Project #3
CS 2210 – Fall 2021
Sam DeCook

I. Requirements: Restate the problem specification, and any detailed requirements

Make a method capable doing simple arithmetic which is inputted in Reverse Polish Notation. Throw an InvalidRPNStringException if the input string contains wrong input, or it attempts to divide by zero. Conduct thorough testing using JUnit.

II. Design: How did you attack the problem? What choices did you make in your design, and why? Show class diagrams for more complex designs.

This problem was relatively simple. I used a while loop to iterate through the input string, adding numbers to the stack and doing calculations when an operator is reached.

III. Security Analysis: State the potential security vulnerabilities of your design. How could these vulnerabilities be exploited by an adversary? What would be the impact if the vulnerability is exploited?

There aren't any security vulnerabilities due to the nature of the code. It performs a simple operation.

IV. Implementation: Outline any interesting implementation details.

When I cleaned up my code, I had to move error checking for the string token (ensuring it was a + , - , * , / ; ensuring the stack had at least two numbers) before the switch statement. They were originally at the end of the switch statement and inside each case, respectively. The code didn't look as clean, but it significantly reduced redundant code.

V. Testing: Explain how you tested your program, enumerating the tests if possible. Explain why your test set was sufficient to believe that the software is working properly, i.e., what were the range of possibilities of errors that you were testing for.

I tested each operator type to ensure it was working, using small and then large numbers as well as integers, doubles, and negatives. Then I tested complex, multistep operations to ensure the while loop and associated logic functioned properly and finally tested error throws. This function is simple enough that this test set is sufficient to determine that my code is working.

VI. Summary/Conclusion: Present your results. Did it work properly? Are there any limitations? If it is an analysis-type project, this section may be significantly longer than for a simple implementation-type project.

My code compiled and ran properly, and produced the expected output (passed the tests).
Sam DeCook

VII. Lessons Learned: List any lessons learned. What might you have done differently if you were going to attack this again.

You can assert that an exception is thrown in JUnit.