SOEN 6011: Project Deliverable 2

Team C, Student N4: Kenlo Dinh Van, 26641652

July 29, 2019

1 Logarithmic Function Implementation

1.1 Repository

The following report, source code and all artifacts are available on the following GitHub repository. [**1**]

1.2 Programming Style

Following Team C discussions, we've decided to follow Google's Java Style Guide [**2**] for all of our respective functions implementation. Two additional IntelliJ plugins were used to ensure the source code would conform to the Google's Java Style Guide.

The **google-java-format plugin** [**3**] integrates within IntelliJ IDE and allows code reformatting at the press of a keyboard shortcut.
Advantages: Convenient and fast way to reformat source code.
Disadvantages: Covers only indentation and formatting.

The **CheckStyle plugin** [**3**] is a development tool that help ensure that the written Java code follows a selected coding standard. In our case it provided underlined warnings whenever code was comforming to Google's Java style.
Advantages: Native support for Sun and Google's coding standard, possibility to add additional style.
Disadvantages: Invasive, Abundance of warnings while programming within the IDE.

1.3 Error Handling

In this implementation we've focused on handling errors due to user input, such as an input value out of the function's domain or textual input instead of numbers. Critical methods such as getBase() and getVariable() were implemented with try/catch blocks, throwing IllegalArgumentException with an error message specifying the non-acceptable or acceptable values.

1.4 Debugging

Description. Throughout the development of this project, we've used IntelliJ's native debugger. It allows placing break points within the source code, which pauses the program's execution at a given point and allow us to visualize each variable value before they are further processed.
Advantages No configuration needed, already well integrated within the IDE.
Disadvantages Usage is a bit complex, required browsing the online documentation.

1.5 Implementation

The **second algorithm** detailed in answer to Problem 3 was implemented. It relied on the Natural Algorithm Approximation followed by a change of base.

$$\ln(x) \;=\; \lim_{n\to\infty} \; n \; (\; x^{1/n} \; - \; 1)$$

Advantages: Simpler implementation.
Disadvantages: Lack of precision. Detailed instructions on how to execute the program are included int the README.md file.

1.6 Utility Functions

In order to not use any native or external Java library or package, additional functions needed to be implemented. These implementation relied on partial solution provided online by fellow programmers.**[5]**[**6**] The square, power and square root functions fall under this category.

1.7 Quality attributes

In order to ensure the quality of the implementation further steps were added to the development process.

Correctness: In order to ensure the provided results were correct, multiple tests and comparisons were done with Java's native logarithmic functions, results were conclusive with a precision of 1.0 x 10^{-7}.

Efficiency: Throughout the testing process, we ensured that all results were displayed within 1 second with no noticeable lag.

Maintainability: The program has been subdivided in 10 functions, with each function carrying one functionality/purpose. The variable names were chosen to make the program more understandable. Each feature was refactored and isolated into its own function once it was confirmed to work.

Robustness: Robustness is ensured by handling most of the possible errors that can occur during the program execution. The errors are mainly triggered by not acceptable or out of domain user inputs.

Usability: Minimal textual interface with 2 choices: terminate the program or use the logarithmic functionality. The function execute within a loop, in order to test multiple log values, until the user decides to exit the program.

2 Unit Testing

All the following unit tests are contained within the class CalculatorTest. All tests were developed using the framework Junit4.

2.1 Requirements Traceability

F4.0.v1 Function Selection: the *testFunctionSelection* insure that the user keyboard input is correctly read, and the logarithmic function has been selected.

F4.1.v1 Logarithm Base Initialization: the *setBaseValue* test insure the desired base value is correctly stored before calculation.

F4.2.v1 Logarithm Base Validation: the tests *baseNotNegative, baseNotZero, baseNotOne, baseMustBeNumber* ensure that user input-ed values are correct and valid for further calculation.

F4.3.v1 Logarithm Variable Initialization: the test *setVariableValue* ensure that the variable value is correctly stored before calculation.

F4.4.v1 Logarithm Variable Validation: the Tests *variableNotNegative, variableNotZero* ensure that the desired variable is within the function's domain.

F4.5.v1 Logarithm Calculation: the test *calculateLogarithm* ensure that the approximated result is comparable to Java's native logarithmic function.

F4.6.v1 Result Display: no further unit test was written to ensure that results were correctly displayed, we relied trusted Java's System. functions.

2.2 JUnit Guideline

The JUnit guide from TutorialsPoint [**7**] was used in order to write the aforementioned unit tests.

References

[**1**] https://github.com/k3nlo/SOEN6011_TeamC_N4
[**2**] https://google.github.io/styleguide/javaguide.html
[**3**] https://plugins.jetbrains.com/plugin/8527-google-java-format
[**4**] https://plugins.jetbrains.com/plugin/1065-checkstyle-idea
[**5**] https://stackoverflow.com/questions/3518973/doubleing-point-exponentiation-without-power-function
[**6**] https://rekinyz.wordpress.com/2015/01/20/implement-simple-sqrt-with-java/
[**7**] https://www.tutorialspoint.com/junit/junit$_{w}riting_{t}ests.htm$