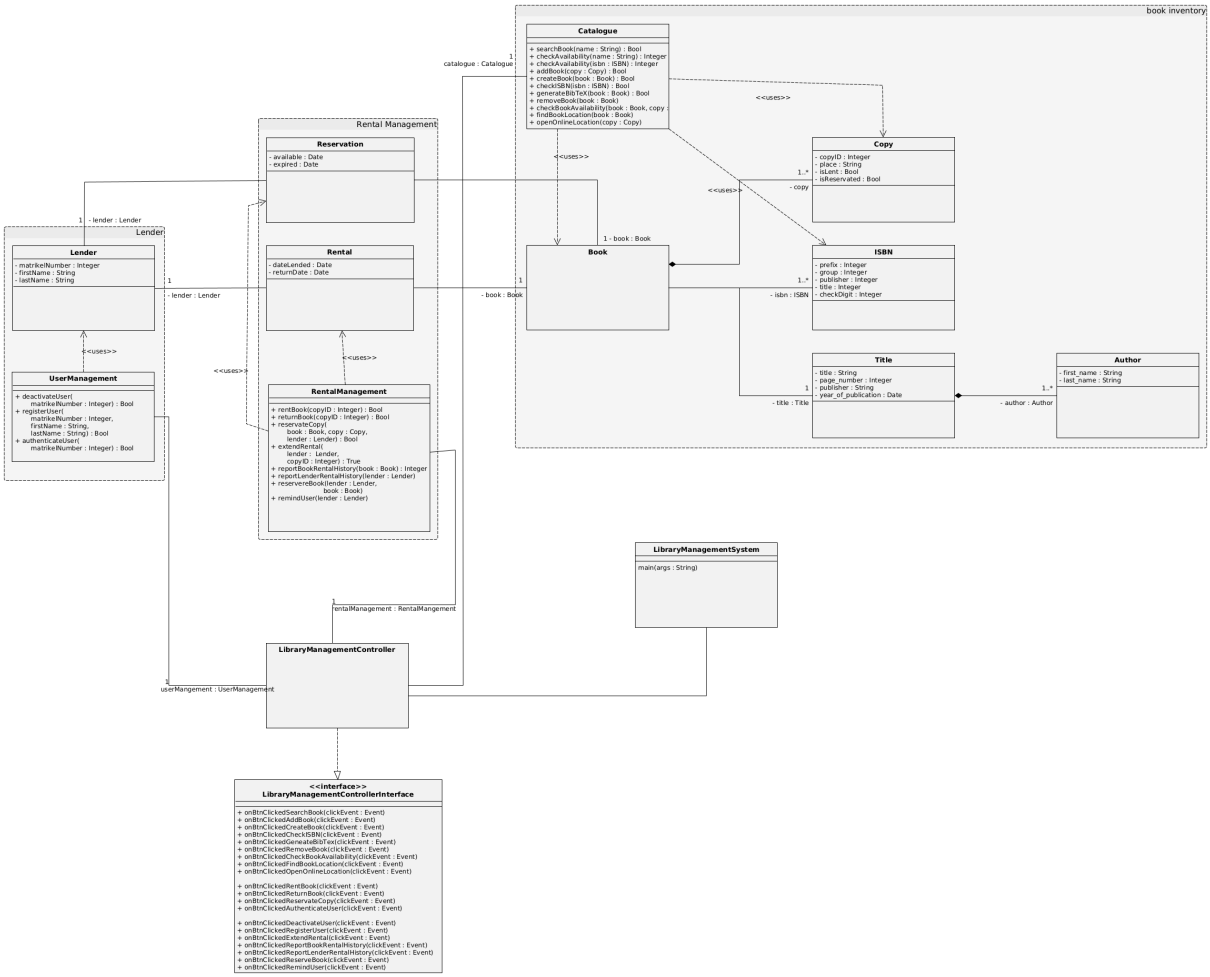
SWE Design - Sheet 3

Exercise 1



- The diagram includes the methods added in exercise 3 and the corresponding controller methods
- https://github.com/ValDavSto/SWE---Design/blob/main/Exercise%203/UML/uml.png
- The java files can be found in the appendix. Arguments were passed either with `0` or `null` to avoid errors. the events of class `Event` marked in the UML diagram were omitted, because they are normally part of the respective GUI frameworks.

UML Description

The UML diagram is based on the diagram from last exercise so only the description of the new class `LibraryManagementController` and the interface `LibraryManagementContorllerInterface`.

To adept our program structure to the GRASP approach we now added a new controller class. This class is called `LibraryManagementController` and implements the interface `LibraryManagementControllerInterface`. The controller connects our single management classes (`UserManagment, RentalMangment, Catalogue`) to the user interface. This can be seen in the methods

which have to be implemented. The methods all have a very similar structure, because they all wait for an
event to happen in our case it is a button click. When the event happens, the method in the controller will
read arguments from the user interface or database and will invoke a method from one of the management classes.

Here an example of the method naming of the methods from the `LibraryManagementControllerInterface`:

```
onBtnClicked<method to invoke>(Event clickEvent){}
```

Exercise 2

Catalogue

`searchBook`; `checkAvailability`; `addBook`; `createBook` ; `checkISBN`; `generateBibTex`

The class `Catalogue` is the **information expert** for the management of the whole library inventory. The `Catalogue` has the knowledge of all books in the inventory and an object `Book` contains all the information about the book and how many and which copies of a certain book exists.
All responsibilities which care about retrieving information's about books/copies or adding/removing
books/copies are assigned to the class `Catalogoue`. This is also the reason why it functions as **creator**
for objects of the classes `Book` and `Copy` which get created with the methods `createBook` and `addCopy`.
To be able to get a high **cohesion** and low **coupling** we include all methods which relate to the book inventory in `Catalogue`, the following methods fall under this responsibility:
`searchBook`, `checkAvailability`, `checkISBN`, `generateBibTex`

UserManagement

`deactivateUser`; `registerUser`;

The class `UserManagement` is the **information expert** for the management of the users using the LibSoft System. Because the class knows everything about the users using the LibSoft System
the responsibilities which are used to administering those users are a part of the class `UserManagement`.
One of those responsibilities is to register new users to the LibSoft System, that is why the method `registerUser`
is part of the class, with this method the class functions as a **creator** for objects of the class `Lender`.
To ensure high **cohesion** and low **coupling** also the method `deactivateUser` is included in the class `UserManagement`.

RentalManagement

`rentBook`; `returnBook`;

The class `RentalManagement` is the **information expert** for the management of the book rentals. The `RentalManagement` knows and tracks all the rented books and reservations and knows to which lender they are
assigned. Also the rental history of books and lender is known by the `RentalMangement`. All responsibilities which fall
under this area get assign to the class, which is for the method `rentBook` the case, which also make `RenatalMangement`
to a **creator** for objects of the class `Rental`.
The method `returnBook` is also included in the `LenderManagment` to fulfill high **cohesion** and low **coupling**.

Exercise 3

Catalogue

`removeBook`; `checkBookAvailability`; `findBookLocation`; `openOnlineLocation`

In Exercise 2 we already mentioned the resposebilitiey

As mentioned above the `Catalogue` class is the **information expert** for the management of the library's inventory and functions a **creator** of instances of the classes `Book` and `Copy`, which are created within the methods. As the methods `removeBook`, `checkBookAvailability`, `findBookLocation`, relate to the the management of the books and and the method `openOnlineLocation` to the management of copies they are part of the `Catalogue`. We enable **low coupling** and **high cohesion** by only taking methods into account which create instances of `Book` or `Copy` (methods like `reserveBook` which can also be seen as management of books were not added to the catalog if they require more objects besides book or copy which would destroy the principle of low coupling and high cohesion).

UserManagement

`authenticateUser`;

The class `UserManagment` is the **information expert** for the management of the users using the LibSoft System. The method `authenticateUser` is **creator** of the objects of the class `Lender`. As only relation to the Class Lender exist we fulfill **low coupling** here.

RentalManagement

`extendRental`; `reportBookRentalHistory`;`reportLenderRentalHistory`; `reserveBook`; `remindUser`

The class `RenderManagment` is the **information expert** for the management of the book rentals. To achieve low **coupling** and high **cohesion** the methods `extendRental`, `reportLenderRentalHistory`, `reserveBook`, `remindUser` are not part of the `UserManagement`, as the class `RentalManagement` already uses the object `Lender` of the class `Rental`. So `RentalManagement` does not directly serves as a **creator** for an object of `Lender` but still has knowledge the object. If they would be part of the Class `UserManagement` we would have a higher coupling as we needed an additional object of `Rental` within the `UserManagement` class. Furthermore the method `reportBookRentalHistory` functions as an **creator** of an object of the Class `Book`.