

JUnit, Mockito

von Florian Goldbach

All die folgenden Dateien sind auf meinem Github Repository „JUnitFeatureExploration“ zu finden. Hier der Link: <https://github.com/Florgol/JUnitFeatureExploration>

Ich habe für das Testing ein build.gradle file benutzt um „gradle test“ auszuführen. Ich hatte zuerst versucht mit vs code extensions zu arbeiten. Allerdings hatte ich dort Schwierigkeiten ein .classpath file zu verändern, dass ich nicht finden konnte. Letztendlich hatte ich Erfolg mit meinem build.gradle file, das ich hier nur zum testen benutze.

Aufgabe 1 + 2

1. Testen Sie Ihr eigenes Projekt / Ihren eigenen Code mit JUnit (so wie wir das mit dem Taschenrechner gemacht haben)
2. Testen Sie dabei ebenfalls auf eine Exception!

Ich begann diese Aufgabe indem ich eine Klasse schrieb mit Methoden, die Strings manipulieren. Diese Klasse heißt **StringManipulator.java**.

Link hier: <https://github.com/Florgol/JUnitFeatureExploration/blob/main/src/main/java/StringManipulator.java>

Anmerkung: Ich habe für diese Klasse die Dokumentation „javadoc-konform“ gehalten. Allerdings habe ich das für die weiteren Klassen nicht fortgesetzt.

Dann habe ich eine Testklasse geschrieben **StringManipulatorTest.java**. Hier teste ich die einzelnen Methoden auf erwarteten Output.

Link hier: <https://github.com/Florgol/JUnitFeatureExploration/blob/main/src/test/java/StringManipulatorTest.java>

Dort ist auch der Test auf eine Exception enthalten.

Hier hatte ich viele Tests, die nicht erfolgreich waren :(

```
>gradle test

> Task :test FAILED

StringManipulatorTest > testIsPalindrome() FAILED
    org.opentest4j.AssertionFailedError at StringManipulatorTest.java:51

StringManipulatorTest > testReverse() FAILED
    org.opentest4j.AssertionFailedError at StringManipulatorTest.java:34

StringManipulatorTest > testCountOccurrencesThrowsExceptionOnNullInput() FAILED
    org.opentest4j.AssertionFailedError at StringManipulatorTest.java:46
    Caused by: java.lang.NullPointerException at StringManipulatorTest.java:46

7 tests completed, 3 failed
```

Den Fehler in der isPalindrome() Methode hatte ich schnell gefunden.

Vorher (ich hatte vergessen, die „whitespaces“ vom „reversed String“ zu entfernen) :

```
public boolean isPalindrome(String str) {  
    String reversed = reverse(str);  
    return str.replace(" ", "").equalsIgnoreCase(reversed);  
}
```

Nachher:

```
public boolean isPalindrome(String str) {  
    String reversed = reverse(str);  
    reversed = reversed.replace(" ", "");  
    return str.replace(" ", "").equalsIgnoreCase(reversed);  
}
```

Die anderen 2 tests konnten auch erfolgreich ausgeführt werden, nachdem ich einen Tippfehler korrigierte und in testOccurrences() die Überprüfung auf „null“-input implementierte.

```
StringManipulatorTest > testToLowerCase() PASSED  
StringManipulatorTest > testCountOccurrences() PASSED  
StringManipulatorTest > testToUpperCase() PASSED  
StringManipulatorTest > testRemoveWhitespaces() PASSED  
StringManipulatorTest > testIsPalindrome() PASSED  
StringManipulatorTest > testReverse() PASSED  
StringManipulatorTest > testCountOccurrencesThrowsExceptionOnNullInput() PASSED
```

Aufgabe 3

Entwickeln oder finalisieren Sie im TDD-Modus die beiden Testklassen mit Testfunktionen für Zeichen to € und Bilder zählen. Falls Ihnen ein besseres / anderes Beispiel einfällt, um erst eine Klasse mit Methoden zu testen und dann dies zu implementieren, verwenden Sie dieses!

Ich habe, im Sinne von TDD (Test-Driven-Development), zuerst 2 Testklassen geschrieben. Die eine zählt die Anzahl der Bilder einer HTML-Seite: **ImageCounterTest.java**. Die andere berechnet die n-te Fibonacci-Zahl und kann feststellen, ob es sich bei angegebener Zahl um eine Fibonacci-Zahl handelt: **FibonacciCalculatorTest.java**.

Link zum Repository mit beiden Test-Klassen:

<https://github.com/Florgol/JUnitFeatureExploration/tree/main/src/test/java>

Daraufhin habe ich tatsächlichen Klassen, **ImageCounter.java** und **FibonacciCalculator.java** geschrieben.

Link zum Repository mit beiden Klassen:

<https://github.com/Florgol/JUnitFeatureExploration/tree/main/src/main/java>

Hier habe ich auch beide Klassen erfolgreich mit dem „gradle test“-Befehl testen können:

```
FibonacciCalculatorTest > testFibonacci() PASSED
FibonacciCalculatorTest > testIsInFibonacci() PASSED
ImageCounterTest > testCountImages() PASSED
```

Aufgabe 4

Schreiben Sie einen eigenen Anwendungsfall, bei dem Sie eine unangenehme Methode „herausmocken“! Wie in der Präsenz (Code).

Zuerst musste ich mockito bei den dependencies meines build.gradle file angeben:

```
dependencies {  
    testImplementation 'org.mockito:mockito-core:3.11.2'
```

Wie im Beispiel in der Präsenz habe ich 2 Klassen. In meinem Fall **ImportantClass.java** und **ExpensiveClass.java**. In der ImportantClass-Klasse gibt es eine importantMethod(), welche die expensiveMethod(), der ExpensiveClass-Klasse aufruft.

Wir sehen die expensiveMethod() für unseren Test als nicht wichtig (und zu teuer) an und versuchen gerade diese in unserer ImportantClassTest-Klasse „herauszumocken“.

ImportantClass.java und **ExpensiveClass.java** sind an dieser Stelle im Repository zu finden:

<https://github.com/Florgol/JUnitFeatureExploration/tree/main/src/main/java>

ImportantClassTest.java ist hier zu finden:

<https://github.com/Florgol/JUnitFeatureExploration/tree/main/src/test/java>

In unserer **ImportantClassTest.java**-Datei importieren wir Mockito:

```
import static org.mockito.Mockito.*;
```

.. und erstellen unser „Mock-Objekt“:

```
expensiveClass = mock(ExpensiveClass.class);
```

Als nächstes legen wir fest, welchen Rückgabewert die expensiveMethod() unseres „Mock-Objekts“ haben sollte:

```
when(expensiveClass.expensiveMethod()).thenReturn("Important data");
```

Wir definieren hier die gleiche Rückgabe, die wir von der echten expensiveMethod() erwarten würden. Wir könnten hier auch einen abweichenden Wert definieren, möglicherweise um später etwas zu unterscheiden.

Zum Abschluss können wir auch noch einmal verifizieren, ob tatsächlich die „Mock Methode“ der expensiveMethod() verwendet wurde:

```
verify(expensiveClass).expensiveMethod();
```

Ist das nicht der Fall, schlägt der Test fehl!

Aufgabe 5

Lassen Sie Test Coverage auf Ihre Tests los!



Ich habe hier in meinem build.gradle file das „jacoco“ plugin benutzt.

```
plugins {  
    id 'java'  
    id 'jacoco'  
}
```

Mit dem „gradle test jacocoTestReport“-Befehl wird ein Bericht in HTML und XML im folgenden Ordner im Projekt erzeugt: .. build/reports/jacoco/test

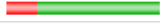
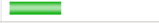




```
>gradle test jacocoTestReport
```

ESA10 - Testing







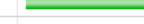




Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
default		89 %		88 %	3	23	5	40	1	14	0	3
Total	17 of 164	89 %	2 of 18	88 %	3	23	5	40	1	14	0	3

Dort kann man durch Anklicken der Packages, Klassen und Methoden zu den noch nicht abgedeckten Stellen im Code gelangen.

default

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
StringManipulator		79 %		100 %	1	11	4	15	1	9	0	1
FibonacciCalculator		96 %		83 %	2	9	1	17	0	3	0	1
ImageCounter		100 %		100 %	0	3	0	8	0	2	0	1
Total	17 of 164	89 %	2 of 18	88 %	3	23	5	40	1	14	0	3

StringManipulator

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
main(String[])		0 %		n/a	1	1	4	4	1	1
isPalindrome(String)		100 %		n/a	0	1	0	3	0	1
countOccurrences(String, char)		100 %		100 %	0	2	0	3	0	1
reverse(String)		100 %		n/a	0	1	0	1	0	1
lambda\$countOccurrences\$0(char, int)		100 %		100 %	0	2	0	1	0	1
removeWhitespaces(String)		100 %		n/a	0	1	0	1	0	1
StringManipulator()		100 %		n/a	0	1	0	1	0	1
toUpperCase(String)		100 %		n/a	0	1	0	1	0	1
toLowerCase(String)		100 %		n/a	0	1	0	1	0	1
Total	15 of 74	79 %	0 of 4	100 %	1	11	4	15	1	9

```
64.         return str.replace(" ", "").equalsIgnoreCase(reverse);  
65.     }  
66.  
67.     /**  
68.      * Removes whitespace characters from a given string.  
69.      *  
70.      * @param str the input string  
71.      * @return a new string with all whitespace characters removed  
72.      */  
73.  
74.     public String removeWhitespaces(String str) {  
75.         return str.replace(" ", "");  
76.     }  
77.  
78.     public static void main(String[] args){  
79.  
80.         StringManipulator sm = new StringManipulator();  
81.         System.out.println(sm.isPalindrome("radar"));  
82.         System.out.println(sm.isPalindrome("nur du Gudrun"));  
83.  
84.     }
```