<h1 align="center">Design a Library Management System</h1>

<h3 align="center">Let's design a Library Management System</h3>

\*\*We'll cover the following:\*\*

\* [System Requirements](#system-requirements)

\* [Use Case Diagram](#use-case-diagram)

\* [Class Diagram](#class-diagram)

\* [Activity Diagrams](#activity-diagrams)

\* [Code](#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.

<p align="center">

<img src="/media-files/library-system.png" alt="Library Management System">

<br />

Library Management System

</p>

### System Requirements

<p align="center">

<b>

<i>

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.

</i>

</b>

</p>

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.

Here is the use case diagram of our Library Management System:

<p align="center">

<img src="/media-files/lib-use-case-diagram.png" alt="Library Use Case Diagram">

<br />

Use Case Diagram for Library Management System

</p>

### 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.

<p align="center">

<img src="/media-files/lib-class-diagram.png" alt="Library Class Diagram">

<br />

Class Diagram for Library Management System

</p>

<p align="center">

<img src="/media-files/lib-uml.svg" alt="Library UML">

<br />

UML for Library Management System

</p>

### Activity Diagrams

\*\*Check-out a book:\*\* Any library member or librarian can perform this activity. Here are the set of steps to check-out a book:

<p align="center">

<img src="/media-files/lib-check-out-book.svg" alt="Check-out Book Activity Diagram">

<br />

Activity Diagram for Library Management System Check-out Book

</p>

\*\*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:

<p align="center">

<img src="/media-files/lib-return-book.png" alt="Return Book Activity Diagram">

<br />

Activity Diagram for Library Management System Return Book

</p>

\*\*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:

<p align="center">

<img src="/media-files/lib-renew-book.svg" alt="Renew Book Activity Diagram">

<br />

Activity Diagram for Library Management System Renew Book

</p>

### 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:

\*\*Code Snippet:\*\*

```python

from abc import ABC

from enum import Enum

class BookFormat(Enum):

HARDCOVER, PAPERBACK, AUDIO\_BOOK, EBOOK, NEWSPAPER, MAGAZINE, JOURNAL = 1, 2, 3, 4, 5, 6, 7

class BookStatus(Enum):

AVAILABLE, RESERVED, LOANED, LOST = 1, 2, 3, 4

class ReservationStatus(Enum):

WAITING, PENDING, CANCELED, NONE = 1, 2, 3, 4

class AccountStatus(Enum):

ACTIVE, CLOSED, CANCELED, BLACKLISTED, NONE = 1, 2, 3, 4, 5

class Address:

def \_\_init\_\_(self, street, city, state, zip\_code, country):

self.\_\_street\_address = street

self.\_\_city = city

self.\_\_state = state

self.\_\_zip\_code = zip\_code

self.\_\_country = country

class Person(ABC):

def \_\_init\_\_(self, name, address, email, phone):

self.\_\_name = name

self.\_\_address = address

self.\_\_email = email

self.\_\_phone = phone

class Constants:

def \_\_init\_\_(self):

self.MAX\_BOOKS\_ISSUED\_TO\_A\_USER = 5

self.MAX\_LENDING\_DAYS = 10

```

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

\*\*Code Snippet:\*\*

```python

# 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.

from abc import ABC

from datetime import datetime

from .constants import \*

from .models import \*

class Account(ABC):

def \_\_init\_\_(self, id, password, person, status=AccountStatus.Active):

self.\_\_id = id

self.\_\_password = password

self.\_\_status = status

self.\_\_person = person

def reset\_password(self):

None

class Librarian(Account):

def \_\_init\_\_(self, id, password, person, status=AccountStatus.Active):

super().\_\_init\_\_(id, password, person, status)

def add\_book\_item(self, book\_item):

None

def block\_member(self, member):

None

def un\_block\_member(self, member):

None

class Member(Account):

def \_\_init\_\_(self, id, password, person, status=AccountStatus.Active):

super().\_\_init\_\_(id, password, person, status)

self.\_\_date\_of\_membership = datetime.date.today()

self.\_\_total\_books\_checkedout = 0

def get\_total\_books\_checkedout(self):

return self.\_\_total\_books\_checkedout

def reserve\_book\_item(self, book\_item):

None

def increment\_total\_books\_checkedout(self):

None

def renew\_book\_item(self, book\_item):

None

def checkout\_book\_item(self, book\_item):

if self.get\_total\_books\_checked\_out() >= Constants.MAX\_BOOKS\_ISSUED\_TO\_A\_USER:

print("The user has already checked-out maximum number of books")

return False

book\_reservation = BookReservation.fetch\_reservation\_details(book\_item.get\_barcode())

if book\_reservation != None and book\_reservation.get\_member\_id() != self.get\_id():

# book item has a pending reservation from another user

print("self book is reserved by another member")

return False

elif book\_reservation != None:

# book item has a pending reservation from the give member, update it

book\_reservation.update\_status(ReservationStatus.COMPLETED)

if not book\_item.checkout(self.get\_id()):

return False

self.increment\_total\_books\_checkedout()

return True

def check\_for\_fine(self, book\_item\_barcode):

book\_lending = BookLending.fetch\_lending\_details(book\_item\_barcode)

due\_date = book\_lending.get\_due\_date()

today = datetime.date.today()

# check if the book has been returned within the due date

if today > due\_date:

diff = today - due\_date

diff\_days = diff.days

Fine.collect\_fine(self.get\_member\_id(), diff\_days)

def return\_book\_item(self, book\_item):

self.check\_for\_fine(book\_item.get\_barcode())

book\_reservation = BookReservation.fetch\_reservation\_details(book\_item.get\_barcode())

if book\_reservation != None:

# book item has a pending reservation

book\_item.update\_book\_item\_status(BookStatus.RESERVED)

book\_reservation.send\_book\_available\_notification()

book\_item.update\_book\_item\_status(BookStatus.AVAILABLE)

def renew\_book\_item(self, book\_item):

self.check\_for\_fine(book\_item.get\_barcode())

book\_reservation = BookReservation.fetch\_reservation\_details(

book\_item.get\_barcode())

# check if self book item has a pending reservation from another member

if book\_reservation != None and book\_reservation.get\_member\_id() != self.get\_member\_id():

print("self book is reserved by another member")

self.decrement\_total\_books\_checkedout()

book\_item.update\_book\_item\_state(BookStatus.RESERVED)

book\_reservation.send\_book\_available\_notification()

return False

elif book\_reservation != None:

# book item has a pending reservation from self member

book\_reservation.update\_status(ReservationStatus.COMPLETED)

BookLending.lend\_book(book\_item.get\_bar\_code(), self.get\_member\_id())

book\_item.update\_due\_date(datetime.datetime.now().AddDays(Constants.MAX\_LENDING\_DAYS))

return True

```

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

\*\*Code Snippet:\*\*

```python

class BookReservation:

def \_\_init\_\_(self, creation\_date, status, book\_item\_barcode, member\_id):

self.\_\_creation\_date = creation\_date

self.\_\_status = status

self.\_\_book\_item\_barcode = book\_item\_barcode

self.\_\_member\_id = member\_id

def fetch\_reservation\_details(self, barcode):

None

class BookLending:

def \_\_init\_\_(self, creation\_date, due\_date, book\_item\_barcode, member\_id):

self.\_\_creation\_date = creation\_date

self.\_\_due\_date = due\_date

self.\_\_return\_date = None

self.\_\_book\_item\_barcode = book\_item\_barcode

self.\_\_member\_id = member\_id

def lend\_book(self, barcode, member\_id):

None

def fetch\_lending\_details(self, barcode):

None

class Fine:

def \_\_init\_\_(self, creation\_date, book\_item\_barcode, member\_id):

self.\_\_creation\_date = creation\_date

self.\_\_book\_item\_barcode = book\_item\_barcode

self.\_\_member\_id = member\_id

def collect\_fine(self, member\_id, days):

None

```

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

\*\*Code Snippet:\*\*

```python

from abc import ABC

from .constants import \*

class Book(ABC):

def \_\_init\_\_(self, ISBN, title, subject, publisher, language, number\_of\_pages):

self.\_\_ISBN = ISBN

self.\_\_title = title

self.\_\_subject = subject

self.\_\_publisher = publisher

self.\_\_language = language

self.\_\_number\_of\_pages = number\_of\_pages

self.\_\_authors = []

class BookItem(Book):

def \_\_init\_\_(self, barcode, is\_reference\_only, borrowed, due\_date, price, book\_format, status,

date\_of\_purchase, publication\_date, placed\_at):

self.\_\_barcode = barcode

self.\_\_is\_reference\_only = is\_reference\_only

self.\_\_borrowed = borrowed

self.\_\_due\_date = due\_date

self.\_\_price = price

self.\_\_format = book\_format

self.\_\_status = status

self.\_\_date\_of\_purchase = date\_of\_purchase

self.\_\_publication\_date = publication\_date

self.\_\_placed\_at = placed\_at

def checkout(self, member\_id):

if self.get\_is\_reference\_only():

print("self book is Reference only and can't be issued")

return False

if not BookLending.lend\_book(self.get\_bar\_code(), member\_id):

return False

self.update\_book\_item\_status(BookStatus.LOANED)

return True

class Rack:

def \_\_init\_\_(self, number, location\_identifier):

self.\_\_number = number

self.\_\_location\_identifier = location\_identifier

```

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

\*\*Code Snippet:\*\*

```python

from abc import ABC

class Search(ABC):

def search\_by\_title(self, title):

None

def search\_by\_author(self, author):

None

def search\_by\_subject(self, subject):

None

def search\_by\_pub\_date(self, publish\_date):

None

class Catalog(Search):

def \_\_init\_\_(self):

self.\_\_book\_titles = {}

self.\_\_book\_authors = {}

self.\_\_book\_subjects = {}

self.\_\_book\_publication\_dates = {}

def search\_by\_title(self, query):

# return all books containing the string query in their title.

return self.\_\_book\_titles.get(query)

def search\_by\_author(self, query):

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

return self.\_\_book\_authors.get(query)

```