<Library Book Management System>

Design Documentation  
Prepared by Team E

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

Summary 2

Domain Model 3

System Architecture 4

Subsystems 5

Book Check-in Fine Calculator 5

Book Check Out State 7

Status of the Implementation 9

Appendix 10

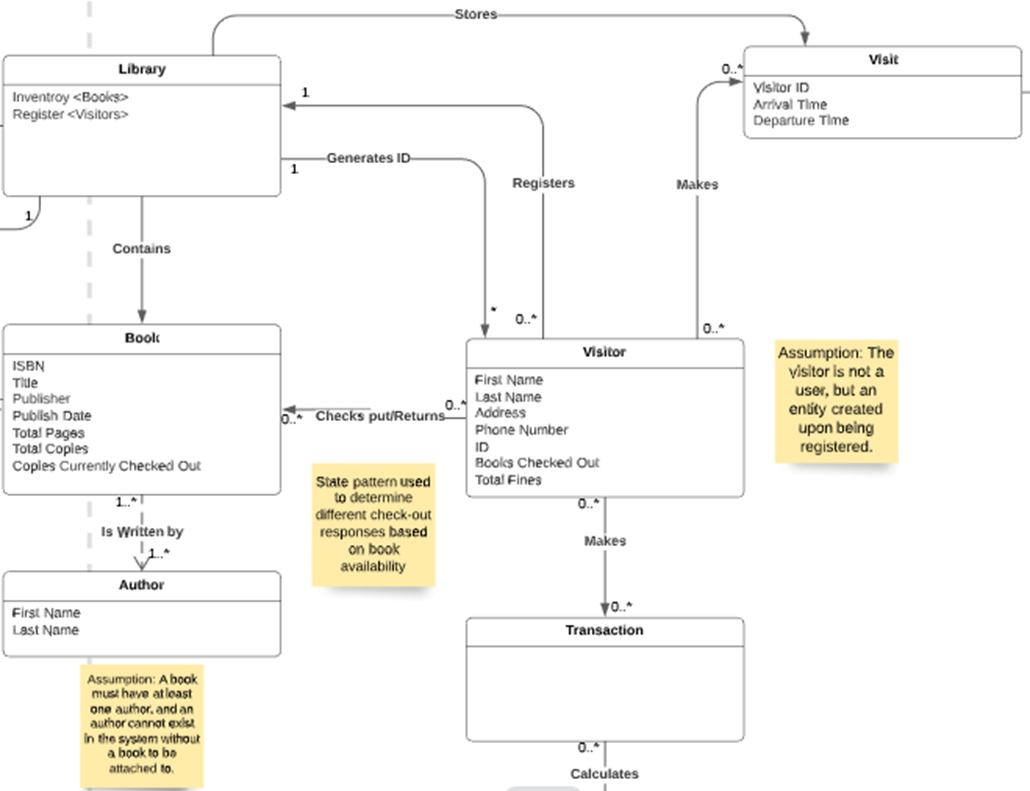
# Summary

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 15 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 on the command line with the intention to further develop a GUI, providing the user with a more pleasant experience. Planning for this future development, the MVC pattern was used to help organize files and directories.

# 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 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.

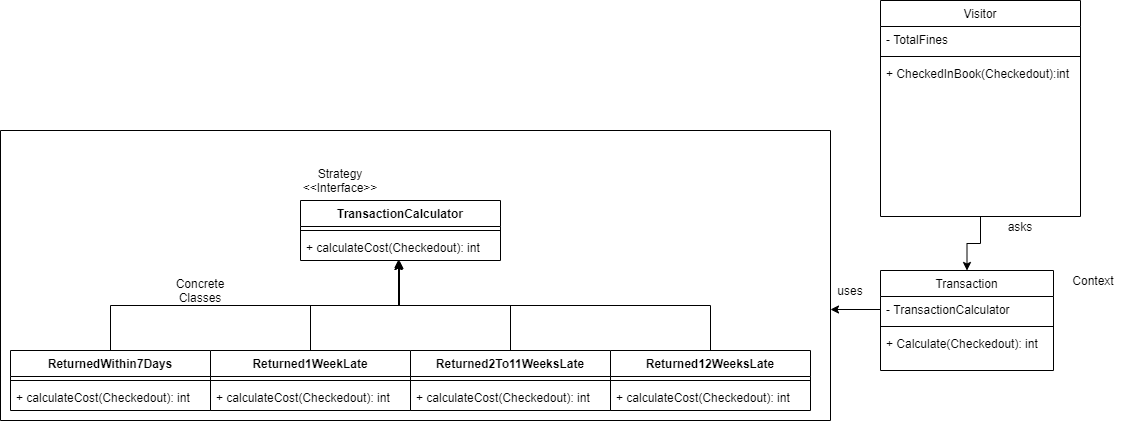
# System Architecture

This section provides a model of the subsystem components that make up the overall software architecture for the project. Draw the subsystems as simple boxes with relationships between them. Provide a narrative that describes the responsibilities of each component and the interfaces that are provided between subsystems.

# Subsystems

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

## Book Check-in Fine Calculator

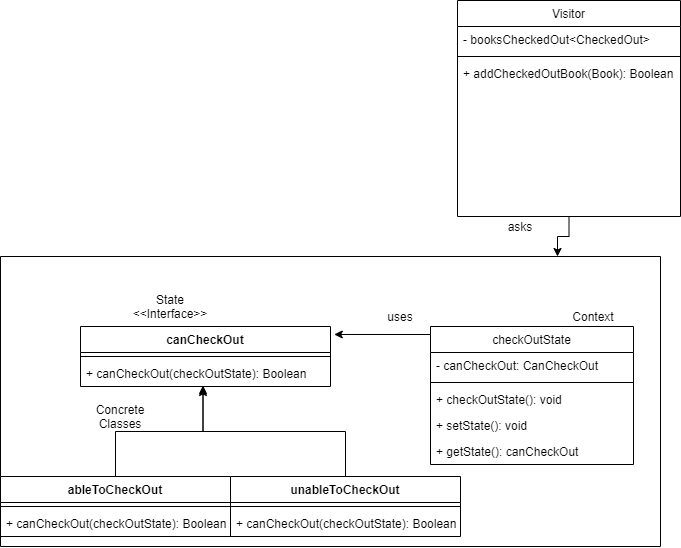


* Sequence diagrams with associated narratives that describe the dynamic behaviors that are primarily located within this subsystem. Within your subsystem design descriptions, you must make sure to provide sequence diagrams for all features listed in the design project problem statement. You may also decide that other features require documentation within the subsystems.

|  |  |  |  |
| --- | --- | --- | --- |
| **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



* Sequence diagrams with associated narratives that describe the dynamic behaviors that are primarily located within this subsystem. Within your subsystem design descriptions, you must make sure to provide sequence diagrams for all features listed in the design project problem statement. You may also decide that other features require documentation within the subsystems.

|  |  |  |  |
| --- | --- | --- | --- |
| **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.** | | | |

# Status of the Implementation

Provide a complete description of the status of your implementation. This should specify all known defects in the system, and indicate requirements that your implementation does not cover.

# 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:** MyClass1 |  |
| **Responsibilities:** ... |  |
| **Collaborators:** ... |  |
| **Users:** ... | **Used by:** ... |
| **Author:** ... |  |