Deliverable 3 **Rahul Sharma**
https://github.com/rahls7/soen6011 **40020034**

Problem 5- Code Review

I avoided using PMD for automated code review because the function uses a lot of System.out.println statements which makes its output very noisy. Instead I chose to focus on manual code review.

Programming Style Comments

1. As per the design document, the tool used by the author was Checkstyle and the style used was from Google. Throughout the code, author has maintained that checkstyle.

2. Javadoc has been provided for every function, however the description for params and args are redundant and author could have explained it better. For instance, @param numerator has description the numerator, which adds no value. This can be seen throughout the code.

3. There were some cases of redundant comments in the code. For instance, automatically generated TO DO comments where left in the code, which should have either been implemented or removed completely.

4. The naming of the variables followed camelCase according to the checkStyle, however the author did not do a great job at chosing descriptive names for the variables. For example, the variable to store all the input data was named *al* which doesn't explain what the variable stores. Instead the author chose to add a comment to explain what the variable does. In future, the author should focus on making variable names descriptive instead of using redundant comments to explain what it does.

Correctness of the code

The author was meticulous in handling all the input of the domain and backed up their development with good test cases. Given the requirements of the function, the code seems to provide fairly accurate approximation of the function.
No use of external library was found and everything was implemented from scratch.

Comments on Readability

1. The length of the functions are very long making it harder to follow and read the code. In general it is a good idea to limit the functions up to 30 lines of code[1]. This ensures the mode is modular, making it more robust and maintainable and also improves the readability of the code.
2. I struggled a bit to follow the code because of poor naming of functions and variable. For instance, upperPowerMultiply and belowPowerMultiply doesn't convey what the function is doing. Similarly the function name converting doesn't describe what the function is suppose to do. A better name for converting would have been getFractionFromDecimal.

Comments on Robustness

The author does a great job in handling errors and making them user friendly. The error messages are to the point and clear.

Comments on Test Cases

The author backs the implementation of the code with good test cases, however it would be nice to have complete code coverage. The author did not test individual methods and left some of them out. For instance, the converting method has no test case in the test suite.

Problem 7- Testing

Test Tool

I primarily used Eclipse and its debugger to test the implemented function. Since the test cases where implemented using jUnit, I made use of the junit plugin for eclipse.

Comments on Test Suite

The author makes use of junit to implement the test cases. However, the author chose to include a separate test file for every method implemented. A common practice is to make a Test suite which eases the process of testing in junit. This would have helped in testing purposes as only one test suite needs to be ran instead of 10 separate files in this case.

Test Cases Analysis

The author has covered all the functional requirements as per Deliverable 1 and all test cases passed.

Functional Testing

I carried out functional testing using the command line interface of the application. The author put in a lot of effort to handle edge cases and give meaningful error messages to the user making it a interactive application.

References

[1] Function Length Rule,
`https://dzone.com/articles/rule-30-%E2%80%93-when-method-class-or`