Code of Library Management System

Let's write the code for the designed classes in different languages.

We'll cover the following

We've gone over the different aspects of the library management system and observed the attributes attached to the problem using various UML diagrams. Let us now explore the more practical side of things, where we will work on implementing the library management system using multiple languages. This is usually the last step in an object-oriented design interview process.

We have chosen the following languages to write the skeleton code of the different classes present in the library management system:

- Java
- C#
- Python
- C++
- JavaScript

Library management

In this section, we will provide the skeleton code of the classes designed in the class diagram lesson.

Note: 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 method functions.

Enumerations

First, we will define all the enumerations required in the library management system. According to the class diagram, there are four enumerations used in the system: `BookFormat`, `BookStatus`, `ReservationStatus`, and `AccountStatus`. The code to implement these enumerations is as follows:

Note: JavaScript does not support enumerations, so we will be using the `Object.freeze()` method as an alternative that freezes an object and prevents further modifications.

```
// definition of enumerations used in library management system
enum BookFormat {
    HARDCOVER,
    PAPERBACK,
    AUDIOBOOK,
    EBOOK,
    NEWSPAPER,
    MAGAZINE,
    JOURNAL
}

enum BookStatus {
    AVAILABLE,
    RESERVED,
    LOANED,
    LOST
}

enum ReservationStatus {
    WAITING,
    PENDING,
    CANCELED,
    NONE
}

enum AccountStatus {
    ACTIVE,
    CLOSED,
    CANCELED,
    BLACKLISTED,
    NONE
}
```

Enum definitions

Address and person

This section contains the code for `Address` and `Person` classes where the `Person` class is composed of an `Address` class. The implementation of these classes can be found below:

```
public class Address {
    private String streetAddress;
    private String city;
    private String state;
    private int zipCode;
    private String country;

}

public class Person {
    private String name;
    private Address address;
    private String email;
    private String phone;

}
```

The Address and Person classes

User

The `User` is an abstract class that represents the various people or actors that can interact with the system. Since there are two types of users, the librarian and the library member, the user can either be a `Librarian` or a `Member`. The implementation of the mentioned classes is shown below:

```
// User is an abstract class
public abstract class User {
    private String id;
    private String password;
    private AccountStatus status;
    private Person person;
    private LibraryCard card;

    public abstract boolean resetPassword();
}

public class Librarian extends User {
    public boolean addBookItem(BookItem bookItem);
    public boolean blockMember(Member member);
    public boolean unBlockMember(Member member);
    public boolean resetPassword() {
        // definition
    }
    public boolean addBookItem(BookItem bookItem) {
        // definition
    }
}

public class Member extends User {
    private Date dateOfMembership;
    private int totalBooksCheckedOut;

    public boolean reserveBookItem(BookItem bookItem);
    private void incrementTotalBooksCheckedOut();
    public boolean reserveBookItem(BookItem bookItem) {
```

User and its child classes

Book reservation, book lending and fine

This component shows the implementation of `BookReservation`, `BookLending`, and `Fine` classes. These classes will be responsible for managing reservations against books, managing reservations, and calculating fine on books. The code is shown below:

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

    public static BookReservation fetchReservationDetails(String bookItemId);
}

public class BookLending {
    private String itemId;
    private Date creationDate;
    private Date dueDate;
    private Date returnDate;
    private String memberId;
    private BookReservation bookReservation;
    private User user;

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

public class Fine {
    private Date creationDate;
    private String bookItemId;
    private String memberId;

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

The BookReservation, BookLending and Fine classes

Book and rack

The `Book` is an abstract class and `BookItem` represents each copy of the book. For example, if there are two copies of the same book then there would only be one `Book` object and two `BookItem` objects. The code to implement these classes is as follows:

```
// User is an abstract class
public abstract class Book {
    private String isbn;
    private String title;
    private String subject;
    private String publisher;
    private int numberOfPages;
    private int numberOfCopies;
    private BookFormat bookFormat;
    private List<Author> authors;
}

public class BookItem {
    private String id;
    private boolean isReferenceOnly;
    private Date borrowed;
    private Date dueDate;
    private double price;
    private BookStatus status;
    private Date dateOfPurchase;
    private Date publicationDate;
    private Rack placedAt;
    private Book book;  // Aggregation: BookItem has a reference to a Book

    // Constructors, getters, and other existing methods...

    public boolean checkout(String memberId) {
        // Implementation for checkout logic
        // Update the status, set due date, etc.
        // Return true if checkout is successful, false otherwise
    }
}
```

The Book, BookItem and Rack classes

Notification

The `Notification` class is another abstract class responsible for sending notifications to the users, with the `PostalNotification` and `EmailNotification` classes as its child classes. The implementation of this class can be found below:

```
// User is an abstract class
public abstract class Notification {
    private String notificationId;
    private Date createdOn;
    private String content;
    private BookLending bookLending;
    private BookReservation bookReservation;

    public abstract boolean sendNotification();
}

public class PostalNotification extends Notification {
    private Address address;

    public boolean sendNotification(){
        // definition
    }
}

public class EmailNotification extends Notification {
    private String email;

    public boolean sendNotification(){
        // definition
    }
}
```

Notification and its derived classes

Search and catalog

The `Search` is an interface used in the efficient searching of library books by various methods, and the `Catalog` class is used to implement the search interface to help in book searching. The code to perform this functionality is presented below:

```
public interface Search {
    // Interface method (does not have a body)
    public List<Book> searchByTitle(String title);
    public List<Book> searchByAuthor(String author);
    public List<Book> searchBySubject(String subject);
    public List<Book> searchByPublicationDate(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) {
        // definition
    }
    public List<Book> searchByAuthor(String query) {
        // definition
    }
    public List<Book> searchBySubject(String query) {
        // definition
    }
    public List<Book> searchByPublicationDate(String query) {
        // definition
    }
}
```

The Search interface and the Catalog class

Library

The final class of LMS is the `Library` class which will be a Singleton class, meaning the entire system will have only one instance of this class. The implementation of this class can be found below:

```
public class Library

    private String name;
    private Address address;

// Private constructor to prevent external instantiation
    private Library() {}

    public Address getAddress();

// The library is a singleton class that ensures it will have only one active instance at a time
    private static Library library = null;

// Created a static method to access the singleton instance of Library class
    public static Library getInstance ()
    {
    if (library == null)
    {
        library = new Library ();
    }
    return library;
    }
}
```

The Library class

Wrapping up

We've explored the complete design of a library management system in this chapter. We've looked at how a basic library management system can be visualized using various UML diagrams and designed using object-oriented principles and design patterns.