<LBMS Project Release 1>

Design Documentation
Prepared by Team 3:

- Hersh Nagpal <hxn6993@rit.edu>
- Luis Gutierrez <lxg8800@rit.edu>
- Michael Kha <mxk5025@rit.edu>
- Jack Li <jl1429@rit.edu>

Summary

Purpose

The Library Book Management System (LBMS) is a server-side system to handle client requests for managing library information. This information includes actions that take place and involve visitors and books. Requests consist of the following: registering visitors, begin visits, ending visits, searching the library for books, borrowing books, finding borrowed books under a visitor, returning books, paying accumulated overdue book fines, searching the bookstore for books, purchasing books from the bookstore for the library, getting statistical reports for a number of days back, and keeping track of the time. Additionally, the system contains features to save and restore the system upon shutdown.

Outcomes

The team designed and implemented the management system. The LBMS provides functionality for storing and manipulating data about the library and its visitors. We implemented a functioning command-line application in only Java without an external database. The main data was stored in databases for visitors, books, and checkouts. Requests interact with the databases directly.

A .jar file is used to run the program. LBMS is executed through a start.bat file which sets up the required environment variables, initializes the program state, and execute the .jar file.

Domain Model

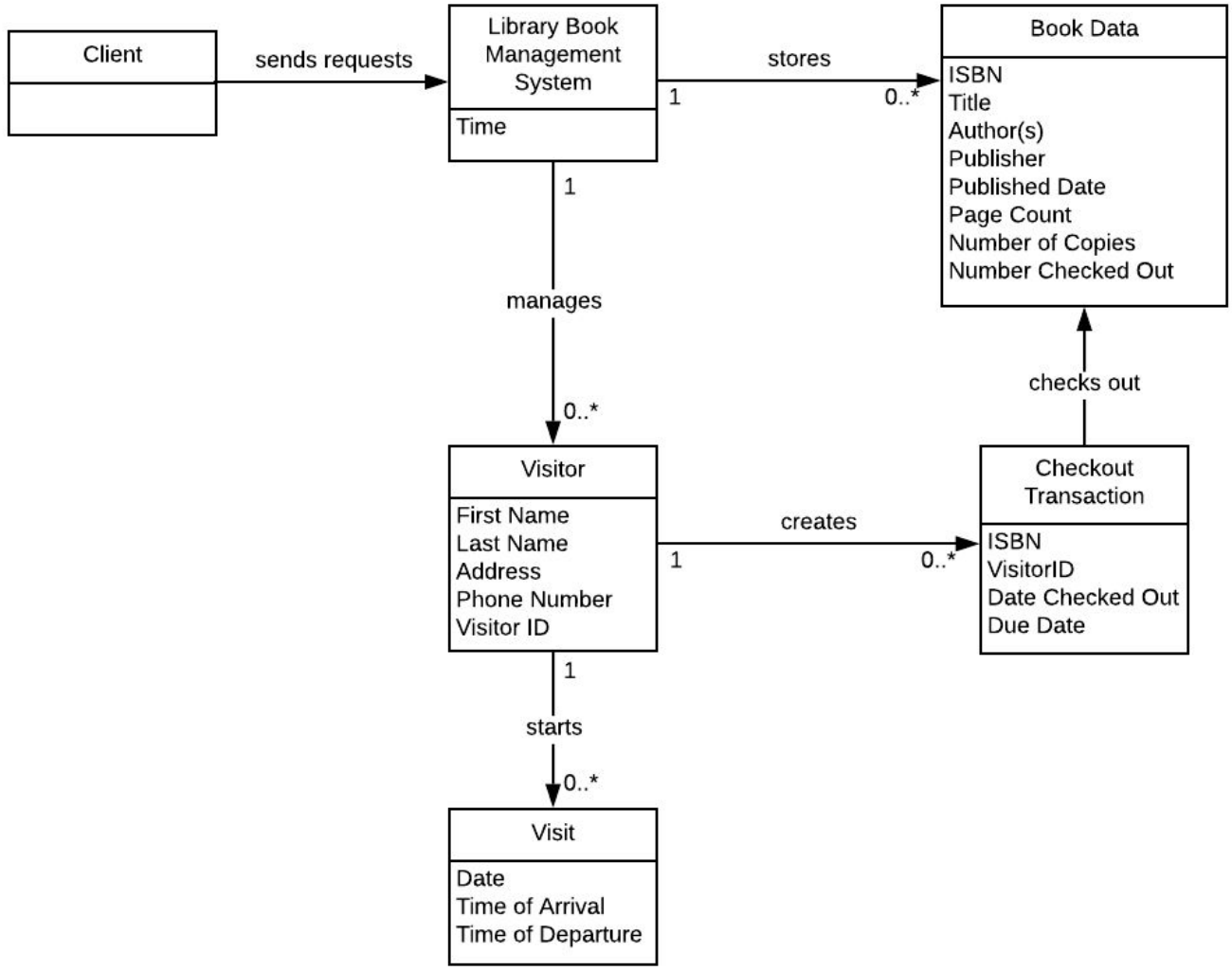


Figure 1. High-level depiction of the entities in the problem statement. Clients must be able to send requests to the LBMS for the purpose of managing the visitor, book, and checkout data.

System Architecture

The model below depicts the architecture of the system. The LB Server initializes the state of the Controller and Model packages to begin taking user input. User input is then delegated to the Request Parser for the creation of the correct requests. A request consists of a command and its parameters. Requests are also self-containing, so that they can execute an operation on its own by calling methods in the Model package. The Model consists of several subsystems for each type of data. The library subsystem maintains the state of the library being open and keeps track of the time. The book subsystem is responsible for maintaining the books. The visitor subsystem is responsible for tracking visitors and visits. The checkout subsystem deals with holding open checkout transactions.

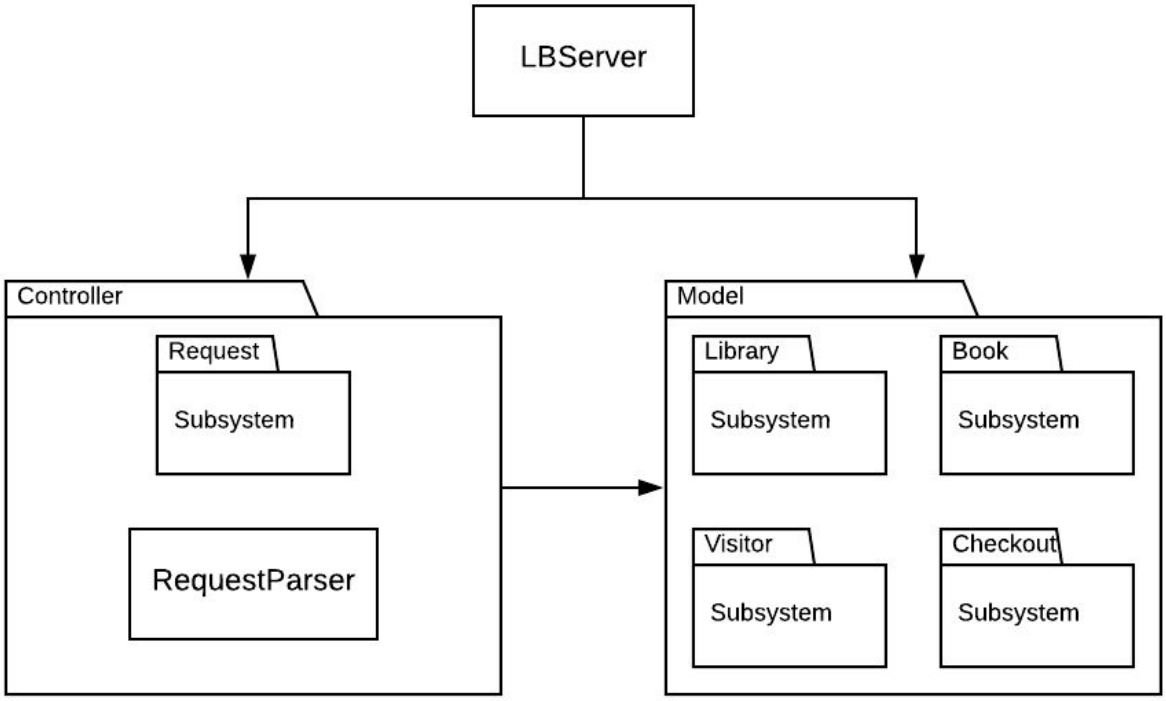


Figure 2. High-level depiction of the entities in the problem statement. Clients must be able to send requests to the LBMS for the purpose of managing the visitor, book, and checkout data.

Subsystems

The following sections provides detailed design for specific subsystems described in the system architecture. The subsystems includes the request, library, book, checkout, and visitor subsystems. We decided to divide the system based on our architecture and general responsibilities. The model subsystems are distinct and do not interact with each other, while the library subsystem is directly coupled to the databases. As a result, we delegated the responsibilities of executing the commands into the library system class and having each individual request's command call methods in the system class. We limited the responsibilities of this system class to call the database methods, perform some error checking, and create the right response.

Request Subsystem

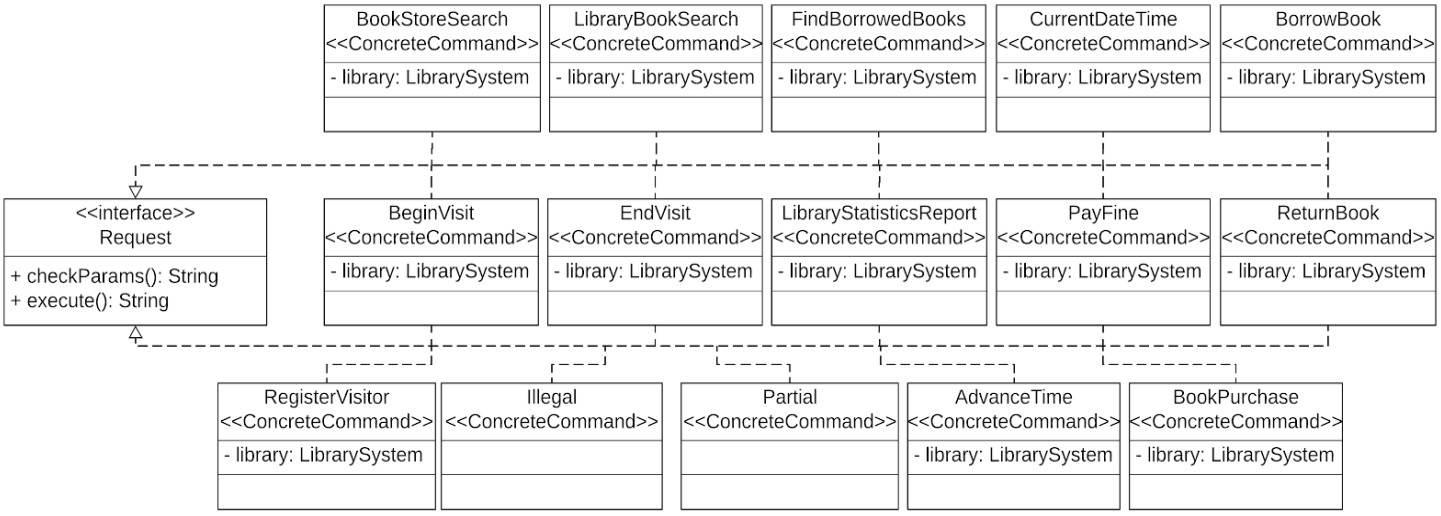


Figure 3. The request subsystem class diagram. Each request is a concrete command. There are also some requests for error checking that do not have a dependency to the library. (source)

The request subsystem consists of an implementation of the command pattern. Each visitor request, representing a concrete command, implements the request interface methods. A decision made for implementing the pattern was to have the requests execute the command by calling a method in the system class rather than introducing coupling directly between the requests and the database classes. The commands (Concrete Request classes) in our request subsystems are not "intelligent" since these classes do not know its implementation of how the requests are performed. The requests simply bind the receiver (LibrarySystem) and the actions needed to be carried to perform the request (methods in specific databases containing behavior needed to complete the specific command). This decision was made because we did not need to make the requests smart since they need to be executed as soon as they are created by the request parser. We did not require the commands to know how they need to be implemented since we never store them or move them across the system for later use.

Subsystem Name: Request		**GoF pattern:** Command
Participants		
Class	**Role in GoF pattern**	**Participant's contribution in the context of the application**
Request	Command Interface	The interface that provides the common implementation for all requests. Allows addition of new commands without creating issues in the system.
LBServer	Client	Allows the request parser to obtain the user's input from the command line by passing input as needed. The shutdown

		and restore functions of the system are done in this object.
RequestParser	Invoker	The parser creates commands through the input given. The parser determines what command is to be created and executed for a response.
LibrarySystem	Receiver	The library system is used to invoke actions that depend on if the library is open or closed. Other actions include reading and changing the time.
RegisterVisitor	Concrete Command	Create a new visitor given their personal information. The new visitor will have a unique ID allowing them to enter the library.
BeginVisit	Concrete Command	Record that a visitor has entered the library and allow them to perform checkouts and other functions.
EndVisit	Concrete Command	Record that a visitor has left the library and prevent any modifications to the system by that visitor.
BookStoreSearch	Concrete Command	Retrieve various information about books in the bookstore given the information the user is looking for and how to sort the search.
BookPurchase	Concrete Command	Purchase books to be added to the library given the IDs from the last bookstore search. A quantity for each book to purchase must be specified.
LibraryBookSearch	Concrete Command	Retrieve various information about books in the library given the information the user is looking for and how to sort the search.
BorrowBook	Concrete Command	Checkout a book from the library given the book's ID from a library book search, the ID of the visitor checking out the book, and other information.
FindBorrowedBooks	Concrete Command	For a given visitor, retrieve the books that are currently borrowed by that visitor and display them in a list with unique IDs.
ReturnBook	Concrete Command	Returns a book to the library given its ID from the find borrowed books request.
PayFine	Concrete Command	Pay the fine for overdue books under a visitor. The amount must be equal to or less than the total fine amount.
LibraryStatisticsReport	Concrete Command	Create the monthly report. The monthly report details the number of books owned, the number of visitors, average amount of

		time spent per visit, books purchased for the month, and the amount of money collected from fines.
AdvanceTime	Concrete Command	Changes the current date and time in the library, modifying all the relevant information along with it (overdue books, etc).
CurrentDateTime	Concrete Command	Gets the current time in the system and displays that time.
Partial	Concrete Command	A temporary command to be used for users that did not send a terminated request.
Illegal	Concrete Command	A command for invalid commands specified in the request. Used for telling the user about invalid input.
Deviations from the standard pattern: Commands are executed immediately when invoked. We made this decision because it's not necessary to hold commands for later use in the system. All visitor requests should be performed as soon as the visitor enters their request.		
Requirements being covered: The application must be able to take in user input that is in CSV format. In order to perform the right request, we have a parser which breaks down the request into a command and its parameters and creates the correct concrete command. This command is then immediately executed since it knows how to perform the operation by calling the appropriate method in the LibrarySystem controller layer.		

Library Subsystem

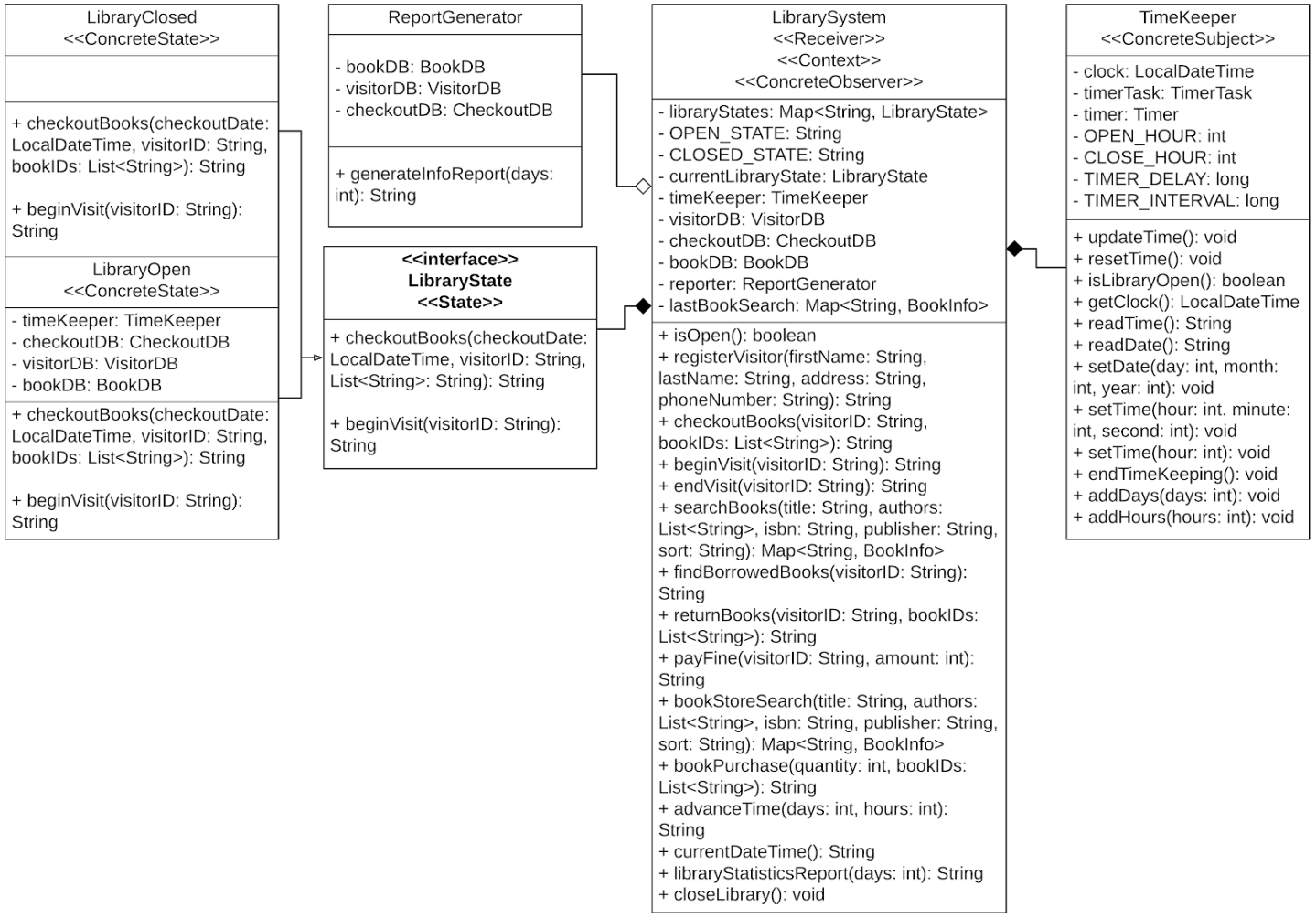
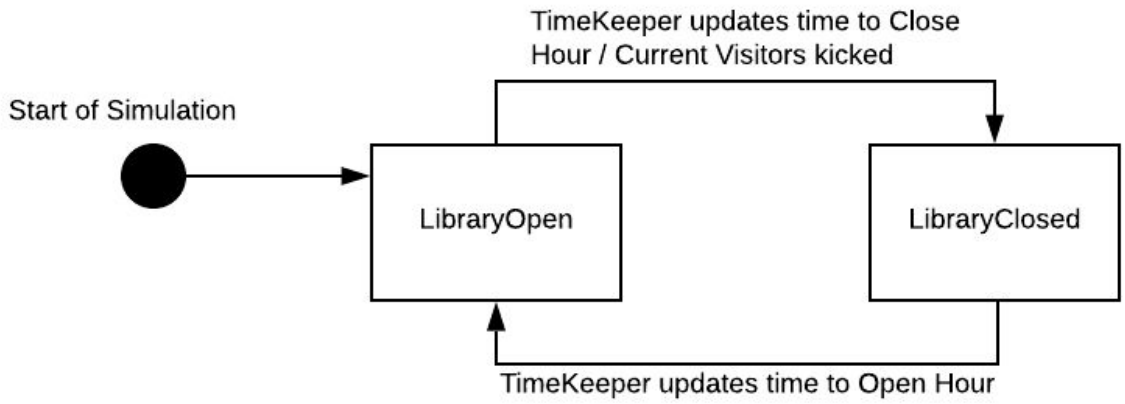


Figure 4. The Library subsystem class diagram. (source)

The Library subsystem controls the flow of information to and from the databases and clients. When a request is made of the library, all tasks are delegated through the LibrarySystem class to whichever databases would have the necessary information or logic to complete the request. The current date and time is managed by the TimeKeeper class and the state of the library (open/closed) is composed of a LibraryState class. Data reports can be requested through the LibrarySystem from the ReportGenerator. Our group decided to add a LibrarySystem class to serve as a control layer between requests and system databases (BookDB, VisitorDB, CheckoutDB). This decision led to less dependencies in our system as we avoided having visitor requests directly coupled with system databases required to complete a command. Instead requests only know about the LibrarySystem which can delegate request functionality to the necessary system databases.



Subsystem Name: Library		**GoF pattern:** State
Participants		
Class	**Role in GoF pattern**	**Participant's contribution in the context of the application**
LibrarySystem	Context	Defines the interface used by requests to interact with the library management system by providing behavior to perform all visitor commands. Maintains an instance of a Concrete LibrarySystem State Class that defines the current library state (Open or Closed).
LibraryState	State	Defines an interface for encapsulating the behavior associated with a Library state. Includes the methods that are shared by Library states (Open or Closed), allowing behavior to be called during all states of the library since concrete states are required to implement these methods.
LibraryClosed	ConcreteState	Defines the concrete behavior for a LibrarySystem in the closed state. This includes the behavior of checking out books and starting visits when the library is closed by returning an error string (Visitors are not allowed to check out or start a visit when the library is closed).
LibraryOpen	ConcreteState	Defines the concrete behavior for a LibrarySystem in the open state. This includes the behavior of checking out books and starting visits when the library is open by delegating request functionality to CheckoutDB, and VisitorDB.
Deviations from the standard pattern: The LibrarySystem (Context) will make its state transitions whenever notified that the library needs to close or open by the TimeKeeper.		

Requirements being covered: The BorrowBook and BeginVisit requests are not allowed to be processed when the library is closed. The LibrarySystem state pattern enforces this requirement by returning error strings to the user when a visitor tries to perform these requests and the library is closed.

Subsystem Name: Library		**GoF pattern:** Observer
Participants		
Class	**Role in GoF pattern**	**Participant's contribution in the context of the application**
LibrarySystem	ConcreteObserver	Defines the interface used by requests to interact with the library management system by providing behavior to perform all visitor commands. Contains a reference to the TimeKeeper (ConcreteSubject) object that will notify the LibrarySystem when the library needs to open and close.
TimeKeeper	ConcreteSubject	Stores the LibrarySystem time which is of interest to the LibrarySystem. Notifies the LibrarySystem when it is time to close or open.
Deviations from the standard pattern: Our design did not include Observer and Subject interfaces since we only had the need for one concrete observer (LibrarySystem) and one concrete subject (TimeKeeper).		
Requirements being covered: The library must have different behavior for borrowing books and starting visits depending on whether the library is open or closed. The Observer pattern allows the library to change state between the close and open library states whenever the time is updated to the open or closed hour.		

Book Subsystem

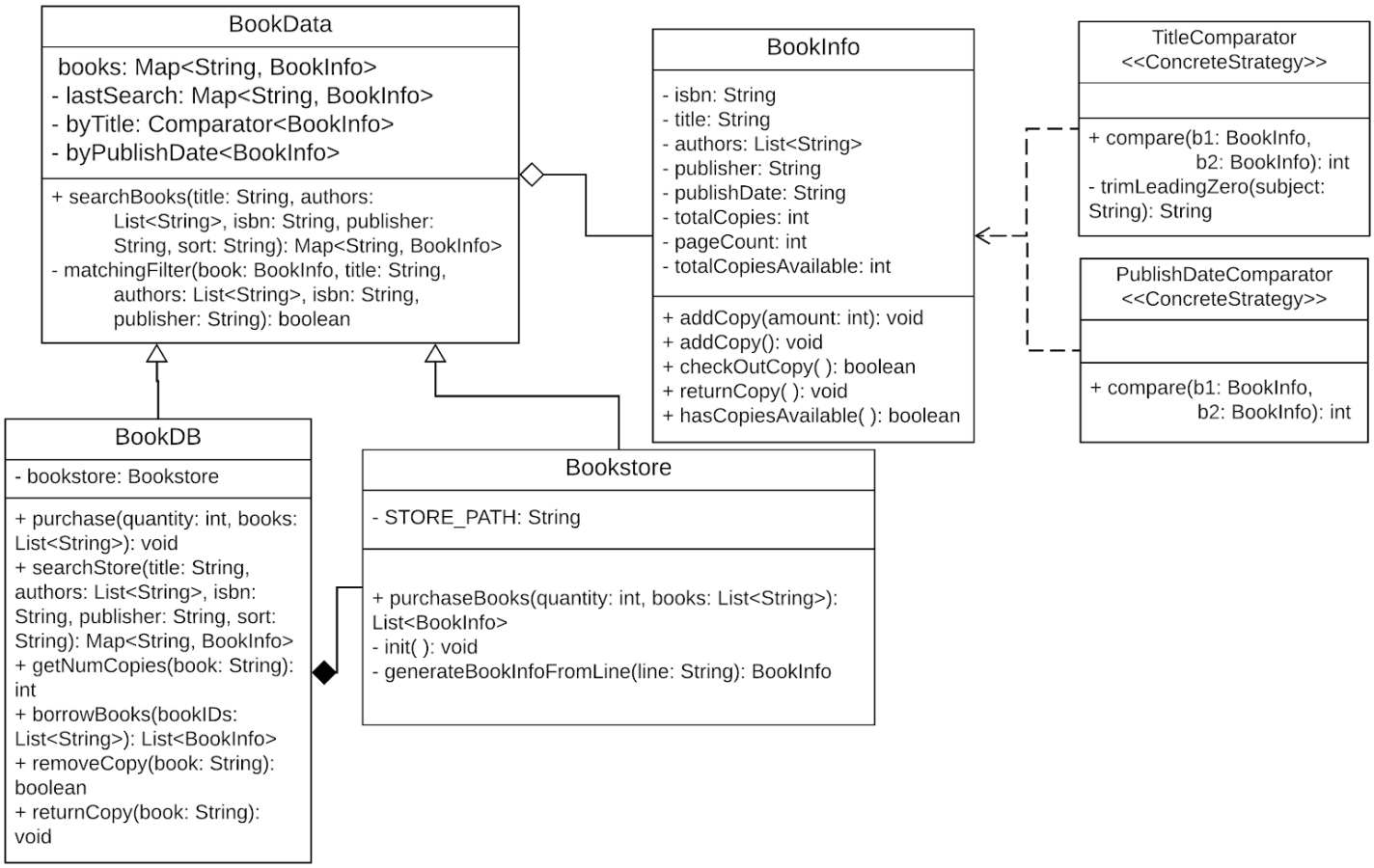


Figure 5. The book subsystem class diagram. (source)

The book subsystem provides a data structure to interact with books. In order to populate the library with books, books must be purchased from the bookstore that contains books from a text file. The library's books are kept in a book database that allows for removing and returning copies. Both the database and the bookstore can also be searched since they extend the book data structure abstract class. This data structure allows searching by certain books through filtering and then sorting the search.

Subsystem Name: Book		**GoF pattern:** Strategy
Participants		
Class	**Role in GoF pattern**	**Participant's contribution in the context of the application**
Comparator	Strategy	Serves to give all implementations a common method of being run. In this case, Comparator gives a way for books to be sorted, and the sorting methods will determine the strategy used.

PublishDate Comparator	ConcreteStrategy	Strategy to sort by publish date of the books.
TitleComparator	ConcreteStrategy	Strategy to sort by title of the book.
Deviations from the standard pattern: None		
Requirements being covered: The searching algorithm for both the bookstore and library book searches required us to be able to sort the contents that the search produced. Since it is up to the user to choose which sorting implementation to use, comparators were used and the sort method was called with the specified comparator.		

The strategy pattern is used in this subsystem for the ordering of books in searches by the comparator interface. The comparator interface is used when calling sort on a list and the one to use is decided by the user's request. The possible sorting algorithms are sorting by title or by publish date. Another sorting option is to filter out copies that do not exist which does not use a comparator.

Checkout Subsystem

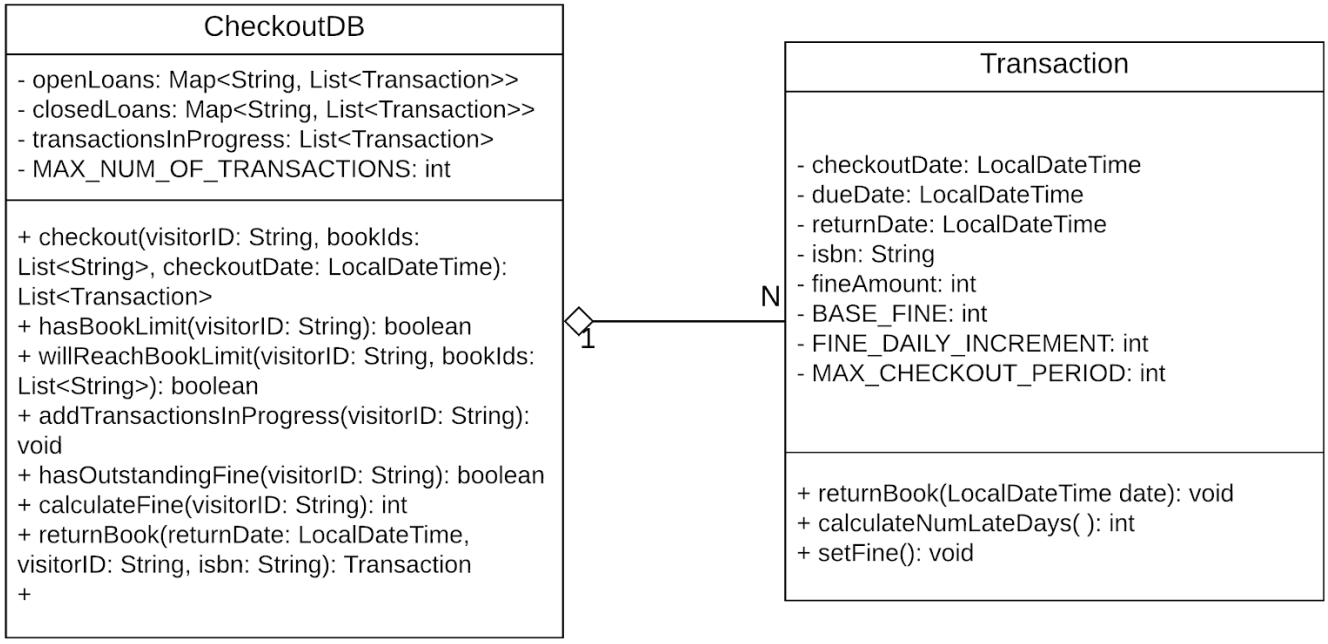
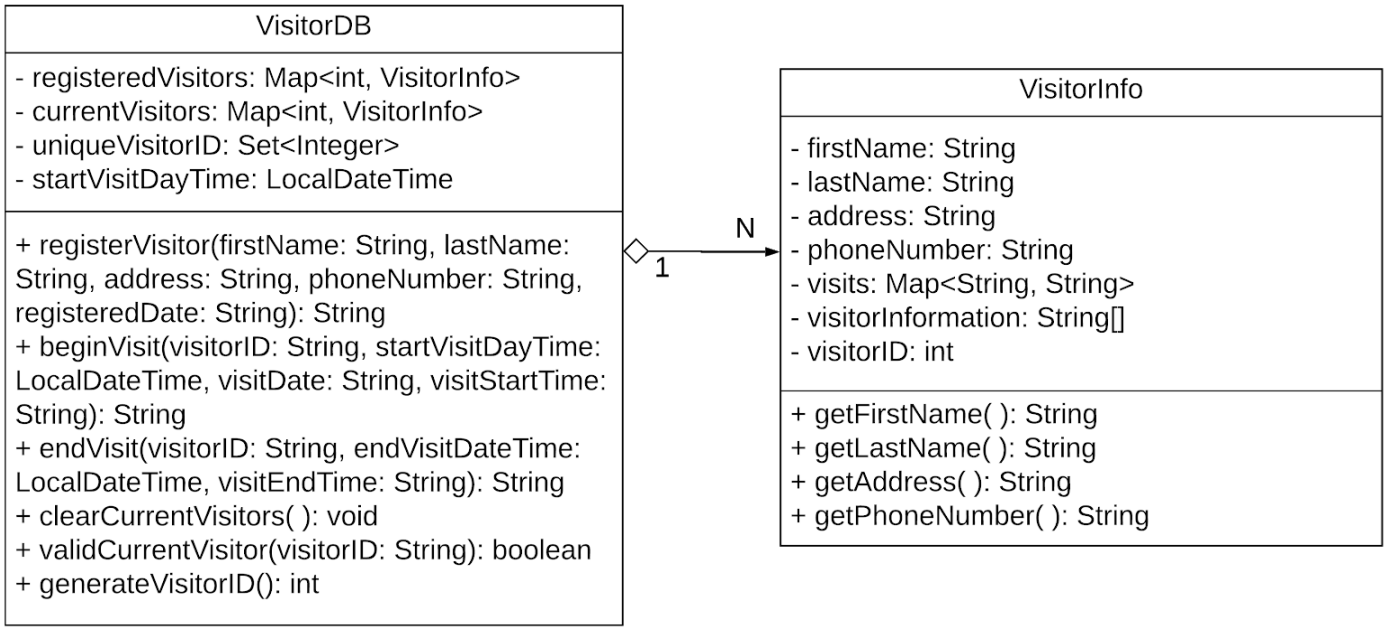


Figure 6. The Checkout subsystem class diagram. (source)

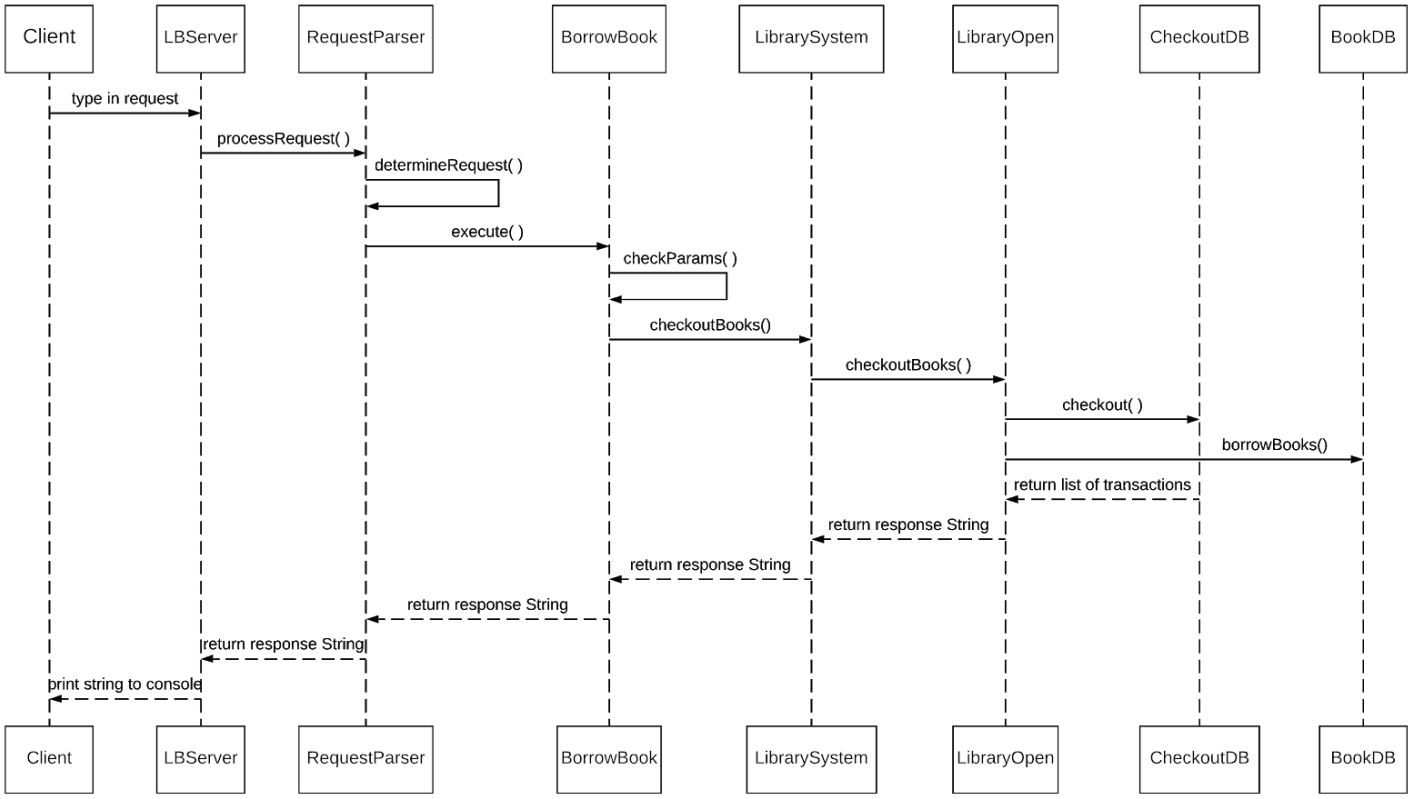
The checkout subsystem provides data structures to interact with visitor transactions in the library system. For example, visitor BorrowBook requests create transactions that are added to the open loans data structure in the checkout subsystem, and visitor ReturnBook requests remove transactions from the open loans collection in CheckoutDB and adds them to the closed loans collection for later use in library statistics reports. Apart from storing transactions, the Checkout subsystem also allows the library system to perform important commands including generating statistics reports, paying fines, returning books, borrowing books, and finding borrowed books. We decided to abstract this subsystem from a general system database to narrow the responsibility of this subsystem to only deal with storing and managing visitor transactions. Whenever a visitor request requires transaction information, the library subsystem will delegate the specific responsibility to behavior in this subsystem. For example, the visitor Checkout request is completed by the librarysubsystem calling the checkout method in CheckoutDB which will add a transaction per book being borrowed to the open loans collection in CheckoutDB.

Visitor Subsystem

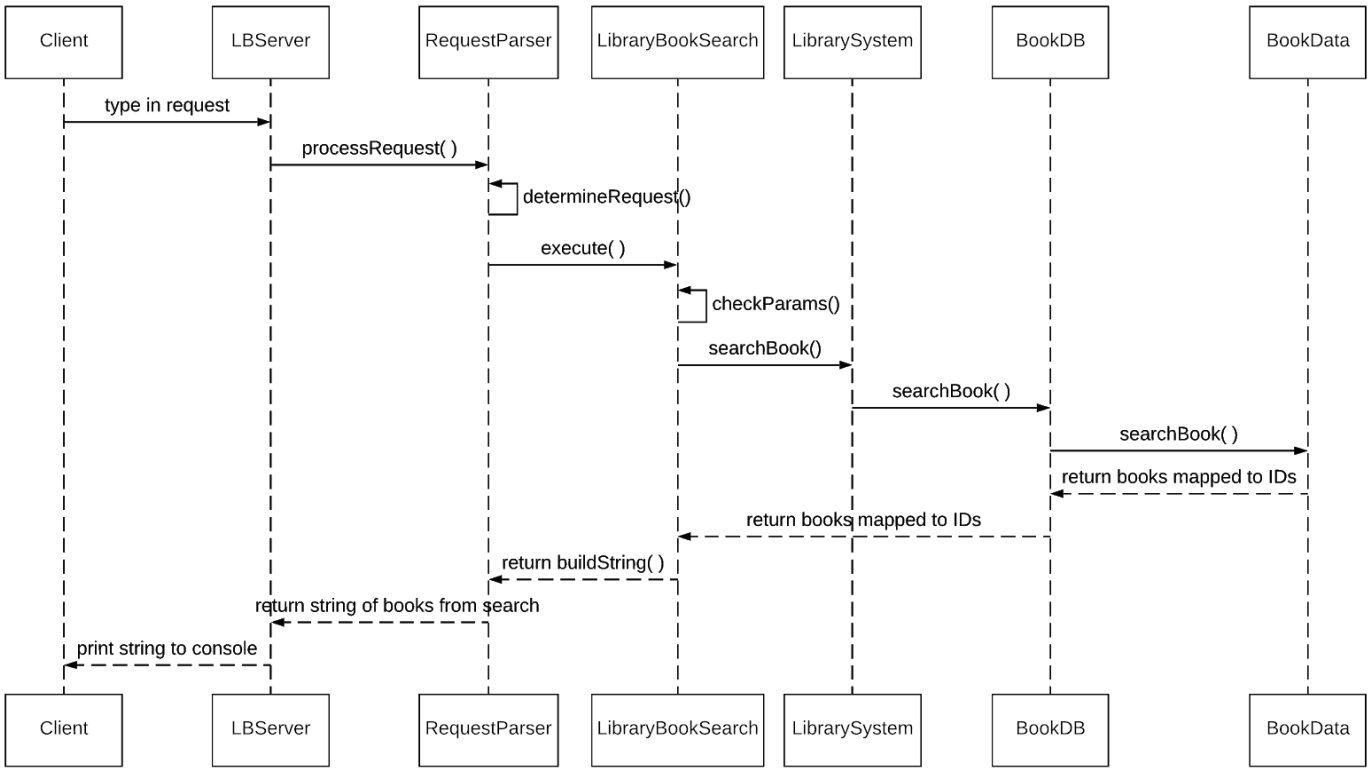


The visitor subsystem provides data structures to interact with current visitors, registered visitors, and individual visitors. The visitor subsystem deals with visitor requests regarding registering new visitors, storing visitors, starting visits, and ending visits. For example, a visitor Register request creates a new visitorInfo representing the visitor registering and adds this visitorInfo to the registered collection of visitors. The VisitorInfo objects represent visitors in our system by storing their first name, last name, address, and phoneNumber . We decided to abstract this subsystem from a general system database to narrow the responsibility of this subsystem to only deal with storing and managing visitor information. Apart from narrowing the focus of model classes, our design of breaking databases into VisitorDB, CheckoutDB, and BookDB also adheres to the principle of information expert. This is because behavior dealing with only visitor information will be placed in the VisitorDB, while behavior dealing with transactions will be placed in the CheckoutDB since these databases contain the required data to manipulate. Whenever a visitor request requires to modify or use visitor information, the library subsystem will delegate the specific responsibility to behavior in this subsystem.

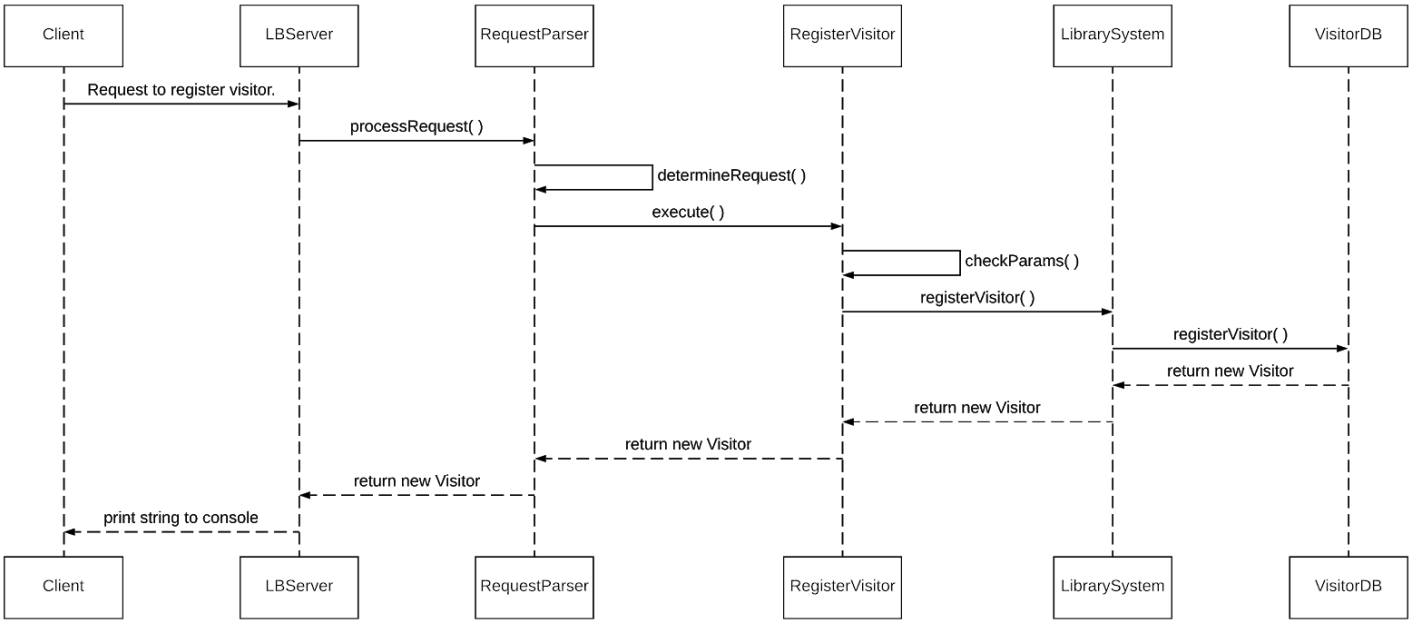
Sequence Diagrams



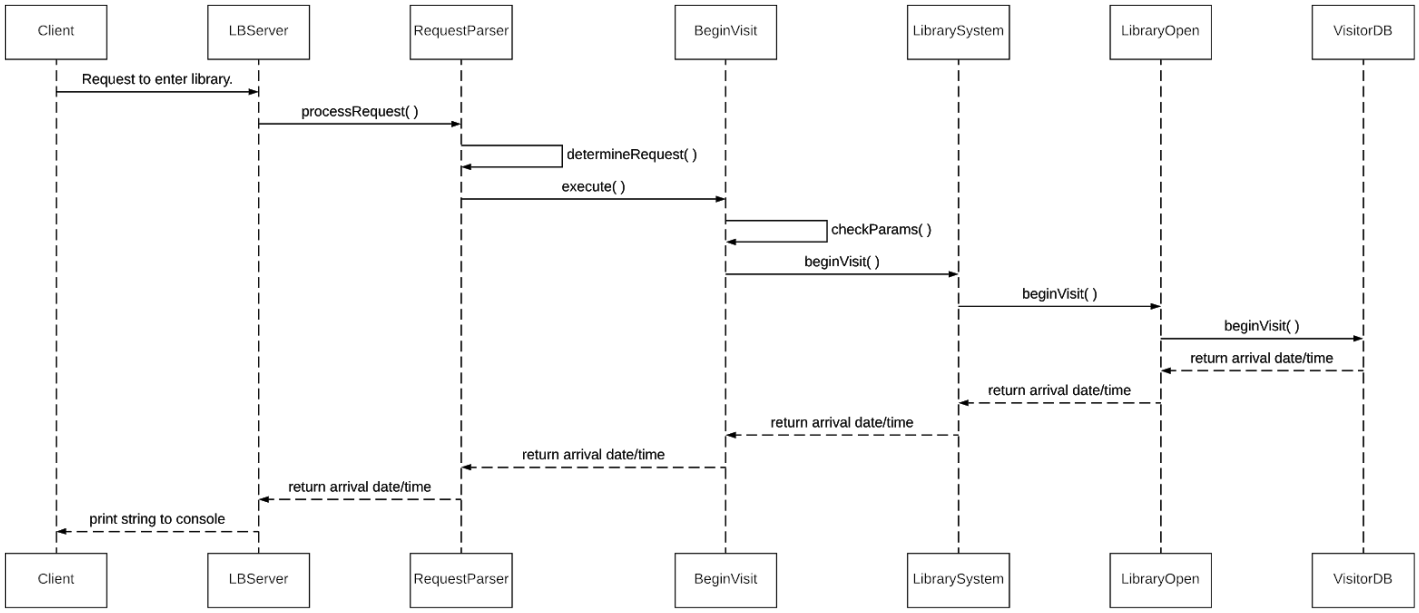
(Source) The above sequence diagram shows the relevant method calls through the system for the client to successfully borrow books from the system.



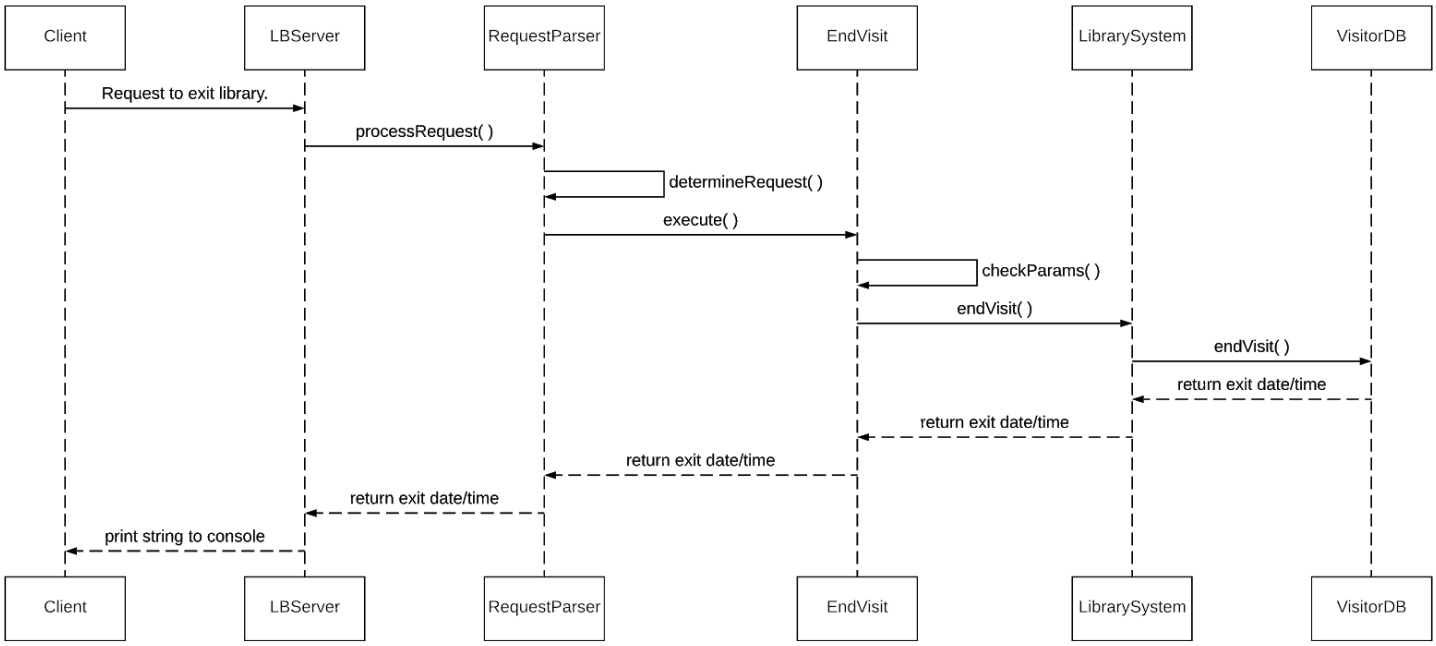
(Source) The above sequence diagram shows the relevant method calls through the system that are made when the client attempts to search for a book in the library.



(Source) The above sequence diagrams shows the relevant method calls made when a visitor first registers with the library.



(Source) The above diagram shows the relevant method calls made when a visitor arrives at the library.



(Source) The above diagram shows the relevant method calls made in the system when a visitor exits the library.

Status of the Implementation

The status of the implementation is as follows. We did not implement the functionality for library statistics report, but did stub out the design for it to be implemented in the next release. All other features are fully working with error responses. Documentation is fully written for class and method headers. At this time, there are no unit tests.

Appendix

Class: LBServer	
Responsibilities: Start the application by initializing the databases. Reads input to be passed to the request parser as needed. Save the state of the application to a file through serializing. Restore the state of the application from a file through deserializing.	
Collaborators:	
Uses: RequestParser, BookDB, VisitorDB, CheckoutDB, TimeKeeper, ReportGenerator, LibrarySystem	**Used by:** None
Author: Michael Kha	

Library Subsystem

Class: TimeUtil	
Responsibilities: Holds default behavior involved with manipulating and reading time. This behavior is shared by classes in the system that need to calculate time durations.	
Collaborators:	
Uses:	**Used by:** TimeKeeper, VisitorDB
Author: Michael Kha	

Class: TimeKeeper	
Responsibilities: Tracks time to be read by all classes that need date or time. Determine if the Library is open. Allows time to be changed.	
Collaborators:	
Uses: Timer, TimerTask, LocalDateTime	**Used by:** Library System
Author: Hersh Nagpal	

Class: LibraryState	
Responsibilities: Library State interface that allows the change in	

behavior regarding visitor requests depending on the state of the Library. Provides methods to concrete Library States (ClosedLibrary and OpenLibrary) that need to be implemented.	
Collaborators:	
Uses:	**Used by:** LibraryClosed, LibraryOpen
Author: Luis Gutierrez	

Class: LibraryClosed	
Responsibilities: Concrete state class which represents the library being closed. Implements behavior when the library is closed to deny visitor actions (Checking out books and starting visits are not allowed).	
Collaborators:	
Uses: LibraryState	**Used by:** Library System
Author: Hersh Nagpal	

Class: LibraryOpen	
Responsibilities: Concrete state class which represents the library being open. Allows visitors to visit, borrow books, and pay fines.	
Collaborators:	
Uses: LocalDateTime, CheckoutDB, VisitorDB, BookDB	**Used by:** Library System
Author: Hersh Nagpal	

Class: LibrarySystem	
Responsibilities: Receives commands and acts as a control layer between visitor commands and delegates functions to Database classes and other helper classes. Holds a Timekeeper.	
Collaborators:	
Uses: LibraryState, TimeKeeper,CheckoutDB,BookDB, VisitorDB	**Used by:** Request Parser, Visitor Request Classes
Author: Hersh Nagpal	

Class: ReportGenerator	

Responsibilities: Responsible for generating the system's monthly reports and for backing up the system's data for a restart. ReportGenerator will use the system databases to extract all necessary information needed to create the reports.	
Collaborators:	
Uses: BookDB, VisitorDB, CheckoutDB	**Used by:** RequestParser
Author: Michael Kha	

Checkout Subsystem

Class: Transaction	
Responsibilities: Record the date of every checkout transaction done by a Visitor and the due date for the book. Contains possible fines accumulated by being returned late.	
Collaborators:	
Uses: LocalDateTime, ChronoUnit	**Used by:** CheckoutDB
Author: Hersh Nagpal	

Class: CheckoutDB	
Responsibilities: Responsible for managing checkout and return transactions, as well as fines and fine payments performed by visitors. In order to manage these requests dealing with transactions, CheckoutDB holds collections of transactions that represent visitor active and closed loans.	
Collaborators:	
Uses: Transaction	**Used by:** Library System
Author: Hersh Nagpal	

Book Subsystem

Class: BookData	
Responsibilities: Abstract class to provide functionality for storing and searching for books. Keeps track of the last search to be referenced by each item's ID.	
Collaborators:	
Uses: BookInfo, PublishDataComparator,	**Used by:** BookDB, Bookstore

TitleComparator, CopiesComparator	
Author: Michael Kha	

Class: BookDB	
Responsibilities: Stores books that have been purchased. Allows books to be checked out and returned. Can add books by first searching the bookstore and then purchasing.	
Collaborators:	
Uses: BookData, Bookstore, BookInfo	**Used by:** DBManager
Author: Michael Kha	

Class: Bookstore	
Responsibilities: Stores all possible books to purchase from a data file. Allow for the database to purchase books.	
Collaborators:	
Uses: BookData, BookInfo	**Used by:** BookDB
Author: Michael Kha	

Class: BookInfo	
Responsibilities: Represents a book by storing its information. Tracks the total number of copies and available copies.	
Collaborators:	
Uses: None	**Used by:** BookData, BookDB, Bookstore
Author: Michael Kha	

Class: TitleComparator	
Responsibilities: Used to sort books by title by implementing the comparator interface. This is a sorting strategy.	
Collaborators:	
Uses: BookInfo	**Used by:** BookData
Author: Michael Kha	

Class: PublishDateComparator	
Responsibilities: Used to sort books by publish date by implementing the comparator interface. This is a sorting strategy.	
Collaborators:	
Uses: BookInfo	**Used by:** BookData
Author: Michael Kha	

Request Subsystem

Class: RequestUtil	
Responsibilities: Holds string constants that are used as a request and a response. Simplifies string building for responses.	
Collaborators:	
Uses:	**Used by:** Request, VisitorDB, BookDB, CheckoutDB
Author: Michael Kha	

Class: RequestParser	
Responsibilities: Reads requests strings and parses those requests into a single concrete command and its parameters to be executed. Returns the response of the executed command back to the LBServer. A request may possibly be a partial request, where the parser continues to read input until it is terminated by a semicolon.	
Collaborators:	
Uses: ALL REQUESTS	**Used by:** LBServer
Author: Michael Kha	

Class: Request	
Responsibilities: An interface defining methods to be used by concrete requests. The included methods are checking parameters and execute the request.	
Collaborators:	
Uses: RequestUtil	**Used by: ALL REQUESTS**
Author: Michael Kha	

Class: AdvanceTime	
Responsibilities: Moves the system time forward by some given number of hours and days. Does not support increments of weeks, months, or other measures of time.	
Collaborators:	
Uses: Request, LibrarySystem	**Used by:** RequestParser
Author: Jack Li	

Class: BeginVisit	
Responsibilities: Concrete visitor request that implements the checking parameters and execute methods in the Request interface. Allows client arrive request to be executed and processed by the library system and returns a formatted string regarding the success of the request to notify visitor of his request status.	
Collaborators:	
Uses: Request, LibrarySystem	**Used by:** RequestParser
Author: Luis Gutierrez	

Class: BookPurchase	
Responsibilities: Purchase books from the most recent book store search by referring to the IDs from the search. Returns a string of all the books purchased and properly formatted.	
Collaborators:	
Uses: Request, LibrarySystem	**Used by:** RequestParser
Author: Michael Kha	

Class: BookStoreSearch	
Responsibilities: Checks to see if the parameters are correct and if so, executes the search store call on the book database. Interprets the mapping of books from the search into a string.	
Collaborators:	
Uses: Request, LibrarySystem	**Used by:** RequestParser
Author: Michael Kha	

Class: BorrowBook	

Responsibilities: Borrows a book and updates the visitor to have the book through a transaction. Borrowing a book may or may not be successful depending on the visitor's condition.	
Collaborators:	
Uses: Request, LibrarySystem	**Used by:** RequestParser
Author: Michael Kha	

Class: CurrentDateTime	
Responsibilities: Outputs a string containing the current date and time in the Library system. Calls the method in the library system.	
Collaborators:	
Uses: Request, LibrarySystem	**Used by:** RequestParser
Author: Jack Li	

Class: EndVisit	
Responsibilities: Ends the visit for the given visitor. This request calls the library system to end the visit.	
Collaborators:	
Uses: Request, LibrarySystem	**Used by:** RequestParser
Author: Michael Kha	

Class: FindBorrowedBooks	
Responsibilities: Find the books that a visitor is currently borrowing. This request uses behavior in library system to access the open loans in the CheckoutDB.	
Collaborators:	
Uses: Request, LibrarySystem	**Used by:** RequestParser
Author: Luis Gutierrez	

Class: Illegal	
Responsibilities: Created when a command read by the request parser could not be identified. When executed, returns a string to tell the user the request had a bad command.	

Collaborators:	
Uses: Request	**Used by:** RequestParser
Author: Michael Kha	

Class: LibraryBookSearch	
Responsibilities: Checks if the parameters are correct and if so, executes the search library call on the book database. Interprets the mapping of books from the search into a string.	
Collaborators:	
Uses: Request, LibrarySystem	**Used by:** RequestParser
Author: Michael Kha	

Class: LibraryStatisticsReport	
Responsibilities: Create the report for a given number of days or from the start of the simulation. The report contains statistics about visitors, books, and checkout transactions.	
Collaborators:	
Uses: Request, LibrarySystem	**Used by:** RequestParser
Author: Michael Kha	

Class: Partial	
Responsibilities: Created when a command read by the request parser did not terminate yet. When executed, returns a string to tell the user the request was only a partial request and to continue providing the rest of the request.	
Collaborators:	
Uses: Request	**Used by:** RequestParser
Author: Michael Kha	

Class: PayFine	
Responsibilities: Pay the fines for a visitor. The payment amount must be within the visitor’s balance.	
Collaborators:	
Uses: Request, LibrarySystem	**Used by:** RequestParser

Author: Michael Kha	

Class: RegisterVisitor	
Responsibilities: Register the visitor given the personal information. The resulting response indicates the visitors new ID and the date of registration.	
Collaborators:	
Uses: Request, LibrarySystem	**Used by:** RequestParser
Author: Michael Kha	

Class: ReturnBook	
Responsibilities: Returns the books for a given visitor from the most recent find borrowed books search. The books may possibly be overdue and the response will indicate the fine amount.	
Collaborators:	
Uses: Request, LibrarySystem	**Used by:** RequestParser
Author: Michael Kha	