Agenda

Integration

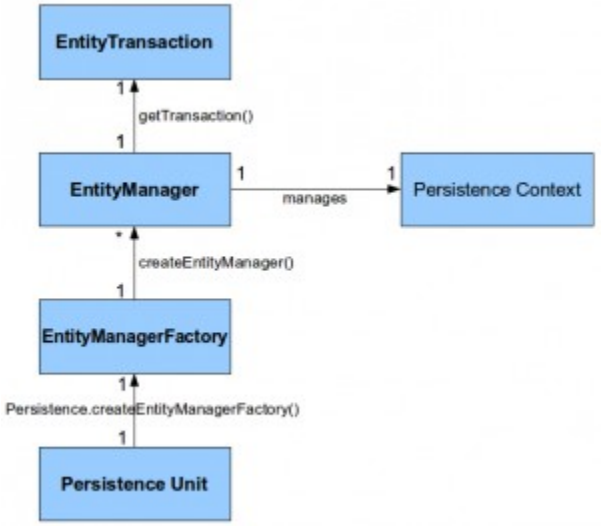
Java SE

Spring

EJB

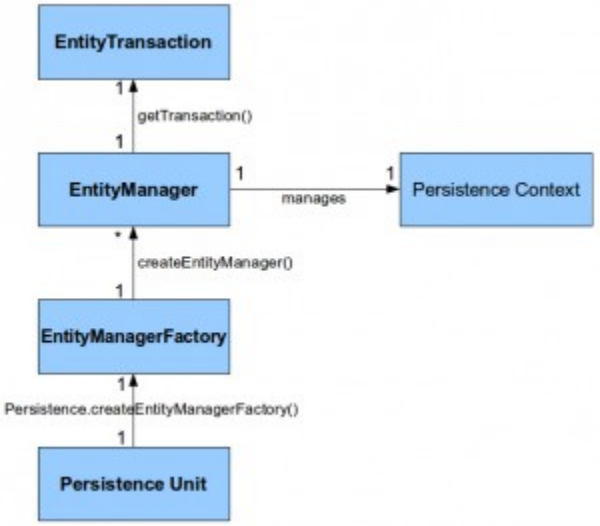
Unmanaged Environment

- Application-Managed Entity Manager
  - Entity Manager wird von der Anwendung selbst verwaltet
  - zum Erzeugen wird eine EntityManagerFactory verwendet
  - Transaktionen durch:
    - JTA
    - lokale Transaktion (z. B. JDBC Transaktion)



Managed Environment

- Container-Managed Entity Manager
  - innerhalb eines Java EE Containers im Einsatz
  - für das Erzeugen und Schließen des Entity Managers verantwortlich
  - Java EE Container verwaltet JTA (Java Transaction API) Transaktionen
  - Verwaltung geschieht transparent für die Anwendung
  - Zugriff auf den Entity Manager erhält die Anwendung durch:
    - Dependency Injection
    - JNDI
    - Definition durch @PersistenceContext



Einsatz in Java SE

- explizites Verbindungs- und Transaktionsmanagement

```
EntityManagerFactory emf = SingleEMF.getInstance() ;
try {
entityManager = emf.createEntityManager() ;
trx = entityManager.getTransaction();
trx.begin();
entityManager.persist(user1);
…
trx.commit ();
   }
catch  (Exception e) {
if  (trx  ! = null)  trx.rollback();
   }
finally {
if  (entityManager  != null)  entityManager.close();
}
```

Aufgabe

- Demo 1: Java-SE-Integration

Agenda

Integration

Java SE

Spring

EJB

Spring

- Open Source Java/Java EE Application Framework
  - steht unter Apache Licence 2.0
- Entstand durch Beispiel zu Buch von Rod Johnson
  - Expert One-on-One JEE Design and Development (WROX 2002)
- Spring wird heute von SpringSource Inc. weiterentwickelt

Spring Ziele

- Vereinfachte und vereinheitlichte API- Schicht über viele Java-SE-APIs, Java-EE-APIs und Open-Source-Frameworks
- Aufbau von Objektnetzen mit Dependency Injection (DI)
- Unterstützung für aspektorientierte Programmierung (AOP)

Einsatzbereiche von Spring

Integration

Java SE

Spring

EJB

- Enterprise Anwendungen basierend auf POJOs (Plain Old Java Objects)
  - Persistenz mit ORM oder nativen JDBC
  - Webanwendungen mit Spring MVC und WebFlow
  - Web Services mit Spring Web Services -u.v.m.

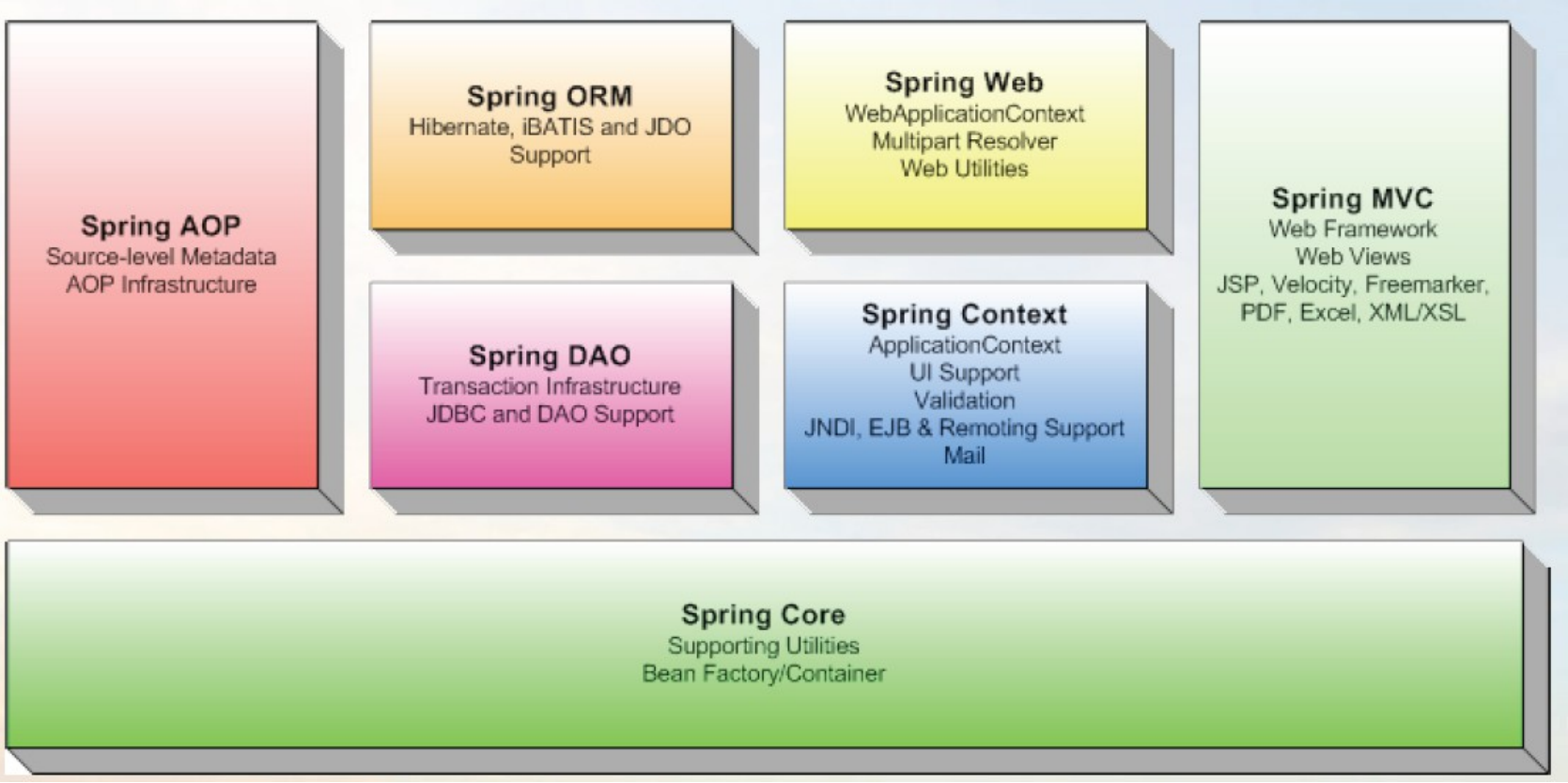
Spring Module

Integration

Java SE

Spring

EJB



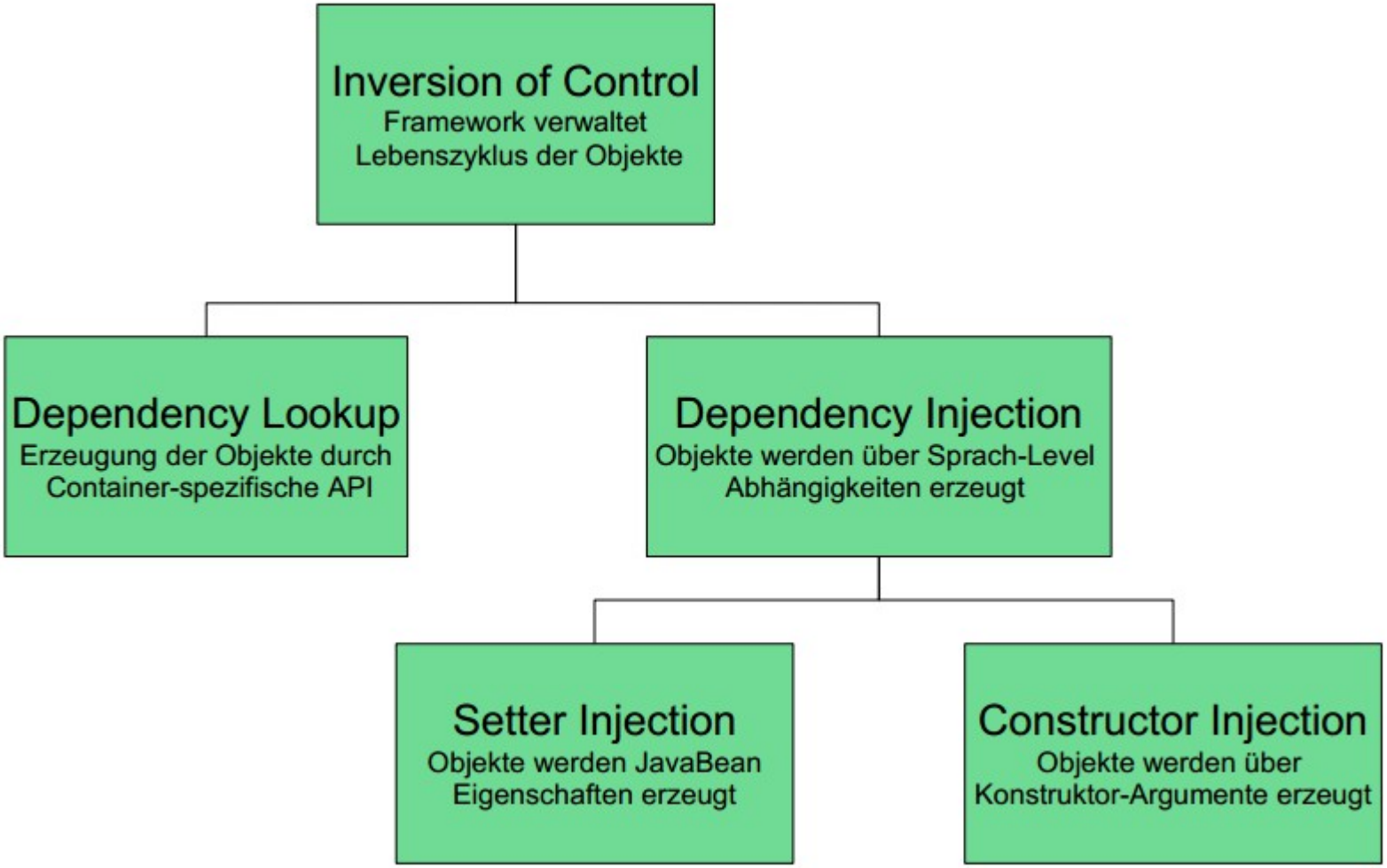
Inversion of Control (IoC)

- IoC Paradigma beschreibt die Arbeitsweise von Frameworks
- Nicht die Anwendung steuert den Kontrollfluss, sondern das Framework
- Beispiel in Java sind Listener
- Sogenanntes „Hollywood-Prinzip":
  "don't call us, we'll call you"

Dependency Injection (DI)

- Ist ein Entwurfsmuster („Fabrik“)
- Dient zur Erzeugung von Objektnetzen
- Überträgt die Verantwortung für das Erzeugen von Objekten an das Framework

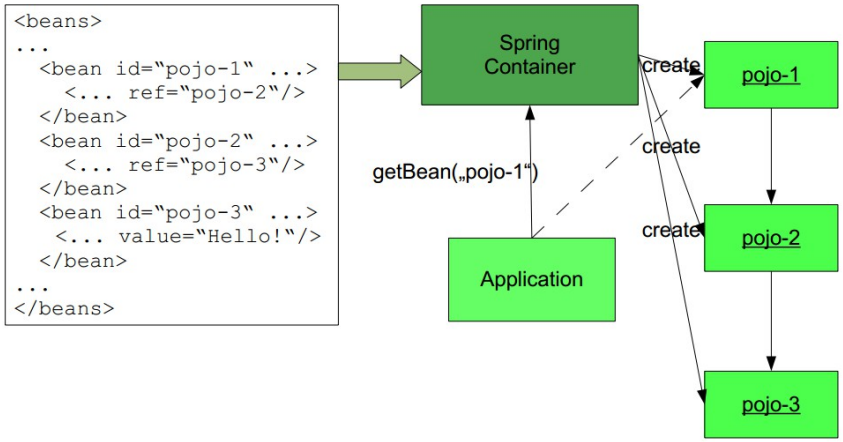
IoC und Formen der DI



Spring Konfigurationsdatei

- Beschreibt das Objektnetz
- Beinhaltet ganz normale Java Objekte (POJOs)
- Die Objekte heißen „Spring-Beans„
- Jedes Spring-Bean hat einen Namen bzw. eine ID
  - -und man kann diesem einen Wert oder eine Referenz zuweisen

```
<beans>
...
  <bean id="pojo-1" ...>
    <... ref="pojo-2"/>
  </bean>
  <bean id="pojo-2" ...>
    <... ref="pojo-3"/>
  </bean>
  <bean id="pojo-3" ...>
    <... value="Hello!"/>
  </bean>
...
</beans>
```



  - alternative Verknüpfung über Autowiring und Annotationen

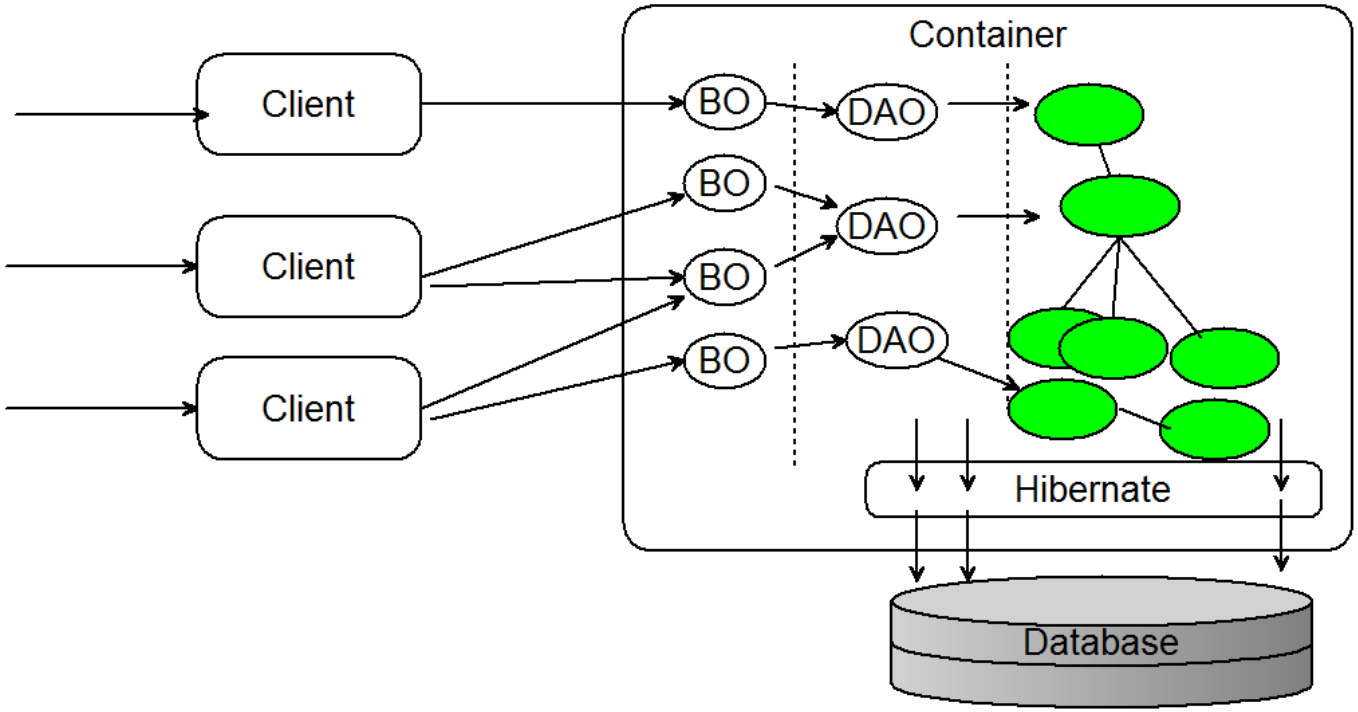
EE Architektur

Integration

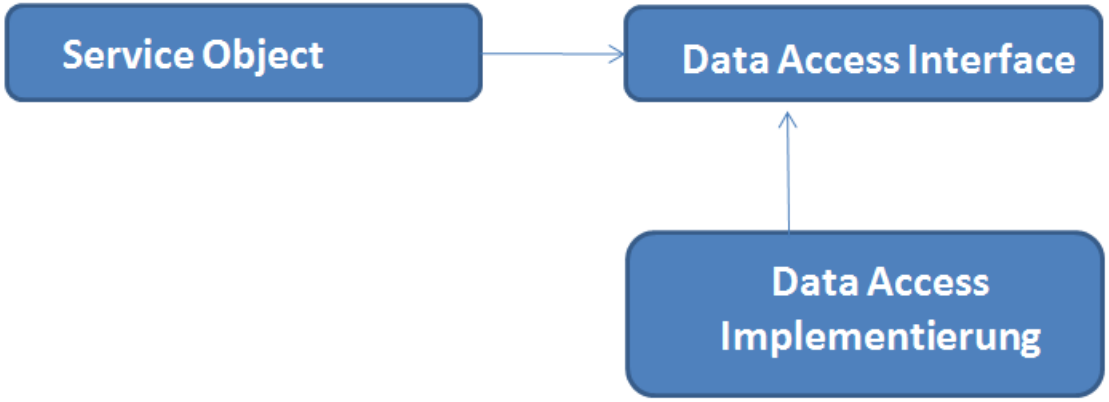
Java SE

Spring

EJB

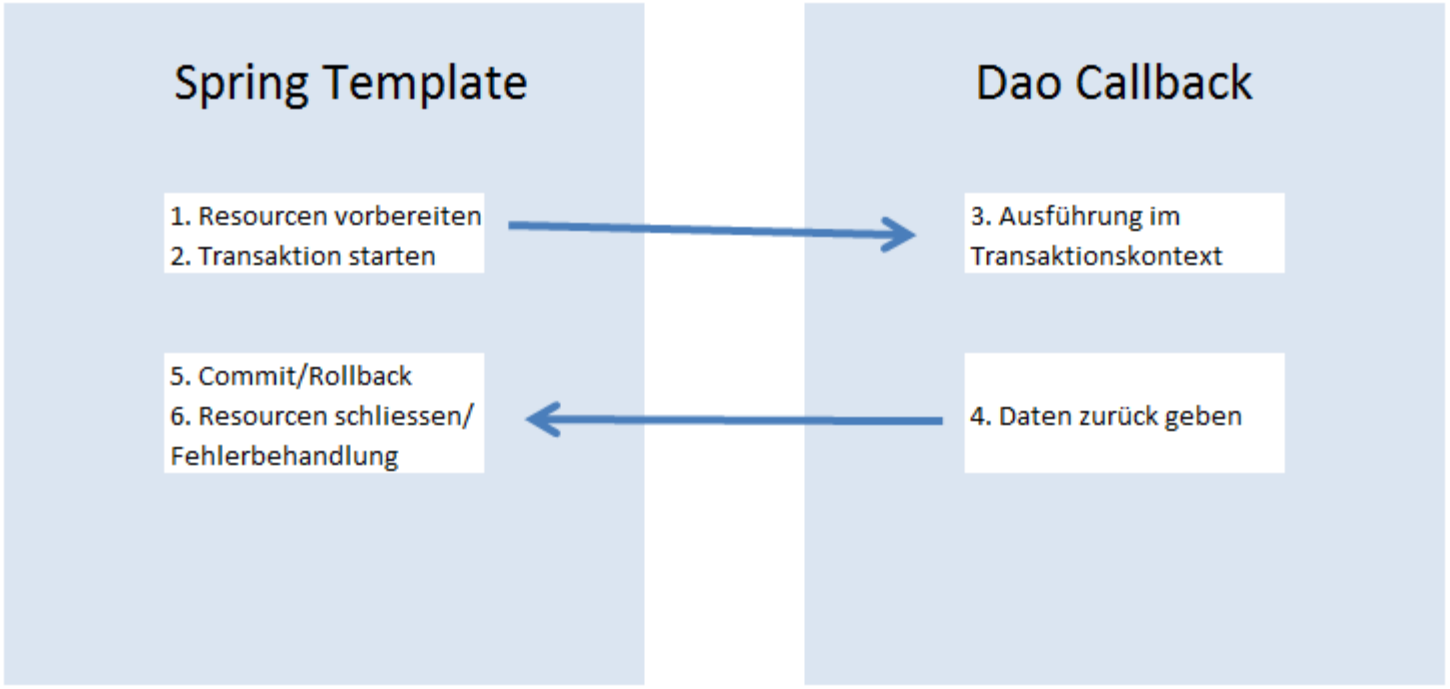


Datenzugriff mit Spring DAO




```
public interface CustomerManagerDAO {
    public abstract Collection getAllCustomers ( ) throws
    DataAccessException;
    public void createNewCustomer(Customer card) throws
      DataAccessException;
}
```

Template und Callback Klassen - Verantwortlichkeiten



Spring DAO mit JPA/Hibernate

- **Spring verwaltet Zugriff auf Session und Transaktion**
  - Spring regelt Transaktionsmanagement
  - Arbeitet mit JTA oder lokalen Transaktionen ohne Änderungen
- **Verwendung von JpaTemplate**
  - Persistenz-Operationen werden im DAO auf der JpaTemplate-Instanz durchgeführt.
  - Intern werden diese an den zugrunde liegenden Entity-Manager delegiert.
  - Einheitliches Exceptionhandling
- **JPA und JDBC in gleicher Transaktion verwendbar**

Beispiel Konfiguration


```
<bean id="emf"
    class = "org.jpa.LocalEntityManagerFactoryBean">
    <property name="persistenceUnitName"
    value="jpaDatabase" />
  </bean>

  <bean id="JpaTemplate"
  class="org.springframework.orm.jpa.JpaTemplate"> '
    <property name="entityManagerFactory" ref="emf"/>
  </bean>

  <bean id="xxxDAO"
  class = "de.example.dao.XXXDAOImpl ">
    <constructor-arg>
            <ref bean="JpaTemplate"/>
    </constructor-arg>
  </bean>
```

DAO-Implementierung


```
@Repository
@Transactional
public class XXXDaoImpl implements XXXDao {
    @PersistenceContext
    private EntityManager em;
    @Autowired
    private JpaTemplate jpaTemplate;
…
}
```

Aufgabe

- Demo 2: Spring-JPA-Integration

Agenda

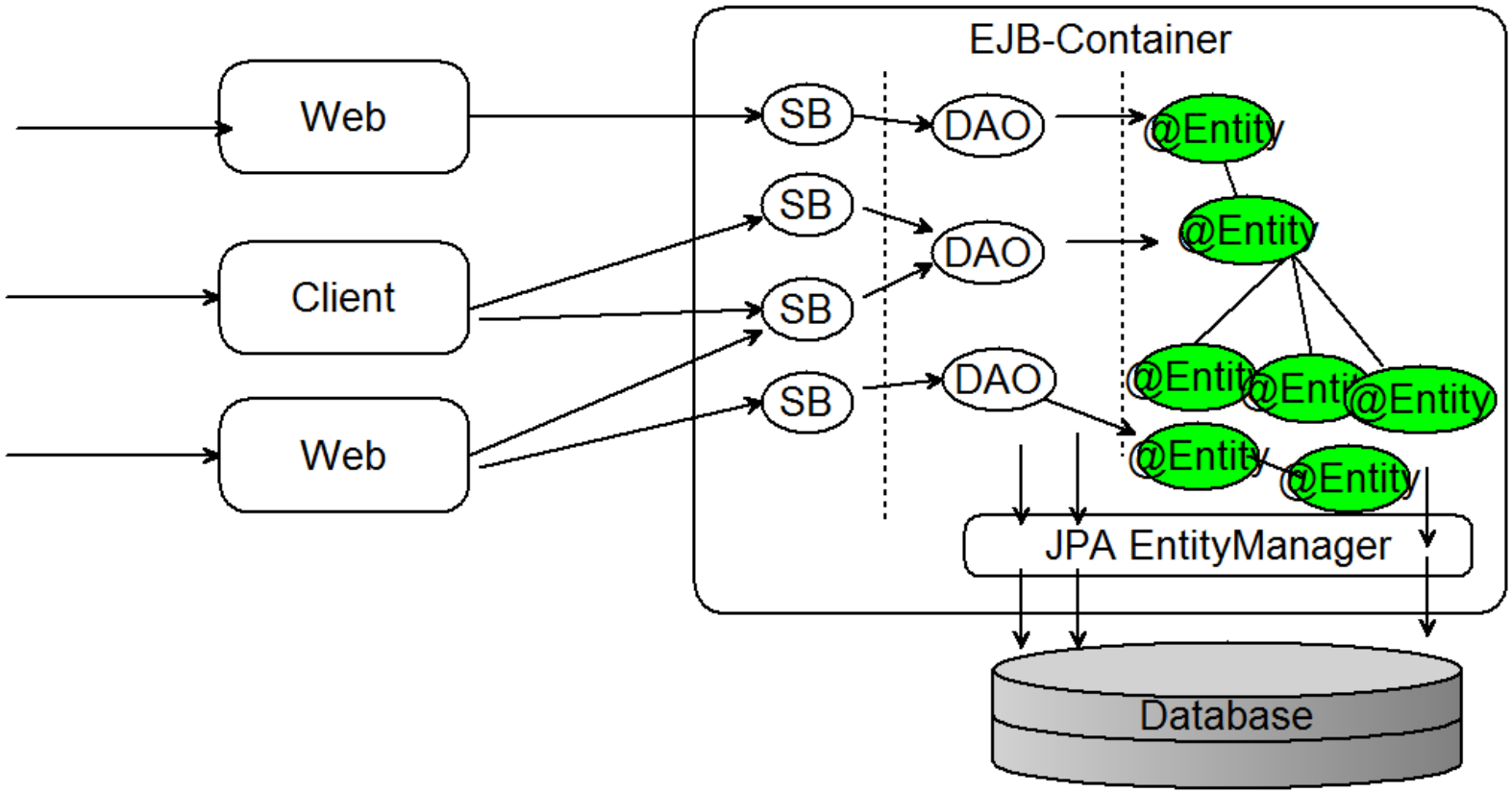
Integration

Java SE

Spring

EJB

Persistenz mit EJB 3



Verwendung EntityManager

- Session Bean braucht einen EntityManager
  - wird durch Container erzeugt und verwaltet
  - kann über DI (Dependency Injection) eingesetzt werden

```
@Stateless
public class PersonServiceBean implements
  PersonService{
     @PersistenceContext(name="kurs")
     private EntityManager manager;
     …
}
```

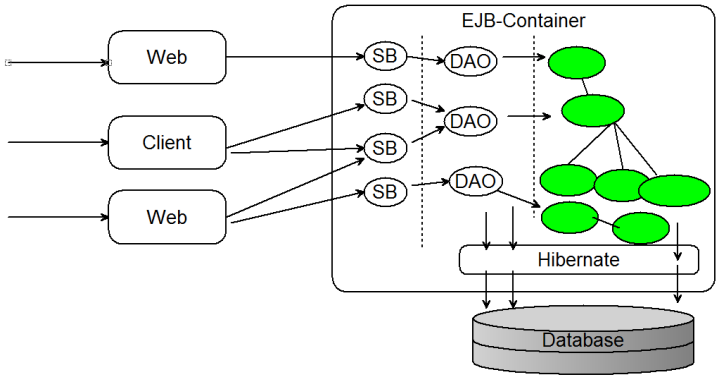
Konversationen (Wizards)


```
@Stateful

public class PersonServiceBean implements PersonService{
       private Person person;
       @PersistenceContext(type=PersistenceContextType.EXTENDED)
       private EntityManager em;
           …
       // Dialog 1,  2,  3,   . . .
       @TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
       public void step1 {
              person = new Person();
              em.persist(person);
       }
       // letzter Dialog
       @Remove // schließt EntityManager und entfernt Stateful SB
       @TransactionAttribute(TransactionAttributeType.REQUIRED)
       public void stepX(String email) {
           person.setEmail(email);
       }

}
```

- stepX() schreibt alle offenen Änderungen in Datenbank, schließt den EntityManager und entfernt die stateful Session Bean

Session Facade EJB 2.1 (Hibernate)

- Entity Beans waren in EJB 2.1 ein Antipattern
- Workaround: Hibernate ersetzt Entity Beans
- SessionFacade arbeitet direkt mit Hibernate (ggf. über DAOs)
  - Session Factory wird an JNDI gebunden
  - Appserver verantwortlich für Connections
  - Hibernate bindet sich automatisch in Transaktionsmanagement vom Container



Hibernate im Application Server

- SessionFactory wird in JNDI abgelegt
  - Konfiguration in hibernate.cfg.xml (automatische Bindung an JNDI)

```
static {
   new Configuration().configure()
   .buildSessionFactory();
…
}
public static SessionFactory getSessionFactory() {
   Context ctx = new InitialContext( ) ;
   String jndiName =
"java:hibernate/HibernateFactory”;
return  (SessionFactory)ctx.lookup(jndiName);
}
```

Aufgabe

- Demo 3: EJB-Integration