Project Name: Homework 4

Unit Test Case Specification

Test Case ID: UnitTestingHomework_TC_3

Test Designed by: Tanner Huynh

Test Priority (Low/Medium/High): Low

Test Designed date: November 6, 2019

Module Name: UnitTestingHomework

Test Title: Multiplier Method

Description: Test the method multiplier, which takes two ints and multiplies them together. Per the interface: int multiplier(int x, int y).

Pre-conditions: N/A

Dependencies: N/A

Case	Given Input	Expected Result	Notes/Environment
1	(0, 0)	0	Zero Power Rule
2	(1 , x)	x	Identity property of multiplication.
3	(-x, -y)	z	Negative numbers produce positive products.
4	(x, - y)	-z	Negative times positive equals negative.
5	(-11000, 328344)	-1758101632	Middle of left partition [-1758101632].

	(11000, 328344)	1758101632	Middle of right partition [1758101632].
6	(MIN_VALUE, 1)	-2147483648	Left negative partition.
7	(MIN_VALUE-1, 1)	-2147483647	One to the right of the left negative partition.
8	(MAX_VALUE, 1)	2147483647	Right positive partition.
9	(MAX_VALUE+1, 1)	2147483646	One to the left of the right positive partition.
10	See table	See table	100 numbers from each partition that does not cause stack overflow.

Post Conditions: N/A

Explanation:
The range of valid inputs for this function, `public int multiplier(int x, int y),` is [`MIN_VALUE…MAX_VALUE`] where the values x and y do not cause a product of stack overflow. It's a range, so we need to test on both sides of the range and inside the range. There are three partitions of [MIN_VALUE … 0] and [0 … MAX_VALUE]. Ideally we will also test for numbers in-between these partitions. Tests should run on the boundaries and one or more past the boundaries.

After that, the next useful tests to add use domain knowledge to define test cases. Here, the example recognizes there are four special cases: zero power rule, identity property, and negative/positive multiplication. The test suite should identify a sample of each of these. The example uses one example per type.

So the minimum total number of tests is nine. This isn't over-whelming for confidence, but it does provide direction for what a larger sampling should entail. After this, every input within a partition is equi-probable. At this point, it's basically guessing inputs and your only limits are the time and resources available to run tests. I'd add in a few hundred samples within each partition, likely creating a few "sub-partitions" as a heuristic – for example: small, medium, and large numbers.

There are some test cases that can't be run without causing a stack overflow. Perhaps, the method should be rewritten to expect bigger numbers.