

Einführung in die Java / Jakarta Enterprise Edition

Kapitel 5 – Das Programmiermodell

Wiederholung: Die Java Enterprise Edition

Beispiele & Folien

Übungen

Jakarta EE Platform

Jakarta EE Web Profile

Expression Language

Server Pages

Authorization

Authentication

CDI

CDI Lite

Activation

Concurrency

WebSocket

JSON Binding

Batch

Persistence

Bean Validation

Annotations

Connectors

Faces

Debugging Support

Interceptors

Mail

Security

Enterprise Beans Lite

Restful Web Services

Messaging

Servlet

Managed Beans

Json Processing

Enterprise Beans

Standard Tag Libraries

Transactions

Dependency Injection

Jakarta EE Core Profile

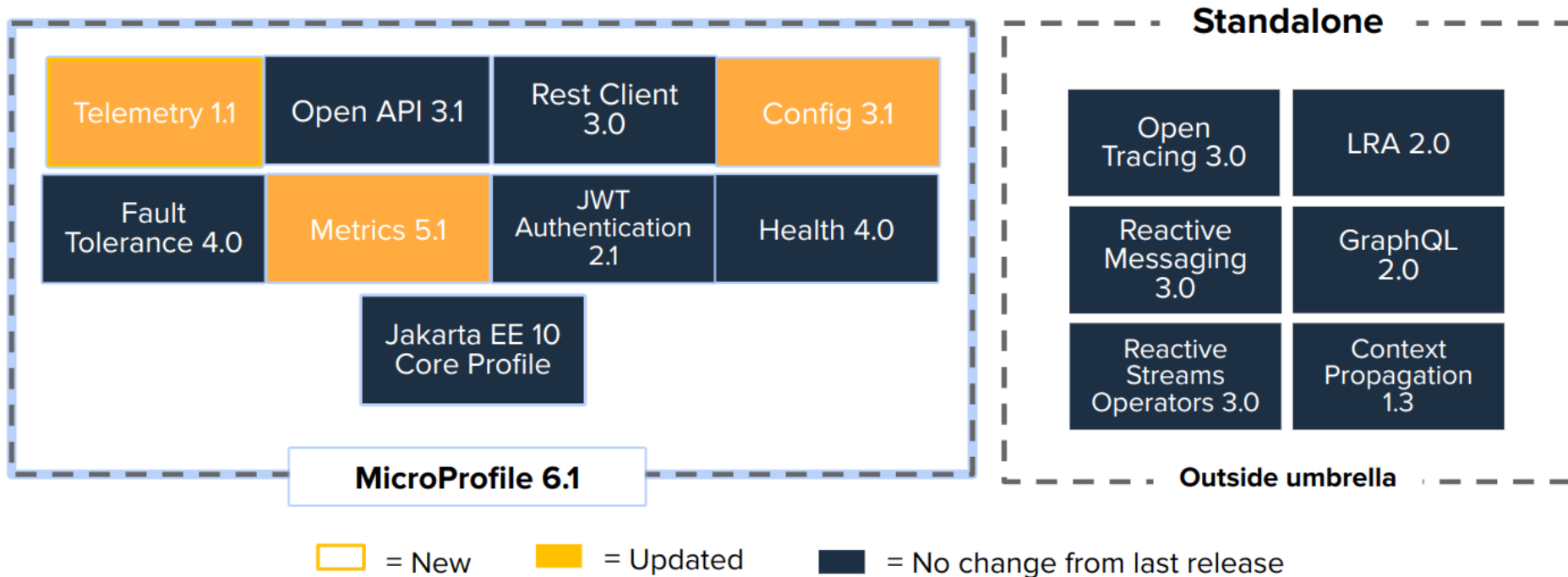
COPYRIGHT (C) 2023, ECLIPSE FOUNDATION. | THIS WORK IS LICENSED UNDER A CREATIVE COMMONS ATTRIBUTION 4.0 INTERNATIONAL LICENSE (CC BY 4.0)

5.1

BASIS

Basis: Microprofile

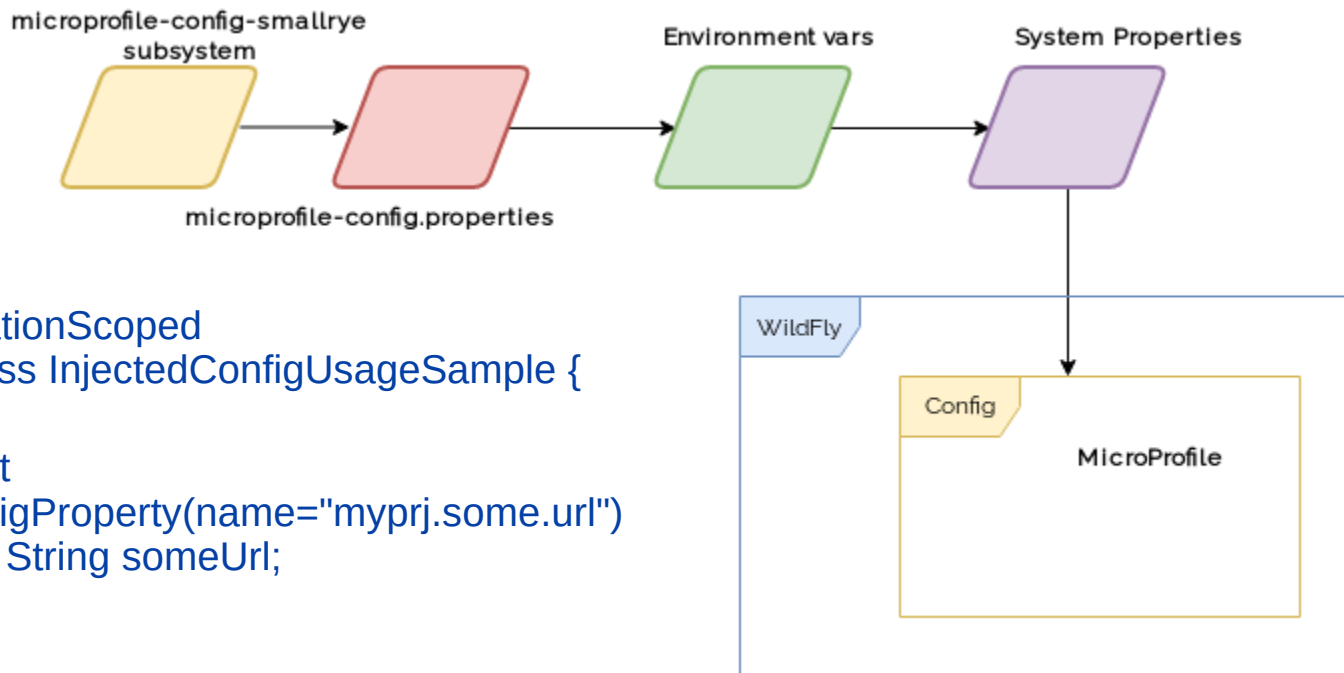
- Microprofile verschiedene kleine stellt Werkzeuge und Helferlein für Jakarta EE 10 bereit
 - Telemetry, Metrics, and Config, OpenApi, Rest Client



Quelle: <https://docs.google.com/presentation/d/1A3Hbr-O7QFepUP5M0X2hKfqFiO1PZgnBWWWhMyovu9JU/edit#slide=id.p8>

Basis: Microprofile Config

- Stellt eine Möglichkeit dar, auf einfache Art und Weise Konfigurationen in die Applikation hineinzugeben, die man später beispielsweise über Umgebungsvariablen ändern möchte



```
@ApplicationScoped
public class InjectedConfigUsageSample {

    @Inject
    @ConfigProperty(name="myprj.some.url")
    private String someUrl;
```

Basis: Microprofile Config - Übung

- Ziel
 - Konfigurationsparameter in eine Applikation hineingeben
- Aufgabe
 - Lege unter dem Verzeichnis resources/META-INF eine property-Datei namens microprofile-config.properties an
 - Definiere eine property, z.B. default-name=Mustermann
 - In der Klasse HelloWorldResource
 - Füge die oben gesetzte Config-Property hinzu
 - Lass dir den default-name statt des Strings „world“ ausgeben, falls kein Namen in der Methode übergeben wurde
 - Teste das Ergebnis mit dem Browser

Basis: Servlet (Web Profile)

- Jakarta Servlets sind eine einfache Art, HTTP-Anfragen als Java-Objekte entgegenzunehmen, ein Java-Objekt als HTTP-Antwort anzubieten und den gesamten Lebenszyklus um sie herum zu verwalten.

```
5
6 @WebServlet("/importtrainings")
7 public class TrainingsImportServlet extends HttpServlet {
8
9 }
10
```

clone()	Object
destroy()	void
doDelete(HttpServletRequest req, HttpServletResponse res)	
doGet(HttpServletRequest req, HttpServletResponse res)	
doHead(HttpServletRequest req, HttpServletResponse res)	
doOptions(HttpServletRequest req, HttpServletResponse res)	
doPost(HttpServletRequest req, HttpServletResponse res)	
doPut(HttpServletRequest req, HttpServletResponse res)	
doTrace(HttpServletRequest req, HttpServletResponse res)	
equals(Object obj)	boolean
finalize()	void
getInitParameter(String name)	String

Basis: Servlets (Web Profile)

- Begriffe
 - Session
 - Timeout
 - Session Replizierung
- Aufruf:
 - http
- Verwendung
 - Als Controller für Web Anwendungen
- Einsatz
 - Als technische Komponente für Web Anwendungen unabdingbar

Basis: Servlet - Übung

- Ziel
 - Web Servlet erstellen

- Aufgabe
 - Wir wollen eine Website mit Trainingsangeboten erstellen.
 - Dazu möchten wir einen Trainingskatalog (csv-Datei) einlesen.

- Schritte
 - Überschreiben Sie die doPost()-Methode in der Klasse TrainingsImportServlet.
 - Geben Sie die eingelesenen Daten im log/auf der Server-Konsole aus. Wir werden sie später verarbeiten.

Packaging von Web-Komponenten

- Alle benötigten Klassen müssen in einem Java-Archiv spezieller Struktur abgelegt werden
 - .war-Dateien
- Spezielles Verzeichnis: WEB-INF
 - Im classes-Verzeichnis sind alle nichtgepackten Klasse
 - lib-Verzeichnis für benötigte Bibliotheken
 - Web-Deskriptor web.xml
- Zusätzlich an beliebiger Stelle weitere Ressourcen
 - Statische Elemente wie HTML-Seiten, Bilder...
 - Dynamische JSP-Seiten

Eine simple Web-Anwendung

- Eine Start-Seite
- Ein Servlet mit der Verarbeitungslogik
- Eine JavaBean, die das Ergebnis hält
- Eine JSP zur Darstellung der Ergebnisse
- Deskriptor
 - web.xml

Für komplexere Anwendungen ist dieser Ansatz nicht mehr zeitgemäß!

- Wie ist eine Webanwendung aufgebaut Welche Datei(en) und Ordner finden man auf jeden Fall?
- Für welche Anwendungsfälle können Servlets verwendet werden?
- Wann braucht man die Microprofile-config?

5.2

CORE PROFILE

Core: Json Binding & Processing

- Jakarta Json Binding stellt ein Default-Mapper für die Serialisierung und Deserialisierung von Java-Objekten bereit
- <https://javaee.github.io/jsonb-spec/getting-started.html>

```
Jsonb jsonb = JsonbBuilder.create();
```

```
Person person = new Person();
```

```
person.name = "Fred";
```

```
Jsonb jsonb = JsonbBuilder.create();
```

```
// serialize to JSON
```

```
String result = jsonb.toJson(person);
```

```
// deserialize from JSON
```

```
person = jsonb.fromJson("{\"name\":\"joe\"}", Person.class);
```

Core: Json Binding & Processing

■ Customized Mapping

- Es können vom Attribut abweichende JSON-Bezeichnungen angegeben werden:

```
public class Person {  
    @JsonProperty("person-name")  
    private String name;  
    private String profession;  
}
```

- Daraus resultiert beispielsweise folgendes Json-Dokument:

```
{  
    "person-name": "Jason Bourne",  
    "profession": "Super Agent"  
}
```

Core: Json Binding & Processing - Übung

- Ziel
 - Umgang mit Json-Dateien üben.
- Aufgabe
 - Wir haben in der vorherigen Übung den Trainingsdatenkatalog als CSV Datei ausgelesen. Wir möchten nun diese Daten im JSON-Format ausgeben.
- Schritte
 - Erweitern Sie das TrainingsImportServlet um eine Methode, welche
 - die Werte aus der CSV-Datei in ein Java-Objekt mappt
 - das Java-Objekt als Json zurückgibt
 - Tauschen Sie getTextFromPart mit der neu geschriebenen Methode aus
 - Rufen Sie das Servlet über die Oberfläche auf uns schauen Sie sich das Resultat Ihrer Änderungen auf der Server-Konsole an
 - Ändern Sie die Json-Properties von Camel-Case in eine Notation mit Bindestrichen, z.B. durationInDays -> duration-in-days

Core: RESTful Webservices

- Representational State Transfer (REST)
- Übertragung von Darstellungen von Ressourcen über Request-Response
- Daten und Funktionen gelten als Ressourcen
- Der Zugriff erfolgt über Uniform Resource Identifiers (URIs)
- Methoden:
 - @GET (auflisten)
 - @PUT (erstellen)
 - @POST (aktualisieren)
 - @DELETE (löschen)

```
@GET
@Produces({ MediaType.APPLICATION_JSON })
public Hello hello(@QueryParam("name") String name) {

    if ((name == null) || name.trim().isEmpty()) {
        name = "world";
    }

    return new Hello(name);
}
```

Core: RESTful Webservice - Übung

- Ziel
 - Bearbeiten eines Objektes mittels RESTful Webservices zum Erstellen, Bearbeiten, Anzeigen und Löschen erstellt werden.
- Aufgabe
 - Wir möchten die importierten Trainings via REST-Schnittstelle bearbeiten können.
- Schritte:
 - Legen Sie einen RESTful Webservice "TrainingResource.java" an und ergänzen sie den Pfad unter dem der Service erreichbar sein soll.
 - Fügen Sie die Methodenrumpfe find(), findAll(), create(), update() und delete() hinzu
 - Ergänzen Sie die zugehörigen REST/HTTP-Methoden (GET, PUT, POST, DELETE)
 - Geben Sie der findAll()-Methode einen Rückgabewert und rufen Sie sie über den Browser auf
 - Wir ergänzen zusammen die Übergabeparameter, Producer, Consumer

Core: Context & Dependency Injection

- CDI beans...
 - sind Quellen von kontextbezogenen Objekten von einer Bean
 - werden über den CDI-Container gemanaged
 - können in andere Objekte im gleichen Kontext injiziert werden
 - können Metadaten welche ihren Lebenszyklus definieren mitbringen
- Eine Bean kann folgende Attribute besitzen:
 - einen nichtleeren Satz von "bean types"
 - einen nichtleeren Satz von "Qualifiers"
 - einen Scope
 - optional einen Name
 - mehrere Interceptor Bindings
 - eine Implementierung

Core: Context & Dependency Injection

- Source: <https://jakarta.ee/specifications/cdi/4.0/jakarta-cdi-spec-4.0.html>
- Definition über Annotations oder beans.xml (nur CDI Full)
- Scopes (= lifecycle context)
 - CDI Lite:
 - RequestScoped
 - ApplicationScoped
 - Dependent
 - CDI Full:
 - SessionScoped
 - ConversationScoped

Core: Context & Dependency Injection

Bean Definieren:

```
...  
@ApplicationScoped  
public class ApplicationScopedCounter implements Serializable {  
    private int counter = 0;  
    public void count() {  
        counter++;  
    }  
    public int getCounter() {  
        return counter;  
    }  
}
```

Bean aufrufen:

```
...  
@Path("hello")  
public class HelloWorldResource {  
  
    @Inject  
    ApplicationScopedCounter applicationScopedCounter;  
    ...  
}
```

Core: Context & Dependency Injection - Übung

- Ziel

- Grundlagen und Scopes von CDI Beans verstehen.

- Aufgabe

- Fügen Sie in das vorhandene Beispiel drei CDI Beans mit jeweils einen Scope Request, Session und Application, die eine Zählfunktion enthalten.
 - Geben Sie das Ergebnis der Funktionen aller drei Beans via Rest Service aus.
 - Rufen Sie den RESTful Service im Browser mehrmals auf - in der existierenden und einer neuen Session. Erklären Sie das Ergebnis.

- Hinweis

- Orientieren Sie sich an der Bean `ApplicationScopedCounter` und dem Rest Service `HelloWorldResource.java`.

Core: Context & Dependency Injection

■ Qualifiers

- Qualifier Types um zwischen verschiedenen Implementierungen zu unterscheiden:

`@CreditCard`

```
class CreditCardPaymentProcessor  
    implements PaymentProcessor {  
    ...  
}
```

`@Cash`

```
class CashPaymentProcessor  
    implements PaymentProcessor {  
    ...  
}
```

- Beans mit Qualifiers aufrufen:

```
@Inject @CreditCard PaymentProcessor paymentProcessor;
```

```
@Inject @Cash PaymentProcessor paymentProcessor;
```

Core: Context & Dependency Injection

■ Qualifiers

- Qualifier jakarta.inject.Named - @Named benutzen:

```
public class Car {  
    @Inject @Named("driver") Seat driverSeat;  
    @Inject @Named("passenger") Seat passengerSeat;  
    ...  
}
```

■ Interceptors CDI Lite:

- @AroundInvoke, @PostConstruct, @PreDestroy, @AroundConstruct
- Aufrufreihenfolge festlegen mit @Priority ist für @PostConstruct und @PreDestroy möglich

Wissenscheck

- Mit welchen Methoden kann man beim Json-Binding Instanzen von Java Objekten in Json-Strings umwandeln et vice versa?
- Mit welcher Annotation kann man der Json-Property einen anderen Namen geben?
- Welche HTTP-Methoden werden durch RESTful Webservices unterstützt und wie sollte man sie verwenden?
- Welche Annotation muss man hinzufügen, damit eine Klasse zum RESTful Webservice wird?
- Welche CDI-Scopes gibt es in CDI Lite und welche werden durch CDI Full zusätzlich unterstützt?

5.3

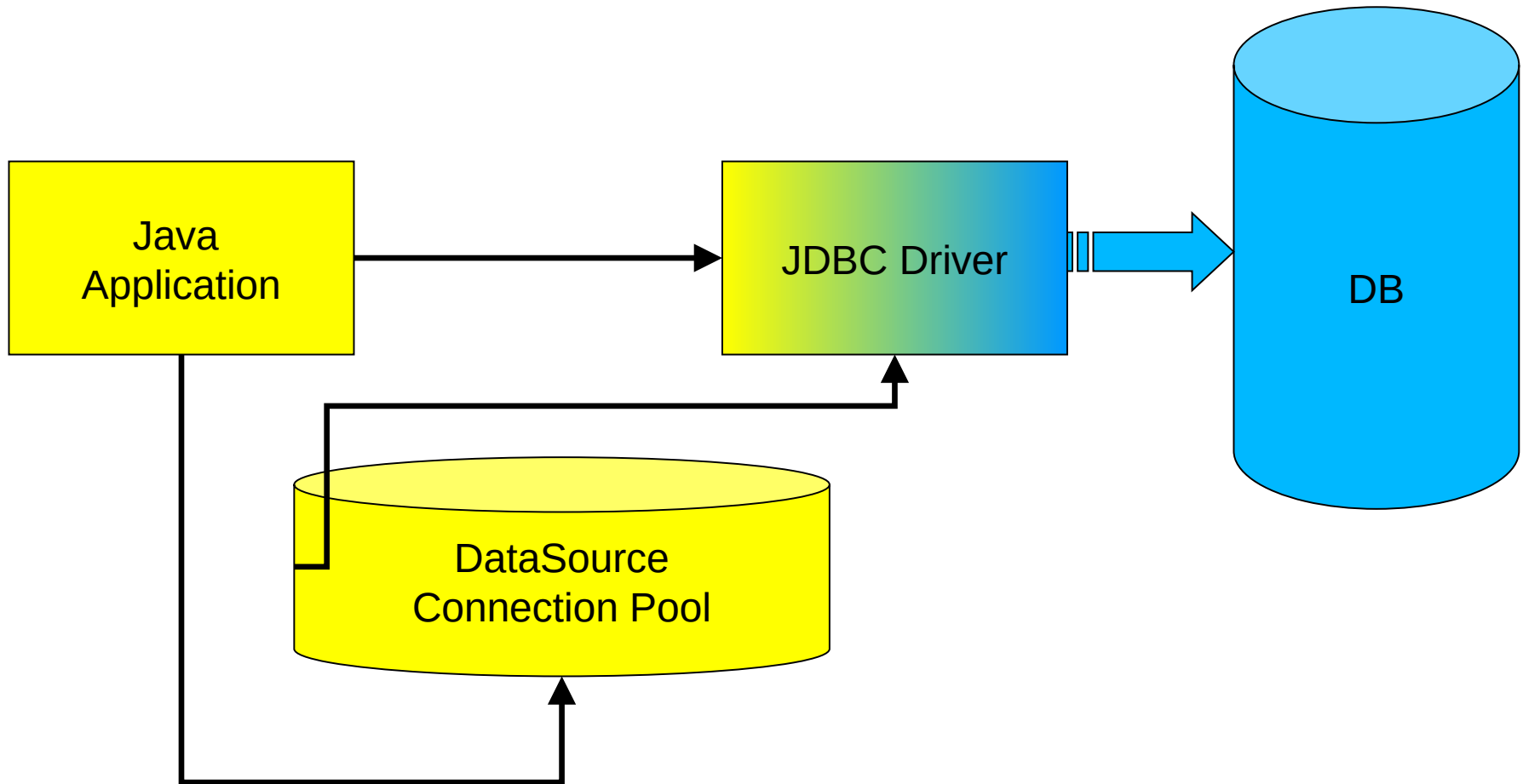
WEB PROFILE

5.3 Web Profile

DATENZUGRIFF & JPA

5.3 Web Profile

NATIVER SQL-ZUGRIFF



- Eine konfigurierte DataSource enthält einen Connection Pool zur Datenbank
- Die Java-Anwendung
 - holt sich von der DataSource eine Connection
 - erzeugt die benötigten Statements
 - setzt diese ab
 - wertet die Ergebnisse aus
 - ResultSet
 - SQLException
 - schließt die Connection
 - diese wird in den Pool zurückgegeben, nicht real geschlossen!

7.2

O/R-MAPPER

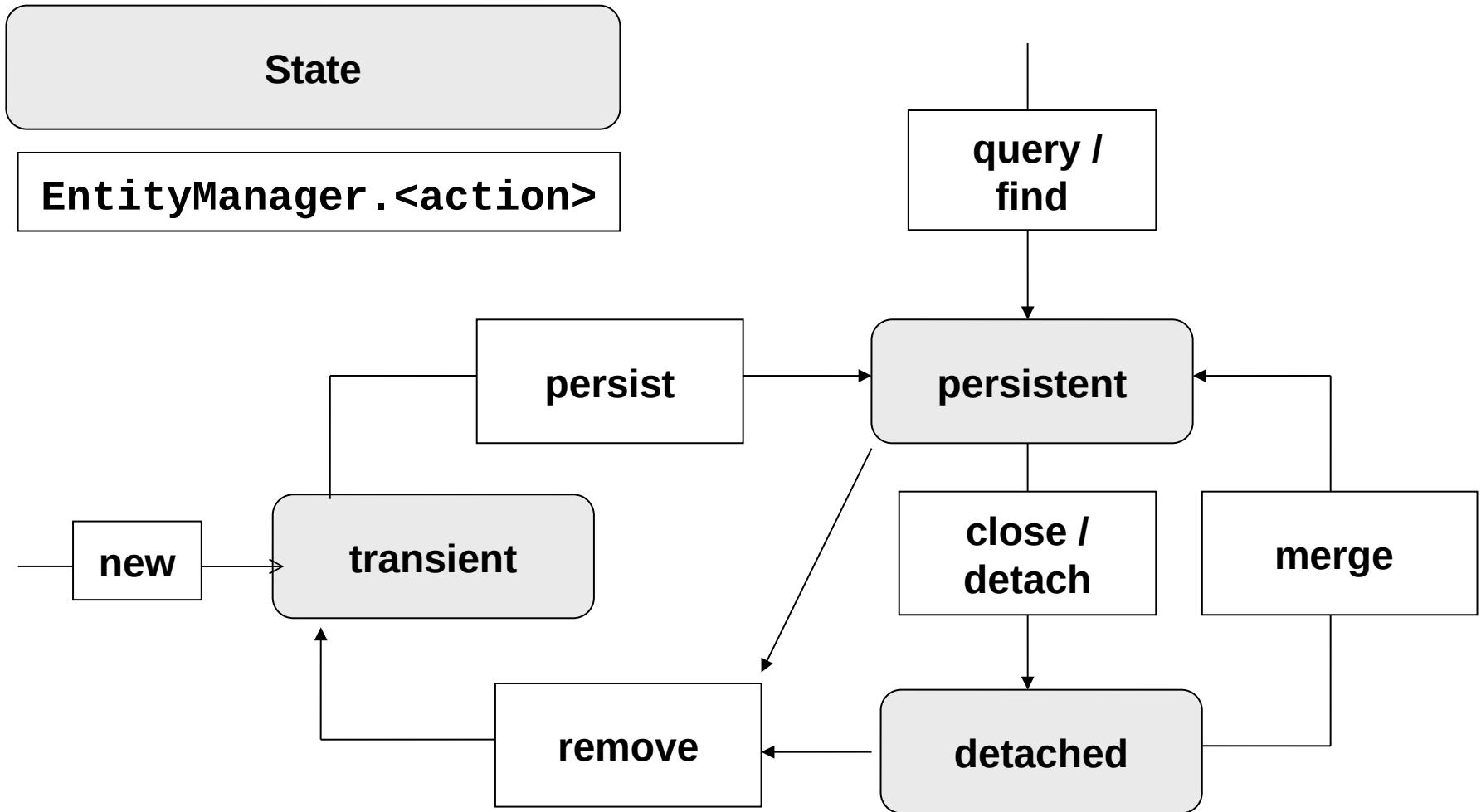
Aufgaben eines O/R-Mappers

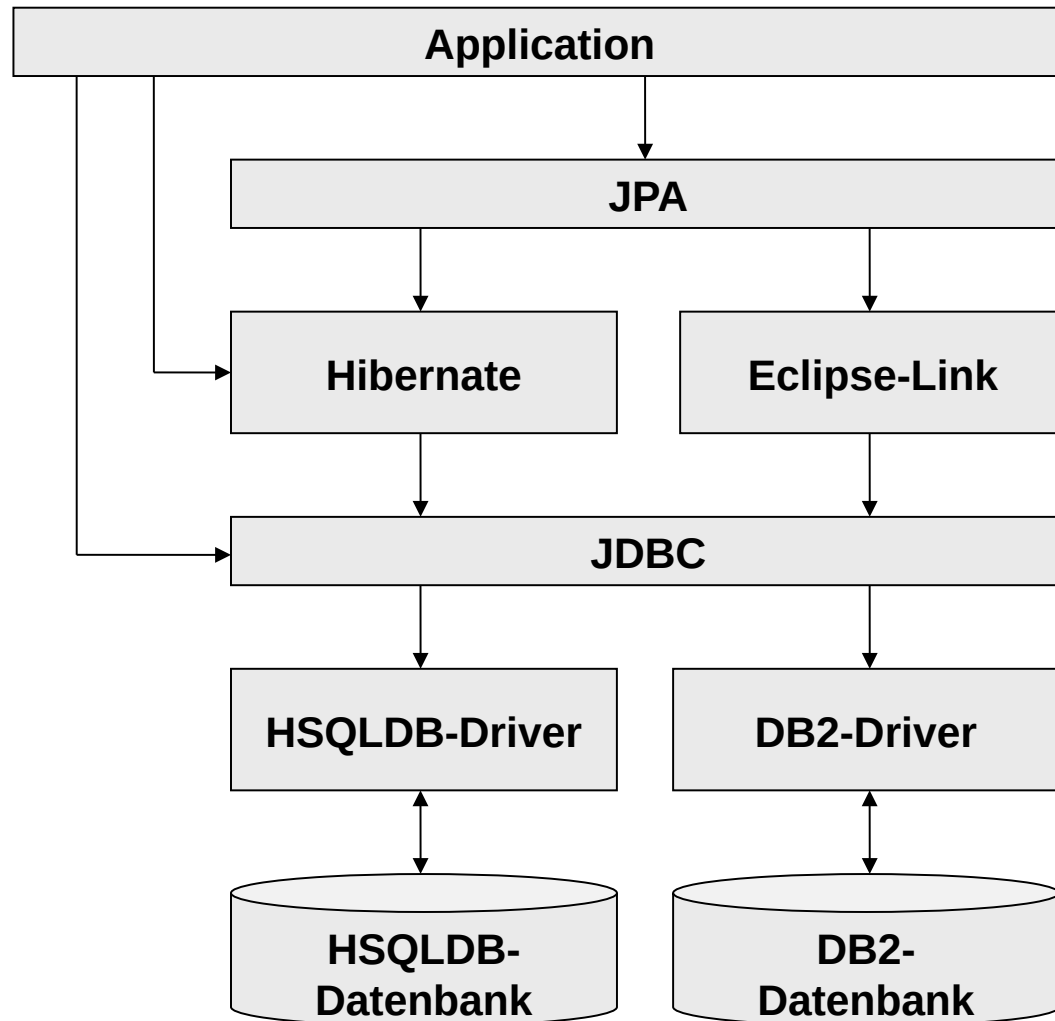
- Abbildung von Tabellenzeilen auf Objekte und umgekehrt
- Abbildung von Primär-/Fremdschlüssel-Beziehungen auf Referenzen
- Generierung von SQL-Statements
- Objektorientierte Abfragesprache
- Identität von Objekten
- Natives SQL

- Die Objekte der Objektorientierung leben im Hauptspeicher und sind somit "flüchtig".
 - Die "Objekte" von Datenbanken sind persistent.
- Die Objektorientierung kennt "intelligente" Objekte – Objekte, die ihren Zustand kapseln. Die Klassen dieser Objekte enthalten Methoden, mittels derer der Zustand der Objekte abfragbar und manipulierbar ist.
 - Relationale Datenbanken dagegen enthalten nur "dumme" Daten.
- Relationale Datenbanken kennen andere Typen als objektorientierte Sprachen.
 - Die Datenbank kennt z.B. die Typen CHAR und VARCHAR; Java dagegen kennt den Typ String.
- In der objektorientierten Welt sind Objekte miteinander über Referenzen (also Pointer) verbunden.
 - In relationalen Datenbanken werden die "Objekte" über Fremdschlüssel-Beziehungen miteinander verbunden. Die Objektorientierung kennt aber keine Fremdschlüssel (und auch keine Primärschlüssel).

- Die Objektorientierung fokussiert individuelle Objekte; zwischen diesen Objekten kann navigiert werden. (Natürlich lassen sich solche individuellen Objekte auch in Collections zusammenfassen.)
 - Die typische Zugriffsweise von Datenbanken ist dagegen der SELECT in Verbindung mit dem JOIN – eine Zugriffsweise, die grundsätzlich Mengen von Zeilen liefert. Im Gegensatz zur Objektorientierung operiert die Datenbank also mengenorientiert.
- Die Objektorientierung kennt das Vererbungskonzept.
 - Relationale Datenbanken dagegen kennen mit wenigen Ausnahmen keine Vererbung.
- Relationale Datenbanken beruhen wesentlich auf dem Konzept der referenziellen Integrität;
 - in der Objektorientierung ist dieses Konzept von Natur aus unbekannt.

- Die Entity ist eine normale Java-Klasse, die spezielle Annotations besitzt
 - Welches Attribut wird in welcher Tabellenspalte abgelegt
 - Optimierung der Datenbankzugriffe mit Lazy Loading, Fetching...
 - Alternativ kann auch eine XML-basierte externen Konfigurationsdatei benutzt werden
- Das Speichern, suchen etc. übernimmt der EntityManager
 - Das Objekt selber ist nicht per se persistent, sondern ist in verschiedenen Zuständen vorhanden:
 - Transient (keine Entsprechung zu einem Datensatz)
 - Persistent (entspricht einem Datensatz, der Entity Manager muss die Objekt-Identität garantieren)
 - Detached (entspricht einem Datensatz, ein detached Objekt ist ein unabhängiger Snapshot des Datenbestandes)
 - Die Zustandswechsel werden vom EntityManager gesteuert





Web: Datenzugriff mit JPA

- Java Persistence API, Offizielles Tutorial
- objekt-/relationales Mapping für die Verwaltung relationaler Daten. Dabei sind Entitäten (@Entity) die Basisklassen:

@Entity

```
public class Coffee implements Serializable {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;
```

- durch den EntityManager ist ein automatisches Transaktionshandling und Management des Datenbank ConnectionPools möglich

@PersistenceContext

```
private EntityManager em;
```

Web: Datenzugriff mit JPA

- besteht aus:
 - Jakarta-Persistence
 - Query Language JPQL
 - Jakarta Persistence Criteria API
 - Objekt-/relationale Mapping-Metadaten
- wird auch von anderen Frameworks wie SpringBoot genutzt

Web: Datenzugriff mit JPA

- um Datenbankverbindungen für JPA zu definieren, benötigt man unter resources/META-INF eine persistence.xml:

```
<persistence version="2.1"
  xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="
    http://xmlns.jcp.org/xml/ns/persistence
    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="testDB">
    <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
    <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
    <properties>
      <property
        name="jakarta.persistence.schema-generation.database.action"
        value="drop-and-create" />
      <property name="hibernate.show_sql" value="true" />
      <property name="hibernate.hbm2ddl.auto" value="create-drop"/>
    </properties>
  </persistence-unit>
</persistence>
```


Web: Datenzugriff mit JPA - Übung

- Ziel:
 - Praktischer Umgang mit Entitäten.
- Aufgabe:
 - Bauen Sie Ihre Klasse Training so um, dass es ebenso als Entity genutzt werden kann.
- Schritte:
 - Fügen Sie @Entity hinzu
 - Fügen Sie ein Attribut id hinzu, dessen Wert beim Anlegen in die Datenbank generiert wird
 - Starten Sie die Server + Anwendung und schauen Sie, ob die Entität auf der Management-Console des Servers angezeigt wird.
 - Überprüfen Sie ebenfalls, ob die über die persistence.xml angelegte Datenbank existiert.

- Wozu dient ein O/R-Mapper?
- Welche Objekte nutzt man für das Mapping, welche Annotation muss man der Klasse hinzufügen?
- Was ist ein EntityManager, welche States verwaltet er?

5.3 Web Profile

ENTERPRISE JAVABEANS ALS FACHOBJEKTE

Aufgaben einer Enterprise JavaBean

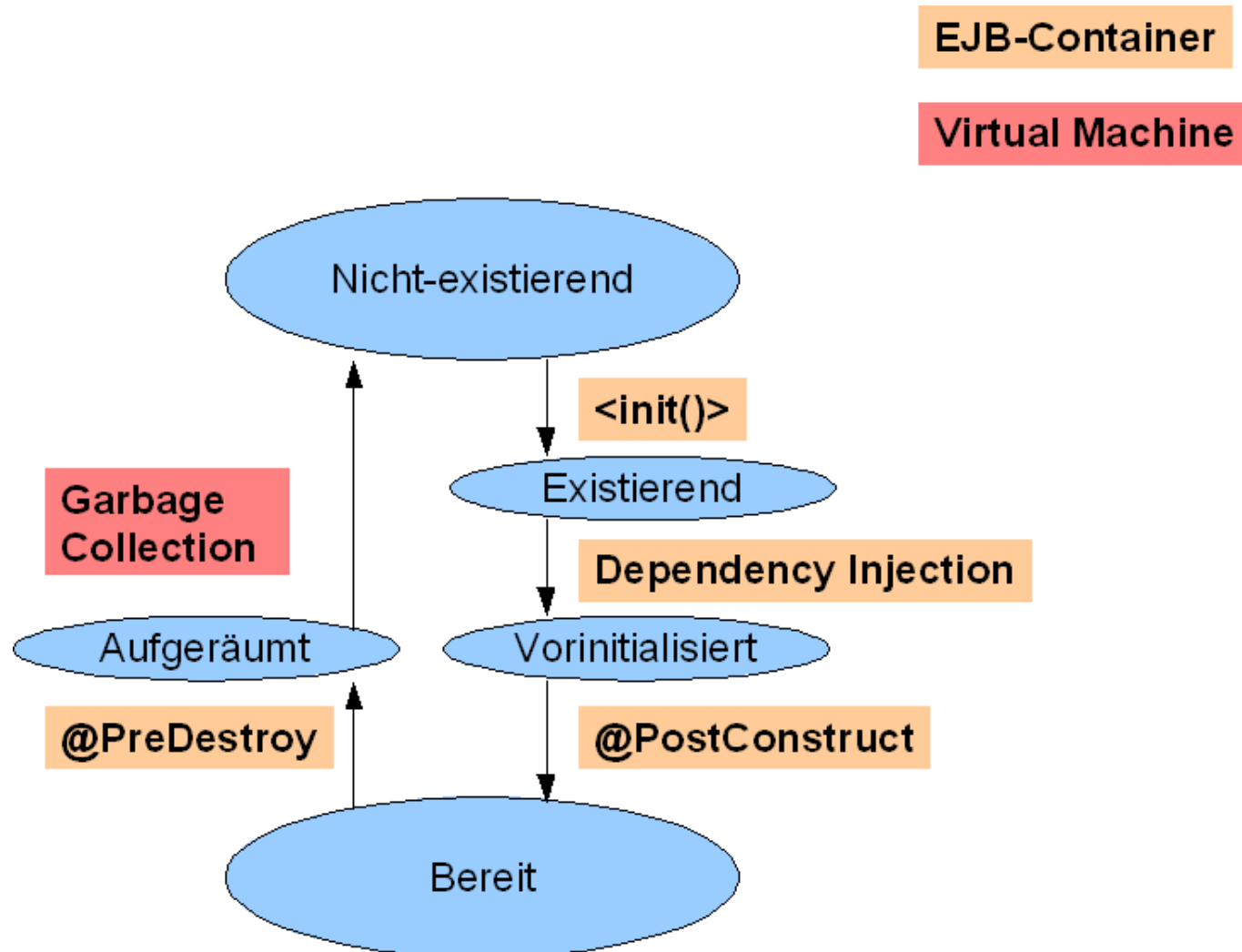
- Definieren das Transaktionsverhalten der Anwendung
- Definieren und Prüfen die Anwendungs-Rollen
- Rufen die eigentliche Geschäftslogik auf
- Können bei Bedarf an einen Inbound Connector gekoppelt und so über das Netzwerk aufgerufen werden
- Eine Stateful SessionBean kann eingesetzt werden, um bei Bedarf Client-Zustand im Server zu halten

Die Aufgaben von Enterprise JavaBeans überschneiden sich mit CDI. Eine Interoperabilität ist jedoch gewährleistet!

- Stateless SessionBeans
 - Gruppieren Funktionen
 - Können bei Bedarf über Java RMI oder SOAP angesprochen werden
 - Mehrere Instanzen werden in einem Pool verwaltet

- MessageDriven Beans
 - Sind Listener an einer JMS-Destination
 - Mehrere Instanzen werden in einem Pool verwaltet

Lebenszyklus einer Zustandslosen EJB



Stateless SessionBeans

- Begriffe
 - Instance Swapping
 - Instance Pooling
- Aufruf:
 - lokal
 - RMI
- Verwendung
 - Aufruf von Geschäftsprozessen, die alle notwendigen Informationen als Parameter bekommen
- Einsatz
 - Uneingeschränkt geeignet
 - Sofort Cluster-fähig und ausfallsicher

MessageDrivenBeans

- Begriffe
 - Instance Swapping
 - Instance Pooling
- Aufruf:
 - JMS
- Verwendung
 - Aufruf von Geschäftsprozessen, die alle notwendigen Informationen als Nachricht bekommen
 - Client erwartet (wenn überhaupt) Antwort über Callback
- Einsatz
 - Uneingeschränkt geeignet
 - Sofort Clusterfähig und Ausfallsicher

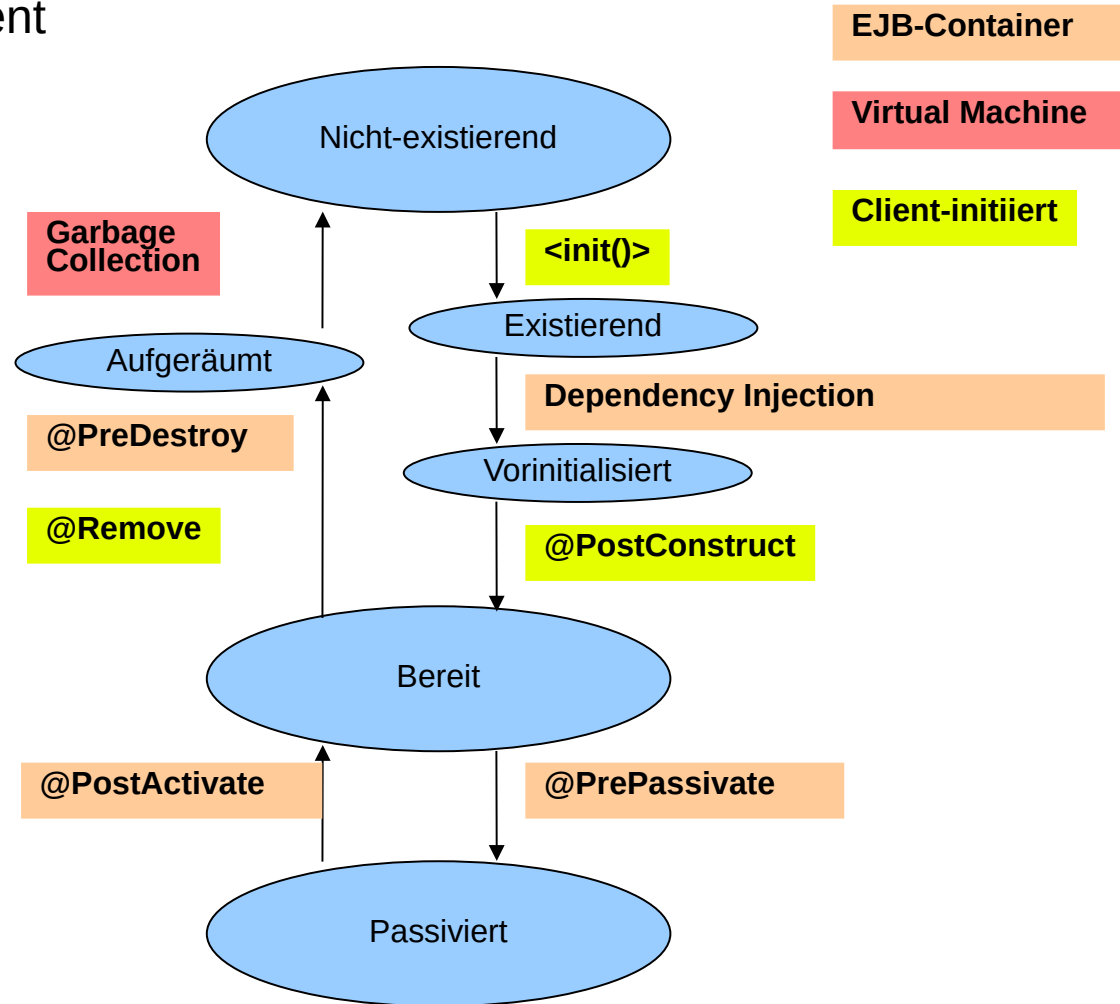
Zustandsbehaftete Enterprise JavaBeans

- Singleton Beans
 - Gruppieren Funktionen
 - Existieren exakt einmal pro Anwendung und Applikationsserver
 - Können von verschiedenen Clients gleichzeitig benutzt werden

- Stateful SessionBeans
 - Gruppieren über ein Interface Funktionen
 - Können bei Bedarf über Java RMI oder SOAP angesprochen werden
 - Haben einen Lebenszyklus, der vom Client gesteuert wird
 - Gleichzeitiger Zugriff mehrere Clients gleichzeitig ist nicht möglich

Lebenszyklus einer Stateful SessionBean

! Eine Instanz pro Client erforderlich



- Begriffe
 - Eine Instanz pro Anwendung
 - Potenziell gleichzeitiger Zugriff muss berücksichtigt werden
- Aufruf:
 - lokal
 - RMI
- Verwendung
 - Als globaler Zwischenspeicher (Daten-Cache) auf der Serverseite
- Einsatz
 - Weniger Overhead als Stateless SessionBeans
 - Multithreading muss beachtet werden

- Begriffe
 - Aktivierung und Passivierung
 - Timeout
 - Session Replizierung
- Aufruf:
 - RMI
 - lokale
- Verwendung
 - Als Zwischenspeicher (Daten-Cache) auf der Serverseite
- Einsatz
 - Eingeschränkt geeignet: Session-Problematik im Cluster
 - Relativ hoher Konfigurationsaufwand
 - Geringere Effizienz als Stateless SessionBeans: Eine Instanz pro Client erforderlich

Web: Enterprise Beans - Annotationen

- Stateful Session Beans: `@Stateful`
- Stateless Session Beans: `@Stateless`
- Singleton Session Beans: `@Singleton`
- Message Driven Beans: `@MessageDriven`

Web: Enterprise Beans – Business Interface

- Stateful / Stateless Enterprise Beans bestehen (historisch begründet) aus einem Business Interface und einer Implementierung.
- Das Business Interface kann separat erstellt werden oder ist implizit mit dem Implementieren der Bean vorhanden
- Arten:
 - @Remote: über Remote Method Invocation (RMI) erreichbar
 - @Local: im Code verwendbar, Default

Web: Enterprise Beans – Übung

- Ziel:
 - CRUD Operationen in einer Stateless Session Bean implementieren. Daten in der Datenbank speichern.
- Aufgabe
 - Die Trainingsdaten sollen in der Datenbank über eine Stateless Session gespeichert werden.
- Schritte:
 - Fügen Sie in der Klasse TrainingService Annotationen hinzu:
 - um den Service als Stateless Session Bean zu markieren
 - um den EntityManager den PersistenceContext zuzuordnen
 - Aktivieren sie die vorbereiteten Methoden im TrainingService, indem Sie die Kommentare entfernen
 - Welches Business Interface hat die Bean?
 - Ergänzen Sie im REST-Service TrainingResource die Stateless Session Bean und entfernen Sie auch hier die Kommentare.
 - Speichern Sie im TrainingsImportServlet die Entitäten. Nutzen Sie dazu die create-Methode des TrainingService.
 - Testen: Importieren Sie die CSV-Datei über die Oberfläche und lassen Sie sich das Ergebnis über REST anzeigen

- Welche Arten von Enterprise Beans gibt es?
- Was passiert, wenn ich in Stateless Session Bean Attribute definiere?
- Wer bestimmt den Lebenszyklus einer Session Bean?
- Welche Standard-Interceptoren gibt es?
- Was ist der Unterschied zwischen Session Beans und CDI Beans?

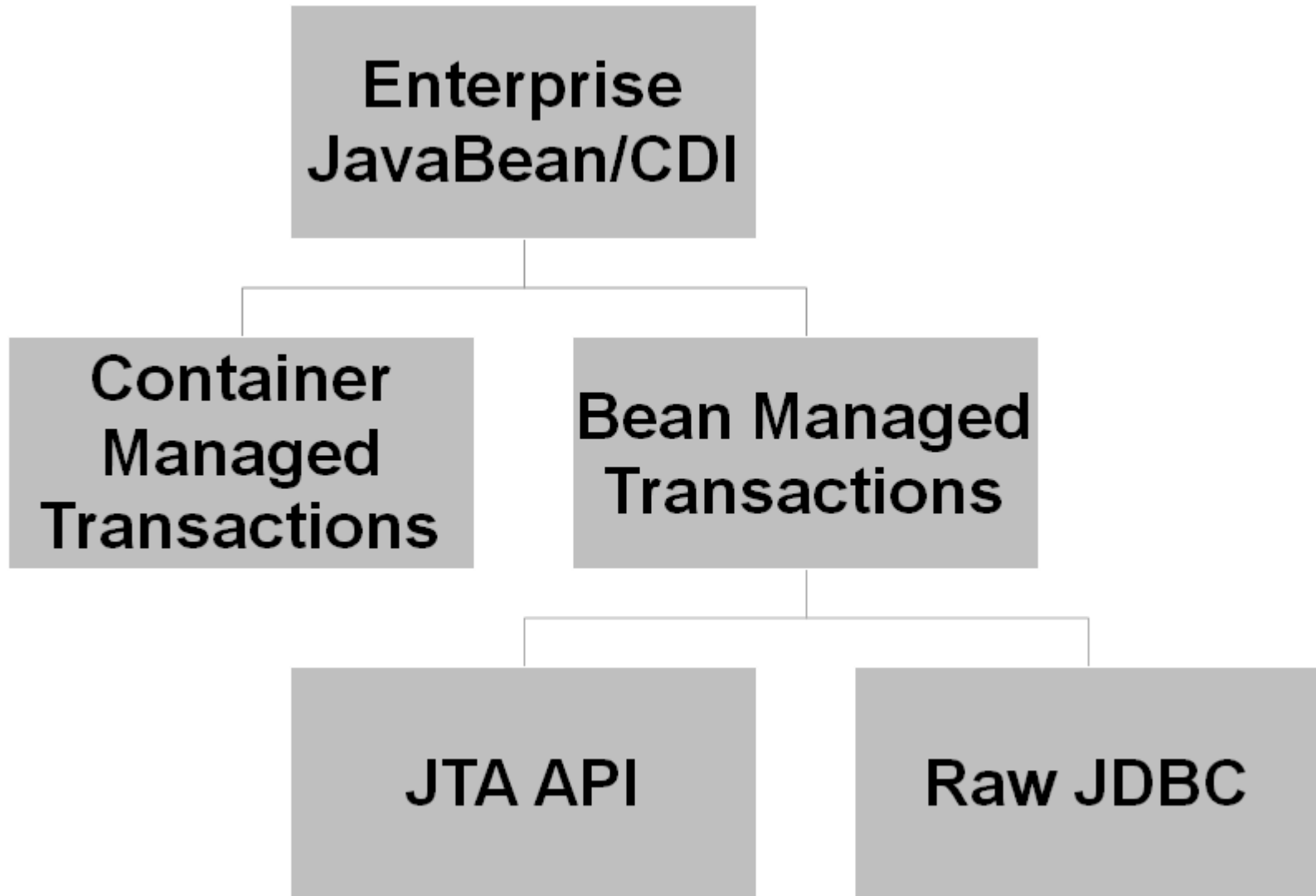
5.3 Web Profile

TRANSAKTIONSSTEUERUNG

Transaktionssteuerung und CDI

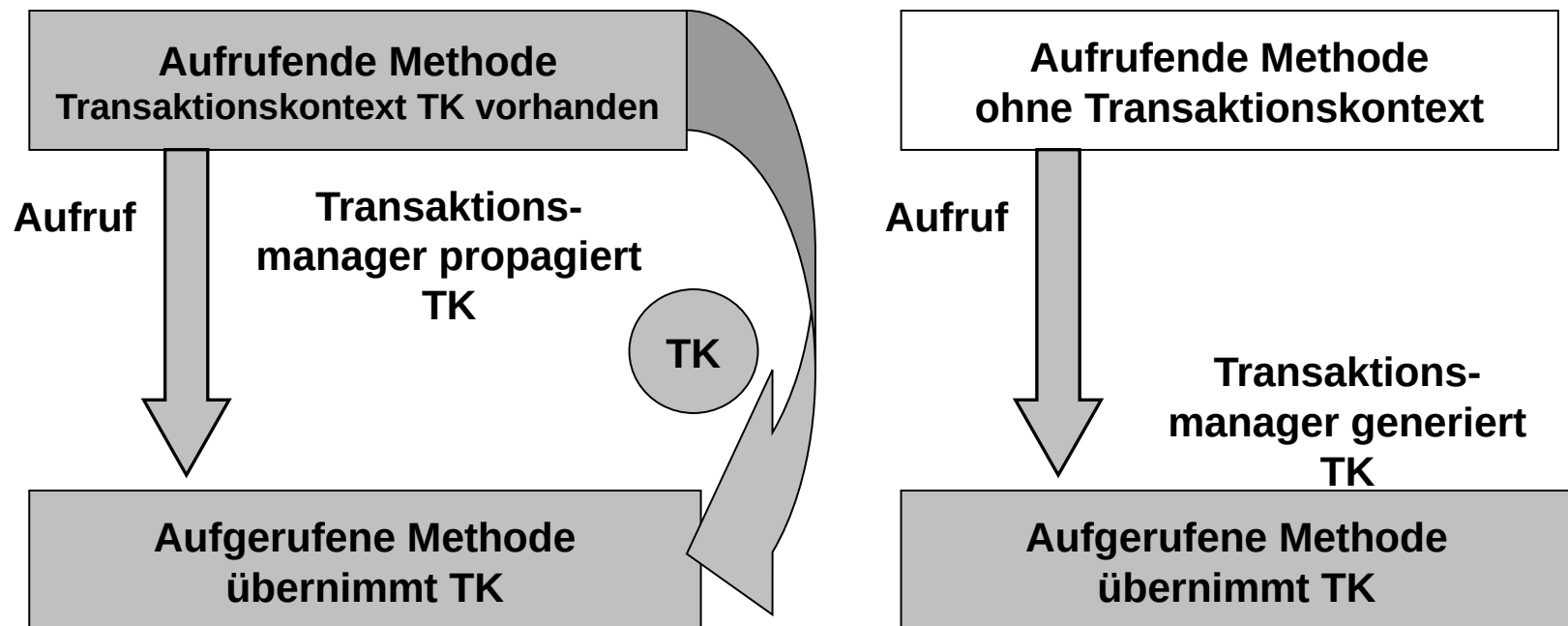
- Ab der JEE 7 unterstützen auch CDI Beans die deklarative Transaktionssteuerung
 - `javax.transaction.Transactional`
- Enterprise JavaBeans sind damit für die Realisierung transaktioneller Fachobjekte nicht mehr notwendig
- Die folgenden Erläuterungen sind für EJBs und CDI gültig

- Deklaratives Transaktions-Management
 - Container übernimmt die gesamte Transaktionsverwaltung
 - Definition eines Transaktions-Attributes auf Methodenebene
 - Die Propagierung der Transaktion wird vom Transaction Manager des Applikationsservers übernommen
- Die Transaktionssteuerung kann auch explizit vom Programmierer vorgenommen werden
 - Dazu wird der Bean vom Container eine `UserTransaction` übergeben
- Die Zuordnung der Transaktionsattribute zu einer Bean-Methode erfolgt durch Annotations
 - Auch eine externe Konfiguration über einen XML-Deskriptor ist möglich

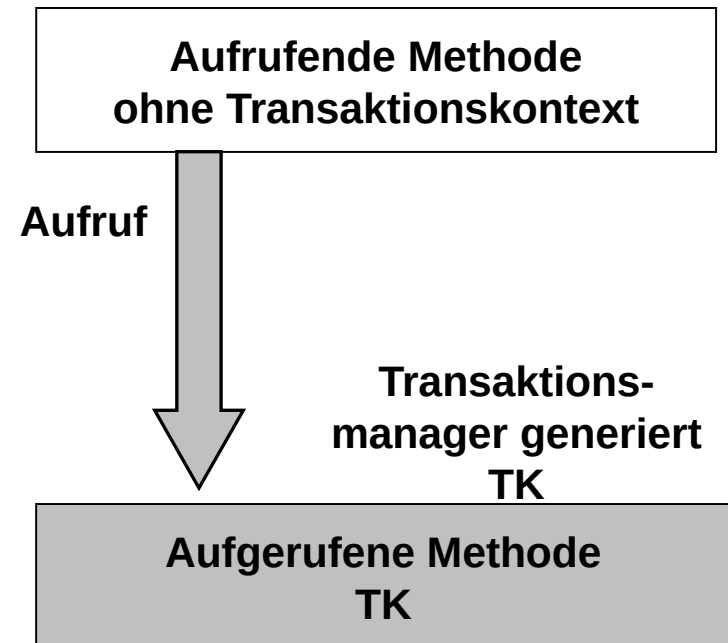
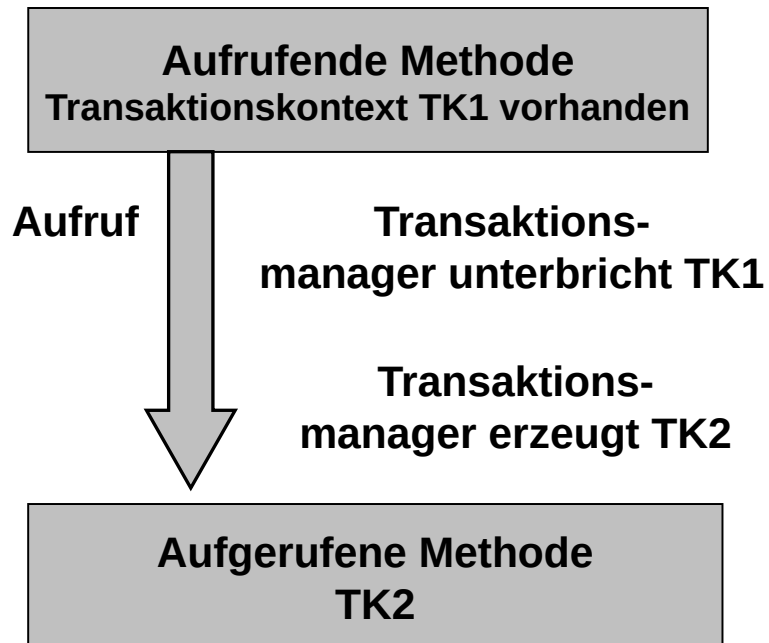


- Es existieren 6 Transaktions-Attribute
 - Attribut:
 - Not Supported
 - Supports
 - Required
 - Requires New
 - Mandatory
 - Never
- Transaktions-Attribute werden auf Methodenebene vergeben
 - Beim Aufruf einer Methode wird an Hand der Attribute eine neue Transaktion gestartet oder ein vorhandener Kontext übernommen

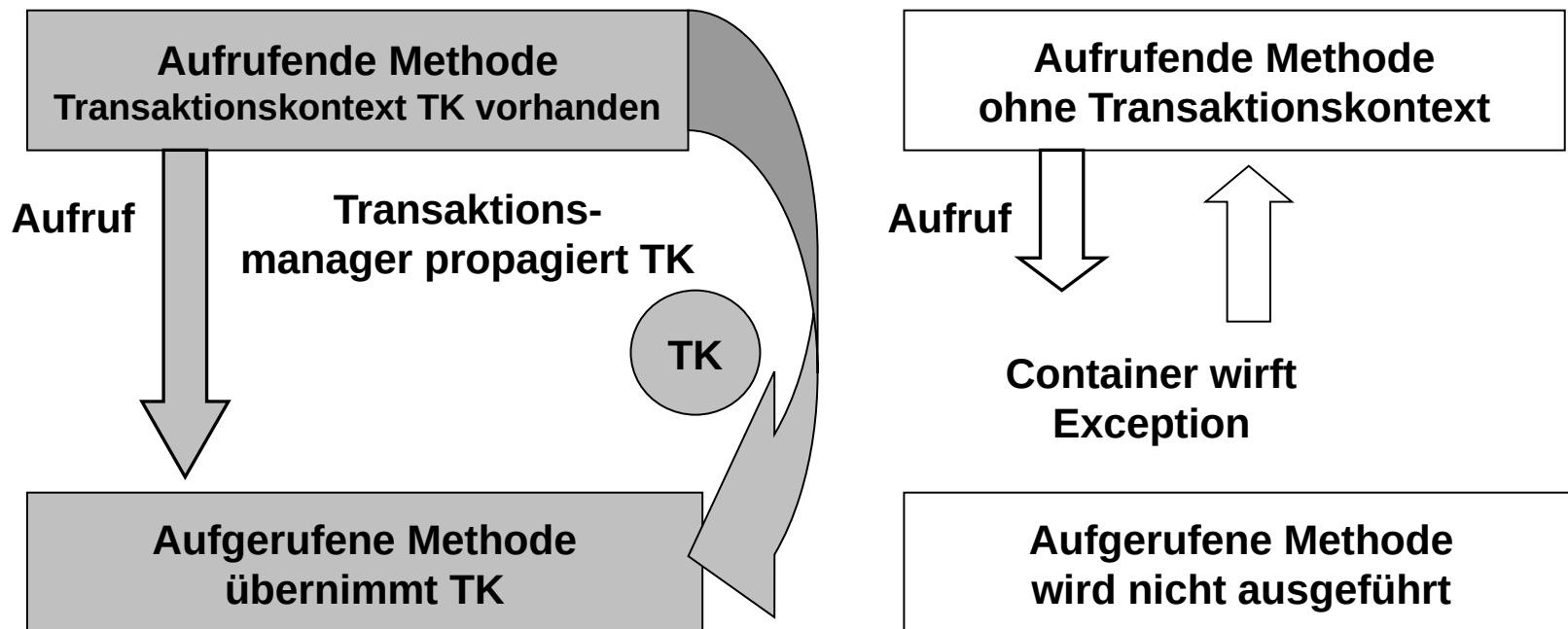
- Die aufgerufene Methode enthält garantiert einen Transaktionskontext



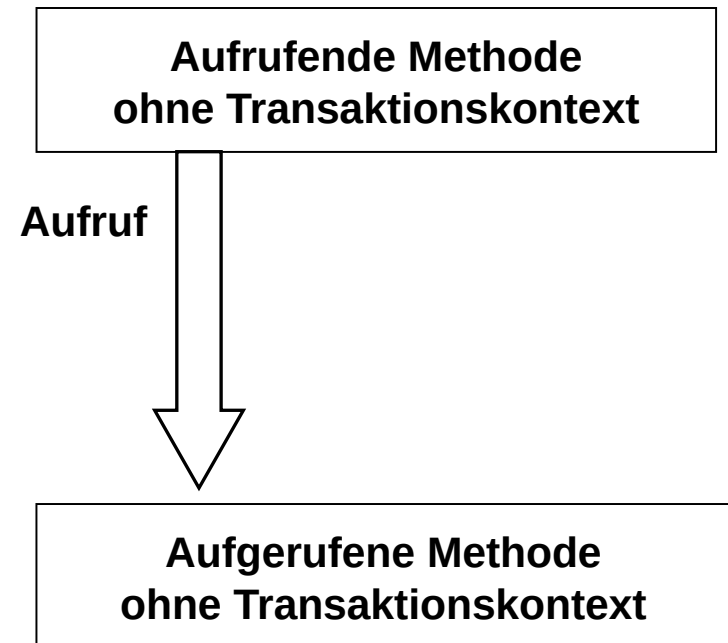
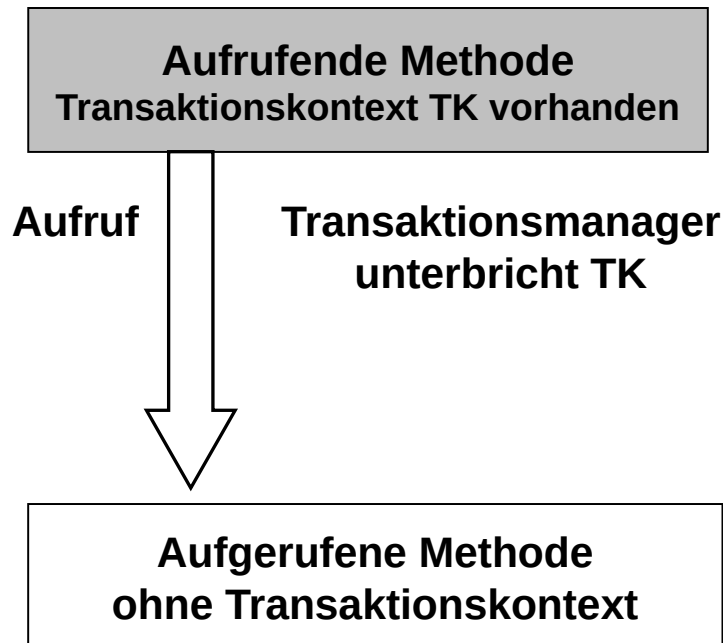
- Die aufgerufene Methode enthält garantiert einen Transaktionskontext



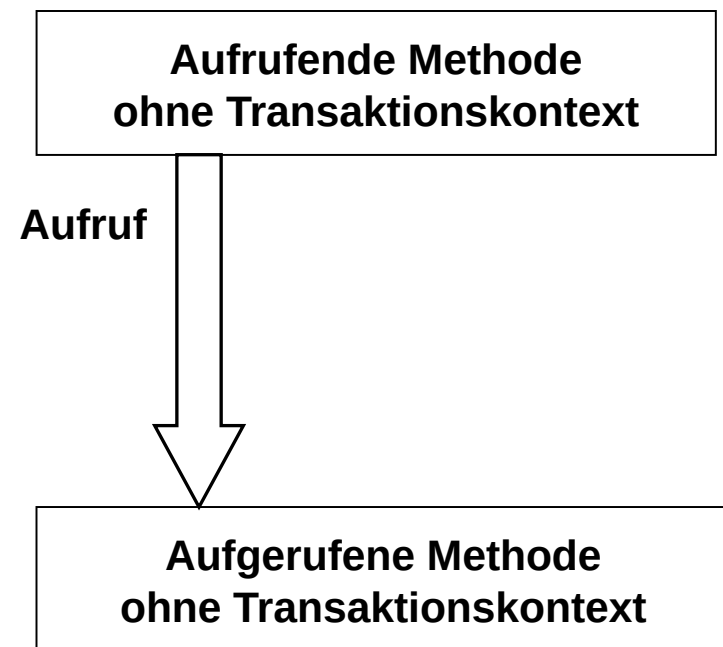
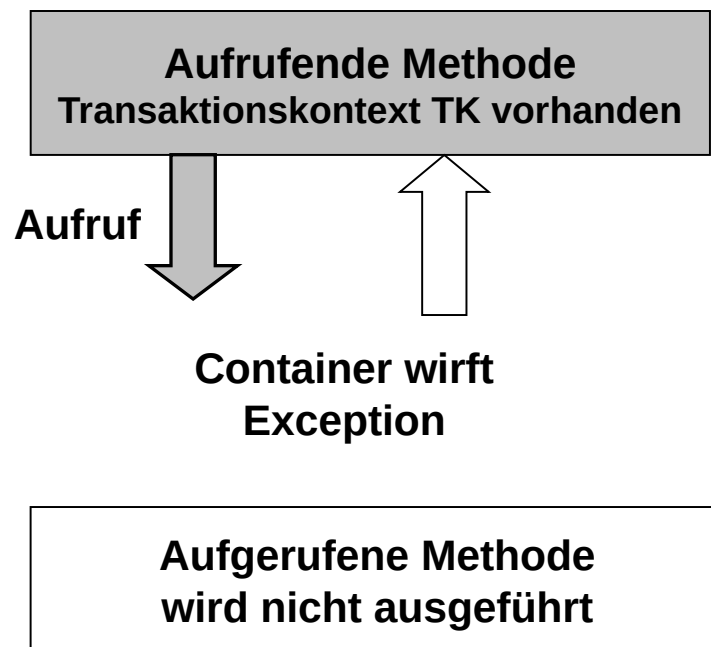
- Der Client muss einen Transaktionskontext besitzen
- Ansonsten: `TransactionRequiredException`



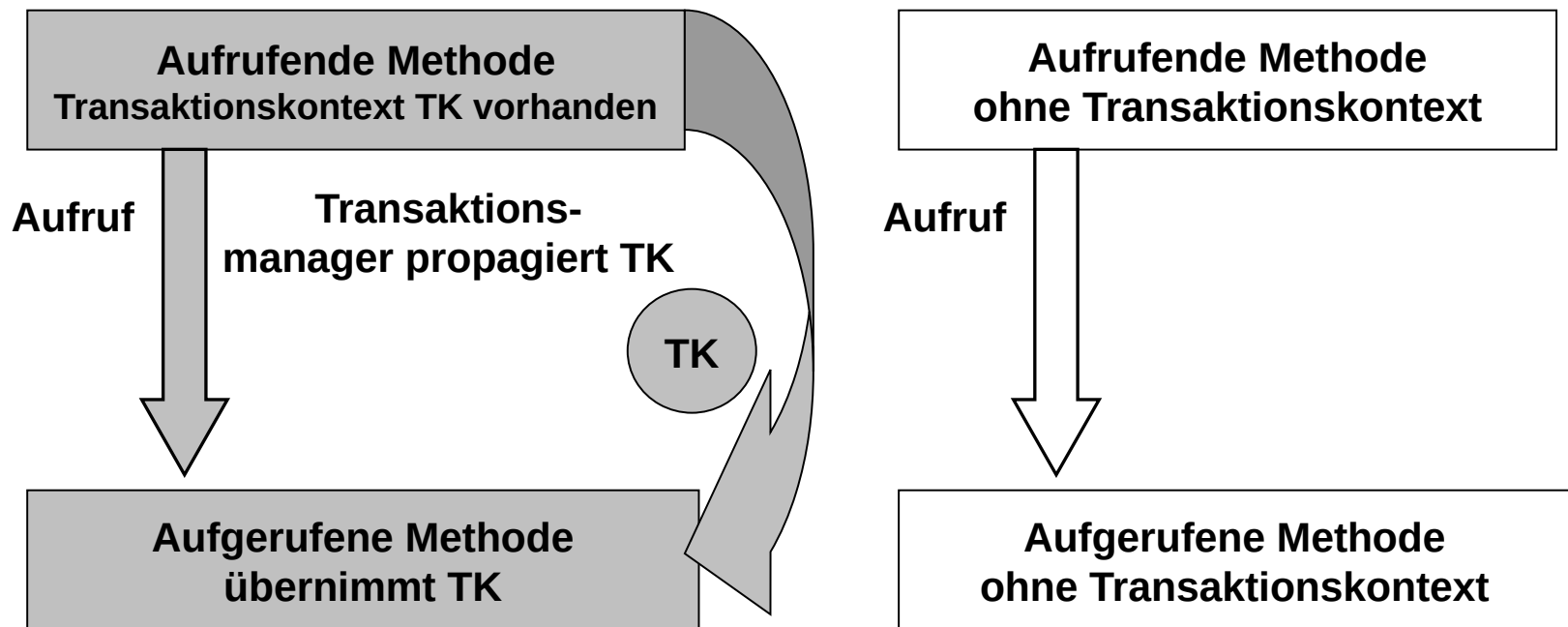
- Die aufgerufene Methode läuft ohne Transaktionskontext



- Der Client darf keinen Transaktionskontext besitzen
- Ansonsten: RemoteException



- Die aufgerufene Methode läuft mit dem Transaktionskontext der aufgerufenen Methode



Web: Jakarta Bean Validation

- Dient zur zur Validierung von Objekten, Objektmitgliedern, Methoden und Konstruktoren
- Die Validierung erfolgt durch Annotationen
- Die einzelnen Validierungen werfen einen Fehlercode zurück, der als Key für Übersetzungsdateien benutzt werden kann
- Typische Annotationen:
 - @NotNull, @NotEmpty, @NotBlank
 - @Max, @Min, @Size
 - @Pattern

Web: Jakarta Bean Validation - Übung

- Aufgabe:
 - Fügen Sie der der TrainingsEntity Validierungen hinzu
 - Prüfen Sie, was passiert, wenn sie über die CSV-Datei Trainings importieren, welche diesen Werten nicht entsprechen

5.3 Web Profile

JAVA SERVER FACES

- Historie
 - Beginn der Spezifikation im Jahr 2001 (JSR-127) (ASF, BEA, Borland, HP, IBM, Novell, Oracle, Sun)
 - März 2004: Final Release Version 1.0 für J2EE 1.4
 - Mai 2006: JSF Version 1.2 als Bestandteil von Java EE 5
 - Dez. 2009: JSF Version 2.0 als Bestandteil von Java EE 6
 - Zwischen Java EE 6 und 7 wurde ein "maintenance-release" JSF 2.1 freigegeben.
 - 2014: JSF Version 2.2 als Bestandteil der JEE 7

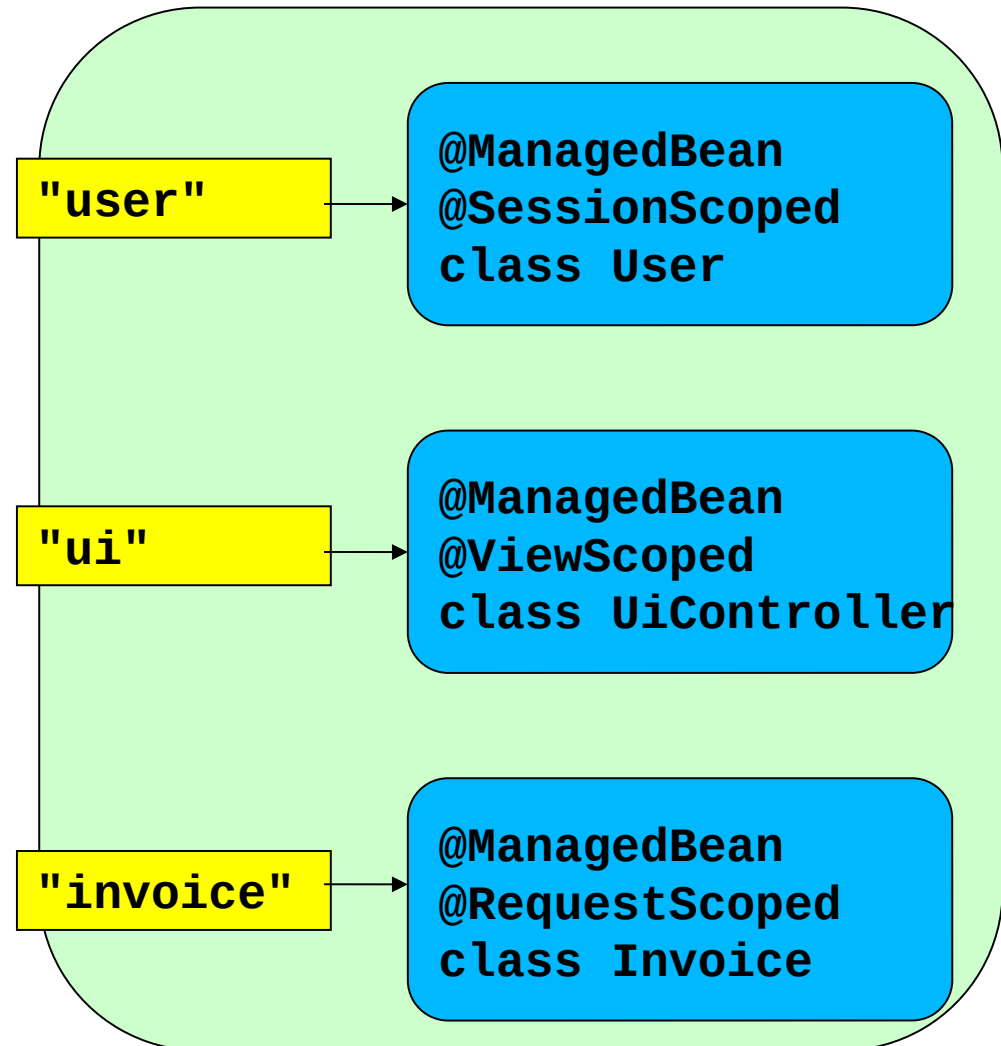
Implementierungen

- Die Hersteller von Applikationsservern und Komponentenbibliotheken verwenden mittlerweile bevorzugt die Referenzimplementierung Mojarra
 - Oracle
 - GlassFish
 - WebLogic
- Erweiterungen sind jedoch immer noch sinnvoll!
 - JBoss RichFaces 4
 - ICEFaces 2
 - PrimeFaces
 - Apache Tomahawk/Trinidad
 - ...

- Objekt-orientierter Programmier-Ansatz
 - „Managed Beans“ halten den Zustand der Web-Anwendung und definieren über Actions ihr Verhalten
- Feingranulare Scopes für die Verwaltung des Datenmodells der Web Anwendung
 - Damit vereinfachte Verwendung der Session
- Ausgefeilter Zyklus zur Verarbeitung eines Requests
 - Integration von Validierung und Konvertierung
 - Die Anwendungsprogramme benötigen das http-nahe Servlet-API nur noch in Ausnahmefällen
- Server-seitige UI-Komponenten mit Event-basiertes Programmiermodel

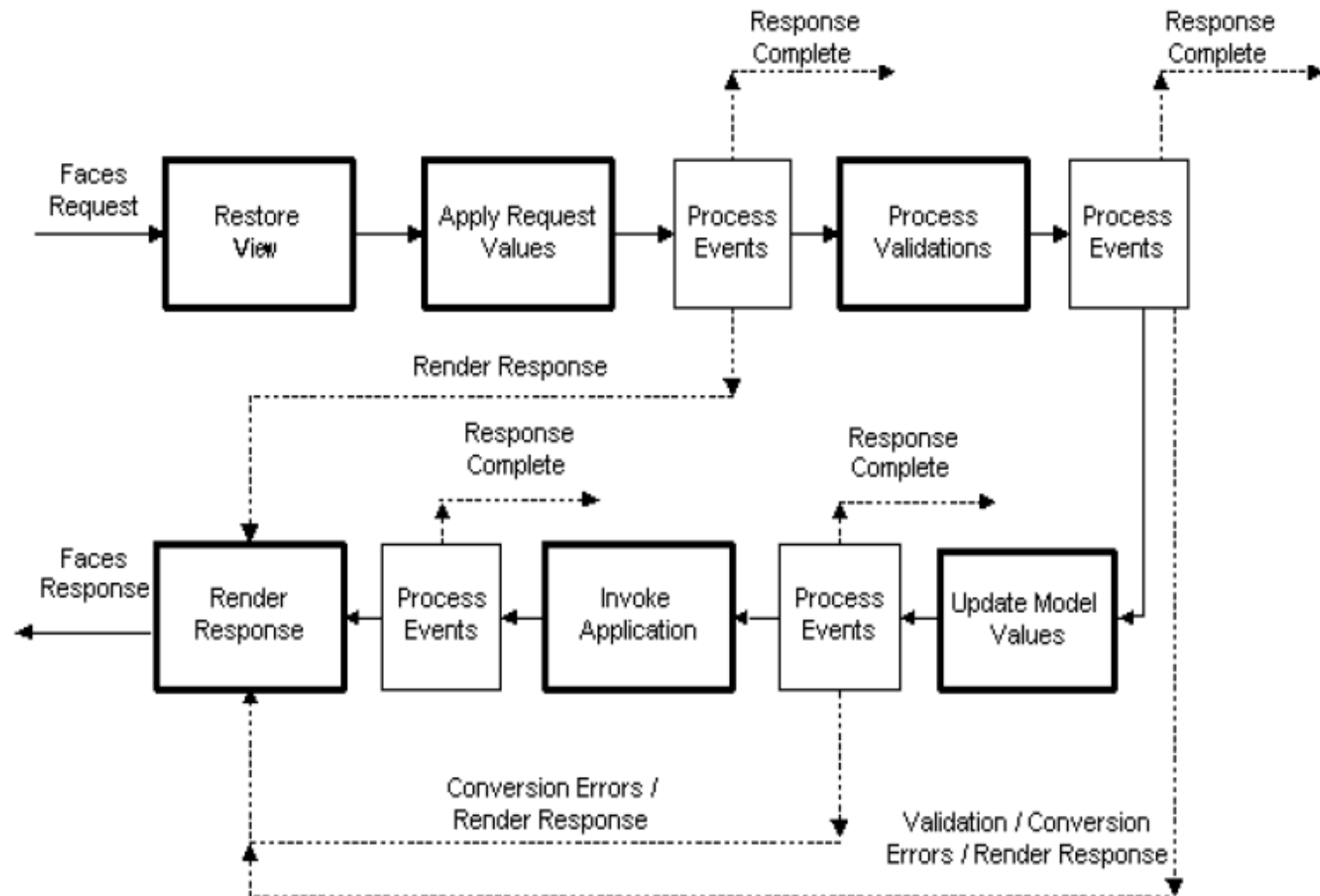
- Die Clients der allermeisten Web-Anwendungen legen Informationen auf dem Server ab
- Managed Beans halten diesen Zustand auf dem Server
- Jede Managed Bean wird durch einen eindeutigen Namen identifiziert
- Die Lebensdauer von Managed Beans wird durch die Angabe von Scopes definiert

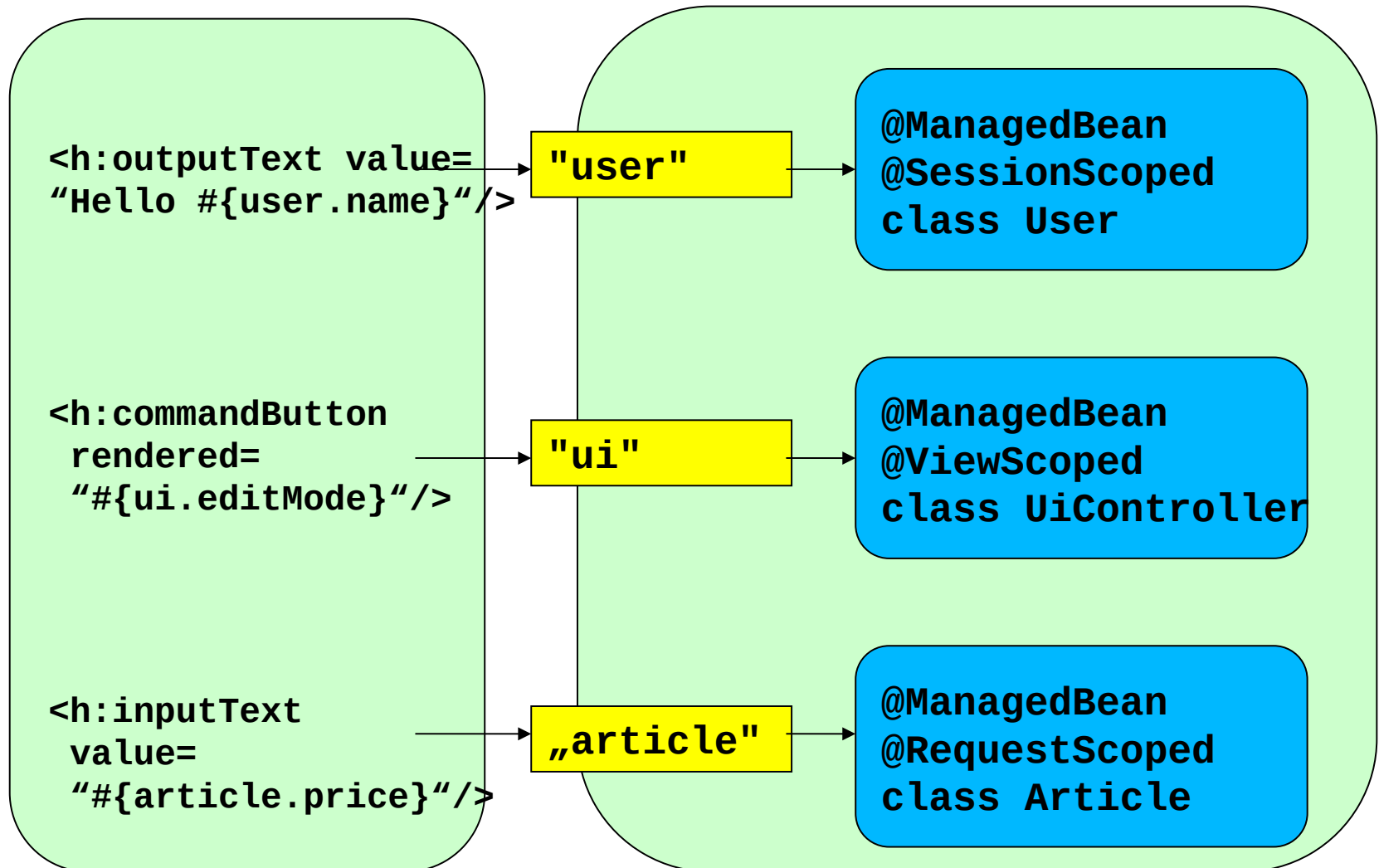
- Application Scope
 - Gültig für die gesamte Zeit der Anwendung
- Session Scope
 - Gültig für die Benutzer-Session
 - „Login – Logout“
- Conversation Scope
 - Gültig für einen Arbeitsablauf
 - „Rechnung erstellen“, „Kunden-Daten aktualisieren“
- View Scope
 - Gültig, so lange keine Seiten-Navigation erfolgt
- Flash Scope
 - Gültig für einen Request-Redirect-Request-Zyklus
- Request Scope
 - Gültig für einen Request



Managed Beans oder CDI?

- Die eben dargestellten Abläufe erinnern sehr stark an CDI
 - JSF enthält seine eigene Implementierung eines CDI-Frameworks
- JSF ist CDI-konform und interoperabel
- Mittelfristig werden die Managed Beans verschwinden





Wissenscheck

- Nennen und erläutern Sie die Transaktionsarten / Transaktionsgrenzen
- Was ist der Unterschied zwischen Container Managed und Bean Managed Transaktionen? Welche Art des Transaktionsmanagement sollte man bevorzugen?
- Was sind Managed Beans?
- Welche Scopes werden durch Managed Beans Unterstützt?
- Welche Schritte werden bei der Verarbeitung eines Requests an eine JSF-Seite durchlaufen?
- Wozu dient Bean Validation?

5.4

PLATFORM PROFILE

Nachtrag Datenbank Locking

- Anwendung:
 - wenn mehrere Anwendungen auf den gleichen Datensatz zugreifen
- Quellen:
 - <https://jakarta.ee/learn/docs/jakartaee-tutorial/current/persist/persistence-locking/persistence-locking.html>
 - <https://www.baeldung.com/jpa-optimistic-locking>
 - <https://www.baeldung.com/jpa-pessimistic-locking>
- Funktionsweise: zu jeder Datenbanktabelle wird eine Spalte „version“ hinzugefügt
- Arten:
 - **Optimistic Locking:** vergleicht die Version kurz vor dem Commit. Wenn die Datensatz geändert wurde (die Versionsnummer höher ist), wird eine OptimisticLock Exception geworfen
 - Lock Mode OPTIMISTIC setzt ein optimistic read lock für alle Entitäten, die das @Version Attribut enthalten
 - **Pessimistic Locking:** long-term Lock auf daten bis die Transaktion abgeschlossen ist
- “By default, persistence providers use optimistic locking”

5.4 Platform Profile

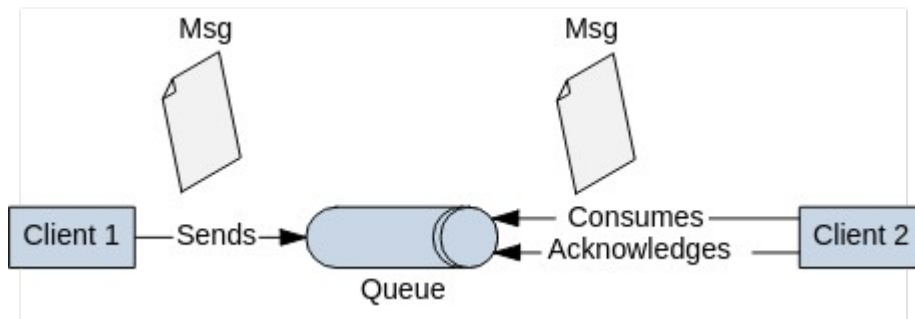
JMS MESSAGING

Plattform: JMS Messaging

- Ermöglicht lose Kopplung zwischen Komponenten durch asynchrone Kommunikation

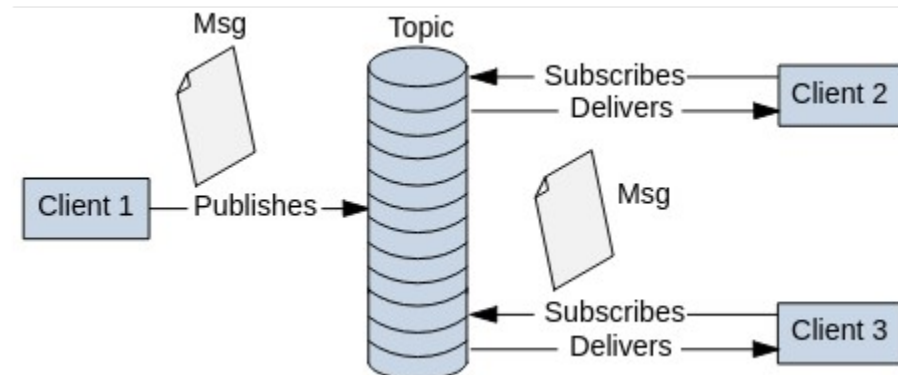
Queue:

- Nachrichten werden in einer Queue zwischengespeicher und von Consumern der Reihe nach abgearbeitet
- Parallele Nachrichtenverarbeitung durch mehrere Consumer (MessageListener, MessageDrivenBean)
- Bei korrekter Verarbeitung wird die Nachricht aus der Queue entfernt, sobald sie von einem Consumer verarbeitet wurde

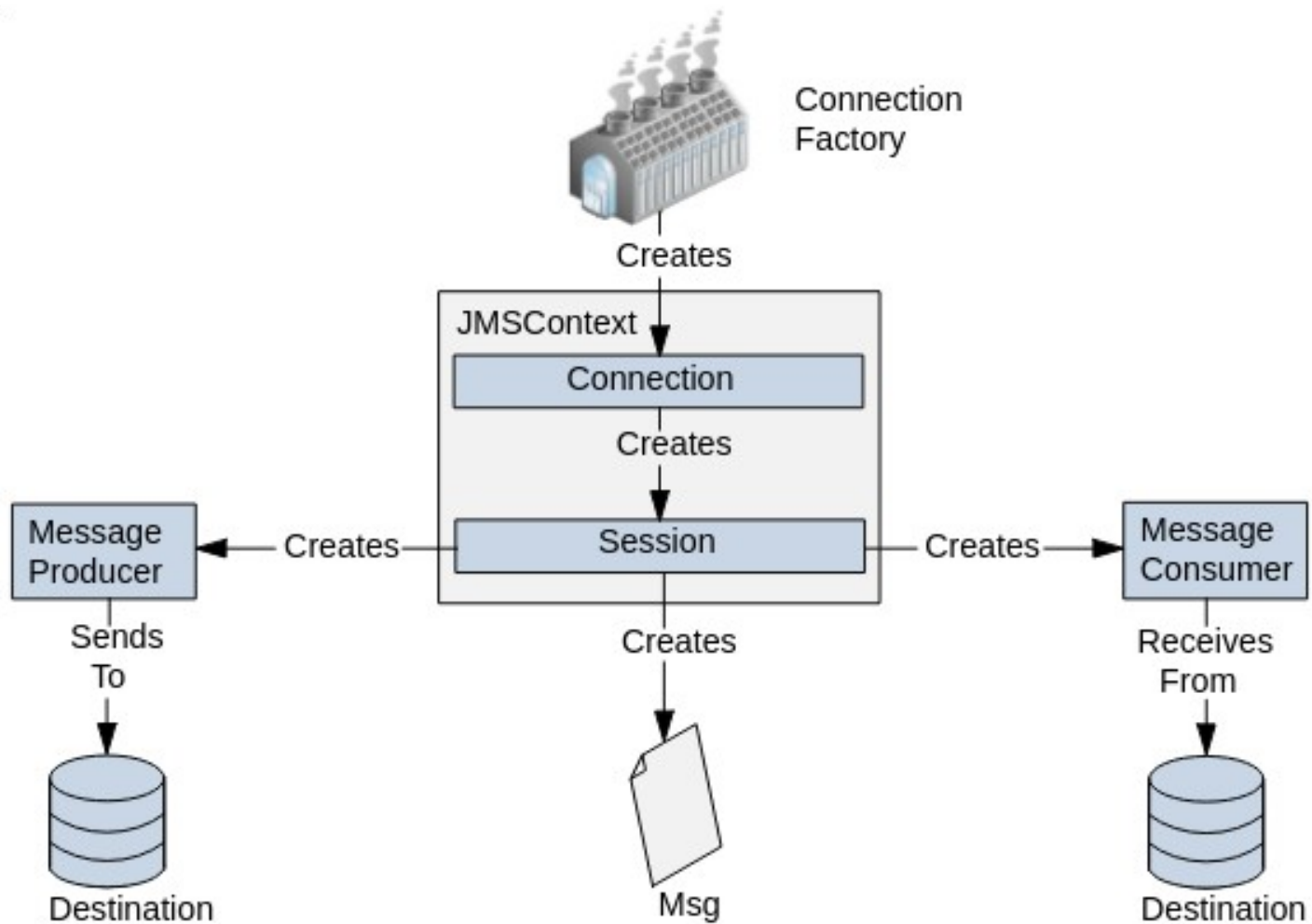


Topic:

- Jede Nachricht kann von einer beliebigen Anzahl von Verbrauchern verarbeitet werden (publish - subscribe)



Platform: JMS Messaging - Bestandteile



Plattform: JMS Messaging - Bestandteile

- **ConnectionFactory:**
 - Vom Provider bereitgestellte Bibliothek, zum verbinden mit einer Queue oder einem Topic
 - Über Resource Injection verfügbar
- **Queues, Topics:**
 - über einem JNDI-Namen und Resource Injection erreichbare, vom Message Provider administrierte Objekte

Platform: JMS Messaging - Bestandteile

- Connection:
 - Virtual Connection zum Message Provider, welche einen offenen TCP/IP Socket repräsentieren kann
- Session:
 - Single-Threaded Context zum Erzeugen+Senden um Empfangen von Nachrichten
- JMSContext:
 - Erzeugt Connection und Session zum Versenden oder Empfangen von Nachrichten

Platform: JMS Messaging - Bestandteile

- Producer:
 - Erzeugt und versendet Nachrichten
(context.createProducer().send(dest, message);
- Consumer:
 - Empfängt Nachrichten (Message m = consumer.receive())
- Message Listener:
 - Asynchroner Event Handler für Messages
 - Implementiert das MessageListener Interface mit der Methode “onMessage”
 - Wird im Consumer gesetzt
 - MyListener myListener = new MyListener();
 - consumer.setMessageListener(myListener);

Plattform: JMS Messaging - Bestandteile

- Message Driven Bean:
 - Implementiert das `MessageListener` Interface mit der Methode `onMessage`
 - Durch direkte Zuordnung der Queues muss kein `MessageListener` mehr implementiert werden

Platform: JMS Messaging - Bestandteile

- Message DrivenBean als MessageListener

```
@MessageDriven(activationConfig = {
    @ActivationConfigProperty(propertyName = "destination", propertyValue = "java:/jms/queue/mySampleQueue"),
    @ActivationConfigProperty(propertyName = "destinationType", propertyValue = "jakarta.jms.Queue")
})
public class MyMessageListener implements MessageListener {
    public void onMessage(Message message) {
        TextMessage textMessage = (TextMessage) message;
        try {
            System.out.println("Message received: " + textMessage.getText());
        } catch (JMSEException e) {
            System.out.println(
                "Error while trying to consume messages: " + e.getMessage());
        }
    }
}
```

Platform: JMS Messaging - Übung

- Ziel:
 - Komponenten sollen mittels JMS Messages asynchron miteinander kommunizieren / Daten austauschen
- Aufgabe
 - Füge eine Queue hinzu
 - Erstelle einen Producer, der mittels Webservice aufgerufen wird
 - Erstelle mittels MessageDrivenBean einen MessageListener
 - Teste das Versenden von Messages. Was passiert, wenn du den MessageListener deaktivierst?

- Durch welche Bestandteile wird das JMS Messaging in Jakarta EE umgesetzt?
- Welche Messagingsoftware ist im Wildfly integriert?
- Welche zwei möglichen Arten des Messaging gibt es?
- Was ist eine XA-Transaktion?
- Unterstützt JMS synchrone oder Asynchrone Kommunikation?

5.4 Platform Profile

BATCH PROCESSING

Plattform: Batch Processing

- Für Aufgaben
 - die ohne Benutzerinteraktion ausgeführt werden können
 - die eine große Anzahl von Daten auf einmal verarbeiten sollen
 - die regelmäßig ausgeführt werden sollen
- Partielles Einlesen der Daten in sogenannten Chunks möglich
- Parallele Verarbeitung von Teilen, die nicht voneinander Abhängen, kann konfiguriert werden
- Beispiele: Abrechnung, Berichtserstellung, Datenformatkonvertierung, Bildverarbeitung

Plattform: Batch Processing - Bestandteile

- Das Batch Processing Framework besteht aus:
 - batch runtime: Ausführen von Jobs
 - Job specification language basierend auf XML
 - Java API

- Ein Job (XML Spezifikation) kann aus folgenden Teilen bestehen:
 - Steps
 - Flows
 - Splits
 - Decision elements

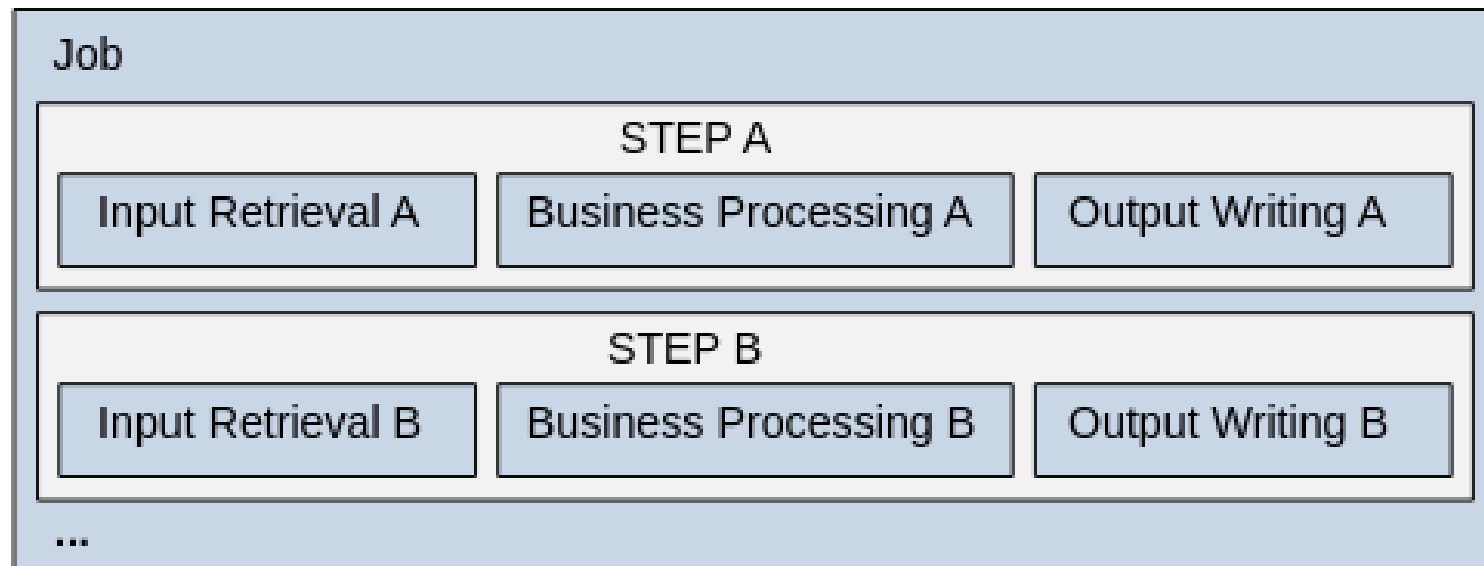
Platform: Batch Processing - Bestandteile

- Job

- Step1
 - Batchlet – task-oriented
- Step2
 - Chunk - chunk-oriented
 - Input Retrieval: Eingabe
 - Business Processing: Verarbeitung (hier sind auch mehrere Schritte möglich)
 - Output Writing: Ausgabe

Platform: Batch Processing - Bestandteile

- Step with Chunks:



Platform: Batch Processing - Bestandteile

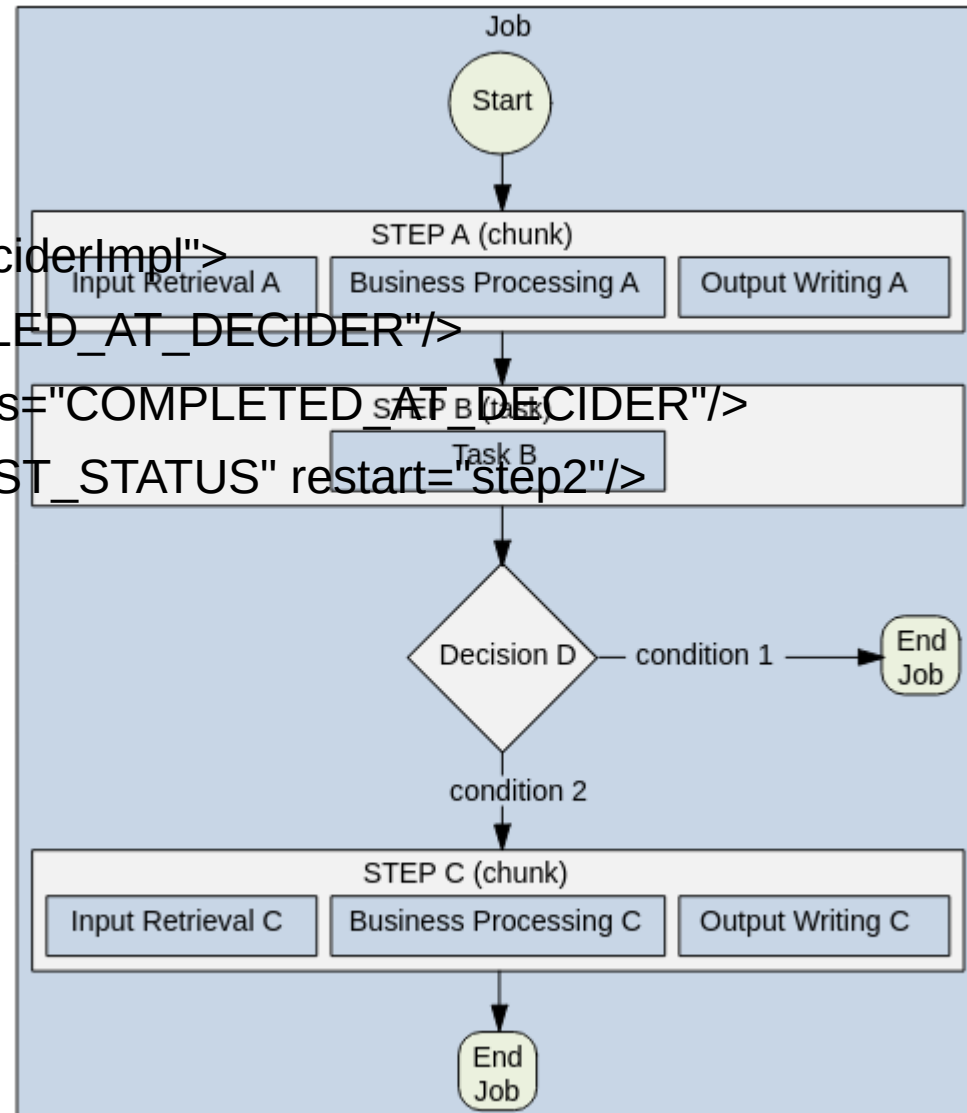
- Batch Status Values für Jobs, Steps, Splits, Flows:

Value	Description
STARTING	The job has been submitted to the batch runtime.
STARTED	The job is running.
STOPPING	The job has been requested to stop.
STOPPED	The job has stopped.
FAILED	The job finished executing because of an error.
COMPLETED	The job finished executing successfully.
ABANDONED	The job was marked abandoned.

Platform: Batch Processing - Bestandteile

- Job with decision element:

```
<decision id="decisionA" ref="MyDeciderImpl">  
  <fail on="FAILED" exit-status="FAILED_AT_DECIDER"/>  
  <end on="COMPLETED" exit-status="COMPLETED_AT_DECIDER"/>  
  <stop on="MY_TEMP_ISSUE_EXIST_STATUS" restart="step2"/>  
</decision>
```



Plattform: Batch Processing - Übung

- Aufgabe

- Der Trainingskatalog soll nun regelmäßig als CSV-Datei importiert werden. Dazu soll ein Batch-Job geschrieben werden

- Schritte:

- Erstelle im Trainingsservice eine JPQL-Query, welche ein Training anhand des Titels zurückgeben kann
- Erstelle einen Scheduled Job, der alle 5 Sekunden die CSV-Datei einliest und die Trainingsdaten neu anlegt oder bei gleichem Titel aktualisiert

- Wie wird der Batch Job in Jakarta EE definiert und spezifiziert?
- Welche Bestandteile kann die Job-Spezifikation haben?
- Welche Arten von Steps gibt es?
- Welche Status-Werte gibt es für Steps?
- Was ist eine XA-Transaktion?
- Nenne Beispiele, bei denen Batch-Processing eingesetzt werden sollte