SQE — Assignment 1:
Unit-Test Library-Management System

1. Introduction

Your task is to create comprehensive unit tests for the *Library* class in a Library Management System. You should aim for exhaustive test and mutation coverage. To achieve this, you will use the JUnit5 framework and Mockito to isolate the Library class.

2. Goals

Practice unit testing, and more specifically:

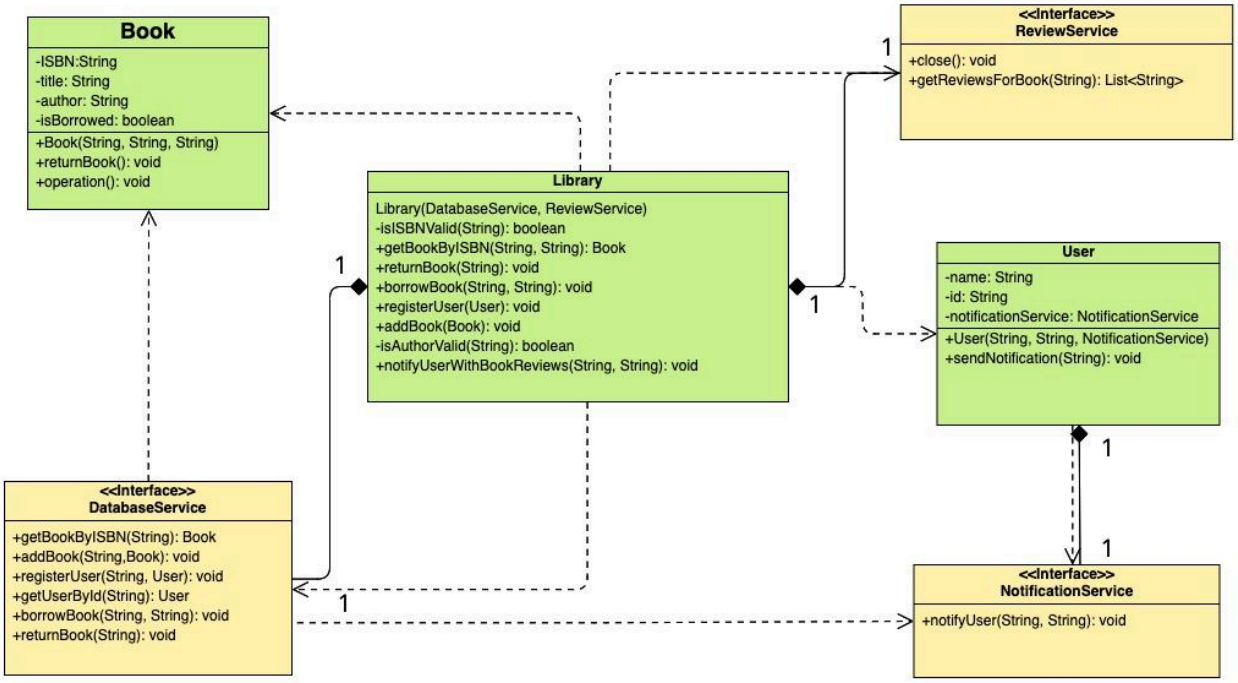
1. Deciding what to test and how to test
2. Isolation
3. Stub/Mock/Spy
4. Test coverage

3. Backstory

In a small town, the local library has decided to modernize its operations by transitioning to a digital management system. Your role is to ensure the reliability and functionality of the *Library* class within this system. The Library Management System integrates several components, including *User* and *Book* classes and services like *DatabaseService*, *NotificationService*, and *ReviewService*.

4. Architecture

Class diagram (a bigger version of the image below)



Classes

- **Library**: Acts as the central orchestrator for all the operations related to the library.
- **User**: Manages attributes and behaviors related to users.
- **Book**: Manages attributes and behaviors related to books.

Services

- **DatabaseService**: Responsible for database interactions.
- **NotificationService**: Sends notifications to users.
- **ReviewService**: Manages book reviews.

Exceptions

- **BookAlreadyBorrowedException**
- **BookNotBorrowedException**
- **BookNotFoundException**
- **NoReviewsFoundException**
- **NotificationException**
- **ReviewException**
- **ReviewServiceUnavailableException**
- **UserNotRegisteredException**

5. Testing Requirements

1. Implement your unit tests using the JUnit5 framework within the *TestLibrary.java* file.
2. Utilize Mockito to isolate the *Library* class during testing.
3. Achieve exhaustive test and mutation coverage to ensure the robustness of the *Library* class.
4. Use the testing naming convention studied in class.
5. Use test annotations correctly (e.g., @BeforeEach, @ParameterizedTest, etc.).
6. You may change only the TestLibrary.java file.

6. Assignment Files and Structure

Download the Assignment files from here (or from Moodle).

```
├── pom.xml - [Project configuration file for Maven]
├── src
│   └── main
│       └── java
│           └── ac
│               └── il
│                   └── bgu
│                       └── qa
│                           ├── Book.java
│                           ├── Library.java - [Class Under Test]
│                           ├── User.java
│                           ├── errors [Exception Classes]
│                           │   ├── BookAlreadyBorrowedException.java
│                           │   ├── BookNotBorrowedException.java
│                           │   ├── BookNotFoundException.java
│                           │   ├── NoReviewsFoundException.java
│                           │   ├── NotificationException.java
│                           │   ├── ReviewException.java
│                           │   ├── ReviewServiceUnavailableException.java
│                           │   └── UserNotRegisteredException.java
│                           └── services [Service Interfaces]
│                               ├── DatabaseService.java
│                               ├── NotificationService.java
│                               └── ReviewService.java
└── test
    └── java
        └── ac
            └── il
                └── bgu
                    └── qa
                        └── TestLibrary.java [Deliverable Testing Class]
```

7. Setup and Execution

1. A **pom.xml** file is included in the repository, configuring all the necessary dependencies. You should not change this file, as it can affect the execution of the automatic evaluation upon submission.
2. To run your tests, execute *mvn clean test* with **Java 17** or higher and the latest Maven version installed.

8. Testing Your Test Suite

We will test the quality of your tests using mutation testing, test coverage, and more.

To verify that we can run your tests, we have prepared a smaller version of our evaluation program, with only a few tests. To run it download assignment1.jar file (can be downloaded from Moodle as well) and then run the following command in shell:

```
java -jar assignment1.jar -t <relative/absolute path TestLibrary.java>
```

This requires **Java 17** or higher.

9. Submission Constraints

- **Deliverables**: Only the TestLibrary.java file should be uploaded on Moodle. All other project files will be provided by us. Therefore, you should not change the POM file or the source classes as it can affect the execution of the automatic evaluation upon submission.

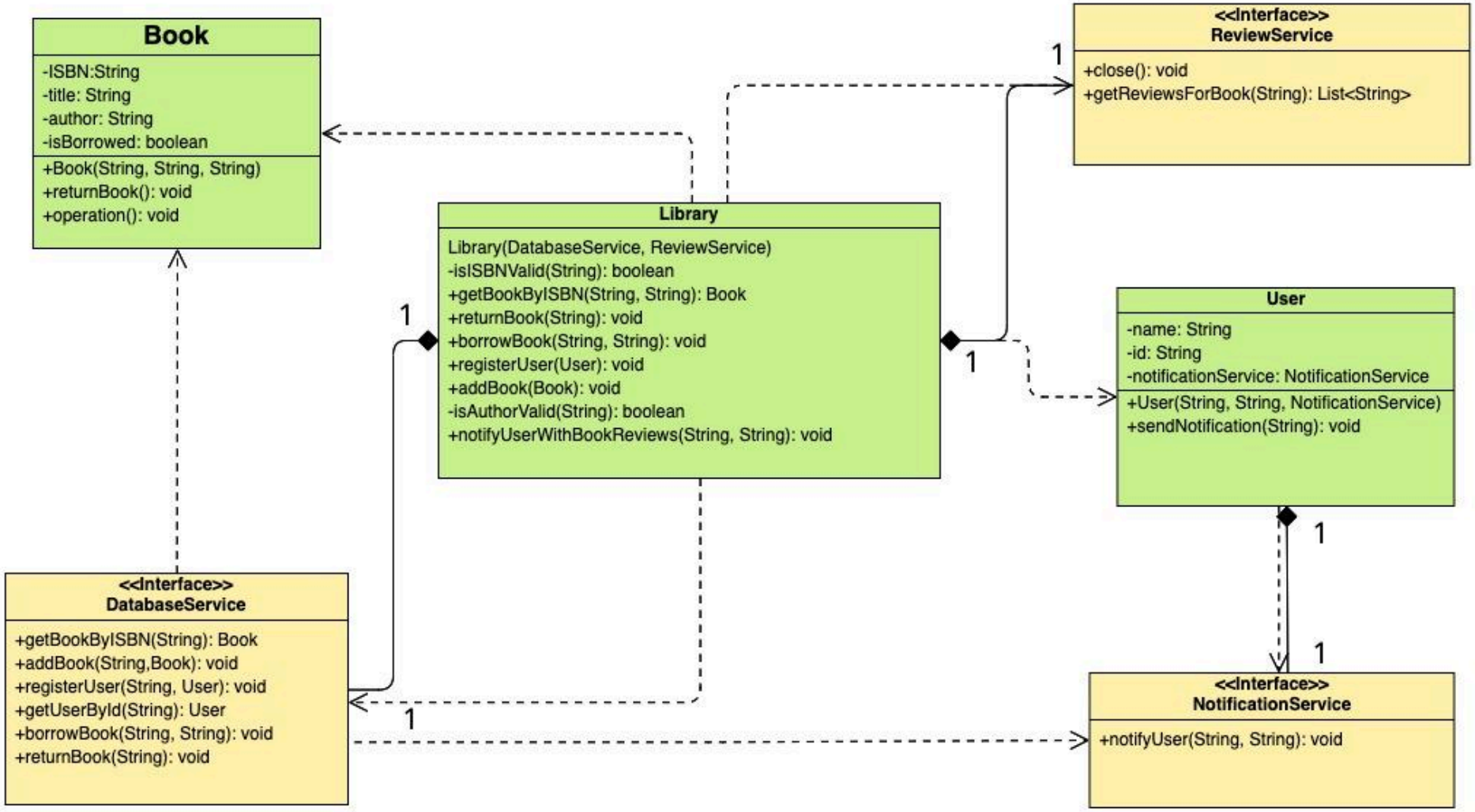
10. Evaluation Criteria

Your submission will be assessed based on the following:
- **Test and Mutation Coverage**: Your aim should be for exhaustive coverage to ensure the Library class's robustness.
- **Code Quality**: The quality and understandability of your code, including the clarity of your annotations and comments.
- **Code Conventions:** We will check that you use the coding conventions taught in class (see the first practical session for more details).
- **Proper Use of Mocks and Stubs**: The correct and effective use of mocks and stubs to isolate the Library class.

11. Deadline

The TestLibrary.java file should be submitted through Moodle by **25/01/2024 23:59**.

GOOD LUCK


Book
-ISBN:String
-title: String
-author: String
-isBorrowed: boolean
+Book(String, String, String)
+returnBook(): void
+operation(): void
<<Interface>>
ReviewService
+close(): void
+getReviewsForBook(String): List<String>
Library
Library(DatabaseService, ReviewService)
-isISBNValid(String): boolean
+getBookByISBN(String, String): Book
+returnBook(String): void
+borrowBook(String, String): void
+registerUser(User): void
+addBook(Book): void
-isAuthorValid(String): boolean
+notifyUserWithBookReviews(String, String): void
User
-name: String
-id: String
-notificationService: NotificationService
+User(String, String, NotificationService)
+sendNotification(String): void
<<Interface>>
DatabaseService
+getBookByISBN(String): Book
+addBook(String,Book): void
+registerUser(String, User): void
+getUserById(String): User
+borrowBook(String, String): void
+returnBook(String): void
<<Interface>>
NotificationService
+notifyUser(String, String): void
1
1
1
1
1
1