JUnit 4 Tutorial

Wolfgang Stöttinger

JUnit 4 Tutorial 1
1 Einführung in JUnit 4 3
1.1 Wie funktioniert JUnit? 3
1.2 Annotations 3
1.2.1 Test Annotation 3
1.2.2 Before Annotation 3
1.2.3 After Annotation 4
1.2.4 BeforeClass Annotation 4
1.2.5 AfterClass Annotation 4
1.3 Asserts 4
1.4 Exceptions erwarten 4
1.5 Testfunktionen ignorieren 5
1.6 Tests mit Timeout 5
2 Einbindung von JUnit in ein Eclipse Projekt 6
3 Erstellen eines JUnit Test Case 7
4 Ausführen eines JUnit Test Case 8

1 Einführung in JUnit 4

JUnit ist ein Tool, welches zum Testen von Java Programmen verwendet wird.
Pro Java Klasse wird eine eigener JUnit Test Case erstellt, welcher in verschiedenen Methoden die Funktionen der Klasse Testet.

Beispiel für einen JUnit Test Case:

```
import junit.framework.Assert;

import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class SimpleClassTestCase
{
	private SimpleClass simple;

	public SimpleClassTest()
	{
		simple = new SimpleClass();
		simple.setText("HelloWorld");
	}

	@Test public void getText ()
	{
		Assert.assertEquals("HelloWorld",
				simple.getText());
	}
}
```

1.1 Wie funktioniert JUnit?

JUnit Tests basieren auf definierten Behauptungen (Asserts), welche ausgewertet werden. Ist ein Assert falsch, so wird eine Exception geworfen und der Test Case schlägt an dieser Stelle fehl. Alle weiteren Testsfunktionen werden nicht mehr aufgerufen. Natürlich können auch manuell Exceptions geworfen werden, welches zu dem selben Resultat führt.

1.2 Annotations

Seit JUnit 4 werden Annotations zur Identifikation der Methoden verwendet, ebenso ist es nu nicht mehr notwendig, von junit.framework.TestCase abzuleiten.

1.2.1 Test Annotation

Die Annotation `@Test` zeigt an, dass es sich bei dieser Funktion um eine Testfunktion handelt, sie wird bei jedem Testlauf ausgewertet und ist erfolgreich, solange keine Exception auftritt.

```
@Test public void getText()
```

1.2.2 Before Annotation

Funktionen mit der Annotation `@Before` werden vor jeder Testfunktion aufgerufen und werden zum Initialisieren verwendet.

```
@Before public void setUp () throws Exception
```

1.2.3 After Annotation

Funktionen mit der Annotation `@After` werden nach jeder Testfunktion aufgerufen und werden verwendet um z.B.: Connections zu schließen.

```
@After public void tearDown () throws Exception
```

1.2.4 BeforeClass Annotation

Im Gegensatz zur Before Annotation kann diese Annotation nur einmal verwendet werden und wird nur ein einziges mal, vor der Instanzierung der Klasse aufgerufen.

```
@BeforeClass public static void setUpBeforeClass ()
              throws Exception
```

1.2.5 AfterClass Annotation

Ähnlich die die BeforeClass Annotation kann diese Annotation auch nur einmal verwendet werden. Diese Funktion wird am Ende des Tests aufgerufen.

```
@AfterClass public static void tearDownAfterClass ()
              throws Exception
```

1.3 Asserts

Es gibt verschiedene Arten von Asserts, welche alle nach demselben Prinzip funktionieren: Wenn sie fehlschlagen, wir deine Exception geworfen und der Test Case schlägt fehl.

Es gibt folgende Arten von Asserts:

```
//Behauptung: beide Werte sind gleich
Assert.assertEquals(42, 40 + 2);
//Behauptung: die Bedingung ist richtig
Assert.assertTrue(true);
//Behauptung: die Bedingung ist falsch
Assert.assertFalse(false);
//Behauptung: die Referenz ist null
Assert.assertNull(null);
//Behauptung: die Referenz ist nicht null
Assert.assertNotNull(new String());
//Behauptung: beide Referenzen sind das selbe Objekt
Assert.assertSame(simple, simple);
//Behauptung: beide Referenzen sind nicht das selbe Objekt
Assert.assertNotSame(simple, simple2);
//Test schlägt fehl
Assert.fail();
Assert.fail("Not yet implemented");
```

1.4 Exceptions erwarten

Es gibt für einen Test die Möglichkeit, eine Exception zu akzeptieren, wenn dies gewünscht wird. Dazu kann man für die Annotation `@Test` Parameter definieren.

```
@Test(expected = NullPointerException.class)
public void anyTestFunction ()
```

1.5 Testfunktionen ignorieren

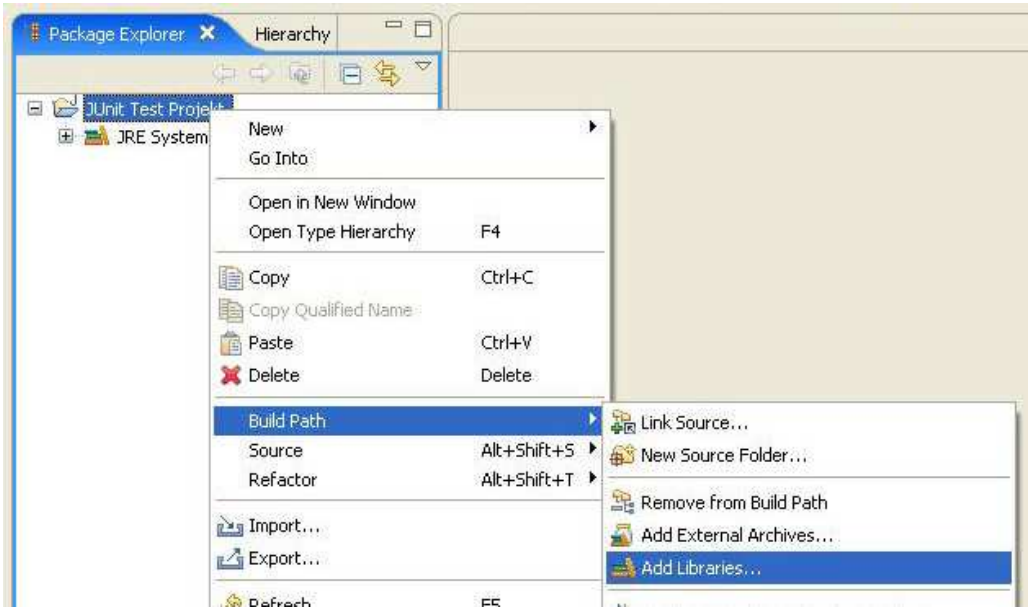
Es besteht die Möglichkeit eine Testfunktion zu ignorieren. Dazu muss die Annotation `@Ignore` vor die Funktion gestellt werden. Zusätzlich kann ein Parameter übergeben werden, warum die Funktion ignoriert wird: `@Ignore("Datenbank offline")`. Dadurch wird die Funktion beim Testlauf zwar angezeigt, aber als ignoriert markiert.

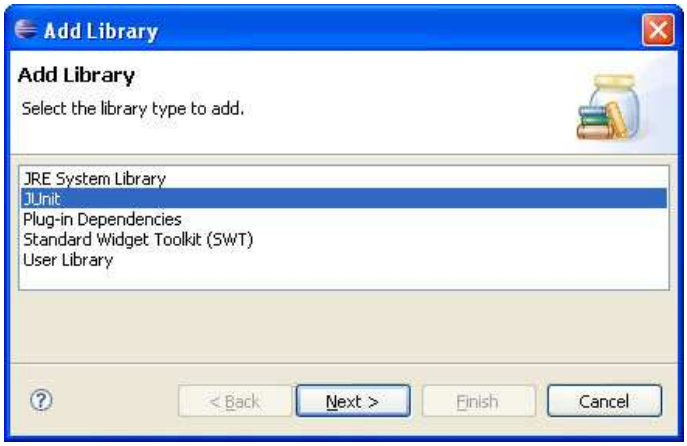
1.6 Tests mit Timeout

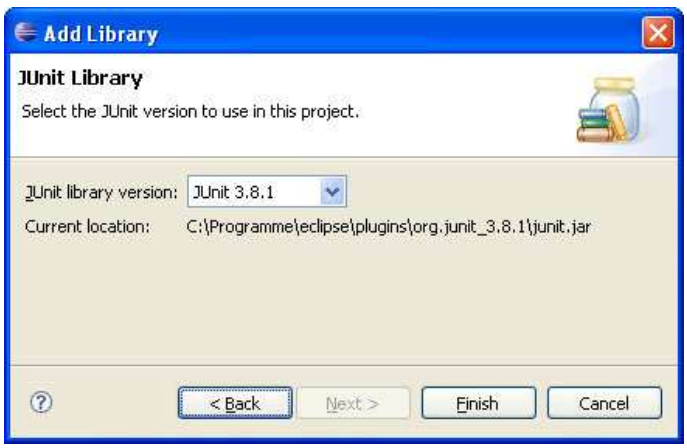
Der Test Annotation kann ein Parameter Timeout übergeben werden, der festlegt, wie lange die Funktion für ihre Durchführung benötigen darf. `@Test(timeout=10) schlägt fehl, sobald die Funktion länger als 10ms benötigt.`

2 Einbindung von JUnit in ein Eclipse Projekt

JUnit wird standardmäßig mit Eclipse 3.2 mitgeliefert und muss daher nur mehr dem Build Path hinzugefügt werden.



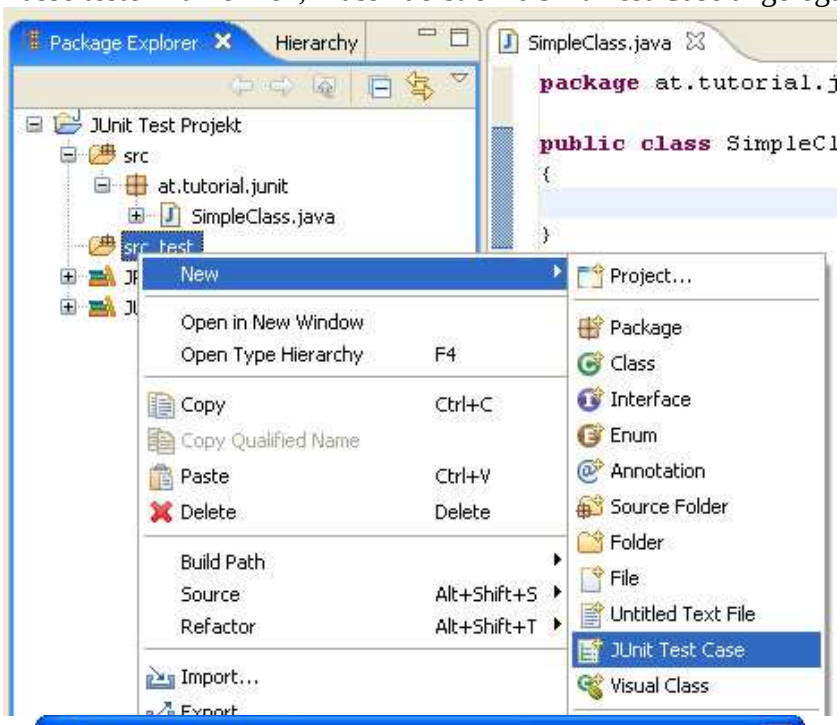


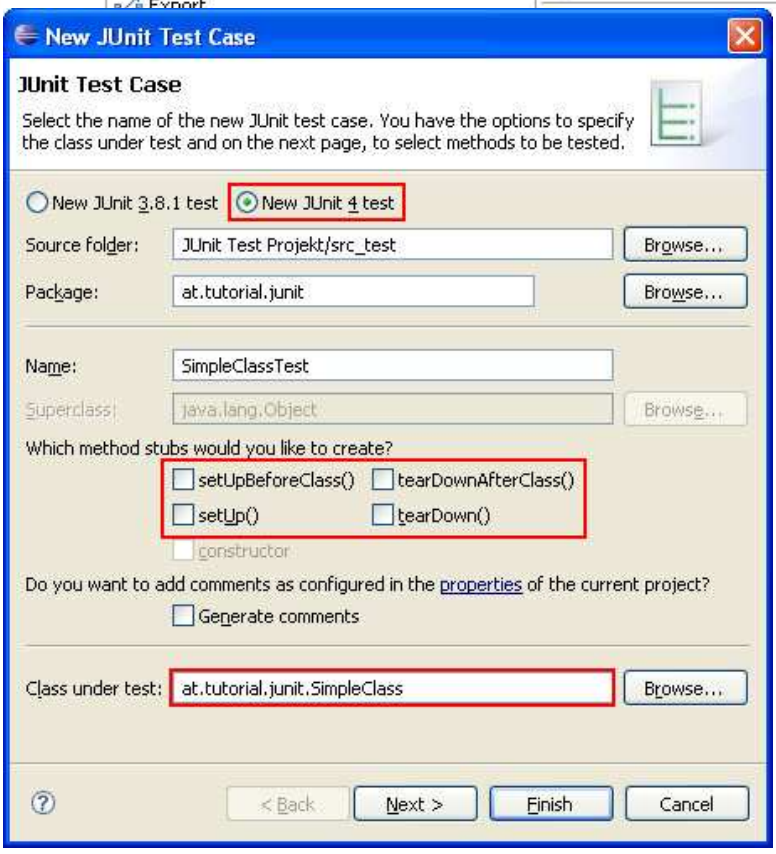


Es können JUnit 3.8.1 und JUnit 4 ausgewählt werden.

3 Erstellen eines JUnit Test Case

Um eine Klasse testen zu können, muss zuerst ein JUnit Test Case angelegt werden.

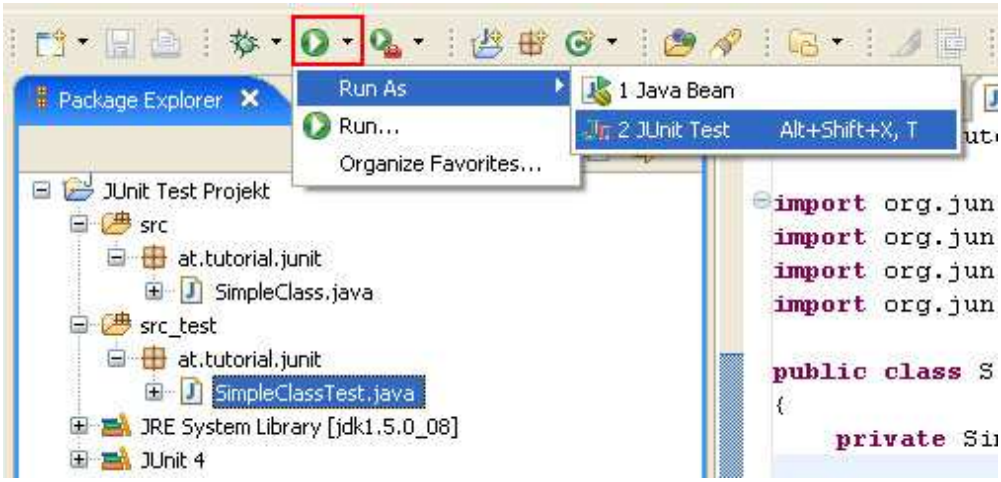




Funktionen für setUp und tearDown können automatisch generiert werden.
Um Testfunktionen generieren zu lassen, muss eine Testklasse ausgewählt werden und auf weiter geklickt werden.

4 Ausführen eines JUnit Test Case

Das Ausführen eines JUnit Test Case in Eclipse ist sehr einfach, dazu muss lediglich die Klasse ausgewählt werden und als JUnit Test gestartet werden.



Danach öffnet sich ein neuer View, in dem der Fortschritt des Tests angezeigt wird. Im oberen Bereich sieht man den Status der jeweiligen Testfunktion und im unteren Bereich sieht man eventuelle Fehlerausgaben.