<Library Book Management System>

Design Documentation
Prepared by Team E

- Joseph S.
- Anthony F.
- Dylan G.
- Alanna M.

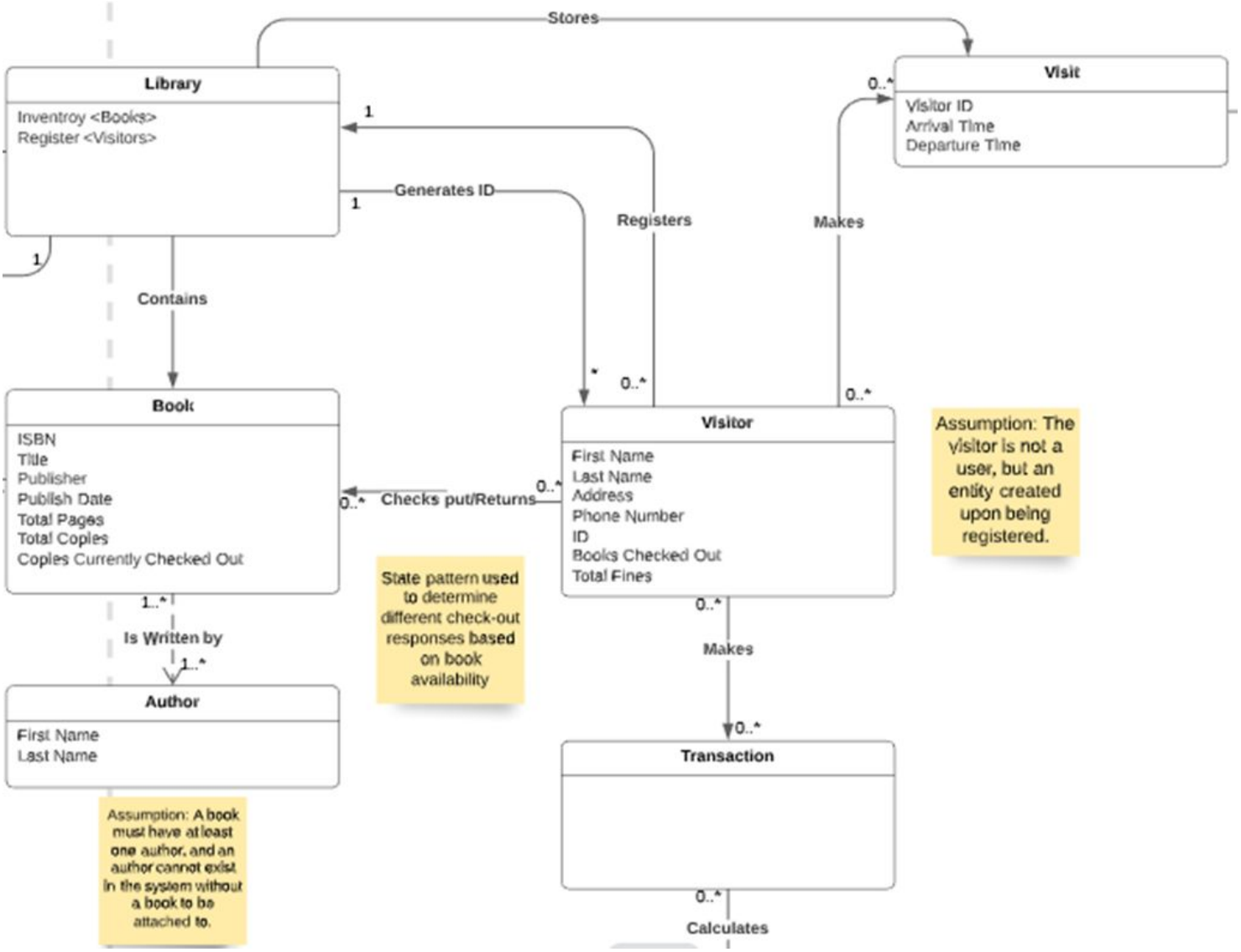
Summary

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 employee's use. The system allows for 14 different requests sent by the employee that interact with the library model. The library model itself is responsible for registering/tracking visitors and books as well as providing the information requested by the user.

This system was designed for reuse and extendibility, with a focus on using Design Patterns to help achieve this goal. The State pattern was used to verify that a visitor is able to check out more books. This library only allows each visitor to check out a max of books at a time, so this pattern helps implement different behavior depending on that state. The Strategy pattern was used to calculate the fine applied when a visitor checks a book back in. This library allows a book to be checked out for 1 week before applying fines and over time the rate changes, so this patterns helps implement different calculations depending on the check in due date and the current date.

Currently, all interactions are displayed via a command line-esqu Graphical User Interface that allows users to enter in commands and will display a separate window with a response.

Domain Model



The broad view of our domain model shows the interactions between key components in the LBMS. To start the Library class holds records of all visitors, visits, and books, making it the ideal place to direct informational requests. Connected to the library is a Library Entry class (not shown) which holds information from the Book class and also monitors the number of available copies for each book. Also connected to the Library is a Visitor class. The Visitor class contains all the information a visitor must enter when they first come to the library, it also interacts with the Transaction (also known as Book Check-in Fine Calculator subsystem) and the Check Out State subsystem (not shown) to determine how to handle check ins and check outs. The Visit class is used to track each individual visit for visitors and is referenced by the Library/visitor classes to calculate system statistics.

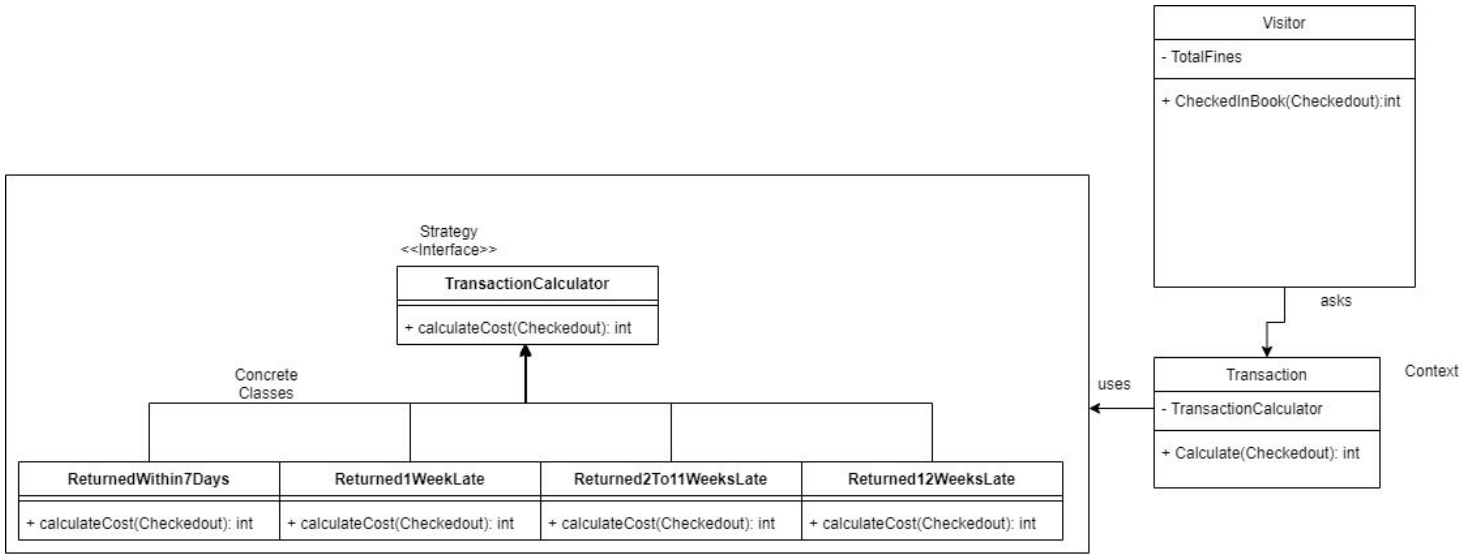
For a more detailed look at our domain model, please follow this link:
https://lucid.app/lucidchart/invitations/accept/39fe6be7-15c3-464d-a2a1-16d5c0021bee

System Architecture

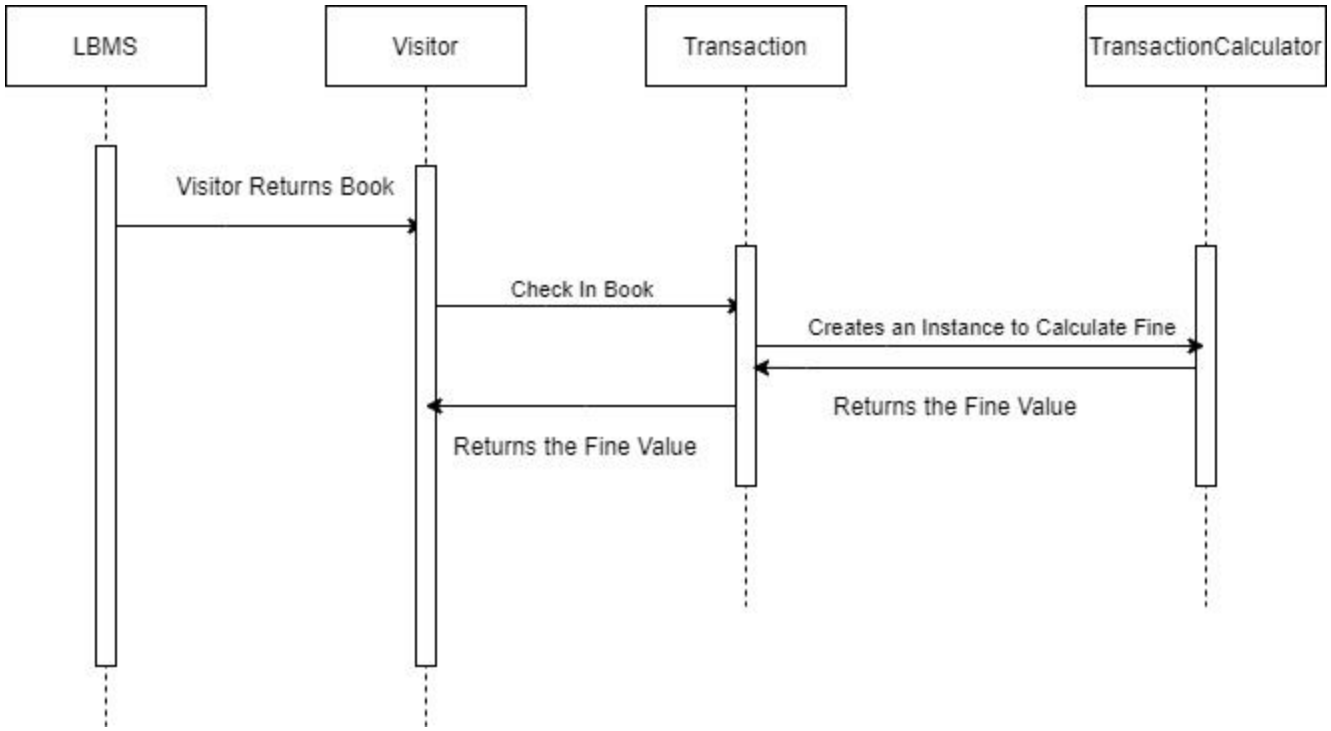
Subsystems

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

Book Check-in Fine Calculator



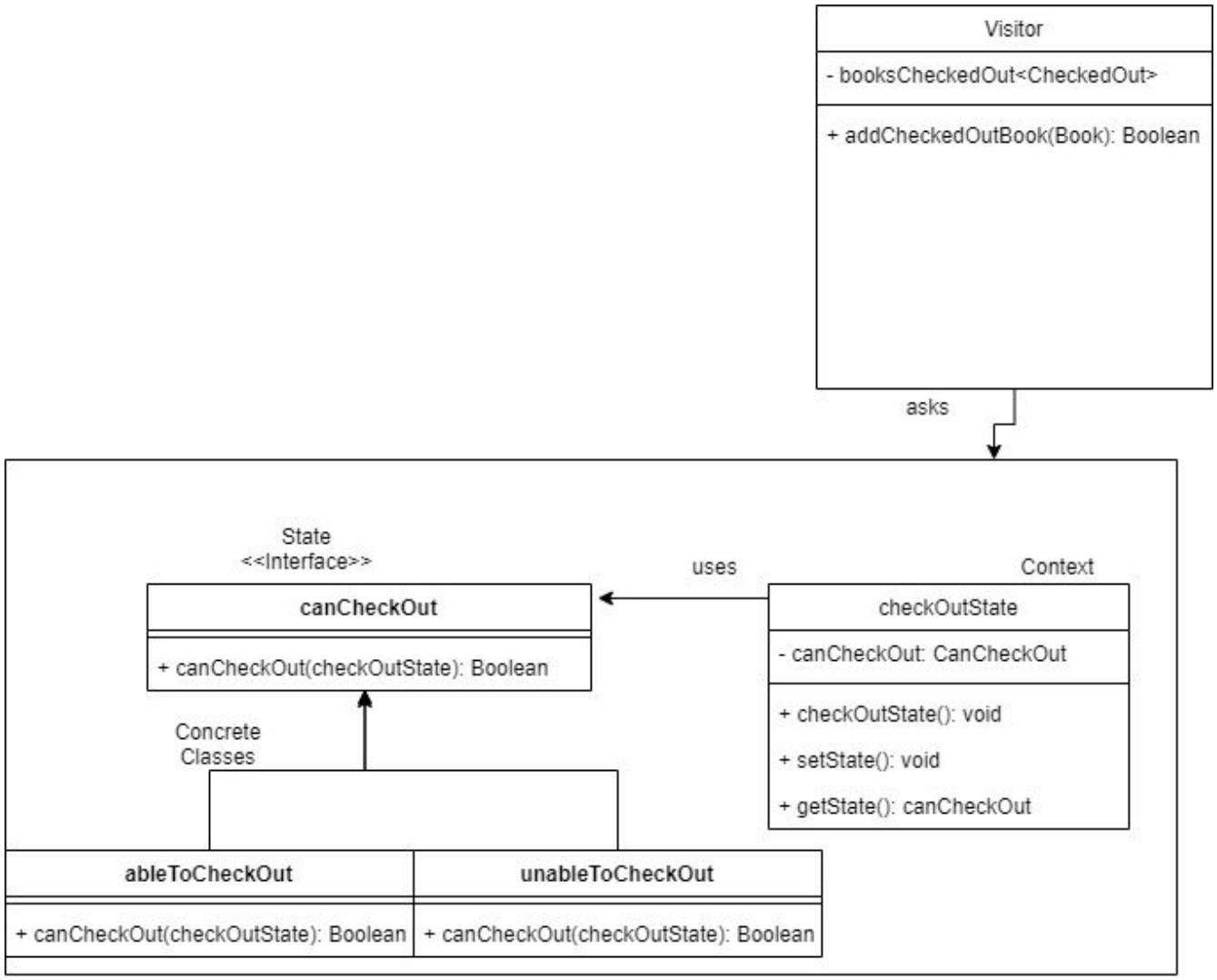
>>>>>>>>>>>



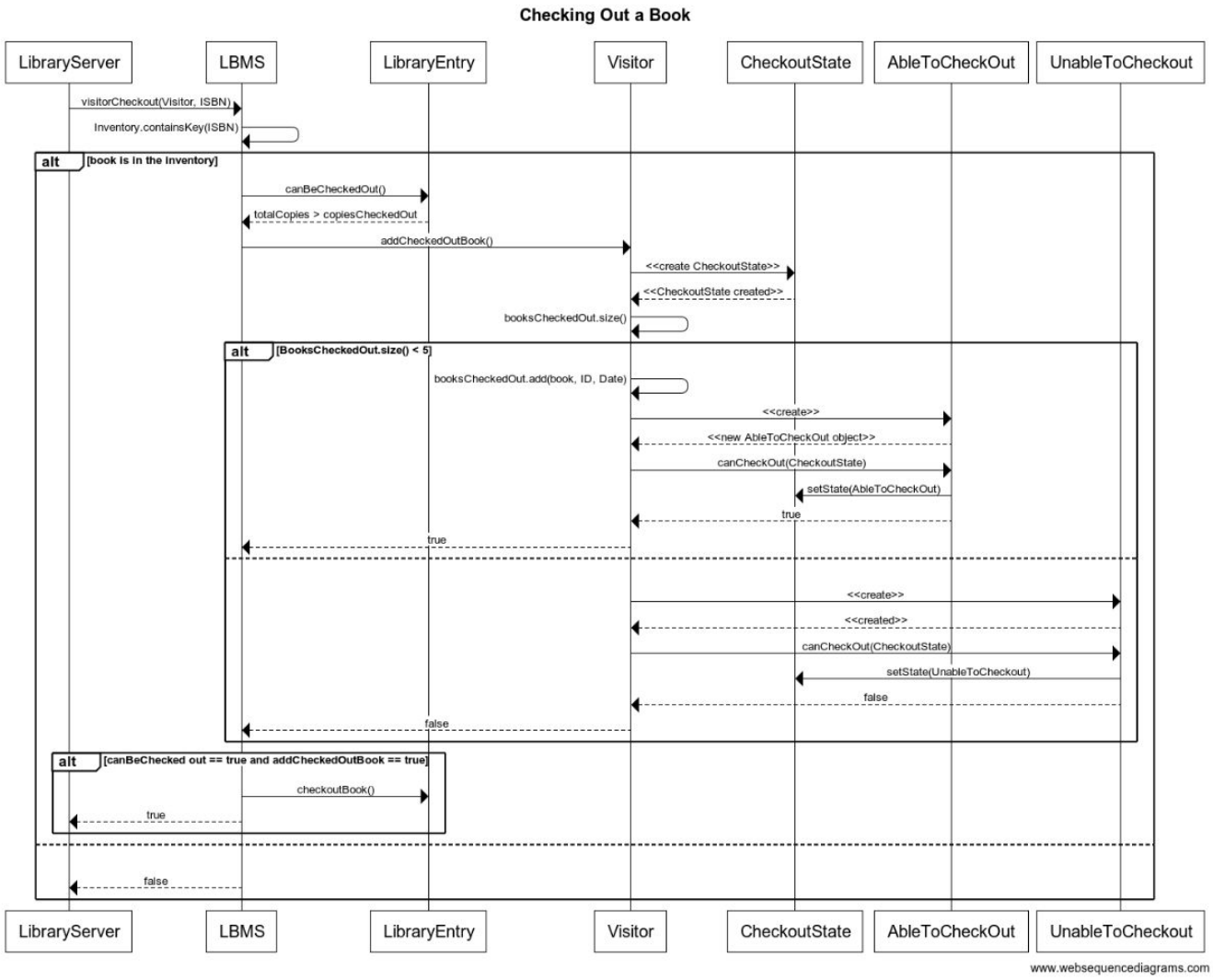
>>>>>>>>>>>

Name: Book Check-in Fine Calculator		**GoF pattern: Strategy**
Participants		
Class	**Role in GoF pattern**	**Participant's contribution in the context of the application**
Visitor	Calling Class	Uses its state to create an instance of the context class referencing the concrete class needed to calculate the fine.
Transaction	Context	Creates an instance of the Strategy Interface and uses it to calculate the fine.
TransactionCalculator	Strategy Interface	The interface implemented by all of the Concrete Classes to calculate the fine.
ReturnedWithin7Days	Concrete Class	An implementation of the TransactionCalculator that will return a fine of 0$.
Returned1WeekLate	Concrete Class	An implementation of the TransactionCalculator that will return a fine of 10$.
Returned2To11WeeksLate	Concrete Class	An implementation of the TransactionCalculator that will return a fine between 12$ to 28$ based on the number of weeks the return is late.
Returned12WeeksLate	Concrete Class	An implementation of the TransactionCalculator that will return a fine of 30$.
Deviations from the standard pattern: N/A		
Requirements being covered: 5 - The LBMS shall track books checked out by visitors.		

Book Check Out State



A visitor must be eligible to check out a book before they can complete a transaction. This is determined by the current number of books the visitor has in their possession (5 is the max). The visitor model already keeps track of a list of currently checked books, so that information can be used to determine the state of the visitors eligibility. This led to the use of the State pattern to verify if a book check out is possible.



Link to better resolution version:

https://tinyurl.com/y6boquj4

The process begins with the server prompting the user to enter a command. For this example we will follow the case of a "Borrow" command.

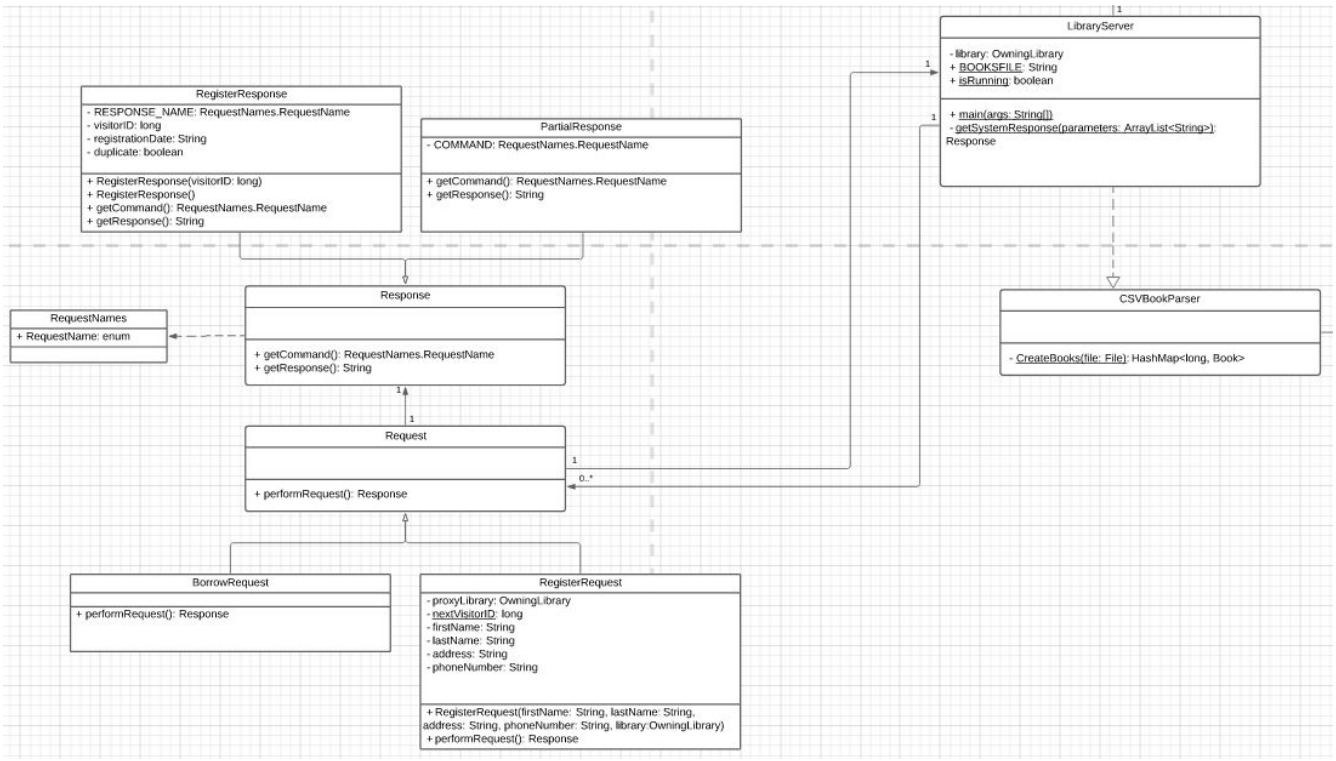
1. First, the server ensures that the book attempting to be returned is a book that this library actually carries.
    a. If not, the response returns an error message, informing the user that the ISBN does not exist.
    b. If so, the LBMS ensures that there is a copy of the entered book available to borrow, and saves the results for later use.
    c. At this point the LBMS attempts to have the visitor check out the book, checking if the visitor is in the AbleToCheckOut State, or the UnableToCheckOut state
        i. This process actually involves creating a new state for the Visitor whenever they attempt to check out a book. This is determined by how many books the visitor currently has checked out
            1. If the visitor has 5 (or more, though it should never end up in this situation), books checked out, the visitor gets an UnableToCheckOut

state
2. If the visitor has less than 5 books checked out, the visitor gets an AbleToCheckOut state

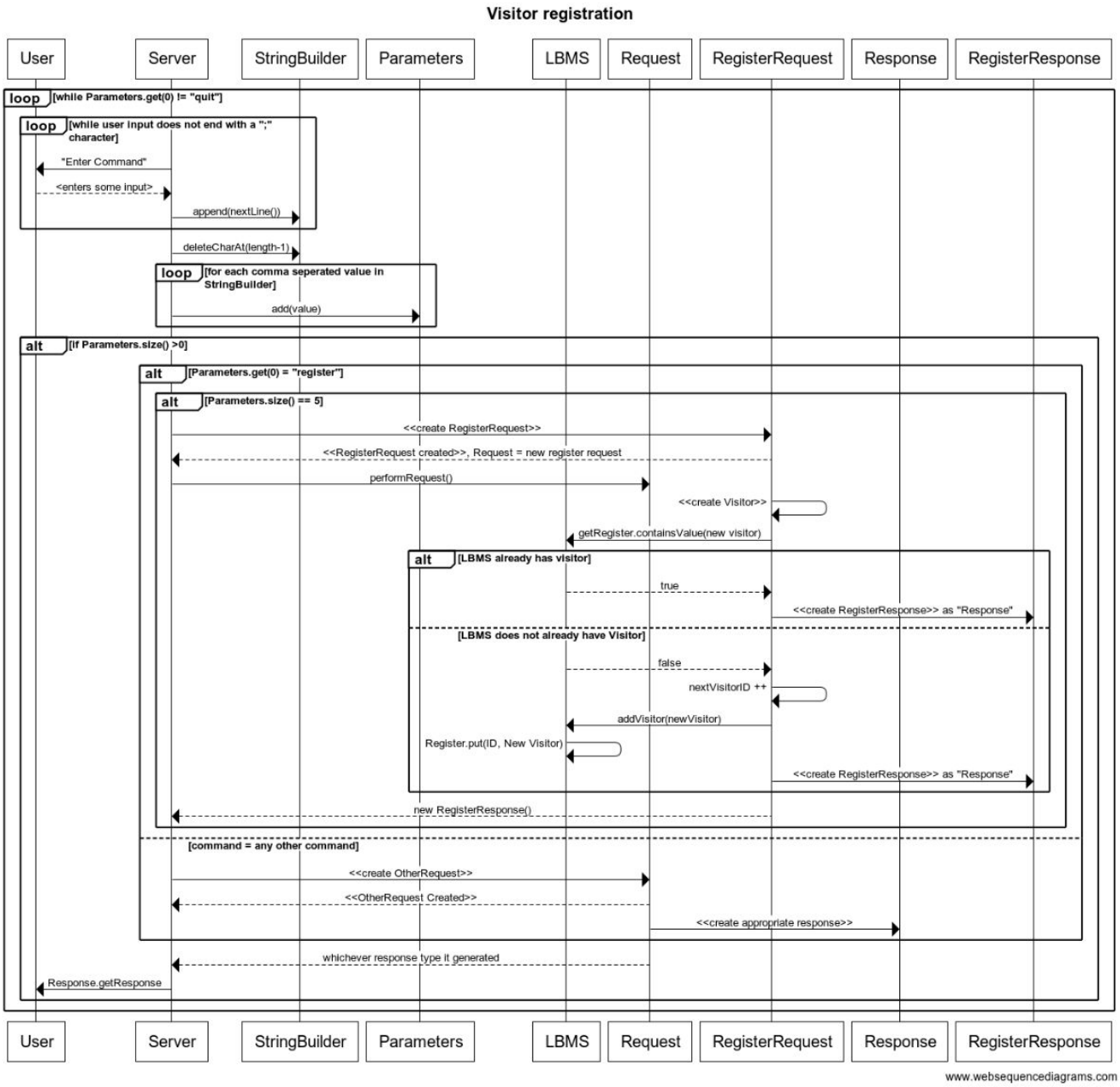
d. At this point, we check that there is a book available, and that the visitor is capable of checking out the book.
   i. If both cases are true, then the book is checked out of the library, and the visitor gets a copy of the book with a due date.
   ii. If either, or both, cases are false, then nothing happens, and the user gets a message detailing what went wrong.

Name: Book Check Out State		**GoF pattern: State**
Participants		
Class	Role in GoF pattern	Participant's contribution in the context of the application
Visitor	Calling Class	Uses its state to create an instance of the context class referencing the concrete class needed to check out a book.
checkOutState	Context	Creates an instance of the State Interface and uses it to set the state, this class also implements a method to return the current State.
canCheckOut	StateInterface	The interface implemented by all of the Concrete Classes to establish the state.
ableToCheckOut	Concrete Class	An implementation of the canCheckOut that will return true.
unableToCheckOut	Concrete Class	An implementation of the canCheckOut that will return false.
Deviations from the standard pattern: N/A		
Requirements being covered: 5 - The LBMS shall track books checked out by visitors.		

Request Response System



>>>>>>>>>>>>>>



Link to better resolution version:

https://tinyurl.com/yxk9ev95

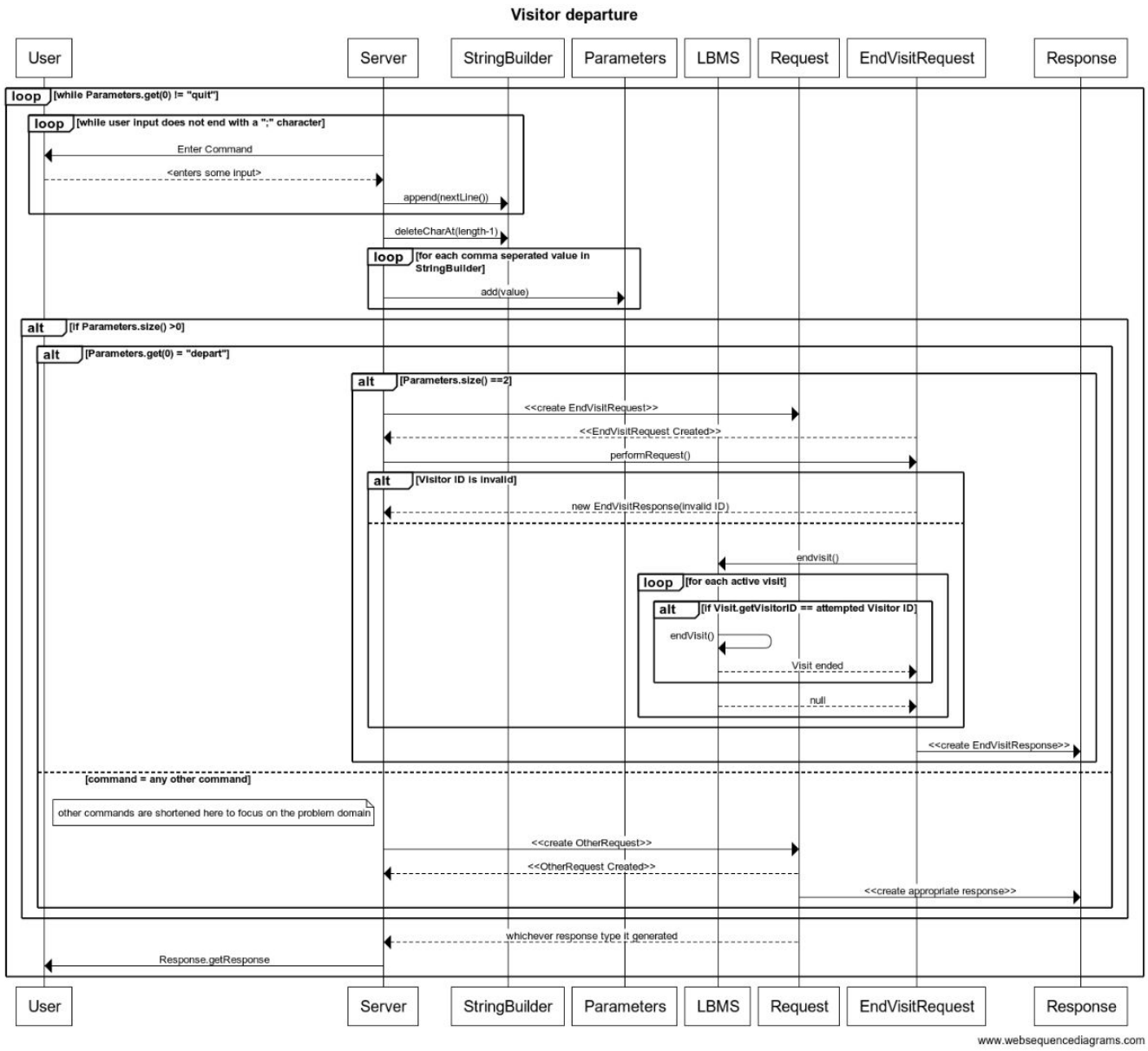
The process begins with the server prompting the user to enter a command. For this example we will follow the case of a "Registration" command.

1. First the system ensure that an appropriate amount of parameters was entered by the user (5 in this case.
    a. If that is true, the server creates a response (in this case, a register response) and calls it's performRequest function.
    b. This prompts the Request to generate an appropriate response based on the parameters entered. There is little fact-checking done here, as we cannot validate every phone

number and address at this stage.

c. The request does ensure that the entered visitor is not already registered by checking that no registered user has the same name, address, and phone number.
    i. If there is a duplicate, the response lists the user information entered and a "failure to register" message
    ii. if there is no duplicate, the visitor is registered

2. If any other type of command is entered, the system will skip the above and perform that Request instead.



Link to better resolution version:

https://tinyurl.com/y2kw2pgn

The process begins with the server prompting the user to enter a command. For this example we will follow the case of a "Depart" command.

3. First the system ensure that an appropriate amount of parameters was entered by the user (2 in this case.
    a. If that is true, the server creates a response (in this case, an End Visit Response) and calls it's performRequest function.
    b. This prompts the Request to generate an appropriate response based on the parameters entered (in this case, simply the visitor's ID number).
    c. The request validates if the ID entered is an actual, registered visitor
        i. If not, the response informs the user that the id is invalid, and lists the ID number

ii. If the visitor does exist it ends any current visit they may be in
    1. Regardless of if a visit was actually closed, the response informs the user that this was a success, as, ultimately, the visitor in question is no longer visiting.

4. If any other type of command is entered, the system will skip the above and perform that Request instead.

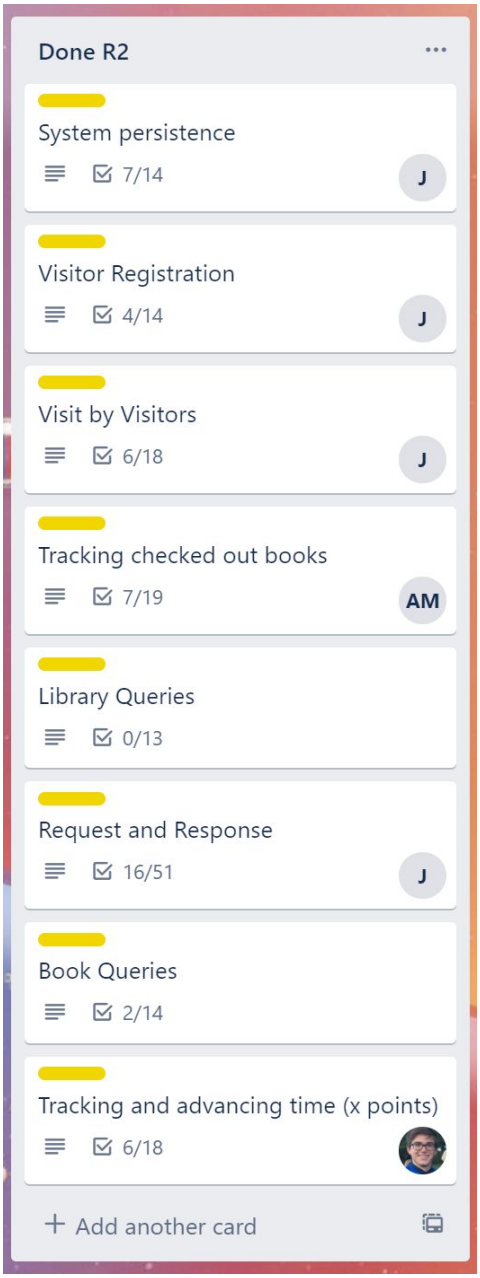
Name: Request Response System		**GoF pattern: Command**
Participants		
Class	**Role in GoF pattern**	**Participant's contribution in the context of the application**
Request	Request	Sends information to the server in a particular format to be processed
Response	Response	Sends information from the server in a particular format after processing the request
LibraryServer [GUI]	Client	
LBMS	Service	Fulfills the request if valid and sends the response as needed
Deviations from the standard pattern: The Client and Service are loosely defined, as the client and server are one application.		
Requirements being covered: 5 - The LBMS shall track books checked out by visitors.		

User/Library Interaction System

Name: GUI System		**GoF pattern: Model-View-Controller**
Participants		
Class	**Role in GoF pattern**	**Participant's contribution in the context of the application**
MyView	View	Presents a display to the user to interact with. Sends the requests to the Library's server.
LibraryServer	Controller	Processes the requests and updates the view.
[State]Library	Models	Provides the backend and organization for the controller to interact with.
Deviations from the standard pattern: N/A		
Requirements being covered: 5 - The LBMS shall track books checked out by visitors.		

Status of the Implementation

The implementation for the LBMS has the necessary functionality to advance time, run from a preserved state saved to files for when it's not running and loaded on startup. In addition, there is a GUI interface which allows the user to run commands and view their results from a window rather than a traditional CLI. This implementation is rudimentary, but meets the requirements. Additional features that were not included in the final product included a way to undo-redo, and a more advanced GUI that did not rely so heavily on commands like the CLI that preceded it.



Appendix

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

Class: `Author`	Author - firstName: String - lastName: String - writtenBookISBN: ArrayList<Integer> + Author(firstName: String, lastName: String, writtenBookISBN: ArrayList<Integer>) + addISBN(ISBN: Integer)
Responsibilities: Represents the Author of a series of books in the LBMS, and contains references to the books they've authored	
Collaborators: N/A	
Uses: N/A	**Used by:** N/A
Author: Alanna M.	

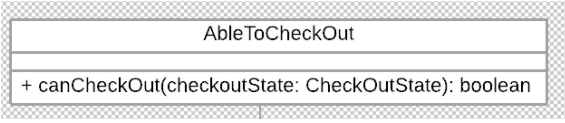
Class: `Visit`	Visit - VisitorID: int - ArrivalTime: Date - DepartureTime: Date + Visit(visitorID: int) + endVisit()
Responsibilities: Represents a full visit of a visitor to the library, particularly a span of time	
Collaborators: N/A	
Uses: N/A	**Used by:** N/A
Author: Alanna M.	

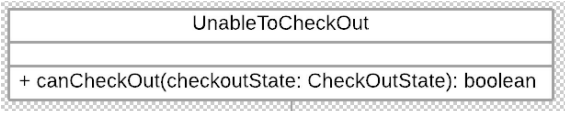
Class: `Validator`	Validator --- # validateAndParseLong(longString: String): long # validateAndParseInt(intString: String): int
Responsibilities: Ensures that server requests are in the correct format, and that the data is valid	
Collaborators: N/A	
Uses: N/A	**Used by:** N/A
Author: Joseph S.	

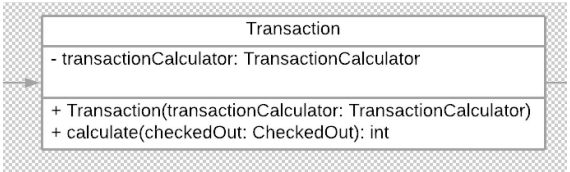
Class: `CanCheckOut`	CanCheckOut --- + canCheckOut(checkoutState: CheckOutState): boolean
Responsibilities: Serve as an interface for the state of a book, and whether it can be checked out or not.	
Collaborators: N/A	
Uses: AbleToCheckOut, UnableToCheckOut	**Used by:** CheckOutState
Author: Alanna M.	

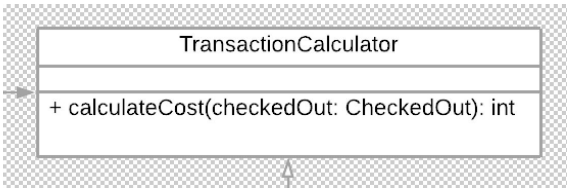
Class: `CheckOutState`	CheckOutState --- - canCheckOut: CanCheckOut --- + CheckOutState() + setState(canCheckOut: CanCheckOut) + getState(): CanCheckOut
Responsibilities: Serves as a container class for the state of the book and whether it can be checked out or not.	

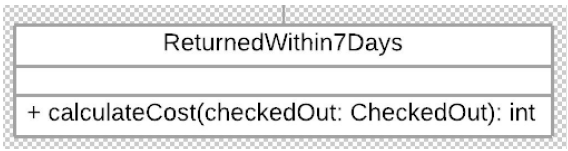
Collaborators: ...	
Uses: CanCheckOut	**Used by:** Visitor
Author: Alanna M.	

Class: `AbleToCheckOut`	
Responsibilities: The state of being able to be checked out immediately by the LBMS	
Collaborators: ...	
Uses: N/A	**Used by:** CanCheckOut
Author: Alanna M.	

Class: `UnableToCheckOut`	
Responsibilities: The state of being unable to be checked out immediately by the LBMS	
Collaborators: ...	
Uses: N/A	**Used by:** CanCheckOut
Author: Alanna M.	

Class: `Transaction`	
Responsibilities: Uses the TransactionCalculator to complete the cost calculation.	
Collaborators: ...	
Uses: TransactionCalculator	**Used by:** Visitor
Author: Alanna M.	

Class: `TransactionCalculator`	
Responsibilities: Interface used to calculate the transaction.	
Collaborators: ...	
Uses: ReturnedWithin7Days, Returned1WeekLate, Returned2To11WeeksLate, , Returned12OrMoreWeeksLate	**Used by:** Transaction
Author: Alanna M.	

Class: ReturnedWithin7Days	

Responsibilities: Calculates a transaction cost of $0 when the book is returned on or before the due date	
Collaborators: ...	
Uses: N/A	**Used by:** TransactionCalculator
Author: Alanna M.	

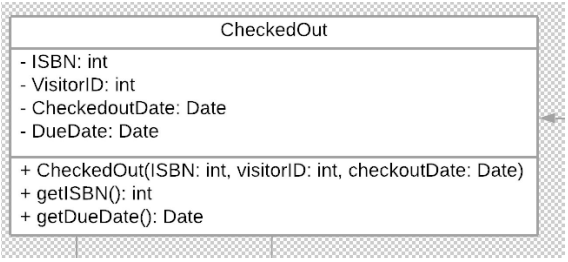
Class: Returned1WeekLate	Returned1WeekLate + calculateCost(checkedOut: CheckedOut): int
Responsibilities: Calculates a transaction cost of $10 when the book is returned 1-7 days past the due date	
Collaborators: ...	
Uses: N/A	**Used by:** TransactionCalculator
Author: Alanna M.	

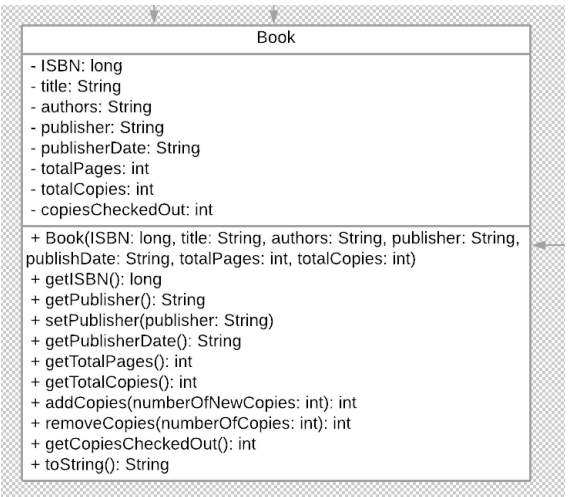
Class: Returned2To11WeeksLate	Returned2To11WeeksLate + calculateCost(checkedOut: CheckedOut): int
Responsibilities: Calculates a transaction cost by adding $2 to the initial fine each week it is late	
Collaborators: ...	
Uses: N/A	**Used by:** TransactionCalculator
Author: Alanna M.	

Class: Returned12OrMoreWeeksLate	Returned12OrMoreWeeksLate + calculateCost(checkedOut: CheckedOut): int
Responsibilities: Calculates a transaction cost of $30 when the book is 12 or more weeks late	
Collaborators: ...	
Uses: N/A	**Used by:** TransactionCalculator
Author: Alanna M.	

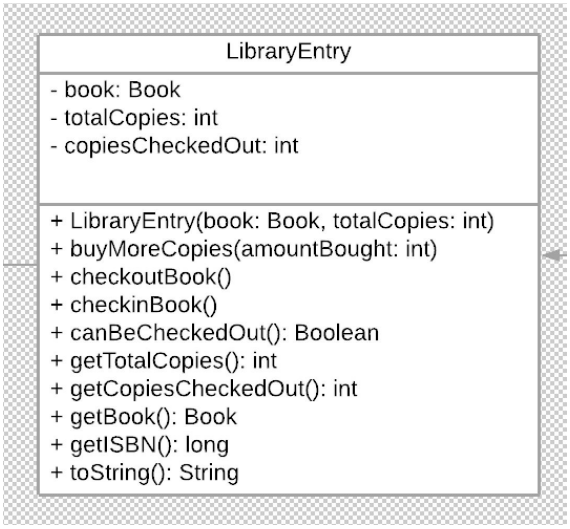
Class: `Visitor`	Visitor - firstName: String - lastName: String - address: String - phoneNumber: String - ID: long - booksCheckedOut: ArrayList<Book> - totalFines: double + Visitor(firstName: String, lastName: String, address: String, phoneNumber: String, ID: long) + getFirstName: String + setFirstName(firstName: String) + getLastName(): String + setLastName(lastName: String) + getAddress(): String + setAddress(address: String) + getPhoneNumber(): String + setPhoneNumber(phoneNumber: String) + getID(): long + addCheckedOutBook(book: Book): boolean + checkInBook(checkedOut: CheckedOut): int + equals(o: Object): boolean + toString(): String
Responsibilities: A model the represents a visitor to a library	

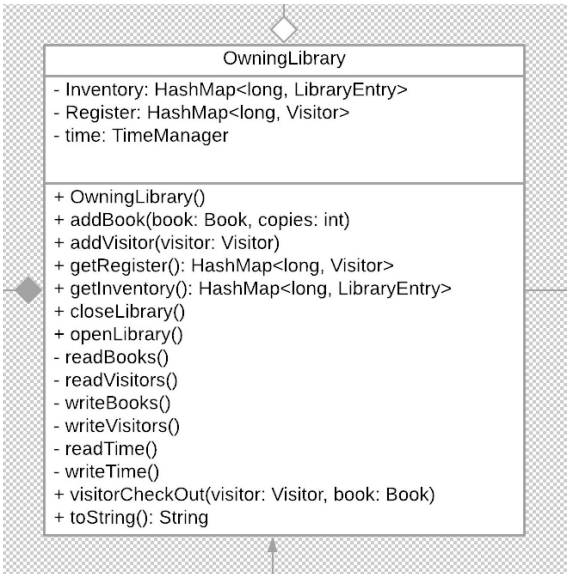
Collaborators: ...	
Uses: CheckedOutState, CheckedOut, Transaction	**Used by:** OwningLibrary
Author: Joseph S.	

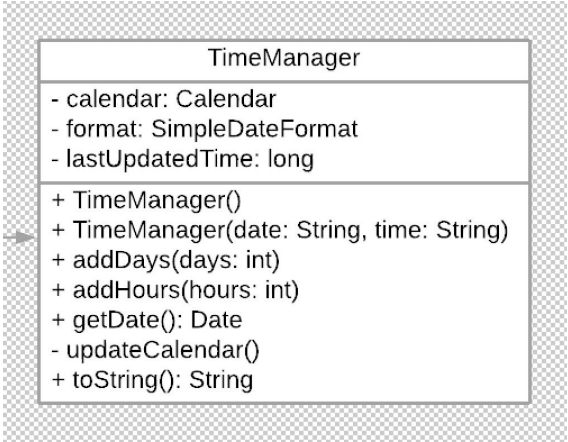
Class: `CheckedOut`	
Responsibilities: A class that holds a checked out book ISBN, the visitor ID, the checkout date, and the due date	
Collaborators: ...	
Uses: Book	**Used by:** Visitor
Author: Alanna M.	

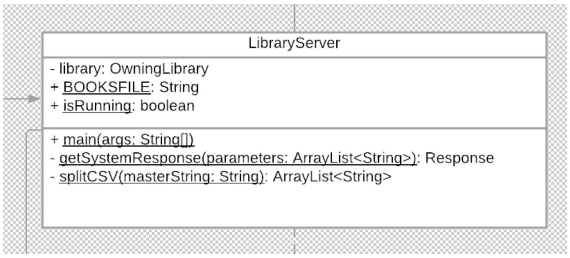
Class: `Book`	

Responsibilities: A model the represents a visitor to a library	
Collaborators: ...	
Uses: N/A	**Used by:** CheckedOut, LibraryEntry
Author: Joseph S.	

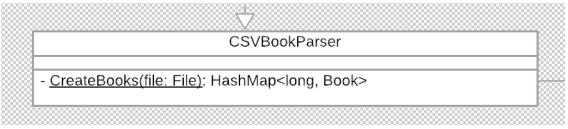
Class: `LibraryEntry`	
Responsibilities: Book class with # of available copies and # of checked out copies	
Collaborators: ...	
Uses: Book	**Used by:** OwningLibrary
Author: Alanna M Joseph S.	

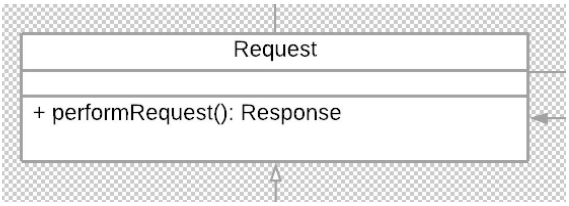
Class: `OwningLibrary`	
Responsibilities: A model that represents a library using the LMS. The library will keep track of all visitors in a hashmap using their ID and all books using their ISBN	
Collaborators: ...	
Uses: TimeManager, LibraryEntry, Visitor	**Used by:** LibraryServer
Author: Alanna M. Joseph S. Dylan G.	

Class: `TimeManager`	
Responsibilities: A utility used to track time in the library, mostly for simulation purposes.	
Collaborators: ...	
Uses: N/A	**Used by:** OwningLibrary
Author: Anthony F.	

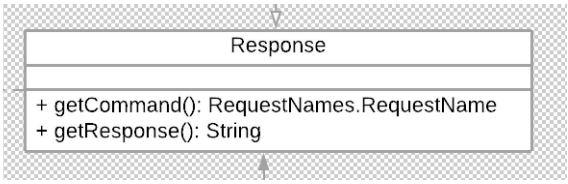
Class: `LibraryServer`	
Responsibilities: The main entry point of the application. Runs the necessary I/O for the LBMS to function.	
Collaborators: ...	
Uses: CSVBookParser, OwningLibrary, Request	**Used by:** N/A
Author: Joseph S.	

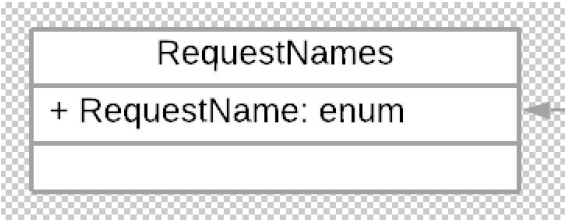
Alanna M. Dylan G. Anthony F.	

Class: `CSVBookParser`	
Responsibilities: This class parces a given text file of books using a format. This is only for use in reading in the Book text file.	
Collaborators: ...	
Uses: N/A	**Used by:** LibraryServer
Author: Joseph S.	

Class: `Request`	
Responsibilities: To serve as a format for receiving user requests and processing them in an orderly fashion.	
Collaborators: ...	
Uses: LibraryServer, Response	**Used by:** LibraryServer, [Request Implementations]

Author: Joseph S.	

Class: `Response`	
Responsibilities: Sending responses to the user after a request is made and processed by the LBMS	
Collaborators: ...	
Uses: N/A	**Used by:** Request, [Response Implementations]
Author: Joseph S.	

Class: `RequestNames`	
Responsibilities: Provide organization to the requests and responses in the LBMS, and identify them.	
Collaborators: ...	
Uses: N/A	**Used by:** [Request Implementations], [Response Implementations]
Author: Joseph S.	

Glossary

CLI - Command-Line Interface: A method of interacting with s system consisting of issuing commands and viewing responses.

GoF - Gang of Four: A name in reference to a set of design patterns.

GUI - Graphical User Interface: A method of interacting with a system consisting of windows and components.

I/O - Input/Output: The streams of data going to and from the system.

LBMS - Library Book Management System: A software project used to oversee libraries.

MVC - Model-View-Controller: A design pattern used for user interfaces.

N/A - Not Applicable: Information for that field does not exist or is not known.