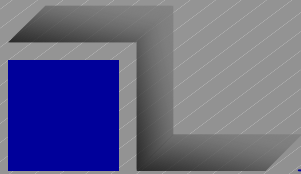


- Motivation
 - Test First
 - Design for Test
- Vorbereitung in Eclipse
- Testfall (eigentlich: Testklasse) zu einer Klasse erstellen
- Beispiel zur Demonstration
- Test-Szenario
- Prüf-Methoden
- [Diverse Ergänzungen](#)



Test First

Testen mit JUnit

- ▶ Motivation
- ▶ Eclipse-Vorbereitung
- ▶ Testfall erstellen
- ▶ Testklasse ausführen
- ▶ Demo-Beispiel
- ▶ Test-Szenario
- ▶ Prüf-Methoden
- ▶ Diverse
- ▶ ...
- ▶ ...
- ▶ ...
- ▶ ...

- "Immer dann, wenn Du in Versuchung kommst, etwas wie eine print-Anweisung oder einen Debugger-Ausdruck zu schreiben, schreibe das stattdessen als Test." [Martin Fowler]
- Beispiele
 - Während der Entwicklung: Schreibe vor dem Hinzufügen neuer Funktionalität zuerst Code, der diese neue Funktionalität benutzt
 - Bei der Fehlersuche, beim Debugging: Wenn ein Fehler entdeckt wurde schreibe Testcode, der den Fehler nachvollzieht, also zuerst fehlschlägt, und erst dann funktioniert, wenn der Fehler beseitigt wurde

Partner-Diskussion: Test First

Testen mit JUnit

- ▶ Motivation
- ▶ Eclipse-Vorbereitung
- ▶ Testfall erstellen
- ▶ Testklasse ausführen
- ▶ Demo-Beispiel
- ▶ Test-Szenario
- ▶ Prüf-Methoden
- ▶ Diverse
- ▶ ...
- ▶ ...
- ▶ ...
- ▶ ...

- Diskutieren Sie mit einem Partner
 - Welche Vorteile bietet es, wenn man Tests unmittelbar beim oder sogar vor dem Programmieren schreibt?
 - Welche Nachteile stehen dem gegenüber?
- Dauer: 3 Minuten



Folie 2

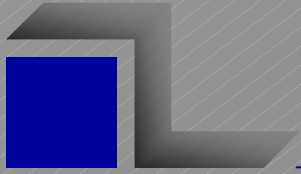
Testen mit JUnit

- ▶ Motivation
- ▶ Eclipse-Vorbereitung
- ▶ Testfall erstellen
- ▶ Testklasse ausführen
- ▶ Demo-Beispiel
- ▶ Test-Szenario
- ▶ Prüf-Methoden
- ▶ Diverse
- ▶ ...
- ▶ ...
- ▶ ...
- ▶ ...

- Design for Test bedeutet
 - Der Fokus beim Entwickeln liegt auf der Schnittstelle, nicht (nur) der Funktionalität
 - Der Code muss bequem aufrufbar sein
 - Der Code muss testbar sein

- "Möglicherweise ist der wichtigste Vorteil von all diesen Tests die Auswirkung auf Architektur und Design. Um ein Modul oder eine Anwendung testbar zu machen, muss sie geeignet entkoppelt sein. **Je testbarer** es ist, **desto modularer** (desto besser gekapselt) ist die Software.
Die Berücksichtigung umfassender Akzeptanz- und Unit-Tests hat einen deutlich positiven Effekt auf die Struktur der Software."

[Robert Martin, Agile Software Development. Prentice Hall 2003]



Testen mit JUnit

- ▶ Motivation
- ▶ Eclipse-Vorbereitung
- ▶ Testfall erstellen
- ▶ Testklasse ausführen
- ▶ Demo-Beispiel
- ▶ Test-Szenario
- ▶ Prüf-Methoden
- ▶ Diverse
- ▶ ...
- ▶ ...
- ▶ ...
- ▶ ...

Fragen?

- Demo
 - JUnit in Eclipse-Projekt einbinden
 - OO -> test.junit -> Calculations

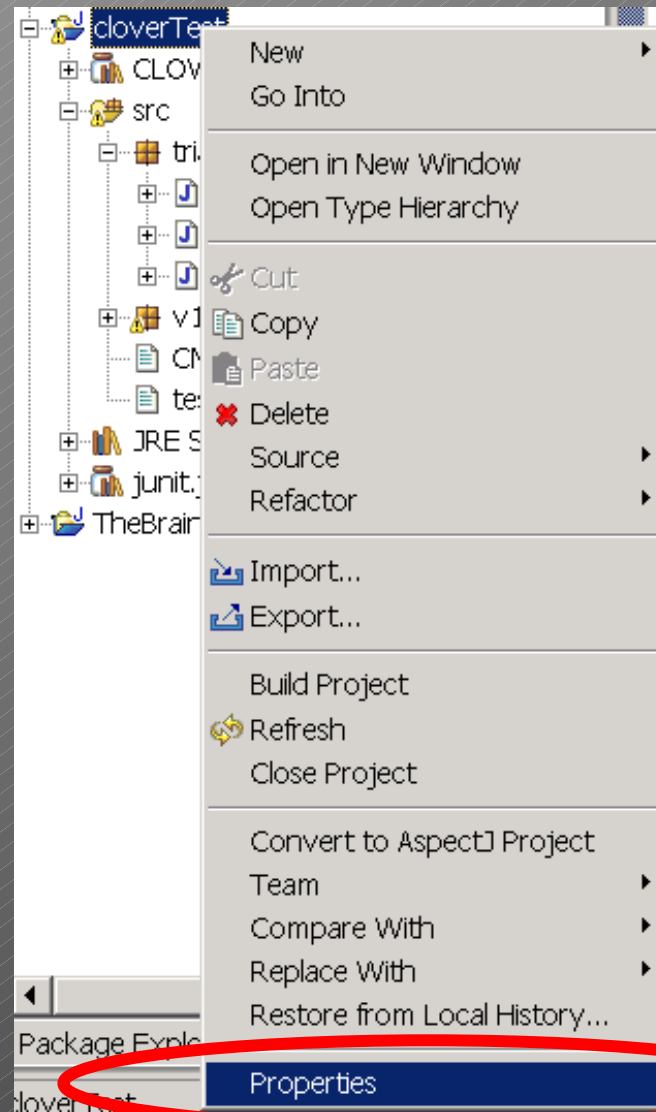


Folie 5

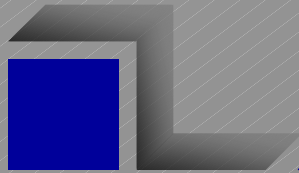
Vorbereitung in Eclipse: Bibliothek einbinden 1/3

Testen mit JUnit

- Motivation
- Eclipse-Vorbereitung
- Testfall erstellen
- Testklasse ausführen
- Demo-Beispiel
- Test-Szenario
- Prüf-Methoden
- Diverse
- ...
- ...
- ...
- ...



Project -> Properties

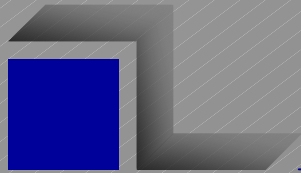


Vorbereitung in Eclipse: Bibliothek einbinden 2/3

Testen mit JUnit

- Motivation
- Eclipse-Vorbereitung
- Testfall erstellen
- Testklasse ausführen
- Demo-Beispiel
- Test-Szenario
- Prüf-Methoden
- Diverse
- ...
- ...
- ...
- ...

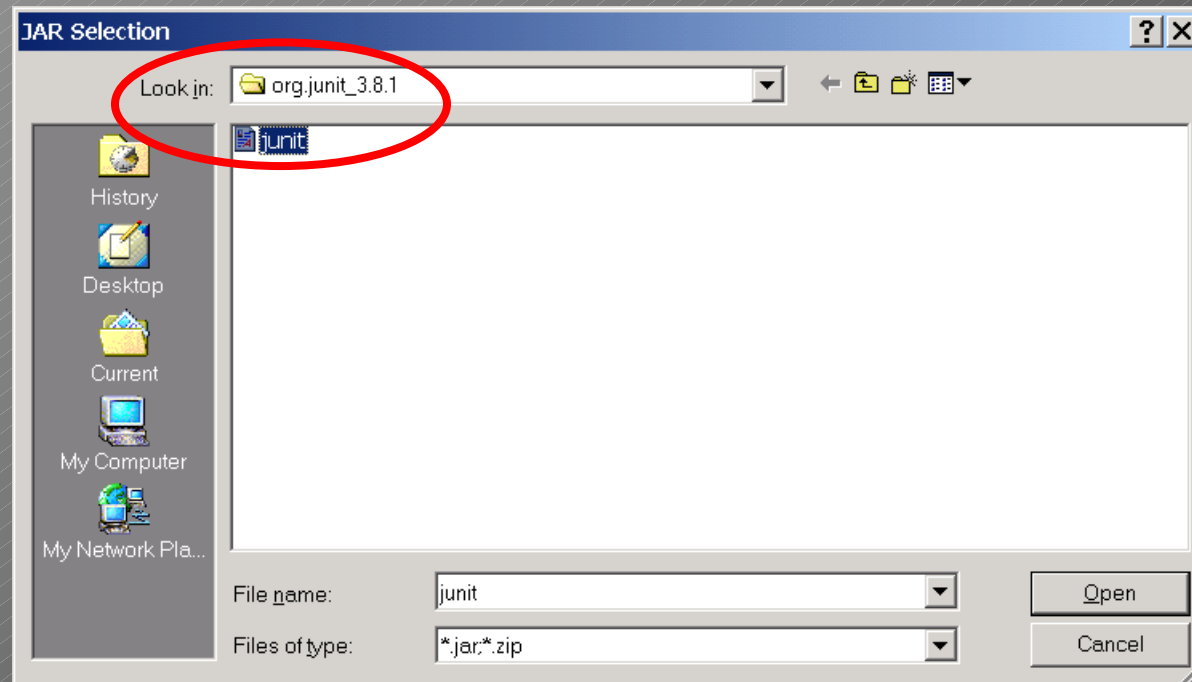
The screenshot shows the 'Properties for classes.ws02' dialog box in Eclipse. The 'Java Build Path' tab is active. The 'Libraries' section is selected, and the 'Add External JARs...' button is highlighted with a red circle. The 'Default output folder' is set to 'classes.ws02'. The 'JRE_LIB - C:\app\j2sdk1.4.0\jre\lib\rt.jar' is listed in the 'JARs and class folders on the build path:' section.



Vorbereitung in Eclipse: Bibliothek einbinden 3/3

Testen mit JUnit

- Motivation
- Eclipse-Vorbereitung
- Testfall erstellen
- Testklasse ausführen
- Demo-Beispiel
- Test-Szenario
- Prüf-Methoden
- Diverse
- ...
- ...
- ...
- ...



C:\Program Files\eclipse\plugins

Testfall zu einer Klasse erstellen (eigentlich: Test*klasse* mit Testfällen erstellen)

Testen mit JUnit

- Motivation
- Eclipse-Vorbereitung
- Testfall erstellen
- Testklasse ausführen
- Demo-Beispiel
- Test-Szenario
- Prüf-Methoden
- Diverse
- ...
- ...
- ...
- ...

The screenshot shows the Eclipse IDE interface. The Package Explorer on the left displays a project structure with a 'src' folder containing various Java files. The file 'TriangleV1.java' is selected, and a context menu is open over it. The 'New' option is expanded, showing a submenu with 'JUnit Test Case' highlighted. Other options in the 'New' submenu include Project..., Package, Class, Interface, Source Folder, Folder, File, Example..., and Other... (Ctrl+N). The main menu bar at the top includes File, Edit, Source, Refactor, Navigate, Search, Project, Run, Window, and Help. The toolbar below the menu bar contains icons for file operations and development actions.

Testfall zu einer Klasse erstellen: Name der Testklasse wählen

Testen mit JUnit

- ▶ Motivation
- ▶ Eclipse-Vorbereitung
- ▶ Testfall erstellen
- ▶ Testklasse ausführen
- ▶ Demo-Beispiel
- ▶ Test-Szenario
- ▶ Prüf-Methoden
- ▶ Diverse
- ▶ ...
- ▶ ...
- ▶ ...
- ▶ ...

New JUnit Test Case

JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify the class under test and on the next page, to select methods to be tested.

Source Folder:

Package:

Name:

Superclass:

Which method stubs would you like to create?

public static void main(String[] args)

Add TestRunner statement for:

setUp()

tearDown()

constructor()

Class Under Test:

< Back Next > Finish Cancel

Testfall zu einer Klasse erstellen: Methoden auswählen, die getestet werden sollen

Testen mit JUnit

- ▶ Motivation
- ▶ Eclipse-Vorbereitung
- ▶ **Testfall erstellen**
- ▶ Testklasse ausführen
- ▶ Demo-Beispiel
- ▶ Test-Szenario
- ▶ Prüf-Methoden
- ▶ Diverse
- ▶ ...
- ▶ ...
- ▶ ...
- ▶ ...

New JUnit Test Case

Test Methods

Select methods for which test method stubs should be created.

Available methods:

- TriangleV1
 - main(String[])
- Object

1 method selected.

Create final method stubs

Create tasks for generated test methods

< Back Next > Finish Cancel

Testfall zu einer Klasse erstellen: generierter Programmcode

Testen mit JUnit

Motivation

Eclipse-
Vorbereitung

Testfall
erstellen

Testklasse
ausführen

Demo-Beispiel

Test-Szenario

Prüf-Methoden

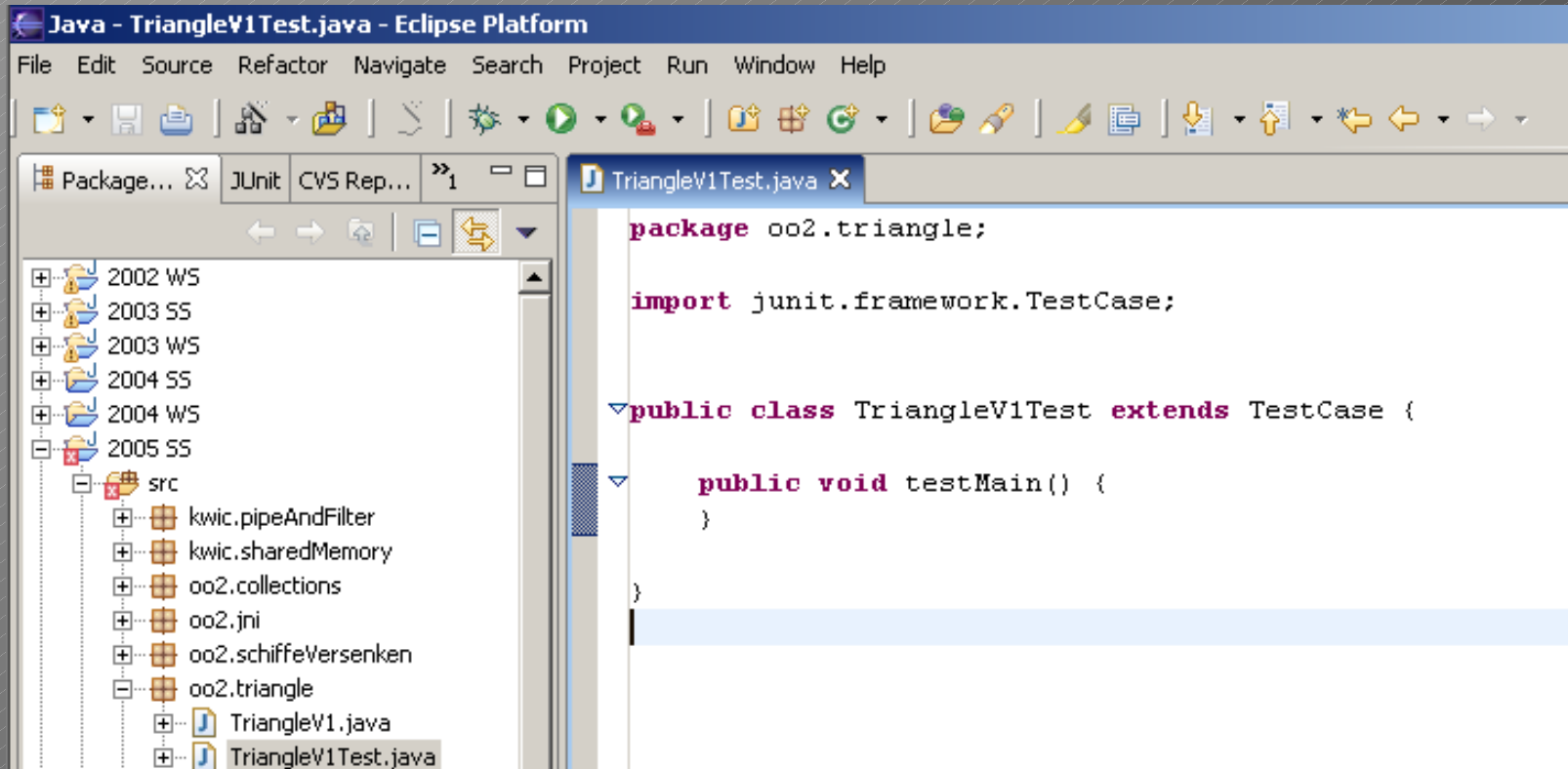
Diverse

...

...

...

...



The screenshot shows the Eclipse IDE interface. The title bar reads "Java - TriangleV1Test.java - Eclipse Platform". The menu bar includes "File", "Edit", "Source", "Refactor", "Navigate", "Search", "Project", "Run", "Window", and "Help". The toolbar contains various icons for file operations and development. The Package Explorer on the left shows a project structure with a "src" folder containing several packages and files, including "TriangleV1Test.java". The main editor window displays the following Java code:

```
package oo2.triangle;

import junit.framework.TestCase;

public class TriangleV1Test extends TestCase {

    public void testMain() {

    }

}
```

Testfall zu einer Klasse ausführen

Testen mit JUnit

- Motivation
- Eclipse-Vorbereitung
- Testfall erstellen
- Testklasse ausführen
- Demo-Beispiele
- Test-Szenario
- Prüf-Methoden
- Diverse
- ...
- ...
- ...
- ...

The screenshot shows the Eclipse IDE interface. The 'Run As' context menu is open, listing various execution options. The 'JUnit Plug-in Test' option is highlighted. The background shows a code editor with a Java class named 'Triangle' and a test class named 'TriangleV1Test' that extends 'JUnit4TestCase'. The test class has a 'testMain()' method that calls 'TriangleV1.main()' with a new String array.

```
// sch  
Triang  
// gl
```

Alle Methoden der Klasse, die als
`public void test...() { ... }`
deklariert sind,
werden nacheinander ausgeführt

Testfall zu einer Klasse ausführen: negatives / positives Ergebnis

Testen mit JUnit

- Motivation
- Eclipse-Vorbereitu
- Testfall erstellen
- Testklasse ausführen
- Demo-Beis
- Test-Szen
- Prüf-Metho
- Diverse
- ...
- ...
- ...
- ...

Package... JUnit CVS Rep... >>1

Finished after 0,01 s

Runs: 1/1 Errors: 0 Failures: 0

Failures Hierarchy

Kein Fehler gefunden

Failure Trace

JUnit Package Explorer

Finished after 0,06 seconds

Runs: 7/7 Errors: 0 Failures: 1

Failures Hierarchy

testCheckTriangle - oo2.triangle.TestableTriangleTest

7 Methoden ausgeführt

Kein Compiler-Fehler

1 fehlgeschlagene Testmethode

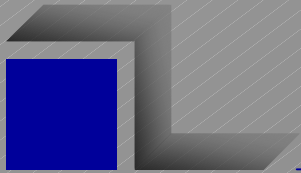
Ein Fehler gefunden

Fehler

Aufruf-Position; Doppelklick möglich

Failure Trace

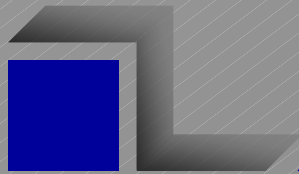
```
junit.framework.AssertionFailedError: expected:<5> but was:<3>  
at oo2.triangle.TestableTriangleTest.testCheckTriangle(TestableTriangleTest.java:26)  
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)  
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:39)  
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.jav  
at junit.framework.TestResult$1.protect(TestResult.java:106)
```



Testen mit JUnit

- ▶ Motivation
- ▶ Eclipse-Vorbereitung
- ▶ Testfall erstellen
- ▶ Testklasse ausführen
- ▶ Demo-Beispiel
- ▶ Test-Szenario
- ▶ Prüf-Methoden
- ▶ Diverse
- ▶ ...
- ▶ ...
- ▶ ...
- ▶ ...

Fragen?



Beispiel: eine zu testende Klasse mit zwei Methoden

Testen mit JUnit

- Motivation
- Eclipse-Vorbereitung
- Testfall erstellen
- Testklasse ausführen
- Demo-Beispiel**
- Test-Szenario
- Prüf-Methoden
- Diverse
- ...
- ...
- ...
- ...

```
package oo2.triangle;

public class Calculations {

    int square( int x ) {
        return x * x;
    }

    static double squareRoot( double x ) {
        return Math.sqrt(x) - 0.000000001;
    }

}
```

Nur zu Demo-Zwecken:
einmal dynamisch,
einmal statisch

- Demo
 - Programm zeigen
 - Testfälle erstellen und ausführen

Idee:
kleiner Rechenfehler,
wie sie bei float/double
manchmal vorkommen



Folie 16

Erstellen der Testklasse

Testen mit JUnit

Motivation

The screenshot shows the 'New' menu in an IDE. The 'JUnit Test Case' option is highlighted with a red circle. The menu also shows other options like 'Project...', 'Package', 'Class', 'Interface', 'Source Folder', 'Folder', 'File', 'Example...', and 'Other...'. The 'Calculations' package is selected in the project browser on the left.

The 'New JUnit Test Case' dialog box is shown. It has a 'Test Methods' section with the instruction 'Select methods for which test method stubs should be created.' Below this is a list of 'Available methods' with checkboxes. The 'Calculations' package is expanded, and 'square(int)' and 'squareRoot(double)' are checked. The 'Object' package is also expanded, showing various methods like 'Object()', 'getClass()', 'hashCode()', 'equals(Object)', 'clone()', 'toString()', 'notify()', 'notifyAll()', and 'wait(long)'. At the bottom, there are two unchecked checkboxes: 'Create final method stubs' and 'Create tasks for generated test methods'. The 'Finish' button is highlighted with a red arrow.

```
package oo2.triangle;

import junit.framework.TestCase;

public class CalculationsTest extends TestCase {

    public void testSquare() {

    }

    public void testSquareRoot() {

    }

}
```

Ausführen der Testklasse

```
package oo2.triangle;  
  
import junit.framework.TestCase;  
  
public class CalculationsTest extends TestCase {  
  
    public void testSquare() {  
        assertEquals(4, new Calculations().square(-2));  
        assertEquals(0, new Calculations().square(0));  
        assertEquals(4, new Calculations().square(2));  
    }  
  
    public void testSquareRoot() {  
  
    }  
  
}
```

Testen mit JUnit

- Motivation
- Eclipse-Vorbereitung
- Testfall erstellen
- Testklasse ausführen
- Demo-Beispiele
- Test-Szenario
- Prüf-Methoden
- Diverse
- ...
- ...
- ...
- ...
- ...

Context menu options:

- Run As
 - 1 Java Applet
 - 2 Java Application
 - 3 Java Bean
 - 4 JUnit Plug-in Test
 - 5 JUnit Test
 - 6 Run-time Workbench
- Run...
- Organize Favorites...

JUnit Test Results:

- Finished after 0,02 seconds
- Runs: 2/2
- Errors: 0
- Failures: 0

"Gründliches" Testen der Wurzel-Methode 1/2

Testen mit JUnit

Motivation

Eclipse-
Vorbereitung

Testfall
erstellen

Testklasse
ausführen

Demo-Beispiel

Test-Szenario

Prüf-Methoden

Diverse

...

...

...

...

Folie 19

Vorlesung Software Engineerin

```
public void testSquareRoot() {  
    assertEquals(2, Calculations.squareRoot(4), 0);  
    assertEquals(0, Calculations.squareRoot(0), 0);  
}
```

Keine Fehlertoleranz

Besser:

```
, 0.0001);  
, 0.0001);
```

JUnit Package Explorer
Finished after 0,01 seconds
Runs: 2/2 Errors: 0 Failures: 1
Failures Hierarchy
testSquareRoot - oo2.triangle.CalculationsTest

Failure Trace
junit.framework.AssertionFailedError: expected: <2.0> but was: <1.999999999>
at oo2.triangle.CalculationsTest.testSquareRoot(CalculationsTest.java:15)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.jav.
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessor
at junit.framework.TestResult\$1.protect(TestResult.java:106)

© Prof. Dr. Peter Knauber
HS Mannheim

"Gründliches" Testen der Wurzel-Methode 2/2

```
public void testSquareRoot() {  
    assertEquals(2, Calculations.squareRoot(4), 0.0001);  
    assertEquals(0, Calculations.squareRoot(0), 0.0001);  
  
    try {  
        Calculations.squareRoot(-4);  
        fail();  
    } catch (ArithmeticException e) {  
        // alles ok, wenn die Exception auftritt  
    }  
}
```

Testen mit JUnit

Motivation

Eclipse-
Vorbereitung

Testfall
erstellen

Testklasse
ausführen

Demo-Beispiel

Test-Szenario

Prüf-Methoden

Diverse

...

...

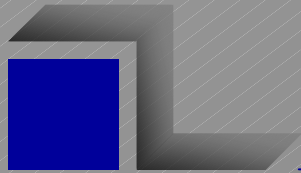
...

...

Folie 20

Vorlesung Software Engineering

JUnit Package Explorer
Finished after 0,02 seconds
Runs: 2/2 Errors: 0 Failures: 1
Failures Hierarchy
testSquareRoot - oo2.triangle.CalculationsTest
Failure Trace
junit.framework.AssertionFailedError
at oo2.triangle.CalculationsTest.testSquareRoot(CalculationsTest.java:20)
at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:42)
at junit.framework.TestResult\$1.protect(TestResult.java:106)



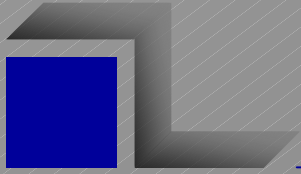
Verbesserter Code der Wurzel-Methode

Testen mit JUnit

- ▶ Motivation
- ▶ Eclipse-Vorbereitung
- ▶ Testfall erstellen
- ▶ Testklasse ausführen
- ▶ Demo-Beispiel

```
double squareRoot( double x ) {  
    if (x <= 0)  
        throw new ArithmeticException("Kein Wurzelziehen von Zahlen kleiner 0 möglich");  
    else  
        return Math.sqrt(x) + 0.00000001;  
}
```

- ▶ ...
- ▶ ...



Testen mit JUnit

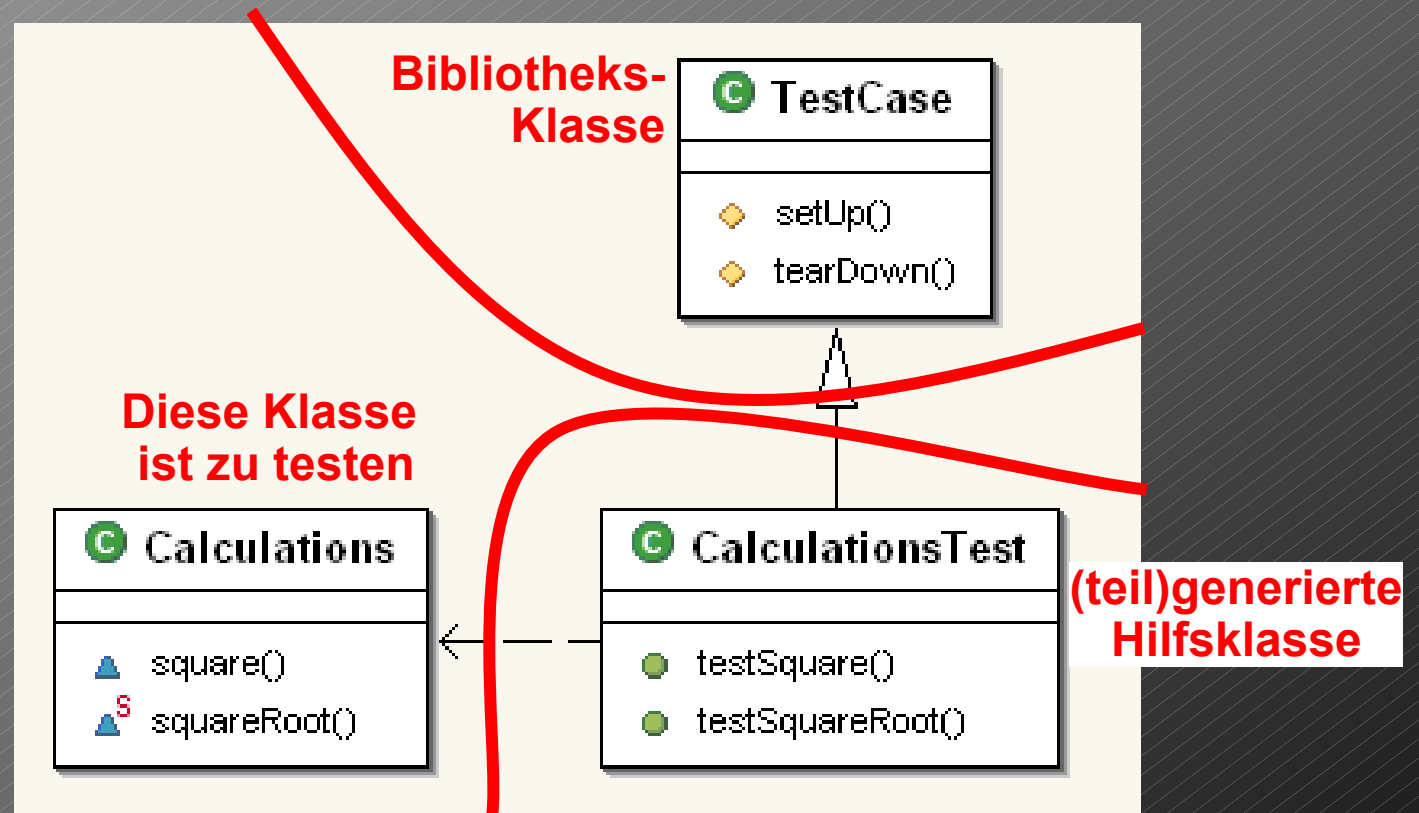
- ▶ Motivation
- ▶ Eclipse-Vorbereitung
- ▶ Testfall erstellen
- ▶ Testklasse ausführen
- ▶ Demo-Beispiel
- ▶ Test-Szenario
- ▶ Prüf-Methoden
- ▶ Diverse
- ▶ ...
- ▶ ...
- ▶ ...
- ▶ ...

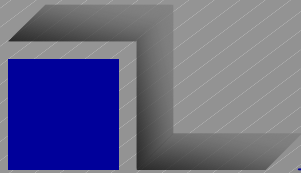
Fragen?

Test-Szenario

Testen mit JUnit

- Motivation
- Eclipse-Vorbereitung
- Testfall erstellen
- Testklasse ausführen
- Demo-Beispiel
- Test-Szenario**
- Prüf-Methoden
- Diverse
- ...
- ...
- ...
- ...



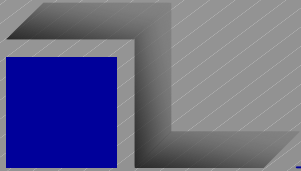


Prüfmethoden (geerbt von *TestCase*)

Testen mit JUnit

- Motivation
- Eclipse-Vorbereitung
- Testfall erstellen
- Testklasse ausführen
- Demo-Beispiel
- Test-Szenario
- Prüf-Methoden
- Diverse
- ...
- ...
- ...
- ...

- Konzept:
An bestimmten Stellen in einem (Test-)Programm soll sichergestellt (engl.: *assert*) werden, dass bestimmte Bedingungen erfüllt sind
- Die Klasse *TestCase* bietet viele Prüf-Methoden
 - *assertTrue(Boole'sche Bedingung)*
 - *assertFalse(Boole'sche Bedingung)*
 - *assertEquals(Soll-Wert, Ist-Wert)*; mögliche Typen: *boolean, byte, char, int, long, short; Object, String* (es wird mit *equals* verglichen)
 - *assertEquals(Soll-Wert, Ist-Wert, delta)*; mögliche Typen: *double, float*
 - *assertNull(Objekt)*
 - *assertNotNull(Objekt)*
 - *assertSame(Objekt1, Objekt2)*
 - *assertNotSame(Objekt1, Objekt2)*
 - *fail()*
- Ist eine der Prüfbedingungen nicht erfüllt, wird ein *AssertionFailedError* ausgelöst
- Alle Methoden existieren auch mit einem zusätzlichen ersten Parameter (Typ *String*) für eine (hilfreiche!) Meldung



Testen mit JUnit

- ▶ Motivation
- ▶ Eclipse-Vorbereitung
- ▶ Testfall erstellen
- ▶ Testklasse ausführen
- ▶ Demo-Beispiel
- ▶ Test-Szenario
- ▶ Prüf-Methoden
- ▶ Diverse
- ▶ ...
- ▶ ...
- ▶ ...
- ▶ ...

Fragen?

Testfälle entwerfen

Testen mit JUnit

Motivation

Eclipse-
Vorbereitung

Testfall
erstellen

Testklasse
ausführen

Demo-Beispiel

Test-Szenario

Prüf-Methoden

Diverse

...

...

...

...

- Jede *test...*-Methode sollte nur einen Testfall prüfen, aber den komplett:
 - Einen geeigneten Zustand vor der zu testenden Methode herstellen
 - Die zu testende Methode aufrufen
 - Das Ergebnis (= Rückgabewert) der Methode überprüfen
 - Den Zustand nach dem Methodenaufruf überprüfen
- Jeder Testfall sollte unabhängig vom Vorgänger funktionieren können
- Die *test...*-Methoden sollten geeignet benannt sein

Runs: 7/7

Errors: 0

Failures: 3

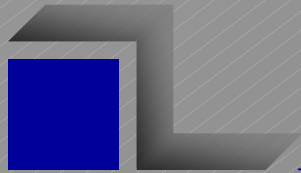
Failures

Hierarchy

testSeitenLängenKleinerNull - oo2.junit.DreieckTest

testZweiSeitenGleichDerDritten - oo2.junit.DreieckTest

testZweiSeitenKleinerDerDritten - oo2.junit.DreieckTest



Vor- und Nachbereitung fürs Testen

Testen mit JUnit

Motivation

Eclipse-
Vorbereitung

Testfall
erstellen

Testklasse
ausführen

Demo-Beispiel

Test-Szenario

Prüf-Methoden

Diverse

...

...

...

...

- Folgende Methode wird *vor jeder test...-Methode* ausgeführt:

```
protected void setUp() throws Exception
```

- Folgende Methode wird *nach jeder test...-Methode* ausgeführt:

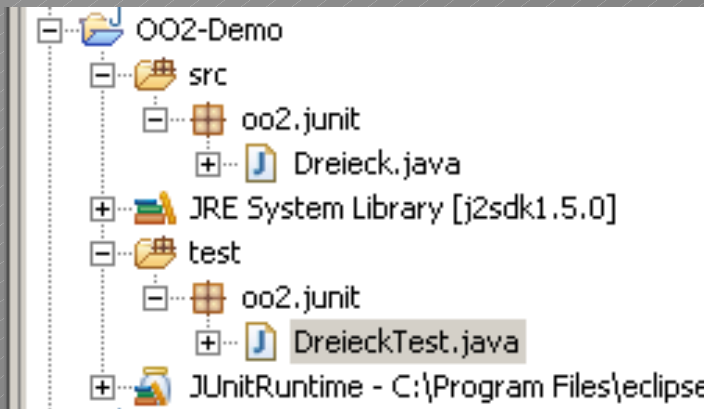
```
protected void tearDown() throws Exception
```

- Man sagt, es wird eine *Test-Fixture* auf- bzw. abgebaut

Trennen von Produktiv- und Test-Code

Testen mit JUnit

- Motivation
- Eclipse-Vorbereitung
- Testfall erstellen
- Testklasse ausführen
- Demo-Beispiel
- Test-Szenario
- Prüf-Methoden
- Diverse
- ...
- ...
- ...
- ...



- Gleiche Package-Struktur in beiden Ordnern
- Export JAR-Datei: nur die Packages (= diejenigen Package-Anteile) im src-Ordner exportieren

```
package oo2.junit;

import junit.framework.TestCase;

public class DreieckTest extends TestCase {
```

```
package oo2.junit;

public class Dreieck {

    static final int GLEICHSEITIG = 1;
    static final int GLEICHSCHENKLIG = 2;
    static final int SCHIEF = 3;
    static final int UNGÜLTIG = -1;
```