

Download and install the Java mutation tool: muJava. Use muJava to test cal().

Use all the operators. Design tests to kill all non-equivalent mutants. Note that a test case is a method call to cal().

(a) How many mutants are there?

The screenshot shows the muJava TestCase Runner interface. The 'Traditional Mutants Viewer' tab is active. The 'Class' is set to 'Cal', the 'Method' is 'int\_cal(int,int,int,int,int)', and the 'TestCase' is 'CalTest'. The 'Time-Out' is set to '3 seconds'. The 'Execute all mutants' radio button is selected. The 'RUN' button is highlighted in yellow.

On the left, there are two tables of operators and their counts:

Op	#	Op	#
AORB	36	IHI	0
AORS	1	IHD	0
AOIU	12	IOD	0
AOIS	74	IOP	0
AODU	0	IOR	0
AODS	0	ISI	0
ROR	34	ISD	0
COR	4	IPC	0
COD	0	PNC	0
COI	7	PMD	0
SOR	0	PPD	0
LOR	0	PCI	0
LOI	19	PCC	0
LOD	0	PCD	0
ASRS	0	PRV	0
SDL	10	OMR	0
VDL	13	OMD	0
CDL	5	OAN	0
ODL	22	JTI	0
		JTD	0
		JSI	0
		JSD	0
		JID	0
		JDC	0
		EOA	0
		EOC	0
		EAM	0
		EMM	0

Total : 237

On the right, the 'Traditional Mutants Result' table shows:

Traditional Mutants Result	
Live Mutants #	30
Killed Mutants #	207
Total Mutants #	237
Mutant Score	87.0%

Below this, there are two lists: 'Live' and 'Killed'. The 'Live' list contains 30 mutants, including ROR\_4, AOIS\_55, AOIS\_56, AOIS\_72, AOIS\_44, AOIS\_43, AOIU\_6, ROR\_22, AOIS\_28, AOIS\_52, AOIS\_73, AOIS\_12, AOIS\_31, AOIS\_15, ROR\_18, AOIS\_36, ROR\_8, AOIS\_32, and AOIS\_74. The 'Killed' list contains 207 mutants, including ROR\_2, AOIU\_1, AOIS\_61, ROR\_11, ROR\_15, VDL\_5, AOIS\_8, LOI\_11, LOI\_2, AOIS\_65, AOIS\_37, ROR\_14, ODL\_6, AORB\_1, AOIU\_10, AOIU\_12, LOI\_7, ODL\_19, and VDL\_13.

On the far right, the 'Class Mutants Result' table shows:

Class Mutants Result	
Live Mutants #	0
Killed Mutants #	0
Total Mutants #	0
Mutant Score	- %

Below this, there are two empty lists: 'Live' and 'Killed'.

**237** mutants in total

(b) How many test cases do you need to kill the non-equivalent mutants?

I design **8** test cases to kill the non-equivalent mutants:

1. `Assert.assertEquals(151, Cal.cal(5, 23, 10, 21, 2022));`
2. `Assert.assertEquals(121, Cal.cal(1, 1, 5, 1, 2020));`
3. `Assert.assertEquals(181, Cal.cal(1, 1, 7, 1, 2100));`
4. `Assert.assertEquals(182, Cal.cal(1, 1, 7, 1, 2000));`
5. `Assert.assertEquals(14, Cal.cal(1, 7, 1, 21, 2100));`
6. `Assert.assertEquals(59, Cal.cal(1, 1, 3, 1, 2022));`
7. `Assert.assertEquals(-6, Cal.cal(8, 7, 8, 1, 2022));`
8. `Assert.assertEquals(0, Cal.cal(7, 7, 7, 7, 2022));`

(c) What mutation score were you able to achieve before analyzing for equivalent mutants?

I achieved **85%** mutant killed before I started to analysis the remaining mutant one by one

And achieved **87%** before submit my homework

(d) How many equivalent mutants are there?

In my test, I found **30** equivalent mutants. Since I didn't check all the 30 remaining mutants, I can't sure there don't exist non-equivalent mutants anymore.

Most of equivalent mutants are in following type:

1. make postfix add (a++) on variables that won't be used in the remaining program (e.g. AOIS\_51)
2. change logic condition mark that won't influence the outcome (e.g. COR\_4)
3. switch the sign of a variable that is used for modular 100 (e.g. AOIU\_5)
4. cases that are already covered by program (e.g. ROR\_4)

Since these faults may produce an error state, they won't produce a failure outcome.

Github: [311551169-ST-2023/Hw3 Report .pdf at main · Laxiflora/311551169-ST-2023 · GitHub](#)