Code review

Overview

The part of code I decided to post for the review is the booking system of our project. Part of it, to be precise. It contains all the logic of booking, return and renew systems. Our projects uses Java and Spring.

Goals

The code itself is working fine, however, you may find bugs in it, or ways to make it better. Can't wait to see your suggestions.

Usage Scenarios

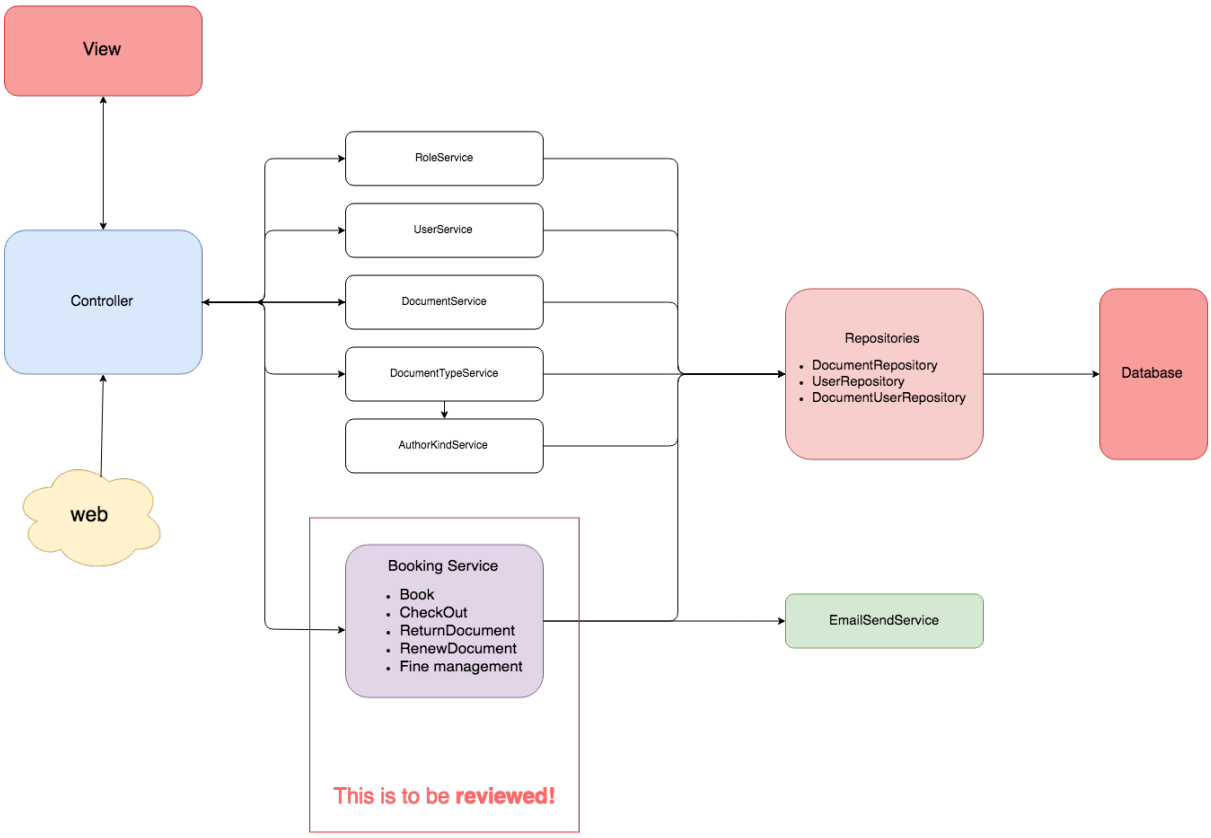
The class itself is used when handling operations such as booking and returning.

Book method is used when a user wants to book a document.

Return - guess what? When a user wants to return a document.

Renew a document is also an action implemented in bookingService.

Class Diagram



Code

Part to review:

Also, a full code of bookingService here:

Full project here:

CatReviewer6ies for review

1. Design decisions

Example:
[1.1] NameTTT: You should use HASH_TABLE for 'yyy' instead of ARRAY.
[1.1.1] Owner: the decision of using ARRAY is based on the fact that ARRAYs are lighter than HASH_TABLES
[1.1.2] NameTTT: That might be true, but you are compromising execution time, 'yyy' is used to perform several searches. Having a HASH_TABLE one can have constant complexity in searching
[1.1.3] Owner: Ok, I will change the design and implementation. [CLOSED]

2. *API design*
3. Architecture (including inheritance hierarchy, if any)
4. Implementation techniques, in particular choice of data structures and algorithms
5. Contracts
6. Programming style, names
7. Comments and documentation

1. Design decisions

[1.1] Reviewer1: In the method maxWeeksNum you should use constants instead of particular numbers

[1.1.1] Owner: Good. Will be fixed. [FIXED]

[1.2] Reviewer2: I think it would be better if the number of maximum weeks (from method **maxWeeksNum**) for each type of Patron will be in the class User. But you should remain this method in the Booking Service class just to check whether the book is reference, bestseller or otherwise.

[1.2.1] Owner: Unfortunately, MVC structure of our project does not allow to build any logic in **model** classes such as *user* class. Therefore, we are to include this method into the *bookingService* class. However, if you have an idea where else to put it, please report. Otherwise, please mark this thread as closed.

[1.2.2] Reviewer2: You may put it as a field of User class.

[1.2.3] Owner: We have a complicated system of roles in our project, so *User* class doesn't actually have anything related to his status. Thus, it could be put as a field to a *Role* class instead. I will think it over. Thank you, anyway. [IMPLEMENTED]

[1.3] Reviewer2: In method **availableCopies** you can combine case statement.

[1.3.1] Owner: Fixed. Thank you for the suggestion! [IMPLEMENTED]

[1.4] Reviewer1: Communication with a user should not be in services, so all strings like “oh man, what are you doing with me” are extremely fun, but should be in servlets or somewhere in web, not here

[1.4.1] Owner: So what you’re saying is that we should handle formally-expressed exceptions informally only in the front-end and keep our real exceptions hidden from the user himself, is that right?

[1.4.2] Reviewer1: i am saying that any communication with user should be in the front, but services we are reviewing are definitely back end.

[1.4.3] Owner: I need to think it over. Probably [TO_BE_IMPLEMENTED]

2. API design

3. Architecture (including inheritance hierarchy, if any)

4. Implementation techniques, in particular choice of data structures and algorithms

[4.1] Reviewer4: in the method public int availableCopies(Document document) you calculate the number of available copies in this way: firstly you find all “taken” and “renewed” documents and then calculate the result. In generally, your implementation of this method works in O(amount of copies of this document/or amount of all copies in your database). But you might store one more field for available copies and in this case this method will work in O(1).

[4.1.2] Owner: Great suggestion, thank you for making our project run faster. [IMPLEMENTED]

[4.2] Reviewer1: Why somewhere you use Calendar() instead of Date()?

[4.2.1] Owner: In general, *Date()* is almost all deprecated, so the question probably should be vise versa, why use *Date* instead of *Calendar*. Maybe we will switch to *Calendar*, but as for me, *Date* has some advantages in the ease of use. Please comment more on that issue.

[4.2.2]Reviewer1: It is not an issue, i was just interested in why did you decide to use different classes for time. Basically question is [CLOSED]

[4.3] Reviewer4: from the method private boolean checkIfAvailableToCheckOut(Document document, User user) I understood that you store all users for a document in one array (users, which already booked the document, and users, which are in queue). In this case you store the status for an user(“new” and so on). Because of that in this method you find all users with status “new” and after that compare their priorities. In this way it is better to divide users in two parts. And after that you will be able to use Priority Queue for waiting

list(which works much faster in most cases than list, ex: removing user from the list).

[4.3.1] Owner: Unfortunately, we can't make changes to the **model** classes, like division the UserDocument class into two dynamically-exchanging ones. One reason for that is that this would force us to move entities from one table to another. [TO_BE_IMPLEMENTED]

[4.4] Reviewer5: It seems that in your method checkOut() user is removed from queue only when he performs some action and not automatically. Is there any way to fix it? It appears that a user can become stuck on the first place of the queue if he does not do anything. It is better to have an automatic method for removing users from queue (efficiently removing human factor of not logging in for days).

[4.4.1] Owner: I checked, it's actually in our "need to implement asap" list. Thank you for the reminder. [TO_BE_IMPLEMENTED]

[4.5] Reviewer5: In your method public void renewDocument (`Document document, User user, Date date`) you initialize the variable int fine and it does not go anywhere.

[4.5.1] Owner: In future (a week later :D) it will be pushed to the frontend, probably. That's why it is now unused. [CLOSED]

5. Exceptions handling - Contracts

[5.1] Reviewer5: You're not using OOP principles when handling exception, because every new type of exception is better to be declared as a separate class and not a base Java exception class.

[5.1.1] Owner: I think the code will just become unclean with exception classes, which all differ just by their message.

[5.1.2] Reviewer5: Okay, I will give some points for your better understanding:

1. You can throw your custom exceptions every time you need them, so it would improve reusability of code.
2. If you introduce custom exception classes, your code will generally be more readable because there will be less constructors with long strings as parameters. If you think that your code will become unclean after replacing several lines without adding new ones, then it is already unclean.
3. It is just a good practice to do it. [TO_BE_IMPLEMENTED]

[5.2] Reviewer5: The text of exceptions a-la "Trying to cheat?" is too informal.

[5.2.1] Owner: It is because we send them directly to the front-end, so that user sees them. It may be a wrong approach, let us discuss this at **[4.1]**

[5.2.2] Reviewer5: It is not HOW you send it, it is WHAT you send I am talking about. The text is too informal and not professional in any way for a project like this. Such comments would be appropriate at school projects and not in a library management system.

[5.2.3] Owner: I get your point, however, it is not unusual in serious projects to be informal when something goes wrong. Just take a look at crashes in Google ("Oooops, something went wrong"), and in Chrome (dinosaur game, not a boring error message). There are multiple examples of such services.

[5.2.4] Reviewer5: It appears to me that your method checkOut() can only be used by Librarian. So the question is, why do your messages (in exceptions) look like they are targeted to User (I mean, you addressed them exactly to User)? And if the statement above is true, and checkOut() is one of the Librarian action, so is it adequate to send messages in such informal way?
If your method can be used by Users, so what is the purpose of it?
And your comparison with Google was inappropriate ~~(because of reasons)~~ as Google can afford to be awesome because it is already at the top. If you don't start seriously, you will never get to that level.

[5.2.5] Owner: Okay, so that confirms that the formal/informal messages is a purely subjective criteria, therefore you can leave it to us to decide. As for the *checkOut()* message, [FIXED]

[5.3] Reviewer6: The text of exception "Cannot return the document. Pay a fine first." is wrong. It should look like this:"It is impossible to return the document. Pay the fine first.". [FIXED]

6. Programming style, names

[6.1] Reviewer2: The number of characters in string is more than 100 that makes your code less readable.

[6.1.1] Owner: Will be fixed. Thank you. [FIXED]

[6.2] Reviewer2: Name of method `availableCopies` seems like it will return the list of available copies (if reader of code doesn't know that copies just of one book)

[6.2.1] Owner: (Tell me if I got your comment wrong) It takes an instance of document as an argument, so that isn't an issue, I suppose.

[6.2.2] Reviewer2: I mean that it would be better if you rename the method such that it is clear that it will return number, not the copies. For example, amountOfCopies or something like that.

[6.2.3] Owner: We will try to think of a better name. Thank you.
[TO_BE_FIXED]

7. Comments and documentation

[7.1] Reviewer5: All methods could really have better JavaDocs (or you could add comments), as just restating the name of the method (e.g. checkOut() - method for checking out the document) is not enough for understanding how exactly method works.

[7.1.1] Will be fixed. Thank you. [FIXED]