

## Servlets

- Java Programmcode der HTML Code erzeugt (println)
- Läuft im Servlet Container

## Servlet Container

- Ablaufumgebung für Servlets
- Z.B. Groovy & Grails

## Serverseitige Skriptsprachen

- Vorteile Groovy Grails zu den anderen
  - Umfang & Vielseitigkeit der Programmiersprache sehr gut
  - Einfache Syntax
  - Dynamische Typisierung
  - Typsicherheit
  - Frameworks
  - Performance
  - Tests

## Zusammenspiel Groovy Java

- Kann Grundsätzlich genau so ablaufen
  - Hat aber zusätzlich den Groovy Interpreter, welche die Schritte Bytecode & Classloader auslässt und direkt eine geladene Klasse erzeugt
- Java in Groovy einbinden
  - Java compilieren und .class einbinden

## Groovy Vereinfachung/Erweiterung zu Java

- Alles sind Objekte (int, double ..)
  - Dynamisch typisierbar als **def**
  - Statisch typisierbar z.B. **int**
- Einfache Nutzung von Collections
- Verzicht auf unnötige Syntax (; , Main Funktion)
- Operatoren überladen überschreiben
  - Collections mit + erweiterbar
  - Für Vektoren oder Matrizenberechnung
- Closures
- Metaprogrammierung
- Assoziative Arrays [key:value]
  - Einfache Syntax
  - Zugriff über Index

## Funktionsdefinition

- Weitestgehend wie in Java
  - Es können aber Default-Werte gesetzt werden z.B. def max(x, y=0)
- Alles Public, wenn nicht weiter angegeben
- Funktionen können außerhalb von Klassen definiert werden -> globale Funktionen
- Klammerentfall bei Aufruf parameterloser Funktionen

## Closures

- Ist ein Objekt vom Typ Closure, kann an Funktionen übergeben oder zurückgegeben werden
- Anwendung
  - Sinnvoll für Asynchrone Aufrufe
  - GSP um G-String zu füllen
  - Liste mit crep-Methoden filtern
- "It" bei keinem oder einem Parameter
- In Java -> Lambdas
- Unterschied zu Methoden
  - Nur über Multimethoden überladbar
  - Funktion zu Closure
    - this.&funktionsname
- Rekursive Closure
  - ```
def fakult
    fakult = { val ->
        val>1 ? val*fakult(val-1) : 1
    }
    println fakult(10)
```

## Operatoren

- Compare und Equals entfallen da mit ==; <; >; >= abgedeckt
- "as" Operator als Datentypumwandlung - selbst definierbar

## Stringliterale

- Mit '
  - Nicht G-String fähig
  - Bei drei "" Mehrzeilig
- Mit "
  - G-String fähig

- Bei """" Mehrzeilig
    - Mit /
      - G-String fähig
  - Slashy-String für Pattern, ansonsten müsste / immer escaped werden
  - G-String
    - Jeder String mit " oder / ist ein G-String, sobald er ein \$-Zeichen enthält, welches nicht escaped ist oder das letzte Zeichen ist
  - Mit Strings kann man rechnen
    - Vorname = "Thomas"
    - Vorname -= "oma"
    - Vorname -> "Ths"
  - Bringt Methoden mit
    - Contains
    - Reverse
    - Size
  - Besitzt Indizes

```
println "Thomas"[3] // gibt 4. Zeichen (m) aus
println "Thomas"[2..4] // gibt 3.bis5.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
```
  - --> Ist jedes Java Programm ein gültiges Groovy Programm?
    - Nicht immer, z.B. Strings und Character
      - Z.B. \$-Zeichen im G-String

## Collections

- Elemente mit + hinzufügen ; - wegnehmen
  - Genauer Zugriff über Indizes
    - Einfügen, überschreiben von Werten
  - FindAll
    - Liefert alles Ergebnisse
  - Find
    - Liefert ein Ergebnis
  - Every
    - Alle Elemente erfüllen Kriterium
  - Any
    - Mindestens ein Element erfüllt Kriterium
  - Assoziative Arrays
    - def profs= [SPE:"Thomas Specht", SMI:"Thomas Smits", KNU:"Peter Knauber"]
  - Ranges

- 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]
```

- Verbesserung gegenüber Java
  - Mit + aneinanderhängen
  - Per Index rückwärts iterieren
  - Problemlos vergrößern (Java Array -> umkopieren)

## Groovy Beans

- Erfüllt Konventionen
- Es gibt keine inneren Klassen
  - Closures ersetzen anonyme innere Klassen
  - Hilfsklassen in der selben Datei
- Groovy Beans = Groovy Klassen
  - Mit parameterlosen Konstruktor
  - Getter Setter für alle Attribute
  - Serializable
- Direkter Attributzugriff -> da grundsätzlich alles public
  - Ruft immer definierten Getter/Setter auf
  - Zum Schutz von Attributen diese private setzen und Getter/Setter implementieren
- Wenn kein anderer Konstruktor definiert ist:
  - impliziter Konstruktor als HashMap mit benannten formalen Parametern für alle Attribute:
  - Syntax: Klassename(attrName1: attrValue1, attrName2: attrValue2)
- Dynamischer Zugriff auf Attribute oder Methoden
  - ```
def attr="vorname"
      println prof."$attr"
```
  - ```
def mn="setKuerzel"
      prof."$mn"("Jens")
```
- Sicheres Dereferenzieren mit ?
  - if (knauber?.vorname)
- Expando
  - Ist dynamisches Objekt vom Typ Closure
  - Beliebig Attribute hinzufügen

## Reguläre Ausdrücke

Klausur z.B.

Kreditkarte, Raumnummer

Pattern immer im /Slashy String/ angeben!

Dient zur Überprüfung von Nutzereingaben z.B. Email im Formularfeld

- Match-Operator
  - Result = testString ==~ pattern
  - Testet kompletten String auf Einhaltung eines regulären Ausdrucks
    - ```
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)
```

- Find-Operator
  - analysiert und zerlegt String anhand eines regulären Ausdrucks
  - Syntax: ergArray= testString =~ pattern
- Capture Group
  - Jede () im Pattern
  - Zugriff über Zweidimensionales Array

## 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

- Pattern-Operator
  - matcher = ~/pattern/
  - parst und compiliert pattern, um künftige Zugriffe zu beschleunigen

## Metaprogrammierung

- Fähigkeit von Programmen (während der Laufzeit)
  - Sich selbst du analysieren
  - Sich selbst zu erweitern
- Einsatzgebiet
  - GORM Methoden z.B. `findByName`
    - Werden zur Laufzeit generiert
    - Kein unnötiger Quellcode
- Vorteile:
  - Ergänzung vorhandener Klassen um
  - neue Methoden

- Konverter und Rechenoperationen zu selbst definierten Datentypen
  - Selbstmodifizierender Code möglich
- Nachteile:
  - geringere Performance
- MetaClass: Dynamisch etwas zur Klasse hinzufügen
  - Dynamisch Attribut neuAttr an MeineKlasse hinzufügen:
    - MeineKlasse.metaClass.neuAttr = initWert
  - Genauso für Methoden und Konstruktoren
- Intercept-Cache-Invoke Pattern
  - Warum: Methoden die fehlen werden während der Laufzeit als Closure hinzugefügt.
    - Zukünftige Aufrufe gehen über neue hinzugefügte Methode
  - Intercept:
    - Abfangen des Aufrufes einer Methode die nicht vorhanden ist
  - Cache:
    - Methode wird per Metaprogrammierung nachgerüstet
  - Invoke:
    - Alle weiteren Aufrufe nutzen die neu erzeugte Methode
- Aspektorientierte Programmierung mit Groovy
  - GroovyInterceptable Interface
    - Vor und nach Methodenaufruf eigenen definierten Code ausführen
- Builder
  - Was ist das?
    - Erstellt aus Domänenklasse-Objekt ein XML
  - Warum?
    - Anstatt html will ich XML/JSON als Endgerät
  - Gegenstück
    - Parser
      - Mit RegEx und Capturegroup wird aus XML ein Objekt erzeugt

## Testen in Groovy

- Wie J-Unit
- Assert boolescheBedingung
  - Stellt sicher das Bedingung true ist, ansonsten Programmabbruch

## ##### Grails Framework

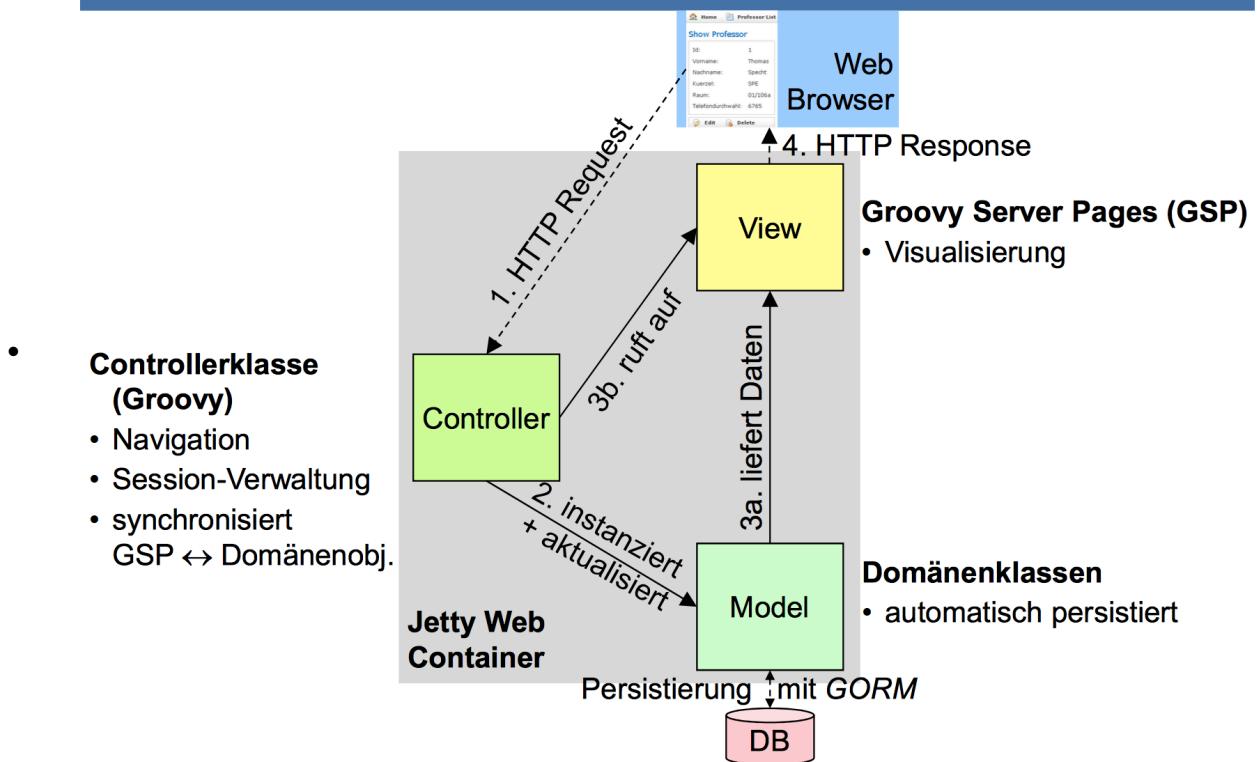
#####

## Einführung in Grails

- Ist Framework für die Präsentationsschicht
  - Liefert vollständige Webseiten
  - In sich selbst MVC
- Convention over Configuration

- Fest vorgegebene Ordner
- Minimaler Konfigurationsaufwand
- Don't repeat yourself
  - Vermeidung unnötiger Coderedundanzen
  - Automatische Generierung aus Domänenklassen
- Agilität
  - Wahlweise statische oder dynamische Typisierung
  - Agile Webentwicklung
  - Rapid Prototyping
  - Einfach zu testen
- Open Source
- MVC
  - Model: Domänenklassen mit automatischer Persistenz (GORM / Hibernate)
  - View: Groovy Server Pages (GSP), typischerweise HTML-basiert
  - Controller: Eigene Instanz für jeden Request
- Scaffolding
  - Statisches
    - Quellcode für Controller und Views aus Domänenklasse generiert
  - Dynamisches
    - Controller und Views während der Laufzeit erzeugen - ohne Quellcode

## MVC in Grails



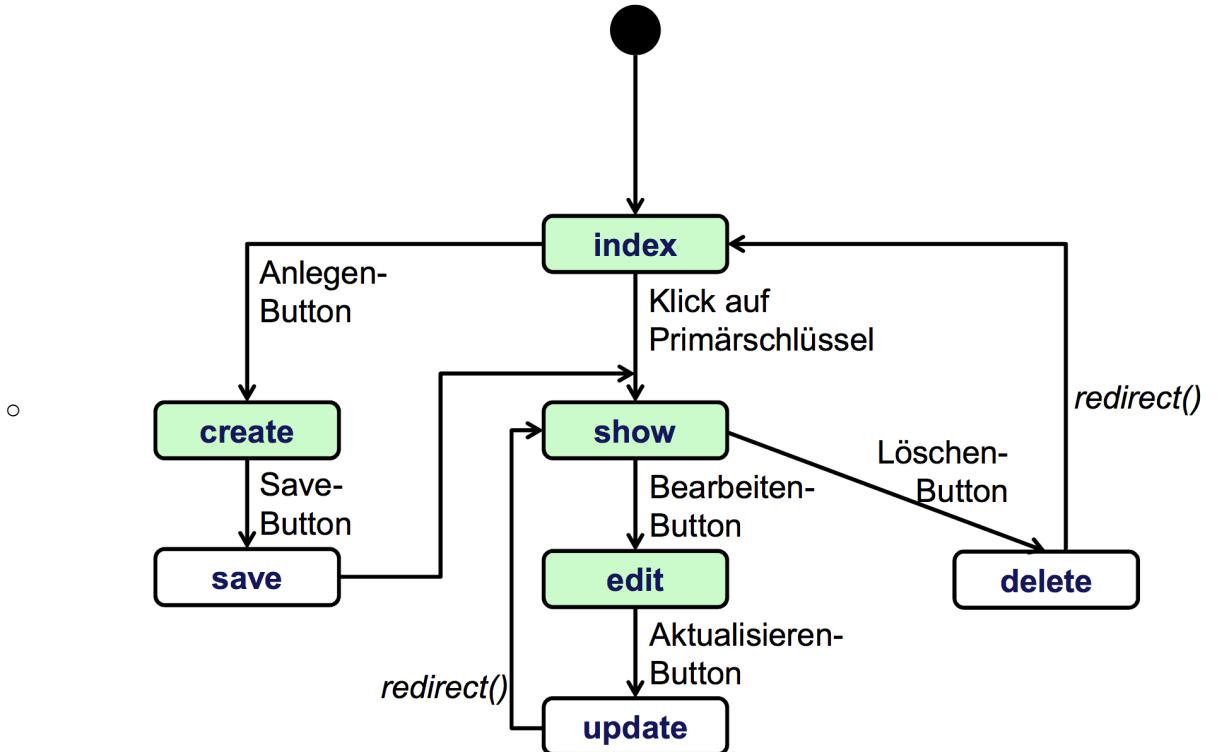
## Domänenklasse

- Angabe von Datentypen -> statische Typisierung
- Grundlage der Persistierung in Grails
  - Pro Domänenklasse eine Datenbanktabelle
  - Instanzen entsprechen Zeilen
  - Attribute entsprechen Spalten

- Realisierung als Groovy Bean
  - Ganz normale Groovy Klasse
- Constraints für Attribute
  - Nullable
  - Blank
- GORM-Methoden
  - Automatisch dynamisch bereitgestellt, ohne Quellcode

## Controller

- Steuert Webseite zu einer Domänenklasse
  - Navigationslogik
  - Enthält Controlleraktionen
  - Schnittstelle zur Anwendungslogik
  - Steuert Persistierung
- Realisierung als Groovy Klasse
- 3-Möglichkeiten zur Erzeugung
  - dynamischesScaffolding
    - dynamische Generierung zur Laufzeit
    - kein Quellcode erzeugt, nicht änderbar
      - Quellcode beinhaltet nur Zuordnung zur Domänenklasse
    - Anwendung: typische CRUD-Oberflächen, z.B. Administrator-Oberflächen
  - statischesScaffolding("Gerüstbau")
    - statische Generierung als Quellcode aus der Domänenklasse
    - beliebig änder- und erweiterbar
    - Anwendung: CRUD-basierte, komplexere Oberflächen
    - Zugehörige Views als Quellcode generieren
  - manuelleProgrammierung
    - Quellcode manuell erstellen
    - höherer Programmieraufwand
    - Anwendung: Oberflächen abseits der üblichen CRUD-Funktionalität
- Standard-Navigationslogik



- Controlleraktionen

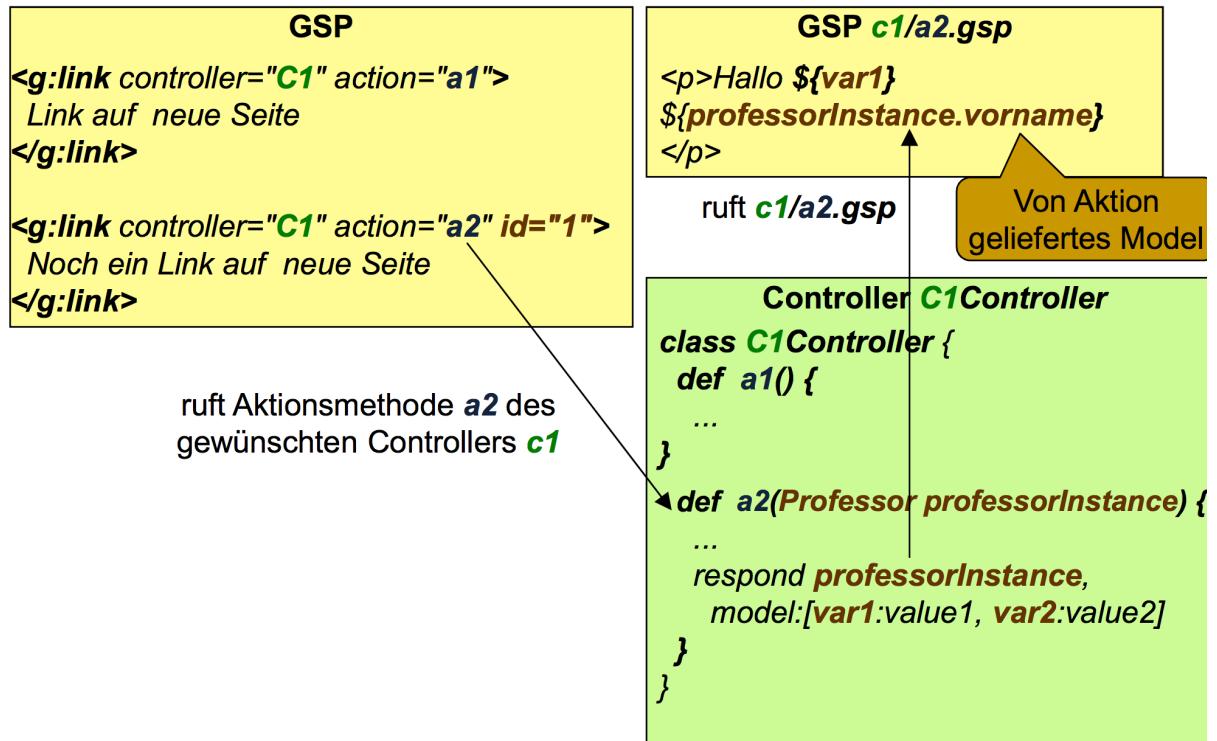
Aktion	zugehöriger View	Bedeutung
<i>index</i>	<i>index.gsp</i>	Einstiegsseite, listet alle Instanzen (Datensätze) der Domänenklasse auf
<i>show</i>	<i>show.gsp</i>	zeigt einzelne Instanz im Detail an
<i>delete</i>	-	löscht einzelne Instanz aus Datenbank
<i>edit</i>	<i>edit.gsp</i>	zeigt einzelne Instanz 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

- Erlaubte HTTP-Methoden

- static allowedMethods = [save: "POST", update: "PUT", delete: "DELETE"]
- Welche Controlleraktion hat welche http Methode

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

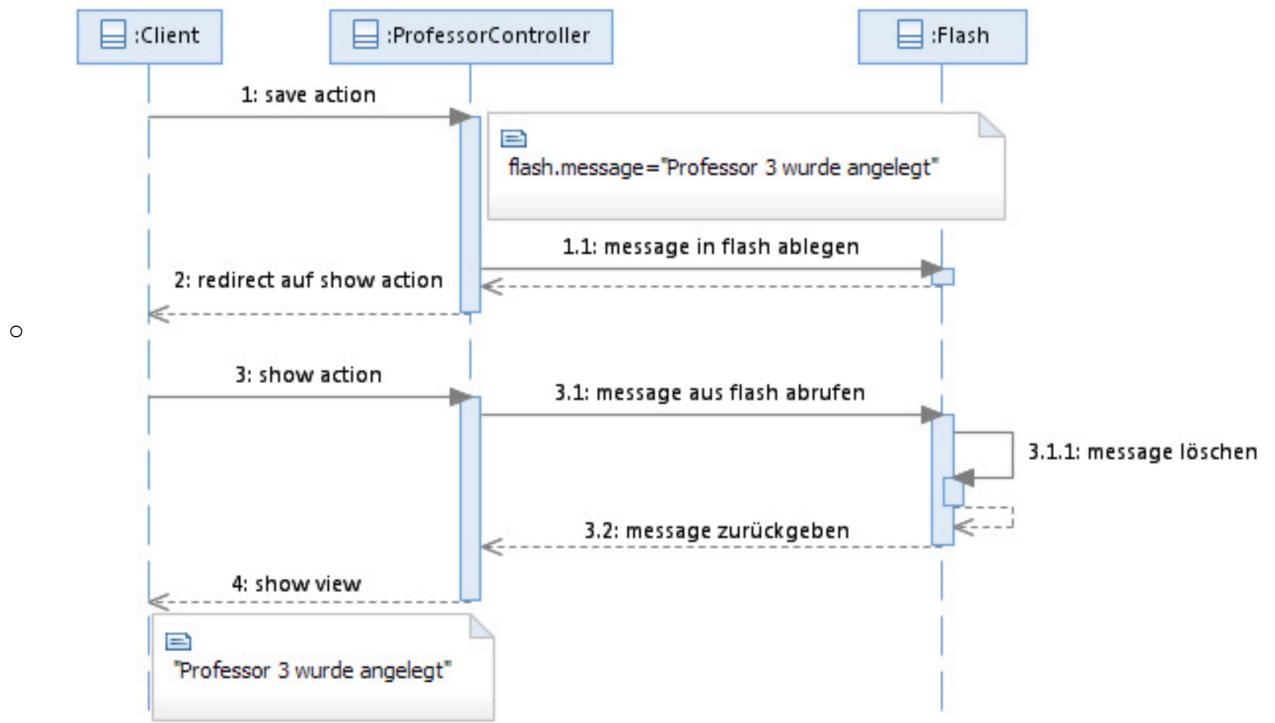
- o Default: alle sind erlaubt
- o Sinvoll: Get für Änderungsaktionen (save, update, delete) verbieten
  - Erschwert URL-Manipulation
  - Verhindert Mitlesen von Parameterwerten auf URL
- Aufruf von Controller-Aktion aus .gsp
  - o Method="DELETE" -> wenn Action fehlt: Aus HTTP-Methode ableiten
  - o Action="save"
- Transaktionssteuerung
  - o Transaktionsbeginn: Beim Aufruf der Controller-Aktion
  - o Transaktionsende: beim Verlassen der Controller-Aktion
  - o Transaktionsarten:
    - Vollwertige Schreibsese-Transaktionen (read/write)
      - Annotiert mit @Transactional
    - Nur lesende Transaktionen (read-only)
      - @Transactional (readOnly = true)
- Zusammenspiel Controller <-> gsp



- Parameterübergabe View <-> Controlleraktion
  - Alle HTTP-Parameter in Umgebungsvariable *params*
  - Erlaubte Parameter
    - Elementare Datentypen + deren Wrapper
    - String
  - Automatische Bereitstellung der Domäneninstanz für id-Parameter
    - Per /idValue an URL angehängt
    - Per ?id=idValue (GET-Parameter)
  - Methodensignatur parameterlos
    - **Beispiel:**  
`def create() {`  
 `respond new Professor(params)`  
 •     }
    - **Aufruf:**  
`http://localhost:8080/WAF_Professor/professor/create`
  - Methodensignatur mit HTTP-Parametern
    - **Beispiel:**  
`def hallo(String vorname, String nachname) {`  
 `...`  
 •     }
    - **Aufruf:**  
`http://localhost:8080/WAF_Professor/professor/hallo?vorname=Thomas&nachname=Specht`
  - Sessionvariable
    - Wird automatisch erzeugt

- Hält Informationen des aufrufenden Clients
- Nutzung: Client identifizieren
- Flash-Objekt
  - Automatisch erzeugt
  - Liefert eine Message an die nächste GSP
  - Wird danach automatisch gelöscht

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



- Logging
  - Von Informationen, Warnungen und Fehlermeldungen
    - Per Logger in LogDatei oder Datenbank umleitbar
    - Appender: Ausgabekanal für Log-Nachrichten

### 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)

- Technische Basis **Logback**
  - Umgebungsvariable **log** steht jedem Controller zur Verfügung

## 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

- Logger Konfiguration in lockback.groovy
    - Logging level dort anpassbar

## 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><br><code> \${nachname}</code> |
| <code>render view:"list", model:[professorList: Professor.list()]</code> | model-Parameter (assoziatives Array)        | <code>list.gsp</code>      | <code> \${professorList[0]}</code>                      |
| <code>render Professor.list() as XML</code>                              | Professorenliste                            | Profes-<br>soren-<br>liste | XML-<br>Format  |
| <code>render Professor.list() as JSON</code>                             |   |                            | JSON-<br>Format   |

- Parameterrückgabe Controller: Model + View

- Beispiel:

- Controller:

```
class Professor {
    // ...
    def index() {
        render view:"list", model:[professorList: Professor.list()]
    }
    // ...
}
```

- Aufgerufene View: **list.gsp**

- Zugriff auf Professorenliste mit  `${professorList}`

- Rendering der Antwort
  - Im Code
    - Render `Professor.list()` as XML/ as JSON
- Content Negotiation
  - Endgerät teilt gewünschtes Format mit
  - Controller-Aktion versucht Wunsch zu erfüllen

### Content Negotiation: Vordefinierte Grails Content-Typen

Grails C.-Typ*	MIME Content-Typ	Anwendung
all	*/*	
atom	application/atom+xml	
css	text/css	Cascading Stylesheets
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`

- **Respond** muss am Ende der Controller-Aktion stehen
- `Request.withFormat` ermöglicht Fallunterscheidung nach Grails-Content-Format

- 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

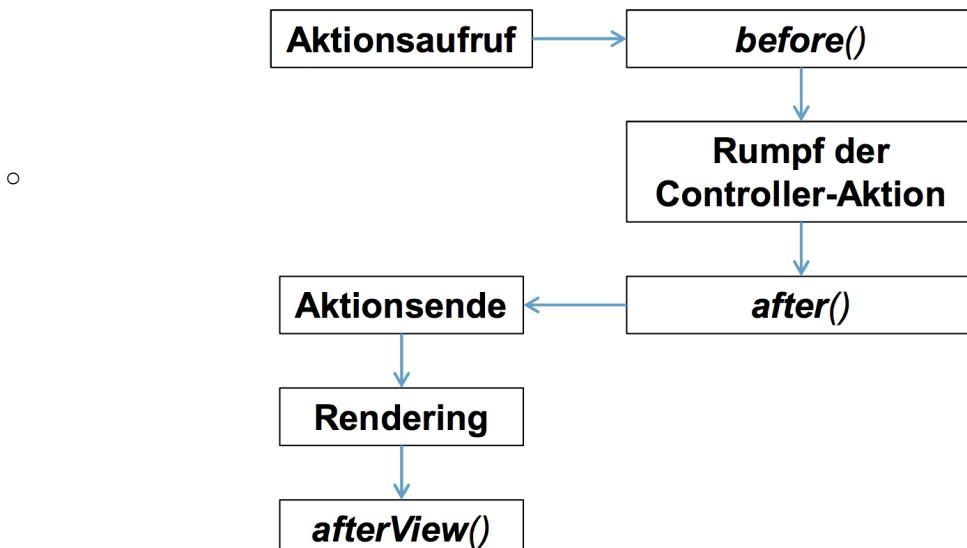
  
`redirect(controller:"Vorlesung", action:show, params:[hochschuleId:params.id])`

- Umgeleitet wird immer auf Aktion eines Controllers, niemals direkt auf eine 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



- Interceptor erstellen

- Create-interceptor Domänenklasse
- Z.B. create-interceptor professorenverwaltung.Professor

- Boolean *before()*

- Wir **vor jeder** Controller-Aktion ausgeführt
- Rückgabewert entscheidet über Fortsetzung der Controller-Aktionen

- Z.B. für
  - Login
  - Überprüfung Parametereingaben
- Boolean after()
  - Wir **nach jeder** Controller-Aktion ausgeführt
  - Rückgabewert entscheidet ob View gerendert wird oder nicht
  - Z.B.
    - Überprüfung und Manipulation der Antwort der Aktion
- Void afterView()
  - Wir nach dem Rendern der View aufgerufen, vor dem Layouten
  - Kein Rückgabewert
- Unbedinger Interzeptor
  - Kein Konstruktor im Interzeptor definiert
  - Muss Domänenklassennamen mit Endung *Interceptor* tragen
  - Gilt für **alle** Aktionen dieses Controllers
- Bedingter Interzeptor
  - Selbstdefinierter parameterloser Konstruktor im Interzeptor
  - Bedinge Interzeptoren
    - Für ein oder mehrere beliebige Controller
    - Für ein oder mehrere Controller-Aktionen
  - Konfiguration durch Match-Aufrufe im Konstruktor  
`package professorenverwaltung`

```
class ProfessorInterceptor {
    ProfessorInterceptor() {
        match controller:'professor', action:~/(login)?+.+/ // Alle Actions außer 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
    }
```

- Manuelle Programmierung eines Controllers
  - Controller nicht zwangsläufig Domänenklasse zugeordnet
    - Ohne Bezug auf Datenbank
    - Repräsentation von mehreren Domänenklassen möglich
    - Volle Flexibilität
  - Z.B. Login-Controller

## Domänenklasse

Klausur: Domainklasse für Student schreiben

- Constrains
  - Beziehungen
  - Warum `toString` überschreiben?
  - Ganze normale Groovy-Klassen mit
    - Attributen die automatisch auf Datenbank abgebildet werden
    - Constrains zur Konfiguration von Integritätsregeln für die Attribute
      - `Nullable`
      - `Blank`
      - `maxSize`
    - `toString()` überschreiben!
  - Werden automatisch persistiert
  - Beziehungen in Domänenklassen
    - 1:n -> `hasMany`
      - `static hasMany = [attributname: Domänenklasse]`
- ```

class Hochschule {
    String name
    String ort
    • static hasMany=[fakultaeten:Fakultaet] static belongsTo=[hochschule:Hochschule]
}
class Fakultaet {
    String name
    String kuerzel
}

```
- 
- ```

Hochschule 1 —n Fakultaet

```
- Fakultät hat Fremdschlüssel HochschulID
    - Hochschule `nullable:false` -> Hochschule **muss** angegeben werden
  - n:1 -> `belongsTo`
    - Zu beliebig vielen anderen Domänenklassen
    - `static belongsTo = [attributname: Domänenklasse]`
    - Löschen des besitzenden Datensatzes löscht auch abhängige Datensätze
    - Besitzer steht in [] -> hier Hochschule
      - Beispiel:
- ```

class Fakultaet {
    String name
    String kuerzel
    • static belongsTo=[hochschule:Hochschule] static hasMany=[fakultaeten:Fakultaet]
}
class Hochschule {
    String name
    String ort
}

```
- 
- ```

Fakultaet n —1 Hochschule

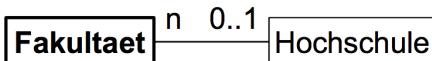
```
- Fakultät hat Fremdschlüssel HochschulID
    - Hochschule `nullable:false` -> Hochschule **muss** angegeben werden
  - n:0..1 -> `belongsTo`
    - Zu beliebig vielen anderen Domänenklassen
    - `static belongsTo = [attributname: Domänenklasse]`
    - Referenzattribut zusätzlich `nullable: true`

- Beispiel:

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

```
class Hochschule {
```

```
    String name
    String ort
```

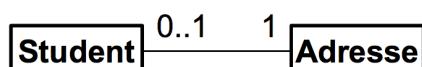


- static **belongsTo**=[referenzierende Klasse:**Hochschule**] static **hasMany**=[referenzierte Klasse:**Fakultaet**]
- Fakultät hat Fremdschlüssel HochschulID
  - Hochschule **nullable:true** -> Hochschule **kann** angegeben werden
- 0..1:1
  - Attribut vom Typ der referenzierten Klasse als Instanzvariable
    - Auf beiden Seiten
  - Auf mindestens einer Seite **nullable: true**

- 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)
    }
}
```



- Student hat Fremdschlüssel adressID
  - Adresse **nullable:false**-> Adresse **muss** angegeben werden
  - In Datenbank kann Fremdschlüssel dennoch null sein
    - Controller und View verhindern das Anlegen ohne Fremdschlüssel
- Bessere Lösung: Einbetten von Objekten
  - Adresse in Klasse Student einbetten
  - Adresse ist **keine** Domänenklasse
  - Vermeidet
    - Unnötige Datenbank-Joins bei Abfragen
    - Eigenen Controller und GSP 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
}
```

```
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"
}
}
```

- Eingebettete Klasse `Adresse` wahlweise als
  - Eigene Domänenklasse
    - Bekommt (überflüssige) Datenbanktabelle
  - Weitere Klasse in Quellcodedatei oder umgebende Klasse `Student`
    - Groovy erlaubt Definition mehrere Klassen in Quellcodedatei
    - Keine eigene Datenbanktabelle
  - Normale Groovy-Klasse

- Keine eigene Datenbanktabelle
- Beste Lösung
- m:n ->hasMany
  - Zu beliebig vielen anderen Domänenklassen
  - Definition über assoziatives Array **hasMany** von beiden Seiten her:
    - static **hasMany**=[attrname : DomainClass]
  - Auf **genau** einer der beiden Seiten zusätzliches **belongsTo**-Attribut notwendig
    - Beispiel:

```

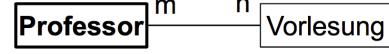
class Professor {
    String vorname
    String nachname
    static hasMany = [vorlesung:Vorlesung]
}

```

```

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

```



- Nachteil direkter m:n Beziehungen
  - Eine der beiden Seiten muss 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

```

class Professor {
    String vorname
    String nachname
    static hasMany = [gehalteneVorlesung:GehalteneVorlesung]
}

```

```

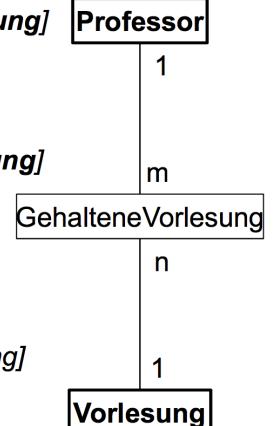
class GehalteneVorlesung {
    String semester
    static belongsTo=[professor:Professor,vorlesung:Vorlesung]
}

```

```

class Vorlesung {
    String name
    String kuerzel
    static hasMany = [gehalteneVorlesung:GehalteneVorlesung]
}

```



- 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
  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                              |                       | ++           | -        | +                          |  |  |

- Fehler
  - Rückgabe mir response funktioniert mit abgeleiteten Klassen nicht
    - Begebung durch klassische Model-Rückgabe
    - Nachteil: Content-Negotiation entfällt

## GORM: Abbildung Domänenklassen → Datenbankschema

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

- GORM-Customizing
  - Datenbankstruktur kann manuell und beliebig angepasst werden z.B.:
    - Tabellennamen
    - Spaltennamen
    - Name der m:n Beziehungstabelle

## 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)                                 |    | Klasse B (n-Seite)            |          | kaskadierendes Löschverhalten |       |
|----------------------------------------------------|----|-------------------------------|----------|-------------------------------|-------|
| Syntax                                             | FK | Syntax                        | FK       | 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]<br/>static belongsTo=B</b> |    | <b>static hasOne=[a:A]</b>    | <b>a</b> | -*                            | ja    |
| <b>static hasMany=[b:B]<br/>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 einer 1:1-Beziehung

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

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

- Transistente Attribute
  - Von Persistierung ausgeschlossen aus z.B. folgenden Gründen
    - Zur Laufzeit berechnet
    - Reine Laufzeitinformationen
    - Komplexe Datentypen, die nicht in Datenbank speicherbar sind
  - Syntax: static transients=['attributname', ...];
- Eingeimpfte Domänenklassen-Methoden
  - Domainclass.list()
    - Liefert Collection mit allen Instanzen dieser Domänenklasse
  - Domainclass.count()
    - Liefert Anzahl der Instanzen dieser Domänenklasse
  - Domainclass.get(id)
    - Liefert Instanz mit Primärschlüssel id
  - Domainclass.getAll(id...)
    - Liefert Instanzen mit angegebenen Primärschlüsseln

- ***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.findByName("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.findAllByAttr1Like(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.findAllByAttr1Ilike(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**Attr1Greater Than** (*value1*)
    - Ergebnis: Alle Datensätze, für die *attr1*>*value1*
  - Domainclass.findAllBy**Attr1Greater Than Equals**(*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)* // Studentennormalter
- 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*