Project #3
CS 2210 – Fall 2022
Christopher LaFave

I. Requirements: Create a method that take a string in RPN format and returns the double calculated from that string. Must handle exceptions and must have thorough testing with JUnit.

II. Design: The main method has three "states" (not sure if states are the right word, btu DLD is coming back to haunt me) and a stack to store information. The three states are: isDouble, isOperator, and else. isDouble and else are both one liners, with isDouble pushing to the stack and else throwing an exception. isOperator figures out what the operator is, uses it on the first two numbers in the stack, and pushes the result back to the stack. After it gets to the end of the input string, it checks for one more error, and then returns the answer.

III. Security Analysis: The only input this takes is a string that must be either a double or an expected operator, so I don't think there are any security risks to this method. I suppose there is always the possibility of overflow/underflow, but I think the double type handles those well.

IV. Implementation: I created a couple of private methods to help me implement the main RPNcalc. isDouble, isOp, and doMath. isDouble checks whether or not the string input is a double without needing messy try-catch blocks in my if statement. isOp does the same thing, but is less important (I just didn't like 4 chained or statements in my code). doMath shortens up the code very nicely, because it can do all 4 operations and tell between them. doMath also throws the Divide By 0 exception for me.
I used isDouble and isOp to clearly organize those two states, and doMath was used in the isOp statement to apply the operators.
I also implemented the InvalidRPNString exception but that was easy.

V. Testing: I did three types of testing. The first as operator testing. I did multiple asserts for each operator to make sure it was working correctly. Then I created a test for every error that could be thrown by my program. Lastly, to check longer inputs, I checked the example input from the project description. All the tests passed, and I don't see need for any other tests.

VI. Summary/Conclusion: The program appears to function properly. It gives the correct answer for all valid inputs and throws the correct exceptions when tested.

VII. Lessons Learned: I learned how to use stacks and apply Junit tests. I got much more comfortable with coding in Java, code organization, and handling/creating exceptions.