Learning goals

Before the next day, you should have achieved the following learning goals:

- Strengthen your knowledge of JUnit4, practicing the use of the new annotations.
- Understand the Test-Driven Development methodology.

1 Practicing TDD:

Write a simple application for keeping track of the books in a library. The functionality will be described incrementally by stages. At every stage, you must follow the following steps:

1. Write (or update) the interface(s) for the new functionality required.
2. Generate the tests for the functionality required at that stage. The tests may not even compile at this point.
3. Write the bare minimum code of the classes implementing the interfaces to make the test-class(es) compile. Check that the new tests fail while the old ones pass.
4. Write the minimum code in the methods of the implementing class(es) that makes all the new tests pass.
5. Refactor the code to make it clearer, if needed. Your first implementation may not be as clear as possible. Think of the next programmer that will come after you: will they understand the code easily? Are variable names clear and descriptive? Can you simplify those `for` loops and/or those `if...else` branches?
6. Document the code if it has not been done yet. Update the JavaDoc documentation (on a separate `www` folder) using the command `javadoc`. Use your browser to check that it works and is easy to read.

Note: Remember than the most usual convention in Java about interfaces and implementing classes is that interfaces get the "normal" name, e.g. interface Book and class BookImpl. Some times, especially in code written by older developers, you will find the opposite convention, e.g. interface IBook and class Book (or, worse, BookInterface and Book). You should be aware of this older convention (and follow it when maintaining legacy code for the sake of coherence) but you should not use it for new code.

1.1

Create a class for books. Books have authors and titles. The class should implement getters for both author name and title, but these must be set at construction time and never be modified afterwards[1].

1.2

Create a class for the users of the library. Users have a name and a library-ID (an `int`), both of which must be unique in a library. The name is set at construction time, but the library-ID is not (more on this on 1.3 and 1.4). The class must implement methods to get the name of the person and their ID.

1.3

Expand the class you have just created to allow users to register with a library. You will need two methods `register(Library)` and `getLibrary()`. The former method is the way to obtain the user-ID, i.e. when a `User` registers with a `Library`, the library returns an `int` which is the ID of this user with this library (more on this on 1.4).

Important. As you do not have a Library class yet, you will need a fake Library object to test your method `register(Library)`. This is called a *mock* object, and it is a common practice when writing testing code because it allows the programmer to test one class at a time —instead of testing several classes at the same time, which is more complex and thus error-prone—. The mock library object does not need to do anything apart from providing a name (so that your class can return it when you call `getLibrary()` and an ID when you call `getID()`. Because it is a mock object and not the real one, it can return trivial values (i.e. the name can be always "Library name" and the ID can be always 13).

[1] An object whose fields cannot be changed after construction is called an *immutable* object.

1.4

Create a class to represent libraries. In addition to all we know about libraries already, libraries also have a name that is set at construction time. They also have a "maximum number of books borrowed by the same person" policy (e.g. max three books per user), which can be updated at any time. Of course, they also have a method to get the maximum number of books to be borrowed at any time (e.g. `getMaxBooksPerUser()`).

1.5

Now add to `Library` a method `getId(String)` that returns the ID of a person for a given name in this library.

If the person does not have an ID yet, a new *unique* ID is created and returned. Any subsequent calls to this method with the same name argument should return the same ID. This is the method that is used to register a user with a library.

Note: This method may already be in your `Library` interface (if you put it there in 1.3). If that is the case, its implementation in `LibraryImpl` and/or `LibraryMock` is probably a bare-bones implementation that does not do much. Now is the time to test and implement this feature.

1.6

Expand your library class a bit further. Add three new methods:

addBook(String title, String author) Adds a new book to the list of books in this library.

takeBook(String title) If the book is not taken, marks the book as taken and returns it. If the book is taken or does not exist, null is returned.

returnBook(Book book) Marks the book as not taken.

Important. At this point you have to make a *design decision*: should the information about the books "being taken" be stored in the book class or in the library class? If this information is part of the book, you must add some functionality to the Book class: maybe methods like `isTaken()` and `setTaken(boolean)`, and some private boolean field; if the responsibility lies on the library's camp, then you must add the adequate memory structures.

1.7

Expand your library once more to include the following methods:

getReaderCount() returns the number of users registered in this library.

getBookCount() returns the number of books in this library.

getBookBorrowedCount() returns the number of borrowed books in this library.

1.8 *

Add a method to your users that returns a list with the titles of all the books they are borrowing at the moment.

1.9 *

Expand your library class one more time with methods that return:

- a list with all the users that are borrowing books at the moment.
- a list with all the users, borrowing or not.
- the name of the person that is borrowing a specific given title at the moment; if nobody is borrowing the book, or the book does not exist in the library, you must return null (not an empty String).

Note: You may need to make some modifications to the interface of the library to be able to record who is borrowing what. If in doubt, remember that it is better to use only one operation (i.e. method call) to borrow a book instead of two.

1.10 *

Modify your method `setMaxBookPolicy(int)` in your library class, so that it does not return `void` anymore. Instead, it should return an array of users that have more books than the new policy allows. The array may be empty.

For example, if the maximum policy is three books per user, Marge, Lisa, and Maggie may have borrowed one, two, and three books. If the policy is then set to a maximum of one book, the method must return the names of Lisa and Maggie (so that the library can track them down and ask them to return the excess books).

1.11 *

Once you have finished implementing all this functionality, write a small script that illustrates these situations:

- A user borrowing one book and then returning it.
- A user trying to borrow more books than allowed. And then returning one of the books they already have to borrow a new one.
- Several users borrow books. The library is then asked who has some specific title, if anyone.
- The "maximum books" is reduced to one, then to zero, then it is increased to the original value.

1.12 *

At the end of the implementation, you may have found yourselves in situation where you thought a different interface would have worked better. Maybe books should be identified by IDs and not by titles? Maybe users should be able to register with more than one library? Make notes about your ideas and discuss them with a member of the faculty.