SOEN 6011: SOFTWARE ENGINEERING PROCESSES

Project Deliverable 3

GitHub Repository : `https://github.com/Hkohli30/SOEN_6011`

Himanshu Kohli
Student Id : 40070839
Applied Computer Science
Concordia University

Problem 5 and 7 **Himanshu Kohli**
SOEN 6011 **40070839**
Prof. Pankaj Kamthan Due Date: August 02, 2019

1 Problem 5: Code Review

1.1 Information

- **Module Name:** B(x,y)
- **Developed By:** Shruthi Kondapura Venkataiah [40091427]
- **Code Review By:** Himanshu Kohli
- **Development End:** July 29,2019

1.2 Introduction

The relation between the set of input values and the output values is specified as the 'Euler Integral of first kind' more formally known as Beta Function. And the input is set of real values. The code review is a standard mechanism to avoid Code Smells and to improve the efficiency of the code. The coder used underlining specifications to write code.

1. **Non repeating code:** The code used structures in the code which are non-repeating which enhances readability and faster execution of code
2. **Proper commenting** : The coder wrote proper java docs which provided explanation of the function and global variables to understand the code. And avoided obscure commenting.

Figure 1: Proper commenting

```
/**
 * To get the logOfGamma of a value.
 * @param lg for any given value
 * @return logOfGamma of a value
 */
public static double logOfGamma(final double lg) {
 double part2;
 double part1;
 double result = 0;
 final double pi = 3.14;
```

3. **Avoided magic numbers:** The author avoided magic numbers all together in the code which leads to error and reduce efficiency
4. **One purpose variable:** Every variable declared has a specific purpose and is not used to hold any other type variables.
5. **Avoided global variables:** Only one global variable is specified in the program with proper modifiers such as static and final so as the functions can use the variable.
6. **Good names:** Most of the functions have variables with self-explanatory names which provide enough information for the reader to understand the code. For a function logOfGamma the two variables declared are doesnt have proper naming.

Figure 2: Improper naming

```
public static double logOfGamma(final double lg) {
 double part2;
 double part1;
 double part3 = 1 / (1188 * powerOf(lg, 9));
 double part4 = 1 / (1680 * powerOf(lg, 7));
 double part5 = 1 / (1260 * powerOf(lg, 5));
 double part6 = 1 / (360 * powerOf(lg, 3));
 double part7 = 1 / (12 * lg);
```

7. **Proper exception handling:** The coder wrote proper exception handling code which manages the code while run-time execution.

Figure 3: Exception Handling

```
try {
 while (temp > limitChecker) {
  result += temp;
  term *= (d - 1) / d;
  temp = term * (1.0 / denominator);
  denominator += 1;
 }
} catch (Exception e) {
 System.out.print("Coud not perfor logOfGreaterThan 1 for a given value");
}
```

8. **Methods returning proper results:** All the methods except main method take valid input argument and return the result to be processed and avoid printing statements int them.

Figure 4: Methods returning results

```
/**
 * To get the expoOfGamma of a value.
 * @param d for any given value
 * @return expoOfGamma of a value
 */
public static double expoOfGamma(final double d) {
 double expoOfG = 0;
 try {
  double logOfG = logOfGamma(d);
  expoOfG = exponential(logOfG);
 } catch (Exception e) {
  System.out.println("There's something wrong while doing expoOfGamma");
 }
 return expoOfG;

}
```

9. **Modularity:** Beta Function uses lots of calculations and every function is divided into separate functional modules which makes modification, maintenance and bug detection easier.

1.3 Coding Standards

1. **Block statements:** The block statement and the presentation of the curly braces is identical to the pre-decided coding standards

2. **Self-descriptive variables:** Some of variable names could have been more descriptive for the reader to understand their proper working.

3. **Proper import statements:** The coder use proper import statements while avoiding the .* statements.

4. **Proper spacing:** Instead of using tab the user has used the 'space bar' which is not according to the pre-decided coding standards.

5. **Naming convention:** The team pre-decided camel-casing methodology for declaring class names, function names and variables which were properly followed in the code.

Figure 5: Naming Conventions

```
public static double getBetaValue(final double x, final double y) {

 double result = (expoOfGamma(x) * expoOfGamma(y)) / expoOfGamma(x + y);
 return result;

}
```

2 Problem 7: Function Testing

2.1 Introduction

The function x^y where x and y are variables and the domain is set of real numbers.

- **Module Name:** x^y
- **Developed By:** Sanchit Kumar [40081187]
- **https://github.com/san089/SOEN-6011**
- **Code Review By:** Himanshu Kohli
- **Development End:** July 29,2019

2.2 Testing Reports

2.2.1 MainTest.java

All the functions check the input with (x^y)

- **Test Case ID:** TC1
  Type: Functional
  Function Name: ZeroRaisedToZero
  Input x,y: 0,0
  Expected Result: 0
  Original Result: 0
  Test Result: PASS
- **Test Case ID:** TC2
  Type: Functional
  Function Name: ZeroRaisedToNegative
  Input x,y: 0,-55
  Expected Result: 0
  Original Result: 0
  Test Result: PASS
- **Test Case ID:** TC3
  Type: Functional
  Function Name: negativeBaseRootError
  Input x,y: -2,-1.99
  Expected Result: NOTHING
  Original Result: NOTHING
  Test Result: PASS

- **Test Case ID:** TC4
  Type: Functional
  Function Name: notANumberInput
  Input x,y: xyz
  Expected Result: Exception [invalid input]
  Original Result: Exception [invalid input]
  Test Result: PASS

2.2.2 pow.java

- **Test Case ID:** TC5
  Type: Functional
  Function Name: getSignToMultiply
  Input x,y: 100,5
  Expected Result: 1E10
  Original Result: 1E10
  Test Result: PASS

- **Test Case ID:** TC6
  Type: Functional
  Function Name: getPowResult
  Input x,y: 100,5
  Expected Result: 1E10
  Original Result: 1E10
  Test Result: PASS

- **Test Case ID:** TC7
  Type: Functional
  Function Name: getPowResultTimeTest
  Input x,y: 311,5
  Expected Result: 1E10
  Original Result: 1E10
  Test Result: PASS

- **Test Case ID:** TC8
  Type: Functional
  Function Name: getExponent
  Input x,y: 16666666666,20
  Expected Result: 2.73511123E204
  Original Result: 2.73511123E204
  Test Result: PASS

- **Test Case ID:** TC9
  Type: Functional
  Function Name: getLn
  Input x: 2150
  Expected Result: 4.07347147E74
  Original Result: 4.07347147E74
  Test Result: PASS

- **Test Case ID:** TC10
  Type: Functional

Function Name: getSmallerValue
Input number,pointValue 123682638168368126386816283 6,28
Expected Result: 0.123682638168368126386816283 6
Original Result: 0.123682638168368126386816283 6
Test Result: PASS

- **Test Case ID:** TC11
  Type: Functional
  Function Name: numSignificantDigits
  Input number 10000.00000
  Expected Result: 5
  Original Result: 5
  Test Result: PASS

- **Test Case ID:** TC12
  Type: Functional
  Function Name: getXValForLog
  Input number -0.8181818182
  Expected Result: 0.1
  Original Result: 0.1
  Test Result: PASS

- **Test Case ID:** TC13
  Type: Functional
  Function Name: getPower
  Input x,y 2.42,2
  Expected Result: 5.8564
  Original Result: 5.8564
  Test Result: PASS

2.2.3 Summary Table

Table 1: Test case summary table.

Test Case ID	Test Case Status
TC1	PASS
TC2	PASS
TC3	PASS
TC4	PASS
TC5	PASS
TC6	PASS
TC7	PASS
TC8	PASS
TC9	PASS
TC10	PASS
TC11	PASS
TC12	PASS

2.2.4 Remarks

1. For all the test cases there were no JavaDocs writeen by the coder.
2. For functions of TC1 and TC2 there is voilation of coding standards.
3. There is use of BigDecimal variables and BigDecimal library in the coding.
4. The results are highly accurate with the calculator results.

References

[1] Robert L. Glass: Facts and Fallacies of Software Engineering; Addison Wesley, 2003. ISBN-13: 978-0321117427.

[2] Oracle Corporation. Code Conventions for the Java Programming Language. `http://www.oracle.com/technetwork/java/codeconvtoc-136057.html`

[3] Google Inc. Google Java Style. `https://google.github.io/styleguide/javaguide.html`

[4] Peter Grogono. Course notes for COMP6441: Advanced Programming Practices. Concordia University, 2007.

[5] Reading 4: Code `http://web.mit.edu/6.005/www/fa15/classes/04-code-review/`

[6] Unit Testing with JUnit - Tutorial `https://www.vogella.com/tutorials/JUnit/article.html`