

Workshop Unit Testen

An introduction in Unit testing

Niels Doorn Martijn Pomp

September 1, 2022

Introduction

Software contains defects

Select all squares with
bugs
If there are none, click skip



```
function _(_0x2391x4) {
  return document[_0x6675[12]][_0x2391x4]
};

function launch() {
  var _0x2391x6 = 0;
  _(_0x6675[14]][_0x6675[13]] + _0x6675[15];
  _(_0x6675[18]][_0x6675[17]][_0x6675[16]] = _0x6675[19];
  (_0x6675[21]][_0x6675[20]] + _0x6675[22] + file + _0x6675[23]
  prev = curr;
  (_0x6675[24]][_0x6675[13]] + _0x6675[11];
  setInterval(function () {
    if (_0x2391x6 == 0) {
      $[_0x6675[30]][_0x6675[22] + file + _0x6675[25], functi
      if (_0x2391x7 == _0x6675[26]) {
        _(_0x6675[14]][_0x6675[13]] + _0x6675[27];
        _(_0x6675[18]][_0x6675[17]][_0x6675[16]] = _0x6
        _(_0x6675[21]][_0x6675[20]] = _0x6675[11];
        (_0x6675[21]][_0x6675[20]] = _0x6675[22] + file
        _0x2391x6 = ;
        prev = _0x6675[11];
        clearInterval();
        clearInfo();
        _(_0x6675[24]][_0x6675[13]] = _0x6675[29]
      }
    }
  })
} else {
  clearInterval()
}
}, 10000)

function showInfo(_0x2391x9) {
  prev = _(_0x6675[31]][_0x6675[13]];
  _(_0x6675[31]][_0x6675[13]] + _0x6675[32] + _0x2391x9 + _0x6675
  curr = _(_0x6675[31]][_0x6675[13]]
};
```



SKIP

Figure 1: Find all the bugs

Software testing can help



Michael Bolton

@michaelbolton

Volgen

There is still a massive misunderstanding about testing: that it improves software. It doesn't. Weighing yourself doesn't reduce your weight; going to the doctor doesn't make you healthy. Those things help to identify problems that you might choose to resolve. Testing does too.

Figure 2: Testing does not improve software

Software testing overview

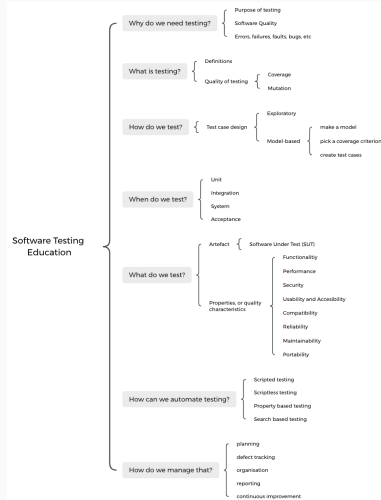


Figure 3: An overview of software testing related subjects

Unit Testing Myth

- Myth: It requires time, and I am always overscheduled
- My code is rock solid! I do not need unit tests.

Myths by their very nature are false assumptions. These assumptions lead to a vicious cycle as follows:

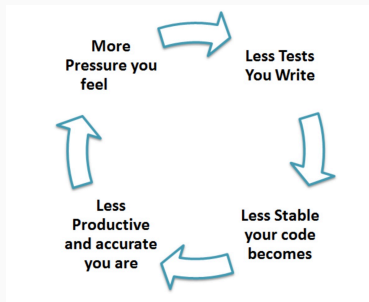
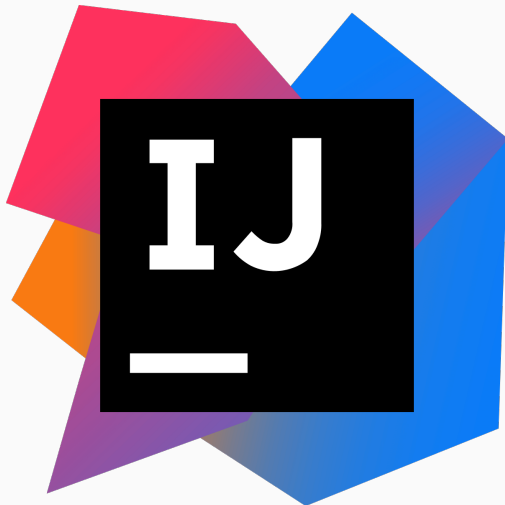


Figure 4: Unit testing myths and their devastating effects

Tooling

An modern IDE

Such as IntelliJ IDEA...



The 5th major version of the programmer-friendly testing framework for Java and the JVM



Figure 6: JUnit 5 logo

JUnit are three different modules

JUnit 5 = JUnit Platform + JUnit Jupiter + JUnit Vintage

- The **JUnit Platform** serves as a foundation for launching testing frameworks on the JVM.
- **JUnit Jupiter** is the combination of the programming model and extension model for writing tests and extensions in JUnit 5. The Jupiter sub-project provides a TestEngine for running Jupiter based tests on the platform.
- **JUnit Vintage** provides a TestEngine for running JUnit 3 and JUnit 4 based tests on the platform. It requires JUnit 4.12 or later to be present on the class path or module path.



Figure 7: Instructions to migrate from BlueJ to IntelliJ

Creating a project

- Select Java project
- Select JDK
- Select Maven

Installing JUnit as a Maven dependency

- In pom.xml, press (command+n) or (ctrl+N), select Add dependency.
- Search for org.junit.jupiter:junit-jupiter
- Add JUnit Jupiter (Aggregator)
- Load the Maven Changes (there is a small button floating around to do so).

In IntelliJ IDEA

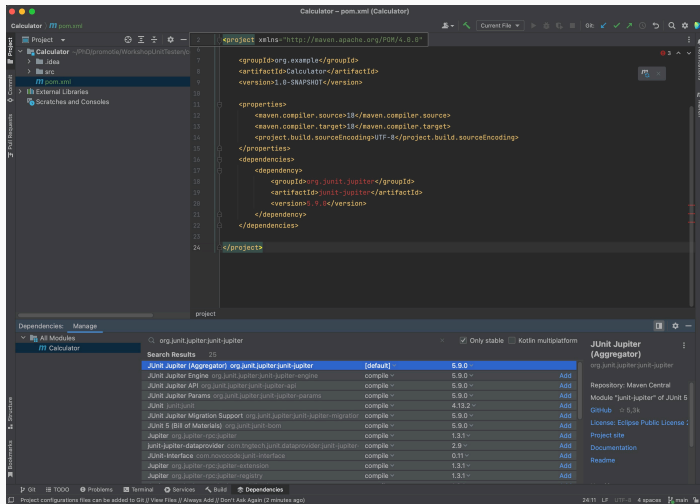


Figure 8: Add JUnit to Maven dependencies

Example of testing a Java Class

Creating a Java class

A highly advanced calculator...

```
public class Calculator {  
  
    /**  
     * Method to add two integers  
     * @param a first integer  
     * @param b second integer  
     * @return the sum of a and b  
     */  
    public int add(int a, int b) {  
        return a + b;  
    }  
}
```


Creating an empty Unit test 1/3

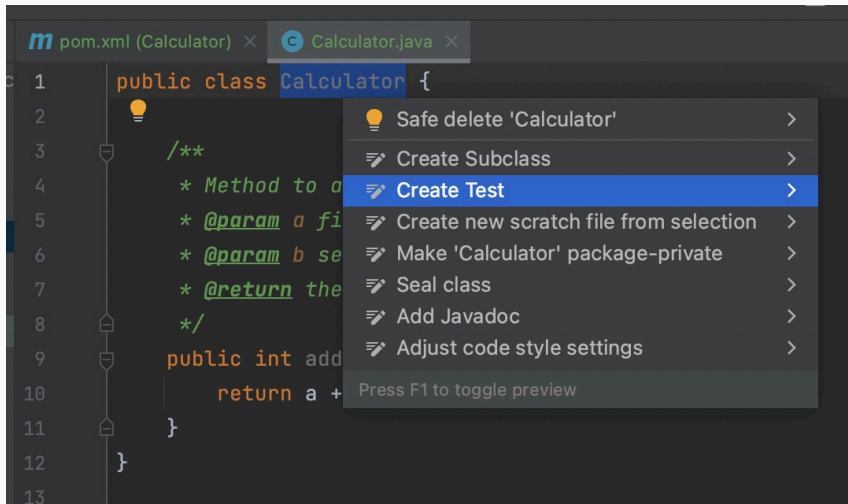


Figure 9: Create test 1/3

Creating an empty Unit test 1/3

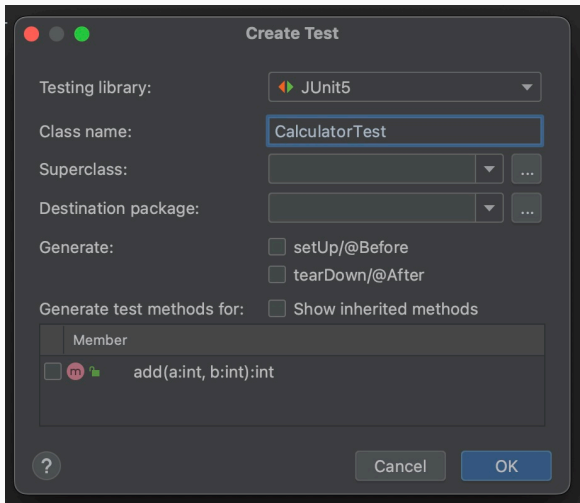
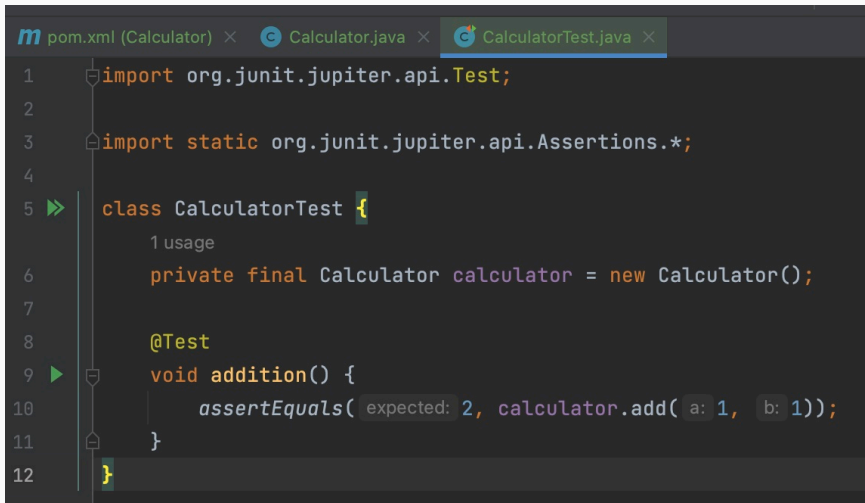


Figure 10: Create test 2/3

Creating an empty Unit test 1/3



The screenshot shows an IDE with three tabs: pom.xml (Calculator), Calculator.java, and CalculatorTest.java. The CalculatorTest.java tab is active, displaying the following code:

```
1  import org.junit.jupiter.api.Test;
2
3  import static org.junit.jupiter.api.Assertions.*;
4
5  class CalculatorTest {
6      1 usage
7      private final Calculator calculator = new Calculator();
8
9      @Test
10     void addition() {
11         assertEquals(expected: 2, calculator.add(a: 1, b: 1));
12     }
13 }
```

The code is written in Java and uses JUnit 5 annotations. The `addition()` method is annotated with `@Test` and uses `assertEquals` to verify the result of `calculator.add(1, 1)` is 2. The IDE interface includes a left margin with line numbers and a vertical scrollbar.

Figure 11: Create test 3/3

How to write a Unit test

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

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

public class CalculatorTest {
    private final Calculator calculator = new Calculator();

    @Test
    void addition() {
        assertEquals(2, calculator.add(1, 1), "This calculator can't add 1 and 1 together");
    }
}
```

Run a test 1/2

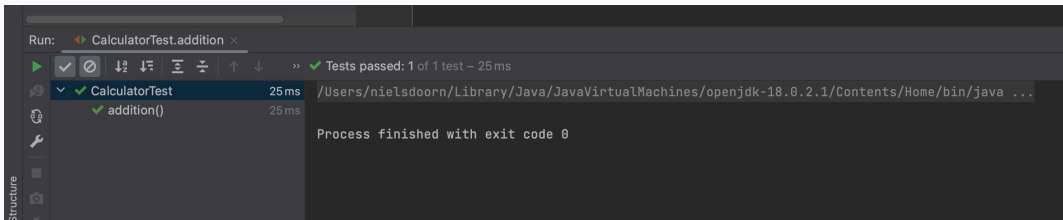


Figure 12: Running a test 1/2

Run a test 2/2

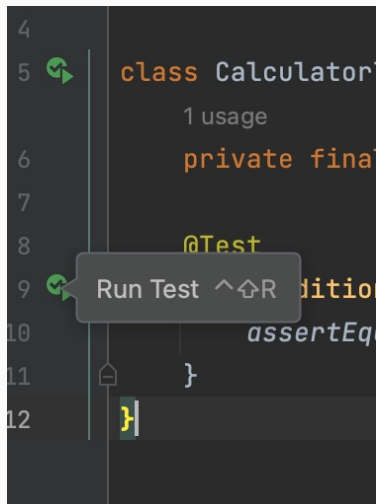


Figure 13: Running a test 2/2

Assertions

```
assertEquals(int expected, int actual, String message)
assertNotEquals(Object expected, Object actual, String message)

assertArrayEquals(int[] expected, int[] actual, String message)

assertNull(Object actual, String message)
assertNotNull(Object actual, String message)

assertSame(Object expected, Object actual, String message)
assertNotSame(Object expected, Object actual, String message)

assertTrue(boolean condition, String message)
assertFalse(boolean condition, String message)

assertEquals(double expected, double actual, double delta, String message)
```

More information

There is much more to learn about JUnit, the user guide can be found here:

junit.org/junit5/docs/current/user-guide/

Exercises

Idea behind the exercises

We have three exercises. Each of them is a bit harder to test. You need to create as much test cases as you think is necessary. At the end, you need to hand in your work via GitHub Classroom.

Money class

All exercises will be in the Money class and in the MoneyTest class

```
public class Money {  
}
```

```
public class MoneyTest {  
}
```

Method to test (easy)

```
/**  
 * Round the amount to nearest integer value  
 * @param amount  
 * @return rounded amount  
 */  
public long round(double amount) {  
    return Math.round(amount);  
}
```

Method to test (moderate)

```
/**
 * Returns the amount with the interest rate
 *
 * @param amount amount to add the interest to
 * @param interestRate the interest rate as a percentage
 * @return
 */
public double addInterest(double amount, double interestRate) {
    return amount + (amount * (interestRate / 100));
}
```

Method to test (difficult)

```
/**
 * Returns the smallest sequence of coins
 * to represent the input argument.
 * Possible coins:
 * 1, 2, 5, 10, 20 and 50 cent and
 * 1 and 2 euro (100 and 200 cent)
 */
public ArrayList<Integer> exchange(int amount) {
    ArrayList<Integer> result = new ArrayList<>();
    int[] coins = {200, 100, 50, 20, 10, 5, 2, 1};
    for (int coin : coins) {
        for (int i=0; i < amount / coin; i++) {
            result.add(coin);
        }
        amount = amount - (amount / coin) * coin;
    }
    return result;
}
```

No more excuses

No more excuses

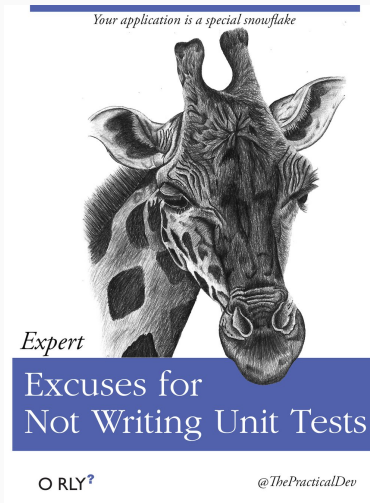


Figure 14: NO MORE EXCUSES