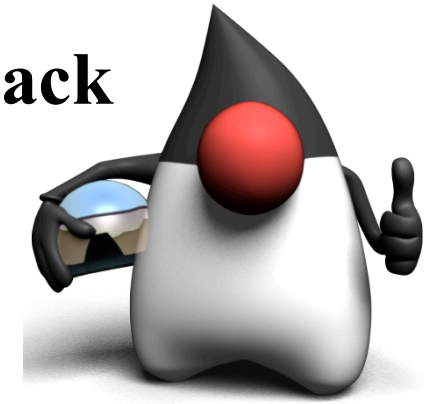
GFU Cyrus AG

Trainer-Workshops

Java-Track



Dipl.-Inform. Thomas Much

Ver.: TM-2008/11/13-GFU

Vorträge im Java-Track (Samstag, 15.11.2008)

08:30 - 09:45	**Java EE, EJB3+JPA**	Thomas Much
09:50 - 11:05	**Spring+Hibernate**	Torsten Friebe
11:15 - 12:30	**JSF, Maven**	Ingo Düppe
12:30 - 13:30	Mittagessen	
13:30 - 15:30	"Die Trainerpersönlichkeit"	Markus Röder
15:30 - 16:45	**Testautomatisierung**	Carsten Siedentop
16:50 - 17:30	**Frage- und Diskussionsrunde (Agenda "Java intensiv"?)**	
17:30 - 18:00	Abschlussbesprechung	Hagen Cyrus

Vortrag "EJB 3 + JPA"

- **Überblick Java-Technologien**
- Schwerpunkt Java EE 5

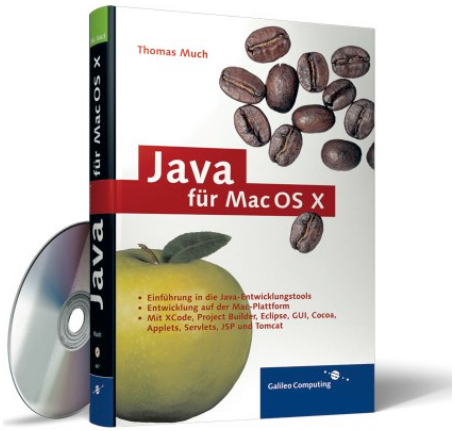
- **EJB 3**
- mit Eclipse 3.4, Ant, Glassfish v2

- **Java Persistence API (JPA)**
- mit Hibernate 3.2
- Standalone und im Application Server

Ihr Trainer

- **Thomas Much**
  thomas@muchsoft.com

- IT-Architekt, Softwareentwickler & Projektcoach:
  - Java seit 1996, C++ seit 1993, davor Object Pascal, Eiffel, ...
  - derzeit v.a. Web- und Enterprise-Applikationen
  - div. Systeme (Windows, Linux, Mac OS X)
- Programmiertrainer seit 1997, für die GFU seit 2001:
  - Java, C, C++, XML, OO, UML
  - http://www.muchsoft.com/
- Autor:
  - „Java für Mac OS X“
    Galileo Computing 2005
    630 Seiten, ISBN 3-89842-447-2



Java EE 5

Java-Versionen

- Java Standard Edition (J2SE, JSE, Java SE)
  - J2SE 1.4 (02/2002), End Of Service Life (EOSL) seit 30.10.2008
  - J2SE 5.0 (09/2004), End Of Service Life am 30.10.2009 (!)
  - **Java SE 6 (12/2006)**
  - Java SE 7 (2009 ?)

- **Java Enterprise Edition (J2EE, JEE, Java EE)**
  - J2EE 1.0 (12/1999)
  - J2EE 1.2 (05/2000)
  - J2EE 1.3 (08/2001)
  - J2EE 1.4 (11/2003)
  - **Java EE 5 (05/2006)**
  - Java EE 6 (2008 ???)

Java EE 5: Zertifizierte Server

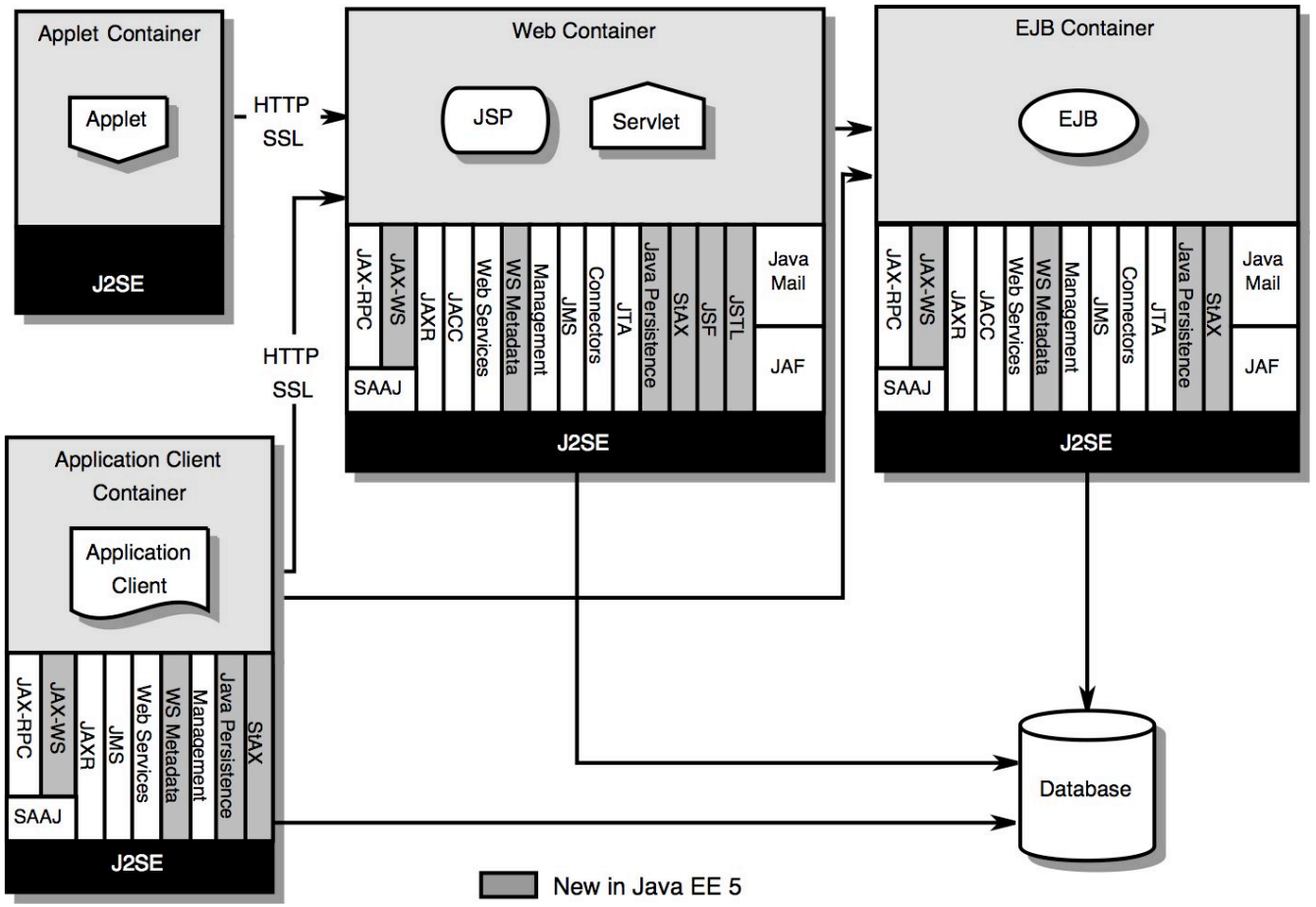
- **Glassfish** (Referenzimplementierung) 
- Sun AS 9

- IBM WebSphere 7
- Bea WebLogic 10
- SAP NetWeaver JEE 5 Edition
- Oracle OC4J 11

- Apache Geronimo 2
- JBoss 5 (derzeit CR, noch nicht final!)

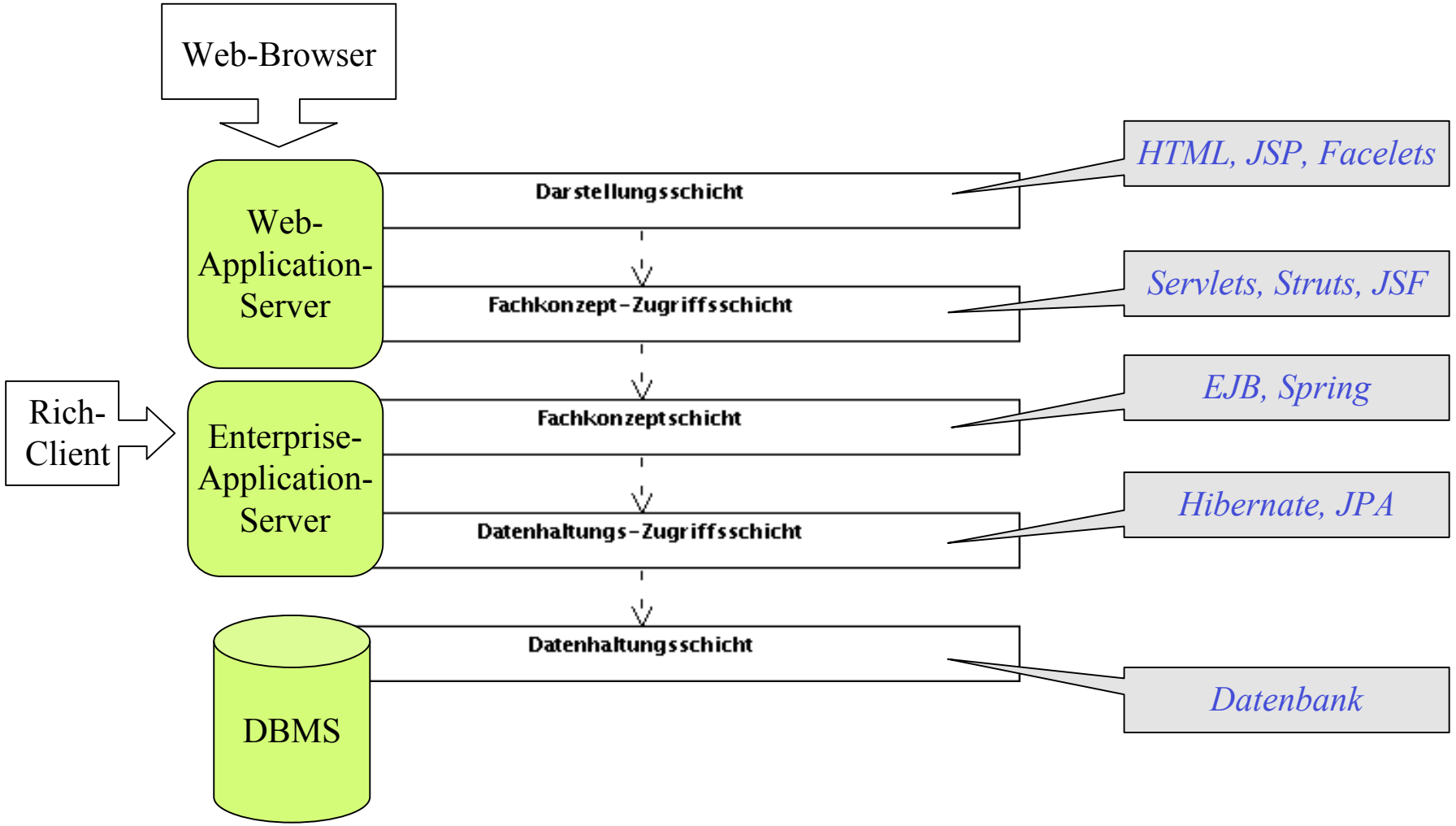
- u.a.

Java EE 5: Technologien

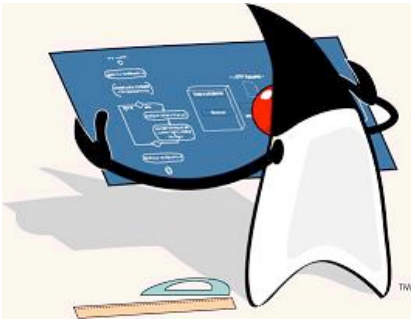


- Wichtige Änderung: **EJB** 3.0
- Neu: **JSF** 1.2, **JSTL** 1.2, **JPA** 1.0, **StAX** 1.0, **JAXB** 2.0, **JAX-WS** 2.0

Architektur einer Enterprise-Applikation



Alternativen und Ausblick



- Alternativen zu JSF:
  - Tapestry 5, Struts 2, Wicket, Stripes (?)
- Alternativen zu JPA:
  - Hibernate, TopLink
- Alternativen und Ergänzungen zu EJB:
  - Spring, Seam
- Ausblick für Seam:
  - WebBeans

Ausblick Java EE 6

- Servlet 3.0
  - Konfiguration alternativ über Annotationen
- EJB 3.1
  - AppServer nicht mehr zwingend, Singleton-Beans u.a.
- JPA 2.0
  - bessere Optimierungs-, Filter- und Suchmöglichkeiten
- sollte eigentlich noch 2008 erscheinen – wird nun vermutlich Mitte 2009

Ausblick Java SE 7

- JVM-Erweiterungen für dynamische (Script-)Sprachen
- Superpackages
- Closures (?)
- u.a.

- soll irgendwann 2009 erscheinen

Enterprise JavaBeans 3.0 (EJB 3)

EJBs ab Java EE 5

```
public class Adresse {

  ...

}
```

+ @ =

POJO Annotation EJB

Enterprise JavaBeans (EJB)

- **Komponenten-Modell**
  - *Session Beans*
    - Stateless Session Beans (vergleichbar mit Web-Services)
    - Stateful Session Beans
  - *Message Driven Beans* (MDB)
  - *Entities* (früher *Entity Beans*)
- **Rahmenwerk (Framework)**
  - Transaktionen, Sicherheit, Pooling, Caching, ...

EJB-Versionen

- EJB 1.0 (≈ J2EE 1.0)
  - Session Beans und (optional) Entity Beans
- EJB 1.1 (≈ J2EE 1.2)
  - u.a. Entity Beans zwingend
- EJB 2.0 (≈ J2EE 1.3)
  - u.a. MDBs, Local-Interface zur Performance-Steigerung
- EJB 2.1 (≈ J2EE 1.4)
  - u.a. Web-Services, Timer

- **EJB 3 + JPA 1.0** (≈ Java EE 5)
  - s. nächste Folie

- EJB 3.1 + JPA 2.0 (≈ Java EE 6)

EJB 3 + JPA

- Enterprise JavaBeans 3.0
- Java Persistence API 1.0

⇨ ab Java EE 5

- **EJB 3**
  - Session Beans, Message Driven Beans
  - Web-Services, Timer, Transaktionen, ...
  - Konfiguration vor allem über "Annotationen", deutlich weniger XML-Dateien nötig
- **JPA**
  - Entities
  - Modellklassen, die persistiert (gespeichert) werden können
  - Ersatz für die alten, problematischen Entity Beans

Session Beans

- **Stateless Session Beans**
- **Stateful Session Beans** (mit *Conversational State*, "Gesprächszustand")

- sind nebenläufig und Thread-sicher
- bieten Transaktionen und Sicherheit
- können entfernt aufgerufen werden

- Stateless Session Beans
  - können auch als Web-Service aufgerufen werden
  - bieten Timer-Dienste und Interceptors
  - bieten automatisches *Pooling*

Stateless Session Bean (SLSB)

HalloWeltBean.java

```
import javax.ejb.*;

@Stateless
public class HalloWeltBean implements HalloWelt {

    public HalloWeltBean() { }

    public String getBotschaft() {
        return "Hallo EJB3-Welt am " + new Date();
    }
}
```

HalloWelt.java

```
@Remote
public interface HalloWelt {

    public String getBotschaft();
}
```

Servlet mit Dependency Injection (DI)

ServletMitDI.java

```
public class ServletMitDI extends HttpServlet {

    @EJB
    private HalloWelt hw;

    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
                          throws ServletException, IOException {

        String botschaft = hw.getBotschaft();

        request.setAttribute( "BOTSCHAFT",  botschaft );

        RequestDispatcher dispatcher = getServletContext()
                        .getRequestDispatcher( "/ausgabe.jsp" );
        dispatcher.forward( request, response );
    }
}
```

JSP-Ausgabe mit der Expression Language (EL)

ausgabe.jsp

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
                      "http://www.w3.org/TR/html4/loose.dtd">
<html>
   <head>
      <title>Ausgabe in einer JSP</title>
   </head>
   <body>

   <h1>Die Botschaft der Bean:</h1>

   <p>${requestScope.BOTSCHAFT}</p>
   <p>${BOTSCHAFT}</p>

   <hr>
   <p><a href="index.html">Zur&uuml;ck</a></p>

   </body>
</html>
```

Local- und Remote-Interfaces

```
@Stateless
public class HalloWeltBean implements HalloWelt, HalloWeltLocal {
    ...
}
```

```
@Remote
public interface HalloWelt {

    public String getBotschaft();
}
```

```
@Local
public interface HalloWeltLocal extends HalloWelt { }
```

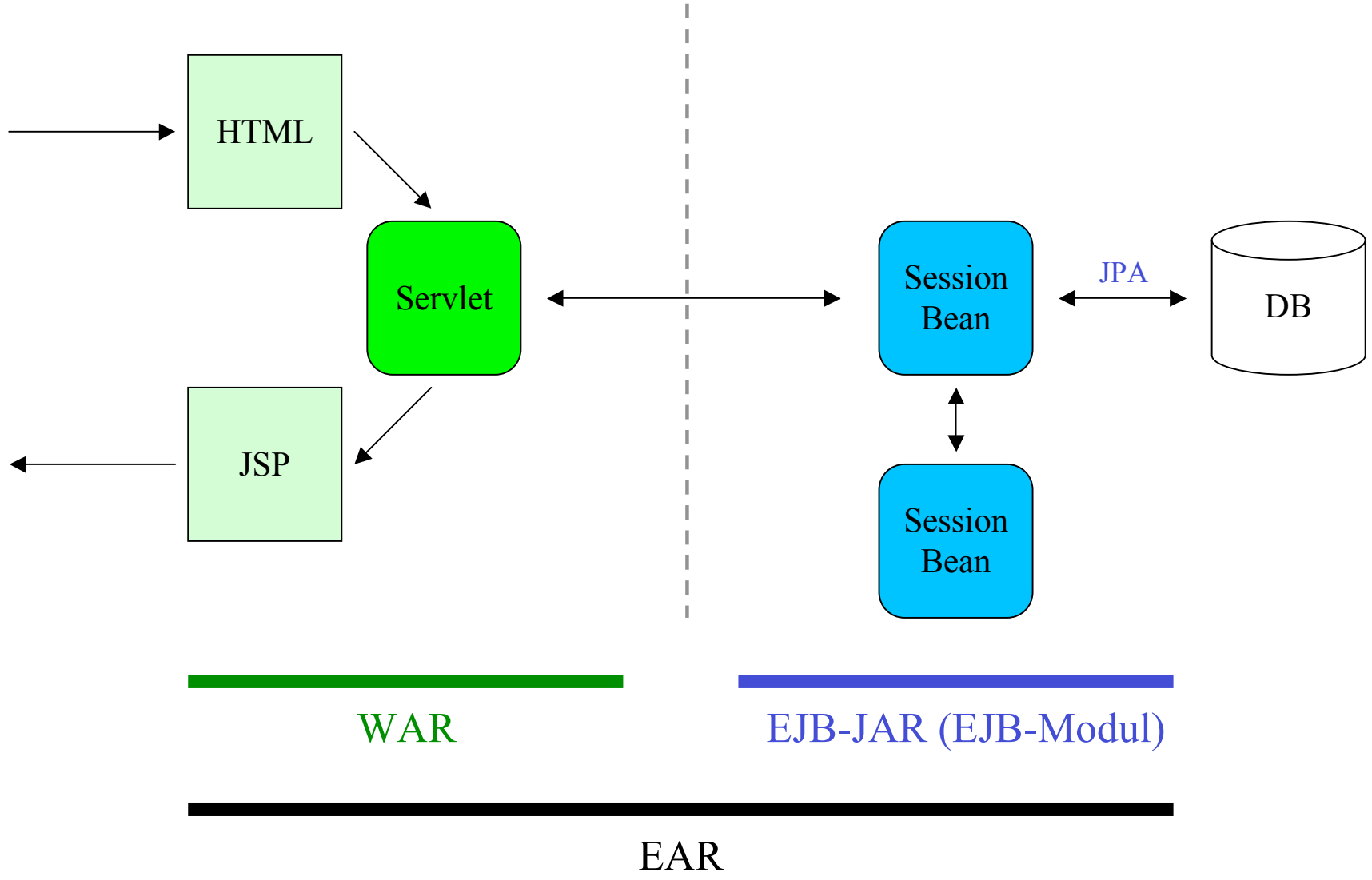
Ab EJB 3.1 vermutlich nicht mehr zwingend

Zugriff mit JNDI-Lookup

ServletMitJNDI.java

```
public class ServletMitJNDI extends HttpServlet {



    @Override
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
                          throws ServletException, IOException {

        HalloWelt hw = ServiceLocator.getBean(HalloWelt.class);

        String botschaft = hw.getBotschaft();

        ...
    }
}
```

Enterprise Application Archive (EAR)



Lebenszyklus von Session Beans

```
@Stateless
public class HalloWeltBean implements HalloWelt {

    public HalloWeltBean() { }

    public String getBotschaft() {
        return "Hallo EJB3-Welt am " + new Date();
    }


    @PostConstruct
    protected void init() {
        System.out.println( "*** init " + this );
    }

    @PreDestroy
    protected void destroy() {
        System.out.println( "*** destroy " + this );
    }
}
```

AOP light: Interceptors

```
@Stateless
@Interceptors(MeinLogger.class)
public class MeineBean implements MeinInterface {

    @Interceptors(GanzSpeziellerInterceptor.class)
    public void methode1() { ... }

    @ExcludeClassInterceptors
    public void methode2() { ... }
}
```

```
public class MeinLogger {

    @AroundInvoke
    public Object log(InvocationContext ctx) throws Exception {

        System.out.println( "Log: " + ctx.getMethod().getName() );

        return ctx.proceed();
    }
}
```

Container-Managed Transactions (CMT) (1)

```
@Stateless
@TransactionManagement(TransactionManagementType.CONTAINER)
public class BestellBean implements BestellFassade {

    @Resource private SessionContext context;

    @TransactionAttribute(TransactionAttributeType.REQUIRED)
    public void bestellen(int kundenNr)
                  throws KreditkartenException, VersandException {

        this.kreditkartePruefen();
        this.versenden();
    }


    private void versenden throws VersandException() {
        ...
    }

    ...
}
```

Container-Managed Transactions (CMT) (2)

KreditkartenException.java

```
@ApplicationException(rollback=true)
public class KreditkartenException extends Exception { }
```

VersandException.java

```
@ApplicationException(rollback=true)
public class VersandException extends Exception { }
```

- *System-Exceptions* = Fehler der technischen Infrastruktur
  - `RemoteExceptions` und `RuntimeExceptions`
  - werden verpackt als `EJBException` weitergeleitet
- *Application-Exceptions* = Fehler der Geschäftslogik
  - werden direkt zum Aufrufer weitergeleitet
  - aber standardmäßig *kein* Rollback!

Java Persistence API 1.0 (JPA)

Persistenz in Java

- von Hand mit SQL (JDBC)
- Serialisierung
- EJB Entity Beans bis inkl. EJB 2.x:
  BMP (Bean Managed Persistence)
  CMP (Container Managed Persistence)
  - weder alle OO- noch alle relationalen Möglichkeiten werden genutzt
  - persistente Klassen müssen speziell für EJB programmiert werden
- JDO (Java Data Objects)
- ...
- O/R-Mapper (ORM)

  ⇨ z.B. Hibernate

  \+ kann alle OO- und relationalen Möglichkeiten nutzen

  \+ erfordert keine spezielle Anpassung der persistenten Klassen

⇨ **JPA**

JPA

- *Entities* - die neuen Entity "Beans"
- alte EJB 2.x Entity Beans werden von Applikations-Servern aus Kompatibilitätsgründen noch unterstützt, werden hier aber nicht mehr besprochen.

- JPA kann auch standalone mit Java SE genutzt werden
- fester Bestandteil ab Java SE 6, Datenbank im JDK enthalten!

- diverse JPA-Implementierungen (Provider): Hibernate, TopLink, OpenJPA

- JBoss nutzt standardmäßig Hibernate
- Glassfish nutzt standardmäßig TopLink

POJOs als Entities

```
@Entity
@Table(name="BOTSCHAFTEN")
public class Botschaft {

    @Id
    @GeneratedValue(strategy=GenerationType.AUTO)
    @Column(name="BOTSCHAFT_PK")
    private Long   id;

    @Column(name="BOTSCHAFT_TEXT", nullable=false, length=160)
    private String text;


    protected Botschaft() { }

    public Botschaft(String text) { this.text = text; }


    public Long getId() { return id; }

    // getText(), setText(), equals(), hashCode() ...
}
```

Objekte speichern (persistieren)

TestSpeichern.java

```
EntityManagerFactory emf =
    Persistence.createEntityManagerFactory( "muchsoft" );

EntityManager manager = emf.createEntityManager();

try {
    manager.getTransaction().begin();

    Botschaft b1 = new Botschaft( "Hallo JPA-Welt!" );

    manager.persist( b1 );

    manager.getTransaction().commit();
}
catch (Exception e) {
    manager.getTransaction().rollback();
}

manager.close();
```

Konfig. für eine Datenbank

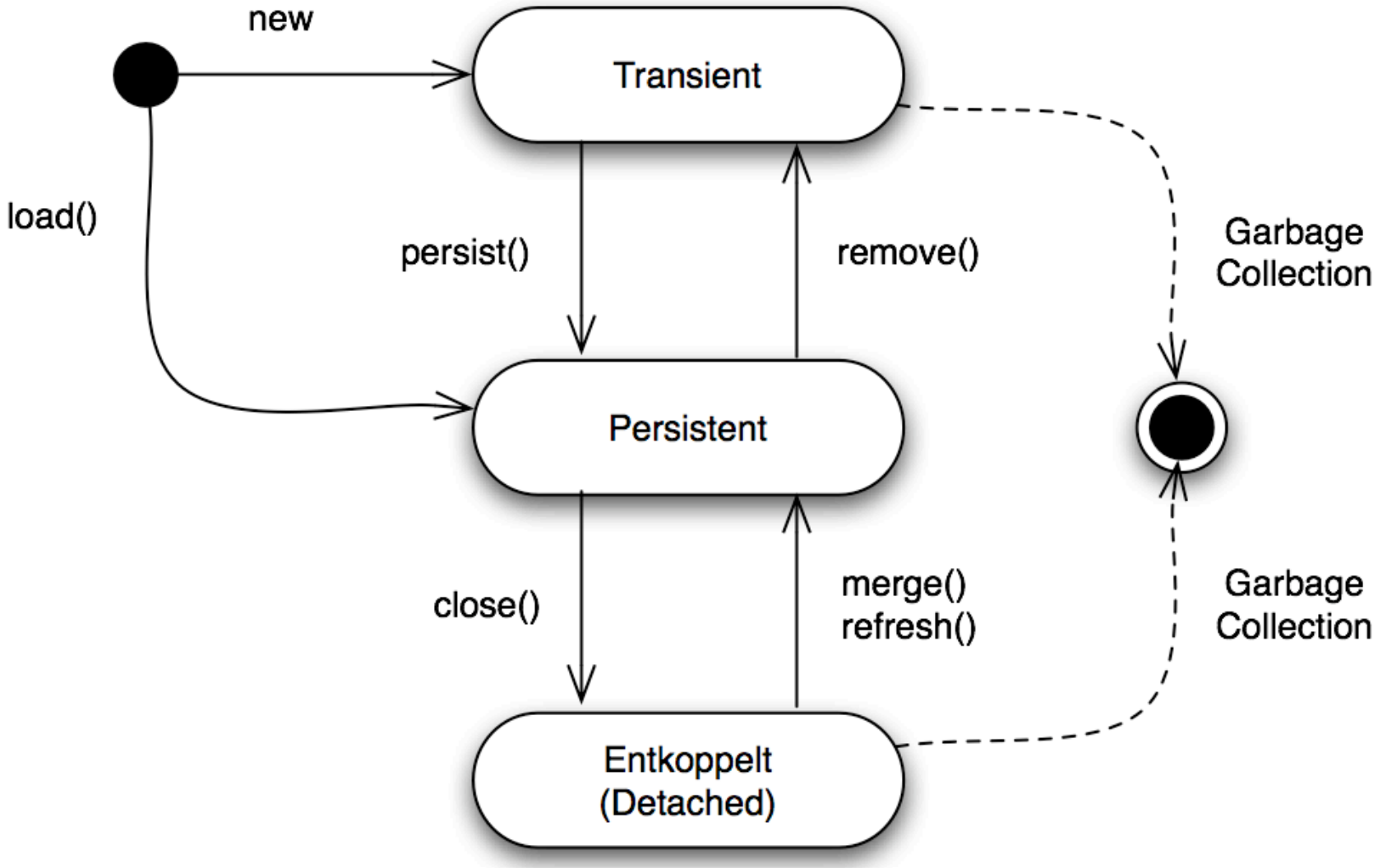
Datenbanksession/-connection

Persistente Objekte laden und ändern

TestLaden.java

```
EntityManagerFactory emf =
    Persistence.createEntityManagerFactory( "muchsoft" );

EntityManager manager = emf.createEntityManager();


Botschaft b1 = manager.find( Botschaft.class, 1L );

System.out.println( b1.getText() );



b1.setText( "das hier wird automatisch gesichert" );


manager.close();

b1.setText( "das hier wird *nicht* mehr gesichert" );

emf.close();
```

Lebenszyklus von Objekten mit JPA



Entkoppelte Objekte aktualisieren

```
EntityManager manager1 = emf.createEntityManager();

Botschaft b1 = manager1.find( Botschaft.class, 1L );

manager1.close();

b1.setText( "wichtige Änderung" );

EntityManager manager2 = emf.createEntityManager();

try {
    manager2.getTransaction().begin();

    Botschaft b2 = manager2.merge( b1 );

    manager2.getTransaction().commit();
}
catch (Exception e) {
    manager2.getTransaction().rollback();
}

manager2.close();
```

JPA konfigurieren (1)

META-INF/persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence>

   <persistence-unit name="muchsoft"
                     transaction-type="RESOURCE_LOCAL">

      <provider>org.hibernate.ejb.HibernatePersistence</provider>


      <class>com.muchsoft.jpa.first.Botschaft</class>


      <properties>
         ...
      </properties>

   </persistence-unit>

</persistence>
```

JPA konfigurieren (2)

META-INF/persistence.xml

```
<persistence-unit name="muchsoft"
                  transaction-type="RESOURCE_LOCAL">
   ...


   <properties>
      <property name="hibernate.dialect"
                value="org.hibernate.dialect.HSQLDialect" />
      <property name="hibernate.connection.driver_class"
                value="org.hsqldb.jdbcDriver" />
      <property name="hibernate.connection.username" value="sa" />
      <property name="hibernate.connection.password" value="" />
      <property name="hibernate.connection.url"
                value="jdbc:hsqldb:hsql://localhost/testdb" />
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.hbm2ddl.auto" value="create" />
   </properties>

</persistence-unit>
```

Abfragen (Queries) mit JPQL (1)

```
EntityManagerFactory emf =
    Persistence.createEntityManagerFactory( "muchsoft" );

EntityManager manager = emf.createEntityManager();


Query query = manager.createQuery( "select b from Botschaft b" );

List<Botschaft> liste = query.getResultList();

for (Botschaft b : liste) {
    System.out.println( b );
}


manager.close();

emf.close();
```

Abfragen (Queries) mit JPQL (2)

```
Query query = manager.createQuery(
        "select a from Adresse a where a.wohnort = 'Hamburg'" );

List<Adresse> lst = query.getResultList();
```

```
Query query = manager.createQuery(
        "select a from Adresse a where a.wohnort = ?1" );

query.setParameter( 1, "Hamburg" );

List<Adresse> lst = query.getResultList();
```

```
Query query = manager.createQuery(
        "select a from Adresse a where a.wohnort = :ort" );

query.setParameter( "ort", "Hamburg" );

List<Adresse> lst = query.getResultList();
```

Many-to-one (N:1) bidirektional (1)

```
@Entity
public class Benutzer {
    @Id @Column(name="BENUTZER_PK")
    private Long benutzerId;
    @ManyToOne
    @JoinColumn(name="BENUTZER_GRUPPE_FK",
                referencedColumnName="GRUPPE_PK")
    private Gruppe gruppe;
}
```

```
@Entity
public class Gruppe {
    @Id @Column(name="GRUPPE_PK")
    private Long gruppeId;
    @OneToMany(mappedBy="gruppe")
    private Set<Benutzer> benutzer = new HashSet<Benutzer>();
}
```

Many-to-one (N:1) bidirektional (2)

```
@Entity
public class Gruppe {
    @Id @Column(name="GRUPPE_PK")
    private Long gruppeId;
    @OneToMany(mappedBy="gruppe")
    private Set<Benutzer> benutzer = new HashSet<Benutzer>();

    ...

    public void addBenutzer(Benutzer b) {
        b.setGruppe( this );
        this.benutzer.add( b );
    }
}
```

"Convenience"-Methode

JPA: Was gibt es noch?

- alle Assoziationstypen (1:1, 1:N, N:M)
- Kaskadierung von Aktionen über Assoziationen

- LazyLoading (inkl. LazyInitializationException...)
- Möglichkeiten zur Fetch-Optimierung

- Komposition
- Vererbung

- ...

Lebenszyklus von Entities

```
@Entity
@EntityListeners(KundenMonitor.class)
public class Kunde {
    ...
}
```

```
public class KundenMonitor {

    public KundenMonitor() { }

    @PrePersist
    @PostPersist
    @PostLoad
    @PreUpdate
    @PostUpdate
    @PreRemove
    @PostRemove
    public void monitorKunde(Kunde k) { ... }
}
```

JPA mit XML-Mappings

META-INF/orm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<entity-mappings ... Schema/DTD ... >

   <persistence-unit-metadata>
      ...
   </persistence-unit-metadata>

   <package>com.muchsoft.schulung</package>

   <entity class="Benutzer" access="PROPERTY"
                              metadata-complete="true">
      <attributes>
         <id name="id">
            <generated-value strategy="AUTO" />
         </id>
      </attributes>
   </entity>

</entity-mappings>
```

JPA im Application Server (1)

META-INF/persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence>

    <persistence-unit name="muchsoft" transaction-type="JTA">

        <provider>
            oracle.toplink.essentials.PersistenceProvider
        </provider>

        <jta-data-source>jdbc/__default</jta-data-source>

        <properties>
            <property name="toplink.ddl-generation"
                      value="drop-and-create-tables" />
        </properties>

    </persistence-unit>

</persistence>
```

JPA im Application Server (2)

```
@Stateless
public class AdresseEAOBean implements AdresseEAO {

    @PersistenceContext(unitName="muchsoft")
    private EntityManager manager;


    public Adresse save(Adresse adr) {
        return manager.merge( adr );
    }

    public Adresse findById(Long id) {
        Adresse adr = manager.find( Adresse.class, id );
        return adr;
    }

    public List<Adresse> findAll() {
        Query q = manager.createQuery("select a from Adresse a");
        return q.getResultList();
    }
}
```

EAO ≈ DAO

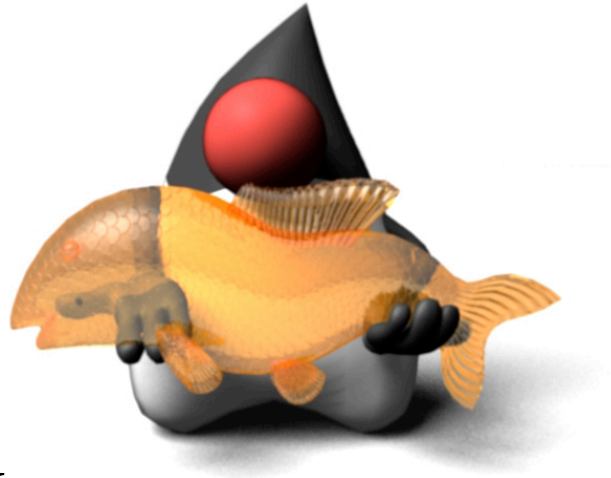
"Session-Fassade"

Fazit

Fazit

- **Vorteile**
  - \+ endlich wieder gute Objektorientierung mit POJOs möglich
  - \+ einfache Programmierung durch Annotationen
  - \+ sinnvolle Defaults spezifiziert , z.B. für EJB-Transaktionssteuerung
  - \+ JPA: gute Version 1.0, durch gute Implementierungen (Hibernate, TopLink) praxiserprobt; auch in Desktop-Anwendungen nutzbar
  - \+ mit dem Glassfish Application Server steht eine sehr gute, auch produktiv einsetzbare Referenzimplementierung zur Verfügung

- **Nachteile**
  - – für EJB 3.0 derzeit noch Application Server /EJBContainer notwendig
  - – derzeit müssen noch Local-Interfaces zur Optimierung verwendet werden
  - – JPA 1.0 ist noch unvollständig (Optimierung, Filterung)



Vielen Dank

für die Aufmerksamkeit

Thomas Much

thomas@muchsoft.com

www.muchsoft.com

Anhang:

JMS + Message Driven Beans
Timer
Security
Web-Services

JMS
+
Message Driven Beans (MDB)

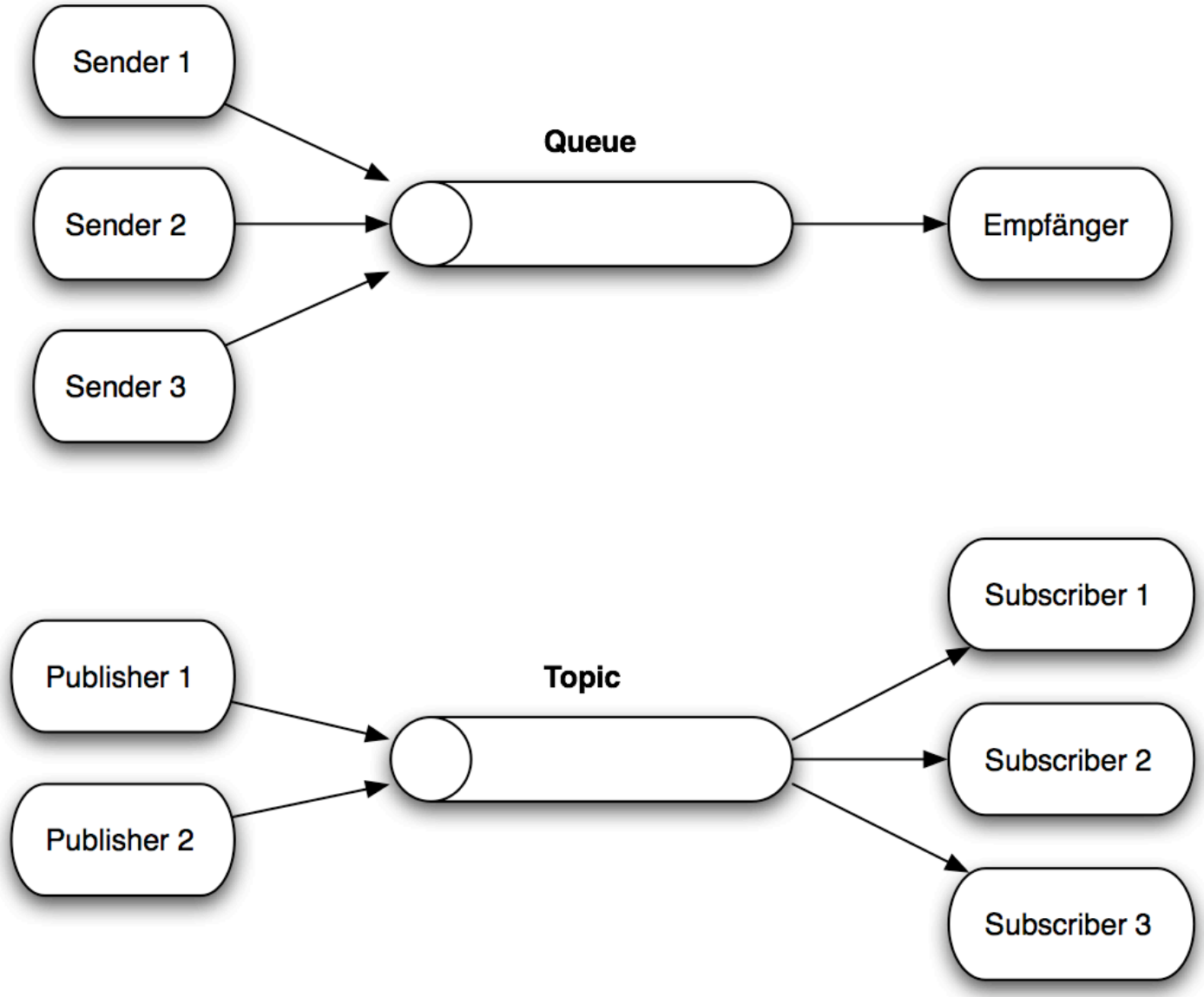
JMS

- *Java Message Service*
- Programmierschnittstelle für Message-Dienste

- Kommunikationsarten:
  - Point-to-Point mit Sendern und einem Empfänger
    ⇨ `Queue`
  - Publish-and-Subscribe als "Schwarzes Brett"
    ⇨ `Topic`

- Bestandteil der Application Server
- oder separater Download von *http://java.sun.com/products/jms/*

JMS: Queues und Topics



Message Driven Beans (MDB) (1)

```
@MessageDriven(
    ...
)
public class DruckerBean implements MessageListener {

    public void onMessage(Message msg) {

        try {
            ObjectMessage objmsg = (ObjectMessage) msg;

            Date datum = (Date) objmsg.getObject();

            System.out.println("Message angekommen: " + datum);

        } catch (JMSException e) {
            e.printStackTrace();
        }

    }
}
```

Message Driven Beans (MDB) (2)

```
@MessageDriven(
    activationConfig = {
        @ActivationConfigProperty(
            propertyName = "destinationType",
            propertyValue = "javax.jms.Queue"),
        @ActivationConfigProperty(
            propertyName = "destinationName",
            propertyValue = "jms/DruckerQueue"),
        @ActivationConfigProperty(
            propertyName = "connectionFactoryJndiName",
            propertyValue = "jms/ConnectionFactory"),
        @ActivationConfigProperty(
            propertyName = "subscriptionDurability",
            propertyValue = "NonDurable")
   },
   mappedName="jms/DruckerQueue"
)
public class DruckerBean implements MessageListener {
   ...
}
```

Timer

EJB Timer Service (1)

- zeitgesteuerte Methodenaufrufe
  - zu einem bestimmten Zeitpunkt oder nach Ablauf einer bestimmten Zeit
  - einmalig oder wiederholt

- Timer sind zustandslos
  ⇨ können also nur in Stateless Session Beans und MDBs genutzt werden

\+ fester Bestandteil der EJB3-Spezifikation (portabel!)

\- nicht für Echtzeitanwendungen gedacht

- nützlich, aber nicht so frei konfigurierbar wie externe Zeitsteuerungen

EJB Timer Service (2)

```
@MessageDriven
public class DruckerBean implements MessageListener {

    @Resource private TimerService timerService;

    public void onMessage(Message msg) {
        ...
        timerService.createTimer( 30*1000, 30*1000, "Info-Obj." );
        System.out.println( timerService.getTimers() );
    }


    @SuppressWarnings("unused")
    @Timeout
    private void zeitgesteuert(Timer timer) {
        System.out.println( "Timer-Aufruf: " + timer.getInfo() );
        timer.cancel();
    }

}
```

Security

Security

- Zwei grundlegende Funktionen:
  - Authentication
    - ⇨ Wer bin ich?
  - Authorization
    - ⇨ Was darf ich?

- Realisiert mit *Benutzern*, *Gruppen* und *Rollen*

- Java EE Security basiert auf dem *Java Authentication and Authorization Service* (JAAS)

- ein gültig authentifizierter Benutzer erhält ein *Principal*-Objekt, dem die Rollen des Benutzers zugeordnet sind

Zugriffs-Konfiguration in web.xml

web.xml

```
<login-config>
    <auth-method>BASIC</auth-method>
    <realm-name>KundenverwaltungsRealm</realm-name>
</login-config>

...

<security-constraint>
    <web-resource-collection>
        <web-resource-name>
            Kundenverwaltung Administration
        </web-resource-name>
        <url-pattern>/admin/*</url-pattern>
    <web-resource-collection>
    <auth-constraint>
        <role-name>ADMIN</role-name>
    <auth-constraint>
</security-constraint>
```

Deklarativer Zugriffsschutz

```
@DeclareRoles("KUNDE","ADMIN","GF")
@Stateless
public class BestellungBean implements Bestellung {

    @RolesAllowed("KUNDE","ADMIN")
    public void bestellungAbschicken() { ... }

    @RunAs("ADMIN")
    @RolesAllowed("GF")
    public List<String> getStatistik() { ... }

    @PermitAll
    public String getBestellStatus() { ... }

    @DenyAll
    public void preisHalbieren() { ... }
}
```

Programmatischer Zugriffsschutz

```
@Stateless
public class BestellungBean implements Bestellung {

    @Resource SessionContext ctx;


    public void bestellungAbschicken() {
        if ( !ctx.isCallerInRole("KUNDE") ||
             !ctx.getCallerPrincipal().getName()
                     .equals(besteller.getName()) ) {

            throw new SecurityException( "Kein Zugriff" );
        }
        ...
    }

}
```

Web-Services

Web-Services

- *Service Orientied Architecture* (SOA):
  - Architektur für lose gekoppelte Systeme, die über Dienste kommunizieren
  - Web-Services sind eine mögliche Realisierung der Dienste

- **SOAP** (*Simple Object Access Protocol*)
  - Format/Aufbau der ausgetauschten Nachrichten
- **WSDL** (*Web Services Description Language*)
  - Beschreibung des angebotenen Dienstes
- **UDDI** (*Universal Description, Discovery and Integration*)
  - "Gelbe Seiten" für Dienste

⇨ im Folgenden werden nur EJB als Web-Services beschrieben

Stateless Session Bean als Web-Service (1)

```
@WebService(serviceName="Drucker",name="DruckWS")
@SOAPBinding(style=SOAPBinding.Style.RPC)
@Stateless
public class DruckServiceBean implements DruckService,
                                         DruckServiceLocal {

    @WebMethod
    @WebResult(name="preisInCent")
    public int drucke( @WebParam(name="drucktext") String txt) {
        System.out.println( "Drucke \"" + txt + "\" ..." );
        return (txt.length() + 1) / 2;
    }

    @WebMethod(exclude=true)
    public void initDrucker() { ... }

    @WebMethod
    @Oneway
    public void loggeDruckerstatus(String aufrufer) { ... }

}
```

Stateless Session Bean als Web-Service (2)

```
@Remote
public interface DruckService {

    public void initDrucker();
    public int drucke(String txt);
}
```

```
@Local
public interface DruckServiceLocal extends DruckService { }
```

- basiert auf JAX-WS 2.0 (Java API for XML-Based Web Services)
- kann deklarative Transaktionen nutzen
- benötigt normalerweise kein APT (Annotation Processing Tool)
- nach Deployment kann WSDL-Dokument abgefragt werden:
  http://localhost:8080/Drucker/DruckWS?wsdl

WS-Zugriffsklassen generieren

build.xml

```
<property name="wsimport"
          location="${J2EE_HOME}/bin/wsimport.bat"/>

...

<target name="ws-generate" depends="j2ee-check" description="">
    <exec executable="${wsimport}" failonerror="true">
        <arg line="-d wssrc" />
        <arg line="-keep" />
        <arg line="-p com.muchsoft.ws.gen" />
        <arg line="http://localhost:8080/Drucker/DruckWS?wsdl" />
    </exec>
</target>
```

⇨ im Eclipse-Projekt Source-Folder ***wssrc*** anlegen!

WS-Client

```
public class WSClient {

    @WebServiceRef(
        wsdlLocation="http://localhost:8080/Drucker/DruckWS?wsdl")
    private static Drucker drucker;


    public static void main(String[] args) {

        DruckWS druckws = drucker.getDruckWSPort();

        String text = "Hallo WebService-Welt!";

        int cent = druckws.drucke( text );

        System.out.println("Drucken kostete " + cent + " Cent");
    }

}
```