Namics. Conf.
Groovy – Workshop. Namics.

Heiko Maaß. Principal Software Engineer.
Max Trense. Software Engineer.

22. Monat 2014

Groovy:
"Like a super version of Java"

JVM-basiert

Einfacher Einstieg

Dynamische Sprache

Closures

Metaprogramming

Funktionale Programmierung

AST-Transformation

Groovy ist eine …

Dynamische Sprache

Erweiterung des Programms zur **Laufzeit**

Use Case
Gradle


```
apply plugin: 'idea'
apply plugin: 'groovy'

project.group = "com.medela.bst"

buildscript {
    repositories {
        mavenCentral()
    }
}

repositories {
    mavenCentral()
}

sourceSets {
    manualtest
}

configurations {
    manualtestCompile.extendsFrom compile
    manualtestRuntime.extendsFrom runtime
}

dependencies {
    compile 'org.codehaus.groovy:groovy-all:2.2.0'
    compile 'junit:junit:4.11'
    compile 'org.hamcrest:hamcrest-all:1.3'
    compile 'commons-httpclient:commons-httpclient:3.1'
    compile 'log4j:log4j:1.2.17'
    compile 'commons-lang:commons-lang:2.6'
    compile 'commons-io:commons-io:2.4'
```

Use Case
Unit + Integrationtests

```
@Test
public void testSignUpUser_minc() {
    SignUpModel signUpModel = signUpHelper.randomSignUp()
    signUpModel.market = "MINC_B2C"
    signUpHelper.postSignUp(signUpModel,
            {
                HttpResponse response, json ->
                    LogHelper.logJson(response, json)
                    assertThat response.statusLine.statusCode, is(200)
                    assertThat(json.code, is("00"))
            },
            {
                HttpResponse response, def reader ->
                    LogHelper.log(response)
                    println reader.text
                    fail("Invalid answer in signup")
            })

    assertSuccessfulSignup(signUpModel)
}
```

Use Case
Web Application Development

→ Convention over Configuration

→ Active Record (GORM)

→ Stark inspiriert von Ruby on Rails

Übungen

Übungen.

Theorie	Übungen
→ Grundlagen / Syntax	→ K1Syntax
→ String Handling	→ K2StringHandling
→ Groovy Beans	→ K3GroovyBeans
→ Operatoren	→ K4Operators
→ Closures	→ K5Closures
→ Collections	→ K6Collections
→ Metaprogramming	→ K7Metaprogramming
→ Builders	→ K8Builders
→ AST	→ K9AST

Idee: Lernen durch Unit-Tests

```
void test_03_rangeOperator() {
    def range = 0..1
    def character_range = 'A'..'B'
    // ------------ START EDITING HERE ----------------------

    // ------------ STOP EDITING HERE ----------------------
    assert range.size() == 5
    assert range[2] == 4

    // ------------ START EDITING HERE ----------------------

    // ------------ STOP EDITING HERE ----------------------
    assert character_range.contains('[')
    assert character_range.contains('A')
    assert ! character_range.contains('z')
}
```

Idee: Lernen durch Unit-Tests

```
void test_03_rangeOperator() {
    def range = 0..1
    def character_range = 'A'..'B'
    // ------------ START EDITING HERE ----------------------
    range = 2..6
    // ------------ STOP EDITING HERE -----------------------
    assert range.size() == 5
    assert range[2] == 4

    // ------------ START EDITING HERE ----------------------
    character_range = 'A'..<'z'
    // ------------ STOP EDITING HERE -----------------------
    assert character_range.contains('[')
    assert character_range.contains('A')
    assert ! character_range.contains('z')
}
```

Installationsanleitung

Übung

→ **Install IntelliJ CE (siehe /install/ide)**

→ **Install Gradle 2.0**

- `$ source setup.sh` / `$ setup.bat`
- Dabei wird Pfad auf `<gradle-2.0>/bin` gesetzt.

→ **Tests initial ausführen**

- `$ gradle clean test`

→ **Lösungen der Übungen entfernen**

- `$ gradle removeSolutions`
- `$ gradle test`

Grundlagen 1/6

→ **Groovy unterstützt Java-Syntax zu 99 %**

→ **Optional:**

- Variablen-Typ (stattdessen typlos durch `def`)
- Semikolon *
- Klammern bei Methodenaufrufen *
- `return` keyword *
- `public` keyword
- `throws` keyword

*** Nicht immer optional. Mehr dazu in der Übung**

Grundlagen 2/6

→ **Alles ist ein Objekt**

```
int i = 42
assert i.floatValue() == 42f
```

→ **Dynamische Methodenauflösung**

- Prüfung erst zur Laufzeit, nie zur Compilezeit

→ **Dynamische Typisierung**

- Typlose Variable mit `def`

Grundlagen 3/6
Truthy

→ **Collections**

```
if ([4,3,2]) {
   println "non-empty list resolves to true"
}
```

→ **Strings**

```
if ("hello") {
   println "non-empty string resolves to true"
}
```

→ **Numbers**

```
if (1) {
   println "non-zero numbers resolves to true"
}
```

Grundlagen 4/6
Gleichheit und Identität

`a == b`	`a.compareTo(b)` oder `a.equals(b)`
`a.is(b)`	`a == b`
Groovy	**Java**

Grundlagen 5/6
Literale für Listen und Maps

```
def list = [5, 6, 7, 8]
```

java.util.List

```
def map = [name: "Peter", uid: 10]
```

java.util.Map

Grundlagen 6/6
Flexible Switch Statements

```
switch (val) {
     case ~/^Switch.*Groovy$/:
          result = 'Pattern match'
          break
     case 60..90:
          result = 'Range contains'
          break
     case [21, 'test', 9.12]:
          result = 'List contains'
          break
     case { it instanceof Integer && it < 70 }:
          result = 'Closure boolean'
          break
     case String:
          result = 'Type equality'
          break
 }
```

K1Syntax

Übung 1

→ **Navigieren zu `/src/test/groovy/workshop`**

→ **Tests starten**

- Rechter Mausklick auf `K1Syntax`
- Run ‘K1Syntax’

→ **Tests fixen ;)**

String Handling 1/2

```
def s1 = "he said 'cheese' once"

def s2 = 'he said "cheese!" again'

def s3 = /he said "(\w){6}!" again/
```

Praktisch für reguläre Ausdrücke

String Handling 2/2

```
def amount = 100

def displayText = "Price: ${amount}"
```

groovy.lang.GString

Auflösung nur bei
doppelten Anführungszeichen

K2StringHandling

Übung 2

→ **Navigieren zu `/src/test/groovy/workshop`**

→ **Tests starten**

- Rechter Mausklick auf `K2StringHandling`
- Run ‘K2StringHandling’

→ **Tests fixen ;)**

Groovy Beans 1/2

```groovy
class Customer {
  Integer id
}
```

```java
public class Customer {
  private Integer id;
  public Integer getId() {
    return this.id
  }
  public setId(Integer id) {
    this.id = I
  }
}
```

GroovyBean ist equivalent zu JavaBean

Groovy Beans 2/2
Named Parameters in Constructor

```
def customer = new Customer(id: 3, name: "Peter")

assert customer.id   == 3
assert customer.name == "Peter"
```

K3GroovyBeans

Übung 3

→ **Navigieren zu `/src/test/groovy/workshop`**

→ **Tests starten**

- Rechter Mausklick auf `K3GroovyBeans`
- Run ‘`K3GroovyBeans`’

→ **Tests fixen ;)**

Operatoren Overloading

→ **Operatoren können für eigene Klassen überlagert werden**

```
a + b    ----------->    a.plus(b)

a[b]     ----------->    a.getAt(b)

a == b   ----------->    a.equals(b)
```

Operatoren
Safe Navigation ?.

→ **Macht vor Property-Zugriff einen `null`-Check**

→ **Vermeidet `if`-Verschachtelungen.**

```
def customer = new Customer()

Date created_at = customer?.cart?.created_at
```

Operatoren
Ranges

→ **Erzeugt eine Liste anhand von Anfangs- und Endwert**

```
5..8      ----------->  [5, 6, 7, 8]

5..<8     ----------->  [5, 6, 7]
```

K4Operators

Übung 4

→ **Navigieren zu `/src/test/groovy/workshop`**

→ **Tests starten**

- Rechter Mausklick auf `K4Operators`
- Run ‘`K4Operators`’

→ **Tests fixen ;)**

Closures 1/4
Definition

→ **Funktion mit Referenzen auf Variablen, die ausserhalb der Funktion deklariert worden sind.**

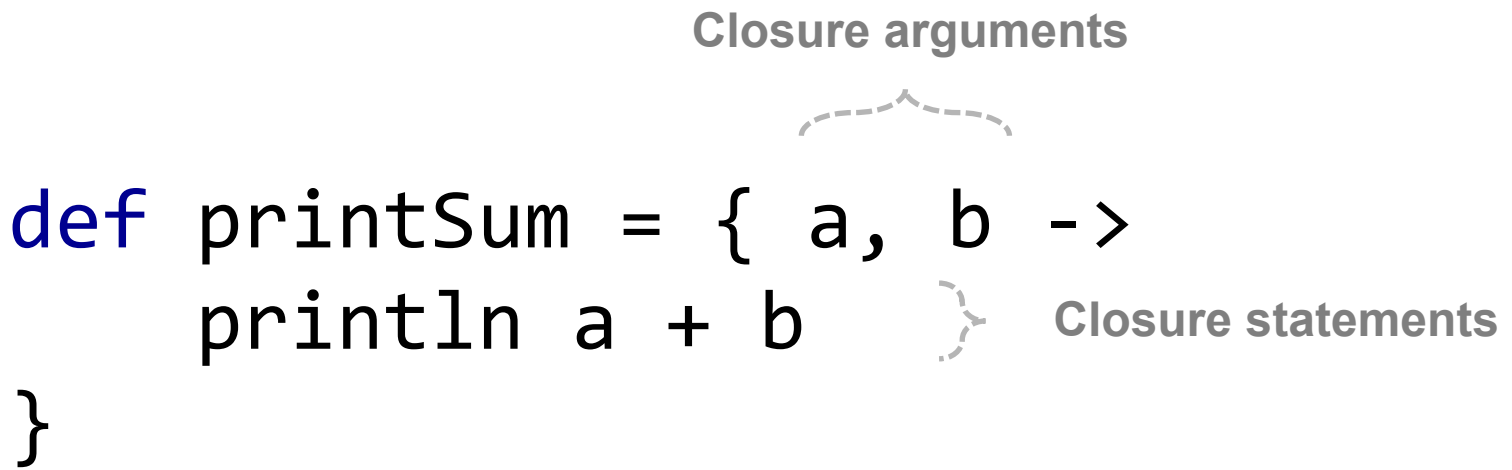
→ **Ein Closure kapselt seinen Erstellungkontext.**

```
def createClosure() {
  def i = 3
  return { println i }
}

def closure = createClosure()

closure() // prints 3
```

Closures 2/4
Syntax



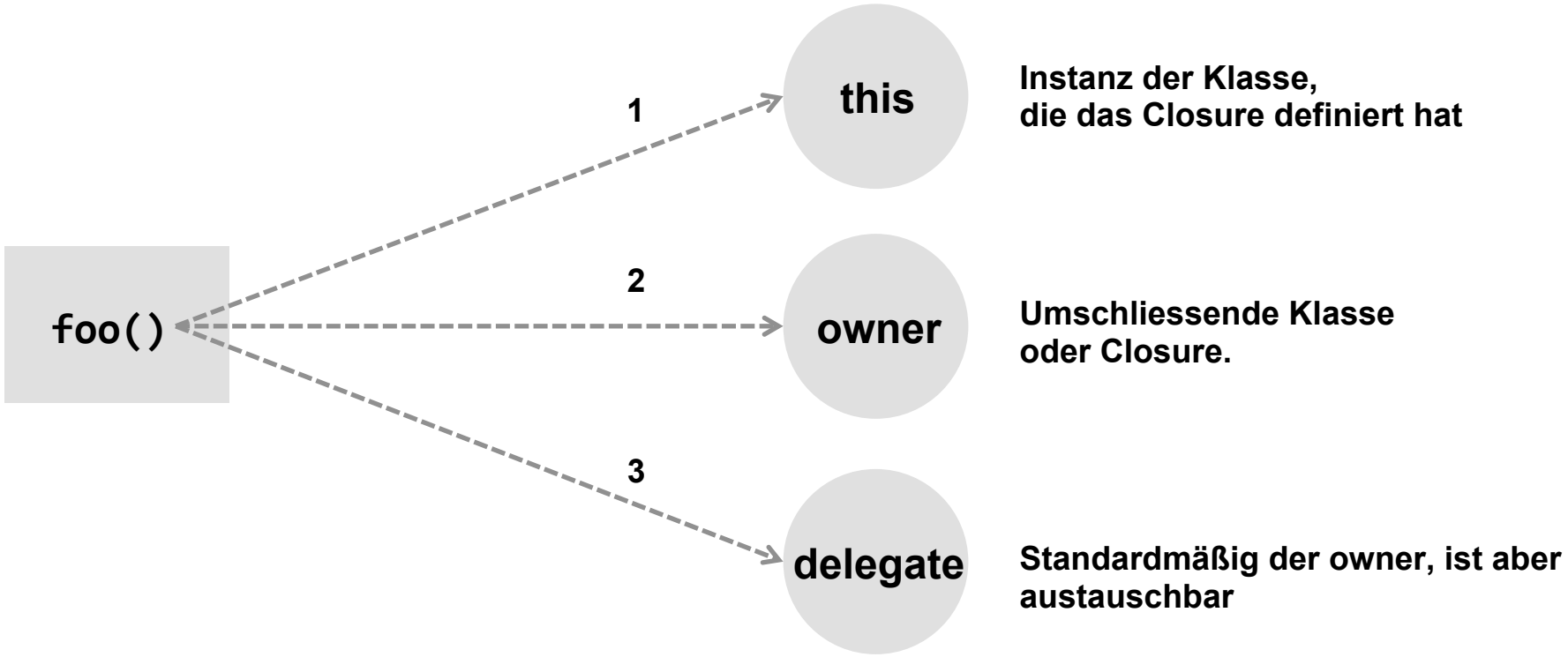
Closures 3/4
Implizite `it`-Variable

→ **Bei nur einer Variable braucht es keine Signatur**

```
{ v ->
  println v
}
```

```
{
  println it
}
```

Closures 4/4



Quelle: Programming Groovy 2

K5Closures

Übung 5

→ **Navigieren zu `/src/test/groovy/workshop`**

→ **Tests starten**

- Rechter Mausklick auf `K5Closures`
- Run ‘`K5Closures`’

→ **Tests fixen ;)**

Collections 1/2
Erweiterung durch Groovy

→ **Iteration mit `each`**

```
[4,3,2].each { print it }
```

------> **432**

→ **Finden mit `findAll`**

```
[4,3,2].findAll { it % 2 == 0}
```

------> **[4,2]**

Collections 2/2
Erweiterung durch Groovy

→ **Transformieren mit `collect`**

```
[4,3,2].collect { it * 2 }
```

------> **[8,6,4]**

→ **Zusammenführen mit `inject`**

```
[4,3,2].inject(0) { sum, i ->
   sum + (i * 2)
}
```

------> **18**

K6Collections

Übung 6

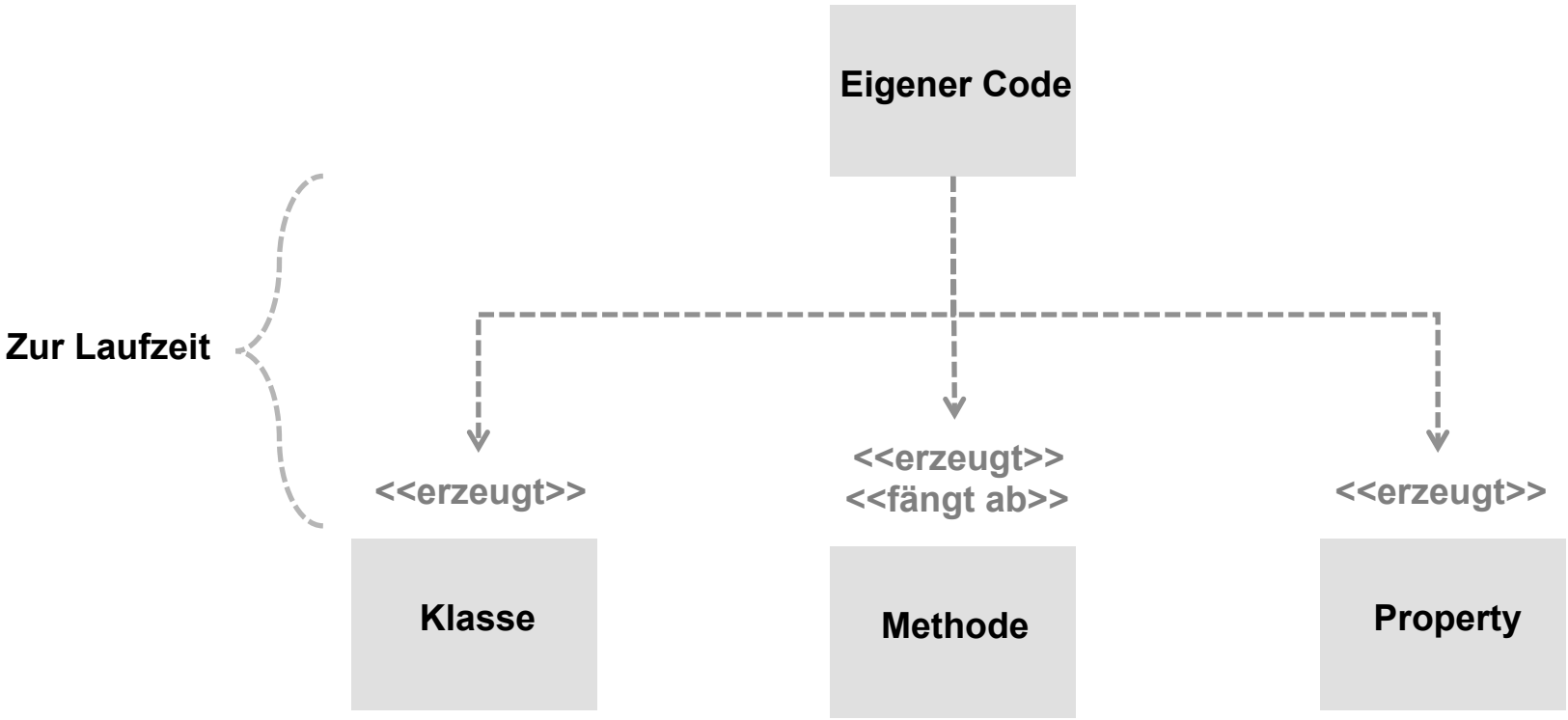
→ **Navigieren zu `/src/test/groovy/workshop`**

→ **Tests starten**

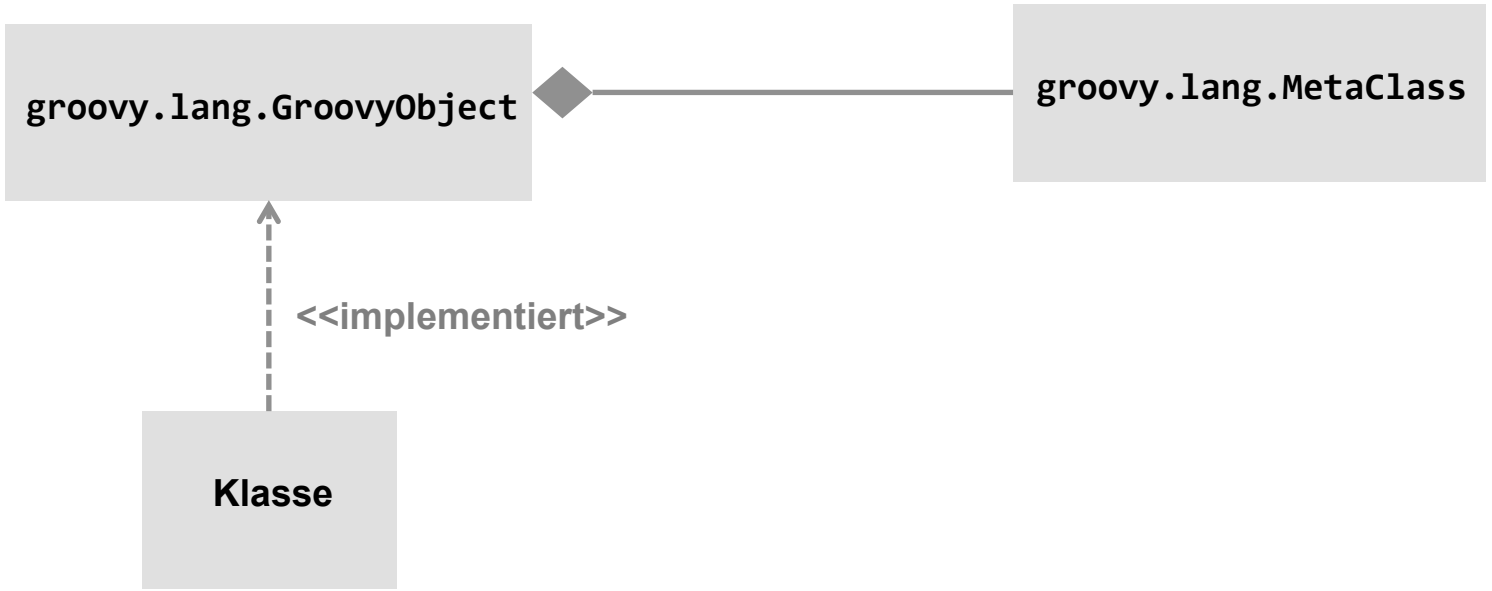
- Rechter Mausklick auf `K6Collections`
- Run ‘`K6Collections`’

→ **Tests fixen ;)**

Metaprogrammierung 1/4



Metaprogrammierung 2/4



Metaprogrammierung 3/4
Hooks

→ **Hooks für das Abfangen von Methodenaufrufen**

```
class A {
    def invokeMethod(String name, args) {
        // intercept existing / non-existing methods
    }

    def methodMissing(String name, args) {
        // intercept non-existing methods
    }
}
```

Metaprogrammierung 4/4
Klassen und Methoden zur Laufzeit erzeugen

→ **Neue Methoden und Properties: `metaClass`**

```
a.metaClass.fancyNewMethod = { ->
   println 'Wow, I was added at runtime'
}
```

→ **Neue Klassen: `Expando`**

```
def smartphone = new Expando
                 (manufactor: "Apple", deviceId: 5)

smartphone.ring = {
   println "ring...“
}
```

K7Metaprogramming

Übung 7

→ **Navigieren zu `/src/test/groovy/workshop`**

→ **Tests starten**

- Rechter Mausklick auf `K7Metaprogramming`
- Run ‘`K7Metaprogramming`’

→ **Tests fixen ;)**

Builder 1/1
XML erzeugen

→ **Einfaches Erzeugen von XML, JSON, HTTPRequests**

→ **Beispiel für XML:**

```
def w = new StringWriter()
def xml = new MarkupBuilder(w)

xml.records() {
  car(name:'HSV Maloo') {
    country('Australia')
  }
  car(name:'P50') {
    country('Isle of Man')
  }
}
```

```
<records>
  <car name='HSV Maloo'>
    <country>Australia</country>
  </car>
  <car name='P50'>
    <country>Isle of Man</country>
  </car>
</records>
```

K8Builder

Übung 8

→ **Navigieren zu `/src/test/groovy/workshop`**

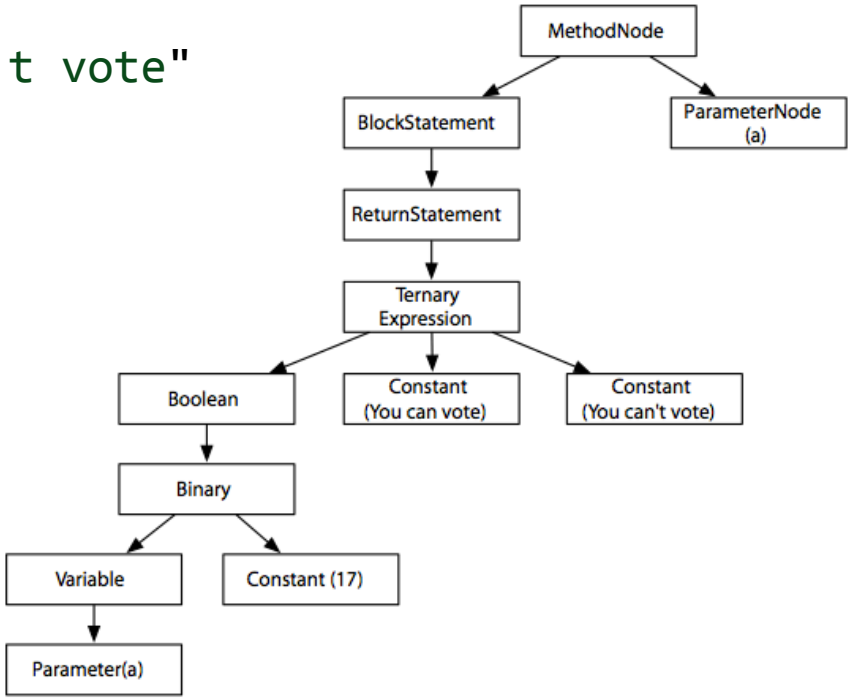
→ **Tests starten**

- Rechter Mausklick auf `K8Builder`
- Run ‘`K8Builder`’

→ **Tests fixen ;)**

Abstract Syntax Tree (AST) Grundlagen

```
def canVote(a) {
  a > 17 "You can vote" : "You can't vote"
}
```



→ **(Byte)code-Analyse**

→ **(Byte)code-Transformation**

- Framework-Integration
- Dry-ing code

Quelle: Programming Groovy 2

Abstract Syntax Tree (AST) Transformationen

→ **Annotationen lösen Transformation zur Compilezeit aus**

```
@Immutable final class Coordinates {
    Double latitude, longitude
}
```

`@Immutable` ----------→

+ equals/hashCode-Implementierung
+ toString-Implementierung
+ finale Felder
+ Kopieren der Rückgabewerte

Abstract Syntax Tree (AST) Transformationen

→ **@Immutable (Klasse)**

→ **@ToString (Klasse)**

- toString()-Methode aus den Feldern einer Klasse

→ **@Delegate (Feld)**

- Delegiert nicht implementierte Methoden an das Objekt

→ **@Singleton (Klasse)**

- Konstruktor → private
- T.instance liefert einziges Objekt zurück

→ **AST-Transformationen sind in Groovy implementiert**

K9AST

Übung 9

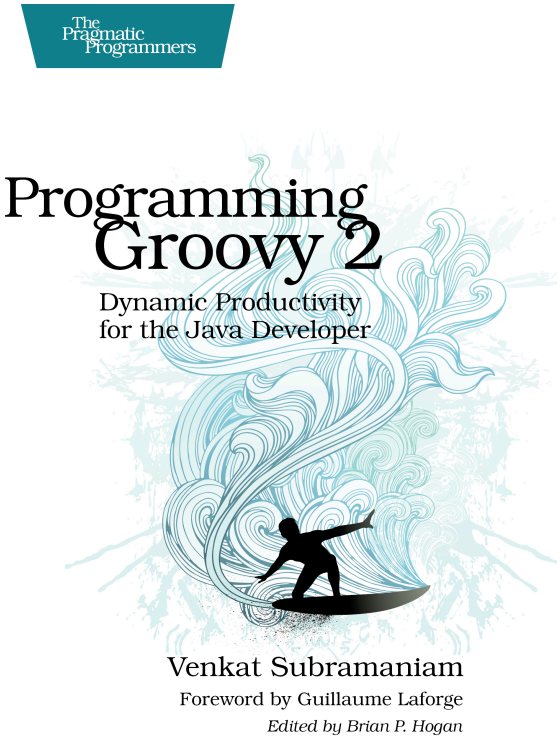
→ **Navigieren zu `/src/test/groovy/workshop`**

→ **Tests starten**

- Rechter Mausklick auf K9AST
- Run ‘K9AST’

→ **Tests fixen ;)**

Buchempfehlung



Vielen Dank.

heiko.maass@namics.com

max.trense@namics.com