



**SOEN 6011 : SOFTWARE ENGINEERING PROCESSES
SUMMER 2021**

SUPER CALCULATOR

PROBLEM - 5
Source Code Review

Authors

Rokeya Begum Keya

Kyle Taylor Lange

Sijie Min

Manimaran Palani

<https://www.overleaf.com/project/610304de4e6b8d24f7c781b6>

Contents

a) Source code Review of F3	2
b) Source code Review of F5	4
d) Source code Review of F2	10

PROBLEM 5 - F2: $\tan(x)$

SOEN 6011 - Summer 2021

Software Engineering Processes

Repository address : <https://github.com/Dakatsu/SOEN6011Calculator>

Rokeya Begum Keya

40183615

Source code Review of F3

The report of source code review for Hyperbolic Sine, $\sinh(x)$ function: Developed by Kyle Taylor Lange.

Manual Code Review

Naming Conventions

The class name, variables, constant variables and methods are named according to the Java Naming Conventions.Java.

JavaDoc Comments

SinhLibrariesTest.java - It would be better to quickly understand the Unit test methods if Javadoc comments are on the test cases.

```
class SinhLibrariesTest {  
  
    @Test  
    void sinhZeroes() {  
        assertEquals(0, SinhLibraries.sinh("0"));  
        assertEquals(0, SinhLibraries.sinh("0."));  
        assertEquals(0, SinhLibraries.sinh(".0"));  
        assertEquals(0, SinhLibraries.sinh("0.0"));  
    }  
  
    @Test  
    void sinhPosOne() {  
        assertEquals(1.1752012, SinhLibraries.sinh("1"), 0.0000001);  
    }  
  
    @Test  
    void sinhNegOne() {  
        assertEquals(-1.1752012, SinhLibraries.sinh("-1"), 0.0000001);  
    }  
  
    @Test  
    void sinhPosTwo() {  
        assertEquals(3.6268604, SinhLibraries.sinh("2"), 0.0000001);  
    }  
}
```

Figure 1: JavaDoc comments in test methods is missing

Automatic Code Review

Automatic source code review is done by **CheckStyle**[1] plug-in in eclipse IDE. I used google-check-global to review the code. I underline four errors which, are needed to be fixed, is given below in the figure 2.

❌ '0.5' is a magic number.	SinhLibraries.ja...	/sinh/src/SinhReview...	<u>line 116</u>	Checkstyle Pro...
❌ '0.5' is a magic number.	SinhLibraries.ja...	/sinh/src/SinhReview...	<u>line 165</u>	Checkstyle Pro...
❌ Array brackets at illegal position.	SinhLibraries.ja...	/sinh/src/SinhReview...	line 221	Checkstyle Pro...
❌ File contains tab characters (this is the first instance)	SinhLibraries.ja...	/sinh/src/SinhReview...	line 4	Checkstyle Pro...
❌ <u>Javadoc comment is placed in the wrong location.</u>	SinhLibraries.ja...	/sinh/src/SinhReview...	<u>line 23</u>	Checkstyle Pro...
❌ Line has trailing spaces.	SinhLibraries.ja...	/sinh/src/SinhReview...	line 220	Checkstyle Pro...
❌ Line has trailing spaces.	SinhLibraries.ja...	/sinh/src/SinhReview...	line 221	Checkstyle Pro...
❌ Line has trailing spaces.	SinhLibraries.ja...	/sinh/src/SinhReview...	line 224	Checkstyle Pro...
❌ Line has trailing spaces.	SinhLibraries.ja...	/sinh/src/SinhReview...	line 225	Checkstyle Pro...
❌ <u>Line is longer than 80 characters (found 100).</u>	SinhLibraries.ja...	/sinh/src/SinhReview...	line 88	Checkstyle Pro...
❌ <u>Line is longer than 80 characters (found 81).</u>	SinhLibraries.ja...	/sinh/src/SinhReview...	line 71	Checkstyle Pro...

Figure 2: Errors found by Checkstyle

PROBLEM 5 - F3: Hyperbolic Sine, $\sinh(x)$

SOEN 6011 - Summer 2021

Software Engineering Processes

Repository address : <https://github.com/Dakatsu/SOEN6011Calculator>

Kyle Taylor Lange

27627696

Source code Review of F5

I used Google's code review guidelines [1] to review Sijie's code for function F5. The *calculate()* function is the main function in F5.java, which returns the value of ab^x . This function simply calls a separate function for b^x and multiplies that result by a , which makes sense from a reuse standpoint.

One confusing aspect was that there are two functions named *power()*, and the comment did not make it apparent that one takes a double exponent while the other takes an integer. The functions could be renamed to make this clearer. e.g. *powerInt* and *powerDouble*, or the main body of the comments could be altered to make this clearer to those unfamiliar with how exponents relate to logarithms.

In any case, the double version of *power* appears syntactically correct. It returns 1 or 0 if the exponent or base are 0, respectively, and then it relies on separating the integer from the decimal portion of the base similarly and calculates the base raised to the integral power times Euler's number raised to a power with a function called *ex*.

The *ex* function has a confusing name, as I presumed it was short for exponent. Its Javadoc comment does clear this up, but perhaps renaming the function to something more detailed, e.g. *eToX* or *naturalExp* would make it clearer to casual users of this library.

The names of the functions and variables all conform to basic style guidelines, with constant values written in all caps and method/variable names written in camelCase. The main exception are the multiple variable names that are written as just a single letter, e.g. a or x . This was difficult to read since I had to constantly refer to the Javadoc comments to remember which variable represented what. There are very few comments in the body of the functions themselves. This may not be necessary for some of the shorter functions, but certain functions took a bit of effort to understand what was occurring.

These functions were unfortunately not integrated into the main calculator program, so I was unable to review how the code worked in practice. Nevertheless, it passes all unit tests.

Bibliography

- [1] How to do a code review, *Google*
<https://google.github.io/eng-practices/review/reviewer/>

PROBLEM 5 - F5

SOEN 6011 - Summer 2021

Software Engineering Processes

Repository address : <https://github.com/Dakatsu/SOEN6011Calculator>

Sijie Min

40152234

Source code Review of F7

Here is the source code review of a Transcendental function (F7) - x^y : Developed by Manimaran Palani.

Manual Code Review

Source File Naming

The code conforms to the google code style, and the method names and variable names in the code conform to the standard of camel case nomenclature

JavaDoc Comments

Javadoc is added to classes and methods to facilitate the formation of highly readable documents with the project and to facilitate code understanding. But there is a lack of comments inside the function, you can add comments appropriately to help understand the code

```
33  /**
34   * Power.
35   *
36   * @param power the power
37   * @param firstRealNumber the first real number
38   * @param secondRealNumber the second real number
39   * @return the double
40   */
41  static double power(double power, double firstRealNumber, double secondRealNumber) {
42      int exponent = (int) secondRealNumber;
43      double base = firstRealNumber;
44      while (exponent != 0) {
45          if ((exponent & 1) != 0) {
46              power *= base;
47          }
48          base *= base;
49          exponent >>= 1;
50      }
51      return power;
52  }
53 }
```

Figure 1: review F7.1

```

21
22- /**
23     * Test Case Id : F7_TestCase_1
24     * Test cases for Requirement Id : F7-R1
25     * test if X(0) to the power of Y(0)=1.0
26     *
27     */
28- @Test
29     public void zeroPowerofZero() {
30         assertEquals(1.0, PowerFunction.calculate(0, 0), 0);
31     }
32

```

Figure 2: review f7.2

Automatic Code Review Review of F7

Use CheckStyle for automatic code review, the following figure PowerFunction.java and CheckStyle violations chart of PowerFunctionTest.java.

	Checkstyle violation type	Occurrences
⚠	'X' has incorrect indentation level X, expected level...	18
⚠	'X' should be separated from previous import grou...	1
⚠	'X' child has incorrect indentation level X, expected...	14

Figure 3: review of f7

I checked the naming convention issues that are easy to ignore through CheckStyle, as shown in the picture

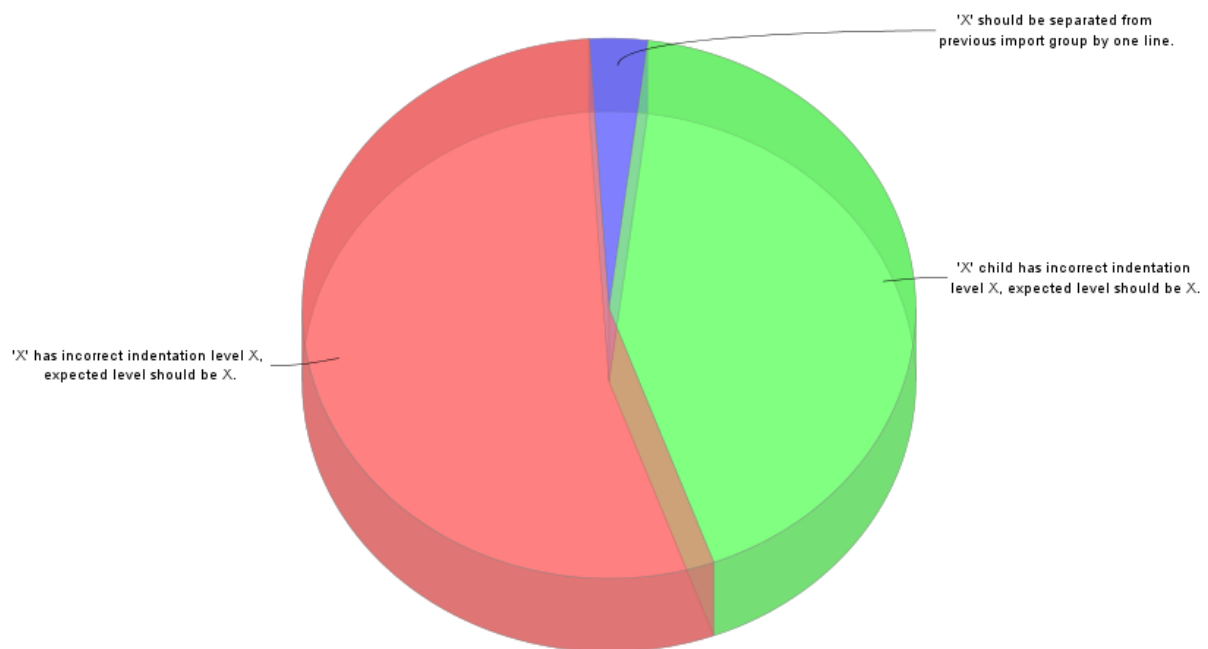


Figure 4: check style of f7

Bibliography

- [1] Mike Spivey. "The fuzz Manual" Manual and software copyright . J. M. Spivey 1988, 1992, 2000

PROBLEM 5 - F7 : x^y

SOEN 6011 - Summer 2021

Software Engineering Processes

Repository address : <https://github.com/Dakatsu/SOEN6011Calculator>

Manimaran Palani

40167543

Source code Review of F2

This sections presents the source code review of a Transcendental function (F2) - $\tan(x)$: Developed by Rokeya Begum Keya.

Manual Code Review

Source File Naming

The class name and methods are named as per Java Naming Conventions. Java uses Camel-Case as a practice for writing names of methods, variables, classes, packages and constants.

JavaDoc Comments

Javadoc is a tool which comes with JDK and it is used for generating Java code documentation in HTML format from Java source code, which requires documentation in a predefined format.

TangentFunction.java - Methods and variables declared in the class can have appropriate JavaDoc to have a better readability of the code.

```
1 package com.calculator.main;
2
3 public class TangentFunction {
4
5     /**
6      *
7      */
8     private static final long serialVersionUID = 1L;
9
10    public static String calculate(double parseDouble) {
11        return getTan(parseDouble);
12    }
13
14
15    // variable start here
16    final static double PI = 3.14159265358979;
17    final static double EPS = 1E-2;
18    final static double EPSValueForMinimumCheck=1E-10;
19    // variable end here
20 }
```

Figure 5: Missing JavaDoc comments in methods and variable declarations

TangentFunctionTest.java - Unit test methods in the class can have Javadoc comments to quickly understand the purpose of the test cases.

```
* Unit test cases for tangent function

package com.calculator.test;

import static org.junit.Assert.*;

public class TangentFunctionTest {

    @Test
    public void tanZeroCheck() {
        assertEquals("0", TangentFunction.getTan(0));
    }

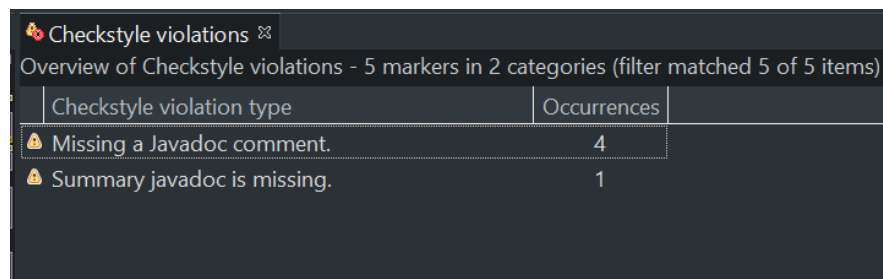
    @Test
    public void tanFortyCheck() {
        assertEquals("0.83910101", TangentFunction.getTan(40));
    }
}
```

Figure 6: Missing JavaDoc comments in test methods

Automatic Code Review

Automatic source code review is done by **CheckStyle**[1] plugin integrated with eclipse IDE which is available in the form of plugin.

Checkstyle inspects Java source code and pointing out items that deviate from a defined set of coding paradigms.



Checkstyle violation type	Occurrences
Missing a Javadoc comment.	4
Summary javadoc is missing.	1

Figure 7: TangentFunction.java - CheckStyle Violation

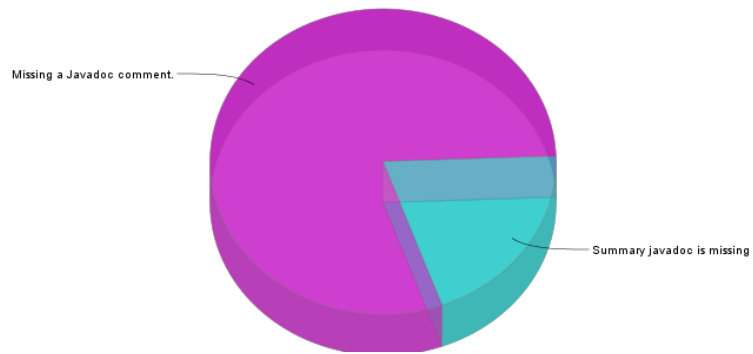


Figure 8: TangentFunction.java - CheckStyle Violation Graph

🔥 Checkstyle violations ✕

Overview of Checkstyle violations - 5 markers in 2 categories (filter matched 5 of 5 items)

Checkstyle violation type	Occurrences
🚨 Missing a Javadoc comment.	4
🚨 Summary javadoc is missing.	1

Figure 9: TangentFunctionTest.java - CheckStyle Violation

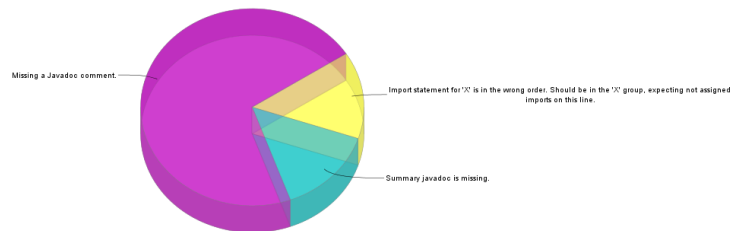


Figure 10: TangentFunctionTest.java - CheckStyle Violation Graph

Bibliography

- [1] CheckStyle. Eclipse Checkstyle Plugin. 2019.
<https://checkstyle.org/eclipse-cs>