



Webarchitekturen und -frameworks (WAF)

Prof. Dr. Thomas Specht

T.Specht@hs-mannheim.de



Hochschule Mannheim University of Applied Sciences

Organisatorisches

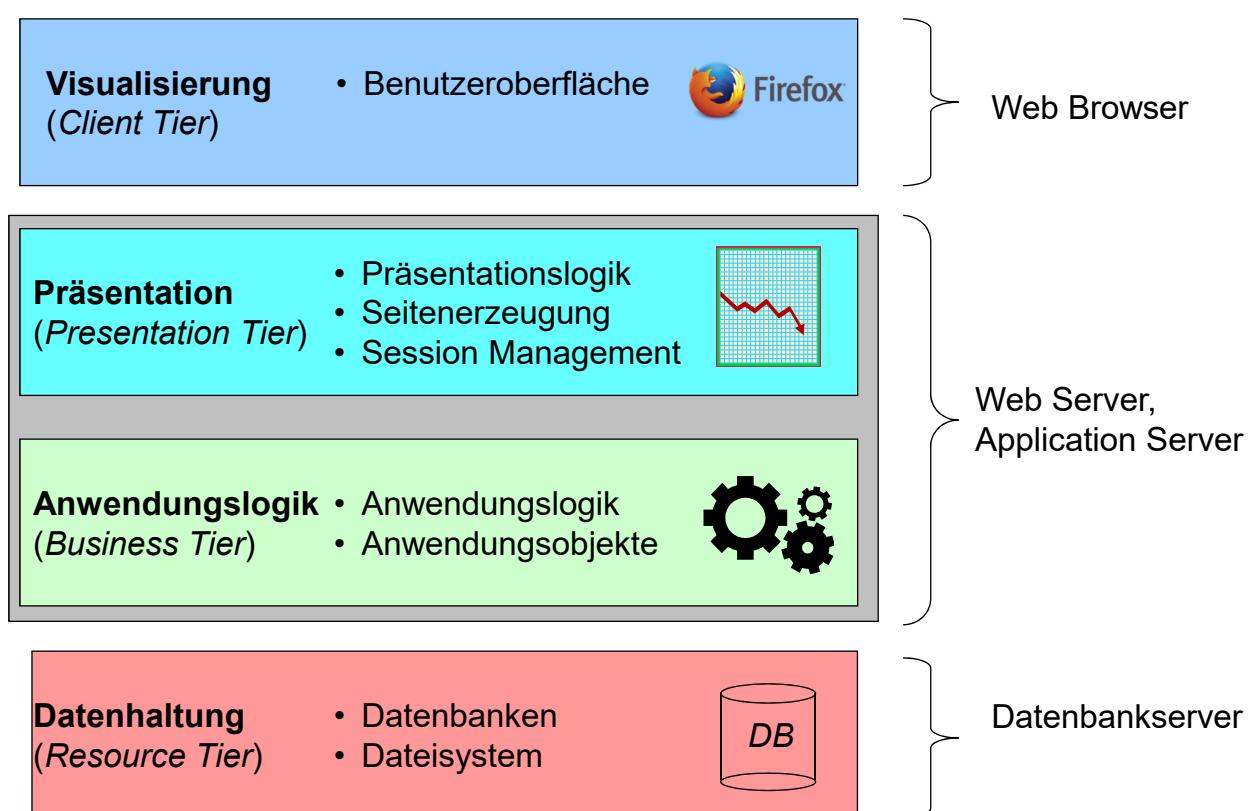
- **Freigegeben für**
 - IB, IMB, UIB
 - alle RGSe
- **Voraussetzungen**
 - Inhalt der Vorlesung WAW: HTML, CSS, JavaScript, DOM, AJAX
- **Vorlesungsunterlagen**
 - immer nach der Vorlesung im Moodle
 - Kurs: WAF_SPE
 - Passwort: Gemmen1Alp7Horn
- **Studienleistung**
 - **alte RGSe: Hausarbeit**
 - **neue RGSe: keine** Studienleistung erforderlich
- **Prüfung**
 - voraussichtlich mündlich



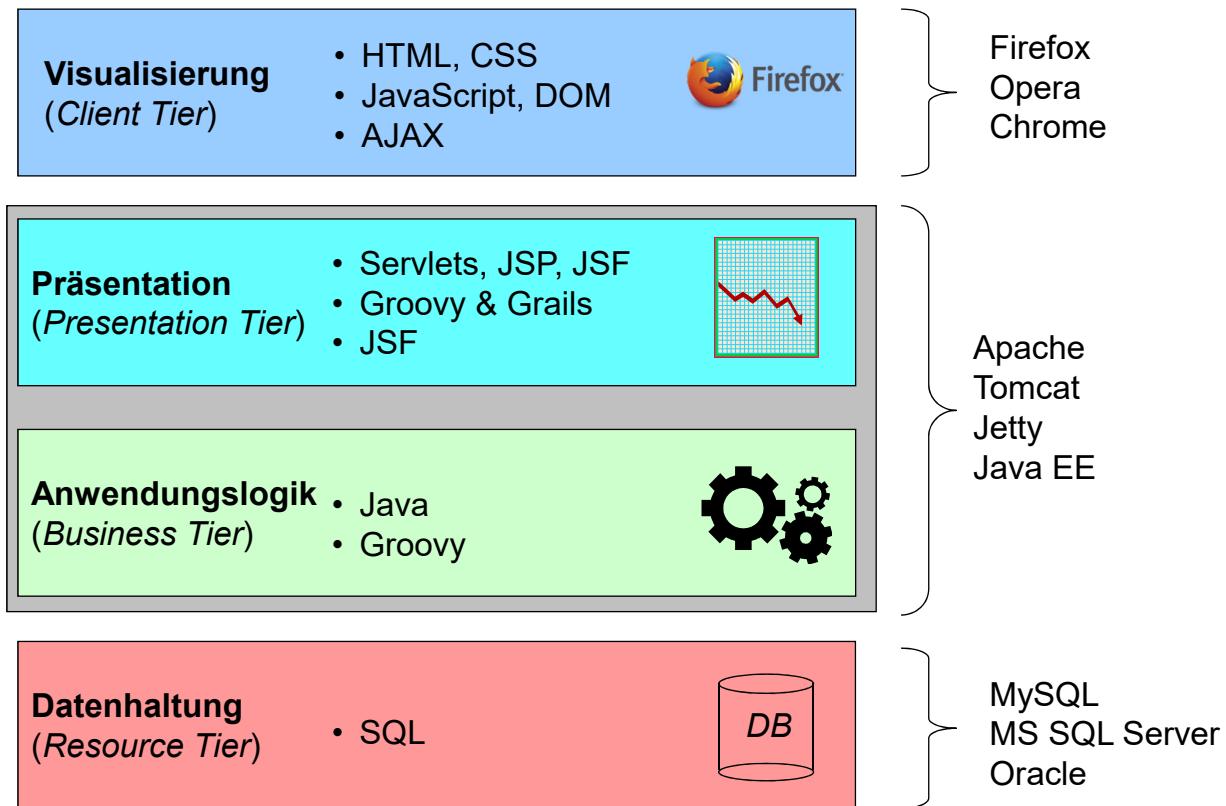
Gliederung

1. Webarchitekturen
2. Groovy
3. Grails-Framework
4. Java-Webtechnologien
5. Java Server Faces

Vierschichtenarchitektur für Webanwendungen



Typische Technologien



Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

5

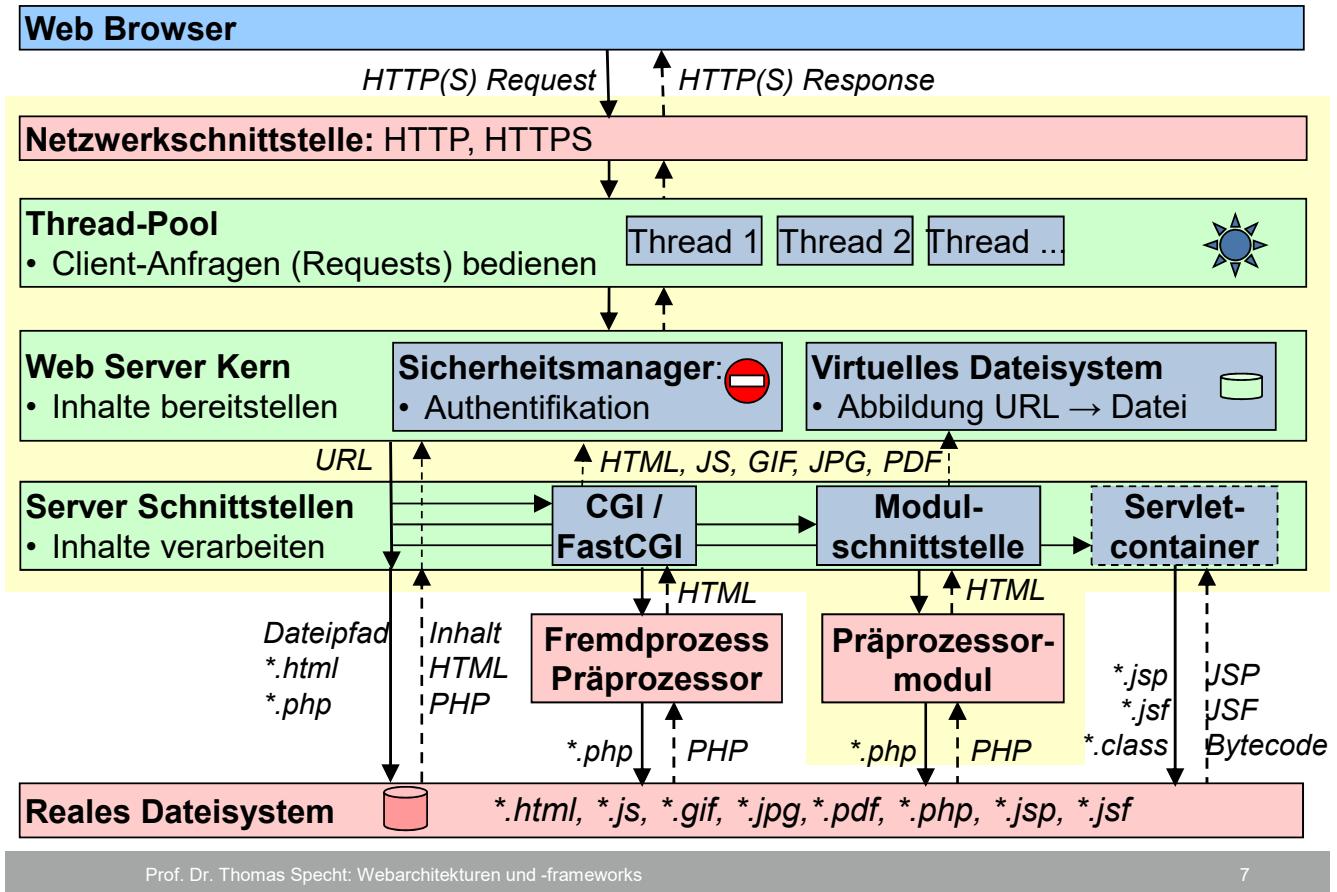
Java Servlets

- **Servlet**: Java-Programmcode, der
 - zur Laufzeit mit `println` HTML-Seiten erzeugt
 - compiliert werden muss
 - im Servlet Container abläuft
 - mühsame LowLevel-Programmierung erfordert
- **Servlet-Container**
 - standardisierte Ablaufumgebung für Servlets, JSP, JSF und Groovy & Grails
 - enthalten in
 - Apache Tomcat
 - Jetty
 - Java EE-Application Servern
 - mittelschnell
 - JSP, JSF, Groovy & Grails vom Servlet-Container beim ersten Aufruf
 - automatisch in Servlet umgewandelt
 - compiliert

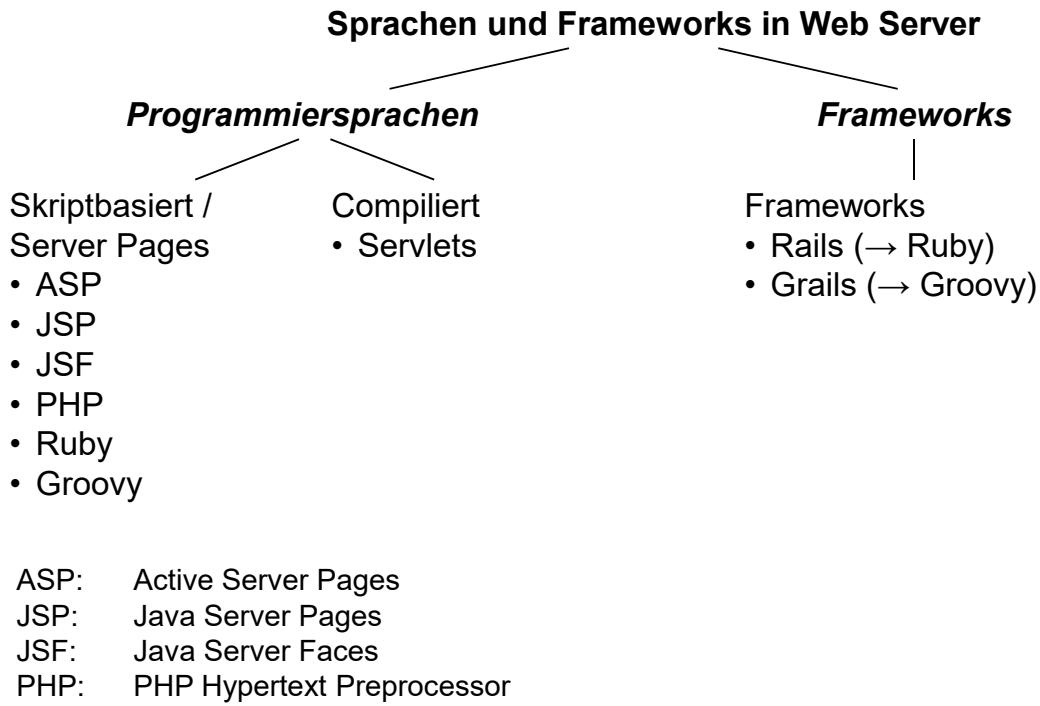
Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

6

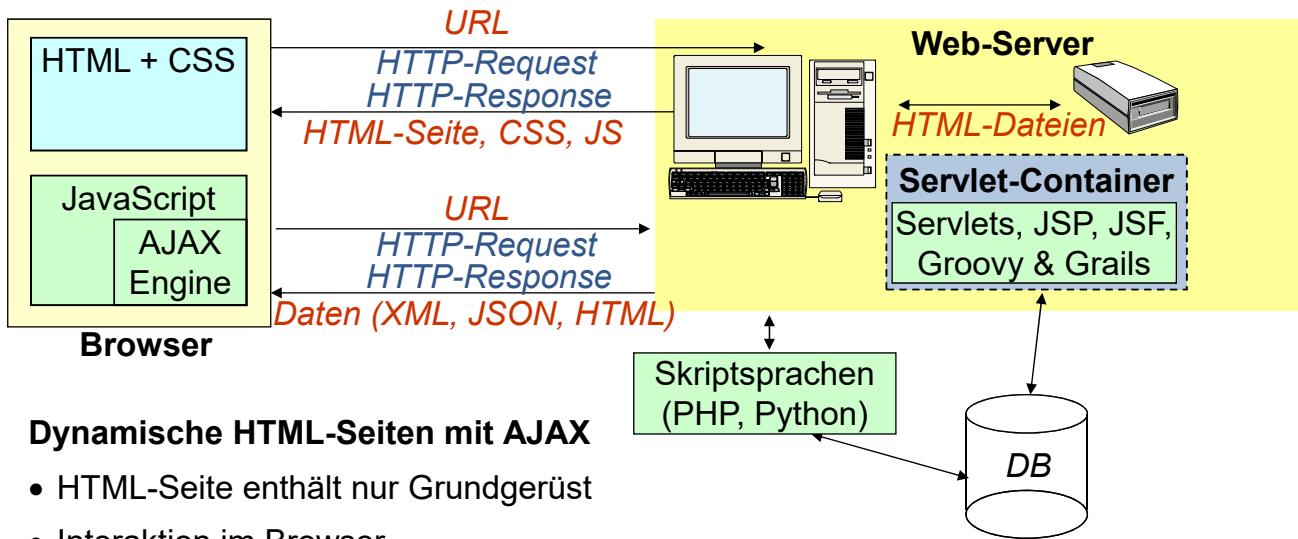
Architektur Web Server



Sprachen und Schnittstellen in Web Server



Dynamische interaktive Webseiten mit AJAX



Dynamische HTML-Seiten mit AJAX

- HTML-Seite enthält nur Grundgerüst
- Interaktion im Browser
 - JavaScript-Ereignisse
 - Nutzdaten (oder auch HTML-Abschnitte) asynchron nachladen
 - Nutzdaten werden neu dargestellt, Rest der Seite bleibt bestehen
- eine HTML-Seite kann beliebig oft Nutzdaten nachladen
 - ganze HTML-Seite wird nur selten ausgetauscht



Gliederung

1. Webarchitekturen
2. **Groovy**
3. Grails-Framework
4. Java-Webtechnologien
5. Java Server Faces



Gliederung

2. Groovy
 1. Einführung in Groovy
 2. Syntax und Kontrollkonstrukte
 3. Funktionen
 4. Closures
 5. Datentypen und Operationen
 6. Collections
 7. Klassen und Groovy Beans
 8. Reguläre Ausdrücke
 9. Metaprogrammierung und Builder
 10. Testen

Anforderungen an Sprachen für dynamische Webseiten

- **allgemein**
 - gute Integration mit (X)HTML, CSS, JavaScript und AJAX
 - Verarbeitung von Formularfeldern und –parametern
 - leicht erlernbar
 - leicht handhabbare Listen, assoziative Arrays (Maps)
 - leichte Datenbankprogrammierung
 - Programmereffizienz
 - skriptfähig
- **für kleinere Systeme**
 - "lasche" Syntaxvorschriften
 - dynamische Typisierung
- **für große Systeme**
 - Typsicherheit (→statische Typisierung)
 - gute Integration mit vorhandenen Bibliotheken und Frameworks
 - Java-Klassenbibliothek
 - EJB, Spring
 - JPA, Hibernate
 - Performanz
 - Testbarkeit

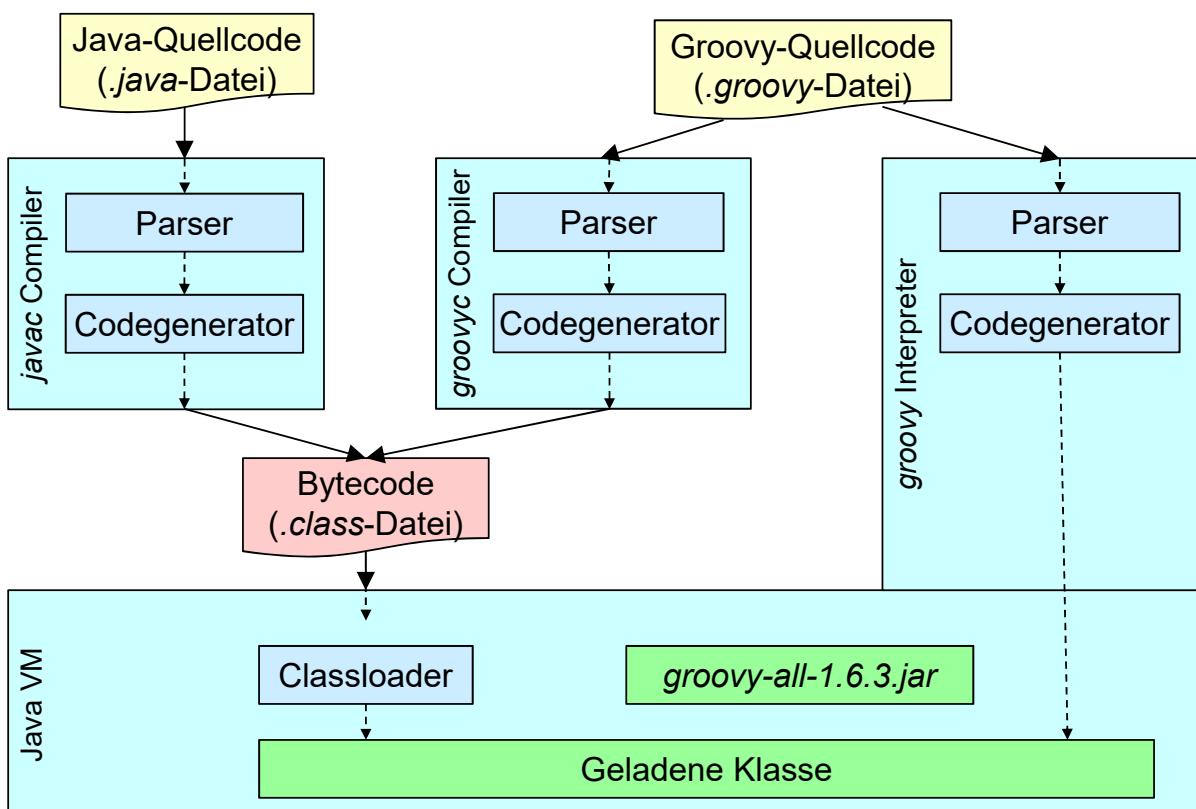
Serverseitige Skriptsprachen und Web-Frameworks

Anforderung	PHP	Java	Serv-let	JSP	JSF	Groovy	Groovy + Grails
Integration (X)HTML, CSS, JS	++	-	o	+		-	++
Formularverarbeitung	++	-	o	+		-	++
Einlernaufwand	+		+		o		++
Listen, Assoziative Arrays	++			o			++
Datenbankprogrammierung	+			+			++
Programmiereffizienz	+		o				++
Skriptfähigkeit	++		--		o		++
"lasche" Syntaxvorschriften	++		--		-		+
Dynamische Typisierung	++			-			++
Statische Typisierung, Typsicherheit	-			++			+
Integration mit Klassenbibliotheken / Frameworks	o			++			++
Performanz	+	++	+	+	o		+
Testbarkeit	o	++		+			++

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

13

Zusammenspiel Groovy, Java und JVM (1/2)



Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

14

Dateikonventionen:

- Java-Quelltexte: *.java
- Groovy-Quelltexte: *.groovy
- Compilierte Klassen: *.class

Starten einer Groovy-Anwendung:

- direkt im Interpreter:
 - groovy SayHello.groovy Thomas
- compilieren
 - groovyc HelloWorldGroovy.groovy
- compilierte Version starten
 - groovy HelloWorldGroovy
 - java -cp %GROOVY_HOME%\embeddable\groovy-all-2.4.7.jar;. HelloWorldGroovy Thomas

Erstes Beispiel: HelloWorld in Java und in Groovy

HelloWorldJava.java

```
public class HelloWorldJava {  
    public static void main(String[] args) {  
        System.out.println("Hello " + (args.length>0 ? args[0] : "unknown")+"!");  
    }  
}
```

HelloWorldJGroovy.groovy

```
println "Hello ${args.length>0 ? args[0] : 'unknown'}!"
```

Java vs. Groovy: Syntax

- (fast) jedes Java-Programm ist auch gültiges Groovy-Programm
- Groovy bringt einige
 - **Vereinfachungen**
 - alle Datentypen (auch elementare) sind Objekte
 - dynamische Typisierung (Datentyp bei Deklaration weglassbar)
 - vereinfachte Nutzung von Listen, Mengen, assoziative Arrays (Map), Ranges
 - Verzicht auf "unnötige" Syntax wie Semikolons, main-Funktion ...
 - **Erweiterungen**
 - Überladen und Überschreiben der Operatoren
 - Closures
 - umfangreiche Metaprogrammierung, domänen spezifische Sprachen

Java vs. Groovy: Laufzeitumgebung

- beide laufen in Java VM
 - plattformunabhängig
 - Groovy und Java problemlos integrierbar, Bytecode-kompatibel
 - Groovy in gängige Applicationserver integrierbar (in Form eines *.jar*)
- Java-Quellcode muss vor Ausführung compiliert werden
- Groovy kann wahlweise compiliert oder direkt als Skript ausgeführt werden
 - implizite Compilierung beim ersten Aufruf

Komplexeres Beispiel in Groovy: Professoren.groovy (1/2)

```
class Professor {  
    def kuerzel  
    def vorname  
    def nachname  
  
    Professor(kuerzel, vorname, nachname) {  
        this.kuerzel = kuerzel;  
        this.vorname = vorname;  
        this.nachname = nachname;  
    }  
  
    String toString() {  
        "$vorname $nachname"  
    }  
}  
  
...
```

Komplexeres Beispiel in Groovy: Professoren.groovy (2/2)

```
...  
  
def professoren = [ new Professor("spe", "Thomas", "Specht"),  
    new Professor("knu", "Peter", "Knauber"),  
    new Professor("grt", "Rainer", "Gerten"),  
    new Professor("sni", "Helmut", "Schnitzspan")]  
  
professoren.each ({println it})  
println()  
  
professoren += new Professor("kae", "Peter", "Kaiser")  
professoren.each { println it }  
println()
```

Dasselbe in Java (1/2): Professor.java

```
public class Professor {  
    String kuerzel;  
    String vorname;  
    String nachname;  
  
    public Professor(String kuerzel, String vorname, String nachname) {  
        this.kuerzel = kuerzel;  
        this.vorname = vorname;  
        this.nachname = nachname;  
    }  
  
    public String toString() {  
        return vorname + " " + nachname;  
    }  
}
```

Dasselbe in Java (2/2): Professoren.java

```
import java.util.ArrayList;  
public class Professoren {  
  
    public static void main(String[] args) {  
        ArrayList<Professor> professoren = new ArrayList<Professor>();  
        professoren.add(new Professor("spe", "Thomas", "Specht"));  
        professoren.add(new Professor("knu", "Peter", "Knauber"));  
        professoren.add(new Professor("grt", "Rainer", "Gerten"));  
        professoren.add(new Professor("sni", "Helmut", "Schnitzspan"));  
  
        for (Professor professor : professoren)  
            System.out.println(professor);  
        System.out.println();  
        professoren.add(new Professor("kae", "Peter", "Kaiser"));  
  
        for (Professor professor : professoren)  
            System.out.println(professor);  
        System.out.println();  
    }  
}
```



Gliederung

2. Groovy
 1. Einführung in Groovy
 - 2. Syntax und Kontrollkonstrukte**
 3. Funktionen
 4. Closures
 5. Datentypen und Operationen
 6. Collections
 7. Klassen und Groovy Beans
 8. Reguläre Ausdrücke
 9. Metaprogrammierung und Builder
 10. Testen

Syntax

- Strichpunkte am Zeilenende können generell entfallen
 - Beispiel: `i=5` statt `i=5;`
 - Einzige Ausnahme: Gewollte Endlosschleifen, z.B. `while(true);`
 - Möglichst genau eine Zeile pro Befehl
- automatisch importierte Packages:
 - `java.lang.*`
 - `java.util.*`
 - `java.net.*`
 - `java.io.*`
 - `groovy.lang..*`
 - `groovy.util.*`

Variablendefinition

- statt Datentyp kann einfach **def** angegeben werden
 - Beispiel: **def j** statt **int j;**
- wenn Variable zugleich initialisiert, braucht nicht einmal **def** dort stehen:
 - Achtung: Deklaration lokaler Variablen benötigt **def** oder Datentypangabe!
 - Beispiel:
j=3; // Bei erstmaliger Verwendung von j: Deklaration mit Initialisierung
- Datentyp ergibt sich aus Initialisierungswert
 - jede Zuweisung kann Datentyp dieser Variablen ändern!
 - dynamische Typisierung
 - Beispiel:
def j=3
`println "j=" + j // j=3
j="Nun ist j ein String!"
println j // Nun ist j ein String!`
- Variablen dürfen auch außerhalb von Klassen deklariert werden
 - "globale" Variablen

Groovy-Wahrheit

Groovy-Wahrheit

- Ausdrücke in Bedingungen brauchen nicht vom Typ Boolean sein
- Automatische Konvertierung nach Boolean gemäß folgender Tabelle

Datentyp	als false interpretiert	als true interpretiert
Boolean	false	true
Ganzzahl	0	alles $\neq 0$
Fließkommazahl	0.0	alles $\neq 0.0$
Character	0	alle Unicodes $\neq 0$
String	leerer String	alle Strings mit mind. einem Zeichen
Collection	leere Collection	nichtleere Collections
Assoziatives Array (Map)	leeres assoziatives Array	nichtleere assoziative Arrays
Matcher	nix gefunden	Pattern mindestens einmal gefunden
Sonstige Objekte	Objektreferenz null	Objektreferenz $\neq null$

Bedingte Abfragen

Bedingte Abfragen mit *if*

- Bedingungsausdruck braucht kein Boolean zu liefern
- sonst wie Java

Bedingte Abfragen mit *switch*

- Switch-Variable darf auch String sein
- in Case-Zweigen auch erlaubt:

```
case 0..9:    println("Einstellige Zahl")           // Range
              break
case [8,9,10]: println("Wert ist 8, 9 oder 10")     // Liste möglicher Werte
              break
case Float:   println("Fließkommazahl")            // Datentyp
              break
case {it%2==0}: println("Gerade Zahl")             // Closure
              break
case ~/[A-Z].*/: println("Beginnt mit Großbuchst.") // Pattern Matching
              break
```

Schleifen

while-Schleife

- Bedingungsausdruck braucht kein Boolean zu liefern
- sonst komplett wie Java

do while-Schleife

- entfällt ersatzlos

for(initialisierung; fortsetzungsbedingung; inkrementer)

- *fortsetzungsbedingung* braucht kein Boolean zu liefern
- sonst wie Java

for(element in iterable)

- iteriert durch Ranges, Collections, Maps, Arrays
- Beispiele:

```
for (zahl in 1..5) print zahl      // 12345
for (zahl in [5,8,13]) print zahl + " " // 5 8 13
def name="Thomas"; for (i in 0..<name.size()) print name[i] + " " // Thomas
```

break und continue

- wie Java

Exception Handling

try {} catch {} finally {}

- auch geprüfte Exceptions brauchen nicht abgefangen werden
- sonst wie in Java
 - {} dürfen nicht weggelassen werden
 - Typ der Exception darf in catch-Zweigen nicht weggelassen werden



Gliederung

2. Groovy
 1. Einführung in Groovy
 2. Syntax und Kontrollkonstrukte
3. Funktionen
4. Closures
5. Datentypen und Operationen
6. Collections
7. Klassen und Groovy Beans
8. Reguläre Ausdrücke
9. Metaprogrammierung und Builder
10. Testen

Formale Positionsparameter

- Zuordnung formale ↔ aktuelle Parameter über Position in Parameterliste
 - aktuelle Parameter an formale Parameter von links nach rechts zugeordnet
 - Defaultwert für fehlende Parameter (sofern bei Funktionsdefinition angegeben)
 - entsprechen formalen Parametern in Java
 - Beispiel:

```
int max(int x, int y) { return x>y ? x : y;}  
println max(4, 7);
```



- Parameterdatentyp
 - durch **def** ersetzbar:
*int square(def x) { return x*x }*
 - darf auch ganz fehlen:
*int square(x) { return x*x }*
→ implizit Typ *Object*
- Parameter
 - kann Defaultwert haben
int blinken(n=1) { n.times {println "blink"}}

Optionale formale Positionsparameter in beliebiger Anzahl

- "letzter" Parameter vom Typ *Object[]*

- Beispiel:

```
def max(x, Object[] z) {  
    def tmp = x  
    z.each { if (it>tmp) tmp=it }  
    return tmp  
}  
println max(2,6)          // max(2,[6])          liefert 6  
println max(2,6, 4)       // max(2, [6,4] as Object[])  liefert 6
```

Benannte Parameter

- Zuordnung formale ↔ aktuelle Parameter über ihren Namen
 - gibt es in Java nicht
- Definition einer Funktion mit benannten Parametern
 - genau ein formaler Parameter vom Typ *Map*
 - nimmt sämtliche benannten Parameter als Assoziatives Array auf
 - Beispiel:

```
def begruessung(Map param){
    def gruss=""
    if (param.vorname)
        gruss += "$param.vorname "
    if (param.nachname)
        gruss += "$param.nachname "
    if (param.ort)
        gruss += "aus $param.ort"
    return gruss
}

println begruessung(vorname:"Thomas")           // Thomas
println begruessung(nachname:"Specht")          // Specht
println begruessung(vorname:"Thomas", nachname:"Specht") // Thomas Specht
println begruessung(vorname:"Thomas", ort:"Ulm")   // Thomas aus Ulm
```

Funktionsdefinition in Java und Groovy

Sprache	Java		Groovy	
Parameterzuordnung	Position	Name	Position	Name
mit Typangabe	<code>int max(int x, int y)</code>	-	<code>int max(int x, int y)</code>	<code>def max(Map map)</code>
ohne Typangabe	-		<code>def max(def x, def y)</code>	<code>def max(def map)</code>
mit Defaultwert	-		<code>def max(x, y)</code>	<code>def max(map)</code>
optionale Parameter	<code>int max(int x, int... yArray)</code>		<code>def max(Object[] o)</code>	implizit
Rumpf	<code>return x>y ? x : y;</code>	-	<code>x>y ? x : y;</code>	<code>map.x>map.y ? map.x : map.y;</code>
Aufruf	<code>int z = max(5,3);</code>	-	<code>def z = max(5,3)</code>	<code>def z = max(x:5, y:3)</code>
Vorteile	+ bequeme Syntax + opt. Parameter	-	+ bequeme Syntax + opt. Parameter + Defaultwerte	<ul style="list-style-type: none"> • Parameter + Reihenfolge bel. + bel. weglassbar
Nachteile	<ul style="list-style-type: none"> • Parameter - Reihenfolge fest - kein Defaultwert 	-	<ul style="list-style-type: none"> • Parameter - Reihenfolge fest 	- unbequeme Syntax

Rückgabewert und -typ

- Rückgabetyp
 - darf durch `def` ersetzt werden:
 - Rückgabetyp implizit aus `return`-Befehl
 - generische Funktionen / Methoden
- letzter Wert einer Funktion ist automatisch Rückgabewert
 - `return`-Schlüsselwort kann dann entfallen:

```
def square(x) { return x*x }
```

```
def square(x) { x*x }
```

Funktions- / Methodendefinition

- Funktionen + Methoden
 - dürfen auch außerhalb von Klassen definiert werden
 - "globale" Funktionen
 - implizit `public`, wenn nicht `protected` oder `private` angegeben

Funktions- / Methodenaufruf

- 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 Funktionsaufrufen dürfen nur äußerste Klammern entfallen!
 - erlaubt: `System.out.println Math.max(3,4)`
 - nicht erlaubt: `System.out.println Math.max 3,4`
- Bei Bildschirmausgaben mit `print` und `println` darf `System.out` weggelassen werden
 - Beispiel: `println "Hello world"` statt `System.out.println("Hello world");`



Gliederung

2. Groovy
 1. Einführung in Groovy
 2. Syntax und Kontrollkonstrukte
 3. Funktionen
 - 4. Closures**
 5. Datentypen und Operationen
 6. Collections
 7. Klassen und Groovy Beans
 8. Reguläre Ausdrücke
 9. Metaprogrammierung und Builder
 10. Testen

Ausgangssituation und Motivation

- bislang sind nur die Daten Objekte
→ nur Daten
 - als Parameter an Funktions- und Methodenaufrufe übergebbar
 - aus Funktionen und Methoden zurückgebar
- warum sollen auch Funktionen / Methoden Objekte sein?
 - als Parameter an Funktions- und Methodenaufrufe übergeben
 - Einsatz z.B. für
 - Callback-Methoden asynchroner Aufrufe
 - Ereignishandler (vgl. *ActionListener* in Swing)
 - Vergleichsfunktion in Sortieralgorithmen
 - Aufruf einer als Parameter übergebenen Funktion für jedes Element einer Collection

Bisherige Realisierung ohne Closures

- C/C++: Zeigervariable auf Funktion (nur in C/C++, nicht in Java)
 - sehr schwer lesbare Syntax
 - fehleranfällig
- Java bis Version 7: **Interface speziell für zu übergebende Funktion erstellen**
 - Parameter / Rückgabe vom Typ des Interfaces
 - für jede Übergabe einer Funktion
 1. Hilfsklasse aus **Interface** erben lassen, die das **Interface** implementiert
 2. Hilfsklasse instanzieren
 3. Instanz als Parameter übergeben
- Konsequenzen
 - hoher Programmieraufwand, unübersichtlicher Quellcode
 - Definition vieler nicht wiederverwendbarer Hilfsklassen
 - manuelle Instanzierung der Hilfsklasse, nur um Funktionsobjekt zu erhalten
 - haufenweise Instanzenmüll für Garbage Collector
- Java ab Version 8: Lambda-Ausdrücke

Beispiel: Array ausgeben

- **Script**

```
def array = ["Thomas Specht", "Thomas Smits", "Peter Knauber",
            "Elena Fimmel"];
array.each { print "$it, "}
```

Array-
iterator

Funktionsobjekt
(Closure)

- **Vorteile der Groovy-Lösung**

- Trennung zwischen Iterationssteuerung und Aktion pro Collection-Element
 - Iterationssteuerung
 - + bereits in Collection-Klassen enthalten
 - + für selbst geschriebene Collection-Klassen leicht implementierbar
 - + keine speziellen Interfaces (vgl. *Iterator* in Java) nötig
→drastische Kürzung des Programmcodes
 - Aktion pro Collection-Element
 - + steht direkt hinter dem Iterator-Aufruf
 - + keine eigene (innere) Hilfsklasse nötig
→leichter lesbarer Code

Beispiel: Dateiinhalt ausgeben

- **Script**

```
try {
    new File("text.txt").eachLine { println it }
} catch(e) {
    println "Konnte Datei nicht einlesen: " + e.getMessage()
}
```

Zeilen-
iterator

Funktionsobjekt
(Closure)

- Das ist alles!
 - selbst die Fehlerabfrage hätten wir noch weglassen können:
`new File("text.txt").eachLine { println it }`
- Reduzierung auf eine Zeile!

Vorteile der Groovy-Lösung

- Trennung zwischen Iterationssteuerung und Aktion pro Zeile
 - Iterationssteuerung
 - + in Ressourcenzugriffsklassen enthalten
 - + für selbst geschriebene Ressourcenhandler leicht implementierbar
 - + übernimmt Verwaltungsaufgaben wie
 - Öffnen und Schließen von Verbindungen
 - Exception Handling
 - drastische Kürzung des Programmcodes
- Aktion pro Zeile
 - + steht direkt hinter dem Zeileniterator-Aufruf
 - leichter lesbarer Code

Was ist eine Closure?

- anonymer Codeblock zwischen {}
- Objekt vom Typ *Closure*, das
 - einer Variablen zugewiesen werden kann:
`def closure= { println "Hallo" }`
 - ausgeführt werden kann:
`closure()`
 - einem Methodenaufruf als Parameter übergeben werden kann:
`def array = ["Thomas Specht", "Sven Klaus", "Peter Knauber", "Elena Fimmel"];`
`array.each ({ print "$it, "})`
 - oder kürzer unter Weglassung der Funktionsaufrufklammern:
`array.each { print "$it, "}`
- aus einer Funktion zurückgegeben werden kann
`return { print "$it, "}`

Parameterübergabe an Closures (1/2)

- Wie definiert man eine Closure mit Parametern?
- Am Anfang der Closure können formale Parameter definiert werden:
 - Syntax:
`{ param1, param2 -> Closurerumpf }`
 - Vor `->` beliebig viele formale Parameter, kommagetrennt
 - nutzt Parameter
- Beispiel:
`{ x, y -> x>=y ? x : y }`
- Vor jedem Closure-Parameter kann stehen
 - `def` (überflüssig)
Beispiel: `{ def x, def y -> x>=y ? x : y }`
 - Typangabe (schränkt Closure auf Parametertyp ein)
Beispiel: `{ int x, int y -> x>=y ? x : y }`

Parameterübergabe an Closures (2/2)

- Parameterlose Closures (ohne Angabe von `->`)
 - können trotzdem mit einem Parameter aufgerufen werden
→ Parameterwert im Rumpf in Variable `it` verfügbar
 - Beispiel:
`def abs2 = { it>=0 ? it : -it }`
`println abs2 (-9) // 9`
 - `it` wird in Closures sehr häufig genutzt

impliziten Parameter
`it` nutzen

Vergleich Funktionen + Methoden ↔ Closures

	Funktion	Methode	Closure
ist Objekt?	-		ja
hat Name?	ja		nur über Zuweisung an Variable
kann Parameter haben?		ja	
ohne Deklaration	<code>def max(x,y) { return x>=y ? x : y }</code>	<code>def max = { x,y -> return x>=y ? x : y }</code>	
mit	<code>def max(x,y=0) { return x>=y ? x : y }</code>	<code>def max2 = { x, y=0 -> return x>=y ? x : y }</code>	
überladbar?	ja		nur über Multimethoden
Aufruf		<code>println max(3, 5)</code>	
Parameter beim Aufruf weglassbar?			nur bei Defaultwert, z.B.: <code>println max2(3)</code>
an Variable zuweisbar?	nur über Adressoperator <code>this.&</code> bzw. <code>ref.&</code> → wird dadurch zur Closure		
als Funktionsparameter übergebbar?			
als Funktionsergebnis zurückgebbar?			ja

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

47

Umwandlung Funktion / Methode → Closure

- mit Adressoperator &
 - bei globalen Funktionen: `this.&funktionsname`
 - bei Methoden eines Objekts `ref. ref.&methodenname`
 - bei Klassenmethoden einer Klasse K: `K.&methodenname`

- Beispiel: "Rekursive" Closure:

```
def spiegeln(int x) {
    return x ? "${x%10}${spiegeln(x.intdiv(10))}${x%10}" : "
```

// Nun eine Closure daraus erstellen ..

```
def spiegelnClosure = this.&spiegeln
```

// ... und testen

```
println spiegelnClosure(4711)
```

`this.` darf nicht weggelassen werden!

Umwandlung überladene Methode → Multimethoden-Closure

- mit Adressoperator &

- Beispiel:

```
def spiegeln(int x) {  
    return x ? "${x%10}${spiegeln(x.intdiv(10))}${x%10}" : "|"  
}  
  
def spiegeln(String s) {  
    def len=s.length() // def muss davorstehen, damit's lokale Variable wird  
    switch (len) {  
        case 0: return "|";  
        case 1: return "${s[len-1]}|${s[len-1]}"  
        default: return "${s[len-1]}${spiegeln(s[0..<len-1])}${s[len-1]}"  
    }  
}  
  
// Nun eine Multimethoden-Closure daraus erstellen ..  
def spiegelnClosure = this.&spiegeln  
// ... und testen  
println spiegelnClosure(4711)  
println spiegelnClosure("4711")
```

nun haben wir eine überladene
Closure (Multimethoden-Closure)

Übergabe von Closures an Funktionen / Methoden (1/3)

- Definition einer Funktion / Methode mit Closure-Parameter
- Typ des Arguments: **Closure**, **def** oder gar nicht angegeben
- Beispiel:

```
def applyForEachChar(string, closure) {  
    def result=""  
    for(i in 0..<string.length())  
        result += closure(string[i])  
    return result  
}
```

Übergabe von Closures an Funktionen / Methoden (2/3)

- Übergabe einer Closure als Parameter
- anonym:
 - `println applyForEachChar("Thomas Specht") { it.toLowerCase()}`
 - nur möglich, wenn Closure letzter Parameter ist
 - Closures möglichst als letzten Parameter
 - `println applyForEachChar("Thomas Specht", { it.toLowerCase()})`
 - geht immer
- via Variable
 - `def closure2={ it.toLowerCase()}`
 - `println applyForEachChar("Thomas Specht",closure)`
- via Adressoperator auf eine Funktion
 - `def lower(c) {`
 - `return c.toLowerCase()`
 - `}`
 - `println applyForEachChar("Thomas Specht",this.&lower)`

Übergabe von Closures an Funktionen / Methoden (3/3)

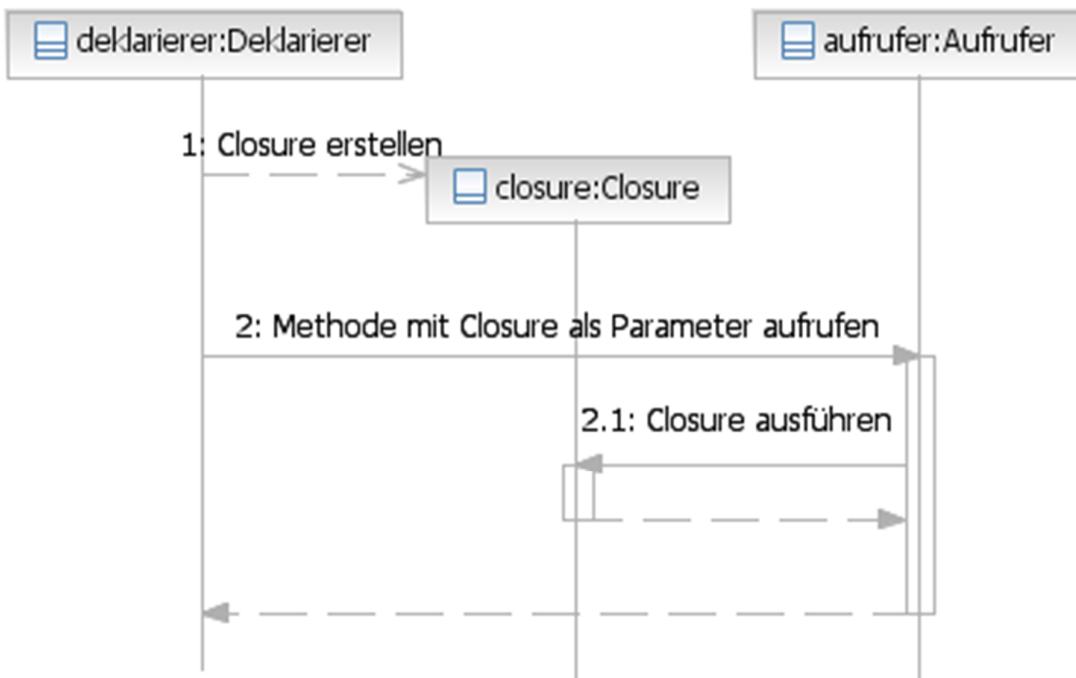
- Gesamtlösung mit selbstgeschriebener Umwandlung in Kleinschrift

```
def applyForEachChar(string, closure) {
    def result=""
    for(i in 0..<string.length()) {
        result += closure(string[i])
    }
    return result
} liefert in Groovy einen String(!) ...

def closure = { c ->
    c in 'A'..'Z' ? (c.toCharacter() + ('a'.toCharacter() - 'A'.toCharacter())) as char
    : c
} ... deshalb Typumwandlung nach Character nötig! auch ein String(!) ... ... Typumwandlung nach Character! Integerrechnung Integer in Character rückverwandeln

println applyForEachChar("Thomas Specht",closure)
```

Ablauf beim Aufruf einer Closure



Sichtbarkeit von Variablen in Closures

- **Deklarationsort** der Closure entscheidet über Sichtbarkeit von Variablen
 - Instanz- und Klassenvariablen der Klasse, in der sie deklariert ist
 - Funktionsparameter und lokale Variablen, wenn sie in Methode deklariert sind
→ vergleichbar mit Sichtbarkeit von Variablen in Methoden
- Empfehlungen
 - Variablenwerte über Closure-Parameter einschleusen
 - Ergebnis über Rückgabewert der Closure zurückgeben
 - Seiteneffekte vermeiden

→ vergleichbar mit Empfehlungen für Methoden

Funktion zur Laufzeitmessung

- **Variante 1:** Auszuführende Befehle als Closure übergeben

```
def zeitmessung(closure) {  
    def start = new Date()  
    closure() // Closure aufrufen  
    def end = new Date()  
    def runtime = end.time - start.time  
    println "$start - $end: $runtime ms"  
}  
  
zeitmessung { for(i=0; i<10000000; i++); }
```

- Auszuführender Code wird als Closure übergeben
- Nachteil: Parameterübergabe an Closure?

Closure zur Laufzeitmessung

- **Variante 2:** Closure zur Laufzeitmessung

```
def startTimer={ name->  
    def start=new Date()  
    return { def end = new Date(); def runtime = end.time - start.time;  
            println "$name: $start - $end: $runtime ms" }  
}  
  
def timer=startTimer("Schleifentimer")  
for(i=0; i<10000000; i++);  
timer() // Closure-Aufruf
```

- **startTimer()**-Funktion liefert Closure zur Laufzeitmessung
 - **name:** Name des Timers
 - **start:**
 - Aufruhrzeit der **startTimer**-Funktion
 - wird bei Rückgabe der Closure bereits ausgewertet eingesetzt
 - Rückgabe: Closure zum Beenden und Auswerten der Laufzeitmessung
- **timer()**-Aufruf ruft Closure zum Beenden und Auswerten der Laufzeitmessung auf
 - **name** und **start**: Werte vom Deklarationszeitpunkt
 - **end** und **runtime**: Werte zum Ausführungszeitpunkt

Closures als Bedingung in Switch-Case-Anweisungen

- Closure
 - bekommt in *it* den Wert der Switch-Variablen
 - liefert
 - true → Zweig ausführen
 - false → Zweig ausführen
- Beispiel

```
i=90
switch(i) {
    case {it>0}: println "positiv"; break;
    case {it==0}: println "null"; break;
    case {it<0}: println "negativ"; break;
}
```

Closure zum Filtern von Collections

- *Collection grep(Closure closure)*
 - liefert Collection mit den Elementen, für die die **Closure** true liefert
 - Beispiel:

```
def noten=[1.3, 1.0, 5.0]
println noten
println "Bestanden: ${noten.grep {it<5.0}}"
```
- Funktioniert auch für Assoziative Arrays (Maps)
 - Zugriff auf Schlüssel mit *it.key*
 - Zugriff auf Wert mit *it.value*
 - Beispiel:

```
def notenA=[824519:1.3, 628928:1.0, 769184:5.0]
println noten
println "Bestanden: ${notenA.grep {it.key<800000 && it.value<5.0}}"
// [628928=1.0]
```



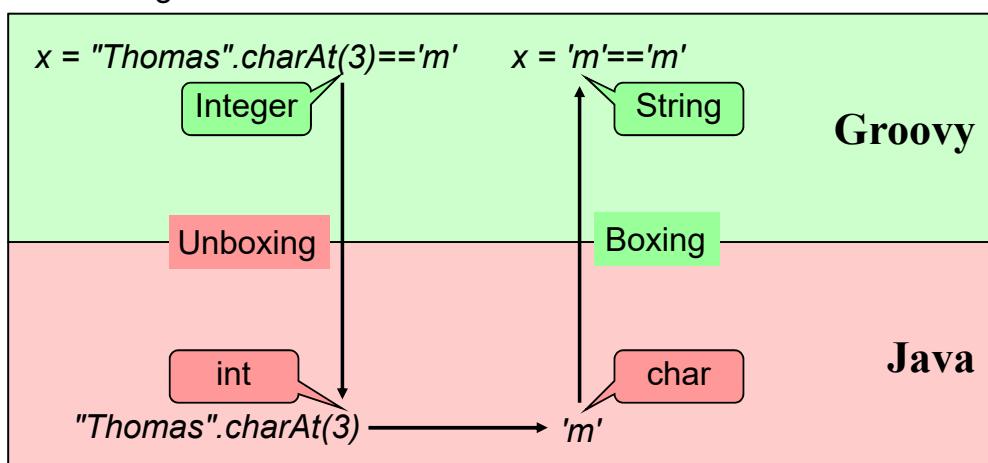
Gliederung

2. Groovy
 1. Einführung in Groovy
 2. Syntax und Kontrollkonstrukte
 3. Funktionen
 4. Closures
 - 5. Datentypen und Operationen**
 6. Collections
 7. Klassen und Groovy Beans
 8. Reguläre Ausdrücke
 9. Metaprogrammierung und Builder
 10. Testen

Elementare Datentypen und Autoboxing

Sämtliche Datentypen sind in Groovy Objekte

- gilt auch für elementare Datentypen!
- *byte, short, int, long, char, float, double und boolean*
 - ersetzt durch Wrapper-Instanzen
 - in Groovy vermeiden, weil sie die Objektorientierung durchbrechen
 - Autoboxing beim Aufruf von Java-Methoden



- Rechenoperationen erfolgen auf Wrapper-Klassen
 - Beispiel: `5+3` steht für `5.plus(3)`

Wichtige Groovy-Datentypen

Typ	Klasse	Verwendung	Beispiel-Literale
Ganzzahl	<i>Byte</i>	\triangleq byte in Java	
	<i>Short</i>	\triangleq short in Java	
	<i>Integer</i>	\triangleq int in Java	29, <i>0xf1fec</i>
	<i>Long</i>	\triangleq long in Java	132 <i>L</i> , 201 <i>I</i>
	<i>BigInteger</i>	beliebige Stellenzahl	165 <i>G</i> , 298 <i>g</i>
Fließkomma-zahl	<i>Float</i>	\triangleq float in Java	1.42 <i>F</i> , 9.13 <i>f</i>
	<i>Double</i>	\triangleq double in Java	1.89 <i>D</i> , 4.2143 <i>d</i>
	<i>BigDecimal</i>	beliebige Stellenzahl	1.23, 1.8e-3
Zeichen(kette)	<i>Character</i>	einzelnes Zeichen	'a'.toCharacter()
	<i>String</i>	String	'Hallo', "Hallo"
Boolesch	<i>Boolean</i>	\triangleq boolean in Java	<i>true</i> , <i>false</i>

Legende:

empfohlene Datentypen
zu vermeidende Datentypen

Vergleichsoperationen

- **==** testet auf inhaltliche Gleichheit
 - entspricht *equals*-Methode
 - gilt auch für Strings, Listen ...
 - überladbar für selbst definierte Klassen
 - Gleichheit der Referenz (also des Zeigers) in Groovy mit *is*-Methode prüfbar
- **!=** testet auf inhaltliche Ungleichheit
 - entspricht *!equals*
 - gilt insbesondere auch für Strings, Listen ...
- **<=>** vergleicht zwei Objekte inhaltlich miteinander
 - entspricht *compare*-Methode:
 - -1, wenn linker Operand kleiner als rechter
 - 1, " linker " größer "
 - 0, " beide Operanden gleich
 - gilt insbesondere auch für Strings
 - überladbar auch für selbst definierte Klassen
- **<, <=, >, >=**
 - funktionieren auch für Strings (lexikalische Reihenfolge)
 - überladbar auch für selbst definierte Klassen

Vergleich von Referenzen

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

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

Überladen der Operatoren (1/3)

Sämtliche Operatoren sind in Groovy überladbar

Operator	Methode	Bereits überladen für
$a + b$	<code>a.plus(b)</code>	<i>Number, String, Collection</i>
$a - b$	<code>a.minus(b)</code>	
$a * b$	<code>a.multiply(b)</code>	
a / b	<code>a.div(b)</code>	<i>Number</i>
$a \% b$	<code>a.mod(b)</code>	<i>Number</i>
$a ^ b$	<code>a.power(b)</code>	<i>Number</i>
$-a$	<code>a.negative()</code>	<i>Number</i>
$+a$	<code>a.positive()</code>	
$a++ bzw.$ $++a$	<code>a.next()</code>	<i>Number, String, Range</i>
$a-- bzw.$ $--a$	<code>a.previous()</code>	

- `next()` und `previous()`
 - geben neu konstruiertes inkrementiertes bzw. dekrementiertes Objekt zurück
 - lassen bestehendes Objekt unverändert!

Überladen der Operatoren (2/3)

Sämtliche Operatoren sind in Groovy überladbar

Operator	Methode	Bereits überladen für
<code>a[b]</code>	<code>a.getValueAt(b)</code>	<code>Object, List, Map, String, Array</code>
<code>a[b]=c</code>	<code>a.putAt(b, c)</code>	<code>Object, List, Map, String, StringBuffer, Array</code>
<code>a b</code>	<code>a.or(b)</code>	Ganzzahl
<code>a & b</code>	<code>a.and(b)</code>	
<code>a ^ b</code>	<code>a.xor(b)</code>	
<code>~a</code>	<code>b.bitwiseNegate()</code>	Ganzzahl (bitweise Negation), <code>String</code> (liefert Pattern für reguläre Ausdrücke)
<code>a<<b</code>	<code>a.leftShift(b)</code>	Ganzzahl (Linksshift) <code>StringBuffer, Writer, File, Socket, List</code> (anhängen)
<code>a>>b</code>	<code>a.rightShift(b)</code>	Ganzzahl
<code>a>>>b</code>	<code>a.rightShiftUnsigned(b)</code>	Ganzzahl

Überladen der Operatoren (3/3)

Sämtliche Operatoren sind in Groovy überladbar

Operator	Methode	Bereits überladen für
<code>switch(a) { case b: }</code>	<code>b.isCase(a)</code>	<code>Object, String, Range, List, Collection, Pattern, Closure</code>
<code>a==b</code>	<code>a.equals(b)</code>	<code>Object</code>
<code>a!=b</code>	<code>! a.equals(b)</code>	
<code>a<b</code>	<code>a.compareTo(b)<0</code>	<code>java.lang.Comparable</code>
<code>a<=b</code>	<code>a.compareTo(b)<=0</code>	
<code>a>b</code>	<code>a.compareTo(b)>0</code>	
<code>a>=b</code>	<code>a.compareTo(b)>=0</code>	
<code>a<=>b</code>	<code>a.compareTo(b)</code>	
<code>a as type</code>	<code>a.asType(Class type)</code>	<code>Collection</code>

Typecast zur erzwungenen Typumwandlung

- wie in Java
- **nur** zwischen
 - Basisklasse und abgeleiteten Klassen (setzt Typprüfung außer Kraft!)
 - Ganzzahldatentypen unterschiedlicher Genauigkeit (Genauigkeitsverlust!)
 - Fließkommadatentypen unterschiedlicher Genauigkeit (Genauigkeitsverlust!)
- **nicht** überladbar für Klassen
- Aufrufsyntax: **(erzwungenerZieltyp)ausdruck**
- Beispiel

```
println ((int)(3/2)) // 1
```

as-Operator zur Konvertierung in anderen Datentyp

- gibt es in Java **nicht**
- in Groovy zwischen
 - **beliebigen** Klassen
 - **selbst definierbare** Umwandlungsfunktion
- Aufrufsyntax: **ausdruck as Zieldatatype**
- Beispiel

```
["Thomas", "Michael", "Rainer", "Gabi"] as Set
```

as-Operator überladen

- in Datentypklasse die Methode `asType(Class TargetClass)` überladen
 - mit `switch(targetClass)` nach Zielklasse unterscheiden
 - für jede Zielklasse einen case-Zweig mit entsprechendem Umwandlungsauftrag
 - alle nicht bekannten Zielklassen an `super.asType(targetClass)` weiterreichen

- Beispiel

```
class Vector2d {  
    BigDecimal x  
    BigDecimal y  
    ...  
    Vector3d asType(Class targetClass) {  
        switch(targetClass) {  
            case Vector3d: new Vector3d(x:x,y:y,z:0); break;  
            default: super.asType(targetClass)  
        }  
    }  
}
```

Zahloperatoren

Division mit /

- schneidet grundsätzlich keine Reste ab
 - auch nicht wenn Zähler und Nenner Ganzzahlen sind
→ inkompatibel zu Java
 - Beispiel
`println 3/2` // 1.5
- Wenn dennoch Ganzzahldivision ohne Rest gewünscht ist:
 - `x.intdiv(y)`
 - Beispiel
`println 3.intdiv(2)` // 1

Ganzzahlmethoden (1/2)

- ermöglichen Zählschleifen via Methodenaufruf
 - *Closure* für alle Zahlen $\leq upperLimit$ durchlaufen
`upto(upperLimit) { Closure }`
 - *Closure* für alle Zahlen $\geq lowerLimit$ rückwärts durchlaufen
`downto(lowerLimit) { Closure }`
 - *Closure* für alle Zahlen bis ausschließlich(!) Grenze mit Schrittweite s durchlaufen
($s > 0$: vorwärts, $s < 0$: rückwärts)
`step(limit, s) { Closure }`
- Beispiele:
 - Zahlen von 3 bis 7 ausgeben, Zählvariable *it*
`3.upto(7) { println it } // 3 4 5 6 7`
 - Zahlen von 7 bis 3 rückwärts ausgeben, Name der Zählvariablen frei wählen
`7.downto(3) { i-> println i } // 7 6 5 4 3`
 - Zahlen von 7 bis 3 rückwärts mit Schrittweite -2 ausgeben
`7.step(3,-2) { println it } // 7 5`

Ganzzahlmethoden (2/2)

- ermöglichen Zählschleifen via Methodenaufruf
 - *Closure n* mal durchlaufen
`n.times() { Closure }`
- Beispiele:
 - 7 Sterne ausgeben
`7.times() { print "*" } // *****`

Datentyp Character: Literale

Character-Literale

- 'c' ist in Groovy ein String!
- "Thomas"[3] ist in Groovy ein String!
- Character-Literale in Groovy nur über Umwege:
`'c'.toCharacter()`

Datentyp Character: Rechenoperationen

- Rechnen mit Character-Codes nur sehr umständlich möglich:
 - Beispiel:

```
def buchstabe='G'
def kleinbuchstabe = buchstabe +'a'-'A'
liefert nicht etwa 'g', sondern 'Ga'
```
 - Verbessert:

```
def buchstabe='G'
def kleinbuchstabe = buchstabe.toCharacter() +'a'.toCharacter()-
'A'.toCharacter()
liefert auch nicht 'g', sondern 103
```
 - Korrekt:

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

Stringliterale

Start / Ende Zeichen	Anzahl	Beispiel	GString-fähig?	Mehrzeilige Strings?	Bedeutung des \ (Backslash)
'	1	'Hallo Thomas'	-	-	\b Rücktaste
	3	""Mehrzeiliger langer String""		ja	\t Tabulator \r Carriage return \n Zeilenvorschub \f Seitenvorschub
"	1	"Hallo \$name"	ja	-	\\" Backslash
	3	"""Mehrzeiliger String für \$name"""		ja	\\$ Dollarzeichen \uhhhh Unicode hex \xooo Unicode oktal \' Apostroph \" Anführungszeichen
/	1	/[a-zA-Z]d/	ja	-	\uhhhh Unicode hex \\$ Dollarzeichen \ i.d.R. kein \\ nötig

- "Das "hier" ist auch ohne Escape-Zeichen erlaubt, \'hier\' muss aber escapet werden"
- "Da 'hier' ist auch ohne Escape-Zeichen erlaubt, \"hier\" muss aber escapet werden"
- /Und sowohl 'hier' als auch "hier" braucht nicht escapet werden/

GStrings

- ermöglichen Ausgabe von Variablen und berechneten Werten in Strings
 - Beispiele:


```
def name="Thomas"
println "Hallo $name, wie geht es Dir?"
-
i=3
j=9
println "$i * $j = ${i*j}"
```
- **\$name** wird ersetzt durch den Wert der Variablen *name*
- **\${ausdruck}** wird ersetzt durch den Wert des Ausdrucks *ausdruck*
 - einzelne Variablenwerte, z.B. `println "Hallo ${name}, wie geht es Dir?"`
 - beliebige Groovy-Ausdrücke, z.B. `ausgabe="Ergebnis: ${i*j}"`
 - vgl. Closure-Syntax
- Jeder **"String"** und jeder **/String/** ist GString, sobald er ein \$-Zeichen enthält, das
 - nicht escapet ist
 - nicht das letzte Zeichen ist
- **'Apostrophierte Strings'** sind grundsätzlich keine GStrings
 - Ausgabe von Variablen und Ausdrücken nur mit Stringverkettung

Stringoperatoren (1/2)

- + Hintereinanderhängen (wie in Java)
- - Erstes Vorkommen des 2. Strings aus 1. String löschen
 - Beispiele: `vorname="Thomas"`
`vorname -= "oma"`
`println vorname` // gibt *Ths* aus
`name="Maja Mayer"`
`name -= "Ma"`
`println name` // gibt ja *Mayer* aus
- Multiplikation String mit Ganzzahl = Mehrfaches Hintereinanderhängen
 - Beispiele: `println "-"*40` // gibt 40 Minuszeichen aus
`vorname="Thomas"`
`vorname *= 40`
`println vorname` // gibt 40 mal hintereinander *Thomas* aus
- Inkrement-Operator ++ ersetzt **letztes** Zeichen durch seinen Unicode-Nachfolger
- Dekrement-Operator -- ersetzt **letztes** Zeichen durch seinen Unicode-Vorgänger
 - Beispiele: `vorname="Thomas"`
`vorname++`
`++vorname`
`println vorname` // Gibt *Thomau* aus

Stringoperatoren (2/2)

- Arrayindizes zum lesenden Zugriff auf einzelne Zeichen oder Zeichenbereiche
 - Index=0: erstes Zeichen
 - Index>0: ab Stringanfang
 - Index<0: ab Stringende rückwärts (-1 für letztes Zeichen)
 - einzelner Index, z.B. [3] für einzelnes (z.B. vierter) Zeichen
 - Indexbereich, z.B. [3..5] für mehrere (z.B. vierter bis sechster) Zeichen
 - oberer Indexwert kann mit ..< ausgeschlossen werden, z.B. [3..<5]
 - Indizes dürfen auch Ausdrücke sein, z.B. [i/3..i/2]
 - Beispiele: `println "Thomas"[3]` // gibt 4. Zeichen (*m*) aus
`println "Thomas"[2..4]` // gibt 3. bis 5. Zeichen (*oma*) aus
`println "Thomas"[2..<4]` // gibt 3. bis 4. Zeichen (*om*) aus
`println "Thomas"[-3..-1]` // gibt drittletztes bis letztes Zeichen (*mas*) aus

Stringmethoden

- `center(n)` zentriert String auf eine Breite von n Leerzeichen
- `center(n, c)` zentriert String auf eine Breite von n Zeichen, füllt mit Zeichen c auf
 - Beispiel: `println "Thomas".center(20, ".") //Thomas.....`
- `contains(substr)` liefert true, wenn `substr` enthalten ist, sonst false
 - Beispiele: `vorname="Thomas"`
`println vorname.contains("oma") // true`
`println vorname.contains("oms") // false`
- `reverse()` dreht Zeichenfolge um
 - Beispiele: `vorname="Thomas"`
`println vorname.reverse() // samohT`
- `size()` liefert Stringlänge (wie `length()`)
 - Beispiele: `vorname="Thomas"`
`println vorname.size() // 6`

StringBuffer-Operatoren (1/2)

- **StringBuffer überlädt []-Operator zum Lesen und Ändern von Zeichen**
- Lesender Zugriff auf einzelne Zeichen(bereiche) wie bei Strings
- Arrayindizes zum Überschreiben einzelner Zeichen(bereiche)
 - einzelner Index, z.B. `[3]` für einzelnes Zeichen
 - Indexbereich, z.B. `[3..5]` für mehrere Zeichen
 - oberer Indexwert kann mit `..<` ausgeschlossen werden, z.B. `[3..<5]`
- Sonderfälle:
 - Reines Einfügen: Indexbereich Länge 0, z.B. `[0..<0]`
`vorname= new StringBuffer("Thomas")`
`vorname[0..<0] = "Hallo "`
`println vorname // Hallo Thomas`
 - Reines Löschen: Ersatzstring Länge 0
`vorname= new StringBuffer("Thomas")`
`vorname[0..1] = "" // Löschen`
`println vorname // omas`

StringBuffer-Operatoren (2/2)

- **StringBuffer überlädt []-Operator zum Lesen und Ändern von Zeichen**
- Lesender Zugriff auf einzelne Zeichen(bereiche) wie bei Strings
- Arrayindizes zum Überschreiben einzelner Zeichenbereiche
 - Indexbereich mit 1 Element, z.B. [3..3] für einzelnes Zeichen
 - einzelner Index, z.B. [3] geht **nicht** mehr!
 - Indexbereich, z.B. [3..5] für mehrere Zeichen
 - oberer Indexwert kann mit ..< ausgeschlossen werden, z.B. [3..<5]
- Sonderfälle:
 - Ersatzstring länger als Bereich: Weitere Zeichen einfügen

```
vorname= new StringBuffer("Thomas")  
vorname[0..0] = "Hallo "           // Überschreiben  
println vorname                  // Hallo homas
```
 - Ersatzstring kürzer als Bereich: Überflüssige Zeichen löschen

```
vorname= new StringBuffer("Thomas")  
vorname[0..1] = "T"                // Löschen  
println vorname                  // Tomas
```



Gliederung

2. Groovy
 1. Einführung in Groovy
 2. Syntax und Kontrollkonstrukte
 3. Funktionen
 4. Closures
 5. Datentypen und Operationen
 - 6. Collections**
 7. Klassen und Groovy Beans
 8. Reguläre Ausdrücke
 9. Metaprogrammierung und Builder
 10. Testen

Collections

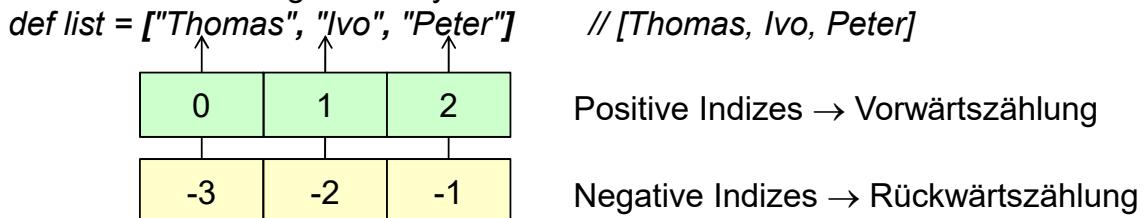
ADT	Eigen-schaf-ten	Abstrakte Basisklasse / Interface	Implementie-rungen (Auszug)	Zugriff		
				Duplikate	sortiert	Num. Index
Array	statisch	(beliebig)	[]	ja	-	ja
	dynamisch					
	assoziativ	Map, AbstractMap	HashMap, TreeMap	-	ja	-
Liste	ja	AbstractSequentialList	LinkedList	(ja)	-	ja
Queue		Queue, AbstractQueue	LinkedList, PriorityQueue			
Stack		(AbstractList)	Stack			
Menge	-	Set, AbstractSet	HashSet, TreeSet	-	-	ja
Sortierte Menge		SortedSet	TreeSet			

* Schlüssel (Keys) müssen jedoch eindeutig sein

Collections: Initialisierung und Elementzugriff

Groovy-Collections

- bauen auf bekannten Java-Collections auf
- erweitern diese um "syntaktischen Zucker", um die Nutzung zu vereinfachen:
 - statische Initialisierung mit Array-Notation:



- Zugriff auf einzelnes Element mit `[` und `]`:

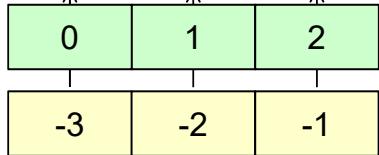
```
println list[1] // Ivo
println list[-1] // Peter
```

- Zugriff auf Bereich mit Range-Indizierung:

```
def sublist1 = list[0..1]; // [Thomas, Ivo]
def sublist2 = list[0..<2]; // [Thomas, Ivo]
def sublist3 = list[1..-1]; // [Ivo, Peter]
def sublist4 = list[2..1]; // [Peter, Ivo]
def sublist5 = list[-1..-2]; // [Peter, Ivo]
```

Collections: Elemente hinzufügen, entfernen

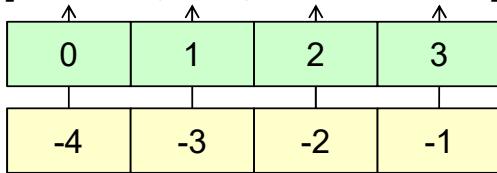
```
def list = ["Thomas", "Ivo", "Peter"]      // [Thomas, Ivo, Peter]
```



- Element hinzufügen mit + und +=

```
list += "Astrid"
```

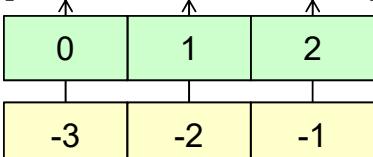
["Thomas", "Ivo", "Peter", "Astrid"]



- Element entfernen mit - und -=

```
list -= "Peter"
```

["Thomas", "Ivo", "Astrid"]



Collections: Positionen ersetzen (1/2)

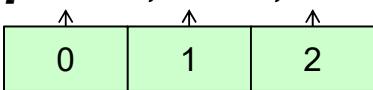
```
def list = ["Thomas", "Ivo", "Peter"]
```



- Positionen ersetzen

```
list[0..1] = ["Markus", "Elena"]
```

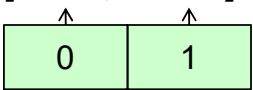
["Markus", "Elena", "Peter"]



- wenn Ersatz kürzer als Original: überzählige Listenelemente werden gelöscht

```
list[0..1] = ["Paul"]
```

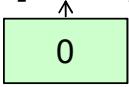
["Paul", "Peter"]



- Spezialfall: Ersatz hat Länge 0 → Listenelemente werden gelöscht

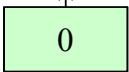
```
list[0..0] = []
```

["Peter"]



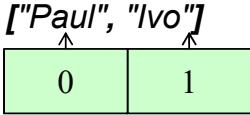
Collections: Positionen ersetzen (2/2)

```
def list = ["Astrid"]
```



- noch Positionen ersetzen

- wenn Ersatz länger als Original: zusätzliche Listenelemente werden eingefügt
 $list[0..0] = ["Paul", "Ivo"]$

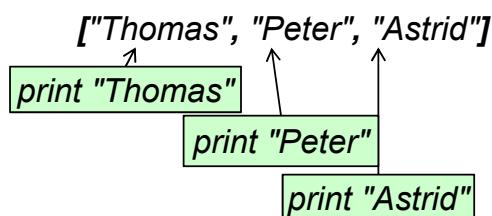


Hier muss ein Range stehen, sonst entsteht verschachtelte Liste!

Collections: Durchiterieren

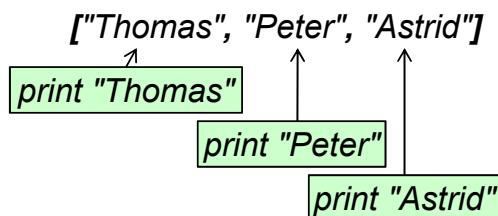
- Iteration durch die Collection per **for**-Schleife

```
for (element in list)  
    print "$element "
```



- Iteration durch die Collection per **each**-Closure

```
list.each { element -> print "$element "} // Thomas Peter Astrid
```



Bequemer Zugriff auf Listen (1/2)

Groovy-Listen

- bauen auf Java-Listen auf (Default: *ArrayList*)
- erweitern diese um "syntaktischen Zucker", um die Nutzung zu vereinfachen:

Aufgabe	Syntax	Listeninhalt
Liste definieren	leer	<code>def leereListe = []</code> []
	initialisiert	<code>def namen = ["Thomas", "Peter", "Helmut", "Miriam", "Paul"]</code> [Thomas, Peter, Helmut, Miriam, Paul]
Entfernen	Element	<code>namen -= "Helmut"</code> [Thomas, Peter, Miriam, Paul]
	Elemente	<code>namen -= ["Miriam", "Paul"]</code> [Thomas, Peter]
Elemente	hinzufügen	<code>namen += ["Markus", "Ivo"]</code> [Thomas, Peter, Markus, Ivo]
	verdoppeln	<code>namen *= 2</code> [Thomas, Peter, Markus, Ivo, Thomas, Peter, Markus, Ivo]
Listen-position	entfernen	<code>namen [1..<4]=[]</code> [Thomas, Thomas, Peter, Markus, Ivo]
	ersetzen	<code>namen [0] = "Elena"</code> [Elena, Thomas, Peter, Markus, Ivo]

Bequemer Zugriff auf Listen (2/2)

Aufgabe	Syntax	Ausgabe
Ganze Liste	<code>println namen</code>	[Elena, Thomas, Peter, Markus, Ivo]
Element zugreifen	von vorne <code>println namen[1]</code>	Thomas
	von hinten <code>println namen[-1]</code>	Ivo
Bereich zugreifen	vorwärts <code>println namen[0..2]</code>	[Elena, Thomas, Peter]
	rückwärts <code>println namen[2..0]</code>	[Peter, Thomas, Elena]
Listenlänge	vorwärts <code>println namen[-2..-1]</code>	[Markus, Ivo]
	rückwärts <code>println namen[-1..-2]</code>	[Ivo, Markus]
Listenlänge	<code>println namen.size()</code>	5
Liste iterieren	for-Schleife <code>for (name in namen) print "\$name "</code>	Elena Thomas Peter Markus Ivo
	each-Closure <code>namen.each { name -> print "\$name "}</code>	Elena Thomas Peter Markus Ivo

Listen durchsuchen

```
def noten=[1.0,1.7,3.0,2.0,4.0, 3.7]
```

- **Alle** Listenelemente liefern, die vorgegebenes Kriterium erfüllen
`println noten.findAll {it<2.0} // [1.0, 1.7]`

Suchkriterium in
Form einer Closure

- **Erstes** Listenelement liefern, das vorgegebenes Kriterium erfüllt
`println noten.find {it>=2.0} // 3.0`

Suchkriterium in
Form einer Closure

Listen sortieren

```
def list = ["Thomas", "Rainer", "Peter"]
```

- Liste sortieren
`list.sort() // ["Peter", "Rainer", "Thomas"]`
- Liste nach Sortierkriterium sortieren, z.B. letzter Buchstabe
`list.sort { it[-1] } // ["Peter", "Rainer", "Thomas"]`

Sortierkriterium in
Form einer Closure

→ eignet sich vorzüglich zum Sortieren von Objekten, z.B. Studenten

Listenquantoren

```
def noten=[1.0,1.7,3.0,2.0,4.0, 3.7]
```

- Prüfen, dass alle Listenelemente ein Kriterium erfüllen: \forall
`println noten.every {it<=4.0} // true`
`println noten.every {it<4.0} // false`
- Prüfen, ob mindestens ein Listenelement ein Kriterium erfüllt: \exists
`println noten.any {it==1.0} // true`
`println noten.any {it==1.3} // false`

Mengen

- können jedes Element nur einmal enthalten
- unterstützen alle bei Listen besprochenen Operatoren und Zugriffe

Mengendefinition:

- unsortiert: `def namen=["Thomas", "Peter", "Helmut", "Miriam", "Ivo"] as Set`
- sortiert: `def namen=["Thomas", "Peter", "Helmut", "Miriam", "Ivo"] as SortedSet`

Bereiche (Ranges) (1/2)

- Datentyp *Range*
- haben wir beim Zugriff auf Listenelemente schon benutzt: `println namen[0..2]`
- benötigen aufzählbaren Datentyp, der
 - Interface *Comparable* implementiert
 - Operatoren ++ und --, d.h. Methoden *next()* und *previous()* definiert
- Anwendungen:
 - Indizierung von Arrays, Listen und Mengen
 - Schleifen:


```
def n = 8
for (i in 1..n/2) { print "${i}" }
println()                                // 1 2 3 4
```
 - Switch-Befehle:

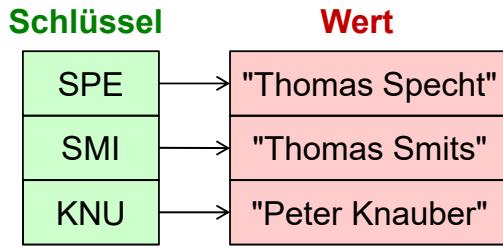

```
c='A'
switch (c) {
    case 'a'..'z': println "Kleinbuchstabe"; break;
    case 'A'..'Z': println "Grossbuchstabe"; break;
    default: println "Sonderzeichen"
}
```

Bereiche (Ranges) (2/2)

Bereich	Syntax	Bereichsinhalt / Ausgabe
Zahlenfolge	aufsteigend <code>range = 0..2</code> <code>range = 0..<2</code>	[0, 1, 2] [0,1]
	absteigend <code>range = 2..0</code> <code>range = 2..<0</code>	[2, 1, 0] [2,1]
Buchstaben -folge	aufsteigend <code>range = 'a'..'f'</code> <code>range = 'a'..<'f'</code>	[a, b, c, d, e, f] [a, b, c, d, e]
	absteigend <code>range = 'f'..'a'</code> <code>range = 'f'..<'a'</code>	[f, e, d, c, b, a] [f, e, d, c, b]
Bereichs- element	auslesen <code>println range[0]</code>	f
	ändern	nicht möglich (Exception)
Bereichslänge	<code>println range.size()</code>	5
Prüfen, ob Wert im Bereich enthalten	<code>println range.contains('a')</code> <code>println range.contains('b')</code>	false true
Bereich iterieren	for-Schleife <code>for (c in 's'..'z') { print "\$c" }</code> <code>for (c in 'z'..'s') { print "\$c" }</code>	s t u v w x y z z y x w v u t s
	each()-Closure <code>range.each { c -> print "\$c" }</code>	f e d c b

Assoziative Arrays (Maps)

- bestehen aus Schlüssel-Werte-Paaren, bestehend aus
 - alphanumerischem Schlüssel (Key)
 - beliebigem Wert (Value)
- Beispiel:
`def profs= [SPE:"Thomas Specht", SMI:"Thomas Smits", KNU:"Peter Knauber"]`



- Anwendung
 - schneller Zugriff auf Datensätze über **Schlüsselattribut**
 - einfache Verzeichnisse (z.B. Zuordnung von **Raumnummern** zu **Namen**)
 - Zählen der **Häufigkeit** von **Wörtern**
 - Zugriff auf GET- und POST-Parameter von HTML-Formularen

Nutzung Assoziativer Arrays (1/2)

Aufgabe	Syntax	Ausgabe
Assoziativer Array definieren	<code>leer</code>	<code>def leereMap=[:]</code>
	<code>initialisiert**</code>	<code>def vor = ['GDI' : 'Grundlagen der Informatik', 'WBT' : 'Webtechnologien']</code>
Element zugreifen	Index	<code>println vor['GDI']</code>
	Property	<code>println vor.WBT</code>
	Methode	<code>println vor.get('WBT')</code>
		<code>println vor.get('OOT')</code>
	Methode mit Defaultwert*	<code>println vor.get('VSY', 'Verteilte Architekturen')</code>
Elemente hinzufügen**	<code>vor += ['LAL':'Lineare Algebra']</code>	
Element entfernen	<code>vor.remove 'GDI'</code>	

* wenn **Schlüssel** noch nicht existiert, wird er mit dem **Defaultwert** nebenbei angelegt

** Apostrophe dürfen beim **Key** weggelassen werden(!), wenn **Key** vom Typ String
 → Wenn **Key** Variable ist, muss diese (**eingeklammert**) werden, z.B.
`def kuerzel='WBT'; def name="Webtechnologien"; def vor=[(kuerzel),name]`

Nutzung Assoziativer Arrays (2/2)

Aufgabe	Syntax	Ausgabe
Anzahl Elemente	<code>println vor.size()</code>	3
Assoziatives Array iterieren	for-Schleife <code>for (v in vor) { println "\${v.key} = \${v.value}" } </code>	WBT = Webtechnologien VSY = Verteilte Systeme KPT = Komponententechnologien
	each-Closure <code>vor.each { key, value -> println "\$key = \$value" } </code> <code>vor.each { pair -> println "\$pair.key = \$pair.value" } </code>	
Menge aller Schlüssel	<code>println vor.keySet()</code>	[WBT, VSY, KPT]
Collection aller Werte	<code>println vor.values()</code>	[Webtechnologien, Verteilte Systeme, Komponententechnologien]

Assoziative Arrays durchsuchen

- `def profs= [SPE:"Thomas Specht", SMI:"Thomas Smits", KNU:"Peter Knauber"]`
- **Alle** Arrayelemente liefern, die vorgegebenes Kriterium erfüllen
 - nur Schlüssel prüfen:
`println profs.findAll { it ==~ /K.*/ } // [KNU: Peter Knauber]`

Schlüssel beginnt mit K
 - auch Wert prüfen:
`println profs.findAll { key, value-> value==~ /Thomas.*/ } // [SPE: Thomas Specht, SMI: Thomas Smits]`

Wert beginnt mit Thomas
- **Ein** Arrayelement liefern, das vorgegebenes Kriterium erfüllt

`println profs.find { it==~ /S.*/ } // [SPE=Thomas Specht]`

`println profs.find { key, value -> value ==~ /P.*/ } // [KNU=Peter Knauber]`



Gliederung

2. Groovy
 1. Einführung in Groovy
 2. Syntax und Kontrollkonstrukte
 3. Funktionen
 4. Closures
 5. Datentypen und Operationen
 6. Collections
 - 7. Klassen und Groovy Beans**
 8. Reguläre Ausdrücke
 9. Metaprogrammierung und Builder
 10. Testen

Klassendefinition

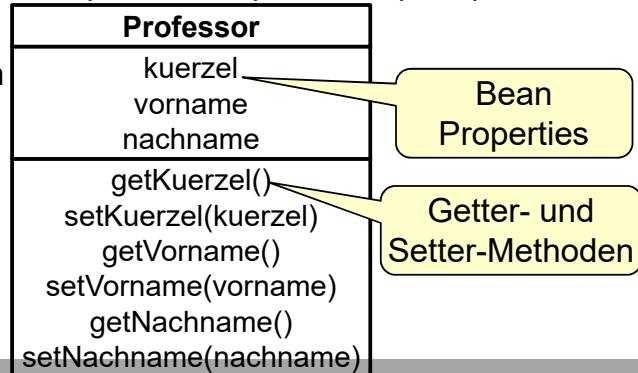
Klassendefinition wie in Java, aber

- Klasse implizit *public*, wenn nicht *protected* oder *private* angegeben
- mehrere Klassen und Skriptcode dürfen in einer Datei definiert werden
→Ersatz für (bislang) fehlende *Inner Classes*
- (bislang) keine *Inner Classes*
→Hilfsklassen in selbe Datei auf obere Ebene als Ersatz
→Closures ersetzen anonyme Inner Classes

Groovy Beans sind Groovy Klassen mit

- Konstruktor mit Map als Parameter
 - implizit, wenn keine eigenen Konstruktoren definiert
 - ansonsten selbst definieren
- Getter- und Setter-Methoden für alle Attribute (*Bean Properties*)
 - implizit für Attribute ohne *public / protected / private* - Angabe
 - ansonsten selbst definieren:

```
def getAttributname()           // Statt "def" kann auch Attributtyp stehen
def setAttributname(value)      // Statt "def" kann auch void stehen
                               // Vor "value" kann auch def oder Datentyp stehen
- Getter- und Setter ohne Angabe von public / protected / private implizit public
```
- serialisierbar
 - implements Serializable* dranschreiben



Ablauf beim Attributzugriff

- **direkter Attributzugriff *ref.attrName***
 - ruft **explizit** definierten **Getter** auf, sofern vorhanden
 - ruft **impliziten Getter** auf, wenn kein expliziter vorhanden
 - greift direkt auf das **Attribut** zu, wenn weder expliziter noch impliziter Getter
 - gilt insbesondere, wenn Attribut weder *public/protected/private*
- Aufruf des **Getters** *ref.getAttributeName()*
 - ruft **explizit** definierten **Getter** auf, sofern vorhanden
 - ruft **impliziten Getter** auf, wenn kein expliziter vorhanden
 - sonst Exception
 - gilt insbesondere, wenn Attribut weder *public/protected/private*
- Attributzuweisung / Setter analog
- **Achtung: Groovy ignoriert Zugriffsrechte**
 - Zugriff auch auf Private-Attribute problemlos möglich
 - Grund: Attribute ohne *public/protected/private* können
 - als privat angesehen werden
 - öffentlicher Zugriff über (explizite oder implizite) öffentliche Getter/Setter
 - Getter/Setter bei Bedarf entsprechend implementieren, sonst weglassen
 - Verbesserung in Groovy 3.0?

Attributdefinition

Attributdefinition wie in Java, aber

- Attribute implizit *public*, wenn *public* / *protected* / *private* fehlt
 - dann implizite Getter- und Settermethoden gemäß GroovyBeans-Standard
- **def** statt Datentypangabe möglich
 - Beispiel:

```
class Complex {  
    def re  
    def im  
}
```

Konstruktoren

Konstruktoren wie in Java, aber

- Konstruktoren implizit *public*, wenn nicht *public* / *protected* / *private* angegeben
- Wenn **kein** anderer Konstruktor definiert ist:
 - impliziter Konstruktor mit benannten formalen Parametern für alle Attribute:
 - Syntax: *Klassenname(attrName1: attrValue1, attrName2: attrValue2)*
 - Attribute ohne Initialisierungsparameter werden *null*
→impliziter parameterloser Konstruktor wie in Java

• Beispiel:

```
class Complex {  
    def re  
    def im  
}  
Attributname  
Attributwert
```

```
def c1 = new Complex(re:9,im:3)  
println "$c1.re $c1.im"      // 9 3  
def c2= new Complex(re:4)  
println "$c2.re $c2.im"      // 4 null  
c3 = new Complex(im:3)  
println "$c3.re $c3.im"      // null 3
```

Methodendefinition

Methodendefinition wie in Java, aber

- Methoden implizit *public*, wenn nicht *public* / *protected* / *private* angegeben
- *def* statt Rückgabetyp möglich
- Parameterdatentypen dürfen fehlen (oder durch *def* ersetzt werden)
- Beispiel:

```
class Complex {  
    def re  
    def im  
    def length() {  
        return Math.sqrt(re*re+im*im)  
    }  
}
```

Sichtbarkeit lokaler Variablen

Lokale Variablen

- mit *def* oder konkretem Datentyp innerhalb Funktion, Closure, Script definiert
- lokal in Bezug auf innerste(n)
 - Befehlsblock { } → sichtbar nur in diesem Befehlsblock und dessen Kindern
 - Funktion → " " " dieser Funktion
 - Closure → " " " dieser Closure
 - Script → " " " diesem Script (nicht in Funktionen / Closures!) in dem / der sie definiert wurden
- erst ab der Stelle, an der sie definiert wurden, sichtbar
- entsprechen lokalen Variablen in Java
 - lokale Variablen des Scripts entsprechen lokalen Variablen der *main*-Methode

Sichtbarkeit von Instanzvariablen

- Variablen auf Klassenebene
 - mit `def` oder konkretem Datentyp
→ Groovy Bean-Property mit impliziten Gettern und Settern
 - mit `public`, `protected` oder `private` gekennzeichnet
- entsprechen Instanzvariablen in Java
- Sichtbarkeitssteuerung über `public`, `protected` oder `private`
 - `public`: überall sichtbar (default)
 - `protected`: sichtbar in eigener und abgeleiteten Klassen
 - `private`: sichtbar in nur eigener Klasse
 - ohne: Bean Property mit impliziter public Getter- und Setter-Methode
- Zugriff aus Methoden und Closures der eigenen oder abgeleiteten Klassen:
 - `variablename`
 - `this.variablename`
- Zugriff von außen über eine Objektreferenz `ref`:
 - `ref.variablename`
 - `ref.getVariablename()` // Impliziter Getter zum Lesen der Variablen
 - `ref.setVariablename(wert)` // Impliziter Setter zum Beschreiben der Variablen

Sichtbarkeit von Klassenvariablen

- mit `static` definierte Variablen auf Klassenebene
- entsprechen Klassenvariablen in Java
- Sichtbarkeitssteuerung über `public`, `protected` oder `private`
 - `public`: überall sichtbar (default)
 - `protected`: sichtbar in eigener und abgeleiteten Klassen
 - `private`: sichtbar in nur eigener Klasse
 - ohne: statische Bean Property mit impliziter public Getter- und Setter-Methode
- Zugriff in Methoden und Closures der eigenen oder abgeleiteten Klassen:
 - `variablename`
 - `this.variablename`
- Zugriff von außen (`ref`: Referenz auf eine Instanz dieser Klasse)
 - `Klassenname.variablename`
 - `Klassenname.getVariablename()` // Impliziter Getter zum Lesen der Var.
 - `Klassenname.setVariablename(wert)` // Impliziter Setter zum Beschreiben r.
 - `ref.variablename`
 - `ref.getVariablename()` // Impliziter Getter zum Lesen der Var.
 - `ref.setVariablename()` // Impliziter Setter zum Beschreiben der Var.

Sichtbarkeit von Globalen Variablen

- ohne `def` und ohne Datentypangabe definiert außerhalb einer Klasse
- sichtbar in
 - Scriptcode derselben Datei
 - ab der Stelle der Definition
 - allen globalen Funktionen / Closures derselben Datei
 - unabhängig von der Reihenfolge der Definition
- nicht sichtbar in
 - Klassen und deren Methoden / Closures (auch nicht derselben Datei!)
- entsprechen etwa `static`-Variablen der Klasse mit der `main`-Methode in Java

Zugriff auf Attribute eines Objekts

Syntax	Beispiel	Attribut-name	Beschreibung
<code>obj.attrName</code>	<code>println prof.vorname</code>	statisch	wie in Java
<code>obj.'attrName'</code>	<code>println prof.'vorname'</code>		in Apostrophen
<code>obj."attrName"</code>	<code>println prof."vorname"</code>		in Anführungszeichen
<code>def attr="attrName"</code> <code>obj."\$attr"</code>	<code>def attr="vorname"</code> <code>println prof."\$attr"</code>	dynam.	Attributname als GString
<code>obj.getAttribute()</code>	<code>println prof.getVorname()</code>		Groovy Bean Getter-Methode
<code>obj.setAttribute(value)</code>	<code>prof.setVorname("Jens")</code>	statisch	
<code>obj['attrName']</code> <code>obj["attrName"]</code>	<code>prof['vorname']="Jens"</code> <code>prof["vorname"]="Jens"</code>	Setter-Methode	
<code>def attr="attrName"</code> <code>obj[attr]</code>	<code>def attr="vorname"</code> <code>obj[attr]="Jens"</code>		dynam.
			Map Syntax wie in Maps / Assoziativen Arrays

- Analoges gilt auch für den Zugriff auf die Closures eines Objektes
 - sind ja praktisch auch Attribute mit Referenz auf Closure

Aufruf der Methoden eines Objekts

Syntax	Beispiel	Attribut-name	Beschreibung
<code>obj.methodName()</code>	<code>prof.setKuerzel("SPE")</code>	statisch	wie in Java
<code>obj.'methodName'()</code>	<code>prof.'setKuerzel'("SPE")</code>		in Apostrophen
<code>obj."methodName"()</code>	<code>prof. "setKuerzel"("SPE")</code>		in Anführungszeichen
<code>def mn="methodName"</code> <code>obj."\$mn"()</code>	<code>def mn="setKuerzel"</code> <code>prof. "\$mn"("Jens")</code>	dynam.	Methodenname als GString

Aufruf der Konstruktoren einer Klasse

Syntax	Beispiel	Beschreibung
<code>new Klassenname()</code>	<code>new Professor()</code>	Parameterloser Konstruktorauftruf
<code>new Klassenname (val1, val2)</code>	<code>new Professor ("SPE", "Thomas", "Specht")</code>	Positionsparametern
<code>def obj= [val1, val2] as Klassenname</code>	<code>def specht= ["SPE", "Thomas", "Specht"] as Professor</code>	
<code>Klassenname obj= [val1, val2]</code>	<code>Professor specht= ["SPE", "Thomas", "Specht"]</code>	Konstruktorauftruf mit benannten Parametern (keine anderen Konstruktoren dürfen definiert sein)
<code>new Klassenname (attr1:val1, attr2:val2)</code>	<code>new Professor (kuerzel:"SPE")</code>	
<code>def obj= [attr1:val1, attr2:val2] as Klassenname</code>	<code>def specht= [kuerzel:"SPE"] as Professor</code>	
<code>Klassenname obj= [attr1:val1, attr2:val2]</code>	<code>Professor specht= [kuerzel:"SPE"]</code>	

Sicheres Dereferenzieren mit ?.

- Motivation:
 - Objektreferenzen können *null* sein
 - bei jedem Attributabruf *objRef.getAttributeName* auf *objRef!=null* prüfen
 - " " Methodenaufruf *objRef.method()* " " "
 - umständlicher Code
- Beispiel:

```
Professor knauber
if (knauber!=null && knauber.vorname)
    println knauber.vorname
```
- Lösung: Sicheres Dereferenzieren mit **?**. statt .
 - Attributabruf über Nullreferenz liefert null, statt Exception
 - Methodenaufruf " " wird ignoriert und " " "
- Beispiel:

```
Professor knauber
if (knauber?.vorname)
    println knauber.vorname      // Gibt Vorname aus, wenn dieser existiert
    println knauber?.vorname     // Gibt Vorname aus, bzw. null wenn nicht existiert
```

Expando

Expando: Dynamisches Objekt

- instanziert aus der Groovy-Klasse *Expando*
- beliebige Attribute hinzufügen durch
 - Angabe als benannter Parameter im Konstruktorauftrag
 - bloßes Beschreiben eines noch nicht vorhandenen Attributs
- Beispiel:

```
Attributname   Attributwert
c1 = new Expando(re:9,im:3)

Attributname   Attributwert
c1.name = "Mein Expando"
println "$c1.name: $c1.re $c1.im"           // Mein Expando: 9 3
```



Gliederung

- 2. Groovy
 - 1. Einführung in Groovy
 - 2. Syntax und Kontrollkonstrukte
 - 3. Funktionen
 - 4. Closures
 - 5. Datentypen und Operationen
 - 6. Collections
 - 7. Klassen und Groovy Beans
 - 8. Reguläre Ausdrücke**
 - 9. Metaprogrammierung und Builder
 - 10. Testen

Match-Operator: String auf Formatpattern testen

- Testet kompletten String auf Einhaltung eines regulären Ausdrucks
 - Syntax: `booleanResult = testString === pattern`

zu überprüfender String

Regulärer Ausdruck als Vergleichsmuster (Pattern)

Match-Operator,
liefert *true* oder *false*
 - Parameter:
 - `testString`: zu überprüfender String
 - `pattern`: Regulärer Ausdruck, der erlaubte Strings formal beschreibt
→ als */Slashy String/*, um Probleme mit Escape-Zeichen \ zu vermeiden
 - Ergebnis:
 - `true`, wenn **kompletter** `testString` exakt zum regulären Ausdruck `pattern` passt
 - `false` sonst
 - Beispiele:
 - `println ('abc' === /[abc]/)` // false (Pattern deckt nur ein Zeichen ab)
 - `println ('abc' === /[abc]*/)` // true (beliebig lange Buchstabenfolge aus a, b, c)
 - `println ('abc' === /^a.*/)` // true (alles, was mit a beginnt)

Reguläre Ausdrücke: Spezielle Zeichen

Pattern	Bedeutung	Beschreibung
x	Einzelnes Zeichen x	Das Zeichen g
\	Backslash-Zeichen	Einzelner Backslash
\t	Tabulatorzeichen (\u00009)	Einzelner Tabulatorsprung
\n	Zeilenvorschub (\u000a)	Direkt am Beginn einer neuen Zeile
\r	Zeilenrücklauf (\u000d)	Direkt nach Zeilenrücklauf
\f	Seitenvorschub (\u000c)	Direkt nach Seitenvorschub
\e	Escape-Zeichen (\001b)	Escape-Steuerzeichen

'a' ==~ /a/
 'b' ==~ /a/
 'aa' ==~ /a/

 'Thomas' ==~ /Thomas/
 'thomas' ==~ /Thomas/
 'Thomase' ==~ /Thomas/

 'Thomas' ==~ /.*\nThomas/
 'Hallo Thomas' ==~ /.*\nThomas/

 ""Hallo
 Thomas"" ==~ /.*\nThomas/

'C:\temp' ==~ /C:\temp/
 /C:\temp/ ==~ /C:\temp/

In Slashy Strings
 \ nicht escapen

In Patterns muss \
 escapet werden

/C:\temp/ ==~ /C:\temp/

Reguläre Ausdrücke: Zeichenklassen

Pattern	Bedeutung	Beispiel
[abc]	eines der genannten Zeichen	genau ein a, b oder c
[^abc]	1 anderes als genannte Zeichen	beliebiges Zeichen ungleich a, b, c
[a-zA-Z]	Bereich(e) inkl. Grenzwerte	ein Groß- oder Kleinbuchstabe a-z
[a-d[m-o]]	gleichbedeutend [a-dm-o]	einer der Kleinbuchst. abcdmno
[[a-z]&&[def]]	Schnittmenge zweier Bereiche*	d, e oder f
[[a-z]&&[^def]]	Differenzmenge zweier Bereiche*	Kleinbuchstabe außer d, e und f
[a-z&&[^d-f]]	gleichbedeutend [a-z&&[^def]]	Kleinbuchstabe außer d, e und f

'a' ==~ /[abc]/ 'a' ==~ /[a-cA-C]/ 'a' ==~ /[[a-z]&&[def]]/ 'a' ==~ /[[a-z]&&[^a-c]]/ 'a' ==~ /[[a-z]&&[^a-c]]/\\
 'c' ==~ /[^abc]/ 'c' ==~ /[^a-cA-C]/ 'b' ==~ /[^a-z]&&[def]/ 'c' ==~ /[^a-z]&&[def]/ 'c' ==~ /[^a-z]&&[def]/\\
 'd' ==~ /[^abc]/ 'd' ==~ /[^a-cA-C]/ 'c' ==~ /[^a-z]&&[def]/ 'd' ==~ /[^a-z]&&[def]/ 'd' ==~ /[^a-z]&&[def]/\\
 'A' ==~ /[^abc]/ 'A' ==~ /[^a-cA-C]/ 'd' ==~ /[^a-z]&&[def]/ 'A' ==~ /[^a-z]&&[def]/ 'A' ==~ /[^a-z]&&[def]/\\
 'C' ==~ /[^abc]/ 'C' ==~ /[^a-cA-C]/ 'A' ==~ /[^a-z]&&[def]/ 'C' ==~ /[^a-z]&&[def]/ 'C' ==~ /[^a-z]&&[def]/\\
 'D' ==~ /[^abc]/ 'D' ==~ /[^a-cA-C]/ 'D' ==~ /[^a-z]&&[def]/ 'D' ==~ /[^a-z]&&[def]/ 'D' ==~ /[^a-z]&&[def]/\\
 'D' ==~ /[^abc]/ 'D' ==~ /[^a-cA-C]/ 'D' ==~ /[^a-z]&&[def]/ 'D' ==~ /[^a-z]&&[def]/ 'D' ==~ /[^a-z]&&[def]/\\
 'A' ==~ /[^abc]/ * && nur zur Verknüpfung von Zeichenbereichen in [] anwendbar!

Reguläre Ausdrücke: Vordefinierte Zeichenklassen

Pattern	Bedeutung	matcht	matcht nicht
.	1 beliebiges Zeichen außer Zeilenumbruch	β	
\d	Eine Ziffer [0..9]	3	a
\D	Ein Zeichen, das <i>keine</i> Ziffer ist	a	0
\s	Ein Whitespace [\0b\t\r\n\f]	Space, Tab, CR	<
\S	Ein Zeichen <i>außer</i> Whitespace)	\n
\w	Eines der Zeichen [a-zA-Z0-9_]	h	ä
\W	Ein Zeichen außer [a-zA-Z0-9_]	ä	h

'9' ==~ ./	'0' ==~ \d/	'' ==~ \s/	'a' ==~ \w/
'19' ==~ ./	'1' ==~ \d/	'\n' ==~ \s/	'aa' ==~ \w/
'a' ==~ ./	'11' ==~ \d/	'a' ==~ \s/	'' ==~ \w/
'abc' ==~ ./	'a' ==~ \d/	'1' ==~ \s/	'1' ==~ \w/
'0' ==~ \D/		'' ==~ \S/	'a' ==~ \W/
'1' ==~ \D/		'\n' ==~ \S/	'aa' ==~ \W/
'11' ==~ \D/		'a' ==~ \S/	'' ==~ \W/
'a' ==~ \D/		'1' ==~ \S/	'1' ==~ \W/
'abc' ==~ \D/			

Reguläre Ausdrücke: Begrenzungen, fressen **kein** Zeichen

Pattern	Bedeutung	Beschreibung
\b	Wortgrenze	alle außer [a-zA-Z0-9_]
\B	Keine Wortgrenze	[a-zA-Z0-9_]
\A	Beginn der Eingabe / Datei	Zeilenbeginn reicht <i>nicht</i>
\Z	Ende der Eingabe (auch als \z)	Zeilenende reicht <i>nicht</i>

'T Specht' ==~ /.+\bS.+/	'T Specht' ==~ /.+\BS.+/	'Thomas' ==~ \A Thomas/
'T.Specht' ==~ /.+\bS.+/	'T.Specht' ==~ /.+\BS.+/	'Hallo Thomas' ==~ \A Thomas/
'T Specht' ==~ /.+\bS.+/	'T Specht' ==~ /.+\BS.+/	"Hello
'TSpecht' ==~ /.+\bS.+/	'TSpecht' ==~ /.+\BS.+/	Thomas" ==~ \A Thomas/
'T9Specht' ==~ /.+\bS.+/	'T9Specht' ==~ /.+\BS.+/	
'T_Specht' ==~ /.+\bS.+/	'T_Specht' ==~ /.+\BS.+/	'Thomas' ==~ /.^ Thomas\Z /
		'Hallo Thomas' ==~ /.^ Thomas\Z /
		"Hello Thomas wie gehts" ==~ /.^ Thomas\Z /

Reguläre Ausdrücke: Quantifizierer

Pattern	Bedeutung	Beispiel		
		Pattern	matcht	matcht nicht
$X?$	0 oder einmal	$a?$	a	aa
X^*	0 oder einmal oder beliebig oft	a^*	aa	b
X^+	einmal oder mehrmals	a	a	
$X\{n\}$	genau n mal	$a\{3\}$	aaa	aa
$X\{n,\}$	mindestens n mal	$a\{3,\}$	$aaaa$	aa
$X\{n,m\}$	mindestens n und höchstens m mal	$a\{3,4\}$	aaa	$aaaa$

" ==~ /a?/ " ==~ /a*/ " ==~ /a{2}/ " ==~ /a{2,4}/
 'a' ==~ /a?/ 'a' ==~ /a*/ 'a' ==~ /a{2}/ 'a' ==~ /a{2,4}/
 'aaa' ==~ /a?/ 'aaa' ==~ /a*/ 'aa' ==~ /a{2}/ 'aa' ==~ /a{2,4}/
 'aaab' ==~ /a?/ 'aaab' ==~ /a*/ 'aaa' ==~ /a{2}/ 'aaa' ==~ /a{2,4}/

 " ==~ /a+/ " ==~ /a{2,}/ 'aa' ==~ /a{2,}/ 'aaaa' ==~ /a{2,4}/
 'a' ==~ /a+/ 'a' ==~ /a{2,}/ 'aa' ==~ /a{2,}/ 'aaaa' ==~ /a{2,4}/
 'aaa' ==~ /a+/ 'aa' ==~ /a{2,}/ 'aaa' ==~ /a{2,}/

Reguläre Ausdrücke: Logische Operatoren

Pattern	Bedeutung	Beispiel		
		Pattern	matcht	matcht nicht
XY	X, gefolgt von Y	ab	ab	a
$X Y$	X oder Y^*	$a b$	a	ab
(X)	X als zusammengehörige Gruppe	$(ab)^*$	ab	aba

* logische Oder-Verknüpfung für einzelnes Zeichen oder ganze Zeichengruppe

" ==~ /ab/ " ==~ /a|b/ " ==~ /(ab)*/
 'a' ==~ /ab/ 'a' ==~ /a|b/ 'a' ==~ /(ab)*/
 'b' ==~ /ab/ 'b' ==~ /a|b/ 'b' ==~ /(ab)*/
 'ab' ==~ /ab/ 'ab' ==~ /a|b/ 'ab' ==~ /(ab)*/
 'ba' ==~ /ab/ 'ba' ==~ /a|b/ 'ba' ==~ /(ab)*/

 'aba' ==~ /(ab)*/ 'abab' ==~ /(ab)*/

Komplexere Beispiele für reguläre Ausdrücke

- Uhrzeit im 12- oder 24-Stunden-Format:
`/[012]?d:[0-5]\d/`
- Datum (max. zweistellige Tag- und Monatsnummer, ohne Jahresangabe)
`/[0123]?d\.[01]?d\./`
- Datum (zweistelliger Tag und Monat, mit Jahresangabe)
`/[0123]?d\.[01]?d\.(d\d){1,2}/`
- Wochentagsangabe
`/(Montag|Dienstag|Mittwoch|Donnerstag|Freitag|Samstag|Sonntag)/`
- E-Mail-Adresse (vereinfacht)
`name = /[a-zA-Z][a-zA-Z0-9_-]*\.[a-zA-Z][a-zA-Z0-9_-]**/
domain = /[a-zA-Z][a-zA-Z0-9_-]*\.[a-zA-Z][a-zA-Z0-9_-]**/
regExp = $name@$domain/`

Find-Operator

- analysiert und zerlegt String anhand eines regulären Ausdrucks
- Syntax: `ergArray= testString =~ pattern`

The diagram illustrates the Find-Operator syntax `ergArray = testString =~ pattern`. It shows four components: `zu zerlegender String` (the string to be analyzed/testString), `Regulärer Ausdruck als Zerlegungsmuster (Pattern)` (the regular expression pattern), `Ergebnis: Array mit Suchergebnissen` (the resulting array of matches), and the `Find-Operator` (`=~`).
- Parameter:
 - `testString`: zu analysierender und zerlegender String
 - `pattern`: Regulärer Ausdruck, der Syntax und Struktur formal beschreibt
→ als `/Slashy String/`, um Probleme mit Escape-Zeichen \ zu vermeiden
- Anmerkungen:
 - nur Teilstring des `testString` muss matchen, nicht gesamter!
→ regulärer Ausdruck kann mehrfach hintereinander in `testString` passen
→ mehrere Ergebniszellen möglich
- Ergebnis:
 - wenn Pattern **ohne** Capture Groups (...) 1-dim Array mit Matches
 - wenn Pattern Capture Groups (...) enthält 2-dim Array (1 Zeile pro Match)

Find-Operator ohne Capture Group

- analysiert und zerlegt String anhand eines regulären Ausdrucks:
 - Syntax: `oneDimArray= testString =~ pattern`
- Ergebnis: 1-dim Array mit Matches

zu zerlegender String

Regulärer Ausdruck als Zerlegungsmuster (Pattern)
- Parameter:
 - `testString`: zu analysierender und zerlegender String
 - `pattern`: Regulärer Ausdruck, der Syntax und Struktur formal beschreibt
 - ohne Capture Groups, d.h. frei von Klammern ()
 - Ergebnis
 - eine Ergebniszeile pro Teilstring des `testString`, der mit `pattern` matcht
 - regulärer Ausdruck kann mehrfach hintereinander in `testString` passen
 - mehrere Ergebniszeilen möglich
 - jedes Matching konsumiert ("frisst") die gematchten Zeichen des `testString`
 - Beispiel:

```
ergebnis= "Die Mittagspause geht von 12:30 bis 13:30" =~ /[0-2]?\\d:[0-5]?\\d/
ergebnis.each { println it }
```

Ausgabe:
12:30
13:30

Find-Operator mit Capture Groups (1/3)

- analysiert und zerlegt String anhand eines regulären Ausdrucks:
 - Syntax: `twoDimArray= testString =~ pattern`
- Ergebnis: 2-dim Array mit Suchergebnissen

zu zerlegender String

Regulärer Ausdruck als Zerlegungsmuster (Pattern)
- Parameter:
 - `testString`: zu analysierender und zerlegender String
 - `pattern`: Regulärer Ausdruck, der Syntax und Struktur formal beschreibt
 - mindestens 1 Capture Group in Klammern ()
 - Ergebnis
 - eine Ergebniszeile pro Teilstring des `testString`, der mit `pattern` matcht,
 - regulärer Ausdruck kann mehrfach hintereinander in `testString` passen
 - mehrere Ergebniszeilen möglich
 - jedes Matching konsumiert ("frisst") die gematchten Zeichen des `testString`
 - Spalte 0: Gematchter Substring
 - für jede *Capture Group* ("Kapergruppe") des regulären Ausdrucks eine Spalte
 - Spalte 1: 1. Capture Group zugeordneter ("gekaperter") Teil des `testString`
 - Spalte 2: 2. Capture Group zugeordneter ("gekaperter") Teil des `testString`
 - usw.

Find-Operator mit Capture Groups (2/3)

- *Capture Group* ("Kapergruppe") innerhalb eines regulären Ausdrucks:
 - Syntax: (X) , wobei X ein beliebiger regulärer Teilausdruck ist
 - jeder rund geklammerte (Teilausdruck) bildet eine *Capture Group*
 - Ausnahme: $(?:X)$ definiert eine Gruppe, die keine *Capture Group* ist

- Beispiel:

```
result = 'Ich bin von 12:00 bis 12:45 beim Mittagessen' =~ /([012]?\d):(\d{2})/
for (match in result) {
  for (group in match[1..<match.size]) {
    print "$group "
  }
  println()
}
```

Nur Capture Groups ausgeben,
auf matchenden Teilstring verzichten

Capture Group 1:
([012]?\d)

Capture Group 2:
(\d{2})

matchender Teilstring ([012]?\d):(\d{2})	Capture Group 1: ([012]?\d)	Capture Group 2: (\d{2})
result[0][0] = '12:00'	result[0][1] = '12'	result[0][2] = '00'
result[1][0] = '12:45'	result[1][1] = '12'	result[1][2] = '45'

Ausgabe:

12 00

12 45

Find-Operator mit Capture Groups (3/3)

- *Capture Group* innerhalb eines regulären Ausdrucks:
 - Syntax: (X) , wobei X ein beliebiger regulärer Teilausdruck ist
 - jeder rund geklammerte (Teilausdruck) bildet eine *Capture Group*
 - Ausnahme: $(?:X)$ definiert eine Gruppe, die keine *Capture Group* ist
- **Dasselbe Beispiel kürzer:**

```
result = 'Ich bin von 12:00 bis 12:45 beim Mittagessen' =~ /([012]?\d):(\d{2})/
result.each { all, group1, group2 -> println "$group1 $group2" }
```

Matchender
Teilstring

Wert
Group 1

Wert
Group 2

Capture Group 1:
([012]?\d)

Capture Group 2:
([012]?\d)

matchender Teilstring ([012]?\d):(\d{2})	Capture Group 1: ([012]?\d)	Capture Group 2: (\d{2})
all= '12:00'	group1= '12'	group2 = '00'
all = '12:45'	group1 = '12'	group2 = '45'

Ausgabe:

12 00

12 45

Quantifizierer-Strategie: greedy, reluctant und possessive

Pattern			Bedeutung
greedy	reluctant	possessive	
$X?$	$X??$	$X?+$	$[0..1]$
X^*	$X^*?$	X^*+	$[0..]$
X^+	$X^+?$	X^++	$[1..]$
$X\{n\}$	$X\{n\}?$	$X\{n\}+$	$[n]$
$X\{n,\}$	$X\{n,\}?$	$X\{n,\}+$	$[n..]$
$X\{n,m\}$	$X\{n,m\}?$	$X\{n,m\}+$	$[n..m]$

X ist einzelnes Zeichen oder geklammerter Ausdruck

Quantifiziererstrategie englisch	deutsch	Sym- bol	Bedeutung
greedy	gierig		Erst mal maximal viele passende Zeichen aus String nehmen, notfalls wieder welche zurückgeben
reluctant	zurückhaltend	?	Erst mal minimal viele passende Zeichen aus String nehmen, notfalls noch welche dazu nehmen
possessive	besitz- ergreifend	+	Maximal viele passende Zeichen aus String nehmen und eisern verteidigen, selbst wenn Pattern scheitert

Quantifizierer-Strategie: Beispiele

Strategie	Pattern	Teststring		
		aabra	aaabra	aaaabra
greedy	(a{1,10})(a+bra)	(a) (abra)	(aa) (abra)	(aaa) (abra)
reluctant	(a{1,10}?) (a+?bra)	(a) (abra)	(a) (aabra)	(a) (aaabra)
possessive	(a{1,10}+) (a++bra)	(aa) †	(aaa) †	(aaaa) †

† Pattern matching scheitert an Possessive-Gier

- Match-Operator $\equiv \sim$
 - liefert bei **greedy** und **reluctant** immer dasselbe (korrekte) Ergebnis
 - kann bei **possessive** false liefern, obwohl das Pattern matcht
- Find-Operator $= \sim$
 - liefert bei **greedy** und **reluctant** immer ein korrektes Ergebniss
 - kann bei **possessive** passende Matchings übersehen
- Empfehlungen
 - Match-Operator: **Greedy**
 - Find-Operator: **Greedy** oder **Reluctant**, je nach Anwendung
 - **Possessive**-Strategie vermeiden

Syntax Quantifizierer-Strategie

- ? (reluctant) und + (greedy)
 - dürfen **hinter** einem Quantifizierer ?, *, +, {} stehen
 - gelten immer nur für diesen **einen** Quantifizierer
 - damit ist die Syntax immer eindeutig:

.?	Optionales Zeichen, greedy-Strategie
.??	Optionales Zeichen, reluctant-Strategie
.?+	Optionales Zeichen, possessive-Strategie
.*	0, 1 oder mehrere Zeichen, greedy-Strategie
.*?	0, 1 oder mehrere Zeichen, reluctant-Strategie
.*+	0, 1 oder mehrere Zeichen, possessive-Strategie
.+	1 oder mehrere Zeichen, greedy-Strategie
.+?	1 oder mehrere Zeichen, reluctant-Strategie
.++	1 oder mehrere Zeichen, possessive-Strategie

Reluctant-Strategie

- am Pattern-Anfang **alle** passenden Zeichen nehmen (wie bei Greedy-Strategie)
 - Grund: String wird von links nach rechts abgearbeitet
 - Pattern-Match zunächst ab String-Index 0 prüfen
 - nur wenn ab String-Index 0 kein Match möglich, dann ab String-Index 1 ...
 - frühestmöglicher Match wird genommen
- mitten im Pattern und am Ende des Patterns **möglichst wenige** Zeichen nehmen

Reluctant-Strategie: Beispiele

- " $aaabb$ " $\approx /a+b/$
 - Match: "aaab"
- " $aaabb$ " $\approx /a+?b/$
 - Match: "aaab"
- " $aaabb$ " $\approx /(a+b+)/$
 - Match: "aaabb"
- " $aaabb$ " $\approx /(a+?b+?)$
 - Match: "aaab"
- "*Das rote Haus neben dem grünen Haus ist neu*" $\approx /(.*)Haus/$ **greedy**
 - Match: "*Das rote Haus neben dem grünen Haus*"
 - CG: "*Das rote Haus neben dem grünen*"
- "*Das rote Haus neben dem grünen Haus ist neu*" $\approx /(.*)?Haus/$ **reluctant**
 - Match 1: "*Das rote Haus*" CG: "*Das rote*"
 - Match 2: "*neben dem grünen Haus*" CG: "*neben dem grünen*"
- "*Das rote Haus neben dem grünen Haus ist neu*" $\approx /(.*)+Haus/$ **possessive**
 - **kein** Match ($.^*$ frisst alles, weil **possessive!**)

Possessive-Strategie

- immer **alle** passenden Zeichen nehmen (wie bei Greedy-Strategie)
 - und keine mehr zurückgeben, auch wenn dadurch Match scheitert

Possessive-Strategie: Beispiele

- "*aaabb*" =~ $/(a+b)b/$
 - Match: "aaabb"

greedy-Strategie
CG: "aaab"
- "*aaabb*" =~ $/(a+b)b^{++}/$
 - Match: "aaabb"

possessive-Strategie
CG: "aaab"
- "*aaabb*" =~ $/(a+b+)b/$
 - Match: "aaabb"

greedy-Strategie
CG: "aaab"
- "*aaabb*" =~ $/(a+b++)b/$
 - **kein** Match

possessive-Strategie
- "*login*" =~ $/(login)?.+/$
 - Match: "login"

greedy-Strategie
CG: ""
- "*login*" =~ $/(login)?+.^/$
 - **kein** Match (matcht auf **alle** Strings außer "login")

possessive-Strategie
- "*login2*" =~ $/(login)?.^/$
 - Match: "login2"

greedy-Strategie
CG: "login"
- "*login2*" =~ $/(login)?+.^/$
 - Match: "login2"

possessive-Strategie
CG: "login"

Anwendungen des Find-Operators (1/2)

Anwendungen des Find-Operators

- Benutzereingaben
 - analysieren
 - auf syntaktische Korrektheit prüfen
 - in einzelne Bestandteile zur Weiterverarbeitung zerlegen

Der Find-Operator kann aber noch mehr!

- liefert in Wirklichkeit gar kein 2-dim. Array, sondern Instanz der Klasse `java.util.regex.Matcher`
- Zugriff über Arrayklammern nur aus Komfortgründen in Groovy hinzugefügt!

Methoden des `java.util.regex.Matcher`

- `int getCount()` Anzahl matchender Teilstrings (Zeilen im `twoDimArray`)
- `size()` äquivalent zu `getCount()`
- `getAt(int i)` i. Zeile des `twoDimArray`, äquivalent zu `matcher[index]`
- `int groupCount()` Anzahl Capture Groups im Pattern dieses Matchers
- `boolean hasGroup()` true, wenn dieser Matcher Captur Groups besitzt
(äquivalent `groupCount()>0`)

Anwendungen des Find-Operators (2/2)

Weitere Methoden des `java.util.regex.Matcher`

- `String replaceFirst(String replacement)`
 - **erstes** Vorkommen des Patterns durch `replacement` ersetzt
 - Beispiel:

```
result = 'Ich bin von 12:00 bis 12:45 beim Mittagessen' =~ /([012]?\d):(\d{2})/
println result.replaceFirst("15:00")
```

Ausgabe: 'Ich bin von 15:00 bis 12:45 beim Mittagessen'
- `String replaceAll(String replacement)`
 - **alle** Vorkommen des Patterns durch `replacement` ersetzt
 - Beispiel:

```
result = 'Ich bin von 12:00 bis 12:45 beim Mittagessen' =~ /([012]?\d):(\d{2})/
println result.replaceAll("15:00")
```

Ausgabe: 'Ich bin von 15:00 bis 15:00 beim Mittagessen'

Pattern-Operator

Pattern-Operator: `matcher = ~/pattern/`

Leerzeichen zwischen `=` und `~`

- parst und compiliert `pattern`, um künftige Zugriffe zu beschleunigen
- liefert `java.util.regex.Pattern` für dieses Pattern
 - schnelle Pattern-Analyse mit `java.util.regex.Matcher matcher(String testString)`
 - `Matcher` entspricht dem Ergebnis des Find-Operators `testString ~= pattern`
- Beispiel:

```
def printMatchResults(parsedPattern, expression) {
    matches = parsedPattern.matcher(expression)
    for (match in matches) {
        for (captureGroup in match[1..<match.size()]) { print "$captureGroup " }
        print ","
    }
    println()
}

parsedPattern = ~/([012]?\d):(\d{2})/
printMatchResults parsedPattern, 'Ich bin von 12:00 bis 12:30 in der Mensa';
printMatchResults parsedPattern, 'Ich bin von 0:00 bis 7:30 im Bett';
```

Ausgabe: 12 00 , 12 30 ,

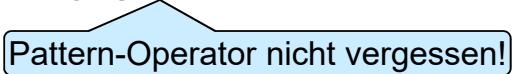
0 00 , 7 30 ,

Weitere Anwendungen des Pattern-Operators (1/2)

Zusätzliche Methoden der Collection-Schnittstelle:

- *Collection grep(Pattern parsedPattern)*
 - liefert Collection der auf geparstes Pattern passenden Elemente
 - Beispiel:

```
def profs=["Thomas Specht", "Peter Kaiser", "Peter Knauber", "Paul Schmücker"]
println profs.grep(~/Peter \w*/)    // Alle Profs mit Vorname Peter heißen
println profs.grep(~/T\w* \w*/)    // Alle Profs, deren Vorname mit T beginnt
```

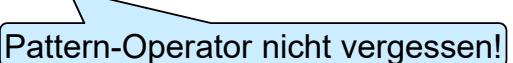


Weitere Anwendungen des Pattern-Operators (2/2)

Pattern Matching in der Switch-Case-Anweisung:

- ```
switch(ausdruck) {
 case ~pattern: // do something ...
}
```
- funktioniert, weil die Pattern-Klasse den **case**-Operator definiert:  
*pattern.isCase(ausdruck)*
  - Beispiel:

```
def profs=["Thomas Specht", "Peter Kaiser", "Peter Knauber",
 "Astrid Schmücker-Schend", "Paul Schmücker"]
profs.each { name->
 switch (name) {
 case ~/Thomas \w*/:
 case ~/Peter \w*/:
 }
}
```



erzeugt Ausgabe:

```
Vorname Thomas
Vorname Peter
Vorname Peter
```

## Weitere Anwendungen von regulären Ausdrücken (1/2)

Zusätzliche Methoden der *String*-Klasse:

- *replaceAll(pattern, replacement)*

ersetzt alle Teilstrings, die auf das Pattern passen, durch *replacement*

- Beispiel:

```
def string ="Das hier ist ein Teststring"
println string.replaceAll(/[aeiou]/, "a") // Das haat ast aan Taststrang
```

- *String[] split(String splitPattern)*

- splittet String an allen Stellen, die das Pattern *splitPattern* erfüllen

- liefert Array aller Teilstrings (ohne *splitPattern* matchende Splitstrings)

- Beispiel:

```
str = "12:00:13"
println "split($str) = ${ str.split(:)}"
```

Ausgabe: *split(12:00:13) = [12, 00, 13]*

## Weitere Anwendungen von regulären Ausdrücken (2/2)

Zusätzliche Methoden der *String*-Klasse:

- *eachMatch(pattern) { Closure }*

führt für jeden Teilstring, auf den das Pattern passt, die angegebene Closure aus

- matchender Teilstring wird in *it* bereitgestellt

- Beispiel:

```
def string ="Ähm Schöne Grüße ähm aus ähm Mannheim"
n=0;
string.eachMatch(/[\Ää]hm/) { n++; println it }
println ("Anzahl ähms: $n") // 3
```



## Gliederung

2. Groovy
  1. Einführung in Groovy
  2. Syntax und Kontrollkonstrukte
  3. Funktionen
  4. Closures
  5. Datentypen und Operationen
  6. Collections
  7. Klassen und Groovy Beans
  8. Reguläre Ausdrücke
- 9. Metaprogrammierung und Builder**
10. Testen

## Metaprogrammierung

- Fähigkeit von Programmen,
  - **Programmcode als Daten** zu behandeln
    - einlesen, analysieren, generieren, transformieren, ändern und ergänzen
  - sich selbst zu **analysieren**
    - Attribute und Methoden einer Klasse zur Laufzeit herausfinden
  - sich selbst zu **erweitern**
    - d.h. sein eigenes Verhalten zu ändern
    - ggf. sogar neuen Programmcode zu erzeugen

Drei Varianten:

1. Zugriff auf Interna der **Laufzeitumgebung** per API
  - erfolgt zur **Laufzeit**
    - 1a) **Reflection**: Werte abfragen und ändern, ohne die Struktur zu ändern
      - Beispiel: Java Reflection Interface
    - 1b) echte **Metaprogrammierung**
      - ermöglicht Änderungen an der Struktur von Klassen und Objekten
        - Attribute zur Laufzeit hinzufügen
        - Methoden zur Laufzeit hinzufügen
      - Beispiel: Groovy Metaprogrammierung
  - 2. Ausführung von **Befehlen**, die in **Strings** oder anderen Datentypen vorliegen
    - werden zur **Laufzeit** konstruiert
      - Beispiel: Evaluation von JSON-Ausdrücken in JavaScript
  - 3. **außerhalb** des eigentlichen **Programms**
    - erfolgt zur **Compilezeit**: Präprozessoren, Präcompiler, Codetransformation
    - Beispiel: IDL-Präcompiler in CORBA

## Reflection

- Programm kennt seine **eigene Struktur** zur Laufzeit
  - vorhandene **Attribute** und **Methoden** einer Klasse bzw. Instanz
  - Sichtbarkeit, Rückgabetyp, Parameter und –datentypen einer Methode
- **Dynamischer Attributzugriff**
  - Name des abzufragenden **Attributs** liegt als **String** vor
- **Dynamischer Methodenaufruf**
  - Name aufzurufender **Methode** liegt als **String** vor
- **Vorteile**:
  - höhere Flexibilität
  - Parametrisierung von Attribut- und Methodennamen
- **Nachteile**:
  - keine Typprüfung durch Compiler mehr möglich
  - geringere Performance
- Beispiel: Java Reflection Interface

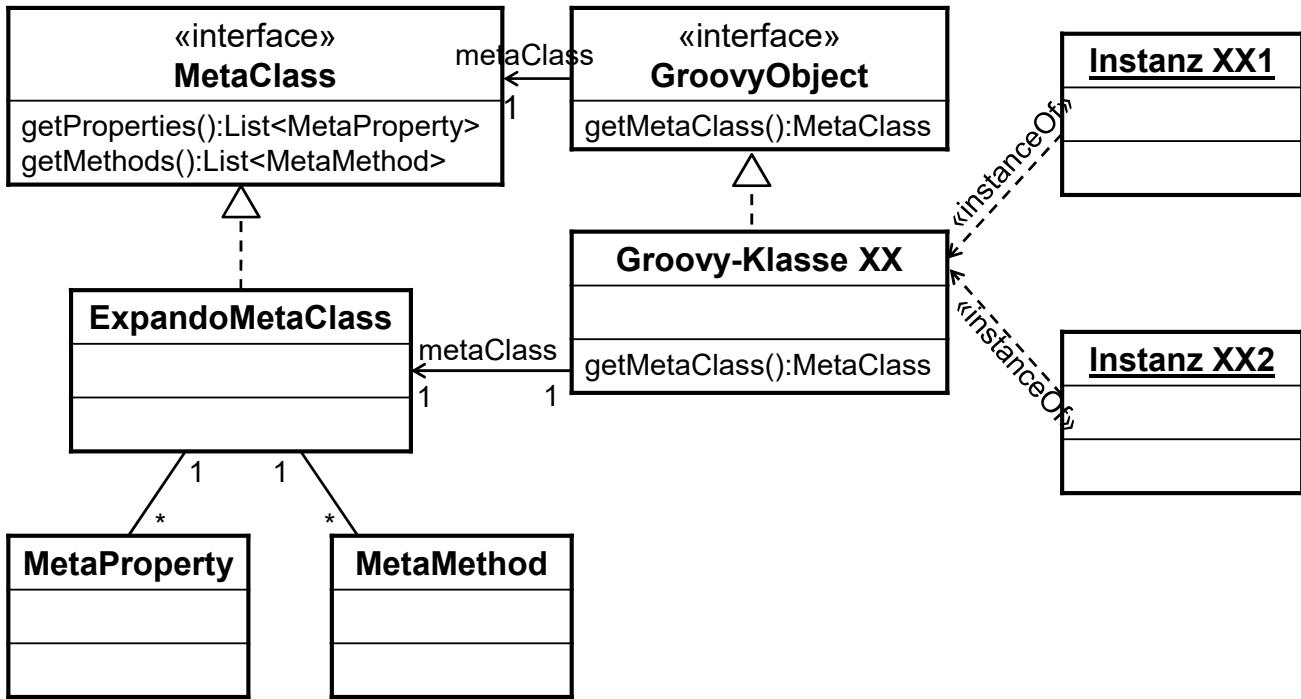
# Echte Metaprogrammierung

- Programm
  - kennt seine **eigene Struktur** zur Laufzeit
    - Reflection
    - Dynamischer Attributzugriff und Methodenaufruf
  - ermöglicht **strukturelle Änderungen** am Programmcode zur Laufzeit
    - Hinzufügen neuer **Attribute** und **Methoden**
    - ohne den Quellcode der Klasse zu besitzen oder zu ändern
    - funktioniert auch für alle mitgelieferten Klassen der Klassenbibliothek
- **Vorteile:**
  - Ergänzung vorhandener Klassen um
    - neue Methoden
    - Konverter und Rechenoperationen zu selbst definierten Datentypen
  - Selbstmodifizierender Code möglich
- **Nachteile:**
  - geringere Performance
- Beispiel: Groovy Metaprogrammierung

## Vergleich Java Reflection / Groovy Metaprogrammierung

| Kriterium | Java Reflection                            |    | Groovy Metaprogrammierung                          |
|-----------|--------------------------------------------|----|----------------------------------------------------|
| Zugriff   | <i>objRef.class</i><br><i>Klasse.class</i> |    | <i>objRef.metaClass</i><br><i>Klasse.metaClass</i> |
| Attribute | auflisten                                  | ja |                                                    |
|           | Auswahl per String                         | ja |                                                    |
|           | Detailinfos abrufen                        | ja |                                                    |
|           | <b>dynamisch hinzufügen</b>                | -  | <b>ja</b>                                          |
|           | Wert auslesen                              | ja |                                                    |
|           | Wert ändern                                | ja |                                                    |
| Methoden  | auflisten                                  | ja |                                                    |
|           | Auswahl per String                         | ja |                                                    |
|           | Details abrufen                            | ja |                                                    |
|           | <b>dynamisch hinzufügen</b>                | -  | <b>ja</b>                                          |
|           | Dynamisch aufrufen                         | ja |                                                    |
|           | <b>Aufrufe abfangen</b>                    | -  | <b>ja</b>                                          |
|           | <b>Interzeptoren</b>                       | -  | <b>ja</b>                                          |

# Groovy Metaprogrammiermodell



## MetaClass (Auszug)

- alle Instanzen der Klasse teilen sich ein **MetaClass**-Objekt

```

interface MetaClass extends MetaObjectProtocol {
 List<MetaProperty> getProperties() // Alle Attribute auflisten
 Object getProperty(Object instance, String property) // Attributwert auslesen
 Object setProperty(Object instance, String property, Object newValue) // ändern

 List<MetaMethod> getMethods() // Alle Methoden auflisten
 MetaMethod getMetaMethod(String name, Object[] args) // Metameth. suchen
 Object invokeMethod(Object instance, String methodName, Object[] args)
 // Methode aufrufen
}

```

- MetaClass**-Interface bildet Grundlage der Metaprogrammierung in Groovy
  - **ExandoMetaClass** implementiert **MetaClass**-Interface
  - verhält sich wie **Expando**

## MetaProperty (Auszug)

- ```
class MetaProperty {
```



```
    String getName()                      // Attributname
```



```
    Class getType()                       // Datentyp des Attributs
```



```
    Object getProperty(Object instance)   // Attributwert für Instanz auslesen
```



```
    Object setProperty(Object instance, Object newValue) // Attributwert schreiben
```


}
- Bemerkungen:
 - `getName()` und `getType()` funktionieren **instanzunabhängig**
 - `getProperty()` und `setProperty()` benötigen **konkrete Instanz**

MetaMethod (Auszug)

- ```
class MetaMethod {
```

```
 String getName() // Methodenname
```

```
 Class getReturnType() // Rückgabedatentyp
```

```
 Class[] getNativeParameterTypes() // Typen der formalen Parameter
```

```
 Object invoke(Object o, Object[] args) // Methode aufrufen auf Objekt o
```

  
}

## MetaClass: Klassenbeschreibung auflisten

```
def printClassDescription(Class c) {
 println "Attribute".center(40,"-")
 c.metaClass.getProperties().sort({it.getName()}).each {
 println "${it.getType()} ${it.getName()}"
 }

 println "Methoden".center(40,"-")
 c.metaClass.getMethods().sort({it.getName()}).each {
 Class returnType = it.getReturnType();
 print "${returnType.getSimpleName()}" // Rückgabetyp (ignoriert Generics)
 print "${it.getName()}(" // Name
 boolean firstParameter=true
 it.getNativeParameterTypes().each {
 print "${firstParameter ? "" : ","}${it.getSimpleName()}" // Ignoriert Generics
 firstParameter=false
 }
 println(")")
 }
}
```

## MetaClass: Dynamisch etwas zur Klasse hinzufügen

- **jeder** bestehenden **Klasse** können dynamisch (d.h. zur Laufzeit)
  - **Attribute** hinzugefügt werden
  - **Methoden** hinzugefügt werden
  - **Konstruktoren** hinzugefügt werden
- Voraussetzungen:
  - Klasse darf in **Groovy** oder **Java** geschrieben sein
  - Quellcode der Klasse dafür **nicht** erforderlich
  - auch für mitgelieferte Klassen aus Bibliotheken
  - auch für *final*-Klassen (z.B. *String*)
- Groovy erweitert auf die Weise viele Klassen der Java-**Klassenbibliothek**
  - z.B.: *Integer*, *BigInteger*, *BigDecimal*, *String*, Collections
  - um Methoden wie z.B:
    - *println* → deshalb darf man *System.out* weglassen!
    - *plus*, *minus* ... → deshalb kann man Rechenoperatoren überladen

## Dynamisch Attribut zur Klasse hinzufügen

- in Groovy-Skript
  - erster Befehl: `ExpandoMetaClass.enableGlobally()`
    - damit auch bereits bestehende Instanzen neue Attribute / Methoden sehen
    - sonst nur die Instanzen, die nach der Erweiterung noch erzeugt werden
  - Dynamisch **Attribut** `neuAttr` an `MeineKlasse` hinzufügen:  
`MeineKlasse.metaClass.neuAttr = initWert`



- Initialwert
  - muss vorhanden sein
  - legt Datentyp fest
  - wird allen bestehenden sowie neu erzeugten Instanzen zugewiesen
- vgl. Attribut zu Expando-Objekt nachträglich hinzufügen
- **static-Attribute** lassen sich **nicht** nachträglich zu Klassen hinzufügen!

## Dynamisch Methode zur Klasse hinzufügen

- Dynamisch **Methode** `neueMethode` an `MeineKlasse` hinzufügen:  
`MeineKlasse.metaClass.neueMethode = { ... }`



- alle Instanzen der Klasse bekommen diese zusätzliche Methode
- Zugriff auf eigenes Objekt mit **delegate** statt `this`
- Rückgabetyp ergibt sich aus `return`-Wert der Closure
- vgl. Methode zu Expando-Objekt nachträglich hinzufügen
- **Überladen** von Methoden möglich
  - selben Methodennamen mit unterschiedlichen Signaturen zuweisen
- **Überschreiben** von Methoden möglich
  - Methodename mit gleicher Signatur zuweisen

## Dynamisch static-Methode zur Klasse hinzufügen

- Dynamisch static-**Methode** `neueStaticMethode` an `MeineKlasse` hinzufügen:  
`MeineKlasse.metaClass.static.neueStaticMethode = { ... }`



- alle Instanzen der Klasse bekommen diese zusätzliche static-Methode
- Rückgabetyp ergibt sich aus `return`-Wert der Closure
- vgl. Methode zu Expando-Objekt nachträglich hinzufügen
- **Überladen** von static-Methoden möglich
  - selben Methodennamen mit unterschiedlichen Signaturen zuweisen
- **Überschreiben** von static-Methoden möglich
  - Methodename mit gleicher Signatur zuweisen

## Dynamisch Konstruktor zur Klasse hinzufügen

- Dynamisch **Konstruktor** an `MeineKlasse` hinzufügen:  
`MeineKlasse.metaClass.constructor = { ... }`



- Rückgabe
  - Typ muss `MeineKlasse` sein
  - Rückgabewert muss neu erzeugte Instanz sein
    - im Gegensatz zu normalen Konstruktoren!
    - neue **Instanz** muss in Konstruktor-Closure **erzeugt** werden
      - Rückgriff auf bestehenden Konstruktor
      - Gefahr der **Endlosrekursion**
- **Überladen** von Konstruktoren möglich
  - `constructor` mit unterschiedlichen Signaturen zuweisen
- **Überschreiben** von Konstruktoren möglich
  - `constructor` mit gleicher Signatur zuweisen

## Beispiel: String-Klasse erweitern

```
// String-Klasse um Konstruktoren erweitern, die den String gleich initialisieren
String.metaClass.constructor={Integer i -> new String(i.toString())}
String.metaClass.constructor={BigInteger i -> new String(i.toString())}
String.metaClass.constructor={Double d -> new String(d.toString())}
String.metaClass.constructor={BigDecimal d -> new String(d.toString())}
String.metaClass.constructor={Boolean b -> new String(b.toString())}

// String-Klasse um neue Methode erweitern
String.metaClass.changeCase={
 String result=""
 delegate.each {
 c = it as Character
 if (c.isLowerCase()) {
 result += it.toUpperCase()
 } else {
 result += it.toLowerCase()
 }
 }
 result
}
```

## MetaClass: Dynamisch zu konkreter Instanz hinzufügen

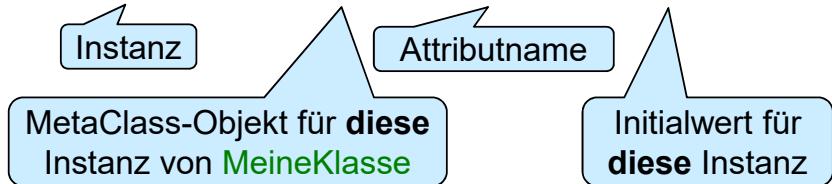
- Bisher: Attribut / Methode an **Klasse** hinzugefügt
  - ein **gemeinsames** MetaClass-Objekt für die Klasse
  - enthält zusätzliche Attribute / Methoden für **alle** Instanzen dieser Klasse
- Nun: Attribut oder Methode an konkrete Instanz hinzufügen
  - ein **exklusives** MetaClass-Objekt für diese **konkrete** Instanz
  - enthält zusätzliche Attribute / Methoden für diese **konkrete** Instanz
- Dynamisch hinzugefügtes Attribut auf **Instanzebene**
  - **verdeckt** gleichnamiges dynamisch hinzugefügte Attribut auf **Klassenebene**
- Dynamisch hinzugefügte Methode auf **Instanzebene**
  - **verdeckt** bei gleicher Signatur dynamisch hinzugefügte Methode auf **Klassenebene**

## Dynamisch Attribut zu konkreter Instanz hinzufügen

- in Groovy-Skript
  - Dynamisch **Attribut** *neuAttr* an *meineInstanz* hinzufügen:

*MeineKlasse meineInstanz*

*meineInstanz.metaClass.neuAttr = initWert*



- Initialwert
  - muss vorhanden sein
  - legt Datentyp fest
  - wird dieser **konkreten** Instanz zugewiesen
- Verdeckt gleichnamiges Attribut, das dynamisch der Klasse zugewiesen wurde

## Beispiel: String-Klasse / Instanz um Attribut erweitern

*ExpandoMetaClass.enableGlobally()*

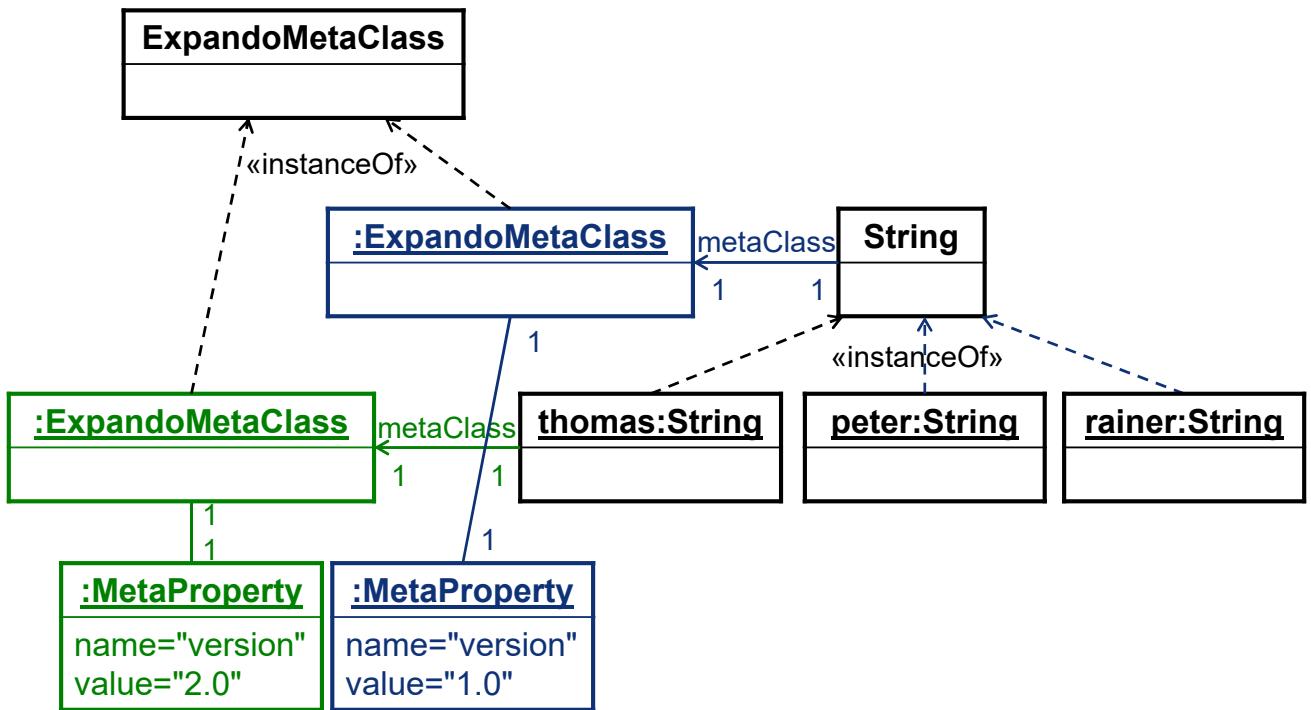
```
def thomas="Thomas"
def peter="Peter"
def rainer="Rainer"

println thomas // Thomas
println peter // Peter
println rainer // Rainer

thomas.metaClass.version="2.0" // Konkrete Instanz um Attribut erweitern
String.metaClass.version="1.0" // Klasse um Attribut erweitern

println()
println thomas.version // 2.0
println peter.version // 1.0
println rainer.version // 1.0
```

## Beispiel: String-Klasse / Instanz um Attribut erweitern



## Dynamisch Methode zu konkreter Instanz hinzufügen

- in Groovy-Skript
  - Dynamisch **Methode neueMethode** an **meineInstanz** hinzufügen:  
`MeineKlasse meineInstanz  
meineInstanz.metaClass.neueMethode = { ... }`
- nur **diese** Instanz bekommt zusätzliche Methode
- Zugriff auf eigenes Objekt mit **delegate** statt **this**
- Rückgabetyp ergibt sich aus *return*-Wert der Closure
- **Überladen** von Methoden möglich
  - selben Methodennamen mit unterschiedlichen Signaturen zuweisen
- **Überschreiben** von Methoden der (Meta-)klasse möglich
  - Methodenname mit gleicher Signatur zuweisen

## Beispiel: String-Klasse / Instanz um Methode erweitern

```
ExpandoMetaClass.enableGlobally()

def thomas="Thomas"
def peter="Peter"
def rainer="Rainer"

println thomas // Thomas
println peter // Peter
println rainer // Rainer

thomas.metaClass.convert = {
 -> delegate.toUpperCase()
}

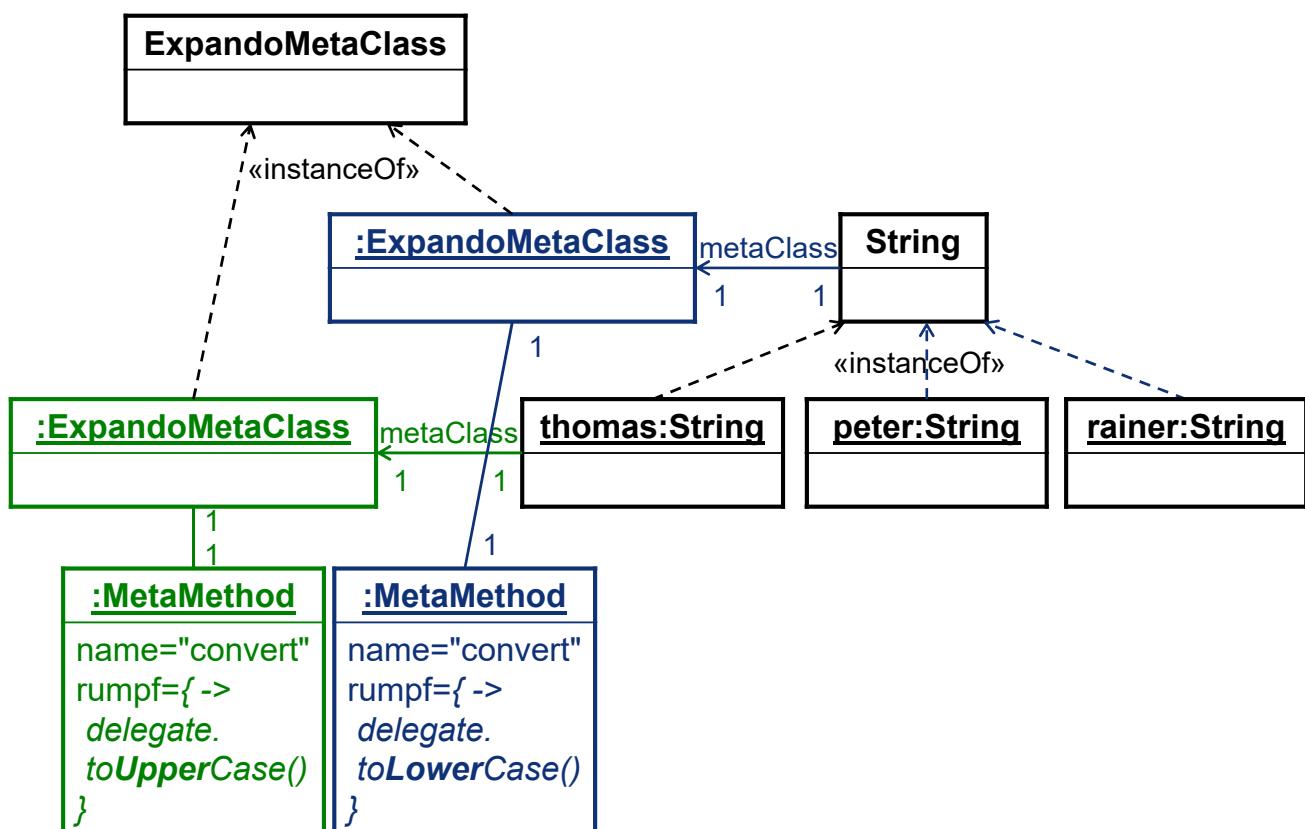
String.metaClass.convert = {
 -> delegate.toLowerCase()
}

println()
println thomas.convert() // THOMAS
println peter.convert() // peter
println rainer.convert() // rainer
```

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

167

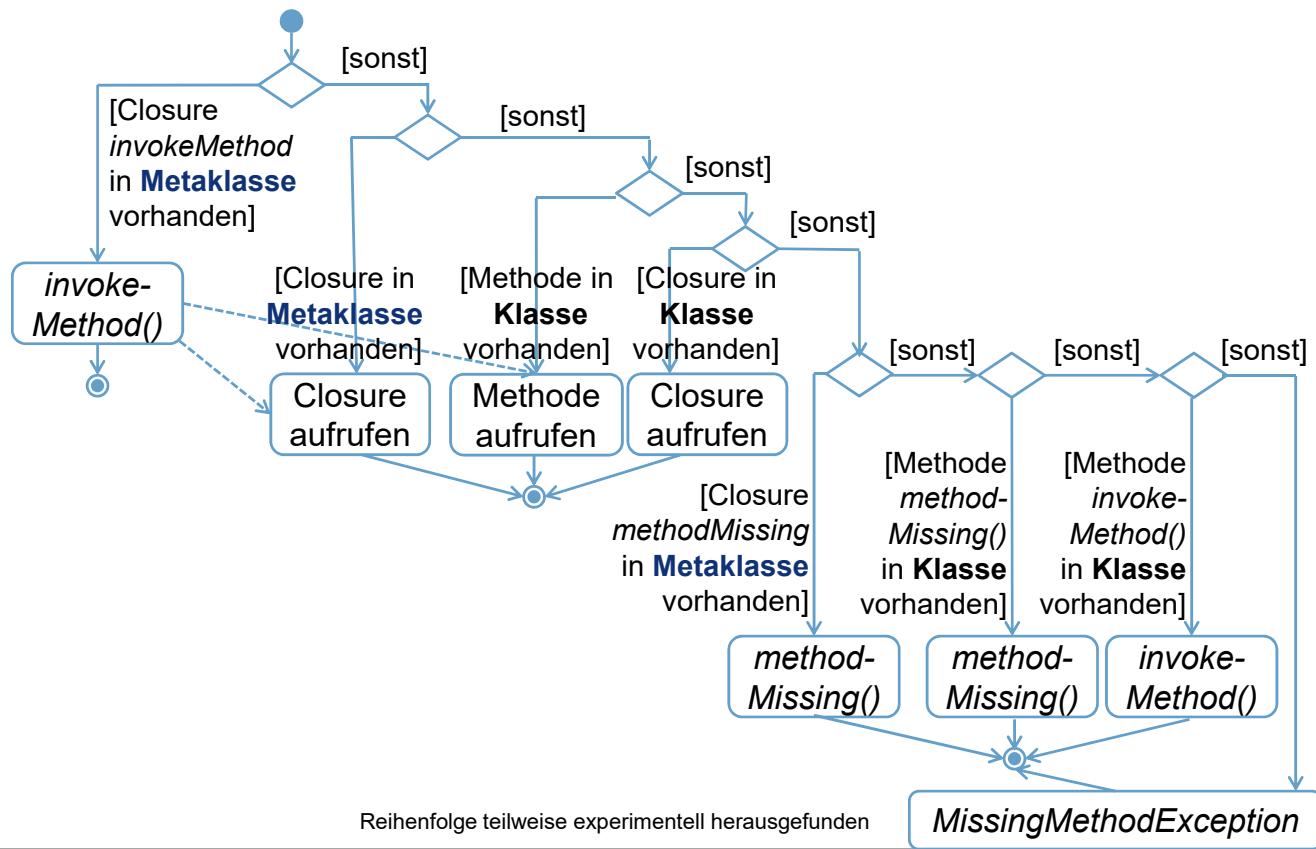
## Beispiel: String-Klasse / Instanz um Methode erweitern



Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

168

## Methodenausführung in Groovy (vereinfacht)



Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

169

## Fehlende Methoden abfangen: methodMissing

- Falls aufgerufene Methode **nicht** existiert
  - weder als Methode noch als Closure in der Klasse
  - auch nicht als Closure in der Metaklasse
- ruft Groovy **methodMissing(String methodename, args)** auf
  - als Methode in der Klasse
  - als Closure in der Metaklasse
- **Anwendung:** Rumpf für fehlende Methode nachliefern
  - oft erst dynamisch zur Laufzeit generiert
  - Anwendung z.B. in Grails für `findXXX`-Methoden der Domänenklassen
- **Vorteile**
  - es brauchen nicht Unmengen von Methoden auf Vorrat programmiert werden
  - automatische Generierung zur Laufzeit
- **Nachteile**
  - geringere Performance
  - Gefahr der Endlosrekursion

## methodMissing: Beispiel

```
ExpandoMetaClass.enableGlobally()

// Multipliziermethoden nachtragen
BigInteger.metaClass.methodMissing = { String methodName, args ->
 if (methodName ==~ /times\d+/) {
 def result = methodName =~ /times(\d+)/ // Faktor extrahieren
 if (result.size()==1) {
 BigInteger factor = new BigInteger (result[0][1]) // Match 0, 1. CG
 return delegate*factor
 }
 } else {
 throw new MissingMethodException(methodName, delegate.class, args)
 }
}

BigInteger b=14
println b.times2() // 28
```

## Intercept – Cache – Invoke-Pattern

- **Problem**
  - fehlende Methode über *methodMissing()* nachliefern kostet Performance
- **Lösungsansatz:** Cache für dynamisch erzeugte Methoden
  - bei **erstmaligem** Aufruf
    - fängt *methodMissing()* den Aufruf der fehlenden Methode ab (**Intercept**)
    - erzeugt dynamisch die neue Methode als Closure
    - fügt die Methode dynamisch der Metaklasse als Closure hinzu (**Cache**)
    - ruft die neue Methode auf, um das Ergebnis zu berechnen (**Invoke**)
  - bei **allen weiteren** Aufrufen
    - ist gewünschte Methode in der Metaklasse vorhanden und direkt aufrufbar
- **Anwendung:**
  - ausgiebige Verwendung in Grails für *findXXX*-Methoden in Domänenklassen
- **Vorteile**
  - gute Performance trotz automatischer Erzeugung dynamischer Methoden

## methodMissing: Beispiel Intercept – Cache – Invoke-Pattern

```
ExpandoMetaClass.enableGlobally()
BigInteger.metaClass.methodMissing = { String methodName, args -> // intercept
 println „Method missing: $methodName“
 if (methodName ==~ /times\d+) {
 def result = methodName =~ /times(\d+)/ // Faktor extrahieren
 if (result.size()==1) {
 BigInteger factor = new BigInteger (result[0][1]) // Match 0, 1. CG
 def closure={ return delegate*factor } // Methodenrumpf
 BigInteger.metaClass."$methodName" = closure // hinzufügen (cache)
 return closure(args) // aufrufen (invoke)
 }
 } else {
 throw new MissingMethodException(methodName, delegate.class, args)
 }
}

BigInteger b=14
BigInteger c=29

println b.times2() // 28
println c.times2() // 58, nun aus Cache bedient ohne methodMissing-Aufruf
```

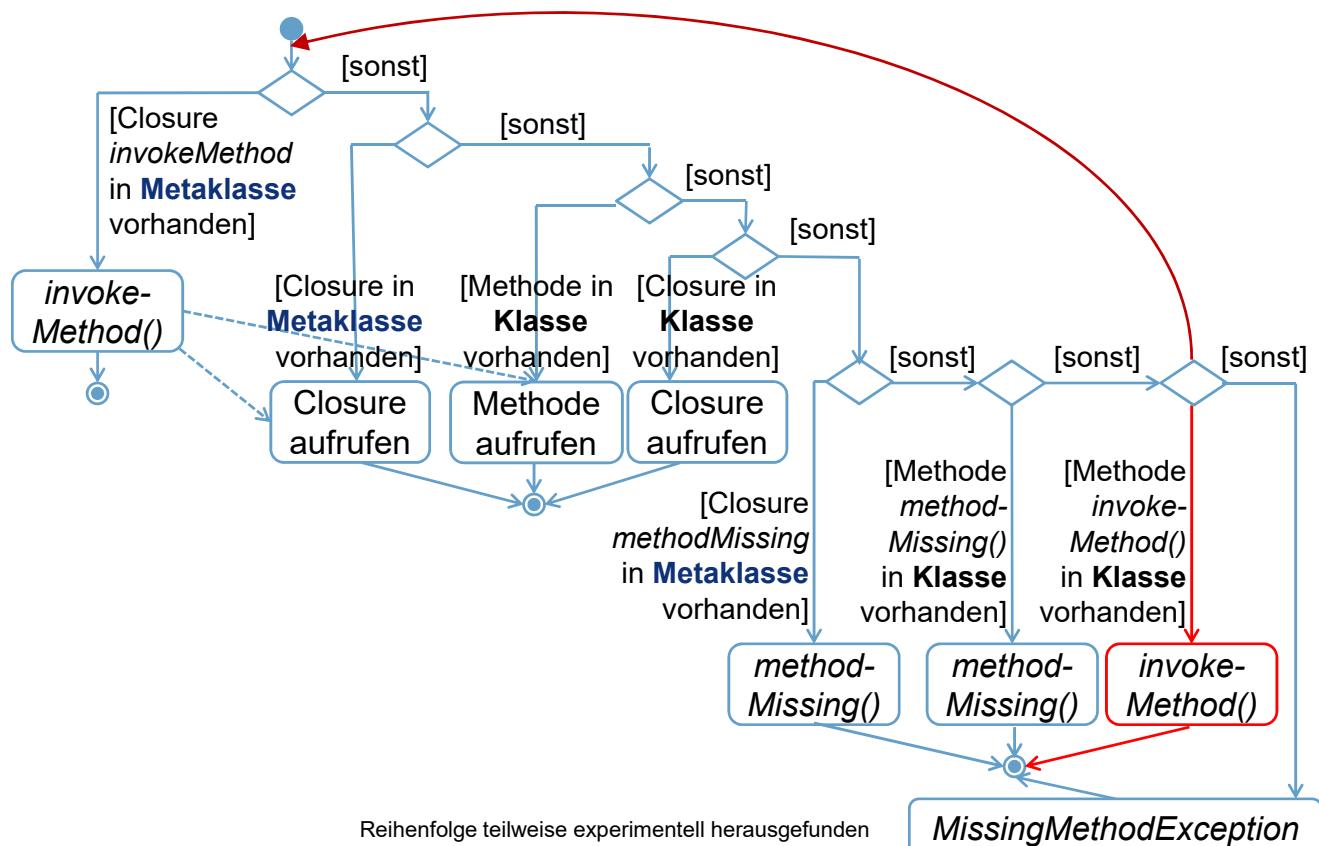
## Aspektorientierte Programmierung (AOP)

- **Ziel:** generische Funktionalitäten (Cross Cutting Concerns)
  - von der eigentlichen Anwendungslogik trennen
  - über Klassen / Methoden hinweg verwenden
- **Anwendungsbeispiele**
  - Protokollierung aller Methodenaufrufe
  - Persistenz und Transaktionssteuerung
  - Security (Benutzerrechteprüfung)
  - Datenvalidierung
  - Fehlerbehandlung
  - Profiling (Flaschenhälse erkennen)

# AOP mit Groovy

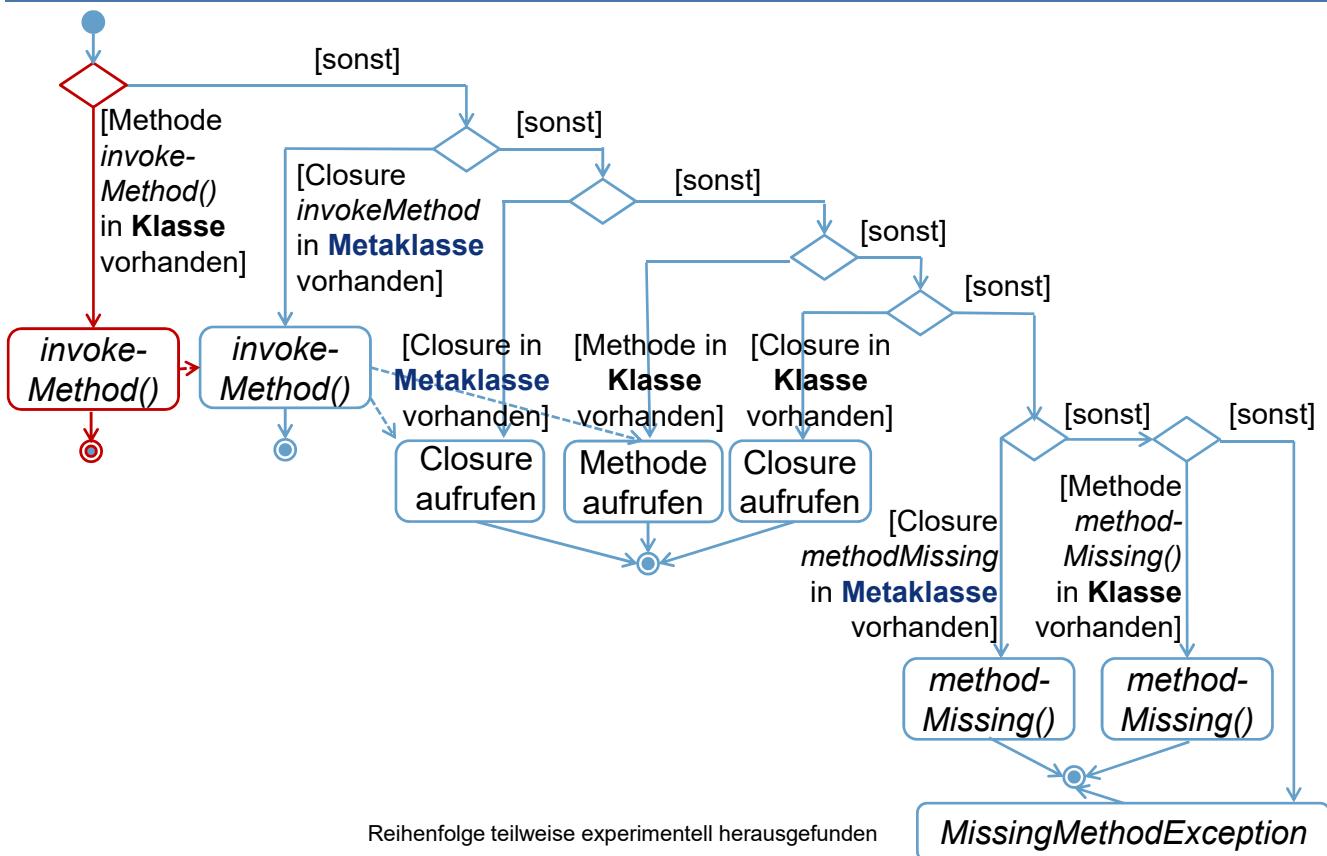
- jede Klasse kann Interface **GroovyInterceptable** implementieren
    - reines Marker-Interface ohne Methoden
    - **alle** Methodenaufrufe laufen dann über *invokeMethod()*-Aufruf der Klasse
    - dort kann
      - **vor** dem Methodenaufruf eigener Code ausgeführt werden
      - die ursprüngliche Methode aufgerufen werden, ggf. unter Bedingungen
      - **nach** dem Methodenaufruf eigener Code ausgeführt werden

## Methodenausführung in Groovy



Reihenfolge teilweise experimentell herausgefunden

## Methodenausführung in Groovy für GroovyInterceptable



Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

177

## invokeMethod(): Methode vs. Closure

- **Methode invokeMethod()**
  - von **Klasse** selbst implementiert
  - falls Klasse **GroovyInterceptable** nicht implementiert
    - nur Aufrufe **nicht vorhandener** Methoden abfangen
    - zur **Ergänzung** nicht vorhandener Methoden  
→ besser *methodMissing()* hierfür verwenden
  - falls Klasse **GroovyInterceptable** implementiert
    - **alle** Methodenaufrufe durch *invokeMethod()* abfangen
      - deshalb Vorziehen in Auswertungsreihenfolge
    - Nutzung für **AOP**
- **Closure invokeMethod() der Metaklasse**
  - in **Metaklasse** als **optionale** Closure
    - wird nur aufgerufen, **sofern vorhanden**
    - fängt bei Vorhandensein **alle** Methodenaufrufe ab
      - steht deshalb immer vorne in Abarbeitungshierarchie

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

178

## AOP mit Groovy: Beispiel

```
class MyClass implements GroovyInterceptable {
 // ...

 def invokeMethod (String name, args) {
 // * Beachte: System.out.println, um Endlosrekursion zu vermeiden
 // println ist Methode in allen Groovy-Klassen, also auch in MyClass!
 System.out.print "Entering method call: $name(" // *
 boolean first=true;
 args.each {
 System.out.print "${first? "" : ", "} ${it}"
 first=false
 }
 System.out.println ")"
 // Im Methodenausführungsprogramm eine Stufe nach rechts
 // Die Closure haben wir nicht definiert → echte Methode aufrufen
 def result = MyClass.metaClass.invokeMethod(this, name, args)
 System.out.println "Exiting method call: $name" // *
 return result
 }
}
```

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

179

## AOP für bestehende Klassen

- Closure `metaClass.invokeMethod` überschreiben
  - darf dann ihrerseits **nicht** `metaClass.invokeMethod` aufrufen
    - `metaClass.invokeMethod` würde in Endlosrekursion geraten
  - wir brauchen Methode mit noch höherer Priorität im Ausführungsdiagramm
    - indirekter Aufruf der abgefangenen Methode über **Metamethode**:  
`delegate.metaClass.getMetaMethod(name,args).invoke(delegate,args)`

```
String.metaClass.invokeMethod = {
 String name, args ->
 System.out.println "Methode $name aufgerufen"
 System.out.println "Argumente:"
 args.each { System.out.println it}
 System.out.println "-----"
 delegate.metaClass.getMetaMethod(name,args).invoke(delegate,args)
}
```

```
String name="Thomas"
nameSub = name.substring(2,4)
println "nameSub: $nameSub"
```

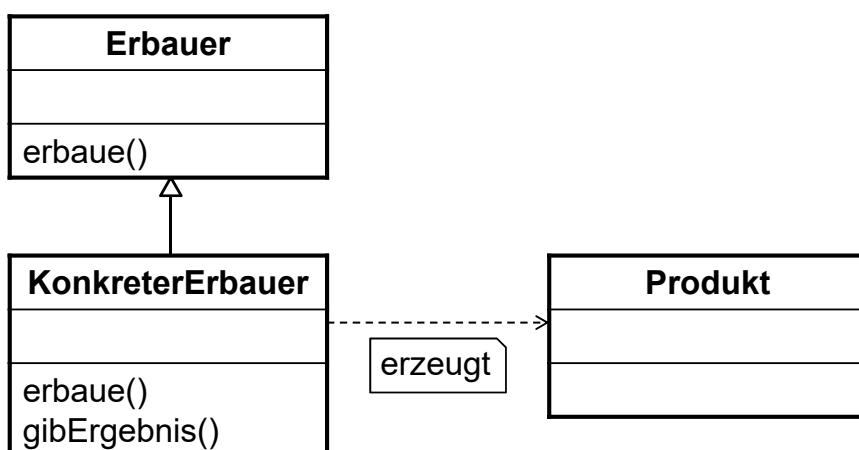
Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

180

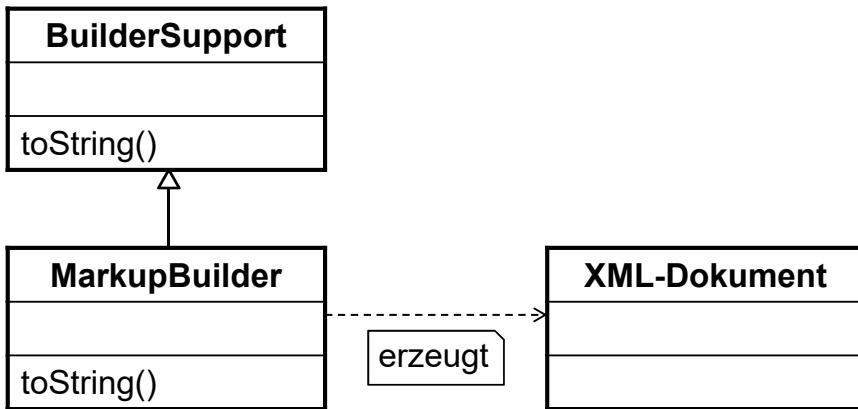
## Builder

- Design Pattern zum Erzeugen komplexer Objekte
  - bekommt Beschreibung als Parameter
  - liefert komplexes Objekt
- Beispiel:
  - bekommt Beschreibung eines XML-Dokuments
  - liefert fertiges XML-Dokument
- Anwendung
  - Verarbeitung domänenspezifischer Sprachen
- Anwendung in Groovy: komfortable Erzeugung von
  - XML-Dokumenten
  - Konfigurationsdaten für OR-Mappern

## Builder Design Pattern (vereinfacht)



## Beispiel: MarkupBuilder in Groovy



## Anwendung des MarkupBuilders in Groovy

```
import groovy.xml.MarkupBuilder
// StringWriter für die Ausgabe anlegen
def writer = new StringWriter()

// Builder für das XML-Dokument anfordern
def meinXML = new groovy.xml.MarkupBuilder(writer)
meinXML.professoren { Name des Root-Elements
 Elementname Attributname Attributwert
 professor(vorname: "Thomas", nachname:"Specht", kuerzel:"SPE") {
 vorlesung (name:"Grundlagen der Informatik", kuerzel: "GDI") < Elementtrumpf
 vorlesung (name:"Objektorientierte Techniken", kuerzel: "OOT")
 vorlesung (name:"Webarchitekturen und -Frameworks", kuerzel: "WAF")
 }
 professor(vorname: "Rainer", nachname:"Gerten", kuerzel:"GRT")
}

println writer.toString()
```

The code demonstrates the use of the `MarkupBuilder` class in Groovy to generate XML. It starts by importing `groovy.xml.MarkupBuilder` and creating a `StringWriter` for output. A `MarkupBuilder` object, `meinXML`, is created with the `writer`. The code then defines a root element `professoren`. Inside this element, two `professor` elements are added, each with attributes `vorname`, `nachname`, and `kuerzel`. Each `professor` element contains a nested `vorlesung` element with attributes `name` and `kuerzel`. The `vorlesung` element is annotated with `< Elementtrumpf`, which is highlighted in green. The code concludes by printing the resulting XML string to the console.

## Ausgabe des Markup Builders

```
<professoren>
 <professor vorname='Thomas' nachname='Specht' kuerzel='SPE'>
 <vorlesung name='Grundlagen der Informatik' kuerzel='GDI' />
 <vorlesung name='Objektorientierte Techniken' kuerzel='OOT' />
 <vorlesung name='Webarchitekturen und -Frameworks' kuerzel='WAF' />
 </professor>
 <professor vorname='Rainer' nachname='Gerten' kuerzel='GRT' />
</professoren>
```

## Selbst definierter Builder (1/4)

```
class MeinXMLBuilder implements GroovyInterceptable{
 int einruecken;
 String result;

 def MeinXMLBuilder() {
 einruecken=0;
 result="";
 }

 def invokeMethod(String name, args) { // erbaue
 // Methodenname = XML-Tag
 if (args.length==0) {
 result+="$'{ '*einruecken}<$name/>\n" // Tag ohne Attribute/Rumpf
 }
 }
}
```

## Selbst definierter Builder (2/4)

```
} else if (args.length==1) {
 switch (args[0]) {
 case Closure: // Rumpf mit weiteren Tags
 result+="${" *einruecken++}<$name>\n"
 def myClosure = args[0]
 myClosure.delegate=this // Im Kontext dieser MeinXMLBuilder-Instanz
 myClosure.call() // Closure aufrufen
 result += "{$ '--einruecken}</$name>\n"
 break;
 case String: // Rumpf als String ohne Tags
 result+="${" *einruecken}<$name>${args[0]}</$name>\n"
 break;
 case Map: // Attribute-Werte-Paare
 result+="${" *einruecken}<$name"
 def params = args[0]
 params.each { key, value -> result += " $key='\$value'" }
 result+= "/>\n"
 break;
 default:
 result += "{$ *einruecken}<$name>Unbekannter Rumpf</$name>\n"
 }
}
```

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

187

## Selbst definierter Builder (3/4)

```
} else if (args.length==2 && args[0] instanceof Map) {
 switch (args[1]) {
 case Closure: // Attribute und Rumpf vorhanden
 result+="${" *einruecken++}<$name"
 def params = args[0]
 params.each { key, value -> result += " $key='\$value'" }
 result+= ">\n"
 def myClosure = args[1]
 myClosure.delegate=this // Im Kontext des Builders abarbeiten
 myClosure.call() // Closure aufrufen
 result+="${" '--einruecken}</$name>\n"
 break;
 case String: // Attribute und String als Rumpf vorhanden
 result+="${" *einruecken}<$name"
 def params = args[0]
 params.each { key, value -> result+= " $key='\$value'" }
 result += ">${args[1]}</$name>\n"
 break;
 default:
 result += "{$ *einruecken}<$name>Unbekannter Rumpf</$name>\n"
 }
}
```

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

188

## Selbst definierter Builder (4/4)

```
 } else {
 result += "Invalid Syntax near '$name\n"
 }
}

String toString() { // Gib Ergebnis
 result
}
}
```

## Aufruf des selbst definierten Builders

```
// Builder für das XML-Dokument anfordern
def meinXML = new MeinXMLBuilder()

def xmldoc = meinXML.professoren {
 professor(vorname: "Thomas", nachname:"Specht", kuerzel:"SPE") {
 vorlesung(name:"Grundlagen der Informatik", kuerzel: "GDI")
 vorlesung(name:"Objektorientierte Techniken", kuerzel: "OOT")
 vorlesung(name:"Webarchitekturen und -Frameworks", kuerzel: "WAF")
 }
 professor(vorname: "Rainer", nachname:"Gerten", kuerzel:"GRT")
 abschluss "Hallo"
 ende()
}

println xmldoc.toString()
```

## Ausgabe des selbst definierten Builders

```
<professoren>
 <professor vorname='Thomas' nachname='Specht' kuerzel='SPE'>
 <vorlesung name='Grundlagen der Informatik' kuerzel='GDI' />
 <vorlesung name='Objektorientierte Techniken' kuerzel='OOT' />
 <vorlesung name='Webarchitekturen und -Frameworks' kuerzel='WAF' />
 </professor>
 <professor vorname='Rainer' nachname='Gerten' kuerzel='GRT' />
<abschluss>Hallo</abschluss>
<ende/>
</professoren>
```



## Gliederung

2. Groovy
  1. Einführung in Groovy
  2. Syntax und Kontrollkonstrukte
  3. Funktionen
  4. Closures
  5. Datentypen und Operationen
  6. Collections
  7. Klassen und Groovy Beans
  8. Reguläre Ausdrücke
  9. Metaprogrammierung und Builder

## 10. Testen

## Einfache skriptbasierte Tests

- **assert boolescheBedingung [: "Fehlerbeschreibung"]**
  - stellt sicher, dass *boolescheBedingung* true ist
  - sonst Programmabbruch
  - Angabe der durch : abgetrennten Fehlerbeschreibung optional
- Beispiele:
  - **assert x>=0**
  - **assert x>=0 : "x darf nicht negativ sein!"**

## Einfache skriptbasierte Tests: Beispiel

```
def abs(BigDecimal x) {
 x>0 ? x : -x
}

println "Quadratwurzelberechnung"
Scanner scanner = new Scanner(System.in)

print "Bitte positive Zahl eingeben: "
BigDecimal x = scanner.nextBigDecimal()
assert x>=0

BigDecimal guess = x/2.0
BigDecimal epsilon = 1E-10

while (abs(x-guess*guess)>epsilon) {
 guess = (guess + x/guess)/2.0
}

println guess
```

## Einfache skriptbasierte Tests: Beispiel

```
def abs(BigDecimal x) {
 x>0 ? x : -x
}

println "Quadratwurzelberechnung"
Scanner scanner = new Scanner(System.in)

print "Bitte positive Zahl eingeben: "
BigDecimal x = scanner.nextBigDecimal()
assert x>=0 : "Quadratwurzel negativer Zahlen nicht definiert!"

BigDecimal guess = x/2.0
BigDecimal epsilon = 1E-10

while (abs(x-guess*guess)>epsilon) {
 guess = (guess + x/guess)/2.0
}

println guess
```

## JUnit-Tests

- JUnit bereits von Java bekannt, nun auch für Groovy
- Voraussetzung: Testklasse benötigt Imports
  - **import org.junit.\***
  - **import static org.junit.Assert**

## Aufbau einer JUnit-Testklasse

```
import org.junit.*
import static org.junit.Assert.*

class BeispielJUnitTest {
 @BeforeClass public static void setUpClass() {
 }

 @AfterClass public static void tearDownClass() {
 }

 @Before public void setUp() {
 }

 @After public void tearDown() {
 }

 // Testmethoden
 @Test public void hello() {}
}
```

## Initialisierung, Ausführung und Aufräumen der Testfälle

	Annotation	Ausführung	Anwendungsbeispiel
Prepare	@BeforeClass static void method()	1* beim Start	DB anlegen und öffnen
	@Before void method()	vor jeder @Test Methode	Testvariablen initialisieren
Cleanup	@After void method()	nach jeder @Test Methode	Speicher freigeben
	@AfterClass static void method()	1* am Ende	DB schließen
Testmethoden	@Test void method()	1*	Testmethode
	@Test(expected= Exception.class) void method()	1*, muss Exception vom Typ Exception.class werfen	Test auf Auslösung einer Exception
	@Test(timeout=100) void method()	1*, darf max. 100 ms brauchen	Laufzeittest
	@Ignore void method()	0*, deaktivierte Testmethode	Unfertige Testmethode

## Anmerkungen

- alle JUnit-Methoden
  - parameterlos
  - Rückgabetypr *void*
  - Methodename beliebig
  - entscheidend ist die Annotation
  - Annotation darf an mehreren Methoden stehen

## Zusicherungen in @Test-Methoden (1/2)

	Zusicherung	sicherstellen, dass
	<i>fail(String message)</i>	Testmethode scheitern lassen
Boolean	<i>assertTrue([String message], boolean condition)</i>	<i>condition==true</i>
	<i>assertFalse([String message], boolean condition)</i>	<i>condition==false</i>
Gleichheit	<i>assertEquals([String message], expected, actual)</i>	<i>actual==expected</i> Prüft bei Arrays + selbstdef. Klassen Referenz, <b>nicht</b> Inhalt!
	<i>assertEquals([String message], expected, actual, tolerance)</i>	<i>actual==expected</i> für die ersten <i>tolerance</i> Dezimalstellen bei Fließkommazahlen
Null	<i>assertNull([String message], object)</i>	<i>object==null</i>
	<i>assertNotNull([String message], object)</i>	<i>object!=null</i>
Referenzen	<i>assertSame([String message], expected, actual)</i>	<i>expected und actual</i> selbes Objekt referenzieren
	<i>assertNotSame([String Message], expected, actual)</i>	<i>expected und actual</i> verschiedene Objekte referenzieren

## Zusicherungen in @Test-Methoden (2/2)

	Zusicherung	sicherstellen, dass
Array	<b>assertArrayEquals</b> (Object[] expected, Object[] actual)	beide Arrays inhaltsgleich sind
Arraygröße	<b>assertLength</b> (int length, char[] array)	array.length==length
	<b>assertLength</b> (int length, int[] array)	
	<b>assertLength</b> (int length, Object[] array)	
Arrayelem.	<b>assertContains</b> (char expected, char[] array)	array das Element expected enthält
	<b>assertContains</b> (int expected, int[] array)	
toStr.	<b>assertToString</b> (Object value, String expected)	value.toString()==expected
Closure	<b>shouldFail</b> (Closure closure)	closure eine Exception wirft
	<b>shouldFail</b> (Exception class, Closure closure)	closure Exception von Typ class wirft

## Zusicherungen: Anmerkungen

- JUnit prüft, ob Zusicherung zutrifft
  - wenn ja: Testmethode weiter abarbeiten
  - wenn nein: Testmethode abbrechen
    - falls message vorhanden: message statt Standardfehlermeldung ausgeben
- **assertEquals** ist fehlerhaft
  - vergleicht Arrayreferenz statt Arrayinhalt
  - vergleicht bei selbstdefinierten Klassen Referenz statt Inhalt
    - auch wenn equals-Methode überladen
    - Ausweg: **assertTrue(actual.equals(expected))**

## JUnit-Tests: Beispiel Bruch.groovy (1/8)

```
class Bruch implements Comparable {
 BigInteger z // Zähler
 BigInteger n // Nenner

 // Konstruktoren
 Bruch(Map m) {
 this.z=m.z
 this.n=(m.n ? m.n : 1)
 if (this.n==0)
 throw new IllegalBruchException("Nenner darf nicht 0 sein!")
 }

 // ...
```

## JUnit-Tests: Beispiel Bruch.groovy (2/8)

```
Bruch(BigInteger z) {
 this.z=z
 this.n=1
}

Bruch(BigInteger z, BigInteger n) {
 this.z=z
 this.n=n
 if (this.n==0)
 throw new IllegalBruchException("Nenner darf nicht 0 sein!")
}

Bruch (Bruch b) {
 this.z=b.z
 this.n=b.n
 if (this.n==0)
 throw new IllegalBruchException("Nenner darf nicht 0 sein!")
}

// ...
```

## JUnit-Tests: Beispiel Bruch.groovy (3/8)

```
// Hilfsfunktionen ggt und kgv
static BigInteger ggt(BigInteger a, BigInteger b) {
 b==0 ? a : ggt(b, a%b)
}

static BigInteger kgv(BigInteger a, BigInteger b) {
 a!=0 && b!=0 ? a*b.intdiv(ggt(a,b)) : 0
}

Bruch kuerzen() {
 def teiler = ggt(z.abs(),n.abs())*n.signum() // Nenner positiv machen
 new Bruch(z.intdiv(teiler), n.intdiv(teiler))
}

// ...
```

## JUnit-Tests: Beispiel Bruch.groovy (4/8)

```
// Grundrechenarten
Bruch plus(BigInteger b) {
 new Bruch(z+b*n, n).kuerzen()
}

Bruch plus(Bruch b) {
 new Bruch(z*b.n+b.z*n, n*b.n).kuerzen()
}

Bruch minus(BigInteger b) {
 new Bruch(z-b*n, n).kuerzen()
}

Bruch minus(Bruch b) {
 new Bruch(z*b.n-b.z*n, n*b.n).kuerzen()
}

// ...
```

## JUnit-Tests: Beispiel Bruch.groovy (5/8)

```
Bruch multiply(BigInteger b) {
 new Bruch(z*b,n).kuerzen()
}

Bruch multiply(Bruch b) {
 new Bruch(z*b.z,n*b.n).kuerzen()
}

Bruch div(BigInteger b) {
 new Bruch (z,n*b).kuerzen()
}

Bruch div(Bruch b) {
 new Bruch (z*b.n,n*b.z).kuerzen()
}

// ...
```

## JUnit-Tests: Beispiel Bruch.groovy (6/8)

```
// Potenzieren
Bruch power(BigDecimal p) {
 new Bruch(z**p, n**p).kuerzen()
}

// Vorzeichen
Bruch negative() {
 new Bruch(-z,n)
}

Bruch positive() {
 new Bruch(z,n)
}

// ...
```

## JUnit-Tests: Beispiel Bruch.groovy (7/8)

```
// Vergleiche
boolean equals(Bruch b) {
 def thisGekuerzt = this.kuerzen()
 def bGekuerzt=b.kuerzen()
 return thisGekuerzt.z==bGekuerzt.z && thisGekuerzt.n==bGekuerzt.n
}

int compareTo(Object b) {
 def diff = this-b
 return diff.z.signum()
}

// ...
```

## JUnit-Tests: Beispiel Bruch.groovy (8/8)

```
// Konvertierungen
Object asType(Class c) {
 if (c==BigDecimal.class) {
 return z/n;
 }
}

String toString() {
 String zString = z.toString()
 String nString = n.toString()
 def bruchstrichLaenge = Math.max(zString.size(), nString.size())
 def bruchstrich = "-"*bruchstrichLaenge

 "${zString.center(bruchstrichLaenge)}\n$bruchstrich\n${nString.center(bruchstrichLaenge)}"
}
```

## JUnit-Tests: Beispiel IllegalBruchException.groovy

```
class IllegalBruchException extends Exception {
 IllegalBruchException(String message) {
 super(message)
 }
}
```

## JUnit-Tests: Beispiel BruchTests.groovy (1/3)

```
import org.junit.*
import static org.junit.Assert.*

class BruchTests {
 Bruch a, b, c, d

 @Before public void setUp() {
 a = new Bruch(3)
 b = new Bruch (4,5)
 c = new Bruch (z:9,n:14)
 d = new Bruch(a)
 }

 // ...
```

## JUnit-Tests: Beispiel BruchTests.groovy (2/3)

```
@Test void testeKonstruktoren() {
 assertEquals 3G, a.z // G steht für BigInteger
 assertEquals 1G, a.n

 assertEquals 4G, b.z
 assertEquals 5G, b.n

 assertEquals 9G, c.z
 assertEquals 14G, c.n

 assertEquals 3G, d.z
 assertEquals 1G, d.n
}

@Test(expected=IllegalBruchException.class) void testeNenner0() {
 d = new Bruch(3G,0G) // Muss IllegalBruchException auslösen
}
```

## JUnit-Tests: Beispiel BruchTests.groovy (3/3)

```
@Test void testeVergleiche() {
 // Vorsicht: assertEquals(a,a) nutzt überladene equals-Methode NICHT!
 // --> direkt die überladene Methode aufrufen
 assertTrue a==a
 assertFalse a==b

 assertTrue a>b
 assertTrue a>=b
 assertFalse a<b
}

@Test void testeRechenoperationen() {
 assertTrue "Addition kommutativ", (a+b==b+a)
 assertTrue "Multiplikation kommutativ", (a*b==b*a)
 assertTrue "a-a gibt 0", ((a-a).z==0)
 assertTrue "Division", (a/b*b==a)
}
}
```



## Gliederung

1. Webarchitekturen
2. Groovy
- 3. Grails-Framework**
4. Java-Webtechnologien
5. Java Server Faces



## Gliederung

3. Grails-Framework
  - 1. Einführung in Grails**
  2. Controller
  3. Domänenklassen
  4. RESTful Web Services
  5. GSP

## Grails: Prinzipien (1/2)

- **Convention over Configuration**
  - fest vorgegebene Ordner für
    - Domänenklassen (Model)
    - Controller
    - Views
    - Internationalisierung (Übersetzungsdateien)
    - Skriptcode
    - Testcode
  - minimaler Konfigurationsaufwand durch
    - sinnvolle Standardwerte für alle Konfigurationsparameter
    - Konfiguration über Groovy-basierte Domänenpezifische Sprachen
    - weitgehenden Verzicht auf XML-Konfigurationsdateien
- **Don't repeat yourself**
  - Vermeidung unnötiger Code-Redundanzen
  - Automatische Generierung aus Domänenklassen
    - Datenbank
    - Controller und Views für CRUD-Operationen per **Scaffolding**

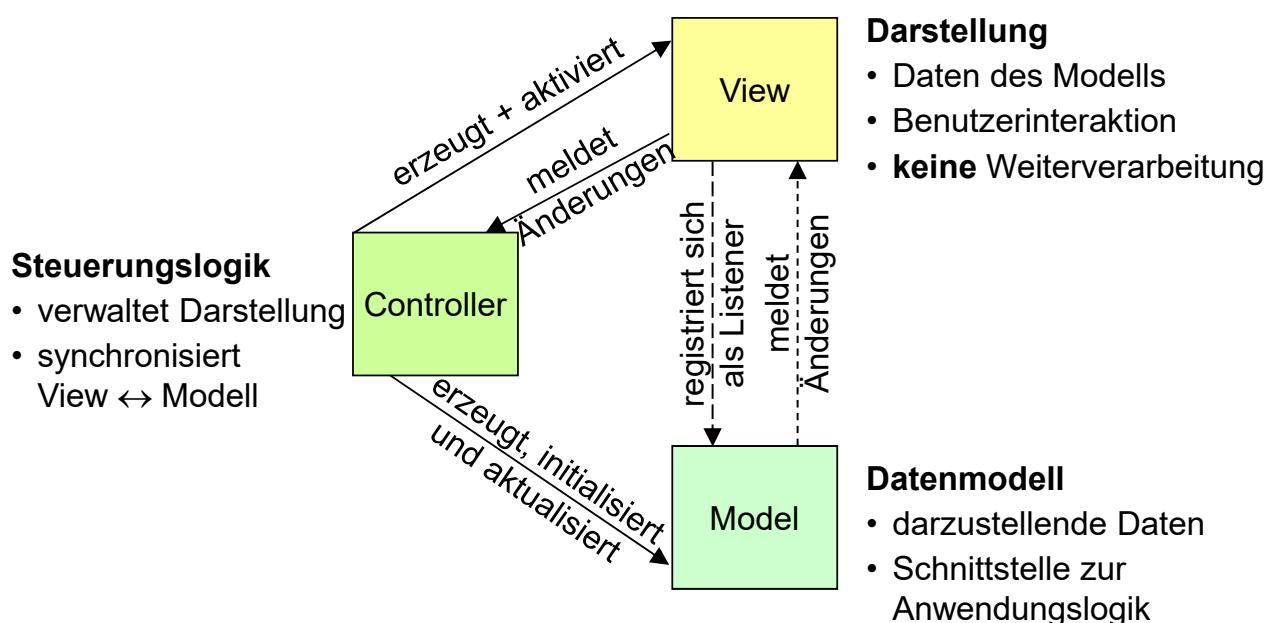
## Grails: Prinzipien (2/2)

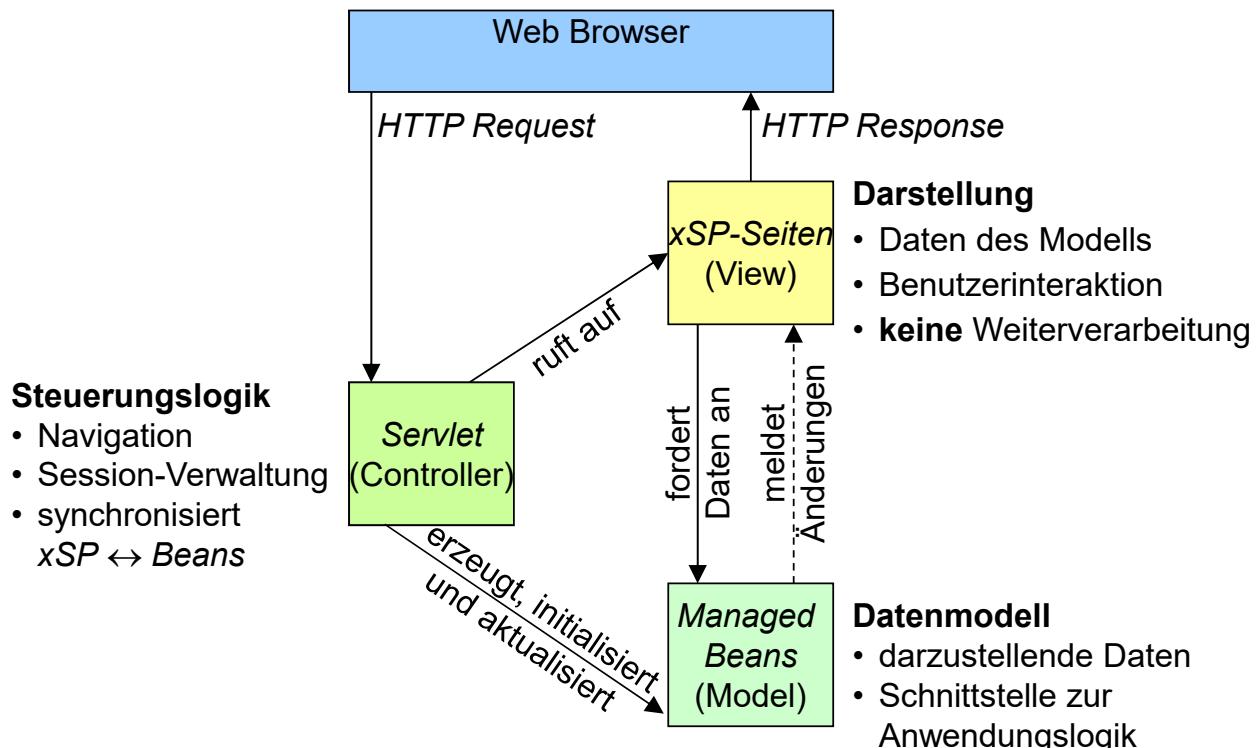
- **Agilität**
  - wahlweise statische oder dynamische Typisierung
  - kurze Entwicklungszyklen
  - agile Webentwicklung
  - Rapid Prototyping
  - leichtes Testen
- **Open Source**
  - Download von <http://www.grails.org>
  - Nutzung bestehender Open Source-Produkte
    - Java Virtual Machine
    - Servlet Container (Tomcat, Jetty oder JEE Application Server)
    - Groovy
    - JUnit (Automatisiertes Testen)
    - Spring (Komponentenmodell)
    - Hibernate (Objektrelationaler Mapper) als Grundlage für GORM
    - Sitemesh (Template-basierte Layout-Engine für GSP)

- **Model View Controller (MVC)**
  - Model: Domänenklassen mit automatischer Persistenz (GORM / Hibernate)
  - View: Groovy Server Pages (GSP), typischerweise HTML-basiert
  - Controller: Eigene Instanz für jeden Request
- **Scaffolding**
  - CRUD-Funktionalität automatisch generiert aus Domänenklassen
  - **statisches** Scaffolding
    - Groovy-Quellcode für Controller aus Domänenklassen generieren
    - Groovy Server Pages für Views aus Domänenklassen generieren
  - **dynamisches** Scaffolding
    - Controller und Views dynamisch zur Laufzeit erzeugen, ohne Quellcode

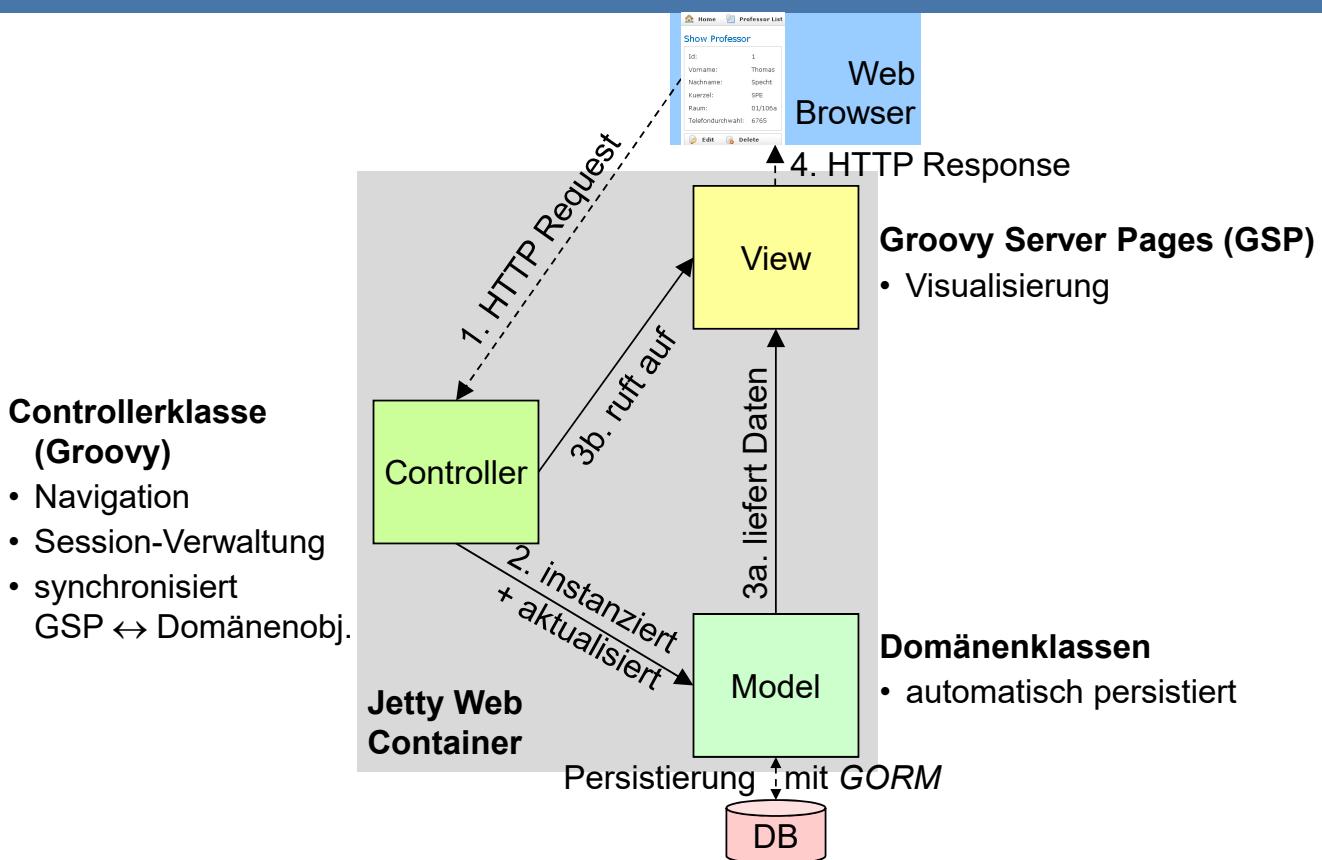
## Model View Controller (MVC)

- Design Pattern für Grafische Benutzeroberflächen
- Grundlage der Swing-Oberflächen in Java





## MVC in Grails



## Grails-Anwendung erstellen

1. Grails-Projekt anlegen
  - erstellt komplette Ordnerstruktur
2. Domänenklasse erstellen
3. Controller erstellen
  - Statisches Scaffolding: aus Domänenklasse als Groovy-Quellcode generieren
  - Dynamisches Scaffolding: zur Laufzeit aus Domänenklasse generieren
  - manuell
4. Views erstellen
  - Statisches Scaffolding: aus Domänenklasse als GSP generieren
  - Dynamisches Scaffolding: zur Laufzeit aus Domänenklasse generieren
  - manuell
5. Anwendung an Application Server **deployen**
6. Anwendung über Browser aufrufen

## Domänenklasse

- Grundlage der **Persistierung** in Grails
  - pro Domänenklasse eine **Datenbanktabelle**
  - **Instanzen** der Domänenklasse entsprechen **Zeilen** der DB-Tabelle
  - **Attribute** der Domänenklasse entsprechen **Spalten** der DB-Tabelle
  - Instanzen werden durch Grails automatisch persistiert
    - technische Basis: GORM (Grails Object Relational Mapping)
    - basiert auf Hibernate
- Realisierung als **Groovy Bean**
  - ganz normale Groovy-Klasse
    - kein Verbrauch der Vererbung
    - kein Konstruktor → impliziter Standardkonstruktor mit Map für Init-Werte
  - implizite Getter und Setter
  - serialisierbar
- Erstellung : *New* → *Grails Domain Class*
  - im Ordner **Domain Classes** (Convention over configuration)

## Domänenklasse Professor

```
class Professor {
 String vorname
 String nachname
 String kuerzel
 String raum
 String telefondurchwahl
 String bemerkungen

 static constraints = {
 vorname(blank:false) <-- Attribut darf nicht leer sein
 nachname(blank:false)
 kuerzel(blank:true, nullable:true) <-- Attribut darf leer sein
 raum(blank:true, nullable:true)
 telefondurchwahl(blank:true, nullable:true)
 bemerkungen(blank:true, nullable:true, maxSize:2000, widget:'textarea')
 }
 String toString() {
 return "$nachname, $vorname"
 }
}
```

Annotations:

- String vorname → Attribute
- String nachname → Constraints + Attributreihenfolge in GSP
- String kuerzel → Attribut darf leer sein
- String raum → Attribut darf leer sein
- String telefondurchwahl → Attribut darf leer sein
- String bemerkungen → Max. 2000 Zeichen, Mehrzeiliges Textfeld
- String toString() → überschriebene toString()-Methode

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

225

## Domänenklasse: Attribute

- **statisch typisiert** (d.h. mit Datentypangabe)
  - Typsicherheit
  - Spaltentypen in DB leiten sich daraus ab
- **automatisch** in DB **persistiert** mit GORM
  - pro Attribut eine Tabellenspalte
  - Spaltentyp leitet sich aus Attributtyp ab
- Beziehungsattribute zu anderen Domänenklassen möglich
  - 1:1, 1:n, m:n-Beziehungen
  - werden später in der Vorlesung behandelt

## Domänenklasse: Constraints

- **Integritätsregeln** für Attribute

- Muss- und Kann-Attribute

- nullable Attribut darf auch null sein boolean
    - blank Attribut darf leeren String enthalten boolean

- Größenbeschränkungen

- maxSize Maximale Anzahl Zeichen im String Integer

- **Klassenvariable constraints**

- definiert Integritätsregeln für Attribute mit Domänspezifischer Sprache (DSL)

- statisch, weil für alle Instanzen gültig

- Name **constraints** fest vorgegeben (Convention over configuration)

- Realisierung als Closure

- keine Parameter, kein Rückgabewert

- Rumpf der Closure:

- pro Attribut ein gleichnamiger Methodenaufruf, benannte Parameter als Map

- Reihenfolge der Aufrufe legt Reihenfolge der Attribute in GSP fest

## Benannte Constraint-Parameter (Standard-Validatoren)(1/2)

Name	Typ	De-fault	Beispiel	Bedeutung für Attributwert
blank	Boolean	true	blank:false	false: darf nicht leer sein
nullable			nullable:false	false: " " null "
creditCard	Boolean	false	creditCard:true	true: muss Kreditkartennummer sein
email			email:true	true: " E-Mail-Adresse "
url			url:true	true: " URL "
unique	Boolean	false	unique:true	true: " eindeutig sein (DB-unique)
min	Integer	-	min:1	kleinster erlaubter Eingabewert
max			max:12	größter " "
range	Range	-	range:[1..12]	erlaubter Wertebereich
scale	Integer	3	scale:2	Fließkomma-Nachkommastellen

## Benannte Constraint-Parameter (Standard-Validatoren) (2/2)

Name	Typ	Default	Beispiel	Bedeutung für Attributwert
<i>inList</i>	String	-	<i>inList:["red","blue"]</i>	Liste erlaubter Werte
<i>notEqual</i>	Object	-	<i>notEqual:"Specht"</i>	dieser Wert ist <b>nicht</b> erlaubt
<i>matches</i>	String	-	<i>matches:/[a-zA-Z]+/</i>	muss mit regulärem Ausdruck matchen
<i>minSize</i>	Integer	-	<i>minSize:3</i>	min. Anzahl Zeichen / Elemente
<i>maxSize</i>			<i>maxSize:3</i>	max. " "
<i>size</i>	Range	-	<i>size:[0..3]</i>	Bereich für erlaubte Anzahl Einträge
<i>widget</i>	String	-	<i>widget:'textarea'</i>	ändert Darstellung in GSP

## Domänenklasse: `toString()`

- **String `toString()` Methode überladen**
  - **muss** String als Rückgabetyp haben
    - sonst funktioniert Überladen nicht
  - Anwendung: "Instanzname" ausgeben, z.B. für
    - Auswahlliste möglicher Instanzen in Fremdschlüsselbeziehungen
    - Debugging
- Weitere Methoden nicht sinnvoll
  - Anwendungslogik gehört in Controller / wird von dort aufgerufen
  - Domänenklassen als reine Datencontainer + Persistenzschnittstelle

## Controller

- steuert die Webseiten zu **einer** Domänenklasse
  - Navigationslogik
  - enthält Controlleraktionen
  - Schnittstelle zur Anwendungslogik
  - steuert Persistierung
- Realisierung als Groovy Klasse
  - kein Verbrauch der Vererbung
  - kein Konstruktor
  - **scaffold**-Attribut, das die verwaltete Domänenklasse angibt
    - nur für dynamisches Scaffolding
    - Beispiel: **static scaffold=Professor**
- Erstellung: *New → Grails Controller*
  - Controllername ohne Endung *Controller* eingeben, wird automatisch ergänzt
  - liegen im Ordner **Controllers** (Convention over configuration)

## Controller für dynamisches Scaffolding : ProfessorController

```
class ProfessorController {
```

```
 static scaffold=Professor
```

Domänenklasse, die  
dieser Controller managt

```
} // def index = {}
```

Achtung, diese automatisch  
erzeugte Zeile auskommentieren!

## Anwendung starten: Startseite mit Controllerliste

The screenshot shows a web browser window for a Grails application. The title bar says "Welcome to Grails". The address bar shows "localhost:8080/Professorenverwaltung/". The main content area has a green header with the "GRAILS" logo. On the left, there's a sidebar with "APPLICATION STATUS" and "INSTALLED PLUGINS" sections. The "APPLICATION STATUS" section lists various system details. The "INSTALLED PLUGINS" section lists "dataBinding - 2.3.1" and "logging - 2.3.1". The main content area starts with a "Welcome to Grails" message and then lists "Available Controllers" with two entries: [grails.plugin.databasemigration.DbdocController](#) and [professorenverwaltung.ProfessorController](#).

**APPLICATION STATUS**

- App version: 0.1
- Grails version: 2.3.1
- Groovy version: 2.1.8
- JVM version: 1.7.0\_71
- Reloading active: true
- Controllers: 2
- Domains: 1
- Services: 2
- Tag Libraries: 13

**INSTALLED PLUGINS**

- dataBinding - 2.3.1
- logging - 2.3.1

Welcome to Grails

Congratulations, you have successfully started your first Grails application! At the moment this is the default page, feel free to modify it to either redirect to a controller or display whatever content you may choose. Below is a list of controllers that are currently deployed in this application, click on each to execute its default action:

**Available Controllers:**

- [grails.plugin.databasemigration.DbdocController](#)
- [professorenverwaltung.ProfessorController](#)

## Listenansicht

The screenshot shows a web browser window titled "Professor Liste". The address bar shows "localhost:8080/Professorenverwaltung/professor/index". The main content area displays a table titled "Professor Liste" with columns: Vorname, Nachname, Kuerzel, Raum, Telefondurchwahl, and Bemerkungen. There are two rows of data: one for "Thomas Specht" (Kuerzel SPE, Raum A105b, Telefondurchwahl 6765) and one for "Michael Gröschel" (Kuerzel GRM, Raum A105b, Telefondurchwahl 6611).

Vorname	Nachname	Kuerzel	Raum	Telefondurchwahl	Bemerkungen
Thomas	Specht	SPE	A105b	6765	
Michael	Gröschel	GRM	A105b	6611	

# Eingabeformular

The screenshot shows a Java Swing application window titled "Professor anlegen". The window has a menu bar with German options: Datei, Bearbeiten, Ansicht, Chronik, Lesezeichen, Extras, Hilfe. Below the menu is a toolbar with icons for back, forward, search, and other functions. The main area is titled "Professor anlegen" and contains the following fields:

- Vorname \* (First Name) - An input field with a red border.
- Nachname \* (Last Name) - An input field with a red border.
- Kuerzel (Abbreviation) - An input field.
- Raum (Room) - An input field.
- Telefondurchwahl (Phone Extension) - An input field.
- Bemerkungen (Remarks) - A large text area.

At the bottom left is a "Anlegen" button with a floppy disk icon. The window has scroll bars on the right and bottom.

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

235



## Gliederung

3. Grails-Framework
  1. Einführung in Grails
  - 2. Controller**
    1. Dynamisches Scaffolding
    2. Statisches Scaffolding
    3. Manuelle Programmierung
  3. Domänenklassen
  4. RESTful Web Services
  5. GSP

## Controller erzeugen: 3 Möglichkeiten

### 1. dynamisches Scaffolding

- dynamische Generierung zur Laufzeit
- kein Quellcode erzeugt, nicht änderbar
- Anwendung: typische CRUD-Oberflächen, z.B. Administrator-Oberflächen

### 2. statisches Scaffolding ("Gerüstbau")

- statische Generierung als Quellcode aus der Domänenklasse
- beliebig änder- und erweiterbar
- Anwendung: CRUD-basierte, komplexere Oberflächen

### 3. manuelle Programmierung

- Quellcode manuell erstellen
- höherer Programmieraufwand
- Anwendung: Oberflächen abseits der üblichen CRUD-Funktionalität



## Gliederung

### 3. Grails-Framework

1. Einführung in Grails
2. Controller
  - 1. Dynamisches Scaffolding
  - 2. Statisches Scaffolding
  - 3. Manuelle Programmierung
3. Domänenklassen
4. RESTful Web Services
5. GSP

## Dynamisches Scaffolding

- Controller verwaltet genau **1 Domänenklasse**
  - gleicher Name wie Domänenklasse, Endung *Controller*
  - Quellcode beinhaltet nur **Zuordnung zur Domänenklasse**, sonst nichts
    - alles andere wird dynamisch zur Laufzeit erzeugt
- Controller erstellen:
  - Grails-Kommando: **create-controller Controllername**
  - Alternative: mit New → *Grails Controller* erstellen
  - Controllernamen ohne Endung Controller eingeben, wird automatisch ergänzt
  - generiert Controllerklasse als Quellcodegerüst:

```
class ProfessorController {
 static scaffold=Professor
 }

 // def index = {}
```

**Diese Zeile ergänzen ...**

**Domänenklasse**

**... und diese auskommentieren / löschen!**



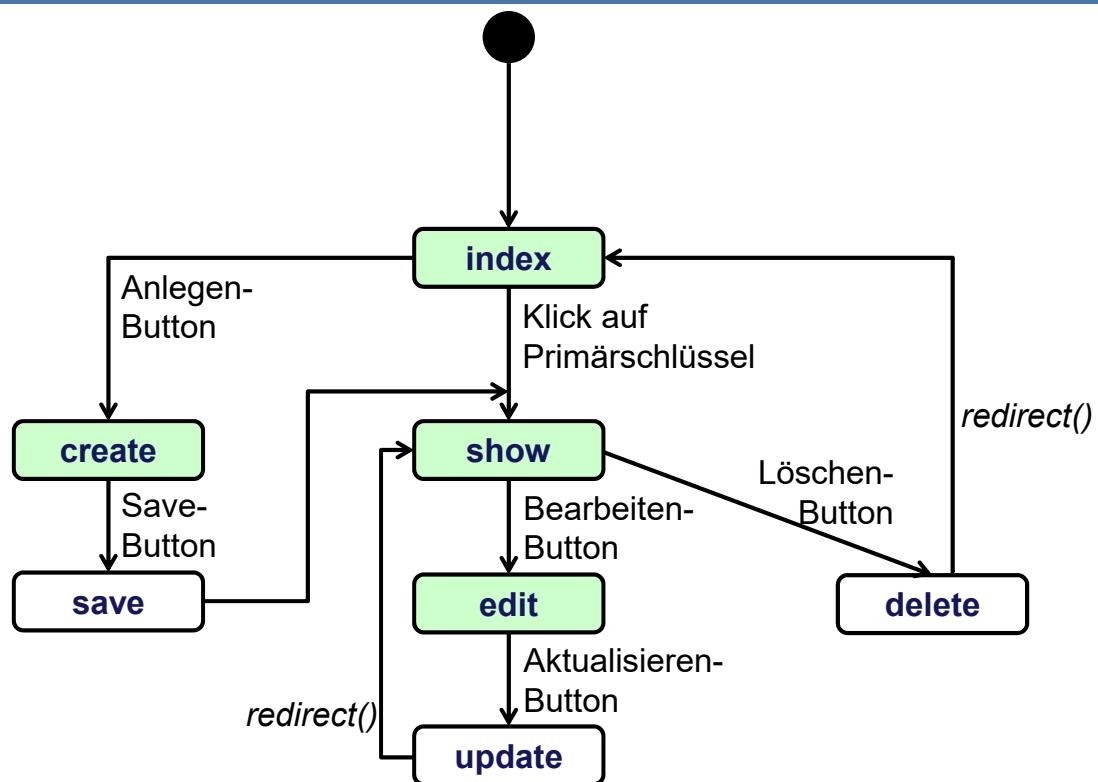
## Gliederung

3. Grails-Framework
  1. Einführung in Grails
  - 2. Controller**
    1. Dynamisches Scaffolding
    - 2. Statisches Scaffolding**
    3. Manuelle Programmierung
  3. Domänenklassen
  4. RESTful Web Services
  5. GSP

## Statisches Scaffolding

- Controller verwaltet genau **1 Domänenklasse**
  - gleicher Name wie Domänenklasse, Endung *Controller*
  - Controller als **Quellcode** generieren
    - enthält Methoden für Aktionen *index, show, delete, edit, update, create, save*
    - beliebig nachbearbeitbar
  - zugehörige Views (GSPs) als **Quellcode** generieren
    - beliebig nachbearbeitbar
- Grails-Kommandos für **statisches** Scaffolding
  - **generate-controller** Domänenklasse Controller als Quellcode generieren
  - **generate-views** Domänenklasse Views als Quellcode generieren
  - **generate-all** Domänenklasse Controller und Views generieren

## Standard-Navigationslogik im Controller



## Controlleraktionen

Aktion	zugehöriger View	Bedeutung
<i>index</i>	<i>index.gsp</i>	Einstiegsseite, listet alle Instanzen (Datensätze) der Domänenklasse auf
<i>show</i>	<i>show.gsp</i>	zeigt einzelne Instanz im Detail an
<i>delete</i>	-	löscht einzelne Instanz aus Datenbank
<i>edit</i>	<i>edit.gsp</i>	zeigt einzelne Instanz zum Editieren an
<i>update</i>	-	schreibt Änderungen an einzelner Instanz in DB
<i>create</i>	<i>create.gsp</i>	zeigt Formular zum Erstellen einer Instanz an
<i>save</i>	-	speichert neu erstellte Instanz in DB

- weitere Controlleraktionen nach Bedarf manuell implementieren

## Statisches Scaffolding: Controller-Kopf

```
package professorenverwaltungdynscaffold

import grails.validation.ValidationException
import static org.springframework.http.HttpStatus.*

class ProfessorController {
 ProfessorService professorService

 static allowedMethods = [save: "POST", update: "PUT", delete: "DELETE"]
```

Legt fest, welche Aktionen mit welcher HTTP-Methode aufgerufen werden dürfen:

- *save* nur mit POST
- *update* nur mit PUT
- *delete* nur mit DELETE
- alle nicht aufgeführten wahlweise mit GET, POST, PUT oder DELETE

## Statisches Scaffolding: index-Aktion

```
Controller-Aktion index
def index(Integer max) {
 HTTP-Parameter max, legt max. Anzahl Datensätze auf Seite fest
 params.max = Math.min(max ?: 10, 100)
 params: Map aller HTTP-Parameter
 10 als Default
 max auf 100 begrenzen
}

View darstellen,
Content Negotiation
list-Methode der Domänen-
klasse liefert Datensätze
Parameter für list-
Methode=HTTP-Parameter
respond professorService.list(params),
model:[professorCount: professorService.count()]
Model: Map aus key-value-Paaren
Key
Value
Zugriff auf Wert im View über ${key}
}

}
```

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

245

## Statisches Scaffolding: show, create

```
id der darzustellenden Instanz
def show(Long id) {
 respond professorService.get(id)
 darzustellende Instanz
 keine zusätzlichen Model-Daten
}

def create() {
 respond new Professor(params)
 darzustellende Instanz
 Konstruktorparameter = HTTP-Parameter
}
```

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

246

## Statisches Scaffolding: save (1/2)

```
def save(Professor professor) {
 if (professor == null) {
 notFound()
 return
 }

 try {
 professorService.save(professor)
 } catch (ValidationException e) {
 respond professor.errors, view:'create'
 return
 }

 // Auf der nächsten Seite geht's weiter
}
```

Annotations:

- notFound() → am Ende des Controllers definiert
- professorService.save(professor) → Änderungen in DB persistieren
- ValidationException e → Constraints verletzt
- professor.errors → darzustellende Fehlermeldungen
- view:'create' → Zurück zur create-View

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

247

## Statisches Scaffolding: save (2/2)

```
// Fortsetzung save-action...
request.withFormat {
 ... HTML-Formular (Browser)
 form multipartForm {
 flash.message = message(
 code: 'default.created.message',
 args: [message(code: 'professor.label',
 default: 'Professor'),
 professor.id
]
)
 redirect professor
 }
}
** { respond professor, [status: CREATED] }
}
```

Annotations:

- request.withFormat { → Je nachdem, welche Formate der Client verarbeiten kann (Content Negotiation)...
- ... HTML-Formular (Browser) → ... HTML-Formular (Browser)
- ... alle anderen Formate → ... alle anderen Formate
- respond professor, [status: CREATED] → Instanz → Model: Statusmeldung

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

248

## Statisches Scaffolding: edit

```
def edit(Long id) {
 respond professorService.get(id)
}
} darzustellende Instanz
```

## Statisches Scaffolding: update (1/2)

```
def update(Professor professor) {
 if (professor == null) {
 notFound() am Ende des Controllers definiert
 return
 }
 try {
 professorService.save(professor) Änderungen in DB persistieren
 } catch (ValidationException e) {
 respond professor.errors, view:'edit'
 return Fehlermeldungen Zurück zum edit-View
 }
 // Auf der nächsten Seite geht's weiter
```

## Statisches Scaffolding: update (2/2)

// Fortsetzung save-action...

```
request.withFormat {
 form multipartForm {
 flash.message = message(
 code: 'default.updated.message',
 args: [message(code: 'professor.label', default: 'Professor'),
 professor.id
]
)
 redirect professor
 }
 "*'{ respond professor, [status: OK] }
}
```

Leitet auf show-Action weiter

## Statisches Scaffolding: delete

```
def delete(Professor professor) {
 if (professor == null) {
 notFound()
 return
 }
 professorService.delete(id)
 request.withFormat {
 form multipartForm {
 flash.message = message(
 code: 'default.deleted.message',
 args: [message(code: 'professor.label', default: 'Professor'),
 professor.id])
 redirect action:"index", method:"GET"
 }
 "*'{ render status: NO_CONTENT }
 }
}
```

## Statisches Scaffolding: notFound-Methode

```
protected void notFound() {
 request.withFormat {
 form multipartForm {
 flash.message = message(
 code: 'default.not.found.message',
 args: [message(code: 'professor.label',
 default: 'Professor'),
 params.id])
 }
 redirect action: "index", method: "GET"
 }
 "*{ render status: NOT_FOUND }"
}
```

Leitet auf index-Aktion weiter

## Erlaubte HTTP-Methoden

- jede Controller-Aktion akzeptiert bestimmte HTTP-Methoden
  - **Default:** alle HTTP-Methoden erlaubt
  - **sinnvoll:** GET für Änderungsaktionen (*save*, *update*, *delete*) verbieten
    - erschwert URL-Manipulationen
    - verhindert Mitlesen von Parameterwerten auf der URL
- Konfiguration
  - Klassenattribut **allowedMethods** der Controllerklasse
  - assoziatives Array mit
    - key: Name der Controller-Aktion, z.B. *save*
    - value: erlaubte HTTP-Methode, z.B. "POST" oder Array erlaubter HTTP-Methoden, z.B. ["GET", "POST"]
  - nicht aufgeführten Controller-Aktionen akzeptieren alle HTTP-Methoden
- Default für durch dynamisches Scaffolding generierte Controller:  
**static allowedMethods = [save: "POST", update: "PUT", delete: "DELETE"]**

## Erlaubte HTTP-Methoden in generierten Controllern

- static allowedMethods = [save: "POST", update: "PUT", delete: "DELETE"]

Controller-Aktion	akzeptiere HTTP-Methode			
	GET	POST	PUT	DELETE
index	X	X	X	X
show	X	X	X	X
create	X	X	X	X
save		X		
edit	X	X	X	X
update			X	
delete				X

## Aufruf von Controller-Aktionen aus GSPs

- Delete-Button in show.gsp ruft delete-Methode auf:

```
<g:form resource="${this.professor}" method="DELETE">
 <fieldset class="buttons">
 <input class="delete" type="submit" value="Delete"/>
 </fieldset>
</g:form>
```

wenn Action fehlt:  
aus HTTP-Methode  
ableiten

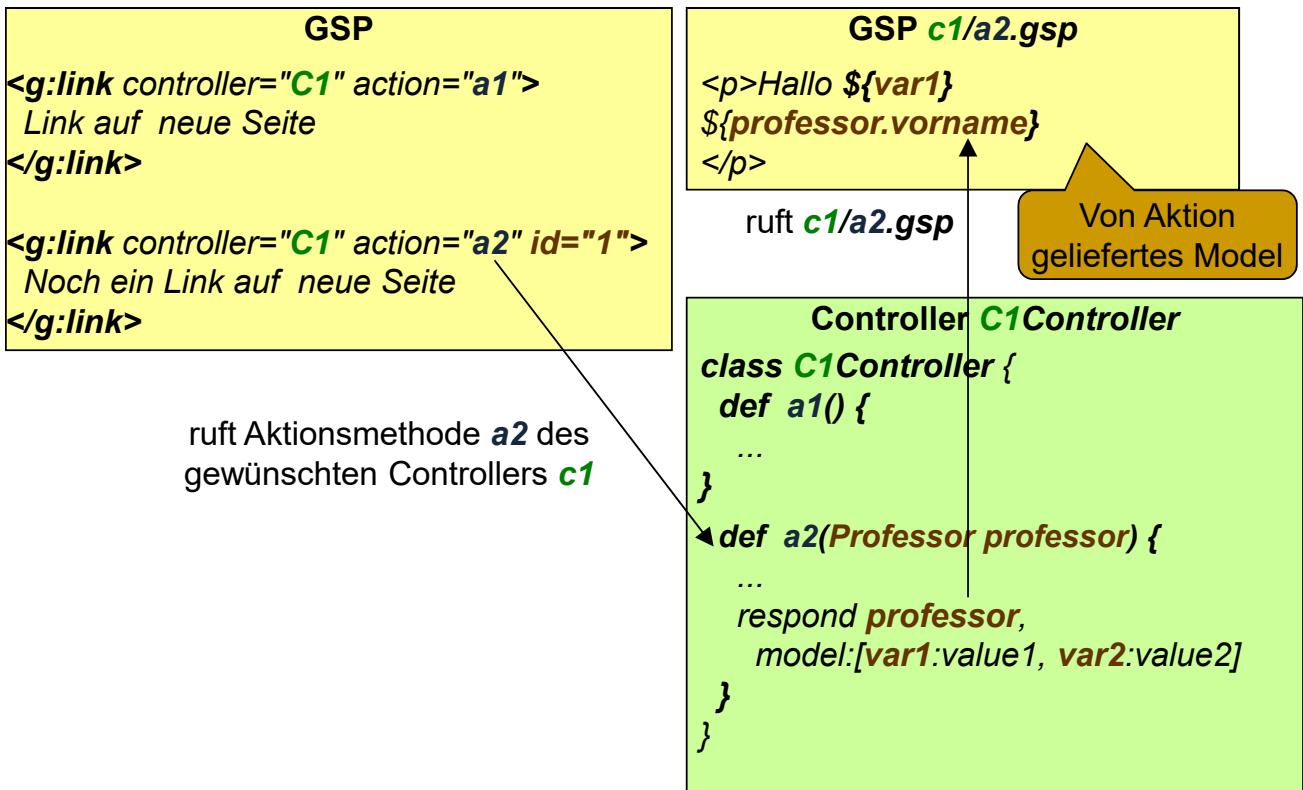
- Edit-Button in show.gsp ruft edit-Action auf:

```
<g:link class="edit" action="edit" resource="${this.professor}">Edit</g:link>
```

- Save-Button in create.gsp ruft save-Methode auf:

```
<g:form resource="${this.professor}" method="POST">
 <fieldset class="buttons">
 <g:submitButton name="create" class="save" value="Create" />
 </fieldset>
</g:form>
```

## Zusammenspiel Controller ↔ GSP



## Parameterübergabe View (GSP) → Controlleraktion

- alle HTTP-Parameter in Umgebungsvariable *params*
  - assoziatives Array
- für jeden HTTP-Parameter (GET/POST) gleichnamiger formaler Parameter
  - freiwillig, d.h. formale Parameter dürfen weggelassen werden
  - Datentyp **muss** in Signatur angegeben sein!
  - **erlaubte** Datentypen:
    - elementare Datentypen + deren **Wrapper**
    - **String**
- automatische Bereitstellung der **Domäneninstanz** für *id*-Parameter
  - HTTP-Parameter muss **id** heißen
    - per */idValue* an die URL angehängt oder
    - per *?id=idValue* (*GET-Parameter*)
  - Parameter muss vom Typ der **Domänenklasse** sein

## Controller-Aktion auslösen: Link in GSP

```
<g:link controller="Professor" action="list">Alle Professoren auflisten</g:link>
```

- Attribute des *g:link* Tags:
  - **controller**: Controller, auf dem *action* aufzurufen ist
    - Controllername ohne Endung *Controller*
    - Defaultwert: eigener Controller
  - **action**: auf dem Controller aufzurufende *action*
    - Defaultwert: Default action, d.h. *index*
  - **id**: mit Link zu übermittelnde Domäneninstanz-ID
  - **params**: assoziatives Array mit HTTP-Parametern (GET-Parameter an URL)
    - umschließende **[ ]** mit angeben!
      - Beispiel: *params*=“[*vorname*:‘Thomas’,*nachname*:‘Specht’]”
      - ganzen Params-Wert in  **\${...}**, nicht einzelne Attributwerte!
      - Beispiel: *params*=“ **\${[kuerzel:professor.kuerzel]}**”
  - **fragment**: Fragment (Anker) in der aufzurufenden Seite (per # in URL)
  - **url**: Assoziatives Array mit Attributwerten wie Controller, Action ..., z.B.  
`<g:link url="[action:'index',controller:'professor']">Professor List</g:link>`

## Controller-Aktion auslösen: Form in GSP

```
<g:form name="ProfessorForm" method="post" >
 <g:textField name="vorname" value="${professor?.vorname}" />
 ...
 <g:actionSubmit value="Update" action="update"/>
</g:form>
```

- Attribute des *g:form* Tags:
  - **method**: GET, POST, PUT oder DELETE
  - weitere wie *g:link*
- Attribute des *g:actionSubmit* Tags:
  - **value**: Buttonbeschriftung
  - **action**: aufzurufende *action* (sonst legt *value* auch *action* fest)
- Anmerkungen:
  - Formular kann mehrere Submit-Buttons haben
    - selber Controller
    - unterschiedliche Aktionen
  - *actionSubmit*-Button wirkt wie ein Methodenaufruf (=Controller-Aktion)

## URL des Projekts



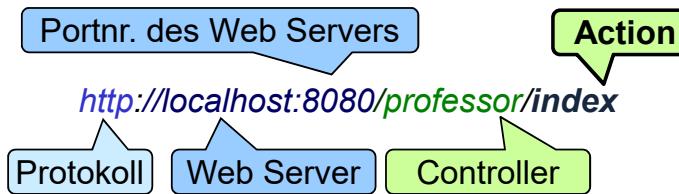
- Servlets verbergen sich hinter
  - **Projektpfad**
    - z.B.: <http://localhost:8080/>  
→Grails-Standard servlet mit Liste der Controller

## URL eines Controllers



- Servlets verbergen sich hinter
  - Projektpfad
    - z.B.: <http://localhost:8080/>  
→Grails-Standard servlet mit Liste der Controller
  - jedem **Controller**
    - z.B.: <http://localhost:8080/professor/>  
→eigenes Servlet, das den Controller implementiert

## URL eines Controllers mit Aktion



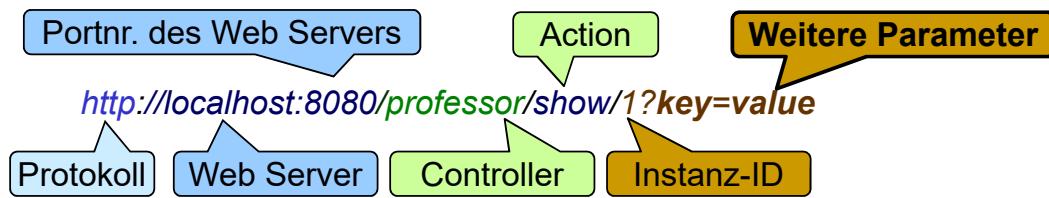
- Servlets verbergen sich hinter
  - Projektpfad
    - z.B.: <http://localhost:8080/>  
→Grails-Standard servlet mit Liste der Controller
    - jedem Controller
      - z.B.: <http://localhost:8080/professor/>  
→eigenes Servlet, das den Controller implementiert
      - Parameter für dieses Servlet:
        - **Name der Aktion**

## URL eines Controllers mit Aktion und Domäneninstanz-ID



- Servlets verbergen sich hinter
  - Projektpfad
    - z.B.: <http://localhost:8080/>  
→Grails-Standard servlet mit Liste der Controller
    - jedem Controller
      - z.B.: <http://localhost:8080/professor/>  
→eigenes Servlet, das den Controller implementiert
      - Parameter für dieses Servlet:
        - Name der Aktion
        - **ID der Domäneninstanz**

## URL eines Controllers mit Aktion, ID + weiteren Parametern



- Servlets verbergen sich hinter
  - Projektpfad
    - z.B.: `http://localhost:8080/`  
→Grails-StandardServlet mit Liste der Controller
    - jedem Controller
      - z.B.: `http://localhost:8080/professor/`  
→eigenes Servlet, das den Controller implementiert
      - Parameter für dieses Servlet:
        - Name der Aktion
        - ID der Domäneninstanz
        - weitere Parameter

## Methodensignatur für parameterlose Aktion

- Signatur der Aktion parameterlos
- Zugriff auf HTTP-Parameter (GET/POST) über Umgebungsvariable **params**
  - assoziatives Array mit allen Formulareingaben
  - Zugriff auf Parameterwerte:
    - Parametername
    - `params.vorname`
    - Parametername
    - `params["vorname"]`
- vgl. `$_GET / $_POST` in PHP
- keine Unterscheidung nach GET- und POST-Methode
- Beispiel:

```
def create() {
 respond new Professor(params)
}
```
- Aufruf:  
`http://localhost:8080/professor/create`

## Methodesignatur für Aktion mit HTTP-Parametern

- zu jedem HTTP-Parameter **gleichnamiger** formaler Parameter in Signatur
  - Datentyp **muss** in Signatur angegeben sein
    - erlaubt: **elementare** Datentypen + deren **Wrapper**, **String**
  - aktueller Parameterwert = HTTP-Parameterwert
  - keine Unterscheidung nach GET- und POST-Methode
  - freiwillig, d.h. formale Parameter dürfen weggelassen werden
  - Zugriff über **params** trotzdem möglich
    - beide Varianten kombinierbar
- Beispiel:

```
def hallo(String vorname, String nachname) {
 ...
}
```
- Aufruf:  
`http://localhost:8080/professor/hallo?vorname=Thomas&nachname=Specht`

## Methodesignatur für Aktion mit HTTP-Parametern

- zu jedem HTTP-Parameter **gleichnamiger** formaler Parameter in Signatur
  - Datentyp **muss** in Signatur angegeben sein
    - erlaubt: **elementare** Datentypen + deren **Wrapper**, **String**
  - aktueller Parameterwert = HTTP-Parameterwert
  - keine Unterscheidung nach GET- und POST-Methode
  - freiwillig, d.h. formale Parameter dürfen weggelassen werden
  - Zugriff über **params** trotzdem möglich
    - beide Varianten kombinierbar
- Beispiel:

```
def index(Integer max) {
 params.max = Math.min(max ?: 10, 100)
 respond professorService.list(params),
 model:[professorCount: professorService.count()]
}
```
- Aufruf:  
`http://localhost:8080/professor/show?max=15`

## Methodesignatur für Aktion mit Domäneninstanz

- HTTP-Parameter ***id*** wird übergeben
  - erster Parameter in Signatur muss vom Typ **Domänenklasse** sein
  - Wenn Datensatz in Datenbank existiert:
    - Grails liefert **Domäneninstanz** mit dieser ID
    - *null*, falls Datensatz nicht gefunden
- für alle weiteren HTTP-Parameter
  - **gleichnamiger** formaler Parameter in Signatur
  - vgl. vorige Folie
- Beispiel:

```
def edit(Professor professor) {
 respond professor
}
```
- Aufruf: <http://localhost:8080/professor/edit/1>

ID des  
Datensatzes

## Methodesignatur für Aktion mit Domäneninstanz

- HTTP-Parameter ***id*** wird übergeben
  - erster Parameter in Signatur muss vom Typ **Domänenklasse** sein
  - Wenn Datensatz in Datenbank existiert:
    - Grails liefert **Domäneninstanz** mit dieser ID
    - *null*, falls Datensatz nicht gefunden
- für alle weiteren HTTP-Parameter
  - **gleichnamiger** formaler Parameter in Signatur
  - vgl. vorige Folie
- Beispiel:

```
def hallo(Professor professor, String ort) {
 [begruebung: "Hallo $professor.name aus $ort"];
}
```
- Aufruf: <http://localhost:8080/professor/hallo/1?ort=Mannheim>

ID des  
Datensatzes

## Umgebungsvariablen im Controller

Umgebungsvariable	Bedeutung	Beispiel
<code>controllerName</code>	Name des Controllers	<code>professor</code>
<code>actionName</code>	Name der Controller-Aktion	<code>index</code>
<code>params</code>	Assoziatives Array mit HTTP-Parametern	<code>[action:index, format:null, controller:professor]</code>
<code>request</code>	<code>HttpServletRequest</code> des Servlet-Aufrufs	Zu Low-Level, deshalb hier nicht behandelt
<code>servletContext</code>	<code>ServletContext</code> des Servlets	
<code>response</code>	<code>HttpServletResponse</code> des Servlet-Aufrufs	
<code>session</code>	<code>HttpSession</code> , Session-Variablen als ass. Array	
<code>flash</code>	Objekt zum Ablegen von Fehlermeldungen	<code>[:]</code>
<code>log</code>	Logger zum Schreiben von Logfiles	

## Zugriff auf Sessionvariablen mit `session`

- Session automatisch erzeugt
  - Browser speichert Session-ID in Cookie
  - überlebt gesamte HTTP-Session
  - Sessionvariablen einfach beschreiben, ohne sie zu deklarieren
- Variable **session** wie assoziatives Array mit allen Sessionvariablen
  - Hinzufügen / Beschreiben einer Sessionvariablen:
    - **Sessionvariable**
    - **session.userID = params.id**
  - Auslesen einer Sessionvariablen:
    - **Sessionvariable**
    - **println session.userID**
  - Entsorgen einer Sessionvariablen:
    - **session.removeAttribute("userID")**
    - Alternative: Sessionvariable auf `null` setzen

## (Fehler-)meldung an nächste GSP liefern mit *flash*

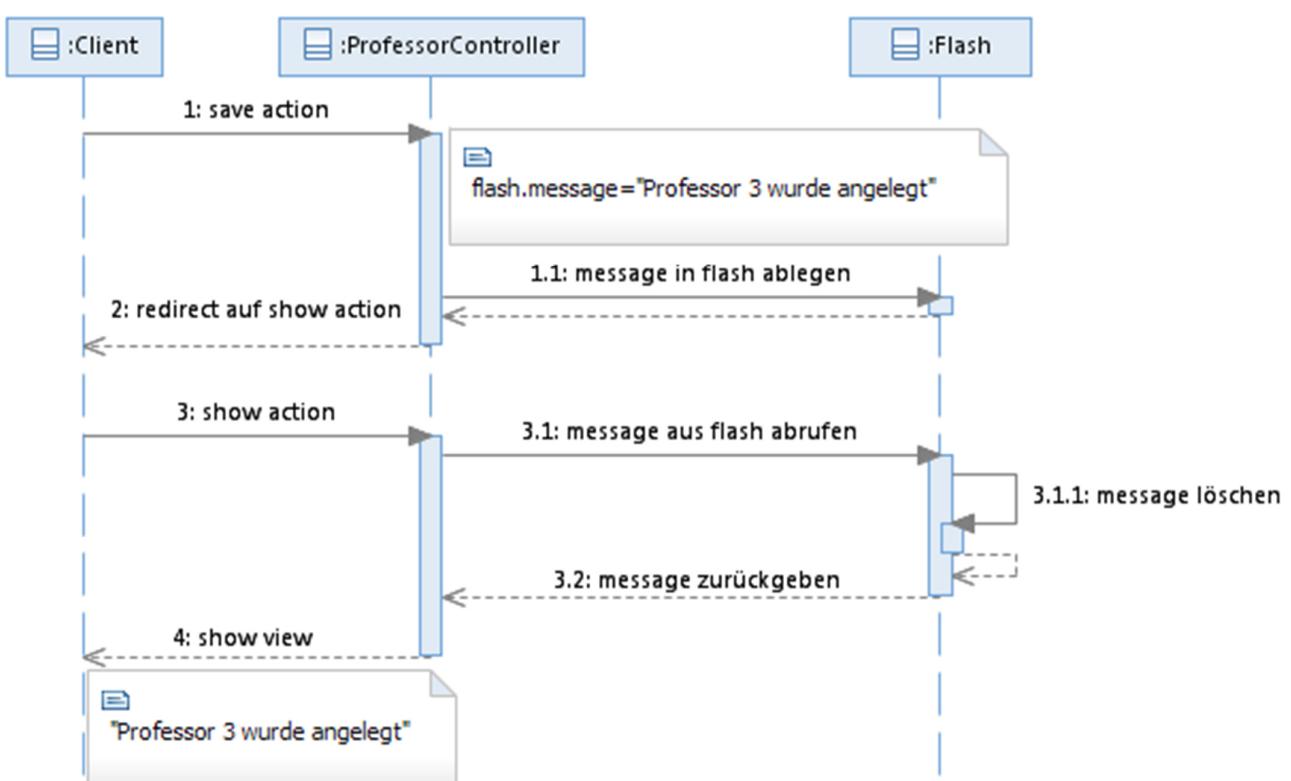
- *flash*-Objekt automatisch erzeugt
  - Controller legt Meldung als String in Attribut *flash.message*
    - Beispiel:

```
Internationalisierungsfunktion
flash.message = message(
 code: 'default.deleted.message',
 args: [message(code: 'Professor.label', default: 'Professor'),
 professorInstance.id])
```

    "{} {} deleted"  
    {0}: Professor  
    {1}: Domäneninstanz-ID
  - unmittelbar nächste GSP
    - zeigt Meldung an mit

```
<g:if test="${flash.message}">
 ${flash.message}
</g:if>
```
    - danach wird *flash.message* automatisch **gelöscht**
      - Meldung wird nur einmal angezeigt

## Ablauf bei (Fehler-)meldungen mit *flash*



## Logging in Grails

- Logging von **Informationen**, **Warnungen** und **Fehlermeldungen**
  - besser per **Logger** statt mit `System.out.println()`
  - umlenkbar in Logdatei oder Datenbanktabelle
- Logging in **Grails**
  - Technische Basis seit Grails 3: **Logback**
    - Logging-Interface: `org.apache.commons.logging.Log`
    - steht allen Controllern als Umgebungsvariable **log** zur Verfügung
  - Standardmäßiger **Root-Logger**
    - gibt nur Fehler (error) und schwere Fehler (fatal error) aus
    - standardmäßig in die Konsole (DOS-Box)
  - Logging-Konfigurationsdatei: `conf/logback.groovy`

## Logging Level

Logging-Level	Was wird geloggt?	Beispiel
off	Logging ausgeschaltet	
fatal	nur schwere, nicht behebbarer Fehler	DB-Verbindung abgebrochen
error	+ Programmfortsetzung trotz Fehler	konnte Daten nicht speichern
warn	+ Warnungen vor kritischen Situationen	unerlaubter Parameterwert
info	+ Informationen zum Programmfortschritt	aufgerufene Controller-Action
debug	+ Debug-Meldungen	wichtige Variablenwerte
trace	+ detaillierte Debug-Meldungen	detaillierter Stack Trace
all	alle oben genannten Levels	

+ Level schließt die darüber stehenden Levels mit ein

## Log-Meldung auslösen

- Umgebungsvariable **log**
  - steht allen Controllern zur Verfügung
  - bietet zu jedem Logging-Level **lev** zwei Methoden:
    - log.lev(Object message)**



- Beispiele:

```
– log.info "Entering index ..." // index-Aktion betreten
– try {
 ...
} catch (IOException ioException) {
 log.error "Fehler beim Dateizugriff", ioException
}
```

## Deklarative Logger-Konfiguration in logback.groovy

- Syntax:  
**logger 'zuLoggendeKomponente', LoggingLevel**  
zu loggende Komponente      Logging-Level
- Logging Level: **FATAL, ERROR, WARN, INFO, DEBUG, TRACE, OFF** oder **ALL**
- Zu loggende Komponente
  - vollständig qualifizierter Package-Pfad zwingend erforderlich
    - Beispiel: `professorenverwaltung.ProfessorController`
    - wenn ohne Klassenname am Ende: alle Klassen des Pakets
    - wenn Klassenname angegeben: nur für diese Klasse

## Deklarative Logger-Konfiguration: Beispiel 1/2

```
import grails.util.BuildSettings
import grails.util.Environment

// See http://logback.qos.ch/manual/groovy.html for details on configuration
appender('STDOUT', ConsoleAppender) {
 encoder(PatternLayoutEncoder) {
 pattern = "%level %logger - %msg%n"
 }
}
root(ERROR, ['STDOUT'])
logger 'professorenverwaltung',INFO

nur Fehlermeldungen vom
Level error und höher loggen
für alles im Paket Professorenverwaltung
vom Level info und höher loggen
```

## Deklarative Logger-Konfiguration: Beispiel 2/2

```
def targetDir = BuildSettings.TARGET_DIR
if (Environment.isDevelopmentMode() && targetDir) {
 appender("FULL_STACKTRACE", FileAppender) {
 file = "${targetDir}/stacktrace.log"
 append = true
 encoder(PatternLayoutEncoder) {
 pattern = "%level %logger - %msg%n"
 }
 }
 logger("StackTrace", ERROR, ['FULL_STACKTRACE'], false)
}
```

## Alles in einem Package loggen

- umfasst
  - Domänenklassen
  - Controller
  - Services
  - Tag-Libraries
- Syntax:  
`logger 'professorenverwaltung', INFO`



## Controller loggen

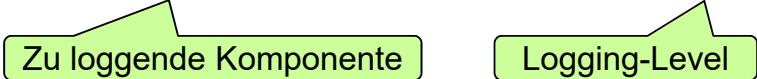
- Bestimmten Controller im Default-Package loggen:  
`logger 'TestController', INFO`  
→ Controller im Default-Package
- Bestimmten Controller im eigenen Package loggen:  
`logger 'professorenverwaltung.ProfessorController', INFO`  
→ Controller mit Package-Pfad

## Beispiel: Controller Professor loggen

```
import grails.util.BuildSettings
import grails.util.Environment

// See http://logback.qos.ch/manual/groovy.html for details on configuration
appender('STDOUT', ConsoleAppender) {
 encoder(PatternLayoutEncoder) {
 pattern = "%level %logger - %msg%n"
 }
}

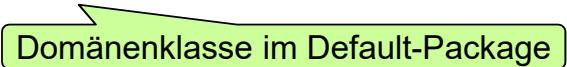
root ERROR, ['STDOUT']
logger 'professorenverwaltung.ProfessorController', INFO
```



## Domänenklasse loggen

- Bestimmte Domänenklasse im Default-Package loggen:

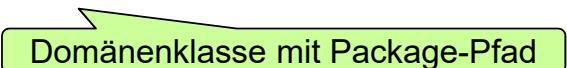
logger 'TestDomaenenKlasse', INFO



Domänenklasse im Default-Package

- Bestimmte Domänenklasse im eigenen Package loggen:

logger 'professorenverwaltung.Professor', INFO



Domänenklasse mit Package-Pfad

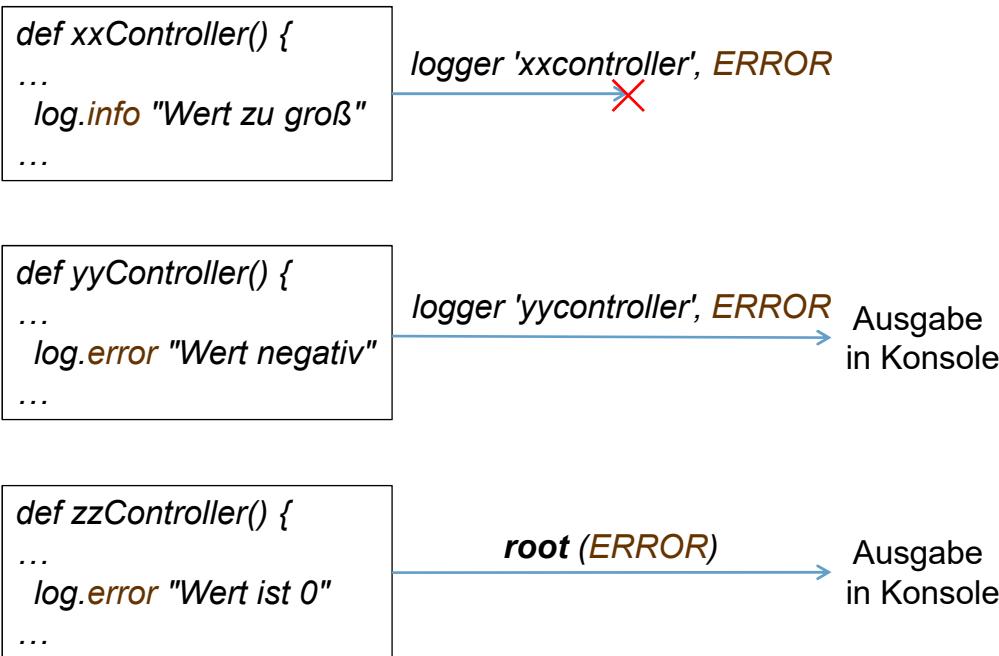
## Root-Logger

- **Root Logger**
  - für alle Log-Nachrichten **ohne** eigene Logging-Regel
  - Standardmäßig generiert als

```
Appender (Ausgabekanal) Ausgabe in Konsole
appender('STDOUT', ConsoleAppender) {
 encoder(PatternLayoutEncoder) {
 pattern = "%level %logger - %msg%n"
 } } Formatstring

root(ERROR, ['STDOUT'])
Logging-Level Liste der Appender (Ausgabekanäle)
```

## Zusammenspiel der Logging-Regeln



## Umleitung der Log-Meldungen mit Appendern

- **Appender:** Ausgabekanal für Log-Nachrichten
- Standard-Appender '**STDOUT**'
  - gibt in Konsole aus
  - automatisch generiert in *config/logback.groovy*:

The diagram shows a snippet of *logback.groovy* configuration for the **STDOUT** appender. It includes callout boxes and arrows pointing to specific parts of the code:

- Name des Appenders:** appender('STDOUT', ConsoleAppender)
- Appender-Typ:** ConsoleAppender
- Logging-Level:** %level
- Logging-Message:** %msg%n
- Ausgabeformat:** pattern = "%level %logger - %msg%n"
- Loggende Komponente:** encoder(PatternLayoutEncoder)

```
appender('STDOUT', ConsoleAppender) {
 encoder(PatternLayoutEncoder) {
 pattern = "%level %logger - %msg%n"
 }
}
```

## Definition eines File-Appenders

- Definition des File-Appenders **PROTOKOLL**

The diagram shows a snippet of *logback.groovy* configuration for the **PROTOKOLL** file appender. It includes callout boxes and arrows pointing to specific parts of the code:

- Name des Appenders:** appender('PROTOKOLL', FileAppender)
- Appender-Typ:** FileAppender
- Name der Logdatei:** file = /c:\temp\testFile.log/
- Bei Neustart nicht löschen:** append = true

```
appender('PROTOKOLL', FileAppender) {
 file = /c:\temp\testFile.log/
 append = true
 encoder(PatternLayoutEncoder) {
 pattern = "%level %logger - %msg%n"
 }
}
```

- Nutzung des File-Appenders **PROTOKOLL**

logger 'professorenverwaltung.ProfessorController',  
INFO,[**PROTOKOLL**]

Liste der Appender (Ausgabekanäle)

## Zuordnung eines Appenders an einen Logger

- Syntax:

```
logger 'zuLoggendeKomponente', LoggingLevel, ['MEINAPPENDER']
```

zu loggende Komponente

Logging-Level

Appenderliste

- Komponenten dürfen auf mehrere Appender geloggt werden
  - kommagetrennte Liste von Appendern
- alle per Appender umgeleiteten Log-Nachrichten
  - gehen **zusätzlich** auf Root-Logger

## Umleitung des Root-Loggers auf einen File-Appender

- Konfiguration des File-Appenders:

```
appender('PROTOKOLL', FileAppender) {
 file = /c:\temp\testFile.log/
 append = true
 encoder(PatternLayoutEncoder) {
 pattern = "%level %logger - %msg%n"
 }
}
```

- Zuordnung des File-Appenders an den Root-Logger

– Logmeldungen ausschließlich an den File-Appender:

```
root(ERROR, ['PROTOKOLL'])
```

Logging-Level

Appender

– Logmeldungen an den File-Appender und auf Konsole:

```
root(ERROR, ['PROTOKOLL','STDOUT'])
```

## Appender-Typen

Appender-Typ	Wohin wird geloggt?
DBAppender	Datenbanktabelle via JDBC-Connection
ConsoleAppender	Konsole der Entwicklungsumgebung / DOS-Box
FileAppender	einzelne Datei, die bei Neustart überschrieben wird
RollingFileAppender	rollierende Dateien (z.B. jeden Tag eine neue)

Dokumentation: <http://logback.qos.ch/manual/appenders.html>

## Encoder-Pattern

Pattern	Beschreibung
%class	Klasse, die geloggt hat
%date{HH:mm:ss.SSS}	Logging-Zeitpunkt
%date{dd MMM yyyy HH:mm:ss.SSS}	Logging-Datum und Zeitpunkt
%level	Datenbanktabelle via JDBC-Connection
%logger	Name des Loggers (inkl. Package-Pfad)
%logger{0}	Name des Loggers ohne Package-Pfad
%msg	Logging-Nachricht
%n	Zeilenvorschub

Dokumentation: <http://logback.qos.ch/manual/layouts.html>

## Ergebnisrückgabe ohne Content Negotiation: Übersicht

Aufruf	Model	View	Zugriff in GSP
<code>render "Das hier ist der Ergebnisstring"</code>	String zur direkten Darstellung ohne Layout		-
<code>[vorname: "Thomas", nachname: "Specht"]</code>	Assoziatives Array	<code>action.gsp</code>	<code> \${vorname}</code> <code> \${nachname}</code>
<code>render view:"list", model:[professorList: professorService.list()]</code>	<i>model</i> -Parameter (assoziatives Array)	<code>list.gsp</code>	<code> \${professorList[0]}</code>
<code>render professorService.list() as XML</code>	Professorenliste	XML-Format	-
<code>render professorService.list() as JSON</code>		JSON-Format	-

## Direktes Rendern der Antwort ohne View

- Controller löst direktes Rendern des Ergebnisses aus
  - kein Model benötigt
  - kein View (GSP) benötigt
    - Ergebnisstring wird direkt zum Client zurückgeschickt
    - keine Einbindung in eine HTML-Seite
    - geeignet z.B. um XML- oder JSON-Daten zu liefern
  - Aufruf: ***render "Das hier ist der Ergebnisstring"***
- Beispiel:

```
class Professor {
 // ...
 def hello() {
 render "Guten Morgen, Herr Specht!"
 }
 // ...
}
```



## Parameterrückgabe Controller: Model

- Controller liefert **Model**
  - Aktions-Methode gibt assoziatives Array mit Variablenwerten für GSP zurück:

```
def hello() {
 Variablenname Variablenwert
 [vorname: "Thomas", nachname: "Specht"]
}
```
  - aufzurufende View: Name der Controller-Aktion + .gsp:
    - Beispiel: *hello.gsp*
- Zugriff auf die Variablenwerte in der GSP
  - mit  **`${variablenname}`**
  - Mengenklammern dürfen **nicht** weggelassen werden
  - Beispiel: *hello.gsp*

```
<body>
 Hallo ${vorname} ${nachname}

</body>
```

## Parameterrückgabe Controller: Model + View

- Controller löst Rendern eines **Views** (GSP) aus und liefert **Model** dazu
  - *myModel*: Assoziatives Array mit Variablen-Werte-Paaren
  - Aufruf: **`render view:"viewName", model:myModel`**
    - Endung *.gsp* im View-Namen weglassen!
  - falls View-Name=Action-Name, reicht bloße Rückgabe des Models
    - vgl. vorige Folie
- Beispiel:
  - Controller:

```
class Professor {
 // ...
 def index() {
 render view:"list", model:[professorList: professorService.list()]
 }
 // ...
}
```
  - Aufgerufene View: ***list.gsp***
    - Zugriff auf Professorenliste mit  **`${professorList}`**

## Rendern der Domäneninstanz(en) ins XML-Format

- Controller löst direktes Rendern des Ergebnisses im XML-Format aus
  - Model: Domäneninstanz oder Liste von Domäneninstanzen
  - kein View (GSP) benötigt
    - Domäneninstanz(en) im XML-Format direkt zum Client zurückgeschickt
  - erfordert Import von **grails.converters.\***
  - Aufruf: **render domainInstance as XML**

- Beispiel:

- Controller:

```
class Professor {
 // ...
 def indexXML() {
 render professorService.list() as XML
 }
 // ...
```

Typecast wandelt Liste von Domänen-Instanzen nach XML um

The screenshot shows a browser window with the URL `http://localhost:8080/Profe`. The page content is an XML document representing a list of professor instances. The XML structure includes a root element `<list>`, followed by a single `<professor id="1">` element, which contains fields like `bemerkungen`, `kuerzel`, `nachname`, `raum`, `telefondurchwahl`, and `vorname`. The `nachname` field has the value `Specht`, and the `vorname` field has the value `Thomas`.

```
<list>
- <professor id="1">
 <bemerkungen/>
 <kuerzel/>
 <nachname>Specht</nachname>
 <raum/>
 <telefondurchwahl/>
 <vorname>Thomas</vorname>
</professor>
</list>
```

## Rendern der Domäneninstanz(en) ins JSON-Format

- Controller löst direktes Rendern des Ergebnisses im XML-Format aus
  - Model: Domäneninstanz oder Liste von Domäneninstanzen
  - kein View (GSP) benötigt
    - Domäneninstanz(en) im JSON-Format direkt zum Client zurückgeschickt
  - erfordert Import von **grails.converters.\***
  - Aufruf: **render domainInstance as JSON**

- Beispiel:

- Controller:

```
class Professor {
 // ...
 def indexJSON() {
 render professorService.list() as JSON
 }
 // ...
```

Typecast wandelt Liste von Domänen-Instanzen nach JSON um

The screenshot shows a code editor window titled "Unbenannt - Editor". The content is a JSON array representing a list of professor instances. Each instance is a object with properties: `class`, `id`, `bemerkungen`, `kuerzel`, `nachname`, `raum`, `telefondurchwahl`, and `vorname`. The `nachname` property has the value `Specht`, and the `vorname` property has the value `Thomas`.

```
[{"class": "professorenverwaltung.Professor",
 "id": 1,
 "bemerkungen": null,
 "kuerzel": null,
 "nachname": "Specht",
 "raum": null,
 "telefondurchwahl": null,
 "vorname": "Thomas"}]
```

## Direktes Rendern der Antwort ins XML-Format

- Controller löst direktes Rendern des Ergebnisses im XML-Format aus
  - Model: Code-Block, der via Builder XML erzeugt
  - kein View (GSP) benötigt
    - Domäneninstanz(en) im XML-Format direkt zum Client zurückgeschickt
  - Aufruf: `render (contentType:"text/xml") { ... }`
- Beispiel:

```
class Professor {
 def hello() {
 render (contentType:"text/xml") {
 professorenliste {
 professor(vorname:"Thomas",nachname:"Specht")
 professor(vorname:"Rainer",nachname:"Gerten")
 }
 }
 }
}
```

Zielformat: XML  
() müssen hier stehen!

- <professorenliste>  
 <professor vorname="Thomas" nachname="Specht"/>  
 <professor vorname="Rainer" nachname="Gerten"/>  
</professorenliste>

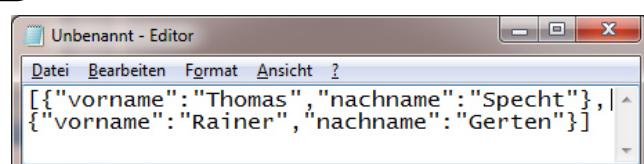
## Direktes Rendern der Antwort ins JSON-Format

- Controller löst direktes Rendern des Ergebnisses im JSON-Format aus
  - Model: Code-Block, der via Builder JSON erzeugt
  - kein View (GSP) benötigt
    - Domäneninstanz(en) im JSON-Format direkt zum Client zurückgeschickt
  - Aufruf: `render (contentType:"application/json") { ... }`
- Beispiel:

```
class Professor {
 def hello() {
 render (contentType:"application/json") {
 [element(vorname:"Thomas",nachname:"Specht"),
 element(vorname:"Rainer",nachname:"Gerten")]
 }
 }
}
```

() müssen hier stehen!  
Zielformat: JSON

JSON-Arrayelemente immer mit element()-Aufruf!  
JSON-Array



## Content Negotiation

- Grails verfügt über eingebaute **Content Negotiation**
  - Endgerät teilt gewünschtes Format mit
  - Controller-Aktion versucht Wunsch zu erfüllen
- Ziel: **Optimales Format** für jedes **Endgerät**, z.B.
  - HTML für Browser
  - XML oder JSON für AJAX-Aufrufe
- Mitteilung des Wunschformats durch Endgerät (z.B. Browser, Mobile App)
  - im **HTTP-Header-Attribut Accept** als MIME Content-Typ
    - Beispiel eines Browsers:  
*Accept: text/html,application/xhtml+xml,application/xml;q=0.9,\*/\*;q=0.8*
    - mit . an **URL** angehängt als **Grails Content-Typ**
      - hinter Controller-Aktion + ggf. id-Wert, vor ? und #
    - Beispiele:
      - *http://localhost:8080/professor/show/1.xml*
      - *http://localhost:8080/professor/index.xml*
  - Antwort der Controller-Aktion mit *respond*-Methode

## Content Negotiation: Vordefinierte Grails Content-Typen

Grails C.-Typ*	MIME Content-Typ	Anwendung
all	*/*	
atom	application/atom+xml	
css	text/css	Cascading Stylesheet
csv	text/csv	
form	application/x-www-form-urlencoded	HTML-Formular
html	['text/html','application/xhtml+xml']	HTML
js	text/javascript	JavaScript
json	['application/json', 'text/json']	JSON
multipartForm	multipart/form-data	
pdf	application/pdf	
rss	application/rss+xml	
text	text/plain	ASCII-/ Unicode-Text
hal	['application/hal+json','application/hal+xml']	
xml	['text/xml', 'application/xml']	XML

\* definiert in *conf/application.yml*

## Ergebnisrückgabe mit Content Negotiation: Übersicht

Aufruf	Model html, form, multipartForm	Model JSON, XML	View	Zugriff in GSP nur für html, form, multipartForm
<b>respond</b> <i>professor,</i> <b>model:[wohnort</b> : "Mannheim"]	[professor: professor, wohnort: "Mannheim"]	{"id":1,"vorname": "Thomas","kuerze I":"SPE","nachna me":"Specht"}		<code> \${professor.vorna me} aus \${wohnort}</code>
<b>respond</b> <i>professorList,</i> <b>model:[wohnort</b> : "Mannheim"]	[professorList: professorList, wohnort: "Mannheim"]		action.gsp	<code> \${professorList[0] .vorname} aus \${wohnort}</code>

## Content Negotiation: respond Domäneninstanz

- Automatische Content Negotiation in Grails per *respond*-Methode:

**respond domänenInstanz, model:[ wohnort: "Mannheim"]**

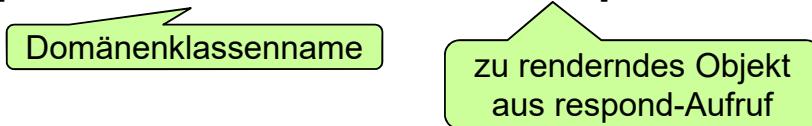


- **domänenInstanz**: Domäneninstanz
- model: erlaubt Übergabe weiterer Parameter an GSP

- **respond** muss am **Ende** der Controller-Aktion stehen

- Falls Grails Content-Typ *html, form* oder *multipartForm*:

- [domänenklassenname: **domänenInstanz**] ins Model hinzugefügt



- aufzurufende View: Name der Controller-Aktion + *.gsp*
- View wird mit diesem erweiterten Model aufgerufen

## Content Negotiation: respond Domäneninstanz: Beispiel

- Controller-Aktion

```
def create() {
 respond new Professor(params)
}
```
- Grails Content-Typ: **html**
  - URL: *http://localhost:8080/professor/create*
  - Model: *[professor: new Professor(params)]*
  - View: *create.gsp*
- Grails Content-Typ: **xml**
  - URL: *http://localhost:8080/professor/create.xml*
  - Model: *new Professor(params) as XML*
  - View: -
- Grails Content-Typ: **json**
  - URL: *http://localhost:8080/professor/create.json*
  - Model: *new Professor(params) as JSON*
  - View: -

## Content Negotiation: respond Domäneninstanz-Liste

- Automatische Content Negotiation in Grails per *respond*-Methode:  
**respond domänenInstanzListe, model:[ wohnort: "Mannheim"]**
  - *domänenInstanzListe*: Objekt mit Domäneninstanz-Liste
  - *model*: erlaubt Übernahme weiterer Parameter für GSP
- **respond** muss am **Ende** der Controller-Aktion stehen
- Falls Grails Content-Typ *html*, *form* oder *multipartForm*:
  - *[domänenklassennameList: domänenInstanzListe]* ins Model hinzugefügt
    - aufzurufende View: Name der Controller-Aktion + *.gsp*
    - View mit diesem erweiterten Model aufrufen

## Content Negotiation: respond Domäneninstanz-Liste: Beispiel

- Controller-Aktion

```
def index(Integer max) {
 params.max = Math.min(max ?: 10, 100)
 respond professorService.list(params),
 model:[professorCount: professorService.count()]
}
```

- Grails Content-Typ: **html**

- URL: *http://localhost:8080/professor/index*
- Model: *[professorList: professorService.list(params), professorCount: professorService.count()]*
- View: *index.gsp*

- Grails Content-Typ: **xml**

- URL: *http://localhost:8080/professor/index.xml*
- Model: *professorList: professorService.list(params) as XML*

- Grails Content-Typ: **json**

- URL: *http://localhost:8080/professor/index.json*
- Model: *professorList: professorService.list(params) as JSON*

## Zugriff auf Model durch die GSP: Beispiel

- Aktions-Methode endet mit respond-Aufruf:

```
def show(Professor professor) {
 respond professor, model:[wohnort: "Mannheim"]
}
```

The code shows a `def show(Professor professor) {` block. Inside, there's a `respond professor, model:[wohnort: "Mannheim"]` statement. Three callout boxes with arrows point to specific parts of this statement:

- A green box labeled "zu renderndes Objekt" points to the variable `professor`.
- A green box labeled "Variablenname" points to the key `wohnort` in the `model` map.
- A green box labeled "Variablenwert" points to the value `"Mannheim"`.

- Zugriff auf Datenobjekt in der GSP

- mit  **`${professor}`**
- Variablenname feststehend: `<nameDerDomänenKlasse>`
- z.B.:  `${professor.vorname} ${professor.nachname} aus ${wohnort}`

## Zugriff auf Model durch die GSP: Beispiel 2

- Aktions-Methode endet mit respond-Aufruf:

```
def index(Integer max) {
 params.max = Math.min(max ?: 10, 100)
 respond professorService.list(params),
 model:[professorCount: professorService.count()]
}
```

zu rendernde Objekte   Variablenname   Variablenwert  
model:[professorCount: professorService.count()]

- Zugriff auf Datenobjekte in der GSP

- mit **`\${professorList}`**
- Variablenname feststehend: **<nameDerDomänenKlasse>List**
- Beispiel:  
`<g:each in="${professorList}" status="i" var="professor">  
 ...  
</g:each>`

- Zugriff auf weitere Model-Variable

- mit **`\${professorCount}`**

## Fallunterscheidung durch Content Negotiation

- request.withFormat** ermöglicht Fallunterscheidung nach Grails Content-Format

- Syntax:

```
request.withFormat {
 Format(e) Closure zur Behandlung
 form multipartForm {
 flash.message = message(
 code: 'default.created.message',
 args: [message(code: 'professor.label', default: 'Professor'), professor.id])
 redirect professor
 }
 Format: Wildcard Closure zur Behandlung
 '*' { respond professor, [status: CREATED] }
}
```

- **Flash-Fehlermeldungen**
  - nur bei Browser-Clients im Klartext zur Einbettung in GSP mitliefern
  - alle anderen Clients bekommen HTTP-Statusfeld auf Konstante gesetzt
- Anpassung des **Ausgabeformats** an Endgeräte
  - in der Regel automatisch über *respond-Aufruf*
- endgerätespezifische **Eingabeparameter**
  - Endgeräte können Eingabedaten z.B. im JSON- oder XML-Format liefern

## Umleitung auf andere Aktion

### Umleitung auf andere Aktion

- des eigenen Controllers

  
`redirect(action:show,id:professor.id)`

- eines anderen Controllers

  
`redirect(controller: "Vorlesung", action:show, id:professor.id)`

- Controllername in "Anführungszeichen" (keine Property der Controllerklasse)
- Endung *Controller* beim Controllernamen nicht mit angeben

- mit Parametern

  
`redirect(controller: "Vorlesung", action:show, params:[hochschuleId:params.id])`

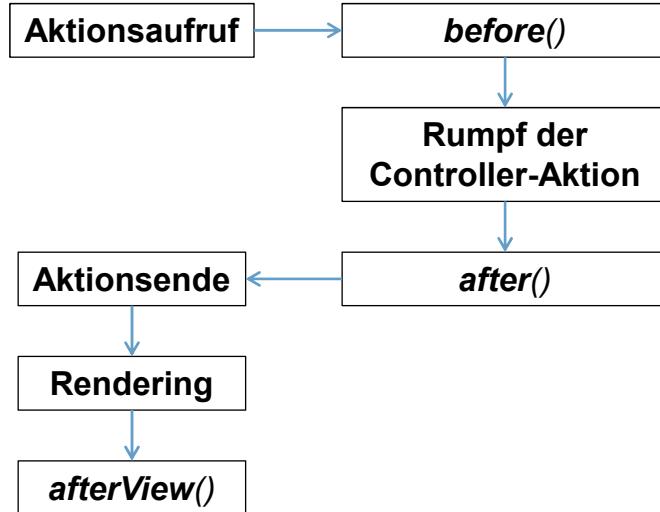
- auf eine URI

  
`redirect(uri: "/")`

- umgeleitet wird immer auf Controller-Action oder URI, niemals direkt auf View!

## Interzeptoren

- unterbrechen Ausführung von Controller-Aktionen
  - *before* Ausführung direkt zu Beginn der Aktion
  - *after* Ausführung direkt nach Abschluss der Aktion
  - *afterView* Ausführung nach Rendern, vor Layouten



## Interzeptor erstellen

- Interzeptor erstellen:  
***create-interceptor Domänenklasse***
  - Beispiel:  
***create-interceptor professorenverwaltung.Professor***
- erzeugter Interzeptor
  - trägt Dateiname *DomänenklassennameInterceptor.groovy*
  - gilt für **alle** Aktionen des Controllers

## Interzeptor: Erzeugtes Grundgerüst

```
package professorenverwaltung

class ProfessorInterceptor {

 boolean before() {
 true
 }

 boolean after() {
 true
 }

 void afterView() {
 // no-op
 }
}
```

## Interzeptor: before

- Signatur: **boolean before()**
- wird **vor** jeder Controller-Action aufgerufen
- selbe Umgebungsvariablen wie im Controller zur Verfügung
  - z.B. *actionName*, *controllerName*, *params*, *session*, *log* ...
- Rückgabewert entscheidet über Fortsetzung der Controller-Action
  - *true*: Controller-Action wird ausgeführt
  - *false*: Controller-Action wird abgebrochen
- Anwendungsbeispiele:
  - Überprüfen der eingegebenen Parameter
  - Login-Handling

## Interzeptor: after

- Signatur: **boolean after()**
- wird **nach** jeder Controller-Action aufgerufen
- selbe Umgebungsvariablen wie im Controller zur Verfügung
  - z.B. *actionName*, *controllerName*, *params*, *session*, *log* ...
- kann anderen **View** zum Rendern auswählen:
  - *view = 'alternativview'* // ohne Endung .gsp
- kann **Model** verändern
  - *model.key=wert* // z.B.: *model.anzahlDatensaetze=20*
- Rückgabewert entscheidet, ob View gerendert wird
  - true: View wird gerendert
  - false: View wird **nicht** gerendert
- Anwendungsbeispiel:
  - Überprüfung und Manipulation der Antwort der Aktion

## Interzeptor: afterView

- Signatur: **void afterView()**
- wird **nach** dem Rendern des Views aufgerufen, vor dem Layouten
- kein Rückgabewert

## Unbedingter Interzeptor

- **kein** Konstruktor im Interzeptor definiert
- Interzeptor muss Domänenklassenname mit Endung *Interceptor* tragen
  - Beispiel:
    - Domänenklasse *Professor*
    - Controllername *ProfessorController*
    - Interzeptorname *ProfessorInterceptor*
- Interzeptor gilt für **alle** Aktionen dieses Controllers

## Bedingter Interzeptor

- **selbstdefinierter parameterloser** Konstruktor im Interzeptor
  - ermöglicht **bedingte Interzeptoren**
    - für einen oder mehrere **beliebige** Controller
    - für eine oder mehrere Controller-Aktionen
  - Konfiguration durch einen oder mehrere match-Aufrufe im Konstruktor
    - Parameter:
      - *controller* Controllername als String oder regulärer Ausdruck
      - *action* Action-Name als String oder regulärer Ausdruck
      - *method* Http-Methode als String oder regulärer Ausdruck
      - *uri* Controller und Action als ant-Pfad '*controller*/*action*'  
Wildcards: ? (0 oder 1 Zeichen) und \* (0, 1 oder mehr Zeichen)
    - Beispiele
      - **match controller: 'professor', action: ~/(edit|show)/**
      - **match controller: ~/.\*/**, action: ~/(edit|show)/
      - **match uri='/professor/\*'**
  - Interzeptorname beliebig mit Endung *Interceptor*

## Interzeptor: Beispiel (1/2)

```
package professorenverwaltung

class AuthenticationInterceptor {
 AuthenticationInterceptor() {
 // Alle Controller + Actions außer login
 match controller:~/(L)login)?+.+/, action:~/(login)?+.+/
 }

 boolean before() {
 println "Controllername: ${controllerName}"
 println "Action-Name: ${actionName}"
 println "Before-Interceptor"
 if (!session.user) {
 session.fromController = controllerName
 session.fromAction = actionName
 session.fromParams = this.params
 redirect action: 'login'
 }
 true
 }
}
```

## Interzeptor: Beispiel (2/2)

```
boolean after() {
 println "After-Interceptor"
 true
}

void afterView() {
 // no-op
}
```



## Gliederung

3. Grails-Framework
  1. Einführung in Grails
  - 2. Controller**
    1. Dynamisches Scaffolding
    2. Statisches Scaffolding
  - 3. Manuelle Programmierung**
  3. Domänenklassen
  4. RESTful Web Services
  5. GSP

## Manuelle Programmierung eines Controllers

- Controller nicht zwangsläufig einer Domänenklasse zugeordnet
  - kann auch Controller ohne Bezug zur Datenbank sein
  - kann auch mehrere Domänenklassen repräsentieren
  - volle Flexibilität
  - alles muss manuell programmiert werden
- Grails-Kommando: *create-controller Klasse*
  - generiert Controllerklasse als Quellcodegerüst:

```
class ProfessorController {
 def index = {}
}
```
  - alle weiteren Aktionen müssen manuell als Methoden erstellt werden

## Manuelle Programmierung der zugehörigen Views

- Views müssen manuell erstellt werden
  - Einstiegsseite **immer** `views/index.html`
    - nicht mit Interzeptoren abfangen
    - nicht umleiten!
  - jede GSP muss `<meta name="layout" content="main">` im Head haben!
    - Beispiel:  
`<head>`  
`<meta name="layout" content="main">`  
`<title>Login</title>`  
`</head>`
    - sonst funktioniert Layouten **nicht!**
- Anwendung eher für GUI-Steuerung

## Vorgehensweise am Beispiel eines Login-Controllers

1. Interzeptor für alle Controller außer Login-Controller definieren
2. `LoginController` erstellen:  
`create-controller Login`
3. Controller-Aktionen implementieren
4. Login-View erstellen

## Interzeptor (1/2)

```
package professorenverwaltung

class AuthenticationInterceptor {
 AuthenticationInterceptor () {
 // Alle Controller und Actions außer login
 match controller:~/(L)login)?+.+/, action:~/(login)?+.+/
 }

 boolean before() {
 println "Controllername: ${controllerName}"
 println "Action-Name: ${actionName}"
 println "Before-Interceptor"
 if (!session.user) {
 session.fromController = controllerName
 session.fromAction = actionName
 session.fromParams = this.params
 redirect controller:'Login', action: 'login'
 }
 true
 }
}
```

## Interzeptor (2/2)

```
boolean after() {
 println "After-Interceptor"
 true
}

void afterView() {
 // no-op
}
```

## Login-Controller (1/2)

```
package professorenverwaltung

class LoginController {
 // Nur eine Controller-Aktion definiert → ist Default Controller-Aktion
 def login(String user, String password) {
 if (user == "specht" && password == "specht") { // Login korrekt
 // Eingeloggten Benutzer merken
 session.user = user
 session.password = password
 // Navigationsherkunft abrufen und in Session-Variablen löschen
 def fromController = session.fromController
 def fromAction = session.fromAction
 def fromParams = session.fromParams
 session.fromController = null
 session.fromAction = null
 session.fromParams = null
 }
 }
}
```

## Login-Controller (2/2)

```
if (fromController && fromAction && fromParams) {
 redirect controller:fromController, action:fromAction, params:fromParams
} else if (fromController && fromAction) {
 redirect controller:fromController, action:fromAction
} else if (fromController) {
 redirect controller:fromController
} else { // Über Controllerauswahlseite eingestiegen
 redirect uri: ""
}
} else {
 if (user || password) { // Falscher User / Passwort bereits eingetragen
 flash.message = "Login oder Passwort falsch!"
 }
 // login.gsp nochmal mit ggf. vorhandenen Eingaben rendern
 render view: 'login', model:[user:user, password:password]
}
}
```

## Login-GSP (1/2)

```
<!DOCTYPE html>
<html>
<head>
 <meta name="layout" content="main"> Grails-Layout auswählen
 <title>Login</title>
 </head>

<body>
 <div class="nav" > Obere Navigationsleiste mit Home-Button

 <g:message code="default.home.label"/>

 </div>
 <div id="list-professor" class="content" > Content-Bereich
 <g:if test="${flash.message}">
 <div class="message" role="status">${flash.message}</div>
 </g:if>
 </div>

```

## Login-GSP (2/2)

```
<g:form controller="login" action="login" method="post">
 <table> Grails-Formular, das eigene Controller-Aktion aufruft
 <tr>
 <td>Login:</td>
 <td><g:textField name="user" value="${user}"></g:textField></td>
 </tr>
 <tr>
 <td>Passwort:</td>
 <td><g:passwordField name="password" value="${password}"></g:passwordField></td>
 </tr>
 </table> Untere Navigationsleiste mit Login-Button
 <fieldset class="buttons">
 <g:submitButton name="Login" value="Login" />
 </fieldset>

 </g:form>
</div>
</body>
</html>
```



## Gliederung

3. Grails-Framework
  1. Einführung in Grails
  2. Controller
  - 3. Domänenklassen**
  4. RESTful Web Services
  5. GSP

## Domänenklasse erstellen

### Domänenklassen

- liegen grundsätzlich im Ordner *Domain Classes*
  - mit *Neu* → *Grails Domain Class* erstellen
- sind ganz normale Groovy-Klassen mit
  - Attributen, die automatisch auf eine Datenbank abgebildet werden
  - *static-Variable constraints* zur Konfiguration von Integritätsregeln für die Attribute
    - **nullable** Attribut darf auch *null* sein **true** oder **false**
    - **blank** Attribut (in der Regel String) darf leer sein **true** oder **false**
    - **maxSize** Maximale Anzahl Zeichen für einen String Ganzzahl
  - Methoden, z.B. überschriebene *toString()*-Methode
    - für Auswahllisten und Zuordnungen gebraucht
- werden automatisch persistiert
  - technische Basis:
    - GORM (Grails Object Relational Mapping)
    - basiert auf Hibernate

## Domänenklasse Professor

```
class Professor {
 String vorname
 String nachname
 String kuerzel
 String raum
 String telefondurchwahl
 String bemerkungen

 static constraints = {
 vorname(blank:false) <-- Attribut darf nicht leer sein
 nachname(blank:false)
 kuerzel(blank:true, nullable:true) <-- Attribut darf leer sein
 raum(blank:true, nullable:true)
 telefondurchwahl(blank:true, nullable:true)
 bemerkungen(blank:true, nullable:true, maxSize:2000, widget:'textarea')
 }
 String toString() {
 return "$nachname, $vorname"
 }
}
```

Annotations:

- Attribut: vorname, nachname, kuerzel, raum, telefondurchwahl, bemerkungen
- Constraint: vorname(blank:false), nachname(blank:false), kuerzel(blank:true, nullable:true), raum(blank:true, nullable:true), telefondurchwahl(blank:true, nullable:true), bemerkungen(blank:true, nullable:true, maxSize:2000, widget:'textarea')
- Max. 2000 Zeichen: maxSize:2000
- mehrzeiliges Textfeld: widget:'textarea'
- Überschriebene Methode: toString()

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

335

## Beziehungen in Domänenklassen: 1:n → hasMany

- über spezielle **static**-Attribute (→ Convention over Configuration)
- 1:n** - Beziehungen
  - zu beliebig vielen anderen Domänenklassen
  - Definition über assoziatives Array **hasMany**:  
**static hasMany=[attrname : DomainClass]**
- Beispiel:

```
class Hochschule {
 String name
 String ort
 static hasMany=[fakultaeten:Fakultaet]
}

class Fakultaet {
 String name
 String kuerzel
 static belongsTo=[hochschule:Hochschule]
}
```

Annotations:

- Attributname: name, ort
- Domainklasse am :n-Ende: fakultaeten, Fakultaet

Relationship diagram:

```
graph LR; Hochschule -- 1 --> Fakultaet -- n --> Fakultaet
```

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

336

## Beziehungen in Domänenklassen: n:1 → belongsTo

- **n:1** - Beziehungen

- zu beliebig vielen anderen Domänenklassen

- Definition über assoziatives Array **belongsTo**:  
**static belongsTo =[attrname : DomainClass]**

Attributname

Domänenklasse am :1-Ende

- Defaults:

- Löschen des besitzenden Datensatzes löscht auch abhängige Datensätze (referenzielle Abhängigkeit)
    - Lazy Loading

- Beispiel:

```
class Fakultaet { class Hochschule {
 String name String name
 String kuerzel String ort
 static belongsTo=[hochschule:Hochschule] static hasMany=[fakultaeten:Fakultaet]
}
} }
Fakultaet n --1-- Hochschule
```

## Vollständige Domänenklasse Hochschule

```
package fakultaeten

class Hochschule {
 String name
 String ort
 static hasMany=[fakultaeten:Fakultaet] // 1:n-Beziehung

 static constraints = {
 name nullable: false, blank: false
 ort nullable:false, blank:false
 fakultaeten nullable: true // Kann-Beziehung
 }

 String toString() { // Benötigt für Auswahlliste der Hochschule in Fakultäts-GSP
 "$name $ort"
 }
}
```

## Datenbanktabelle Hochschule

	Feld	Typ	Kollation	Attribute	Null	Standard	Extra	
<input type="checkbox"/>	<b>id</b>	bigint(20)			Nein	Kein	AUTO_INCREMENT	 
<input type="checkbox"/>	<b>version</b>	bigint(20)			Nein	Kein		 
<input type="checkbox"/>	<b>name</b>	varchar(255)	utf8_bin		Nein	Kein		 
<input type="checkbox"/>	<b>ort</b>	varchar(255)	utf8_bin		Nein	Kein		 
<input type="checkbox"/> Alle auswählen / Auswahl entfernen markierte:       								

 Druckansicht  Tabellenstruktur analysieren   
 1 Felder hinzufügen  An das Ende der Tabelle  An den Anfang der Tabelle  Nach **id** 

### Indizes:

Aktion	Name	Typ	Unique	Gepackt	Feld	Kardinalität	Kollation	Null	Kommentar
 	PRIMARY	BTREE	Ja	Nein	id	2	A		

## Vollständige Domänenklasse Fakultaet

```

package fakultaeten

class Fakultaet {
 String name
 String kuerzel
 static belongsTo=[hochschule:Hochschule] // n:1-Beziehung

 static constraints = {
 name nullable:false, blank:false
 kuerzel nullable:false, blank: false
 hochschule nullable:false // Muss-Beziehung
 }

 String toString() {
 name
 }
}

```

## Datenbanktabelle Fakultaet

	Feld	Typ	Kollation	Attribute	Null	Standard	Extra	Aktion			
<input type="checkbox"/>	<b>id</b>	bigint(20)			Nein	Kein	AUTO_INCREMENT				
<input type="checkbox"/>	<b>version</b>	bigint(20)			Nein	Kein					
<input type="checkbox"/>	<b>hochschule_id</b>	bigint(20)			Nein	Kein					
<input type="checkbox"/>	<b>kuerzel</b>	varchar(255)	utf8_bin		Nein	Kein					
<input type="checkbox"/>	<b>name</b>	varchar(255)	utf8_bin		Nein	Kein					

Alle auswählen / Auswahl entfernen markierte:

Druckansicht Tabellenstruktur analysieren   
 1 Felder hinzufügen  An das Ende der Tabelle  An den Anfang der Tabelle  Nach id

### Indizes:

Aktion	Name	Typ	Unique	Gepackt	Feld	Kardinalität	Kollation	Null
	<b>PRIMARY</b>	BTREE	Ja	Nein	id	5	A	
	<b>FK_cpcw33cyi1hwwhbfd6usto94j</b>	BTREE	Nein	Nein	hochschule_id	0	A	

## Tabellenstruktur in GORM

### • Attribute

- *id* Künstlicher Primärschlüssel, automatisch generiert
- *version* Versionsnummer des Datensatzes für optimistisches Sperren
- *xxx\_id* Fremdschlüsselattribute nach Bedarf
- ... Attribute der Domänenklasse, alphabetisch(!) sortiert

### • Constraints

- *PRIMARY* Attribut *id* eindeutiger (Primär-)schlüssel
- *FK\_xxx* Index für jeden Fremdschlüssel

### • Primärschlüsselerzeugung

- abhängig von zugrunde liegender Datenbank
- in MySql: *id*-Attribut mit Extra *AUTO\_INCREMENT*

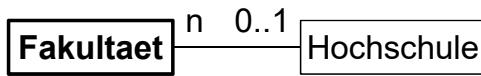
## Beziehungen in Domänenklassen: n:0..1 → belongsTo

- **n:0..1 - Beziehungen**
  - zu beliebig vielen anderen Domänenklassen
  - Definition über assoziatives Array **belongsTo**:  
**static belongsTo =[attrname : DomainClass]**
  - Referenzattribut zusätzlich **nullable:true**
  - Defaults
    - Löschen des Datensatzes, zu dem wir gehören, löscht auch uns selbst  
(referenzielle Abhängigkeit)
    - Lazy Loading

- Beispiel:

```
class Fakultaet {
 String name
 String kuerzel
 static belongsTo=[hochschule:Hochschule]
 static constraints = {
 hochschule(nullable:true)
 }
}
```

```
class Hochschule {
 String name
 String ort
 static hasMany=[fakultaeten:Fakultaet]
}
```



## Datenbanktabelle Fakultaet

	Feld	Typ	Kollation	Attribute	Null	Standard	Extra	Aktion
<input type="checkbox"/>	<b>id</b>	bigint(20)			Nein	Kein	AUTO_INCREMENT	
<input type="checkbox"/>	<b>version</b>	bigint(20)			Nein	Kein		
<input type="checkbox"/>	<b>hochschule_id</b>	bigint(20)			Ja	NULL		
<input type="checkbox"/>	<b>kuerzel</b>	varchar(255)	utf8_bin		Nein	Kein		
<input type="checkbox"/>	<b>name</b>	varchar(255)	utf8_bin		Nein	Kein		

Alle auswählen / Auswahl entfernen markierte:

Fremdschlüssel darf Null sein

Druckansicht Tabellenstruktur analysieren

1 Felder hinzufügen  An das Ende der Tabelle  An den Anfang der Tabelle  Nach id OK

### Indizes:

Aktion	Name	Typ	Unique	Gepackt	Feld	Kardinalität	Kollation	Null
	<b>PRIMARY</b>	BTREE	Ja	Nein	id	0	A	
	<b>FK_cpcw33cyi1hwwhbfd6usto94j</b>	BTREE	Nein	Nein	hochschule_id	0	A	YES

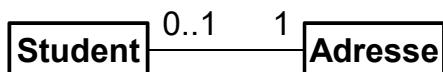
Fremdschlüssel darf Null sein

## Beziehungen in Domänenklassen: 0..1:1

- **1:0..1 - Beziehungen**
  - Attribut vom Typ der referenzierten Klasse als Instanzvariable
    - auf beiden Seiten
    - auf mindestens einer Seite **nullable:true** (→Henne - Ei - Problem)
- Beispiel:

```
class Student {
 String vorname
 String nachname
 Adresse adresse
 static constraints = {
 adresse(nullable:false)
 }
}

class Adresse {
 String strasse
 String hausnummer
 String plz
 String ort
 Student student
 static constraints = {
 student(nullable:true)
 }
}
```



## Vollständige Domänenklasse Student

```
package student

class Student {
 String vorname
 String nachname
 Adresse adresse
 static constraints = {
 vorname nullable:false, blank: false
 nachname nullable: false, blank: false
 adresse nullable:false // Muss-Beziehung
 }

 String toString() {
 "$vorname $nachname"
 }
}
```

## Datenbanktabelle Student

	Feld	Typ	Kollation	Attribute	Null	Standard	Extra	Aktion	
<input type="checkbox"/>	<b>id</b>	bigint(20)			Nein	Kein	AUTO_INCREMENT		
<input type="checkbox"/>	<b>version</b>	bigint(20)			Nein	Kein			
<input type="checkbox"/>	<b>adresse_id</b>	bigint(20)			Ja	<u>NULL</u>			
<input type="checkbox"/>	<b>nachname</b>	varchar(255)	utf8_bin		Nein	Kein			
<input type="checkbox"/>	<b>vorname</b>	varchar(255)	utf8_bin		Nein	Kein			

Alle auswählen / Auswahl entfernen markierte:

Seltsamerweise:  
Fremdschlüssel darf Null sein

Druckansicht Tabellenstruktur analysieren

1 Felder hinzufügen  An das Ende der Tabelle  An den Anfang der Tabelle  Nach

### Indizes:

Aktion	Name	Typ	Unique	Gepackt	Feld	Kardinalität	Kollation	Null
	PRIMARY	BTREE	Ja	Nein	id	0	A	
	FK_j4usl0f38nx35suuip1vqih6d	BTREE	Nein	Nein	adresse_id	0	A	YES

- Controller und View verhindern jedoch Anlegen eines Studenten ohne Adresse

Seltsamerweise:  
Fremdschlüssel darf Null sein

## Vollständige Domänenklasse Adresse

```
package student

class Adresse {
 String strasse
 String hausnummer
 String plz
 String ort
 Student student // 1:0..1-Beziehung

 static constraints= {
 strasse nullable: true, blank: true
 hausnummer nullable: true, blank: true
 plz nullable: true, blank: true
 ort nullable: false, blank: false
 student nullable: true // Kann-Beziehung
 }

 String toString() {
 "$strasse $hausnummer, $plz $ort"
 }
}
```

## Datenbanktabelle Adresse

	Feld	Typ	Kollation	Attribute	Null	Standard	Extra	Aktion
<input type="checkbox"/>	<b>id</b>	bigint(20)			Nein	Kein	AUTO_INCREMENT	
<input type="checkbox"/>	<b>version</b>	bigint(20)			Nein	Kein		
<input type="checkbox"/>	<b>hausnummer</b>	varchar(255)	utf8_bin		Ja	NULL		
<input type="checkbox"/>	<b>ort</b>	varchar(255)	utf8_bin		Nein	Kein		
<input type="checkbox"/>	<b>plz</b>	varchar(255)	utf8_bin		Ja	NULL		
<input type="checkbox"/>	<b>strasse</b>	varchar(255)	utf8_bin		Ja	NULL		
<input type="checkbox"/>	<b>student_id</b>	bigint(20)			Ja	NULL		

Alle auswählen / Auswahl entfernen markierte:

Druckansicht Tabellenstruktur analysieren   
 1  Felder hinzufügen  An das Ende der Tabelle  An den Anfang der Tabelle  Nach **id**

### Indizes:

Aktion	Name	Typ	Unique	Gepackt	Feld	Kardinalität	Kollation	Null
	<b>PRIMARY</b>	BTREE	Ja	Nein	<b>id</b>	0	A	
	<b>FK_iwj6vm0e5a12idcpuq47g1132</b>	BTREE	Nein	Nein	<b>student_id</b>	0	A	YES

## Einbetten von Objekten

- Bessere Lösung: Attribut vom Typ *Adresse* in Klasse *Student* einbetten
  - Datentyp *Adresse* im Ordner *src/main/groovy* (ist **keine** Domänenklasse)
  - vermeidet
    - Henne-Ei-Problem
    - unnötige Datenbank-Joins beim Abruf der Daten
    - eigenen Controller und eigene GSPs für Adresse
  - Einbetten über Liste *embedded*:  
**static embedded=['attrname']**
- Beispiel:

```
class Student {
 String vorname
 String nachname
 Adresse adresse

 static constraints = {
 adresse(nullable:true)
 }
 static embedded = ["adresse"]
}
```

```
class Adresse {
 String strasse
 String hausnummer
 String plz
 String ort
}
```

## Vollständige Domänenklasse Student

```
package student
import datatypes.Adresse

class Student {
 String vorname
 String nachname
 Adresse adresse // 1:0..1-Beziehung

 static constraints = {
 vorname nullable:false, blank: false
 nachname nullable: false, blank: false
 adresse nullable: true
 }

 static embedded = ['adresse']

 String toString() {
 "$vorname $nachname"
 }
}
```

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

351

## Vollständiger Datentyp Adresse

```
package datatypes

class Adresse {
 String strasse
 String hausnummer
 String plz
 String ort

 static constraints = {
 strasse nullable: true
 hausnummer nullable: true
 plz nullable: true
 ort nullable: true
 }
}
```

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

352

## Datenbanktabelle Student mit eingebetteter Adresse

	Feld	Typ	Kollation	Attribute	Null	Standard	Extra
<input type="checkbox"/>	<b>id</b>	bigint(20)			Nein	Kein	AUTO_INCREMENT
<input type="checkbox"/>	<b>version</b>	bigint(20)			Nein	Kein	
<input type="checkbox"/>	<b>adresse_hausnummer</b>	varchar(255)	utf8_bin		Ja	NULL	
<input type="checkbox"/>	<b>adresse_ort</b>	varchar(255)	utf8_bin		Ja	NULL	
<input type="checkbox"/>	<b>adresse_plz</b>	varchar(255)	utf8_bin		Ja	NULL	
<input type="checkbox"/>	<b>adresse_strasse</b>	varchar(255)	utf8_bin		Ja	NULL	
<input type="checkbox"/>	<b>nachname</b>	varchar(255)	utf8_bin		Nein	Kein	
<input type="checkbox"/>	<b>vorname</b>	varchar(255)	utf8_bin		Nein	Kein	

Alle auswählen / Auswahl entfernen markierte:

Druckansicht Tabellenstruktur analysieren 1 Felder hinzufügen  An das Ende der Tabelle  An den Anfang der Tabelle  Nach id

Indizes:

Aktion	Name	Typ	Unique	Gepackt	Feld	Kardinalität	Kollation	Null	Kommentar
	PRIMARY	BTREE	Ja	Nein	id	0	A		

## Einbetten von Objekten: Anmerkungen

- Views `create.gsp` + `edit.gsp` in Grails 3.0 **fehlerhaft**, erzeugen leeren Bildschirm
  - vermutlich Bug im Grails `fields` plugin
  - Korrektur: in `create.gsp` und `edit.gsp` jeweils
 

```
<f:all bean="student"/>
```

 ersetzen durch die einzelnen Attribute:
 

```
<f:field bean="student" property="vorname"/>
<f:field bean="student" property="nachname"/>
<f:field bean="student" property="adresse"/>
```
- Eingebettete Klasse `Adresse` wahlweise
  - als **eigene Domänenklasse**
    - bekommt (überflüssige) Datenbanktabelle
  - als **weitere Klasse** in Quellcodedatei der umgebenden Klasse `Student`
    - Groovy erlaubt Definition mehrerer Klassen in einer Quellcodedatei
    - keine eigene Datenbanktabelle
  - als **normale Groovy-Klasse** im Ordner `src/main/groovy`
    - keine eigene Datenbanktabelle
    - beste Lösung

## Beziehungen in Domänenklassen: m:n → hasMany

- **m:n** - Beziehungen

- zu beliebig vielen anderen Domänenklassen

- Definition über assoziatives Array **hasMany** von beiden Seiten her:

**static hasMany=[attrname : DomainClass]**

Attributname

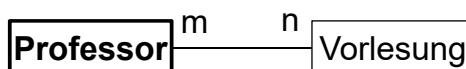
Domänenklasse am anderen Ende

- Auf **genau einer** der beiden Seiten zusätzliches **belongsTo**-Attribut notwendig

- Beispiel:

```
class Professor {
 String vorname
 String nachname
 static hasMany = [vorlesung:Vorlesung]
}

class Vorlesung {
 String name
 String kuerzel
 static hasMany = [professor:Professor]
 static belongsTo= Professor
}
```



## Vollständige Domänenklasse Professor

```
package mnbeziehung

class Professor {

 String vorname
 String nachname
 static hasMany = [vorlesung:Vorlesung]

 static constraints = {
 vorname nullable: false, blank: false
 nachname nullable: false, blank: false
 }

 String toString() {
 "$vorname $nachname"
 }
}
```

## Datenbanktabelle Professor

	Feld	Typ	Kollation	Attribute	Null	Standard	Extra
<input type="checkbox"/>	<b>id</b>	bigint(20)			Nein	Kein	AUTO_INCREMENT
<input type="checkbox"/>	<b>version</b>	bigint(20)			Nein	Kein	
<input type="checkbox"/>	<b>nachname</b>	varchar(255)	utf8_bin		Nein	Kein	
<input type="checkbox"/>	<b>vorname</b>	varchar(255)	utf8_bin		Nein	Kein	



Alle auswählen / Auswahl entfernen markierte:



Druckansicht Tabellenstruktur analysieren



1 Felder hinzufügen An das Ende der Tabelle An den Anfang der Tabelle Nach **id**

### Indizes:

Aktion	Name	Typ	Unique	Gepackt	Feld	Kardinalität	Kollation	Null	Kon
	<b>PRIMARY</b>	BTREE	Ja	Nein	<b>id</b>	0	A		

## Vollständige Domänenklasse Vorlesung

```
package mnbeziehung

class Vorlesung {

 String name
 String kuerzel
 statichasMany = [professor:Professor]
 static belongsTo= Professor

 static constraints = {
 name nullable: false, blank: false
 kuerzel nullable: false, blank: false
 professor null:true
 }

 String toString() {
 name
 }
}
```

## Datenbanktabelle Vorlesung

	Feld	Typ	Kollation	Attribute	Null	Standard	Extra
<input type="checkbox"/>	<b>id</b>	bigint(20)			Nein	Kein	AUTO_INCREMENT
<input type="checkbox"/>	<b>version</b>	bigint(20)			Nein	Kein	
<input type="checkbox"/>	<b>kuerzel</b>	varchar(255)	utf8_bin		Nein	Kein	
<input type="checkbox"/>	<b>name</b>	varchar(255)	utf8_bin		Nein	Kein	



Alle auswählen / Auswahl entfernen markierte:



Druckansicht



Tabellenstruktur analysieren



1

Felder hinzufügen



An das Ende der Tabelle



An den Anfang der Tabelle



Nach



Nach

### Indizes:

Aktion	Name	Typ	Unique	Gepackt	Feld	Kardinalität	Kollation	Null	K
	PRIMARY	BTREE	Ja	Nein	id	0	A		

## Beziehungstabelle Professor\_Vorlesung

	Feld	Typ	Kollation	Attribute	Null	Standard	Extra	Aktion					
<input type="checkbox"/>	<b>professor_id</b>	bigint(20)			Nein	Kein		<input type="checkbox"/>					
<input type="checkbox"/>	<b>vorlesung_id</b>	bigint(20)			Nein	Kein		<input type="checkbox"/>					
<input type="checkbox"/>								<input type="checkbox"/>					



Alle auswählen / Auswahl entfernen markierte:



### Indizes:

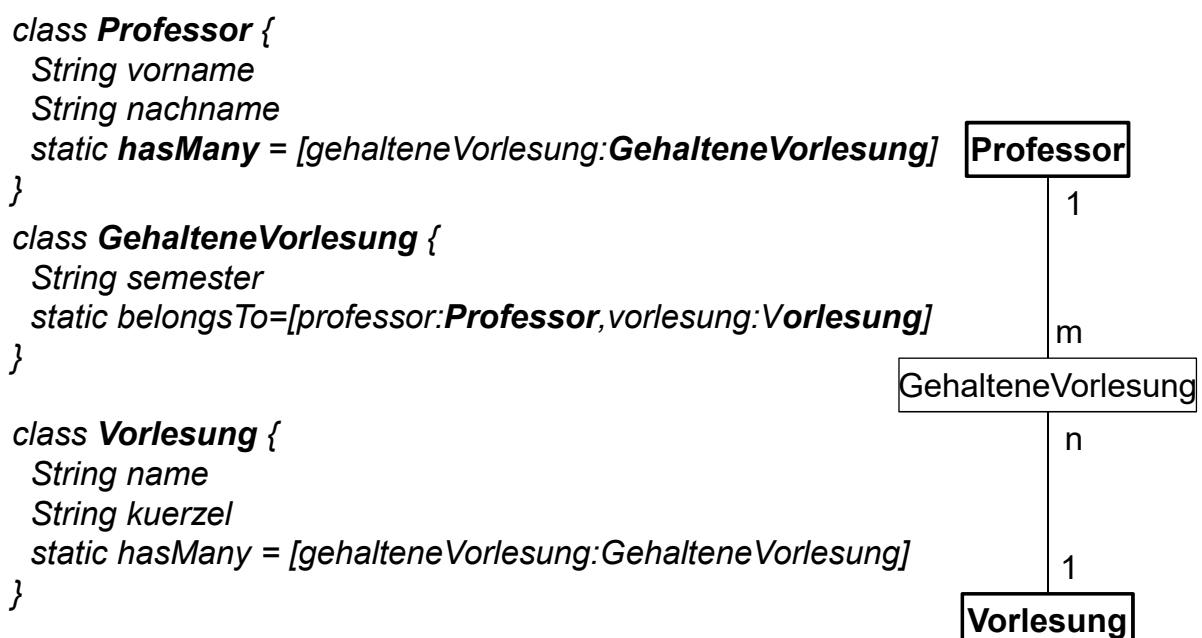
Aktion	Name		Typ	Unique	Gepackt	Feld	Kardinalität	Kollation	Null
	PRIMARY		BTREE	Ja	Nein	professor_id	0	A	
						vorlesung_id	0	A	

## Nachteil direkter m:n-Beziehungen

- **eine** der beiden Seiten muss der Besitzer sein
  - nur dort ist Beziehung in der GSP editierbar
  - andere Seite abhängig mit **belongsTo**, kann Beziehung nicht editieren

## m:n-Beziehung durch Zwischendomäne auflösen

- künstliche Domänenklasse zur Auflösung der m:n-Beziehung
  - hat n:1-Beziehung zur beiden Enden der m:n-Beziehung



## Vollständige Domänenklasse Professor

```
package mnbeziehung

class Professor {

 String vorname
 String nachname
 statichasMany = [gehalteneVorlesung:GehalteneVorlesung]

 static constraints = {
 vorname nullable: false, blank: false
 nachname nullable: false, blank: false
 gehalteneVorlesung nullable: true
 }

 String toString() {
 "$vorname $nachname"
 }
}
```

## Vollständige Domänenklasse Vorlesung

```
package mnbeziehung

class Vorlesung {

 String name
 String kuerzel
 statichasMany = [gehalteneVorlesung:GehalteneVorlesung]

 static constraints = {
 name nullable: false, blank: false
 kuerzel nullable: false, blank: false
 gehalteneVorlesung nullable: true
 }

 String toString() {
 name
 }
}
```

## Vollständige Domänenklasse GehalteneVorlesung

```
package mnbeziehung

class GehalteneVorlesung {
 String semester
 static belongsTo=[professor:Professor,vorlesung:Vorlesung]

 static constraints = {
 professor nullable: false
 vorlesung nullable: false
 semester nullable: true, blank: true
 }

 String toString() {
 "$vorlesung im $semester bei $professor"
 }
}
```

## Beziehungstabelle Gehaltene\_Vorlesung

	Feld	Typ	Kollation	Attribute	Null	Standard	Extra	Aktion
<input type="checkbox"/>	<b>id</b>	bigint(20)			Nein	Kein	AUTO_INCREMENT	
<input type="checkbox"/>	<b>version</b>	bigint(20)			Nein	Kein		
<input type="checkbox"/>	<b>professor_id</b>	bigint(20)			Nein	Kein		
<input type="checkbox"/>	<b>semester</b>	varchar(255)	utf8_bin		Ja	NULL		
<input type="checkbox"/>	<b>vorlesung_id</b>	bigint(20)			Nein	Kein		

Alle auswählen / Auswahl entfernen markierte:

---

Druckansicht Tabellenstruktur analysieren   
 1 Felder hinzufügen An das Ende der Tabelle An den Anfang der Tabelle Nach **id**

---

**Indizes:**

Aktion	Name	Typ	Unique	Gepackt	Feld	Kardinalität	Kollation	Null
	<b>PRIMARY</b>	BTREE	Ja	Nein	id	3	A	
	<b>FK_nhgb31v4canlpb1113ylbei9r</b>	BTREE	Nein	Nein	professor_id	0	A	
	<b>FK_sy3psy7jdb7u5vo1nlent0wdf</b>	BTREE	Nein	Nein	vorlesung_id	0	A	

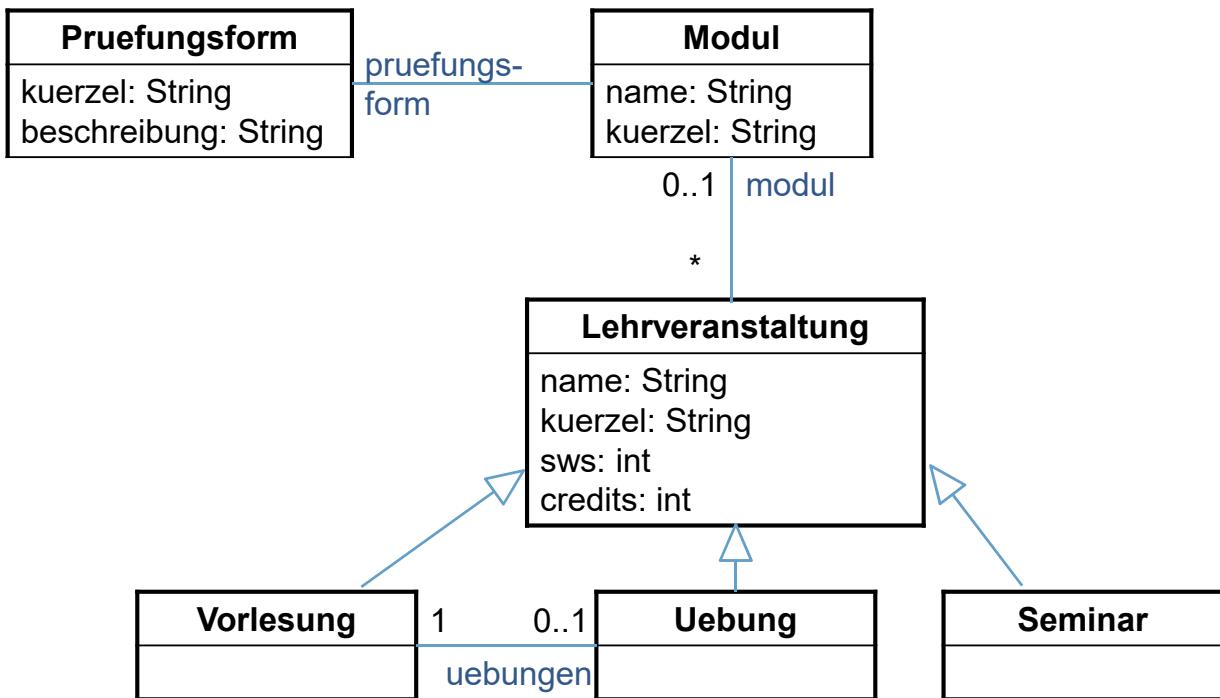
# Vererbung

- Domänenobjekte sind **POGOs** (Plain Old Groovy Objects)
  - Vererbung
    - nicht für OR-Mapping zweckentfremdet
    - für Anwendungszwecke einsetzbar
- GORM beherrscht **Vererbung** von Domänenklassen
  - ganz normale *extends*-Klausel
  - Attribute und Beziehungen werden vererbt
  - Constraints werden vererbt
    - können in abgeleiteter Klasse ergänzt / überschrieben werden
- **Abbildungsvarianten** auf Datenbank
  1. Table per **Hierarchy**: 1 Tabelle pro Vererbungshierarchie (Default)
  2. Table per **Subclass**: 1 Tabelle pro Klasse (Basis + abgeleitete)
  3. Table per **Concrete Class**: je eine Tabelle pro konkreter Klasse, umfasst Attribute der Basisklasse

## Abbildungsvarianten der Vererbung auf die Datenbank

		Table per				
		Hierarchy	Subclass	Concrete Class		
Basisklasse	Tabelle	ja	ja	wenn nicht <i>abstract</i>		
	Attribute	eigene + Sub	eigene	eigene		
Abgeleitete Klasse	Tabelle	-	ja	wenn nicht <i>abstract</i>		
	Attribute		eigene	eigene + Basis		
JOIN-freier Instanzabruf	Basisklasse	ja				
	Abgeleitete Klasse	ja	-	ja		
UNION-freier Abruf aller Instanzen	Basisklasse	ja				
	1 abgeleitete Klasse	ja				
	>1 abgeleitete Klasse	ja	-			
nur <i>nullable</i> Subklassen-Attribute	ja	-				
Unterstützung	Hibernate	ja				
	Grails	ja		-		
Performance	++	-	+			

## Vererbungshierarchie



## Vererbung: Basisklasse Lehrveranstaltung

```
package lehrveranstaltungsverwaltung
class Lehrveranstaltung {
 String name
 String kuerzel
 int sws
 int credits
 static belongsTo=[modul:Modul]
 static constraints = {
 name(blank: false, nullable: false)
 kuerzel(blank: false, nullable: false)
 sws()
 credits()
 modul(nullable: true)
 }
 String toString() {
 kuerzel
 }
}
```

## Vererbung: Abgeleitete Klasse Vorlesung

```
package lehrveranstaltungsverwaltung
class Vorlesung extends Lehrveranstaltung {
 defhasOne=[uebungen:Uebung]
 static constraints = {
 uebungen(nullable: true)
 }
 String toString() {
 kuerzel
 }
}
```

## Vererbung: Abgeleitete Klasse Uebung

```
package lehrveranstaltungsverwaltung
class Uebung extends Lehrveranstaltung {
 static constraints = {
 }
 String toString() {
 kuerzel
 }
}
```

## Vererbung: Abgeleitete Klasse Seminar

```
package lehrveranstaltungsverwaltung
class Seminar extends Lehrveranstaltung {
 static constraints = {
 }
}
```

## Domänenklasse Modul

```
package lehrveranstaltungsverwaltung
class Modul {
 String name
 String kuerzel
 Pruefungsform pruefungsform
 statichasMany=[lehrveranstaltungen:Lehrveranstaltung]
 static constraints = {
 name(nullable:false, blank: false)
 kuerzel(nullable: false, blank: false)
 pruefungsform(nullable: true, blank:true)
 lehrveranstaltungen(nullable: true)
 }

 String toString() {
 kuerzel
 }
}
```

## Domänenklasse Pruefungform

```
package lehrveranstaltungsverwaltung
class Pruefungsform {
 String kuerzel
 String beschreibung
 static constraints = {
 kuerzel(blank:false, nullable: false)
 beschreibung(blank: false, nullable: false)
 }
 String toString() {
 kuerzel
 }
}
```

## Nachbearbeitung der Controller und Views 1/2

- Zwei **Fehler** im Grails 3 Scaffolding bei Domänenklassen Vererbung
  - **index**-Aktion des Basisklassencontrollers
    - Rückgabe mit `respond` funktioniert nicht für abgeleitete Klassen  
`respond Lehrveranstaltung.list(params),  
model:[lehrveranstaltungCount: Lehrveranstaltung.count()]`
    - liefert oft leere Liste statt Instanzen der Basis- und aller abgeleiteten Klassen
    - **ersetzen** durch klassische Model-Rückgabe:  
`[lehrveranstaltungList: Lehrveranstaltung.list(params),  
lehrveranstaltungCount: Lehrveranstaltung.count()]`
    - Content Negotiation entfällt somit, bis der Bug behoben ist

## Nachbearbeitung der Controller und Views 2/2

### – **index.gsp** der Basisklasse

- nimmt Klasse des ersten Datensatzes als Basisklasse
- Lösung: Datensatzaufstellung der *index.gsp*  
`<f:table collection="${lehrveranstaltungList}" />`
- ersetzen durch  
`<f:table collection="${lehrveranstaltungList}"  
domainClass="lehrveranstaltungsverwaltung.Lehrveranstaltung"/>`

## Variante 1: DB-Tabelle für ganze Vererbungshierarchie

- Default-Einstellung
- alle Attribute **aller** abgeleiteten Klassen in Tabelle der Basisklasse
  - Diskriminatorspalte *class* zur Erkennung des Instanztyps
- **Vorteile:**
  - Abruf einer Instanz ohne DB-JOIN
  - Abruf aller Instanzen sämtlicher abgeleiteter Klassen ohne DB-UNION
  - weniger DB-Tabellen
- **Nachteil:**
  - Attribute abgeleiteter Klassen müssen *nullable* sein

Attribute abgeleiteter  
Klassen nullable

Diskriminatorspalte

<b>id</b>	<b>version</b>	<b>credits</b>	<b>kuerzel</b>	<b>modul_id</b>	<b>name</b>	<b>sws</b>	<b>class</b>	<b>uebungen_id</b>
1	1	4	WAW	1	Webanwendungen	4	lehrveranstaltungsverwaltung.Vorlesung	NULL
2	0	4	VAR	1	Verteilte Architekturen	4	lehrveranstaltungsverwaltung.Vorlesung	NULL
3	1	0	GDI	2	Grundlagen der Informatik	0	lehrveranstaltungsverwaltung.Vorlesung	4
4	0	3	GDIL	2	Grundlagen der Informatik Labor	3	lehrveranstaltungsverwaltung.Uebung	NULL

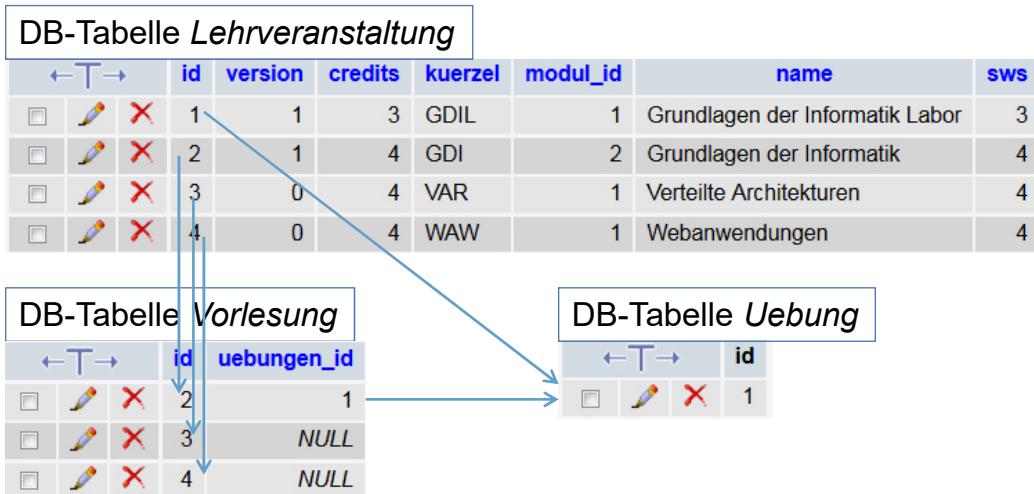
## Variante 2: DB-Tabelle je Klasse (Table per Subclass)

- Basisklasse ergänzen um

```
static mapping = {
 tablePerHierarchy false
}
```

Kein Doppelpunkt!
- jede abgeleitete Klasse bekommt eigene DB-Tabelle
  - zusätzlich zur Basisklassentabelle
- Nachbearbeitung der Controller und Views wie bei Eintabellenlösung
- **Vorteile:**
  - Spalten brauchen nicht *nullable* sein
  - keine Diskriminator-Spalte nötig
- **Nachteile:**
  - mehr DB-Tabellen
  - Abruf einer abgeleiteten Instanz erfordert DB-Join
  - Abruf aller Instanzen der Vererbungshierarchie erfordert DB-UNION

## Variante 2: DB-Tabellen



## Konfiguration der GORM-Datenbankverbindung

- Konfigurationsdatei: `conf/application.yml`
  - Abschnitt `dataSource` definiert Defaultwerte für alle 3 Konfigurationen
  - Abschnitt `environments` definiert 3 Konfigurationen:
    - `development` Entwicklungskonfiguration, auf Entwicklerproduktivität optimiert
    - `test` Testkonfiguration
    - `production` Produktivbetrieb, auf Performance optimiert
- Beim Austausch der Datenbank beachten:
  - JDBC-Treiber für `mySql`
    - runterladen von [https://mariadb.com/my\\_portal/download/java-client/1.5](https://mariadb.com/my_portal/download/java-client/1.5)
    - Jar-Datei nach `jdk/jre/lib/ext` und `jre/lib/ext` kopieren
  - oder alternativ
    - Konfigurationsdatei `build.gradle` im Abschnitt `dependencies` ergänzen um `runtime "org.mariadb.jdbc:mariadb-java-client"`

## GORM-Konfiguration für H2-DB (1/3)

### `dataSource`:

```
pooled: true
jmxExport: true
driverClassName: org.h2.Driver
username: sa
password:
```

### `environments`:

#### `development`:

```
dataSource:
 dbCreate: create-drop
 url: jdbc:h2:mem:devDb;MVCC=TRUE;
 LOCK_TIMEOUT=10000;DB_CLOSE_ON_EXIT=FALSE
```

#### `test`:

```
dataSource:
 dbCreate: update
 url: jdbc:h2:mem:testDb;MVCC=TRUE;
 LOCK_TIMEOUT=10000;DB_CLOSE_ON_EXIT=FALSE
```

## GORM-Konfiguration für H2-DB (2/3)

***production:***

***dataSource:***

```
dbCreate: update
url: jdbc:h2:./prodDb;MVCC=TRUE;
 LOCK_TIMEOUT=10000;DB_CLOSE_ON_EXIT=FALSE
```

## GORM-Konfiguration für H2-DB (3/3)

***properties:***

```
jmxEnabled: true
initialSize: 5
maxActive: 50
minIdle: 5
maxIdle: 25
maxWait: 10000
maxAge: 600000
timeBetweenEvictionRunsMillis: 5000
minEvictableIdleTimeMillis: 60000
validationQuery: SELECT 1
validationQueryTimeout: 3
validationInterval: 15000
testOnBorrow: true
testWhileIdle: true
testOnReturn: false
jdbcInterceptors: ConnectionState
defaultTransactionIsolation: 2 # TRANSACTION_READ_COMMITTED
```

## GORM-Konfiguration für MariaDB

```
dataSource:
 pooled: true
 jmxExport: true
 driverClassName: org.mariadb.jdbc.Driver
 username: hochschule
 password: hochschule

environments:
 development:
 dataSource:
 dbCreate: create-drop
 url: jdbc:mariadb://localhost/hochschule
 test:
 dataSource:
 dbCreate: update
 url: jdbc:mariadb://localhost/hochschule
 production:
 dataSource:
 dbCreate: update
 url: jdbc:mariadb://localhost/hochschule
```

## GORM: Abbildung Domänenklassen → Datenbankschema

Domänenklasse	Datenbankschema	Bemerkungen
Klassenname	Tabellenname	CamelCase ersetzt durch camel_case
Attributname	Spaltenname	
-	Spalte <i>id</i>	Künstlicher Primärschlüssel
-	Spalte <i>version</i>	Versionsnr. optimist. Sperren
hasMany=[attr:Klasse]	Fremdschlüsselattribut in gegenüberliegender Tabelle	
hasOne=[attr:Klasse]		
belongsTo=[attr:Klasse]	Attribut <i>besitzerklasse_ID</i>	Fremdschlüsselbeziehung
Vererbung	Tabelle pro Hierarchie oder pro Klasse	
-	PRIMARY-Constraint	Primärschlüssel-Constraint
constraints= {}	DB-Constraint, Index	

## GORM-Customizing

- Convention over Configuration:
  - Default OR-Mapping (vgl. vorige Folie)
- Bei Bedarf (fast) beliebig anpassbar
  - Adaption an bestehendes DB-Schema
    - neue Anwendung zu bestehendem Datenmodell
  - Einhaltung von Konventionen, z.B.
    - zu jeder Tabelle einheitlicher Attribute-Präfix, z.B. `lv` für `lehrveranstaltung`
    - Attributnamen dieser Tabelle beginnen mit Präfix, gefolgt von `_`  
Beispiel: `lv_id`, `lv_name`, `lv_kuerzel`

## GORM-Customizing: Einfaches Beispiel

- Customizing des OR-Mappers mit **statischem mapping-Attribut**:

```
class Lehrveranstaltung {
 String name
 String kuerzel
 int sws
 int credits

 static belongsTo=[modul:Modul]

 static mapping = {
 table 'Lehrveranstaltungen' // Name der DB-Tabelle vorgeben
 name column:'lv_name' // Spaltenname vorgeben
 modul column:'lv_modul' // Spaltenname vorgeben für Beziehung
 kuerzel column:'lv_kuerzel' type:'text' // Hibernate-Spaltentyp vorgeben
 }
 // ...
}
```

**GORM-Customizing**

**Tabellenkonfiguration**

**Attributname**

**Assoziatives Array mit Customizing-Konfiguration**

## GORM-Customizing: m:n-Beziehung

```
• class Professor {
 String vorname
 String nachname
 static hasMany = [vorlesung:Vorlesung]

 static mapping = {
 vorlesung joinTable:'Professor_Vorlesung_Verknuepfung'
 }
 // ...
}

• class Vorlesung {
 String name
 String kuerzel
 static hasMany = [professor:Professor]
 static belongsTo= Professor

 static mapping = {
 professor joinTable:'Professor_Vorlesung_Verknuepfung'
 }
 // ...
}
```

Name der m:n-Beziehungstabelle

## GORM-Customizing: Tabelle

Attribut	Bedeutung	Default
<b>table 'Tabellenname'</b>	Datenbanktabellenname	Domänenklassenname
<b>sort 'sortierattribut'</b>	Sortierattribut festlegen	unsortiert
<b>sort sortierattribut:'asc'</b>	aufsteigend	aufsteigend
<b>sort sortierattribut:'desc'</b>	absteigend	
<b>tablePerHierarchy true</b>	Tabelle je Vererbungshierarchie	true
<b>tablePerHierarchy false</b>	Tabelle je Klasse	
<b>version true</b>	Optimistisches Sperren mit injiziertem <i>version</i> -Attribut	true
<b>version false</b>	Pessimistisches Sperren ohne <i>version</i> -Attribut	
<b>discriminator attr:value ...</b>	Diskriminator-Konfiguration	eigene Folie
<b>attributname attr:value ...</b>	Attribut-Konfiguration	eigene Folie

## GORM-Customizing: Tabelle: Syntaxbeispiel Basisklasse

```
static mapping = {
 tablePerHierarchy false // je 1 Tabelle pro Klasse
 table 'lehrveranstaltung' // Tabellenname für Basistabelle
 discriminator column:'lv_lehrveranstaltungstyp' // Diskriminatorspaltenname
 kuerzel column: 'lv_kuerzel' // Name der Attributspalte
}
```

## GORM-Customizing: Diskriminator

Attribut	Bedeutung	Default
<b>column:</b> 'Spaltenname'	Name für Diskriminatorspalte (nur für Basisklasse)	<i>class</i>
<b>type:</b> <i>Diskriminatorspaltentyp</i> '	Hibernate-Datentyp für Diskriminator	<i>string</i>
<b>value:</b> ' <i>Diskriminatorwert</i> '	Diskriminatorwert für diese Domänenklasse	Name Domänenklasse inkl. Packagepfad

Syntaxbeispiel für abgeleitete Klasse:

```
static mapping = {
 table vorlesung' // Tabellenname für Basistabelle
 discriminator value:'Vorlesung' // Diskriminatorwert für diese abgeleitete Klasse
}
```

## GORM-Customizing: Spalten

Attribut	Bedeutung	Default
<b>column:'Spaltenname'</b>	Spaltenname	Attributname
<b>type:'Spaltentyp'</b>	Hibernate-Datentyp f. Spalte	je nach Attributtyp
<b>joinTable:'Spaltentyp'</b>	Name m:n-Beziehungstabelle	klassenname_refattribut
<b>index:'Indexname'</b>	Index für Spalte anlegen	Indizes nur für Primär und Fremdschlüsselattribute
<b>lazy:true</b>	Referenzierte Instanz erst bei Zugriff laden	<i>true</i>
<b>lazy:false</b>	Referenzierte Instanz sofort laden	

Syntaxbeispiel:

```
statichasMany = [gehalteneVorlesung:GehalteneVorlesung]
```

```
mapping = {
 gehalteneVorlesung column:'haelt_vorlesung', lazy:false
}
```

## Einfluss der Constraints auf das Datenmodell

Constraint-Attribut	Konsequenz für Datenmodell
<b>max:maxAnzahlZeichen</b>	Spalte vom Typ Langtext (TEXT) statt Text (varchar(255))
<b>inList:[...]</b>	Stringlänge auf längsten Eintrag der Auswahlliste setzen
<b>maxSize:laenge</b>	Stringlänge auf <i>laenge</i> setzen
<b>size:laenge</b>	Stringlänge auf <i>laenge</i> setzen

## Kaskadierendes Löschen

- **static belongsTo**-Attribut legt in Beziehungen die abhängige Seite fest
  - Löschen der Besitzerseite kaskadiert auf abhängige Seite
  - Beispiele:
    - Fakultät *belongsTo* Hochschule, Hochschule *hasMany* Fakultäten
      - Löschen der Hochschule löscht all ihre Fakultäten
      - Löschen einer Fakultät löscht **nicht** die Hochschule
    - Hochschule *belongsTo* Fakultät, Hochschule *hasMany* Fakultäten
      - Löschen einer(!) Fakultät löscht die Hochschule → unsinnig!
      - Löschen der Hochschule löscht **nicht** deren Fakultäten
- **Regeln:**
  - In 1:n-Beziehungen immer *hasMany* auf der 1-Seite, *belongsTo* auf der n-Seite
  - In 1:1-Beziehungen immer *belongsTo* auf **mindestens** einer Seite
  - In m:n-Beziehungen immer *belongsTo* auf **genau** einer Seite

## Kaskadierendes Löschen einer 1:n-Beziehung

Klasse A (1-Seite) Syntax	FK	Klasse B (n-Seite) Syntax	FK	kaskadierendes Löscherhalten	
				a → b	b → a
<b>static hasMany=[b:B]</b>		<b>static hasOne=[a:A]</b>	<b>a</b>	-*	-
<b>static hasMany=[b:B]</b>		<b>static belongsTo=[a:A]</b>	<b>a</b>	ja	-
<b>static hasMany=[b:B]</b> <b>static belongsTo=B</b>		<b>static hasOne=[a:A]</b>	<b>a</b>	-*	ja
<b>static hasMany=[b:B]</b> <b>static belongsTo=B</b>		<b>static belongsTo=[a:A]</b>	<b>a</b>	ja	ja**

FK = Fremdschlüsselattribut (Foreign Key)

\* a wird gelöscht → Fremdschlüssel von b hängt in der Luft → Zugriffsfehler in Views

\*\* b wird gelöscht → kaskadierend a → kaskadierend alle mit a assoziierten b

## Kaskadierendes Löschen: 1:n-Beziehung

- 1:n-Beziehung n-Seite **belongsTo** 1-Seite:
  - Löschen der 1-Seite kaskadiert auf n-Seite
  - Beispiel: Löschen der Hochschule löscht auch deren Fakultäten

### Klasse A

```
package kaskadierendesloeschen

class Hochschule {
 String name
 String ort

 static hasMany = [fakultaeten:Fakultaet]
}
```

### Klasse B

```
package kaskadierendesloeschen

class Fakultaet {
 String name
 String kuerzel

 static belongsTo =
 [hochschule:Hochschule]
}
```

## Kaskadierendes Löschen einer 1:1-Beziehung

Klasse A (1-Seite) Syntax	FK	Klasse B (1-Seite) Syntax	FK	kaskadierendes Löscherhalten	
				a → b	b → a
<b>static hasOne=[b:B]</b>	<b>b</b>	<b>static hasOne=[a:A]</b>	<b>a</b>	-*	-*
<b>static hasOne=[b:B]</b>	-	<b>static belongsTo=[a:A]</b>	<b>a</b>	ja	-
<b>static belongsTo=[b:B]</b>	<b>b</b>	<b>static hasOne=[a:A]</b>	-	-	ja
<b>static belongsTo=[b:B]</b>	<b>b</b>	<b>static belongsTo=[a:A]</b>	<b>a</b>	ja	ja
<b>B b</b>	<b>b</b>	-	-	-	-
<b>B b</b>	<b>b</b>	<b>static belongsTo=A</b>	-	ja	-

FK = Fremdschlüsselattribut (Foreign Key)

\* Zugriffsfehler in Views bereits beim Anlegen der Datensätze!

## Kaskadierendes Löschen einer m:n-Beziehung

Klasse A (1-Seite)		Klasse B (1-Seite)		kaskadierendes Löschverhalten	
Syntax	FK	Syntax	FK	a → b	b → a
<b>static hasMany=[b:B]</b>	-	<b>static hasMany=[a:A]</b>	-	nicht erlaubt	
<b>static hasMany=[b:B]</b>	-	<b>static hasMany=[a:A] static belongsTo=A</b>	-	-*	-
<b>static hasMany=[b:B] static belongsTo=B</b>	-	<b>static hasOne=[a:A]</b>	-	-	-*
<b>static hasMany=[b:B] static belongsTo=B</b>	-	<b>static hasMany=[a:A] static belongsTo=A</b>	-	nicht erlaubt	

\* Löscht nur den Datensatz in der Beziehungstabelle, **nicht** in A bzw. B Fremdschlüsselattribute a und b in Beziehungstabelle

## Transiente Attribute

- von Persistierung ausgeschlossen, z.B. aus folgenden Gründen:
  - zur Laufzeit berechnet
  - reine Laufzeitinformationen
  - komplexe Datentypen, die nicht in Datenbank speicherbar sind
- Syntax: **static transients=['attributname', ...];**

## GORM-Methoden: 3 Aufrufvarianten

1. direkt auf **Domänenklasseninstanz**
  - **save** und **delete**, weil diese beiden Methoden ein **konkretes Objekt** betreffen
    - eingeimpft via **Metaprogrammierung**
2. als **statische** Methode der **Domänenklasse**
  - viele weitere GORM-Methoden (**list**, **count**, **get**, **findXXX**)
    - **instanzunabhängig** aufrufbar
    - eingeimpft via **Metaprogrammierung**
3. über Variable **xxxService** im Controller (xxx=Domänenklasennname)
  - gleichwertig zu 2.
  - eingeimpft via **Metaprogrammierung**

## Eingeimpfte Domänenklassen-Methoden (1/3)

- **Domainclass.list()**
  - liefert Collection mit allen Instanzen dieser Domänenklasse
  - Benannte Parameter:
    - **max:** maximale Anzahl zu liefernde Instanzen
    - **offset:** Offset (relativ zu 0), ab dem die Instanzen geliefert werden
    - **sort:** Attribut, nach dem die Instanzen sortiert werden sollen
    - **order:** **asc** (vorwärts sortieren) oder **desc** (rückwärts sortieren)
    - **ignoreCase:** **true**: Groß-/Kleinschreibung beim Sortieren ignorieren (Default)  
**false**: Groß-Kleinschreibung unterscheiden
- **Domainclass.count()**
  - liefert Anzahl der Instanzen dieser Domänenklasse

## Eingeimpfte Domänenklassen-Methoden (2/3)

- **Domainclass.get(id)**
  - liefert Instanz mit dem Primärschlüssel id
  - Methodenparameter:
    - *id*: Primärschlüssel des gewünschten Datensatzes
- **Domainclass.getAll(id ...)**
  - liefert Instanzen mit angegebenen Primärschlüsseln
  - Methodenparameter:
    - *id*: Kommagetrennte Aufzählung von Primärschlüsseln darf auch ein Parameter vom Typ Liste sein

## Eingeimpfte Domänenklassen-Methoden (3/3)

- **domainobject.save()**
  - persistiert diese Instanz in die Datenbank (→ SQL INSERT oder UPDATE)
  - Benannte Parameter:
    - *flush*: *true* → Änderungen sofort in DB schreiben  
*false* → Änderungen spätestens bei Commit schreiben (Default)
    - *validate*: *true* → Attributwerte validieren (Default)  
*false* → Attributwerte nicht validieren
- **domainobject.delete()**
  - löscht diese Instanz aus der Datenbank (→ SQL DELETE)
  - Benannte Parameter:
    - *flush*: *true* → Änderungen sofort in DB schreiben  
*false* → Änderungen spätestens bei Commit schreiben (Default)

## Dynamische Finder: findBy

- **Domainclass.findByAttr(value)**
  - MethodenParameter
    - Attr: Attribut der Domainclass, erster Buchstabe groß geschrieben
    - value: zu suchender Wert
  - Ergebnis:
    - **Einzig**e Instanz dieser Domänenklasse mit *attr=value*
    - Exception, wenn mehrere Instanzen das Suchkriterium erfüllen
  - Beispiel: *Hochschule.findByNombre("Hochschule Mannheim")*

## Dynamische Finder: findAllBy

- **Domainclass.findAllByAttr(value, benannteParameterMap)**
  - Parameter
    - Attr: Attribut der Domainclass, erster Buchstabe groß geschrieben
    - value: zu suchender Wert
  - Benannte Parameter (am Ende der Parameterliste als Assoziatives Array)
    - max: maximale Anzahl zu liefernde Instanzen
    - offset: Offset (relativ zu 0), ab dem die Instanzen geliefert werden
    - sort: Attribut, nach dem die Instanzen sortiert werden sollen
    - order: asc (vorwärts sortieren) oder desc (rückwärts sortieren)
    - ignoreCase: true: Groß-/Kleinschreibung beim Sortieren ignorieren (Default)  
false: Groß-Kleinschreibung unterscheiden
  - Ergebnis:
    - Collection **aller** Instanzen dieser Domänenklasse mit *attr=value*
  - Beispiel: *Hochschule.findAllByOrt("Mannheim", [sort:'Name', max:10])*
- auf folgenden Folien *findBy* und *findAll* analog, auch wenn nur eine genannt ist

## Dynamische Finder: Ähnlichkeitssuche

- Domainclass.findAllBy**Attr1Like(sqlpattern)**
  - Parameter
    - *sqlPattern*: Like-Pattern für WHERE ... LIKE Klausel (SQL-Syntax)
      - % 0, 1 oder mehrere Zeichen
      - Genau ein Zeichen
  - Ergebnis: Alle Datensätze, die SQL-Pattern entsprechen
  - Beispiel: *Student.findAllByStudiengangLike('%B')* // Bachelor-Studenten
- Domainclass.findAllBy**Attr1Ilike(sqlpattern)** // Groß-Ignore, klein-I
  - wie Attr1Linke, ignoriert aber Groß-/Kleinschreibung

## Dynamische Finder: Vergleiche

- Domainclass.findAllBy**Attr1LesserThan(value1)**
  - Parameter
    - *value1*: Vergleichswert
  - Ergebnis: Alle Datensätze, für die *attr1<value1*
  - Beispiel: *Student.findAllByAlterLesserThan(18)* // Jugendliche
- Domainclass.findAllBy**Attr1LesserThanEquals(value1)**
  - Ergebnis: Alle Datensätze, für die *attr1<=value1*
- Domainclass.findAllBy**Attr1GreaterThanOrEqual(value1)**
  - Ergebnis: Alle Datensätze, für die *attr1>value1*
- Domainclass.findAllBy**Attr1GreaterThanOrEqual(value1)**
  - Ergebnis: Alle Datensätze, für die *attr1>=value1*

## Dynamische Finder: Bereichsangaben und Listen

- Domainclass.findAllBy**Attr1Between**(*value1, value2*)
  - Parameter
    - *value1*: Untergrenze
    - *value2*: Obergrenze
  - Ergebnis: Alle Datensätze, für die *value1<=attr1<=value2*
  - Beispiel: *Student.findAllByAlterBetween(18,25)* // Studenten normaler
- Domainclass.findAllBy**Attr1InList**(*list*)
  - Parameter
    - *list*: Groovy-Liste von Werten
  - Ergebnis: Alle Datensätze, für die *value1* in *list* enthalten
  - Beispiel: *Student.findAllBySemesterInList([1,2,3,6,7])* // Theoriesemester

## Dynamische Finder: Null

- Domainclass.findAllBy**Attr1IsNull()**
  - Ergebnis: Alle Datensätze, für die *attr1=null*
  - Beispiel: *Student.findAllByAdresseIsNull()* // Wohnsitzlose
- Domainclass.findAllBy**Attr1IsNotNull()**
  - Ergebnis: Alle Datensätze, für die *attr1!=null*

## Dynamische Finder: Logische Verknüpfung mehrerer Attribute

- Domainclass.findBy**Attr1AndAttr2**(value1,value2)
  - Ergebnis:
    - Einzige Instanz dieser Domänenklasse mit attr1=value1 und attr2=value2
    - Exception, wenn mehrere Instanzen das Suchkriterium erfüllen
- Domainclass.findAllBy**Attr1AndAttr2**(value1, value2)
  - Alle Instanzen dieser Domänenklasse mit attr1=value1 und attr2=value2
- Domainclass.findBy**Attr1OrAttr2**(value1,value2)
  - Ergebnis: Instanz dieser Domänenklasse mit attr1=value1 oder attr2=value2
- Domainclass.findBy**Attr1NotEqual** (value1)
  - Ergebnis: Instanz dieser Domänenklasse mit attr1!=value1

## Dynamische Finder: Beispiel für komplexe Bedingung

- Finder-Methoden lassen sich kombinieren:

```
// Alle Java-Bücher in den letzten 30 Tagen erschienenen Java-Bücher
def books = Book.findAllByTitleLikeAndReleaseDateGreaterThan(
 "%Java%", new Date() - 30)
```
- Finder-Methodename
  - darf nur *And*- oder nur *Or*-Verknüpfungen enthalten
  - niemals beide

## Dynamische Finder: Suche nach Beziehungen

- Auch nach **Beziehungsattributen** kann gesucht werden
  - referenzierten Datensatz suchen mit
    - `def referenziert = ReferenzierteDomänenklasse.findByXXX()`
    - `def referenziert = ReferenzierteDomänenklasse.get(id)`
  - **Instanz** an dynamische Finder-Methode übergeben
    - Primärschlüssel reicht **nicht**
  - Beispiel:

```
def hochschule = Hochschule.get(hochschule_id)
found = Fakultaet.findAllByNameLikeAndKuerzelLikeAndHochschule(
 name ?: '%',
 kuerzel ?: '%',
 hochschule,
 params)
```

referenzierten Datensatz suchen

referenzierte Instanz

## Beispielprojekt mit Suchfunktion: Hochschulen und Fakultäten

- Ziel:
  - Anwendung zur Verwaltung der Fakultäten mehrerer Hochschulen
  - Suchfunktion nach Hochschulen und Fakultäten
  - Automatische Bereitstellung von Testdaten nach dem Start der Anwendung
- **Vorgehensweise:**
  - **Domänenklassen** *Hochschule* und *Fakultät* anlegen
  - **Controller** und Views mit statischem Scaffolding generieren
  - **Testdaten** in *conf/BootStrap.groovy* bereitstellen
  - Controller um **Suchaktionen** *search* und *searchResults* ergänzen
  - **Template** für **Suchparametereingabe** durch Kopieren aus *\_form.gsp* erstellen
    - Anpassungen nötig
  - **View** für **Suchparametereingabe** durch Kopieren aus *create.gsp* erstellen
    - Anpassungen nötig
  - **View** für **Suchergebnisse** durch Kopieren aus *index.gsp* erstellen
    - Anpassungen nötig

## Domänenklasse Hochschule

```
package fakultaetssuche

class Hochschule {
 String name
 String ort

 statichasMany = [fakultaeten:Fakultaet]

 static constraints = {
 name nullable: false, blank: false
 ort nullable: true, blank: true
 }

 String toString() {
 name
 }
}
```

## Domänenklasse Fakultaet

```
package fakultaetssuche

class Fakultaet {
 String name
 String kuerzel

 static belongsTo=[hochschule:Hochschule]

 static constraints = {
 name nullable: false, blank: false
 kuerzel nullable: true, blank: true
 hochschule nullable: true
 }

 String toString() {
 name
 }
}
```

## Start der Anwendung – BootStrap.groovy

- während der **Entwicklungsphase**
  - lässt man meist die DB-Tabellen bei jedem Start neu generieren
  - müsste man deshalb bei jedem Programmstart die Testdaten erneut eingeben
    - hoher Zeit- und Tippaufwand
- Abhilfe: *init/BootStrap.groovy*
  - definiert zwei Closures:
  - **def init = { servletContext -> ... }**
    - wird beim Starten der Anwendung ausgeführt
    - für Initialisierung + Ressourcen-Anforderung
    - kann für Erzeugung von Testdatensätzen genutzt werden
    - defaultmäßig leere Closure
  - **def destroy = { ... }**
    - wird beim Beenden der Anwendung ausgeführt
    - für Ressourcen-Freigabe
    - kann zum Löschen der Testdatensätze genutzt werden
    - defaultmäßig leere Closure

## Testdaten in BootStrap.groovy generieren

- innerhalb der *init*-Closure für jeden Testdatensatz:
  - **Domänenobjekt erzeugen** durch Instanzierung der Domänenklasse
    - impliziter Konstruktor mit benannten Parametern steht zur Verfügung
    - Beispiel:  
`def hsMannheim = new Hochschule(name:"HS Mannheim", ort: "Mannheim")`
  - **Domänenobjekt mit `save(flush:true)`** in Datenbank schreiben
    - Beispiel:  
`hsMannheim.save flush:true`
- Tipps:
  - Domänenobjekte so in der Reihenfolge erzeugen und saven, dass
    - **referenzierte** Objekte **vor** den **referenzierenden** eingefügt werden
    - nachträgliches Setzen und Update von Referenzattributen vermieden wird
  - Einhaltung der Constraints für die Initialisierungsdaten sicherstellen
    - werden durch `save flush:true` geprüft

## Testdaten in *init/BootStrap.groovy* generieren (1/2)

```
import fakultaetssuche.*;

class BootStrap {

 def init = { servletContext ->
 println "Initialisiere Datenbank ..."

 def hsMannheim = new Hochschule(name:"HS Mannheim", ort: "Mannheim")
 def uniMannheim = new Hochschule(name: "Uni Mannheim", ort:"Mannheim")
 def hsLudwigshafen = new Hochschule(name:"HS Ludwigshafen",ort: "Ludwigshafen")
 hsMannheim.save flush:true
 uniMannheim.save flush: true
 hsLudwigshafen.save flush: true
 }
}
```

## Testdaten in *init/BootStrap.groovy* generieren (2/2)

```
def infHsMannheim = new Fakultaet(
 name:"Informatik", kuerzel: "I", hochschule: hsMannheim)
infHsMannheim.save flush: true

def tInfHsMannheim = new Fakultaet(
 name:"Technische Informatik", kuerzel: "N", hochschule: hsMannheim)
tInfHsMannheim.save flush: true

def infUniMannheim = new Fakultaet(
 name:"Informatik", kuerzel: "I", hochschule: uniMannheim)
infUniMannheim.save flush: true

def wInfUniMannheim = new Fakultaet(
 name:"Wirtschaftsinformatik", kuerzel: "WI", hochschule: uniMannheim)
wInfUniMannheim.save flush: true
println "Datenbank initialisiert"
}

def destroy = {
}
}
```

## Einschub: Grails-Environments

- definiert in *grails-app/conf/application.yml*
- Standard-Environments
  - *development* für Entwicklungsphase (default)
  - *test* für Testphase
  - *production* für Qualitätssicherung + Betrieb
- Auswahl des Environments beim Starten:
  - grails *development* run-app
  - grails *test* run-app
  - grails *production* run-app
- Feststellen des Environments zur Laufzeit
  - Umgebungsvariable *grails.util.Environment.current.name*
  - Beispiel:

```
if (grails.util.Environment.current.name == "development") {
 ...
}
```

## HochschuleController: Suchaktionen ergänzen

```
// Suchformular
def search() {
 respond new Hochschule(params)
}

// Suchergebnisse
def searchresults(String name, String ort, int max) {
 params.max = Math.min(max ?: 10, 100) // Für Paginator
 Collection <Hochschule> results =
 Hochschule.findAllByNameLikeAndOrtLike(name?:'%', ort?:'%', params);
 respond results, model:[hochschuleCount: results.size()]
}
```

Auto-Parameterübergabe  
der Suchkriterien

Leerlassen als Wildcard

Paginator-Parameter

## hochschule/index.gsp: Suchbutton ergänzen

```
<div class="nav" role="navigation">

<g:link class="create" action="create">
<g:message code="default.new.label" args="[entityName]" />+
</g:link>

<g:link class="create" action="search">Suche ${entityName}
</g:link>

</div>
```

Nur der änderungsrelevante  
Ausschnitt ist abgedruckt!

## hochschule/search.gsp erstellen

- **Kopervorlage:** *hochschule/create.gsp*
    - ist die View für die *create*-Aktion (neue Hochschule erstellen)
    - muss geringfügig angepasst werden
      - Suchmaske soll keine Pflichtsuchattributte enthalten
      - Nicht-Suchattribute von der Suchmaske entfernen
    - Vorgehen
      - *<f:all ...>* ersetzen durch *<f:field ... required="false">*
- alle** Attribute darstellen  
(als Pflicht/Kann abh. von Constraints)
- einzelne** Attribute darstellen  
(als Kann)-Attribzut)
- Beispiel: *<f:all bean="hochschule"/>* ersetzen durch
    - <f:field bean="hochschule" property="name" required="false"></f:field>*
    - <f:field bean="hochschule" property="ort" required="false"></f:field>*

```
<!DOCTYPE html>
<html>
<head>
<meta name="layout" content="main" />
<g:set var="entityName"
 value="${message(code: 'hochschule.label', default: 'Hochschule')}" />
<title>Suche ${entityName}</title>
</head> Fenstertitel
<body>

 <g:message code="default.link.skip.label" default="Skip to content…" />

<div class="nav" role="navigation">

 ${createLink(uri: '/')}">
 <g:message code="default.home.label"/>
 <g:link class="list" action="index">
 <g:message code="default.list.label" args="[entityName]" /></g:link>

</div>
```

```
<div id="create-hochschule" class="content scaffold-create" role="main">
<h1>Suche ${entityName}</h1>
 Seitenüberschrift
<g:if test="${flash.message}">
 <div class="message" role="status">${flash.message}</div>
</g:if>
<g:hasErrors bean="${this.hochschule}">
 <ul class="errors" role="alert">
 <g:eachError bean="${this.hochschule}" var="error">
 <li <g:if test="${error instanceof org.springframework.validation.FieldError}">
 data-field-id="${error.field}"
 </g:if>
 >
 <g:message error="${error}" />

 </g:eachError>

</g:hasErrors>
```

```
<g:form action="searchresults">
 <f:field bean="hochschule" property="name" required="false"></f:field>
 <f:field bean="hochschule" property="ort" required="false"></f:field>
</fieldset>
<fieldset class="buttons">
 <g:submitButton name="searchresults" class="save" value="Suchen" />
</fieldset>
</g:form>
</div>
</body>
</html>
```

Annotations:

- Controller-Aktion: Annotates the line `<g:form action="searchresults">`.
- Controller-Aktion: Annotates the line `<g:submitButton name="searchresults" class="save" value="Suchen" />`.
- Buttonbeschriftung: Annotates the line `value="Suchen"`.

## hochschule/searchResults.gsp erstellen

- **Kopervorlage:** `hochschule/index.gsp`
  - ist die View für die index-Aktion (Liste aller Hochschulen)
  - kann nahezu unverändert übernommen werden
    - damit Suchkriterien bei erneuter Suche erhalten bleiben:  
`</i>`  
`<g:link class="create" action="search" params="${params}">Suche`  
 `${entityName}</g:link>`
    - Suchkriterien übergeben
  - ggf. kleine textuelle Anpassungen der Beschriftungstexte

## Bugs in Grails bei der Anzeige der Suchergebnisse

- Fields-Plugin
  - <f:table ...> zur Darstellung der Ergebnisliste
    - gibt Suchkriterium nicht weiter
    - bei Änderung der Sortierreihenfolge geht Suchkriterium verloren
  - Abhilfe: Template `views/templates/fields/_table.gsp`
    - von [https://github.com/grails3-plugins/fields/blob/master/grails-app/views/templates/\\_fields/\\_table.gsp](https://github.com/grails3-plugins/fields/blob/master/grails-app/views/templates/_fields/_table.gsp) kopieren
    - gemäß folgender Folie patchen
- Paginator
  - wird nicht dargestellt, wenn Suchergebnisse nicht auf eine Seite passen

## Bugfix im Template \_table.gsp 1/2

```
<table>
<thead>
<tr>
<g:each in="${domainProperties}" var="p" status="i">
<g:set var="propTitle">${domainClass.propertyName}.${p.name}.label</g:set>
<g:sortableColumn property="${p.name}"
 title="${message(code: propTitle, default: p.naturalName)}"
 params="${params}"/>
</g:each>
</tr>
</thead>
```

Suchparameter beim  
Sortieren erhalten

## Bugfix im Template \_table.gsp 2/2

```
<tbody>
<g:each in="${collection}" var="bean" status="i">
 <tr class="${(i % 2) == 0 ? 'even' : 'odd'}">
 <g:each in="${domainProperties}" var="p" status="j">
 <g:if test="$j==0">
 <td><g:link method="GET" resource="${bean}">
 <f:display bean="${bean}" property="${p.name}"
 displayStyle="${displayStyle?:'table'}" />
 </g:link></td>
 </g:if>
 <g:else>
 <td>
 <f:display bean="${bean}" property="${p.name}"
 displayStyle="${displayStyle?:'table'}" />
 </td>
 </g:else>
 </g:each>
 </tr>
</g:each>
</tbody>
</table>
```

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

431

## FakultaetController: Suchaktionen ergänzen

```
// Suchformular
def search() {
}

// Suchergebnisse
def searchResults(Integer max, String name, String kuerzel, Long hochschule_id) {
 params.max = Math.min(max ?: 10, 100)

 def found=null
 if (hochschule_id) {
 def hochschule = Hochschule.get(hochschule_id)
 found = Fakultaet.findAllByNameAndKuerzelLikeAndHochschule(
 name ?: '%', kuerzel ?: '%', hochschule, params)
 }
 respond found, model:params+[fakultaetInstanceCount: found.size()]
}
```

**Für Paginator**

**Auto-Parameterübergabe der Suchkriterien**

**Leerlassen als Wildcard**

**Paginator-Parameter**

**Such- und Paginatorparameter mitgeben**

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

432

## fakultaet/index.gsp: Suchbutton ergänzen

```
<div class="nav" role="navigation">

<g:link class="create" action="create">
<g:message code="default.new.label" args="[entityName]" />+
</g:link>

<g:link class="create" action="search">Suche ${entityName}</g:link>

</div>
```

Nur der änderungsrelevante  
Ausschnitt ist abgedruckt!

## Alt: Template fakultaet/\_formsearch.gsp erstellen

- **Kopervorlage:** *fakultaet/\_form.gsp*
  - ist das Standardtemplate für die *create-* und *edit*-Aktion
- muss geringfügig **angepasst** werden (Änderungen in **rot** markiert):
  - Muss-Felder zu Kann-Feldern herabstufen
    - sonst **müssten** Mussfelder als Suchkriterium angegeben werden
  - In Fremdschlüsselfeldern . durch \_ ersetzen
    - Beispiel: *hochschule\_id* statt *hochschule.id*
    - Grund: Controller-Aktion kann mit Punkt-Schreibweise nicht umgehen
      - Fehler bei erneutem Aufruf der Ergebnisse durch Paginator / Sortieren

## Alt: Template fakultaet/\_formsearch.gsp erstellen (1/2)

```
<%@ page import="fakultaetssuche.Fakultaet" %>

<div class="fieldcontain ${hasErrors(bean: fakultaetInstance, field: 'name', 'error')} required">
 <label for="name">
 <g:message code="fakultaet.name.label" default="Name" />
 *
 </label>
 <g:textField name="name" required="" value="${fakultaetInstance?.name}"/>
</div>

<div class="fieldcontain ${hasErrors(bean: fakultaetInstance, field: 'kuerzel', 'error')} ">>
 <label for="kuerzel">
 <g:message code="fakultaet.kuerzel.label" default="Kuerzel" />
 </label>
 <g:textField name="kuerzel" value="${fakultaetInstance?.kuerzel}"/>
</div>
```

## Alt: Template fakultaet/\_formsearch.gsp erstellen (2/2)

```
<div class="fieldcontain ${hasErrors(bean: fakultaetInstance, field: 'hochschule', 'error')} ">>
 <label for="hochschule">
 <g:message code="fakultaet.hochschule.label" default="Hochschule" />
 </label>
 <g:select id="hochschule"
 name="hochschule_id" hochschule.id durch
hochschule_id ersetzen
 from ="${fakultaetssuche.Hochschule.list()}"
 optionKey="id"
 value ="${fakultaetInstance?.hochschule?.id}"
 class="many-to-one" noSelection="['null': '']"/>
</div>
```

## Alt: fakultaet/search.gsp erstellen (1/3)

```
<!DOCTYPE html>
<html>
<head>
 <meta name="layout" content="main">
 <g:set var="entityName" value="${message(code: 'fakultaet.label', default: 'Fakultaet')}" />
 <title>Fakultät suchen</title>
</head>
<body>
 Fenstertitel
 <g:message code="default.link.skip.label" default="Skip to content…" />
 <div class="nav" role="navigation">


```

## Alt: fakultaet/search.gsp erstellen (2/3)

```
<div id="create-fakultaet" class="content scaffold-create" role="main">
 <h1>Fakultät suchen</h1>
 Seitenüberschrift
 <g:if test="${flash.message}">
 <div class="message" role="status">${flash.message}</div>
 </g:if>
 <g:hasErrors bean="${fakultaetInstance}">
 <ul class="errors" role="alert">
 <g:eachError bean="${fakultaetInstance}" var="error">
 <li <g:if test="${error instanceof org.springframework.validation.FieldError}">
 data-field-id="${error.field}"</g:if>><g:message error="${error}" />
 </g:eachError>

 </g:hasErrors>
```

## Alt: fakultaet/search.gsp erstellen (3/3)

```
<g:form url="[resource:fakultaetInstance, action:'searchResults']>
 <fieldset class="form">
 <g:render template="formsearch" />
 </fieldset>
 <fieldset class="buttons">
 <g:submitButton name="searchResults" class="save" value="Suchen" />
 </fieldset>
</g:form>
</div>
</body>
</html>
```

The diagram shows the structure of the search.gsp code. It highlights several parts with callout boxes:

- A blue box labeled "Controller-Aktion" points to the line `action:'searchResults' ]"`.
- A blue box labeled "Template mit Eingabefeldern" points to the line `<g:render template="formsearch" />`.
- A blue box labeled "Controller-Aktion" points to the line `<g:submitButton name="searchResults" ...>`.
- A blue box labeled "Buttonbeschriftung" points to the line `value="Suchen" />`.

## Alt: fakultaet/searchResults.gsp erstellen

- **Kopervorlage:** `fakultaet/index.gsp`
  - ist die View für die index-Aktion (Liste aller Fakultäten)
- muss geringfügig **angepasst** werden (Änderungen in rot markiert):
  - Alle `g:sortableColumn` müssen `params` wieder an Controller-Aktion übergeben
  - `g:paginator` muss `params` wieder an Controller-Aktion übergeben
  - Beispiel:

```
<g:sortableColumn
 property="name"
 title="${message(code: 'fakultaet.name.label', default: 'Name')}"
 params="${params}" />

<g:paginate total="${fakultaetInstanceCount ?: 0}" params="${params}" />
```
  - Grund: Sortieren und Paginator rufen Controller-Aktion erneut auf
    - Suchparameter müssen erhalten bleiben

## Alt: fakultaet/searchResults.gsp erstellen

```
<thead>
<tr>
 <g:sortableColumn
 property="name"
 title="${message(code: 'fakultaet.name.label', default: 'Name')}"
 params="${params}">

 <g:sortableColumn
 property="kuerzel"
 title="${message(code: 'fakultaet.kuerzel.label', default: 'Kuerzel')}"
 params="${params}">

 <th><g:message code="fakultaet.hochschule.label" default="Hochschule" /></th>
 </tr>
</thead>

<!-- hier ist einiges ausgelassen -->
<div class="pagination">
 <g:paginate total="${fakultaetInstanceCount ?: 0}" params="${params}" />
</div>
```

## Where-Queries

- **Where-Klauseln in Groovy-DSL** spezifizieren
  - Syntax ähnlich Groovy-Ausdrücken
    - übliche Groovy-Operatoren
  - Realisierung als Closure
    - durch Compiler überprüfbar
  - komplexe Verknüpfungen möglich
    - im Gegensatz zu Dynamischen Finder-Methoden auch
      - And / Or gemischt
      - Klammerung zur Festlegung der Auswertungsreihenfolge
  - automatische Umsetzung nach SQL
    - Selektion erfolgt durch Datenbankserver, **nicht** durch GORM
    - künftig auch für nichtrelationale Datenbanken

## Where-Queries: Query-Definition

- Domänenattribut mit Wert vergleichen

```
Domänenklasse
def query = Professor.where {
 Attribut der Domänenklasse Vergleichswert
 firstName == "Thomas" && lastName == "Specht"
}
Vergleichsoperator Logische Verknüpfung
```

- Zwei Domänenattribute vergleichen

```
Attribut der Domänenklasse Attribut der Domänenklasse
def query = Timer.where {
 startDate <= endDate
}
Vergleichsoperator
```

## Where-Queries: Query ausführen

- vordefinierte Query ausführen:

```
query.list()
```

- gleiche Parameter wie List-Methode der Domänenklasse

- liefert Collection von Domäneninstanzen

- Beispiel:

```
def query = Professor.where {
 kuerzel="SPE"
}
def found = query.list()
```

## Where-Queries: Query direkt ausführen

- Where-Kriterium direkt als Parameter an überladene `find` / `findAll` übergeben:
  - `Domänenklasse.find {}`
  - `Domänenklasse.findAll {}`
- Beispiele:
  - `def specht = Professor.find { kuerzel=="SPE" }`
  - `def erstsemester = Student.findAll { matikelnummer in 1420000..1439999 }`

## Where-Queries: Vergleichsoperatoren Attribut <-> Wert

Vergleich	= Dynamischer Finder	Bedeutung
<code>==</code>		Attributwert gleich
<code>!=</code>	<code>notEqual</code>	Attributwert ungleich
<code>== null</code>		Attributwert ist null
<code>!= null</code>	<code>notEqual</code>	Attributwert ist ungleich null
<code>&lt;</code>	<code>LesserThan</code>	Attributwert kleiner als
<code>&lt;=</code>	<code>LesserThanEquals</code>	Attributwert kleiner gleich
<code>&gt;</code>	<code>GreaterThan</code>	Attributwert größer als
<code>&gt;=</code>	<code>GreaterThanEquals</code>	Attributwert größer gleich
<code>==~</code>	<code>Like</code>	Match mit SQL-Pattern <code>(%,_)*</code>
<code>=~</code>	<code>Ilike</code>	Match mit SQL-Pattern <code>(%,_)</code> , case insensitive*
<code>in [...]</code>	<code>InList</code>	Attributwert in Liste enthalten
<code>in s..e</code>	<code>between</code>	<code>s&lt;=Attributwert&lt;=e</code>

\* erlauben wahlweise als rechten Operanden:

- String mit SQL-Wildcards `%`, `_`
- Groovy Pattern der Form `~/xxxx/`, sofern der DB-Server reguläre Ausdrücke kennt

## Where-Queries: Vergleichsoperatoren Attribut <-> Attribut

Vergleich	= Dynamischer Finder	Bedeutung
==		Attributwert gleich
!=	notEqual	Attributwert ungleich
<	LesserThan	Attributwert kleiner als
<=	LesserThanEquals	Attributwert kleiner gleich
>	GreaterThan	Attributwert größer als
>=	GreaterThanOrEqual	Attributwert größer gleich

## Where-Queries: Logische Verknüpfungen

Vergleich	= Dynamischer Finder	Bedeutung
&&	And	und
	Or	oder
!	(Not)	nicht
()	-	Klammerung für Auswertungsreihenfolge

## Where-Queries: Suche über x:1-Beziehungen hinweg

- **:1-Beziehungen** in Domänenklassen
  - wie **Attribute** vom Typ der Zielklasse behandelt

– Beispiel:

```
class Fakultaet {
 String name
 String kuerzel

 // Attribut hochschule vom Typ Hochschule
 static belongsTo=[hochschule:Hochschule]
}
```

definiert ein Attribut *hochschule* vom Typ *Hochschule*

- Zugriff auf Attribut des referenzierten Objekts mit der Punktschreibweise
  - *hochschule.id == hochschule\_id*

Attribut der referenzierten Hochschule

Attributwert in der Variablen *hochschule\_id*

## Where-Queries: Beispiel für die Fakultätssuche

```
def searchResults(Integer max, String name, String kuerzel, Long hochschule_id)
{
 params.max = Math.min(max ?: 10, 100)

 def query = Fakultaet.where {
 if (hochschule_id) {

 Attribut der Domänenklasse
 Formaler Methodenparameter

 name =~ (name ?: '%') && kuerzel =~ (kuerzel ?: '%') &&
 hochschule.id == hochschule_id
 } else {
 name =~ (name ?: '%') && kuerzel =~ (kuerzel ?: '%')
 }
 }

 def found=query.list(params)
 respond found, model:params+[fakultaetInstanceCount: found.size()]
}
```

## Where-Queries: Anmerkungen zum Beispiel

- oft heißen formale Parameter der Controller-Aktion gleich dem Domänenattribut
- wie **unterscheidet** man die in der Closure?
  - gar nicht, denn
    - **links** vom Vergleichsoperator steht **immer** ein **Attributname**
    - **rechts** vom Vergleichsoperator darf stehen
      - Wertliteral (konkrete Zahl, String in "", " oder //)
      - Variable mit dem Wert (im Beispiel formaler Methodenparameter)
      - Attributname
        - nur dann, wenn es keine gleichnamige Variable gibt (missingProperty)

## Where-Queries: Alternative Syntax mit findAll()

```
def searchResults(Integer max, String name, String kuerzel, Long hochschule_id) {
 params.max = Math.min(max ?: 10, 100)

 def found=Fakultaet.findAll(params)
 // Linksseitig: Suchattribut, rechtsseitig: Methodenparameter
 if (hochschule_id) {
 name =~ (name ?: "%") && kuerzel =~ (kuerzel ?: "%") &&
 hochschule.id == hochschule_id
 } else {
 name =~ (name ?: "%") && kuerzel =~ (kuerzel ?: "%")
 }
 }

 respond found, model:params+[fakultaetInstanceCount: found.size()]
}
```

## Where-Queries: Suche mit x:n-Beziehungen

- **:n-Beziehungen** in Domänenklassen
  - wie **Attribute** vom Typ der Collection der Zielklasse behandelt
  - Beispiel:

```
class Hochschule {
 String name
 String ort

 statichasMany = [fakultaeten:Fakultaet]
}
```
  - definiert ein Attribut *fakultaeten* vom Typ Collection von Fakultäten
- Anwendung: Prüfen der Elementanzahl
  - `def query = Hochschule.where { fakultaeten.size()>0 }`

## Where-Queries: Subqueries mit Aggregatfunktionen

- aggregieren Werte **aller** gefundenen Datensätze für Vergleiche

Methode	Bedeutung
min	Minimum aller Werte
max	Maximum aller Werte
avg	Durchschnitt aller Werte
sum	Summe aller Werte
count	Anzahl aller Werte
property	Eigenschaft der gefundenen Datensätze

- Vgl. SQL-Aggregatfunktionen
- Beispiele:
  - Finde alle Studenten, deren Note besser ist als der Durchschnitt:  
`def guteStudenten = Student.findAll { note<avg(note) }`
  - Finde besten Studenten:  
`def besterStudent = Student.find { note==min(note) }`

## Where-Queries: Aggregation mit Einschränkungen

- Aggregatbildung einer Subquery auf bestimmte Datensätze **einschränken**:
  - **of**-Methode auf Aggregatsfunktion aufrufen
  - Parameter der **of**-Methode: Closure
    - Subquery, die zur Aggregation heranzuziehende Datensätze sucht
  - Beispiel:

Durchschnittsnote  
aller Studenten, ...

... die bestanden  
haben

```
def besteStudenten = Student.findAll { note < avg(note).of { note <= 4.0 } }
```

## Where-Queries: Property-Subquery

- *property*-Methode
  - Vergleichsoperator erfolgt gegen anderen Datensatz der selben Tabelle
  - Beispiel:
    - Alle Hochschulen, deren Anzahl Studenten ...
    - ... größer ist als die Anzahl Studenten ...
    - ... der Hochschule Mannheim
  - `anzahlStudenten > property(anzahlStudenten).of { name == "HS Mannheim" }`

## Massen-Update und Massen-Delete mit Where-Klauseln

- Massen-Update:

- Query **definieren**:

```
def query = Fakultaet.where { name == "Informatik" }
```

- Massen-Update **ausführen**:

**Attributname**      **neuer Wert**

```
query.updateAll(name:"Computer Science")
// Benennt Informatik-Fakultäten in Computer Science um
```

- Massen-Delete:

- Query **definieren**:

```
def query = Fakultaet.where { name == "Informatik" }
```

- Massen-Delete **ausführen**:

```
query.deleteAll()
// Löscht Informatik-Fakultäten
```

## QueryByExample

- Suchattribute in **Beispielinstanz** eintragen

- `def example = new Fakultaet(name:'Wirtschaftsinformatik')`

- eingeschränkte Suchmöglichkeiten

- keine Vergleiche
    - keine Ähnlichkeitssuche
    - keine Suche nach Beziehungsattributen

- anhand **Beispielinstanz** suchen (Query by Example)

- **Einzelinstanz** suchen: `def result = Domänenklasse.find(example)`

- **mehrere Instanzen** suchen: `def results = Domänenklasse.findAll(example)`

- Beispiele:

- `def result = Fakultaet.find(new Fakultaet(name:'Wirtschaftsinformatik'))`  
liefert die **einzige** Fakultät, deren Name Wirtschaftsinformatik ist

- `def results = Fakultaet.findAll(new Fakultaet(name:'Informatik'))`  
liefert die **alle** Fakultäten, deren Name Informatik ist

- Criteria API
  - von Hibernate übernommen
  - domänenpezifische Sprache
    - ähnlich Where-Queries, aber Methoden statt Operatoren
    - unübersichtlichere Syntax
- HQL (Hibernate Query Language)
  - SQL-ähnliche Abfragesprache
  - Datenbankbefehle in Strings
    - fehlende Überprüfung durch Compiler
    - fehleranfällig



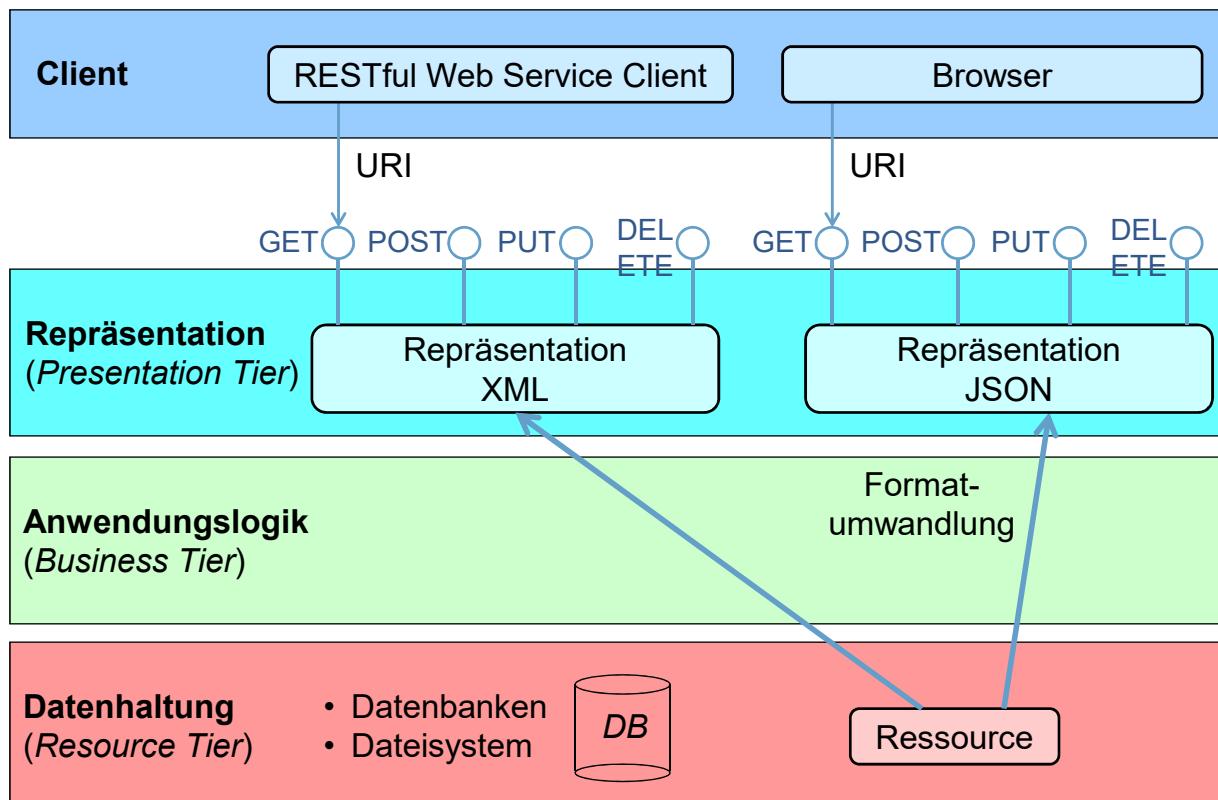
## Gliederung

3. Grails-Framework
  1. Einführung in Grails
  2. Controller
  3. Domänenklassen
- 4. RESTful Web Services**
5. GSP

# REST als Architekturstil

- REST: **Representation State Transfer**
  - Client-Server-basierter **Architekturstil**
  - formalisiert / verallgemeinert Architektur des WWW
  - im Jahr 2000 von Roy T. Fielding in seiner Dissertation entwickelt
- Server
  - **zustandslos**
  - bietet Clients **Repräsentation** von **Ressourcen** an
    - meist Dateien oder in Datenbank gespeicherte Domänenobjekte
    - Adressierung über eindeutige URI (Uniform Resource Identifier)
    - Client darf sich **Repräsentation** (Format) wünschen
      - Übermittlung im HTTP Header oder in URI
    - Server liefert dem Client die gewünschte Repräsentation der Ressource
- Client
  - adressiert Ressource per URI & wählt Repräsentation aus
  - Zustand (State) der Anwendung
    - festgelegt durch abgerufene Repräsentation einer Ressource ("Seite")

# REST-Architekturstil



## WWW setzt REST-Architekturstil um

- **URL** als Spezialisierung der URI
  - **URL**: Uniform Resource **Locator**
    - realer Speicherort, z.B. Dateiname im Webserver
    - einfacher Seitenabruf
  - **URI**: Uniform Resource **Identifier**
    - symbolischer Ressourcenidentifikator
    - meist kein Bezug mehr zu Speicherort oder Dateiname
    - moderne Webframeworks verwenden generell URI
      - z.B. Grails
- **GET**- und **POST**-Methode (HTTP-Verben)
  - GET zum Abrufen von Seiten und Ressourcen
  - POST zum Absenden (Speichern) von Ressourcen
  - PUT und DELETE im WWW normalerweise nicht genutzt
- **Repräsentation** im Original-Dateiformat
  - HTML, CSS, JS, GIF, JPG ...
  - meist keine Formatumwandlung

## RESTful Web Service setzt REST-Architekturstil um

- **URI** identifiziert gewünschte Ressource(n)
  - einzelne Ressource (z.B. Hochschule mit ID=1)
  - Menge von Ressourcen (z.B. alle Hochschulen)
- **Methoden** ("HTTP-Verben")

Methode	Beispiel-URI	Bedeutung		idem-potent	cache-bar	URL zeigt auf Res-source
		Ressource	Verarbeitung			
<b>GET</b>	/hochschule	≥0 abrufen		ja	ja	ja
	/hochschule/3	1 abrufen		ja	ja	ja
<b>POST</b>	/hochschule	1 anlegen	starten	-	-	-
<b>PUT</b>	/hochschule/3	1 updaten		ja	-	ja
<b>DELETE</b>	/hochschule/3	1 löschen		ja	-	ja

- **Repräsentation**
  - meist XML oder JSON
  - erst beim Abruf generiert

## RESTful Web Services vs. SOAP/WSDL Web Services

	<b>RESTful Web Service</b>	<b>SOAP/WSDL Web Service</b>
<b>Architekturstil</b>	ressourcenbasiert (REST)	Entfernter Methodenaufruf
<b>Philosophie</b>	einfach, high level	alle Fälle abdecken
<b>Transportprotokoll</b>	HTTP(S)	beliebig, meist HTTP(S)
<b>Repräsentation / Format</b>	beliebig, oft XML / JSON	XML
<b>Ressourcenidentifikation</b>	URI	- (Methodenparameter)
<b>Schnittstellen</b>	<b>Methoden</b>	GET, POST, PUT, DELETE
	<b>Parameter</b>	URI / HTTP-Header
	<b>Standards</b>	ja (feste HTTP-Verben)
	<b>Fehlerbehandl</b>	HTTP-Fehlercodes
	<b>Beschreibung</b>	- (feste HTTP-Verben)
<b>Performance</b>	+	- (hoher Protokolloverhead)
<b>Caching möglich</b>	GET-Aufrufe	-

## Repräsentationen von Ressourcen, Content Negotiation

- **Ressourcen** liegen oft vor als
  - strukturierte Daten in einer Datenbank
  - Objekte einer Programmiersprache
- Gewünschte **Repräsentation** wird erst zur Laufzeit beim Abruf erzeugt
- **Content Negotiation**
  - Aushandeln der Repräsentation zwischen Client und Server
  - Client
    - teilt mit, welche MIME-Format(e) er verarbeiten kann
  - Server beim Abruf der Ressource
    - Repräsentation der Ressource im gewünschten Format erstellen
    - via HTTP-Protokoll zum Client schicken

## URI zur Ressourcenidentifikation in Grails



- Aufbau ähnlich Grails-URI für Webanwendungen
  - **Ressourcenname** statt Controller und Controller-Aktion
    - standardmäßig: Ressourcenname = Domänenklassenname
    - Controller-Aktionen ersetzt durch HTTP-Verben
  - **Repräsentation** optional wählbar über
    - HTTP-Header
    - Formatendung `.fff` am Ende der URL, z.B.
      - `http://localhost:8080/WAF_Professor/professor.xml`
      - `http://localhost:8080/WAF_Professor/professor/1.json`

## Content Negotiation: Vordefinierte Grails Content-Typen

Grails Content-Typ*	MIME Content-Typ	Anwendung
all	<code>/*</code>	
atom	<code>application/atom+xml</code>	
css	<code>text/css</code>	Cascading Stylesheet
csv	<code>text/csv</code>	
form	<code>application/x-www-form-urlencoded</code>	HTML-Formular
html	<code>['text/html', 'application/xhtml+xml']</code>	HTML
js	<code>text/javascript</code>	JavaScript
json	<code>['application/json', 'text/json']</code>	JSON
multipartForm	<code>multipart/form-data</code>	
rss	<code>application/rss+xml</code>	
text	<code>text/plain</code>	ASCII-/ Unicode-Text
hal	<code>['application/hal+json', 'application/hal+xml']</code>	
xml	<code>['text/xml', 'application/xml']</code>	XML

\* definiert in `Config.groovy`

## 1. Annotation der Domänenklasse mit @Resource

- für Domänenklasse darf **keine** Controllerklasse existieren
  - **kein** statisches Scaffolding
  - **kein** dynamisches Scaffolding
- Controller wird **automatisch** erzeugt (ohne Quellcode)
  - vgl. dynamisches Scaffolding für Webanwendungen
- nur sinnvoll für RESTful Web Services **ohne** GUI

## 2. Controller durch **statisches** Scaffolding generieren

- Controller-Aktionen implementieren RESTful Web Service
- @Resource-Annotation in Domänenklasse wird dann **ignoriert**
- Controller MUSS also Rest selbst implementieren!

## 3. Controller manuell programmieren

- Resource-Annotation in Domänenklasse wird **ignoriert**
- Controller-Aktionen **müssen** Rest selbst implementieren!

## Variante 1: Annotierte Domänenklasse Hochschule

```
package restfakultaeten
```

```
import grails.rest.Resource
```

URI  
(Root: Projekt-URI)

Lieferbare Repräsentationen  
(Grails Content-Typen)

```
@Resource (uri="/resthochschulen", formats=['json', 'xml'])
```

```
class Hochschule {
```

```
 String name
```

```
 String ort
```

```
 static hasMany=[fakultaeten:Fakultaet]
```

Falls Client keine Repräsentation wählt:  
1. Listeneintrag ist Defaultformat

```
 static constraints = {
```

```
 name blank: false
```

```
 ort blank: true
```

```
 fakultaeten nullable:true
```

```
 }
```

```
}
```

## Attribute der Resource-Annotation

- **uri**
  - bezieht sich auf Projekt-Root `http://localhost:8080/Projektname`
  - sollte Plural des Domänenklassennamens sein
  - hier im Beispiel bewusst mal anderen Namen gewählt
  - Default: Domänenklassenname
- **formats**
  - spezifiziert Liste Formate, die der RESTful Web Service liefern kann
  - Formate sind Grails Content-Typen (vgl. nächste Folie)

## Lieferbare Repräsentationen

- vorkonfigurierte Grails Content-Typen in `Config.groovy`:

```
grails.mime.types = [// the first one is the default format
 all: '*/*', // 'all' maps to '*' or the first available format in withFormat
 atom: 'application/atom+xml',
 css: 'text/css',
 csv: 'text/csv',
 form: 'application/x-www-form-urlencoded',
 html: ['text/html','application/xhtml+xml'],
 js: 'text/javascript',
 json: ['application/json', 'text/json'],
 multipartForm: 'multipart/form-data',
 rss: 'application/rss+xml',
 text: 'text/plain',
 hal: ['application/hal+json','application/hal+xml'],
 xml: ['text/xml', 'application/xml']
]
```

- Automatische Erzeugung von XML und JSON über mitgelieferte Builder

## Annotierte Domänenklasse Fakultaet

```
package restfakultaeten

import grails.rest.Resource

@Resource (uri="/restfakultaeten", formats=['json', 'xml'])
class Fakultaet {
 String name
 String kuerzel
 static belongsTo=[hochschule:Hochschule]

 static constraints = {
 name blank: false
 kuerzel blank: false
 hochschule nullable:true
 }
}
```

## Datensätze füllen in Bootstrap.groovy (mangels GUI)

```
import restfakultaeten.*

class BootStrap {

 def init = { servletContext ->
 Hochschule hsma =
 new Hochschule(name:"Hochschule Mannheim",ort:"Mannheim")
 hsma.save()

 Hochschule hslu =
 new Hochschule(name:"Hochschule Ludwigshafen",ort:"Ludwigshafen")
 hslu.save()

 Fakultaet hsmalnf =
 new Fakultaet (name:"Informatik", kuerzel:"I", hochschule: hsma)
 hsmalnf.save()
 }

 def destroy = {
 }
}
```

## Aufruf mit Web Browser: Alle Hochschulen

The screenshot shows a web browser window with the title "The Grails Framework 2.4.4". The address bar displays "localhost / localhost / grail..." and the URL "http://localhost:8080/RESTFakultaeten/resthochschulen". The browser interface includes standard buttons like back, forward, and search.

The main content area displays the following JSON response:

```
[{"class": "restfakultaeten.Hochschule", "id": 1, "fakultaeten": [{"class": "restfakultaeten.Fakultaet", "id": 1}], "name": "Hochschule Mannheim", "ort": "Mannheim"}, {"class": "restfakultaeten.Hochschule", "id": 2, "fakultaeten": [], "name": "Hochschule Ludwigshafen", "ort": "Ludwigshafen"}]
```

Annotations with callouts point to specific parts of the JSON output:

- An annotation labeled "ID: 1" points to the "id" field in the first object.
- An annotation labeled "Fakultäten: I (ID=1)" points to the "fakultaeten" array in the first object.
- An annotation labeled "Name: Hochschule Mannheim" points to the "name" field in the first object.
- An annotation labeled "Ort: Mannheim" points to the "ort" field in the first object.

Wamp-Server mit mysql starten!

## Aufruf mit Web Browser: Hochschule Mannheim

The screenshot shows a web browser window with the title "The Grails Framework 2.4.4". The address bar displays "localhost / localhost / grail..." and the URL "http://localhost:8080/RESTFakultaeten/resthochschulen/1". The browser interface includes standard buttons like back, forward, and search.

The main content area displays the following JSON response:

```
{"class": "restfakultaeten.Hochschule", "id": 1, "fakultaeten": [{"class": "restfakultaeten.Fakultaet", "id": 1}], "name": "Hochschule Mannheim", "ort": "Mannheim"}
```

The status bar at the bottom of the browser shows the URL "https://jonathan.sv.hs-mannheim.de/mediawiki/index.php/Hauptseite".

# RESTful Web Service-Client Tools

- Kommandozeilentool: **curl**
  - mühsam zu bedienen
  - POST und PUT unter Windows kaum machbar
- Firefox Browser Plugin **HttpRequester**
  - komfortable GUI
  - Download unter <https://addons.mozilla.org/de/firefox/addon/httprequester/>
- **Achtung:** Grails ignoriert die *Accept* Header gängiger Browser
  - Grund:
    - ältere Browser-Versionen senden beim Websurfen falsche Accept-Listen
    - würde zur Darstellung von XML-Inhalten statt HTML führen
  - Abhilfe:
    - in *Config.groovy* folgende Zeile abändern:
    - *grails.mime.disable.accept.header.userAgent*s = [~~'Gecko', 'WebKit', 'Presto', 'Trident'~~]
    - Vollständiges Löschen der Zeile reicht **nicht**

## Aufruf mit HttpRequester: GET resthochschulen

The screenshot shows the HttpRequester plugin interface. On the left, the 'REQUEST' tab displays a URL of `http://localhost:8080/RESTFakultaeten/resthochschulen` and a 'GET' button highlighted with a blue box. On the right, the 'RESPONSE' tab shows the JSON data returned from the server. The JSON structure represents a collection of high schools, each with an ID, name, and location. Annotations highlight the 'Statuscode' (200 OK) and the 'alle Hochschulen' (all high schools) data. At the bottom, a red bar contains the command: `curl -i -X GET -H "Accept:application/json" localhost:8080/RESTFakultaeten/resthochschulen`.

```
[{"class": "restfakultaeten.Hochschule", "id": 1, "fakultaeten": [{"class": "restfakultaeten.Fakultaet", "id": 1}], "name": "Hochschule Mannheim", "ort": "Mannheim"}, {"class": "restfakultaeten.Hochschule", "id": 2, "fakultaeten": [], "name": "Hochschule Ludwigshafen", "ort": "Ludwigshafen"}]
```

`curl -i -X GET -H "Accept:application/json" localhost:8080/RESTFakultaeten/resthochschulen`

## GET resthochschulen/1

HttpRequester

**REQUEST**

URL: http://localhost:8080/RESTFakultaeten/resthochschulen/1

Method: GET

**RESPONSE**

Status: 200 OK

Content:

```
{ "class": "restfakultaeten.Hochschule", "id": 1, "fakultaeten": [{ "class": "restfakultaeten.Fakultaet", "id": 1 }], "name": "Hochschule Mannheim", "ort": "Mannheim" }
```

**HEADERS**

Server	Apache-Coyote/1.1
Content-Type	application/json;charset=UTF-8
Transfer-Encoding	chunked
Date	Wed, 07 Jan 2015 05:28:07 GMT

curl -i -X GET -H "Accept:application/json" localhost:8080/RESTFakultaeten/resthochschulen/1

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks 479

## GET restfakultaeten/1

HttpRequester

**REQUEST**

URL: http://localhost:8080/RESTFakultaeten/restfakultaeten/1

Method: GET

**RESPONSE**

Status: 200 OK

Content:

```
{ "class": "restfakultaeten.Fakultaet", "id": 1, "hochschule": { "class": "restfakultaeten.Hochschule", "id": 1 }, "kuerzel": "I", "name": "Informatik" }
```

**HEADERS**

Server	Apache-Coyote/1.1
Content-Type	application/json;charset=UTF-8
Transfer-Encoding	chunked
Date	Wed, 07 Jan 2015 05:31:13 GMT

curl -i -X GET -H "Accept:application/json" localhost:8080/RESTFakultaeten/restfakultaeten/1

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks 480

## GET resthochschulen/1.xml

HttpRequester

**REQUEST**

URL: http://localhost:8080/RESTFakultaeten/resthochschulen/1.xml

Method: GET

Content Type: application/xml

Content Options: Base64, Parameter Body

Content: Content

**RESPONSE**

Status: 200 OK

```
<?xml version="1.0" encoding="UTF-8"?>
<hochschule id="1">
 <fakultaeten>
 <fakultaet id="1" />
 </fakultaeten>
 <name>Hochschule Mannheim</name>
 <ort>Mannheim</ort>
</hochschule>
```

**Hochschule Mannheim**

**HISTORY**

```
curl -i -X GET -H "Accept:application/xml" localhost:8080/RESTFakultaeten/resthochschulen/1
```

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks 481

## Neue Hochschule anlegen: POST

HttpRequester

**REQUEST**

URL: http://localhost:8080/RESTFakultaeten/resthochschulen

Method: POST

Content Type: application/json

Content Options: Base64, Parameter Body

Content: Content

**RESPONSE**

Status: 201 Created

```
{
 "class": "restfakultaeten.Hochschule",
 "id": 3,
 "fakultaeten": [],
 "name": "Uni Mannheim",
 "ort": "Mannheim"
}
```

**Erstellte Ressource**

**JSON-Repräsentation der Ressource:**

```
{
 "class": "restfakultaeten.Hochschule",
 "id": 3,
 "fakultaeten": [],
 "name": "Uni Mannheim",
 "ort": "Mannheim"
}
```

**HEADERS**

- Server: Apache-Coyote/1.1
- Content-Type: text/xml;charset=UTF-8
- Transfer-Encoding: chunked
- Date: Wed, 07 Jan 2015 23:50:17 GMT

## Vorhandene Hochschule ändern: PUT resthochschulen/3

The screenshot shows the HttpRequester tool interface. In the REQUEST tab, the URL is `http://localhost:8080/RESTFakultaeten/resthochschulen/3`, the method is set to **PUT**, and the Content Type is `application/json`. The content body contains the JSON representation of the university to be updated:

```
{ "name": "Universität Mannheim" }
```

An annotation labeled **JSON-Format** points to the content body. Another annotation labeled **PUT** points to the method button. A large callout box labeled **JSON-Repräsentation der zu ändernden Attribute:** contains the JSON object.

In the RESPONSE tab, the status is **200 OK**. The response body shows the updated university record:

```
{ "class": "restfakultaeten.Hochschule", "id": 3, "fakultaeten": [], "name": "Universität Mannheim", "ort": "Mannheim" }
```

An annotation labeled **Geänderte Daten** points to the updated name in the response. The Headers section shows standard HTTP headers.

## Vorhandene Hochschule löschen: DELETE resthochschulen/3

The screenshot shows the HttpRequester tool interface. In the REQUEST tab, the URL is `http://localhost:8080/RESTFakultaeten/resthochschulen/3`, the method is set to **DELETE**, and the Content Type is `application/json`. An annotation labeled **DELETE** points to the method button. A large callout box labeled **kein Content nötig** (no content needed) points to the empty content body area.

In the RESPONSE tab, the status is **204 No Content**. An annotation labeled **Statuscode: Ressource gelöscht** (Status code: resource deleted) points to the status message.

The Headers section shows standard HTTP headers.

```
curl -i -X DELETE localhost:8080/RESTFakultaeten/resthochschulen/1
```

## Groovy RESTful Web Service Client

- **Client-Zugriff** auf RESTful Web Service
  - einfaches Groovy-Skript reicht
  - kein Grails nötig
  - benötigt *rest-client-builder* Plugin (oder dergleichen)
- **Client-Projekt :**
  - ganz normales **Grails-Projekt** erstellen
  - *rest-client-builder* Plugin hinzufügen
    - in **BuildConfig.groovy** gegen Ende die **grüne** Zeile einfügen:

```
plugins {
 // ...
 compile ':rest-client-builder:2.0.3' // Zeile hinzufügen
}
```
  - im Ordner *scripts*:
    - auf nächsten 3 Folien abgedruckten *FakultaetenClient.groovy* anlegen
  - Script *FakultaetenClient.groovy* starten
    - **nicht** das Grails-Projekt, denn das würde wieder einen Tomcat starten  
→ Portkonflikt auf Port 8080

## Groovy RESTful Web Service Client (1/3)

```
import grails.plugins.rest.client.RestBuilder

includeTargets << grailsScript("_GrailsInit")

target(restFakultaetenClient: "The description of the script goes here!") {

 // GET Datenstz mit ID=1
 String uriGET = "http://localhost:8080/RESTFakultaeten/resthochschulen/1"
 def response = new RestBuilder().get(uriGET) {
 contentType 'application/json'
 }
 println "Response: ${response.text}"
}
```

The diagram illustrates the execution flow of the Groovy code. It starts with the URI 'http://localhost:8080/RESTFakultaeten/resthochschulen/1'. This URL is passed to the 'get' method of the RestBuilder object. The 'get' method is annotated with 'Methode: GET'. Finally, the 'contentType' property is set to 'application/json', which is annotated with 'Content Negotiation'.

## Groovy RESTful Web Service Client (2/3)

```
// Füge neuen Datensatz ein (-->ID=3)
String uriPOST = "http://localhost:8080/RESTFakultaeten/resthochschulen"
new RestBuilder().post(uriPOST) {
 contentType 'application/json'
 json {
 name="Uni Mannheim"
 ort="Mannheim"
 fakultaeten: []
 }
}
```

Content (Rumpf) des HTTP Requests

```
// Ändere Datensatz mit ID=3
String uriPUT = "http://localhost:8080/RESTFakultaeten/resthochschulen/3"
new RestBuilder().put(uriPUT) {
 contentType 'application/json'
 json {
 name="Universität Mannheim"
 }
}
```

## Groovy RESTful Web Service Client (3/3)

```
// Löscht Datensatz mit ID=2
String uriDELETE = "http://localhost:8080/RESTFakultaeten/resthochschulen/2"
new RestBuilder().delete(uriDELETE) { }
```

```
// Listet alle Datensätze auf
String uriGETall = "http://localhost:8080/RESTFakultaeten/resthochschulen"
def responseAll = new RestBuilder().get(uriGETall) {
 contentType 'application/json'
}
println "ResponseAll: ${responseAll.text}"
```

```
}
```

```
setDefaultTarget(restFakultaetenClient)
```

## Groovy RESTful Web Service Client: Ergebnis

**Response:** {

```
"class":"restfakultaeten.Hochschule",
"id":1,
"fakultaeten":[{"class":"restfakultaeten.Fakultaet","id":1}],
"name":"Hochschule Mannheim",
"ort":"Mannheim"
}
```

**ResponseAll:** [

```
{ "class":"restfakultaeten.Hochschule",
 "id":1,
 "fakultaeten":[{"class":"restfakultaeten.Fakultaet","id":1}],
 "name":"Hochschule Mannheim",
 "ort":"Mannheim"
},
{ "class":"restfakultaeten.Hochschule",
 "id":3,
 "fakultaeten":[],
 "name":"Universität Mannheim",
 "ort":"Mannheim"
}
]
```

## RESTful Web Services Variante 2: Statisches Scaffolding

- per **statischem** Scaffolding generierte Controller
  - Controller-Aktionen implementieren RESTful Web Service wie folgt:

HTTP-Methode	URI	Controller Action
GET	/hochschule	index
	/hochschule/create	create*
POST	/hochschule	save
	/hochschule/1	show
GET	/hochschule/1/edit	edit*
	/hochschule/1	update
DELETE	/hochschule/1	delete

- keine manuelle Nacharbeit nötig
- Ressourcenname = Domänenklassenname
- `@Resource`-Annotation in Domänenklasse wird ignoriert

\* nur für Controller mit HTML-Schnittstelle nötig

## RESTful Web Services Variante 3: Manuelle Programmierung

- **manuell** programmierte Controller
  - Controller-Aktionen müssen RESTful Web Service selbst implementieren:

HTTP-Methode	URI	Controller Action
GET	/hochschule	index
	/hochschule/create	create*
POST	/hochschule	save
GET	/hochschule/1	show
	/hochschule/1/edit	edit*
PUT	/hochschule/1	update
DELETE	/hochschule/1	delete

\* nur für Controller mit HTML-Schnittstelle nötig

- @Resource-Annotation in Domänenklasse wird ignoriert

## RESTful Web Services Variante 3: Manuelle Programmierung

- Alternative: aus *RestfulController* erben und Aktionen bei Bedarf überschreiben

```
class HochschulController extends RestfulController {
 static responseFormats = ['json', 'xml']

 HochschulController() {
 super(Hochschule) // Konstruktor des RestfulController aufrufen
 }
}
```

- @Resource-Annotation in Domänenklasse wird ignoriert



## Gliederung

3. Grails-Framework
  1. Einführung in Grails
  2. Controller
  3. Domänenklassen
  4. RESTful Web Services
  - 5. GSP**

## Groovy Server Pages

- HTML-Seiten mit Groovy-spezifischen Erweiterungen:
  - **<%-- Kommentar --%>**
    - Kommentar, der vom Groovy-Interpreter ignoriert wird
    - gut zum Auskommentieren von Groovy-Code
  - **<% out<<"Hello GSP!" %>**
    - führt beliebigen Groovy-Code aus
    - Ausgabe in die HTML-Seite per out<<"Ausgabetext"
  - **<%= "Hello GSP!" %>**
    - fügt Wert eines Groovy-Ausdrucks in die HTML-Seite ein
    - besser: **\$()** verwenden
  - **\${ variablename }**
    - gibt Wert der Variablen **variablename** auf die HTML-Seite aus
    - Variablen = keys im von der Controller-Aktion gelieferten assoziativen Array
  - **\${ closurecode }**
    - gibt Wert auf die HTML-Seite aus, den der **closurecode** liefert
    - **closurecode** könnte z.B. ein Funktionsaufruf sein

## Groovy Server Pages: Internationalisierung: Konzept

- in dem GSPs
  - symbolische Properties statt hart eincodierter Texte
  - plus Defaultwert, wenn kein passendes Message Bundle vorhanden
- Beispiel:

```
<g:sortableColumn property="kuerzel"
title="${message(code: 'professor.kuerzel.label', default: 'Kuerzel')}" />
```

Text im Message  
Bundle nachschlagen

symbolischeProperty

Defaultwert, falls kein  
passendes Message Bundle  
vorhanden

- Message Bundles

- im Ordner Message Bundles
- pro Sprache eine Properties-Datei
  - Dateiname: messages\_de.properties

fester  
Dateiname

Ländercode,  
z.B. de

Dateiendung  
.properties

- Inhalt
  - pro symbolischer Property eine Zeile
  - Beispiel: professor.kuerzel.label=Kürzel

Prof. Dr. Thomas Specht: Webarchitekturen und -frameworks

495

## Groovy Server Pages: Int. mit parametrisierten Texten

- message-Methode im Detail

```
message(code: 'default.not.found.message', default: '{0} with id {1} not found',
args: ['Professor', 17])
```

aktueller Wert für  
Message-Parameter {1}

symbolischeProperty

Defaultwert, falls kein passendes  
Message Bundle vorhanden

formaler Message-  
Parameter

- steht auch als Tag in Grails Tag Library zur Verfügung:

```
<g:message code='default.not.found.message' default='{0} with id {1} not
found' args='["Professor", 17]'/>
```

- Deutsche Übersetzung in messages\_de.properties:

```
default.not.found.message={0} mit der id {1} wurde nicht gefunden
```

## Groovy Server Pages: Internationalisierung: Vorgehen

- Domain Classes, Controller und Views in Defaultssprache erstellen
  - in internationalen Projekten: Englisch
- alle hart encodierten Texte durch symbolische Properties ersetzen
  - in scaffolding-generierten Klassen standardmäßig schon der Fall
  - als Defaultwert jeweils den Text in der Defaultssprache angeben
- für weitere Landessprachen entsprechende Properties-Dateien erstellen

## Grails Tag Library

- Grails Tag Library
  - grundlegende Ablaufkonstrukte (Zuweisungen, Bedingte Abfragen, Schleifen)
  - Integration der HTML-Elemente
  - Komfortfunktionen (Paginator, Internationalisierung, Fehlermeldungen ...)
- Grails Tags
  - beginnen mit dem Prefix `g:`
  - in der *Grails Tag Library* definiert
  - automatisch importiert

## Grails Tag Library: Variablen und Gültigkeitsbereich

- Variablenwert setzen (Default-Gültigkeitsbereich: *page*)

```
<g:set var="i" expr="${id+1}" />
```

Variable

Ausdruck (zuzuweisender Wert)

- Variablenwert setzen mit Angabe des Gültigkeitsbereichs:

```
<g:set var="i" expr="${id+1}" scope="session" />
```

Gültigkeit (Scope)	Lebensdauer	Entsprechung / Verwendung	Anwendungsbeispiel
<b>page</b>	aktuell angezeigte GSP	Lokale Variable	Schleifeniterator
<b>request</b>	Abarbeitung dieses Requests	HTTP-Parameter (GET / POST)	Datensatz-ID
<b>flash</b>	bis zum nächsten Request	wird im Flash Scope abgelegt, beim nächsten Request abrufbar	Mitteilung an Benutzer, Fehlermeldung
<b>session</b>	gesamte Session	vgl. Session-Variablen in PHP	Benutzer-ID
<b>application</b>	Lebensdauer Web Container ( $\rightarrow$ solange Server läuft)	Globale Variablen, in allen Sessions sichtbar	Seitenabrufzähler

## Grails Tag Library: Bedingte Abfragen

- **<g:if test="\${condition}">**

```
...
</g:if>
```

- **<g:if test="\${condition}">**

```
...
</g:if>
<g:elseif test="${condition}">
...
</g:elseif>
<g:else>
...
</g:else>
```

## Grails Tag Library: Schleifen

- *foreach*-Schleife

```
<g:each in="${professorInstanceList}" var="professorInstance">
```

Zu durchlaufende Collection

Laufvariable für Collection-Element:  
Falls var-Attribut fehlt: *it*

```
 ${professorInstance.name}
```

```
</g:each>
```

- *while*-Schleife

```
<g:while test="${condition}" >
```

```
 ...
```

```
</g:while>
```

## Grails Tag Library: Suchen und Filtern einer Collection

- *foreach*-Schleife

```
<g:findAll in="${books}" expr="it.author == 'Graeme Rocher'">
```

Zu durchlaufende Collection

Filterkriterium

```
 <p>Title: ${it.title}</p>
```

```
</g:findAll>
```

- *grep*-Schleife

```
<g:grep in="${books.title}" filter="~/.*?Groovy.*?/">
```

Zu durchlaufende Collection + Attribut

Regulärer Ausdruck als Filterkriterium (Pattern-Operator)

```
 <p>Title: ${it}</p>
```

```
</g:grep>
```

## Grails Tag Library: Controller Action mit Link aufrufen

```
<g:link controller="professor" action="list">Alle Professoren auflisten</g:link>
```

- Attribute des *g:link* Tags:
  - *controller*: Controller, auf dem *action* aufzurufen ist
    - Controllername in Kleinschrift und ohne die Endung *Controller*
    - Defaultwert: eigener Controller
  - *action*: auf dem Controller aufzurufende *action*
    - Defaultwert: Default action, d.h. *index*
  - *id*: mit diesem Link zu übermittelnde Datensatz-*id*
  - *params*: assoziatives Array mit HTTP-Parametern (GET-Parameter an URL)
    - umschließende [] mit angeben!
  - *fragment*: Fragment (Anker) in der aufzurufenden Seite (per # in URL)
  - *url*: Assoziatives Array mit Attributwerten wie Controller, Action ..., z.B.  
`<g:link url="[action:'index',controller:'professor']">Professor List</g:link>`

## Grails Tag Library: Formular und Submit-Button

```
<g:form name="ProfessorForm" method="post" >
 <g:textField name="vorname" value="${professorInstance?.vorname}" />
 ...
 Beschriftung Aktion
 <g:actionSubmit value="Update" action="update"/>
</g:form>
```

- Attribute des *g:form* Tags:
    - *method*: GET, POST, PUT oder DELETE
    - weitere wie *g:link*
  - Attribute des *g:actionSubmit* Tags:
    - *value*: Beschriftung des submit-Buttons
    - *action*: auf dem Controller aufzurufende *action*  
Defaultwert: *value* legt auch *action* fest
  - Anmerkungen:
    - Formular kann mehrere Submit-Buttons haben mit
      - auf selbem Controller
      - unterschiedliche Actions
- *actionSubmit*-Button wirkt wie ein Methodenaufruf (=Controller-Action)

## Grails Tag Library: Eingabefelder

- Einzeiliges Textfeld:

```
<g:textField name="vorname" value="${professorInstance?.vorname}" />
```

Feldname  
→ Key in params

Initialisierungswert  
(Default: leer)

- Passwortfeld:

```
<g:passwordField name="myPasswordField" value="${myPassword}" />
```

- Hidden-Feld:

```
<g:hiddenField name="myField" value="myValue" />
```

- Mehrzeiliges Textfeld

```
<g:textArea name="bemerkungen" value="myValue" rows="5" cols="40" />
```

- Weitere Attribute:

– *required="true"* Feld darf nicht leer sein, Prüfung in JavaScript

– *readonly="true"* Nurlesefeld

## Grails Tag Library: Checkboxen und Radio Buttons

- Checkbox:

```
<g:checkbox name="myCheckbox" value="${true}" />
```

Feldname  
→ Key in params

Initialisierungswert

- Radio Buttons:

```
<g:radio name="geschlecht" value="m"/>
```

```
<g:radio name="geschlecht" value="w" checked="true"/>
```

Feldname  
→ Key in params

Auswahlwert

angekreuzt  
oder nicht

## Grails Tag Library: Auswahllisten

- Auswahlliste  
`<g:select name="age" from="${18..65}" noSelection="[':-Choose your age-']"/>`
- Attribute des g:select Tags:
  - `name`: Feldname → Key in params
  - `from`: Collection mit den Auswahlwerten der Select-Liste
  - `keys`: Collection mit den Absendewerten der Select-Liste
    - gleiche Reihenfolge der Auswahloptionen wie `from`-Attribut
    - Defaultwert: gleiche Werte wie `from`-Attribut
  - `noSelection`: **Assoziatives Array** mit einem Element
    - Key: Abzusendender Wert (meist ", also leerer String)
    - Value: Anzuzeigender Text
  - `multiple`: Wenn gesetzt: aufgeklappte Mehrfachauswahlliste

## Grails Tag Library: Sortierbare Tabellen (1/2)

```
<table>
<thead>
<tr>
 <g:sortableColumn property="vorname" title="Vorname" />
 <g:sortableColumn property="nachname" title="Nachname" />
 <g:sortableColumn property="kuerzel" title="Kuerzel" />
 ...
</tr>
</thead>
<tbody>
<g:each in="${professorInstanceList}" status="i" var="professorInstance">
 <tr class="${(i % 2) == 0 ? 'even' : 'odd'}">
 <td><g:link action="show" id="${professorInstance.id}">
 ${fieldValue(bean: professorInstance, field: "vorname")}</g:link></td>
 <td>${fieldValue(bean: professorInstance, field: "nachname")}</td>
 <td>${fieldValue(bean: professorInstance, field: "kuerzel")}</td>
 ...
 </tr>
</g:each>
</tbody>
</table>
```

## Grails Tag Library: Sortierbare Tabellen (2/2)

- Funktionsweise:
  - jede Spalte bekommt Link als Spaltenkopf
  - Anklicken des Spaltenkopfes
    - wählt Spalte als Sortierkriterium aus
      - zunächst aufsteigend sortiert (asc), bei erneutem Klicken absteigend (desc)
    - löst die Controller-Action erneut auf
      - zusätzliche URL-Parameter: `sort=kuerzel&&order=asc`
  - Controller-Action muss Datensatzliste sortiert bereitstellen
    - Trick: `list(params)` auf Domänenklasse aufrufen
      - verarbeitet die Parameter `kuerzel` und `order` vollautomatisch
      - kein manuelles Sortieren nötig

Sortierattribut

Sortierrichtung

## Grails Tag Library: Paginator

- ```
<g:paginate total="${professorInstanceCount ?: 0}" max="2" />
```
- Stellt Seitenweises Blättern durch die Ergebnisliste zur Verfügung
 - Funktionsweise:
 - falls Anzahl Datensätze $\leq max$
 - alle Datensätze auf eine Seite
 - keine Links zum Durchblättern
 - falls Anzahl Datensätze $> max$
 - Paginator als Links in HTML-Seite (Vorherige, Seitennummern, Nächste)
 - Vorherige: Wechselt auf vorherige Seite (sofern vorhanden)
 - Seitennummern: Wechselt auf entsprechende Seite
 - Nächste: Wechselt auf nächste Seite (sofern vorhanden)
 - löst die Controller-Action erneut auf
 - zusätzlicher URL-Parameter: `offset=4&&max=2`
 - Controller-Action gewünschte Datensatzliste bereitstellen
 - Trick: `list(params)` auf Domänenklasse aufrufen
 - verarbeitet die Parameter `offset` und `max` vollautomatisch
 - kein manuelles Selektieren nötig

offset=4 überspringt die
Datensätze 1-4, startet mit dem 5.

Anzahl dargestellte
Datensätze

Grails Tag Library: Paginator-Attribute (1/2)

- total:
 - Gesamtanzahl Datensätze
 - daraus wird Anzahl benötigter Seiten berechnet: aufrunden (total/max)
 - falls Datensätze auf einer Seite unterkommen: Paginator nicht darstellen
- max
 - maximale Anzahl Datensätze pro Seite
 - nur verwendet, wenn *params.max* leer
 - *params.max* bindet also stärker als das Attribut *max*
- offset
 - Offset, ab dem die Datensätze dargestellt werden
 - nur verwendet, wenn *params.offset* leer
 - *params.offset* bindet also stärker als das Attribut *max*
- controller
 - Controller, der bei Benutzung der Paginator-Links gerufen wird
 - Default: gleicher wie aktueller (fast immer sinnvoll)

Grails Tag Library: Paginator-Attribute (2/2)

- action
 - Controlleraction, der bei Benutzung der Paginator-Links gerufen wird
 - Default: gleiche wie die aktuell angezeigte (fast immer sinnvoll)
- params
 - Weitere Parameter für die aufzurufende Controller-Action
- prev:
 - Text für den Previous-Link
 - Default: *default.paginate.prev* Property im Message Bundle
- next:
 - Text für den Next-Link
 - Default: *default.paginate.next* Property im Message Bundle

Fields Plugin zur Erleichterung der GSP-Erstellung

- automatische Auswahl des gerenderten HTML-Elements: `<f:field ...>`
 - abhängig vom Attributtyp der Domänenklasse
 - Beispiel: Booleans mit `g:checkbox` rendern
- automatische Darstellung **aller** Domänenklassenattribute mit `<f:all ...>`
 - zusätzliche Domänenklassenattribute erfordern keine Anpassung der GSPs
- Verkleinerung der (per Scaffolding generierten) GSP-Dateien

Fields Plugin: `<f:field ...>`

- `<f:field bean="hochschule" property="name"/>`
 - Variable mit Domänenklasseninstanz
 - Attributname in Domänenklasse
- Wirkung
 - Domänenklasse = Datentyp der Domänenklasseninstanz-Variable
 - Label = Attributname (groß geschrieben)
 - zu renderndes HTML-Element aus Property-Datentyp ableiten
 - Erzeugt Label und Eingabefeld auf HTML-Seite
- Rendering anpassbar durch Templates

<f:field ...>: Wichtige Attribute

| Attributname | muss? | Default | Beschreibung |
|--------------|-------|-------------------|-----------------------------|
| bean | ja | - | Domäneninstanzvariable |
| property | ja | - | Name Domänenklassenattribut |
| required | - | → blank(nullable) | Muss-/Kann-Attribut |
| label | - | → Attributname | Feldbeschriftung |

- alle weiteren Attribute werden an das Rendering-Template durchgereicht

Fields Plugin: <f:all ...>

- <f:all bean="hochschule"/>
Variable mit
Domänenklasseninstanz
- Wirkung
 - Domänenklasse = Datentyp der Domänenklasseninstanz-Variable
 - alle Attribute der Domänenklasse mit <f:field ...> rendern (inkl. Labels)

<f:all ...>: Wichtige Attribute

| Attributname | muss? | Default | Beschreibung |
|--------------|-------|---------|--|
| bean | ja | - | Domäneninstanzvariable |
| except | - | "" | Kommagetrennte Liste vom Rendering auszuschließender Attribute |

- Attribute *id*, *version*, *dateCreated* und *lastUpdated* grundsätzlich nicht rendern
- alle weiteren Attribute werden an das Rendering-Template durchgereicht

Fields Plugin: <f:with ...>

- <f:with bean="hochschule">
Variable mit
Domänenklasseninstanz
 - <f:field property="name"/>
 - <f:field property="ort"/>
 - <f:field property="plz"/>
- </f:with>
- gibt *bean*-Property für alle eingebetteten <f:field ...> vor
 - hilfreich, wenn viele Attribute einer Domänenklasse auf die HTML-Seite sollen