*Fast and Furious:*

# Professionelle Java Entwicklung mit Maven und Spring

Fachhochschule Köln
Cologne University of Applied Sciences

- Lars Haferkamp, cyber:con
  - Diplom Informatiker
  - Consultant in der Software-Entwicklung
- Matthias Richter, cyber:con
  - M.Sc. Medien-Informatik
  - Consultant in der Software-Entwicklung

# Eure Erwartungen?

# Unsere Vorstellungen

... wissen, was Maven ist, wissen was Spring ist

...technische Projektstrukturen erstellen

...Programmkomponenten entkoppeln

Unsere Vorstellungen

Dies ist ein *Workshop*...



- Ablauf

# 1. Tag: Maven

| | Inhalt | Dauer |
|---|---|---|
| 1. Teil: Praxis | Praktischer Direktstart:<br>· gemeinsamer Kick-Start einer eigenen Anwendung durch Maven-Archetype<br>· Hallo, Welt! mit Maven<br>· Eclipse Projekt erstellen<br>· etc. | 1h |
| 2. Teil: Theorie | · Einführung in Apache Maven<br>· Der Maven Lifecycle<br>· Die Maven Projektstruktur<br>· Dependency Management | 40 Min. |
| 3. Teil: Praxis | · Einbindung einer Dependency: Logging<br>· Einbindung eines Plugins: maven-jar-plugin für test-jar<br>· Erweitern der Projektstruktur um resources Verzeichnis mit log4j.properties<br>· Einbindung eines WS und testen der Anwendung | 2 h |
| Mittagspause | | |



| | Inhalt | Dauer |
|---|---|---|
| Fortsetzung<br>3. Teil | · Abschließen der Assignments<br>· Gemeinsamen Arbeitsstand feststellen | 20 Min. |
| 4. Teil: Theorie | · Entstandene Fragen aus Teil 3 klären<br>· Rekapitulation<br>· Spezielle Anpassungen des Builds (Properties)<br>· Maven Profiles und persönliche Settings<br>· Paketierung (JARs, WARs, EARs)<br>· Multi-Module Projekte<br>· Deployment Plugins<br>· Reporting Plugins | 1 h |
| 5. Teil: Praxis | · Aufgabenteilung:<br>· Multi-Module Projekt aufsetzen<br>· Datenbank (HQSQL) integrieren und abfragen + DBUnit<br>· Persönlicher Build und Deployment mit Profilen | Ende offen;<br>mind. jedoch<br>2h 40 Min. |
| 6. Teil: Abschluß | · Fazit / Probleme bei Aufgabenteilung analysieren<br>· Ausblick auf den nächsten Tag | 10 Min. |



# Ladies and Gentlemen,

# Start your Engines!



*Fast and Furious:*

# Apache Maven 2

Fachhochschule Köln
Cologne University of Applied Sciences

# Rekapitulation

# Nach den ersten Aufgaben…

- Kickstart hat einen *Projekt-Zyklus* abgebildet
  - Projekt aufsetzen / Klassen schreiben, testen / Auslieferung vorbereiten
  - Maven hat den Entwicklungszyklus von Anfang bis Ende unterstützt



# Im Vergleich mit Maven

# Maven im Vergleich

- make (1977)
  - in C geschrieben
  - erfreut sich weiterhin großer Beliebtheit
  - Explizite Beschreibung des Verhältnisses zwischen Dateien
  - Erweiterung durch shell-Programme
  - Strukturierung durch Whitespaces und Tabs (!)

```
project.exe : main.obj io.obj
     tlink c0s main.obj io.obj, project.exe,, cs /Lf:\bc\lib
main.obj : main.c
     bcc –ms –c main.c
io.obj : io.c
     bcc –ms –c io.c
```

aus: http://www.opussoftware.com/tutorial/TutMakefile.htm



# Maven im Vergleich

- Ant (2000)
  - „Another Neat Tool“
  - in Java geschrieben
    - Erweiterung durch Java-Klassen
- Konfiguration in XML
  - Entwickler stellen sich eine Baum-Struktur an Ausführungszielen zusammen
- Write once, run everywhere
- Notfalls auch shell-Befehle möglich

```
<project name="MyProject" default="dist" basedir=".">
    <description>
        simple example build file
    </description>
  <!-- set global properties for this build -->
  <property name="src" location="src"/>
  <property name="build" location="build"/>
  <property name="dist"  location="dist"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}"/>
  </target>

  <target name="compile" depends="init"
        description="compile the source " >
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <target name="dist" depends="compile"
        description="generate the distribution" >
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib"/>

    <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
    <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
  </target>

  <target name="clean"
        description="clean up" >
    <!-- Delete the ${build} and ${dist} directory trees -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>
```

aus: http://ant.apache.org/manual/using.html



# Apache Maven 2



# Apache Maven 2

- Software Project Management und *comprehension* tool*
  - Zentrales Konzept: das Project Object Model (POM)

* (Verständnis, Bedeutungsumfang)



# Anforderungen

- Build ist soll nicht unnötig kompliziert sein
- Entwickler müssen gleiche Versionen von eigenen Komponenten und Fremdbibliotheken haben
  - falls nicht: „Auf meiner Kiste läufts aber!“
- Wissen über den Build darf nicht zentralisiert sein



# Ziele von Maven

- Vereinfachung des Build-Prozesses
- Einen gleichförmigen Build bieten
- Qualitative Informationen über das Projekt geben
- Guidelines für Best Practices
- Einfaches updaten von Plugins und Maven selbst

* (Verständnis, Bedeutungsumfang)



# Features

- Standard Lifecyles
- Dependency Management
- Multi-Modul Projekte
- Erweiterbarkeit durch Plugins



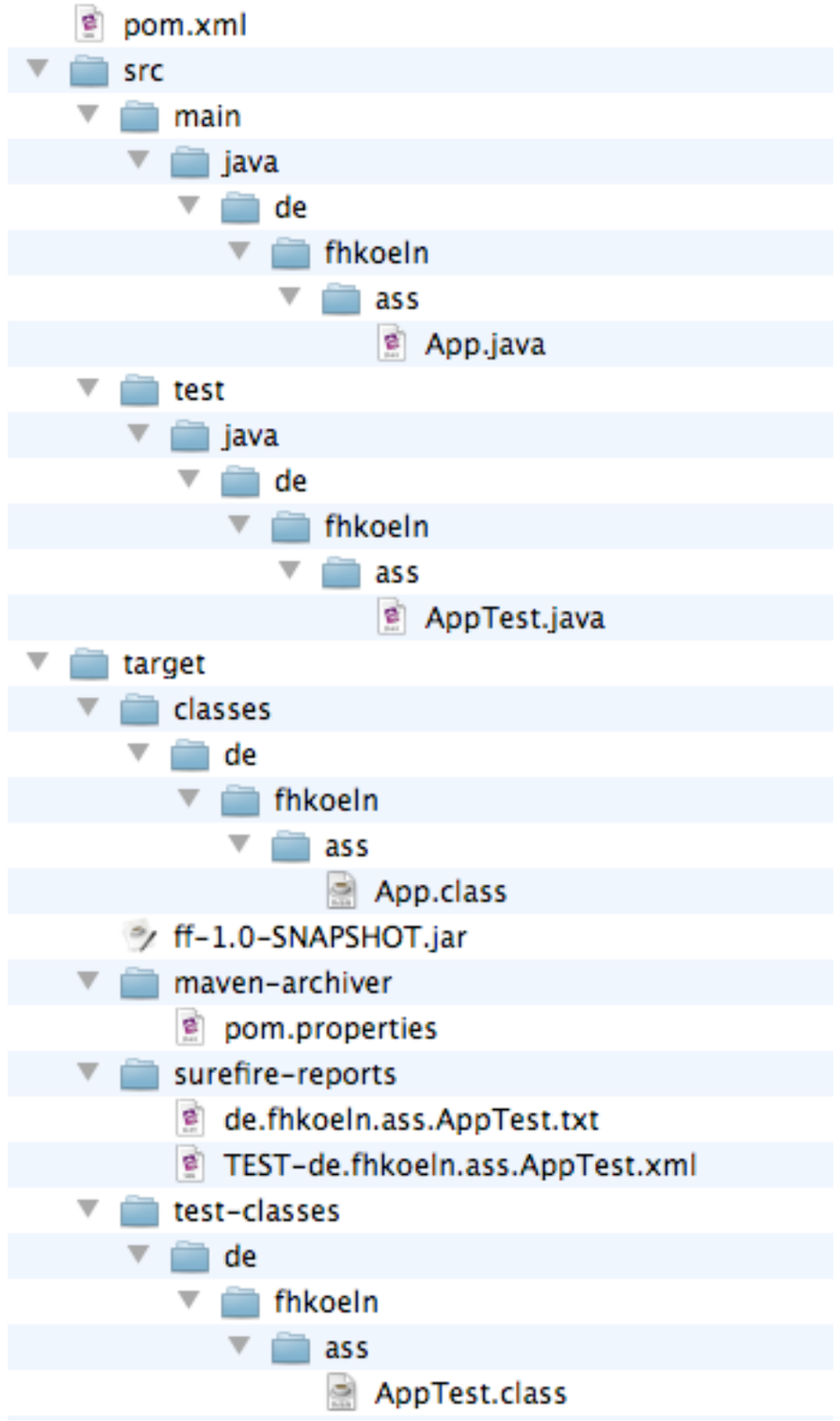
# Prinzipien von Maven

- Convention over Configuration
- Deklarative Ausführung
- Wiederverwendung von Build-Logik
- Kohärente Organisation von Abhängigkeiten



- POM/Projektstruktur, Lifecycle und Dependency Management im Detail

# Projektstruktur im Finder



# POM

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>de.fhkoeln.ass</groupId>
  <artifactId>ff</artifactId>
  <packaging>jar</packaging>
  <version>1.0-SNAPSHOT</version>

  <name>ff</name>
  <url>http://maven.apache.org</url>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

</project>
```



# POM - <dependencies>

- Download bei Maven-Aufruf
- Download aus öffentlichen Repositories
- Ablage in lokalem Repo
- ~/.m2/repository/groupid/artifactid/version/artefaktname-versionsnummer.jar

```
<project ...>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
    </dependency>
  </dependencies>
</project>
```



# POM - <build>

- Angabe von Ressourcen und weiteren Infos
- Einbindung und Konfiguration von Plugins

```
<project ...>
    ...
      <build>
            <finalName>Name der Anwendung</finalName>
            <resources>
               <resource>
                     <directory>src/main/resources</directory>
               </resource>
            </resources>
            <testResources>
               <testResource>
                     <directory>src/test/resources</directory>
               </testResource>
         </testResources>

         <plugins>
               <plugin>
                     <artifactId>maven-compiler-plugin</artifactId>
                     <version>2.0.2</version>
                     <configuration>
                           <source>1.5</source>
                           <target>1.5</target>
                     </configuration>
               </plugin>
         </plugins>
      </build>
    ...
</project>
```



# Transitive Abhängigkeiten

- Verwendete Libraries verwenden ihrerseits weitere Libraries.
- Wenn zum Beispiel Hibernate log4j in der Version 1.2.x verwendet, Struts aber die Version 1.1.x, können zur Laufzeit sonderbare Effekte auftreten (NoSuchMethodException/IncompatibleClassChangeError).
- mvn dependency:tree zeigt transitive Abhängigkeiten, verlässt sich dabei auf die pom.xml-Dateien aus den Repositories
- Ohne maven muss das manuell überwacht werden, was in der Praxis meist „vergessen wird“



# Beispiel dependency:tree

```
[INFO] ------------------------------------------------------------------------
[INFO] Building Fast and Furious - Multi Module Example Project ** Data Layer **
[INFO]    task-segment: [dependency:tree]
[INFO] ------------------------------------------------------------------------
[INFO] [dependency:tree {execution: default-cli}]
[INFO] de.fhkoeln.ass:data-layer:jar:1.0-SNAPSHOT
[INFO] +- javax.persistence:persistence-api:jar:1.0:compile
[INFO] +- org.springframework:spring:jar:2.5.6:compile
[INFO] |  \- commons-logging:commons-logging:jar:1.1.1:compile
[INFO] +- org.apache.cxf:cxf-rt-frontend-jaxws:jar:2.2.3:compile
[INFO] |  +- xml-resolver:xml-resolver:jar:1.2:compile
[INFO] |  +- org.apache.geronimo.specs:geronimo-jaxws_2.1_spec:jar:1.0:compile
[INFO] |  |  \- org.apache.geronimo.specs:geronimo-activation_1.1_spec:jar:1.0.2:compile
[INFO] |  +- org.apache.geronimo.specs:geronimo-ws-metadata_2.0_spec:jar:1.1.2:compile
[INFO] |  +- asm:asm:jar:2.2.3:compile
[INFO] |  +- org.apache.cxf:cxf-api:jar:2.2.3:compile
[INFO] |  |  +- org.apache.cxf:cxf-common-utilities:jar:2.2.3:compile
[INFO] |  |  |  +- org.apache.geronimo.specs:geronimo-stax-api_1.0_spec:jar:1.0.1:compile
[INFO] |  |  |  +- wsdl4j:wsdl4j:jar:1.6.2:compile
[INFO] |  |  |  \- commons-lang:commons-lang:jar:2.4:compile
[INFO] |  |  +- org.apache.ws.commons.schema:XmlSchema:jar:1.4.5:compile
[INFO] |  |  +- org.apache.geronimo.specs:geronimo-annotation_1.0_spec:jar:1.1.1:compile
[INFO] |  |  +- org.codehaus.woodstox:wstx-asl:jar:3.2.8:compile
[INFO] |  |  +- org.apache.neethi:neethi:jar:2.0.4:compile
[INFO] |  |  \- org.apache.cxf:cxf-common-schemas:jar:2.2.3:compile
[INFO] |  +- org.apache.cxf:cxf-rt-core:jar:2.2.3:compile
[INFO] |  |  +- com.sun.xml.bind:jaxb-impl:jar:2.1.12:compile
....
```



# Der Lifecycle

- default Lifecycle
  - Bau und Auslieferung des Projekts
- clean Lifecycle
  - Aufräumen von Generaten, Kompilaten usw.
- site Lifecycle
  - Erstellen der Projektdokumentation
- Für alle gilt: das Aufrufen einer Phase beinhaltet den Aufruf aller vorhergehenden Phasen!



# default Lifecycle

| | |
|---|---|
| `validate` | Validierung von Informationen, Vollständigkeit |
| `generate-sources` | Generiert Sourcen |
| `compile` | Kompiliert sämtliche Sourcen |
| `test` | Führt die Testklassen aus |
| `package` | Legt die Binaries im Auslieferungsformat ab |
| `integration-test` | Liefert das Paket in eine Auslieferungsumgebung zum Test aus |
| `verify` | Prüft auf Qualitätskriterien |
| `install` | Installiert das Paket im lokalen Repository, ermöglicht die Einbindung als Dependency |
| `deploy` | Kopiert das Paket in ein entferntes Repository, ermöglicht die Einbindung als Dependency für andere |



# site Lifecycle

| | |
|---|---|
| `pre-site` | Ausführung von Prozessen vor der Generierung der Dokumentation |
| `site` | Generiert Dokumentation und Reports |
| `post-site` | Postprocessing der Dokumentation und Vorbereitung der Auslieferung |
| `site-deploy` | Deployt die Dokumentation auf Webserver |



# clean Lifecycle

| | |
|---|---|
| `pre-clean` | Ausführung von Prozessen vor dem Löschen von target/ |
| `clean` | Löschen von target/ |
| `post-clean` | Ausführung von Prozessen nach dem Löschen von target/ |



# Aufruf einer Phase

```
$ mvn compile
```



# Ladies and Gentlemen,

# Start your Engines!



*Fast and Furious:*

# Apache Maven 2

## Der Theorie zweiter Teil

Fachhochschule Köln
Cologne University of Applied Sciences

- Fragen zum letzten praktischen Teil

# Euere Meinung?
# Vorteile & Nachteile?

- Builds weiter anpassen

# Properties strukturieren das POM weiter

- Properties sind key/value Paare

```
<project ...>
   <properties>
       <!-- Application settings -->
       <copyright.year>2008<copyright.year>
       <!-- Framework dependency versions -->
       <appfuse.version>2.0.2</appfuse.version>
       <spring.version>2.5.4</spring.version>
   </properties>
</project>
```

- Können im POM beliebig verwendet werden

```
<project ...>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring</artifactId>
      <version>${spring.version}</version>
    </dependency>
  </dependencies>
</project>
```



# Properties

- Properties können aber auch
  - geerbt werden (aus Super-POM oder übergeordnetem Projekt)
  - extern gesetzt werden
    - per Kommandozeilen-Parameter
    - per profiles.xml / settings.xml



- Profiles und Settings

# Profiles

- Profiles ermöglichen umgebungsabhängige Builds
  - Umgebungsabhängigkeit und Plattformunabhängigkeit bilden Zielkonflikt
- Profiles.xml ist ein Subset der pom.xml
- Profiles modifizieren den Build zur Laufzeit



# Profiles

- Profiles sollen ähnliche-aber-unterschiedliche Parameter ermöglichen
  - bspw. die Server URL und Socket für Umgebungen (Entwicklungsumgebung, Testumgebung, Produktivumgebung)
- Profiles verursachen unterschiedliche Builds



# Beispiel - Profiles.xml

```
<profilesXml>
    <profiles>
        <profile>
            <id>local</id>
            <activation>
                <property>
                    <name>env</name>
                    <value>local</value>
                </property>
            </activation>
            <properties>
                <weblogic.version>9.0</weblogic.version>
                <host>localhost</host>
                <port>7001</port>
                <protocol>t3</protocol>
                <user>weblogic</user>
                <password>geheim</password>
                <target>myLocalServer</target>
            </properties>
        </profile>
    </profiles>
</profilesXml>
```

# Definition von Profiles

- im Maven Settings File (meist `<your -home-directory>/.m2/settings.xml`)
- in profiles.xml (liegt im gleichen Verzeichnis wie pom.xml)
- im POM selbst

# Definition von Profiles

| | Gültigkeit im System | Gültigkeit im Team | Pfad |
|---|---|---|---|
| Maven Settings File | für alle Maven-Projekte auf einem System | 1 pro Person | ~/.m2/settings.xml |
| Profile | nur in Projekt und allen Unterprojekten | 1 pro Person | profiles.xml neben pom.xml |
| POM | nur in Projekt und allen Unterprojekten | 1 für Alle | pom.xml |

# Definition von Profiles

| | Gültigkeit im System | Gültigkeit im Team | Pfad |
|---|---|---|---|
| Maven Settings File | für alle Maven-Projekte auf einem System | 1 pro Person | ~/.m2/settings.xml |
| Profile | nur in Projekt und allen Unterprojekten | 1 pro Person | profiles.xml neben pom.xml |
| POM | nur in Projekt und allen Unterprojekten | 1 für Alle | pom.xml |

# Definition von Profiles

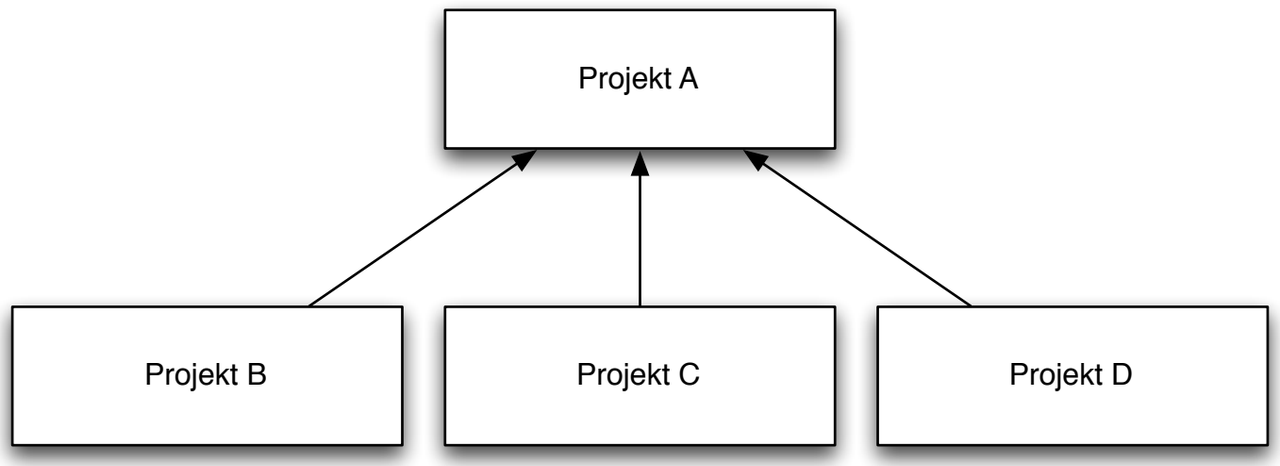
| | Gültigkeit im System | Gültigkeit im Team | Pfad |
|---|---|---|---|
| Maven Settings File | für alle Maven-Projekte auf einem System | 1 pro Person | ~/.m2/settings.xml |
| Profile | nur in Projekt und allen Unterprojekten | 1 pro Person | profiles.xml neben pom.xml |
| POM | nur in Projekt und allen Unterprojekten | 1 für Alle | pom.xml |

# Multi-Modul Projekte



# Eltern-Kind Beziehung für Projekte

- Projekte B,C,D erben Dependencies von A
- B,C,D haben eigene Plugins und weitere Dependencies



# Multi-Module Projekte

## Parent

```
<project ...>
    <modelVersion>4.0.0</modelVersion>
    <groupId>de.fhkoeln.ass</groupId>
    <artifactId>ff-parent</artifactId>
    <packaging>pom</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>ff-parent</name>


    <modules>
        <module>A</module>
        <module>B</module>
        <module>C</module>
    </modules>
...
```

## Modul A

```
<project ...>
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>de.fhkoeln.ass</groupId>
        <artifactId>ff-parent</artifactId>
        <version>1.0-SNAPSHOT</version>
    </parent>
    <artifactId>A</artifactId>
    <packaging>jar</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>ff-A</name>
...
```

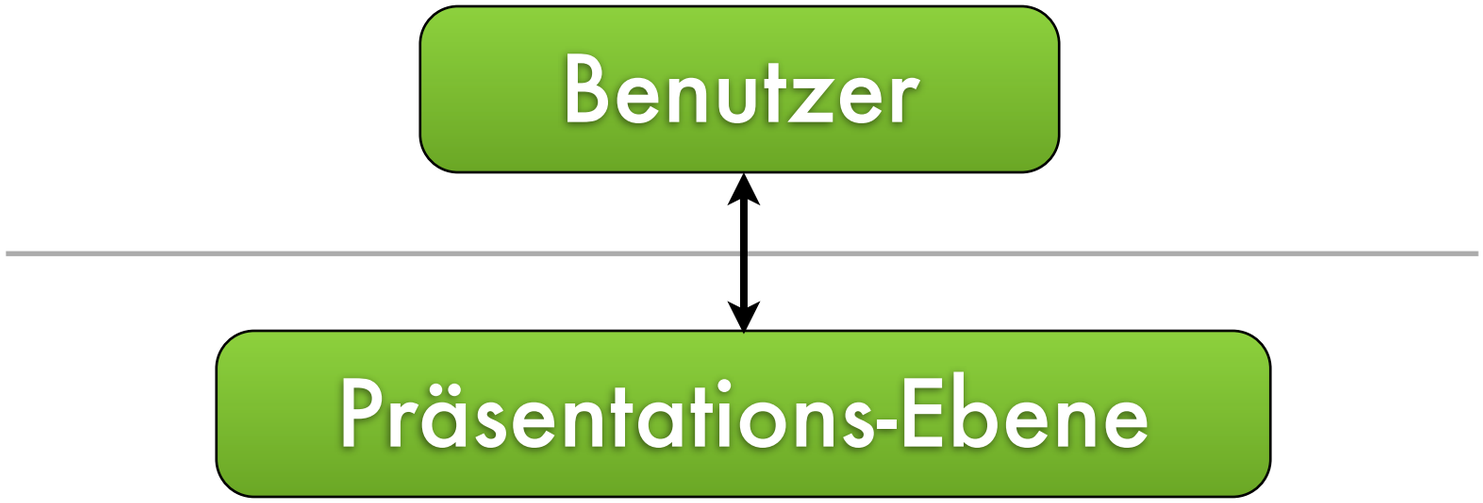


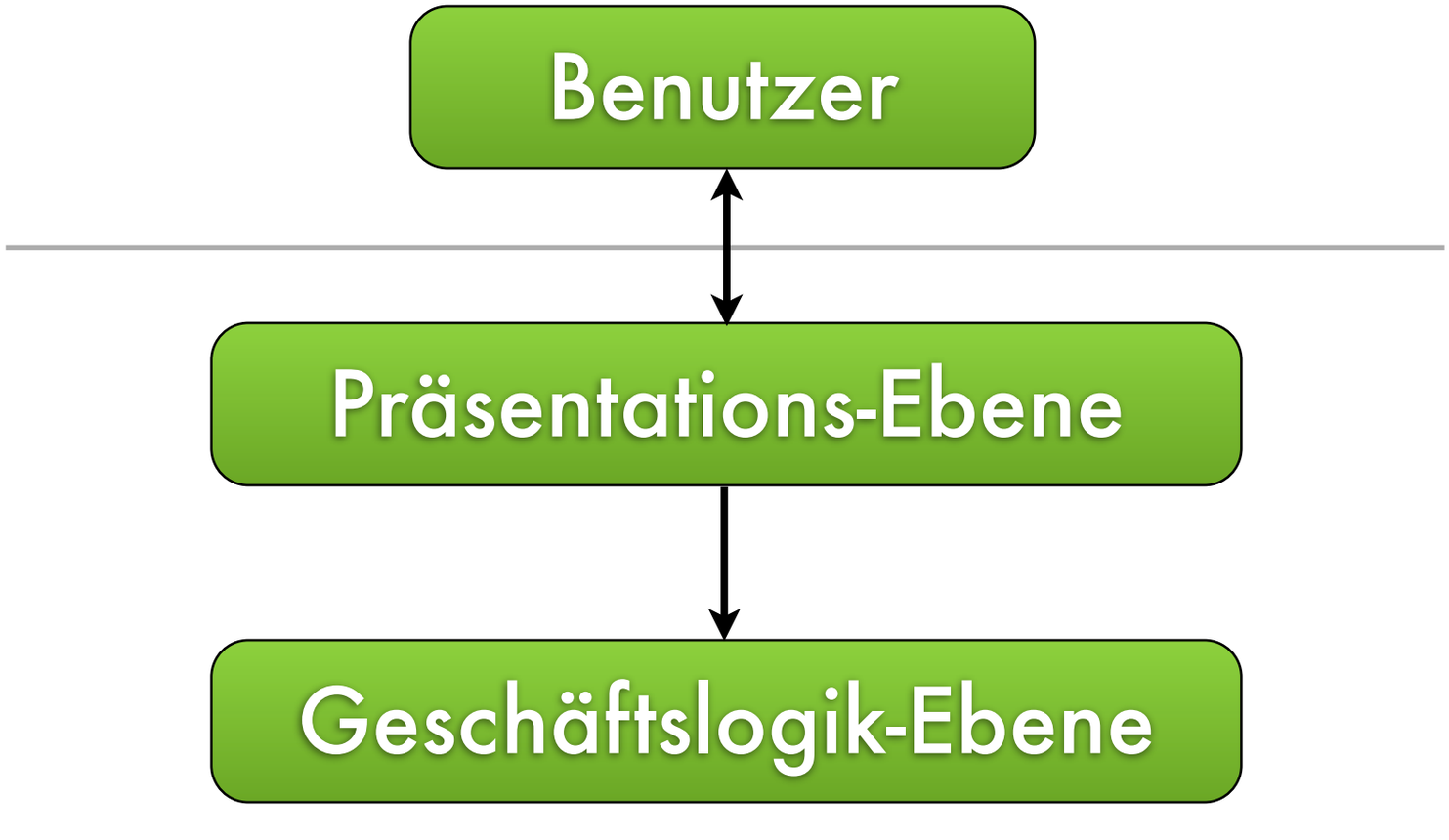
# Modularisierung

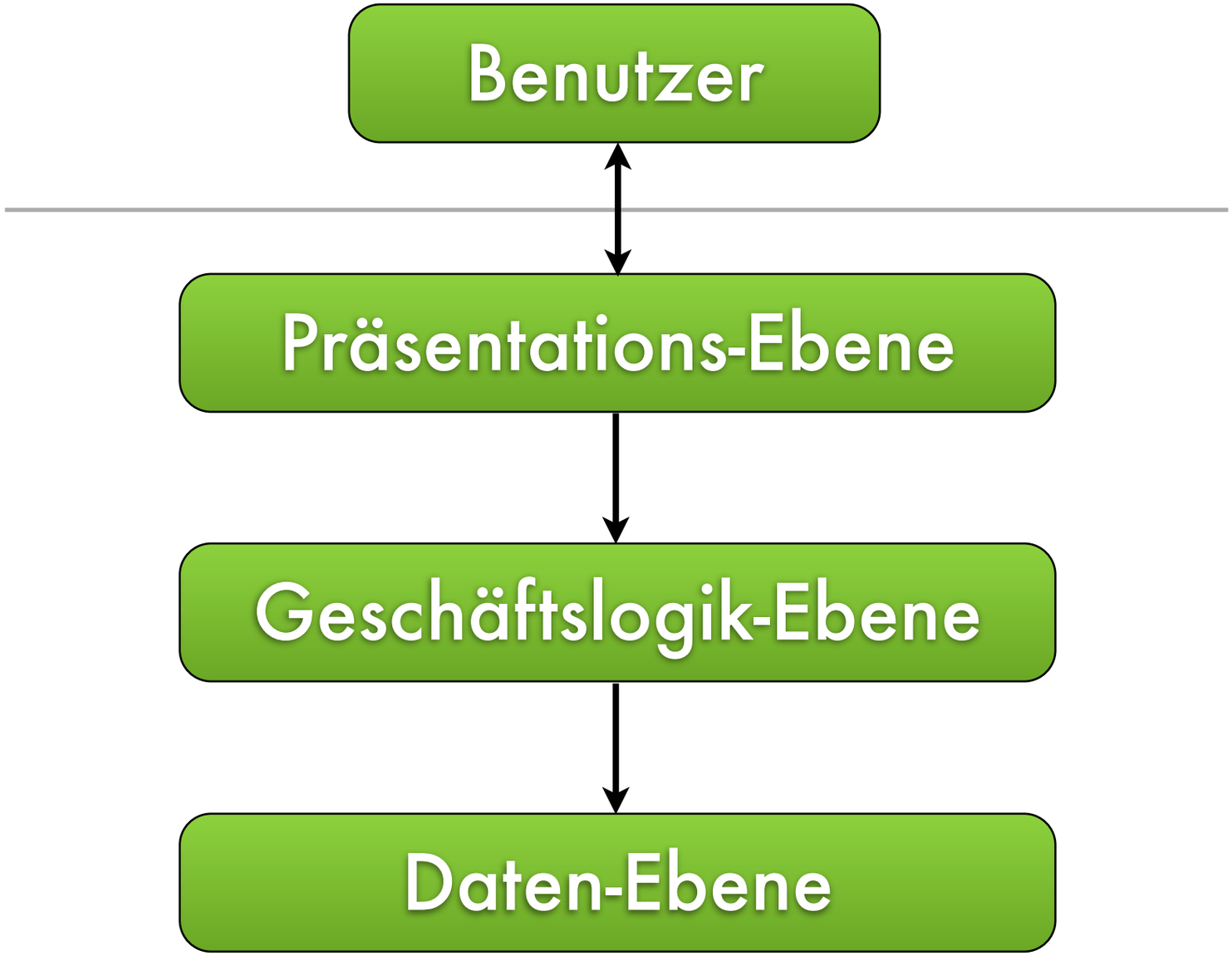
- „Teile und Herrsche“ – Angewandt!
- Vertikale oder Horizontale Modularisierung
- Typischer Fall: n-Tier Architektur

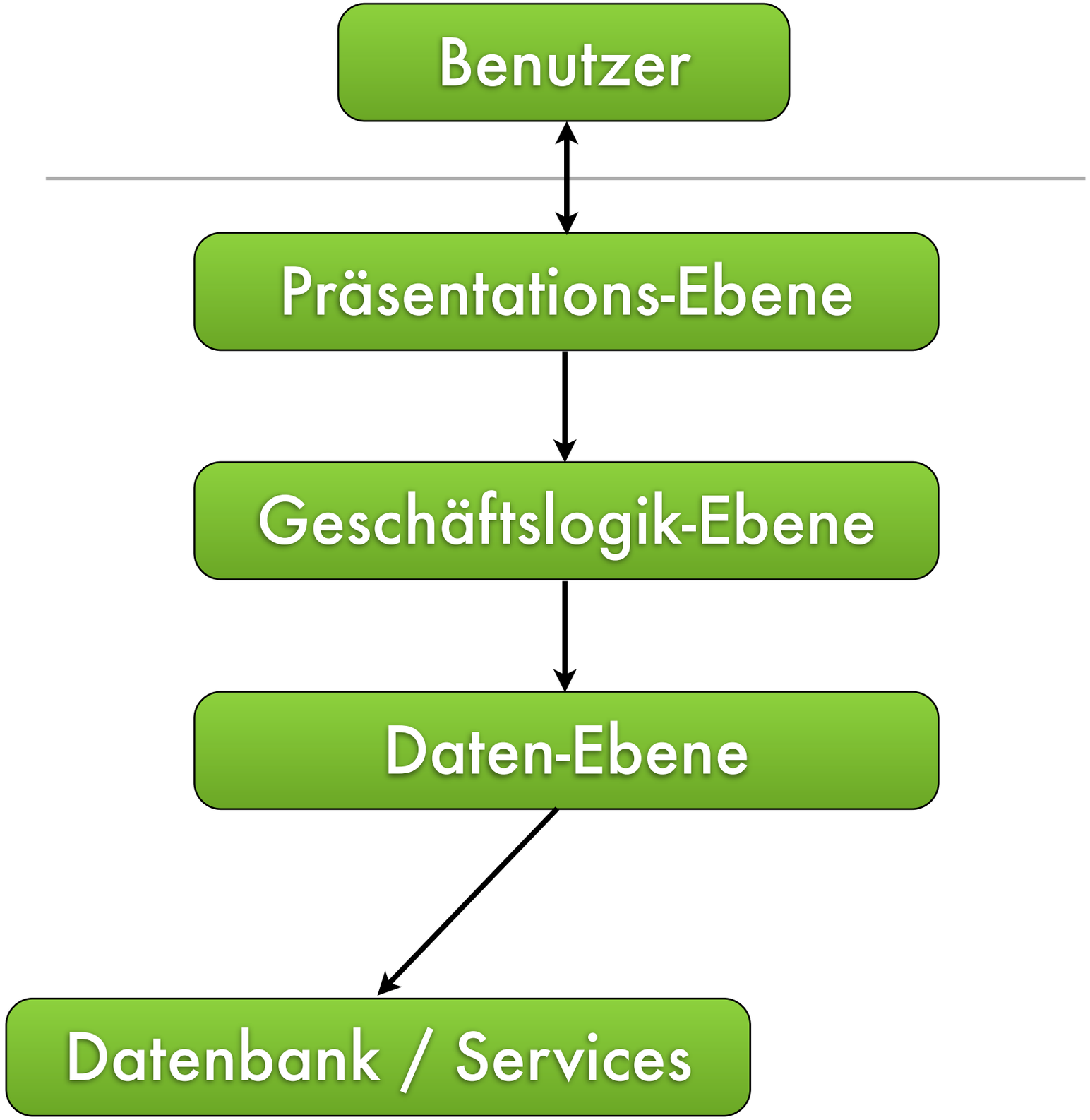


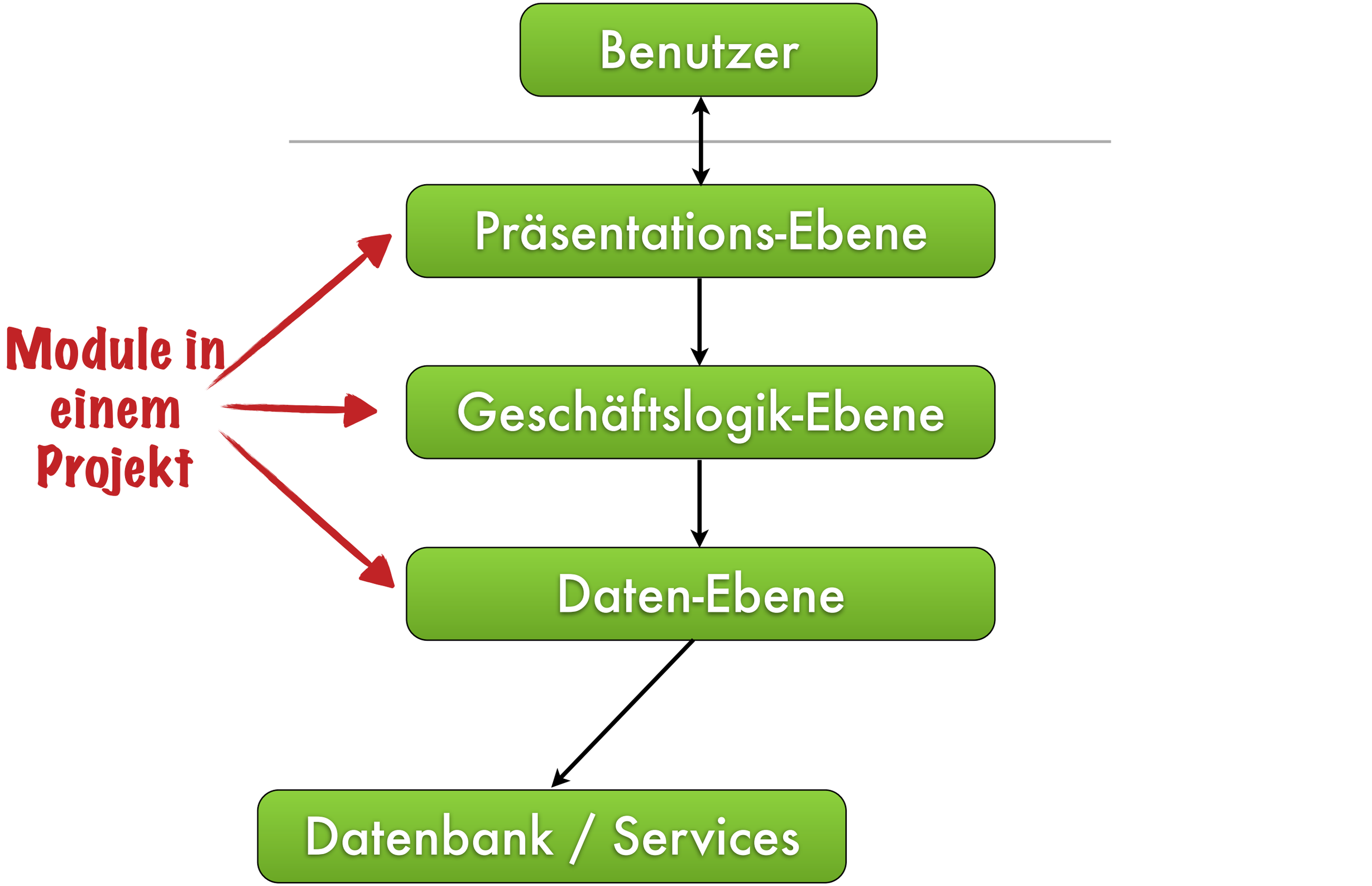
# Benutzer


Benutzer
Präsentations-Ebene


Benutzer
Präsentations-Ebene
Geschäftslogik-Ebene


Benutzer
Präsentations-Ebene
Geschäftslogik-Ebene
Daten-Ebene


Benutzer
Präsentations-Ebene
Geschäftslogik-Ebene
Daten-Ebene
Datenbank / Services


Benutzer
Präsentations-Ebene
Module in einem Projekt
Geschäftslogik-Ebene
Daten-Ebene
Datenbank / Services

# Interne Projektabhängigkeiten

- Präsentationsschicht nutzt Geschäftslogik nutzt Daten-Ebene
- Abhängigkeiten werden in Modul-POM definiert
- Problem: wie können Entwickler der Präsentationsschicht auf Pakete der Geschäftslogik zugreifen?



# Dependency Management

# Repositories

| | |
|---|---|
| lokal | ~/.m2/repository |
| LAN | http://<br>server.mycompany.de/... |
| WAN | Ibiblio, etc. |



# Repositories

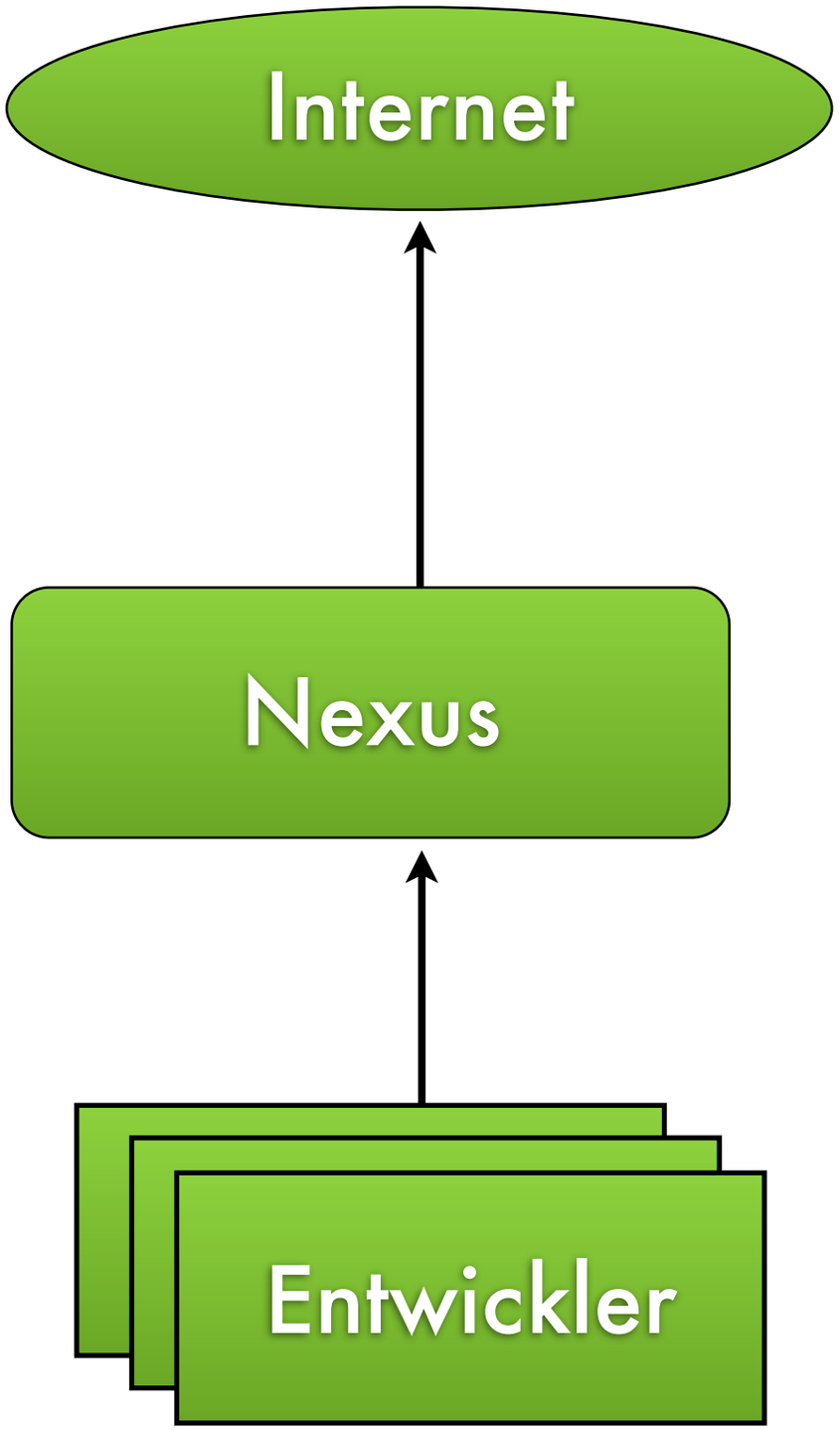
| | |
|---|---|
| lokal | ~/.m2/repository |
| LAN | http://<br>server.mycompany.de/... |
| WAN | Ibiblio, etc. |



# Nexus Repository Manager

- Proxy zwischen lokalem und entferntem Repository
  - verringert Traffic und Ladezeit beim Download von Dependencies
- Managing von Release und Snapshot Dependencies
  - Nexus prüft Snapshots regelmäßig
- Sharing von Binaries innerhalb eines LANs




Internet
Nexus
Entwickler

# Ladies and Gentlemen,

# Start your Engines!



*Fast and Furious:*

# The Spring Framework

Fachhochschule Köln
Cologne University of Applied Sciences

# Rekapitulation

# Nach den ersten Aufgaben…

- Was ist Kopplung? Was ist das Problem mit Interfaces? Was ist das Problem mit „new“?
- Kickstart soll Problematik im Aufbau von Objektnetzen zeigen



# Im Vergleich mit Maven

# Vergleich mit Maven

- Maven arbeitet „an“ der Software
  - generiert Sourcen, kompiliert
  - sorgt für Paketierung, Auslieferung
  - ist aber im Betrieb nicht mehr präsent
- Spring wirkt „in“ der Software
  - konfiguriert zur Laufzeit
- Maven und Spring haben keine Schnittmenge!



# The Spring Framework



# Spring

- Spring ist eine Applikations-Plattform für Java
- Spring ist ein leichtgewichtiger Container
  - Programmcode ist nicht an Spring gebunden, bzw muss es nicht sein
- Spring erlaubt die Konfiguration von Applikations-Komponenten
- Spring integriert Frameworks von Drittanbietern



## DAO
Spring JDBC
Transaction Management

## ORM
Hibernate
JPA
TopLink
JDO
OJB
iBatis

## JEE
JMX
JMS
JCA
Remoting
EJBs
Email

## Web
Spring Web MVC
Spring Portlet MVC
Struts
WebWork
Tapestry
JSF
Rich Views Support
Velocity
Freemarker
PDF
Jasper Reports

## AOP
Spring AOP
AspectJ Integration

## Core
Spring IoC Container



## DAO

Spring JDBC
Transaction Management

## ORM

Hibernate
JPA
TopLink
JDO
OJB
iBatis

## JEE

JMX
JMS
JCA
Remoting
EJBs
Email

## Web

Spring Web MVC
Spring Portlet MVC
Struts
WebWork
Tapestry
JSF
Rich Views Support
Velocity
Freemarker
PDF
Jasper Reports

## AOP

Spring AOP
AspectJ Integration

## Core

Spring IoC Container

- Grundlegene Funktionen von Spring
- Dependency Injection
- `BeanFactory` / `ApplicationContext` **als zentrale Konzepte**

- Entkopplung von: [Konfiguration und Spezifikation von Abhängigkeiten] / Programm Logik

**Core**
Spring IoC Container

DAO
Spring JDBC
Transaction
Management

- eigene JDBC-Abstraktion
  - kein mühsames JDBC-Coding mehr
  - kein DB-spezifisches SQL mehr
- Programmatisches und deklaratives Transaktions-Management



DAO
Spring JDBC
Transaction
Management

ORM
Hibernate
JPA
TopLink
JDO
OJB
iBatis

- Integration für Objekt-Relationale Mapper
- Nutzung in Kombination mit Springs Transaktions-Management

Web
Spring Web MVC
Portlet MVC
Struts
WebWork
JSF
Rich Views Support
Velocity
Freemarker
PDF
Jasper Reports

AOP
Spring AOP
AspectJ Integration

Core
Spring IoC Container

# Konfiguration

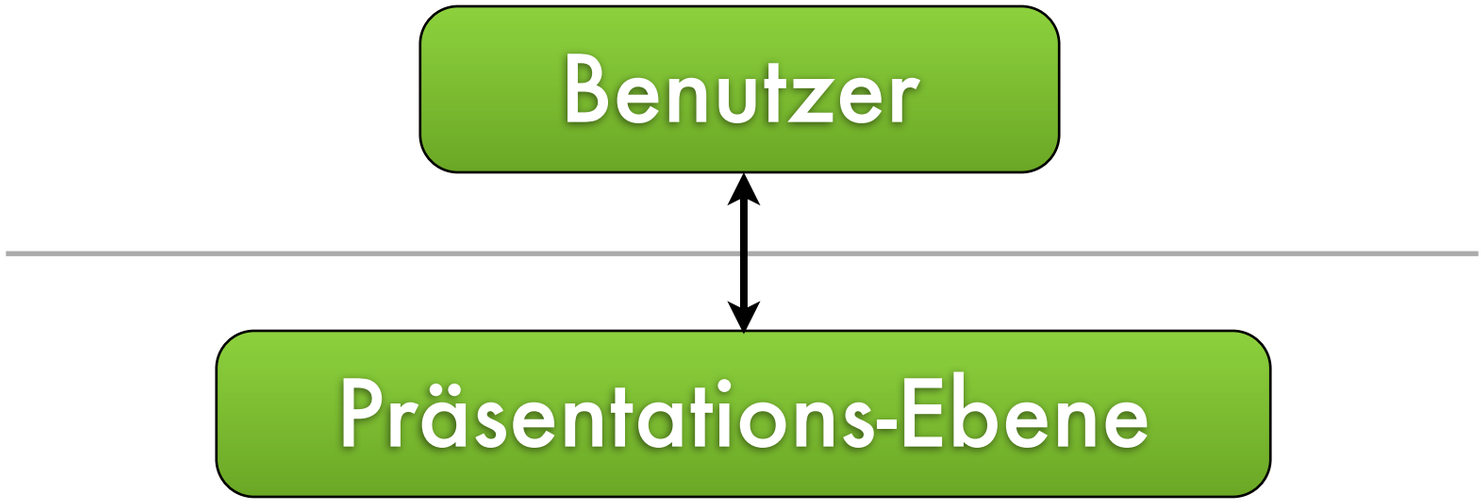
- applicationContext.xml / spring-config.xml / …
  - Keine Namenskonvention, überlicherweise hierarchische Aufteilung in mehreren Dateien
- Meist zentraler Ort für Konfiguration der Software
- Verrät Infos über Architektur im Großen und Kleinen

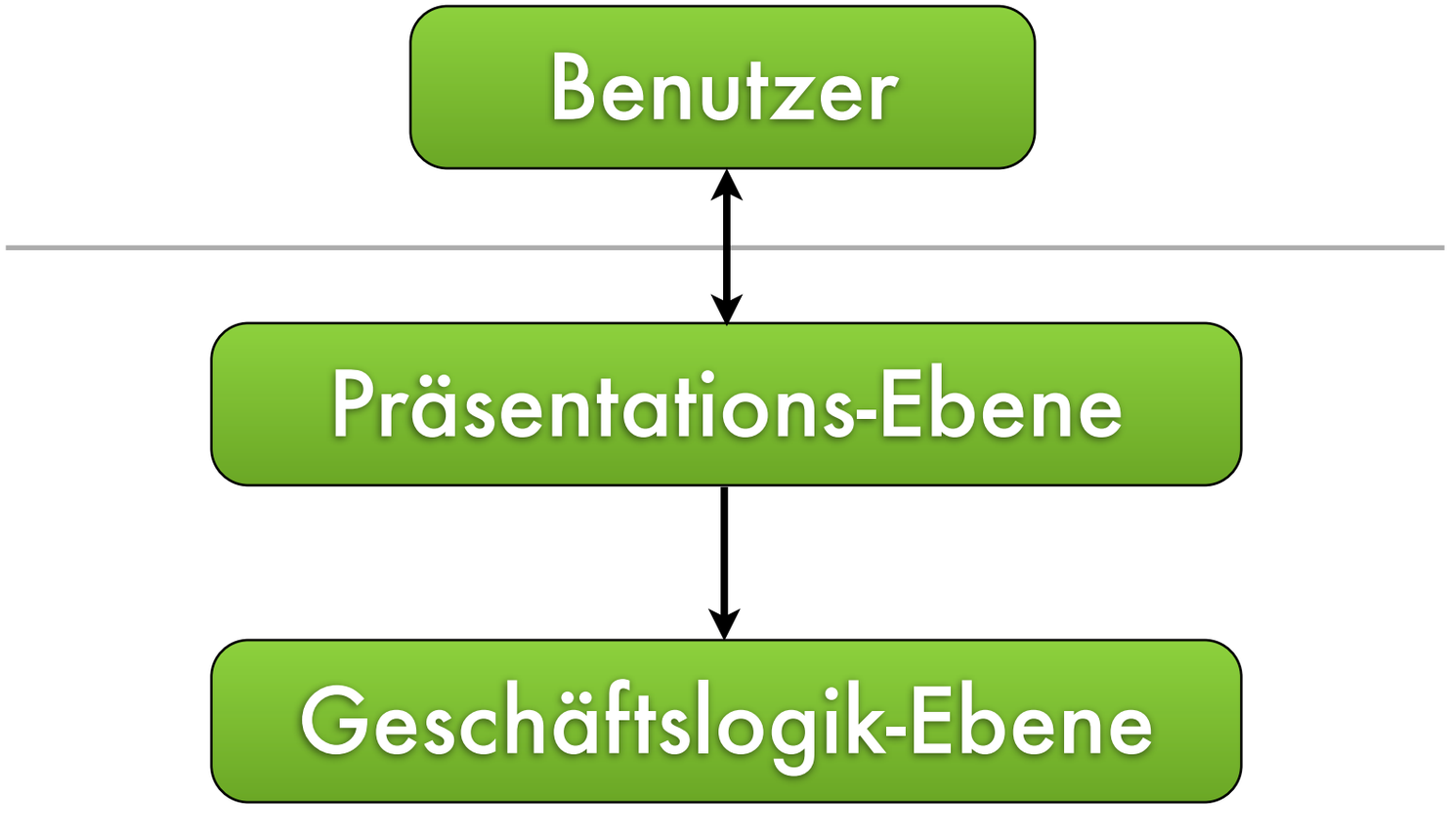


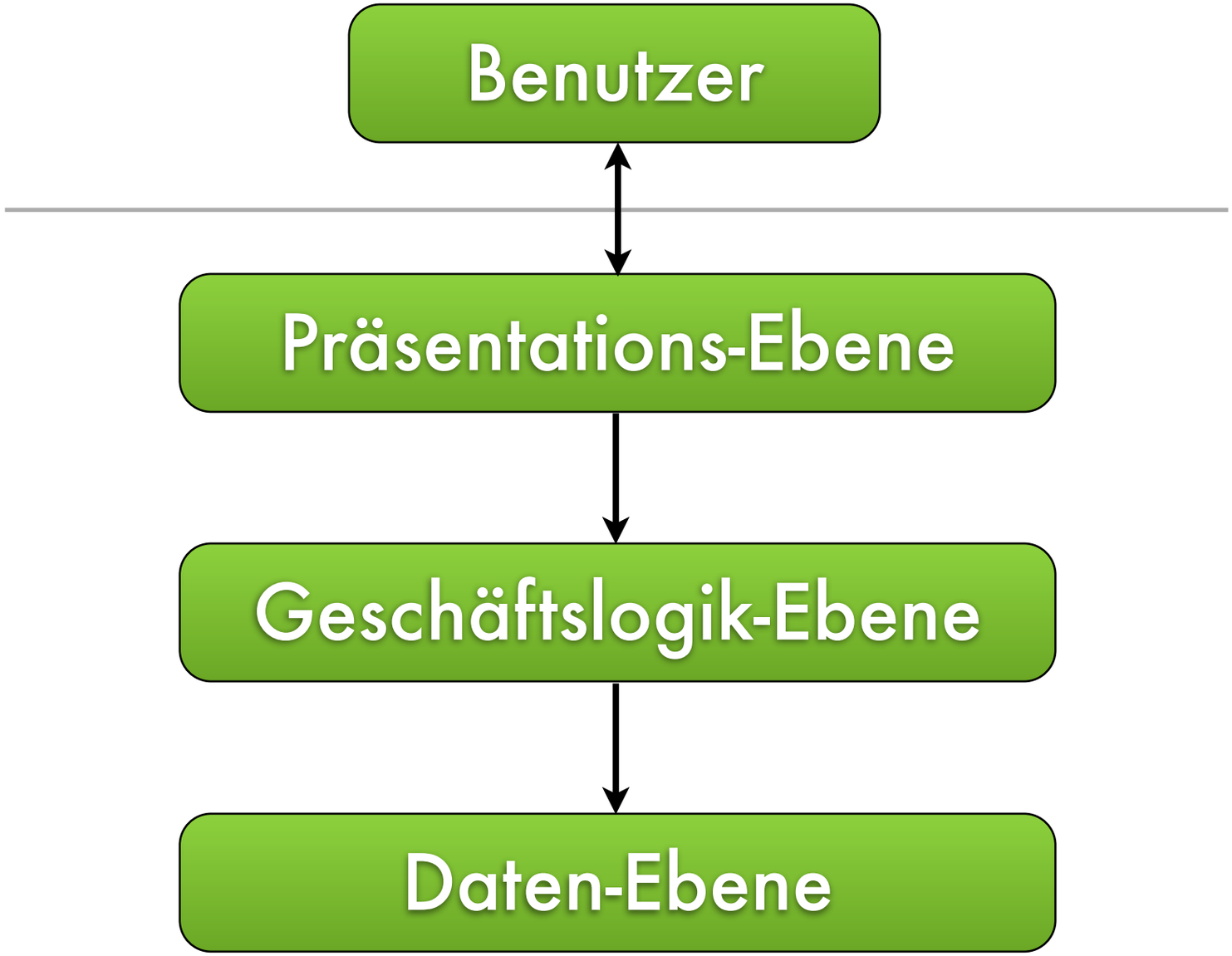
# Dependency Injection mit Spring

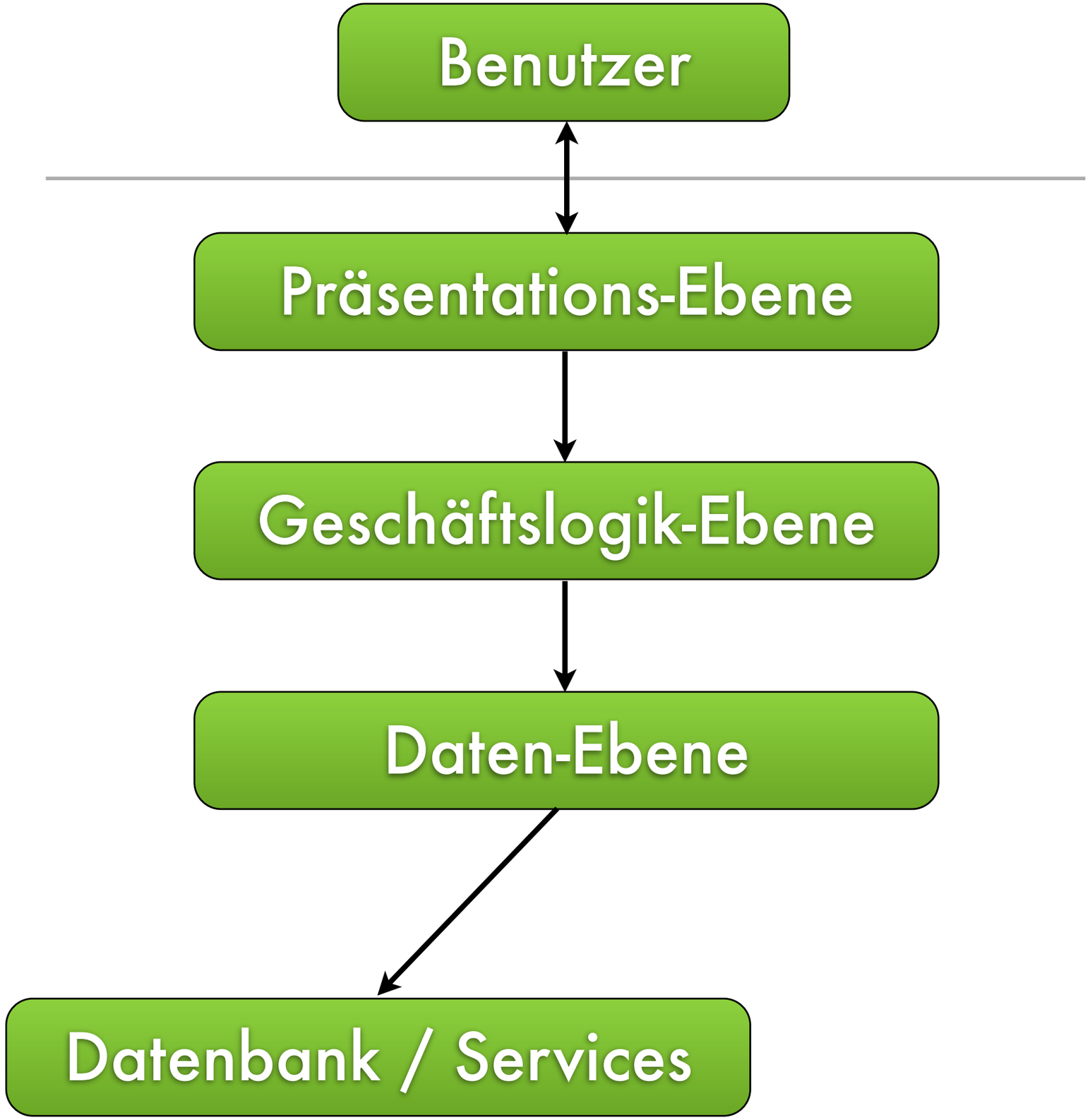


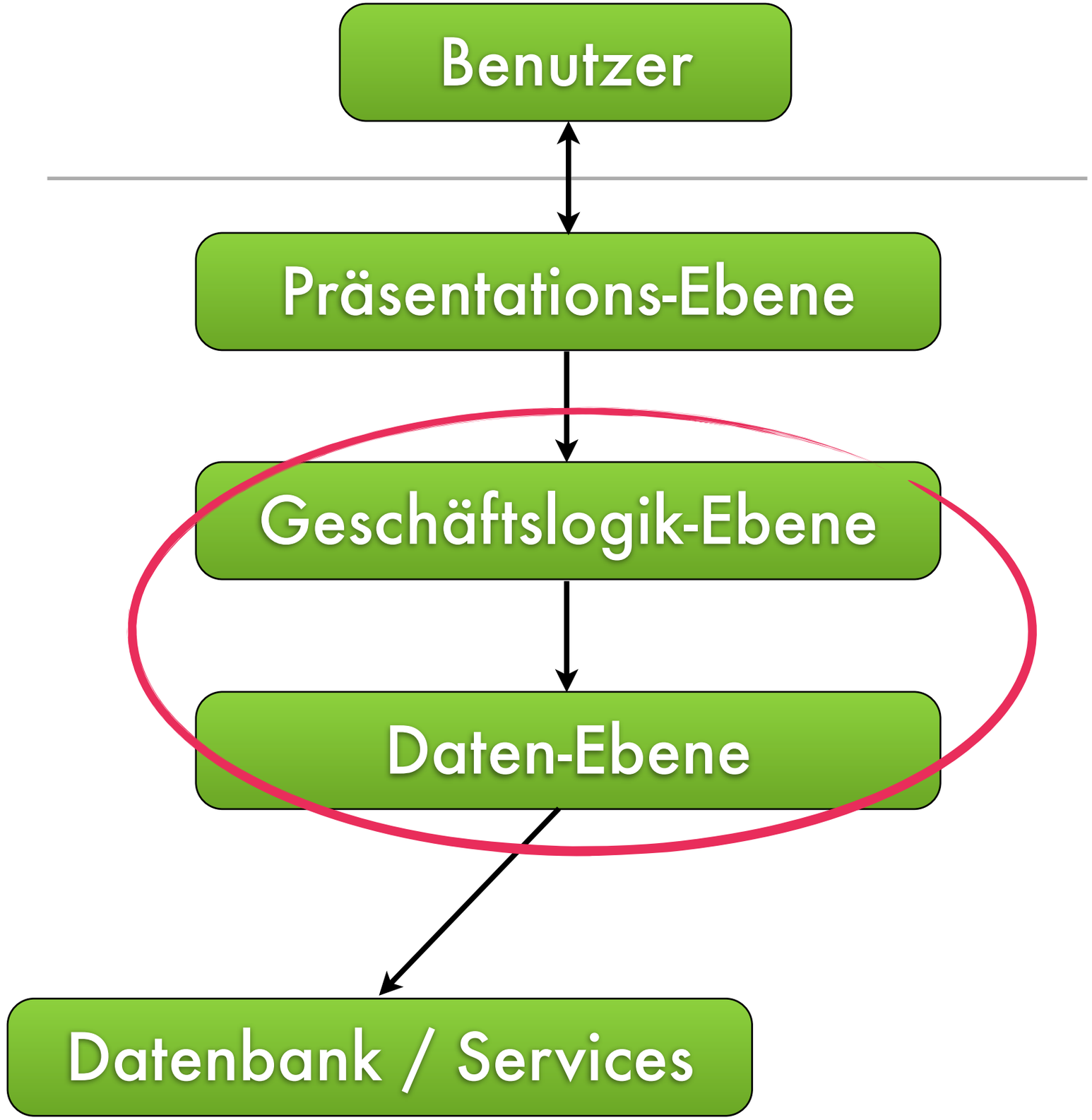
# Benutzer


Benutzer
Präsentations-Ebene


Benutzer
Präsentations-Ebene
Geschäftslogik-Ebene

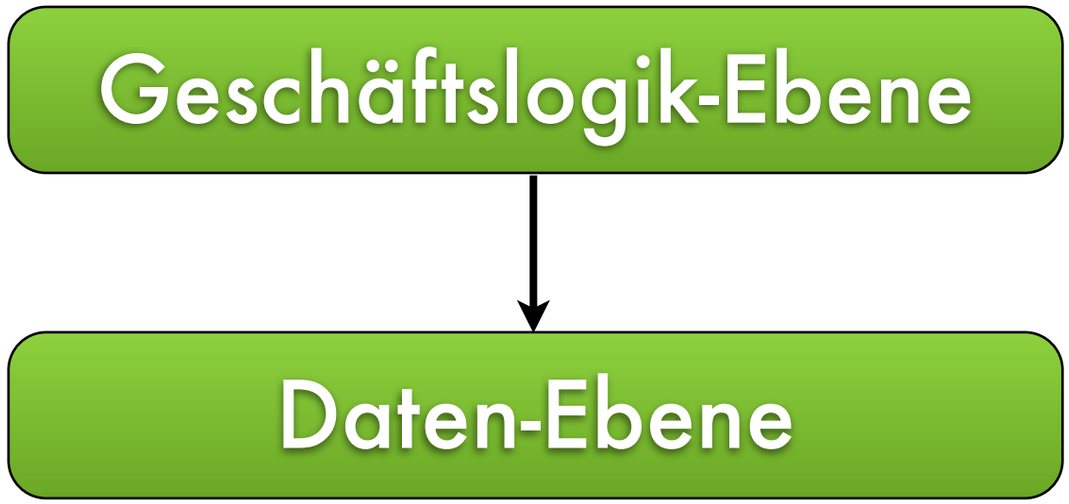

Benutzer
Präsentations-Ebene
Geschäftslogik-Ebene
Daten-Ebene


Benutzer
Präsentations-Ebene
Geschäftslogik-Ebene
Daten-Ebene
Datenbank / Services


Benutzer
Präsentations-Ebene
Geschäftslogik-Ebene
Daten-Ebene
Datenbank / Services

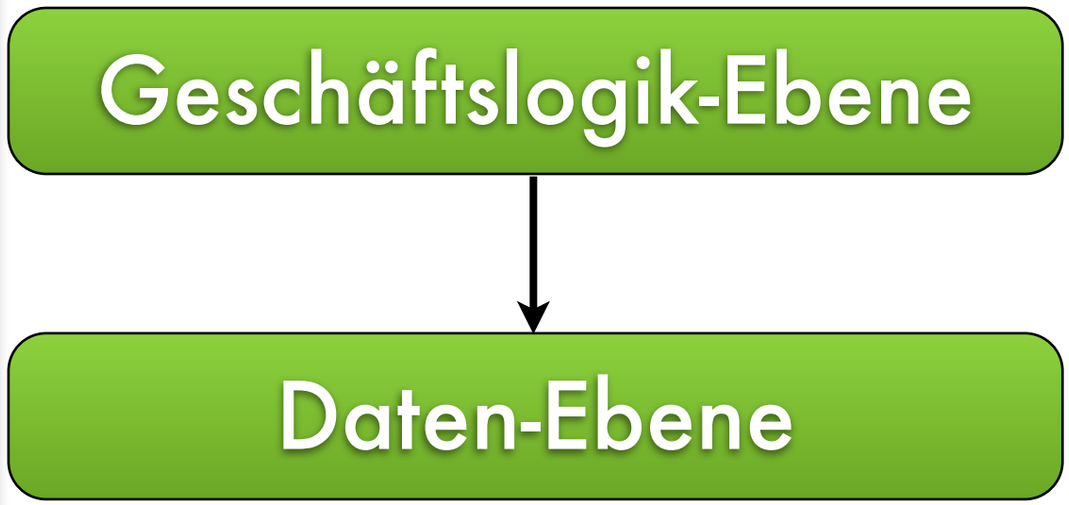
# Beispiel

- Geschäftslogik greift auf Daten-Ebene zu
  - aber nicht andersherum!



# Beispiel

```
public class BusinessLogic {
   private SpecialDataLayer dataLayer;

   public void doBusinessLogic() {
      if(that) {
         dataLayer.doThis();
      }
   }
}


public class SpecialDataLayer
   implements DataLayer {

   public void doThis() {
      this.accessDB();
   }
}
```



# Beziehung ohne Kopplung

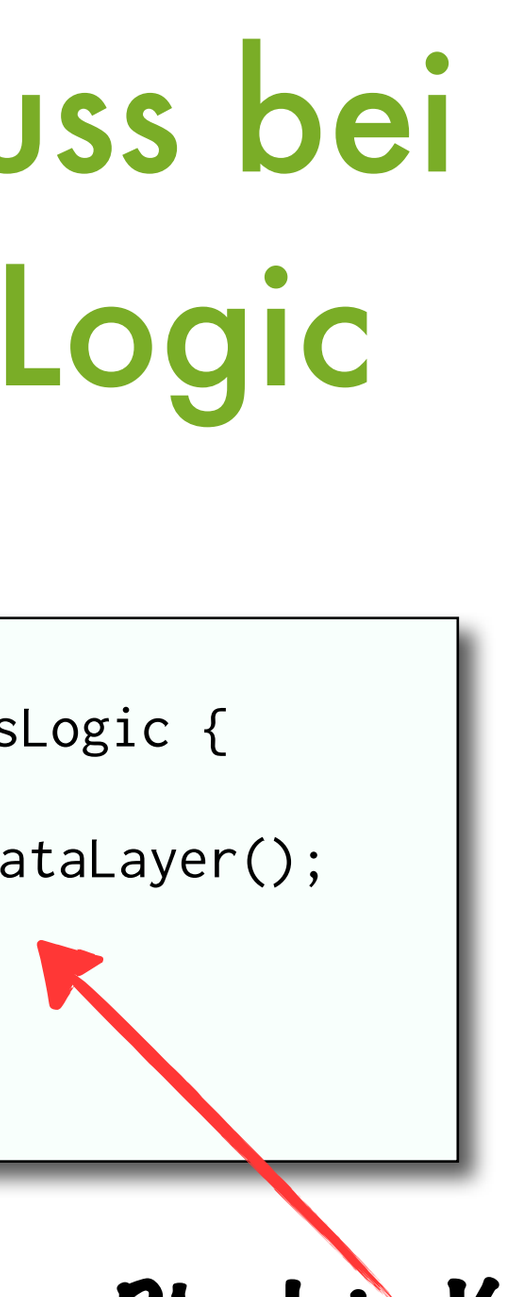
- BusinessLogic-Klasse ist an DataLayer-Interface gebunden
- Problem: Interfaces sind nicht instanziierbar
- Lösung: Dependency Injection

```
public class BusinessLogic {
  private DataLayer dataLayer;

  public void doBusinessLogic() {
    if(that) {
      dataLayer.doThis();
    }
  }
}


public class SpecialDataLayer
  implements DataLayer {

  public void doThis() {
    this.accessDB();
  }
}
```



# Kontrollfluss bei BusinessLogic

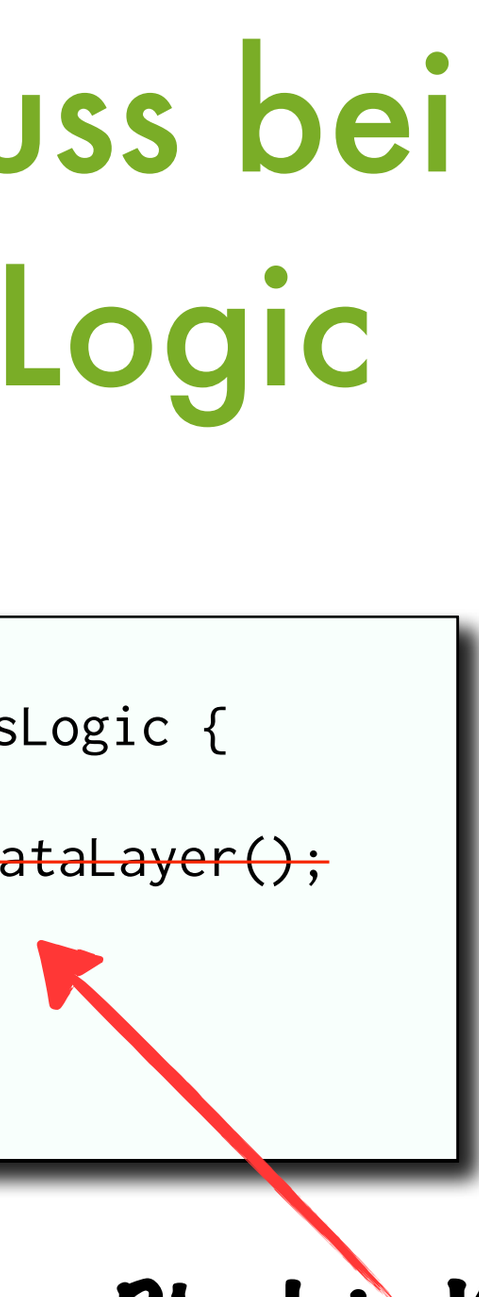
```
public class BusinessLogic {

   ... = new SpecialDataLayer();


}
```



Direkte Kopplung!

# Kontrollfluss bei BusinessLogic

```
public class BusinessLogic {

   ... = new SpecialDataLayer();


}
```

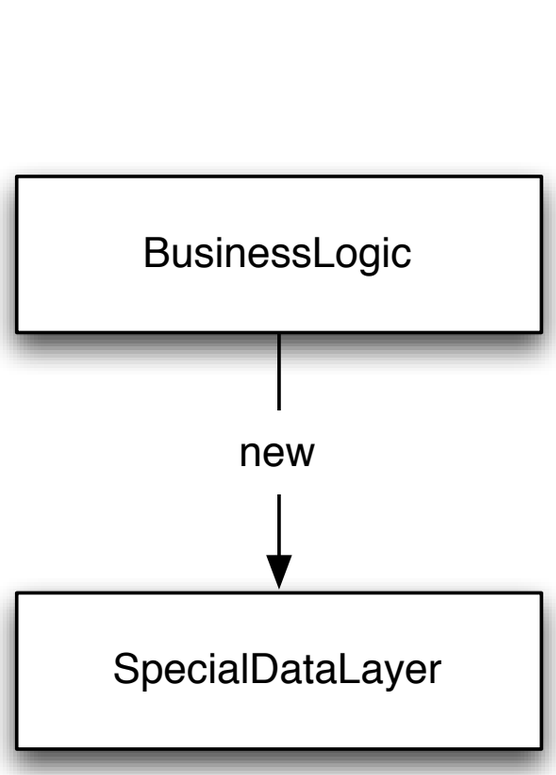


Direkte Kopplung!

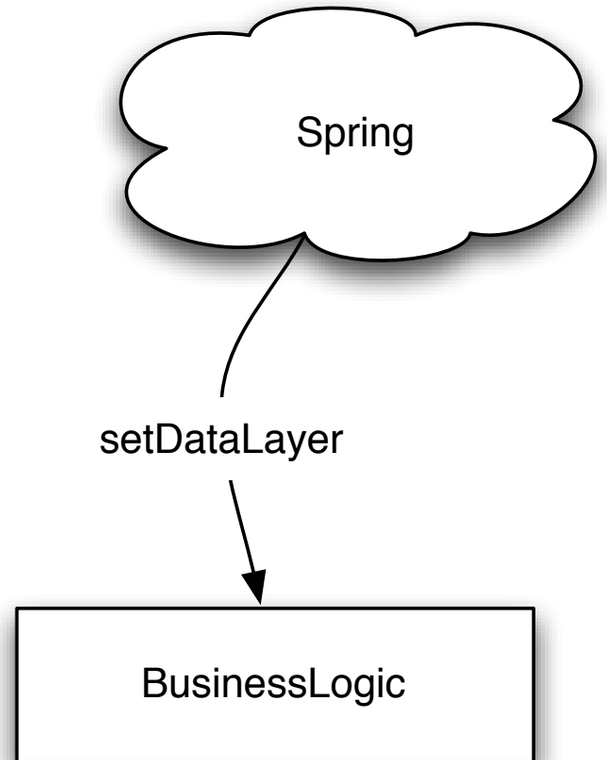
# Alternative: setter

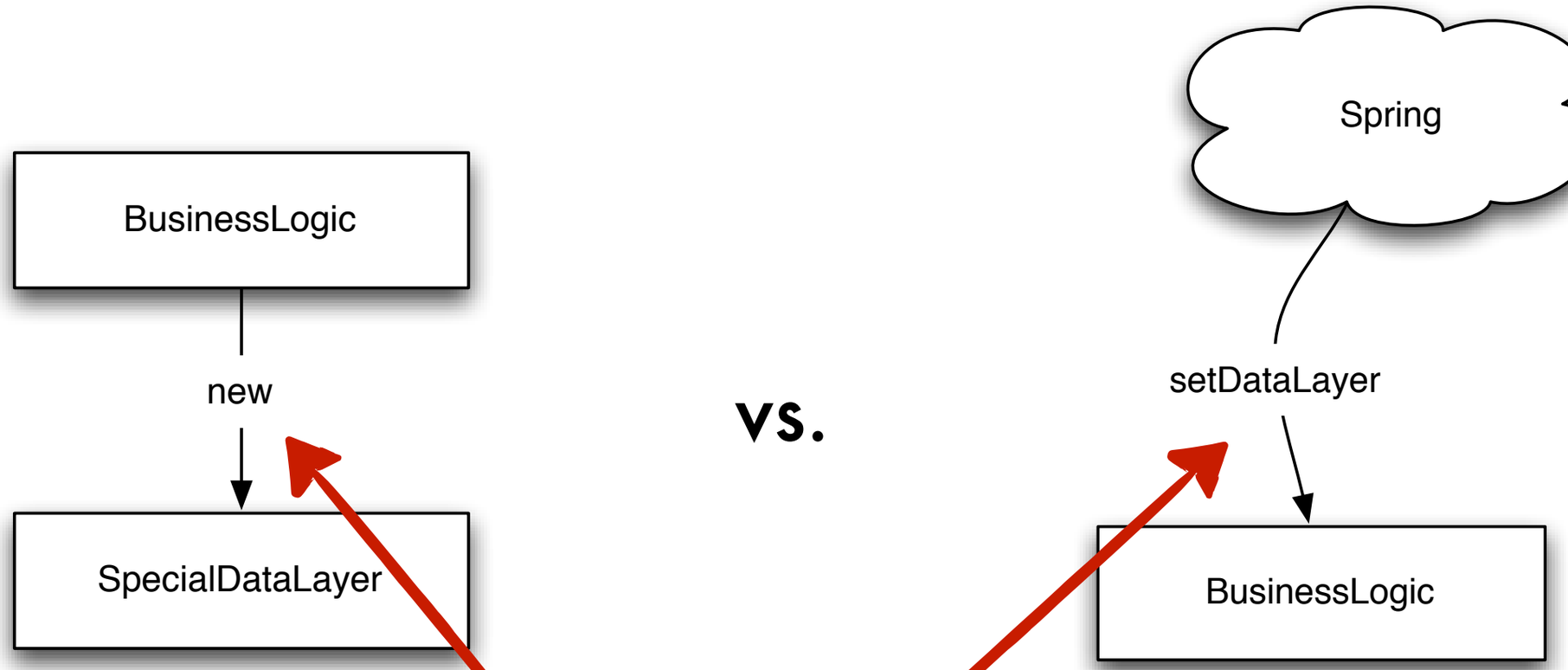
```
public class BusinessLogic {

   private DataLayer dataLayer;


   public void setDataLayer(DataLayer dataLayer) {
      this.dataLayer = dataLayer;
   }

}
```




BusinessLogic
new
SpecialDataLayer

vs.


Spring
setDataLayer
BusinessLogic




# Umkehrung der Kontrolle über Abhängigkeiten!

# Spring Konfiguration

```
<beans>

  <bean name="businessLogic" class="de.fhkoeln.ass.ff.business.BusinessLogic">
    <property name="dataLayer" ref="specialDataLayer" />
  </bean>

  <bean name="specialDataLayer"
  class="de.fhkoeln.ass.ff.datalayer.SpecialDataLayer" />

</beans>
```



# Spring JDBC



# JDBC-Unterstützung

- JDBC ist Schnittstelle zwischen DB und Java
- JDBC Coding
  - aufwändig
  - fehlerbehaftet



```
Statement stmt = conn.createStatement();
try {
    ResultSet rs = stmt.executeQuery( "SELECT * FROM MyTable" );
    try {
         doStuffWith(rs);
        }
    } finally {
        rs.close();
    }
} finally {
    stmt.close();
}

...

conn.close()
```



```
Statement stmt = conn.createStatement();
try {
    ResultSet rs = stmt.executeQuery( "SELECT * FROM MyTable" );
    try {
         doStuffWith(rs);
        }
    } finally {
        rs.close();
    }
} finally {
    stmt.close();
}

...

conn.close()
```

Viel Overhead
für ein Select



# Vereinfachung von Spring: JDBC-Template

- Erzeugung aus DataSource, durch Dependency Injection oder Vererbung
- JdbcTemplate enthält Methoden zum Ausführen von SQL-Statements:
  - `execute(), query(), update(), batchUpdate()`
  - selbständige Erzeugung von Connections und Aufräumen von Ressourcen
  - notfalls auch Arbeit mit JDBC-Klassen direkt möglich.



# Beispiel

```
getJdbcTemplate()
   .update(„DELETE FROM KUNDE WHERE ID=?“, new Object[] {new Integer(id)} );
```



# Ladies and Gentlemen,

# Start your Engines!



*Fast and Furious:*

# Wahnsinnige Spring Features

Fachhochschule Köln
Cologne University of Applied Sciences

# Rekapitulation

# Spring

- Fragen zu Dependency Injection



# Spring Testing



# Unit / Integration Testing

- Unit Testing
  - Unit als kleinste kompilierbare Einheit (In Java: Klasse)
  - Beinhaltet nicht den Aufruf von Sub-Komponenten oder jegliche sonstige Kommunikation mit anderen Komponenten
  - Besteht zu anderen Komponenten ein Abhängigkeitsverhältnis
    - Wird die Komponente ge-mockt
    - simuliert
    - oder durch vertrauenswürdige Komponenten ersetzt



# Unit / Integration Testing

- Integration Testing
  - Komponenten werden in ihrem Zusammenspiel getestet
  - Beinhaltet Aufruf von Sub-Komponenten
  - Im Gegensatz zum Unit-Testing muss kompletter Kontext / Infrastruktur vorhanden sein
  - Längere Laufzeiten



# Spring und Testing

- Unit-Testing
  - Mocks für JNDI, Servlet API und Portlet API
  - Supportclasses vor allem für die Nutzung von Reflections und Testing von Spring MVC
- Integration Testing
  - Testing ohne Deployment
  - Testet korrekte IoC-Konfiguration
  - Support für Transaktionen (Rollbacks), Test Fixtures, uvm.



# Spring Transaction-Management



# Transaktionen

- einzelne und unteilbare Transaktionen müssen im Ganzen gelingen oder scheitern
- ACID Kriterien
- Commit oder Rollback der Transaktion



# Spring Transaktionen

- Reichhaltige Unterstützung von Transaktionen
- Einheitliches Transaktionsmodell über andere Transaktions-APIs (JTA, JDBC, Hibernate, usw)
- Deklarativ oder Programmatisch
- Integriert mit Springs Datenzugriffs-Abstraktionen



# Beispiel

```
public class ServiceClass {

  @Transactional
  public void update(Thing thing) {
    dao.update(thing);
    ...
  }

}
```

# Spring ORM Integration



# ORM

- (O)bject (R)elational (M)apper
  - Bilden Objekte auf relationale Strukturen ab
  - ORM-Tools automatisieren Zugriff auf Datenbank

# Springs ORM Support

- Integration mit Hibernate, JDO, Oracle TopLink, iBatis, JPA
- Unterstützung mit einheitlichem Transaktionsmanagement und Exception-Hierarchien
- Nutzung von Spring DAO Templates oder direkte Nutzung der ORM API



# Beispiel

```
public class HibernateProductDao extends HibernateDaoSupport implements
ProductDao {
   public Collection loadProductsByCategory(String category) throws
   DataAccessException, MyException {
      Session session = getSession(false);
      try {
         Query query = session.createQuery("from test.Product product where
         product.category=?");
         query.setString(0, category);
         List result = query.list();
         if (result == null) {
            throw new MyException("No search results.");
         }
         return result;
      }
      catch (HibernateException ex) {
         throw convertHibernateAccessException(ex);
      }
   }
}
```



# Spring Web Integration



# Web Frameworks

- Qual der Wahl bei Web Frameworks
- Spring MVC
  - eigenes request-basiertes MVC Framework
- Integration anderer Frameworks
  - Struts, JSF 1 & 2, WebWork 2, Tapestry 3 & 4



# Spring Web Service Integration



# Unterstützung von Standard WS-APIs

- Veröffentlichen mit JAX-RPC
- Zugreifen mit JAX-RPC
- Veröffentlichen mit JAX-WS
- Zugreifen mit JAX-WS



# Veröffentlichen eines WS mit JAX-WS

- JAX-WS funktioniert weitgehend mit Annotationen
- SpringBeanAutowiringSupport-Klasse als Basis
  - Wrapping unserer Services



```
@WebService(serviceName="AccountService")
public class AccountServiceEndpoint extends SpringBeanAutowiringSupport {

   @Autowired
   private AccountService biz;

   @WebMethod
   public void insertAccount(Account acc) {
      biz.insertAccount(acc);
   }

   @WebMethod
   public Account[] getAccounts(String name) {
      return biz.getAccounts(name);
   }
}
```



- Spring 3.0



# Spring 3.0

- Seit Mitte Dezember ist 3.0 final
- basiert auf Java 5, unterstützt Java 6
- kompatibel mit J2EE 1.4, JEE 5 und teilweise mit JEE 6



- Spring Expression Language
- IoC Verbesserungen
- Objekt / XML Mapping (OXM, aus Spring Web Services Projekt)
- leistungsfähiger REST Support
- @MVC Annotations
- Deklarative Validierung
- Unterstützung eingebetteter Datenbanken



# Ladies and Gentlemen,

# Start your Engines!