# Library management System

Requirements:

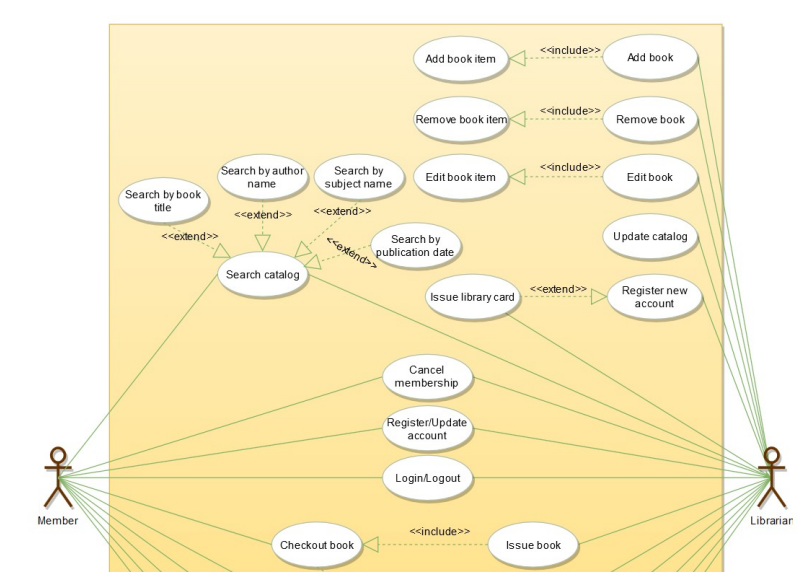
1. Any library member be it customer or a librarian should be able to search books by their title, author, subject, category as well by the publication date.
2. Each book will have a unique identification number and other details including rack number which will help to physically locate the book.
3. There could be more than one copy of a book, and library members should be able to check-out and reserve any copy. We will call each copy of a book, a book item.
4. The system should be able to retrieve information like who took a particular book or what are the book checked out by a specific library member.
5. There should be a maximum limit on how many books a member can check out.
6. There should be a maximum limit on how many days a member can keep a book.
7. The system should be able to collect fines for books returned after the due date.
8. Members should be able to reserve books that are not currently available.
9. The system should be able to send notification whenever a reserved book is available.
10. Each book and member card will have a unique barcode.

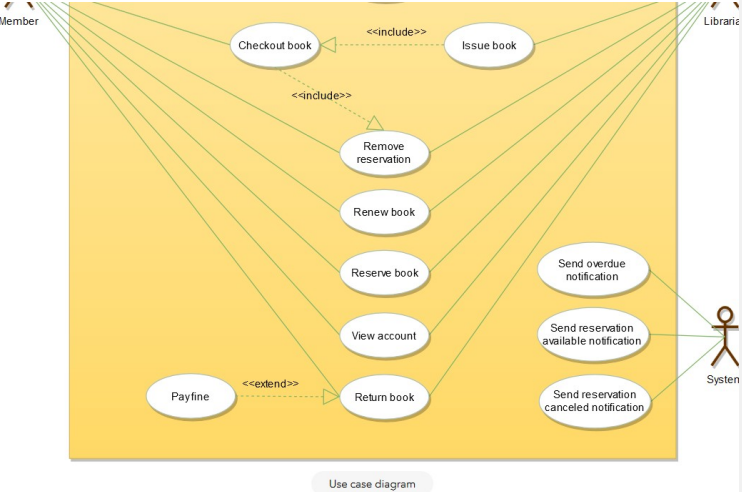
**Use case diagram:** we have three main actors:

1. Librarian: responsible for adding and modifying books, book items, and users. It can issue, reserve and return book items too.
2. Member: they can search the catalog, as well as checkout, reserve, renew and return a book.
3. System: responsible for sending notifications for overdue books, reservation canceled etc.

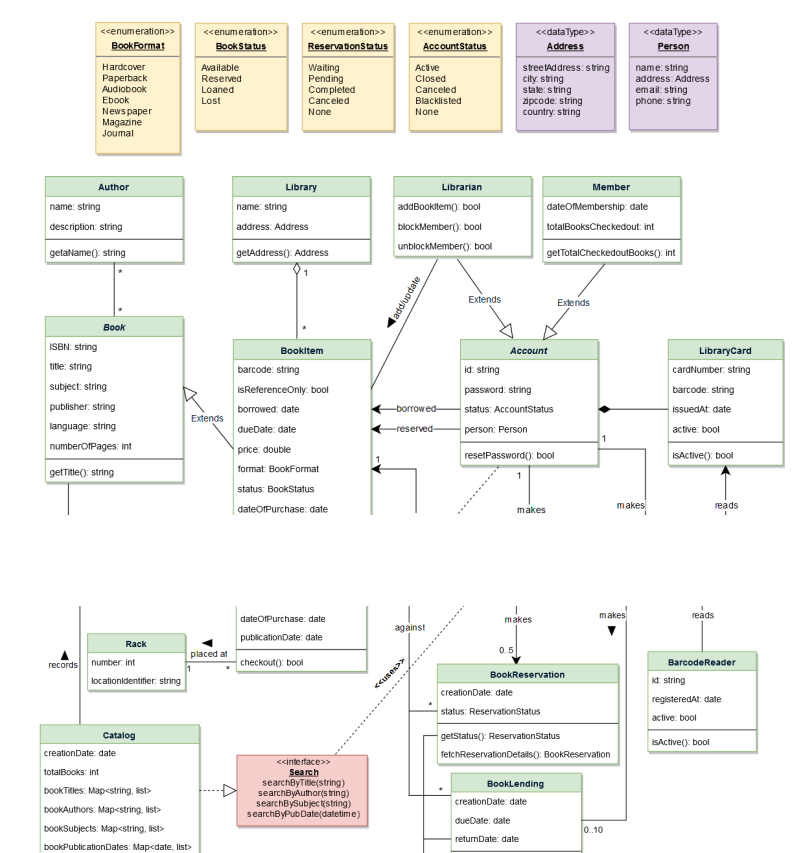
**Top use case for library:**

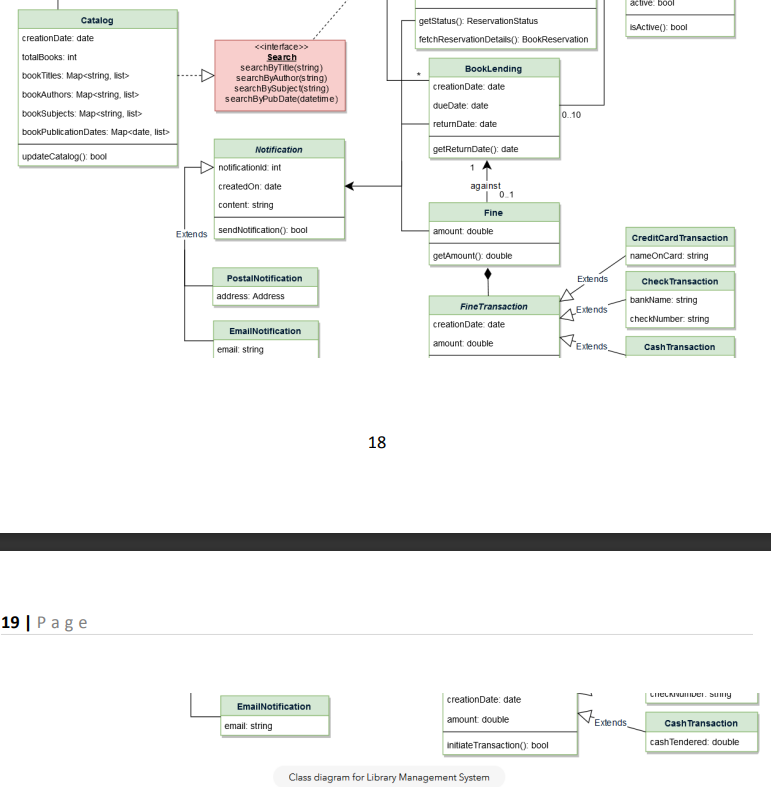
* **Add/remove/edit book.**
* **Search catalog**
* **Register new account.**
* **Checkout book**
* **Renew book.**
* **Return book.**

****

****

**Class diagram:**

****

****

**ode**

**Here is the code for the use-cases mentioned above 1) Check-out a book, 2) Return a book,**

**and 3) Renew a book.**

**Note: This code only focuses on the design part of the use-cases. Since you are not supposed**

**to write a fully executable code in an interview, the reader can assume parts of the code to**

**interact with the database and payment system, etc.**

**Enums and Constants: Here are the required enums, data types, and constants:**

**public enum BookFormat {**

**HARDCOVER,**

**PAPERBACK,**

**AUDIO\_BOOK,**

**EBOOK,**

**NEWSPAPER,**

**MAGAZINE,**

**JOURNAL**

**}**

**public enum BookStatus {**

**AVAILABLE,**

**RESERVED,**

**LOANED,**

**LOST**

**}**

**public enum ReservationStatus{**

**WAITING,**

**PENDING,**

**CANCELED,**

**NONE**

**}**

**public enum AccountStatus{**

**ACTIVE,**

**CLOSED,**

**CANCELED,**

**BLACKLISTED,**

**NONE**

**}**

**public class Address {**

**private String streetAddress;**

**private String city;**

**private String state;**

**private String zipCode;**

**private String country;**

**}**

**23**

**24 | P a g e**

**public class Person {**

**private String name;**

**private Address address;**

**private String email;**

**private String phone;**

**}**

**public class Constants {**

**public static final int MAX\_BOOKS\_ISSUED\_TO\_A\_USER = 5;**

**public static final int MAX\_LENDING\_DAYS = 10;**

**}**

**Account, Member, and Librarian: These classes represent different people that interact with our**

**system:**

**// For simplicity, we are not defining getter and setter functions. The reader can**

**// assume that all class attributes are private and accessed through their respective**

**// public getter method and modified only through their public setter method.**

**public abstract class Account {**

**private String id;**

**private String password;**

**private AccountStatus status;**

**private Person person;**

**public boolean resetPassword();**

**}**

**public class Librarian extends Account {**

**public boolean addBookItem(BookItem bookItem);**

**public boolean blockMember(Member member);**

**public boolean unBlockMember(Member member);**

**}**

**public class Member extends Account {**

**private Date dateOfMembership;**

**private int totalBooksCheckedout;**

**public int getTotalCheckedoutBooks();**

**public boolean reserveBookItem(BookItem bookItem);**

**private void incrementTotalBooksCheckedout();**

**public boolean checkoutBookItem(BookItem bookItem);**

**public boolean returnBookItem(BookItem bookItem);**

**public boolean renewBookItem(BookItem bookItem);**

**public boolean checkoutBookItem(BookItem bookItem) {**

**if(this.getTotalBooksCheckedOut() >=**

**Constants.MAX\_BOOKS\_ISSUED\_TO\_A\_USER ) {**

**ShowError("The user has already checkedout maximum number of books");**

**return false;**

**24**

**25 | P a g e**

**}**

**BookReservation bookReservation =**

**BookReservation.fectchReservationDetails(bookItem.getBarcode());**

**if( bookReservation != null && bookReservation.getMemberId() != this.getId() ) {**

**// book item has a pendening reservation from another user**

**ShowError("This book is reserved by another member");**

**return false;**

**} else if( bookReservation != null ) {**

**// book item has a pendening reservation from the give member, update it**

**bookReservation.updateStatus(ReservationStatus.COMPLETED);**

**}**

**if(!bookItem.checkout(this.getId())) {**

**return false;**

**}**

**this.incrementTotalBooksCheckedout();**

**return true;**

**}**

**private void checkForFine(String bookItemBarcode) {**

**BookLending bookLending = BookLending.fectchLendingDetails(bookItemBarcode);**

**Date dueDate = bookLending.getDueDate();**

**Date today = new Date();**

**// check if the book has been returned within the due date**

**if(todaye.compareTo(dueDate) > 0) {**

**long diff = todayDate.getTime() - dueDate.getTime();**

**long diffDays = diff / (24 \* 60 \* 60 \* 1000);**

**Fine.collectFine(memberId, diffDays);**

**}**

**}**

**public void returnBookItem(BookItem bookItem) {**

**this.checkForFine();**

**BookReservation bookReservation =**

**BookReservation.fectchReservationDetails(bookItem.getBarcode());**

**if(bookReservation != null) {**

**// book item has a pendening reservation**

**bookItem.updateBookItemStatus(BookStatus.RESERVED);**

**bookReservation.sendBookAvailableNotification();**

**}**

**bookItem.updateBookItemStatus(BookStatus.AVAILABLE);**

**}**

**public bool renewBookItem(BookItem bookItem) {**

**this.checkForFine();**

**BookReservation bookReservation =**

**BookReservation.fectchReservationDetails(bookItem.getBarcode());**

**if( bookReservation != null && bookReservation.getMemberId() !=**

**member.getMemberId() ) {**

**25**

**26 | P a g e**

**// book item has a pendening reservation from another member**

**ShowError("This book is reserved by another member");**

**member.decrementTotalBooksCheckedout();**

**bookItem.updateBookItemState(BookStatus.RESERVED);**

**bookReservation.sendBookAvailableNotification();**

**return false;**

**} else if( bookReservation != null ){**

**// book item has a pendening reservation from this member**

**bookReservation.updateStatus(ReservationStatus.COMPLETED);**

**}**

**BookLending.lendBook(bookItem.getBarCode(), this.getMemberId());**

**bookItem.updateDueDate(LocalDate.now().plusDays(Constants.MAX\_LENDING\_DAYS));**

**return true;**

**}**

**}**

**BookReservation, BookLending and Fine: These classes represent a book reservation, lending and**

**fine collection respectively.**

**public class BookReservation {**

**private Date creationDate;**

**private ReservationStatus status;**

**private String bookItemBarcode;**

**private String memberId;**

**public static BookReservation fectchReservationDetails(String barcode);**

**}**

**public class BookLending {**

**private Date creationDate;**

**private Date dueDate;**

**private Date returnDate;**

**private String bookItemBarcode;**

**private String memberId;**

**public static void lendBook(String barcode, String memberId);**

**public static BookLending fectchLendingDetails(String barcode);**

**}**

**public class Fine {**

**private Date creationDate;**

**private double bookItemBarcode;**

**private String memberId;**

**public static void collectFine(String memberId, long days) {}**

**}**

**26**

**27 | P a g e**

**BookItem: To encapsulate a book item. This class will be responsible for processing reservation,**

**return and renew of a book item.**

**public abstract class Book {**

**private String ISBN;**

**private String title;**

**private String subject;**

**private String publisher;**

**private String language;**

**private int numberOfPages;**

**private List<Author> authors;**

**}**

**public class BookItem extends Book {**

**private String barcode;**

**private boolean isReferenceOnly;**

**private Date borrowed;**

**private Date dueDate;**

**private double price;**

**private BookFormat format;**

**private BookStatus status;**

**private Date dateOfPurchase;**

**private Date publicationDate;**

**private Rack placedAt;**

**public boolean checkout(String memberId) {**

**if(bookItem.getIsReferenceOnly()) {**

**ShowError("This book is Reference only and can't be issued");**

**return false;**

**}**

**if(!BookLending.lendBook(this.getBarCode(), memberId)){**

**return false;**

**}**

**this.updateBookItemStatus(BookStatus.LOANED);**

**return true;**

**}**

**}**

**public class Rack {**

**private int number;**

**private String locationIdentifier;**

**}**

**Search interface and Catalog: Catalog will implement Search to facilitate searching of books.**

**public interface Search {**

**public List<Book> searchByTitle(String title);**

**public List<Book> searchByAuthor(String author);**

**27**

**28 | P a g e**

**public List<Book> searchBySubject(String subject);**

**public List<Book> searchByPubDate(Date publishDate);**

**}**

**public class Catalog implements Search {**

**private HashMap<String, List<Book>> bookTitles;**

**private HashMap<String, List<Book>> bookAuthors;**

**private HashMap<String, List<Book>> bookSubjects;**

**private HashMap<String, List<Book>> bookPublicationDates;**

**public List<Book> searchByTitle(String query) {**

**// return all books containing the string query in their title.**

**return bookTitltes.get(query);**

**}**

**public List<Book> searchByAuthor(String query) {**

**// return all books containing the string query in their author's name.**

**return bookAuthors.get(query);**

**}**

**}**