CalculatorTest.java			
Test #	Test Name	Expected Outcome	Actual Outcome
1	addNumberToList	6	6 ✓
2	substractionTest	10	10 ✓
3	multiplicationTest	8280	8280 ✓
4	additionTest	88	88 ✓
5	divisionTest	2.4	2.4 ✓
6	tooManyNumbersTest	8280	8280 ✓
7	invalidInput	690	690 ✓

- Test #1: Every number is separated with a spacing; operations and numbers are first added as a whole string and then the string is separated with white spacing.

- Test #2, #5: A whole string is parsed when the calculator instance is created. This string contains two numbers and are followed by the operand. The tests call the doOperation() method which pushes the numbers onto the stack and takes into account the operand and returns the calculated result.

- Test #3, #4: These tests are the same as tests #2 and #5 but the operation is extended with an additional operation. The end-result from the first operation is placed onto the stack and then used again to give a new final result.

- Test #6: Same as test #3 but with an extra integer added at the end of the two operations. The end-result is that the implementation returns the result up until that point and ignores the additional integer as it does not fulfill the requirements for a valid operation.

- Test #7: This test takes into account an operation and a character. The character is ignored as it is not an operand and the operation outcome before it is returned.

StackTest.java			
Test #	Test Name	Expected Outcome	Actual Outcome
1	testStackSize	1	1 ✓
2	testStackPush	“Test2”	“Test2” ✓
3	testStackPop	3	3 ✓
4	testEmptyStack	True	True ✓
5	testExpandSize	200	200 ✓
6	testGetElements	Object[] elements	Object[] elements ✓

Test #1: A new instance of MyStack is created in all tests and is parameterized to take a String. We then call the push() method in MyStack which is responsible for increasing the array size and to return the current element position. After we push two elements onto the stack, the counter value is set to two. Then, when we call the pop() method, the first element into the stack is removed and the counter is decremented to one.

Test #2: We push an element onto the stack. The pop() method returns the object that is to be popped based on which position the counter is at.

Test #3: Four elements are pushed onto the stack. Therefore, the counter is currently four. After calling the pop() method, the counter is decremented and is compared in the assertEquals() method in the test class.

Test #4: This test calls the emptyStack() method that, which has the purpose of checking if the stack is already empty. A stack is created, and the method emptyStack() is called on it, the result is compared with an assertEquals() method.

Test #5: This test first returns the stack instance array object and then calls the expandSize() method which returns a new array with a doubled capacity.

Test #6: Returns the Object array of elements.

Created By:

Angel Petrov - 266489@via.dk
Christian Schou Sørensen - 267142@via.dk
Ionel-Cristinel Putinica - 266123@via.dk