**Task 3: Demonstrate Refactoring**

Fowler (2018) describes refactoring as “the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure” (p. 9). He elaborates further with the idea that software engineers often devise ways to improve the design while going through the process of building a system. He explains that making small changes to the code can cumulatively have a significant effect. Fowler (2018) describes several benefits of refactoring, including the ability to simplify the design and improve readability which can in turn help to increase extensibility and reduce the chance of creating bugs when adding enhancements or engaging in system maintenance procedures.

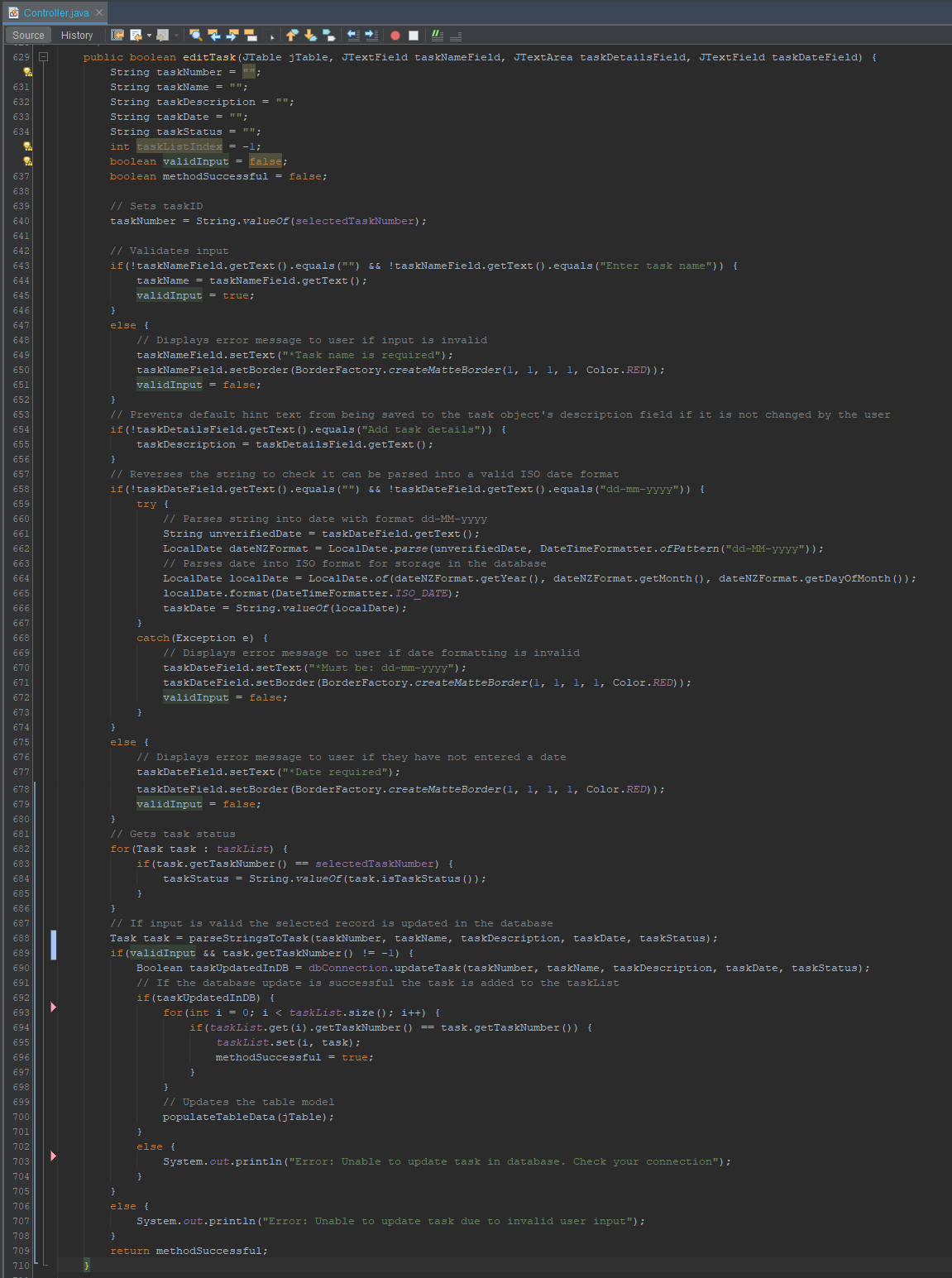
Kaya (2016) argues that long and complex methods can negatively affect readability and in turn make maintenance and code extension more difficult, leading to an overall reduction in system maintainability. He proposes the ‘Extract Method’ refactor technique be used to resolve these issues and describes it as breaking large methods down into smaller, simpler ones which can then be called as required by the original method(s).

In the below screenshot labelled ‘C*ode Before Refactoring*’ it can be observed that the *editTask(…)* method is unduly long and over-complicated to the point where readability and extensibility may be compromised. For these reasons it was determined that the *editTask(…)* method would be refactored using the ‘Extract Method’ technique.

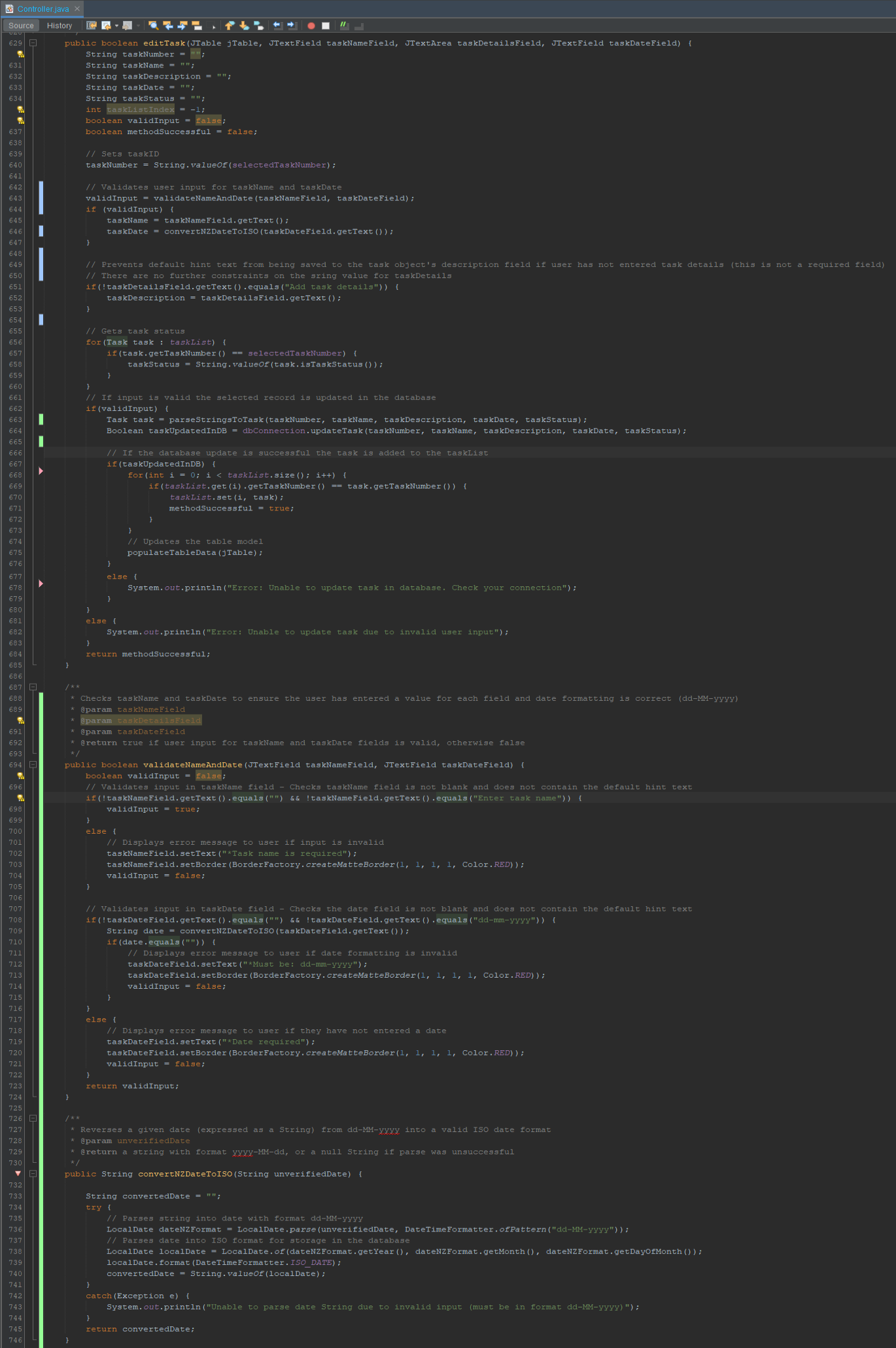
Kaya (2016) explains how carefully analysing variable declarations and uses can assist in the identification of ‘Extract Method’ refactoring opportunities. Following this approach it was observed that a large portion of the method (highlighted in blue) was used to validate user input. As a result, the method was refactored by removing the validation code and re-writing it into a new method called *validateNameAndDate(…)*. The new method can now be called to handle user input validation as and when required, which also enables other methods (e.g. the *addTask()* method) to access this same functionality without having to duplicate the same code.

In the interests of not making the same mistake twice, the opportunity was taken to further simplify the *validateNameAndDate(…)* method by removing the necessity for it to convert dates from NZ formatting to ISO standard formatting. Consequently, a further *convertNZDateToISO(…)* method was established which can now be called by *validateNameAndDate(…)* whenever it needs an NZ formatted date string converted into an ISO formatted date string.

After completing the refactor, the application test suite was employed to confirm that no new bugs had been introduced. It was found that the system’s external behaviour has not changed despite the modifications made to the code. It is therefore concluded that the refactor was a success. The resulting code can be observed in the screenshot below labelled ‘Code After Refactoring’ (changes are highlighted in green).

**Code Before Refactoring:**

**Code After Refactoring:**

****

**References:**

Fowler, M. (2018). *Refactoring: Improving the design of existing code.* Addison-Wesley Professional.

www.academia.edu/download/62290045/Martin\_Fowler\_-\_Refactoring\_-\_Improving\_the \_Design\_of\_Existing-By\_www.LearnEngineering.in20200305-13250-1kf3a2o.pdf

Kaya, M. (2016). Identification of Extract Method Refactoring Opportunities through Analysis of Variable Declarations and Uses*. International Journal of Software Engineering and Knowledge Engineering.* https://www.researchgate.net/profile/James-Fawcett-2/ publication/310601298\_Identification\_of\_Extract\_Method\_Refactoring\_Opportunities\_ through\_Analysis\_of\_Variable\_Declarations\_and\_Uses/links/5b7aedc392851c1e1223a9b0/ Identification-of-Extract-Method-Refactoring-Opportunities-through-Analysis-of-Variable -Declarations-and-Uses.pdf