SOEN 6011 - Scientific Calculator

Saraswati Saud
Student ID: 40115097

1 Problem 1: Gamma function, Γ (x)

1.1 Description

The gamma function represented by Γ(x), is an extension of the factorial function to complex numbers. It is defined for all the complex numbers except the non-positive integers.

Specifically, Γ(n) = (n-1)! where n > 0.

1.2 Domain

Any complex number that is not a negative integer is in the domain of the function.
So, the domain of the Gamma function is (0, $+\infty$).

1.3 Co-domain

The co-domain is the set of all real numbers ($-\infty$, $+\infty$).

1.4 Characteristics of Gamma Function

1. The gamma function is uniquely defined for all positive integers and complex numbers with positive real parts.
2. For real values of argument 'n', the value of the gamma function Γ(n) are real (or infinity). The gamma function is not equal to zero.
3. Γ(n+1) = n!, for integer n > 0.
4. Γ(n+1) = n Γ(n) (function equation).

References

[1] MathWorld: Gamma Function
`https://mathworld.wolfram.com/GammaFunction.html`

[2] Geeksforgeeks
`https://www.geeksforgeeks.org/gamma-function`

2 Problem 2: Requirements Specification

2.1 Definitions and Abbreviations

Terms	**Definitions**
FR	Functional Requirement
NFR	Non-Functional Requirement
User	Someone who interacts with the system
System	Software Program for calculation of Gamma Function

2.2 Constraints and Assumptions

- User should provide input for ‘x’.
- Based on the function characteristics, the value of ‘x’ should either be positive integer or half-integer value.
- The input value cannot be 0.
- The output will always be a positive decimal.

2.3 Requirements

2.3.1 Functional Requirements

ID:	FR1
TYPE:	Functional
VERSION:	1.0
DIFFICULTY:	Easy
PRIORITY:	1
OWNER:	Saraswati Saud
DESCRIPTION:	System should prompt the user to enter the value of ‘x’.
RATIONALE:	To get user input and start calculation.

ID:	FR2
TYPE:	Functional
VERSION:	1.0
DIFFICULTY:	Medium
PRIORITY:	2
OWNER:	Saraswati Saud
DESCRIPTION:	System should display an error message when entered value is not a number.
RATIONALE:	For calculations, input should only be numbers.

ID:	FR3
TYPE:	Functional
VERSION:	1.0
DIFFICULTY:	Difficult
PRIORITY:	1
OWNER:	Saraswati Saud
DESCRIPTION:	System should display an error message if a user entered negative, zero or positive non-half integer.
RATIONALE:	For calculations, input should only be positive integer or half-integer value.

ID:	FR4
TYPE:	Functional
VERSION:	1.0
DIFFICULTY:	Easy
PRIORITY:	3
OWNER:	Saraswati Saud
DESCRIPTION:	User should have an option to exit the program anytime during the use.
RATIONALE:	If user is done with the calculations.

2.3.2 Non-Functional Requirements

ID:	NFR1
TYPE:	Non-Functional
VERSION:	1.0
DIFFICULTY:	Medium
PRIORITY:	2
OWNER:	Saraswati Saud
DESCRIPTION:	The error message displayed should be appropriate and helpful for the user.
RATIONALE:	User should be able to know what went wrong.

ID:	NFR2
TYPE:	Non-Functional
VERSION:	1.0
DIFFICULTY:	Easy
PRIORITY:	3
OWNER:	Saraswati Saud
DESCRIPTION:	The text-based interface should be user friendly.
RATIONALE:	It should be easy for the user to use the system.

ID:	NFR3
TYPE:	Non-Functional
VERSION:	1.0
DIFFICULTY:	Medium
PRIORITY:	2
OWNER:	Saraswati Saud
DESCRIPTION:	The displayed result should be as accurate as possible.
RATIONALE:	Incorrect output should not be displayed.

ID:	NFR4
TYPE:	Non-Functional
VERSION:	1.0
DIFFICULTY:	Difficult
PRIORITY:	1
OWNER:	Saraswati Saud
DESCRIPTION:	Calculation time should be less than 1 second.
RATIONALE:	Waiting a long time for the output might not be desired for the user.

3 Problem 3: Algorithm and Pseudocode

Following are the algorithms and the pseudo-codes of function F4, $\Gamma(x)$:

Algorithm 1 Recursive Approach - $\Gamma(x)$

```
procedure $functionF4(x)$
    in: double x
    out: double result
    if (x < 0) then return "Wrong Input"
    else
        if (x == x.5) then
            result = halfFactInteger(x)
        else
            result = factInteger(x)
        return result

procedure $halfFactInteger(x)$
    in: double x
    out: double result
    if (($x == 0.5$) then
        return 1.77
    else
        return $(x - 1) * halfFactInteger(x - 1)$

procedure $factInteger(x)$
    in: double x
    out: double result
    if (($x == 1$) then
        return 1
    else
        return $(x - 1) * factInteger(x - 1)$
```

Algorithm 2 Iterative Approach - $\Gamma(x)$

```
procedure functionF4(x)
    in: double x
    out: double result
    if (x < 0) then return "Wrong Input"
    else
        if (x == x.5) then
            result = halfFactInteger(x)
        else
            result = factInteger(x)
        return result

procedure halfFactInteger(x)
    in: double x
    out: double result

    double fact = 1.77
    double result = 1.0

    for (int i = 1; i < x; i++) do
        double value = x - i
        result *= value
    return fact * result

procedure factInteger(x)
    in: double x
    out: double result

    double fact = 1.0

    for (int i = 1; i < x; i++) do
        fact = fact * i
    return fact
```

3.1 Algorithm Description

3.1.1 Algorithm 1

Following are the details of algorithm 1:
Time Complexity: O(n)
Space Complexity: O(n)
Approach: Recursion

Advantages

- Reduces time complexity.
- Adds clarity and reduces the time needed to write and debug code.

- Reduces unnecessary calling of function and length of a code.

Disadvantages

- Recursion is usually slower due to the overhead of maintaining the stack.
- It usually uses more memory for the stack.
- Recursive methods will often throw a StackOverflowException when processing big sets.

3.1.2 Algorithm 2

Following are the details of algorithm 2:
Time Complexity: O(n)
Space Complexity: O(1)
Approach: Iterative

Advantages

- Algorithm 2 does not suffer from stack overflow because all operations are conducted on the heap.
- The space complexity of Algorithm 2 is O(1).

Disadvantages

- An infinite loop for iteration occurs when the condition never fails.
- Not efficient for larger inputs as it requires more time to execute.

3.2 Conclusion

Algorithm 1 has greater space requirements than Algorithm 2 as all the functions will remain in the stack until the base case is reached. In addition to this, algorithm 1 (i.e. recursive approach) has greater time requirements because of function calls and returns overhead. Therefore, iterative algorithm is preferred over recursive approach.

References

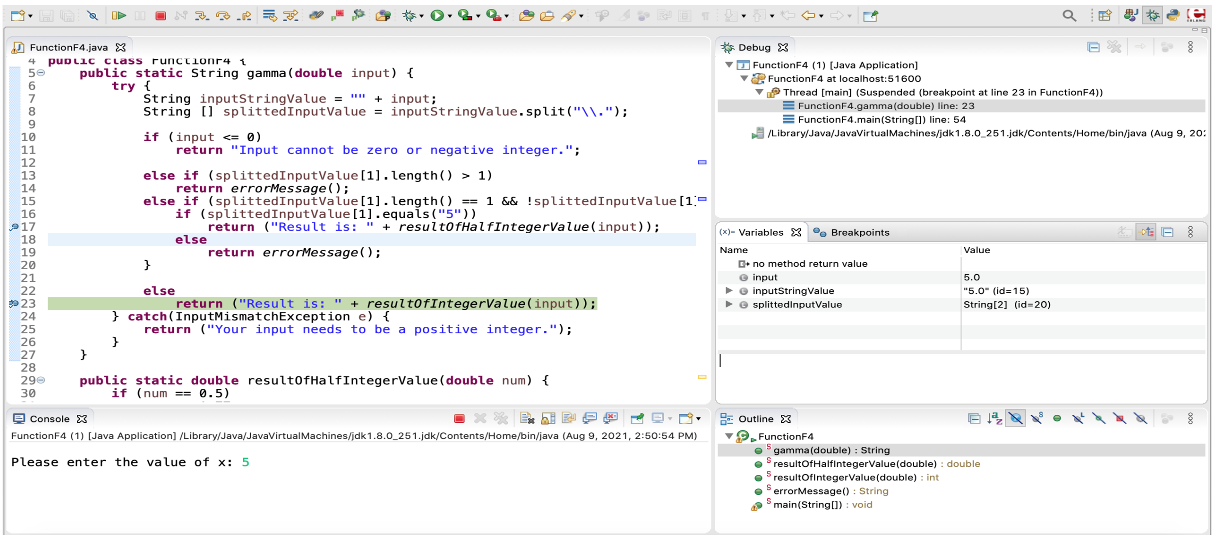
[1] Medium: Recursion - Pros and Corns
`https://medium.com/@williambdale/recursion-the-pros-and-cons-76d32d75973a`

[2] Geeksforgeeks: Recursive Function,
`https://www.geeksforgeeks.org/recursion/`

4 Problem 4: Debugger, Quality Attributes and Check-Style

4.1 Eclipse Debugger

Eclipse allows us to start a Java program in Debug mode. So, I am using the inbuilt Eclipse debugger for debugging my code. I have chosen Eclipse debugger mainly for two reasons. First, it is free and open source. Second, because of its familiarity as I have been using Eclipse IDE for a very long time.


FunctionF4.java

```
 4  public class FunctionF4 {
 5      public static String gamma(double input) {
 6          try {
 7              String inputStringValue = "" + input;
 8              String [] splittedInputValue = inputStringValue.split("\\.");
 9
10              if (input <= 0)
11                  return "Input cannot be zero or negative integer.";
12
13              else if (splittedInputValue[1].length() > 1)
14                  return errorMessage();
15              else if (splittedInputValue[1].length() == 1 && !splittedInputValue[1
16                  if (splittedInputValue[1].equals("5"))
17                      return ("Result is: " + resultOfHalfIntegerValue(input));
18                  else
19                      return errorMessage();
20              }
21
22              else
23                  return ("Result is: " + resultOfIntegerValue(input));
24          } catch(InputMismatchException e) {
25              return ("Your input needs to be a positive integer.");
26          }
27      }
28
29      public static double resultOfHalfIntegerValue(double num) {
30          if (num == 0.5)
```

Debug

```
FunctionF4 (1) [Java Application]
  FunctionF4 at localhost:51600
    Thread [main] (Suspended (breakpoint at line 23 in FunctionF4))
      FunctionF4.gamma(double) line: 23
      FunctionF4.main(String[]) line: 54
  /Library/Java/JavaVirtualMachines/jdk1.8.0_251.jdk/Contents/Home/bin/java (Aug 9, 20
```

Variables

Name	Value
no method return value	
input	5.0
inputStringValue	"5.0" (id=15)
splittedInputValue	String[2] (id=20)

Console

```
FunctionF4 (1) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_251.jdk/Contents/Home/bin/java (Aug 9, 2021, 2:50:54 PM)

Please enter the value of x: 5
```

Outline

```
FunctionF4
  gamma(double) : String
  resultOfHalfIntegerValue(double) : double
  resultOfIntegerValue(double) : int
  errorMessage() : String
  main(String[]) : void
```

4.1.1 Advantages

- It is free and open source.
- It is easy to use – setting breakpoints, executing code line by line, and inspecting the value of variables and expressions.
- It also supports other programming languages than Java for investigating and fixing problems with the code.

4.1.2 Disadvantages

- It is usually slower than other debuggers like Visual Studio.
- The independent processes cannot be attached to the eclipse debugger to debug.

4.2 Quality Attributes

Following are the efforts that has been made in achieving the quality attributes:

4.2.1 Correctness

- 'Standard' programming style for the source code has been followed.
- Related test cases have been added on the assigned functions.

4.2.2 Efficient

- Program takes less time to execute.
- Unnecessary variables, methods and the statements have been removed.

4.2.3 Maintainability

- Programming style has been made identical throughout the team to remove ambiguity.
- Necessary comments have been added where required.

4.2.4 Robustness

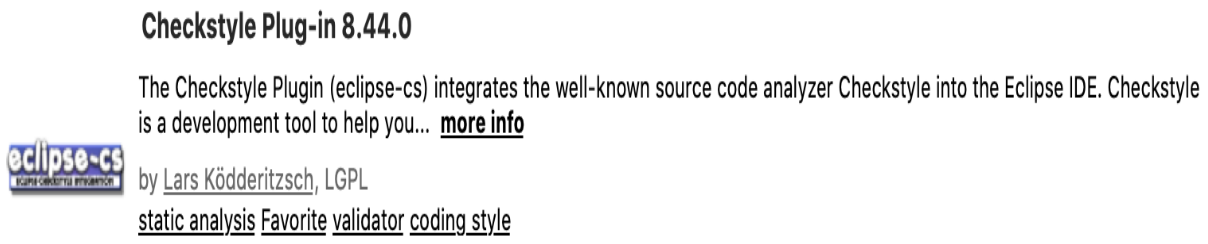
- Proper error handling has been done for the wrong user inputs.
- Use of Try…Catch block for handling java exceptions.

4.2.5 Usable

- Simple textual interface has been added for user input and output.
- Proper error messages have been given for the wrong inputs.
- Proper display of an output has been done for the right inputs.

4.3 CheckStyle

I have used the Eclipse plugin - Checkstyle Plug-in 8.44.0 to check the quality of my source code. It is a development tool that helps programmers to write Java code that follows good programming standards. It helps to improve the quality, readability, re-usability of the code and may reduce the development cost.



Checkstyle Plug-in 8.44.0

The Checkstyle Plugin (eclipse-cs) integrates the well-known source code analyzer Checkstyle into the Eclipse IDE. Checkstyle is a development tool to help you... more info

by Lars Ködderitzsch, LGPL

static analysis Favorite validator coding style

4.3.1 Advantages

- It is portable between IDEs.
- It is easier to integrate with external tools as it was designed as a standalone framework.
- It has an ability of creating own rules. Eclipse has a large set of styles but checkstyle has more and we can add our own custom rules.

4.3.2 Disadvantages

- Checkstyle is limited to a single file static analysis tool.
- It is also limited to the presentation of the code and hence does not confirm the correctness or completeness of the code.

5 Problem 5: Source code review of Function F5, ab^x

5.1 Introduction

The function provided for the source code review purpose was ab^x is a power function. The code was reviewed based on the coding standards that the developer has followed while writing the code. For this, I reviewed coding practices, non-functional requirements and overall structure of the code.

5.2 Code Review Checklist

Following are the checklists of the standard code review guidelines which was used for reviewing the source code:

1. Don't repeat yourself (Avoid duplication)
2. Functionality of the code should work properly
3. Use of meaningful class, method and variables names
4. Don't ignore the exception in the code
5. Avoid creating unnecessary class, methods, variables and objects.
6. Proper comments have been added
7. Unnecessary packages should not be imported
8. Class, function should not be too lengthy
9. Code must have unit test cases
10. Proper error message should be displayed

5.3 Review Approach

Several steps were followed while reviewing the source code.

- First, the code was reviewed well enough to check the naming convention, proper comments in the class and methods, and test cases.
- Second, the code was run to check whether the functionality properly works or not.
- Third, test cases were run to check whether all the test cases pass or not.

5.4 Code Review Results

Checklist	Description	Result
1	Duplicate Code	PASS
2	Functionality of the code	PASS
3	Naming Convention	PASS
4	Exception Handling	PASS
5	No unnecessary class, methods and variables	PASS
6	Useful comments	PASS
7	No unnecessary imports	PASS
8	No lengthy code	PASS
9	Must have unit test cases	PASS
10	Proper error message	PASS

5.5 Conclusion

From the above, it is noted that the code is well written and all the necessary quality attributes are fulfilled. The only flaw I could found was missing few comments in the function body.

6 Problem 6: Unit Testing - Standard Guidelines

6.1 Description

Following are the standard guidelines for implementing unit testing for function F4, $\Gamma(x)$:

- Test case contains the function name used for the unit test case
- Test shall cover all cases such as actual and expected value.
- For readability, each test case must show proper display message.
- Each test cases must be traceable.
- Test cases must map all the functional requirements.

6.2 Implemented Unit Test

- Junit – a unit testing framework is used.
- The sample practice used in the test cases are as follows:
  ID: TC1
  Test Case: Description about the test case

7 Problem 7: Testing of Function F7, x^y

7.1 Introduction

The function provided for the testing purpose is x^y which is a power function. The team member has provided all the required documents to perform the testing of a function.

7.2 Test Step

1. The source code is imported into Eclipse and compiled.
2. The program is run for different inputs and results were verified.

7.3 Test Results

Requirements	Description	Result
R1	Valid inputs within the given range	PASS
R2	Any base value is raised to the power zero	PASS
R3	Base value zero is raised to any exponent value	PASS
R4	Non-numeric throws exception	PASS

7.4 Conclusion

From the above, all the methods of power function are tested and found a positive response. Also, there are no errors found during the testing process. The test cases cover all the requirements as specified in Problem 2. In addition to this, test cases are fast and precise.