**Library 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, or 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 equals LOAN\_LIMIT

returns false otherwise

**hasFinesPayable**: boolean

returns true if the fines currently owed 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