Groovy --> Java-Code

Statische Typen und optionale Typen

Java : String text = "Hallo";
Groovy : text = "Hallo"

Listen und Maps

liste = [1,2,3,4,5]
map = [abs : "AAA", dghd : "hjkhk"]

Funktionsobjkte (Closures)

function = {Daten -> print daten}

Frameworks

Grails - Web Framework - Basiert auf Spring
GSP - Groovy Servlet Pages
Griffon - Desktaop Anwendungen (Swing basiert / JavaFX)

Groovy Development Kit

Standard Biblotheken - Groovy Hilfklassen, JDBC, XML
Hilfsbibliotheken - Logging, Testcases
Compiler --> Groovyc --> class-Dateien
Tooling --> Shell Interpreter (ohne Compilierung für Skripte)
Konsole --> Texteditor

Homepage : groovy-lang.org --> Download

Groovy-Verzeichnis

lib : Normale Biblotheken
indy : Performante Biblotheken für die neuste Java-Version
bin : Tools

System --> PATH erweitern plus GROOVY_HOME

1.Groovy-Anwendung

bin/groovyConsole --> Editor --> println "Hallo"

Groovy Interpreter --> Runetime-Interpreter

groovysh.bat (Ende mit ":exit")

:help

script.groovy anlegen
println "Hallo"
groovysh.bat script.groovy
groovy.bat script.groovy

groovyConsole --> Editor plus Ausgabeconsole

Compiler --> .groovy --> .class
Mischung von Java und Groovy (-j Flag)
Über Java kann man dann den Code starten

Gradle (Buildtool wie maven oder ant)

Gradle-Projekt erzeuigen
build.gradle --> apply plugin : 'groovy' setzen
Dependencies -->
mvcrepository.com --> home > org.codehaus.groovy > groovy-all
2.4.0-beta-4
Gradle --> Ausdruck kopieren

compile 'org.codehaus.groovy:groovy-all:2.4.0-beta4'

Ordner erstellen (src/main/groovy)
Ordner erstellen (src/main/java)
Ordner erstellen (test/groovy)

GroovyTest

```
class GroovyTest extends GroovyTestCase {
    public static testFunktion(){
        def helloWord = new HelloWorld()
    }
}
```

Abhängigkeiten auflösen (Grape)

Maven Repository
@Grad Annotation

Funktionen als Objekte

Funktionen können als Objekte Daten "umschliessen"
Funktionen können "ausgetauscht" werden
Keine statische Klassenbindung
Keine "Inner Classes"

<u>Beispiel;</u>

```
def function = { name -> println "Hallo $name" }
function("Christian")
```

Def Keyword (lokale Variable) - Objekttyp wird angenommen
Groovy arbeitet mit Reflection
Semikolon sind OPTIONAL
Return Keyword OPTIONAL Terminierung mit einem Identifier
Funktionen > Klammern sind OPTIONAL (wenn nur ein Parameter übergeben wird)

POGO > Groovy Bean (wie Java Bean)

Deklaration von Variablen
Generieren von Getter(Setter Methoden
Setter(Getter unsichtbar für Groovy Nutzer
Steuerung von Sichtbarkeiten durch readonly/final
@ Operator für Feldzugriff
Konstruktor > Automatische Erzeugung

Beispiel:

```
import groovy.transform.toString
@ToString           <-- ALle Properties bekommen eine ToString-Methode
class Person {
   String name
   String vorname
}

def person = new Person(vorname:'Christian' name:'Müller');
person.@vorname = "Christian"
person.vorname = "Markus"
person.setVorname = "Klaus"

def map = [ vorname:'Christian' name:'Müller' ]
def person = new Person(map);
```

String Interpolation

String Konstante : 'text'
GroovyString (GString) : "text" --> Plus interpolation; In Java mit StringFormat
Referenz mit $-Identifier

Beispiel:

```
def name = "Christian"
println "Mein Name ist $name"      --> Christian
println 'Mein Name ist $name'      --> $name
println "Mein Name ist ${name}"    --> Christian
```

Mehrspaltige Texte (entweder """ oder ''')

```
println """
   Das ist ein Text\\n
   Das ist ein Text
   """
```

**Regex Matching (Reguläre Ausdrücke
)**

==~ Operator - Evaluiert zu Boolean

Excaping umgehen - Mechanik aus JavaScript
Deklaration eines Redex-Strings mit /[0-9]{2}/

=~ Operator - Redex Grouping; Macht ein Matching mit Gruppen; Durchsuchung längere Texte

```
def muster = ~/Beispiel/
assert muster.matcher('Beispiel').matches()
def redex = /Audi A[1-8]/
assert 'Audi A7' ==~ redex        --> true
assert 'Audi A9' ==~ redex        --> false
assert 'Audi A7aghag' =~ redex     --> true (enthält)

def text = '''
   AMD stellt CPU Einheiten her.
   AMDs Produktlinie besteht aus Grafikkarten und CPUs.
   AMD ist Konkurrent von Intel
   '''
// Suche AMD oder AMDs wenn danach ein Leerzeichen und ein P oder s folgt.
def result = (text =~/AMD([s]? [P|s])/).replaceAll('Intel$1)
```

Ergebnis:

```
   Intel stellt CPU Einheiten her.
   Intels Produktlinie besteht aus Grafikkarten und CPUs.
   AMD ist Konkurrent von Intel
```

Null immer false --> if (result) { do() }
Leere Collections immer false
Leere String immer false
Equals --> ["abc", "def"] == ["abc", "def"] bzw. "Text" == "Text"
Switch ()

```
   case xxx: .... break
   default: ... break
```

Nullpointer mit Elvis lösen

? Operator
Nutung um If-Statement zu vermeiden
(test == true) ? "Wert1" : "Werte2"
?: Operator (Elvis Operator)
Redudanzen vermeiden
(text == null) ?: "Standard Text" <--- Wenn Null > Standard Text
?. Operator
Safe-Dereference Operator
Methoden/Property Zugriff absichern
Null Objekte Statements werden nicht ausgeführt
people.collect{ p -> p.job?.salary }.max()

Iteratoren – Loops

For Loop
Iteration über Ranges
for (i in 0..10) {}
for (val in ['Montag', 'Dienstag', 'Mittwoch']) {}
Collection Iteration
Closure forEach/each Methode
def liste = [‘AAA‘, 'BBB', 'CCC'];
liste.each { println it }
liste.eachWithIndex { text, index --> println "$index : $text" }

Listentypen und Maps

Listentyp
Simple Deklaration mit []
Type-Casting it as Keyword (["A", "B", "C"])
Diverse Aggregationsfuntkionen : min, max, sum
Diverse Verarbeitungsfunktionen
*. Operator
def liste = ['liste'. 12. true, Boolean]
assert liste == ['liste'. 12. true, Boolean]
def n = 0..5 as Integer[]

Maptyp
Simple Deklaration mit [key : "value"]
Erstellung mit [:]
Intuitiver Zugriff map.key
*. Operator
Beliebige Map-Tiefe
def map = [key: 'wert', key2: 'wert2']
assert map.key == 'wert'

Operatoren überladen (Java nicht möglich)

Möglichkeit der Definition - Nur definierte Operatoren
Vorsicht !!!!

Formlose Objekte (Expando)

Definition zur Laufzeit
Vergleichbar mit einem JSON Objekt

```
def expando = new Expando()
expando.text = "Hallo"
expando.methode = { nummer -> (int)Math.pow(nummer, 2) }
println expando.methode(2)
```

Meta-Programmierung

Definition : Programme schreiben die Programme schreiben
"Code schreiben der Code schreibt"