LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Datenbankpraktikum, Gruppe F

JPA mit Hibernate

Alexander Stautner, Harald Zauner, Sascha Schreier

Agenda

- Objektrelationales Mapping
- JPA / Hibernate
- Demo

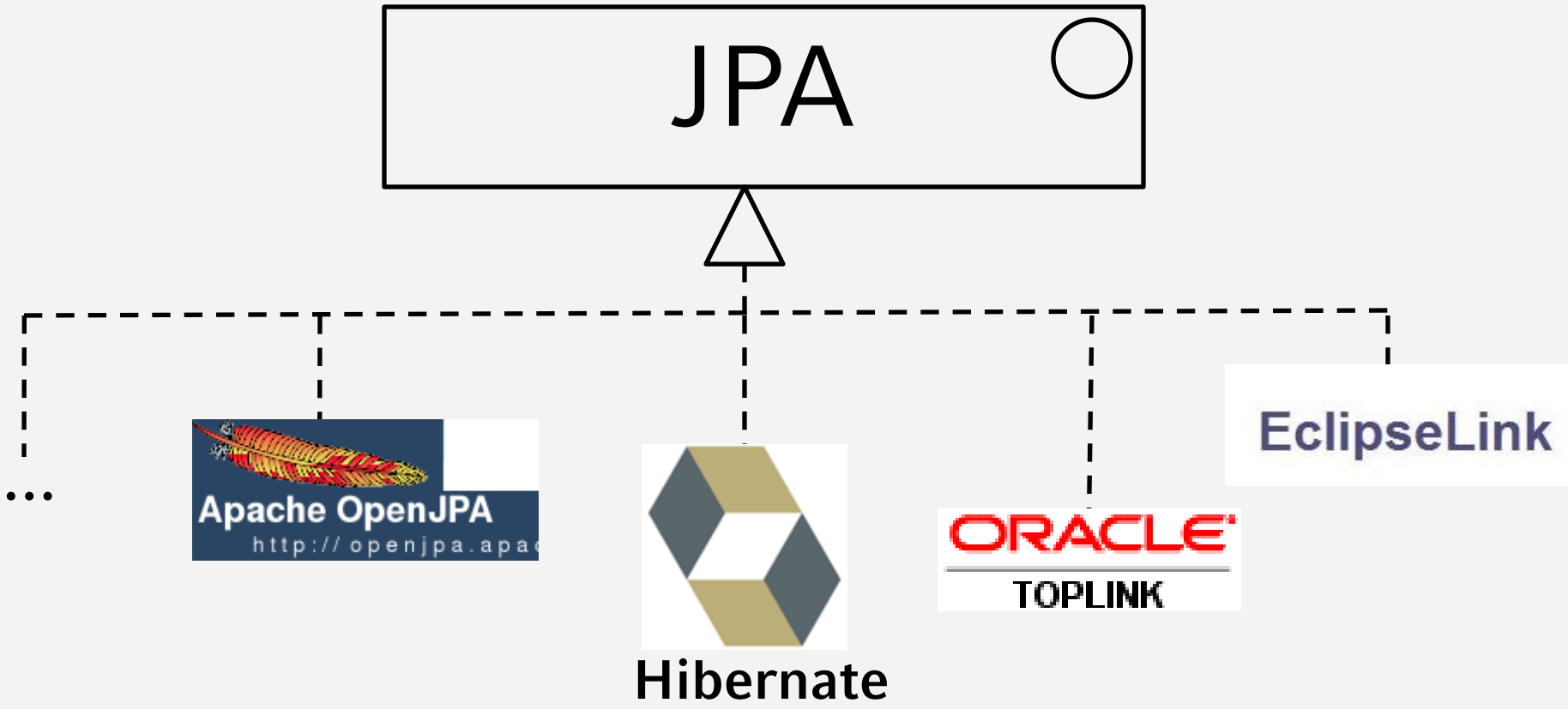
Objektrelationales Mapping

Objektorientierte Welt	Relationale Welt
➢ Objekte	➢ Tabellen/Relationen
– Objektidentität	– Primärschlüssel
– komplexe Attribute	– Grunddatentypen als Spalten
– Kapselung	– ?, ggf. Views
➢ Assoziationen	➢ Beziehungen („Relationships“)
– auch unidirektional	– bidirektional
➢ Vererbung	➢ ?
➢ ?, ggf. über Methoden	➢ Kaskadierendes Löschen

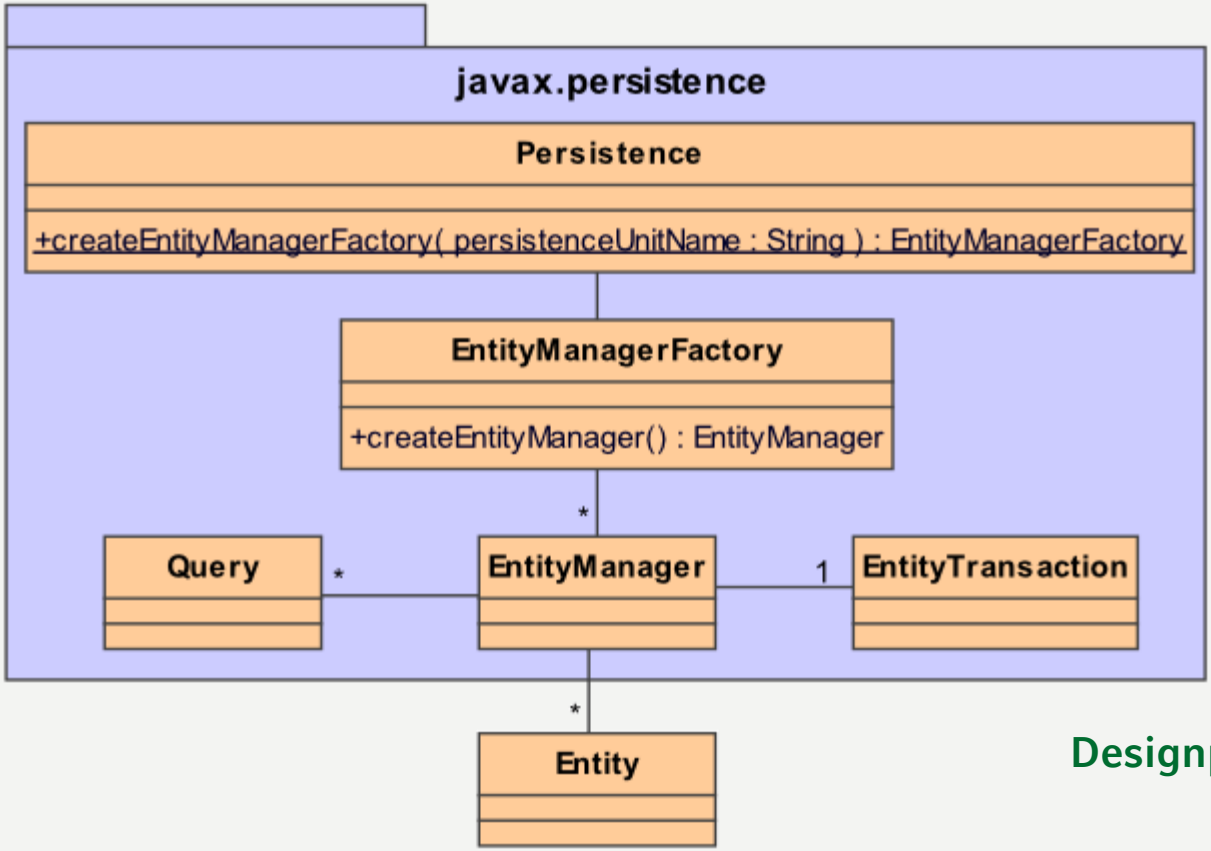
Objektrelationales Mapping (ORM):
Mapping von Objekten auf die Datenbank

JPA: Java Persistence API (seit 2006)

JPA: „Java-API for the management of persistence and objectrelational mapping“



JPA: Java Persitence API
Architektur



Designprinzipien

- für JAVA SE als auch Java EE
- Configuration by Convention

persistence.xml: Konfiguration der Persistenzeinheiten (Name, zugehöriger Provider)

- Properties zur Konfiguration des Persistenzproviders (z.B. Hibernate)

META-INF/persistence.xml

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd" version="1.0">

<persistence-unit name="vortrag">
<provider>org.hibernate.ejb.HibernatePersistence</provider>
<properties>
<!-- JDBC - Konfiguration -->
<property name="hibernate.connection.driver_class" value="oracle.jdbc.OracleDriver" />
<property name="hibernate.connection.url" value="jdbc:oracle:thin:@localhost:1521:xe" />
<property name="hibernate.connection.username" value="username" />
<property name="hibernate.connection.password" value="password" />

<!-- Der ausgewählte Hibernate Dialekt -->
<property name="hibernate.dialect" value="org.hibernate.dialect.Oracle10gDialect" />

<!-- Steuerung der Tabellengenerierung -->
<property name="hibernate.hbm2ddl.auto" value="create" />
</properties>
</persistence-unit>

</persistence>
```

Objektrelationales Mapping

per XML — per Annotationen

META-INF/orm.xml

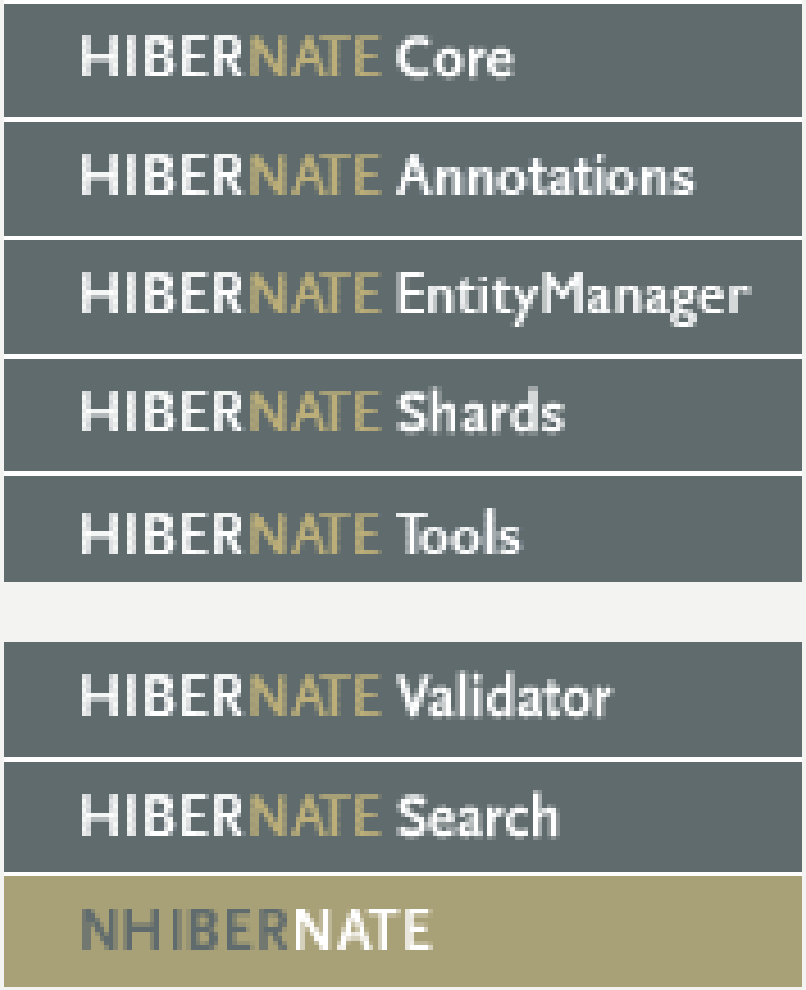
```
<entity-mappings
xmlns="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm
http://java.sun.com/xml/ns/persistence/orm_1_0.xsd"
version="1.0">
  <package>model</package>
  <entity class="Person" access="FIELD">
    <attributes>
      <id name="id">
        <generated-value />
      </id>
    </attributes>
  </entity>
</entity-mappings>
```

Im Zweifelsfall wird **orm.xml** bevorzugt!

model/Person.java

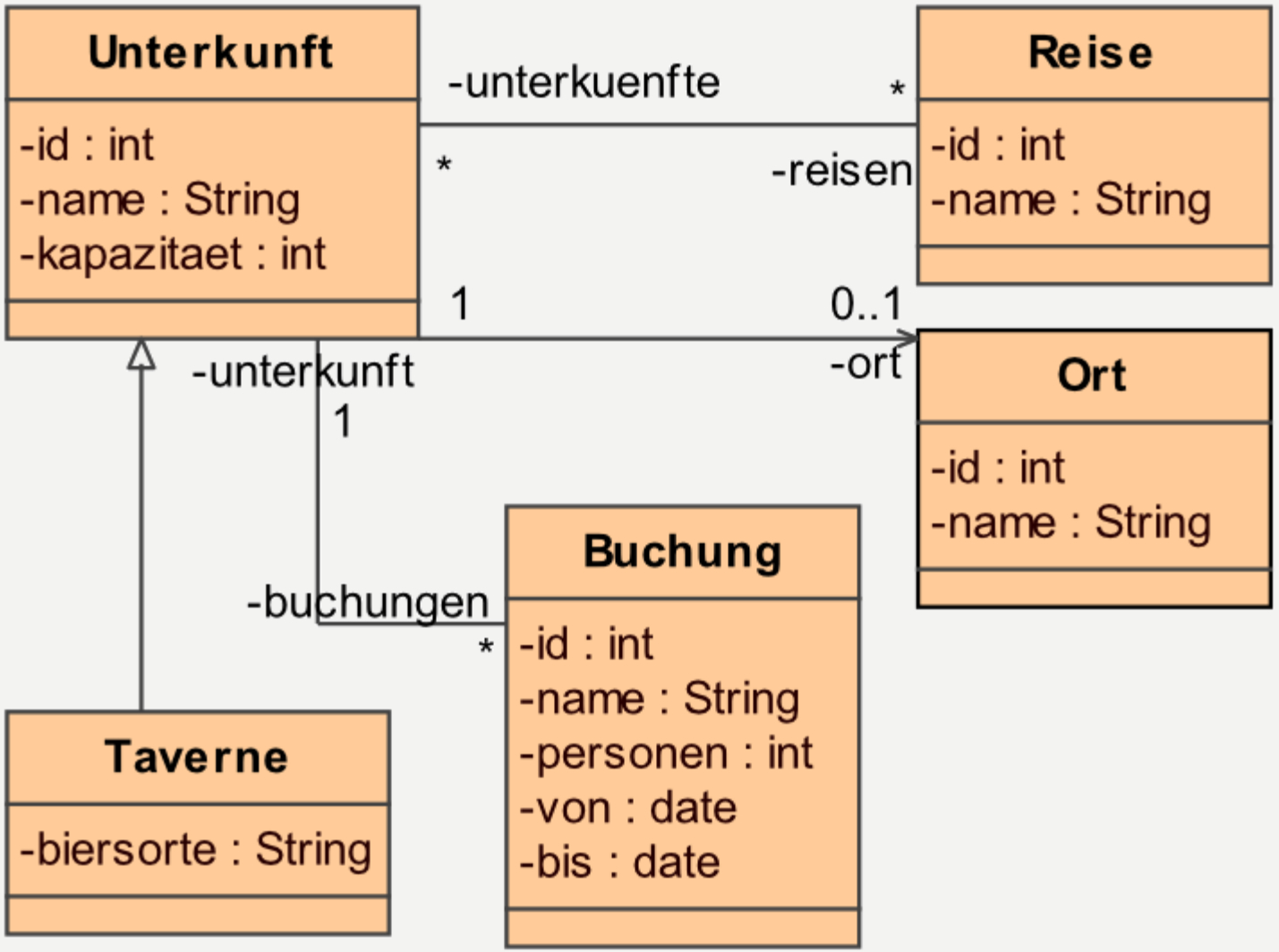
```
package model;

@Entity
public class Person implements Serializable {
  @Id
  @GeneratedValue
  private int id;
  private String name;
  public Person() {}
  public int getId() { return this.id; }
  public String getName() { return this.name; }
  public void setName(String name) {
    this.name = name;
  }
}
```

Hibernate (seit 2002)



Quelle:
Hibernate.org

Demo
Klassenmodell



Modellierung
Entität

Unterkunft.java

```
@Entity
public class Unterkunft
implements Serializable {
...
@Id
@GeneratedValue
private int id;

@Basic(optional = false)
private String name;

private int kapazitaet;

public Unterkunft() {
}
...
```

`@Entity` („An entity is a lightweight persistent domain object“)

Klasse sollte Java Beans Konventionen folgen:

- (evtl. impliziter) Standardkonstruktor, Sichtbarkeit **`public`**
- Getter/Setter-Methoden
- `java.io.Serializable` implementieren

`@Id` (Festlegung des Primärschlüsselattributs)

`@GeneratedValue` (Generierung von Primärschlüsseln)

- **strategy** = **AUTO** | IDENTITY | SEQUENCE | TABLE

`@Basic`

Steht (implizit) vor jedem Attribut

- **optional** = **true** | false
- **fetch** = **EAGER** | LAZY

Primitive Attribute sind nie optional!

EntityManager

EntityManagerFactory

- erzeugt EntityManager für eine Persistenzeinheit (→ **persistence.xml**)

Persistenzkontext

- Menge von Objekten/Entitäten
- Lebensdauer: transaktionsgebunden (Java EE) oder erweitert (Java SE/EE)

EntityManager

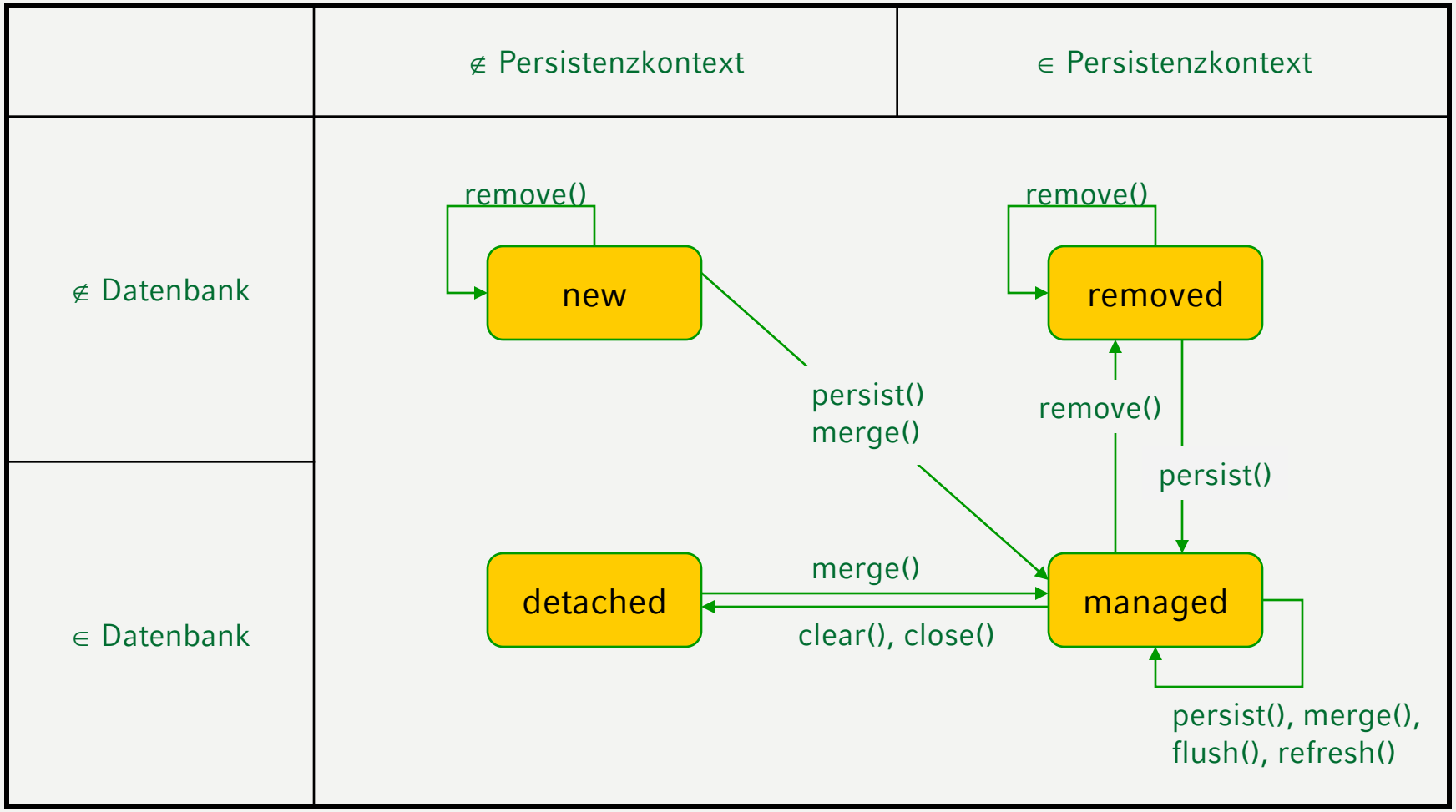
- verwaltet Persistenzkontext: **clear()** , **contains()**
- Transaktionsmanagement: **getTransaction()**
- Synchronisation: **flush()** (Objekt → DB), **refresh()** (DB → Objekt)

EntityTransaction

- **begin()**, **commit()** / **rollback()**
- WICHTIG: spätestens bei **commit()** wird ein **flush()** ausgeführt, also alle managed Objekte persistiert

Objektlebenszyklus

Zustandsübergänge



Unterkunft.java (besitzende Entität)

```
...
@OneToOne
private Ort ort;
...
```

Ort.java

```
@Entity
public class Ort implements
Serializable {
...
@Id
@GeneratedValue
private int id;

@Basic(optional = false)
private String name;
...
```

Modellierung einer 1:1 Beziehung
(besitzende Entität beliebig)

`@OneToOne`

- **fetch**, **optional** wie bei `@Basic`
- **mappedBy**: gibt zugehöriges Feld an (nur bei der nicht besitzenden Entität benötigt)
- **cascade** $\subseteq$ { MERGE, PERSIST, REFRESH, REMOVE }
  gibt an, welche Operationen (des `EntityManagers`) kaskadiert werden sollen

WICHTIG: JPA kümmert sich ausschließlich um die besitzende Entität

Modellierung
1:n Beziehung

Unterkunft.java

```
...
@OneToMany(cascade = CascadeType.ALL,
mappedBy = "unterkunft")
private Collection<Buchung> buchungen;
...
```

Buchung.java (besitzende Entität)

```
@Entity
public class Buchung
implements Serializable {
...
@Id
@GeneratedValue
private int id;

@Basic(optional = false)
private String name;

@ManyToOne(cascade = {
CascadeType.MERGE,
CascadeType.PERSIST },
optional = false)
private Unterkunft
unterkunft;
...
```

Modellierung einer 1:n Beziehung
(besitzende Entität ist diejenige mit der Multiplizität n)

`@OneToMany`

- **fetch** = EAGER | LAZY
- **mappedBy**, **cascade** wie bei `@OneToOne`

`@ManyToOne`

- **fetch**, **cascade**, **optional** wie bei `@OneToOne`

Unterkunft.java

```
...
@ManyToMany(mappedBy =
"unterkuenfte")
private Collection<Reise> reisen;
...
```

Reise.java (besitzende Entität)

```
@Entity
public class Reise implements
Serializable {
...
@ManyToMany
private Collection<Unterkunft>
unterkuenfte;
...
```

Modellierung einer n:m Beziehung
(besitzende Entität beliebig)

`@ManyToMany`

- alle Attribute (**fetch**, **mappedBy**, **cascade**) wie bei `@OneToMany`

Unterkunft.java

```
@Entity
@Inheritance(strategy =
InheritanceType.JOINED)
public class Unterkunft
implements Serializable {
...
```

Taverne.java

```
@Entity
public class Taverne extends
Unterkunft {
...
@Basic(optional = false)
private String biersorte;
```

Modellierung einer is-a Beziehung

`@Inheritance`

- **strategy** = **SINGLE_TABLE** | JOINED | TABLE_PER_CLASS

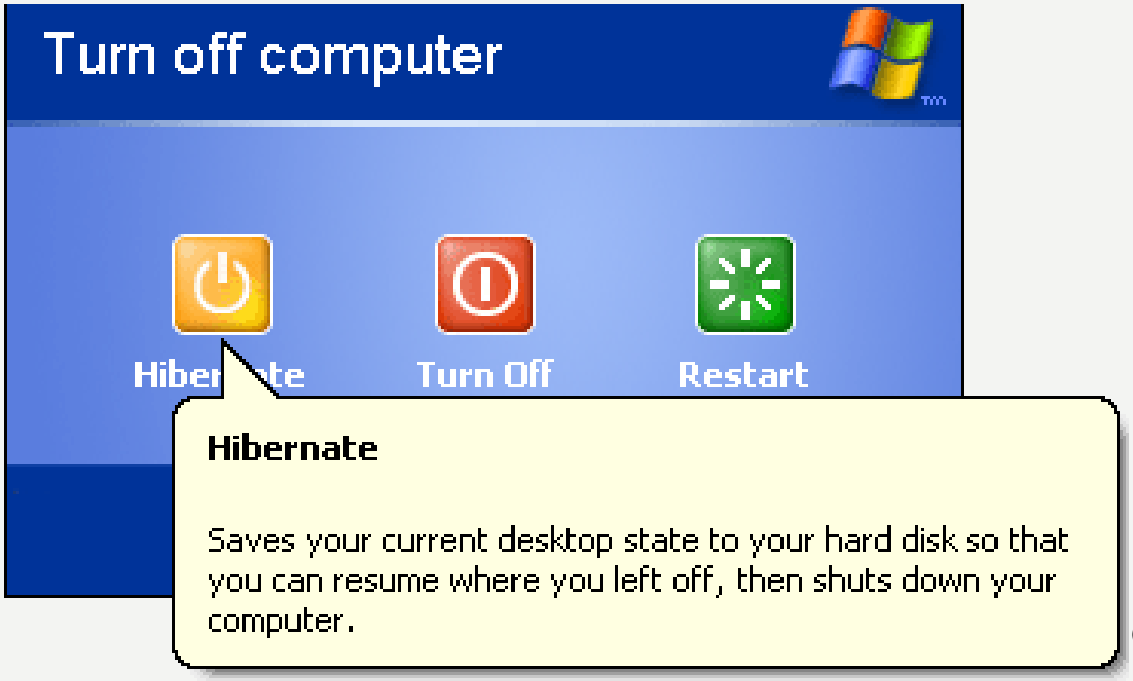
Query Language

Falls Primärschlüssel bekannt

- EAGER Fetching: `entityManager.find(Unterkunft.class, 1);`
- LAZY Fetching: `entityManager.getReference(Unterkunft.class, 1);`

JPQL: Java Persistence Query Language (SELECT, UPDATE, DELETE)

- SQL-ähnliche Syntax
- Deklaration von Variablen in der FROM-Klausel: SELECT u FROM Unterkunft u
=> getypt, polymorph
- Parameter werden gesetzt über
  - die Position
    SELECT u FROM Unterkunft u WHERE u.kapazitaet BETWEEN ?1 AND ?2
  - den Namen
    SELECT u FROM Unterkunft u WHERE u.kapazitaet BETWEEN :lower AND :upper
- Erzeugen mittels `entityManager.createQuery("SELECT u FROM Unterkunft u");`
- Native Queries: `entityManager.createNativeQuery("SELECT * FROM Unterkunft");`
- Abfrage des Ergebnisses: `.getResultList()`



© Microsoft

Vielen Dank für eure Aufmerksamkeit !

Literatur:

- *JSR 220: Enterprise JavaBeans 3.0*
- *Müller, Wehr: Java-Persistence-API mit Hibernate*