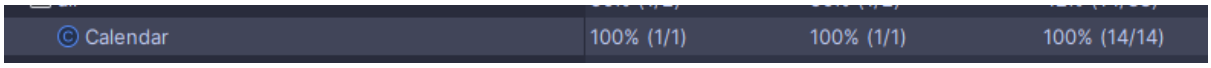
Report *Kevin Koljonen*

Task 1

Test Coverage:



PITest:

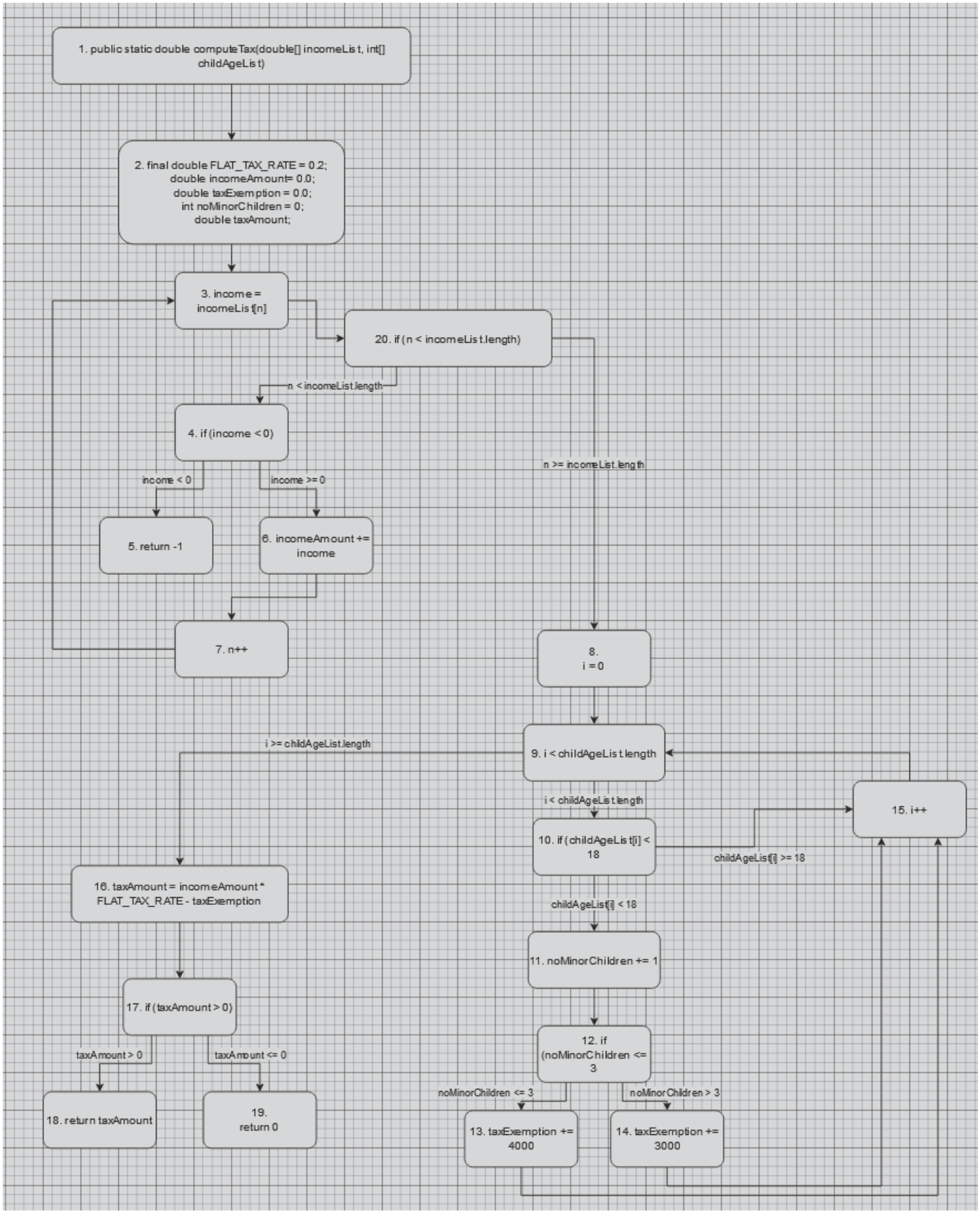
Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	93%	14/15	69%	11/16	69%	11/16

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
Calendar.java	93%	14/15	69%	11/16	69%	11/16

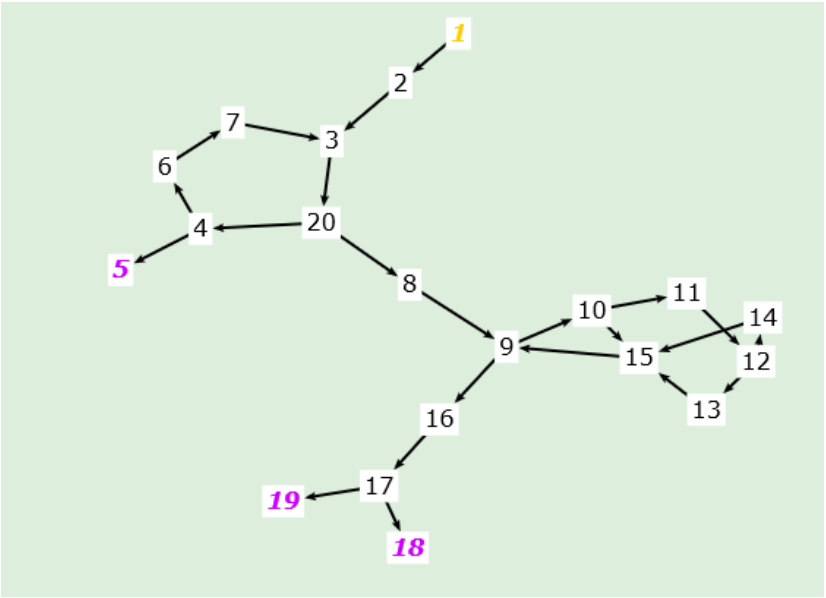
Task 2

1.


1. public static double computeTax(double[] incomeList, int[] childAgeList)
2. final double FLAT_TAX_RATE = 0.2;
double incomeAmount= 0.0;
double taxExemption = 0.0;
int noMinorChildren = 0;
double taxAmount;
3. income = incomeList[n]
20. if (n < incomeList.length)
n < incomeList.length
4. if (income < 0)
n >= incomeList.length
income < 0
income >= 0
5. return -1
6. incomeAmount += income
7. n++
8.
i = 0
i >= childAgeList.length
9. i < childAgeList.length
i < childAgeList.length
15. i++
10. if (childAgeList[i] < 18
childAgeList[i] >= 18
16. taxAmount = incomeAmount * FLAT_TAX_RATE - taxExemption
childAgeList[i] < 18
11. noMinorChildren += 1
17. if (taxAmount > 0)
12. if (noMinorChildren <= 3
taxAmount > 0
taxAmount <= 0
noMinorChildren <= 3
noMinorChildren > 3
18. return taxAmount
19.
return 0
13. taxExemption += 4000
14. taxExemption += 3000

2.

20 requirements are needed for Nodes
[1]
[2]
[3]
[20]
[4]
[5]
[6]
[7]
[8]
[9]
[10]
[11]
[12]
[13]
[14]
[15]
[16]
[17]
[18]
[19]



27 requirements are needed for Edge-Pairs
1. [1,2,3]
2. [2,3,20]
3. [3,20,4]
4. [3,20,8]
5. [4,6,7]
6. [6,7,3]
7. [7,3,20]
8. [8,9,10]
9. [8,9,16]
10. [9,10,11]
11. [9,10,15]
12. [10,11,12]
13. [11,12,13]
14. [11,12,14]
15. [12,13,15]
16. [12,14,15]
17. [13,15,9]
18. [14,15,9]
19. [15,9,10]
20. [15,9,16]
21. [9,16,17]
22. [16,17,18]
23. [16,17,19]
24. [20,4,5]
25. [20,4,6]
26. [20,8,9]
27. [10,15,9]

3.

incomeAmount	[2,6] [2,16] [6,6] [6,16]
taxAmount	[16,17] [16,18] [2,17] [2,18]

4.

taxAmount	[1,2,3,20,8,9,16,17,18]

5.

incomeAmount	[1,2,3,20,4,6,7,20,4,5] [1,2,3,20,8,9,16,17,18] [1,2,3,20,4,6,7,20,4,6,7,20,4,5] [1,2,3,20,4,6,7,20,8,9,16,17,18]

6.

- P1: n < incomeList.length
  - Reachability: True
- P2: income < 0
  - Reachability: P1 = true
- P3: i < childAgeList.length
  - n < incomeList.length && income >= 0 || n >= incomeList.length
  - Reachability: (P1 && !P2) || !P1
  - Reachability simplified: !P2
- P4: childAgeList[i] < 18
  - (n < incomeList.length && income >= 0 || n >= incomeList.length) && i < childAgeList.length
  - Reachability: ((P1 && !P2) || !P1) && P3
  - Reachability simplified: !P2 && P3
- P5: noMinorChildren <= 3
  - (n < incomeList.length && income >= 0 || n >= incomeList.length) && i < childAgeList.length && childAgeList[i] < 18
  - Reachability: ((P1 && !P2) || !P1) && P3 && P4
  - Reachability simplified: !P2 && P3 && P4
- P6: taxAmount > 0
  - (n < incomeList.length && income >= 0 || n >= incomeList.length) && (i < childAgeList.length && childAgeList[i] < 18 && (noMinorChildren <= 3 ||

noMinorChildren > 3)) || (n < incomeList.length && income >= 0 || n >= incomeList.length) && i >= childAgeList.length

- Reachability: ((P1 && !P2) || !P1) && (P3 && P4) || (((P1 && !P2)|| !P1) && !P3)
- Reachability simplified: (!P2 && (P3 && P4)) || (!P2 && !P3)

7.

Test Requirements for P5:

- P5 = true
  - noMinorChildren <= 3
- P5 = false
  - noMinorChildren > 3

8 and 9.

4 test paths are needed for Edge-Pair Coverage using the prefix graph algorithm

Test Paths	Test Requirements that are toured by test paths directly
[1,2,3,20,8,9,10,11,12,14,15,9,16,17,18]	[1,2,3], [2,3,20], [3,20,8], [8,9,10], [9,10,11], [10,11,12], [11,12,14], [12,14,15], [14,15,9], [15,9,16], [9,16,17], [16,17,18], [20,8,9]
[1,2,3,20,4,6,7,3,20,4,5]	[1,2,3], [2,3,20], [3,20,4], [4,6,7], [6,7,3], [7,3,20], [20,4,5], [20,4,6]
[1,2,3,20,4,6,7,3,20,8,9,10,11,12,13,15,9,10,15,9,10,15,9,16,17,19]	[1,2,3], [2,3,20], [3,20,4], [3,20,8], [4,6,7], [6,7,3], [7,3,20], [8,9,10], [9,10,11], [9,10,15], [10,11,12], [11,12,13], [12,13,15], [13,15,9], [15,9,10], [15,9,16], [9,16,17], [16,17,19], [20,4,6], [20,8,9], [10,15,9]
[1,2,3,20,8,9,16,17,19]	[1,2,3], [2,3,20], [3,20,8], [8,9,16], [9,16,17], [16,17,19], [20,8,9]

Test	Test path in graph	Input	Expected output	Edge Coverage	Edge pair Coverage	Prime path Coverage
T1	[1,2,3,20,8,9,10,11,12,14,15,9,16,17,18]	incomeList[] = [] childAgeList[] = [4]	0		X	
T2	[1,2,3,20,4,6,7,3,20,4,5]	incomeList[] = [500, -50]	-1		X	
T3	[1,2,3,20,4,6,7,3,20,8,9,10,11,12,13,15,9,10,15,9,10,15,9,16,17,19]	incomeList[] = [500] childAgeList[] = [2, 20, 25]	0		X	
T4	[1,2,3,20,8,9,16,17,19]	incomeList[] = [] childAgeList[] = []	0		X	

10.

```java
class TaxCalculatorTest {

    @Test
    public void T1(){
        double[] incomeList = new double[0];
        int[] childAgeList = new int[]{4};

        assertEquals( expected: 0, TaxCalculator.computeTax(incomeList, childAgeList));
    }

    @Test
    public void T2(){
        double[] incomeList = new double[]{500, -50};
        int[] childAgeList = new int[0];

        incomeList = new double[]{500, -50};

        assertEquals( expected: -1, TaxCalculator.computeTax(incomeList, childAgeList));
    }

    @Test
    public void T3(){
        double[] incomeList = new double[]{500};
        int[] childAgeList = new int[]{2, 20, 25};

        assertEquals( expected: 0, TaxCalculator.computeTax(incomeList, childAgeList));
    }

    @Test
    public void T4(){
        double[] incomeList = new double[0];
        int[] childAgeList = new int[0];

        assertEquals( expected: 0, TaxCalculator.computeTax(incomeList, childAgeList));
    }

}
```

11.

TaxCalculator	100% (1/1)	100% (1/1)	89% (17/19)

Number of Classes	Line Coverage		Mutation Coverage		Test Strength	
1	85%	17/20	37%	7/19	41%	7/17

Breakdown by Class

Name	Line Coverage		Mutation Coverage		Test Strength	
TaxCalculator.java	85%	17/20	37%	7/19	41%	7/17