WAF-ZUSAMMENFASSUNG

Mehmet [redacted]

Inhalt

Es wurden keine Einträge für das Inhaltsverzeichnis gefunden.

Inhalt

Mögliche Klausurfragen und Antworten ... 3

1.Auf welcher Schicht läuft PHP? ... 3

2.Auf welcher Schicht läuft JavaScript? ... 3

3.Was kann man mit Ranges machen? ... 3

4.Warum hat Java so viele Sachen über Collections? ... 3

5.Was ist eine Range? ... 3

6.Was sind assoziative Arrays? ... 3

7.Welche Vorteile hat Groovy gegenüber Java? ... 3

8.Erstellen Sie eine Klasse Vektor mit X und Y Koordinaten. ... 3

9.Reguläre Ausdrücke: Schreiben Sie ein Pattern hin z.B. für eine Kreditkarte, Straße, HS-Räume etc. ... 3

10.Was kann man mit Metaprogrammierung machen? ... 3

11.Was ist statisches Scaffolding? Was kann man damit machen? ... 3

12. Was ist dynamisches Scaffolding? Was kann man damit machen? ... 3

13.Was bedeutet das, wenn man Datenbanken austauschen kann? ... 3

14. Wieso sollte man Konstruktoren in Domänenklassen machen? ... 3

Was ist der Unterschied zwischen Java Bean und Groovy Beans? ... 3

Was ist ein flash Objekt? ... 4

Was ist der Unterschied zwischen Expando und Metaprogrammierung? ... 4

Was ist der Unterschied zwischen Java Reflection und Metaprogrammierung? ... 4

GORM-Methoden? ... 4

Wo läuft Grails? ... 4

Servelet Container ... 4

Was sind die Vorrausetzungen für Grails? ... 4

Wo läuft Ruby, Phyton, PHP? ... 4

Webarchitektur ... 5

Groovy und Grails ... 6

Groovy ... 6

Syntax und Kontrollkonstrukte ... 6

Closures ... 6

Datentypen und Operationen ... 9

Collections ... 11

Klassen und Groovy Beans ... 14

Reguläre Ausdrücke ... 15

Metaprogrammierung und Builder ... 17

Grails – Framework ... 19

Interzeptor 21
Manuelle Programmierung eines Controllers 21
Domänenklasse 21
MVC 22

Mögliche Klausurfragen und Antworten

1.Auf welcher Schicht läuft PHP?

Presentation Tier (Präsentation)?

Buisness Tier (Anwendungslogik)?

2.Auf welcher Schicht läuft JavaScript?

Client Tier (Virtualisierung)

3.Was kann man mit Ranges machen?

Vollwertiges Objekt
Verändern
An Methoden übergeben
Auf alle Elemente zugreifen

4.Warum hat Java so viele Sachen über Collections?

Aus Performancegründen
Operatoren konnte man nicht überladen

5.Was ist eine Range?

6.Was sind assoziative Arrays?

Es sind sozusagen Maps. Man greift über Schlüssel auf die Werte drauf.
Vorteil: schneller Zugriff auf Datensätze über KEY, einfache Verzeichnisse, Zählen der Häufigkeit von Wörtern

7.Welche Vorteile hat Groovy gegenüber Java?

Datentypen austauschen
Ranges anwenden
Mit dem each Iterator über Collections iterieren
Default Werte für Methodenparameter geben

8.Erstellen Sie eine Klasse Vektor mit X und Y Koordinaten.

9.Reguläre Ausdrücke: Schreiben Sie ein Pattern hin z.B. für eine Kreditkarte, Straße, HS-Räume etc.

10.Was kann man mit Metaprogrammierung machen?

11.Was ist statisches Scaffolding? Was kann man damit machen?

12. Was ist dynamisches Scaffolding? Was kann man damit machen?

13.Was bedeutet das, wenn man Datenbanken austauschen kann?

Keine SQL Befehle schreiben
Wie gewohnt Domänenklassen erstellen ohne sich um den Rest zu kümmern
In Zukunft kann man auch Java Code mittels nur UML generieren

14. Wieso sollte man Konstruktoren in Domänenklassen machen?

Braucht man nicht, da man automatisch assoziative Konstruktoren hat → Es muss so sein damit Hypernate es aufrufen kann.

Was ist der Unterschied zwischen Java Bean und Groovy Beans?

Java Beans: Braucht Konstruktor und private Attribute und Getter Setter Methoden
Grovvy Beans: Konstruktoren und Getter und Setter sind selbst definiert

Was ist ein flash Objekt?

An gewünschten Stellen kann man diesem Objekt auf serverseite messages hinzufügen, die dann beim Client aufpoppen. Objekt zum Ablegen von Fehlermeldungen

Was ist der Unterschied zwischen Expando und Metaprogrammierung?

Expando: Da kann man eine erstelle Instanz um ein Attribut ohne Probleme erweitern aber man kann keine Methoden hinzufügen

Was ist der Unterschied zwischen Java Reflection und Metaprogrammierung?

Java Reflection: Man kann da nur die Methoden und Attribute und Klasse nur auslesen und ändern aber keine dynamisch hinzufügen und man kann keine Aufrufe abfangen und haben keine Interzeptoren

GORM-Methoden?

List, get(id), count, save, delete, findbyXXXX

Wo läuft Grails?

Im Servelet Contrainer

Servelet Container

Da läuft Grails, JSP, Webservices, Webanwendung, JSF

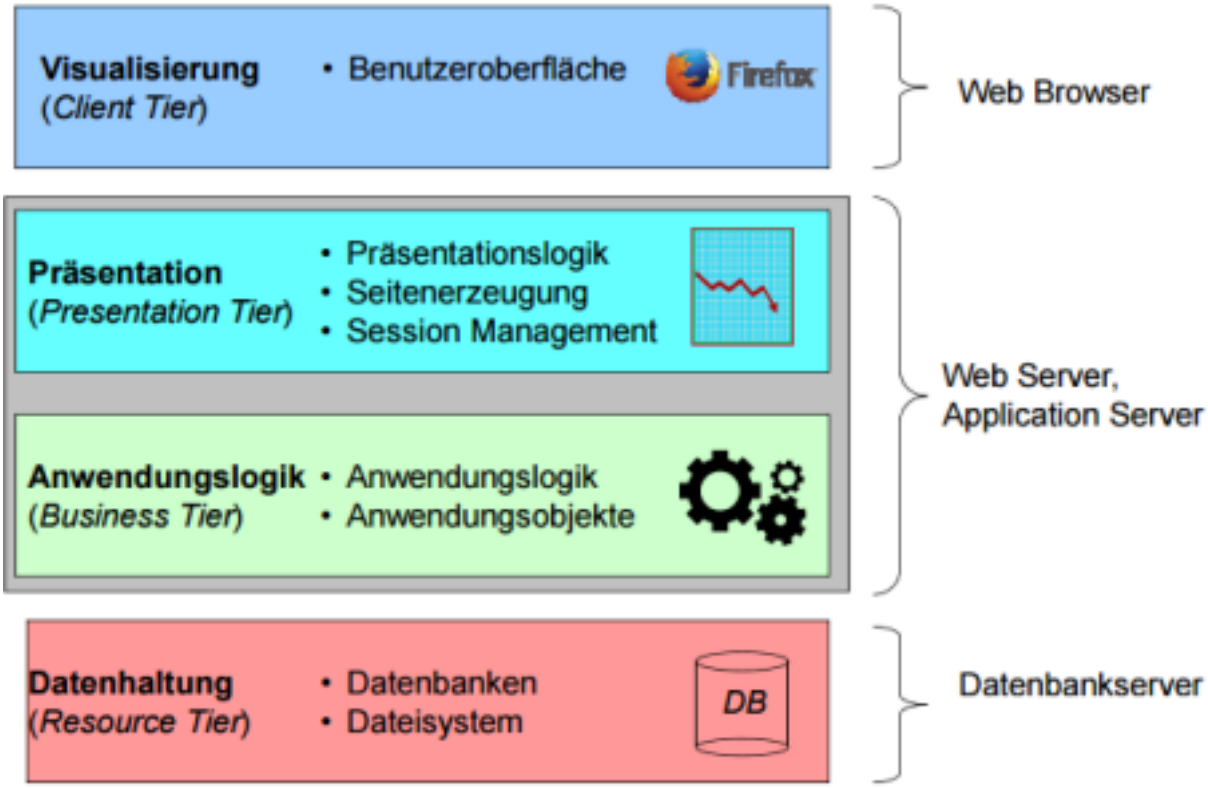
Was sind die Vorrausetzungen für Grails?

Ein Webserver mit integriertem Servelet Container z.b. Tomcat.

Wo läuft Ruby, Phyton, PHP?

Servelet-Engine: Modulschnittstelle CGI

Webarchitektur



Web Browser:	FireFox, Opera, Chrome
Visualisierung:	HTML, CSS, JavaScript
Web Server, App. Server:	Apache, Tomcat, JavaEE
Präsentation:	JSP, Servelets, Groovy, JSF
Anwendungslogik:	Java, Groovy
Datenbanken, Datenbanksystem:	MySQL, Oracle
Datenhaltung:	SQL

Ziel: Möglichst mit dem Browser ohne Plungins (Modulschnittstelle) auskommen → da Sicherheitslücke

CGI-Schnittstelle ist sicherer und ist plattformunabhängig, da es standardisiert, ausgereift und zuverlässig ist.

Servelet-Container:

- ➔ Sicher
- ➔ Wenn das JRE eine Sicherheitslücke hat, hat alles was darauf aufsetzt auch eine Sicherheitslücke
- ➔ Läuft in der VM ab

Benutzereingaben werden auf der Browserseite sowie Serverseite überprüft → besser wegen Last

Groovy und Grails

ER-Modelle nicht mehr notwendig → kein SQL mehr, Java Klassen ohne Methoden (generiert automatisch Datenbanktabellen)

; und Klammern weglassen → Syntax nicht mehr so streng wie in Java

Eher für größere Systeme.
Performance nachrangig

Groovy

- ➔ Kompilierter Groovy Code ist dasselbe wie Java Bytecode
- ➔ Speckt soweit wie möglich den Code ab → erhöht die Übersicht
- ➔ Fast jedes Java Programm ist auch ein Groovy-Programm
- ➔ Java Quellcode muss vor Ausführung kompiliert werden
- ➔ Groovy kann wahlweise kompiliert werden oder direkt als Script ausgeführt werden

```
...

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()
```

Syntax und Kontrollkonstrukte

Statt des Datentyps kann auch einfach def angegeben werden

```
def max(x,y){
    x > y ? x : y // gillt für alle Datentypen, welche für > definiert ist
}
```

Closures

- ➔ Codeblöcke = Objekte
- ➔ Returnen immer das letzte Zeichen
- ➔ Werden aufgerufen wie eine Methode
- ➔ Sind in Java anonyme Klassen
- ➔ Wenn eine Closure mit einem String zusammengehängt wird → "$it" ansonsten ohne
- ➔ Ist ein anonymer Codeblock zwischen { }← Ist ein Objekt
- ➔ Mit dem ‚it' Parameter kann man nur arbeiten, wenn man nur einen Parameter hat

---

```
def fak = {
    n -> result = 1;
        for(i=2; i <= n; i++)
            result*= i;
        result
}
```

```groovy
// Jetzt mit Referenz auf den Codeblock
def fakRec;
fakRec = {
    n -> n==1 ? 1 : n*fakRec(n-1)
}

println fakRec(5)
```

```groovy
// Rekursive Clousure erstellen
def spiegeln(int x){
    return x ? "${x%10} ${spiegeln(x.intdiv(10))} ${x%10}" : "|"
}

// Nun eine Clousre erstellen
def spiegelnClosure = this.&spiegeln

//wichtig den Übergabeparameter nicht vergessen
println spiegelnClosure(4711) // 1 1 7 4 | 4 7 1 1
```

```groovy
// Jetzt mit Referenz auf den Codeblock
// verkürzte Schreibweise, da ein Parameter
def fakRec;
fakRec = {
   it == 1 ? 1 : it*fakRec(it-1)
}
```

```groovy
// Clousure als Parameter übergeben mit zwei Parameter
def applyMethod(x,y,clousure){
    return clousure(x,y)
}

println applyMethod(4,8,{q,p -> q > p ? p : q}) // 4

//******Globale Methoden******
def min(x,y){
    x < y ? x : y;
}

def max(x,y){
    x > y ? x : y;
}

println applyMethod(3,8, this.&max) // 8
```

```groovy
//Array werte mit beliebig vielen Werten
def meinMaxOhneClousure(x, Object[] werte) {
    def ergebnis = x;
    werte.each {
        if(it > x){
            ergebnis = it
        }
    }
    ergebnis
}
println meinMaxOhneClousure(9,8,17,5) // 17

//Array werte mit beliebig vielen Werten
def meinMaxMitClousure(x, Object[] werte) {
    def ergebnis = x;
    for (i=0; i<werte.size(); i++) {
        if (werte[i]>ergebnis)
            ergebnis = werte[i]
    }
    ergebnis
}
println meinMaxMitClousure(9,8,17,5) // 17
```

```groovy
// Bei einem String die Character-Methode toLowerCase verwenden
def applyForEachChar(string, closure) {
    def result=""
    for(i in 0..<string.length())
        result += closure(string[i] as Character)
    return result
}

println applyForEachChar("ThOmAs", Character.&toLowerCase) // thomas
```

```groovy
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<1000000; i++); } // Tue Dec 22 00:46:31 CET 2015 - Tue Dec 22 00:46:32 CET 2015: 488 ms
```

```groovy
//***Eine anderes Beispiel zur Zeitmessung
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<1000000; i++);
timer()         // Closure-Aufruf
```

Sichtbarkeit:

- Merkt sich den Kontext
- Alle Variablen lokale und formale Parameter werden zum Definitionszeitraum eingesetzt
- Clousures erben seinen Definitionskontext

Datentypen und Operationen

- 5plus(3) → kann man überladen z.B. für Vektorrechnung
- == vergleicht auf den Inhalt (nicht die Referenz) → immer überschreiben wenn wir Klassen schreiben
- .is(Objekt) auf Referenz vergleichen

```
class Bruch implements Comparable<Bruch> {
    int zaehler
    int nenner

    Bruch(int zaehler,int nenner=1) {
        this.zaehler=zaehler
        this.nenner=nenner
    }

    Bruch kuerzen() {
        new Bruch(zaehler/ggt(zaehler,nenner) as int,
                nenner/ggt(zaehler,nenner) as int)
    }

    Bruch plus(Bruch b) {
        new Bruch(zaehler*b.nenner+b.zaehler*nenner,nenner*b.nenner).kuerzen()
    }

    Bruch minus(Bruch b) {
        new Bruch(zaehler*b.nenner-b.zaehler*nenner,nenner*b.nenner).kuerzen()
    }

    Bruch multiply(Bruch b) {
        new Bruch(zaehler*b.zaehler,nenner*b.nenner).kuerzen()
    }

    Bruch multiply(int b) {
        new Bruch(zaehler*b,nenner).kuerzen()
    }

    Bruch div(Bruch b) {
        new Bruch(zaehler*b.nenner,nenner*b.zaehler).kuerzen()
    }
```

```
    Bruch power(int potenz) {
        new Bruch(zaehler**potenz, nenner**potenz).kuerzen()
    }

    Bruch negative() {
        new Bruch(-zaehler,nenner).kuerzen()
    }

    Bruch positive() {
        new Bruch(zaehler,nenner).kuerzen()
    }

    boolean equals(Bruch b) {
        Bruch aGek = this.kuerzen()
        Bruch bGek = b .kuerzen()
        aGek.zaehler== bGek.zaehler && aGek.nenner==bGek.nenner
    }

    int compareTo(Bruch b) {
        this.zaehler*b.nenner-b.zaehler*this.nenner
    }

    String toString() {
        nenner==1 ? zaehler  :"($zaehler/$nenner)"
    }

    static int ggt(int a, int b) {
        a%b ? ggt(b, a%b) : b
    }
}
```

```
def a = new Bruch(4,6).kuerzen()
def b = new Bruch(1,28)

println a // (2/3)
println b // (1/28)
println a+b // (59/84)
println a-b // (53/84)
println a*b // (1/42)
println a/b // (56/3)
println b+a*3 // (57/28)

println a**3 // (8/27)

println (-a) // (-2/3)
println a==new Bruch(2,4) // false

a= new Bruch(2,3)
b = new Bruch(3,4)
println (a>=a) // true
```

Collections

- Arbeiten mit den eckigen Klammern
- Datentypen sind egal
- Element hinzufügen list+="Astrid"
- list.sort({-1}) → sortiert nachdem letzten Buchstaben
- Range = auch ein Objekt mit Ober und Untergrenzen (benötigt aufzählbare Typen die Comparable implementieren → muss aufzählbar sein)
- Java: for(int i = 0; i <= 9; i++)
- Groovy: for(i in 1..9)
- Iteration durch die Collection per each-Closure → list.each {println "$it"}

```groovy
def list = ["Thomas", "Astrid", "Ivo"]
list.each {println it} // Thomas Astrid Ivo

list *=2
list.each {println "$it"} // Thomas Astrid Ivo Thomas Astrid Ivo

list[1..4] = [] // Strings die im Bereich 1 bis 4 sind löschen
list.each {println "$it"} // Thomas Ivo

list+= ["Mehmet","Devran"]
list+= "Michele"
list.each {println "$it"} // Thomas Ivo Mehmet Devran Michele

list[1..3] = "Paul"
list.each {println "$it"} // Thomas Paul Michele
println list[-2] // Paul
println list[1..2] // [Paul, Michele]

list -= ["Michele", "Paul"]
list.each {println "$it"} // Thomas

println list[-1] // Thomas
println list[0] // Thomas

list = [] // Liste komplett löschen
```

```groovy
def noten=[1.0,1.7,3.0,2.0,4.0, 3.7]
println noten.findAll {it < 2.0} // [1.0, 1,7] // Alle Listenelement die das Kreterium erfüllt liefern
println noten.find {it >= 2.0} // 3.0 // Erstes Listenelement liefer die das Kreterium erfüllt
println noten.every {it <= 4.0} // true // Überprüft ob alle dieses Kriterium erfüllen
println noten.every {it < 4.0} // false
println noten.any {it == 1.0} // ture // Überprüft ob min. ein Element dieses Kreterium erfüllt
println noten.any {it == 1.3} // false
```

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"
}
```

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

---

Nutzung Assoziativer Arrays (1/2)

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

* 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]

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

/../ → Benutzt man um Reguläre Ausdrücke zu beschreiben und ==~ ist ein Match Operator

Klassen und Groovy Beans

- Syntax wie Java
- Klassen ohne Angaben sind public
- Java Bean → Serialisierbar, Parameterlose Konstruktoren, Setter und Getter (werden automatisch erstellt)
- Für Groovy und Java müssen alle Klassen Serializable implementieren
- Man braucht keine Konstruktoren mehr zu schreiben → nichts anderes als ein assoziatives Array → wenn man einen Konstruktor haben will, dann in Java-Style (man muss ihn selbst erstellen)
- Swing ist auch Java Bean
- Globale Variablen ohne des und ohne Datentypangabe
- Methodennamen kann während der Laufzeit festgelegt werden
- Bei Domain Klassen die Datentypen hinschreiben wegen DB und Typsicherheit
- Um einen assoziativen Array als Konstruktor zu machen → vorher: new Student(vorname:“Mehmet“, nachname:“[redacted]“,…) → nacher: 123666: new Student(….)
- Ein leeres assoziatives Array → Map noten = [:] → Im Konstruktor noten:[GDI:1.0, OOT:2.0

```
def wafTeilnehmer = [
        127661:new Student(vorname: "Michael", nachname: "Müller",matrNr: 127661,
                noten: [GDI:1.0, OOT: 1.3, WAF:1.3]),

        0125431:new Student(vorname: "Anja", nachname: "Schmidt", matrNr: '0125431',
                noten: [GDI: 1.0, OOT:3.0, WAF:1.0]),
        451121:new Student(vorname: "Martina", nachname: "Heise",matrNr: 451121,
                noten: [GDI:3.0, ADS: 1.7, TPE:3.0])
]
println wafTeilnehmer // [127661:Müller, Michael, Matrikelnr: 127661,
                      // 43801:Schmidt, Anja, Matrikelnr: 0125431, 451121:Heise, Martina, Matrikelnr: 451121]
wafTeilnehmer+= [344321:new Student(vorname: "Florian",nachname: "Meyer",matrNr: 344321)]
Student peters= new Student(vorname: "Heinz Peter",nachname: "Peters")
println peters // Peters, Heinz Peter

wafTeilnehmer.each { println "$it.value. ${it.value.noten ? 'Noten: ' + it.value.notenToString():''}"}
/**
 * Müller, Michael, Matrikelnr: 127661. Noten: GDI: 1.0, OOT: 1.3, WAF: 1.3
 Schmidt, Anja, Matrikelnr: 0125431. Noten: GDI: 1.0, OOT: 3.0, WAF: 1.0
 Heise, Martina, Matrikelnr: 451121. Noten: GDI: 3.0, ADS: 1.7, TPE: 3.0
 Meyer, Florian, Matrikelnr: 344321.
 */
println "Best Student Award"
wafTeilnehmer.findAll {key, value ->value.noten.any {it.value==1.0}}.each
        { println "$it hat 1.0 in ${it.value.noten.findAll{it.value==1.0}.keySet()}"}
/*
127661=Müller, Michael, Matrikelnr: 127661 hat 1.0 in [GDI]
43801=Schmidt, Anja, Matrikelnr: 0125431 hat 1.0 in [GDI, WAF]
 */
```

- Metaprogrammierung: Neue Methoden und Attributwerte einfügen auch zu Bestandklassen
- Println 1+a Burch a = new Buch(1,7) → 1.plus(a) → die 1 vor dem .plus(a) ist delegate
- Derefernzieren: if(knauber?.vorname) → überprüft auf null etc. deshalb kein NullPointer
- Expanos: hat auch mit Metaprogrammierung zutun

```
// Expandos: dynamisches Objekt
// Beliebige Attribute hinzufügen durch bloßes beschreiben eines nicht vorhanden Attributes
c1 = new Expando(re:1,im:3)
c1.name = "Mein Expando"
println "$c1.name $c1.re $c1.im"
```

Reguläre Ausdrücke

- Match-Operator → booleanResult = testString ==~ pattern
- . → beliebiges Zeichen
- Find-Operator → ergArray = testString =~ pattern → wenn Pattern ohne Capture Group () dann ist Array 1 Dim → wenn Pattern mit Capture Group dann zwei 2Dim (1 Zeile pro Match)
- **? reluctant und + greedy**

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*'

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]/
'c' ==~ /[abc]/
'd' ==~ /[abc]/
'A' ==~ /[abc]/

'a' ==~ /[^abc]/
'c' ==~ /[^abc]/
'd' ==~ /[^abc]/
'A' ==~ /[^abc]/

'a' ==~ /[a-cA-C]/
'c' ==~ /[a-cA-C]/
'd' ==~ /[a-cA-C]/
'A' ==~ /[a-cA-C]/
'C' ==~ /[a-cA-C]/
'D' ==~ /[a-cA-C]/

'a' ==~ /[[a-z]&&[def]]/
'b' ==~ /[[a-z]&&[def]]/
'c' ==~ /[[a-z]&&[def]]/
'd' ==~ /[[a-z]&&[def]]/
'A' ==~ /[[a-z]&&[def]]/
'D' ==~ /[[a-z]&&[def]]/

'a' ==~ /[[a-z]&&[^a-c]]/
'c' ==~ /[[a-z]&&[^a-c]]/
'd' ==~ /[[a-z]&&[^a-c]]/
'A' ==~ /[[a-z]&&[^a-c]]/
'D' ==~ /[[a-z]&&[^a-c]]/

* && nur zur Verknüpfung von Zeichenbereichen in [] anwendbar!

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*	*aaaaa*

'' ==~ /a?/
'a' ==~ /a?/
'aaa' ==~ /a?/
'aaab' ==~ /a?/

'' ==~ /a*/
'a' ==~ /a*/
'aaa' ==~ /a*/
'aaab' ==~ /a*/

'' ==~ /a+/
'a' ==~ /a+/
'aaa' ==~ /a+/
'aaab' ==~ /a+/

'' ==~ /a{2}/
'a' ==~ /a{2}/
'aa' ==~ /a{2}/
'aaa' ==~ /a{2}/

'' ==~ /a{2,}/
'a' ==~ /a{2,}/
'aa' ==~ /a{2,}/
'aaa' ==~ /a{2,}/

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

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'  ==~ /ab/
'b'  ==~ /ab/
'ab' ==~ /ab/
'ba' ==~ /ab/
```

```
''   ==~ /a|b/
'a'  ==~ /a|b/
'b'  ==~ /a|b/
'ab' ==~ /a|b/
'ba' ==~ /a|b/
```

```
''     ==~ /(ab)*/
'a'    ==~ /(ab)*/
'b'    ==~ /(ab)*/
'ab'   ==~ /(ab)*/
'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/

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) †

- Greedy = nimm dir so viel was du nehmen kannst (Greedy, Reluctant, Prossesive sind Strategien)
- Reluctant = nimm dir so wenig wie es geht
- Prossesive = ich reiß alles an mir, egal ob das Pattern scheitert
- Find-Operator eigentlich gar kein 2Dim Array sondern Instand der Klasse java.util.regex.Matcher
- Mit dem Find-Operator kann man einen String in einem Text suchen
- Pattern-Operator: matcher = ~/pattern/ → Verwendung bei Performance

Metaprogrammierung und Builder

- Metaprogrammierung = Nicht nur einfach programmieren, sondern eine Ebene höher gehen
- Durch Metaprog. Sieht man z.B. bei einer String Variable „a“, wenn man a. die Vorschlag Methoden
- Reflection: Programm kennt seine eigene Struktur zur Laufzeit→Vorteil: höhere Flexibilität, Parametrisierung on Attribut- und Methodennamen →Nachteil: Keine Typprüfung durch Compiler möglich, geringe Performance
- Metaprogrammierung: Vorteil: Konverter und Rechenoperationen zu selbst definierten Datentypen, Selbstmodifizierter Code möglich → Nachteil geringe Performance
- Groovy kann zur Laufzeit, Klassen und Methoden ändern
- Die Klasse class ist auch ein Objekt
- Metaprogrammierung → Nachteil → Performance

```
// Konstruktoren zu existierenden Klassen hinzufügen
String.metaClass.constructor = {BigInteger bi -> return bi.toString()}
String meineZahl = new String(3); // Dies würde normalerweise nicht Funktionieren, dank Metaprog. funktioniert es
println meineZahl // 3
```

- Wie funktioniert das? →Groovy greift auf die Attribute und Methoden in der Laufzeit zu und schaut ob es dafür eine Metaprogrammierung gibt → wenn nicht arbeitet Groovy den Code als ganz normales Java Code ab
- MetaClass-Interface bildet die Grundlage in Metaprogrammierung in Groovy

```
// Metaprogrammierung wird irgendwann unlesbar
// Domänklasse Beispiel:
// hier fehlt die ID, diese wird durch Metaprog. auto. hinzugefügt, sowie die Getter- und Settermethoden
class Professor{
    String name
    String vorname
    String kuerzel
}
```

- ExpandoMetaClass.enableGlobaly() → acht die Änderung auch auf bereits existierende Objekte.
- Invoke-Method wird immer aufgerufen → fängt alle Methoden aufrufe ab → z.B. Protokollieren welche Methode aufgerufen worden sind

```
// String-Klasse / Instanz um Attribut erweitern
ExpandoMetaClass.enableGlobally()
def thomas = "Thomas"
def peter = "Peter"
def rainer = "Rainer"
thomas.metaClass.version="2.0" // Konkrete Instanz um Attribut erweitern
String.metaClass.version="1.0" // Klasse um Attribut erweitern
println thomas.version  // 2.0
println peter.version   // 1.0
println rainer.version  // 1.0
```

```
// String-Klasse / Instanz um Methode erweitern
ExpandoMetaClass.enableGlobally()
def thomas = "Thomas"
def peter = "Peter"
def rainer = "Rainer"
thomas.metaClass.convert = {
    delegate.toUpperCase()
}

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

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

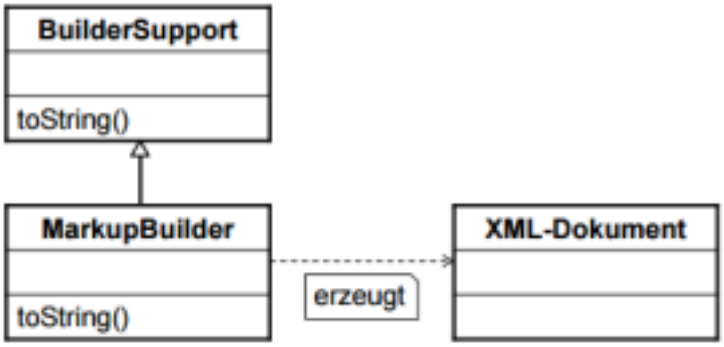
```groovy
// Invoke Method Beispiel
String.metaClass.invokeMethod = {
    String name, args ->
        System.out.println "Methode $name aufgerufen"
        System.out.println "Argumente:"
        args.each { System.out.println it}
        System.out.println "---------------"
        delegate.metaClass.getMetaMethod(name,args).invoke(delegate,args)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

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

- methodMissing() wird aufgerufen, wenn die aufrufende Methode nicht existiert → Erstmaliger Aufruf leidet die Performance, beim zweiten oder n-ten mal wird es manuell gecached

```groovy
ExpandoMetaClass.enableGlobally()
// Multiplizierermethoden nachtragen
BigInteger.metaClass.methodMissing = { String methodName, args ->
    println "MissingMethod aufgerufen: $methodName"
    if (methodName =~ /times\d+/) {
        def result = methodName =~ /times(\d+)/             // Faktor extrahieren
        if (result.size()==1) {
            // Methode für nachfolgende Aufrufe in der Klasse BigInteger ergänzen
            BigInteger factor = new BigInteger (result[0][1])     // Match 0, 1. CG
            BigInteger.metaClass."$methodName" = {
                delegate*factor;
            }
            delegate."$methodName"()
        }
    } else {
        throw new MissingMethodException(methodName, delegate.class, args)
    }
}
BigInteger b=14
BigInteger b2=10;
println b.times2()          //MissingMethod aufgerufen: times2 // 28
println b2.times2()         //Methode wurde gespeichert deshalb kein Aufruf String//20
println b2.times3()         //MissingMethod aufgerufen: times3 // 30
```

- Builder-Pattern → liefert komplexes Objekt, DSL (Domain Spicific Language)
- meinXML.professsoren ← root Element ← greift es mit methodMissing() ab und merkt, dass es dieses Objekt nicht gibt
- Builder dient dazu etwas zu erstellen was Groovy nicht beherrscht



MarkupBuilder → eigene Sprachen bauen

Grails – Framework

- Webframework für Groovy
- Stärke von Grails ist Groovy
- Grails gibt eine Ordnerstruktur an
- Alles einheitlich
- Grails Objekt Relation Mapper sorgt dafür, dass die Domänenklasse gemappt wird
- In Domänenklassen keine Anwendungslogik reinschreiben
- Scraffholding = Grundgerüst automatisch generieren lassen und da ergänzen wo ich will
- Model = ist unsere Domänenklasse
- Constraint = Einschränkung, deklarative Programmierung, Eigenschaften wie nicht null, nicht leer etc. → constraint ist eine Closure
- Controller erzeugen 3 Möglichkeiten → dynamisches Scaffolding: Generierung zur Laufzeit, nicht änderbar, kein Quellcode erzeugt, Adminoberfläche → statisches Scaffolding: beliebig änder- und erweiterbar, für komplexere Oberflächen → manuelle Programmierung: Quellcode manuell erstellen, höherer Programmieraufwand, Anwendung: Oberflächen abseits der üblichen CRUD-Funktionalitäten

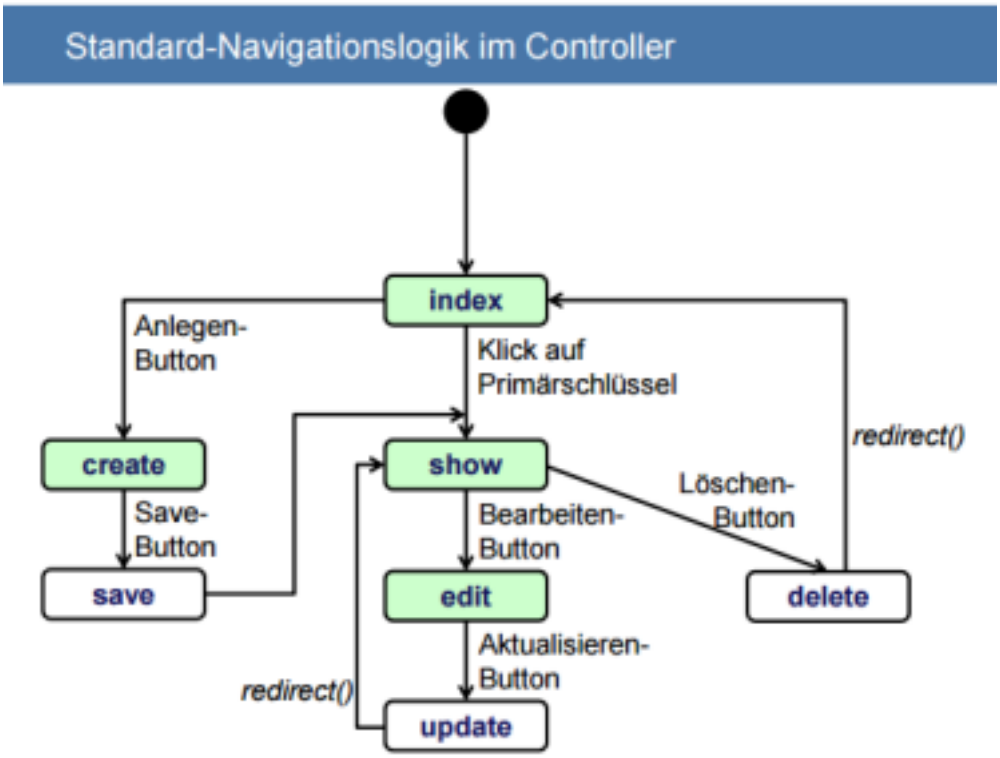
```
package vorlesungsverwaltung

class Vorlesung {
    String name
    String kuerzel
    int sws
    int credits

    static constraints = {
        name(nullable:false, blank:false)
        kuerzel(nullable:false, blank:false)
        sws()
        credits()
    }

    String toString() {
        "$name ($kuerzel)"
    }
}
```

- generate-all → erzeugt Controller und View → im Beispiel siehe oben generate-all vorlesungsverwaltung.Vorlesung
- localhost:8080/vorlesung/edit/3 → Domänenklasse, Action, ID
- Man kann beim Starten von Grails automatisch einen Scriptcode ausführen
- GORM-Methoden → Schnittstelle
- Domänenklassen haben einen Standartkonstruktor → sind Groovy Klassen → verbrauchen keine Vererbung
- def scaffold = Professor → Professor = Domänenklasse → Kontrolle und View wird in der Laufzeit erzeugt (muss unter ProfessorController)
- Ein Kontroller verwaltet nur eine Domäne



- save mit POST (Grund: Manipulierbarkeit der URL)
- update mit PUT
- delete mit DELETE
- alle GET-Parameter werden an max angehängt
- wenn die list() Methode unnötige Attribute bekommt → schmeißt sie diese weg
- model: stellt der aufgerufenen ESP den Model zu Verfügung
- respond: gibt diese Liste an die betroffene View weiter
- default.list.label = Entitätsname im Titel
- max = zeigt die Anzahl Profs. pro Seite
- show?id=1→ zeigt mir den Prof. mit der ID = 1 an
- Sicherheitsprüfung immer im Controller
- def create(params) → damit beim Fehlversuch nicht die Felder erneut angegeben werden müssen → wird noch nicht gespeichert bis man auf den Button create drückt
- def save(Vorlesung vorlesung) → hasErrors ruft die static contraints auf (überprüft auf Contraints) → request.withFormat {form multiPartorm…(Browser)} → '*' alle anderen Endgeräte
- Wie kommen Parameter von der View in den Controller? → params(assoziatives Array) → In der Controller-action Formular Parameter → id (http-Parameter muss id heißen)
- submitButton = Methodenaufruf
- bei constraint = {…} wird auch die Reihenfolge festgelegt
- standartaction ist index
- Zugriff auf Parameter params.vorname oder params["vorname"] kein Unterschied nach GET- und POST-Methode
- Linkaufruf ist Methodenaufruf (aus dem GSP heraus Methoden Aufruf)
- Controller ist ein Servelet
- flash-Objekt → Fehlermeldung auf die nächste Seite und Statusmeldung
- Logger: logger 'zuLogendeKomponente', Loggin Level

- Der speziellere Logger setzt sich immer durch
- respond = konvertiert automatisch in das Zielformat im Gegensatz zu render
- REST = Jede Resource wird durch eine Seite repräsentiert
- Content negotiation → passiert durch respond (tauscht nur die View aus)→ conf/application.vml → verschiedene Endgeräte durch einen Controller unterstützen
- Es gibt Controller actions ohne einer GSP z.B. update()

Interzeptor

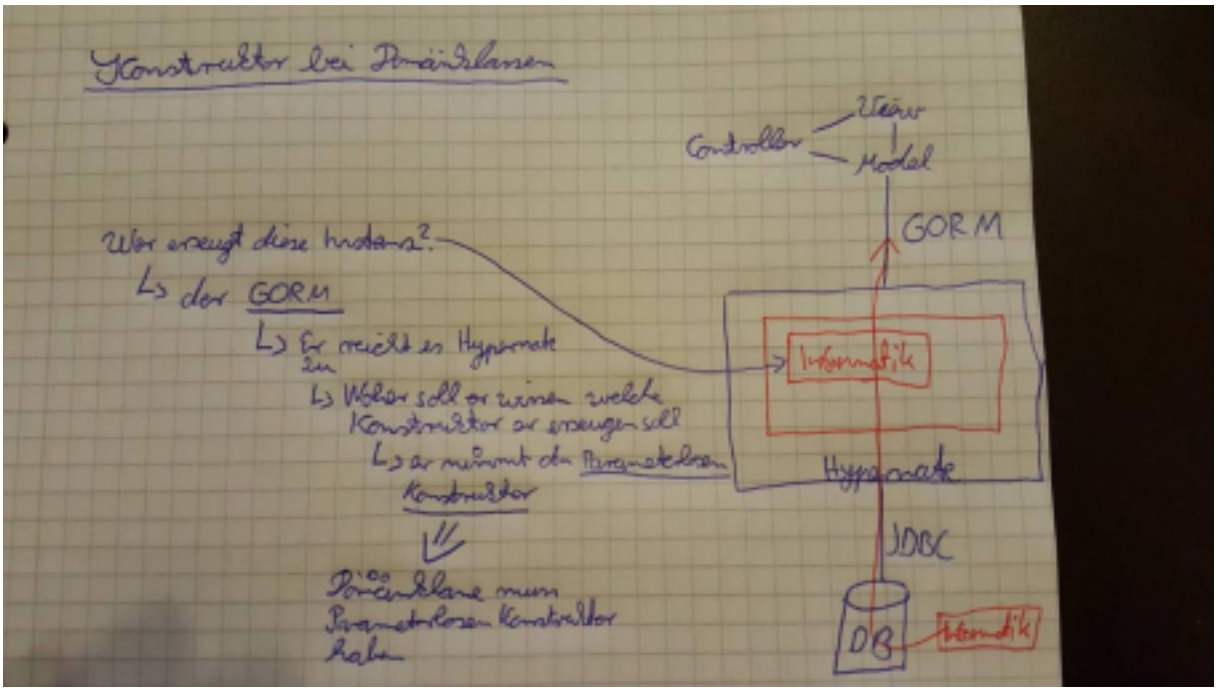
- Bei jedem Controller muss überprüft werden, ob der Benutzer angemeldet ist → Lösbar durch Interzeptoren → in der befor() Methode überprüfen, ob der Benutzer angemeldet ist → Ich brauche eine Möglichkeit das ich auf einmal in alle Controller dies überprüfen kann → Einen Interzeptor zentral für alle Controller schreiben (in einer externen Datei, nicht bei den Controller drinnen) → zunächst als Beispiel für den Prof. → create-interceptor professorverwaltung.Professor
- after() wird aufgerufen zum Rendern
- Bedingter Interzeptoren → heißt: wir wollen nicht für jeden Controller die befor() und after() Methoden aufrufen → z.B. ProfessorInterceptor(){match controller:‘professor‘, action: ~/(login)/} → alle actions außer Login
- Bedingter Interzeptor → parameterlosen Konstruktor aufrufen
- Interzeptoren biten die Möglichkeit vor jedem Methodenaufruf und nach jedem Methodenaufruf einzugreifen
- after() Überprüfung und Manipulation der Antwort

Manuelle Programmierung eines Controllers

- Tools->Grails->Run Target-> create-login controller → erstellt einen Controller ohne Inhalt
- def index{leer d.h. ohne redirect wird automatisch eine index.gsp aufgerufen}
- def login(String login, String password){…}
- Jetzt müssen wir die GSP bauen für das Login → index.gsp (beim Aufrufen landet man immer in der index aktion) → Bei ProfessorInterceptor redirect controller:‘login‘, action:‘index‘ → controller:‘login‘ sagt mir das ich den login controller verwende
- Anwendungslogik ohne Domänenklasse im Vergleich zu statisches Scaffolding

Domänenklasse

- Werden durch den GORM persistiert
- Domänenklassen sind Klassen ohne Methoden → gut um Datensätze zu erstellen
- Standartmäßig wird eine H2 Memory verwendet → wenn man z.B. MySQL verwenden will den JDBC-Treiber runterladen und in der Datei application.yml Änderungen vornhemen
- static hasMany=[fakultaeten:Fakultaet] (in eckigen Klammern weil man mehrere Beziehungen haben kann)
- Spalte version in MySQL: Wenn zwei denselben Datensatz bearbeiten → ist am Anfang 0 wenn ich am Datensatz arbeite wird die version um 1 erhöht und wenn ein anderer auch an dem selben Datensatz arbeitet und ladet den Datensatz auch hoch, dann wird geprüft welche version der zweite Typ hat und wenn die kleiner ist wie die aktuelle wird diese version nicht übernommen. → Auf die Art kann man auf die Sperrung verzichten
- Wenn man ganz normal Objektorientiert Programmiert, braucht man keine NORMALFORM nicht mehr.
- GORM übernimmt Attributnamen in den Tabellennamen → Brücke zu Hypernate (Kapselt Hypernate)

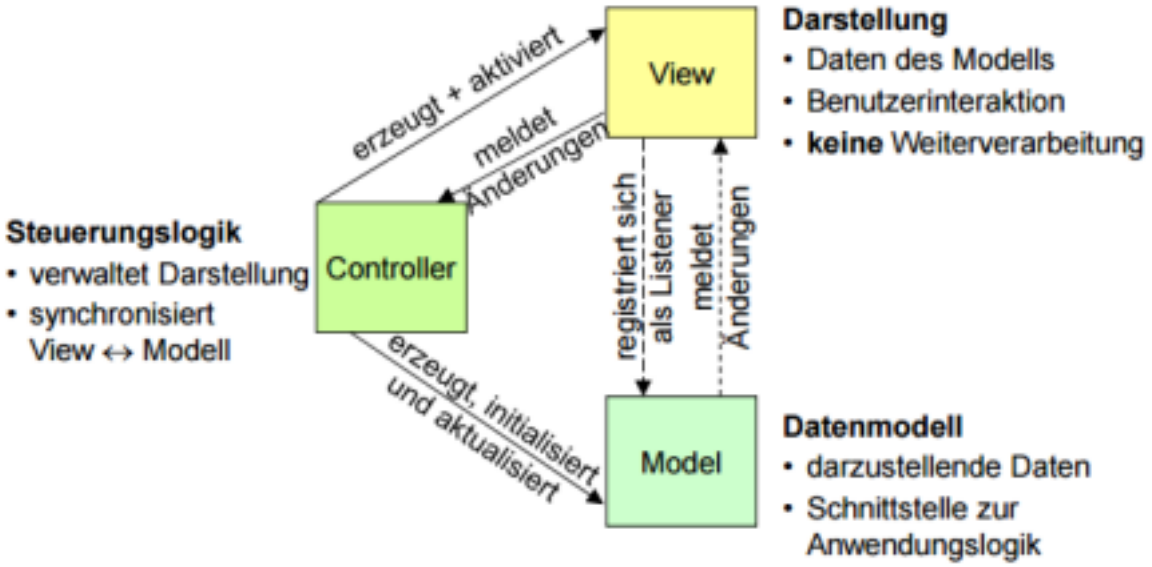


- GORM = OR –Mapper → GORM ruft Methoden auf → man kann sich bei Grails nicht mehr entscheiden → Grails zwingt uns die Infrastruktur einzubehalten
- Grails setzt auf Inversion-Controll → Ich verzichte auf Programmieraufwand → Vorteil:bei Agilen Prozessen können wir schnell auf einen Prototypen kommen, Nachteil: Ich verzichte auf Freiheit.

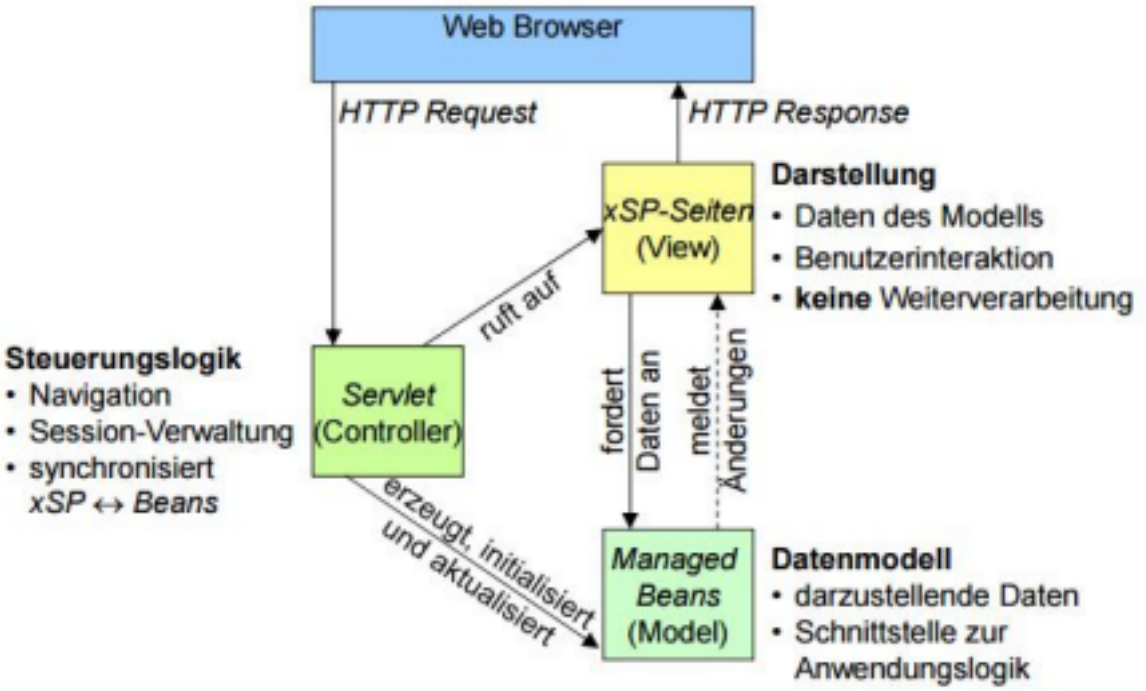
MVC

Model View Controller (MVC)

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



Spezialisierung MVC für Webframeworks



MVC in Grails