# SOEN 6011 Deliverable 2 for Question 4 and 6

Anqi Wang 40057695

July 27, 2019

F8 B(x, y)
GitHub Anqi Wang's individual repo address: https://github.com/AnqiAngelineWang/SOEN6011.git

## 1 Correctness

This source code calculates Beta Function B(X, Y), for $x, y \subset (0, +\infty)$. The theory behind this program is based on mathematics integral calculation. The answer is obtained through solving mathematics integral by using java code implementation. This calculation does not import any packages, nor any build-in java functions. It combines empirical equation into Beta function:
$B(x, y) = \Gamma(x)\Gamma(y)/\Gamma(x + y)$
$B(x, y) = \int_0^\infty t^{x-1} \times (1 - t)^{y-1} dt$
This can generate accurate answers. For this program, the maximum length allowance number is followed double data type 1.79E308. It has the accuracy for almost 15 digits after decimal point. For calculating input value greater than 171.624, the answer approaches to infinity.

## 2 Efficiency

This program works with double datatype. This is a primitive datatype and it does not require much system memories. Generally speaking, for all values satisfying within the valid input range, and working on an average speed computer in 21 centuries, the program can generate results in milliseconds. The accuracy also decides the code efficiency. This program can estimate the solution in the accuracy of 15 digits after decimal point.

## 3 Maintainability and Program Style

This source code has followed the standard Google Java Style Guide, which is corresponding to the whole team's program style. The team has confirmed

that source code file structure is correspondence. Naming convention has applied. This provides a strong foundation for future maintenance. It is also easier for other developers to follow the current progress. Besides, documentation is necessary for engineers to trace back the developing history, which is an important resource to set up current maintainable decisions. This Beta Calculator program generates Javadoc, such as public class declaration and methods explanation have been added. What's more, copyright information has been included as well. On the other hand, corresponding team's program style has kept with the same format, block indentation and statement has been regulated. The code follows line wrapping and line break strictly. There is no extra whitespace. Exactly each section has one blank line to separate. For annotations, override, and static members have been used carefully with quality. Thus, by following the standard programming style, readability and maintainability have increases. Software engineers are easy to understand and catch up with other people's work.

# 4 Robustness

Robustness is essential for a program, especially after the program has been released to market. This program has been built based on mathematics theory, which is a strong supporting background for Beta Function analyzation. Also, this program's algorithm chooses 64 bit double data type. Double data type has 15 decimal digits of precision. This solution is robust and trustful. Moreover, Beta Function calculator is built by Java language and it is object-oriented. This program has a simple hierarchy inheritance. The architecture is robust. This robustness program is an advantage for future refactor or development.

# 5 Feasibility, Error handling, Usability

This program has a clear user interface. It is straight forward for user to understand the logic and instructions. This program also kindly reminds user if he has input a valid value. If error occurs, error handling exceptions shall be invoked. Try and catch blocks keep the program functioning, and exceptions have thrown error messages to remind user. This program also paid attentions in feasibility for user in the design phase, and has built to pursuit user-friendly aspect.

# 6 Debugger

There are three debuggers have been tried for this project during implementation. But, Jetbrains InteliJ Idea IDE build-in debugger is the major tool. It has several advantages. It is an efficient tool to find mistakes in the code. It can place breakpoints at any statements. It supports the quick glance by hovering mouse onto code lines, or clicking onto the breakpoint to see its properties in the

tooltip. This helps developers to suspend the execution of code. A detailed step by step processes will be displayed during debugging. Using this tool during implementation is essential for programmers to solve problems. This procedure is good for figuring out tiny mistakes as well. Also, breakpoint's properties are easy to be changed even after it has been placed. There is a hot key for programmers to relocate breakpoints in their codes and continue onto solving other issues. What's more, this debugger has a clear interface and straight forward working logic. Users can set their own debugging configurations based on their specific situation. The stepping toolbar contains options in Step Over, Step Into, Force Step Into, Step Out, Drop Frame and Run to Cursor. This support programmers to manipulate the code. They can also use Evaluate Expression and Trace Current Stream Chain. In the debugging environment, each step shows the current value, the memory address and the times this line of code has been invoked. You can also see Java Virtual Machine status during debugging. Although debugger is a good tool and assists you while coding, it reduces your abilities to think. Debuggers may point out the problem immediately, but your brain will get used to follow the instructions, and you will rely onto it as time goes. Your problem-solving abilities will decrease.

# 7 Checkstyle

Last but not least, source code quality is important. This source code Beta Function calculator is a java program. This program uses Checkstyle development tool during software implementation. There is a Checkstyle plugin to add into IDE. This tool is able to check coding standard automatically. This makes software developer's work easier by indicating design problems, like code formats and layout. Furthermore, java is an object-oriented programming language. Multiple classes and methods are this program's feature. Checkstyle tool can also help with auditing code structure in classes and methods. For example, while implementing this program, Checkstyle plugin reminds the programmer that the format is not regulated. This diminishes the chances for problems to occur. However, the disadvantages for Checkstyle is that this tool can not determine the full hierarchy of the program. If the java program has multiple inheritance, this tool may not be applicable.

# 8 JUnit Testing

This program contains Java JUnit testing. JUnit testing is significant for developers to test the software's properties. This program contains two files, the calculation .java file (F8 Beta Function.java) and Junit assert .java file (AssertTests.java). In order to satisfy client's requirement, and matches with user assumptions, detailed Junit Assert test cases have been attached to the program. The program has passed all test cases. Software testing is believed to be useful to check program accuracy, satisfy requirements, and increase soft-

ware robustness. Test cases ID and user (assumptions) requirements ID have matched, and explain below:

Test case ID: 1
Description: Requirement 1 has matched with Test case 1
Summary of Test: It tests if user has a valid input, not a character. The user has input a String "1.0". It uses assertFalse to check answer. So, the answer should receive "false" to pass the result.
Test Data: "1.0"
Expected Result: false
Actual Result: false
Status (Fail/Pass): Pass

Test case ID: 2
Description: Requirement 2 has matched with Test case 2
Summary of Test: It tests if user has an input number, which is not in the Beta Function's acceptable range. The user has input 0.1. User should not input a negative value. It uses assertFalse to check answer. So, the answer should receive "false" to pass the result.
Test Data: 0.1
Expected Result: false
Actual Result: false
Status (Fail/Pass): Pass

Test case ID: 3
Description: Requirement 3 has matched with Test case 3
Summary of Test: It tests if user has an input number, which is a complex number. The system couldn't calculate complex number. The user has input PI, which is a complex number. It uses throw exception: IllegalArgumentException to check answer. So, the answer has received correct exception.
Test Data: Math.PI
Expected Result: The system couldn't calculate complex number, only real number
Actual Result: The system couldn't calculate complex number, only real number
Status (Fail/Pass): Pass

Test case ID: 4
Description: Requirement 4 has matched with Test case 4
Summary of Test: It tests if user has an input number, which is a very large number that has out of the machine memory. This system can not support for this calculation. It uses throw exception: IllegalArgumentException to check answer. So, the answer has received correct exception.
Test Data: Math.pow(999999999, 999999999)
Expected Result: The input value is too large that has out of the machine memory

Actual Result: The input value is too large that has out of the machine memory"
Status (Fail/Pass): Pass

Test case ID: 5
Description: Requirement 5 has matched with Test case 5
Summary of Test: It tests if user has an input number, which is a very big number that the system couldn't handle. This system can not support for this calculation. It uses throw exception: IllegalArgumentException to check answer. So, the answer has received correct exception.
Test Data: Math.pow(9999,99999)
Expected Result: The input a big number that the system couldn't handle
Actual Result: The input a big number that the system couldn't handle
Status (Fail/Pass): Pass

Test case ID: 6
Description: Requirement 6 has matched with Test case 6
Summary of Test: It tests if user has an input number, and modified again. This program will calculate based on the first input. User could not modify the input again after his first input
Test Data: 3.5
Expected Result: 1.2009736023470743
Actual Result: 1.2009736023470743
Status (Fail/Pass): Pass

# 9 Reference

G. (n.d.). Google Java Style Guide. Retrieved July 26, 2019, from
*https* : *//google.github.io/styleguide/javaguide.html*

Jetbrains. (n.d.). Debugging Your First Java Application. Retrieved July 26, 2019, from
*https* : *//www.jetbrains.com/help/idea/debugging* − *your* − *first* − *java* − *application.html*

Ivanov, R. (2019, June 22). Checkstyle Overview. Retrieved July 26, 2019, from
*https* : *//checkstyle.sourceforge.io/*