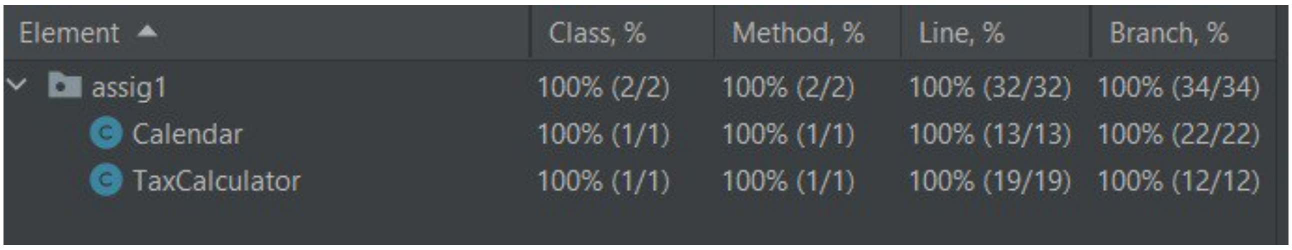
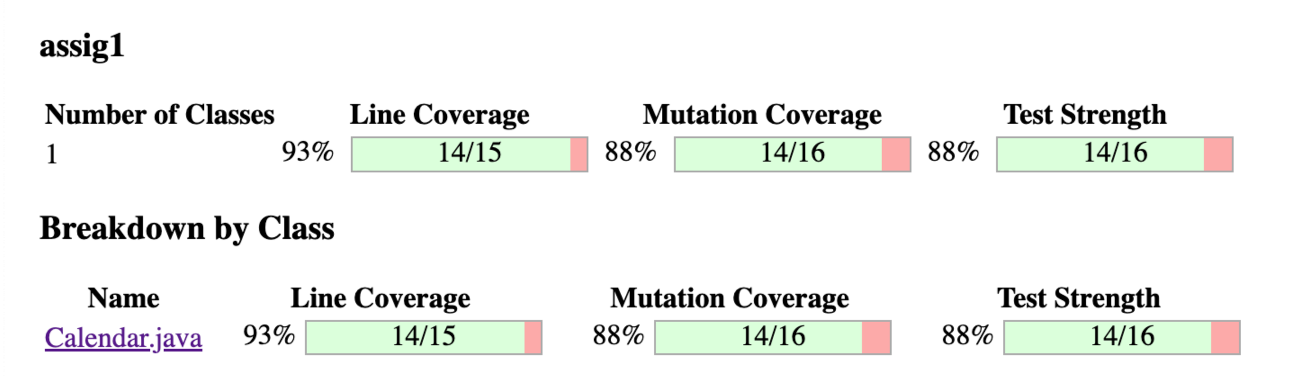
Assignment 1

Task1 - Calendar



Element	Class, %	Method, %	Line, %	Branch, %
assig1	100% (2/2)	100% (2/2)	100% (32/32)	100% (34/34)
Calendar	100% (1/1)	100% (1/1)	100% (13/13)	100% (22/22)
TaxCalculator	100% (1/1)	100% (1/1)	100% (19/19)	100% (12/12)

I achieved 100 % class, method, line, and branch coverage. I could have performed as well in the mutation testing with less branch coverage, but I chose to score high on both. It is not necessary to list every month in tests for the mutation score.



assig1

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	93%	14/15	88%	14/16	88%	14/16

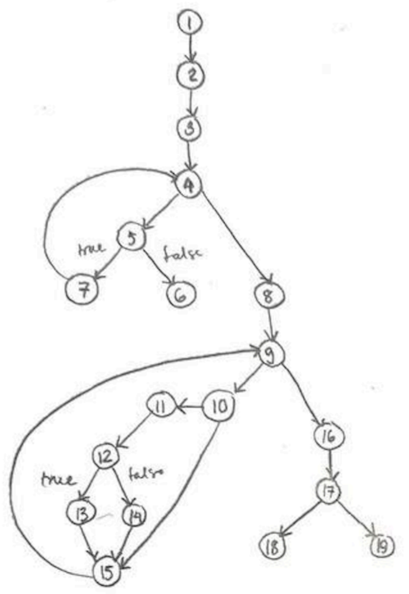
Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
Calendar.java	93%	14/15	88%	14/16	88%	14/16

My test strength is 88 %. All mutants except the two on the print lines were killed.

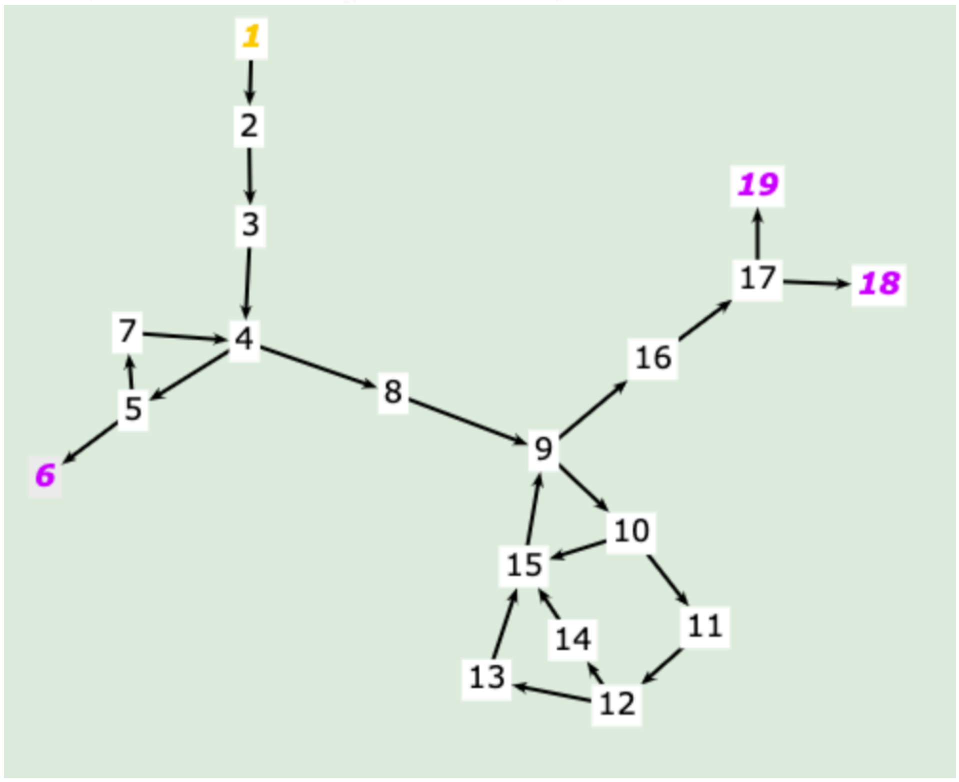
Task 2 – Task Calculator

Control Flow Graph

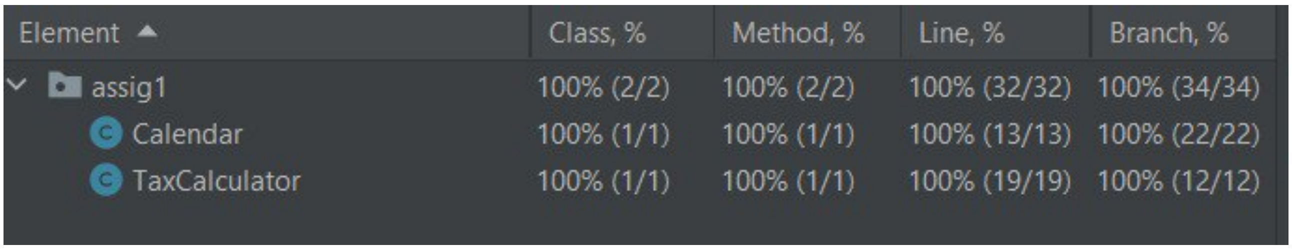


1 initialize incomeList & childAgeList
2 print-statement
3 initialize rest of the variables
4 dummy node (for)
5 if
6 return (income <0)
7 add income to incomeAmount
8 initialize i =0
9 dummy node
10 first if
11 minorchildren statement
12 second if
13 if statement true
14 if statement false
15 increment i++
16 calc tax Amount
17 new if
18 if - true
19 if false

starting nodes : 1
ending nodes : 6, 18, 19



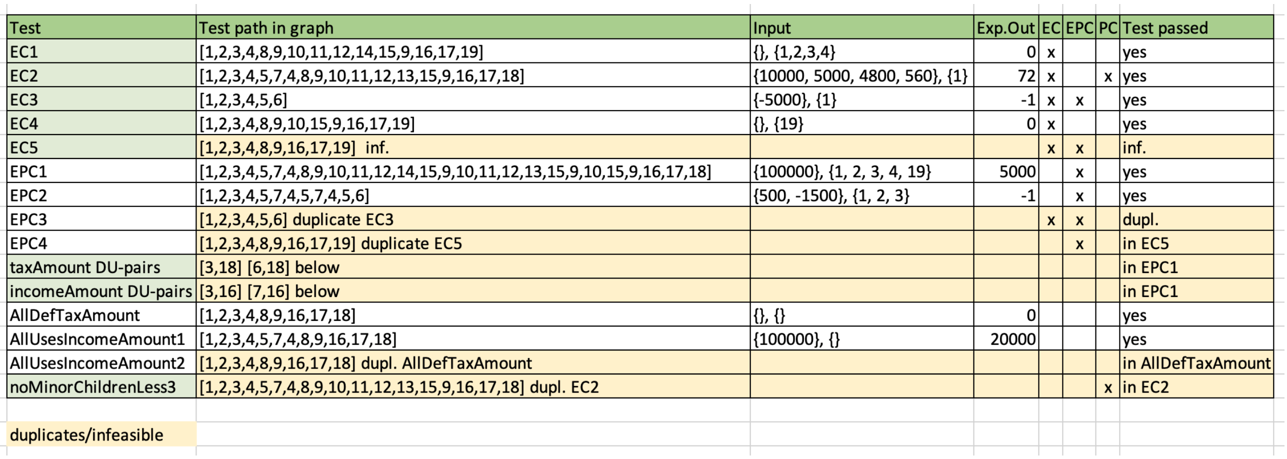
Code Coverage



Element	Class, %	Method, %	Line, %	Branch, %
assig1	100% (2/2)	100% (2/2)	100% (32/32)	100% (34/34)
Calendar	100% (1/1)	100% (1/1)	100% (13/13)	100% (22/22)
TaxCalculator	100% (1/1)	100% (1/1)	100% (19/19)	100% (12/12)

I have 100 % class, method, and line coverage in Tax Calculator as well.

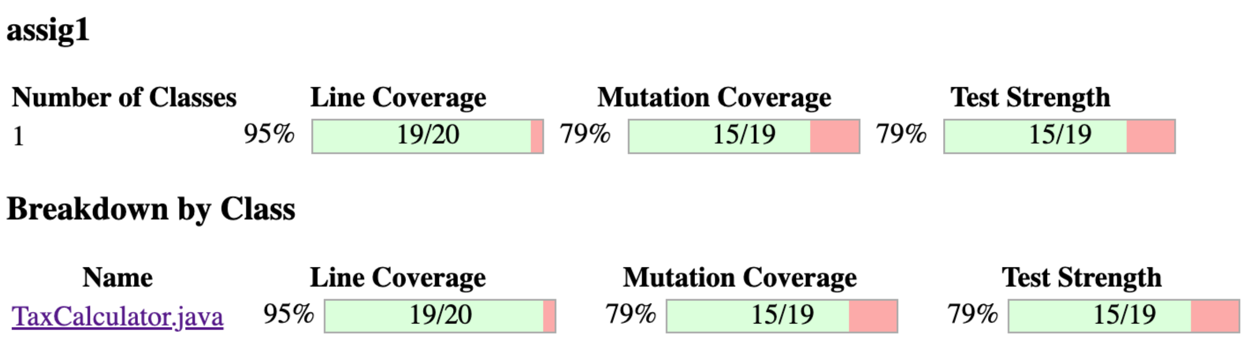
Table for test requirements



Test	Test path in graph	Input	Exp.Out	EC	EPC	PC	Test passed
EC1	[1,2,3,4,8,9,10,11,12,14,15,9,16,17,19]	{}, {1,2,3,4}	0	x			yes
EC2	[1,2,3,4,5,7,4,8,9,10,11,12,13,15,9,16,17,18]	{10000, 5000, 4800, 560}, {1}	72	x		x	yes
EC3	[1,2,3,4,5,6]	{-5000}, {1}	-1	x	x		yes
EC4	[1,2,3,4,8,9,10,15,9,16,17,19]	{}, {19}	0	x			yes
EC5	[1,2,3,4,8,9,16,17,19] inf.			x	x		inf.
EPC1	[1,2,3,4,5,7,4,8,9,10,11,12,14,15,9,10,11,12,13,15,9,10,15,9,16,17,18]	{100000}, {1, 2, 3, 4, 19}	5000		x		yes
EPC2	[1,2,3,4,5,7,4,5,7,4,5,6]	{500, -1500}, {1, 2, 3}	-1		x		yes
EPC3	[1,2,3,4,5,6] duplicate EC3			x	x		dupl.
EPC4	[1,2,3,4,8,9,16,17,19] duplicate EC5				x		in EC5
taxAmount DU-pairs	[3,18] [6,18] below						in EPC1
incomeAmount DU-pairs	[3,16] [7,16] below						in EPC1
AllDefTaxAmount	[1,2,3,4,8,9,16,17,18]	{}, {}	0				yes
AllUsesIncomeAmount1	[1,2,3,4,5,7,4,8,9,16,17,18]	{100000}, {}	20000				yes
AllUsesIncomeAmount2	[1,2,3,4,8,9,16,17,18] dupl. AllDefTaxAmount						in AllDefTaxAmount
noMinorChildrenLess3	[1,2,3,4,5,7,4,8,9,10,11,12,13,15,9,16,17,18] dupl. EC2					x	in EC2
duplicates/infeasible							

In this table, I have all the tests generated from the assignment instructions. By themselves, I get a mutation coverage and test strength of 79 %.

EC5 is marked as infeasible since you cannot get a negative tax outcome if you do not have children, therefore we cannot end up in node 19 on this path.



assig1

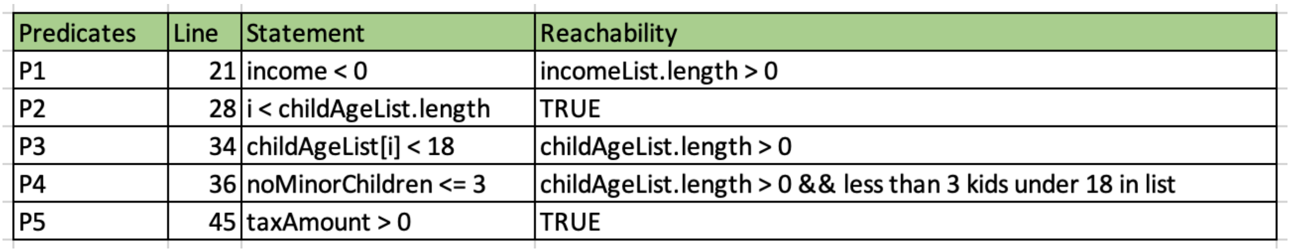
Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	95%	19/20	79%	15/19	79%	15/19

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
TaxCalculator.java	95%	19/20	79%	15/19	79%	15/19

The tests did not cover some of the conditional boundary changes, so I added some more tests to target those. They are described at the end of this report.

Table for predicates and their reachability conditions



Predicates	Line	Statement	Reachability
P1	21	income < 0	incomeList.length > 0
P2	28	i < childAgeList.length	TRUE
P3	34	childAgeList[i] < 18	childAgeList.length > 0
P4	36	noMinorChildren <= 3	childAgeList.length > 0 && less than 3 kids under 18 in list
P5	45	taxAmount > 0	TRUE

Predicate 1 is within the first for-loop, which is only visited if the incomeList has elements in it.

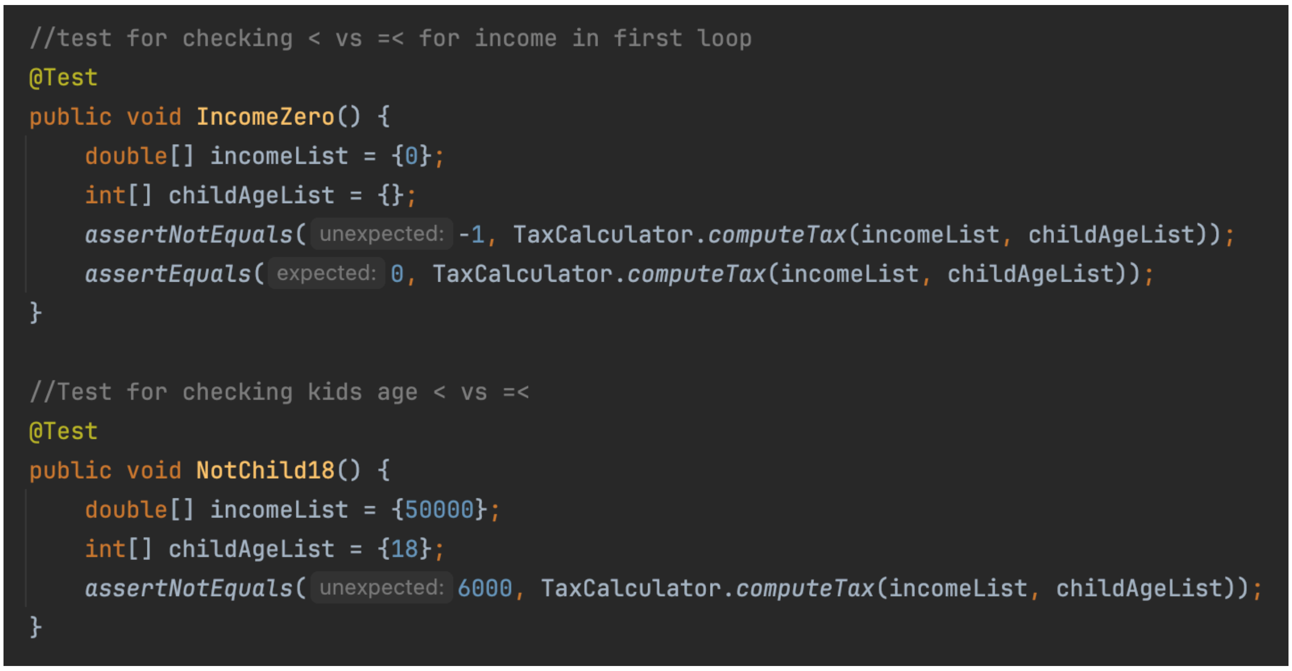
Predicate 2 is specified in the for-loop regarding children, and it is visited even if the list for children is empty.

Predicate 3 is within the for-loop regarding children, and hence the list cannot be empty if we wish to reach this predicate.

Predicate 4 is within the if-statement of the for-loop, and it is required that the list is not empty and that the child age list elements satisfy the condition.

Predicate 5 is reached regardless of the paths taken, even if both income and children's age lists are empty.

Added tests for conditional boundaries

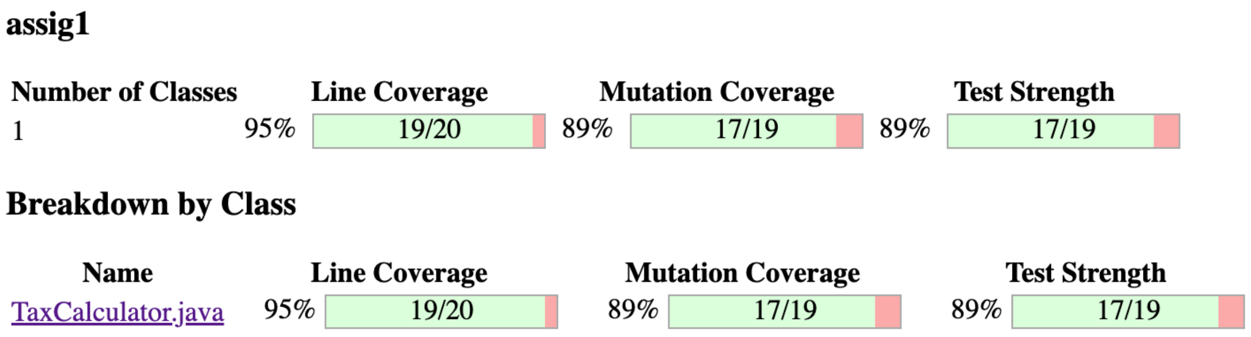


```java
//test for checking < vs =< for income in first loop
@Test
public void IncomeZero() {
    double[] incomeList = {0};
    int[] childAgeList = {};
    assertNotEquals( unexpected: -1, TaxCalculator.computeTax(incomeList, childAgeList));
    assertEquals( expected: 0, TaxCalculator.computeTax(incomeList, childAgeList));
}

//Test for checking kids age < vs =<
@Test
public void NotChild18() {
    double[] incomeList = {50000};
    int[] childAgeList = {18};
    assertNotEquals( unexpected: 6000, TaxCalculator.computeTax(incomeList, childAgeList));
}
```

Since my mutation score was not as high as I'd hoped, I added two tests to target conditional boundary errors. The first test targets line 21, as we do not wish to return -1 if the < is accidentally changed to <=.

The second test targets cases where we might make a human error and change the childAgeList[i] < 18, and change the < to <=, counting 18-year-olds as minors.



assig1

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	95%	19/20	89%	17/19	89%	17/19

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
TaxCalculator.java	95%	19/20	89%	17/19	89%	17/19

This brings my mutation score and test strength up to 89 % from the previous 79 %, leaving the print statement and the changed conditional boundary “if (taxAmount > 0)” surviving.

Testing this last conditional boundary would not be as straightforward as the ones described and tested above, since the expected output of the if-branch would be 0, which also the else-branch returns.