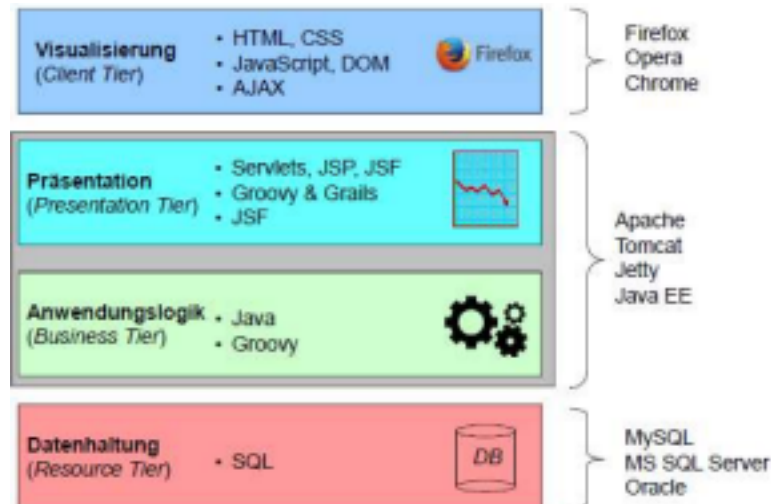


WAF Klausurfragen

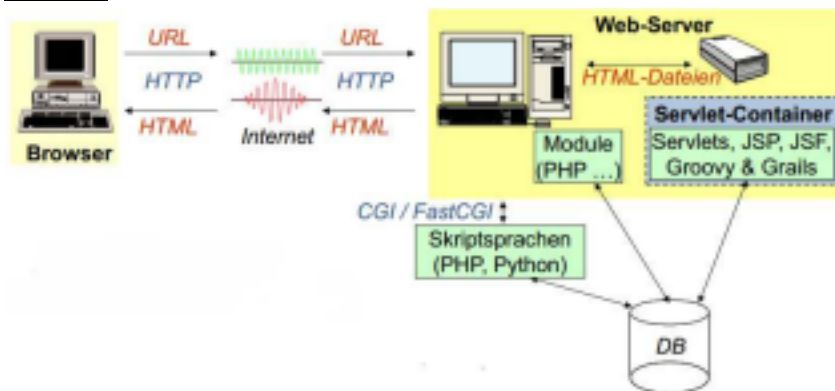
1. Skizzieren Sie ein Vierschichten-Modell für Web-Anwendungen und schreiben Sie die entsprechenden Groovy und Grails Technologien rein.

Antwort:



2. Skizzieren Sie Zusammenspiel zwischen Webserver, Datenbank und Skriptsprache am Beispiel von Groovy und Grails. Welche Protokolle werden verwendet?

Antwort:



3. Warum wurde schon wieder eine neue Skriptsprache entwickelt? Hätte man nicht einfach Java verwenden können?

Antwort:

- „Lasche“ Syntaxvorschriften und die dynamischen Typisierung
- gute Integration von Klassenbibliotheken und Frameworks (Java-Klassenbibliothek)
- einfacher Einlernaufwand für Java-Programmierer
- Skriptfähigkeit

4. Was sind die wesentlichen Eigenschaften/Ziele von Groovy?

Antwort:

- Überladen und überschreiben von Operatoren möglich
- Vereinfachung von Listen, Mengen, assoziative Arrays (Maps)
- Nutzung von Closures
- Alle Datentypen sind Objekte

- Dynamische Typisierung
 - Verzicht auf unnötige Syntax wie Semikola, main-Funktion, return-Statement
 - Das Ziel von Groovy ist es Sprachfunktionen mit vergleichbarer Wirkung für die Java-Plattform bereitzustellen, aber zugleich das Java-Objektmodell zu befolgen und die Perspektive eines Java-Programmierers einzunehmen
5. Ist Groovy interpretiert oder kompiliert?
- Antwort:**
Groovy kann wahlweise kompiliert oder direkt als Skript ausgeführt werden, dabei wird beim ersten Aufruf implizit kompiliert
6. Was mache ich wenn ich Java-Code verwendet habe? -> Kann ich es weiternutzen?
- Antwort:**
Ja, denn (fast) jedes Java-Programm ist auch ein gültiges Groovy-Programm. Ausnahme bei der Division von Ganzzahlen.
7. Was sind die Vor- und Nachteile von Groovy und Grails? (Im Vergleich zu einer anderen Programmiersprache wie z.B. PHP)
- Antwort:**
Vorteile:
- Leichter erlernbar
 - Datenbankprogrammierung wird vereinfacht
 - Integration von Klassenbibliotheken
 - Testbarkeit
- Nachteile:
- Performanz durch die dynamische Typisierung
8. Wodurch unterscheidet sich Groovy zu Java? (Syntax)
- Antwort:**
- alle Datentypen (auch elementare) sind Objekte
 - Datentyp kann bei Deklaration weggelassen werden (dynamische Typisierung)
 - vereinfachte Nutzung von Listen, Mengen, assoziative Arrays (Map), Ranges
 - Überladen und Überschreiben der Operatoren
 - Closures
 - Verzicht auf "unnötige" Syntax wie Semikolons, main-Funktion ...
9. Warum verwendet man Groovy und nicht gleich Java?
- Antwort:**
Groovy ist besser an Webentwicklung angepasst.
10. Wie sieht es mit den if-Bedingungen in Groovy aus?
- Antwort:**
Bedingungsausdruck braucht kein Boolean liefern, sonst wie in Java
11. Was hat es mit den „Ranges“ auf sich?
- Antwort:**
Kann man beim Zugriff auf Listenelemente nutzen -> `println namen[0..2]`
benötigt aufzählbaren Datentyp, der
- Interface Comparable implementiert
 - der Operatoren ++ und --, definiert
12. Erklären Sie mir die Regulären Ausdrücke. Was ist das überhaupt? -> Beispiel Kreditkartennummer als pattern schreiben können.
Erklären Sie den Find-Operator, Capture-Group und Match-Operator

Antwort:

Ein Regulärer Ausdruck ist eine Möglichkeit Suchmuster zu definieren, die einen bestimmten Ausdruck suchen und dann true oder false zurückgeben.

```
creditNumberTest = „1234123412341234“
```

```
creditNumberResult = creditNumberTest =~ /^d\{16}/
```

```
println creditNumberResult -> true
```

Find-Operator: analysiert und zerlegt String anhand eines regulären Ausdrucks

Match-Operator: Testet kompletten String auf Einhaltung eines regulären Ausdrucks

Capture Group: Ein regulärer Ausdruck wird in Capture Group aufgeteilt um einen Teilausdruck zu finden.

13. Was hat es sich mit den 3 Strategien greedy, reluctant und possessive auf sich? Wann nehme ich welche?

Antwort:

- greedy: Erst mal maximal viele passende Zeichen aus String nehmen, notfalls wieder welche zurückgeben
- reluctant: Erst mal minimal viele passende Zeichen aus String nehmen, notfalls noch welche dazu nehmen
- possessive: Maximal viele passende Zeichen aus String nehmen und eisern verteidigen, selbst wenn Pattern scheitert

14. Was hat es mit den benannten Parametern auf sich?

Antwort:

Zuordnung formale « aktuelle Parameter über ihren Namen

Definition einer Funktion mit benannten Parametern

- genau ein formaler Parameter vom Typ Map
- nimmt sämtliche benannten Parameter als Assoziatives Array auf

15. Was ist denn ein Groovy Bean?

Antwort:

Groovy Beans sind Groovy Klassen mit

- parameterlosem Konstruktor
 - 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

16. Was ist ein Closure? Schreiben Sie eine Closure, die eine Fakultät berechnet (rekursiv)

Antwort:

- 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, " }

17. Für was verwendet man Closures? Was kann ich damit anfangen?

Antwort:

- als Parameter an Funktions- und Methodenaufrufe übergeben

18. Nennen Sie ein paar Beispiele wo Closures vorkommen.

Antwort:

- Callback-Methoden asynchroner Aufrufe
- Ereignishandler (vgl. ActionListener in Swing)
- Vergleichsfunktion in Sortieralgorithmen
- Aufruf einer als Parameter übergebenen Funktion für jedes Element einer Collection

19. Wie kann ich Parameter bei Closures übergeben?

Antwort:

Am Anfang der Closure können formale Parameter definiert werden:

- Syntax:

{ param1, param2 -> Closuresrumpf }

Parameterlose Closures (ohne Angabe von ->)

- können trotzdem mit einem Parameter aufgerufen werden

--> Parameterwert im Rumpf in Variable it verfügbar

20. Kann man Closures überladen?

Antwort:

Man kann Closures nur mit Multimethoden überladen dazu benötigt man den Adressoperator &

Bsp.:

```
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
```

21. Wie sieht es mit den Datentypen in Groovy aus? Sind Integer auch Objekte?

Antwort:

Alle Datentypen sind in Groovy Objekte

22. Wie sieht es mit den Operatoren beim überladen aus? Wofür kann man sowas verwenden?

Antwort:

Sämtliche Operatoren sind in Groovy überladbar

23. Nennen Sie Beispiele bei denen Operatoren sinnvoll überladen sind?

Antwort:

a + b ⇨ a.plus(b)

a - b ⇨ a.minus(b)

a * b ⇨ a.multiply(b)

a / b → a.div(b)

24. Was hat es mit dem „GString“ auf sich?

Antwort:

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}"`

25. Was ist der Unterschied von Strings mit Slash, Anführungszeichen und Hochkomma? Wann verwende ich welche von denen?

Antwort:

Beim Slashed-String muss weder das Stringliteral mit Apostroph noch das mit Anführungszeichen escaped werden. Beim Stringliteral mit Apostroph oder Anführungszeichen muss, dass jeweilige zweite Vorkommen dann mit Backslash escaped werden.

26. Welche Vereinfachung hat Groovy für Collections und warum?

Antwort:

- bauen auf bekannten Java-Collections auf

- erweitern diese um "syntaktischen Zucker", um die Nutzung zu vereinfachen:

- statische Initialisierung mit Array-Notation:

`def list = ["Mueller", "Meier", "Schulze"] // [Mueller, Meier, Schulze]`

- Zugriff auf einzelnes Element mit [und]:

`println list[1] // Meier`

`println list[-1] // Schulze`

- Zugriff auf Bereich mit Range-Indizierung:

`def sublist1 = list[0..1]; // [Mueller, Meier]`

`def sublist2 = list[-1..-2] // [Schulze, Meier]`

27. Assoziative Arrays werden auch als Map bezeichnet. Wie funktionieren diese?

Antwort:

- bestehen aus Schlüssel-Werte-Paaren, bestehend aus

- alphanumerischem Schlüssel (Key)

- beliebigem Wert (Value)

- 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

28. Wie sieht es mit den Konstruktoren in Groovy aus?

Antwort:

Konstruktoren wie in Java, aber:

- Sichtbarkeit ist implizit public wenn nicht anders angegeben.

- wenn kein anderer Konstruktor definiert ist gibt es einen impliziten Konstruktor mit benannten formalen Parametern → Bsp. `Klassenname(attrName1: attrValue1, attrName2: attrValue2)`

- Attribute ohne Initialisierungswerte werden null → impliziter Parameterlosen Konstruktor wie in Java

29. Was ist ein Expando? Wofür kann man das gebrauchen?

Antwort:

Expando: Dynamisches Objekt

- instanziiert aus der Groovy-Klasse Expando
- beliebige Attribute hinzufügen durch
 - Angabe als benannter Parameter im Konstruktoraufbau
 - bloßes Beschreiben eines noch nicht vorhandenen Attributs

30. Was ist Meta-Programmierung? Was kann ich damit anfangen?

Antwort:

- 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

31. Was ist Grails?

Antwort:

Grails ist ein Web Application Framework für die Programmiersprache Groovy.

32. Warum hat man Grails entwickelt?

Antwort:

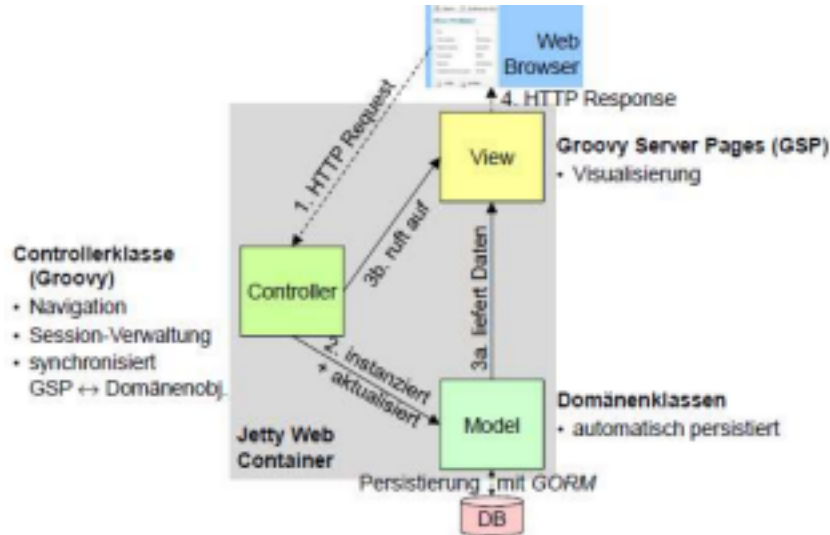
33. Was sind typische Prinzipien die sich in Grails wiederfinden?

Antwort:

- Convention over Configuration
 - fest vorgegebene Ordner für
 - Domänenklassen (Model)
 - Controller
 - Views
- Don't repeat yourself
 - Vermeidung unnötiger Code-Redundanzen
 - Automatische Generierung aus Domänenklassen
- Agilität
 - wahlweise statische oder dynamische Typisierung
 - kurze Entwicklungszyklen
- 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)

34. Wie funktioniert das Zusammenspiel mit MVC?

Antwort:



- Die Steuerung(Controller) nimmt Anfragen von Benutzern entgegen und vermittelt zwischen dem Modell und der Präsentation(View).
- Die Präsentation(View) ist für die Aufbereitung der von der Steuerung erzeugten Daten zuständig. Hier dienen die Groovy Server Pages als Visualisierung.
- Das Modell(Model) enthält die Datenstruktur sowie die Geschäftslogik der Anwendung. Es wird in Domain-Klassen abgelegt und automatisch von Hibernate in ein relationales Datenbankschema überführt.

- Model: Domänenklassen mit automatischer Persistenz (GORM / Hibernate)
- View: Groovy Server Pages (GSP), typischerweise HTML-basiert
- Controller: Eigene Instanz für jeden Request

35. Wie funktioniert das statische und dynamische Scaffolding? Bin ich darauf angewiesen den Container zu belegen?

Antwort:

Dynamisches Scaffolding

- Controller verwaltet genau 1 Domänenklasse
 - gleicher Name wie Domänenklasse, Endung Controller
 - Quellcode beinhaltet nur Zuordnung zur Domänenklasse, sonst nichts
- Controller und Views dynamisch zur Laufzeit erzeugen, ohne Quellcode

Statisches Scaffolding

- Groovy-Quellcode für Controller aus Domänenklassen generieren
- Groovy Server Pages für Views aus Domänenklassen generieren
- 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

36. Was ist eine Domain-Klasse? Was kann ich alles darunter deklarieren?

Antwort:

- liegen grundsätzlich im Ordner Domain Classes
- 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

37. Wo kommt das Datenmodell her? Wird es automatisch generiert oder aus der Domain-Klasse?

Antwort:

Es wird aus der Domain-Klasse mit den dort verwendeten constraints generiert.

38. Schreiben Sie ein paar typische Constraints. -> nicht null, bestimmte range

Antwort:

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)

39. Was steht in der GSP drin?

Antwort:

Im Grund die ganze Visualisierung, d.h. Form, Textarea, Links, Input-Felder

40. Was hat es mit dem „Flash“ auf sich?

Antwort:

- flash-Objekt automatisch erzeugt
 - Controller legt Meldung als String in Attribut flash.message
 - unmittelbar nächste GSP
 - zeigt Meldung an
 - danach wird flash.message automatisch gelöscht
 - Meldung wird nur einmal angezeigt

41. Was sind die Aufgaben des Controllers?

Antwort:

- 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: `def scaffold=Professor`

42. Was versteht man unter Controller?

Antwort:

Ein Grails-Controller ist eine Klasse, die für die Beantwortung von Anfragen an die Anwendung verantwortlich ist.

43. Was sind die einzelnen Controller-Actions? Was machen diese?

Antwort:

index: Einstiegsseite, listet alle Instanzen (Datensätze) der Domänenklasse auf

save: speichert neu erstellte Instanz in DB

show: zeigt einzelne Instanz im Detail an

update: schreibt Änderungen an einer einzelnen Instanz in DB

delete: löscht einzelne Instanz aus Datenbank

edit: zeigt einzelne Instanz zum Editieren an

create: zeigt Formular zum Erstellen einer Instanz an

44. Wo bekommt die Controller-Action ihre Daten her?

Antwort:

- Daten können über GORM aus der Datenbank geholt werden.
- Request-Parameter kommen die Daten aus der URI
- FORM-Daten kommen aus dem Request Body
- Bei einer Session bekommt man durch eine Map die Session spezifische Daten

45. Wie liefert die Controller-Action die Daten?

Antwort:

Über eine Daten-Map die im entsprechenden Controller implementiert ist.

46. Wie wird die Controller-Action beim Aufruf ausgewählt?

Antwort:

In der GSP kann man mit `<g:link>` und den Attributen `controller` und `action` den gewünschten Controller und seine Aktionsmethode aufrufen.

47. Wie kann ich die Controller-Action auslösen?

Antwort:

48. Wie kann ich Parameter an die Controller-Action zurückgeben?

Antwort:

Entweder über URI mit `params(Map)` oder über den Request-Body bekommt man die FORM-Parameter

49. Wie kommen die Parameter in die Controller-Action rein?

Antwort:

- Controller liefert Model
 - Aktions-Methode gibt assoziatives Array mit Variablenwerten für GSP zurück
 - aufzurufende View: Name der Controller-Aktion + Endung + `.gsp`
- Zugriff auf die Variablenwerte in der GSP
 - mit `${variablenname}`
 - Mengenklammern dürfen nicht weggelassen werden

50. Wie sieht eine URI typischerweise aus?

Antwort:



51. Was hat es mit den Umgebungsvariablen auf sich?

Antwort:

controllerName □ Name des Controllers Bsp: professor

actionName □ Name der Controller-Aktion Bsp: index

controllerUri □ URI des Controllers im Projektkontext Bsp: /professor

actionUri □ URI der Controller-Aktion im Projektkontext Bsp: /professor/index

params □ Assoziatives Array mit HTTP-Parametern Bsp: [action:index, format:null,

controller:professor]

session □ HttpSession, Session-Variablen als assoziatives Array

flash □ Objekt zum Ablegen von Fehlermeldungen Bsp: [:]

log □ Logger zum Schreiben von Logfiles

52. Was ist Content Negotiation?

Antwort:

• 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

53. Welche Methoden gibt es in der Domain-Klasse? Welche definieren Sie, und wo kommen die anderen her?

Antwort:

Man erbt von der Klasse Object, z.B. toString

54. Kann ich Methoden in der Domain-Klasse auch dynamisch anhängen?

Antwort:

Ja, mit MethodMissing werden zur Laufzeit die fehlenden Methoden automatisch generiert.

55. Kann man auch Vererbung abbilden? Welche Möglichkeiten gibt es?

Antwort:

• GORM beherrscht Vererbung von Domänenklassen

- ganz normale extends-Klausel

- Attribute und Beziehungen werden vererbt

- Constraints werden vererbt

- können in abgeleiteter Klasse ergänzt / überschrieben werden

56. Wie kann ich Daten aus der Grails-Anwendung rausschleusen und wie kann ich sie reingeben?

Antwort:

Entweder mit SOAP oder mit RESTful Web-Service

57. Was ist REST?

Antwort:

- REST ist ein Architekturstil für Netzwerk-basierte Systeme

- REST ist eine grundsätzliche Denkweise wie man Softwaresysteme entwerfen kann
- **Representational State Transfer**

58. Was ist eine Ressource?

Antwort:

- Ressourcen liegen oft vor als
 - strukturierte Daten in einer Datenbank
 - Objekte einer Programmiersprache

59. Wie kann ich in Grails einen RESTful Service programmieren?

Antwort:

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!

Fragen an Johannes :

zeichne MVC

was ist controller domain gsp

was ist rest

wie geht es wie sieht so ein http aufruf aus

welche http verben gibts

was ist metaprogrammierung

wofür domainklassen

wie ruft der client eine action ausm controller auf

was ist eine map

hat eine domainklasse methoden ? wenn ja welche da sie keine hat bis auf toString wie kann man dann sachen aus der datenbank abfragen? -> metaprogrammierung die methode wird dynamisch erstellt

schreibe eine domainklasse

was ist or mapping ... welche technologie (hibernate)