Softwaretechnik – Übung

Qualitätssicherung: Exkurs JUnit 5

Prof. Dr. Bodo Kraft

FH AACHEN
UNIVERSITY OF APPLIED SCIENCES

Einführung JUnit
Werkzeuge zur Testautomatisierung

JUnit

- ist ein Open-Source Test-Framework (http://junit.org)
- ermöglicht wiederholt ausführbare Unit-Tests (Regressionstests)
- **Fokus:** Modultests
- baut auf xUnit-Architektur auf
- Ursprünglich geschrieben von: Erich Gamma, Kent Beck

Kent Beck

JUnit

Erich Gamma

Basis-Syntax JUnit
Werkzeuge zur Testautomatisierung

Testmethoden

- Seit JUnit 4.X über Annotation definiert (**@Test**)
- Signatur:
  - Sichtbarkeit **nicht private**
  - **Nicht static**
  - Rückgabetyp **void**
  - Methodenname frei wählbar
  - Keine* Parameter

Implementierung Testmethode (AAA):

1) **A**rrange: Test-Fixture festlegen
2) **A**ct: Aufruf zu testender Methode
3) **A**ssert: Abgleich Soll/Ist-Zustand

```
public class Simple {

  public int sum(int a, int b){
    return a + b;
  }
}
```

```
public class SimpleTest {

  @Test
  public void testSum(){

1)  int a = 4; int b=5;
    Simple simple = new Simple();
    int expected = 9;

2)  int actual = simple.sum(a, b);

3)  assertEquals(expected, actual,
     „Calculation of the sum of
     two integers did not work.“);
  }
}
```

@BeforeEach und @AfterEach
Test-Fixture

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

public class SampleTest {
 private java.util.List emptyList;

 /**
  * Sets up the test fixture.
  * (Called before every test case method.)
  */
  @BeforeEach public void setUp() {
    emptyList = new java.util.ArrayList();
  }

  /**
   * Tears down the test fixture.
   * (Called after every test case method.)
   */
   @AfterEach public void tearDown() {
     emptyList = null;
   }

   @Test public void testSomeBehavior() {
     assertEquals(0, emptyList.size(),
       "Empty list should have 0 elements");
   }
}
```

Auf Methodenebene:

@BeforeEach

Gemeinsame Voraussetzungen fest-legen für mehrere Test-Methoden

@AfterEach

Gemeinsame Aufräum-Prozedur festlegen für mehrere Test-Methoden

Auf Klassenebene:

@BeforeAll

Die Methode wird genau einmal ausgeführt beim Laden der Testklasse (vor allen anderen Methoden der Klasse)

@AfterAll

Die Methode wird genau einmal ausgeführt bei der Speicher-bereinigung der Klasse. (nach allen anderen Methoden der Klasse)

Übersicht der verantwortlichen Klassen
Abgleich Soll/Ist-Zustand

Das Testergebnis wird geprüft:

- **Falls Standardfall erwartet:**
  - über die Klasse org.junit.jupiter.api.Assertions
  - Enthält viele statische Prüf-Methoden

- **Falls Exception erwartet:**
  - über assertThrows – Methode in org.junit.jupiter.api.Assertions
  - Assert, wenn keine oder die falsche Exception auftritt
  - Wenn die Exception auftritt, wird diese zurückgegeben, damit sie ggf. detailliert geprüft werden kann

Methoden der Assert-Klasse
Abgleich Soll/Ist-Zustand

```
import static org.junit.jupiter.api.Assertions.*;
[…]

@Test public void assertions() {

  assertTrue(true, "failure - should be true");
  assertFalse(false, "failure - should be false");

  assertNull(null, "should be null");
  assertNotNull(new Object(), "should not be null");

  byte[] expected = "trial".getBytes();
  byte[] actual = "trial".getBytes();
  assertArrayEquals(expected, actual, "failure - byte arrays not same");
  assertEquals("text", "text", "failure - strings are not equal");

  Integer aNumber = Integer.valueOf(768);
  assertSame(aNumber, aNumber, "should be same");
  assertNotSame(new Object(), new Object(), "should not be same Object");

}
```

Erwartete Ausnahmen
Abgleich Soll/Ist-Zustand

- **assertThrows** bekommt als Argument die Klasse der erwarteten Exception und eine Methode, die die Exception auslösen soll.
- **assertThrows** schlägt fehl:
  - wenn keine Exception geworfen wird
  - wenn der Typ der geworfenen Exception falsch ist (nicht **instanceOf** „expected“)

```
[…]
private java.util.List emptyList = new java.util.ArrayList();

[…]

@Test
public void testForException() {
  assertThrows(IndexOutOfBoundsException.class, () -> {
      emptyList.get(0);
  }
}

[…]
```

Erwartete Ausnahmen
Abgleich Soll/Ist-Zustand

Der Inhalt einer Exception kann wie folgt geprüft werden:

```
@Test
public void shouldTestExceptionMessage() throws IndexOutOfBoundsException {
    List<Object> list = new ArrayList<Object>();

    IndexOutOfBoundsException ex = assertThrows(IndexOutOfBoundsException.class,
        () -> list.get(0)
    );

    assertEquals("Index: 0, Size: 0", ex.getMessage());
}
```

Wenn die richtige Exception aufgetreten ist, gibt **assertThrows** diese als Ergebnis zurück.

Allgemeine Richtlinien
JUnit Best Practices

- Nutze denselben Standard für alle Tests
- Test Code wird für Lesbarkeit optimiert, nicht Performance! (lange Methodennamen hier OK)
- Teste immer nur eine Sache pro Test
- Verwende mindestens eine Testklasse pro zu testender Klasse
  - Beginne mit 1:1 Beziehung
  - Aufsplitten um zum Bsp.: seperation of concern zu erreichen
- Test-Klassennamen(Fixture) enden mit Suffix „Test“
  - Bsp.: Die Klasse **PersonTest** beinhaltet alle Tests für Objekte der Klasse **Person**.

Testmethoden benennen
JUnit Best Practices

- Methodenname sollte wie konkrete Anforderungs-Spezifikation lesbar sein (ohne Codebetrachtung)
- Mache deutlich, was jeder Testzustand ist
- Sei spezifisch bezüglich des erwarteten Verhaltens
- Vermeide Prefix „test“ wenn möglich (Relikt aus JUnit3)

Testmethoden benennen
JUnit Best Practices

Testmethodennamen werden aufgebaut nach dem Schema:

```
<methodUnderTest>_<scenario>_<expectedBehaviour>
```

als Orientierung:

„When I call method X with a Y value, then it should do Z“

- **When-Teil:** ermöglicht alphabetische Sortierung + vereinfacht Suche
- **With-Teil:** beschreibt konkrete Parameter für den Testaufruf
- **Then-Teil:** das erwartete Ergebnis nach Aufruf

Beispiele:

```
@Test public void sum_WithNegativeNumberAs1stParam_shouldThrowException()
@Test public void sum_WithNumberBiggerThan1000_shouldIgnoreNumber()
@Test public void
      analyzeFile_FileWith3LinesAndFileProvider_ReadsFileUsingProvider()
```

[Quelle: „The Art of Unit Testing, 2nd Edition, Seite 181 ff“]

Vielen Dank für die Aufmerksamkeit!

Fragen?