Groovy – Eine Einführung

Dr. Joachim Baumann, Xinaris GmbH

Xinaris

Inhalte – Tour de Force

- Spracheigenschaften
- Anwendungsfälle
- Alles ist ein Objekt
- Ranges, Listen und Maps
- Kontrollstrukturen
- Operator-Überladung
- Closures: Funktionen höherer Ordnung
- Erweiterung von (Java-)Klassen durch Categories

Xinaris

Groovy

- Ist eine Programmiersprache
  - Skriptsprache
  - reguläre Programmiersprache
- Stammt von Java ab
- Ist eine **dynamische** Sprache
- Ist von Sun unterstützt im Java Community Process
  - JSR-241 (Java Specification Request)
- Wird auf der Standard-Virtual Machine von Java ausgeführt (Standard-Byte-Code)

Ein Beispiel (Java)

```
public class HelloWorld  {

   public static void main(String [ ] args)  {

    System.out.println(„HelloWorld“);

   }

}
```

- Einfachstes existierendes Java-Programm

Ein Beispiel (Groovy)

Anmerkung: Auch das Beispiel auf der vorherigen Seite ist korrekter Groovy-Code

Anwendungsfälle

- Programmierung von Skripten anstelle von
  - Sh, Perl, Python, Ruby …
- Unit-Tests
  - Deutlich einfachere Formulierung der Testfälle
  - Unterstützung für Mock-Objekte
- Eigenständige Applikationen
  - Frameworks: Grails als Web-Applikations-Framework
- Integration in großen Applikationen
  - Als „Glue-Sprache“
  - Gleichwertig zu Java z.B. in Spring
  - Als Konfigurationssprache
  - Als Domain-Specific Language

Spracheigenschaften

- Sehr Java-ähnliche Syntax
  - damit leichte Erlernbarkeit
- Java-Objektmodell wird verwendet
- Sämtliche Java-Klassen können verwendet werden
  - Hohe Anzahl von existierenden Bibliotheken
- Übersetzte Groovy-Klassen können von Java verwendet werden
- Integration in beide Richtungen
- Starke und schwache Typisierung (je nach Bedarf)

Alles ist ein Objekt

- In Java gibt es eine Unterscheidung zwischen primitiven Datentypen und Objekten
  - Ursprünglicher Grund: Laufzeitprobleme
  - Führt zu vielen Problemen bei der Programmierung
- Groovy hat nur Objekte
  - Auch Zahlen sind Objekte

```
5.times { println „Hallo Welt“ }
```

Hallo Welt
Hallo Welt
Hallo Welt
Hallo Welt
Hallo Welt

Typisierung ist optional

- „Duck Typing“: Bekannt aus Skriptsprachen
- Erleichtert die Programmierung
- Der Typ einer Variable muss nicht angegeben werden
- Aber: Groovy unterstützt explizites Typing (im Zweifel hinter den Kulissen)

```
def a = 1
```
← a ist vom Typ Integer

Ranges

- Definieren eine Menge von Elementen innerhalb eines Bereiches

```
1..5
```
← Bereich von 1 - 5

- Ranges sind Objekte

```
def r = 1..<5
```
← Bereich von 1 - 4

- Über die Elemente kann iteriert werden

```
r.each { n -> println n }
```
←
```
1
2
3
4
```

- Klassen, die Comparable sind und die Methoden next() und previous() implementieren, können in Ranges verwendet werden.

Listen

- Vereinfachte Notation (erweitert den Java-Typ List)

```
def list = [1, 2]
```

- Hinzufügen und Entfernen von Werten:

```
list += 3
```
→ [1, 2, 3]

```
list = list - [2]
```
→ [1, 3]

- Über die Elemente kann iteriert werden

```
list.each { n ->println n }
```
→ 1
3

Listen: Subskript-Operationen

- Subskript-Operationen werden unterstützt

```
def list = [1, 3]
list [1..1] = [1, 2, 3, 4]
println list
```
→ [1, 1, 2, 3, 4]

```
list [1..2] = [ ]
println list
```
→ [1, 3, 4]

Maps

- vereinfachte Notation (erweitert den Java-Typ Map)

```
def map = [ „name“: „Groovy“, „typ“: „Sprache“ ]
```

- Zugriff auf Elemente

```
map.each { entry -> println „$entry.key: $entry.value“}

map [„name“] = „English“
```

name: Groovy
typ: Sprache

- Verkürzte Zugriffsform

```
println map.name
```

English

Kontrollstrukturen (1)

- For-Schleife ähnlich wie in Java 5 (alte Variante wird unterstützt)

```
for ( i in 1..5 )
    println „Hallo $i“
```

Hallo 1
Hallo 2
Hallo 3
Hallo 4
Hallo 5

- Alle normalen Kontrollstrukturen von Java
- Boolesche Entscheidungen allgemeiner durch „Groovy Truth“

Kontrollstrukturen (2)

- Switch deutlich allgemeiner als in Java
  (Beispiel mit Typ, Range und Closure)

```
switch ( val ) {
   case Number:    println „$val ist eine Zahl“
   case 5..9:      println „$val ist im Bereich 5 – 9“
   case { it > 3 }:    println „$val ist größer 3“
}
```


Operator-Überladung (1)

- Gibt es in Java nicht
- Existiert in Groovy, da es nur Objekte gibt
- Für viele existierende Java-Klassen bereits zur Verfügung gestellt
- Funktioniert für alle gängigen Operatoren durch Implementierung der entsprechenden Methode
  Beispiel:
  Operator + Methode plus()
- Eigene Implementierungen sehr einfach

Operator-Überladung (2)

```
class Complex {
   BigDecimal r
   BigDecimal i
   def plus (Complex c) {
        new Complex (r : this.r + c.r, i : this.i + c.i)
   }

}

def res = new Complex(r:1, i:-1)
                    + new Complex (r:1, i:1)

println „$res.r + $res.i i“
```

2 + 0 i

Closures: Funktionen höherer Ordnung

- Idee: Methoden sind Objekte
  - bekannt aus funktionaler Programmierung
  - Funktionen erster (und höherer) Ordnung
- „Anonyme innere Klassen auf Steroiden“
- Aspekte
  - Trennung von Iteration und Logik
  - Ressourcen-Handling zentral
  - Funktionale Programmierung

Closures: Definition (1)

- Wir haben (anonyme) Closures schon kennengelernt

```
map.each { entry -> println „$entry.key: $entry.value“}
```

```
case { it > 3 }:   println „$val ist größer 3“
```

- Definition allgemein:

```
def c = { p1, p2, p3 = 0 ->

   [ Quelltext der Closure]

}
```

Parameter können Default-Werte erhalten
Wenn keine Parameter, dann implizit Parametername **it**

Resultat der letzten Anweisung ist Return-Wert (alternativ return)

Closures: Definition (2), Beispiel

```
def square = { x -> x * x }

println square(2)

r = 1..5

r.each { println square(it) }


def cubed = { x -> x * square (x) }

r.each { println cubed(it) }
```

4

1
4
9
16
25

impliziter Parameter it

1
8
27
64
125

Closures: Funktionale Programmierung (1)

- „Currying (auch: Schönfinkeln) ist die Umwandlung einer Funktion mehrerer Argumente in mehrere Funktionen einzelner Argumente. Das Verfahren ist benannt nach Haskell Brooks Curry (obwohl es schon vorher von Moses Schönfinkel und Gottlob Frege erfunden wurde), nach dem auch die funktionale Programmiersprache Haskell benannt ist.“ – *Wikipedia*
- Bewirkt die Vorbelegung einzelner Parameter mit spezifischen Werten
- Erlaubt auf elegante Weise die Verknüpfung von Funktionen miteinander

Closures: Funktionale Programmierung (2)

```
def mul = { x, y -> x * y }
def mul2 = mul.curry(2)
def mul5 = mul.curry(5)
println mul2(3)
println mul5(2)

def komp = { g, f, x -> g( f(x) ) }
def mul10 =  komp.curry(mul2, mul5)
println mul10(5)
```

entspricht { y -> 2 * y }

entspricht { y -> 5 * y }

6

10

g∘f (x) := g(f(x))

Currying mit Closures

50

Erweiterung von Klassen durch Categories

- Klassen, die nicht unter eigener Kontrolle sind, benötigen trotzdem manchmal zusätzliche Funktionalität
- Categories fügen existierenden Klassen **zur Laufzeit** neue Methoden hinzu

```
static class HalloCategory {
    public static hallo(String i) {  return "Hallo, $i"}
}
def testString = "kleiner Prinz"
use(HalloCategory) {
    println testString.hallo()
}
```

Hallo, kleiner Prinz

GDK – Erweiterungen der Standard-Klassen

- Aktuell über 400 Methoden, die die Standardklassen des JDK erweitern
- Gleichen Versäumnisse im JDK aus
- Sorgen für intuitive Verwendbarkeit des GDK
- Beispiel Klasse java.io.File
  (in Summe 40 neue Methoden)
  - eachDir(Closure closure)
  - eachFile(Closure closure)
  - eachFileMatch(Object filter, Closure closure)
  - eachFileRecurse(Closure closure)
  - …

Groovy: Hoher Abstraktionsgrad

- Sprache hat höheren Abstraktionsgrad als Java
- Zunehmende Abstraktion ermöglicht gesteigerte Produktivität
- Deshalb kann Groovy produktiver genutzt werden als Java

- Gegenargument: Laufzeitgeschwindigkeit ist niedriger
- Aber

	Java / C++
1995	1:100 – 1:1000
heute	häufig 1:1

Was haben wir ausgelassen?

- GPath
- Unit-Tests und Mock-Objekte
- XML-Behandlung (siehe Anhang)
- Groovy SQL
- Meta Object Protocol (siehe Anhang)
- Grails, GORM
- GString (siehe Anhang)
- Bean-Unterstützung (siehe Anhang)
- Reguläre Ausdrücke (siehe Anhang)
- Builder-Unterstützung (siehe Anhang)
- …

Weitere Informationen

- Website

  http://groovy.codehaus.org

- Special Interest Group “SIG Groovy” der JUGS

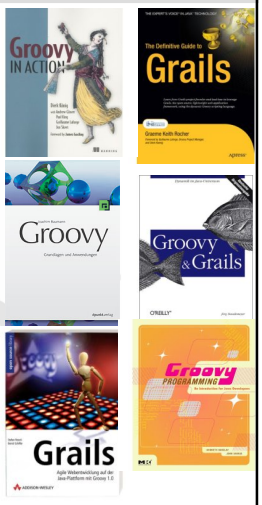
  http://www.jugs.de/sig-groovy.html

- Englisches Buch (ab Oktober in deutscher Übersetzung)
  „Groovy in Action“, *Dierk König*, Manning Publishing

  http://www.manning.com/koenig

- Deutsches Buch
  “Groovy – Grundlagen und Anwendungen”
  erscheint 4. Quartal des Jahres 2007 im dpunkt-Verlag

  http://www.groovybuch.de



Vielen Dank für Ihre Aufmerksamkeit

Haben Sie Fragen?

Email-Adresse
Dr. Joachim Baumann joachim.baumann@xinaris.de

Historie der Sprache, Standardisierung

- James Strachan und Bob McWhirter sind die Erfinder
- 2003: James Strachan und seine Frau warteten am Flughafen auf einen verspäteten Flug
  - Er sah sich die Sprache Python an
  - Überlegung: Wie wäre es, wenn man die gleiche Eleganz wie in Python in Java hätte?
- In Folge wiederholt zu Bob: „Wäre es nicht **groovy**, wenn das und das ginge?“
- 2004: Standardisierung mit dem JSR-241
- Aktueller Projektleiter Guillaume LaForge
- Version 1.0 ist am 3. Januar 2007 erschienen

Groovy Strings

- Zeichenketten in Groovy können Referenzen auf Variablen enthalten
  - Diese werden erst zum Auswertungszeitpunkt eingesetzt

```
date = new Date(12, 5, 23)
text = "Datum: "
gstring = "$text $date"
println gstring
date.date += 15324
text = “GString ändert sich nicht"
println gstring
```

Datum: Sun Jun 23 00:00:00 CET 1912

Datum: Mon Jun 07 00:00:00 CET 1954

Meta Object Protocol (1)

- Normalerweise bietet ein Objekt eine feste Schnittstelle von Methoden (und Attributen) an.
- Was passiert, wenn das Objekt von Mal zu Mal entscheiden kann, wie es einen Methodenaufruf interpretiert?
- Erlaubt einem Objekt, reflektiv zu operieren
- Damit sinnvolle Reaktion auf Aufrufe möglich, die zur Implementierungszeit unbekannt sind

Meta Object Protocol (2): Einfaches Beispiel

- Reagieren auf eine nicht implementierte Methode (bekannt aus Smalltalk)

wird aufgerufen für alle nicht implementierten Methoden

```
class MOPBeispiel {
    public invokeMethod(String method, Object params) {
     println „Ich wurde aufgerufen mit $method“
    }

}

bsp = new MOPBeispiel()

bsp.helloWorld()
```

Ich wurde aufgerufen mit helloWorld

XML-Verarbeitung: Eine XML-Datei (1)

```
<groovy><music>
  <tune>
    <title>The 59th Street Bridge Song</title>
    <artist>Simon and Garfunkel</artist>
  </tune>
  <tune>
    <title>Monday Monday</title>
    <artist>The Mamas and The Papas</artist>
  </tune>
```


XML-Verarbeitung: Eine XML-Datei (2)

```
  <tune>
    <title>Goodnight Moon</title>
    <artist>Shivaree</artist>
  </tune>
  <tune>
    <title>Suddenly</title>
    <artist>Soraya</artist>
  </tune>
</music></groovy>
```

XML-Verarbeitung: Das Groovy-Skript

XML-Verarbeitung: Die Ausgabe

The 59th Street Bridge Song, Simon and Garfunkel

Monday Monday, The Mamas and The Papas

Goodnight Moon, Shivaree

Suddenly, Soraya

Umgang mit Beans (1): Java

```
public class JavaBean implements Serializable {

    private int meinAttribut;

    public int getMeinAttribut ( ) {

     return meinAttribut;

    }

    public void setMeinAttribut (int neuerWert) {

     this.meinAttribut = neuerWert;

    }

}
```

Bean-Eigenschaft
entsteht aus
-Attribut*
-Getter
-Setter

* vereinfacht

Umgang mit Beans (2): Groovy

```
class GroovyBean {

    int meinAttribut

}
```

- Setter und Getter werden automatisch bereitgestellt
- Zugriff aber normalerweise in vereinfachter Form

```
def gb = new GroovyBean ( meinAttribut : 1 )

println gb.meinAttribut
```

Vereinfachte Initialisierung

Vereinfachter Zugriff

1

Reguläre Ausdrücke (1)

- Vereinfachte Notation (angelehnt an Python oder Perl)
- „=~“ erzeugt Matcher-Objekt

```
def text = "Groovy ist cool"

def m = text =~ /\w+/

println m[0]
println m[1]
println m[2]
```

Groovy
ist
cool

Reguläre Ausdrücke (2)

- Matcher erlaubt Zugriff auf Gruppen (Teile des regulären Ausdrucks mit Klammern) über Feldindex

```
def text = "Groovy ist cool"

def m = text =~ /\w(\w+)/

println m[0][1]
println m[1][1]
println m[2][1]
```

roovy
st
ool

Builder-Unterstützung

- Erbauer (Builder)
  „Trenne die Konstruktion eines komplexen Objekts von seiner Repräsentation, so dass derselbe Konstruktionsprozess unterschiedliche Repräsentationen erzeugen kann.“
  - *Entwurfsmuster, Gamma et al.*
- Builder erlauben die Erzeugung von HTML, XML, Ant Tasks oder auch Swing-GUIs
- Verwendung ist immer gleichartig
- Builder-Objekt bietet (Builder-)Methoden an mit Parametern
  - Map (Argumente)
  - Closure(s)

Builder-Unterstützung: Beispiel (1)

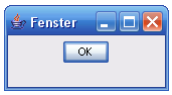
```
import groovy.swing.SwingBuilder

def sb = new SwingBuilder()

def closeOp = javax.swing.WindowConstants.EXIT_ON_CLOSE

def gui = sb.frame(title:'Fenster', size:[60, 80],
                   defaultCloseOperation:closeOp) {
    panel() {
        button(actionPerformed:{
            println „Ich wurde geklickt mit ${it}“
            }, text:'OK')
    }

}

gui.visible = true
```

Parameter

Innere Closure

Äußere Closure

Builder-Unterstützung: Beispiel (2)



Ich wurde geklickt mit
java.awt.event.ActionEvent[ACTION_PERFORMED,cmd=OK,when=1158135503891,
modifiers=Button1] on
javax.swing.JButton[,54,5,47x23,alignmentX=0.0,alignmentY=0.5,border=com.sun.java.swing.plaf.windows.XPStyle$XPEmptyBorder@f892a4,flags=296,maximumSize=,minimumSize=,preferredSize=,defaultIcon=,disabledIcon=,disabledSelectedIcon=,margin=javax.swing.plaf.InsetsUIResource[top=2,left=14,bottom=2,right=14],paintBorder=true,paintFocus=true,pressedIcon=,rolloverEnabled=true,rolloverIcon=,rolloverSelectedIcon=,selectedIcon=,text=OK,defaultCapable=true]

…