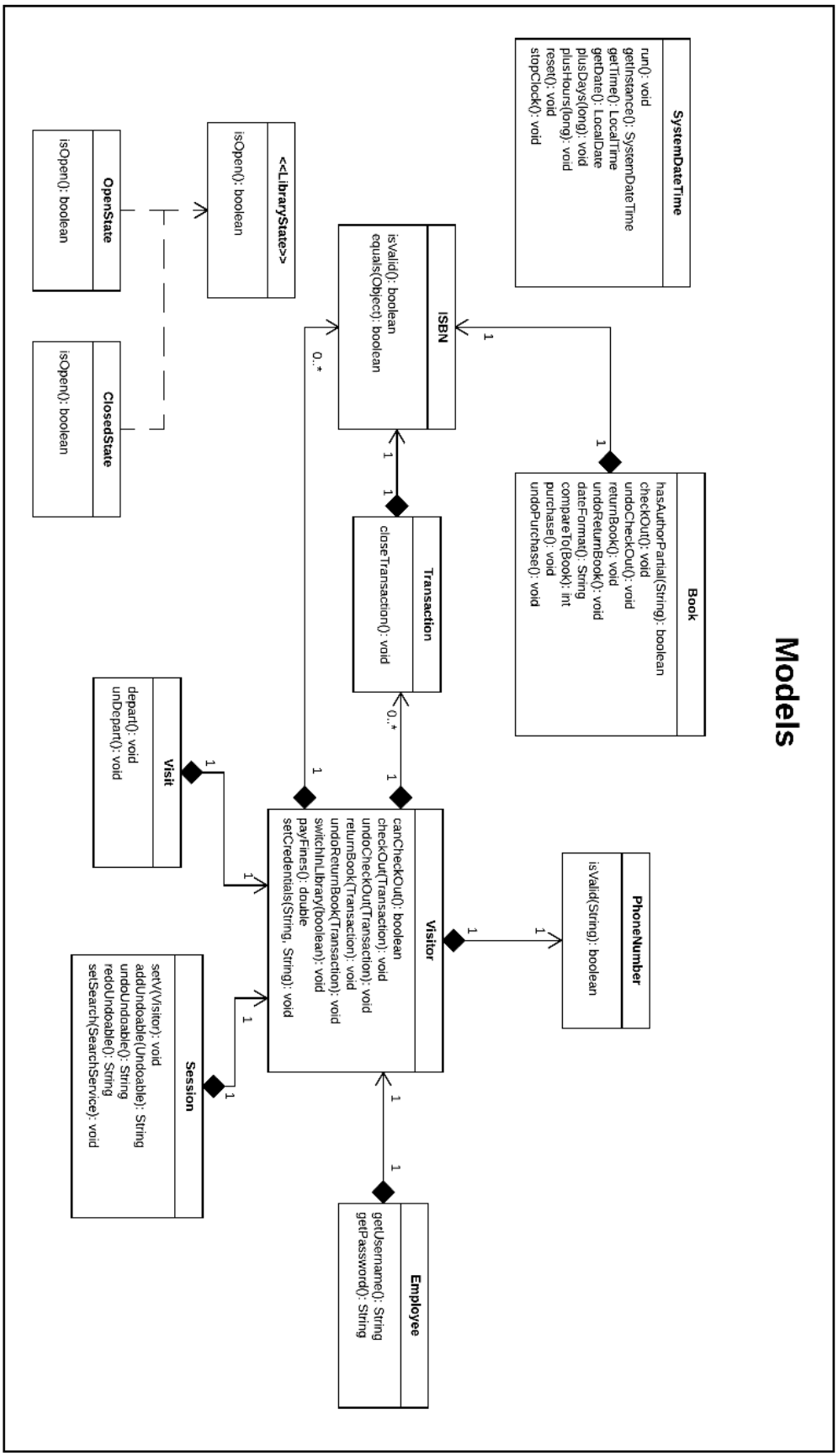
StoreSearch	ConcreteCommand	This class defines the steps specific to searching the bookstore for books available for purchase by the library which match input criteria.
Undo	ConcreteCommand	This class defines the steps specific to undoing an undoable command.
LBMS	Receiver	This class contains the state of the system and therefore receives the actions executed by the commands.

Deviations from the standard pattern:
- Addition of a second interface to implement commands that can be "undone".
- Custom exception that is thrown in constructors to signal an improper request format that it was given.

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.

MODELS SUBSYSTEM

The Models Subsystem is just a grouping of all the models we implemented and their interactions. These classes were separated into their own package to follow the MVC pattern and to make it clear what was part of the system data. 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, because there will only ever be one instance of SystemDateTime. The State pattern is also implemented in the models as the library itself is either in a closed or open state depending on the time of day. The state the library is in changes the actions that can be performed.

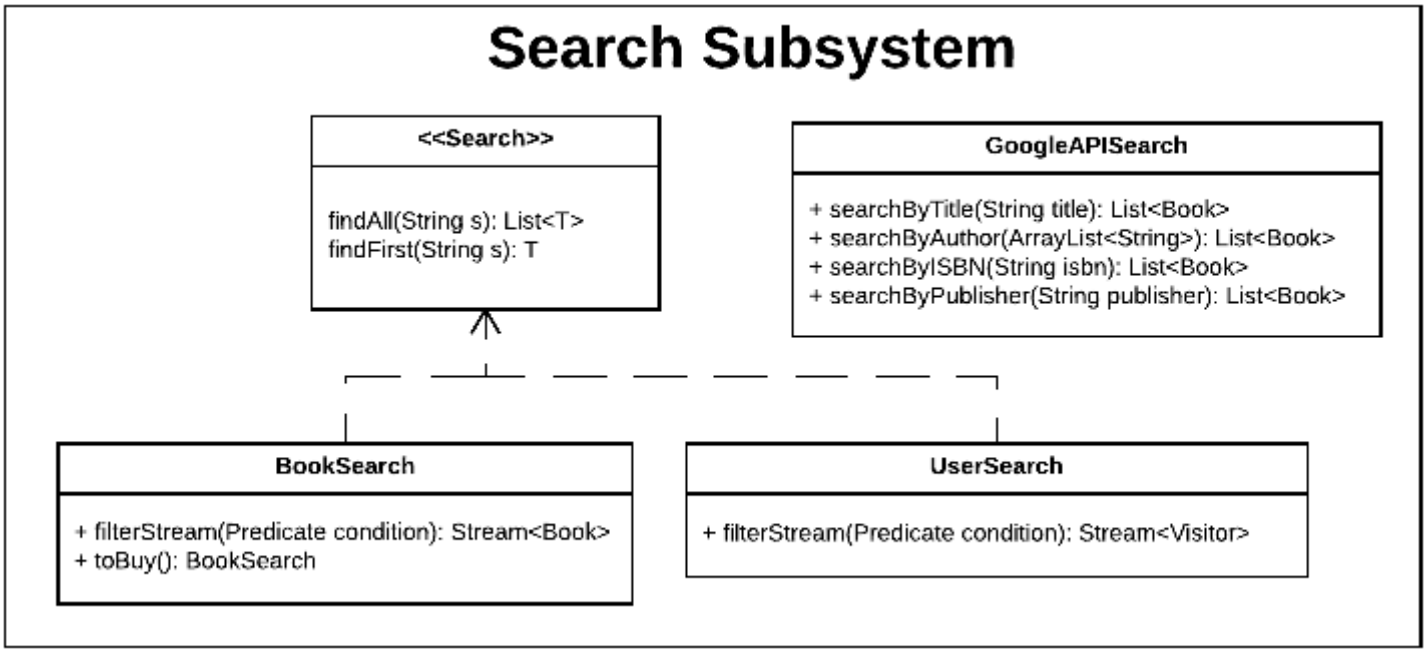

Models
SystemDateTime
run(): void
getInstance(): SystemDateTime
getTime(): LocalTime
getDate(): LocalDate
plusDays(long): void
plusHours(long): void
reset(): void
stopClock(): void
Book
hasAuthorPartial(String): boolean
checkOut(): void
undoCheckOut(): void
returnBook(): void
undoReturnBook(): void
dateFormat(): String
compareTo(Book): int
purchase(): void
undoPurchase(): void
ISBN
isValid(): boolean
equals(Object): boolean
Transaction
closeTransaction(): void
<<LibraryState>>
isOpen(): boolean
OpenState
isOpen(): boolean
ClosedState
isOpen(): boolean
Visit
depart(): void
unDepart(): void
Visitor
canCheckOut(): boolean
checkOut(Transaction): void
undoCheckOut(Transaction): void
returnBook(Transaction): void
undoReturnBook(Transaction): void
switchInLibrary(boolean): void
payFines(): double
setCredentials(String, String): void
PhoneNumber
isValid(String): boolean
Session
serv(Visitor): void
addUndoable(Undoable): String
undoUndoable(): String
redoUndoable(): String
setSearch(SearchService): void
Employee
getUsername(): String
getPassword(): String
1
0..*

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.		

Name: Library State		**GoF pattern:** State
Participants		
Class	**Role in GoF pattern**	**Participant's contribution in the context of the application**
LibraryState	Interface	This is the interface for all the state classes. It requires a state class to have an isOpen() method.
OpenState	ConcreteState	This class is used to represent the state of the library between the hours of 0800 and 1900 (System Time).
ClosedState	ConcreteState	This class is used to represent the state of the library between the hours of 1900 and 0800 the following day (System Time). When in this state, the library restricts access to commands.
Deviations from the standard pattern: None		
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, such as open and closed functionality differing.		

SEARCH SUBSYSTEM

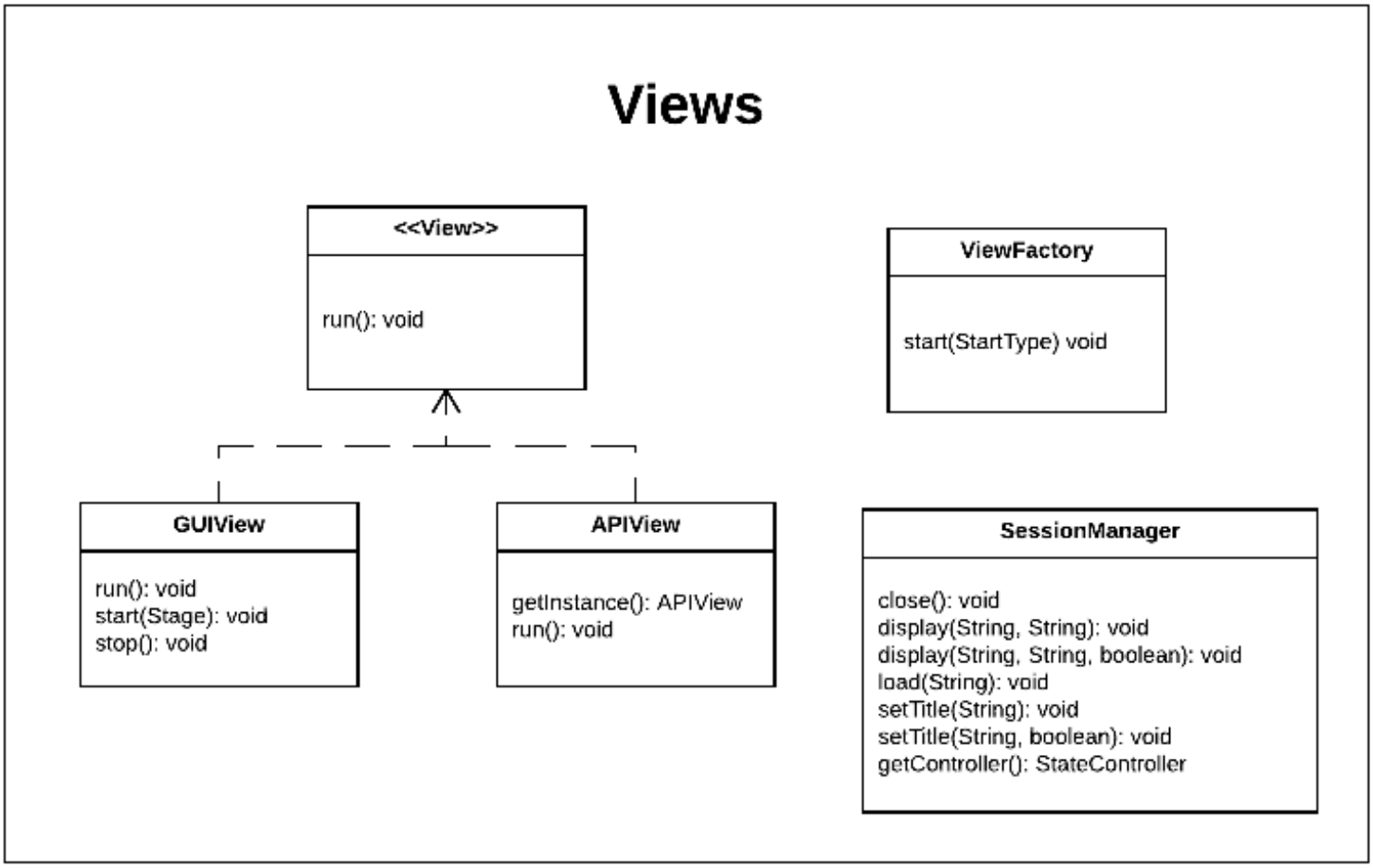
The Search Subsystem implements the Strategy design pattern, inheriting from a single interface, Search, and implementing different algorithms in each subclass. The strategy design patterns allow us to add additional searching algorithms quickly and efficiently, without having to modify the rest of the system. We used two main classes, BookSearch and UserSearch, to search the system. Both of those classes searched the local data of the program. For R2 we added another class, GoogleAPISearch, which uses Google Books to find additional books that can be purchased. However, it does not implement the Search interface and is not really part of the design pattern.



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.		

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 and Factory patterns. The Library Book Management System can only have one mode at a time that is decided on with program arguments when run. The two available modes of the system are API or GUI, API sends requests to the system and prints responses, GUI launches the graphical user interface. The mode of the system is controlled by the ViewFactory class, which uses the Factory pattern.



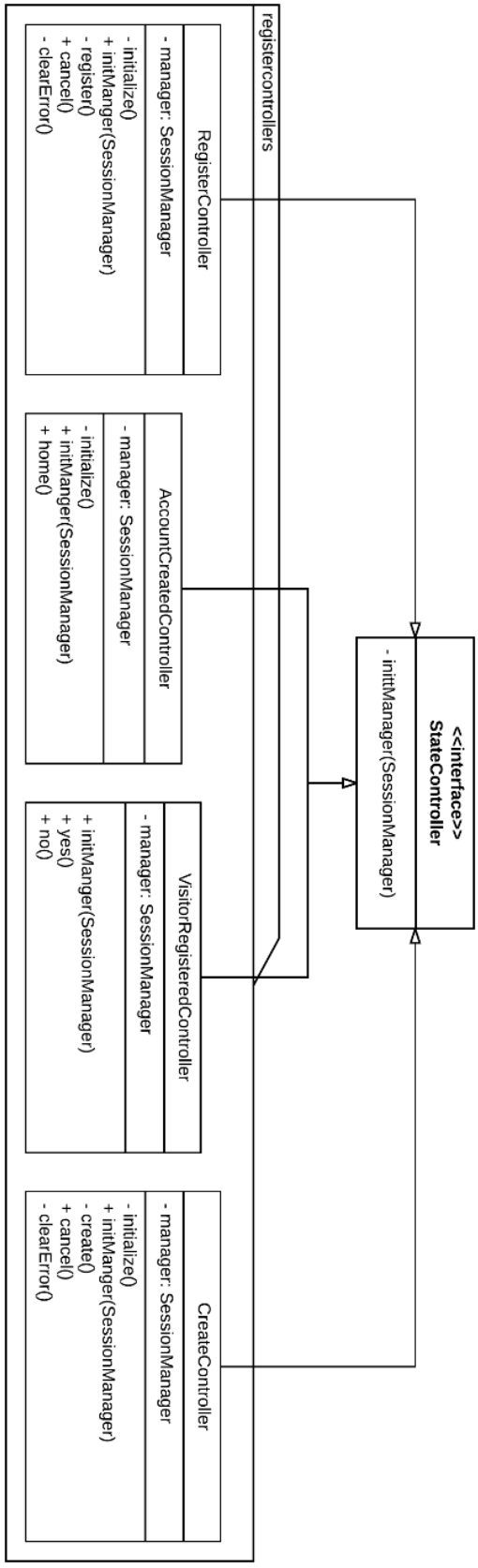
Name: View State		**GoF pattern:** State
Participants		
Class	**Role in GoF pattern**	**Participant's contribution in the context of the application**
View	Interface	This is the interface for all the state classes, it requires a state class to have the run() method.
APIView	ConcreteState	This class is used for starting the API mode of the Library Book Management System.
GUIView	ConcreteState	This class is used for starting the graphical user interface of the Library Book Management System.
Deviations from the standard pattern: None		
Requirements being covered: The system can only run one mode at a time, either API or GUI.		

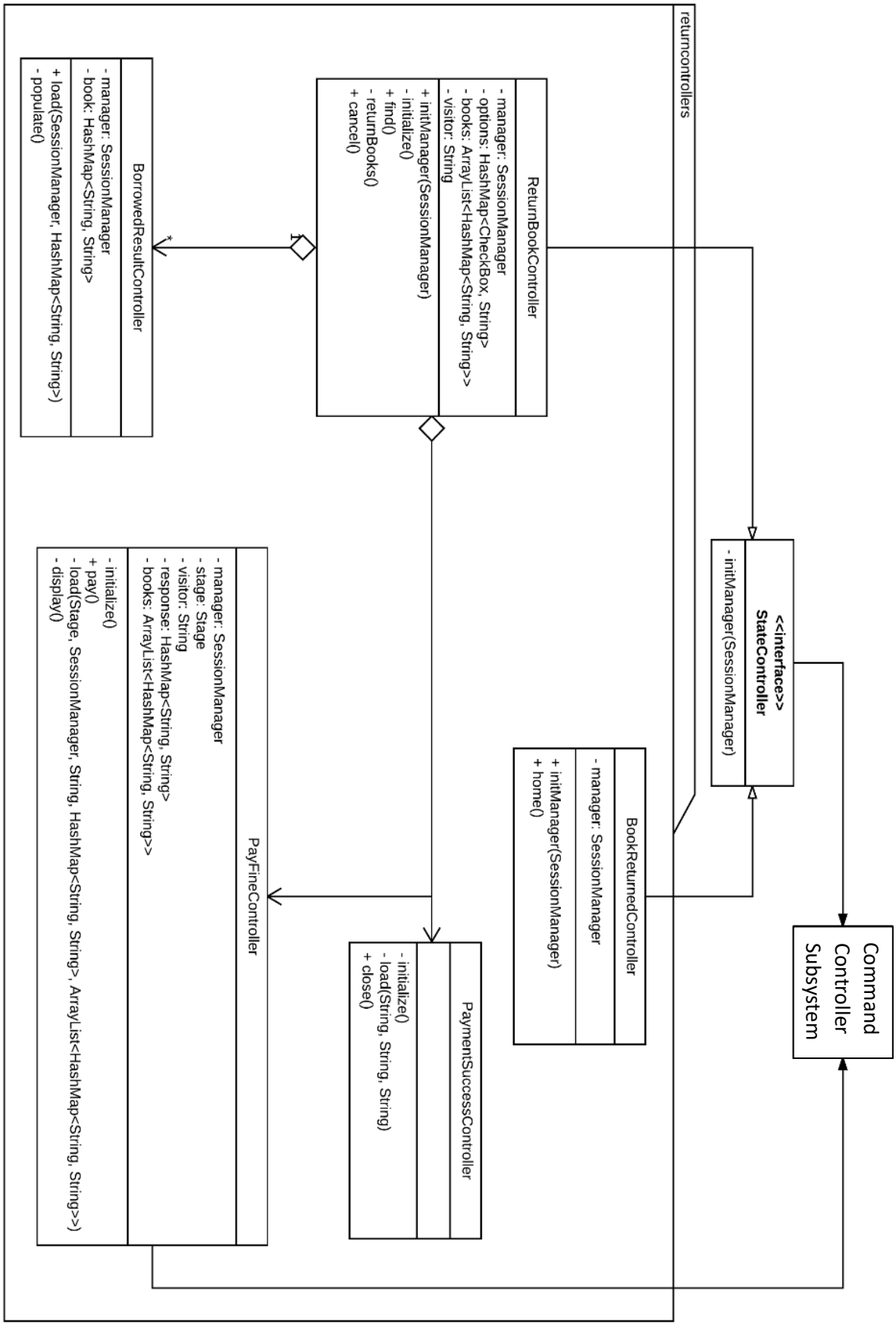
Name: ViewFactory		**GoF pattern:** Factory Method
Participants		
Class	**Role in GoF pattern**	**Participant's contribution in the context of the application**
ViewFactory	ConcreteCreator	This class selects the appropriate View and runs it.
View	Product	This is the interface the ConcreteProducts must implement.
APIView	ConcreteProduct	This class is used for starting the API mode of the Library Book Management System.
GUIView	ConcreteProduct	This class is used for starting the graphical user interface of the Library Book Management System.
Deviations from the standard pattern: No interface used for the creator.		
Requirements being covered: The system runs one of two modes, API or GUI, and the specific subclass that it needs to create cannot be determined until runtime as it is determined by the program arguments.		

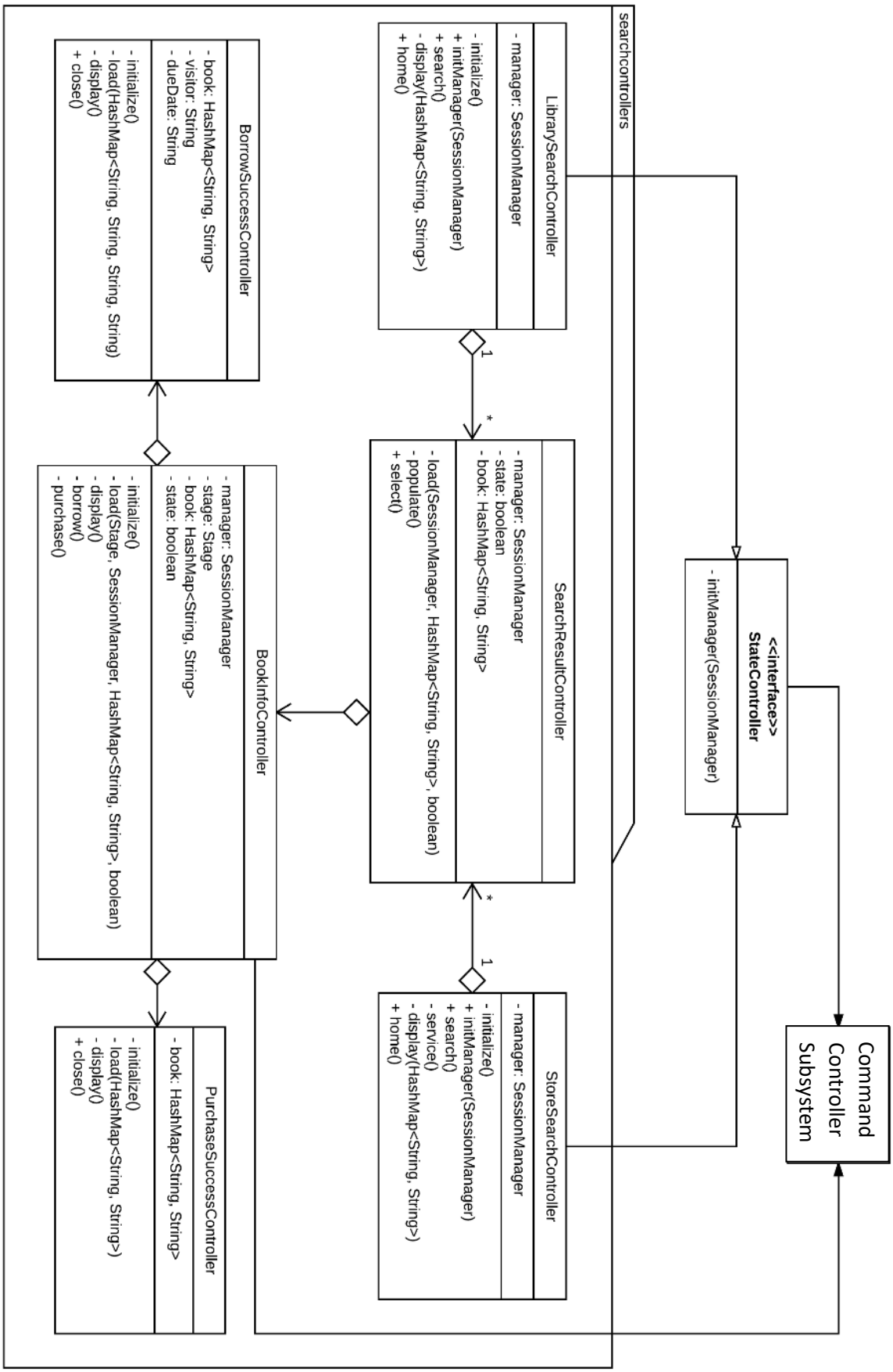
CONTROLLERS SUBSYSTEM

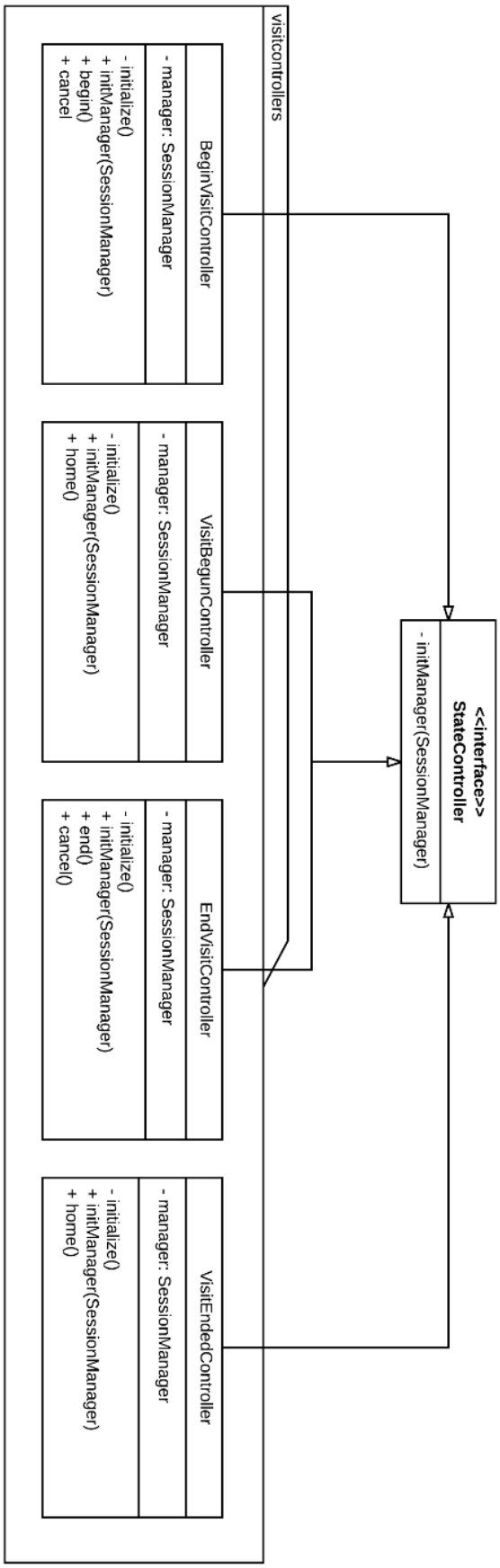
The Controllers Subsystem is contained in the controllers package and contains two sub packages, commandproxy and guicontrollers. The commandproxy package is used to control the Command Subsystem and incorporates the Proxy pattern. This allows us to determine if a command can be executed based on user permissions, allowing visitors and employees to have different abilities within the system. It also contains the ParseResponseUtility class which is not part of the design pattern, but is used to take the information outputted by a command and format it for use with the GUI. The guicontrollers package contains several sub-packages and classes that interact with the GUI, following the MVC pattern. These controllers are the part of the system that connect the JavaFX code to the Command Subsystem.

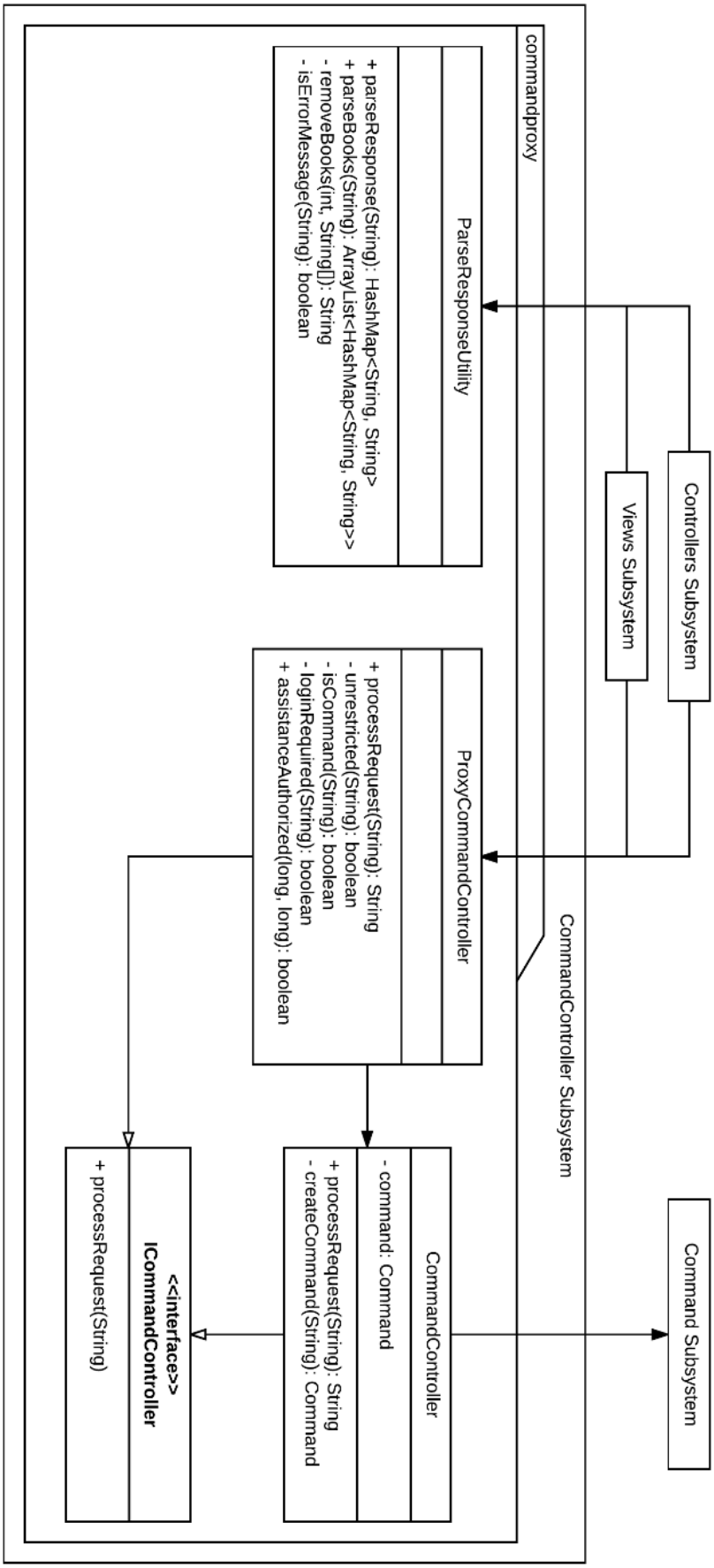
guicontrollers
RegisterControllers
ReturnControllers
SearchControllers
VisitControllers
SystemController
- manager: SessionManager
- initialize()
+ initManager(SessionManager)
+ home()
+ advance()
+ report()
+ command()
LoginController
- manager: SessionManager
- initializer()
+ initManager(SessionManager)
- execute()
MainVisitorController
- manager: SessionManager
- initialize()
+ initManager(SessionManager)
+ search()
- beginVisit()
- endVisit()
+ logout()
MainEmployeeController
- manager: SessionManager
- initManager(SessionManager)
- initialize()
+ search()
+ searchStore()
+ beginVisit()
+ endVisit()
+ returnBook()
+ register()
+ create()
+ settings()
+ logout
<<interface>>
StateController
- initManager(SessionManager)
ClientController
- stop: Boolean
- tabs: TabPane
- initialize()
- addTab()
+ stop()
- createMenuBar()
Command
Controller
Subsystem
Views Subsystem
Views
SessionManager
- clientId: Long
- tab: Tab
- controller: StateController
- visitorID: Long
- user: String
+ SessionManager(Tab)
+ close(boolean)
+ display(String, String)
+ display(String, String, boolean)
- load(String)


registercontrollers
RegisterController
- manager: SessionManager
- initialize()
+ initManger(SessionManager)
- register()
+ cancel()
- clearError()
AccountCreatedController
- manager: SessionManager
- initialize()
+ initManger(SessionManager)
+ home()
VisitorRegisteredController
- manager: SessionManager
+ initManger(SessionManager)
+ yes()
+ no()
CreateController
- manager: SessionManager
- initialize()
+ initManger(SessionManager)
- create()
+ cancel()
- clearError()
<<interface>>
StateController
- inittManager(SessionManager)

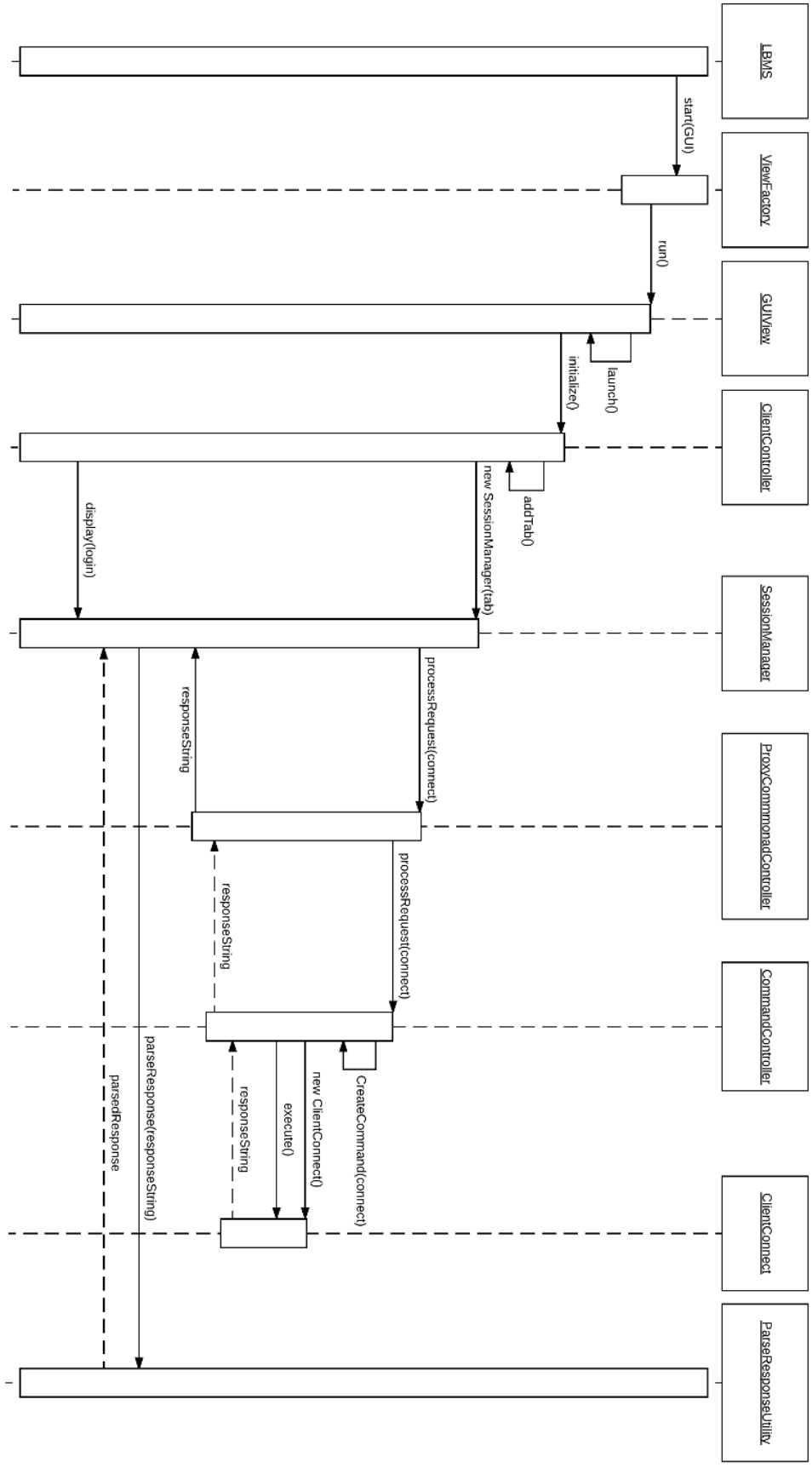

returncontrollers
<<interface>>
StateController
- initManager(SessionManager)
Command
Controller
Subsystem
ReturnBookController
- manager: SessionManager
- options: HashMap<CheckBox, String>
- books: ArrayList<HashMap<String, String>>
- visitor: String
+ initManager(SessionManager)
- initialize()
+ find()
- returnBooks()
+ cancel()
BorrowedResultController
- manager: SessionManager
- book: HashMap<String, String>
+ load(SessionManager, HashMap<String, String>)
- populate()
BookReturnedController
- manager: SessionManager
+ initManager(SessionManager)
+ home()
PayFineController
- manager: SessionManager
- stage: Stage
- visitor: String
- response: HashMap<String, String>
- books: ArrayList<HashMap<String, String>>
- initialize()
+ pay()
- load(Stage, SessionManager, String, HashMap<String, String>, ArrayList<HashMap<String, String>>)
- display()
PaymentSuccessController
- initialize()
- load(String, String, String)
+ close()

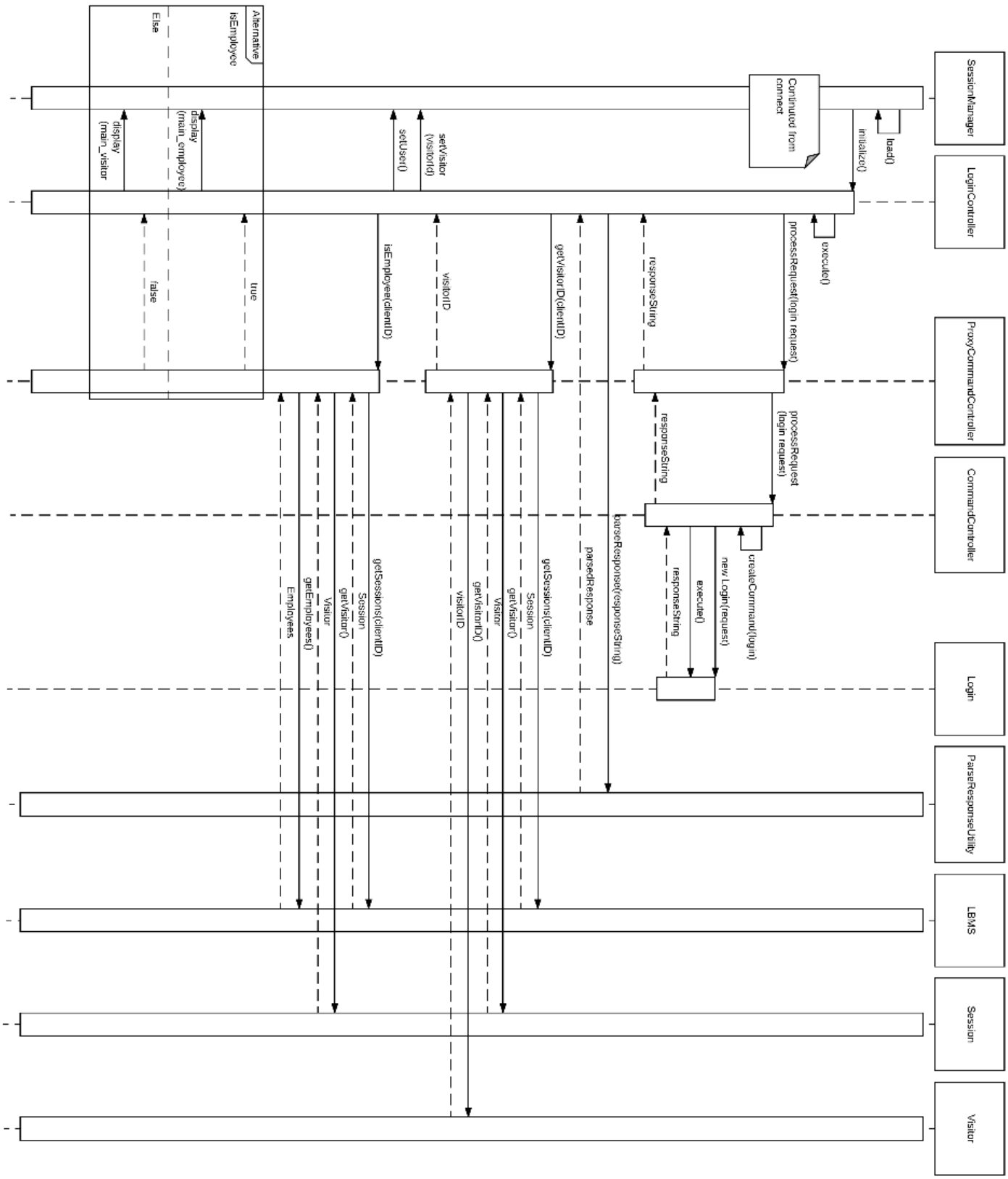

searchcontrollers
LibrarySearchController
- manager: SessionManager
- initialize()
+ initManager(SessionManager)
+ search()
- display(HashMap<String, String>)
+ home()
<<interface>>
StateController
- initManager(SessionManager)
Command
Controller
Subsystem
SearchResultController
- manager: SessionManager
- state: boolean
- book: HashMap<String, String>
- load(SessionManager, HashMap<String, String>, boolean)
- populate()
+ select()
StoreSearchController
- manager: SessionManager
- initialize()
+ initManager(SessionManager)
+ search()
- service()
- display(HashMap<String, String>)
+ home()
BorrowSuccessController
- book: HashMap<String, String>
- visitor: String
- dueDate: String
- initialize()
- load(HashMap<String, String, String, String, String)
- display()
+ close()
BookInfoController
- manager: SessionManager
- stage: Stage
- book: HashMap<String, String>
- state: boolean
- initialize()
- load(Stage, SessionManager, HashMap<String, String>, boolean)
- display()
- borrow()
- purchase()
PurchaseSuccessController
- book: HashMap<String, String>
- initialize()
- load(HashMap<String, String>)
- display()
+ close()
1
*
*
1

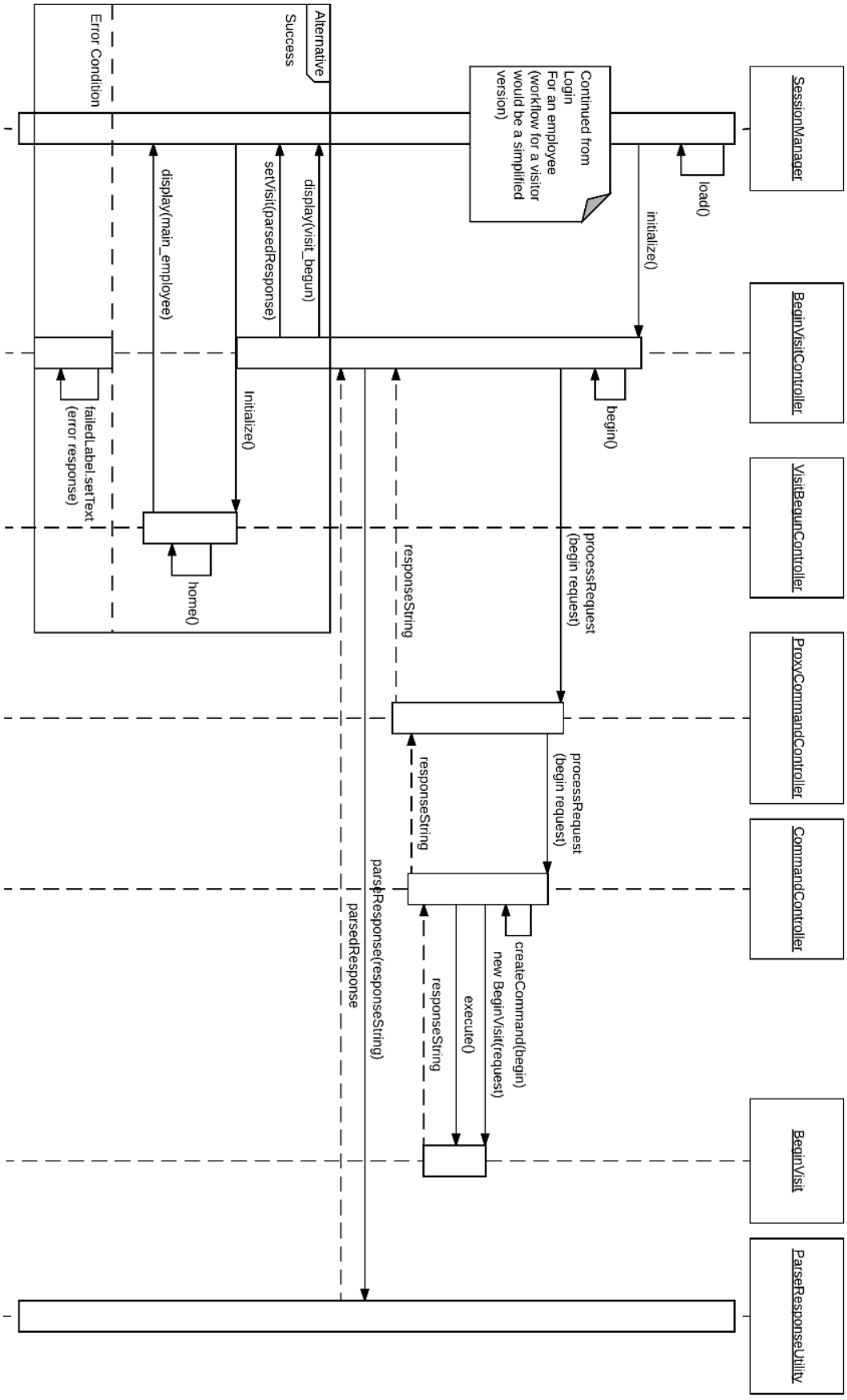

visitcontrollers
BeginVisitController
- manager: SessionManager
- initialize()
+ initManager(SessionManager)
+ begin()
+ cancel
VisitBegunController
- manager: SessionManager
- initialize()
+ initManager(SessionManager)
+ home()
EndVisitController
- manager: SessionManager
- initialize()
+ initManager(SessionManager)
+ end()
+ cancel()
VisitEndedController
- manager: SessionManager
- initialize()
+ initManager(SessionManager)
+ home()
<<interface>>
StateController
- initManager(SessionManager)


Controllers Subsystem
Views Subsystem
CommandController Subsystem
Command Subsystem
commandproxy
ParseResponseUtility
+ parseResponse(String): HashMap<String, String>
+ parseBooks(String): ArrayList<HashMap<String, String>>
- removeBooks(int, String[]): String
- isErrorMessage(String): boolean
ProxyCommandController
+ processRequest(String): String
- unrestricted(String): boolean
- isCommand(String): boolean
- loginRequired(String): boolean
+ assistanceAuthorized(long, long): boolean
CommandController
- command: Command
+ processRequest(String): String
- createCommand(String): Command
<<interface>>
ICommandController
+ processRequest(String)

Name: Proxy Command		**GoF pattern:** Proxy
Participants		
Class	**Role in GoF pattern**	**Participant's contribution in the context of the application**
ICommandController	Subject	Interface for the command controller, requires the processRequest method that takes a string as a parameter and returns a string.
ProxyCommandController	Proxy	This class implements the ICommandController interface. It checks for the user's status (Visitor or Employee) and calls processRequest in the CommandController if it allowable.
CommandController	RealSubject	This class also implements the ICommandController interface. It creates the appropriate command and executes it, directly interacting with the Command Subsystem.
Deviations from the standard pattern: None		
Requirements being covered: Uses the Proxy design pattern to control how commands are processed by the system based on the user and state of the library.		


LBMS
ViewFactory
GUIView
ClientController
SessionManager
ProxyCommmonadController
CommandController
ClientConnect
ParseResponseUtility
start(GUI)
run()
launch()
initialize()
addTab()
new SessionManager(tab)
display(login)
processRequest(connect)
responseString
processRequest(connect)
responseString
CreateCommand(connect)
new ClientConnect()
execute()
responseString
parseResponse(responseString)
parsedResponse


SessionManager
LoginController
ProxyCommandController
CommandController
Login
ParseResponseUtility
LBMS
Session
Visitor
load()
initialize()
Continued from connect
execute()
processRequest(login request)
responseString
getVisitorID(clientID)
visitorID
isEmployee(clientID)
true
false
Alternative
isEmployee
Else
display (main_visitor)
display (main_employee)
setUser()
setVisitor (visitorId)
processRequest (login request)
responseString
createCommand(login)
new Login(request)
execute()
responseString
parseResponse(responseString)
parsedResponse
getSessions(clientID)
Session
getVisitor()
Visitor
getVisitorID()
visitorID
getSessions(clientID)
Session
getVisitor()
Visitor
getEmployees()
Employees


SessionManager
BeginVisitController
VisitBegunController
ProxyCommandController
CommandController
BeginVisit
ParseResponseUtility
Continued from Login For an employee (workflow for a visitor would be a simplified version)
load()
initialize()
begin()
processRequest (begin request)
responseString
processRequest (begin request)
responseString
createCommand(begin)
new BeginVisit(request)
execute()
responseString
parseResponse(responseString)
parsedResponse
Alternative
Success
Error Condition
display(visit_begun)
setVisit(parsedResponse)
display(main_employee)
Initialize()
failedLabel.setText (error response)
home()

APPENDIX

Main

Class: LBMS	
Responsibilities: The LBMS class is responsible for storing all data related to the library such as the books, visitors, transactions, etc. Depending on the program argument given, the program will use its API form or GUI form. This class is capable of saving updated data through serialization after shutdown and deserialization during startup. When the LBMS is started up with a clean slate, an initial admin account is created. It is also responsible for 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: ProxyCommandController, ViewFactory, Book, Employee, ISBN, PhoneNumber, Session, SystemDateTime, Transaction, Visit, Visitor	**Used by:** BeginVisit, BookPurchase, Borrow, ClientConnect, CreateAccount, Disconnect, EndVisit, FindBorrowed, LibrarySearch, LogIn. LogOut, Undo, Redo, RegisterVisitor, Return, SetBookService, StatisticsReport, StoreSearch, BookSearch, UserSearch, ViewFactory
Author: Edward Wong	

Commands

Class: Command	
Responsibilities: All commands used to change the data stored in the library inherit from this interface. Each class inheriting from this interface implements a execute() method. The execute() method manipulates the data according to the input.	
Collaborators	
Uses: None	**Used by:** Undoable, AdvanceTime, BeginVisit, BookPurchase, Borrow, ClientConnect, CloseLibrary, CreateAccount, Disconnect, EndVisit, FindBorrowed, GetDateTime, Invalid, LibrarySearch, PayFine, RegisterVisitor, ResetTime, Return, StatisticsReport, StoreSearch
Author: Edward Wong	

Class: Undoable	
Responsibilities: The Undoable interface extends the Command interface. Some commands must be capable of being undone and redone. This interface requires these command classes to implement an unExecute() method which basically does the opposite of the command's execute() method.	
Collaborators	
Uses: Command	**Used by:** BeginVisit, BookPurchase, Borrow, EndVisit, PayFine, Return
Author: Edward Wong	

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: Edward Wong	

Class: BeginVisit	

Responsibilities: This class is responsible for adding a visit to the LBMS. The user may give a visitorID if they choose to. If the user is an employee, they can enter anyone's visitorID as long as it is valid. If the user is a visitor, they can only enter their own 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 no visitorID is mentioned, the visitorID chosen is the visitorID of the user using the client. If the visit was successfully started, a success message will be displayed with the date and time the visit started. This command can be undone and redone.
Collaborators

Uses: Command, Undoable, LBMS, ProxyCommandController, SystemDateTime, Visit, Visitor, UserSearch	**Used by:** CommandController
Author: Edward Wong	

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 store search, the user can choose which books to buy along with the quantity. 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 it's 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. This command is undoable and redoable.	
Collaborators	
Uses: Command, Undoable, LBMS, Book	**Used by:** CommandController
Author: Edward Wong	

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 book was borrowed. Otherwise, an error message is returned stating why the book(s) could not be borrowed. This command is redoable and undoable.	
Collaborators	
Uses: Command, Undoable, LBMS, ProxyCommandController, Book, SystemDateTime, Transaction, Visitor, UserSearch, MissingParametersException	**Used by:** CommandController
Author: Edward Wong	

Class: ClientConnect	
Responsibilities: This class is responsible for starting a new client in order for a user to access the LBMS. It starts a new session and adds that session to the LBMS.	
Collaborators	
Uses: Command, LBMS, Session	**Used by:** CommandController
Author: Edward Wong	

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: Edward Wong	

Class: CreateAccount	
Responsibilities: The CreateAccount class makes a new account for a registered visitor. An account consists of a username, password, role, and the visitorID of the visitor who will own the account. The role can only be entered as “visitor” or “employee” This will decide the permissions of the user. If the role is entered as neither of these words, an error message occurs. An error message also occurs if the given visitorID does not exist.	
Collaborators	
Uses: Command, LBMS, Employee, Visitor, MissingParametersException	**Used by:** CommandController
Author: Edward Wong	

Class: Disconnect	
Responsibilities: Using the clientID, this command disconnects a client. LBMS then removes this session from its hashmap of stored sessions. If there is no session attached to the given clientID, an error message appears.	
Collaborators	

Uses: Command, LBMS	**Used by:** CommandController
Author: Edward Wong	

Class: EndVisit	
Responsibilities: EndVisit removes a visitor from the library. It also adds a visit to the total visits that the LBMS records. If the visitorID is not mentioned, the visitorID will be assumed to be the visitorID of the user operating the client. It won't work if the visitorID does not exist or the visitor 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. This command is undoable and redoable.	
Collaborators	
Uses: Command, Undoable, LBMS, ProxyCommandController, SystemDateTime, Visit, Visitor, UserSearch	**Used by:** CommandController
Author: Edward Wong	

Class: FindBorrowed	
Responsibilities: Given a valid visitorID, this command presents the number of books a visitor has borrowed and which specific books were borrowed. If the visitorID is not given, the visitorID of the user operating the client will be used. 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: Edward Wong	

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

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: Edward Wong	

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, ISBN, BookSearch, MissingParametersException	**Used by:** CommandController
Author: Edward Wong	

Class: LogIn	
Responsibilities: This class allows a visitor or employee to log in into their LBMS account. The session operating will be obtained from the LBMS and the visitor logging in will be attached to it. If they input an invalid username and/or password, an error message will be presented to the user.	
Collaborators	
Uses: LBMS, Visitor, MissingParametersException	**Used by:** CommandController
Author: Edward Wong	

Class: LogOut	
Responsibilities: This class logs a visitor or employee out of their account with a given clientID. The session operating is obtained from the LBMS and the visitor that is attached to that session detached from it. If the clientID is not valid, an error message will appear.	
Collaborators	
Uses: LBMS, Session	**Used by:** CommandController

Author: Edward Wong

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 parameters and which parameters are missing.	
Collaborators	
Uses: None	**Used by:** Borrow, EndVisit, LibrarySearch, LogIn, RegisterVisitor, StatisticsReport, StoreSearch
Author: Edward Wong	

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 a visitorID is not in the request, it will assumed to be the visitorID of the user operating the current client. 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. This command is undoable and redoable.	
Collaborators	
Uses: Command, UserSearch	**Used by:** CommandController
Author: Edward Wong	

Class: Undo	
Responsibilities: This class allows for the undoing of some commands. These commands include beginning a visit, purchasing a book, borrowing a book, ending a visit, paying a fine, and returning a book. The only times where this command will not work is when the undo stack is empty (none of the previous commands have been executed) or when you attempt to undo after a search clears the undo and redo stacks.	
Collaborators	
Uses: LBMS	**Used by:** CommandController
Author: Edward Wong	

Class: Redo	

Responsibilities: The Redo class allows certain commands that were undone to be redone. The only times this command will not work is when there is no commands in the redo stack (no commands have been undone) or when a new search has cleared out both the redo and undo stack. When this is the case, an error message is returned.	
Collaborators	
Uses: LBMS	**Used by:** CommandController
Author: Edward Wong	

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, PhoneNumber, SystemDateTime, Visitor, UserSearch, MissingParametersException	**Used by:** CommandController
Author: Edward Wong	

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: Edward Wong	

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 visitorID is not given, the visitorID of the user operating the client will be used. If the book is returned overdue, a fine will be added to the visitor’s total balance and the overdue books are not returned until the fines have been paid. 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, Undoable, LBMS, Book, SystemDateTime, Transaction, Visitor, UserSearch	**Used by:** CommandController
Author: Edward Wong	

Class: SetBookService	
Responsibilities: This class is responsible for enabling the user to switch between the local book service and the Google Books service. The local book service contains the books parsed from books.txt while the Google Books service contains books obtained with the Google API. This command obtains the current session and changes the service to search from. If the user enters a service other than “local” and “google,” an error message appears.	
Collaborators	
Uses: LBMS, MissingParameterException	**Used by:** CommandController
Author: Edward Wong	

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 statistics recorded since the beginning of the simulation.	
Collaborators	
Uses: Command, LBMS, Book, SystemDateTime, Visit, Visitor, MissingParametersException	**Used by:** CommandController
Author: Edward Wong	

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 by being given a temporary ID.	
Collaborators	

Uses: Command, LBMS, Book, ISBN, BookSearch, GoogleAPISearch, MissingParametersException	**Used by:** CommandController
Author: Edward Wong	

Controllers

Class: ICommandController	
Responsibilities: This interface defines the functionality required by the CommandController and its proxy. The parseResponse() method is required to be defined by classes implementing this interface. This method is used to take in a request string and subsequently output the proper response, usually involving multiple checks on the state, type, and validity of the request.	
Collaborators	
Uses: None	**Used by:** CommandController, ProxyCommandController
Author: Charles Barber	

Class: ProxyCommandController	
Responsibilities: This class is used to implement a protection proxy. It restricts access to the CommandController based on the authorization status of the user/command type pairing. In this way, a check is performed and a response is generated without a command object ever being created. This proxy is always used to access the underlying CommandController and calls the CommandController if appropriate.	
Collaborators	
Uses: LBMS, Invalid, Visitor, Employee, Session, Visit, SystemDateTime	**Used by:** LBMS, BeginVisit, Borrow, EndVisit, APIView, SessionManager
Author: Charles Barber	

Class: ParseResponseUtility	
Responsibilities: Utility class designed to provide a centralized location for parsing all types of response strings into a useable format. Mainly used by the views to parse response data to be displayed in the GUI.	
Collaborators	
Uses: Book, BookSearch	**Used by:** SessionManager
Author: Charles Barber	

Class: AccountCreatedController	

Responsibilities: This class handles the GUI view for the page displayed after the user has created an account. Once an account has been created, a confirmation page is displayed with detail about the action that just occurred. This class is responsible for displaying that page. Also on this page is the option to return to the main screen by pressing a finish button. This class handles the event execution of the button press and directs the system display back to the main screen.
Collaborators

Uses: StateController, SessionManager	**Used by:** CreateController
Author: Charles Barber	

Class: CreateController	
Responsibilities: This class handles the GUI view for the page used to create an account. The page to create an account displays relevant text input fields as well as create and cancel buttons. This class handles the submission of the input fields as well as the event execution of each button press. As all the fields are required, this class will prevent execution on incomplete data and will output an error message to the user when applicable. It should be noted that this class is directly related to the CreateAccount command class and therefore generates the appropriate request and interprets the response for that command.	
Collaborators	
Uses: ParseResponseUtility, ProxyCommandController, StateController, SessionManager	**Used by:** VisitorRegisteredController
Author: Charles Barber	

Class: RegisterController	
Responsibilities: This class handles the GUI view for the page used to register a visitor. The page to create an account displays relevant text input fields as well as register and cancel buttons. This class handles the submission of the input fields as well as the event execution of each button press. As all the fields are required, this class will prevent execution on incomplete data and will output an error message to the user when applicable. It should be noted that this class is directly related to the RegisterVisitor command class and therefore generates the appropriate request and interprets the response for that command.	
Collaborators	
Uses: ParseResponseUtility, ProxyCommandController, StateController, SessionManager	**Used by:** None
Author: Charles Barber	

Class: VisitorRegisteredController	
Responsibilities: This class handles the GUI view for the page displayed after the user has registered a visitor. Once visitor has been registered, a confirmation page is displayed with detail about the action that just occurred. This class is responsible for displaying that page. Also on this page is the option to return to the main screen by pressing a finish button. This class handles the event execution of the button press and directs the system display back to the main screen.	
Collaborators	
Uses: StateController, SessionManager	**Used by:** RegisterController
Author: Charles Barber	

Class: BookReturnedController	
Responsibilities: This class handles the GUI view for the page displayed after the user has successfully returned a book. Once one or more books are returned, a confirmation page is displayed with detail about the action that just occurred. This class is responsible for displaying that page. Also on this page is the option to return to the main screen by pressing a finish button. This class handles the event execution of the button press and directs the system display back to the main screen.	
Collaborators	
Uses: StateController, SessionManager	**Used by:** None
Author: Charles Barber	

Class: BorrowedResultController	
Responsibilities: This class is used to link a book from the borrowed book query to a checkbox which can then be displayed in the GUI. When a user looks for all the books they have currently borrowed from the library, a checklist is displayed in the GUI. This class handles the display of a book as a checkbox.	
Collaborators	
Uses: SessionManager	**Used by:** PayFineController, ReturnBookController

Author: Charles Barber

Class: PayFineController	
Responsibilities: This class handles the GUI view for the page used to pay overdue fines. This page can only be accessed after an attempt to return a book has revealed a late fee for one or more of the books currently borrowed by the visitor. It should be noted that this class is directly related to the PayFine command class and therefore generates the appropriate request and interprets the response for that command.	
Collaborators	
Uses: ParseResponseUtility, ProxyCommandController, SessionManager	**Used by:** ReturnBookController
Author: Charles Barber	

Class: PaymentSuccessController	
Responsibilities: This class handles the GUI view for the page displayed after the user has successfully payed a fine. Once a fine is paid, a confirmation page is displayed with detail about the action that just occurred. This class is responsible for displaying that page. Also on this page is the option to return to the main screen by pressing a finish button. This class handles the event execution of the button press.	
Collaborators	
Uses: None	**Used by:** PayFineController
Author: Charles Barber	

Class: ReturnBookController	
Responsibilities: This class handles the GUI view for the page used to return borrowed library books. The page displayed to return books contains input in the form of checkboxes and button presses. This class handles the event execution for these items as well as checks the submission status of any text fields. It should be noted that this class is directly related to the Return command class and therefore generates the appropriate request and interprets the response for that command.	
Collaborators	
Uses: ProxyCommandController, ParseResponseUtility, SessionManager, StateController	**Used by:** None

Author: Charles Barber

Class: BookInfoController	
Responsibilities: This class handles the GUI view for the page used to display the results of a book search. From the results of a book search, a user can perform actions on those results such as borrowing and purchasing (the latter only if the user is an employee). It should be noted that this class is directly related to the Borrow and PurchaseBook command classes and therefore generates the appropriate requests and interprets the responses for those commands.	
Collaborators	
Uses: ProxyCommandController, ParseResponseUtility, SessionManager	**Used by:** SearchResultController
Author: Charles Barber	

Class: BorrowSuccessController	
Responsibilities: This class handles the GUI view for the page displayed after the user has successfully borrowed a book. Once one or more books are borrowed, a confirmation page is displayed with detail about the action that just occurred. This class is responsible for displaying that page, specifically, as a pop-up window.. Also on this page is the option to exit the popup window displays. This class handles the event execution of the close button press.	
Collaborators	
Uses: None	**Used by:** BookInfoController
Author: Charles Barber	

Class: LibrarySearchController	
Responsibilities: This class handles the GUI view for the page used to search the library for books. Both input fields and buttons exist on this page. This controller class handles the submission of the input fields and the action execution of any existing buttons. It should be noted that this class is directly related to the LibrarySearch command class and therefore generates the appropriate request and interprets the response for that command.	
Collaborators	
Uses: ProxyCommandController, ParseResponseUtility, SessionManager, StateController	**Used by:** MainEmployeeController, MainVisitorController

Author: Charles Barber

Class: PurchaseSuccessController	
Responsibilities: This class handles the GUI view for the page displayed after the user has successfully purchased a book. Once one or more books are purchased, a confirmation page is displayed with detail about the action that just occurred. This class is responsible for displaying that page, specifically, as a pop-up window. Also on this page is the option to exit the popup window displays. This class handles the event execution of the close button press.	
Collaborators	
Uses: None	**Used by:** BookInfoController
Author: Charles Barber	

Class: SearchResultController	
Responsibilities: This class is used to display a specific search result in the GUI. This search result is specifically resulting from a book search of either the library or the store.	
Collaborators	
Uses: SessionManager	**Used by:** LibrarySearchController, StoreSearchController
Author: Charles Barber	

Class: StoreSearchController	
Responsibilities: This class handles the GUI view for the page used to search the store for books. Both input fields and buttons exist on this page. This controller class handles the submission of the input fields and the action execution of any existing buttons. It should be noted that this class is directly related to the LibrarySearch command class and therefore generates the appropriate request and interprets the response for that command. Similarly, this class is related to the SetBookService command class as the option to set the book service is included on the same page as the store search for convenience.	
Collaborators	
Uses: ProxyCommandController, ParseResponseUtility, SessionManager, StateController	**Used by:** MainEmployeeController
Author: Charles Barber	

Class: BeginVisitController	
Responsibilities: This class handles the GUI view for beginning a visit. An employee can begin a visit for any visitor and therefore this class must display and handle the submission of a text field. It should be noted that this class is directly related to the BeginVisit command class and therefore generates the appropriate request and interprets the response for that command.	
Collaborators	
Uses: ProxyCommandController, ParseResponseUtility, SessionManager, StateController	**Used by:** None
Author: Charles Barber	

Class: EndVisitController	
Responsibilities: This class handles the GUI view for ending a visit. An employee can end a visit for any visitor and therefore this class must display and handle the submission of a text field. It should be noted that this class is directly related to the EndVisit command class and therefore generates the appropriate request and interprets the response for that command.	
Collaborators	
Uses: ProxyCommandController, ParseResponseUtility, SessionManager, StateController	**Used by:** None
Author: Charles Barber	

Class: VisitBegunController	
Responsibilities: This class handles the GUI view for the page displayed after the user has successfully begun a visit. Once a visit has been started, a confirmation page is displayed with detail about the action that just occurred. This class is responsible for displaying that page, specifically, as a pop-up window. Also on this page is the option to exit the popup window displays. This class handles the event execution of the close button press.	
Collaborators	
Uses: SessionManager, StateController	**Used by:** BeginVisitController
Author: Charles Barber	

Class: VisitEndedController	

Responsibilities: This class handles the GUI view for the page displayed after the user has successfully ended a visit. Once a visit has ended, a confirmation page is displayed with detail about the action that just occurred. This class is responsible for displaying that page, specifically, as a pop-up window. Also on this page is the option to exit the popup window displays. This class handles the event execution of the close button press.	
Collaborators	
Uses: SessionManager, StateController	**Used by:** EndVisitController
Author: Charles Barber	

Class: SystemController	
Responsibilities: This class handles the GUI view for the entire system settings page. This page is only accessible as an employee and can be used to perform any system action through a built in command line. Additionally, this page has input fields and buttons for generating a system report and advancing the system time. All field submissions are handled by this class along with all event actions from button presses. It should be noted that this class is directly related to the any command class through the built in command line and therefore generates the appropriate request and interprets the response for any command. Also note that this class automatically prepends the clientID to what is entered in the command line and so this input is unnecessary to perform manually.	
Collaborators	
Uses: ParseResponseUtility, ProxyCommandController, SessionManager	**Used by:** None
Author: Charles Barber	

Class: ClientController	
Responsibilities: This class handles the GUI view for the initial client connection. Each client connection is housed within its own UI tab. This class initializes each tab and informs the SessionManager which page to display on tab startup, namely login. Additionally, this class is responsible for the part of the GUI which is shared among all client connections. The most notable of which being the system clock in the bottom right corner and the menu bar at the top. This class is also responsible for handling hotkey input (e.g. CTRL-T open a new tab).	
Collaborators	
Uses: CommandContoller, SessionManager	**Used by:** GUIView
Author: Charles Barber	

Class: LoginController	

Responsibilities: This class handles the GUI view for user login. The first page shown in any new tab prompts for username and password fields and allows the user to submit input. This class handles the submission of input fields through the event action of the button press. As all the fields are required, this class will prevent execution on incomplete data and will output an error message to the user when applicable. It should be noted that this class is directly related to the LogIn command class and therefore generates the appropriate request and interprets the response for that command.
Collaborators
Uses: ParseResponseUtility, ProxyCommandController, SessionManager **Used by:** None
Author: Charles Barber

Class: MainEmployeeController	
Responsibilities: This class handles the GUI view for the main employee screen. This screen is displayed upon successful employee login and is the “home screen” for all further GUI interaction. This class is responsible for handling event execution for buttons displayed. Most of these executions simply change the view and accompanying controller. This class is also responsible for handling a search inputted from the text fields displayed on the main page. This handling involves delegation to the appropriate controller for request and response processing as well as a change in view.	
Collaborators	
Uses: ProxyCommandController, LibrarySearchController, StoreSearchController, SessionManager	**Used by:** None
Author: Charles Barber	

Class: MainVisitorController	
Responsibilities: This class handles the GUI view for the main visitor screen. This screen is displayed upon successful visitor login and is the “home screen” for all further GUI interaction. This class is responsible for handling event execution for buttons displayed. Most of these executions simply change the view and accompanying controller, however, some button actions are directly related to commands. In order to handle the execution of these button presses, this class generates the request and interprets the response for appropriate commands. This class is also responsible for handling a search inputted from the text field displayed on the main page. This handling involves delegation to the appropriate controller for request and response processing as well as a change in view. It should be noted that this class handles a simplified employee view which is presented to the visitor. In this way, the actions of the visitor are limited to those he or she is authorized to perform.	
Collaborators	
Uses: ParseResponseUtility, ProxyCommandController, LibrarySearchController, SessionManager	**Used by:** None

Author: Charles Barber

Class: StateController	
Responsibilities: This interface unites controllers under the pretense that they are each handle events for a specific system state. This interface requires those classes which implement it to define an initManager() method which ties the controller to the SessionManager.	
Collaborators	
Uses: SessionManager	**Used by:** SessionManager, AccountCreatedController, CreateController, RegisterController, VisitorRegisteredController, BookReturnedController, BorrowedResultController, PayFineController, PaymentSuccessController, ReturnBookController, BookInfoController, BorrowSuccessController, LibrarySearchController, PurchaseSuccessController, SearchResultController, StoreSearchController, BeginVisitController, EndVisitController, VisitBegunController, VisitEndedController, LoginController, MainEmployeeController, MainVisitorController, SystemController
Author: Charles Barber	

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, BookSearch
Author: Team B	

Class: LibraryState	
Responsibilities: This interface holds a method isOpen(). The classes inheriting from it determines if the library is open or closed.	
Collaborators	
Uses: None	**Used by:** ClosedState, OpenState
Author: Team B	

Class: ClosedState	
Responsibilities: This class inherits from LibraryState to show that the library is closed.	
Collaborators	
Uses: LibraryState	**Used by:** ProxyCommandController
Author: Team B	

Class: OpenState	
Responsibilities: This class inherits from LibraryState to show that the library is open.	
Collaborators	
Uses: LibraryState	**Used by:** ProxyCommandController
Author: Team B	

Class: Employee	
Responsibilities: This class holds the state and behaviors for an employee. The only state an employee has is a visitor as employees are considered visitors also.	
Collaborators	
Uses: None	**Used by:** CreateAccount
Author: Team B	

Class: ISBN	
Responsibilities: This class stores the ISBN of a book. The methods in ISBN check if the given ISBN is valid in its 10 digit and 13 digit forms. There is also an equals() method that checks if two ISBNs are equal which is used in book searching.	
Collaborators	
Uses: None	**Used by:** LBMS, LibrarySearch, StoreSearch
Author: Team B	

Class: PhoneNumber	
Responsibilities: This class holds the state for a phone number. A phone number has the areaCode (first 3 digits), the exchangeCode (the 3 digits after), and the extension which is the last 4 digits. The class also has a toString() method to convert the phone number into a string.	
Collaborators	
Uses: None	**Used by:** LBMS, RegisterVisitor
Author: Team B	

Class: Session	
Responsibilities: When a user logins into their account, they start a session. Each session has its own clientID, an associated visitor, the type of service (local or Google), a stack to hold commands to undo, a second stack to hold commands to redo, and a bookSearch arraylist which holds the books from the last search.	
Collaborators	
Uses: None	**Used by:** LBMS, ClientConnect, LogOut
Author: Team B	

Class: SystemDateTime	
Responsibilities: This class is a custom made datetime class for the LBMS. It is used to store the system's date and time. Through this class, the system time can be advanced but the time can functions on its own like a real clock.	
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
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 unless	

their books are overdue. If they are overdue, the visitor's total fines increases based on the fines stored in the transactions. After the visitor has completely paid all of their fines, they can finally return their books.	
Collaborators	
Uses: LBMS	**Used by:** LBMS, BeginVisit, Borrow, CreateAccount, EndVisit, FindBorrowed, LogIn, RegisterVisitor, Return, StatisticsReport, UserSearch
Author: Team B	

Search

Class: Search	
Responsibilities: Search is a generic interface to facilitate the two classes that implement it (BookSearch and UserSearch). It contains two main methods, findAll() and findFirst(). FindAll() finds all objects that fit the criteria given in a search while findFirst() only finds the first object that matches. This interface has two other methods, createPredicate() and filterStream() which aids in searching.	
Collaborators	
Uses: None	**Used by:** BookSearch, UserSearch
Author: Nicholas Feldman	

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. This class implements the createPredicate() and filterStream() methods in the Search	
Collaborators	
Uses: Search, LBMS, Book	**Used by:** BookPurchase, FindBorrowed, LibrarySearch, StoreSearch
Author: Nicholas Feldman	

Class: GoogleAPISearch	
Responsibilities: This class allows the user to search the Google Books API. They can search for books by title, author, ISBN, and publisher. The class uses a URL to retrieve JSON data from Google Books which is then parsed to form a list of Book objects.	
Collaborators	
Uses: None	**Used by:** StoreSearch
Author: Nicholas Feldman	

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, RegisterVisitor, Return
Author: Nicholas Feldman	

Views

Class: View	
Responsibilities: The View interface only has one method, run() which is used to start a view.	
Collaborators	
Uses: None	**Used by:** APIView, GUIView
Author: Edward Wong	

Class: ViewFactory	
Responsibilities: This class uses the StartType held in the LBMS to determine what view to run. There are only two views to choose from: API and GUI.	
Collaborators	
Uses: LBMS, APIView, GUIView	**Used by:** LBMS
Author: Edward Wong	

Class: APIView	
Responsibilities: This class establishes the initial API view of LBMS. From here, the ProxyCommandController takes in user input and processes the inputted requests in order to execute commands.	
Collaborators	
Uses: ProxyCommandController, View	**Used by:** ViewFactory
Author: Edward Wong	

Class: GUIView	
Responsibilities: This class establishes the GUI view of LBMS. It also has a stop() method to discontinue the view. After starting, the user can begin to interact with GUI.	
Collaborators	
Uses: ClientController, View	**Used by:** ViewFactory
Author: Edward Wong	

Class: SessionManager	

Responsibilities: This class operates a session (aka a tab in the GUI). It holds a close() method which log outs the visitor logged into the session and the client disconnects. It has other methods to display the session and to load a file to format the GUI.	
Collaborators	
Uses: ParseResponseUtility, ProxyCommandController, StateController	**Used by:** LoginController, MainEmployeeController, MainVisitorController, SystemController, AccountCreatedController, CreateController, RegisterController, VisitorRegisteredController, BookReturnedController, BorrowedResultController, PayFineController, ReturnBookController, BookInfoController, LibrarySearchController, SearchResultController, StoreSearchController, BeginVisitController, EndVisitController, VisitBegunController, VisitEndedController, ClientController
Author: Edward Wong	