Test mit JUnit und TestObject schreiben.

Joshua Gleitze

14. Januar 2015

Um Klassen zu testen, eignet sich die Bibliothek JUnit gut. Diese bietet viele nützliche Funktionen zum Testen von Klassen. Um mit unbekannten Klassen und (evtl. unvollständigen) Implementierungen umgehen zu können, habe ich zusätzlich die Klasse TestObject geschrieben. Wie man gute Tests schreiben kann, möchte ich hier kurz erläutern.

Namen in diesem Dokument

In diesem Dokument verwende ich verschiedene Namen, die ich hier kurz erläutern möchte.

getestete Klasse Die Klasse, die von einem Test getestet werden soll.

Test, Testklasse Der JUnit-Test, der die getestete Klasse testet.

Implementierte Klasse Die getestete Klasse, wie sie sich bei demjenigen, der den Test ausführt, befindet. Über diese wissen wir *nichts*.

JUnit Eine Bibliothek um Klassen zu testen, siehe junit.org.

JUnit-Methoden Die hier vorgestellten Methoden kommen alle aus dem Paket `org.junit.*`, die meisten aus `org.junit.Assert`.

Inhaltsverzeichnis

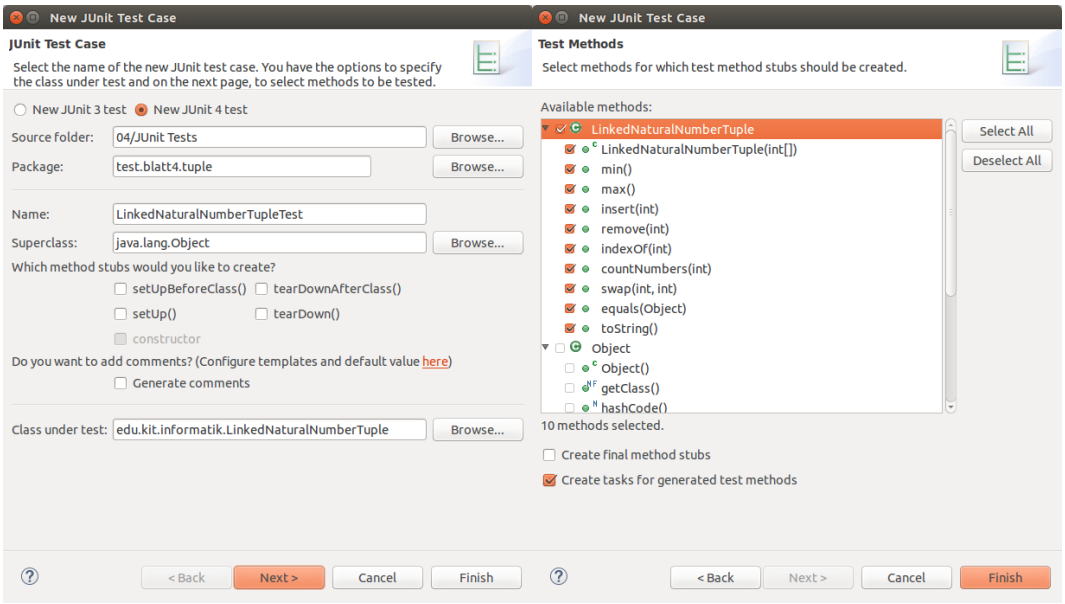

Abbildung 1: Einstellungen für den Test

1 JUnit-Test in Eclipse erstellen

Eine Testklasse lässt sich gut in dem Projekt entwickeln, in dem auch die getestete Klasse geschrieben wird. Hierzu muss zuerst der JUnit-Tests-Ordner zum Projekt hinzugefügt werden, wie es in der Anleitung beschrieben ist. Ein neuer Test lässt sich über Rechtsklick -> New -> JUnit Test Case erstellen. Existiert die getestete Klasse bereits, so empfiehlt es sich, den Rechtsklick auf diese Klasse auszuführen. Dadurch wird Eclipse den Test automatisch benennen und ihm für jede Methode in der getesteten Klasse eine Test-Methode hinzufügen. Die Optionen im Fenster „New JUnit Test Case“ sollte man wie folgend wählen:

Source Folder `Projektordner`/JUnitTests

Package tests.blatt`x`.`name` (`x`: Nummer des Blatts, `name`: Griffiger Name für die Aufgabe)

Name `GetesteteKlasse`Test

Class under test ggf. die getestete Klasse, um aus ihr Methoden zu übernehmen.

2 Aufbau eines Tests

Ein Test sollte für jede Methode der getesteten Klasse, die er testen will, eine eigene Methode besitzen. Wenn `methodName` der Name einer zu testeten Me-

thode in der getesteten Klasse ist, sollte der Test für sie eine Methode namens testMethodName enthalten. Eine Test-Methode ist parameterlos, vom Typ **void** wird mit der Annotation `@Test` eingeleitet. Eine Testklasse sieht beispielsweise so aus:

```
import static org.junit.Assert.*;

public class FooTest {
    @Test
    public void testMethod1() {
        // do the testing here
    }

    @Test
    public void testMethod2() {
        // do the testing here
    }

    // more test methods
}
```

> Test-Methoden werden in (praktisch) zufälliger Reihenfolge aufgerufen. Daher müssen sie vollkommen unabhängig voneinander sein!

3 TestObject

Die Klasse `test.TestObject` ist ein Stellvertreter für die implementierte Klasse. Sie hält einen Verweis auf die implementierte Klasse. Sie lässt sich (fast) wie die getestete Klasse instanziieren und man kann die Methoden der getesteten Klasse mit `run()` und *`runStatic()`* aufrufen. Letztere führen eine (statische) Methode aus, deren Name als `String` übergeben wird. Die Konstruktoren und die run-Methoden können entweder parameterlos oder mit einem Parameter aufgerufen werden. Will man sie mit mehreren Parametern aufrufen, fasst man die Parameter in einem Array vom Typ `Object[]` zusammen (Die run-Methoden brauchen als ersten Parameter natürlich immer den Namen der Methode):

```
TestObject testObject = new TestObject(); // call the standard
    constructor of the implemented class
Object result1 = testObject.run("foo", 4); // call foo(4) on the
    instance of the implemented class
Object result2 = TestObject.runStatic("bar", new Object[]{3, 9, "bar"})
    ; // call static bar(3, 9 , "b") on the implemented class
```

TestObject gibt ausführliche Fehlermeldungen mit *`fail()`* aus, wenn bei der Ausführung eines Konstruktors oder einer Methode ein Fehler auftritt. Das kann beispielsweise sein, dass keine passende Methode in der implementierten Klasse gefunden wurde, dass zwar eine Methode gefunden wurde sie aber die falsche Sichtbarkeit hast, oder dass eine Exception bei der Ausführung aufgetreten ist.

Die weiteren Möglichkeiten und Methoden von TestObject sind in der ausführlichen JavaDoc-Dokumentation erklärt.

4 JUnit

Die Verwendung von JUnit wird in diesem Artikel sehr gut beschrieben. Er enthält auch eine nützliche Tabelle der wichtigsten Methoden. Selbstverständlich findet man auf junit.org auch umfangreiche Erklärungen.

5 Dokumentation

Damit ein Test auch von anderen überprüft und von dem, der ihn ausführt, verstanden werden kann, ist eine detailliert Dokumentation wichtig. Diese sollte grundsätzlich in englischer Sprache verfasst werden. Ein Test, der die Klasse [Name] der Aufgabe [X] überprüft und von [Autor] geschrieben wurde, könnte diesen Javadoc-Kommentar erhalten:

```
/**
 * A test for [Name] (Task [X]) <br>
 * <br>
 * People that checked this test for being correct and complete:
 * <ul>
 *      <li>[Autor]</li>
 * </ul>
 * <br><br>
 * Things that are currently not tested, but should be:
 * <ul>
 *      <li>...</li>
 * </ul>
 *
 * @author [Name]
 */
```

Eine Test-Methode, die die Methode [name] mit den Parametern (x) der implementierten Klasse auf die Eigenschaften [prop1], [prop2], ... testet, könnte den folgenden Javadoc-Kommentar erhalten:

```
/**
 * Tests the {@code [name](x)} method. Asserts that:
 * <ul>
 *      <li>[prop1]</li>
 *      <li>[prop2]</li>
 *      <li>...</li>
 * </ul>
 */
```

6 Beispiele

Üblicher Aufbau einer Methode

Ein üblicher Ablauf einer Test-Methode wäre, ein `TestObject` zu instanziieren, eine Methode darauf auszuführen und das Ergebnis der Operation auszugeben:

```
@Test
public void testRemove() {
    TestObject testObject;  // holds an instance of the tested class
    Object result;          // the result of doSomething()

    testObject = new TestObject(); // get the instance of the tested
        class
    result = testObject.run('doSomething'); //run the method
        doSomething()
    assertEquals("doSomething() is supposed to do this and that",
        expected, result); //check the result of doSomething()
}
```

main() aufrufen und System.out lesen

Die main-Methode einer Java-Klasse erwartet als einzigen Parameter einen Array vom Typ String. Übergibt man einer run-Methode aber einen `String[]`-Array, so wählt der Java-Compiler die run-Methode mit der Signatur `Object[] parameters` aus und übergibt jeden String einzeln an die Methode. Daher casted man den `String[]`-Array davor explizit zu `Object`. Um die Ausgabe einer Klasse ins Terminal zu überprüfen, kann man die Hilfsklasse `test.Sysout` verwenden.

```
@Test
public void testMain() {
    String output;
    String[] outputLines;
    String[] expectedResults = new String[] {
        /* ... */
    };

    Sysout.observe(); // start observing the Sysout
    TestObject.runStatic("main", (Object) new String[]{par1, par2}); //
         run main
    output = Sysout.getAll(); // get what was written to the Sysout
    Sysout.stopObserving(); // Resets everything. Never forget to stop
        observing!

    // If the program outputs multiple lines, it's wise to check it
        line by line, to provide more detailed error messages
    outputLines = output.split("\n");

    /*
     * You should definitely check your arrays here! Otherwise
     * the next part might throw NullPointerExceptions. And you
     * never ever want such Exceptions in a test method!
     */
```

```
    for (int i = 0; i < outputLines.length; i++) {
        assertEquals("message", expectedResults[i], outputLines[i]);
    }
}
```

Optionale Methode testen

TestObject gibt selbständig ausführliche Fehlermeldungen über die JUnit-Methode `fail()` aus, wenn etwas schief läuft. So auch, wenn eine Methode mit den run-Methoden aufgerufen wird, die in der implementierten Klasse nicht vorhanden ist. Möchte man eine Methode testen, deren Implementierung nicht vorausgesetzt werden kann, so kann man `TestObject.hasMethod()` verwenden:

```
/*
 * test only if the method is implemented. If the method is not
 * implemented, no error message is created. Only if it is present
 * but not working properly, this test will fail.
*/
@Test
public void testFoo() {
    if (TestObject.hasMethod("foo")) {
        TestObject testObject;
        Object result;

        testObject = new TestObject();
        result = testObject.run("foo");
        assertEquals("message", expected, result);
    }
}
```

Package und Konstruktor überprüfen

Mit `TestObject.getPackageName()` kann das Package der implementierten Klasse geprüft werden. Dies kann man in einer eigenen Test-Methode machen. Jene, die den Konstruktor prüft, ist aber auch gut hierfür geeignet.

```
@Test
public void testFoo() {
    TestObject testObject;
    testObject = new TestObject("input"); // if there is no such
        constructor or anything fails while running it, TestObject will
         output error messages for us.
    assertNotNull("message", testObject); // check that the constructor
         really did its job

    // Test only that a package is set:
    assertNotNull("You are not allowed to use the default package!",
        TestObject.getPackageName());
    // Test that the package is a specific one:
    assertEquals("Please set the package name correctly!", "edu.kit.
        informatik", TestObject.getPackageName());
}
```