JUnit 4

KW 28, Zentralübung Informatik II

2006-07-11

JUnit

- Entwickelt von:
  - Kent Beck („eXtreme Programming“)
  - Erich Gamma („Design Patterns“).
- Hat fast 8 Jahre ohne grössere Änderungen überdauert.

Lektionen von JUnit 3

- Das Composite-Pattern: (u.a.) zum Zusammenbauen von Tests ist zu kompliziert.
- Vielfältig eingesetzt: Es stand nicht zu erwarten, dass so viele Leute JUnit erweitern würden (Integration in IDEs etc.).
- Gute Ideen alleine genügen nicht: Passende Werkzeuge sind essentiell für ihre Verbreitung.
  Andere Beispiele hierfür (bestehende Ideen, klever neu verpackt, kostenlos verbreitet übers Internet): Java, Eclipse.
- Kulturelle Veränderungen brauchen Jahrzehnte um sich durchzusetzen: Die grundlegenden Ideen des Testens gibt es seit langem, JUnit und eXtreme Programming haben für eine allgemeinere Verbreitung gesorgt.

Quelle: http://blogs.zdnet.com/Burnette/?p=118

JUnit 4

- Weiterentwicklung von JUnit 3.
- Einiges ist einfacher, vor allem die Definition von Test-Suites.
- Setzt Java-5-Features ein (und voraus): Static imports, Annotationen.
- Beim neuesten Eclipse (3.2, Build Date: 29. Juni 2006) schon mit dabei!

Java 5: Static Imports

```
import static java.lang.Math.*;

public class StaticImports {
    public static void main(String[] args) {
        Math.cos(2 * Math.PI);
        cos(2 * PI); // kürzer per "import static"
    }
}
```

Java 5: Annotationen

Annotationen schreibt man vor eine Definition, direkt vor *Modifiers* wie `public` oder `static`.

- Annotiert werden können (die Definition von): Typen (Klassen, Interfaces, . . .), Attribute, Methoden, Parameter, Konstruktoren, lokale Variablen und Packages.

- Arten von Annotationen:
  - Marker (0 Argumente): `@Override`
  - Single-Value (1 Argument): `@SuppressWarnings("unchecked")`
  - Full (mind. 1 Argument):
    `@Test(expected=IndexOutOfBoundsException.class, timeout=5000)`

- Kann man auch selbst definieren.

Java 5: Standard-Annotationen

- `@Override` (wichtig): Manchmal denkt man nur, man überschreibt (Tippfehler, falsche Signatur), mit `@Override` kann man sicher gehen.

```
@Override
public String toStirng() { // TIPPFEHLER ==> Fehlermeldung
    return "";
}
```

- `@Deprecated` (für API-Entwickler): Auch wenn kein JavaDoc da ist.

- `@SuppressWarnings` (selten): Vor allem für Generics-Tricksereien.

JUnit 3: Test-Klasse

```
import junit.framework.TestCase;

public class SomeTest3 extends TestCase { // TestCase erweitern
    public void testLength() { // Namenspräfix "test"
        assertEquals(3, "abc".length());
    }
}
```

JUnit 3: Test-Suite

```
import junit.framework.Test;
import junit.framework.TestSuite;

public class MainSuite3 {
    public static Test suite() { // standardisierter Name
        TestSuite suite = new TestSuite();
        suite.addTestSuite(SomeTest3.class); // Test-Klasse
        suite.addTest(SubSuite.suite()); // Test-Suite
        return suite;
    }
}
```

Welches Pattern wird verwendet?

JUnit 4: Test-Klasse

```
// assert*-Methoden per import static:
import static org.junit.Assert.assertEquals;

import org.junit.Test;

public class SomeTest4 { // kein "extends" nötig
    @Test // kennzeichnet Test-Methode
    public void lengthOK() { // beliebiger Name
        assertEquals(3, "abc".length());
    }
}
```

JUnit 4: Test-Suite

- Nicht mehr nötig!
- Man kann jetzt (in Eclipse) beliebige Ordner als Test „ausführen“.

Vor/nach jedem Test ausführen (JUnit 4)

```
import org.junit.*;

public class BeforeAfter4 {
    @Before // JUnit3: setUp()
    public void vorJedemTest() { }
    @After // JUnit3: tearDown()
    public void nachJedemTest() { }

    @BeforeClass
    public void vorAllenTests() { }
    @AfterClass
    public void nachAllenTests() { }
}
```

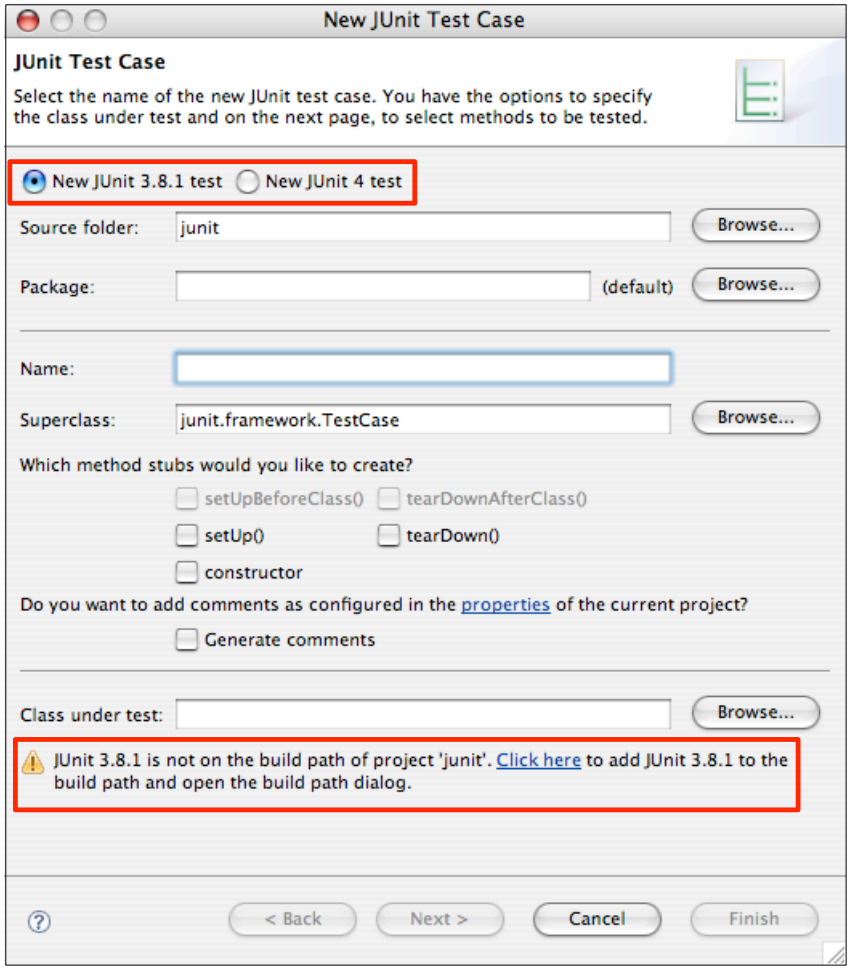
Exceptions fordern (JUnit 3)

```
import junit.framework.TestCase;

import org.junit.Test;

public class ExceptionTest3 extends TestCase {
    public void testDivisionByZero() {
        try {
            int tmp = 3 / 0;
            fail();
        } catch(ArithmeticException e) {
            // success
        }
    }
}
```

Exceptions fordern (JUnit 4)

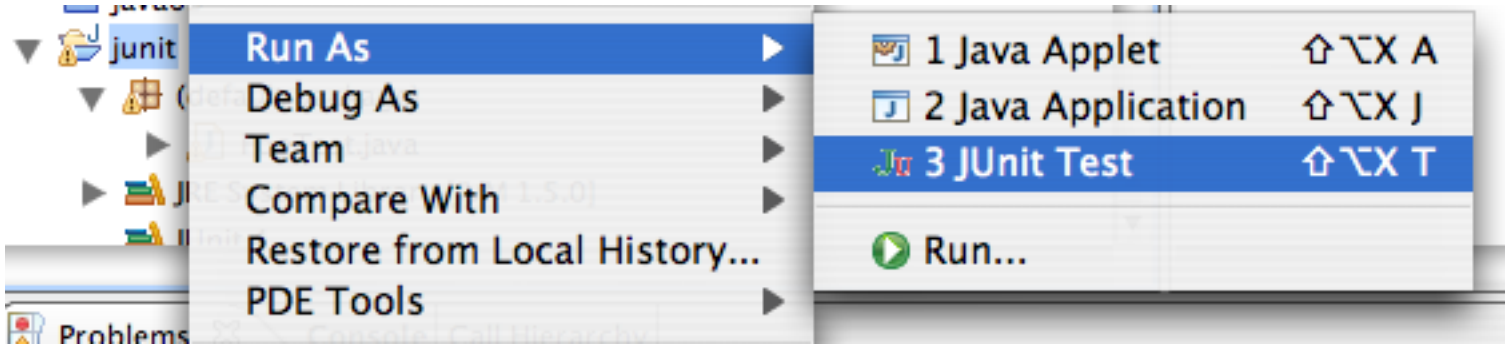
```
import org.junit.Test;

public class ExceptionTest4 {
    @Test(expected = ArithmeticException.class)
    public void divisonByZero() {
        int tmp = 3 / 0;
    }
}
```

Eclipse – Test erstellen: New → JUnit Test Case

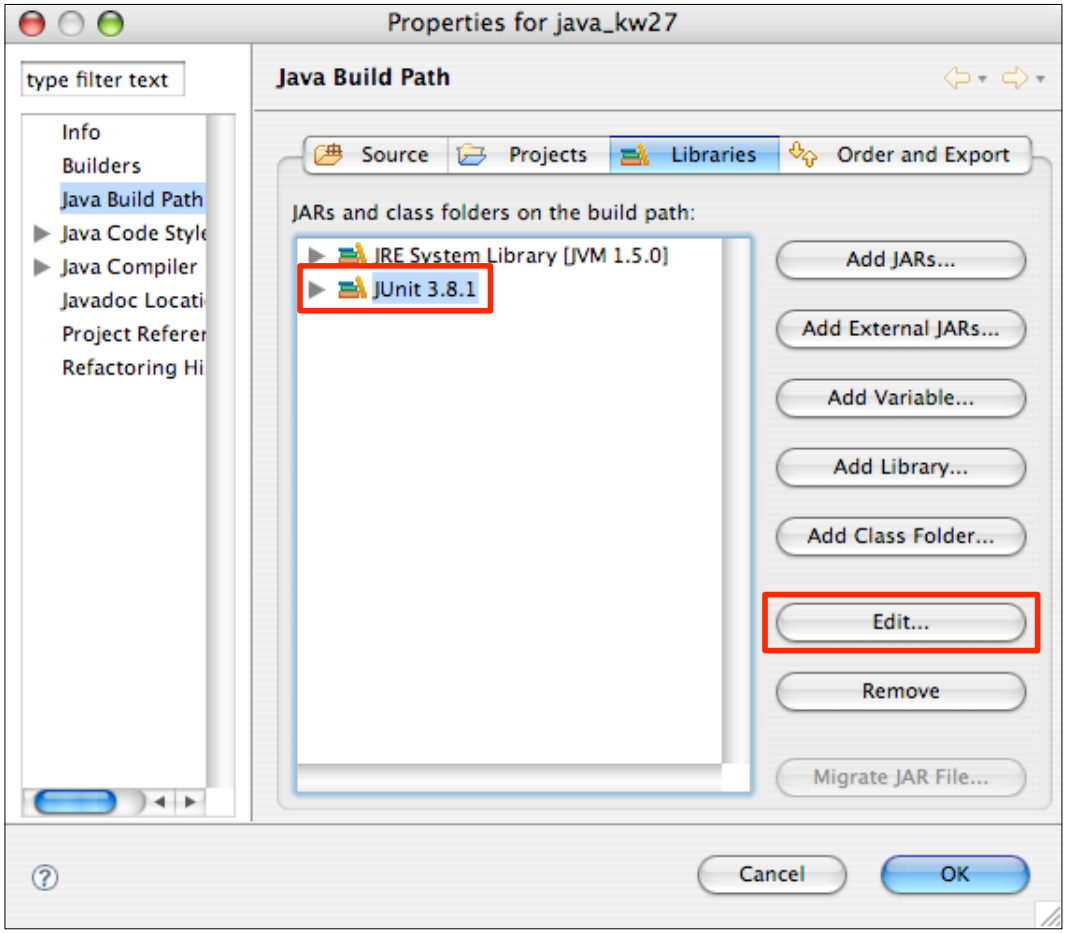


Version 3.2!

JUnit 4: Tests laufen lassen



Von JUnit 3 zu JUnit 4 wechseln



Project → Properties

TestRunner

Neue Tests mit altem TestRunner einsetzen: Adapter-Klasse `JUnit4Test`.

```
public static junit.framework.Test suite() {
  return new JUnit4TestAdapter(JUnit4Test.class);
}
```

- Alte Test laufen auch mit dem neuen TestRunner.
- Der neue TestRunner hat keine GUI mehr (das wird dem IDE überlassen).

Zusammenfassung: JUnit 3 und JUnit 4

	JUnit 3	JUnit 4
Package	`junit.framework`	`org.junit`
Test-Klasse markieren	extends TestCase	nicht nötig
Test-Methoden markieren	Namenspräfix `test`	`@Test`
`assert*`	geerbt (statisch)	`static import Assert.*`
Testmengen definieren	`static Test suite()` gibt TestSuites zurück	„Run as Test“ von Verzeichnissen
Vor jeder Methode ausführen	setUp()	@Before
Nach jeder Methode ausf.	tearDown()	@After
Vor allen Methoden ausführen	(Simuliere per `TestSetup`)	@BeforeClass
Nach allen Methoden ausf.	(Simuliere per `TestSetup`)	@AfterClass
Exception E fordern	`try{}catch(E){fail}`	`@Test(expected=E.class)`