Vorlesungsevaluierung

- zu viel Theorie – zuviel Übungen / Meilensteine
  - 3 Stunden Theorie langweilig
- Vorlesungsausfall
- Mehr Hilfestellung zu Meilensteinen (zuviel selbst ergoogeln)
- zu schnell gesprochen / mehr Beispiele – zu langsam
- zu laut / Vorlesung in anderem Raum
- Katze
- keine Hilfe bei Meilensteinen
- zuviele Technologien

Vorlesungsevaluierung

- Themen bauen nicht aufeinander auf
- zuviel geschmipft / kein Verständnis
  - „ich will nicht ausgelacht werden“
- Mehr Freiheiten bei den Übungen
- uninteressante Themen
- eigenen Laptop mitbringen
- zuviel Ausfall, dann alles auf einmal
- Ersatzvorlesung mit Studierenden absprechen

Webanwendungen
Vorlesung – Hochschule Mannheim

Groovy

Inhaltsverzeichnis

Einführung

Warum noch eine Sprache?

Für kleinere Systeme

- “lasche” Syntaxvorschriften
- Dynamische Typisierung

Für große Systeme

- Typsicherheit (statische Typisierung)
- Gute Integration mit vorhandenen Bibliotheken und Frameworks
- Performance
- Testbarkeit

Warum noch eine Sprache?

Anforderung	PHP	Java	Servlet	Groovy	Groovy + Grails
Integration CSS, JS	+	-	0	-	++
Formularverarbeitung	++	-	0	-	++
Einlernaufwand	+	+	+	++	++
Listen, Assoziative Arrays (Maps)	++	0	0	++	++
Datenbankprogrammierung	+	+	+	++	++
Programmiereffizienz	+	0	0	++	++
Skriptfähigkeit	++	--	--	++	++
Lasche Syntaxvorschriften	++	--	--	+	+
Dynamische Typisierung	++	-	-	++	++
Typsicherheit	-	++	++	+	+
Integration	0	++	++	++	++
Performance	0	++	+	0	0
Testbarkeit	0	++	+	++	++

Groovy

- 2003 konzeptioniert von James Stracham
- Weiterentwickelt von „The Codehaus“
- Aktuelle Version: 2.4.3 – 23.März 2015
- Objektorientierte Skriptsprache
- Teilweise statisch und dynamisch typisiert
- .groovy Dateien werden vor dem interpretieren in Java-Bytecode übersetzt
- Engine: JVM

Beispiel HelloWorld

HellowWorldJava.java

```
public static void main(Sring[]args){
System.out.printlln(“Hello“ + (args.length>0 ? args[0] : “unknow“));
}
```

HelloWorldGroovy.groovy

```
println “Hello ${args.length>0 ? args[0] : ‘unknow‘}!“
```

Grundlagen

Syntax

- Kommentare: // bzw /*…..*/
- Stringkonkatenation: ‘a‘ + ‘b‘
- Strichpunkte am Zeilenende können entfallen:
  - foo = 11
- Variablendefinition:

  def foo

  - dynamische Typisierung
  - Wird die Variable direkt initialisiert wird das *def* nicht benötigt

– foo = 11

Datentypen

- byte
- short
- int
- long
- java.lang.BigInteger
- float
- double
- java.lang.BigDecimal
- java.lang.String.
- java.util.ArrayList
- java.util.List

Groovy Wahrheit (Boolean)

Datentyp	false	True
Boolean	false	True
Ganzzahl	0	Alles != 0
Fließkommazahl	0.0	Alles != 0.0
Character	0	Alle Unicodes != 0
String	Leerer String “”	Mind. 1 Zeichen
Collection	Leere Collection	Mind. 1 Element
Assoziatives Array	Leeres Array	Mind. 1 Element
Matcher	Nichts gefunden	gefunden
Objekte	null	!null

Bedingte Abfragen

- Bedingte Abfragen mit if
  - Ausdruck braucht kein Boolean zu liefern
  - Wie in Java

- Bedingte Abfragen mit switch-case
  - Switch Variable darf auch String sein
  - In den Case Zweigen sind auch erlaubt:
  - case 0…9: //range
  - case [8,9,19]: //Liste mit möglichen Werten
  - case Float: //Datentyp
  - case ~[A-Z].*/ //Pattern Matching

Schleifen

- while—Schleife
- for(initialisierung; fortsetzungbedingung; imkrementer)
- for(element in iterable)
- break
- continue

Funktionen

- Ähnlich zu Java:
  - int max(int x, int y) {return x>y ? x : y;}
- Defaultwerte in der Methodensignatur möglich:
  - int max(int x , int y = 11)
- Parameterdatentyp durch *def* ersetzbar oder ganz fehlen:
  - int max(def x)
  - int max(x)
- Rückgabetyp darf durch *def* ersetzt werden:
  - def square(x)

Funktionen

- Benannte Parameter
  - Zuordnung formale – aktuelle Parameter über ihren Namen
  - Genau ein formaler Parameter vom Typ *Map*

```
def begruessung(Map paran){
def gruss =“““
if(param.vorname) gruss += “$param.vorname“
if(param.nachname) gruss += “$param.nachname“
if(param.ort) gruss += “aus $param.ort“
Return gruss
}
println begrussung(vorname:“Martina“) //Martina
println begrussung(nachname:“Kraus“) //Kraus
println begrussung(vorname:“Martina“, nachname:“Kraus“) //Martina Kraus
println begrussung(vorname:“Martina“, ort:“Berlin“) //Martina aus Berlin
```

Funktionsaufruf

Klammern können bei parameterbehafteten Aufrufen entfallen

- Beispiel: *System.out.println “Hello world“* statt

  System.out.println (“Hello world“);

Bei parameterlosen Aufrufen dürfen Klammern nicht entfallen

Bei verschachtelten Aufrufen dürfen nur die äußersten Klammern entfallen:

- erlaubt: *System.out.println Math.max(3,4)*
- nicht erlaubt: *System.out.println Math.max 3,4*

Bei Bildschirmausgaben darf *System.Out weggelassen werden.:*

println “Hello world“

Vergleichsoperatoren

== testet auf inhaltliche Gleicheit

- entspricht der equals Methode (für Strings, Listen …)
- Testet nicht auf Gleichheit der Referenzen

!= testet auf inhaltliche Ungleicheit

- entspricht der equals Methode (für Strings, Listen …)
- Testet nicht auf Gleichheit der Referenzen

<=> vergleicht zwei Objekte inhaltlich miteinander

- entspricht der compare Methode
- -1 links größer als recht, 1 rechts größer als links, 0 gleich groß

<, <=, >, =>

- funktioniert auch für String (lexikalische Reihenfolge)

Vergleich von Referenzen

- Vergleich der Referenz mit der Methode *is(Object o)* der Klasse *Object*
  - liefert *true*, wenn beide Objekte an der selben Speicheradresse liegen, sonst *false*

```
def c1=new ComplexNumber(1,2)
def c2=new ComplexNumber(1,2)
def c3=new ComplexNumber(1,3)
println c1.is(c1) //true
println c1.is(c2) //false
println c1.is(c3) //false

def name1="Martina"
def name2="Martina"
def name3=new String("Martina")
println name1==name2 //true
println name1.is(name1) //true
println name1.is(name3) //false
```

Überladen von Operatoren

- Sämtliche Operatoren sind in Groovy überladbar:

Operator	Methode	Bereits überladen für:
a+b	a.plus(b)	Number,String, Collection
a-b	a.minus(b)	
a * b	a.multiply(b)	
a/b	a.div(b)	Number
a % b	a.mod(b)	
-a	a.negative()	
+a	a.positive()	
a++ bzw. ++a	a.next()	Number, String
a-- bzw. --a	a.previous()	

Überladen von Operatoren

- Sämtliche Operatoren sind in Groovy überladbar:

Operator	Methode	Bereits überladen für:
a[b]	a.getAt(b)	Object, List, Map, Array, String
a[b]	a.putAt(b,c)	Object, List, Map, Array, String, StringBuffer
a | b	a.or(b)	Ganzzahl
a & b	a.and(b)	
a ^ b	a.xor(b)	
a<<b	a.leftShift(b)	Ganzzahl,StringBuffer
a>>b	a.rightShift(b)	Ganzzahl
a>>>b	a.rightShiftUnsigned(b)	

Überladen von Operatoren

- Sämtliche Operatoren sind in Groovy überladbar:

Operator	Methode	Bereits überladen für:

switch(a){
case b:
}
	b.isCase(a)	Obect, List, Collectiom

a==b	a.equals(b)	Object
a != b	!a.equals(b)	
a<b	a.compareTo(b)<0	java.lang.Comparable
a <= b	a.compareTo(b)<=0	
a>b	a.compareTo(b)>0	
a>=b	a.compareTo(b)>=0	
a <=> b	a.compareTo(b)	

Ganzzahlmethoden

- ermöglichen Zählschleifen via Methodenaufruf
- ***upto(upperLimit)***
- Zahlen von 3 bis 7 ausgeben, Zählvariable it

  3.upto(7) {println it} //3 4 5 6 7
- ***downto(lowerLimit)***
- Zahlen von 7 bis 3 rückwärts ausgeben

  7.downto(3) {println it} //7 6 5 4 3
- ***step(limit, s)***
- Zahlen von 7 bis 3 rückwärts mit Schrittweite -2 ausgeben

  7.step(3,-2) {println it} //7 5 3

Ganzzahlmethoden

- ***n.times()***
- Methodenrumpf n-mal durchlaufen

  Beispiel:

  11 Sterne ausgeben

  11.times() { print “x“} // xxxxxxxxxxx

Datentyp Character

- ‘m‘ ist in Groovy ein String!
- “Martina“[0] ist in Groovy ein String!
- Character-Literal in Groovy nur über Umwege:

  ‘c‘.**toCharacter()**

Rechnen mit Character-Codes sehr umständlich:

```
def buchstabe = ‘G‘
def kleinbuchstabe = buchstabe + ‘a‘-‘A‘ // ‘Ga‘


def buchstabe = ‘G‘
def kleinbuchstabe = (buchstabe.toCharacter() +
‘a‘.toCharacter()-‘A‘.toCharacter()) as char // ‘g‘
```

- ohne *as char* würde 103 herauskommen

Stringoperatoren

- **Addition (+)** == Stringkonkatination (wie in Java)
- **Subtraktion (-)** == String 2 aus String 1 löschen:

```
vorname = “Martina“, neu = vorname – “Mar“
println neu // Tina
```

- **Multiplikation(*)** == Mehrfaches Hintereinanderhängen

```
println “x“*3 // gibt 3 mal x aus: xxx
```

- **Inkrement-Operator++** ersetzt letztes Zeichen durch seinen Unicode-Nachfolger
- **Dekrement-Operator--** ersetzt letztes Zeichen durch seinen Unicode-Vorgänger

Stringmethoden

- center(n, c) zentriert den String auf eine Breite von n Zeichen, füllt mit Zeichen c auf

  println "Martina".center(11,"X") //XXMartinaXX

- contains(substr) liefert true, wenn **substr** enthalten ist, sonst false

  vorname="Martina"

  println vorname.contains("oma") //true

  println vorname.contains("oms") //false

- reverse() dreht Zeichenfolge um

  vorname="Martina"

  println vorname.reverse() //anitraM

- size() liefert Stringlänge (wie length())

Zugriff auf Listen

- bauen auf Java-Listen auf
- erweitern diese um “syntaktischen Zucker“

Aufgabe		Syntax	Listeninhalt
Liste definieren	leer	def leereListe=[]	[]
	initialisieren	def namen=[“Jan“, “Thomas“, “Dirk“]	[Jan, Thomas, Dirk]
Entfernen	Element	namen -= “Thomas“	[Jan, Dirk]
	Elemente	namen -=[“Thomas“,“Dirk“]	[Jan]
Elemente	hinzufügen	namen += [“Matt“, “Rod“]	[Jan, Matt, Rod]
	verdoppeln	namen *= 2	[Jan, Matt, Rod, Jan, Matt, Rod]
Listen-position	entfernen	namen[1..3]=[]	[Jan, Matt, Rod]
	ersetzen	namen[0] = “Heiko“	[Heiko,Matt,Rod]

Zugriff auf Listen

Aufgabe		Syntax	Listeninhalt
Ganze Liste		println namen	[Heiko, Matt, Rod]
Element zugreifen	von vorne	println namen[1]	Matt
	von hinten	println namen[-1]	Rod
Bereich zugreifen	vorwärts	println name[0..2]	[Heiko, Matt]
	rückwärts	println name[2..0]	[Matt, Heiko]
	vorwärts	println name[-2..-1]	[Matt, Rod]
	rückwärts	println name[-1..-2]	[Rod, Matt]
Listenlänge		println namen.size()	3
Liste iterieren	for-Schleife	for(name in namen)	Heiko Matt Rod
	each	namen.each{name->print “$name“}	Heiko Matt Rod

Listen durchsuchen

def noten = [1.0, 2.3, 4.0, 1.0, 1.3]

- **Alle** Listenelemente liefern, die vorgegebenes Kriterium erfüllen

  *println.noten.**findAll**{ it < 2.0 } // [1.0, 1.0, 1.3]*

- **Erstes** Listenelement liefern, die das vorgegebene Kriterium erfüllen

  *println.noten.**find**{ it >= 2.0 } // [2.3]*

Listen sortieren

def tiere= [“katze“, “vogel“, “drache“, “maus“]

- **Liste sortieren**

  list.sort() // [“drache“,“katze“, “maus“ “vogel“]

- **Liste nach Sortierkriterium sortieren z.B. letzter Buchstabe**

  list.sort { it[-1] } //[“drache“, “katze“, “vogel“, “maus“]

Klassendefinition

- Ähnlich zu Java aber:
  - Klasse implizit *public*, wenn nichts anderes angegeben
  - mehrere Klassen dürfen in einer Datei definiert werden

- Groovy Beans
  - sind Groovy Klassen

```
class Complex {
def katze
int pi
}
```

Beispiel Klasse

```
class Professor {
    def kuerzel
    def vorname
    def nachname

   Professor(kuerzel, vorname, nachname) {
    this.kuerzel = kurzel;
    this.vorname = vorname;
    this.nachname = nachname;
   }
   String toString(){
    “$vorname $nachname“
   }
}
```

Beispiel Klasse

```
def professoren = [
                  new Professor(“kmt“, “Martina“, “Kraus“),
                  new Professor(“smt“, “Thomas“, “Smits“),
                  new Professor(“gmi“, “Michael“, “Gröschel“)
                  ]

professoren.each{prof-> println prof}
 // kmt Martina Kraus
 // smt Thomas Smits
 // gmi Michael Gröschel
```

Groovy Beans

- parameterloser Konstruktor
  - implizit, wenn keine eigenen Konstruktoren definiert
  - ansonsten selbst definieren
- Getter- und Setter-Methoden für alles Attribute (Bean Properties)
  - implizit für Attribute ohne *public* | *protected* | *private* - Angabe
  - Ansonsten selbst definieren:

```
def getAttributname()
def setAttributname(value)
```

Konstruktoren

- implizit *public*
- Wenn kein anderer Konstruktor definiert ist:

  Konstruktor mit benannten formalen Parametern für alle Attribute

```
class Complex {
def real
def imaginaer
}
```

```
def c1 = new Complex(real:9, imaginaer:3) println “$c1.real $c1.imaginaer“
// 9 3
def c2 = new Complex(real:9) println “$c1.real $c1.imaginaer“
// 9 null
def c3 = new Complex(imaginaer:3) println “$c1.real $c1.imaginaer“
// null 3
```

Sichtbarkeit von Variablen

- lokale Variablen
  - mit def oder Datentyp definierte Variable innerhalb Fúnktion, Script
  - lokal im Bezug auf innerste(n)
    - Befehlsblock { }
    - Funktion
    - Script
  - erst ab der Stelle, an der sie definiert wurde, sichtbar
  - entsprechen lokalen Variablen wie in Java

Sichtbarkeit von Variablen

Instanzvariablen (Attribute) von Klassen

- werden Groovy Bean-Property genannt
- haben implizite Getter und Setter
- entsprechen Instanzvariablen in Java
- Sichtbarkeitssteuerung private, protected oder private

Zugriff von der eigenen Klasse:

```
this.variablenname
```

Zugriff von außerhalb:

```
object.getVariablenname()
object.setVariablenname(wert)
```

Sichtbarkeit von Variablen

Klassenvariablen

- mit static definierte Variablen auf Klassenebene
- entsprechen Klassenvariablen Java
- Sichtbarkeitssteuerung private, protected oder private

Zugriff von der eigenen Klasse:

```
this.variablenname
```

Zugriff von außerhalb:

```
Klassenname.getVariablenname()
Klassenname.setVariablenname(wert)
```

Sichtbarkeit von Variablen

Globale Variablen

- ohne def und ohne Datentyp definierte Variablen außerhalb einer Klasse
- sichtbar
  - in Scriptcode derselben Datei
  - allen globalen Funktionen
- nicht sichtbar
  - Klassen und deren Methoden

Expandos

- dynamisches Objekt
- instanziiert aus der Groovy-Klasse Expado
- beliebige Attribute können hinzugefügt werden
  - Angabe als benannter Parameter im Kontruktoraufruf
  - bloßes Beschreiben eines noch nicht vorhandenen Attributs

```
c1 = new Expando(real:9, imagniaer:3)


c1.name = “Mein Expando“
println “$c1.name: $c1.real $c1.imaginar“ //Mein Expando 9 3
```

Klausurbesprechung

- 90 Minuten Klausur
  - 60 Minuten VAR
  - 30 Minuten WAW

- Themen:
  - 15 Punkte Web-Server Theorie
  - 5 Punkte CSS Theorie
  - 5 Punkte JavaScript Theorie
  - 5 Punkte PHP Theorie
  - Kein HTML oder Groovy
  - Kein „auf Papier Programmieren“

Klausurbesprechung

Fragen?