Wie man mit Spring Anwendungen bauen kann

Eberhard Wolff
Managing Director
Interface21 GmbH
Interface21 – Spring from the Source

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Interface21

- Produkte u.a. Spring Framework
- Spring from the Source
- Consulting, Training, Support
- Kürzlich: 10 Mio $ Venture Capital von Benchmark (MySQL, JBoss, Red Hat)

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Über mich

- Managing Director Interface21 Deutschland
- Java Champion
- Fokus: Java EE, Spring ...
- Autor (z.B. Java Magazin, 3 Bücher...)
  - Z.B. Spring (Neu: 2. Auflage)

- http://www.spring-buch.de/

- Blog: http://JandlandMe.blogspot.com/

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Überblick

- Zwei Worte über Spring

- Typische Spring Architekturen

- Layering

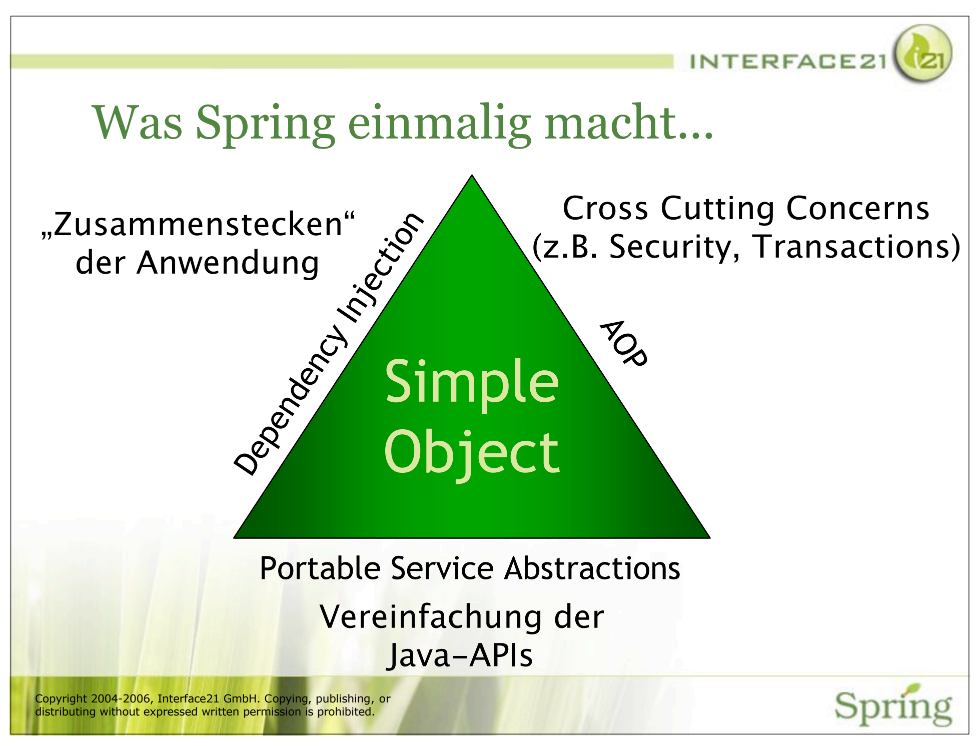
- Ist das eine objektorientierte Architektur?

- Fazit

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

INTERFACE21
ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK
Zwei Worte über Spring
Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.
Spring


INTERFACE21
Was Spring einmalig macht...
„Zusammenstecken“ der Anwendung
Dependency Injection
Cross Cutting Concerns (z.B. Security, Transactions)
AOP
Simple Object
Portable Service Abstractions
Vereinfachung der Java-APIs
Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.
Spring

INTERFACE21

Dependency Injection

```
public class OrderService implements IOrderService {
    private ICustomercDAO customerDAO;

    public void setCustomerDAO(ICustomercDAO customerDAO) {
        this.customerDAO = customerDAO;
    }
}

<bean id="orderService" class="order.service.OrderService">
   <property name="customerDAO" ref="customerDAO " />
</bean>

<bean id="customerDAO" class="customer.dao.CustomerDAO" />
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

INTERFACE21
ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK
Das ist ja XML!
Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.
Spring

INTERFACE21
Aber das ist ja XML
• Einfaches XML
• Lesbar und leicht editierbar
• Mit Tool-Support
• Aber: Man kann auch stattdessen Java & Annotationen verwenden
• Neues Projekt: Spring JavaConfig
Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.
Spring

INTERFACE21

Spring JavaConfig

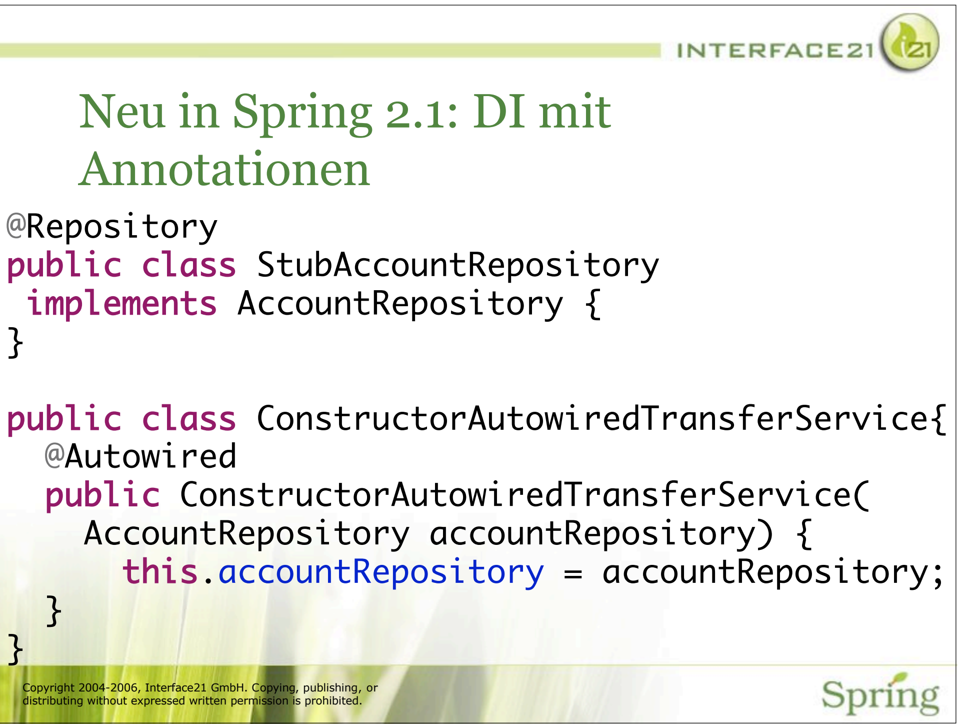
```
@Configuration
public class SpringConfiguration {
  @Bean Ein Singleton...
  public ICustomerDAO customerDAO() {
    return new CustomerDAO(dataSource());
  }

  @Bean
  public IOrderService orderService() {
    return new OrderService(customerDAO());
  }
}
```

- ...und Spring 2.1 hat auch einiges...
- Spring 2.1: Frisch vom Entwickler

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Warum Dependency Injection?

- Klassen sind unabhängig von der Umgebung
  - Im Gegensatz zum Beispiel zu JNDI
  - ...oder der Benutzung von Factories
- Leichter testbar
  - Mocks statt eigentlichen Klassen zuweisbar
- Leichter wiederverwenbar
  - Klare Abhängigkeiten
- Konfiguration von Objektnetzen
  - Verbessert z.B. Nutzung des Strategy-Patterns

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

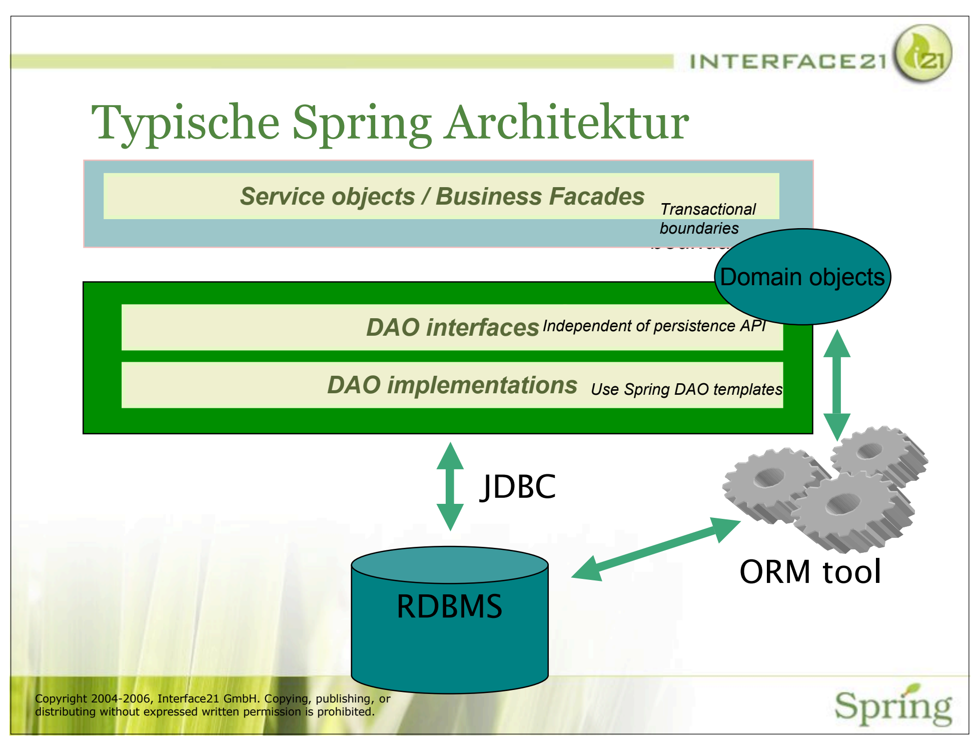
Was einem Spring also an die Hand gibt...

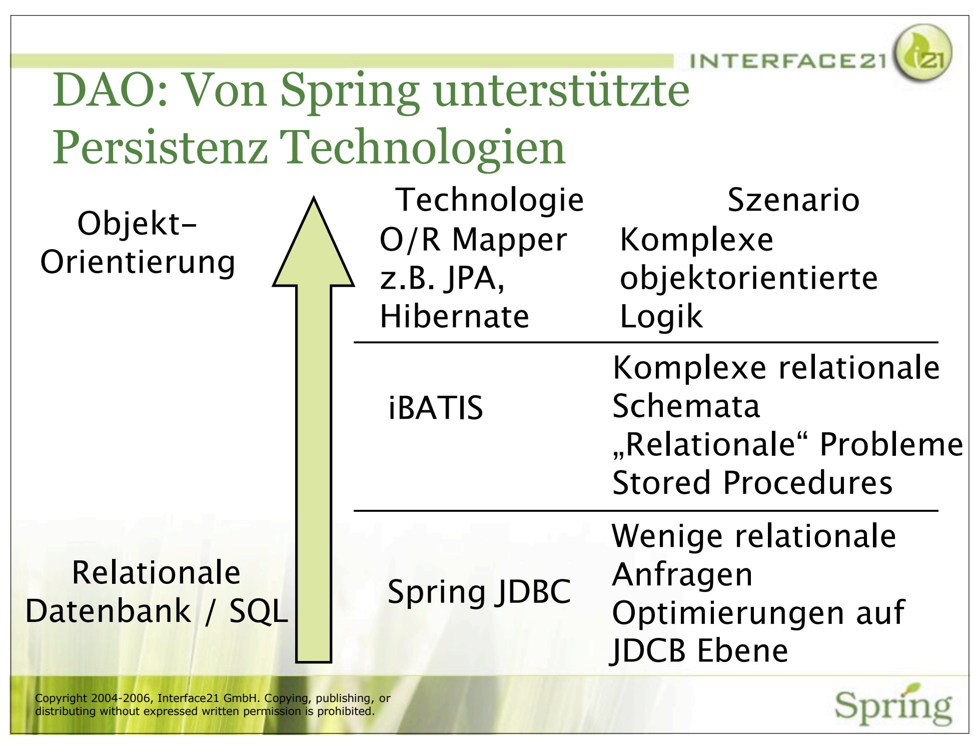
- Möglichkeit zur Strukturierung der Anwendung
- Elemente sind „normale“ Java *Klassen*
  - Keine Abhängigkeiten von Spring
  - Konstruktor- oder Setter-Dependency Injection
  - Integration von Factories usw. möglich
  - Integration anderer Bibliotheken möglich
  - Migration vorhandener Anwendungen möglich
- Feingranularer als eine Architektur
- Wie sieht darauf aufbauend eine Anwendung aus?

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

INTERFACE21
ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK
Typische Spring Architekturen
Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.
Spring


INTERFACE21
Typische Spring Architektur
Service objects / Business Facades
Transactional boundaries
Domain objects
DAO interfaces
Independent of persistence API
DAO implementations
Use Spring DAO templates
JDBC
ORM tool
RDBMS
Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.
Spring



Warum DAOs?

- Ziel nicht so sehr die mögliche Migration der Persistenz-Technologie...
- ...sondern Mocking für Tests (DAO-Interfaces)

- Mit Spring kann man mehrere Technologien in der DAO-Schicht nutzen (90% O/R Mapper + 10% JDBC)
- Dabei notwendig: Durch Spring vereinheitlichten Exceptions und einheitliches Transaktionshandling

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Service Objects / Business Facades

- Bieten fachliche motivierte Services an
- Steuern Transaktionen (z.B. über Annotationen)
- Auch Sicherheit kann hier ansetzen

- Technologie-unabhängig

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Domänen-Objekte

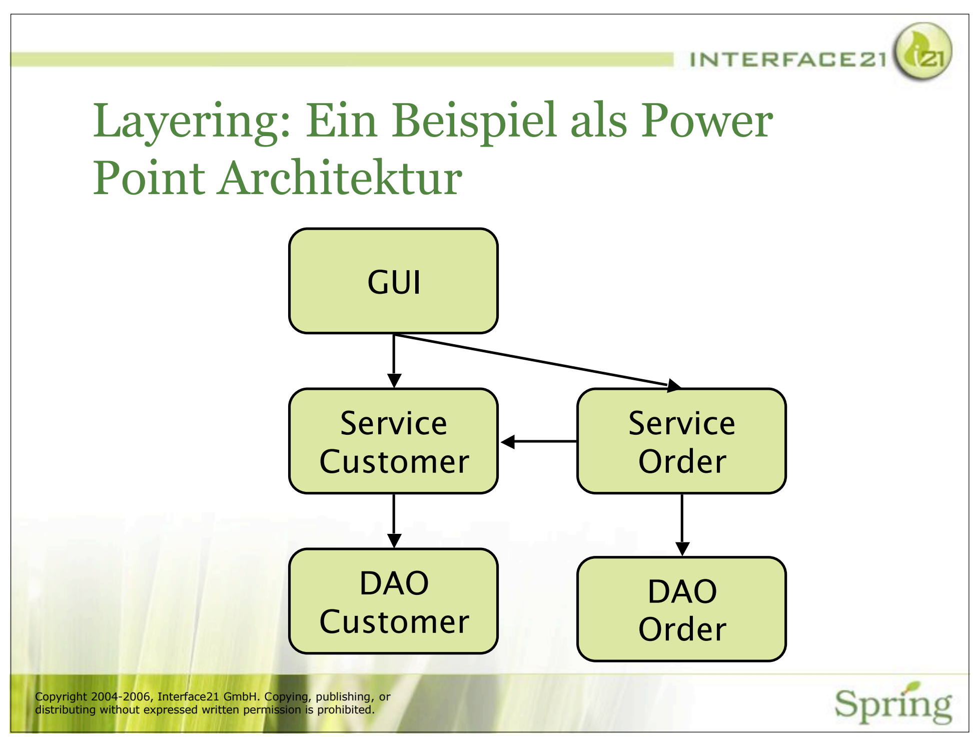
- Modellieren fachliche Konzepte
- Damit enthalten sie Daten
- ....und Logik?

- Können Layer-übergreifend genutzt werden
- Werden nicht von Spring erzeugt

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

INTERFACE21
ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK
Layering
Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.
Spring


INTERFACE21
Layering: Ein Beispiel als Power Point Architektur
GUI
Service Customer
Service Order
DAO Customer
DAO Order
Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.
Spring

Wie werden Layer zu Software-Artefakten?

- Spring hat scheinbar „nur“ Klassen als Komponenten
- Zu feingranular

- Also: Spring Konfigurationen
- Entweder durch mehrere Konfigurations-Dateien oder `ApplicationContext`-Hierarchien
- Macht Konfiguration auch übersichtlicher

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Layer mit Spring-Konfiguration: <import>

```
<beans>
    <import resource="dao-customer.xml" />
    <import resource="dao-order.xml" />
    <import resource="services-customer.xml" />
    <import resource="services-order.xml" />
    <import resource="gui.xml" />
</beans>
```

Allerdings können alle Spring-Beans alle anderen Spring-Beans sehen und das Layering verletzen.

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Oder: Spring-OSGi

- OSGi definiert ein Komponentenmodell (*Bundles*)
  - Bundles bieten *Services* an
  - Können individuell gestartet und geupdatet werden
- Ziel des Spring-OSGi Projekts: OSGi als grobgranulare Komponenten-Infrastruktur über Spring
  - Bundles können gestartet, gestoppt und geupdated werden
  - ...und damit auch individuelle Updates und Wartungen dieser grobgranularen Komponenten möglich
- Zur Zeit in 1.0M1
- http://groups.google.com/group/spring-osgi
- http://www.springframework.org/osgi

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK

INTERFACE21

Nur Spring-Beans - Was ist mit dem Code?

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

INTERFACE21

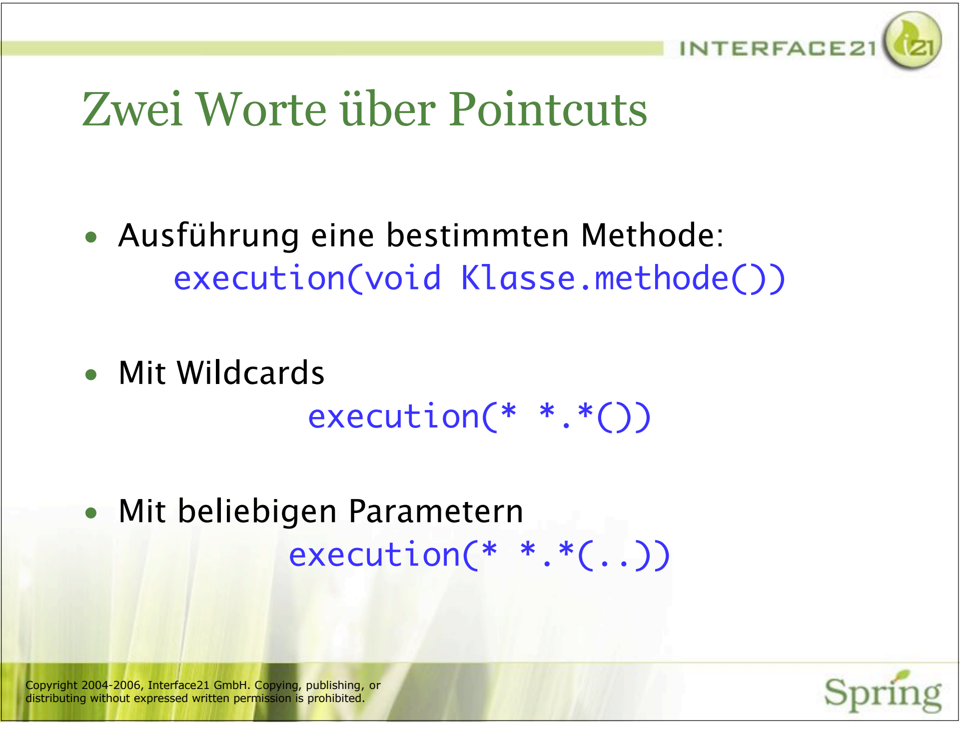
Code: Service

```
package order.service;

public interface IOrderService {
  void order(ShoppingCart shoppingCart)
   throws OrderException;
}
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Man kann Layer als Spring 2.0 AOP Pointcuts definieren...

- Pointcuts definieren, wo Code durch Aspekte erweitert werden soll

```
@Pointcut("execution(* order.service.*.*(..))")
public void orderServiceLayer() {}
@Pointcut("execution(* order.dao.*.*(..))")
public void orderDaoLayer() {}

@Pointcut("execution(* *.service.*.*(..))")
public void serviceLayer() {}
@Pointcut("execution(* *.dao.*.*(..))")
public void daoLayer() {}
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

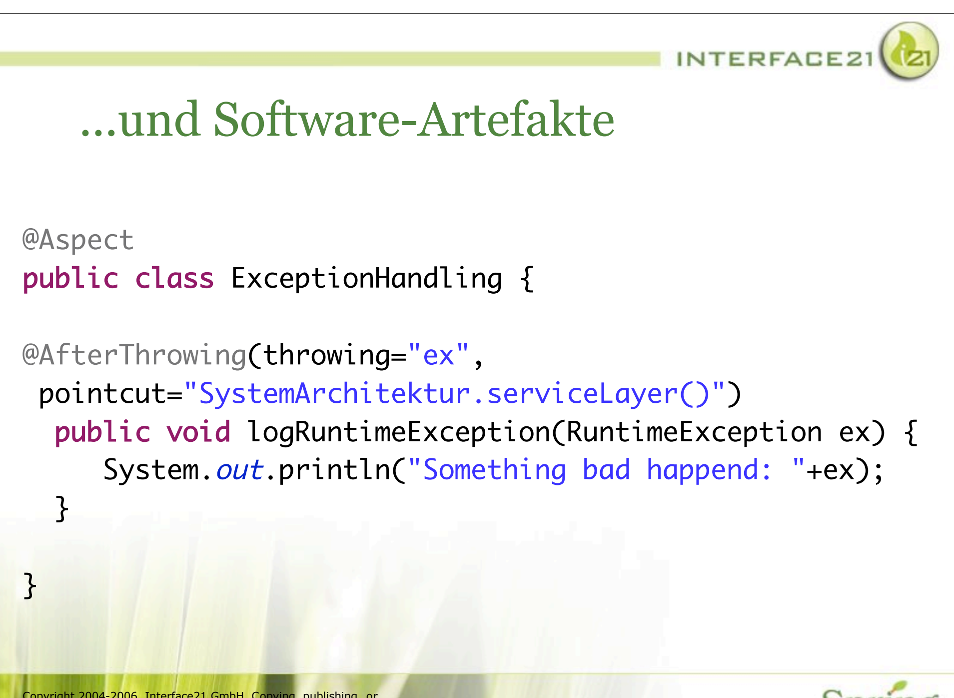
INTERFACE21

Power Point Architektur

- „Runtime Exceptions werden im Service Layer gefangen und geloggt.“

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring



Power Point Architektur

- „Die Benutzung von JDBC ist nur im DAO Layer erlaubt.“
- „Exceptions müssen geloggt werden. Aufrufe von `printStackTrace()` sind nicht zulässig.“

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

INTERFACE21

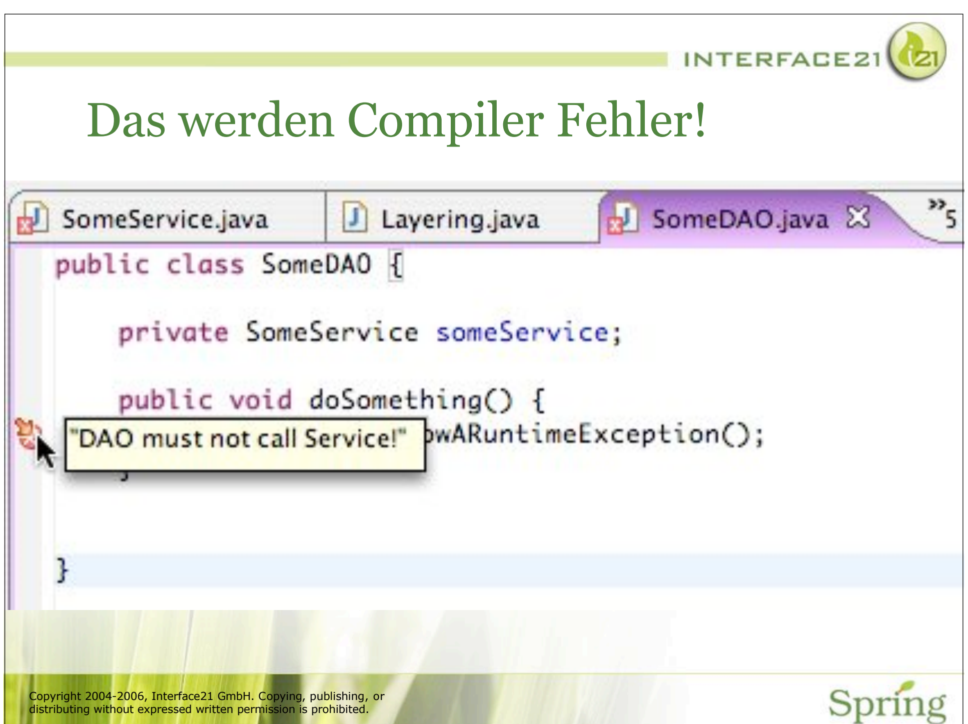
...und Software Artefakt (AspectJ)

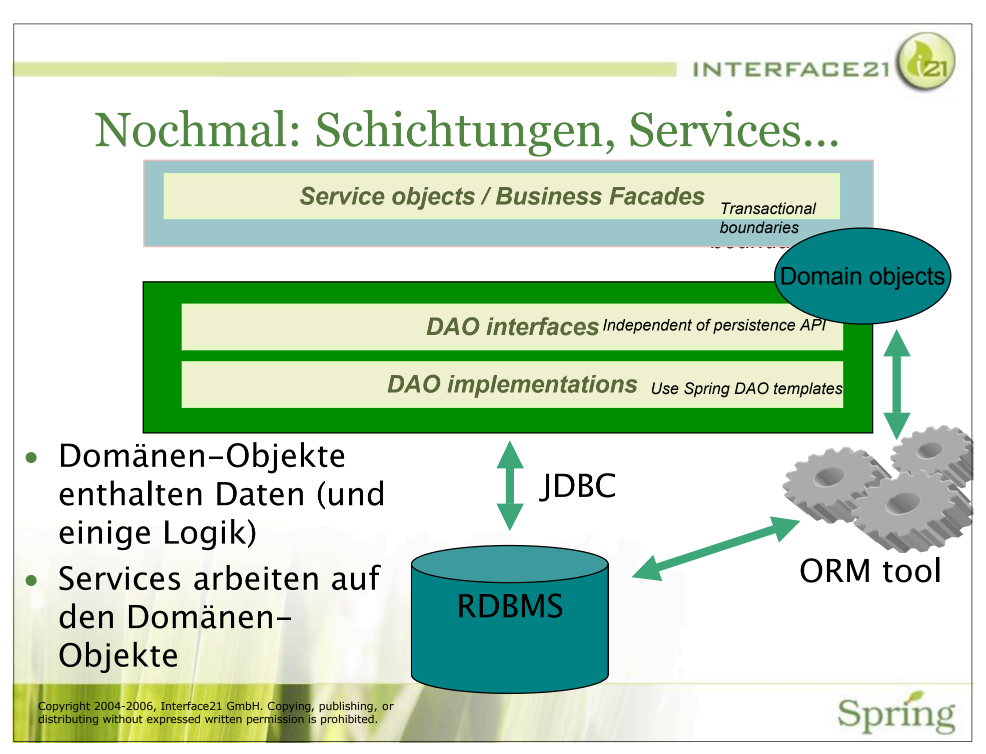
```
@DeclareError(" (call(* java.sql.*.*(..)) && " +
  "!within(*.dao.*) ) ")
  public static final String JdbcOnlyInDAOs =
   "JDBC only in DAOs!";


@DeclareError("call(void " +
 "java.lang.Throwable+.printStackTrace())")
    public static final String NoPrintStackTrace =
   "Please log exception!";
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring




INTERFACE21
Nochmal: Schichtungen, Services...
Service objects / Business Facades
Transactional boundaries
Domain objects
DAO interfaces
Independent of persistence API
DAO implementations
Use Spring DAO templates
• Domänen-Objekte enthalten Daten (und einige Logik)
• Services arbeiten auf den Domänen-Objekte
JDBC
RDBMS
ORM tool
Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.
Spring

INTERFACE21
Ein Beispiel...
private IOrderService order;
public void setOrder(IOrderService order) {
this.order = order;
}
public void doSomething() {
ShoppingCart shoppingCart = new ShoppingCart();
...
order.order(shoppingCart);
}
Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.
Spring

Objekt-Orientierung

- Ein Objekt hat einen internen Zustand (aka Daten)
- ...den es nicht nach außen veröffentlicht (Information Hiding)
- Der Zustand kann intern anders dargestellt werden

- Wichtiger als die Daten sind die Umgangsformen (Methoden)
- Z.B. das Bestellen eines `ShoppingCart`s

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Wie es also eigentlich sein sollte...

```
public void doSomething() {
  ShoppingCart shoppingCart = new ShoppingCart();
  ...
  shoppingCart.order();
}
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Services und Domänen-Objekte

- Domänen-Objekte enthalten vor allem Daten
- Services die Logik
- Zumindest Instanz-Übergreifende Logik ist da gut aufgehoben
- Aber Domänen-Modelle neigen dazu, keine Logik zu enthalten („blutleer“)
- Anaemic Domain Model

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Spring 2.0 hilft

- @Configurable: Dependency Injection bei Klassen, die per `new` erzeugt werden (z.B. Domänen-Objekte)

```
@Configurable
public class OrderableShoppingCart {
  private IOrderService orderService;
  public void setOrderService(IOrderService o) {
    this.orderService = o;
  }

  public void order() throws OrderException {
    orderService.order(this);
  }
}
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Hilft das?

- Eine objektorientierte *Facade* für eine Service-Architektur
- Es sieht also „nur“ anders aus
- Aber ist das ganze überhaupt ein Problem?
- Objekt-Orientierung ist kein Wert an sich
- ...sondern nur ein Technik

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Problem

- Normalerweise kann man von ShoppingCart erben
- ...beispielsweise einen RebatedShoppingCart
- ...der bei order() noch einen Rabatt abzieht

- Das kann man mit den Services nicht so elegant modellieren!

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

INTERFACE21

Code für den RebatableShoppingCard mit Services

```
public class OrderService implements
  IOrderService {

  public void order(OrderableShoppingCart cart)
   throws OrderException {
    if (cart instanceof RebatedShoppingCart) {
      // Logik für den Rabatt
    }
  }
}
```

instanceof ist ein Code-Smell

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Objekt-orientierter Code für den RebatedShoppingCart

```
package demo;

public class RebatedShoppingCart
 extends ShoppingCart {

  public void order() throws OrderException {
    // Logik für den Rabatt
    super.order();
  }

}
```

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

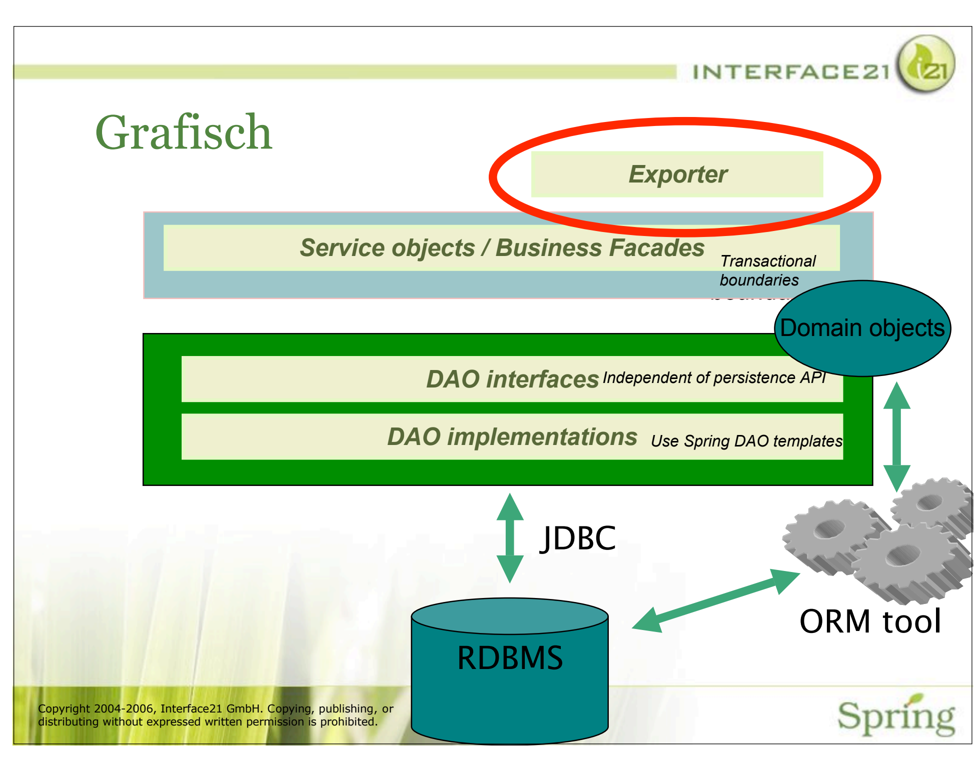
Aber...

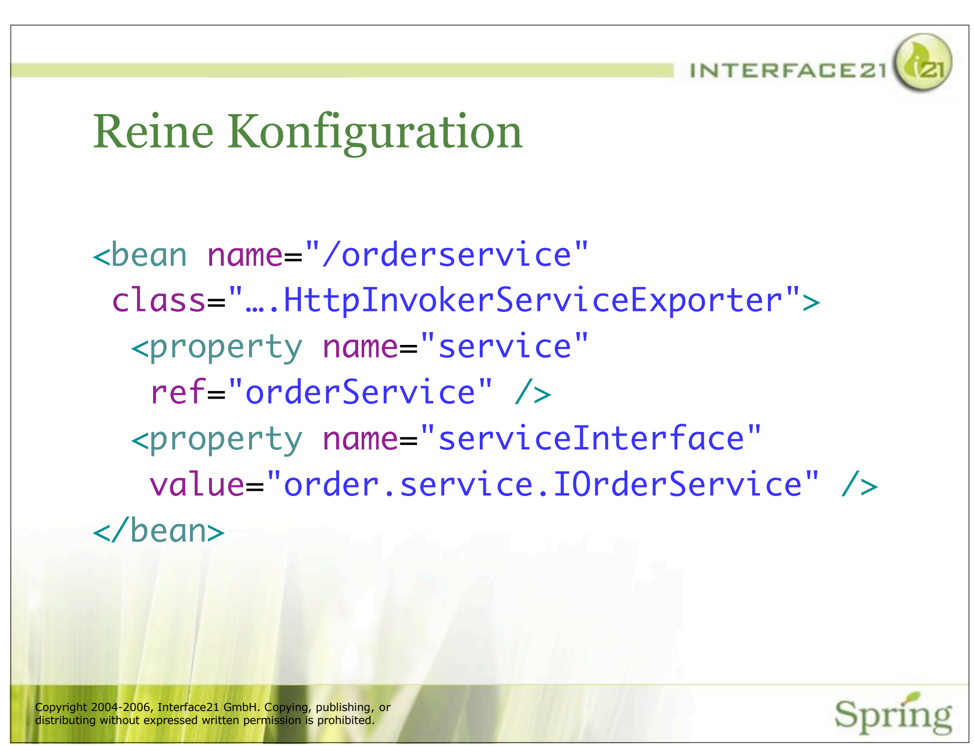
- ...man kann die Service Schicht sehr einfach an Clients im Netz exportieren
- Wenn die Domänen-Objekte reine Datencontainer sind, können sie als Data Transfer Objects (DTOs) zum Client geschickt werden
- ...der dann kein Java-Client sein muss
- ...da er keine Logik aufrufen können muss

- Die Architektur ist leicht verteilbar
- Mit Spring ist die technische Implementierung trivial

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring


INTERFACE21
Grafisch
Exporter
Service objects / Business Facades
Transactional boundaries
Domain objects
DAO interfaces
Independent of persistence API
DAO implementations
Use Spring DAO templates
JDBC
ORM tool
RDBMS
Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.
Spring


INTERFACE21
Reine Konfiguration
<bean name="/orderservice"
class="….HttpInvokerServiceExporter">
<property name="service"
ref="orderService" />
<property name="serviceInterface"
value="order.service.IOrderService" />
</bean>
Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.
Spring

Sind Domänen-Objekte DTOs?

- Domänen-Objekte sind die interne Datenrepräsentation
- ...die man allen anderen zeigt
- Exhibitionismus statt Information Hiding...

- Clients müssen bei Änderungen der internen Datenstrukturen nachgezogen werden
- ...oder die Datenmodellierung der Clients zieht sich in den Server

- Unschön

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring

Und überhaupt...

- Man sollte eigentlich Contract-First arbeiten

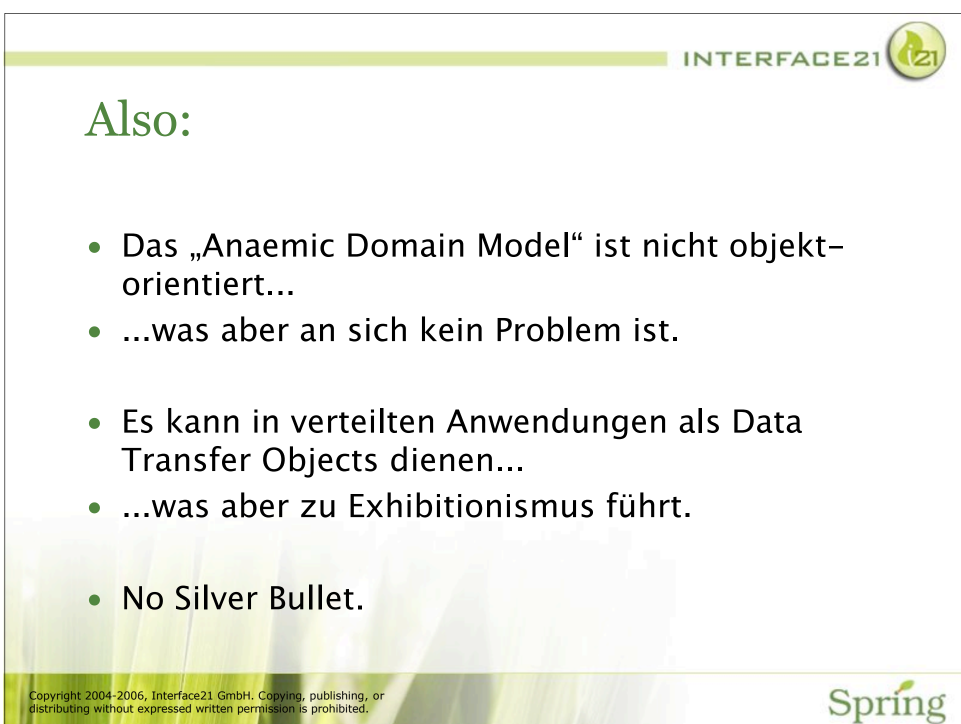
- Schnittstelle zwischen Client und Server definieren...
- ...bevor man die Implementierung des Clients und Servers vornimmt

- Eigentlich triviale Einsicht: Erst Schnittstelle definieren
- Man kann die Schnittstelle plattformneutral (z.B. WSDL) definieren
- Oder mit XML Schema (Spring Web Services)

- Dann braucht man aber eine Adapter-Schicht

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring


INTERFACE21
Also:
Das „Anaemic Domain Model“ ist nicht objekt-orientiert...
...was aber an sich kein Problem ist.
Es kann in verteilten Anwendungen als Data Transfer Objects dienen...
...was aber zu Exhibitionismus führt.
No Silver Bullet.
Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.
Spring

INTERFACE21
ENTERPRISE DEVELOPMENT SERVICES DIRECT FROM THE CREATORS OF SPRING FRAMEWORK
Fazit
Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.
Spring

Fazit

- Spring bietet durch Dependency Injection ein feingranulares Komponentenmodell
- Darauf aufbauend entstehen Layer (Service, DAO)
- Auch Layer können mit Dependency Injection Software Artefakte werden
- Layer können mit AOP Semantik bekommen
- Das Vorgehen ist nicht zwangsläufig objektorientiert
- ...aber auch nicht objektorientierte Entwürfe haben Vorteile

Copyright 2004-2006, Interface21 GmbH. Copying, publishing, or distributing without expressed written permission is prohibited.

Spring