

## Testen mit JUnit

### 1 Theoretische Grundbegriffe

Im Rahmen rund um das Thema Testen gibt es einige zentrale Grundbegriffe. Informieren Sie sich kurz stichpunktartig über folgende Begriffe (Beschreibung, Anwendungsbeispiele,...):

- Schreibtisch-Test
- Black-Box-Test
- White-Box-Test
- Testautomatisierung
- Unit Tests
- Integrationstests
- Systemtests
- Systemintegrationstests
- F.I.R.S.T-Prinzipien

In welcher Verbindung zu diesen Begriffen stehen **JUnit-Tests**?

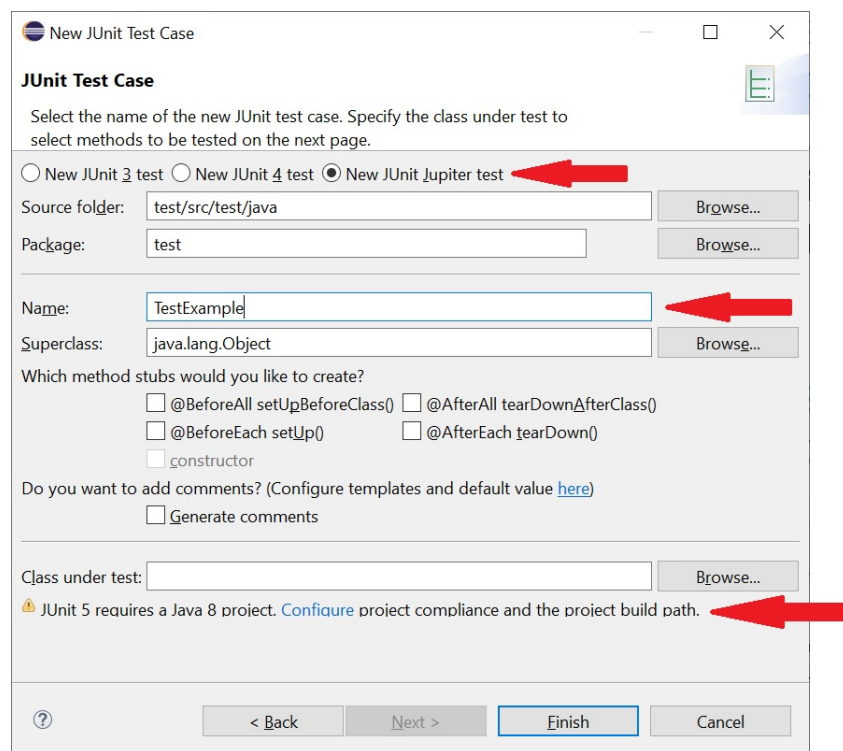
## 2 Erstellen von Tests mit JUnit

Im folgenden soll eine Methode getestet werden, welche zählt, wie oft ein bestimmtes Zeichen in einem String vorkommt:

```
public class ToTest {  
  
    public static int testMethod(String text, char toCount) {  
        int counter = 0;  
        if (text != null) {  
            char[] chars = text.toCharArray();  
            for (char c : chars) {  
                if (c == toCount) {  
                    counter++;  
                }  
            }  
        }  
        return counter;  
    }  
}
```

Eine erste entsprechende JUnit Test Case kann in Eclipse folgendermaßen erzeugt werden:

- Rechtsklick auf den Ordner/das Package, in dem die JUnit-Tests angelegt werden sollen. (in einem Maven-Projekt sind dafür die Ordner unter `src/test` vorgesehen)
- New → Other → JUnit Test Case auswählen
- Anschließend öffnet sich nachfolgendes Fenster, in dem Sie u.a. die JUnit-Version auswählen (JUnit Jupiter ist die aktuellste) und den Namen der Testklasse eingeben können. Achten Sie darauf, dass für JUnit Jupiter mindestens Java Version 8 für das Projekt eingestellt sein muss.



Nun kann die erstellte Klasse für einen JUnit Test folgendermaßen angepasst werden:

```
class TestExample {  
  
    private static final String testString = "Test-String";  
  
    @Test  
    @DisplayName("Korrekte Anzahl, falls das Zeichen häufiger im Text vorkommt")  
    void charMoreTimesInText() {  
        char toCount = 't';  
        int occurrence = 2;  
        int result = ToTest.testMethod(testString, toCount);  
        assertEquals(occurrence, result);  
    }  
}
```

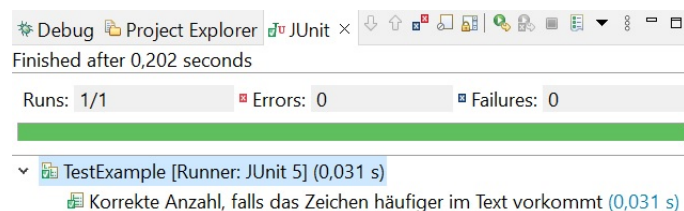
Alternativ kann in einem Maven-Projekt (ebenfalls mindestens Java-Version 8 notwendig) folgende Dependency in die pom-Datei eingetragen werden, um anschließend eine Klasse für JUnit-Tests zu erstellen:

```
<dependency>  
    <groupId>org.junit.jupiter</groupId>  
    <artifactId>junit-jupiter-engine</artifactId>  
    <version>5.9.2</version>  
    <scope>test</scope>  
</dependency>
```

Dieser Test kann nun in Eclipse folgendermaßen ausgeführt werden:

Rechtsklick auf die Testdatei → „Run as“ auswählen → „JUnit Test“ auswählen.

Nach der Ausführung öffnet sich folgender Tab, in dem das jeweilige Ergebnis aller Tests erscheint:



Im Rahmen von Testen mit JUnit werden einige Annotationen verwendet. Informieren Sie sich kurz stichpunktartig, wofür folgende Annotationen verwendet werden.

- Test
- DisplayName
- BeforeEach
- AfterEach
- BeforeAll
- AfterAll
- TestMethodOrderer

### 3 Wie viele Testfälle benötigt man - unterschiedliche Arten von Testabdeckung

#### 3.1 Zeilenüberdeckung

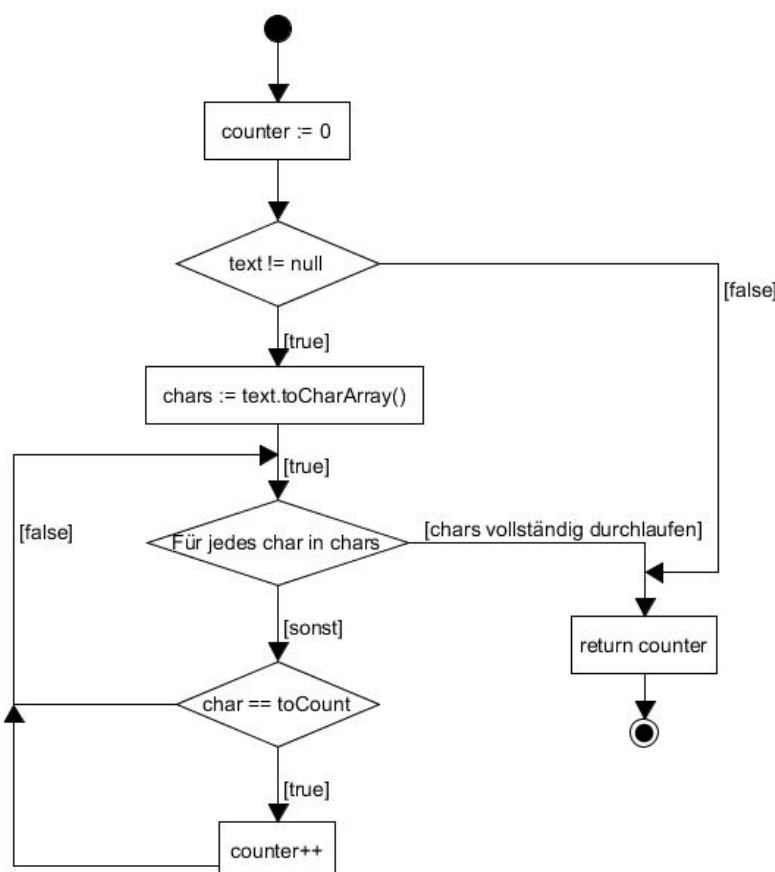
Bei dieser Überdeckung werden keine Anweisungen betrachtet, sondern nur die Anzahl durch die Testfälle ausgeführter Zeilen. Wurden alle Zeilen des zu testenden Codes mindestens einmal ausgeführt, spricht man von **vollständiger Zeilenüberdeckung**. Diese Abdeckung spielt in der Praxis aber kaum eine Rolle, da sich die Abdeckung nur bei einer anderen Formatierung des zu testenden Codes schon verändern kann.

#### 3.2 Anweisungsüberdeckung (Statement Coverage)

Bei dieser Überdeckung werden die ausgeführten Anweisungen gezählt. Der Grad der Überdeckung wird dabei wie folgt berechnet:

$$\frac{\text{Anzahl der durchlaufenen Anweisungen}}{\text{Gesamtanzahl der Anweisungen}} \cdot 100\%$$

Wird jede Anweisung mindestens einmal durchlaufen, spricht man von **vollständiger Anweisungsüberdeckung**. Für diese und den folgenden Abdeckungen ist es hilfreich, einen so genannten **Kontrollflussgraphen** des Programms zu zeichnen:



Bei der vollständigen Anweisungsüberdeckung muss also bei dem Kontrollflussgraphen jeder Knoten (Rechteck bzw. Raute) mindestens einmal durchlaufen werden.

Durch diese Testabdeckung können allgemein nicht ausführbare Anweisungen im Code ermittelt werden. Die Anweisungsüberdeckung reicht alleine jedoch in der Praxis häufig nicht aus und wird mit strengeren Überdeckungsarten kombiniert.

**Aufgabe:** Wird durch den obigen JUnit Test `charMoreTimesInText` vollständige Anweisungsüberdeckung erreicht? Begründen Sie Ihre Antwort. Falls nicht, ergänzen Sie entsprechende Tests, so dass vollständige Anweisungsüberdeckung erreicht wird.

### 3.3 Zweigüberdeckung (Branch Coverage)

Diese Überdeckung umfasst die Anweisungsüberdeckung, jedoch werden ,anstelle von ausgeführten Anweisungen, die Anzahl ausgeführter Zweige gezählt und berechnet sich wie folgt:

$$\frac{\text{Anzahl der ausgeführten Zweige}}{\text{Gesamtanzahl aller Zweige}} \cdot 100\%$$

Wird jeder Zweig mindestens einmal durchlaufen, so spricht man von **vollständiger Zweigüberdeckung**. Im Kontrollflussgraph müssen also für eine vollständige Zweigüberdeckung alle Kanten (Pfeile) mindestens einmal durchlaufen werden.

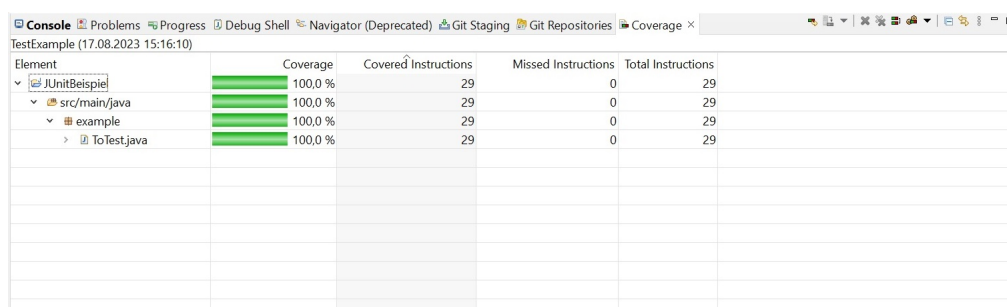
**Aufgabe:** Wird durch den obigen JUnit Test `charMoreTimesInText` vollständige Zweigüberdeckung erreicht? Begründen Sie Ihre Antwort. Falls nicht, ergänzen Sie entsprechende Tests, so dass vollständige Zweigüberdeckung erreicht wird.

### 3.4 Ermittlung der Testabdeckung in Eclipse

In aktuellen Eclipse IDEs ist das Testabdeckungswerkzeug EclEmma schon integriert. Dieses Werkzeug kann die Testabdeckung sehr einfach ermitteln:

Rechtsklick auf die Testdatei → „Coverage as“ → „JUnit Test“ auswählen

Danach öffnet sich der Coverage-Tab, in dem die Testabdeckung für das gesamte Projekt bzw. jeder einzelnen Klasse angezeigt wird:



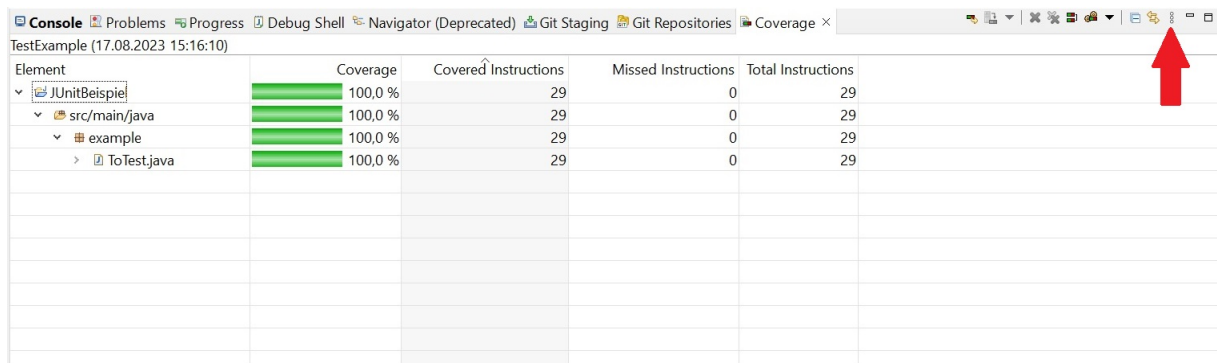
Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
JUnitBeispiel	100.0 %	29	0	29
src/main/java	100.0 %	29	0	29
example	100.0 %	29	0	29
ToTest.java	100.0 %	29	0	29

Da sich die JUnit Tests ebenfalls im Projektordner befinden, diese jedoch häufig für die Testabdeckung eines Projekts nicht miteingerechnet werden sollen, kann der Ordner mit den JUnit Tests von der Berechnung wie folgt herausgenommen werden:

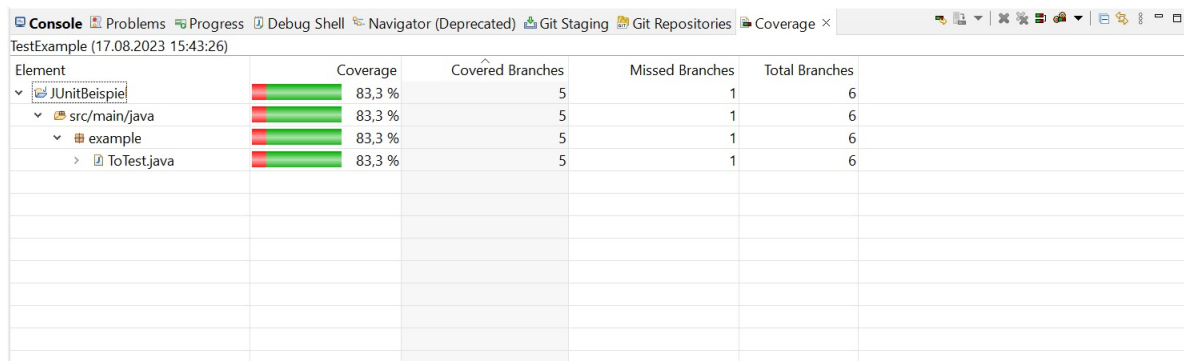
Rechtsklick auf die Testdatei → „Coverage as“ → „Coverage Configurations“ auswählen

Anschließend kann im Tab „Coverage“ der Haken beim Ordner mit den JUnit Tests entfernt werden.

Man kann auch anstelle der Anweisungsüberdeckung die Zweigüberdeckung auswählen, dies geht über den Punkt „View Menu“ (siehe Pfeil) und der anschließenden Auswahl von „Branch Counters“:



Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
JUnitBeispiel	100,0 %	29	0	29
src/main/java	100,0 %	29	0	29
example	100,0 %	29	0	29
ToTest.java	100,0 %	29	0	29



Element	Coverage	Covered Branches	Missed Branches	Total Branches
JUnitBeispiel	83,3 %	5	1	6
src/main/java	83,3 %	5	1	6
example	83,3 %	5	1	6
ToTest.java	83,3 %	5	1	6

Dort erkennt man, dass der vorhin festgestellte fehlende Test für eine vollständige Zweigüberdeckung noch nicht implementiert wurde, dies bestätigt auch der Blick in die Klasse ToTest:

```

5 public static int testMethod(String text, char toCount) {
6     int counter = 0;
7     if (text != null) {
8         char[] chars = text.toCharArray();
9         for (char c : chars) {
10             if (c == toCount) {
11                 counter++;
12             }
13         }
14     }
15     return counter;
16 }

```

## 3.5 Für Experten: Weitere Überdeckungsarten

Mit den bisherigen Überdeckungen werden Schleifen und zusammengesetzte komplexere Bedingungen nur unzureichend getestet, sowie Abhängigkeiten zwischen Zweigen nicht berücksichtigt. Hierfür gibt es die folgenden Überdeckungsarten.

### 3.5.1 Pfadüberdeckung

Bei einer vollständigen Pfadüberdeckung sollen alle möglichen Pfade von Start bis zu Ende getestet werden. Bei Schleifen ist dies jedoch kaum zu erreichen, vor allem ist die Pfadanzahl bei unbestimmten Schleifen möglicherweise unbeschränkt.

Daher gibt es beispielsweise die abgeschwächte Form der **Boundary Interior Tests**, die im Gegensatz zu allgemeinen Pfadüberdeckung praktisch anwendbar sein soll. Hierbei werden zwei Gruppen von Tests gebildet:

- Tests, die die Schleife kein oder nur einmal betreten und alle möglichen Pfade innerhalb der Schleife testen. (Boundary Tests)
- Tests, die die Schleife mindestens zweimal durchlaufen und dabei alle möglichen Pfade testen. (Interior Tests)

### 3.5.2 Bedingungsüberdeckung

Hierbei sollen komplexere zusammengesetzte Bedingungen genauer getestet werden. Dabei gibt es drei verschiedene Varianten:

- Einfache Bedingungsüberdeckung:  
Jede einzelne atomare Bedingung muss einmal mit `true` und einmal mit `false` getestet werden.
- Mehrfachbedingungsüberdeckung:  
Jede Kombination an Wahrheitswerten für jede atomare Bedingung soll getestet werden. (Bei  $n$  Bedingungen  $2^n$  Kombinationen)
- Minimale Mehrfach-Bedingungsüberdeckung:  
Jede einzelne atomare und zusammengestellte Bedingung muss einmal mit `true` und einmal mit `false` getestet werden.