# Environment QA Engineer

## Practical Assignment

### 1. Assignment Description

In this assignment, you are required to design an Object-Oriented Program that implements a library management system with specific requirements. You need to submit the following

a. A system diagram that represents the classes, methods (functions) and the relation between the different classes of the program (You can find a diagram example for another program in the draft, kindly note that you are not required to submit a similar diagram, just use it an example not as a reference)

b. A code using your preferred programming language (Python – Java – C++), preferred python, with applying the needed object-oriented programming concepts.

Notes:

i. You don't need to submit a runnable program, if your code doesn't compile or has syntax errors you can still submit it.

ii. You don't need to test your code with a main function or any examples. Just submit the main classes.

iii. Do not bother googling the code. You will be asked to represent your work in details.

iv. You have to apply the object-oriented concepts in the code, any code that doesn't apply this will not be accepted.

## 2. Problem Description

### *a. Requirements:*

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.

## b. *Overview*

We have three main actors in our system:

i. **Librarian**: Mainly responsible for adding and modifying books, book items, and users. The Librarian can also issue, reserve, and return book items.
ii. **Member**: All members can search the catalog, as well as check-out, reserve, renew, and return a book.
iii. **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.

Note: Those are only the top use cases (methods or functions), there should be other use cases according to your design.

# Draft

An example of a diagram for designing a parking lot program.

The green boxes represent the classes

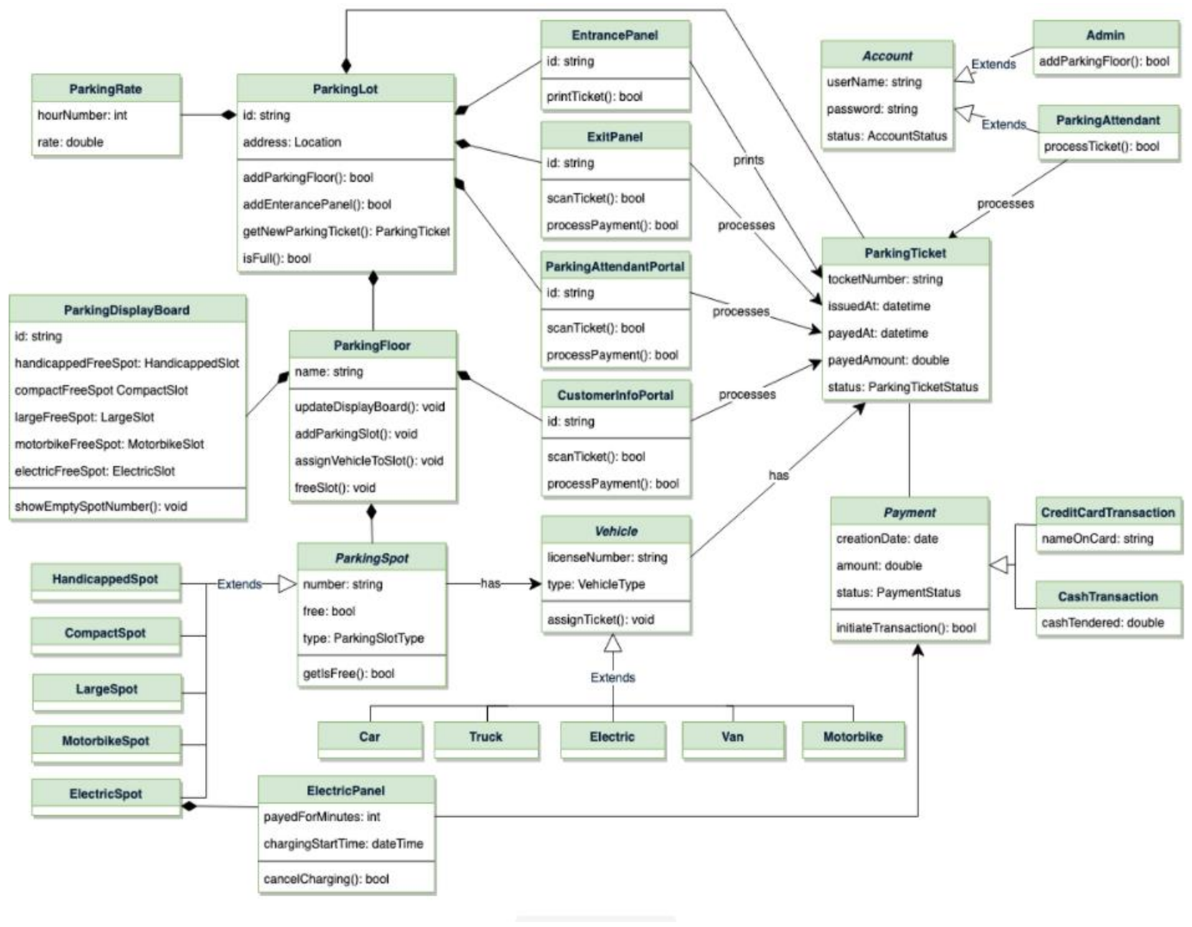
The white boxes are the class attributes (first section) and the class methods (second section)

Extends -> stands for inheritance

Id: String -> is an example of a class attribute

Processes -> means it modifies the class attribute

> For example, Exit Panel class has processPayment() that will make changes to Parking Ticket class attributes.



# Object Oriented Programming (OOP) Concepts

## Required:

- Inheritance

  (You have to implement your classes based on a certain hierarchy and an efficient inheritance)

## Preferred:

- Abstraction
- Polymorphism
- Encapsulation

  (Applying these concepts will only be counted as a bonus)