Answer to Problem 4

Arnab Roy

August 9, 2021

1 Information about the debugger used

For the purpose of debugging the method written for calculating the tangent of an angle, the Intellij IDEA debugger was used. This debugger comes pre-bundled with the Intellij IDEA IDE. This makes coding and debugging streamlined.

1.1 Advantages

Using a pre-bundled IDE comes with many advantages, one of which is ease of debugging when writing code. Some of the capabilities that this debugger provides is:

- Shows the variables and their updated values for each line of code. Values are shown in the code inline for each variable and also all variables and values are shown in the variables window.
- Allows jumping to variable, variable type and method declaration.
- Allows changing variable values during debugging, change program flow and execute conditional code branches.
- Allows evaluation of complex code expressions on the fly.
- Enables pausing the debugging procedure and check the call stack and the running threads.
- Enables setting breakpoints on lines. The debugger stops code execution at the breakpoint and the user can manually execute the remaining code statements, one by one.
- Allows dragging a breakpoint and dropping it on another line to move a breakpoint from one line to another. Also allows setting multiple breakpoints and disabling them.

- Enables setting conditional breakpoints, where the application is stopped at that line if a certain condition is met.
- Allows stepping over a method, stepping into a method or stepping out of a method during debugging. JDK methods are stepped over by default and the focus is towards user written methods and statements.
- Enables "run to cursor" feature, where the debugger does not stop at lines until the one at which the mouse cursor is at.

1.2 Disadvantages

The debugger is very resource intensive. It can easily slow down a machine with 8GB RAM and Intel Core i7 processor for very large programs.

2 Effort to make the program better

When writing the program, focus was given to make it correct, efficient, maintainable, robust, and usable. An explanation of the steps taken to make them such is given below.

2.1 Correctness

According to the requirements, the program needs to be correct up-to 7 decimal digits of a Casio scientific calculator. The result of the tangent function in the program depends on the results of the sine and the cosine functions. These functions were implemented using the Taylor series and severals steps were taken to increase their accuracy.

- At first, only 3 terms from the Taylor series were considered, which later was increased to 9. The inclusion of more terms increased the accuracy but was not enough.
- It was observed that for smaller angle values, the calculations were very accurate. But for higher values, the results started to deviate. So, all angles were first converted within 360 degrees.
- To further increase the accuracy, they were converted to 90 degrees and the quadrant was calculated to determine the sign (+ or -) of the result. This also eliminated a problem during calculation of cosine of 180 which outputted 1.0000008 instead of 1.
- For many calculations, Java shows -0. These results were checked so that the output for 0 does not have any sign.

2.2 Efficiency

To calculate the tangent of an angle, the Taylor series algorithm was implemented which is a $O(n^2)$ algorithm because of the calculation of the factorials for each term. To make it efficient, the values of the factorials were calculated beforehand at once and stored in an array. To calculate the factorial of n, n was multiplied with the value of (n-1)th index in the array and the result was stored at nth index of the array. This algorithm takes place in O(n) time and the values can be accessed in O(1) time. Thus the run time complexity of the algorithm was reduced to O(n) and the space complexity increased to O(n). The rounding function runs in O(1) time complexity.

2.3 Maintainability

To make the program maintainable, the lines of code were properly documented by comments. Most code, which would be repeatable and reusable, were separated into separate small functions. The methods return appropriate error messages. The code was written in a way so that it is easier to read and meaningful variable names were used.

2.4 Robustness

For the tangent function, one pitfall is when the value of cosine is 90 degrees. For this case, an `ArithmeticException` was thrown with an error message that could be clearly understood by the user.

2.5 Usability

The functions written for the tangent, cosine and sine functions do not have input prompts themselves. They require driver programs to take the input from the user and pass it as arguments. The output and error messages from the tangent function notifies the user about the result as a clearly understandable string.

3 Description of Checkstyle

For this project, an extension of Checkstyle for IntelliJ IDEA had been used. The source code was analyzed with the Google style. It generated 121 warnings, most of which were related to code indentation. It suggested a tab width of 2 spaces instead of 4. This type of indentation makes the code clearer to read. Moreover, another type of warning was to

comment before classes and methods using JavaDoc. It also warned about comments that are more than 100 words long on a single line.
The advantages of using CheckStyle is that it enforces strict coding conventions which is useful across teams. The code analysis was also very fast.
One disadvantage of the extension was that, it did not show the errors inline, like most static source code analysers do.