

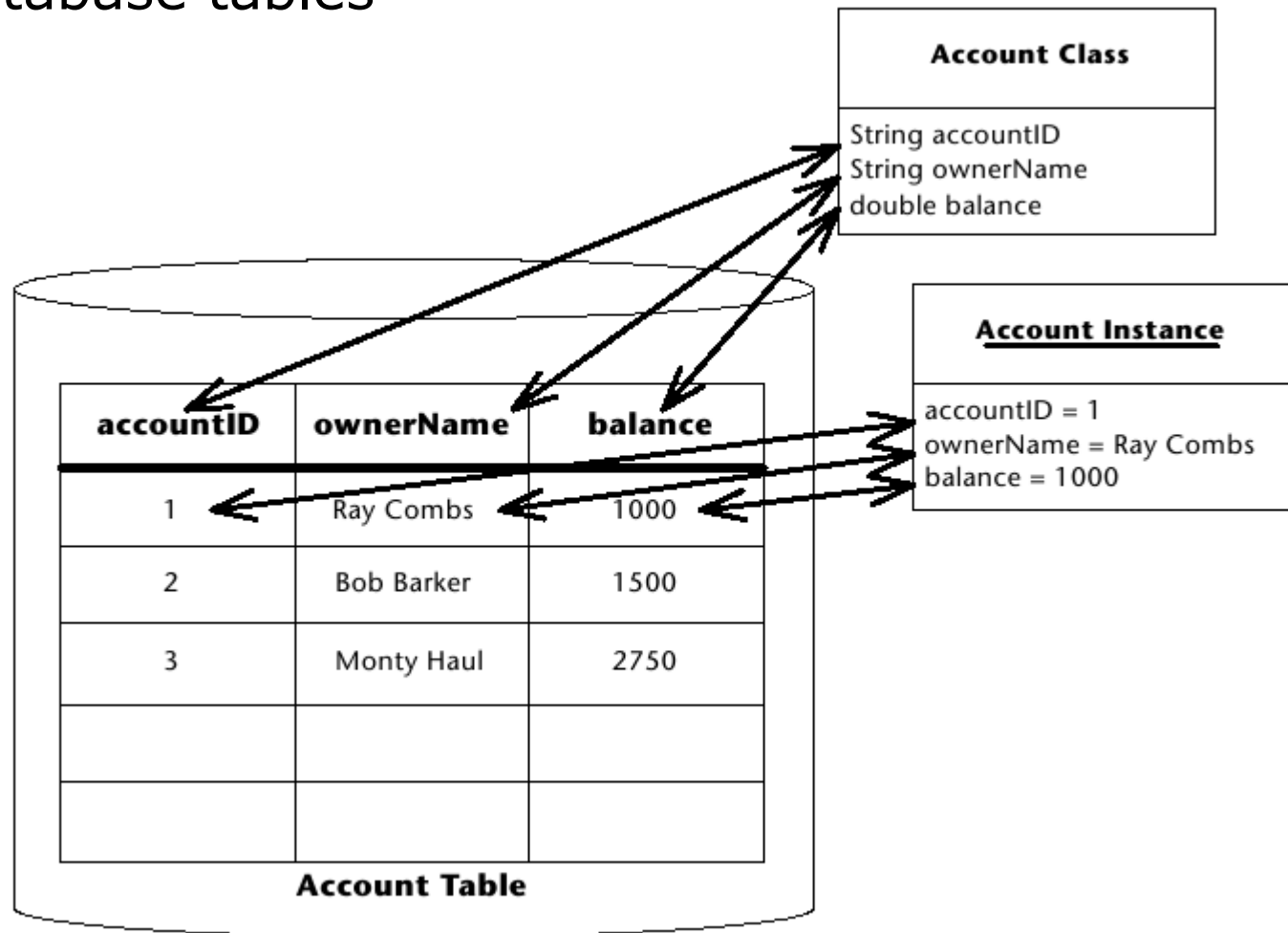
Programming with Java EE 6

Module 3

DI Michael Schaffler-Glößl

Object relational Mapping

OR-Mapping is the mapping of objects to database tables



The simplest case is always one object to a one table manner.

Foreign key relationships in the database will be implemented by associations on the class side.

There are 3 cases of foreign key relationships:

- 1:1 relationship (1 customer, 1 creditcard)

- 1: n relationship (1 customers, n orders)

- n: m relationship (n orders, m articles)

Table Customer

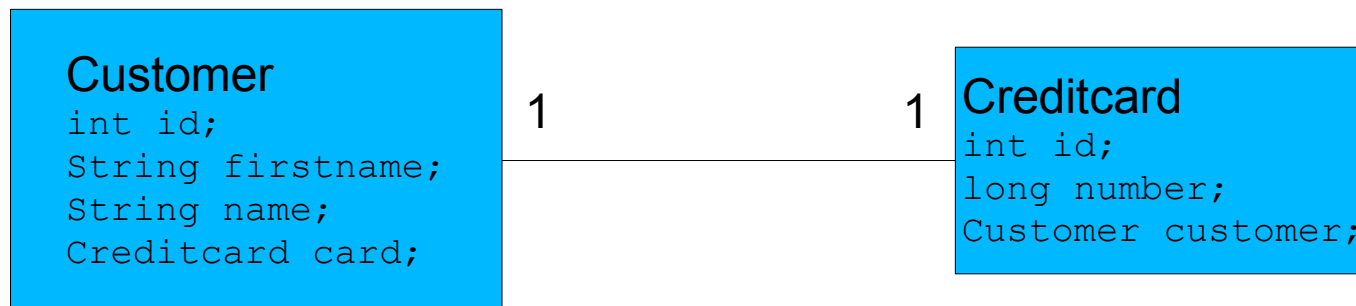
ID	Firstname	Name	ID_Card
1	Heinz	Zach	4
2	Katharina	Horvath	2
3	Martina	Clement	5

1:1

Table Creditcard

ID	Number
4	6543342
2	3232345
5	2345676

1:1 Relationships are not usually split to 2 tables
One table is sufficient



1:n

Table Customer

Firstname	Name	ID_Karte
Heinz	Zach	4
Katharina	Horvath	2
Martina	Clement	5

Table Order

ID	ID_Customer
1	1
2	1
3	1
4	2
5	2
6	3

1:n Relationships are usually mapped on 2 tables

Normally, only one table has the other table foreign key as the primary key

BUT:

1:n Relations can be also defined by a separate intermediate table

A table can contains the primary key of the other and vice versa (both tables refer mutually)

1:n

Table Customer

Firstname	Name	ID_Karte
Heinz	Zach	4
Katharina	Horvath	2
Martina	Clement	5

Table Order

ID	ID_Customer
1	1
2	1
3	1
4	2
5	2
6	3



m:n

Table Oder

ID	ID_Customer
1	1
2	1
3	1
4	2
5	2
6	3
7	1

Table Article_to_order

ID_Order	ID_ARTICLE
1	3
1	4
1	5
2	4
2	3
2	2
2	1

Table Article

ID	Name	Modelname
1	HP Computer	SX 4000
2	Disketten	Verbatim 1.44
3	CD	740 MB
4	Toner	Samsung SCX2000
5	Drucker	Samsung SCX2000



An entity class is a simple Java class.

You can also say => POJO Plain Old Java Object

For each entity class, there is a mapping to one (or more) table(s) in a relational database. This mapping will be declared using Java Annotations.

Each entity class has a primary key, the objects of the class can be uniquely identified.

Entity classes (in contrast to the previous Entity Beans) do not need EJB container, they can also exist in the Web container or in standalone Java applications.

Definition of the Mappings to the Database

```
@Entity()  
public class Student implements Serializable {  
  
    public Student() {  
    }  
  
    @Id()  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private Long id;  
  
    @Column()  
    private Long matrikelnummer;  
  
    public Long getId() {return this.id;}  
    public void setId(Long id) {this.id = id;}  
  
    public Long getMatrikelnummer() {return matrikelnummer;}  
    public void setMatrikelnummer(Long matrikelnummer)  
        {this.matrikelnummer = matrikelnummer;}  
}
```

- The Entity Manager is an instance, that controls the lifecycle of the entity classes .
- It instantiates Entity classes by doing queries against the database.
- It synchronizes Entity objects with the database.

Entity Manager

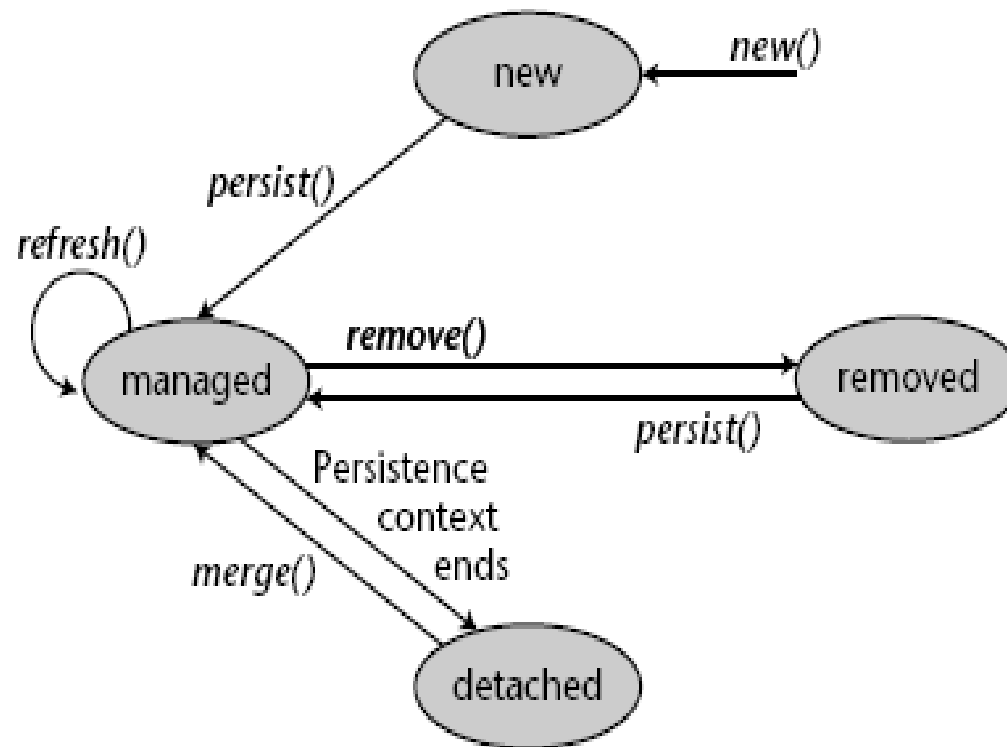
```
@Stateless
@WebService()
public class StudentServiceBean implements
    StudentServiceRemote, StudentServiceLocal {
    @PersistenceContext()
    EntityManager em;

    /** Creates a new instance of StudentServiceBean */
    public StudentServiceBean() {
    }

    public Student getStudentByID(Long id) {
        Student student = em.find(Student.class, id);
        return student;
    }

    public Jahrgang[] getAllJahrgaenge() {
        Query query =
em.createNamedQuery("findAllJahrgaenge");
        List<Jahrgang> jahrganglist = query.getResultList();
        return jahrganglist.toArray(new Jahrgang[10]);
    }
}
```

Lifecycle of an Entity Class



Lifecycle of Entity Objects

New: The object has been created using the new operator in the Java Virtual Machine (JVM). The object has no equivalent in the relational database, it will not be managed by the Entity Manager.

Managed: The object is managed by the Entity Manager. The Entity Manager knows whether attributes of an object has been changed. It decides when the changed attributes will be written in the database and when the new object will be read from the database. The writing in the database and the refresh can be enforced programmatically.

Removed: The object still exists in the JVM, it will be managed by the entity manager, but is marked for deletion in the database.

Detached: The object exists in the database and in the JVM, but it will no longer be managed by the Entity Manager (it occurs when the object is being transported between JVMs, or the transaction has ended).

`Persist()`: An object in the state **new** is given to the Persistence Manager (managed) to be managed and will be written to the database

`refresh()`: An object that is in the state **managed**, will be read again from the database

`remove()`: An object that is in the state **managed** is marked for deletion, and with the termination of the transaction will be removed from the database. The Java object remains.

`merge()`: An object that will not be managed by the Persistence Manager, but it is with the primary key already in the database, will be managed by the Persistence Manager. The data of the object will be written to the database when the transaction ends.

`flush()`: All changes of the object are written to the database

`find()`: The object with the specified primary key will be read from the database.

`createQuery(„select ...“)`: A custom database query will be defined and using `query.getResultList` the result will be read in the form of Java objects.

```
@Stateless
@WebService()
public class StudentServiceBean implements
    StudentServiceRemote, StudentServiceLocal {
    @PersistenceContext() EntityManager em;

    /** Creates a new instance of StudentServiceBean */
    public StudentServiceBean() {
    }

    public Student getStudentByID(Long id) {
        Student student = em.find(Student.class, id);
        return student;
    }

    public Jahrgang[] getAllJahrgaenge() {
        Query query =
            em.createNamedQuery("findAllJahrgaenge");
        List<Jahrgang> jahrganglist = query.getResultList();
        return jahrganglist.toArray(new Jahrgang[10]);
    }
}
```

```
@Stateless
@WebService()
public class StudentServiceBean implements StudentServiceRemote,
    StudentServiceLocal {

    @PersistenceContext()
    EntityManager em;

    public void createSomeTestData() {
        Adresse adresse = new Adresse();
        Jahrgang jahrgang = new Jahrgang();

        jahrgang.setJahr(2004);
        em.persist(jahrgang);

        adresse.setStrasse("Thomas Edison Straße");
        adresse.setHausnummer("2");
        adresse.setOrt("Eisenstadt");
        adresse.setPlz("7000");
        em.persist(adresse);

        Student student1 = new Student();
        student1.setVorname("Peter");
        student1.setNachname("Wind");
        student1.setMatrikelnummer(123456L);
        student1.setEmail("peter.wind@abc.com");
        student1.setAdresse(adresse);
        student1.setJahrgang(jahrgang);
        em.persist(student1);
    }
}
```

As in case of Session Beans the Callbackmethods will be marked with Annotations :

@PrePersist, @PostPersist

@PreRemove, @PostRemove

@PreUpdate, @PostUpdate

@PostLoad

Mapping Annotations

The mapping of Objects to the Database will be done via Annotations:

`@Entity` ... defines the class as an entity class; as default - table name = class name

`@Id` ... defines an attribute as a primary key of an Entity class

`@GeneratedValue(strategy = GenerationType.AUTO)` ... says to the Persistence Manager, that the primary key of the Entity class should be automatically generated

`@Column` ... defines an attribute to persist; as default - the attribute name = column name

Mapping Annotations

```
@Entity()
public class Student implements Serializable {

    public Student() {
    }

    @Id()
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Column()
    private Long matrikelnummer;

    public Long getId() {return this.id;}
    public void setId(Long id) {this.id = id;}

    public Long getMatrikelnummer() {return matrikelnummer;}
    public void setMatrikelnummer(Long matrikelnummer)
        {this.matrikelnummer = matrikelnummer;}

}
```

By the definition of foreign key relationships should always been taken into the consideration, on which side of the two cross reference tables is the foreign key located (see next slide):

Table Customer

ID	Firstname	Name
1	Heinz	Zuber
2	Katharina	Horvath
3	Martina	Clement

1:n

Table Order

ID	ID_Customer
1	1
2	1
3	1
4	2
5	2
6	3

The most important annotations for the relationship of foreign key relationships are:

@JoinColumn ... defines an attribute as foreign key

@OneToOne ... defines a 1:1 relationship

@OneToMany, @ManyToOne ... defines a 1:n relationship

@ManyToMany ... defines a m:n relationship

1:1

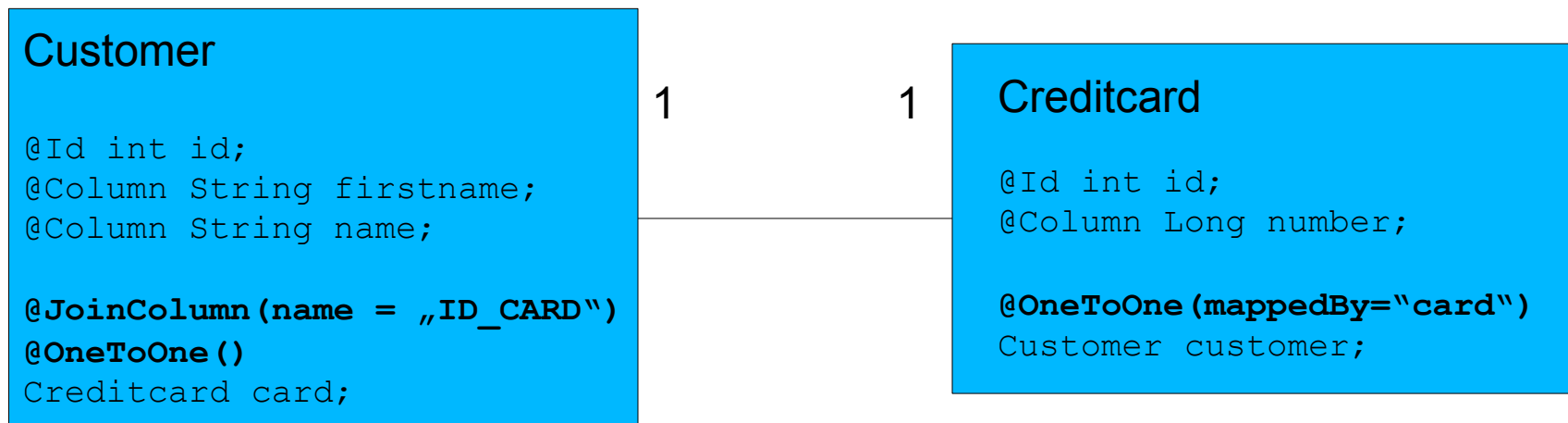
Table Customer

ID	Firstname	Name	ID_CARD
1	Heinz	Zuber	5
2	Katharina	Horvath	2
3	Martina	Clement	4

Table Creditcard

ID	Number
4	6543342
2	3232345
5	2345676

1:1 Relationships are not usually split to 2 tables
One table is sufficient



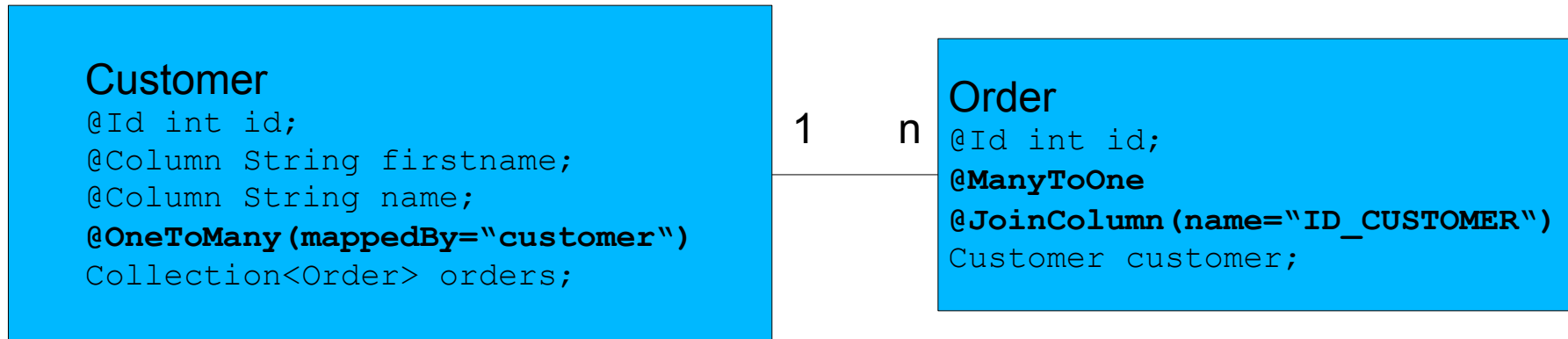
1:n

Table Customer

ID	Fistname	Name
1	Heinz	Zuber
2	Katharina	Horvath
3	Martina	Clement

Table Order

ID	ID_Customer
1	1
2	1
3	1
4	2
5	2
6	3



m:n

Table Order

ID	ID_Customer
1	
2	1
3	1
4	2
5	2
6	3

Table Article_to_order

ID_ORDER	ID_ARTICLE
1	3
1	4
1	5
2	4
2	3
2	2

Table Article

ID	Name	Modelname
1	HP Computer	SX 4000
2	Disketten	Verbatim 1.44
3	CD	740 MB
4	Toner	Samsung SCX2000
5	Drucker	Samsung SCX2000

Order

```
@Id int id;
@ManyToMany()
@JoinTable(name="ARTICLE_TO_ORDER",
    joinColumns={@JoinColumn(name="ID_ORDER")},
    inverseJoinColumns={@JoinColumn(name="ID_ARTICLE")})
Collection<Article> arcticlelist;
```

n

Article

```
@int id;
@Column String name;
@Column String modelname;
@JoinTable(name="ARTICLE_TO_ORDER",
    joinColumns={@JoinColumn(name="ID_ARTICLE")},
    inverseJoinColumns={@JoinColumn(name="ID_ORDER")})
Collection <Order> orders;
```

m

The next lecture

The next module:

- EJB Query Language
- Optimistic / Pessimistic Locking
- Managing Cascading Operations / Constraints
 - Generation of database tables and generation of objects from the database model

Example

The Netbeans Project Example can be downloaded under
<http://javatraining.at/web/guest/javaee6>

Exercise

Work on the Tutorial:

<http://www.netbeans.org/kb/61/javaee/persistence.html>

Create a data model with an m:n relationship and implement an EJB module with corresponding entity classes. Create a Web Service using a Stateless Session Bean for testing of your entity classes.