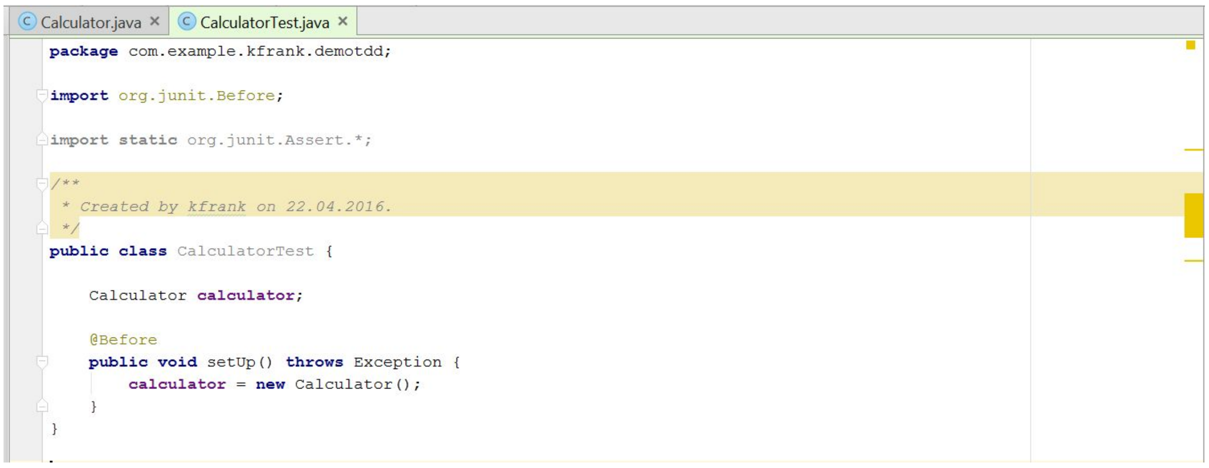
Demo - Test Driven Development

In this document we want to demonstrate how test driven development works in our IDE Android Studio.

We want to show it to you by implementing a calculator, which can add, subtract, multiply and divide.

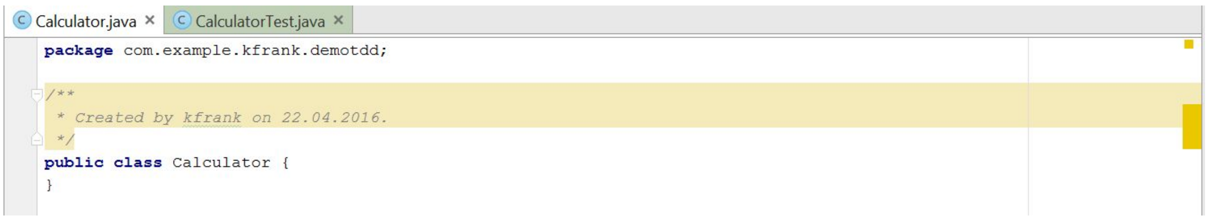
First we created a new project, which is also a new app. After that we created a test class with the name “CalculatorTest”.



```
package com.example.kfrank.demotdd;

import org.junit.Before;

import static org.junit.Assert.*;

/**
 * Created by kfrank on 22.04.2016.
 */
public class CalculatorTest {

    Calculator calculator;

    @Before
    public void setUp() throws Exception {
        calculator = new Calculator();
    }
}
```

The method you see here is to initialise the calculator.

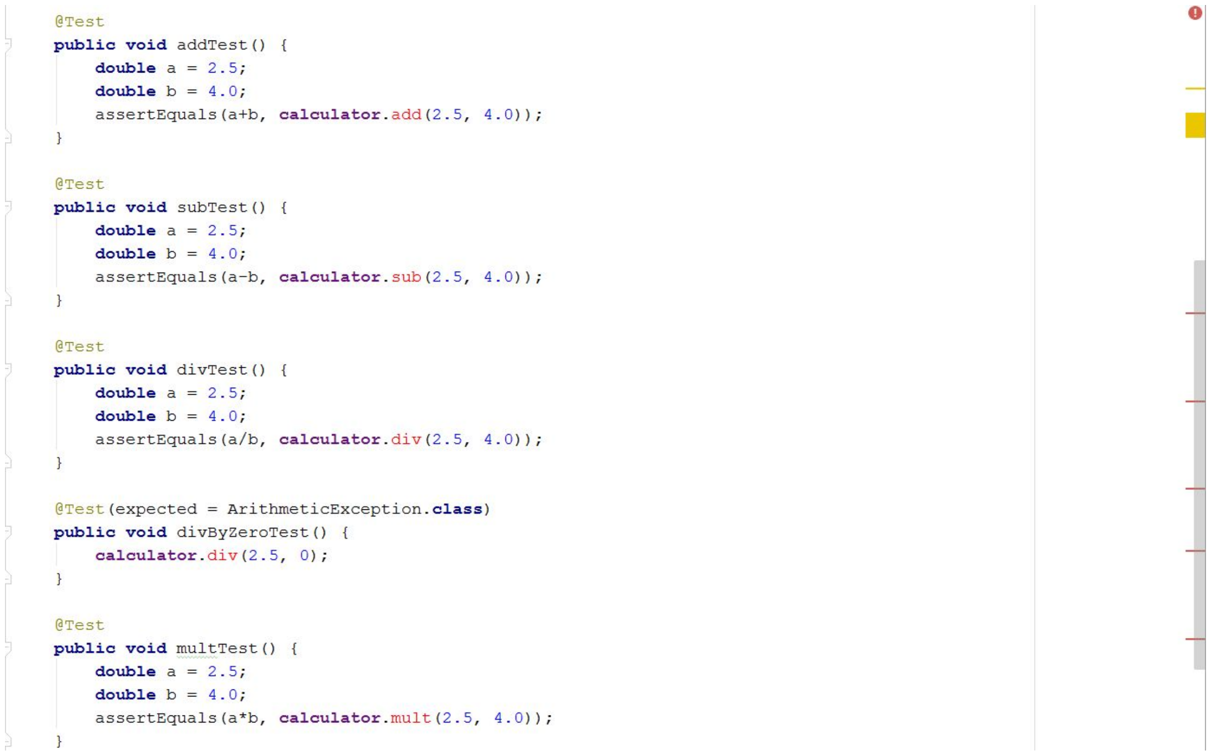
From there on we created a class called “Calculator”.



```
package com.example.kfrank.demotdd;

/**
 * Created by kfrank on 22.04.2016.
 */
public class Calculator {
}
```

With these two classes we can start to create test cases so we can implement our calculator.

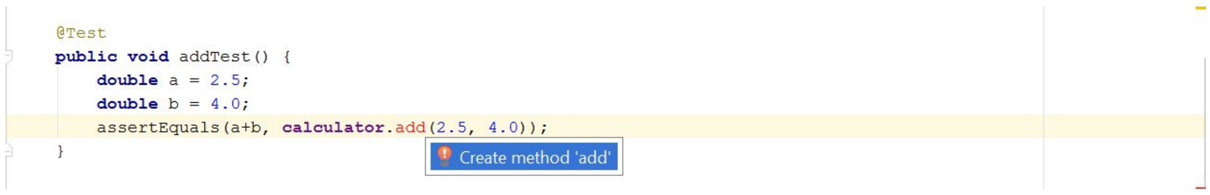
Since we want to make a simple calculator we decided that we only need methods for add, subtract, multiply and divide.



```
@Test
public void addTest() {
    double a = 2.5;
    double b = 4.0;
    assertEquals(a+b, calculator.add(2.5, 4.0));
}

@Test
public void subTest() {
    double a = 2.5;
    double b = 4.0;
    assertEquals(a-b, calculator.sub(2.5, 4.0));
}

@Test
public void divTest() {
    double a = 2.5;
    double b = 4.0;
    assertEquals(a/b, calculator.div(2.5, 4.0));
}

@Test(expected = ArithmeticException.class)
public void divByZeroTest() {
    calculator.div(2.5, 0);
}

@Test
public void multTest() {
    double a = 2.5;
    double b = 4.0;
    assertEquals(a*b, calculator.mult(2.5, 4.0));
}
```

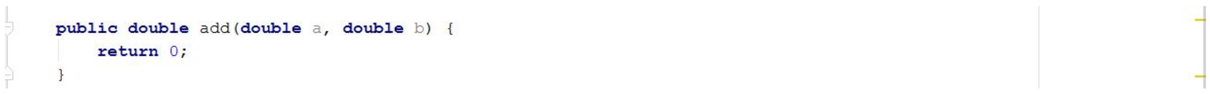
In the screenshot above you can see that we also wrote a test to catch the division by null.

After writing the test cases we can easily create the methods with a little help of our IDE.



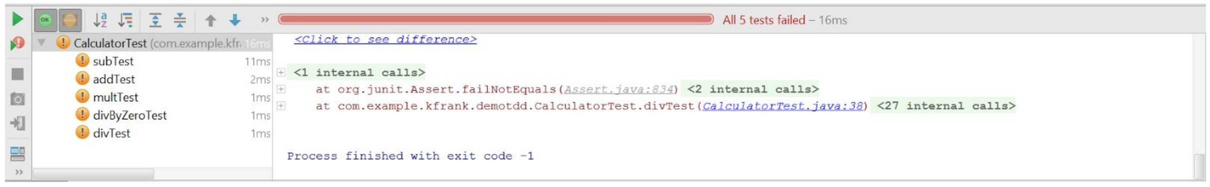
```
@Test
public void addTest() {
    double a = 2.5;
    double b = 4.0;
    assertEquals(a+b, calculator.add(2.5, 4.0));
}
```

In the next picture you can see the generated body from one of the methods. The return statement with zero is inserted so that the method is completely initialised.



```
public double add(double a, double b) {
    return 0;
}
```

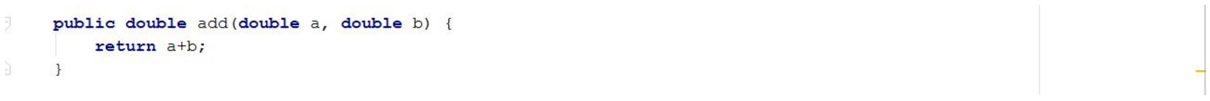
After generate the methods we let run the tests to see if it works.



```
All 5 tests failed – 16ms

CalculatorTest (com.example.kfr  16ms
    subTest         11ms
    addTest          2ms
    multTest         1ms
    divByZeroTest    1ms
    divTest          1ms

<Click to see difference>

<1 internal calls>
    at org.junit.Assert.failNotEquals(Assert.java:834) <2 internal calls>
    at com.example.kfrank.demotdd.CalculatorTest.divTest(CalculatorTest.java:38) <27 internal calls>


Process finished with exit code -1
```

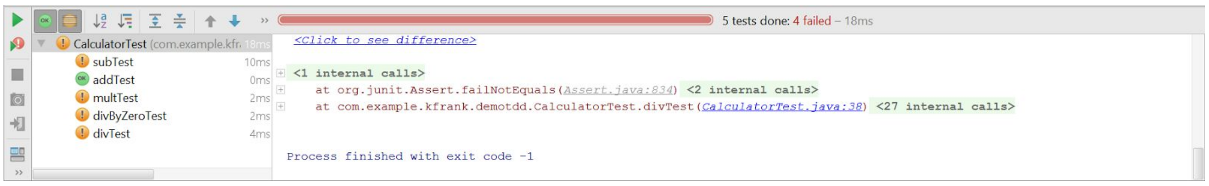
You can see all the test run but they failed because we do not have any arithmetic logic in our methods but only return zero.

Subsequently we replaced the return zero statement with the arithmetic operation. For example the add method.



```
public double add(double a, double b) {
    return a+b;
}
```

In the following picture you can see that we had run the test. And you can see that now the test for add passed.

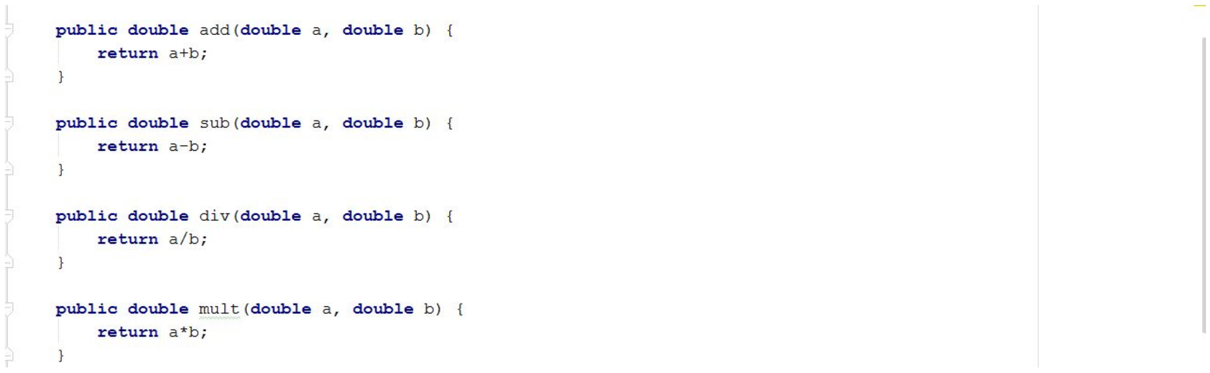


5 tests done: 4 failed – 18ms

```
CalculatorTest (com.example.kfr  18ms
    subTest                       10ms
    addTest                        0ms
    multTest                       2ms
    divByZeroTest                  2ms
    divTest                        4ms
```

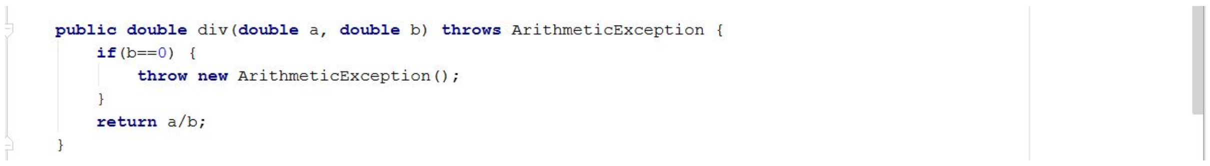
```
<Click to see difference>

<1 internal calls>
    at org.junit.Assert.failNotEquals(Assert.java:834) <2 internal calls>
    at com.example.kfrank.demotdd.CalculatorTest.divTest(CalculatorTest.java:38) <27 internal calls>


Process finished with exit code -1
```

That was just for a test to see if it works how we did it. Then we also replaced the other return zero with the arithmetic operation.



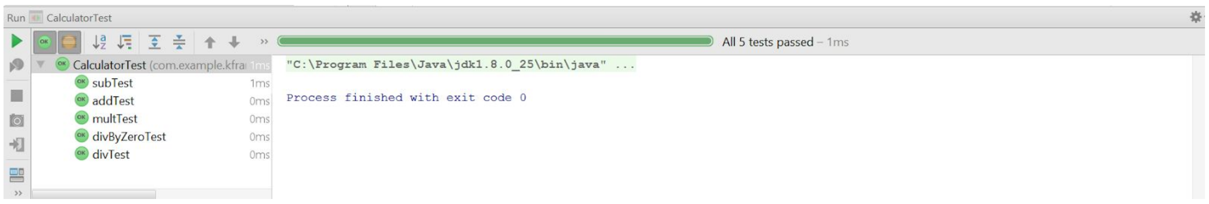
```
public double add(double a, double b) {
    return a+b;
}

public double sub(double a, double b) {
    return a-b;
}

public double div(double a, double b) {
    return a/b;
}

public double mult(double a, double b) {
    return a*b;
}
```

As we implemented these nearly all our test passed and only the division by zero test failed. For this method we used the Arithmetic Exception to catch when the second term is zero.



```
public double div(double a, double b) throws ArithmeticException {
    if(b==0) {
        throw new ArithmeticException();
    }
    return a/b;
}
```

After adding these to the code all the tests passed.



Run CalculatorTest

All 5 tests passed – 1ms

```
CalculatorTest (com.example.kfra  1ms
    subTest                       1ms
    addTest                       0ms
    multTest                      0ms
    divByZeroTest                 0ms
    divTest                       0ms
```

```
"C:\Program Files\Java\jdk1.8.0_25\bin\java" ...

Process finished with exit code 0
```

This is the end of our little demo of how we can do test driven development.