**Jacob Pitkin’s Calculator**

While working on this calculator, I wanted to first identify what kinds of objects and abstraction I could get out of the calculator. The only real fit I found that seemed to work well was to create abstracted operation strategies (the exponentiation, multiplication, division, addition, and subtraction). This way we don’t really have to know how they’re calculating things, just that they take a left and a right value (left in exponentiation being the base value and the numerator in division). This gives us the ability to create more mathematical operations or alter already made ones without changing any of the business logic within the calculator itself.

When I first started, I was inclined to include parenthesis in the operation abstraction. However, I found that thinking of parenthesis as their own sub expressions helped a lot more since I could zero in on each “window” of the expression and evaluate those on their own, then bubble the expression back up replacing the expression in the parenthesis with its result.

To make sure I’m actually working with valid equations, I set up two checks right at the beginning of my evaluation. The first is a regular expression double checking that I’ve only got characters valid in a mathematical expression (parenthesis, periods, numbers, \*, /, ^, and -). The second check evaluated how many open and closing parenthesis exist and throws an exception if they’re not equal.

To actually perform each of the mathematical operations, I put those strategies in an array to loop over and apply their mathematical operations as I came across them. The benefit of doing it this way is I can set up their order of operation up front in the constructor of the Calculator class, and as I loop over the array in order, I’m performing those mathematical operations in order according to PEMDAS. I actually struggled a good bit with this since I was also looping over the expression in its own array so I could just grab the numbers to each side of the operator, but capturing negative values on either side required a lot more thought and array resizing to keep things in check.

Overall this project took me approximately 8 hours, which I’d split up into about 2 hours relearning Visual Studio and researching C# documentation, 4 hours planning and thinking about the problem, and then 2 hours actually coding the project.