

# Programmierung 2

## Studiengang MI / WI

Dipl.-Inf., Dipl.-Ing. (FH) Michael Wilhelm

Hochschule Harz

FB Automatisierung und Informatik

[mwilhelm@hs-harz.de](mailto:mwilhelm@hs-harz.de)

Raum 2.202

Tel. 03943 / 659 338

# Inhalt der Vorlesung

## Überblick:

- Objekte und Methoden
- Swing
- Exception
- I/O-Klassen / Methoden
- Threads
- Algorithmen (Das Collections-Framework)
- Design Pattern
- Graphentheorie
- **JUnit**

# Softwareentwicklung

- Analyse
- Modellierung
- Codierung
- Testen der einzelnen Module
- Testen aller Module
- Ausliefern

# Testroutinen

## ■ Blackbox-Test

- Werden von speziellen Testern durchgeführt. Sie sind nicht Teil des Entwicklungsteam. Haben keinen Source-Code

## ■ Unit, Komponenten, Whitebox-Test

- Die Komponenten werden einzeln unabhängig von Einsatz getestet. Tester sind Teil des Entwicklungsteam. Haben Zugriff auf den Source-Code.

# Testroutinen

## ■ Akzeptanztest

- Werden von Kunden bei der Abnahme durchgeführt. Prüfung des Pflichtenhefts

## ■ Integrationstest, Interaktionstest

- Sie testen das Zusammenspiel von einzelnen Komponenten

## ■ Performanz- Lasttest

- Testen Antwortzeiten der Benutzer, Arraygrößen, Zeiten für Suchalgorithmen etc.

# Testkriterien für Test-Werkzeuge

- Testcode getrennt von Quellcode
  - sonst Auslieferung des Testcodes an den Kunden?
  - sonst große Klassen
- Unabhängige Testfälle
  - Sie testen das Zusammenspiel von einzelnen Komponenten
- Ergebnis der Tests
  - Sofortiger Überblick
  - Keine log-Files etc.

# Test-Werkzeuge: JUnit

## ■ Autoren

- Kent Beck (Vater der eXtreme Programming)
- Erich Gamma (Gang of Four)

## ■ Version

- Aktuell: 4,10                      enthält nicht mehr swingui

## ■ Internet-Seiten

- <http://www.junit.org>
- <http://www.xprogramming.com/>

# Integration von JUnit in IDEs

- PlugIn für Eclipse ab Version 2.1
- Integration in JBuilder (ab Version 8.x)
- Integriert in NetBeans

# Testbeispiel: Projekt Junit\_bsp1

```
public class Mathe {
```

```
    public double add(double x, double y) {  
        return x+y;  
    }
```

```
    public int add(int x, int y) {  
        return x+y;  
    }  
}
```

# Beispiel mit einer eigenen Testroutine

```
public void test1() {  
    int a,b,c;  
    Mathe myclass = new Mathe();    // neue Klasse  
    c = myclass.add(5, 4);  
    if (c != 9 ) {  
        System.out.println("Fehler beim 1. Test Add: "+c );  
    }  
}
```

```
public static void main(String[] args) {  
    ( new Mathe() ).test1();  
}
```

# Beispiel mit einer eigenen Testroutine

```
public void test2() {  
    double a,b,c;  
    Mathe myclass = new Mathe();    // neue Klasse  
  
    c = myclass.add(5.5, 4.75);  
    if (c != 10.25) {  
        System.out.println("Fehler beim 2. Test Add: "+c );  
    }  
} // test2
```

```
public static void main(String[] args) {  
    ( new Mathe() ).test2();  
}
```

# Benutzeroberflächen von JUnit

- `junit.textui.TestRunner`
  - Textbasierte Ausgabe
- `junit.swingui.TestRunner`
  - Testumgebung mit Swing-Klassen (empfohlen)
  - Nur bis Version 3.8.1
- `junit.awtui.TestRunner`
  - Testumgebung mit AWT-Klassen

# JUnit-Testen

- Erstellen eines normalen Projektes
- Erstellen einer Testsuite
- **Typisches Verfahren**
  - JUnit trennt den Testcode vom Quellcode
  - Jede Testklasse wird von „TestCase“ abgeleitet
  - **Tests** beginnen mit dem Namen „test“,

**sind public**

**sind nicht statisch**

**haben keine Parameter**

**@Test**

**// notwendig in Eclipse**

- **Ergebnis:**

- Grün: Erfolgreich
- Rot: Mindestens ein Failure oder Error ist aufgetreten
  - Failure (Test versagt)
  - Error (Abbruch durch eine Exception)

# Test:

## Aufruf auf der Konsole

```
C:\Daten>java -cp junit.jar;  
                junit.textui.TestRunner  junit.samples.AllTests
```

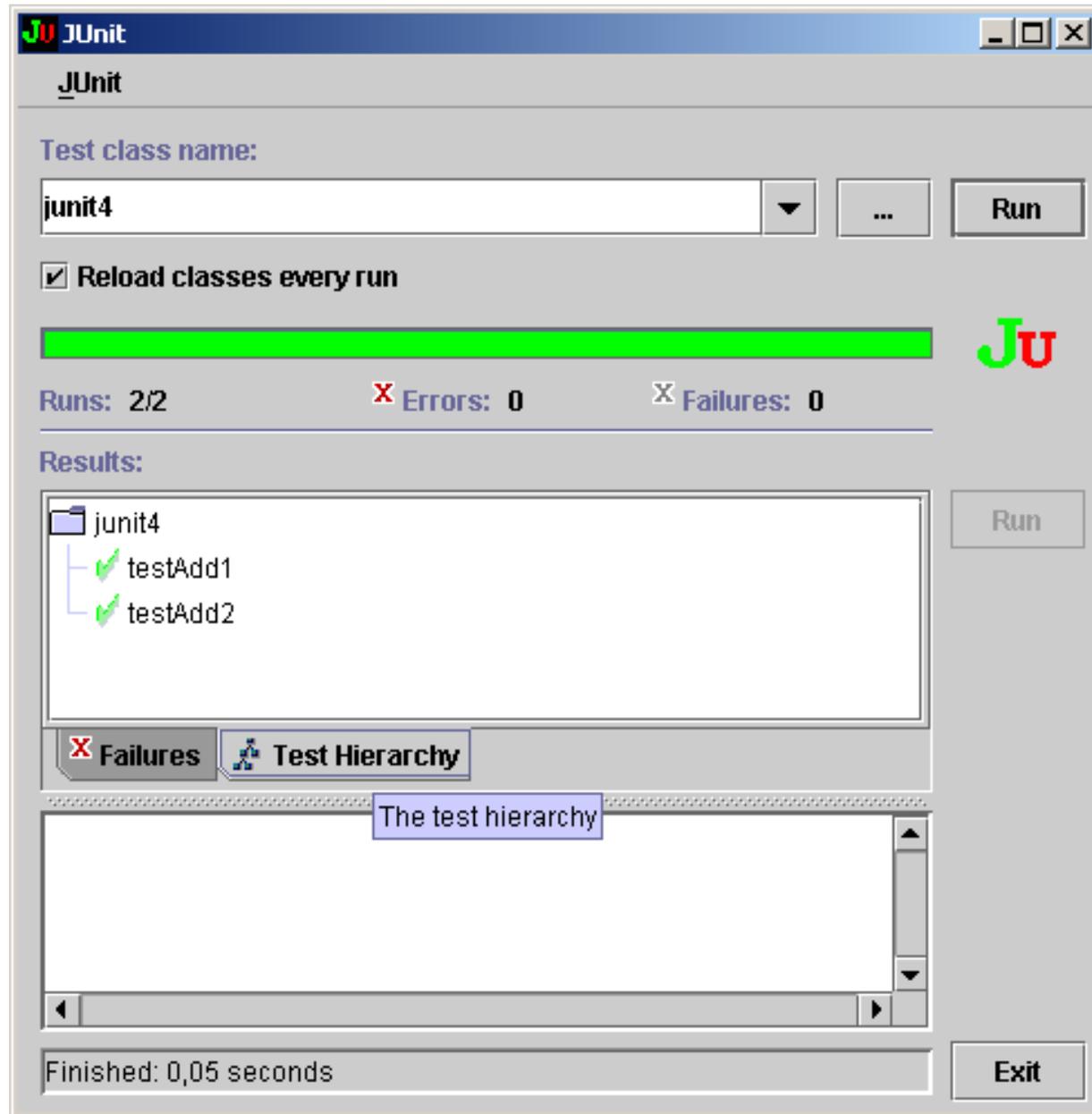
```
.....  
.....
```

Time: 0,861

OK (119 tests)

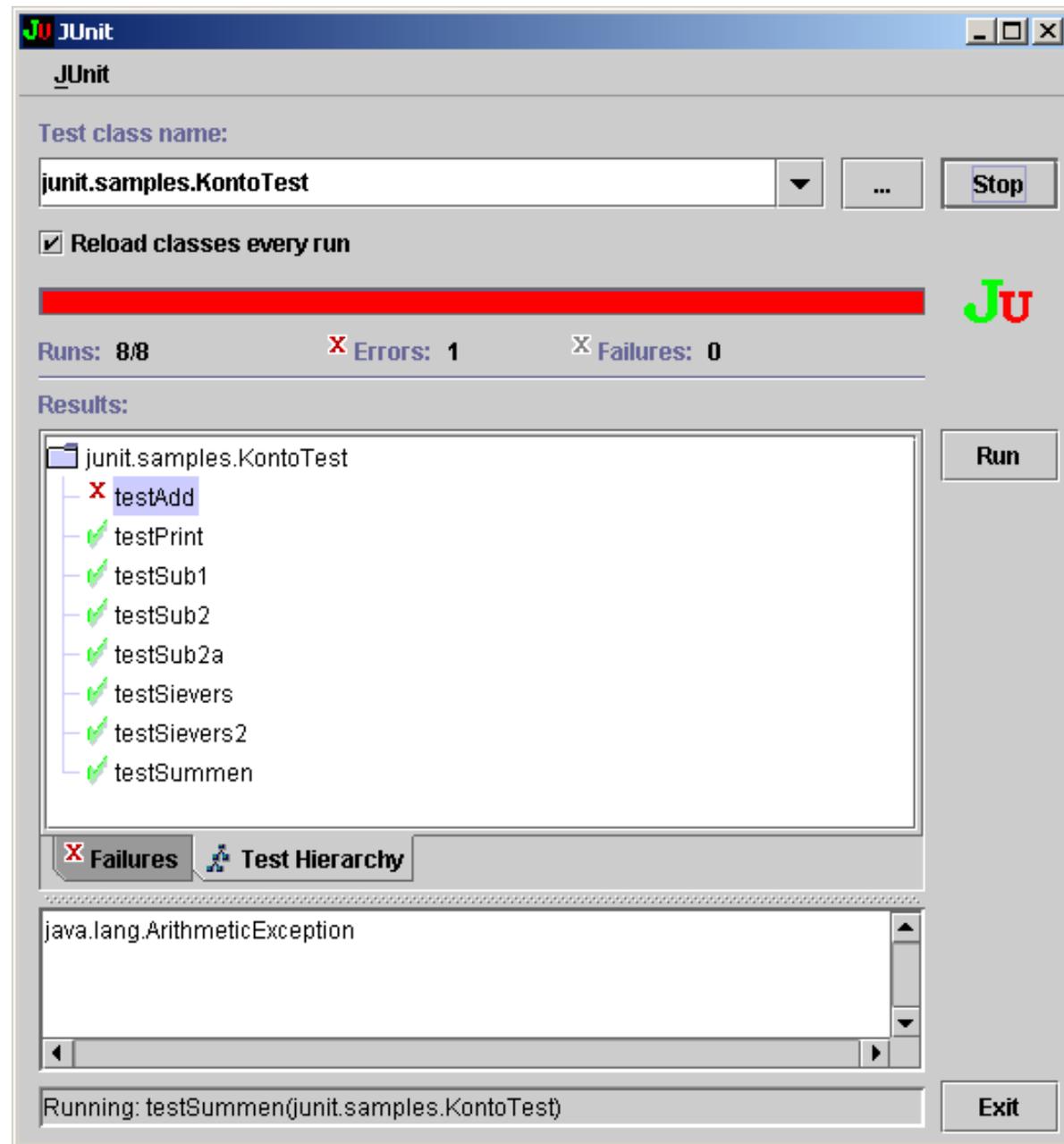
```
C:\Daten>
```

# GUI



# GUI-Test:

Methode  
TestAdd  
ist fehlerhaft



# Asserts

- `assertFalse(boolean expected, boolean actual)`
  - Bedingung ist falsch. Wenn gleich  $\Rightarrow$  Fehler
- `assertTrue(boolean expected, boolean actual)`
  - Bedingung ist wahr. Wenn ungleich  $\Rightarrow$  Fehler
- `assertEquals(int expected, int actual)`
- `assertEquals(double expected, double actual) ????`
  - Die zwei Parameter haben den gleichen Wert (Klasse)
- `assertNotNull`
  - Wenn Parameter vorhanden  $\Rightarrow$  okay
- `assertNull`
  - Wenn Parameter ist null  $\Rightarrow$  okay
- `assertSame`
  - gleiche Objekte

# Asserts

- `assertFalse(boolean expected, boolean actual)`
  - Bedingung ist falsch. Wenn gleich  $\Rightarrow$  Fehler
- `assertTrue(boolean expected, boolean actual)`
  - Bedingung ist wahr. Wenn ungleich  $\Rightarrow$  Fehler
- `assertEquals(int expected, int actual)`
- `assertEquals(double expected, double actual, double delta)`
  - Die zwei Parameter haben den gleichen Wert (Klasse)
  - Rundungsfehler
- `assertNotNull`
  - Wenn Parameter vorhanden  $\Rightarrow$  okay
- `assertNull`
  - Wenn Parameter ist null  $\Rightarrow$  okay
- `assertSame`
  - gleiche Objekte

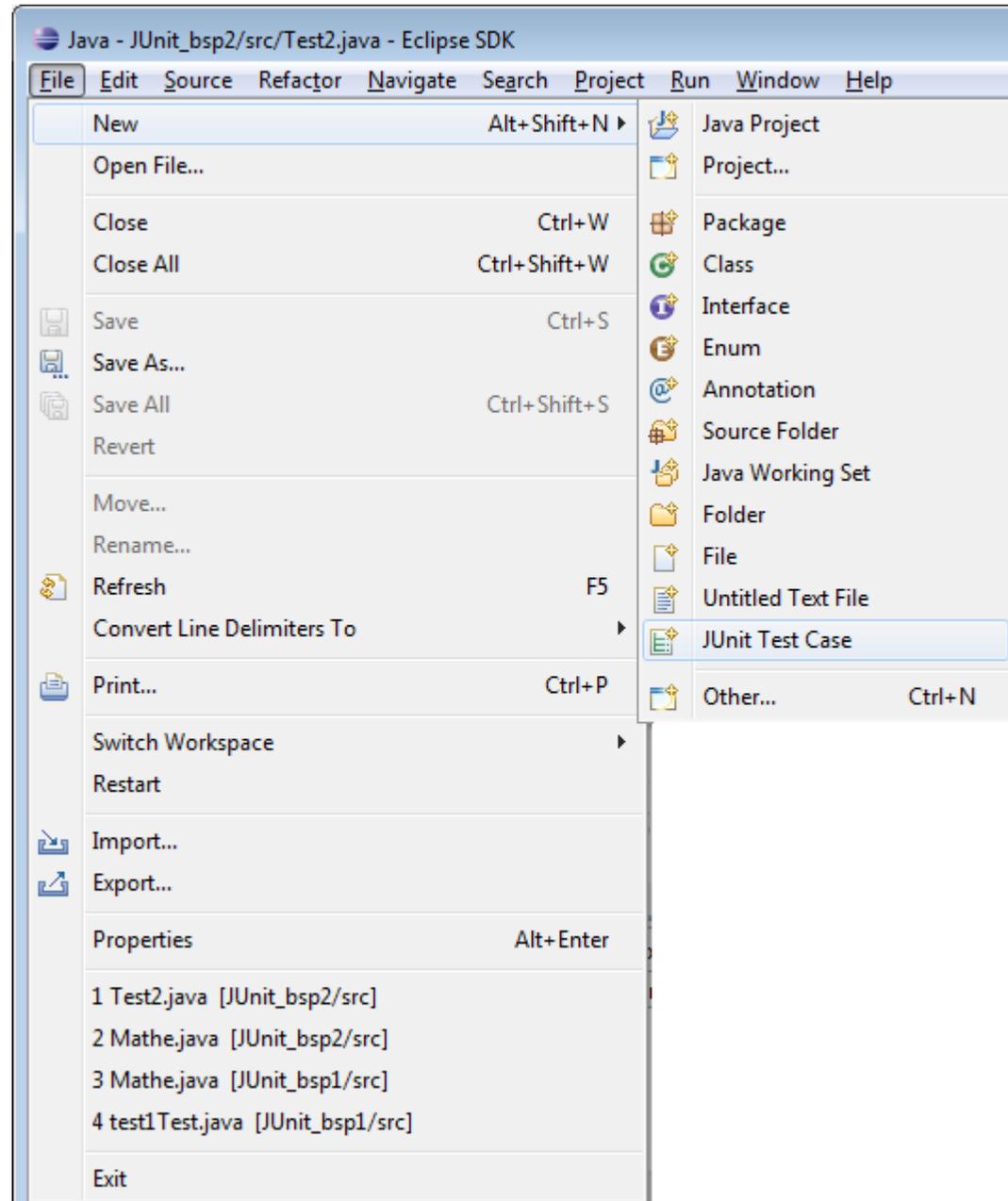
# Eclipsebeispiel: Projekt Junit\_bsp2 mit JUnit

```
public class Mathe {  
  
    public double add(double x, double y) {  
        return x+y;  
    }  
  
    public int add(int x, int y) {  
        return x+y;  
    }  
}
```

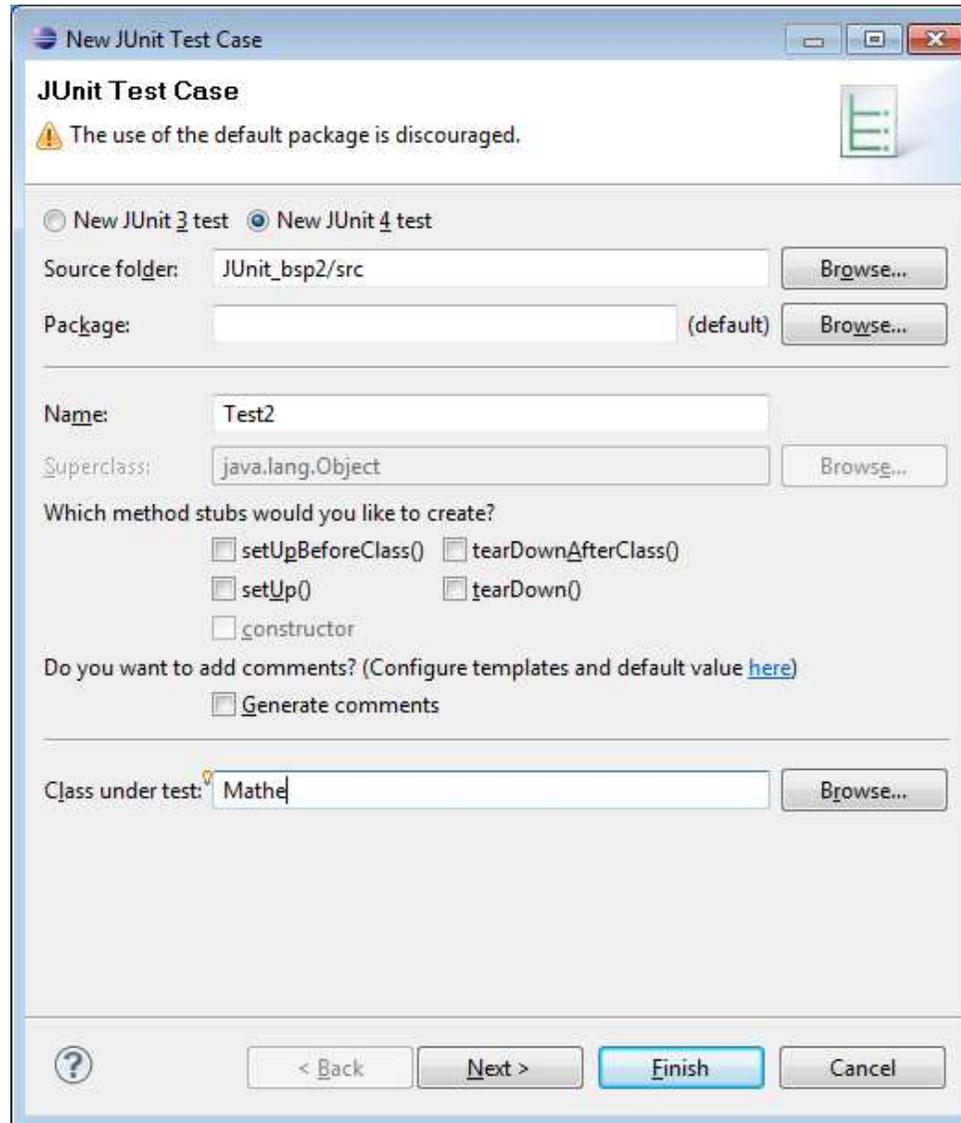
# Menü File:

New

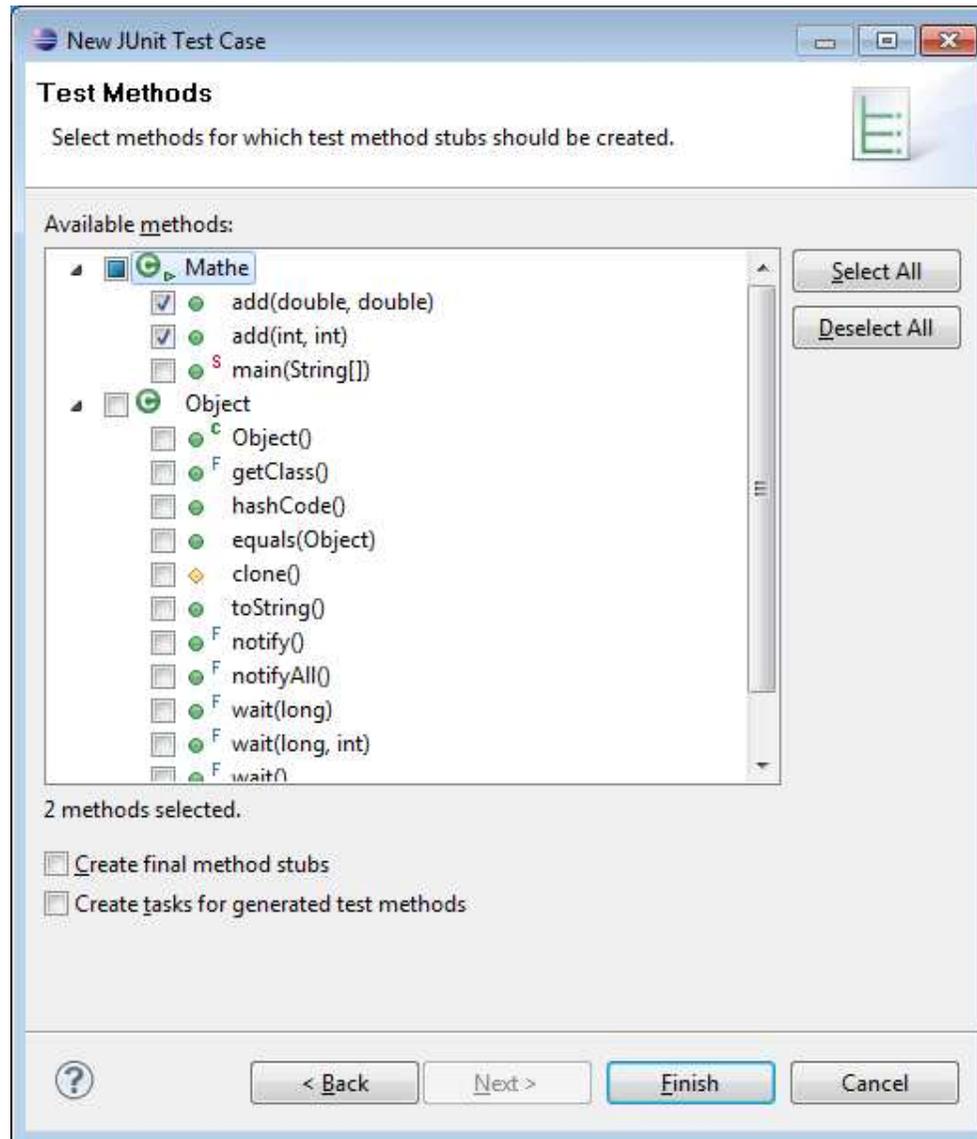
JUnit Test Case



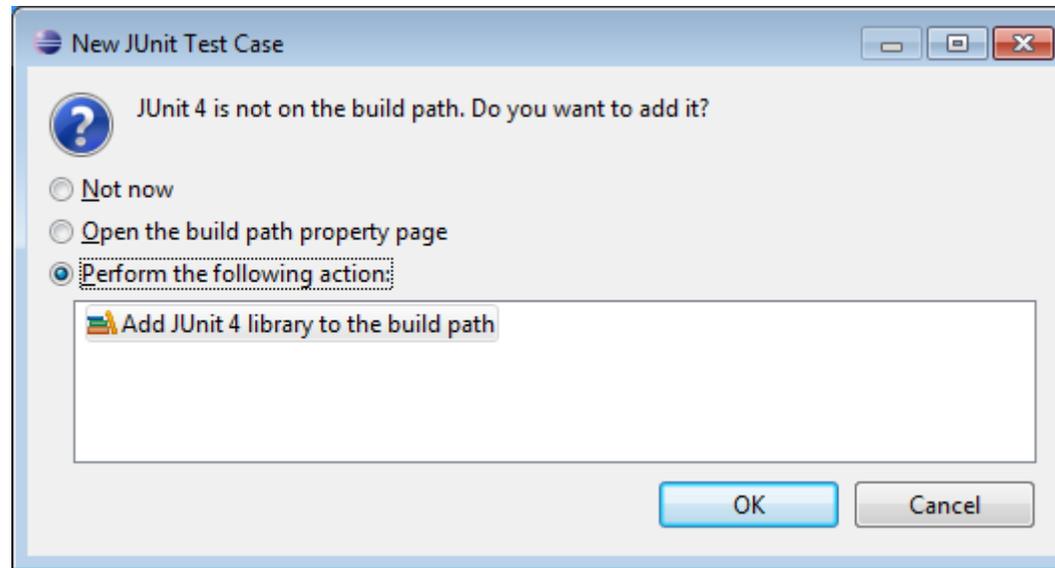
# Testbeispiel: Projekt JUnit\_bsp2 mit JUnit



# Testbeispiel: Projekt Junit\_bsp2 mit JUnit



# Testbeispiel: Projekt Junit\_bsp2 mit JUnit



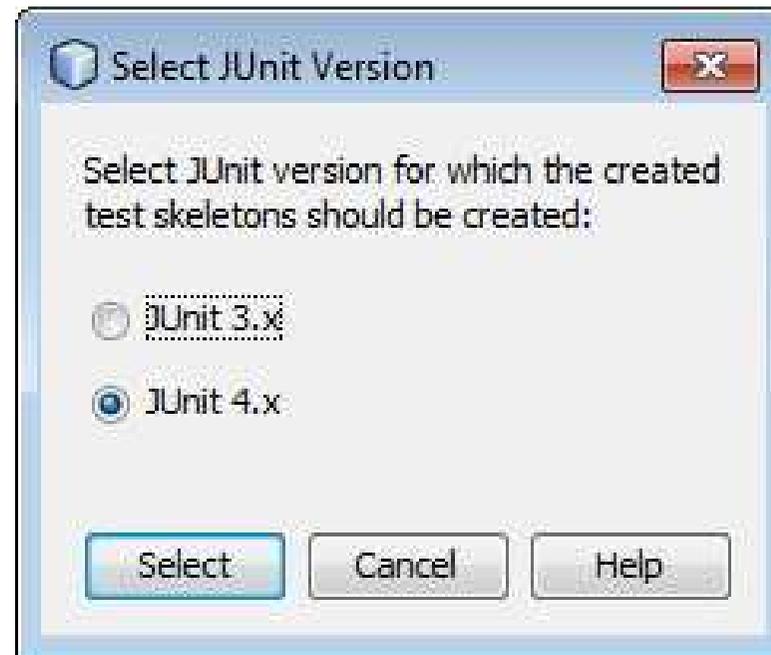
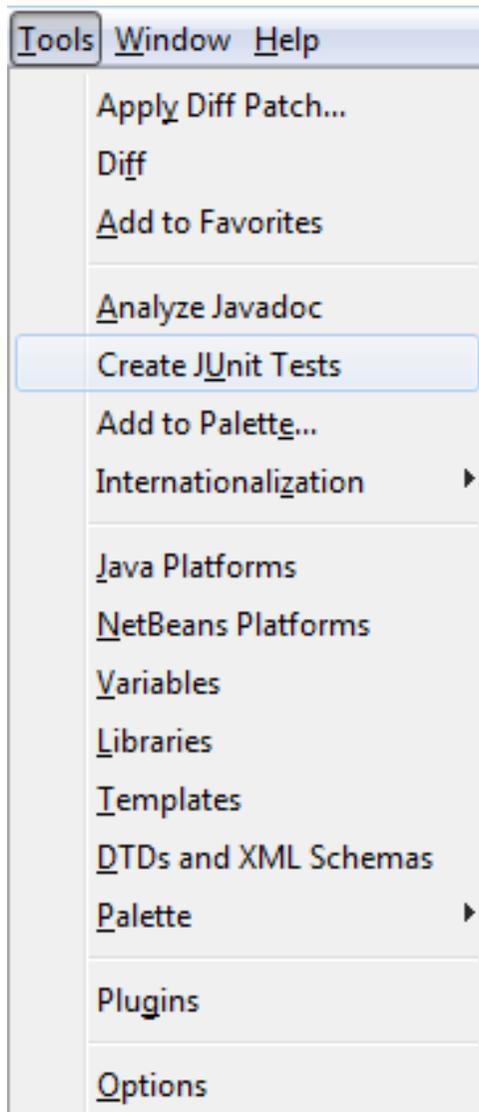
# Testbeispiel: Projekt JUnit\_bsp2 mit JUnit

The screenshot shows the Eclipse IDE interface for a Java project named 'JUnit\_bsp2'. The main editor displays the source code for 'Test2.java':

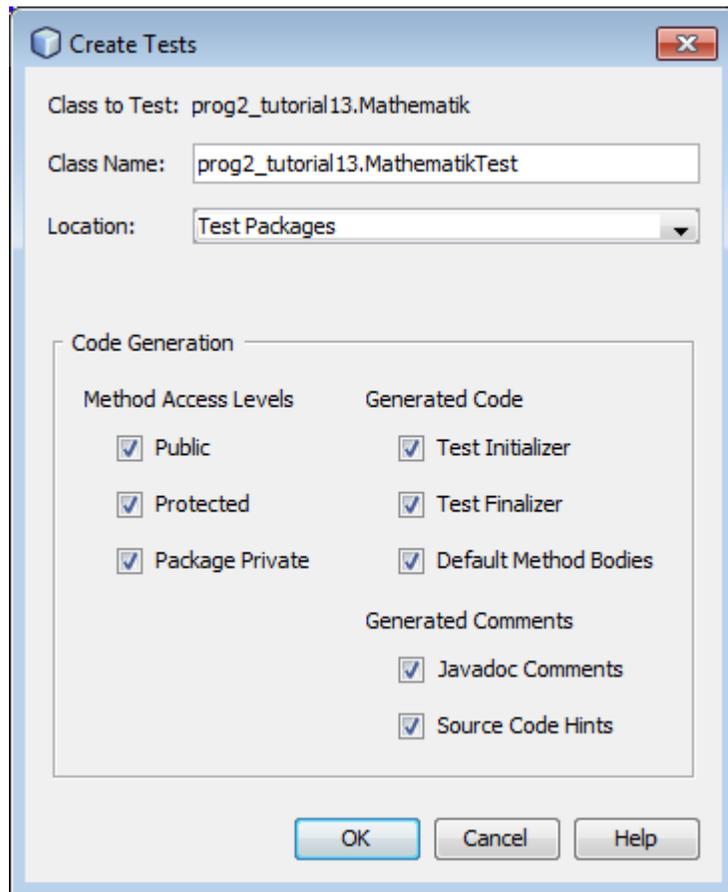
```
1 import static org.junit.Assert.*;
4
5
6 public class Test2 {
7
8     @Test
9     public void testAddDoubleDouble() {
10         fail("Not yet implemented1");
11     }
12
13     @Test
14     public void testAddIntInt() {
15         fail("Not yet implemented2");
16     }
17 }
```

The left sidebar shows the 'JUnit' view with a progress bar and summary: 'Finished after 0 seconds', 'Runs: 2/2', 'Errors: 0', and 'Failures: 2'. Below this, a tree view shows the test results for 'Test2 [Runner: JUnit 4] (0,000 s)', including 'testAddDoubleDouble (0,000 s)' and 'testAddIntInt (0,000 s)'. The 'Failure Trace' view shows the error: 'java.lang.AssertionError: Not yet implemented1 at Test2.testAddDoubleDouble(Test2.java:10)'. The bottom console shows the command: '<terminated> Test2 [JUnit] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (17.12.2011 10:32:45)'. The status bar at the bottom indicates 'Writable', 'Smart Insert', and the time '15 : 35'.

# Netbeans mit JUnit



# Netbeans mit JUnit



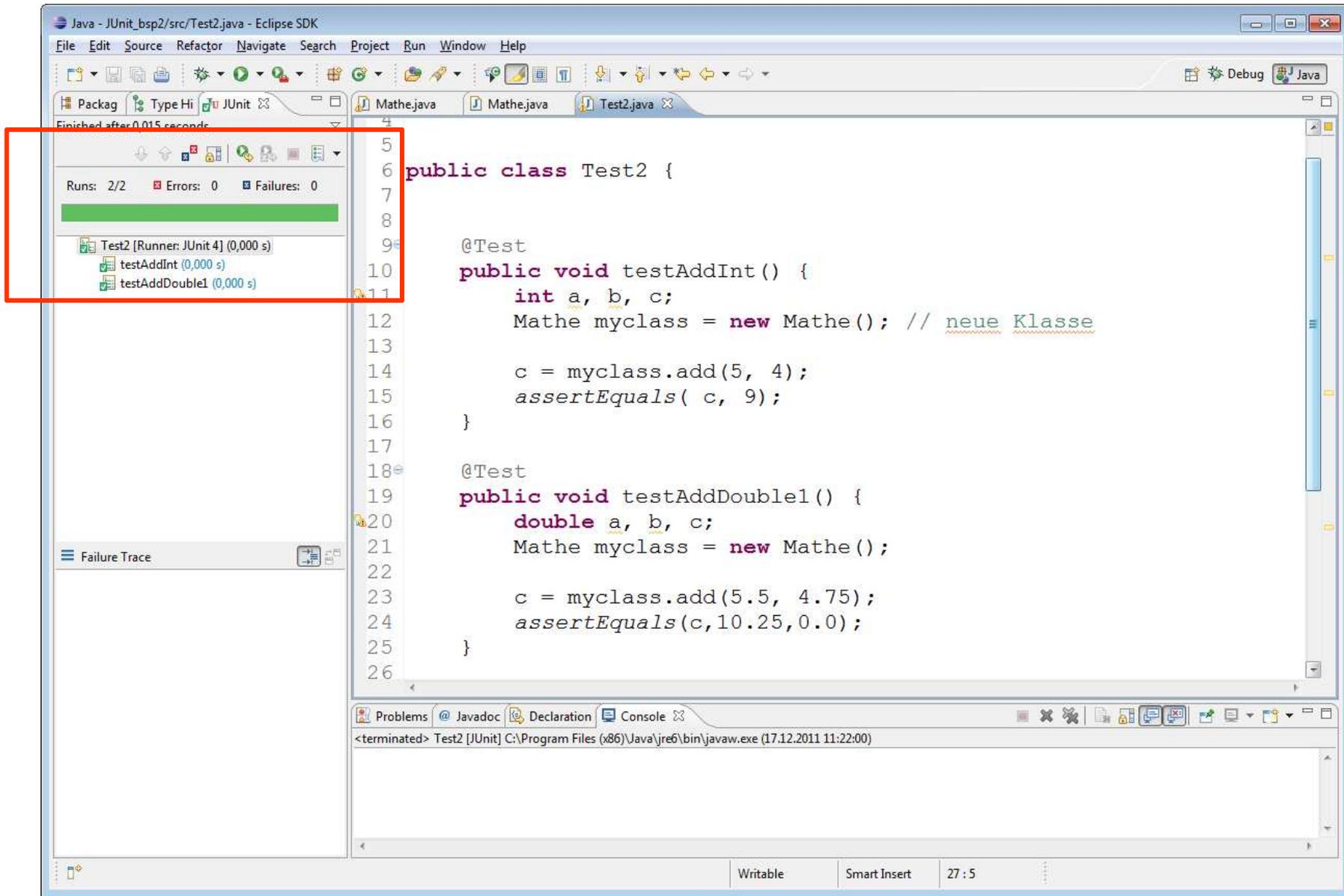
- Implementieren der Tests
- Starten mit Alt+F6

# Test-Case

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

```
public class Test2 {  
    @Test  
    public void testAddInt() {  
        int a, b, c;  
        Mathe myclass = new Mathe(); // neue Klasse  
  
        c = myclass.add(5, 4);  
        assertEquals( c, 9);  
    }  
  
    @Test  
    public void testAddDouble1() {  
        double a, b, c;  
        Mathe myclass = new Mathe();  
  
        c = myclass.add(5.5, 4.75);  
        assertEquals(c,10.25,0.0);  
    }  
}
```

# Testbeispiel starten: Register, STRG+F11



# Beispiel mit einer eigenen Testroutine

```
public void test2() {  
    double a,b,c;  
    Mathe myclass = new Mathe();    // neue Klasse  
  
    c = myclass.add(5.033, 4.343);  
    if (c != 9.376 ) {  
        System.out.println("Fehler beim 2. Test Add: "+c );  
    }  
}  
  
public static void main(String[] args) {  
    ( new Mathe() ).test2();  
}
```

# Testbeispiel: Projekt Junit\_bsp3 mit Matrix

$$\begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \cdot 1 + 4 \cdot 2 \\ 2 \cdot 1 + 5 \cdot 2 \\ 3 \cdot 1 + 6 \cdot 2 \end{pmatrix} = \begin{pmatrix} 9 \\ 12 \\ 15 \end{pmatrix}$$

$$\begin{pmatrix} 1,1 & 4,4 \\ 2,2 & 5,5 \\ 3,3 & 6,6 \end{pmatrix} \cdot \begin{pmatrix} 1,6 \\ 2,5 \end{pmatrix} = \begin{pmatrix} 1,1 \cdot 1,6 + 4,4 \cdot 2,5 \\ 2,2 \cdot 1,6 + 5,5 \cdot 2,5 \\ 3,3 \cdot 1,6 + 6,6 \cdot 2,5 \end{pmatrix} = \begin{pmatrix} 12,76 \\ 17,27 \\ 21,78 \end{pmatrix}$$

# Testbeispiel: Projekt Junit\_bsp3 mit Matrix

```
public double [] mult(double A[][], double B[] ){
    int i,j,m, n;
    double s;
    double C[];
    m = A.length;
    n = B.length;
    C = new double[m];
    for (i=0; i<m; i++ ) {
        s = 0;
        for (j=0; j<B.length; j++) {
            s += A[i][j] * B[j];
        }
        C[i] = s;
    }
    return C;
}
```

```

public final void testMult1() {
    double A[][];
    double B[];
    double C[];
    MatrixMult m;
    A = new double[3][2];
    B = new double[2];
    double[] CErg = {9.00,12,15};

    A[0][0]=1;  A[1][0]=2;   A[2][0]=3;   A[0][1]=4;
    A[1][1]=5;  A[2][1]=6;   B[0]=1;      B[1]=2;
    m = new MatrixMult();
    C = m.mult(A,B);
    for (int i=0; i<C.length; i++ ) {
        assertEquals( "Vectortest i"+i,C[i],CErg[i],0.0);
    }
} // test1

```

# Testbeispiel: Projekt Junit\_bsp3 mit Matrix

The screenshot shows the Eclipse IDE interface. The main editor displays the code for `MatrixMultTest.java`. The code includes a `@test` annotation and a `testMult3()` method. The method defines three double arrays: `A` (3x2), `B` (2x1), and `C` (1x3). It then creates a `MatrixMult` object and assigns values to `A` and `C`.

```
69 @test
70 public final void testMult3() {
71     double A[][];
72     double B[];
73     double C[];
74     MatrixMult m;
75     A = new double[3][2];
76     B = new double[2];
77     double[] CErg = {11.0385,14.7048,18.3711};
78
79     A[0][0]=1.123;
80     A[1][0]=2.234;
```

The left sidebar shows the JUnit test runner. It indicates that the tests finished after 0,016 seconds. The test results are as follows:

- Runs: 3/3
- Errors: 0
- Failures: 2

The test runner shows three tests: `testMult1`, `testMult2`, and `testMult3`. The `testMult2` test is highlighted with a red box, indicating it failed. The failure trace shows the error: `java.lang.AssertionError: Vergleich der Vektoren: at MatrixMultTest.testMult2(MatrixMultTest.java`.

The console output shows the results of the tests:

```
<terminated> MatrixMultTest [JUnit] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (17.12.2011 12:25:20)
Matrix C: 17.270000128746034
Matrix C: 21.779999763965606

Test3:
Matrix C: 11.038500000000003
Matrix C: 14.7048
Matrix C: 18.371100000000002
```

# Testbeispiel: Projekt Junit\_bsp4 mit Konto

- Kontoverwaltung
- Betrag / Name
- add
- sub
- toString
- Test:
  - add
  - sub
  - print
  - doppelte Buchführung (Summe vorher = Summe nachher)

# Klasse Konto

```
public class Konto {  
    private double _betrag;  
    private String _name;  
  
    public Konto(double betrag, String name) {  
        _betrag = betrag;  
        _name = name;  
    }  
  
    public void add (double betrag) {  
        _betrag += betrag;  
    }  
  
    public void sub (double betrag) {  
        _betrag -= betrag;  
    }  
}
```

## Klasse Konto (2)

```
public double getBetrag() {  
    return _betrag;  
}  
public String getName() {  
    return _name;  
}  
public String toString() {  
    return new String( "[ Name: "+getName()+  
        " Betrag: "+getBetrag()+" ]");  
}  
}
```

# Testbeispiel: Projekt Junit\_bsp4 mit Konto

```
package junit.samples.Konto;
```

```
import junit.framework.*;
```

```
public class KontoTest extends TestCase {
```

```
    private Konto k1 = new Konto(100,“Müller“);;
```

```
    private Konto k2;
```

```
    private Konto k3;
```

```
    private Konto k4;
```

```
    public static void main(String args[]) {
```

```
        junit.textui.TestRunner.run(KontoTest.class);
```

```
    }
```

# Testbeispiel: Projekt Junit\_bsp4 mit Konto

```
public void testAdd() {  
    // 100 + 20 = 120  
    k1.add(20);  
    assertEquals(k1.getBetrag(), 120.0, 0.0001);  
}  
  
public void testSub() {  
    // 100 - 20 = 80  
    k1.sub(20);  
    assertEquals(k1.getBetrag(), 80.0 , 0.0001);  
}  
  
public void testPrint() {  
    assertEquals("[ Name: Müller Betrag: 100.0 ]", k1.toString());  
}
```

# Testbeispiel: Projekt Junit\_bsp4 mit Konto

```
public void testSummen() {  
    double s1, s2;  
    s1 = k1.getBetrag() + k2.getBetrag() + k3.getBetrag() + k4.getBetrag();  
        k1.add(20.0);  
        k2.sub(20.0);  
  
        k1.add(60.0);  
        k4.sub(60.0);  
  
        k3.add(25.0);  
        k2.sub(25.0);  
  
    s2 = k1.getBetrag() + k2.getBetrag() + k3.getBetrag() + k4.getBetrag();  
    assertEquals(s1, s2, 0.0001);  
} // testSummen
```

```

public class KontoTest {
    private Konto k1 = new Konto(100.0, "Müller");
    private Konto k2 = new Konto(200.0, "Schmidt");
    private Konto k3 = new Konto(300.0, "Meyer");
    private Konto k4 = new Konto(400.0, "Schulze");

    public void testAdd() {
        // 100 + 20 = 120
        k1.add(20);
        assertEquals(k1.getBetrag(), 120.0, 0.0001);
    }

    public void testSub1() {
        // 100 - 20 = 80}
        k1.sub(20);
        assertEquals(k1.getBetrag(), 80.0,0.0);
    }

    public void testSub2() {
        // 100 - 20 = 80}
        k1.sub(20);
        assertEquals(k1.getBetrag(), 80.0 , 0.0001);
    }
}

```

# Testbeispiel: Projekt Junit\_bsp4 mit Konto

// Wird jeweils pro Test neu aufgerufen !!

```
protected void setUp() {  
    k1 = new Konto(100.0, "Müller");  
    k2 = new Konto(200.0, "Schmidt");  
    k3 = new Konto(300.0, "Meyer");  
    k4 = new Konto(400.0, "Schulze");  
}
```

// Wird jeweils pro Test am Ende aufgerufen !!

```
protected void tearDown() {  
  
}
```

# Testbeispiel: Projekt Junit\_bsp4 mit Konto

The screenshot shows the Eclipse IDE interface. The main editor displays the following Java code for `KontoTest.java`:

```
7  
8  
9 public class KontoTest {  
10  
11     private Konto k1;  
12     private Konto k2;  
13     private Konto k3;  
14     private Konto k4;  
15  
16  
17     @Test  
18     public void testAdd() {  
19         // ...  
20     }  
21 }  
22
```

The left sidebar shows the JUnit runner results for `KontoTest` (Runner: JUnit 4) with a duration of 0,000 s. The results are as follows:

Test Method	Duration (s)	Status
testAdd	0,000	Success
testPrint	0,000	Success
testSub1	0,000	Success
testSub2	0,000	Success
testNumerik	0,000	Failure
testSummen	0,000	Success

The bottom console shows the output of the failed test:

```
<terminated> KontoTest [JUnit] C:\Program Files (x86)\Java\jre6\bin\javaw.exe (17.12.2011 12:47:03)  
s1: 1000.0  
s2: 1000.0
```

The Failure Trace in the bottom left shows the error details:

```
java.lang.AssertionError: expected:<0.3000000000  
at KontoTest.testNumerik(KontoTest.java:52)
```

# Ablauf eines Testes

- Bestimmen der TestSuites
- Bestimmen der Testfälle (test\*, public, ())
- **@BeforeClass**
  - public static void setUpClass() throws Exception { }
- Pro Test:
  - Initialisierung mit setUp()
  - Aufruf des Testroutine
  - De-Initialisierung mit tearDown()
  - Beim Auftreten eines Fehlers ⇒ Abbruch des Einzel-Testes
- **@AfterClass**
  - public static void tearDownClass() throws Exception { }
- Ausgabe des Ergebnisses

## Was passiert, wenn JUnit die Tests dieser Klasse ausführt?

- Das Test-Framework durchsucht die Testklasse mit Hilfe des Reflection API nach öffentlichen Methoden, die mit test beginnen und weder Parameter noch Rückgabewert besitzen.
- JUnit sammelt diese Testfallmethoden in einer Testsuite und führt sie voneinander isoliert aus. Die Reihenfolge, in der Testfallmethoden vom Framework gerufen werden, ist dabei prinzipiell undefiniert.
- Damit zwischen einzelnen Testläufen keine Seiteneffekte entstehen, erzeugt JUnit für jeden Testfall ein neues Exemplar der Testklasse und damit eine frische Test-Fixture.

## Was passiert, wenn JUnit die Tests dieser Klasse ausführt?

- Der Lebenszyklus dieses Exemplars ist so gestaltet, dass vor der Ausführung eines Testfalls jeweils die setUp Methode aufgerufen wird, sofern diese in der Unterklasse redefiniert wurde.
- Anschliessend wird eine der test... Methoden ausgeführt.
- Nach der Ausführung des Testfalls ruft das Framework die tearDown Methode, falls diese redefiniert wurde, und überlässt das Test Objekt dann der Speicherbereinigung.
- Dieser Zyklus wird vereinfacht erklärt ab Schritt 3 solange wiederholt, bis alle Testfälle jeweils einmal ausgeführt wurden.

# Literatur

## **JUnit: Profi-Tipps**

Klaus Meffert

Entwickler.press

ISBN: 3-935042-76-0

2006

## **Softwaretests mit JUnit**

Verlag: Dpunkt Verlag; 2. Auflage, 2005

ISBN-10: 3898643255

ISBN-13: 978-3898643252

# Literatur

- Johannes Link

- UnitTests mit Java, dpunkt-Verlag, 2002

- Frank Westphal

- Testgetriebene Entwicklung, dpunkt-Verlag, 2004

- Elfriede Dustin

- Automated software testing, Addison-Wesley, 1999

- Alistair Cockburn:

- Agile software development, Addison-Wesley, 1999