Info 3

Laboratory 7

25.06.2020

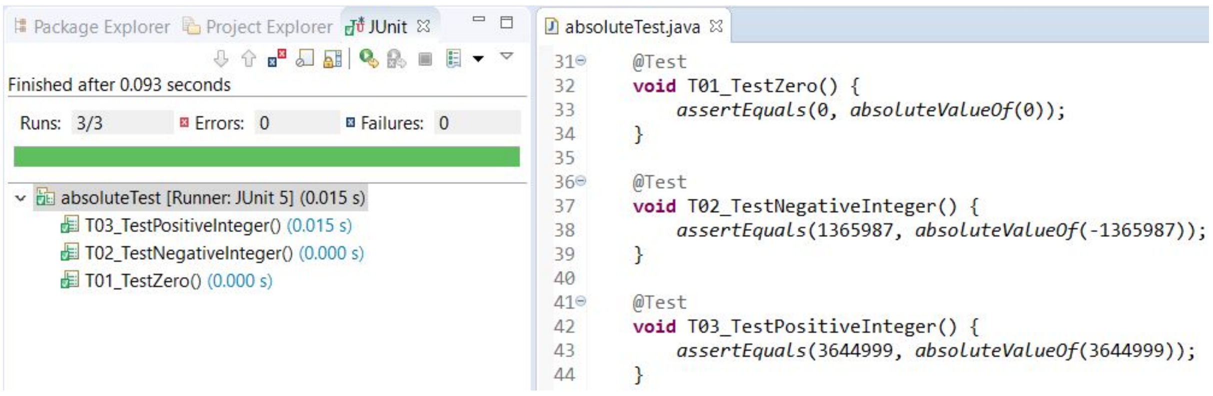
Lab 7: Testing

Niklas Lengert s0563290
Pavel Tsvyatkov s0559632
Robin Jaspers s0568739
Nataliia Azarnykh s0568691

1. **Getting started:** Together with your team write down the equivalence classes for testing a method that returns the absolute value of an integer passed in as a parameter using the black-box methodology. **Don't peek at the code yet!** Now download the implementation absolute.java and give the equivalence classes for a white-box test. Develop one test case for each equivalence class. Now implement the test case using JUnit, if you are programming in Java. You are welcome, however, to use any programming language you with.

Since we don’t know how the program behaves, using the black-box methodology, we discussed and suggested the following equivalence Classes:

- negative Integers
- positive Integers
- zero

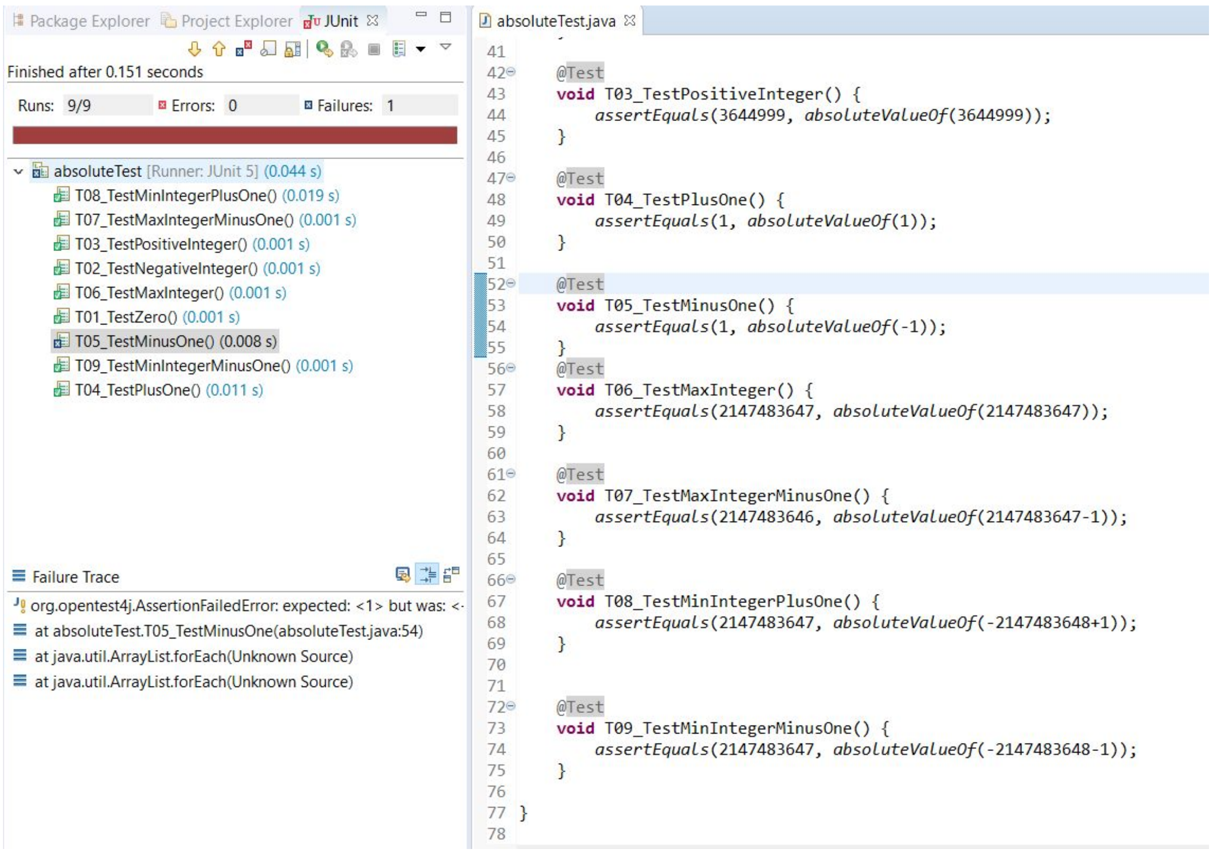


```
Finished after 0.093 seconds
Runs: 3/3    Errors: 0    Failures: 0

absoluteTest [Runner: JUnit 5] (0.015 s)
    T03_TestPositiveInteger() (0.015 s)
    T02_TestNegativeInteger() (0.000 s)
    T01_TestZero() (0.000 s)
```

```java
    @Test
    void T01_TestZero() {
        assertEquals(0, absoluteValueOf(0));
    }

    @Test
    void T02_TestNegativeInteger() {
        assertEquals(1365987, absoluteValueOf(-1365987));
    }

    @Test
    void T03_TestPositiveInteger() {
        assertEquals(3644999, absoluteValueOf(3644999));
    }
```

The tests were successful, but we had some more ideas and continued to test the program further.

We thought the **boundary-value analysis technique** would be efficient in order to find exceptions or bugs here.



```
Finished after 0.151 seconds

Runs: 9/9    Errors: 0    Failures: 1

absoluteTest [Runner: JUnit 5] (0.044 s)
    T08_TestMinIntegerPlusOne() (0.019 s)
    T07_TestMaxIntegerMinusOne() (0.001 s)
    T03_TestPositiveInteger() (0.001 s)
    T02_TestNegativeInteger() (0.001 s)
    T06_TestMaxInteger() (0.001 s)
    T01_TestZero() (0.001 s)
    T05_TestMinusOne() (0.008 s)
    T09_TestMinIntegerMinusOne() (0.001 s)
    T04_TestPlusOne() (0.011 s)

Failure Trace
org.opentest4j.AssertionFailedError: expected: <1> but was: <-
at absoluteTest.T05_TestMinusOne(absoluteTest.java:54)
at java.util.ArrayList.forEach(Unknown Source)
at java.util.ArrayList.forEach(Unknown Source)
```

```java
41
42      @Test
43      void T03_TestPositiveInteger() {
44          assertEquals(3644999, absoluteValueOf(3644999));
45      }
46
47      @Test
48      void T04_TestPlusOne() {
49          assertEquals(1, absoluteValueOf(1));
50      }
51
52      @Test
53      void T05_TestMinusOne() {
54          assertEquals(1, absoluteValueOf(-1));
55      }
56      @Test
57      void T06_TestMaxInteger() {
58          assertEquals(2147483647, absoluteValueOf(2147483647));
59      }
60
61      @Test
62      void T07_TestMaxIntegerMinusOne() {
63          assertEquals(2147483646, absoluteValueOf(2147483647-1));
64      }
65
66      @Test
67      void T08_TestMinIntegerPlusOne() {
68          assertEquals(2147483647, absoluteValueOf(-2147483648+1));
69      }
70
71
72      @Test
73      void T09_TestMinIntegerMinusOne() {
74          assertEquals(2147483647, absoluteValueOf(-2147483648-1));
75      }
76
77  }
78
```

We noticed there’s an error when testing the method with -1 as input. The result should be 1, but was -1 instead.

2. **Black-box test**: Look at this grading scale at the bottom of the page. Give the equivalence classes for a program that loops until a -1 is entered, asking for the number of points on this scale, prints the appropriate letter grade (A-F), and then prints the average number of points when -1 has been entered. Develop one test case for each equivalence class. Test your test cases with **GradingScale.class**—this is a Java program. No fair decompiling the class! Report on the results. Did you find any errors? Document the errors found, but don't correct the code.

Equivalence Classes(partitions):

Invalid input	Valid input	Invalid input
Less than 0	Between 0 and 100	More than 100

Then we decided to write down the points and the corresponding letter grade for easier tests.

1. Points 90-100 → A
2. Points 75-89 → B
3. Points 60-74 → C
4. Points 50-59 → D
5. Points 0-49 → F
6. Enter -1 → Average Points

Develop one test case for each equivalence class

We expect a generic invalid input message to be shown if we test with values from the invalid input (below 0 and above 100).

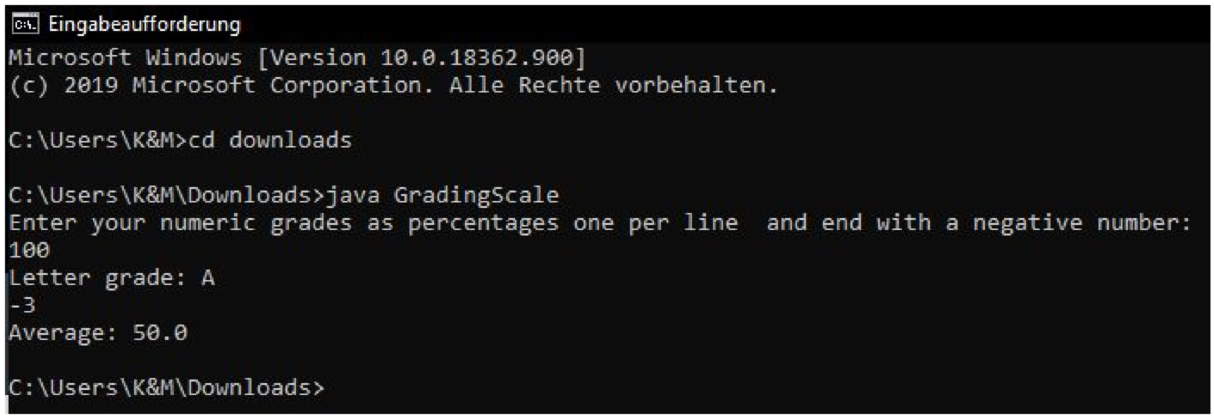
assertEquals("some invalid input message", -5)
assertEquals(A, 91)
assertEquals("some invalid input message", 150)

We wanted to think of some additional tests and decided to check the boundaries again.

For input 50, we expect the output to be D
For input 60, we expect the output to be C
For input 75, we expect the output to be B
For input 90, we expect the output to be A

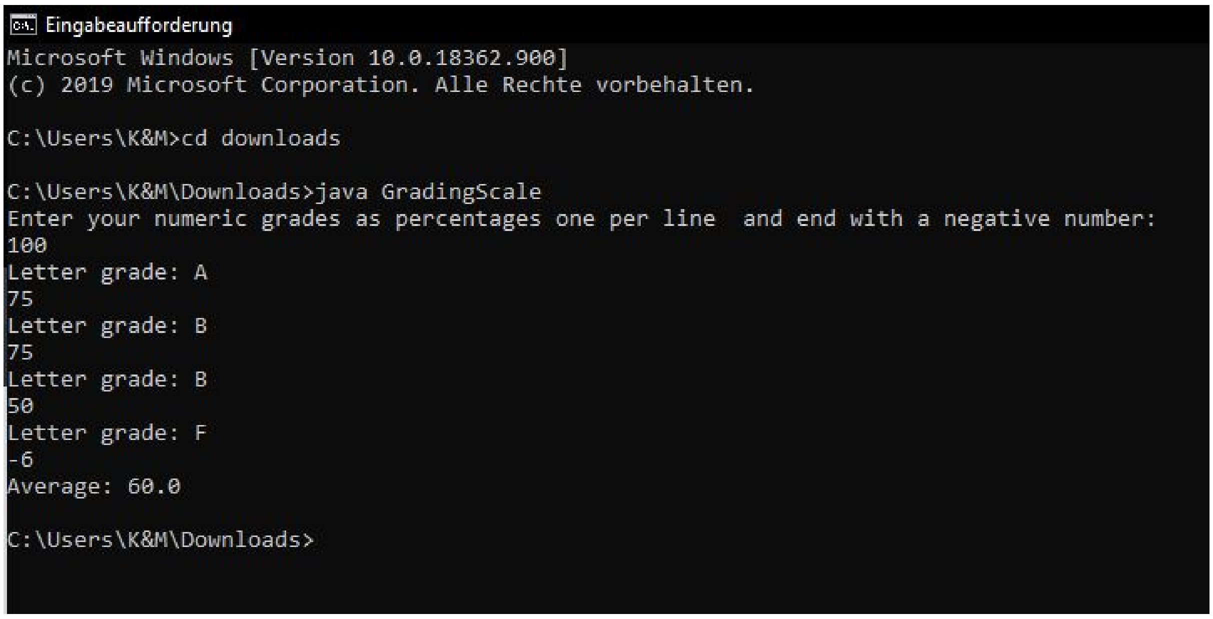
Errors we found:

The calculation of the average score is wrong. Usually you would take the total points and divide them by the amount of tests.
However, the negative input that should trigger the average score calculation increases the number of tests by one. The following example takes one test with 100 points, so the average should also be 100 points, but it says 50, because it does not divide the total by one, but by **two**.



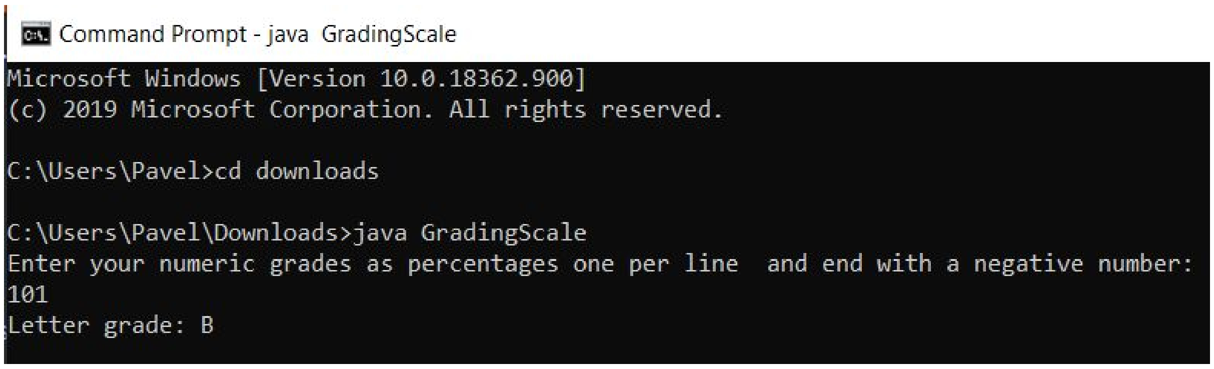
```
Eingabeaufforderung
Microsoft Windows [Version 10.0.18362.900]
(c) 2019 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\K&M>cd downloads

C:\Users\K&M\Downloads>java GradingScale
Enter your numeric grades as percentages one per line  and end with a negative number:
100
Letter grade: A
-3
Average: 50.0

C:\Users\K&M\Downloads>
```

The same happens in the next example where the average score should be 75 but it is 60.



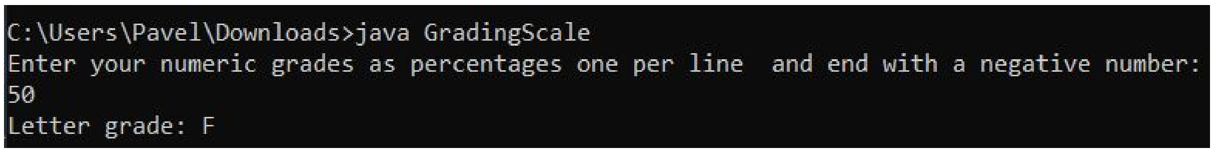
```
Eingabeaufforderung
Microsoft Windows [Version 10.0.18362.900]
(c) 2019 Microsoft Corporation. Alle Rechte vorbehalten.

C:\Users\K&M>cd downloads

C:\Users\K&M\Downloads>java GradingScale
Enter your numeric grades as percentages one per line  and end with a negative number:
100
Letter grade: A
75
Letter grade: B
75
Letter grade: B
50
Letter grade: F
-6
Average: 60.0

C:\Users\K&M\Downloads>
```

We decided to test with an invalid input. Entering any number above 100 would always return “B”.


Command Prompt - java GradingScale

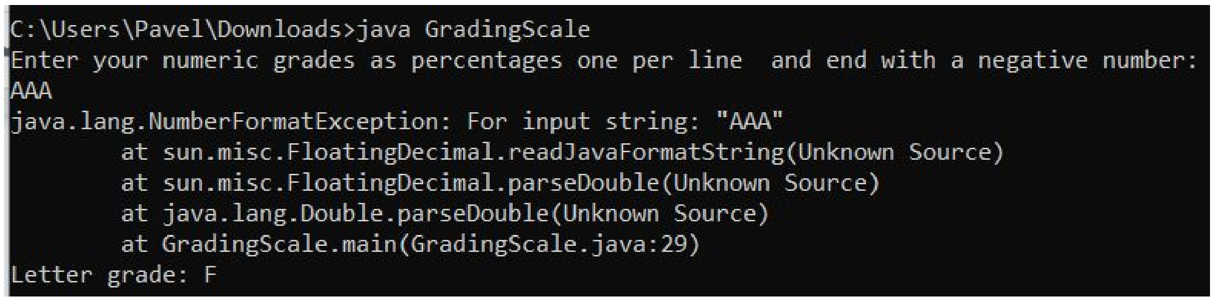
```
Microsoft Windows [Version 10.0.18362.900]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\Pavel>cd downloads

C:\Users\Pavel\Downloads>java GradingScale
Enter your numeric grades as percentages one per line  and end with a negative number:
101
Letter grade: B
```

We noticed that testing with input 50 was returning F instead of D.



```
C:\Users\Pavel\Downloads>java GradingScale
Enter your numeric grades as percentages one per line  and end with a negative number:
50
Letter grade: F
```

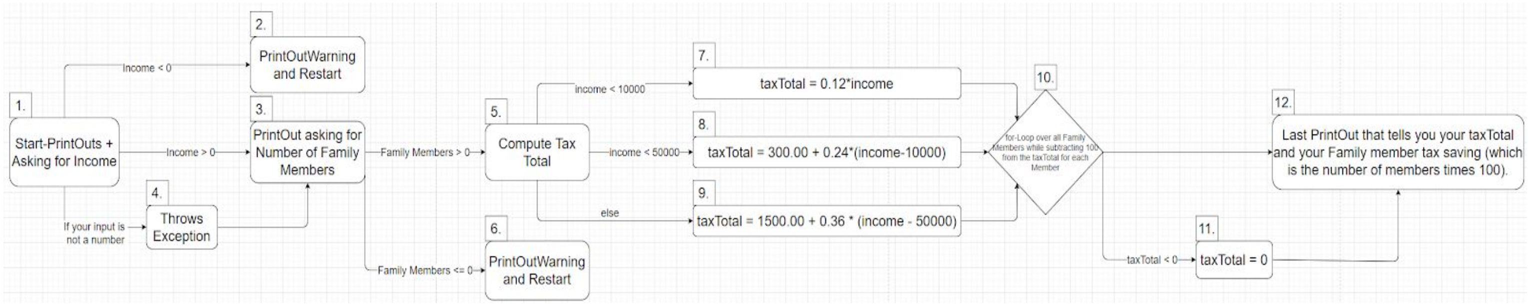
Another issue we found was that even after entering a non-numeric value the program returns “Letter grade:F”

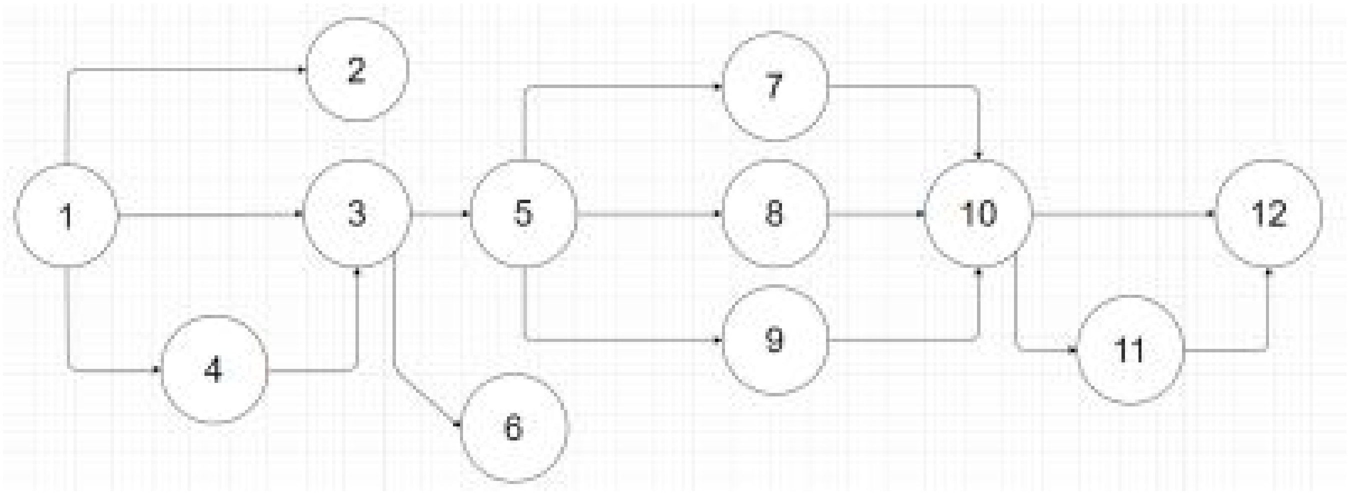


```
C:\Users\Pavel\Downloads>java GradingScale
Enter your numeric grades as percentages one per line  and end with a negative number:
AAA
java.lang.NumberFormatException: For input string: "AAA"
        at sun.misc.FloatingDecimal.readJavaFormatString(Unknown Source)
        at sun.misc.FloatingDecimal.parseDouble(Unknown Source)
        at java.lang.Double.parseDouble(Unknown Source)
        at GradingScale.main(GradingScale.java:29)
Letter grade: F
```

3. White-box test / path coverage: Examine the code for **TaxTime.java**. Draw a code graph of the main class! How many independent paths are there? What are the conditions that cause each of the paths to be taken? Draw up a table giving you an overview of the conditions. Design and implement test cases that exercise each path. Are there any errors in the program (besides the size of the tax bite)? Document the errors found, don't correct the code.

We drew the following chart and graph for the program.


1.
Start-PrintOuts + Asking for Income
Income < 0
2.
PrintOutWarning and Restart
Income > 0
3.
PrintOut asking for Number of Family Members
If your input is not a number
4.
Throws Exception
Family Members > 0
Family Members <= 0
5.
Compute Tax Total
6.
PrintOutWarning and Restart
income < 10000
7.
taxTotal = 0.12*income
income < 50000
8.
taxTotal = 300.00 + 0.24*(income-10000)
else
9.
taxTotal = 1500.00 + 0.36 * (income - 50000)
10.
for-Loop over all Family Members while subtracting 100 from the taxTotal for each Member
taxTotal < 0
11.
taxTotal = 0
12.
Last PrintOut that tells you your taxTotal and your Family member tax saving (which is the number of members times 100).



How many independent paths are there?

We wanted to make sure we understand what an independent path is exactly. Source:https://binaryterms.com/basis-path-testing.html

An independent path is one that represents a path in the program that traces a new set of procedural statements or conditions. If we take it in the context of a flow graph, the independent path traces the edges in the flow graph that are not traversed before the path is defined.

Independent Paths in the TaxTime program:

Path 1: 1-2

Path 2: 1-3-5-7-10-11-12

Path 3: 1-4-3-6

Path 4: 1-3-5-8-10-12

Path 5: 1-3-5-9-10-12

What are the conditions that cause each of the paths to be taken?

To take path 1: income has to be < 0

To take path 2: income has to be > 0 and < 10 000, nFamilyMembers has to be > 0, taxTotal < 0

To take path 3: if income is not a number, nFamilyMembers has to be 0

To take path 4: income has to be > 0 and < 50 000, nFamilyMembers has to be > 0, taxTotal > 0

To take path 5: nFamilyMembers has to be > 0, income has to be >= 50 000, taxTotal > 0

To take path 6: nFamilyMembers has to be >0, income has to be a word

Draw up a table giving you an overview of the conditions.

Condition	Path that will be taken
Condition 1: If input for income is smaller than 0.	Print Out and restart
Condition 2: If input for family members is smaller or equal to 0.	Print Out and restart
Condition 3: If your income is smaller than 10000.	Calculation of your taxTotal: taxTotal = 0.12*income
Condition 4: If your income is smaller than 50000.	Calculation of your taxTotal: taxTotal = 300.00 + 0.24*(income-10000)
Condition 5: If your income is bigger than 50000 or equals it.	Calculation of your taxTotal: taxTotal = 1500.00 + 0.36*(income-50000)
Condition 6: If your taxTotal is smaller than 0.	Set taxTotal to 0.
Condition 7: If input for income is a word	Set taxTotal to 0.

Design and implement test cases that exercise each path.

Path 1 test: input -1 for income, expected output: program prints a message("Start over") and stops.

Path 2 test: input number 8 000 for income, input 1 for FamilyMembers.

Expected output: tax is 760.0 €, family member tax saving is 100€.

Path 3 test: input a word "Test" for income. Input "0" for FamilyMembers.

Expected output: program prints an exception after first input(input of a word "Test") and prints "Did you forget to count yourself? Start over." and stops after the second input ("0").

Path 4 test: input an income 45 000, input 1 for Family members.

Expected output: tax is 8500.0 €. Family member tax saving is 100€.

Path 5 test: input an income 50000, input 2 for Family members.

Expected output: tax is 1200.0 €. Family member tax saving is 200€.

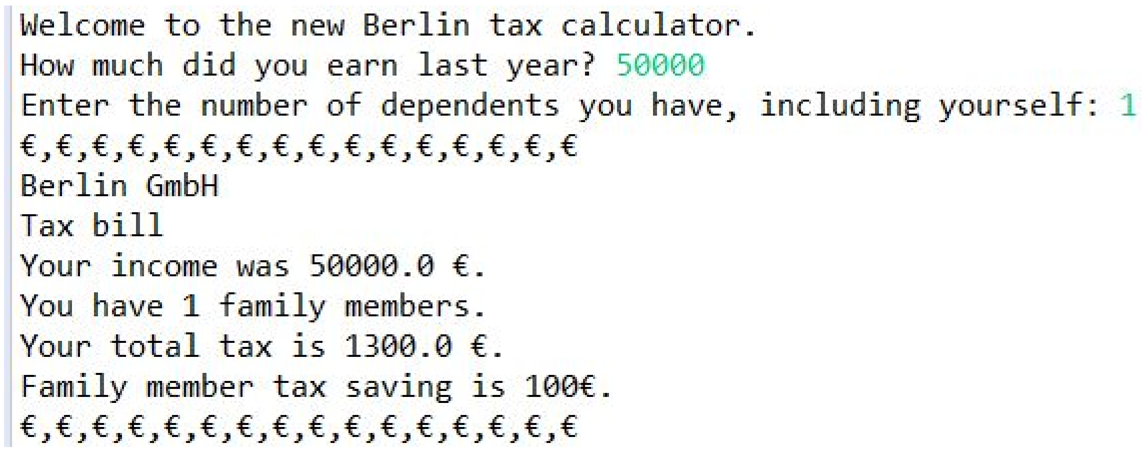
Additional test: input an income “Test”, input 1 for Family members.

Expected output: program prints exception after the first input. Tax is 0 €. Family member tax saving is 100€.

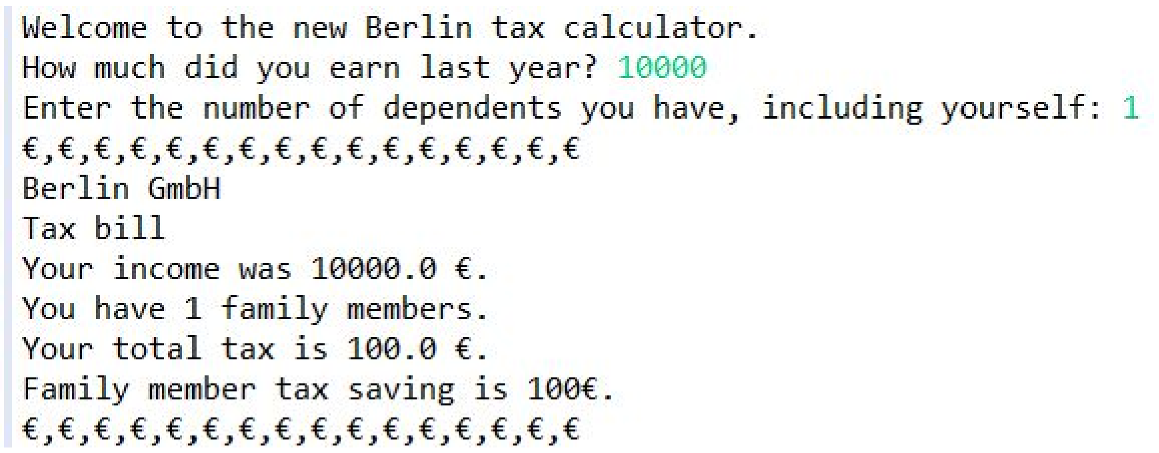
We found the following errors in the program:

A problem that we have with the current version is that the tax for exactly 10.000 and 50.000 income is wrong, because of how the conditions in the tax total computation are set. Both the 10 000 and 50 000 inputs go into a computation where they cause one of the multipliers to become zero. This should definitely not happen, because it changes the result dramatically for those two numbers.

If we input income 50 000 and 1 dependent, the TotalTax is 1300. That’s not correct and it happens, because we step into the else statement which multiplies with (income - 50 000), which evaluates to 0. In the end the for loop.

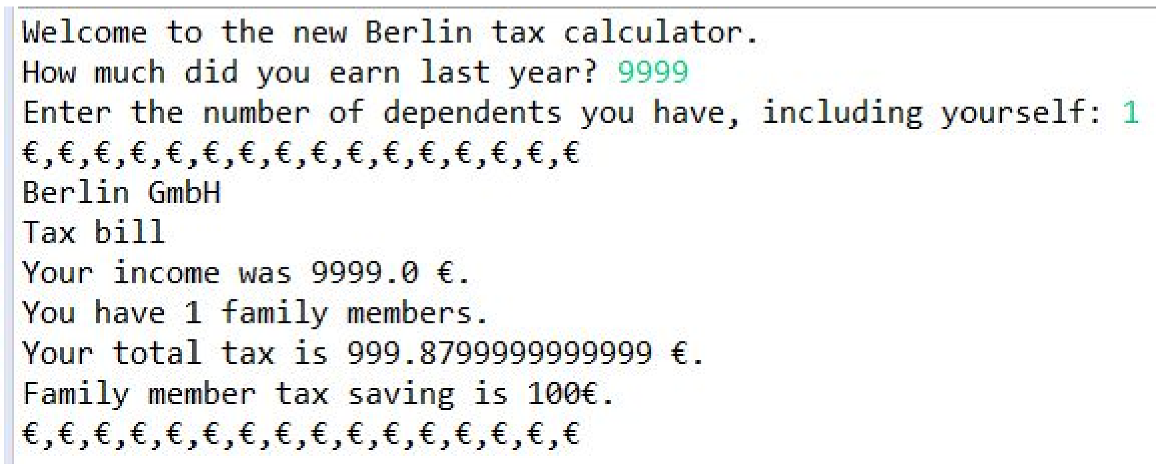


```
Welcome to the new Berlin tax calculator.
How much did you earn last year? 50000
Enter the number of dependents you have, including yourself: 1
€,€,€,€,€,€,€,€,€,€,€,€,€,€,€,€
Berlin GmbH
Tax bill
Your income was 50000.0 €.
You have 1 family members.
Your total tax is 1300.0 €.
Family member tax saving is 100€.
€,€,€,€,€,€,€,€,€,€,€,€,€,€,€,€
```



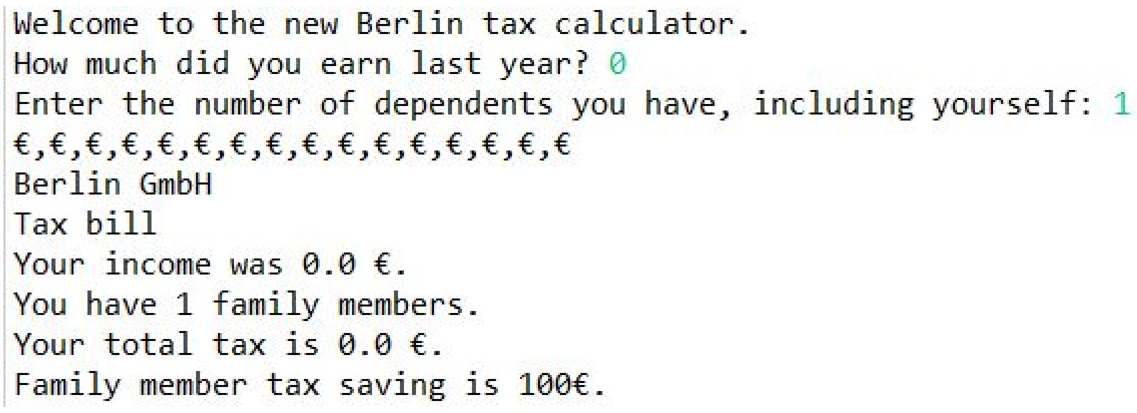
```
Welcome to the new Berlin tax calculator.
How much did you earn last year? 10000
Enter the number of dependents you have, including yourself: 1
€,€,€,€,€,€,€,€,€,€,€,€,€,€,€,€
Berlin GmbH
Tax bill
Your income was 10000.0 €.
You have 1 family members.
Your total tax is 100.0 €.
Family member tax saving is 100€.
€,€,€,€,€,€,€,€,€,€,€,€,€,€,€,€
```

If we input income 10 000 and 1 dependent, the TotalTax is 100. We thought that it’s not the correct behaviour, because we tested it with income 9999 and the TotalTax was almost 1000.



```
Welcome to the new Berlin tax calculator.
How much did you earn last year? 9999
Enter the number of dependents you have, including yourself: 1
€,€,€,€,€,€,€,€,€,€,€,€,€,€,€,€
Berlin GmbH
Tax bill
Your income was 9999.0 €.
You have 1 family members.
Your total tax is 999.8799999999999 €.
Family member tax saving is 100€.
€,€,€,€,€,€,€,€,€,€,€,€,€,€,€,€
```

It didn’t make sense for us that the family member tax saving is separated from your own income and total tax because in the current version of the program you can get tax savings even though you do not work (have 0.00€ income). There is a private variable at the beginning which is called exemption which is never used.



```
Welcome to the new Berlin tax calculator.
How much did you earn last year? 0
Enter the number of dependents you have, including yourself: 1
€,€,€,€,€,€,€,€,€,€,€,€,€,€,€,€
Berlin GmbH
Tax bill
Your income was 0.0 €.
You have 1 family members.
Your total tax is 0.0 €.
Family member tax saving is 100€.
```

4. Test Cases: Define some test cases (at least one per team member) for your Corona-App!

Test Case 1:

An test case for a symptom input would be:
user.healthStatus.addSymptoms(“temperature”);
assertEquals(“temperature”, user.getHealthStatus());

If a user has inputted a symptom “temperature”, the expected output when we call the user’s getHealthStatus() is “temperature”

Test Case 2:

If a user inputs a symptom, we expect that the proper timestamp is saved(shown).

Test Case 3:

If a user start the app for the first time, he should be taken to the “First-Time User Page” accordingly

Test Case 4:

If a user inputs a previous condition, it should be properly shown(outputted) when we check the user’s health status.

Test Case 5:

When a user inputs the intensity for their symptom, it should be of type int.

Test Case 6:
When we check a user’s health status, we want to find out whether a symptom is part of the previous conditions or not.

Input: Symptom as previous condition, Expected output: true
Input: Symptom not as previous condition, Expected output: false

5. Reflection: Consider the last examples TaxTime and GradingScale. What makes these programs hard to test? How would you refactor TaxTime to make it testable automatically with JUnit?

TaxTime would be testable with JUnit if we would refactor it in a way so that it does not have only one method, which in this case is the main method. If TaxTime had separate methods which handle each task, like one for the taxTotal, one for the inputs of family members and your income, then it would be much easier to test. We would be able to call methods and write JUnit assertions and be able to automate the test cases. The tax saving should also not be calculated in the print message by multiplying nFamilyNumbers with 100. It would make more sense if we made the tax saving an additional variable and use it in the print message.

The problem about GradingScale is that we can't access the methods which are needed for JUnit. We have to manually test everything we would like to test and run the program over and over after every negative input, until we have finally tested all the inputs that we would like to check.

It is also very complicated to grab information from console without touching the initial code.

Reflection:

Niklas: This was very refreshing to work with some code and actually opening an IDE after almost an entire semester. We struggled a little bit with the visualization of the code path but everything else was doable. We thought about the tests a lot but only to see if we had all the options considered.

Pavel: It was interesting to test the classes and write JUnit assertions. I also got to use the Coverage tool in Eclipse which highlighted the code based on the condition that was executed according to what I had as input. It was really helpful and it helped us find some of the errors in the TaxTime program.

Robin: It's been a while for me using java so this exercise was very refreshing. One of the harder things of this exercise was probably setting up JUnit for intellij.It took some time to find out how it works and without internet i wouldn't be able to. JUnit definitely makes life easier when you are able to do white-box testing. All in all this exercise was about attention to not miss any errors.

Natalia:
This time we had an opportunity to work with some code. It was very pleasant. This exercise was extremely important and interesting for me, because somehow related to my job. Taxes bites are indeed very funny here. What I missed is acceptance criteria for the TaxTime program. How it should work? What does "Family member tax saving" means? Is it saving for each member or for the whole family? If taxes are 1600, and the amount of family members is 2, should we subtract 200 euro, which stays in "Family member tax saving" or should we subtract 300 euro, like it stays in the program? If it was a real situation, I would ask a project manager about the logic that stays behind and about expected behaviour. Now I am not sure if it's a bug or a feature. I it's my favourite Lab so far.

Time spent:

Task 1	2h
Task 2	2h
Task 3	3h
Task 4	30 min
Task 5	30 min

Appendix:

Absolute.java

```
public class absolute {


        // Code for White-box test exercise
        public static int absoluteValueOf (int x) {
          if (x < -1)
             return -x;
          else
             return x;
        }

        public static void main(String[] args) {


                System.out.println(absoluteValueOf(55));


        }
}
```

AbsoluteTest.java

```
import static org.junit.jupiter.api.Assertions.*;
import static org.junit.Assert.*;

import org.junit.After;
import org.junit.Before;
import org.junit.jupiter.api.Test;
```

```
class absoluteTest {

        static int a = Integer.MIN_VALUE +1;
        public static int absoluteValueOf (int x) {
                  if (x < -1)
                     return -x;
                  else
                     return x;
                }
         public absoluteTest()
            {
            }

            @Before
            public void setUp()
            {
            }

            @After
            public void tearDown()
            {
            }

        @Test
        void T01_TestZero() {
                assertEquals(0, absoluteValueOf(0));
        }

        @Test
        void T02_TestNegativeInteger() {
                assertEquals(1365987, absoluteValueOf(-1365987));
        }

        @Test
        void T03_TestPositiveInteger() {
                assertEquals(3644999, absoluteValueOf(3644999));
        }

        @Test
        void T04_TestPlusOne() {
                assertEquals(1, absoluteValueOf(1));
        }

        @Test
        void T05_TestMinusOne() {
                assertEquals(1, absoluteValueOf(-1));
        }
        @Test
```

```
	void T06_TestMaxInteger() {
		assertEquals(2147483647, absoluteValueOf(2147483647));
	}

	@Test
	void T07_TestMaxIntegerMinusOne() {
		assertEquals(2147483646,
absoluteValueOf(2147483647-1));
	}

	@Test
	void T08_TestMinIntegerPlusOne() {
		assertEquals(2147483647,
absoluteValueOf(-2147483648+1));
	}

	@Test
	void T09_TestMinIntegerMinusOne() {
		assertEquals(2147483647,
absoluteValueOf(-2147483648-1));
	}
}
```

TaxTime.java

```
package TaxTask;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

/*
 * Created on 06.01.2006, updated 2020-06-18
 */

/**
 * @author weberwu
 * Calculating a Berlin tax that could be done on a beer mat.
 */
public class TaxTime {
    static int nFamilyMembers;
    static int exemption;
    static double income;
    static double taxTotal;

    public static void main(String[] args) {
        // A Reader stream to read from the console
        BufferedReader in =
            new BufferedReader(new InputStreamReader(System.in));

        System.out.println ("Welcome to the new Berlin tax calculator.");
        System.out.print   ("How much did you earn last year? ");
        try {
            income = Double.parseDouble(in.readLine());
        } catch (NumberFormatException e1) {
            e1.printStackTrace();
        } catch (IOException e1) {
            e1.printStackTrace();
        }

//      check income

        if (income < 0) {
            System.out.println ("Even in Berlin, no one has a negative
income!");
            System.out.println ("Start over.");
            System.exit (-1);
        }

        System.out.print("Enter the number of dependents you have,
including yourself: ");
        try {
            String s = in.readLine();
            nFamilyMembers = Integer.valueOf(s).intValue();
        } catch (IOException e) {
            e.printStackTrace();
        }

//      check number of family members

       if (nFamilyMembers <= 0) {
          System.out.println("Did you forget to count yourself?");
          System.out.println ("Start over.");
          System.exit (-1);
          }

//     compute tax total

      if (income < 10000)
         taxTotal = 0.12 * income;
    //  System.out.println("in the if");}
      else if (income < 50000)
         taxTotal = 300.00 + 0.24 * (income - 10000);
     //    System.out.println("in the else if");}
      else
        //  System.out.println("in the else");
         taxTotal = 1500.00 + 0.36 * (income - 50000);

      for (int i = 0; i <= nFamilyMembers; i++){
          taxTotal = taxTotal - 100;
      }

//    check negative tax

   if (taxTotal < 0) // In case of negative tax
       taxTotal=0;

    System.out.println ("€,€,€,€,€,€,€,€,€,€,€,€,€,€,€,€");
    System.out.println ("Berlin GmbH");
    System.out.println ("Tax bill");
    System.out.println ("Your income was " + income + " €.");
    System.out.println ("You have " + nFamilyMembers + " family
members.");
    System.out.println ("Your total tax is " + taxTotal + " €.");
    System.out.println ("Family member tax saving is " +
nFamilyMembers*100 + "€.");
    System.out.println ("€,€,€,€,€,€,€,€,€,€,€,€,€,€,€,€");
    }
}
```