Testen mit JUnit (JUnit-Basics)

Carsten Gips (HSBI)

JUnit: Ergebnis prüfen

Klasse `org.junit.Assert` enthält diverse **statische** Methoden zum Prüfen:

```
// Argument muss true bzw. false sein
void assertTrue(boolean);
void assertFalse(boolean);

// Gleichheit im Sinne von equals()
void assertEquals(Object, Object);

// Test sofort fehlschlagen lassen
void fail();

...
```

To “assert” or to “assume”?

- Mit `assert*` werden Testergebnisse geprüft
  - Test wird ausgeführt
  - Ergebnis: OK, Failure, Error

- Mit `assume*` werden Annahmen über den Zustand geprüft
  - Test wird abgebrochen, wenn Annahme nicht erfüllt
  - Prüfen von Vorbedingungen: Ist der Test hier ausführbar/anwendbar?

Beispiel: junit4.TestAssume

Setup und Teardown: Testübergreifende Konfiguration

```java
private Studi x;

@Before
public void setUp() { x = new Studi(); }

@Test
public void testToString() {
    // Studi x = new Studi();
    assertEquals(x.toString(), "Heinz (15cps)");
}
```

`@Before` wird **vor jeder** Testmethode aufgerufen

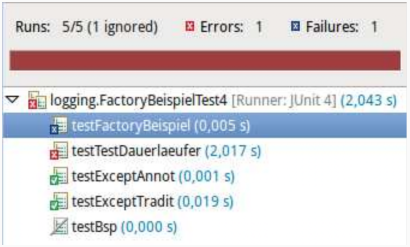
`@BeforeClass` wird **einmalig** vor allen Tests aufgerufen (`static`!)

`@After` wird **nach jeder** Testmethode aufgerufen

`@AfterClass` wird **einmalig** nach allen Tests aufgerufen (`static`!)

Vermeidung von Endlosschleifen: Timeout

```
@Test(timeout = 2000)
void testTestDauerlaeufer() {
    while (true) { ; }
}
```



Test von Exceptions: Expected

```java
@Test
public void testExceptTradit() {
    try {
        int i = 0 / 0;
        fail("keine ArithmeticException ausgeloest");
    } catch (ArithmeticException aex) {
        assertNotNull(aex.getMessage());
    } catch (Exception e) {
        fail("falsche Exception geworfen");
    }
}
```

Test von Exceptions: Expected

```java
@Test
public void testExceptTradit() {
    try {
        int i = 0 / 0;
        fail("keine ArithmeticException ausgeloest");
    } catch (ArithmeticException aex) {
        assertNotNull(aex.getMessage());
    } catch (Exception e) {
        fail("falsche Exception geworfen");
    }
}
```

```java
@Test(expected = java.lang.ArithmeticException.class)
public void testExceptAnnot() {
    int i = 0 / 0;
}
```

Parametrisierte Tests

```
class Sum {
    public static int sum(int i, int j) {
        return i + j;
    }
}

class SumTest {
    @Test
    public void testSum() {
        Sum s = new Sum();
        assertEquals(s.sum(1, 1), 2);
    }
    // und mit (2,2, 4), (2,2, 5), ...????
}
```

Parametrisierte Tests: Konstruktor (JUnit 4)

```java
@RunWith(Parameterized.class)
public class SumTestConstructor {
    private final int s1;
    private final int s2;
    private final int erg;

    public SumTestConstructor(int p1, int p2, int p3) { s1 = p1;  s2 = p2;  erg = p3; }

    @Parameters
    public static Collection<Object[]> values() {
        return Arrays.asList(new Object[][] { { 1, 1, 2 }, { 2, 2, 4 }, { 2, 2, 5 } });
    }

    @Test
    public void testSum() {
        assertEquals(Sum.sum(s1, s2), erg);
    }
}
```

Parametrisierte Tests: Parameter (JUnit 4)

```java
@RunWith(Parameterized.class)
public class SumTestParameters {

    @Parameter(0)  public int s1;
    @Parameter(1)  public int s2;
    @Parameter(2)  public int erg;

    @Parameters
    public static Collection<Object[]> values() {
        return Arrays.asList(new Object[][] { { 1, 1, 2 }, { 2, 2, 4 }, { 2, 2, 5 } });
    }

    @Test
    public void testSum() {
        assertEquals(Sum.sum(s1, s2), erg);
    }
}
```

Beispiel: junit4.SumTestConstructor, junit4.SumTestParameters

Testsuiten: Tests gemeinsam ausführen (JUnit 4)

```java
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({
    // Hier kommen alle Testklassen rein
    PersonTest.class,
    StudiTest.class
})

public class MyTestSuite {
    // bleibt leer!!!
}
```

Wrap-Up

JUnit als Framework für (Unit-) Tests; hier JUnit 4 (mit Ausblick auf JUnit 5)

- Testmethoden mit Annotation `@Test`
- `assert` (Testergebnis) vs. `assume` (Testvorbedingung)
- Aufbau der Testumgebung `@Before`
- Abbau der Testumgebung `@After`
- Steuern von Tests mit `@Ignore` oder `@Test(timout=XXX)`
- Exceptions einfordern mit `@Test(expected=package.Exception.class)`
- Tests zusammenfassen zu Testsuiten

LICENSE