Data Structure Design

PR2: Practice 2

Introduction

PR2 involves a programming exercise that will be reusable for Practice 3 (PR3), where you will apply the knowledge acquired. Specifically, in this activity, we will focus on implementing the data structures defined and designed with the information volume restrictions outlined in the CAA1, enabling us to store information about **The library and its loan management.**

Previous considerations

It is mandatory that the coding of this exercise is based on the data structures defined in the CAA1 solution and that it is mandatory to use the data structures offered in the course's ADTs library. Although it could be implemented using Java collections, their use in this exercise will be penalized. If any additional structure is necessary for any of the functionalities, it can be added with a justified explanation.

Please note that to pass the course, you need a final practical grade (P) of 5 or higher (P = 10% PR1 + 35% PR2 + 55% PR3). Additionally, the activity must be completed individually. Any lack of authenticity in authorship or originality; copying or plagiarism; fraudulent attempts to obtain a better academic result; collaboration, concealment, or facilitation of copying; or the use of unauthorized materials or devices during the evaluation, among other irregular conducts, can have serious academic and disciplinary consequences.

AI use is not allowed

In this activity, the use of artificial intelligence tools is prohibited. In the teaching plan and on the UOC's website on academic integrity and plagiarism, you will find information about what constitutes irregular behavior in assessments and the potential consequences.

Statement

In PR2 we will work with the following entities and concepts:

- **Reader**: Library client
- **Loan**: Loan of a book to a client
- **Book**: Book stored in the library
- **CatalogedBook**: Cataloged books in the library
- **Workers**: Library workers

functionalities

The functionalities required for PR2, already specified in CAA1, are:

- addReader(readerId, name, surname, IdDoc, birthDate, birthPlace, address)
- addWorker(workerId, name, surname)
- lendBook(loanId, readerId, bookId, workerId, date, expirationDate)
- givebackBook(loanId, date)
- storeBook(bookId, isbn, title, editorial, publicationYear, edition, author, theme)
- catalogBook(workerId)
- timeToBeCataloged(bookId, lotPreparationTime, bookCatalogTime): Integer
- getAllLoansByReader(readerId): Iterator
- getAllLoansByState(readerId, state): Iterator
- getAllLoansByBook(bookId): Iterator
- getReaderTheMost(): Reader
- getMostReadBook(): Book

Additionally, new operations have been defined that allow inspecting the data structure and validating the test sets:

- getReader(String id): Reader
- getBook(String id): Book
- getLoan(String id): Loan
- numReaders(): int
- numBooks(): int
- numLoan(): int
- …

All the operations required in PR2 are defined in the **Library** interface (Library.java). Additionally, some constants are provided to define the bounded containers and enumerations.

Start-up of the project

To develop this practice, a base project is provided along with the statement, which has the following structure:

- readme.md: A text file in Markdown format included in the project to provide information about it.

- src/main/java/uoc.ds.pr. **Library.java**: Interface that specifies the ADTs operations to implement.
- src/test/java/uoc.ds.pr. **FactoryLibrary.java**: Class that implements the factory design pattern[1] and that initializes the data structures with initial values.
- src/test/java/uoc.ds.pr. **LibraryPR2Test.java: Library** ADTs test class.
- src/test/java/**/: Additional test classes to assist in implementing the project
- src/test/java/uoc.ds.pr.util. **DateUtil.java**: A utility class for working with dates.
- Classes pending implementation:
    - **LibraryPR2Impl**, which implements the interface defined in **Library** and the model entities.
    - Additionally, all exceptions defined in the **LibraryPR2Impl** interface must be implemented, which must inherit from the **DSException class** provided in the statement.

The unit tests (**LibraryPR2Test** .java and **FactoryLibrary** .java files) have been implemented with the JUNIT 4 framework [2].

JUNIT uses annotations (https://docs.oracle.com/javase/tutorial/java/annotations/) which are metadata that provide additional information about classes or parts of classes. Methods that have the following JUNIT annotations will have the following responsibility:

[1] https://en.wikipedia.org/wiki/Factory_Method_(patr%C3%B3n_de_dise%C3%B1o)
[2] https://github.com/junit-team/junit4/wiki/Getting-started

- @BeforeClass – Will always be executed once when the class **is initialized.**
- @AfterClass - Will be executed after **all** tests are executed.
- @Before: will always be executed **before** each test method.
- @After: will always be executed **after** each test method.
- @Test – Defines a **test method**.

With this in mind, the implemented tests work as described in detail below:

1. In our project, the BeforeClass and AfterClass annotations will not be used, since we do not have a need that justifies it.
2. **LibraryPR2Test** class implements the *setUp method (@Before), the tearDown* method (@After), and some utility operations to handle data types (e.g. dates).
3. The *setUp method* initializes the data structures with an initial set of data. It does this by using the provided Factory (**FactoryLibrary**).
4. The setUp method will always be executed BEFORE each @Test and the *tearDown method* will always be executed AFTER each @Test. This guarantees that the INITIAL state of each and every @Test and the final state is known. Before each @Test the system data structure will have exactly the data that has been initialized with the *setUp method*. This allows for avoiding side effects between different test operations.
5. **LibraryPR2Test** class implements a set of test operations that are associated with the operations that have been designed in the **Library ADT**.
6. To execute the project, it must be executed (Run as) only on the **LibraryPR2Test class**. In the Annex (example of implementation of a project) the details of the execution of the project and captures of its operation are shown. The project provided with the utterance is pending completion of its development and for this reason, it will not work until its implementation is complete.
7. Unless otherwise stated, test classes should NOT be modified. For the improved test case, you can create a new test class.

Planning and tasks to be carried out

The planning that we propose to develop the PRA1 is the following (2/11 - 29/11):

1. (31/10-4/11). Analyze the solution of CAA1 and identification of the ADTs of the library.

2. (5/11-11/11). Analyze the test suite (**src/test/java/****) and validate that the data structures proposed in CAA1 are sufficient to meet the test suite requirements. If there is any modification, it must be indicated in the delivery text document. With this action, we are following a test-driven development methodology, *Test-driven development (TDD)* [3].

3. (12/11-24/11). Implement the manager and models necessary to carry out the operations defined by the provided interface.

4. (25/11-27/11). Testing, validation of test sets, and updating if necessary. Additionally, it is requested that the set of tests be expanded with additional cases to those provided.

Delivery format

The delivery must be made through the "EC Delivery and Registration" space, publishing a single ZIP file following the instructions detailed below:

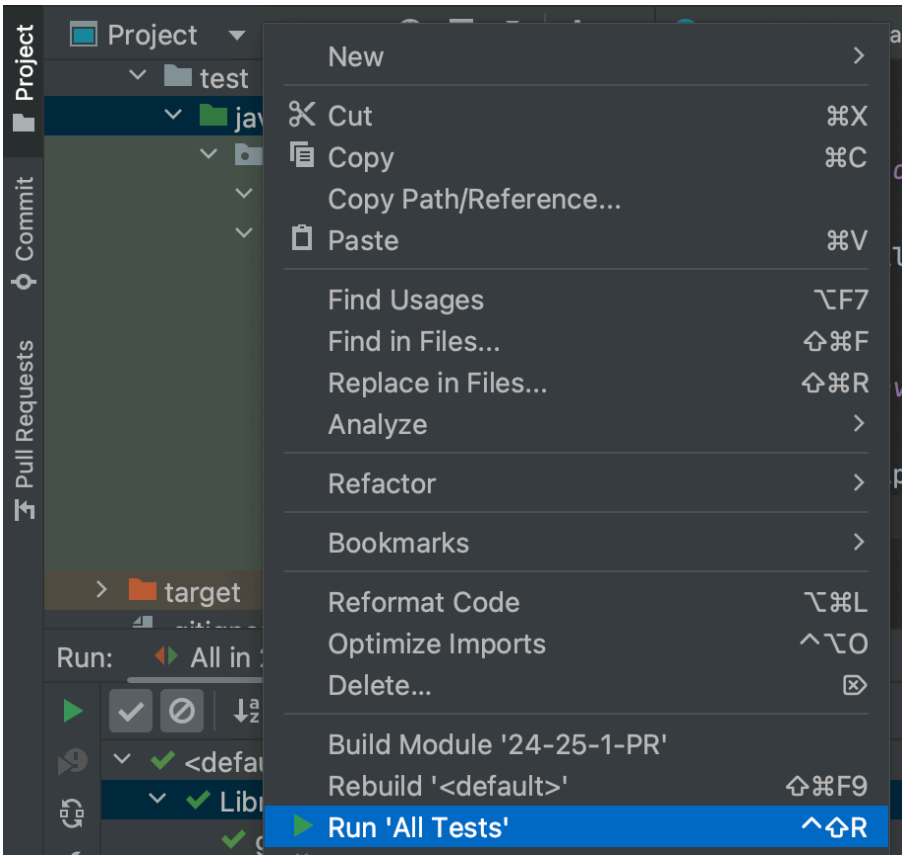
- In the main directory, a text document (readme.md) should appear indicating the NAME, email address, the scope of the delivery, modifications, and/or updates made to the initially proposed design (official CAA1 solution) with its justification if there has been some additional data structure, hints on the new test suites, problems, and additional comments where necessary.
- The code (src/**/java/**/) maintaining the project structure. DO NOT ATTACH either .class or the ADTs library of the course in the ZIP file that will be published. Remember that in addition to the set of tests that we provide you, you must expand that set of tests with new cases. The new test files (src/test/java/**) must be included in the ZIP.
- IT IS PROHIBITED TO MODIFY the code of the test files (src/test).

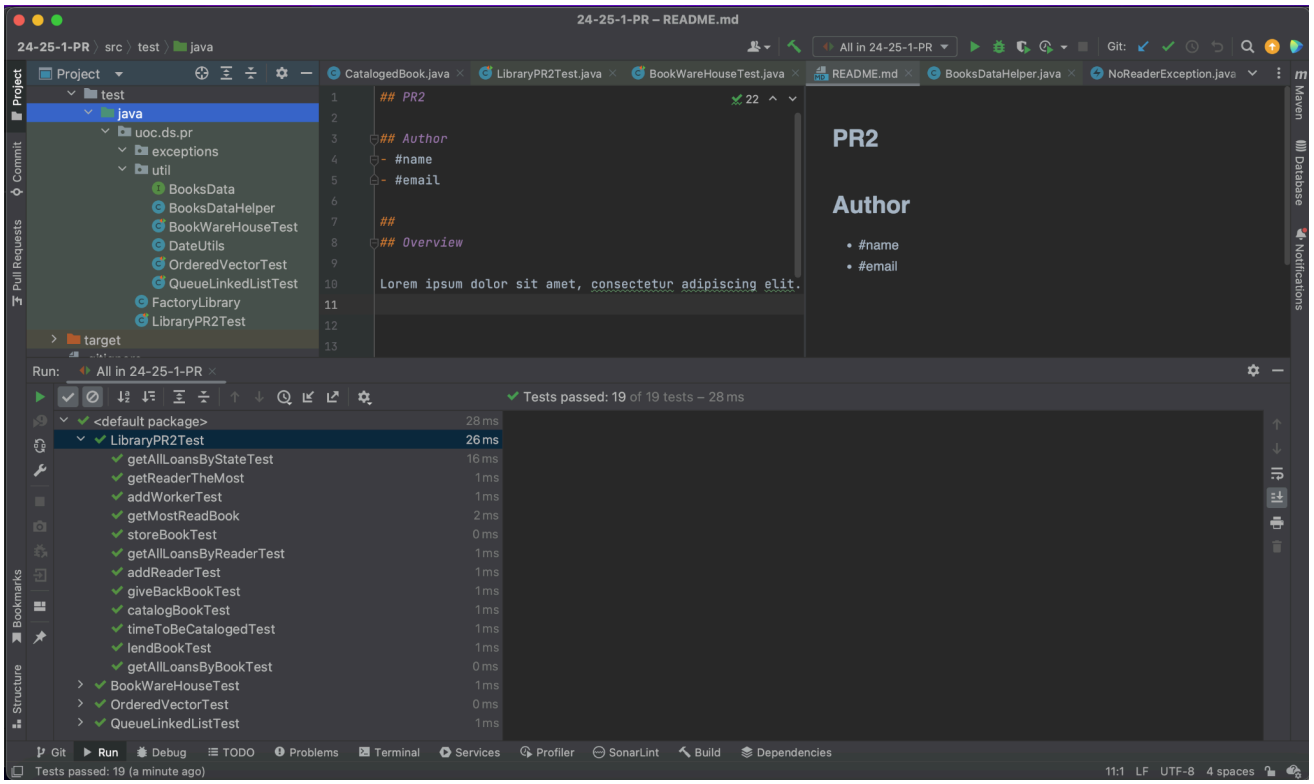
[3] https://en.wikipedia.org/wiki/Development_guided_by_tests

Annex I: Example of implementation of a PR2 project

To start the project, the following steps must be carried out:

1. Import the project, in the same way as PR0 was done, using the sources provided.
2. Make sure that in the project properties, the sources of both the main components (**src/main/****) and the Test components (**src/test/**)** are defined and that the SDK of the project is informed.
3. Run the project (after all features have been implemented) by running the **LibraryPR2Test class**. The result should be similar to the followings images:





6. In case there is a defect in your code, it will be displayed graphically and the console will give indications of the error. The following image shows an example of a defect in a part of a project in which a value was expected and the component returns another.