Report for Assignment 4

Project

Name: JabRef

URL: https://github.com/JabRef/jabref

A tool for managing papers and their bibliography.

Onboarding experience

After discussing whether to keep our existing project mockito or change, we decided to change. This decision was made due to mockito not reaching our requirements for how they handle their issues. They have no tags and the feedback from their maintainers is low. Therefore, we felt it was hard to select an appropriate issue. Instead, we looked at several other projects within the “git repositories” sheet, and also on the list of good beginner projects.

From this, we were hooked by JabRef’s superior issue handling and their activity as well as multiple good beginner issues to choose from. After selecting an issue, we received some valuable guidance on how to begin and implement the issue through additional comments made by the core developers of JabRef. This was incredibly useful and helped us form our understanding of the requirements of the solution.

Regarding the onboarding process, this project had better documentation than mockito, but was harder to set up due to it being larger and more complex. However, their setup guide was very helpful and it would be quite hard without it for newcomers. One side note though, we all failed at cloning the repository (we did a regular git clone …). The first page of JabRef setup guide stated that cloning had to be recursive to include third-party libraries. We discovered our mistake while running the test suite which generated a lot of failed tests that were not expected to fail.

Lesson learned*: you can git clone in more than one way and always read the documentation.*

We also encountered several failed test cases on Windows machines. In short, it was due to differences in LF and CRLF line breakers. There also seemed to be a difference between running tests on Mac and Windows, with a disparity in the number of tests that were executed.

The build system for JabRef is gradle, something we as a group are all familiar with from our last assignment. Therefore, setting up the build environment is less of a hassle than last time. There are existing unit tests, gradle tests, and existing coverage report generation tools using JaCoCo. However, running one branch coverage generation was, in reality, too long, as running one coverage task would take around 15 min. Therefore, we ended up not running them many times in our local development. Additionally, for viewing branch coverage, Jabref has an external source available.

Effort spent

Plenary Meetings

[10:30] 1 hr 30 min, Feb.27	Discussed and settled the project and the issues to work on.
[15:00] 2 hr 30 min, Feb.28	Set up the project and start creating sub-issues (divide work).

- Do a run-down of the project, and the selected issue.
- Be familiar with the contribution guidelines of the project.
- Identify the specific issue requirements and their technical details

[10:00] 1 hr , March 4	Gather group and conclude work

	Anders (h)	Elliot (h)	Hannes (h)	Ed (h)	Yening (h)
1. meetings	6	6	6	5	5
2. discussions	1	1	1	1	1
3. read doc	2	1	2	2	2
4. cfg/setup	2	2	1	2	2
5. analyze code/output	3	3	2	4	2
6. writing doc	3		1	3	1
7. writing code	3	9	4	3	2
8. running code	1	2	1	2	2
Total	20	23	17	22	17

Regarding point 4, we discovered that each of us had cloned the repository the wrong way, as we got tests that failed, which should not have failed. After everyone had managed to run the test suite, we compared our results and saw that one specific category of tests failed. These were related to a third-party library called Citation-Styles (specifically these two: styles and locales). Going back to the setup guide we discovered that we had to use recursive git cloning. After solving these missing libraries some of us passed all tests whereas there were still tests that failed for Anders.

Anders: I failed 9 tests and when checking what the expected/actual output was, they were the same! But the difference was that due to I'm on Windows 11, these tests had files with CRLF line endings and not LF line endings, which caused the test suite to fail. This took longer than expected to realize because I went through the setup guide to double-check if I missed anything.

Yening: Had unique setup issues which took some extra time and effort to solve. It was regarding getting the correct Java version to run and limitations of authority. After that I run the updated software several times to test its function.

Elliot: A lot of time was spent both doing the setup two times, as well as debugging some error issues. The first time was done without following the setup guide (as I did not know it existed). This resulted in some errors, and when talking in a meeting I realized I had many fewer test runs on my machine than my colleagues. They recommended the setup guide, and I redid the setup following the guide. However, the results were the same, so I did some debugging and checked the guide again to make sure I did the setup correctly. After talking to some other groups with the same project, I came to the realization that Mac (the hardware I am using) has fewer tests run than on other machines because other groups where some also use Mac have the same test count as me. All my colleagues use Windows or Linux which produces a different test count. This seems like a reasonable explanation for the disparity.

Overview of the issue(s) and work done.

Title: Replacing String Constants During Copy-Paste

URL: https://github.com/JabRef/jabref/issues/10872

A feature request where the copy and paste commands: ctrl+C and ctrl+V, should include string constants. Currently, when copying from the BibTex entry, you can not

(1) Paste with existing string constant and add it to JabRef library;

(2) Copy with existing string constants.

The user is requesting a new option in the copy drop-down that allows them to copy with string constant.

After group internal discussion, also pointed out by the project core developer, we have reached the conclusion that the new option in the drop-down was not what the user was really asking. We want to achieve the function all “behind the scene”. New string constants will be added on pasting, and will also be copied with ctrl+C.

Scope (functionality and code affected):

MainTable.java@copy()	add new functionality
MainTable.java@paste()	add new functionality
ClipBoardManager.java	add a new method
ConstantPropertiesView.java	add new functionality

Work done:

https://github.com/DD2480-Group1/jabref/pull/2 (canceled work, requirement change)

https://github.com/DD2480-Group1/jabref/pull/11

https://github.com/DD2480-Group1/jabref/pull/12

https://github.com/DD2480-Group1/jabref/pull/14

https://github.com/DD2480-Group1/jabref/pull/16

https://github.com/DD2480-Group1/jabref/pull/17

https://github.com/DD2480-Group1/jabref/pull/19

Requirements for the new feature or requirements affected by functionality being refactored

Requirement 1: Copying (Label in GitHub issue tracker as [Copy]):

When copying BibTex entries from the library, the referenced string constant should be added to the clipboard. This involves:

1. Export the string constants from the database;
2. Check for referenced LaTex string constants;
3. Add them to the clipboard with proper checking;

Requirement 2: Pasting (Label in GitHub issue tracker as [Paste]):

When pasting BibTex entries from the clipboard to the JabRef library, if there are string constants in the pasting texts, they should be added to the database's string constants for later use. This involves:

1. Upon pasting, parse the string to check if there are string constants;
2. Add the string constants to the library with collision checking. I.e. if there exist string constants with the same name, do proper handling. Add other string constants.
3. Check for the string constants to be in the correct format;

Requirement 3: Testing (Label in GitHub issue tracker as [Tests]):

Proper unit tests should be added to newly added methods and the refactored methods when implementing the above requirements. They should follow the project's development and testing guidelines and have valid assertions. Due to the nature of test-driven development, most of the unit tests added should fail before the development, and they would all pass after the development.

Code changes

Patch

(copy your changes or the add git command to show them)

Test results

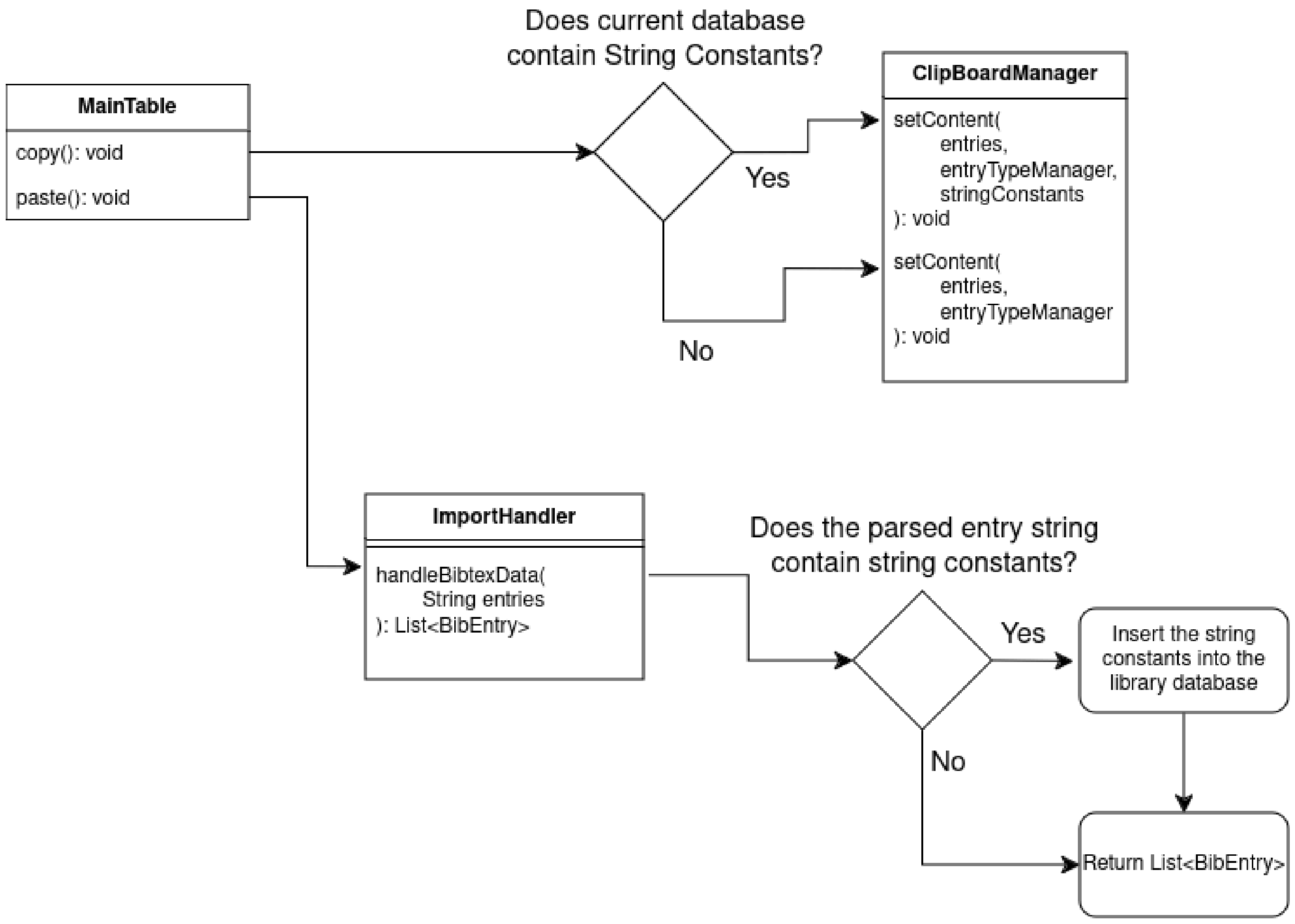
Overall results with link to a copy or excerpt of the logs (before/after refactoring).

UML class diagram and its description

UML diagram of the new feature and its related classes. The feature begins in the MainTable class where either copy or paste is called.

For the copy method, new logic has been added where a simple check if the database contains string constants is made. If yes, then a new setContent method will be called which serializes the string constants.

For the paste method, changes have been made to the importHandler@handleBibTeXData method. As is, this method already parses given string constants in from the clipboard, but it does nothing with the parsed values. New functionality has been added to fetch string constants from the parser and then add them into the library. When doing this there are also some extra checks to ensure that the constants are correctly formatted and that there are no duplicate constants in the library.


Does current database contain String Constants?
MainTable
copy(): void
paste(): void
Yes
No
ClipBoardManager
setContent(entries, entryTypeManager, stringConstants): void
setContent(entries, entryTypeManager): void
ImportHandler
handleBibtexData(String entries): List<BibEntry>
Does the parsed entry string contain string constants?
Yes
No
Insert the string constants into the library database
Return List<BibEntry>

Key changes/classes affected

Optional (point 1): Architectural overview.

Optional (point 2): relation to the design pattern(s).

Overall experience

Project experience

Overall, the project experience can be summarized as great. The project contains good documentation including high-level explanations and decision explanations. However, some parts are a bit unclear, such as the unit testing standards. There is not really any guidance on creating unit tests except what libraries they use. Any unit tests that interact with both the backend (logic) and frontend (UI) are burdensome to implement and get working correctly. We suggest that the core developers create an official UI unit tests development guide, since currently it is very difficult to implement UI unit tests without extensive knowledge of the codebase. Since this project contains a high percentage of graphical UI elements, many parts will involve UI to some extent, from buttons and tables to the clipboard (in our case). Such observation was also noted by the core developers in this issue.

Another key highlight of the project is the good direction for new developers, such as ourselves. A detailed label tracker for beginner-friendly such as a good first issue tag. Additionally, responses from the core developers were efficient and informative.

Team and the way of working under Essence

Main Takeaway

Optional (point 6): How would you put your work in context with the best software engineering practice?

Optional (point 7): Is there something special you want to mention here?

Optional (8): where do you put the project that you have chosen in an ecosystem of open-source and closed-source software? Is your project (as it is now) something that has replaced or can replace similar proprietary software? Why (not)?