NTNU
Kunnskap for en bedre verden

IT1901 - Informatics, Project I

Autumn 2023

Student number: 561159

Group number: 60

Subject: 02 - Code Quality

Word count: 1398

Introduction to Code Quality

In object oriented programming a few core concepts and principles are often utilised to improve the code quality of the project. The most essential of these principles is the Object Oriented Design model, also known as OOD. This model is built upon the already famously used object oriented programing model, improving the work process creating the project. OOD is often defined as following the SOLID principle; *Single responsibility principle, Open-closed principle, Liskov substitution principle, Interface segregation principle, and the Dependency inversion principle*. Together these principles can be followed to create a *solid* application, which is easier to read, maintain, and develop. Abstraction further helps with the readability in the project and builds upon the interfaces and encapsulation principles. Creating classes with simple methods that can be called by other classes, without detail of what's actually going on in that class, nor how the methods function. These classes can –and if they're subject to change in the future, should– be created using an interface; making maintenance upon change a simple matter of creating the same methods regardless of how they internally function in the new class. Here the importance of naming conventions come into play, where descriptive names on methods in the classes will drastically improve readability throughout the project. Finally an essential part of any programming project –and language– is test coverage, making sure that the methods being created function as intended and as expected. Great test coverage safeguard your code and project to eliminate bugs and issues, again improving upon the overall code quality, leaving no stone unturned forgetting edge cases, user input checks, and other issues that could surface at runtime.

Positive Aspects in our Code Quality

For our development we have utilised the principles of quality code. Mainly we've focused on encapsulation of our classes and testing. In addition to this we've used the popular CheckStyle analytic tool to safeguard our code against issues that could occur; however it mostly found unutilised variables on scans, which were swiftly solved. SpotBugs was also used to ensure bug free code, analysing our code for uncaught issues. Private methods for the class only, protected methods for directory wise access and final variables were used to ensure good encapsulation of our code; in addition to securing constants from being reassigned. The project has been completed with simple and easy to understand code, with JavaDoc to document the code well. We decided to utilise JavaDoc from the beginning to ensure everyone's understanding of the methods being created, something which has greatly improved the development experience. This has also helped us not repeat ourselves any more than necessary, helping us see if a method achieving the wanted functionality already exists with quick glances. We've taken great care of the naming conventions throughout our project, ensuring that methods have a clear name describing its function. The same goes for our class names and variable names, making sure they are easy to read by a third party, or for code maintenance. One of the most important measures we took to secure our code quality was testing, to make sure our test coverage was up to par we utilised the JaCoCo tool. Testing has allowed us to find the bugs in our code early on, letting us fix our mistakes and oversights before it would have been too late. Naturally we could have testing even more than currently implemented, which could help us discover uncaught issues, something that should definitely be taken into consideration in future projects. Personally I would say our nesting is up to par with what is to be expected, not using any more indents than we find necessary. Any function we thought could use a method rather than further nesting in our code was implemented to ensure this. For the front end of our project we decided to use some robust libraries to simplify and improve the project. Here we used the frontend library React, in addition to the widely used TailwindCSS to improve the code quality, erasing common CSS issues that could occur.

Negative aspects and potential for improvement

Clear areas of future improvement in similar projects would include taking advantage of the dependency injection model commonly used in object oriented programming. We could have achieved a vastly more maintainable and updatable application by dividing some of our modules using this principle. Specifically and especially the input-output class would've been highly relevant to implement using an interface; This would have made it easier to replace the module if needed, in addition to clearly defining an abstraction to what's going on within. We encountered an issue by this class's location where we wanted to move the persistence layer completely into its own directory at the root level as a module; we were too far into the process to achieve this within a reasonable time due the refactor it would require. The decision not to put our efforts into the matter depended on whether we would need a parent connecting class to achieve the current functionality, or we could completely rewrite this part of the project. Naturally we realised an issue with our design not properly following OOD principles, thus creating this issue. Had we utilised abstractions and interfaces it could have been helpful for the update we wanted. The refactor would make our application clearer and more concise to maintain, avoiding the current nested modules. Although we have tested our code to what we found necessary, we have some methods that are not properly tested, specifically in our JavaFX module. Leaving some stones unturned and allowing for issues here to remain uncaught, this is a clear oversight on our part taking away from our wanted code quality and test coverage. By researching the used technologies more than we have currently, we could have achieved a higher code quality on earlier tries, avoiding our many rewrites of the same methods and functionality, leaving room for errors to occur due to incompetence with the technology. Moving on to the client side of our project we have some clear issues to be resolved, we chose to use the production robust framework React, here the component sizes are arguably off the wanted sizes, leaving some as large as three hundred and seventy five lines long, whilst some are as short as ten. The larger file in question should've been split into its own components to ensure the code's maintainability and readability. In addition to these issues we ended up not linting our code fully, leaving some issues behind in production which is unfortunate.

Conclusion

In conclusion our code quality can be improved to follow the above mentioned principles, this project has shown the ways to take these into consideration moving forwards. Tests could be created before any methods other than interfaces have been created, ensuring the code to be functional, and removing bias with creating said tests, allowing us to create safer and more robust code. The biggest issue encountered is the maintainability of our project, something we discovered late after a mistake in our original design. As mentioned before these could have been relieved by using proper abstraction, preferably using a dependent injection, which is generally good OOP- and widely used practises already. Further it would be beneficial to create more solid plans for the code development to ensure 'spaghetti code' is not created, somewhat related to our persistence issue where we could not easily move the module to a new directory. This ties tightly to the OOD, where we have to take the principles further into account before starting any code, again tying closely to SOLID, and especially interfacing; which would give us a simple rough draft of the expected methods to be used, letting us design other classes based on what we are allowed to call in our design. In addition to this new technologies the project group find necessary should be thoroughly researched before hand to get an idea of how it's used, rather than learning as we go which creates room for error and frustration, leading to more rushed and poor quality solutions.

References

Stoica, G. A. (u.å.). *Object Oriented Design Principles* [Lysarkpresentasjon]. Blackboard. https://ntnu.blackboard.com/ultra/

Stoica, G. A. (u.å.). *Testing and Code Quality* [Lysarkpresentasjon]. Blackboard. https://ntnu.blackboard.com/ultra/