ITC205/515 Assignment 3 Design materials

Assignment 3 revolves around developing support for a self-service Library system. The main aim of the new system is to allow borrowers to self-support in terms of finding and borrowing books. The system should also help staff to manage books.

It has been determined that the most important functionality required by the new system is the ability to borrow a book. Consequently, initial analysis and design activity have centred on developing a specification for the use case **'Borrow a Book'**.

What follows are some documents detailing the requirements for the 'Borrow a Book' use case.

These documents include:

1. A use case diagram for the library system
2. A domain model for the library system.
3. A full use case description for the 'Borrow Book' use case.
4. An activity diagram equivalent to the full use case.
5. A design class diagram for the entity and data access classes necessary to support the use case.
6. A set of 3 state diagrams, one for each entity class.
7. A sequence diagram showing a sequence of messages which can support the use case.
8. A software requirement specification document detailing the requirements for those entity and data access classes
9. A set of system operation specifications detailing the requirements for the system operations required to support the use case.

Please Note: in the event of any conflict between a diagram and the textual specifications, the textual specifications take precedence.

Assignment Guidelines

The assignment centres on a hierarchy of dynamic testing, starting with low level unit testing, low level integration testing, higher level integration testing, 'system' testing and eventually user acceptance testing.

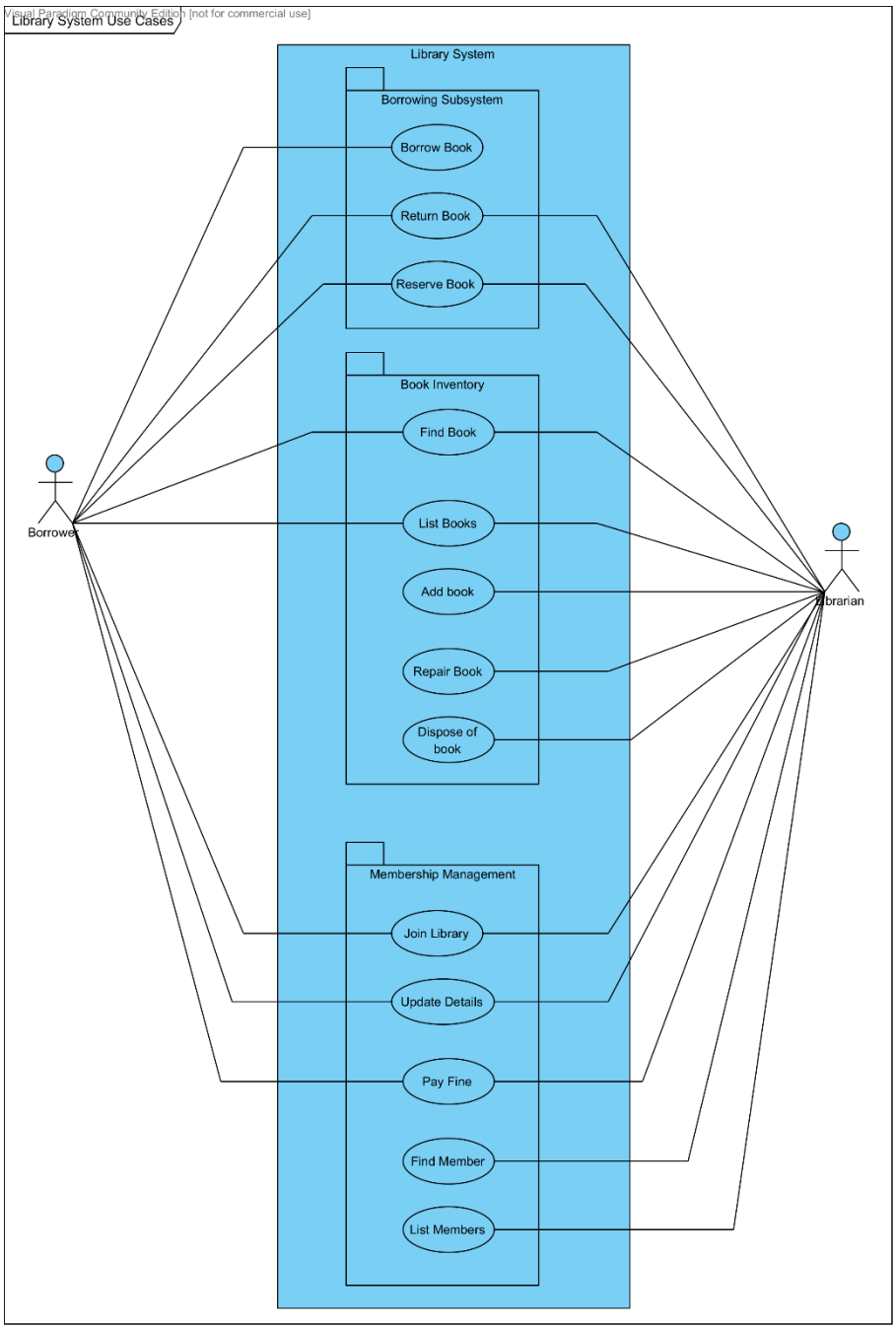
Carry out the following testing steps:

1. Implement and unit test any of the Member, Loan, or Book objects.*
2. Implement and unit test the associated DAO and helper obejcts.*
3. Carry out low level integration testing of the Entity, DAO, and helper object interaction.
4. Carry out high level integration testing by implementing and testing the 'swipe card', 'scan book' and 'confirm loan' system operations.**
5. Carry out 'system' level testing by simulating use case scenarios to drive the 'borrow book' control class.
6. Carry out user acceptance testing by integrating the supplied UI classes, and designing and carrying out UAT using the supplied UAT templates.

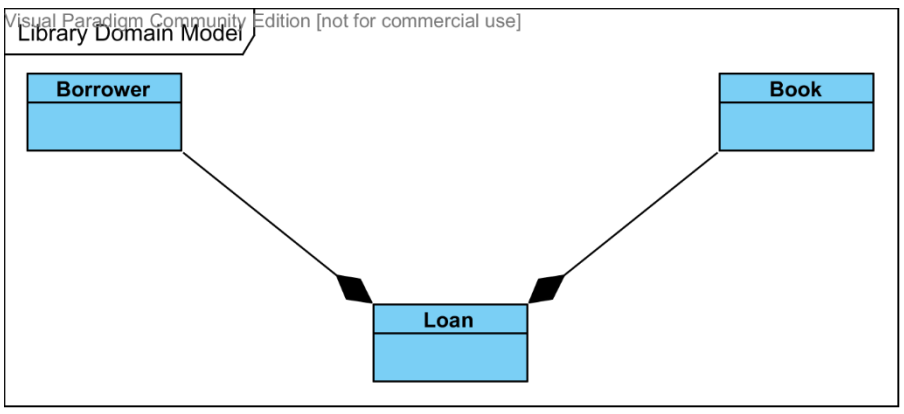
*Use mock objects to isolate the object subjected to unit tests.

**You may contact the lecturer to provide implementations of those sections of the application you have not implemented and tested in steps 1, 2, and 3.

Library System Use Case Diagram



Library System Domain Model



Full Use Case Description: Borrow Book

1 Brief Description

A borrower swipes their library card. If there are no borrowing restrictions they then scan the barcode of each book they wish to borrow. When they have scanning all the books they wish to borrow, a loan slip is printed.

2 Actor Brief Descriptions

2.1 Borrower: A member of the library who wishes to borrow a book.

3 Preconditions

3.1 The borrower must be a member of the library with a valid membership card.

3.2 All books selected must be shelved in the public area, and have valid book barcodes identifying them to the library system.

4 Normal Flow of Events

Step	Actor	System
1	1.1 The use case begins when the borrower initiates a loan at the self-service station.	1.2 The system prompts the user to swipe their library card
2	2.1 The borrower swipes their library card through the library card reader.	2.2 The system retrieves the borrower's details. 2.3 The system displays the borrower's details. 2.4 The system enables the scanner
3	3.1 The borrower scans the book barcode using the book barcode scanner	3.2 The system retrieves the book's details. 3.3 The system displays the book's details. 3.4 The system creates a new pending loan for the book 3.5 The system adds the new pending loan to the pending loan list. 3.6 The system displays the current pending loan list.
4	4.1 The borrower indicates they have finished scanning books.	4.2 The system disables the scanner. 4.3 The system displays the finalized pending loan list.
5	5.1 The borrower confirms the finalized pending loan list.	5.2 The system records the confirmed loans. 5.3 The system prints a borrowing slip. 5.4 The use case terminates.

5 Alternate Flows

1.1 Borrowing Restrictions Apply

If at step 2.2 of the basic flow the system finds that the borrower has borrowing restrictions due to overdue loans, having borrowed the maximum number of books permitted, or exceeding the maximum amount of fines owing, then:

Step	Actor	System
2.2		2.2.1 The system displays a borrowing restriction message. 2.2.2 The system disables the book scanner
	2.2.3 The user cancels the book borrowing process.	

The use case terminates.

1.2 Borrowing Multiple Books

If at step 4.1 the borrower indicates that they want to scan more books:

Step	Actor	System
4.1	4.1.1 The borrower indicates they wish to scan more books.	4.1.2 The system leaves the scanner enabled

The use case repeats from the beginning of step 3.

1.3 Loan Limit Reached

If at step 4.1.1 the borrower indicates that they want to scan more books, but borrowing an additional book would cause them to exceed the loan limit:

Step	Actor	System
4.1.1		4.1.2.1 The system displays a message telling the user they have reached the loan limit.

The normal flow is resumed at step 4.2.

1.4 Finalized Loan List Rejected

If at step 5.1 the borrower changes their mind about what books to borrow:

Step	Actor	System
5.1	5.1.1 The borrower rejects the finalized loan list.	5.1.2 The system clears the current pending loan list.

The normal flow is resumed at step 3.

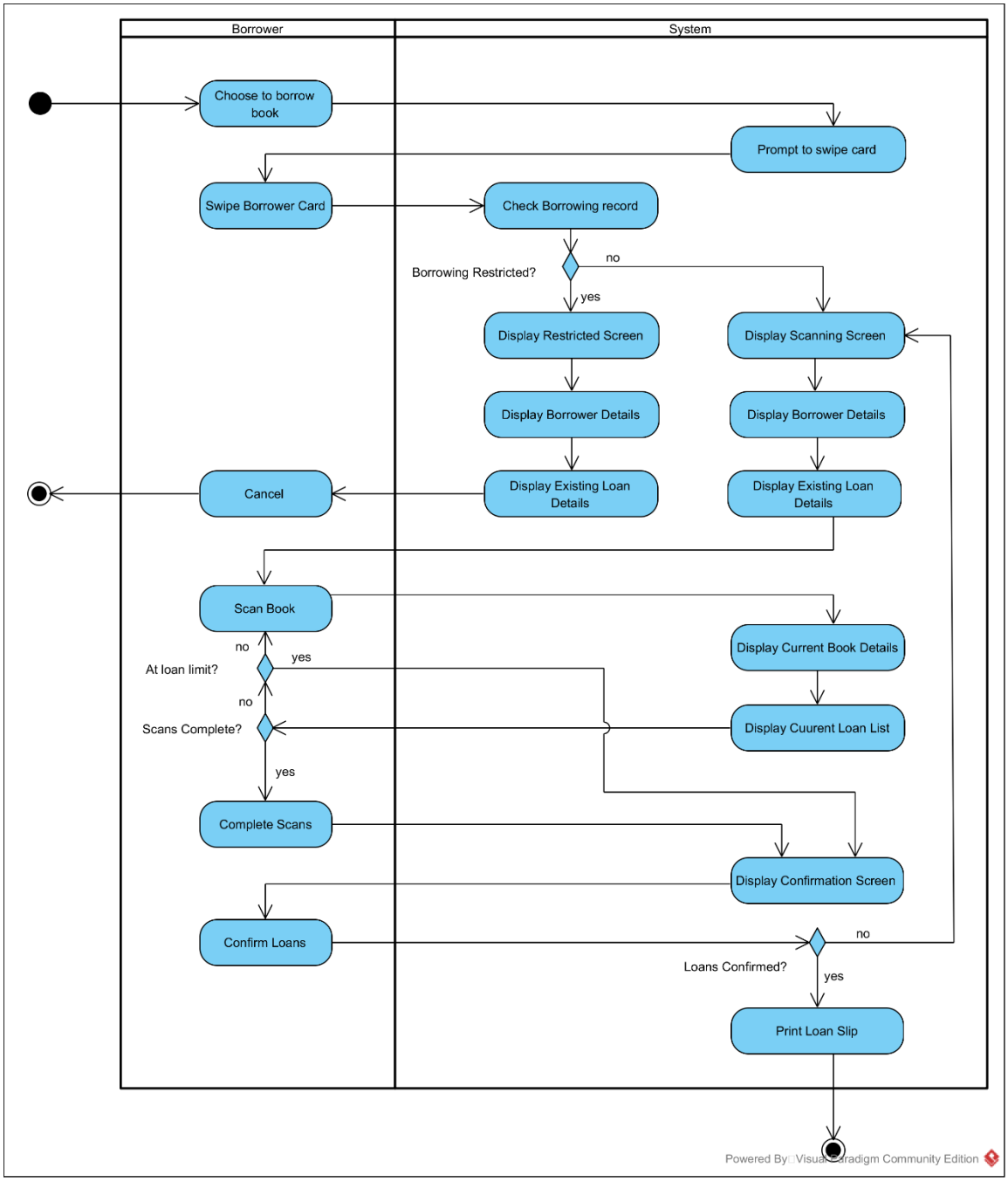
1.5 Borrowing Cancelled

If at any step, up until step 5.1, the user decides not to borrow any books:

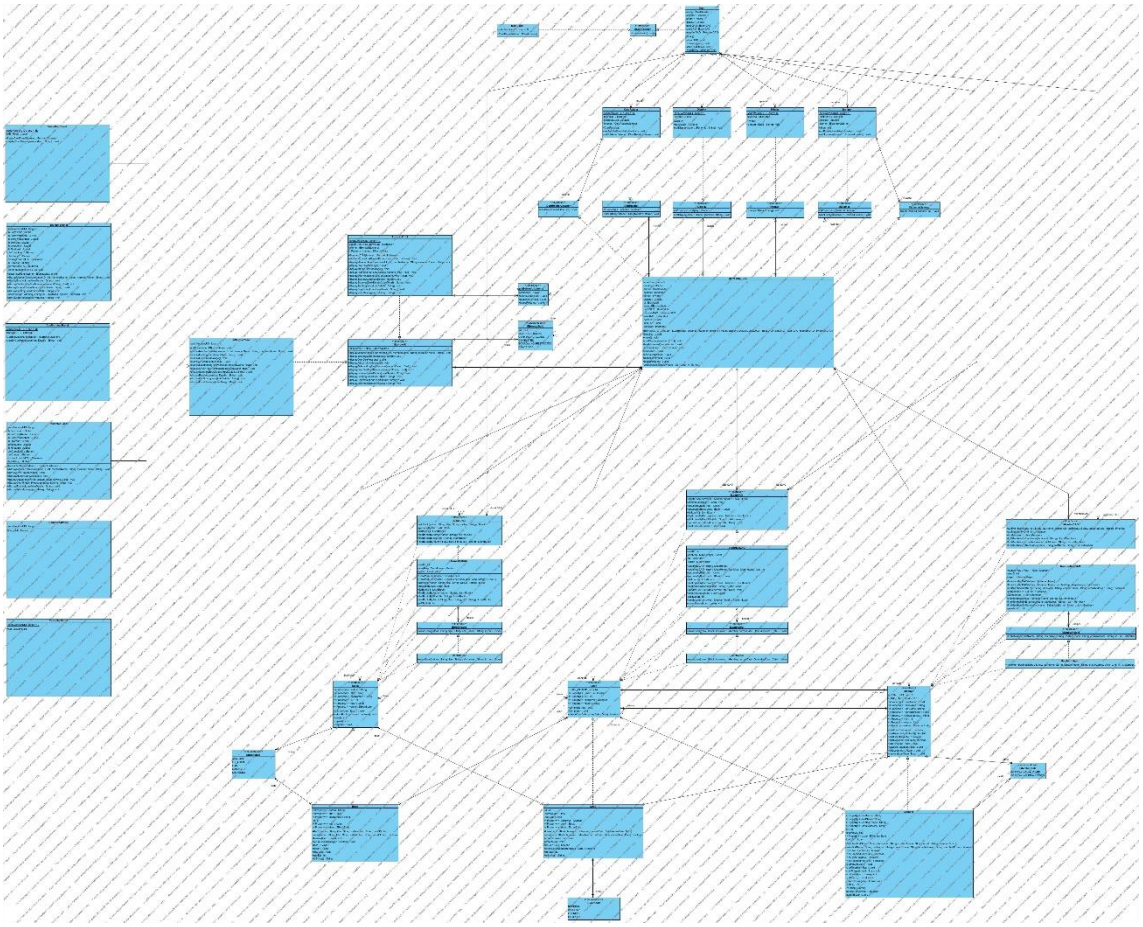
Step	Actor	System
X	X.1 The user cancels the book borrowing process.	X.2 The system clears the pending loan list

The use case terminates

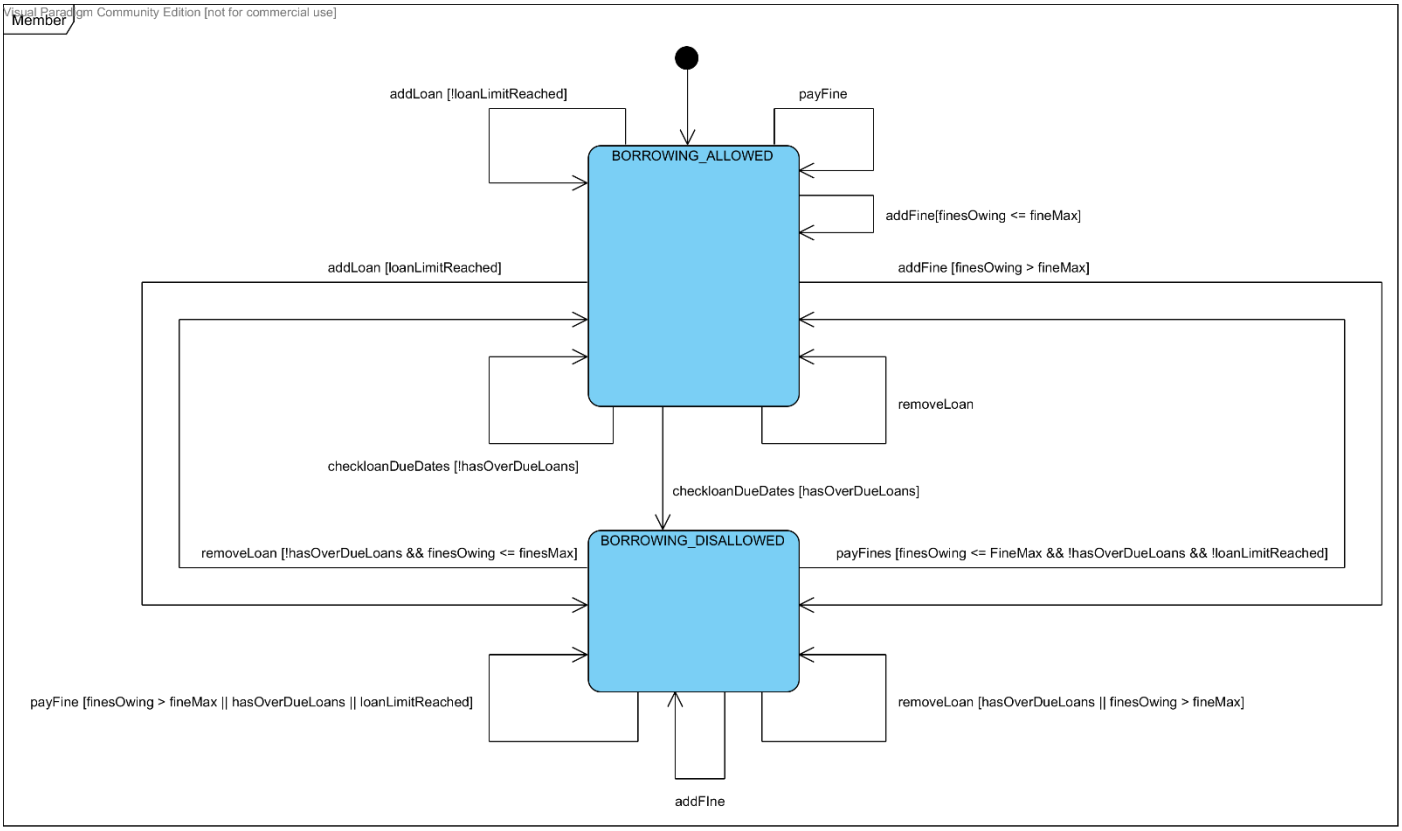
Activity Diagram: Borrow Book



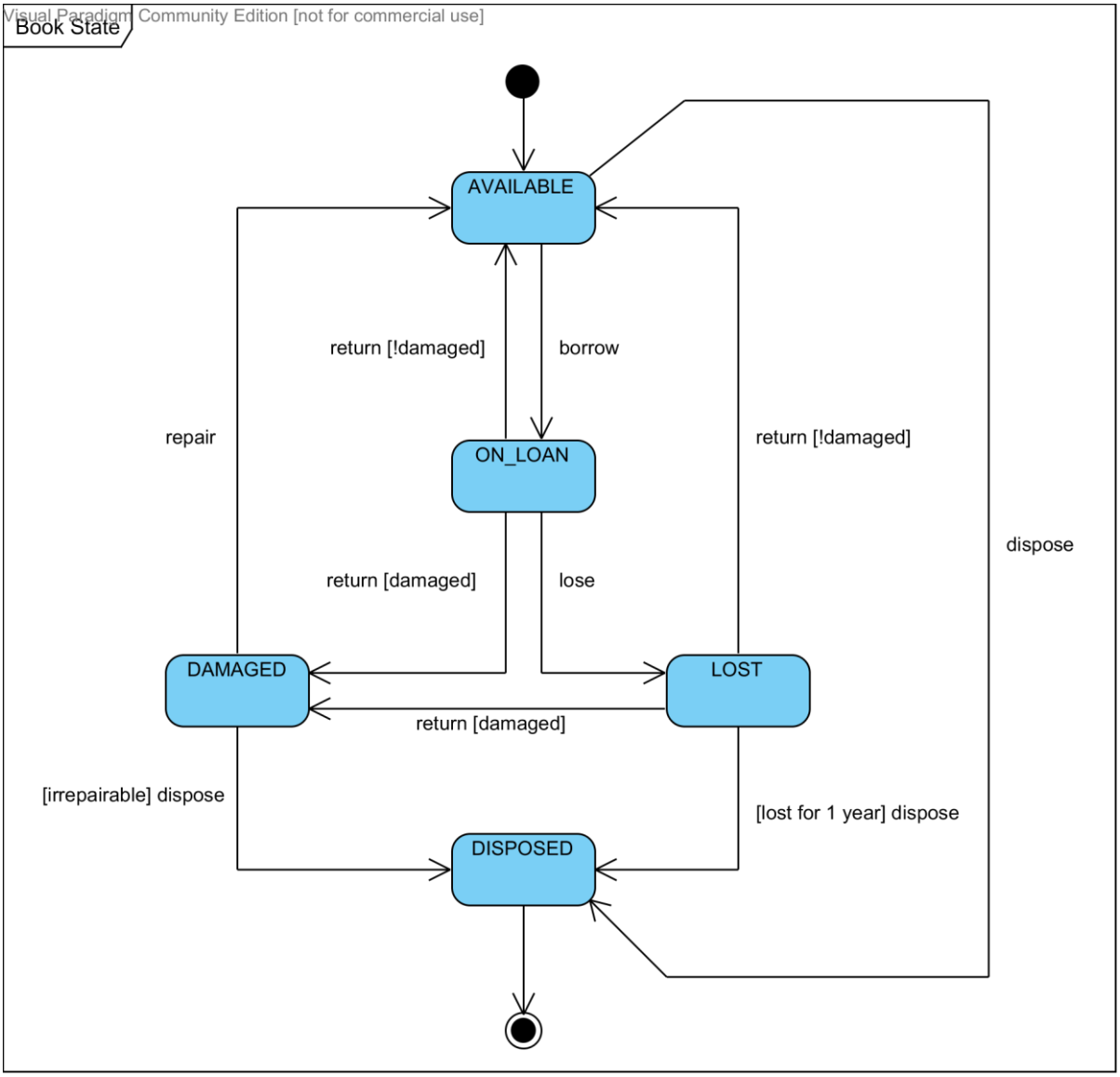
Design Class Diagram: Borrow Book Collaboration



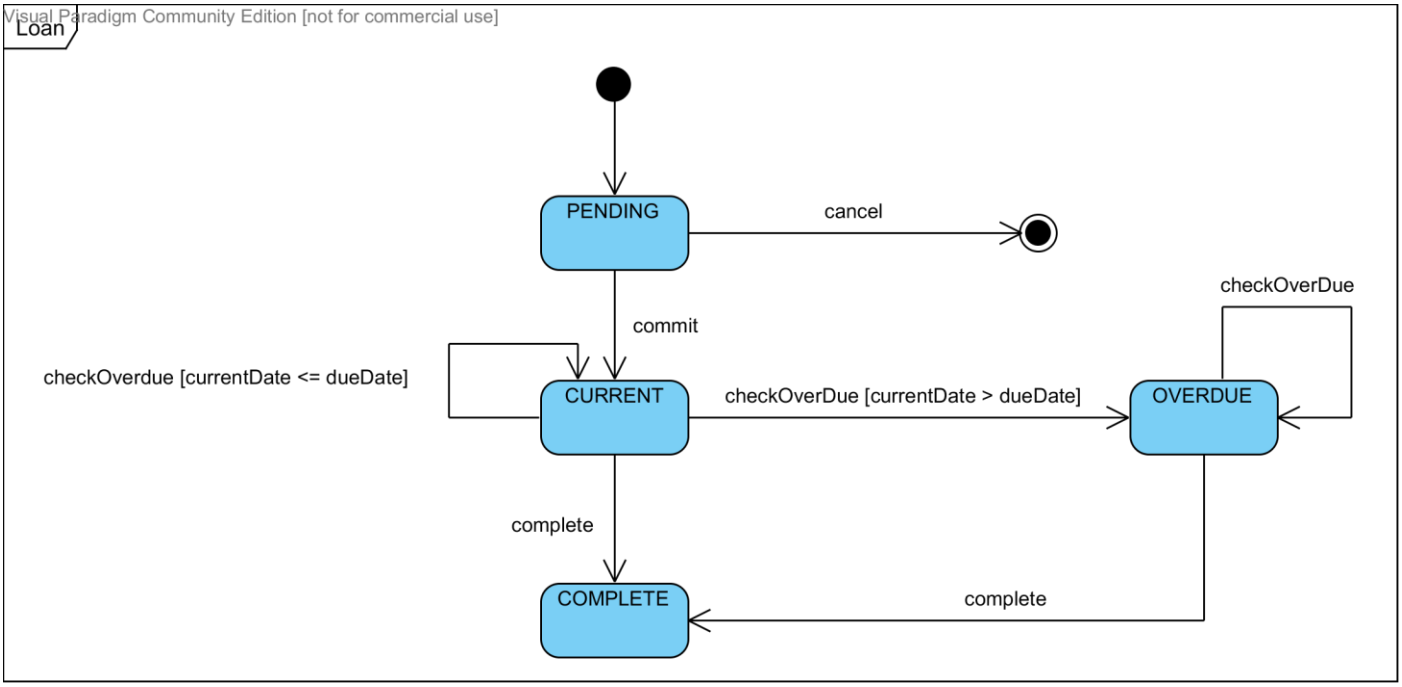
State Diagram: Borrower



State Diagram: Book

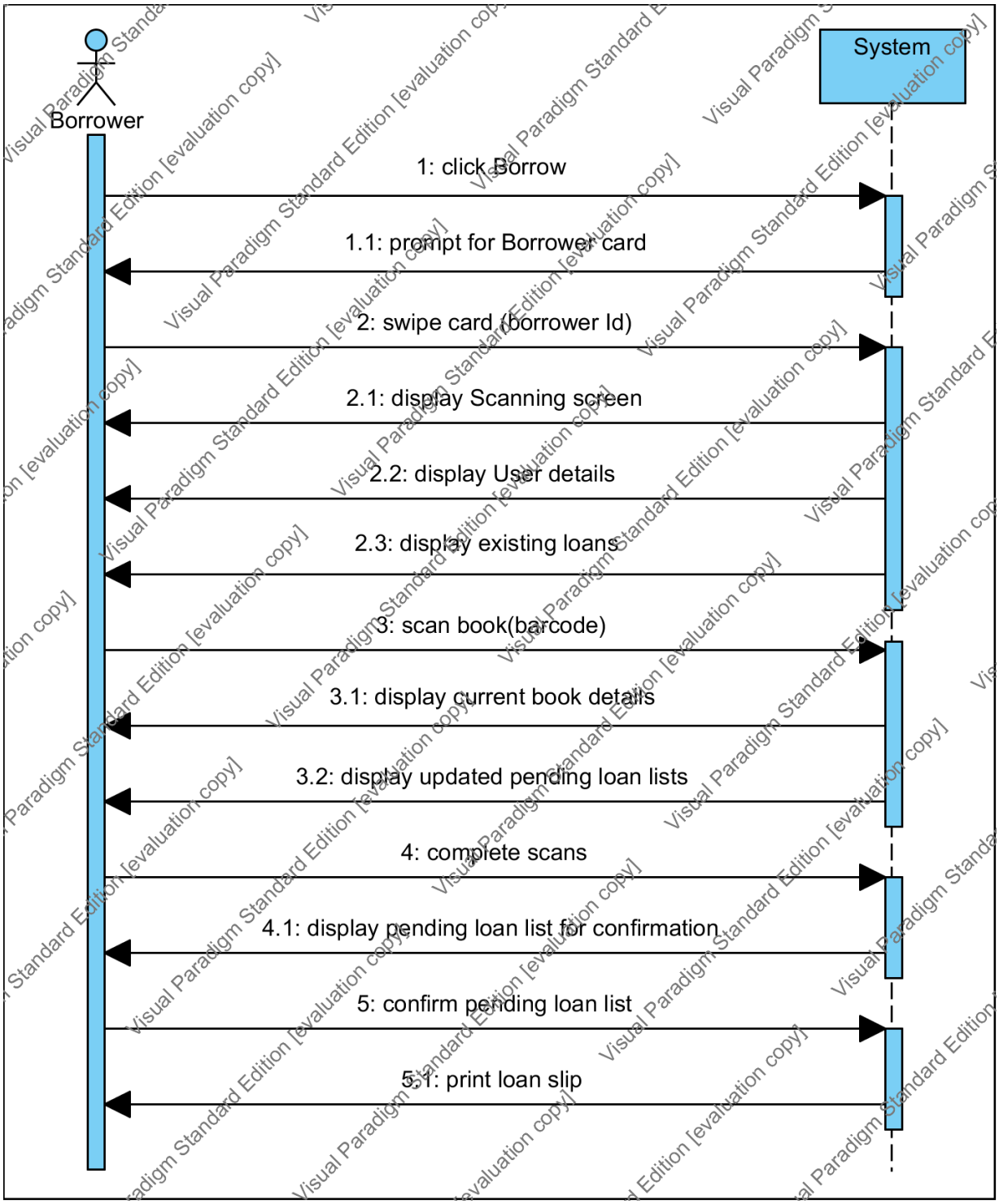


State Diagram: Loan

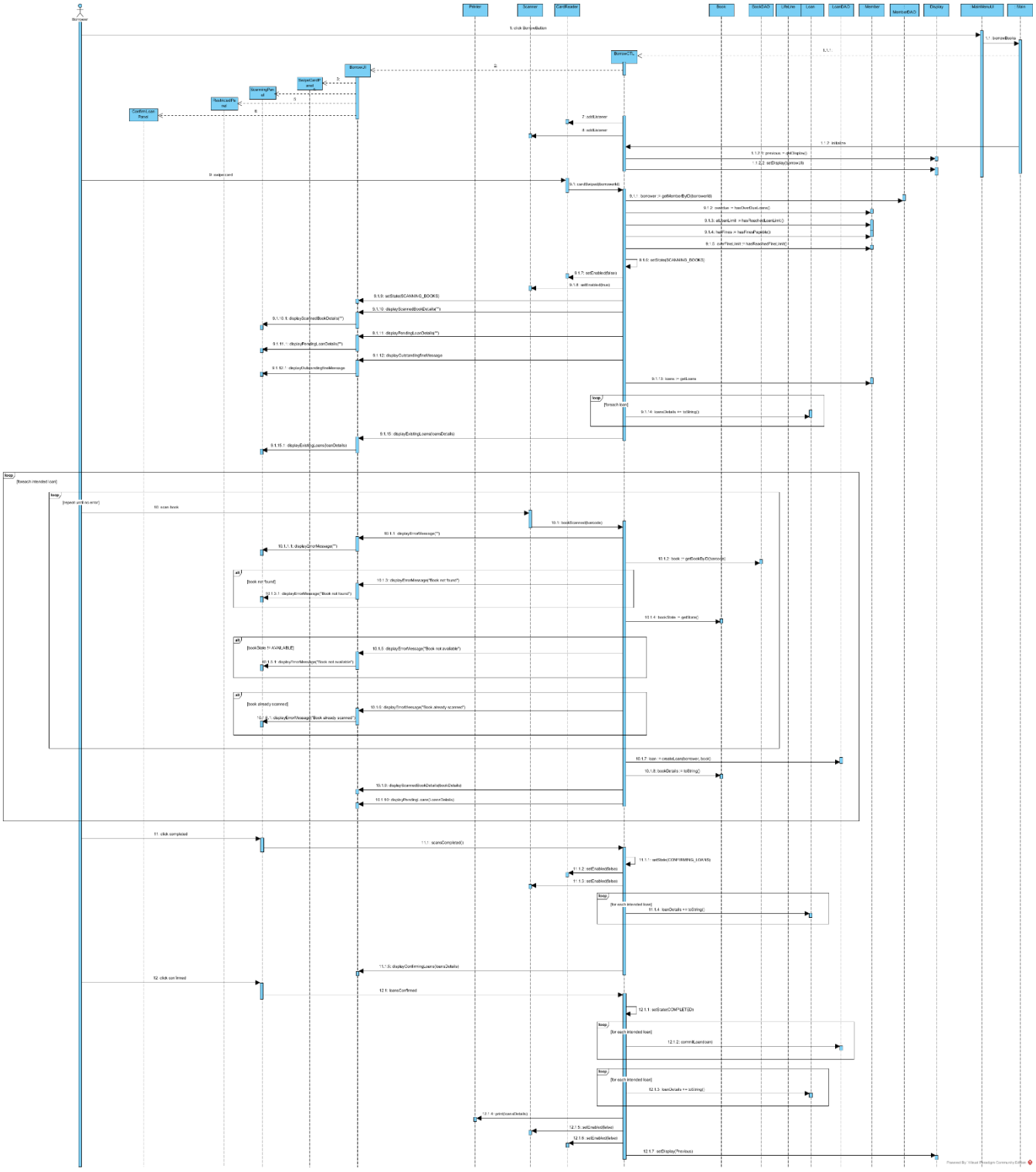


Sequence Diagram: Borrow Book

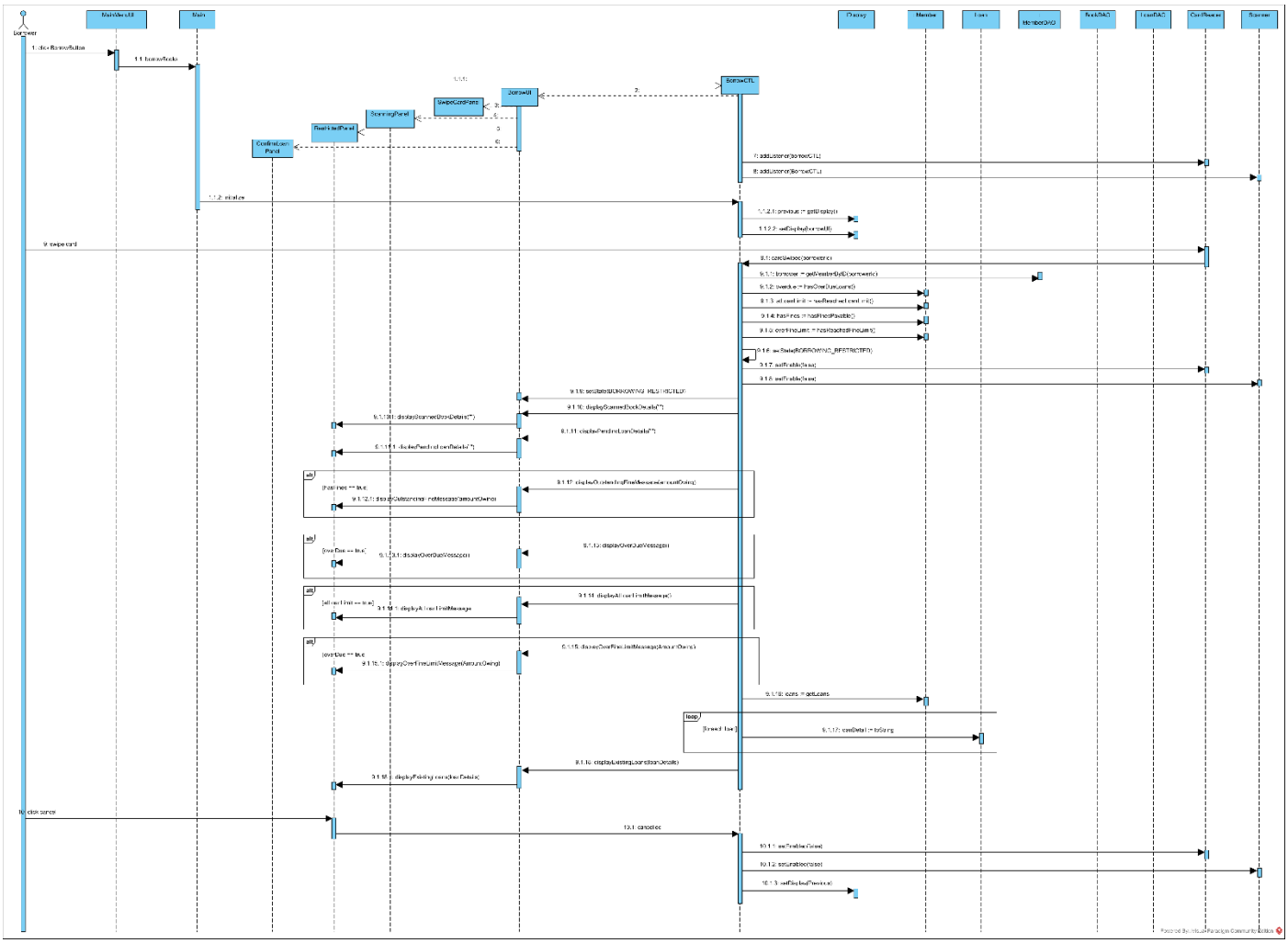
Borrow Book SSD – Normal Flow



Borrow Book – Borrowing Not Restricted



Borrow Book – Borrowing Restricted



Library System Domain Entities Software Specification

Entities

Member – implements IMember

Constructor : Member
Parameters:
firstName:string, lastName:string, contactPhone :string, emailAddress:string, id:int
throws IllegalArgumentException if:
any of firstName, lastName, contactPhone, emailAddress are null or blank id is less than or equal to zero

hasOverDueLoans: boolean
returns true if:
any existing loan is overdue
returns false otherwise

hasReachedLoanLimit: boolean
returns true if:
number of existing loans equal LOAN_LIMIT
returns false otherwise

hasFinesPayable: boolean
returns true if the fines owing by the Member exceeds zero
returns false otherwise

hasReachedFineLimit: boolean
returns true if:
the fines owing by the Member is equals or is greater than FINE_MAX
returns false otherwise

getFineAmount: float
returns the amount of fines owing by the Member

addFine: void
Parameters: amount:float
Increments the fines owing by amount
Throws IllegalArgumentException if:
amount is negative

payFine: void
Parameters: amount:float
Decrements the fines owing by amount
Throws IllegalArgumentException if:
amount is negative, or amount exceeds fines owing

addLoan: void
Parameters: loan:ILoan
Adds loan to the Members list of current loans
Throws IllegalArgumentException if:
loan is null ||
member is currently BORROWING_DISALLOWED

getLoans: List<ILoan>
returns a copy of the Members current loan list

removeLoan: void
Parameters: loan:ILoan
Removes a loan from the Members list of current loans
Throws IllegalArgumentException if:
loan is null, or loan not present in current loans

getState: EMemberState
returns the current MemberState of the Member

getFirstName: string
returns the firstName of the Member

getLastName: string
returns the lastName of the Member

getContactPhone: string
returns the contactPhone of the Member

getEmailAddress: string
returns the emailAddress of the Member

getID: int
returns the unique id of the member

Book – implements IBook

Constructor: Book
Parameters: author:String, title:String, callNumber:String, bookID:int
throws IllegalArgumentException if:
any of author, title, or callNumber are null or blank ||
bookID is less than or equal to zero

borrow: void
Parameters: loan:ILoan
associates the loan with the book
throws RuntimeException if:
the book is not currently AVAILABLE

getLoan: ILoan
returns the loan associated with the book
returns null if:
the book is not current ON_LOAN

returnBook: void
Parameters: damaged:boolean
removes the loan the book is associated with if:
damaged sets the BookState to DAMAGED
else
sets the BookState to AVAILABLE
throws RuntimeException if:
the book is not currently ON_LOAN

lose: void
sets the BookState to LOST
throws RuntimeException if:
the book is not currently ON_LOAN

repair: void
sets the BookState to AVAILABLE
throws RuntimeException if: the book is not currently DAMAGED

dispose: void
sets the BookState to DISPOSED
throws RuntimeException if:
the book is not currently AVAILABLE, DAMAGED, or LOST

getState: EBookState
returns the book's current BookState

getAuthor: string
returns the book's author

getTitle: string
returns the book's title

getCallNumber: string
returns the book's call number

getID: int
returns the book's unique id

Loan – implements ILoan

Constructor: Loan
Parameters: book:IBook, borrower: IMember, borrowDate:Date, dueDate:Date, loanID:int
throws IllegalArgumentException if:
any of book, borrower, borrowDate, or dueDate are null
dueDate is less than borrowDate
loanID is less than or equal to zero

commit: void
sets the current state of the Loan to CURRENT
calls book.borrow with itself as parameter
calls borrower.addloan with itself as parameter
throws a RuntimeException if:
the loan's current LoanState is not PENDING

complete: void
sets the current state of the Loan to COMPLETE
throws a RuntimeException if:
the loan's current LoanState is not CURRENT or OVERDUE

isOverDue: boolean
returns true if current LoanState of loan is OVERDUE

checkOverDue: boolean
Parameters: currentDate:Date
Sets current LoanState to OVERDUE if currentDate > loan's dueDate
throws a RuntimeException if:
the loan's current LoanState is not CURRENT or OVERDUE

getBorrower: IMember
returns the borrower associated with the loan

getBook: book:IBook
returns the book associated with the loan

getID: id:int
returns the book's unique id

Data Access Objects

BookDAO – implements IBookDAO

Constructor: BookDAO
Parameters: helper:IBookHelper
throws IllegalArgumentException if helper is null

addBook: IBook
Parameters: author:string, title:string, callNumber:string
uses helper to create a new Book with a unique book ID
adds the new book to a collection of Books
returns the new book

getBookByID: IBook
Parameters: id:int
returns the book identified by the id from the Book collection
returns null if book not found

listBooks: List<IBook>
returns a list of all books in the Book collection

findBooksByAuthor: List<IBook>
Parameters: author:string
returns a list of all books in the collection written by author
returns empty list if no books found

findBooksByTitle: List<IBook>
Parameters: title:string
returns a list of all books in the collection whose title is title
returns empty list if no books found

findBooksByAuthorTitle: List<IBook>
Parameters: author:string , title:string
returns a list of all books in the collection whose title is title and whose author is author
returns empty list if no books found

MemberDAO – implements IMemberDAO

Constructor: MemberDAO
Parameters: helper:IMemberHelper
throws IllegalArgumentException if helper is null

addMember: IBook
Parameters: firstName:string, lastName :string, contactPhone :string, emailAddress:string
uses helper to create a new Member with a unique member ID
adds the new member to a collection of Members
returns the new member

getMemberByID: IMember
Parameters: id:int
returns the member identified by the id from the Member collection
returns null if member not found

listBooks: List<IMEMBER>
returns a list of all members in the Member collection

findMembersByLastName: List<IMember>
Parameters: lastName:string
returns a list of all members in the collection whose last name is lastName
returns empty list if no members found

findMembersByEmailAddress: List<IMember>
Parameters: emailAddress:string
returns a list of all members in the collection whose email address is emailAddress
returns empty list if no members found

findMembersByNames: List<IMember>
Parameters: firstName:string, lastName:string
returns a list of all members in the collection whose first name is firstName and whose last name is lastName
returns empty list if no members found

LoanDAO – implements ILoanDAO

Constructor: LoanDAO
Parameters: helper:ILoanHelper
throws IllegalArgumentException if helper is null

createLoan: ILoan
Parameters: borrower:IMember, book:IBook
Uses helper to create a new loan with default id of zero
Returns new loan
throws IllegalArgumentException if borrower or book is null

commitLoan: ILoan
Assigns the loan a unique id
Stores the loan

getLoanByID: ILoan
Parameters: id:int
Returns the loan in the committed loan collection identified by id
Returns null if loan not found

getLoanByBook: ILoan
Parameters: book:IBook
Returns the loan in the committed loan collection associated with book
Returns null if loan not found

listLoans: List<ILoan>
Returns a list of all loans in the committed loan collection

findLoansByBorrower: List<ILoan>
Parameters: borrower:IMember
Returns a list of all loans in the committed loan collection associated with borrower

findLoansByBookTitle: List<ILoan>
Parameters: title:string
Returns a list of all loans in the committed loan collection associated with books with the title 'title'

updateOverDueStatus: void
Parameters: date:Date
Iterates through the committed loan collection updating the overdue status of current
loans according to date.

findOverDueLoans: List<ILoan>
Returns a list of all loans in the committed loan collection which are currently overdue

Helpers

LoanHelper– implements ILoanHelper

makeLoan: ILoan
Parameters: book:IBook, borrower:IMember, borrowDate:Date, dueDate:Date
returns a new loan with a default id of zero

BookHelper– implements IBookHelper

makeBook: IBook
Parameters: author: string, title: string, callNumber:string, id:int
returns a new book with an id specified by id

MemberHelper– implements IMemberHelper

makeMember: IMember
Parameters: firstName: string, lastName:string, contactPhone:string, emailAddress:string, id:int
returns a new book with an id specified by id

Borrow Book System Operation Specifications

BBUC_Op1: Begin Use Case
Intent: Initialize Borrow Book Use Case
Signature: initialize()
PreConditions:

BorrowBookCTL class exists

memberDAO, loanDAO
bookDAO, display, cardReader,
scanner, printer all exist

BorrowBookCTL added as listener to
cardReader and scanner

BorrowBookCTL state == ***CREATED***

PostConditions:

BorrowBookUI is displayed
SwipeCard panel of BorrowBookUI displayed
Cancel button enabled
cardReader is enabled
scanner is disabled
BorrowBookCTL state == ***INITIALIZED***

BBUC_Op2: Swipe Borrower Card
Intent: Identify borrower to system
Signature: cardSwiped(borrowerId)
PreConditions:

cardReader visible and enabled

BorrowBookCTL added as listener to
cardReader

memberDAO exists
BorrowBookCTL state == ***INITIALIZED***

PostConditions:
(member exists and not restricted)

Scanning panel of BorrowBookUI displayed
Complete, Cancel buttons enabled
cardReader is disabled
scanner is enabled
borrower details displayed
existing loans displayed
scan count initialized as number of
existing loans
existing fine message displayed if relevant
BorrowBookCTL state == ***SCANNING_BOOKS***

(member exists and is restricted)

Restricted panel of BorrowBookUI displayed
Cancel button enabled
cardReader is disabled
scanner is disabled
borrower details displayed
existing loans displayed
existing fine message displayed if relevant
overdue message displayed if relevant
atLoanLimit message displayed if relevant
borrowing restricted error message displayed
BorrowBookCTL state ==
BORROWING_RESTRICTED

BBUC_Op3: Scan Book
Intent: Add book to list of intended/pending loans
Signature: bookScanned(barcode)
PreConditions:

BorrowBookCTL class exists

scanner exists

BorrowBookCTL added as listener to scanner

BorrowBookCTL state == ***SCANNING_BOOKS***

PostConditions:

(book not found)

BorrowBookUI is displayed
Scanning panel of BorrowBookUI displayed
Cancel button enabled
cardReader is disabled
scanner is enabled
book not found error message displayed
BorrowBookCTL state == ***SCANNING_BOOKS***

(book not available)

BorrowBookUI is displayed
Scanning panel of BorrowBookUI displayed
Cancel button enabled
cardReader is disabled
scanner is enabled
book not available error message displayed
BorrowBookCTL state == ***SCANNING_BOOKS***

(book already scanned)

BorrowBookUI is displayed
Scanning panel of BorrowBookUI displayed
Cancel button enabled
cardReader is disabled
scanner is enabled
book already scanned error message displayed
BorrowBookCTL state == ***SCANNING_BOOKS***

(scan count < loan limit)

BorrowBookUI is displayed
Scanning panel of BorrowBookUI displayed
Cancel button enabled
cardReader is disabled
scanner is enabled

scanCount incremented by 1
scanned books details displayed
new (pending) loan created
(pending) loan added to pending loan list
Current pending loan list displayed
BorrowBookCTL state == ***SCANNING_BOOKS***

(scan count == loan limit)

BorrowBookUI is displayed
ConfirmingLoans panel of BorrowBookUI displayed
final pending loan list displayed
Cancel button enabled
Reject button enabled
cardReader is disabled
scanner is disabled

scanCount incremented by 1
BorrowBookCTL state == ***CONFIRMING_LOANS***

BBUC_Op4: Complete Scans
Intent: Finish scanning books
Signature: scansCompleted()
PreConditions:

BorrowBookCTL class exists
Pending loan list exists

BorrowBookCTL state == ***SCANNING_BOOKS***

PostConditions:

BorrowBookUI is displayed
Confirming Loan panel of BorrowBookUI
Is displayed
List of pending loans displayed
Cancel button enabled
cardReader is disabled
scanner is disabled
BorrowBookCTL state == ***CONFIRMING_LOANS***

BBUC_Op5: Confirm Loans
Intent: complete the borrowing book process
Signature: loansConfirmed()
PreConditions:

BorrowBookCTL class exists
Pending loan list exists

BorrowBookCTL state == ***CONFIRMING_LOANS***

PostConditions:

Main Menu is displayed
All pending loans are committed and recorded
Loan slip of committed loans printed
cardReader is disabled
scanner is disabled
BorrowBookCTL state == ***COMPLETED***

BBUC_Op6: Reject Loans
Intent: reject the current list of pending loans and
start the scanning process again
Signature: loansRejected()
PreConditions:

BorrowBookCTL class exists
Pending loan list exists

BorrowBookCTL state == ***CONFIRMING_LOANS***

PostConditions:

BorrowBookUI is displayed
Scanning panel of BorrowBookUI displayed
Borrower details displayed
Existing loan details displayed
pending loan list is empty
scan count == number of existing loans
Cancel button enabled
cardReader is disabled
scanner is enabled
BorrowBookCTL state == ***SCANNING_BOOKS***