

Testen von Java Code mit **JUnit**

```
public void runTest() {  
    for (Enumeration e) {  
        if (e.hasMoreElements()) {  
            make  
            last  
        }  
    }  
}
```

Tests

Markus Wichmann

Demotivation...

Am Anfang war der Zeitdruck...

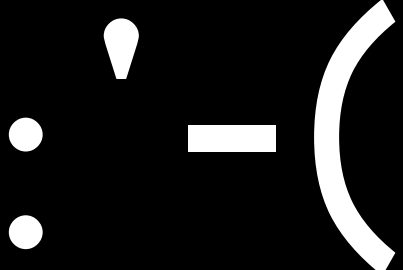
- **Hilfe, ich habe doch keine Zeit zum Testen!**
- **Ich schreibe einfach keine Tests, dadurch werde ich schneller fertig sein...**
(...und mein Chef wird mich loben)

Stunden oder Tage später...

- **Oh, meine Applikation wird instabil**
(...wie kann das sein? Ich bin ein guter Programmierer!)
- **Jetzt läuft mit die Zeit erst recht weg!**
- **Wie gut, dass ich meine Zeit nicht auch noch mit Tests verschwendet habe!**



Demotivation...



Motivation

- Das kennt Ihr doch:
 - Code schreiben
 - Applikation starten
 - Neue Funktionalität testen
 - Versuchen, Fehler zu provozieren
- Probleme dabei:
 - Starten der Applikation und Navigation zur neuen Funktionalität kostet Zeit
 - Neue Funktionalität nicht immer einfach testbar
 - Vergessen ist menschlich
 - **(sehr wichtig)** Testende Kollegen kennen die Fallstricke des Codes nicht!
- Wäre es nicht schön,...



Wäre es nicht schön, etwas zu haben,...

- das einen schnelleren Entwicklungszyklus ermöglicht?
- das das Starten der Applikation zunächst überflüssig macht?
- das keinen Test vergisst?
- **(sehr wichtig)** das andere im Team nachvollziehen und selbst testen können?
- das knallhart und ehrlich **ok** oder **nicht ok** sagt?
- was also gleichzeitig schnellere **und** bessere Entwicklung ermöglicht?

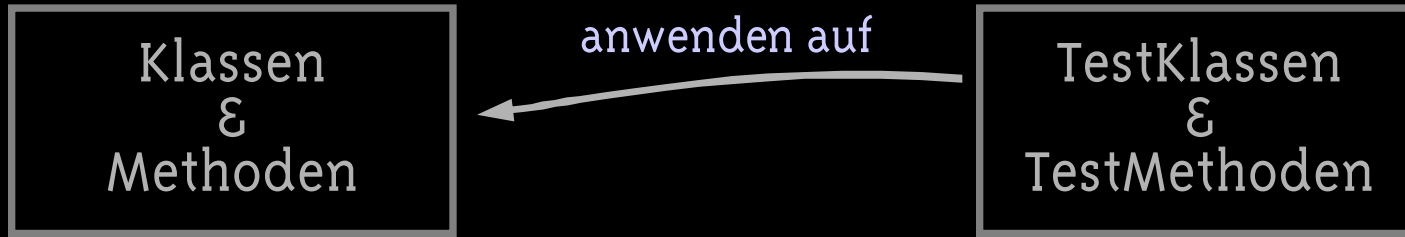


Unit Testing – Merkmale

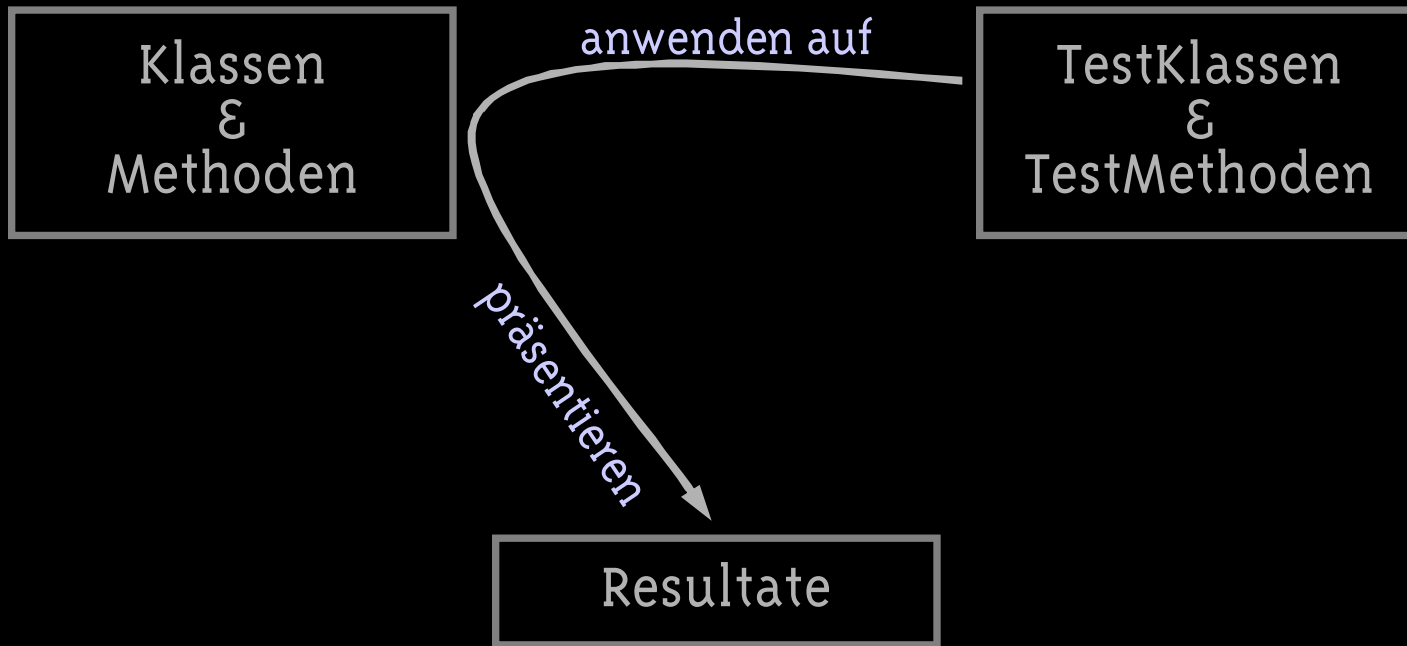
- erfolgt mit Test-Frameworks (hier: mit JUnit)
- sollte von Beginn des Projekts eingesetzt werden
- wird von IDEs unterstützt (siehe spätere Demo)
- ist ein iterativer Prozess
- arbeitet granular (testet einzelne Konstrukte)
 - Units sind meist Methoden und Klassen
- testet nicht die Gesamt-Funktionalität
- sind White Box Tests
 - Personen, die den Code sehen und ändern können, testen



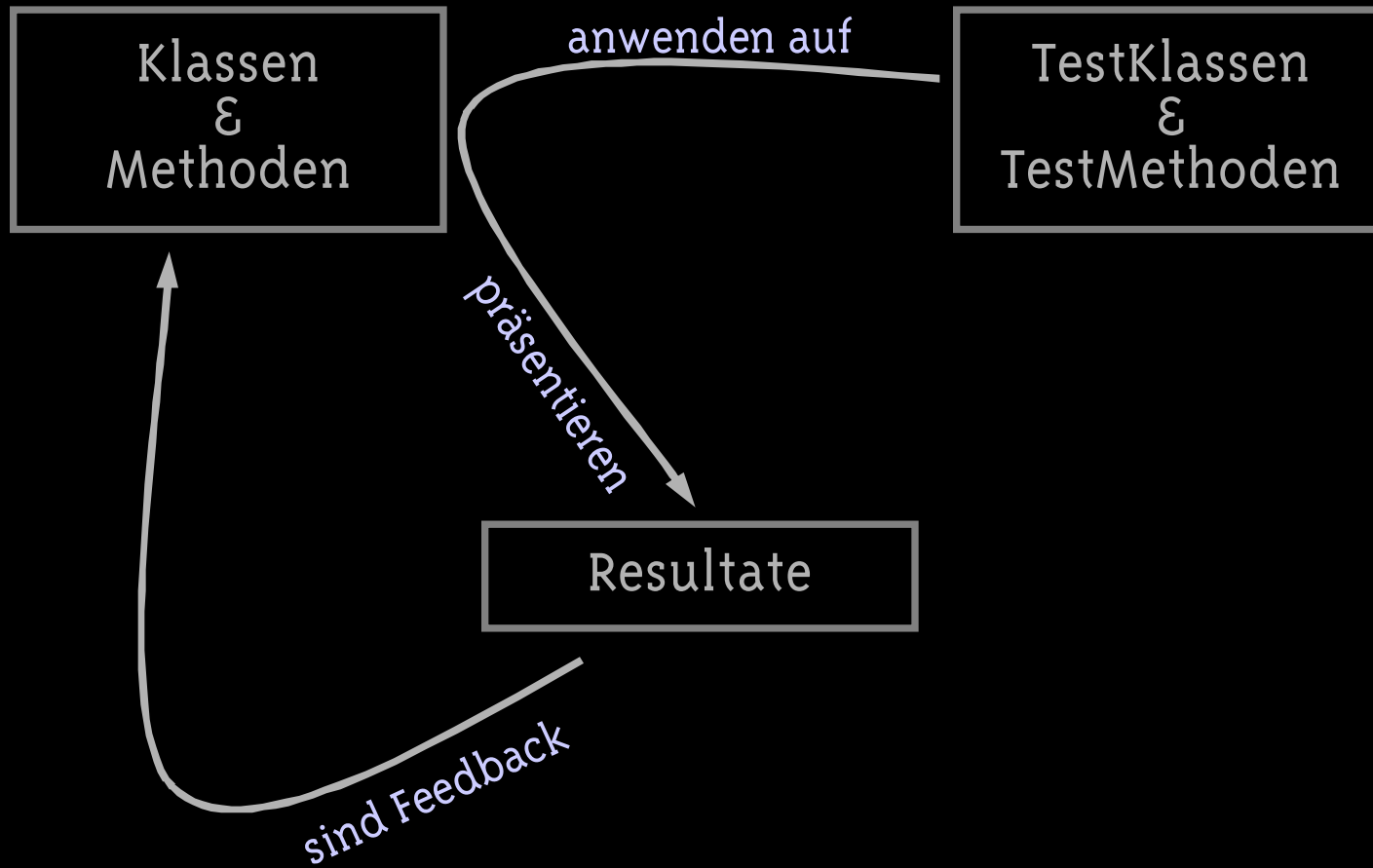
Unit-Testing: Ablauf



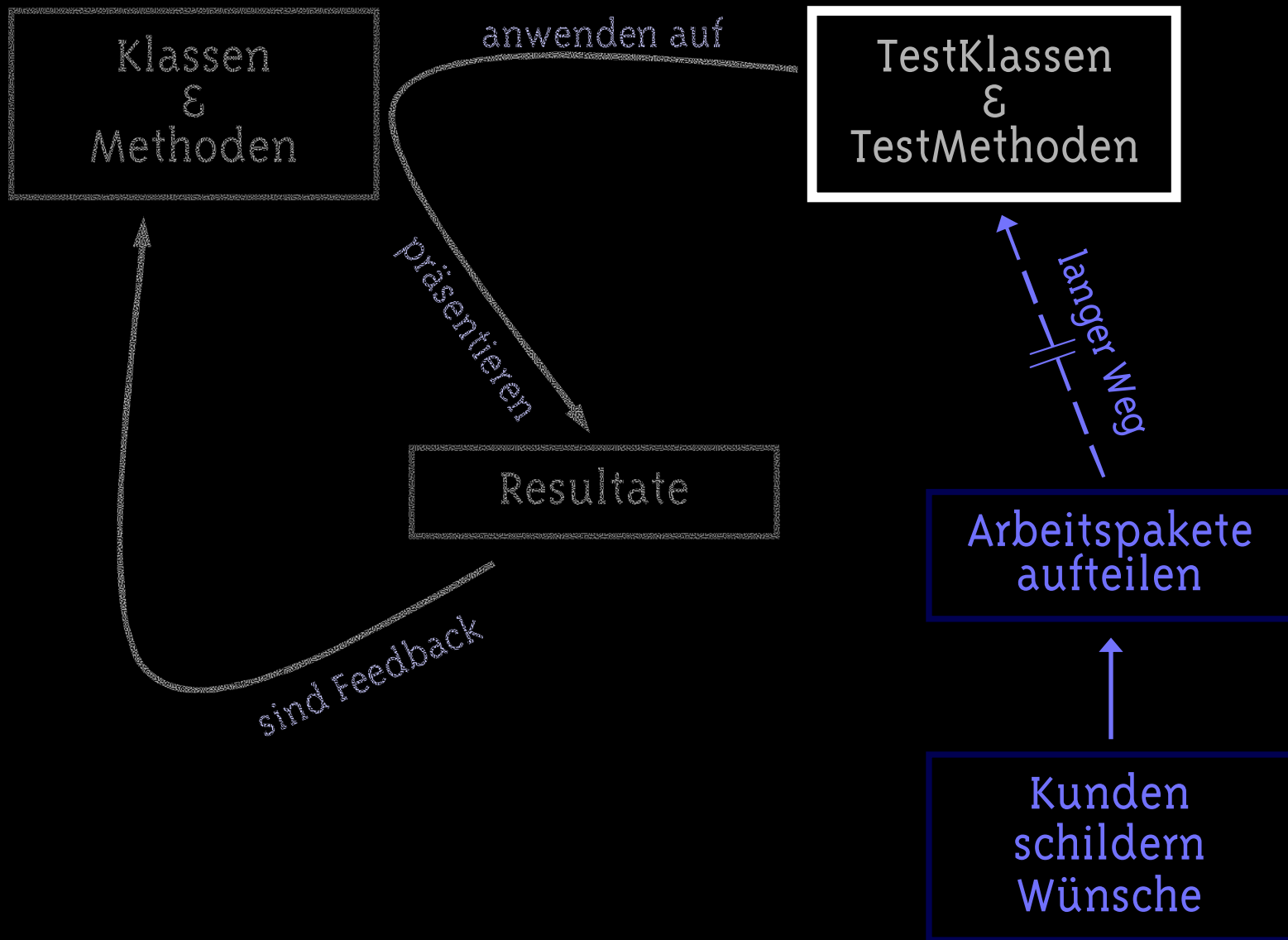
Unit-Testing: Ablauf



Unit-Testing: Ablauf



Henne oder Ei?



Ein paar Formalien...

- Es gebe eine Spezifikation **S** und ein Programm **P**
- Ein **Test Case**
 - ist ein Paar aus Eingabe- und Ausgabewert $(i, S(i))$ auf Spezifikations-Seite
- Ein **Test** ist ein implementierter Vergleich
 - $S(i) == P(i)$?
- Eine **Test Suite** ist eine Reihe von Tests/Test Cases, welche verschiedene Möglichkeiten abdeckt



Wir wollen Geld...

...durch unseren Code verwalten:

- erzeugen
- zu noch mehr Geld addieren

```
Money m12EUR= new Money(12, "EUR");  
Money m14EUR= new Money(14, "EUR");
```

```
Money result= m12EUR.add(m14EUR);
```

auf Vollständigkeit prüfen

```
Money expected= new Money(26, "EUR");  
Assert.assertTrue(expected.equals(result));
```



Der Geld-Test

```
public class MoneyTest extends TestCase {
    public void testSimpleAdd() {
        Money m12EUR= new Money(12, "EUR");
        Money m14EUR= new Money(14, "EUR");
        Money expected= new Money(26, "EUR");
        Money result= m12EUR.add(m14EUR);
        Assert.assertTrue(expected.equals(result));
    }
}
```

Gelingt der Test, ist add(...) richtig implementiert.

Scheitert er, so ist dies nicht der Fall.

Frage an Euch: Was wird passieren? 😊



JUnit Demo

Quelltexte als ZIP unter
m2w2.de/articles.html



JUnit Framework: Klasse Assert

- stellt in Test-Methoden statische Methoden für zu verifizierende Annahmen zur Verfügung:
 - `assertTrue(boolean)`
 - `assertFalse(boolean)`
 - `assertSame(Object, Object)`
 - `assertNotSame(Object, Object)`
 - `assertEquals(primitiveType, primitiveType)`
 - jeweils für 2 primitive Java-Typen



JUnit Framework: Klasse TestCase

- Spezialisierungen haben bestimmte Merkmale:
 - alle Methoden, deren Name mit `test...()` beginnt, werden vom Framework als Test-Methoden angesehen
 - Test-Methoden können automatisch aufgerufen werden (Reflection wird benutzt)
 - `junit.textui.TestRunner.run(MoneyTest1.class);`
 - `TestCase testB = new MoneyTest3("testEquals");`
`testB.run();`
 - Die Methode `suite()` wird gerufen, wenn vorhanden
- Überschreiben der Methoden `setUp()` und `tearDown()` zum Sammeln von gemeinsamem Code aller Tests

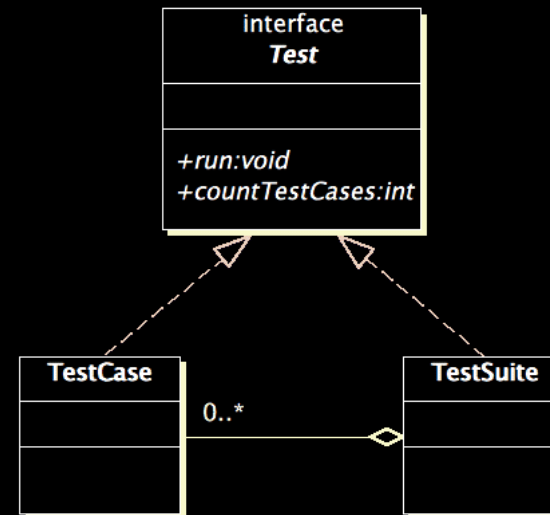


JUnit Framework: Klasse TestSuite

- Eine TestSuite besteht aus mehreren Tests
- Jede Klasse mit mehreren Test-Methoden kann als Suite benutzt werden

```
TestSuite aSuite = new  
    TestSuite(MoneyTest3.class);
```

- stellt Meta-Methoden zur Überwachung des Testablaufs zur bereit (VariousSuites.java)
 - countTestCases()
 - wasSuccessful()
 - ...



Weitere Vorteile von Unit Testing

- Codefremde Entwickler können codieren und sofort sehen, ob sie etwas kaputt gemacht haben
- Je einfacher ein Test zu machen ist, desto häufiger wird er eingesetzt
- Je häufiger er eingesetzt wird, desto schneller werden Fehler repariert



Noch mehr Vorteile

- Schreiben von Unit Tests zwingt zum Im-Auge-Behalten vom Zweck
 - der Methode, der Klasse, der Anwendung
- Granularität von Unit Tests animiert zum Entkoppeln von Methoden und Klassen
- Entwickler nehmen Standpunkt des Aufrufers ein
 - Code muss leicht aufrufbar sein
 - Code muss leicht testbar sein
- Nebeneffekt: Code des Unit Tests dient als Referenz zum Aufrufen des eigentlichen Codes
- Schreiben von Tests im voraus verbessert Design



Kritische Anmerkungen

- Unit Tests ungenügend um ein ganzes System zu testen
- Erfüllen von Kundenwünschen wird nicht getestet
- Schreiben von Tests für bestehenden Code schwierig
 - Stark gekoppelte Methoden und Klassen
 - Ver-Antipatternder Code ist schwer testbar (The Blob, Spaghetti-Code, Monolithic JSPs, ...)



Warum JUnit JavaDoc zu empfehlen ist...

- weil es weniger ist als es aussieht
- weil es Spaß macht ;-)

	<code>java.lang.Class theClass)</code>
<code>void</code>	<code>addTestSuite(java.lang.Class testClass)</code> Adds the tests from the given class to the suite
<code>int</code>	<code>countTestCases()</code> Counts the number of test cases that will be run by this test.
<code>static Test</code>	<code>createTest(java.lang.Class theClass, java.lang.String name)</code> ...as the moon sets over the early morning Merlin, Oregon mountains, our intrepid adventurers type...
<code>static java.lang.String</code>	<code>exceptionToString(java.lang.Throwable t)</code> Converts the stack trace into a string
<code>java.lang.String</code>	<code>getName()</code> Returns the name of the suite.
<code>reflect.Constructor</code>	<code>getTestConstructor(java.lang.Class theClass)</code> Gets a constructor which takes a single String as its argument or a no arg



Auch Lesenwert

Robert C. Martin, James W. Newkirk, Robert S. Koss:
Agile Software Development (ISBN 0-13-597444-5)

<http://www.extremeprogramming.org>

<http://www.junit.org> (Die JUnit WebSite)

www.martinfowler.com/articles/newMethodology.html

<http://ccm.redhat.com/doc/core-platform/5.0/engineering-standards/testing-standards.html>

<http://junit.sourceforge.net/doc/testinfected/testing.htm>

<http://www.xprogramming.com/software.htm> (Liste von Unit Test Frameworks für alle erdenklichen Sprachen)



```
Assert.assertTrue(esIstSchonSpaet);
```

**Vielen Dank
für Eure Aufmerksamkeit!**

