Report for Assignment 4

Group 1
Authors: Anders Blomqvist, Elliot Darth, Hannes Stig, Yening Liu, ZOU Hetai

All 5 group members are attempting P+

Attempted P+ points: **(All P+ requirements are discussed at the end of the report)**

- Point 3: Relevant test cases are traced to requirements.
- Point 4: The patch is clean with no obsolete comments/code and no unnecessary whitespaces.
- Point 5: Patches are accepted by the project, or considered for acceptance.
- Point 6: Argue critically about the benefits and drawbacks of your work.
- Point 8: Where do you put the project that you have chosen in an ecosystem of open-source and closed-source software?

Project

Name: JabRef

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

An app for managing papers and their bibliography (.bib) reference files.

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] 2 hr,	March 4	Gather group and conclude work

	Anders (h)	Elliot (h)	Hannes (h)	Ed (h)	Yening (h)
1. meetings	6	6	6	6	6
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	2
7. writing code	3	9	4	3	2
8. running code	1	2	1	2	2
Total	21	24	18	23	19

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.

Hannes: The setup took an hour but not because it was difficult or there were any real issues. Most of the setup time came from setting up the IntelliJ IDEA which I had not used yet. Other than that I found the setup process very straightforward and easy to understand. After I was done I could run and build the project while passing all tests.

Overview of the issue 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 a 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
ImportHandler.java@handleBibTexData()	add new functionality
ImportHandler.java@importStringConstantsWithDuplicateCheck()	add new method
BibtexParser.java@getStringValues()	add new method
ClipBoardManager.java	add new method
ConstantPropertiesView.java@storeSettings()	add new functionality

Work done:

See the combined final work done in the patch section.

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

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; link to github requirement issue
2. Check for referenced LaTex string constants; link to github requirement issue
3. Add them to the clipboard with proper checking; link to github requirement issue

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;

link to github requirement issue

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.

link to github requirement issue

Code changes

Comparison between our **`fix-for-issue-10872`** branch to jabref **`main`** branch.

https://github.com/JabRef/jabref/compare/main...DD2480-Group1:jabref:fix-for-issue-10872

Patch

Pull request to main Jabref repository https://github.com/JabRef/jabref/pull/10992. Our changes implement the requested feature, however, it does not add a new "merge dialog" as mentioned by koppor.

Test results

Copy of the test file before the changes: Before

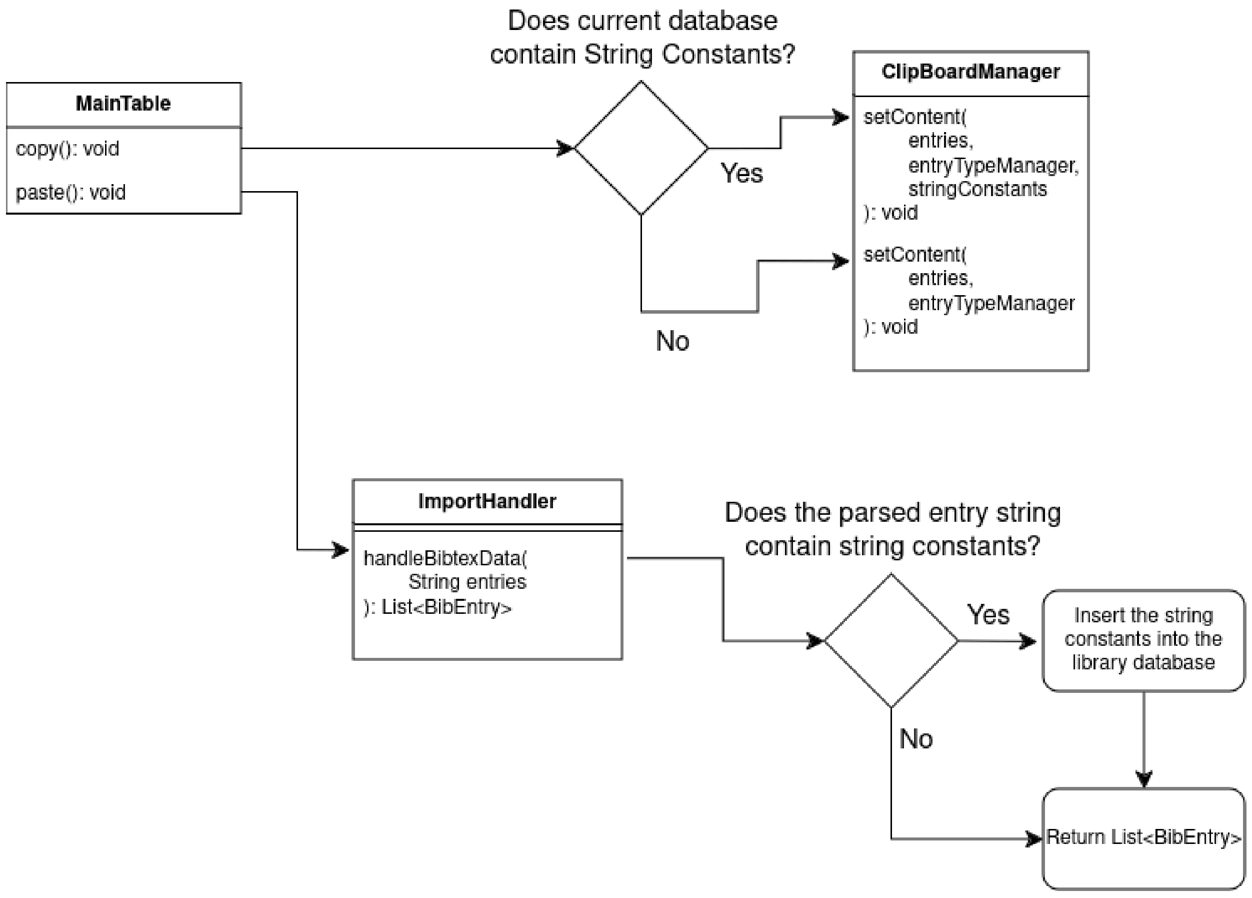
Copy of test log after the changes: After

UML class diagram and its description

Our new feature begins in the MainTable class where either copy or paste is called. The MainTable class is the primary GUI section where a user can edit their bibliography library. This class contains a lot of functionality, but we are only interested in the copy-and-paste methods, which are fired when a user presses “ctrl+C” or “ctrl+V”.

For the copy method, new logic has been added to check if the library database contains string constants. If yes, then we will call our new setContent method which serializes the given string constants and the rest of the bibliography entry to a simple string value that will be placed in the clipboard. The new setContent method is within another class called ClipBoardManager, which as its name suggests, manages the clipboard.

For the paste method, changes have been made to the importHandler@handleBibTeXData method. The ImportHandler is managing how new data would be added to the user's bibliography library. Before our feature implementation, this method already parsed given string constants from the clipboard, but it did 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.


MainTable
copy(): void
paste(): void
Does current database contain String Constants?
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>

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

Looking at the states of the Team table, we believe to have reached and completed the state of **Performing**. All the states of **Seeded** had already been achieved with the previous assignments, as well as the **Formed**-state. Some points in the formed state were re-identified for this assignment, such as ‘external collaborators’ (Kopper, the main project developer, comments for instance). For the **Collaborating** state we did not really work as a cohesive unit for this assignment but rather collectively through our communication channel. Still it was very organized and could count towards working together cohesively, just not working at the same time.

For the **Performing** state we believe to have achieved all the points mentioned. We coordinate the work that we would perform to prevent any mishaps such as overlapping work or wasted work. In cases of working becoming obsolete this was identified and clearly communicated in the group, resulting in us quickly adapting. Overall, we feel like we have achieved the main points of all the team states except for ‘adjourned’ which do not apply.

In terms of the team's way of working, we believe that we have made improvements since our last self-assessment and have progressed to the state of '**Working well**.' Common practices and tools have been established based on previous working experiences, resulting in fewer issues in their application. These practices and tools include git/GitHub conventions, coding IDEs, testing tools, continuous integration (CI), and coverage tools, among others. We have mainly improved our approach by regularly reflecting on our way of working during meetings. Our work is adaptable, allowing us to respond effectively to potential obstacles.

P+ Points

Relevant test cases are traced to requirements (P+ requirement 3):

The tests created were related to the requirements of the issue. Essentially the requirements can be viewed as two parts: the copy action and the paste action. An attempt was made to test both functionalities that were to be changed. The issue however is that Jabref does not so good documentation on test creation, resulting in especially UI tests being very difficult to construct. This led to one of the requirements "pasting" essentially being extremely difficult to test accurately. However, copy-tests were able to be made since the clipboard could be accessed through other means than JavaFX (the UI library used in Jabref).

This resulted in the main test being for the "copy" action, and the tests running in the "ClipBoardManagerTest.java" file. These tests simulate the copy action of a .bib reference with string constants and check that the constants were handled correctly; by being appended to the clipboard with the reference. Two tests were thus created, one to check the changed functionality of the copy function when copying a .bib reference, and an additional test that is similar but the copy also includes a string constant. This was done to ensure that when the changed method was added, nothing broke the old functionality (because the test was developed independently of the new functionalities). This means that initially, one test would pass while the other did not until the change was added. With this, we could ensure that our new functionality did not break anything old.

Furthermore, some other parts of the code had to be changed in order to accommodate the new copy functionalities added. One of these parts was tested as well in order to ensure that nothing old would break with our added feature. Specifically in "ConstantsPropertiesViewModelTest.java" tests were added to ensure that our new feature did:

1. Not break the old functionality of the method tested
2. Could handle the new method case when also having string constants included

This was done for the same reason as the aforementioned tests.

A Clean Patch (P+ requirement 4)

We believe our patch meets the requirements to be considered clean. We do not have any commented-out old code. We don't use any unnecessary debug output or error messages, other than in places where they are expected such as when the user tries to paste an ill-formatted entry into the library. The code also does not have any extraneous whitespaces or newlines. The project itself uses a style-checker to check for unnecessary whitespace and the patch passes this stylecheck. If you look at the files changed in the patch you can see that whitespaces and newlines are used sparingly, and we don't add or change lines outside the functions we have worked on.

An overview of the files changed in the patch can be found here

An Accepted Patch (In Progress) (P+ Requirement 5)

A draft pull request is currently open for the patch, minor changes might be needed for it to actually get merged. However, we have already gotten feedback saying the patch seems to be going in the right direction https://github.com/JabRef/jabref/pull/10992

Our work in context with the best software engineering practice (P+ Requirement 6)

In this project, we tried to put the goal into small parts and assigned it to different group mates. For example, fixing, reviewing, running and testing, and writing the report. Generally, the outcome is quite good and we succeeded in solving the problem. However, drawbacks still exist. Since there are many people trying to update the software, we can’t make sure that the changes we make will not lead to conflicts. When we were trying to merge two branches in our group, we had architectural failures and spent some time solving them. We think that it would be better if we can update and merge more often and we can get the newest results.

One thing that has been new to all group members during this course is using the Essence standard to evaluate our way of working. Using a standard like this to reflect on your own work feels in line with how you might work in a more official organization. In some ways, it seems like a good way of reflecting on your process as a team and how you could improve your work. We have found some use for the Essence standard, in part it helped us reflect on our feedback process and that we should implement feedback sessions more regularly during our work. However, we still aren’t entirely sold on the use of the Essence standard. While it does feel useful in some sense, it also feels like “a bit of a drag” sometimes having to write and document things that might be easier to just have discussions about.

Our project in the context of similar software (P+ Requirement 8)

JabRef works as a reference management software. There are closed-source tools that do similar things to JabRef. For example, some text editors like Microsoft Word and Google Docs have a simple reference management system built into them. JabRef could be used as a stand-alone tool to replace these features in those pieces of proprietary software. Of course, since JabRef is a stand-alone piece of software you are not tied to using it with any specific program. You could choose to use it with any piece of software that you use for writing documents, whether that be a plaintext editor like Notepad or a custom LaTex setup.

We are unsure if JabRef could entirely replace proprietary offerings that are built into platforms like Google Docs and Word. These offerings are, in our experience, much simpler tools that work best for managing references for smaller projects. On the other hand, JabRef offers a much larger set of tools that could be more useful for managing a very large set of references across multiple different projects. Even though JabRef could replace some of the simpler built-in offerings, users who only need the simple functions that these built-in tools offer might prefer to stick to them rather than use a separate more complex piece of software like JabRef.

There also exists other reference management software that functions more like JabRef does. For example, Zotero which is also open source, or Mendely which is created by Elsevier. We’re unsure how JabRef compares to these offerings, but on the surface, they seem to offer comparable features and could probably be used for similar purposes.