Code review

InLibrary Manager

Overview

This is a part of Patron.java class which is responsible for main patron actions, storing and transferring patron data.

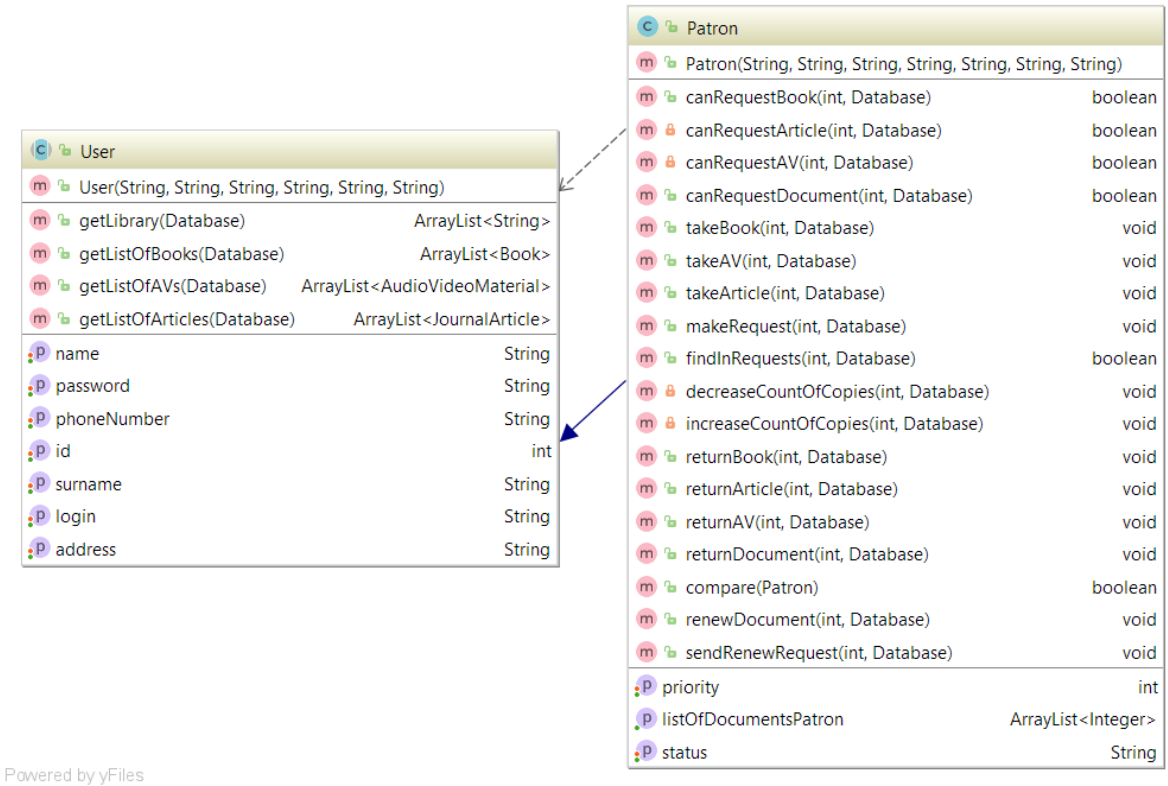
Goals

Clean the code, make it more readable and effective.

Usage Scenarios

Getting patron from database, requesting, renewing and returning documents

Class Diagrams



Code

Categories for review

1. Design decisions

[1.1] Reviewer1: You shouldn`t hardcode numbers and strings into your algorithms, in case you want to change something you will have to go through all your code and change this value. Instead, you could create a constants file and store all those values there.
[1.1.1] Owner: Could you please, give me an example of strings or constants?
[1.1.2] Reviewer1: Every time you use SetTime function there is a multiplication of some numbers or names of columns when you access your database.
[1.1.3] Owner: Got it. **[TO BE IMPLEMENTED]**

2. API design

3. Architecture (including inheritance hierarchy, if any)

[3.1] Reviewer1: Checking if strings are equal in order to get UserType makes code much less readable and not OOP. It would be better if you used dynamic binding technique and just called method canRequest or takeBook for some 'User' object which inherits all UserTypes, so correct version of method would be called.
[3.1.1] Owner: Patron type is a field in our system. It would be hard enough to inject new classes at the current stage.
[3.1.2] Reviewer1: I guess you can at least use enumerators then. But next time you should expand inheritance tree, since it is obvious that checking out a book (for example) is different for different UserTypes.
[3.1.3] Owner: OK, we're currently working on replacing strings with enumerators. **[TO BE FIXED]**

[3.2] Reviewer1: Instead of writing separate method for book every time, you could use dynamic binding technique by implementing methods for checking canTake condition or for getting expireDate and just call it for some 'Document' object which inherits all other DocumentTypes.
[3.2.1] Owner: It might produce a lot of changes, so we cannot quickly fix this issue. Nevertheless, it is a good suggestion and I think it's better to discuss it later. **[TO BE DISCUSSED]**
[3.2.2] Owner: We will make a general method for booking. **[TO BE IMPLEMENTED]**

[3.3] Reviewer2: You should separate your logic from the working module. It would be more readable and more convenient to change the condition of booking and returning systems.
[3.3.1] Owner: What do you mean by "logic"?
[3.3.2] Reviewer2: The requirements to booking the documents and returning them.
[3.3.3] Owner: Got it. We are currently working on it. **[TO BE IMPLEMENTED]**

[3.4] Reviewer3: It is better not to output system information to the console, but store this data in log or (and) send notifications to users.
[3.4.1] Owner: Can you suggest any logging frameworks?
[3.4.2] Reviewer3: You can use java.util.logging, Log4j 2.x or Logback.
[3.4.3] Owner: We are working on logging system now. We'll try some of these, thank you. **[TO BE IMPLEMENTED]**

[3.5] Reviewer4: It was noticed that you do not have "information hiding". That is, all basic database operations are performed directly in the "Patron" class. I propose to make a special intermediate file, which will connect this class with the database class simultaneously and all requests will be stored exactly in it. Eventually, in the Patron class, for example, the "Renew" function, you can write a simple condition for checking the user and refer to the intermediate file calling as "`<IntermediateFileName>.UserDocumentRenew(<certain parameters>)`".
[3.5.1] Owner: *Database* class is exactly what you mean. It was designed to separate SQL routines from the other parts of system. **[CLOSED]**

[3.6] Reviewer5: I would suggest to use enum types in Patron.status for easy code understanding and for avoiding string comparison.
[3.6.1] Owner: We are working now on replacing strings with enumerators. **[TO BE IMPLEMENTED]**

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

5. Exceptions handling - ~~Contracts~~

[5.1] Reviewer1: I believe that default case (takeBook) should throw an exception since such userType is not exist which means that there is a mistake somewhere else.
[5.1.1] Owner: Fixed. **[TO BE FIXED]**

[5.2] Reviewer2: Incorrect processing exceptions. You should process the exceptions on the current level (you should not pass them to the upper levels).
[5.2.1] Owner: We are currently working on cleaning up exception handlers. We will move SQLException handlers to lower level, but we have to handle NoSuchElementException on current level.
[5.2.2] Owner: Fixed. **[TO BE FIXED]**

[5.3] Reviewer6: I think, that in AV material, you set time for week, if status is "vp", maybe I don't understand clearly, what you mean by "vp", but if it is patron, I believe, that there should be 2 week in any cases.
[5.3.1] Owner: Visiting professors allowed to take AVs only for 1 week. **[CLOSED]**

[5.4] Reviewer4: There are a lot of “try-catch” statements in this code. Therefore, you can notice that the program can crash, and this is bad idea in programming. Usually it is used as a temporary measure against the bug, which in the future is eliminated. It would be better to use the standard “if-else” operator against the current and “throw” to raise an exceptions.

```
if (<condition>){
      <certain operations>
}
else {
      throw(SqlExceptionId)
}
```

[5.4.1] Owner: We usually work with methods which operates with database. SQLite JDBC methods have “throws SQLException” in their signature, so finally we have to use try/catch constructions somewhere. However, we know about this problem and working now on cleaning up exception handlers.
[5.4.2] Owner: We have cleaned up most of useless exceptions and try-catch statements, but we want you to provide us with some examples. I suppose, we need to discuss this. **[TO BE DISCUSSED]**
[5.4.3] Owner: We moved exception handling to the Database class level, so now we don’t need to use try-catch statements everywhere. **[TO BE FIXED]**

[5.5] Reviewer5: It is better to use System.err.println() for output in exception handling.
[5.5.1] Owner: We are implementing more advanced logging system instead of writing in console. **[TO BE IMPLEMENTED]**

6. Programming style, names

[6.1] Reviewer6: Not understandable name “vp”, maybe there should be “patron”
[6.1.1] Owner: “vp” used for *visiting professor*. **[CLOSED]**

[6.2] Reviewer6: Instead of “Not copies” “No copies”, it should be correctly.
[6.2.1] Owner: Fixed. **[TO BE FIXED]**

[6.3] Reviewer6: Mistake in takeBook, “28L” instead of just “28”
[6.3.1] Owner: “28L” suggests Java to use *long* type instead of just *int*. Thus, we have to keep “28L”. **[CLOSED]**

[6.4] Reviewer3: What are date and date2 variables in takeAV method? It is not clear what they are intended for.
[6.4.1] Owner: Fixed. **[TO BE FIXED]**

[6.5] Reviewer5: Calling variables “documentId” instead of “idDocument” is more syntactical and more correct.
[6.5.1] Owner: Fixed. **[TO BE FIXED]**

7. Comments and documentation

[7.1] Reviewer3: There is no documentation in canRequestDocument and takeAV methods.
[7.1.1] Owner: There is documentation for these methods in full code. I just did not include documentation into frozen snippet. **[CLOSED]**

[7.2] Reviewer4: There are no comments in the code. It's about the comments inside the function. For instance, in the "canRequestBook" function there are lengthy conditions, and it is very hard to recognize their meaning. It is necessary to leave small comments so that make reading easier and understand what is happening here. Moreover, it will be useful for you after a long time.
[7.2.1] Owner: We have got your suggestion. We are going to add comments as soon as possible. **[TO BE IMPLEMENTED]**

[7.3] Reviewer5: “No patron registered with ID” sentence is incorrect. “There is no registered patron with such ID” looks more proper.
[7.3.1] Owner: Fixed. **[TO BE FIXED]**