Black-Box Testing

Milestone 2

George Dimitrov, Yasir Al-Bender, Cory Ebner & Will Nguyen

ABSTRACT

Correctness and Accuracy are critical in the field of Mathematics. As such, a calculator application must provide both of these requirements in order to be taken seriously as a tool for mathematics. Correctness and Accuracy will be black-box tested by using the following techniques: Equivalence Partitioning, Boundary-Value Analysis and Error Guessing. Using the results from these tests, one can assess the effectiveness of the tests and the functionality of the calculator.

Table of Contents

1. Test-Case Design for Functional Requirements

Since this program is mainly a calculator, it remains logical only to use numbers as inputs. A variety of mathematical functions were identified and selected to ensure functionality of the calculator. The main requirements to test are correctness. This differs from accuracy in the sense that accuracy test for correct number of decimal places while correctness tests for the correct answer. Test cases can be divided into 3 main categories: Equivalence Partitioning, Boundary-Value Analysis and Error Guessing.

The procedure for testing functional requirements is as follows:

1. Determine the input domain of a function based on documentation and requirements.
2. Based on the input domain and function, identify characteristics in which to define partitions.
3. Based on partitions, identify which type of testing to use.
4. Create test cases based on partitions in JUnit.
    i. For Equivalence Partition – Ensure partition satisfy 2 conditions: Completeness and Disjoint
    ii. For Boundary-Value Analysis – Identify boundaries for each partitions and uses those values in testing
    iii. Error Guessing – Use prior mathematic knowledge and experience to use values in testing
5. Record test results in cloud-based spreadsheet.

For most functions, Equivalence Partition testing was utilized. Most if not all functions take the same types of inputs. The main partitions used:

- Large Positive Values
- Small Positive Values
- Large Negative Values
- Small Negative Values
- Zero
- Infinity

For more special cases of testing, like trigonometric functions, Boundary-value Analysis was used. However, the upper bounds in most cases resulted in the test failing. This is due to the max value being too great for the calculator to handle. Since it allows for 64-bit IEEE-754 format, it took that as input but it did not compute it properly. Still, upper bound and lower bound values were determined and used for trigonometric testing.

Lastly, bitwise and some logarithmic functions are tested using Error Guessing. Using prior knowledge about certain properties about these mathematic concepts, the correctness of the implementation can be tested. For example:

$$\ln e^x = x$$

This property was tested by using java.Math.log to get the expected answer. Then comparing that answer to the actual one received from the function.

2. Test Case Design for Non-Functional Requirements

The main Non-Functional requirements are accuracy and usability. Accuracy was tested by using a leniency of 0.1. Meaning actual results from the functions must not have more of a difference of 0.1 with the expected results. This was to ensure the precision of all outputs from the functions were mathematically exact. Although the leniency is 0.1, test cases can be tested to the 0.000000000000008 to test for accuracy according to the developer. All test failed upon changing the delta to the developer stated number so testing will need to be redone to with new numbers to check for accuracy.

Usability will be tested in a future date and was not examined in this milestone. The team concluded that usability of the calculator remained largely within the graphical user interface (GUI). Since, the GUI will be inspected more in depth in a later milestone, it was decided that the GUI will not be tested right now.

3. Report Results

For actual results, see Appendix A. – Results.

4. Conclusions and Recommendations

The degree of effectiveness in the testing was very high considering it was the team's first attempt at using formal testing techniques. The preparation and definition of the partitions early in testing certainly helped synchronized the team. Another important factor was the independence and trust given to each tester. Each member of the team had freedom to choose which types of tests were to be performed. Although this factor was mainly positive, a downside to the flexibility given to each tester was inconsistent communication and inconsistent naming scheme. A more centralized approach may be taken for future testing.

It is a consensus among the team that Equivalence Partition was the best method for testing in a project like this one. The partitions defined were certainly a factor in the ease of testing experienced by the team. Error Guessing was only helpful if the tester had a large amount of knowledge regarding the function. It was effective in testing the correctness of a function if the tester knew what the function was base off. Boundary-Value Analysis was somewhat difficult to work with since it required prior knowledge about which numbers worked and which didn't within the boundaries of mathematical concepts used in the functions.

From the results, we can conclude that the calculator's functions were, for the most part, implemented correctly. However, the tests performed showed some precision problems with some functions. Precision is a very important requirement especially when dealing with mathematics. The other interesting result was the Java Overflow error experienced when testing the Combination function. It is particularly intriguing because the program, itself, does not handle the error.

Appendix A. – Results

Tester	Type Of Tests	Package Tested	Class Tested	Test Scenario	Results	Notes
George	Black-Box	jscicalc.pobject	Cube	Large Negative Value	Pass	
				Small Negative Value	Pass	
				Zero	Pass	
				Small Positive Value	Pass	
				Large Positive Value	Pass	
				Positive Decimal	Pass	
				Negative Decimal	Pass	
			Square	Large Negative Value	Pass	
				Small Negative Value	Pass	
				Zero	Pass	
				Small Positive Value	Pass	
				Large Positive Value	Pass	
				Positive Decimal	Pass	
				Negative Decimal	Pass	
			Inverse	Large Negative Value	Pass	
				Small Negative Value	Pass	
				Zero	Pass	
				Small Positive Value	Pass	
				Large Positive Value	Pass	
				Positive Decimal	Pass	

Factorial	Negative Decimal	Pass	
	Large Negative Value	N/A	
	Small Negative Value	Pass	Only negative test done, rest are redundant, pass condition is an exception is thrown.
	Zero	Pass	
	Small Positive Value	Pass	
	Large Positive Value	Pass	
	Positive Decimal	Fail	Using 2.5, Actual Result: Arithmetic Exception, Expected Result: 1.875 or 2
	Negative Decimal	N/A	

Tester	Type Of Tests	Package Tested	Class Tested	Test Scenario	Results	Notes
Cory	Black-Box	jscicalc.pobject	Combination	x is negative, y is negative	Pass	
				x is negative, y is positive	Pass	
				x is positive, y negative	Pass	
				x is zero, y is positive	Pass	
				x is positive, y is zero	Fail	
				x is zero, y is zero	Fail	
				x is larger positive, y is smaller positive	Pass	
				x is smaller positive, y is larger positive	Pass	
				x is large positive, y is large positive	Fail	Java throws stack overflow. Manually testing an error message is displayed, the function tested does not handle this.
				x is decimal < .5, y is non decimal positive	Pass	
				x is decimal >= .5, y is non decimal	Pass	

	positive		
	x is decimal < .5, y is decimal < .5	Pass	
	x is decimal >= .5, y is decimal >= .5	Pass	
	x is non decimal positive, y is decimal < .5	Pass	
	x is non decimal, y is decimal >= .5	Pass	
	Large Negative Value: -8000000	Fail, uses complex numbers as answers. Says it uses real numbers	Expected: -200, Actual: NaN
	Small Negative Value: -2	Fail, uses complex numbers as answers. Says it uses real numbers	Expected: -1.259921, Actual: NaN
Cube Root	Zero	Pass	
	Small Positive Value	Pass	
	Large Positive Value	Pass	
	Positive Decimal	Pass	
	Negative Decimal: -576.78654	Fail, uses complex numbers as answers. Says it uses real numbers	Expected: -8.324121, Actual: NaN
Logarithm (Base 10)	Large Negative Value: -8000000	Fail, uses complex numbers as answers. Says it	Expected: Error Message, Actual: NaN

		uses real numbers	
	Small Negative Value: -2	Fail, uses complex numbers as answers. Says it uses real numbers	Expected: Error Message, Actual: NaN
	Zero	Pass	
	Small Positive Value	Pass	
	Large Positive Value	Pass	
	Positive Decimal	Pass	
	Negative Decimal: -576.78654	Fail, uses complex numbers as answers. Says it uses real numbers	Expected: Error Message, Actual: NaN
	Large Negative x: -8000000, Large Negative y: -8000000	Pass	
	Small Negative x: -2, Large Negative y: -8000000	Pass	
Addition	Small Negative x: -2, Small Negative y: -2	Pass	
	Large Negative x: -8000000, Small Negative y: -2	Pass	
	x = Zero, non zero y = 5	Pass	
	x = Zero, y = zero	Pass	

Non-zero x = 5, y = zero	Pass
Low Positive x: 50, Low Positive y: 60	Pass
Large Positive x: 479001600, Small Positive y = 44	Pass
Large Positive x: 479001600, Large Positive y = 479001600	Pass
Small Positive x: 47, Large Positive y = 479001600	Pass
Positive Decimal x: 56.29, Positive Decimal y: 56.29	Pass
Positive Decimal x: 56.29, Negative Decimal y: -56.29	Pass
Negative Decimal x: -56.29, Positive Decimal y: 56.29	Pass
Negative Decimal x: -56.29, Negative Decimal y: -56.29	Pass

Tester	Type Of Tests	Package Tested	Class Tested	Test Scenario	Results	Notes
Will	Black-Box	jscicalc.pobject	Natural Logarithm	Property: ln(1) = 0	Pass	
				Property: ln(x^y) = y*ln(x)	Pass	
				Property: ln(e^y) = y	Pass	
				Property: ln(x)+ln(y) = ln(x * y)	Pass	
				Zero	Pass	
				Negative Non-Zero: -1	Pass	
				Infinity	Pass	
				Negative Zero	Pass	
				Large Positive Value: 1000	Pass	
				Small Positive Value: 0.0001	Pass	
				Zero	Pass	
				Property: 10^(x+1)/10 = 10^x	Pass	
			Inverse Logarithm	Property: 10^1 = 10	Fail	Small Precision Difference
				Negative Value: -1	Fail	Small Precision Difference
				Large Negative Value: -1000	Fail	Small Precision Difference
				Testing Negative Property Equality: [10^(2+1)]/10 = 10^2	Fail	Small Precision Difference
				Large Positive Value: 1000	Pass	
				Small Positive Value: 0.0001	Pass	

AND	Property: 0 AND 0	Pass	
	Property: 0 AND 1	Pass	
	Property: 1 AND 0	Pass	
	Property: 1 AND 1	Pass	
	Large Positive Values: 1110 AND 1001	Pass	
	Infinity AND Infinity	Pass	Pass Condition: Exception was Thrown
XOR	Property: 0 AND 0	Pass	
	Property: 0 AND 1	Pass	
	Property: 1 AND 0	Pass	
	Property: 1 AND 1	Pass	
	Large Positive Values: 1110 AND 1001	Pass	
	Infinity AND Infinity	Pass	Pass Condition: Exception was Thrown

Tester	Type Of Tests	Package Tested	Class Tested	Test Scenario	Results	Notes
Yasir	Black-Box	jscicalc.pobject	Sine	DegreesUpperBound	Pass	
				DegreesLowerBound	Pass	
				DegreesNegativeUpperBound	Pass	
				DegreesNegativeLowerBound	Pass	
				DegreesZero	Pass	
				DegreesFifty	Pass	
				RadiansUpperBound	Fail	
				RadiansLowerBound	Pass	
				RadiansNegativeUpperBound	Fail	
				RadiansNegativeLowerBound	Pass	
				RadiansZero	Pass	
				RadiansFifty	Pass	
			Cosine	DegreesUpperBound	Fail	
				DegreesLowerBound	Pass	
				DegreesNegativeUpperBound	Fail	
				DegreesNegativeLowerBound	Pass	
				DegreesZero	Pass	
				DegreesFifty	Pass	
				RadiansUpperBoun	Pass	

	d	
	RadiansLowerBound	Pass
	RadiansNegativeUpperBound	Pass
	RadiansNegativeLowerBound	Pass
	RadiansZero	Pass
	RadiansFifty	Pass
	DegreesUpperBound	Fail
	DegreesLowerBound	Pass
	DegreesNegativeUpperBound	Fail
	DegreesNegativeLowerBound	Pass
	DegreesZero	Pass
Tangent	DegreesFifty	Pass
	RadiansUpperBound	Pass
	RadiansLowerBound	Pass
	RadiansNegativeUpperBound	Pass
	RadiansNegativeLowerBound	Pass
	RadiansZero	Pass
	RadiansFifty	Pass
	DegreesUpperBound	Pass
Inverse Cosine	DegreesLowerBound	Pass
	DegreesNegativeUp	Pass

perBound	
DegreesNegativeLowerBound	Pass
DegreesZero	Pass
DegreesFifty	Pass
RadiansUpperBound	Pass
RadiansLowerBound	Pass
RadiansNegativeUpperBound	Pass
RadiansNegativeLowerBound	Pass
RadiansZero	Pass
RadiansFifty	Pass

Appendix B. – References

Wolfram Alpha Computational Engine - http://www.wolframalpha.com/
Official Sourceforge Website - http://jscicalc.sourceforge.net/

- http://jscicalc.sourceforge.net/precision.php
- http://jscicalc.sourceforge.net/trig.php
- http://jscicalc.sourceforge.net/factorial.php
- http://jscicalc.sourceforge.net/roots.php
- http://jscicalc.sourceforge.net/modes.php
- http://jscicalc.sourceforge.net/test.php

Official JUnit GitHub Wiki - https://github.com/junit-team/junit/wiki