Back To Course Home

Grokking the Object Oriented Design Interview

0% completed

Search Course

Object-Oriented Design and UML

- Object-Oriented Basics
- OO Analysis and Design
- What is UML?
- Use Case Diagrams
- Class Diagram
- Sequence diagram
- Activity Diagrams

Object Oriented Design Case Studies

- Design a Library Management System
- Design a Parking Lot
- Design Amazon - Online Shopping System
- Design Stack Overflow
- Design a Movie Ticket Booking System
- Design an ATM
- Design an Airline Management System
- Design Blackjack and a Deck of Cards
- Design a Hotel Management System
- Design a Restaurant Management system
- Design Chess
- Design an Online Stock Brokerage System
- Design a Car Rental System
- Design LinkedIn
- Design Cricinfo
- Design Facebook - a social network

Appendix

- Contact us
- Other courses

Course Certificate

Mark Course as Completed

Design a Library Management System

We'll cover the following

- System Requirements
- Use case diagram
- Class diagram
- Activity diagrams
- Code

A Library Management System is a software built to handle the primary housekeeping functions of a library. Libraries rely on library management systems to manage asset collections as well as relationships with their members. Library management systems help libraries keep track of the books and their checkouts, as well as members' subscriptions and profiles.

Library management systems also involve maintaining the database for entering new books and recording books that have been borrowed with their respective due dates.

System Requirements

Always clarify requirements at the beginning of the interview. Be sure to ask questions to find the exact scope of the system that the interviewer has in mind.

We will focus on the following set of requirements while designing the Library Management System:

1. Any library member 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 a 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 books checked-out by a specific library member.
5. There should be a maximum limit (5) on how many books a member can check-out.
6. There should be a maximum limit (10) 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 notifications whenever the reserved books become available, as well as when the book is not returned within the due date.
10. Each book and member card will have a unique barcode. The system will be able to read barcodes from books and members' library cards.

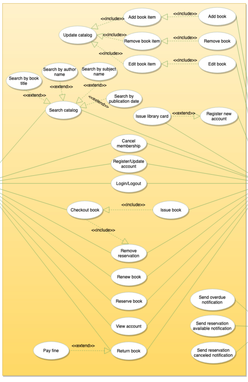
Use case diagram

We have three main actors in our system:

- **Librarian:** Mainly responsible for adding and modifying books, book items, and users. The Librarian can also issue, reserve, and return book items.
- **Member:** All members can search the catalog, as well as check-out, reserve, renew, and return a book.
- **System:** Mainly responsible for sending notifications for overdue books, canceled reservations, etc.

Here are the top use cases of the Library Management System:

- **Add/Remove/Edit book:** To add, remove or modify a book or book item.
- **Search catalog:** To search books by title, author, subject or publication date.
- **Register new account/cancel membership:** To add a new member or cancel the membership of an existing member.
- **Check-out book:** To borrow a book from the library.
- **Reserve book:** To reserve a book which is not currently available.
- **Renew a book:** To reborrow an already checked-out book.
- **Return a book:** To return a book to the library which was issued to a member.

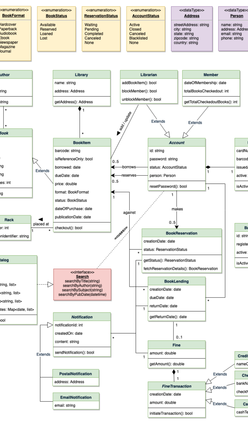


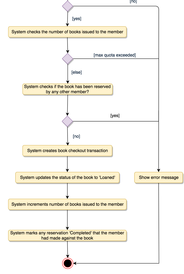
Use case diagram

Class diagram

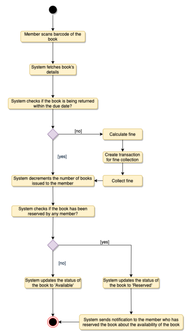
Here are the main classes of our Library Management System:

- **Library:** The central part of the organization for which this software has been designed. It has attributes like 'Name' to distinguish it from any other libraries and 'Address' to describe its location.
- **Book:** The basic building block of the system. Every book will have ISBN, Title, Subject, Publishers, etc.
- **BookItem:** Any book can have multiple copies, each copy will be considered a book item in our system. Each book item will have a unique barcode.
- **Account:** We will have two types of accounts in the system, one will be a general member, and the other will be a librarian.
- **LibraryCard:** Each library user will be issued a library card, which will be used to identify users while issuing or returning books.
- **BookReservation:** Responsible for managing reservations against book items.
- **BookLending:** Manage the checking-out of book items.
- **Catalog:** Catalogs contain list of books sorted on certain criteria. Our system will support searching through four catalogs: Title, Author, Subject, and Publish-date.
- **Fine:** This class will be responsible for calculating and collecting fines from library members.
- **Author:** This class will encapsulate a book author.
- **Rack:** Books will be placed on racks. Each rack will be identified by a rack number and will have a location identifier to describe the physical location of the rack in the library.
- **Notification:** This class will take care of sending notifications to library members.

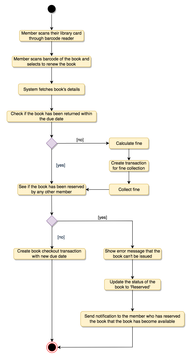

«enumeration» BookFormat
Hardcover
Paperback
Audiobook
Ebook
Newspaper
Magazine
Journal
«enumeration» BookStatus
Available
Reserved
Loaned
Lost
«enumeration» ReservationStatus
Waiting
Pending
Completed
Canceled
None
«enumeration» AccountStatus
Active
Closed
Canceled
Blacklisted
None
«nodata»Types» Address
streetAddress: string
city: string
state: string
zipcode: string
country: string
«nodata»Types» Person
name: string
phoneNumber: string
email: string
phone: string
Author
name: string
description: string
getAuthor(): string
Library
name: string
address: Address
getAddress(): Address
Librarian
addBook(item): bool
blockMember(): bool
unblockMember(): bool
Member
dateOfMembership: date
totalBooksCheckedout: int
getTotalCheckedoutBooks(): int
Book
ISBN: string
title: string
subject: string
publisher: string
language: string
numberOfPages: int
getTitle(): string
BookItem
barcode: string
isReferenceOnly: bool
borrowed: date
dueDate: date
price: double
format: BookFormat
status: BookStatus
dateOfPurchase: date
publicationDate: date
checkout(): bool
Account
id: string
password: string
status: AccountStatus
person: Person
resetPassword(): bool
LibraryCard
cardNumber: string
barcode: string
issuedOn: date
active: bool
isActive(): bool
BarcodeReader
id: string
registeredAt: date
active: bool
isActive(): bool
Rack
number: int
locationIdentifier: string
Catalog
creationDate: date
totalBooks: int
searchTitle: Map<string, list>
searchAuthors: Map<string, list>
searchSubjects: Map<string, list>
searchPublicationDate: Map<date, list>
updateCatalog(): bool
«interface» Search
searchByTitle(string)
searchByAuthor(string)
searchBySubject(string)
searchByPublicationDate(date)
BookReservation
creationDate: date
status: ReservationStatus
getStatus(): ReservationStatus
lastReservationDate(): BookReservation
BookLending
creationDate: date
dueDate: date
returnDate: date
getReturnDate(): date
Notification
notificationId: int
creationOn: date
content: string
sendNotification(): bool
PostalNotification
address: Address
EmailNotification
email: string
Fine
amount: double
getAmount(): double
FineTransaction
creationDate: date
amount: double
makeTransaction(): bool
CreditCardTransaction
nameOnCard: string
CheckTransaction
bankName: string
checkNumber: string
CashTransaction
cashTendered: double
Extends
Extends
Extends



Return a book: Any library member or librarian can perform this activity. The system will collect fines from members if they return books after the due date. Here are the steps for returning a book:



Renew a book: While renewing (re-issuing) a book, the system will check for fines and see if any other member has not reserved the same book, in that case the book item cannot be renewed. Here are the different steps for renewing a book:



Code

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 required to write a fully executable code in an interview, you can assume parts of the code to interact with the database, payment system, etc.

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

Java Python

```java
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,
```

Account, Member, and Librarian: These classes represent various people that interact with our system:

Java Python

```java
// 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 methods and modified only through their public methods function.

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 getTotalBooksCheckedout();

  public boolean reserveBookItem(BookItem bookItem);
```

BookReservation, BookLending, and Fine: These classes represent a book reservation, lending, and fine collection, respectively.

Java Python

```java
public class BookReservation {
  private Date creationDate;
  private ReservationStatus status;
  private String bookItemBarcode;
  private String memberId;

  public static BookReservation fetchReservationDetails(String barcode);
}

public class BookLending {
  private Date creationDate;
  private Date dueDate;
  private Date returnDate;
  private String bookItemBarcode;
  private String memberId;

  public static boolean lendBook(String barcode, String memberId);
  public static BookLending fetchLendingDetails(String barcode);
}

public class Fine {
  private Date creationDate;
  private double bookItemBarcode;
  private String memberId;

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

BookItem: Encapsulating a book item, this class will be responsible for processing the reservation, return, and renewal of a book item.

Java Python

```java
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)){
```

Search interface and Catalog: The Catalog class will implement the Search interface to facilitate searching of books.

Java Python

```java
public interface Search {
  public List<Book> searchByTitle(String title);
  public List<Book> searchByAuthor(String author);
  public List<Book> searchBySubject(String subject);
  public List<Book> searchPubDate(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 bookTitles.get(query);
  }

  public List<Book> searchByAuthor(String query) {
    // return all books containing the string query in their author's name.
    return bookAuthors.get(query);
  }
}
```