**Time Complexity Analysis of ArithmeticCalculator.java**

Based on how the code operates, the time complexity of the algorithm is **O(n2)** as we have to first loop through the n length of each expression string, thus worst case is **O(n)** for the loop of the expression alone. We know that stack operations run in constant time **O(1)** so anytime an operator or an operand gets pushed or popped from their respective stack, it runs in **O(1)** time thus it does not affect the overall runtime of the algorithm. There is one exception to that and that is the stack *push()* method. *push()* is **O(1)** except when the stack is full. The stack must expand its size by constant *c*, thus the stack will have to loop through all of its contents to store into a copy stack and then copy those values back into the extended stack thus worst case time complexity of the push method alone is **O(n + c).** Since in the worst case, we have looped through the entire string expression and we push into a full stack, this means we loop through n elements twice, thus we have **O(n2)**.

Furthermore, we know that on average, every element inside the stack is used only once or twice. This means that each element will be run only a maximum of two times, thus guaranteeing that our code will not exceed **O(n2)** as the stack cannot expand twice with one *push()* method.

For the memory complexity of the algorithm, the stack will only occupy a max of n elements which represent the n length of the expression string, thus the memory complexity of the algorithm is **O(n).**