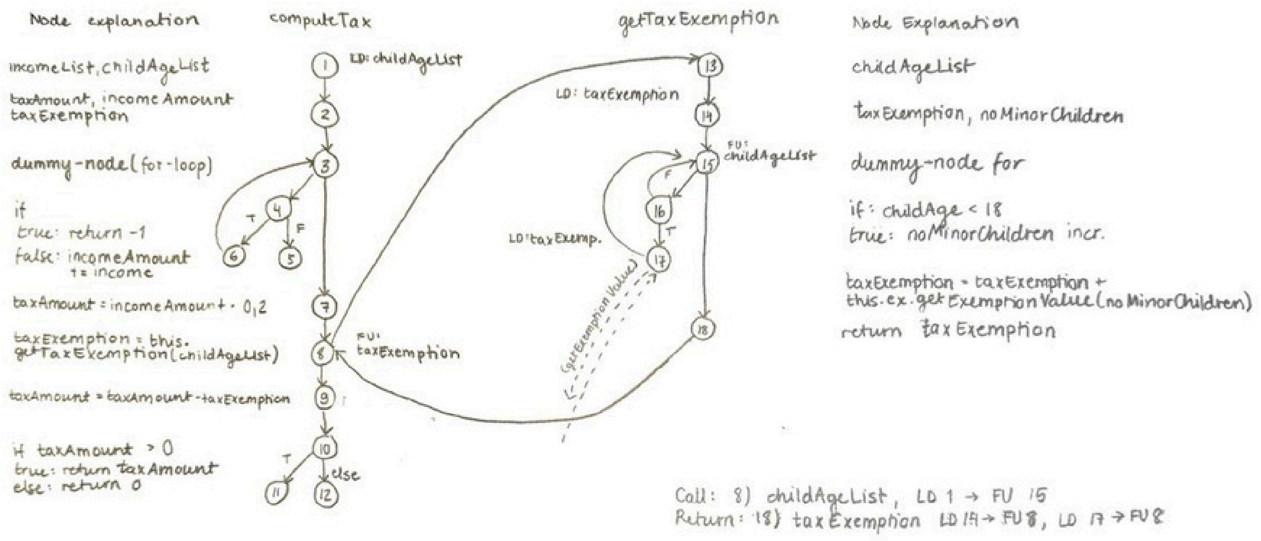
Assignment 2

In this assignment, I explore integration testing for the methods computeTax and getTaxExemption within the class TaxCalculator2 and explore Mockito with class Exemption.

Task 1: Design of Integration Tests

Here are the CFG graphs of computeTax and getTaxExemption. There is one method call and one return between these two methods. In node 8, there is a method call to getTaxExemption with the childeAgeList, and in node 18, taxExemption is returned to computeTax. These variables are passed between these methods.


Node explanation
computeTax
getTaxExemption
Node Explanation
incomeList, childAgeList
LD: childAgeList
taxAmount, incomeAmount taxExemption
dummy-node (for-loop)
if
true: return -1
false: incomeAmount += income
taxAmount = incomeAmount * 0,2
taxExemption = this.getTaxExemption(childAgeList)
FU: taxExemption
taxAmount = taxAmount - taxExemption
if taxAmount > 0
true: return taxAmount
else: return 0
T
F
else
LD: taxExemption
LD: taxExemp.
FU: childAgeList
getExemptionValue
childAgeList
taxExemption, noMinorChildren
dummy-node for
if: childAge < 18
true: noMinorChildren incr.
taxExemption = taxExemption + this.ex.getExemptionValue(noMinorChildren)
return taxExemption
Call: 8) childAgeList, LD 1 → FU 15
Return: 18) taxExemption LD 14 → FU 8, LD 17 → FU 8

childAgeList

LD: only defined in node 1, when it is given as an input to ComputeTax

FU: in node 15, when it is used in the for loop to see if there are minors in the family

taxExemption

LD1: in node 14, if the childAgeList is empty or there is an adult child in the family

LD2: in node 17, if there are children under the age of 18 in the family

FU: in both cases, taxExemption is updated directly in node 8, into the variable taxExemption that was defined within computeTax

This does not leave much room for many DU pairs. As in the case of All-coupling def's, where a path is chosen from every last-def to a first-use, would result in the same pairs as All-coupling use, where a path is chosen from every last-def to every first-use.

	Path 1	Path 2	variable
Nodes 1-15	1,2,3,7,8,13,14,15	1,2,3,4,6,7,8,13,14,15	childAgeList (call)
Nodes 14-8	14,15,18,8	14,15,16,15,18,8	taxExemption (return)
Nodes 17-8	17,15,18,8	17,15,16,15,18,8	taxExemption (return)

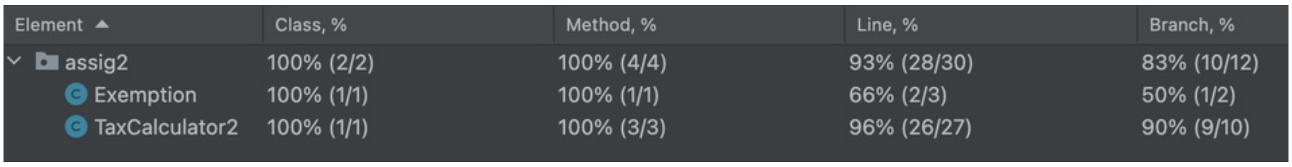
These are the possible paths from last-def to first-use for each variable that is passed between these methods. Since there is only one first-use for each for each last-def, the test requirements are the same for All-coupling def and All-coupling use. Either Path 1 or Path 2 is chosen for each pair. If I were to choose all six paths, I would get All-coupling DU paths relatively easily for this small implementation.

1,2,3,7,8,13,14,15,16,15,18,8	1-15 path 1	14-8 path 2
1,2,3,4,6,7,8,13,14,15,16,17,15,16,15,18,8	1-15 path 2	17-8 path 2

Here is one possible way to pair paths for the final test paths. Here, both paths are chosen for the pair 1-15, and then one of each for 14-8 and 17-8. For taxExemption, I chose the longer paths in the Path 2 column.

Test	Path in Graph	Input	Exp. Output	Test Passed
TestPath1	1,2,3,7,8,13,14,15,16,15,18,8,9,10,12	{0}{18}	0	yes
TestPath2	1,2,3,4,6,7,8,13,14,15,16,17,15,16,15,18,8,9,10,11	{40000}[0, 18}	4000	yes

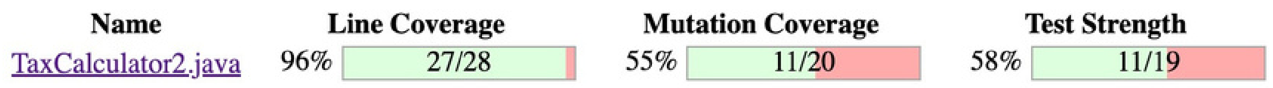
This results in a minimum number of two test paths, as seen above. Input values for children are chosen from the edges of the allowed values.



Element ▲	Class, %	Method, %	Line, %	Branch, %
assig2	100% (2/2)	100% (4/4)	93% (28/30)	83% (10/12)
Exemption	100% (1/1)	100% (1/1)	66% (2/3)	50% (1/2)
TaxCalculator2	100% (1/1)	100% (3/3)	96% (26/27)	90% (9/10)

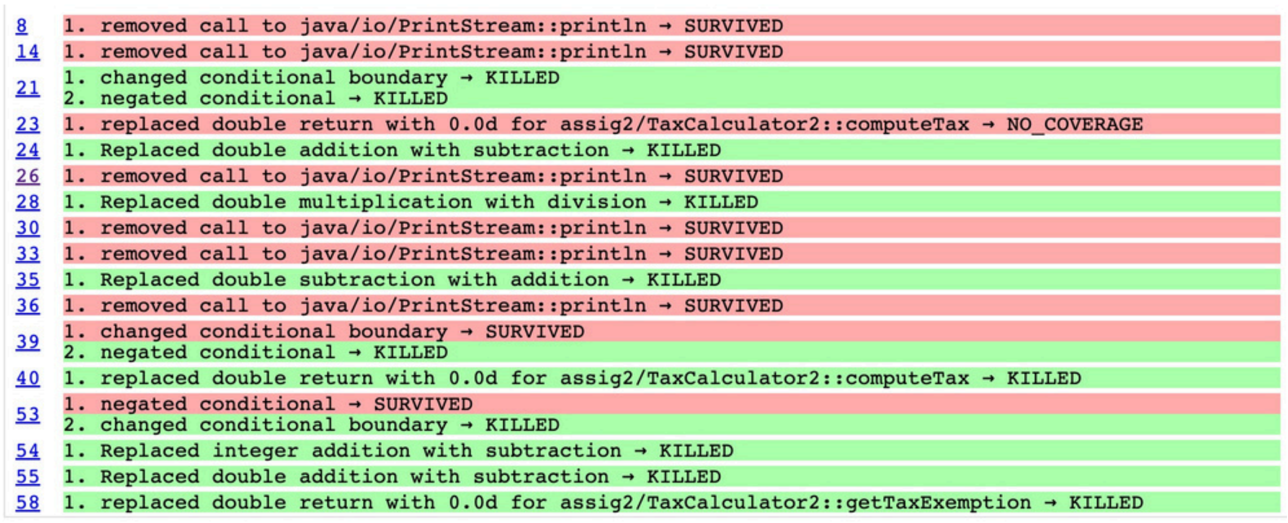
These two tests alone give a decent coverage of the TaxCalculator2 class. This covers the whole class except the branch for negative income, which is not necessary for testing integration between these two functions.

Task 2: Mutation Analysis



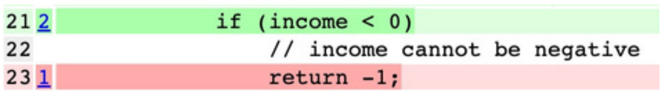
Name	Line Coverage		Mutation Coverage		Test Strength	
TaxCalculator2.java	96%	27/28	55%	11/20	58%	11/19

The overall mutation score for just two tests is not that good, but most of the mutations are not considered by the integration part, and 6 of them are about the print statements. This is the list of all mutations:



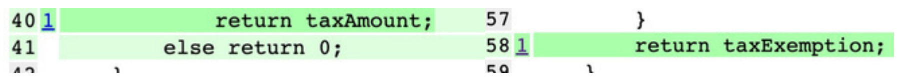
```
8   1. removed call to java/io/PrintStream::println → SURVIVED
14  1. removed call to java/io/PrintStream::println → SURVIVED
21  1. changed conditional boundary → KILLED
    2. negated conditional → KILLED
23  1. replaced double return with 0.0d for assig2/TaxCalculator2::computeTax → NO_COVERAGE
24  1. Replaced double addition with subtraction → KILLED
26  1. removed call to java/io/PrintStream::println → SURVIVED
28  1. Replaced double multiplication with division → KILLED
30  1. removed call to java/io/PrintStream::println → SURVIVED
33  1. removed call to java/io/PrintStream::println → SURVIVED
35  1. Replaced double subtraction with addition → KILLED
36  1. removed call to java/io/PrintStream::println → SURVIVED
39  1. changed conditional boundary → SURVIVED
    2. negated conditional → KILLED
40  1. replaced double return with 0.0d for assig2/TaxCalculator2::computeTax → KILLED
53  1. negated conditional → SURVIVED
    2. changed conditional boundary → KILLED
54  1. Replaced integer addition with subtraction → KILLED
55  1. Replaced double addition with subtraction → KILLED
58  1. replaced double return with 0.0d for assig2/TaxCalculator2::getTaxExemption → KILLED
```

There are not that many integration mutations generated. They are on lines 23, 40, and 58. All three mutants are of type “replaced double return with 0.0d”. I have excluded the ones for the call to print stream.



```
21 2            if (income < 0)
22                  // income cannot be negative
23 1                return -1;
```

Line 23 is never visited, if we wish to reach the method call in node 8. On lines 41 and 58, the mutants are killed. Thus, the mutation score for integration mutations is 66,67% (2/3).



```
40 1            return taxAmount;        57          }
41          else return 0;               58 1        return taxExemption;
```

Faults found

In Assignment 2, I noticed that negative ages are allowed for children. It is probably the same case for the implementation in Assignment 1.