

## JUnit Intro

Patrick Gleichmann ([patrick@feedface.com](mailto:patrick@feedface.com)), Letzte Änderung: 11.12.2003

### 1. Test und TestCase

Folgende Klasse soll getestet werden:

```
1 package com.feedface.junitdemo;
2
3 public class MyNumber {
4     int value;
5
6     public MyNumber(int value)        { this.value = value; }
7     public int getValue()             { return value; }
8     public boolean equals(int value)  { return getValue() == value; }
9 }
```

Die dazugehörige Testklasse:

```
1 package com.feedface.junitdemo;
2
3 import junit.framework.*;
4
5 public class MyNumberTest extends TestCase {
6     private MyNumber number;
7
8     public void setUp() {
9         number = new MyNumber(23);
10    }
11    public void tearDown() {
12        // Clean up code - z.B. Datei schließen
13    }
14
15    public void testEquals() {
16        Assert.assertTrue(number.equals(24));
17    }
18
19    public static void main(String[] args) {
20        junit.textui.TestRunner.run(MyNumberTest.class);
21    }
22 }
```

Code Besprechung:

- 
- ① Der Import **junit.framework.\*** ist, wie auch die Datei "junit.jar", immer nötig.
  - ② Der **TestCase** ist der Container für die einzelnen Tests.
  - ③ Falls mehrere Tests (siehe ④) die gleiche Vorbereitung, z.B. Initialisierung von Variablen, benötigen, kann man diese in die optionalen „Fixture“ Routinen packen:
- 

**setUp**      Ausführung vor dem Test  
**tearDown**    Ausführung nach dem Test

Damit es zu keinen Nebenwirkungen kommt, werden die Routinen für jeden Test aufgerufen.

---

- 
- ④ Der Name einer Testmethode beginnt immer mit **test**.  
Das eigentliche Testen wird immer mit den statischen Methoden der Klasse Assert gemacht – hier z.B. **Assert.assertTrue**, das ein zurückgeliefertes False als einen Fehler werten würde.  
Eine Auflistung der bekannten Asserts folgt weiter unten.
- 
- ⑤ Der **TestRunner** sorgt für die Ausführung der einzelnen Tests und die Ausgabe, u.a. der Fehler und des Testfortschritts. Falls keine Suite (siehe 2.) existiert, so werden alle Tests, d.h. alle Methoden die mit „test“ beginnen, durchgeführt.  
Folgende TestRunner existieren:

|                          |   |
|--------------------------|---|
| junit.textui.TestRunner  | Textmodus – praktisch für automatisierte Tests. |
| junit.swingui.TestRunner | Swing GUI                                       |
| junit.awt.ui.TestRunner  | AWT GUI   |

Die Alternative zu der Angabe des TestRunners in main(), ist die schlichte Angabe des gewünschten TestRunner auf der Kommandozeile, also für die Swing Version hier:

---

```
java junit.swingui.TestRunner MyNumberTest
```

---

## 2. Suite

```
1 package com.feedface.junitdemo;
2
3 import junit.framework.*;
4
5 public class SuiteDemo extends TestCase {
6     public SuiteDemo(String name) { super(name); }
7
8     public void testOne { /* Code */ }
9     public void testTwo { /* Code */ }
10
11     public static Test suite() {
12         TestSuite suite = new TestSuite();
13         suite.addTest(new SuiteDemo("testOne"));
14         suite.addTest(
15             new SuiteDemo("two") {
16                 public void runTest() { testTwo(); }
17             }
18         );
19         suite.addTestSuite(MyNumberTest.class);
20         return suite;
21     }
22 }
```

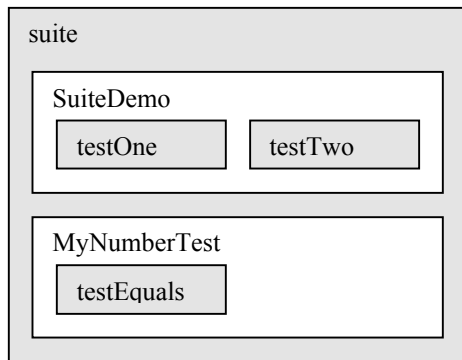
① { ② { ③ { ④ {

Besprechung der Neuerungen gegenüber dem vorherigen TestCase:

- 
- ① Dank einer **suite** ist es möglich nur bestimmte Tests ausführen zu lassen, und auch die Reihenfolge festzulegen.
- 
- ② Mittels **addTest** fügt man Tests hinzu, wobei es sich hier um die dynamische Variante handelt.  
Der Nachteil liegt darin, das JUnit Schreibfehler nicht erkennt und der Test dann schlicht nicht ausgeführt wird.
-

- ③ Ähnlich ②, jedoch die statische Variante. Es wird die Instanz einer Klasse erwartet, die das Interface Test implementiert.
- ④ Um andere Suites ebenfalls ausführen zu lassen, bedient man sich **addTestSuite**.

Das ganze als „Schalenmodell“:



### 3. Die statischen Methoden der Klasse Assert

| Name                          | Anz.Arg. <sup>*1</sup> | T: Erlaubte Argumenttypen<br>W: Erwarteter Wert bzw. Zustand, sonst Fehler |
|-------------------------------|------------------------|--|
| assertEquals                  | 2                      | T: Alle Basistypen (byte, boolean, ...)<br>W: Sind gleich                  |
| assertTrue /<br>assertFalse   | 1                      | T: boolean<br>W: Wert ist true / false                                     |
| assertNull /<br>assertNotNull | 1                      | T: Object<br>W: Object ist null / not null                                 |
| assertSame /<br>assertNotSame | 2                      | T: Object<br>W: Gleiches Objekt (= Adresse) / nicht gleich                 |
| fail                          | 0                      | W: Erzwungener Fehler  |

\*1: Optional kann den nötigen Argumenten noch eine Fehlerbeschreibung (Typ String) vorangestellt werden, also z.B.:

```
assertEquals(String fehlernachricht, byte wert1, wert2)
```

### 4. Nützliches und Wissenswertes

Behandlung von Exceptions:

```
try {
    // Aufruf einer Methode, die eine Exception wirft
} catch (Exception e) {
    Assert.fail("Exception geworfen");
}
```

Quellcodehierarchie:

```
src/
├── com/feedface/junitdemo/
└── tests/ — com/feedface/junitdemo/
```

## 5. Ant

```
1 <project name="JUnit Demo" default="runtests" basedir=".>
2   <property name="src.dir"      value="src" />
3   <property name="build.dir"    value="build" />
4   <property name="testlog.dir"  value="testlog" />
① — 5   <property name="junit.jar"  value="/J2EE/junit/junit.jar" />
6
7   <path id="classpath">
8     <pathelement path="{build.dir}" />
② — 9     <pathelement path="{junit.jar}" />
10  </path>
11
12  <target name="compile">
13    <mkdir dir="{build.dir}" />
14    <javac srcdir="{src.dir}/com" destdir="{build.dir}"
15      classpathref="classpath" />
16  </target>
17  <target name="compiletests" depends="compile">
③ { 18    <javac srcdir="{src.dir}/test" destdir="{build.dir}"
19      classpathref="classpath" />
20  </target>
21  <target name="runtests" depends="compiletests">
④ { 22    <mkdir dir="{testlog.dir}" />
23    <junit haltonerror="true" haltonfailure="true"
24      printsummary="true">
25      <classpath refid="classpath" />
26      <test name="com.feedface.junitdemo.SuiteDemo"
27        todir="{testlog.dir}" />
28      <formatter type="plain" />
29    </junit>
30  </target>
31 </project>
```

Kurze Erläuterung der zusätzlich, zu den Standardeinträgen, hinzugekommenen Zeilen:

- 
- ① Definition des JUnit JAR Pfades.

---

  - ② Hinzufügen von JUnit zum Classpath.

---

  - ③ Der „compile“ Task wird um die Tests erweitert.

---

  - ④ Ant bringt standardmäßig die **junit** Task Erweiterung mit.  
In diesem Fall wird die SuiteDemo als einziger **test** festgelegt und die Testausgabe in das „testlog.dir“ geschrieben, wobei das Resultat, wegen **formatter**, eine Textdatei ist.
-