Concordia University

SOEN 6011 - Software Engineering Process

ETERNITY: FUNCTIONS
tax(x)

Deliverable final

Apekshaba Gohil
Student ID : 40203058

https://github.com/Apekshaba/SOEN-6011

Problem 1

1 Introduction

F2: $f(x) = tan(x)$ is a trigonometric tangent function where x is an angle at which the tangent is found. Here, x is an angle in rad. This function is one of the most commonly used function in mathematics.

2 Domain & Co-Domain

2.1 Domain

- it includes all the real numbers, but not the values for x where cos(x) is zero, because tan can also be written as the ratio of sine and cosine functions. So the domain will contain all the real numbers excluding $(2n + 1)\frac{\pi}{2}$

2.2 Co-Domain

- For the negative value of an angle ,the tangent will also be negative.
- For the positive values of an angle, the tangent will also be positive.

2.3 Range

- The range of the tan function is R where, R is a set of all real numbers.

2.4 Restrictions

- for the angle value of 90°tan is not defined.

3 Characteristics

- ***Growth*** :When the value of angle lies between every alternate section of $\frac{\pi}{2}$ the tangent function seems to be increasing until it reaches the multiple of 90°, where it's not defined.
- ***Decay*** : When the value of angle starting from $\frac{\pi}{2}$ to the next subsection of $\frac{\pi}{2}$, tangent function seems to be decreasing until it reaches the undefined state. as shown in the $Figure$[1].
- **Injectivity & Surjectivity** : The tangent function is both injective and surjective and so it is called bijective function.

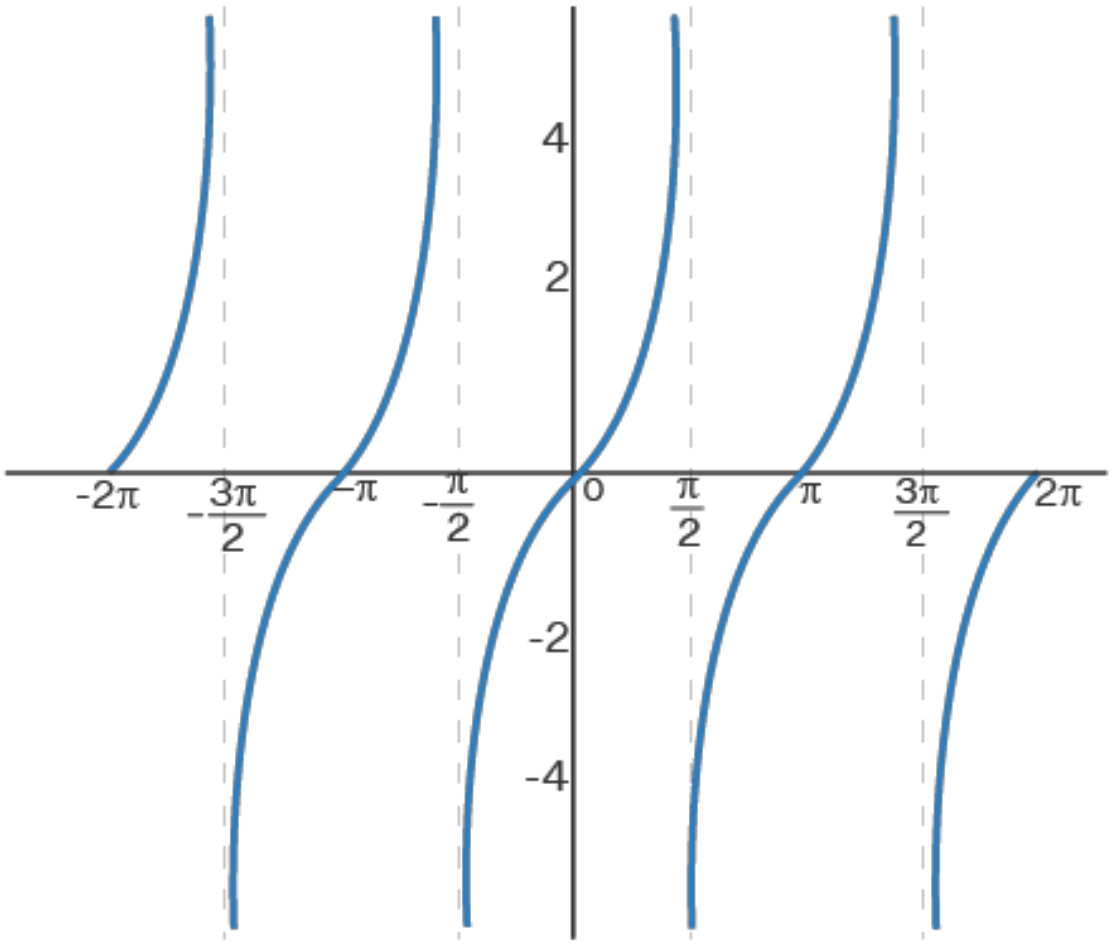

Figure 1: curve growth and Decay graph of tangent function.

Problem 2

Requirements

First Requirement

- **ID =** FR1
- **Type =** Functional Requirements
- **Version =** 1.0
- **Difficulty =** Easy
- **Description =** System shall take an input X which is nothing but the angle for which the tangent is to be found, ahich then is converted into radians and the output which is a tangent value for that value of angle is given.
  Example :for $x = 45, tan(x) = 0.99924342$

Second Requirement

- **ID =** FR2
- **Type =** Functional Requirements
- **Version =** 1.0
- **Difficulty =** Easy
- **Description =** $tan(x)$ is dependent on the $sin(x) and cos(x)$ functions which are expanded using Taylor series expansion.

Third Requirement

- **ID =** FR3
- **Type =** Functional Requirements
- **Version =** 1.0
- **Difficulty =** Easy
- **Description =** User shall give an input from all real numbers for x but the values for which cos(X) is zero.

Fourth Requirement

- **ID =** FR4
- **Type =** Functional Requirements
- **Version =** 1.0
- **Difficulty =** Easy

- **Description =** when user gives other input then integer that is string then system shall not accept and will ask to provide proper response after checking it using the Util function inValidRange.

Fifth Requirement

- **ID =** FR5
- **Type =** Functional Requirements
- **Version =** 1.0
- **Difficulty =** Easy
- **Description =** when user provides a value that is out of range for the Double variable's valid range, It says value out of range.

Problem 3

Pseudocode and Algorithm

Calculate: $f(x) = tan(x)$

Consider $tan(x) = \frac{sin(x)}{cos(x)}$

ALGORITHM 1: Iterative algorithm to calculate $tan(x)$ using Taylor series expansion

1. function **sin(x)**
in: double radian
out: double sinX
2. $pow \leftarrow 1$
3. $sinX \leftarrow 0.0$
4. for $Iterator \leq 15$ do
5. $current\_term \leftarrow 0.0$
6. if $Iterator \mod 2 == 0$
7. $current\_term \leftarrow -tan.power(radians, pow)$
8. else
9. $current\_term \leftarrow tan.power(radians, pow)$
10. $sinX \leftarrow sinX + current\_term$
11. $pow \leftarrow pow + 2$
12. end for
13. return sinX
14. function **cos(x)**
in: double radian
out: double cosX
15. $pow \leftarrow 0$
16. $cosX \leftarrow 0.0$
17. for $Iterator \leq 15$ do
18. $current\_term \leftarrow 0.0$
19. if $Iterator \mod 2 == 0$
20. $current\_term \leftarrow tan.power(radians, pow)$
21. else
22. $current\_term \leftarrow -tan.power(radians, pow)$
23. $cosX \leftarrow cosX + current\_term$
24. $pow \leftarrow pow + 2$
25. end for
26. return cosX
27. function **getFact(power)**
in: Integer power
out: Integer fact
28. $fact \leftarrow 1$
29. for $Iterator \leq power$ do
30. $fact \leftarrow fact * Iterator$
31. end for
32. return fact
33. function **power(radian, exponent)**
in: double radian, exponent
out: double res
34. $res = 1$
35. while $exponent! = 0$ do
36. $res = res * radian$
37. $--exponent$
38. end while
39. return res

Here, I keep track of our result in a variables called sinX and cosX in two different functions, which are initially set to be equal to 0.0. after that we are simply calling a power function for the angle x which keeps track of the results into *current_term* which was then appended to the final results be them sinX and cosX and returned at the time of a call from tan(X) function call

ALGORITHM 2: Recursive algorithms in order to receive the results of tanx using exponential functions

1.epowerx$x$
2.**for** $i \leftarrow 0$ *to* $maxdepth$ **do**
end
3.$result = 1 + x * result/i$
4. endFor
5. endProcedure
6.$epowerX = epowerx(x)$
7.$epowernegativeX = epowerx(-x)$
8.$tanX = (epowerX - epowernegativeX)/iterator(epowerX + epowernegativeX)$

Here, there's a common exponential function has been defined which returns the positive and negetive exponential power of constant e of x times, The result of tanX is finally stored using the function and exponential function based on the formula of $tan(x) = \frac{(e^{ix} - e^{-ix})}{i(e^{ix} + e^{-ix})}$

Advantages and Disadvantages

Algorithm 1:

Advantages:
1. In terms of space complexity, iterative algorithms don't suffer from stack overflow because all operations are done on the heap. and later alocated the space to store the final result
2. Easy to read and follow since the format is easy to understand.
Disadvantages:
1. The time complexity of the iterative algorithm is $O(n)$, hence it is not very efficient for larger inputs in terms of time.
2. Proper terminating condition for loop is needed or else we might get stuck in infinite loop.

Algorithm 2:

Advantages:
1. The time complexity of our recursive algorithm is $O(logn)$. Our version of recursive algorithm is optimized and tail recursive so that we don't get stack overflow error. here the implementation and soc is lesser and easier compare to iterative algorithm.
2. Recursion has higher maintainability than loop. There's little to no modification needed if we handle base case correctly
Disadvantages:
1. When using recursion, system continuously allocates memory space, which leads to the stack overflow issue.
2. The algorithm is highly crucial at the time of handling base cases and might misbehave at the time of edge cases. 3. It is not really very readable since most of the part is in binary mathematical format.

Conclusion

My decision of going with the Iterative algorithm using Taylor series was mostly because of code re-usability simple and modular design, even though it might take a little extra time than a recursive algorithm using Exponential function would take having the computer space managed quite well it has been proven the best decision.

Problem 4

3.1 Debugger

In the project, I have used the Intellij IDE debugger to debug the code. The Intellij IDE provides many debugging tools and views grouped in debug perspective to debug effectively and efficiently. this debugger provides functions to run the code step by step and track the values of variables while debugging.

Advantages and Disadvantages

Advantages:

• The values of variables can be changed at debugging mode on the fly.
• It has step filtering functions to step into or step over a statement
• It offers a feature to show the logical structure that allows viewing the object in another meaningful structure/view.
• It offers event based breakpoints.

Disadvantages:

• Debugging real time/multi-thread programs becomes harder as it involves huge lines of code and it's better to perform testing using test cases.

3.2 Quality Attributes:

3.2.1 correctness:

The program is tested for all the possible values of an angle Tangent function can return.

3.2.2 space-efficient:

The function implemented is space-effiecient since all the variables are tacked into heap and stored into memory at the time of passing it.

3.2.3 portable and maintainable:

The program is devided into three classes which make it effective and easy to understand and maintain, that is breakout into one class will be easier to handle without making any possible damage to the other files.

3.2.4 Robust:

The program is tested for all faulty inputs and outputs, and all possible outliers to check it's robustness.

3.2.5 Usable:

The program is made useable so that anyone can easily understand and make changes according to their requirements.

3.2.6 time-efficient:

The program is made to be less time consuming by efficiently using the loops to not interrupt abruptly.

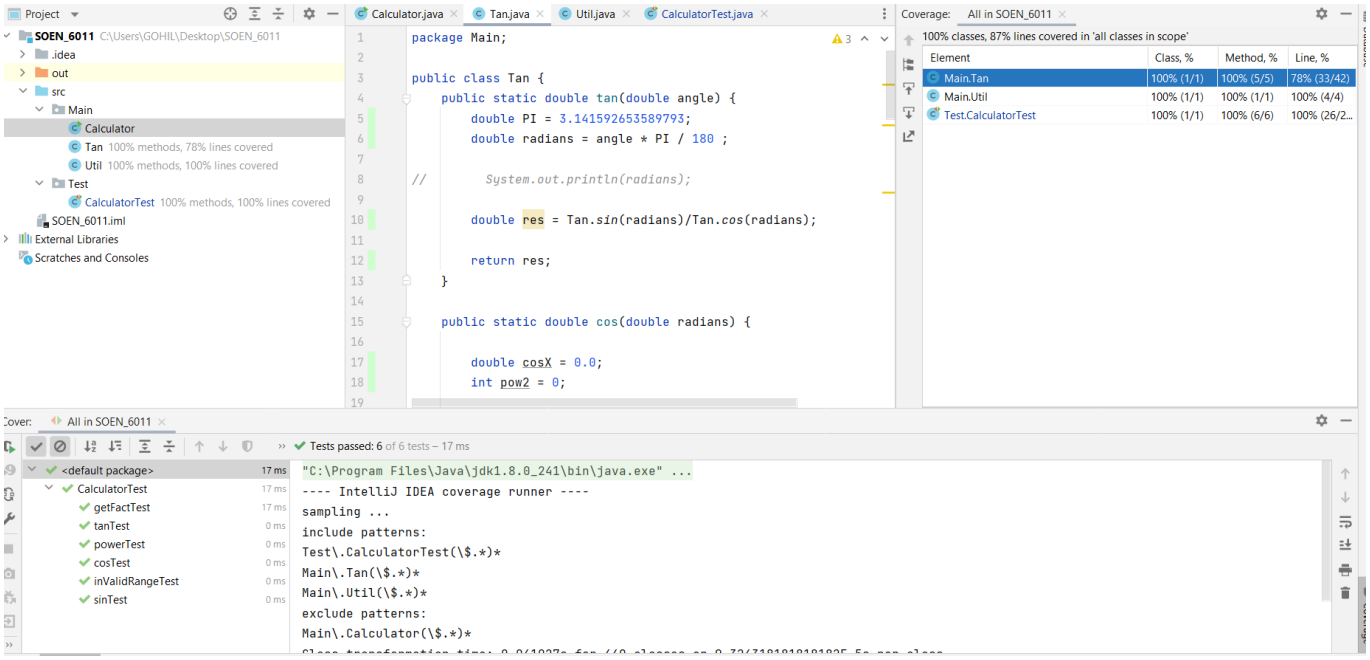

Figure 2: The snapshot of the debugger which has included the coverage of the functions and the display of the successful tests has been shown .

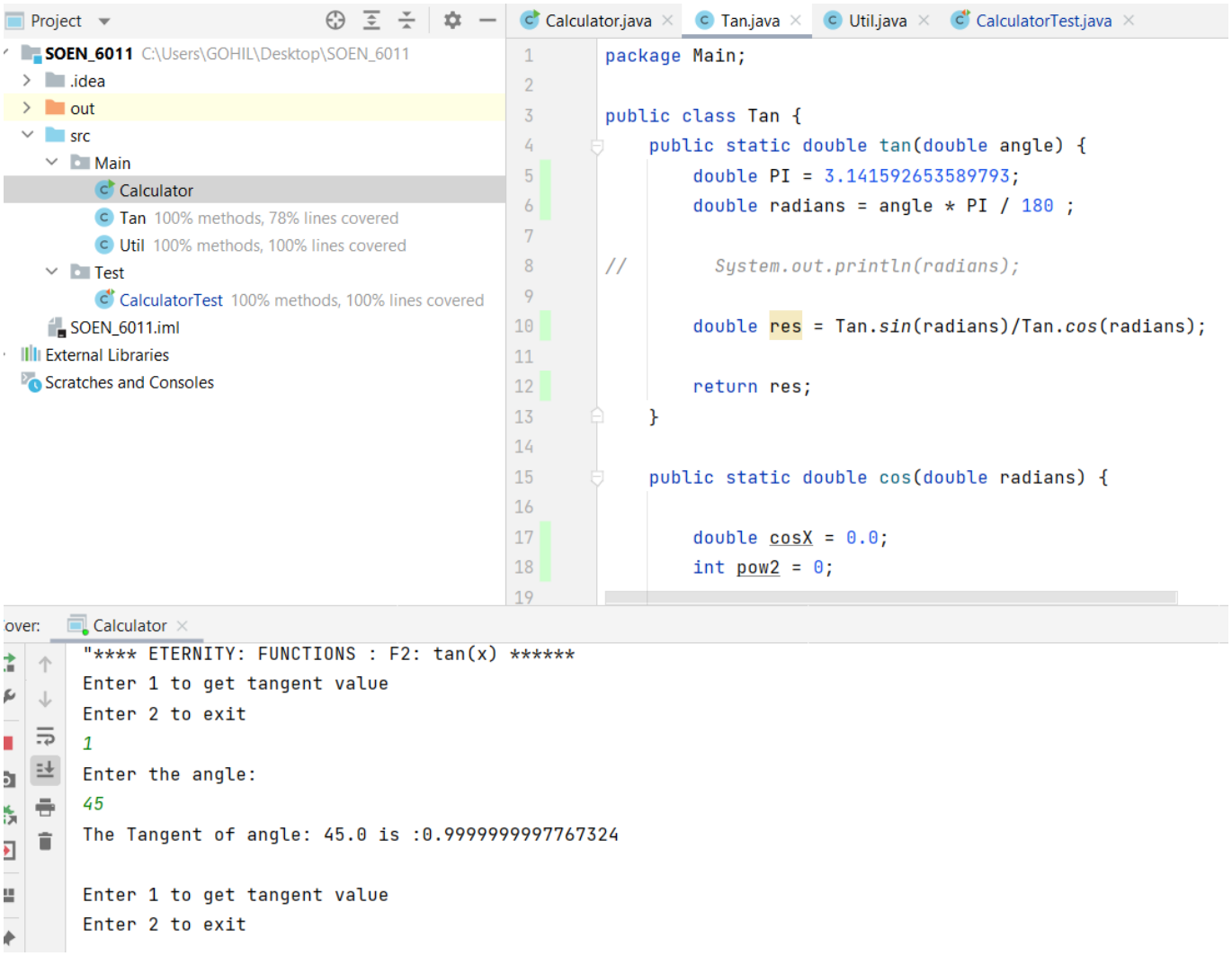

Figure 3: The snapshot of the debugger which has a case where it ran perfectly for the value in range

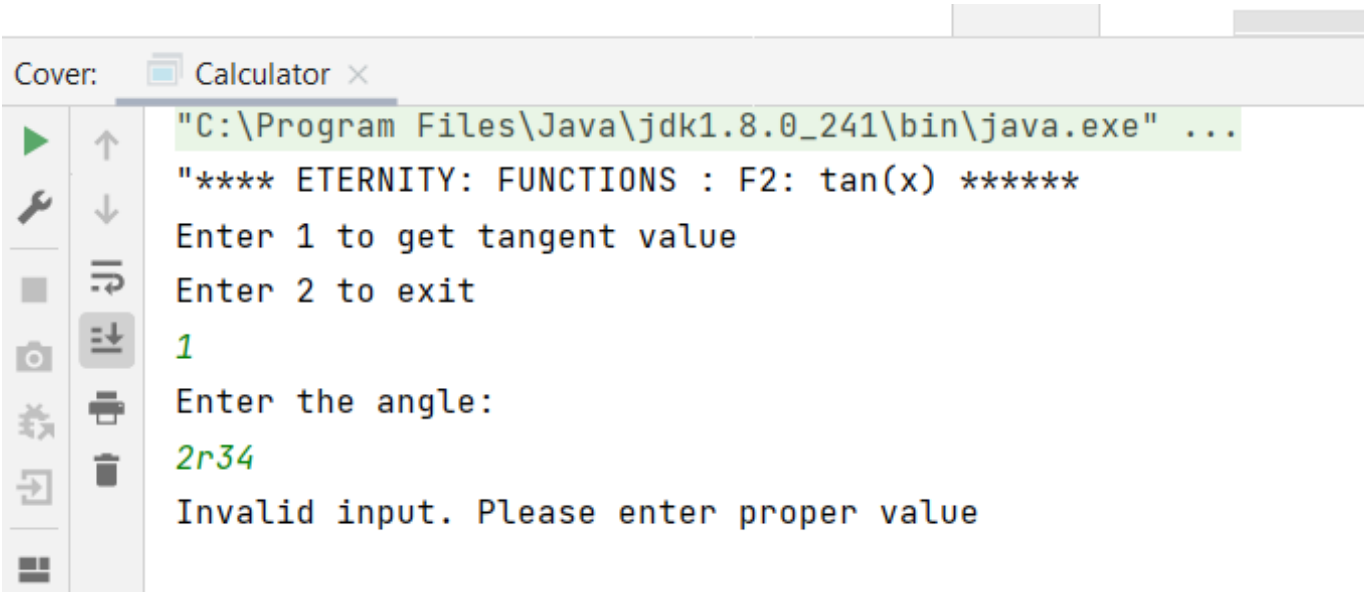

Figure 4: The snapshot of the debugger where the error and exceptions has been handled.

References

[1] MathBitsNotebook,
`https://mathbitsnotebook.com/Algebra1/FunctionGraphs/FNGTypeExponential.html`

[2] TutorialsPoint,
`https://www.tutorialspoint.com/java/lang/math_pow.htm`

[3] R. T. Barker and D. W. Biers, "Software usability testing: Do user self-consciousness and the laboratory environment make any difference?" in Part 2 (of 2), October 24, 1994 - October 28, 1994, .

[4] S. L. Sparagen and A. Riback, "Flexible software interface design," Ergonomics Des., vol. 7, (4), pp. 4-8, 1999.

[5] M. Hon, G. Russell and M. Welch, "Open source software considerations for law enforcement," IT Professional, vol. 12, (6), pp. 18-23, 2010. Available: http://dx.doi.org/10.1109/MITP.2010.121. DOI: 10.1109/MITP.2010.121.

[6] J. Ragot, D. Maquin and A. Kondo, "Control design for loop transfer recovery," in Part 1 (of 3), October 2, 1994 - October 5, 1994, .

[7] B. Haberman and H. Averbuch, "The case of base cases: Why are they so difficult to recognize? student difficulties with recursion," in ITiCSE 2002 - Proceedings of the 7th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, June 24, 2002 - June 28, 2002, Available: http://dx.doi.org/10.1145/544414.544441. DOI: 10.1145/544414.544441.