KATEDRA POČÍTAČOVÉ GRAFIKY A INTERAKCE

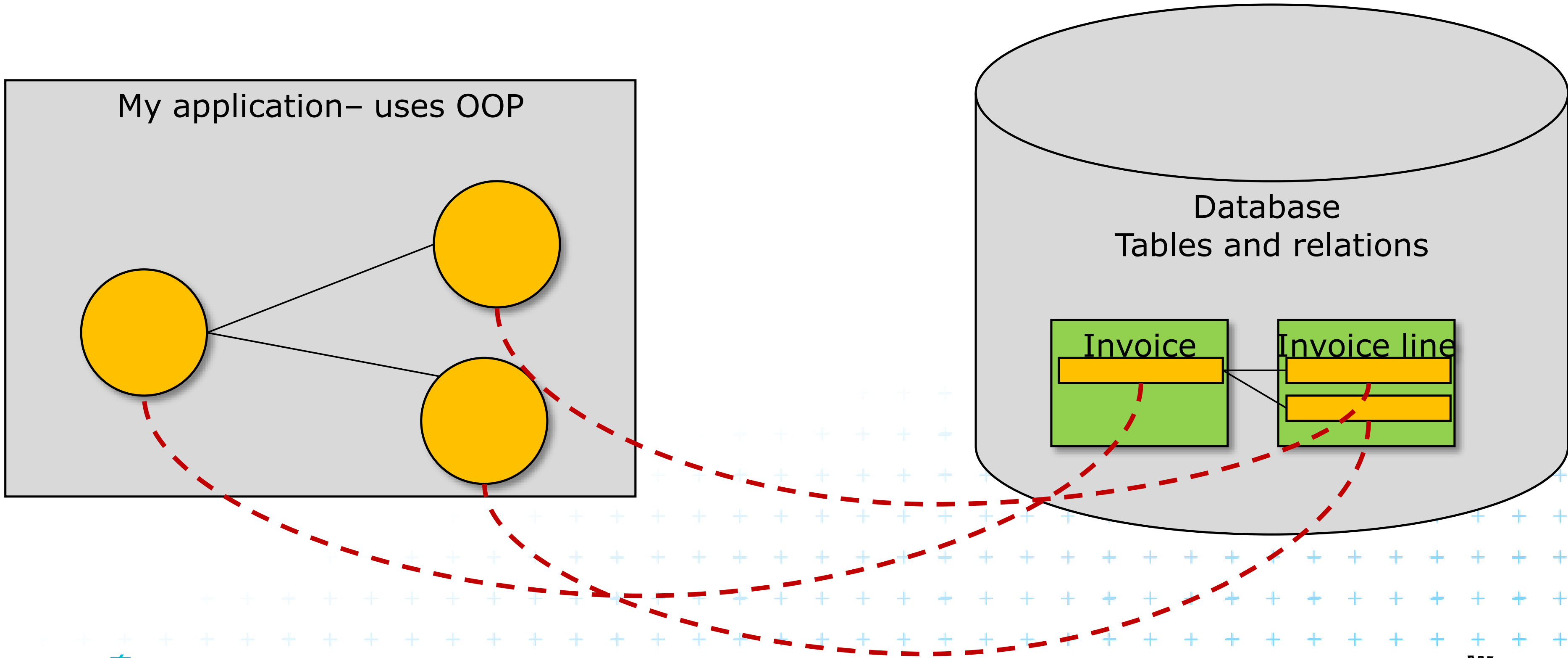
JPA

WA2

Martin Klíma

JPA – Java Persistence API

- JPA is a standardized API to ORM
- ORM = Object – Relational Mapping



ORM – what is it and why we need it?

- OOP works with classes and their instances
- There are object databases that work with OOP natively
  - Performance problems
  - Standardization problems
- Commercial solutions are usually based on RDB
- Direct work with RDB is possible using JDBC
  - Not comfortable
  - Error prone
  - Degrease of code readability
  - Mixing RDB an OOP approach

ORM in Java EE

Until J2EE 1.4 there were special Entity Beans

- Set of interfaces and classes
- A LOT of config files
- Complicated and hard to use

- but …
- Persistence fully managed by a container
- Transactions solved
- Load balancing

Is there an option?

… yes

Hibernate framework

- came with a simple to use XML mapping from a POJO to RDB

JPA

- a modern version of HIBERNATE, does the same using annotations

The core idea

- Use POJO only
- Implicit mapping using variable names
- Annotate exceptions only
- Container takes care of resource injection
- Objects use inheritance

Entity Manager

- Takes care of work with entities, connection.
- Can be obtained using resource injection

```
@PersistenceContext(unitName="school_persistence_unit")
    private EntityManager em;

//…follows work with em
```

- or using factory

```
javax.persistence.EntityManager em;
        em = javax.persistence.
       Persistence.createEntityManagerFactory(
   " school_persistence_unit ").createEntityManager();

//…follows work with em
```

Entity manager - configuration

- Configuration is stored in file persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence s: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">
  <persistence-unit name="school_persistence_unit" transaction-type="JTA">
    <provider>oracle.toplink.essentials.PersistenceProvider</provider>
    <jta-data-source>jdbc/school</jta-data-source>
    <properties>
      <property name="toplink.ddl-generation" value="drop-and-create-tables"/
    </properties>
  </persistence-unit>
</persistence>
```

Name of PU will be used for DI

provider = JPA implementation (several exist)

JNDI name

strategy for table generation

- More PU can be defined in one file
- Provider – implementation
  - Hibernate, Oracle Toplink, OpenJPA

Entity Manager

persist(Object entity)

saves entity into DB

refresh(Object entity)

reloads the entity from DB

merge(T entity)

merges / connects an object to a persistent context

remove(Object entity)

deletes from DB

find(Class<T>entityClass, Object primaryKey)

finds an entity of T type using primary key

flush()

makes sure to write to DB

create*Query(String sql, …)

query to DB

Code example

POJO class

```
@Entity
public class TeacherEntity implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String firstName;
    private String lastName;
    @OneToMany(mappedBy = "supervisor")
    private Collection<StudentEntity> students;

    public Long getId() {  return id;  }
    public void setId(Long id) {  this.id = id; }
    public String getFirstName() {  return firstName; }
    public void setFirstName(String firstName) {     this.firstName = firstName;  }
    public String getLastName() {  return lastName;   }
    public void setLastName(String lastName) {  this.lastName = lastName;   }
    public Collection<StudentEntity> getStudents() {
        return students;
    }
}
```

Consider this POJO an entity

An Entity must have a primary key

Properties should have getters and setters

Collection of related entities

One teacher may supervise multiple students

Code example cont.

Session EJB

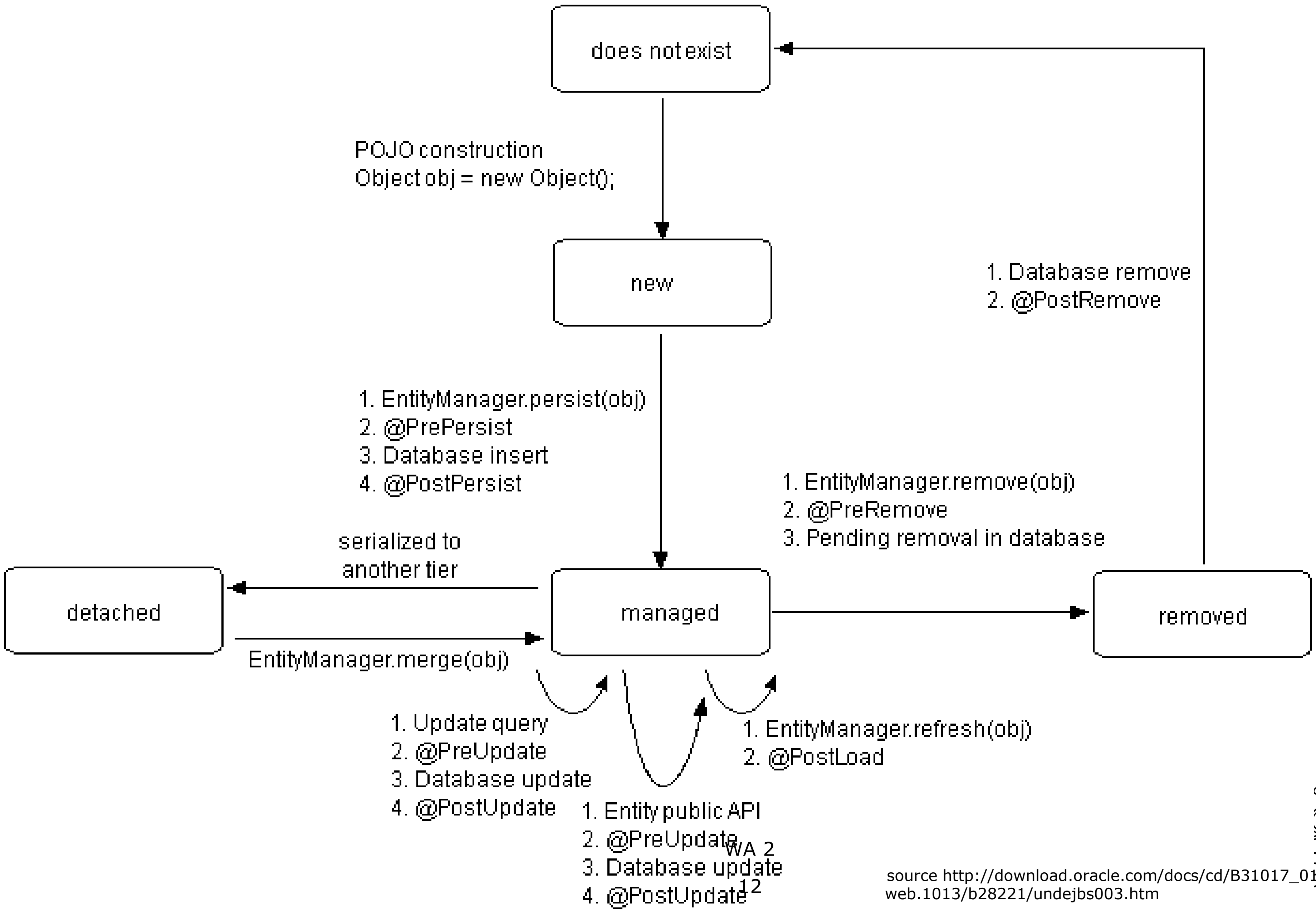
```
@Stateless
public class SchoolSessionBean implements SchoolSessionBeanRemote,
SchoolSessionBeanLocal {

    @PersistenceContext(unitName = "school_persistence_unit")
    private EntityManager em;

    public StudentEntity addStudent(final String firstName, final String lastName) {

        StudentEntity newStudent = new StudentEntity();
        newStudent.setFirstName(firstName);
        newStudent.setLastName(lastName);

        em.persist(newStudent);
        return newStudent;
    }
```

DI of Entity Manger instance

Let's create a new instance of Student

Let it persist into DB

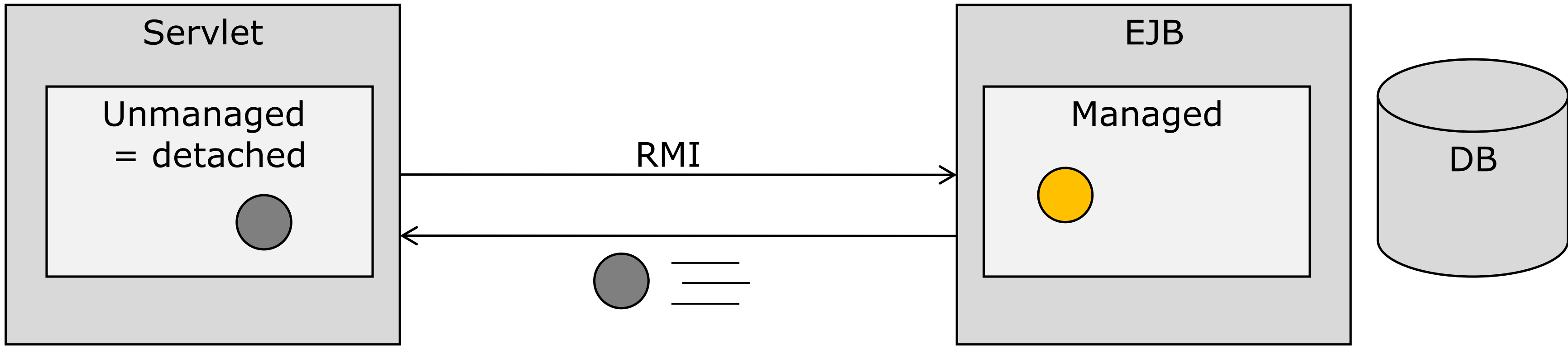
Entity lifecycle



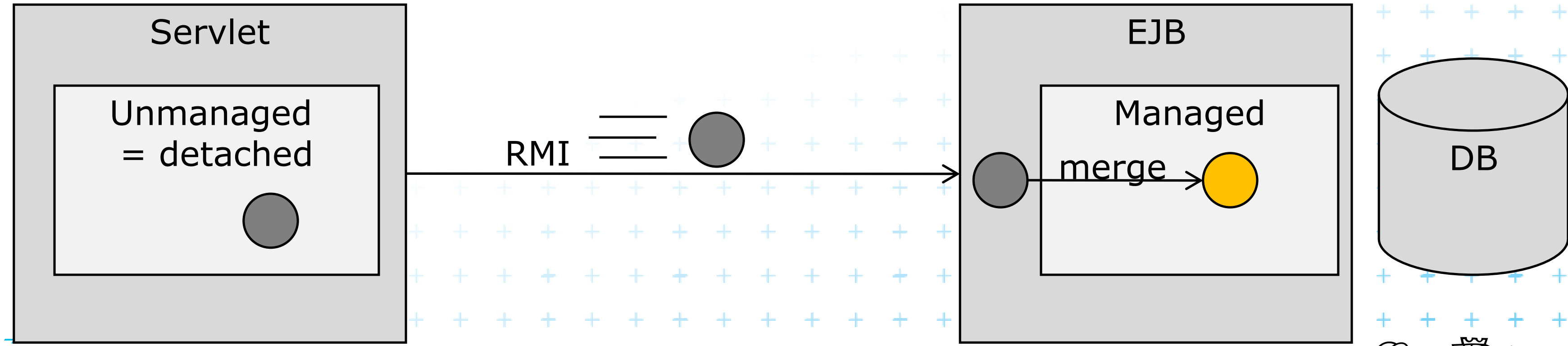
source http://download.oracle.com/docs/cd/B31017_01/web.1013/b28221/undejbs003.htm

merge

- typical example



- Servlet changes some property, then merge



Code example

```
@Stateless
public class SchoolSessionBean implements SchoolSessionBeanRemote,
SchoolSessionBeanLocal {

   @PersistenceContext(unitName = "school_persistence_unit")
   private EntityManager em;

   public void setSupervisor(final StudentEntity student, final TeacherEntity supervisor) {
      student.setSupervisor(supervisor);
      em.merge(student);
   }
 }
```

Entity

- must have empty *public* or *protected* constructor
- must have annotation of *javax.persistence.Entity*
- must not be *final,* neither its methods and properties
  - due to the fact, that container will yet extend it internally
- if it will be serialized, for example due to RMI call, must implement *Serializable*
- must have a primary key @Id

- Annotated can be either properties or methods, not both!

Either

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

Or

```
@Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  public Long getId() {
     return id;
  }
```

Entity relations

Relations like in RDB

- 1:N, 1:1, M:N

Relation *Owner* is that Entity, that holds the foreigh key

- For example: 1:N

  Owner

- by M:N any of the Entities can be the owner

Unidirectional and bidirectional relations

- Unidirectional
  - onle one party (owner) knows about the peer

```
@ManyToOne ()
```

- Bidirectional
  - both parties know about the peer
  - one of them is the owner

```
@ManyToOne ()
```

  - the other one is informed

```
@OneToMany(mappedBy = "supervisor")
```

Example

```
public void setSupervisor(final StudentEntity
student, final TeacherEntity supervisor) {
        student.setSupervisor(supervisor);
        em.merge(student);
//        Attention, this does not work, must call
//        supervisor.getStudents().add(student);
//        em.merge(supervisor);
    }
```

Example @ManyToMany

Subject

```
@Entity
public class SubjectEntity implements Serializable {
   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;
   private String name;

   @ManyToMany
   private Collection<TeacherEntity> teachers;

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

…

   public Collection<TeacherEntity> getTeachers() {
         return teachers;   }
   public void setTeachers(Collection<TeacherEntity>
teachers) {    this.teachers = teachers;   }
}
```

Owner

Teacher

```
@Entity
public class TeacherEntity implements Serializable {
   @ManyToMany(mappedBy = "teachers")
   private List<SubjectEntity> subjectEntitys;
   private static final long serialVersionUID = 1L;
   @Id
   @GeneratedValue(strategy = GenerationType.AUTO)
   private Long id;

   @ManyToMany(mappedBy = "teachers")
   private List<SubjectEntity> subjectEntitys;

   public Long getId() { return id; }

   public void setId(Long id) {  this.id = id; }
…
   public Collection<StudentEntity> getStudents()
      return students;  }
   public List<SubjectEntity> getSubjectEntitys() {
        return subjectEntitys;   }
}
```

Informed

```
public void LinkTeacherSubject (SubjectEntity subject, TeacherEntity teacher){
      subject.getTeachers().add(teacher);
      em.merge(subject);
      // but not:
      // teacher.getSubjectEntitys().add(subject);
      // em.merge(teacher);
 }
```

relation

Anotace - details

- Many annotation do have additional parameters, see http://www.oracle.com/technology/products/ias/toplink/jpa/resources/toplink-jpa-annotations.html
- @OneToMany(cascade="ALL")
  - ALL - all cascading operations performed on the source entity are cascaded to the target of the association.
  - MERGE - if the source entity is merged, the merge is cascaded to the target of the association.
  - PERSIST - if the source entity is persisted, the persist is cascaded to the target of the association.
  - REFRESH - if the source entity is refreshed, the refresh is cascaded to the target of the association.
  - REMOVE - if the source entity is removed, the target of the association is also removed.
- @JoinColumn
  - name, referendedColumnName, unique, nullable, insertable, columnDefinition, table
- @Transient
  - items that should not be written to DB

Annotations cont.

- @GeneratedValue(strategy=
  - Sequence
  - AUTO – JPA will choose a strategy atumaticaly
  - TABLE – an extra dedicated table will be used to ensure unique ID
  - IDENTITY – DB will ensure unique identifier

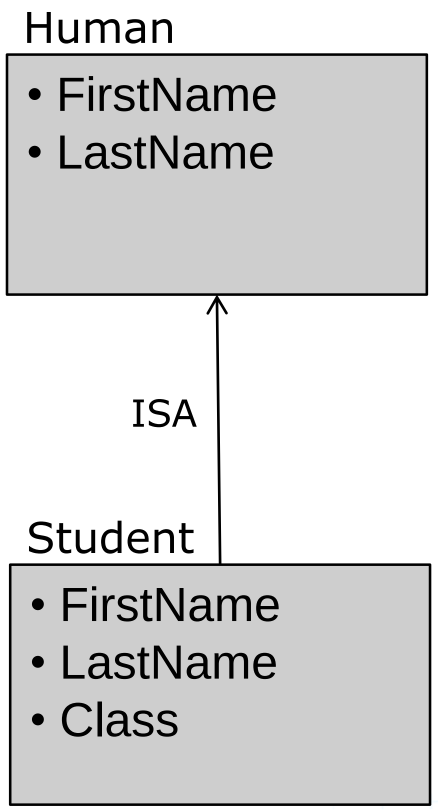
- @LOB
  - property will be mapped to BLOB datatype

Reverse generation of StudentEntity

```
@Entity
@Table(name = "STUDENTENTITY")
@NamedQueries({
   @NamedQuery(name = "Studententity.findAll", query = "SELECT s FROM Studententity s"),
   @NamedQuery(name = "Studententity.findById", query = "SELECT s FROM Studententity s WHERE s.id = :id"),
   @NamedQuery(name = "Studententity.findByLastname", query = "SELECT s FROM Studententity s WHERE s.lastname =
:lastname"),
   @NamedQuery(name = "Studententity.findByFirstname", query = "SELECT s FROM Studententity s WHERE s.firstname =
:firstname")})
public class Studententity implements Serializable {
   private static final long serialVersionUID = 1L;
   @Id
   @Basic(optional = false)
   @Column(name = "ID")
   private Long id;
   @Column(name = "LASTNAME")
   private String lastname;
   @Column(name = "FIRSTNAME")
   private String firstname;
   @JoinColumn(name = "SUPERVISOR_ID", referencedColumnName = "ID")
   @ManyToOne
   private Teacherentity supervisorId;

   public Studententity() {    }

   public Studententity(Long id) {   this.id = id;   }

   public Long getId() {   return id;    }

…

}
```

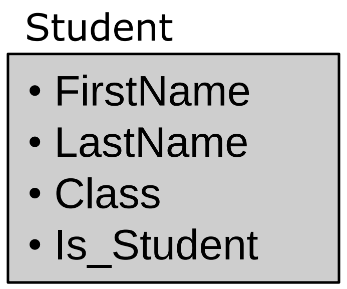
Inheritance = ISA relations

Logical level

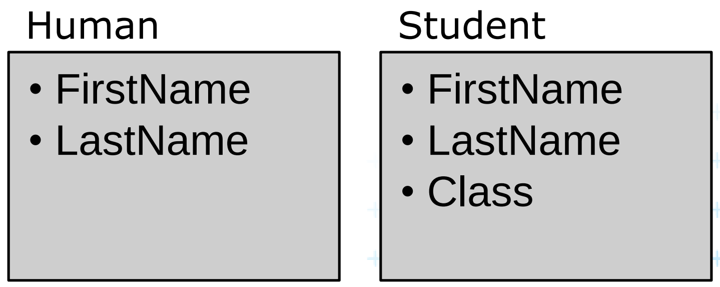


Physical level

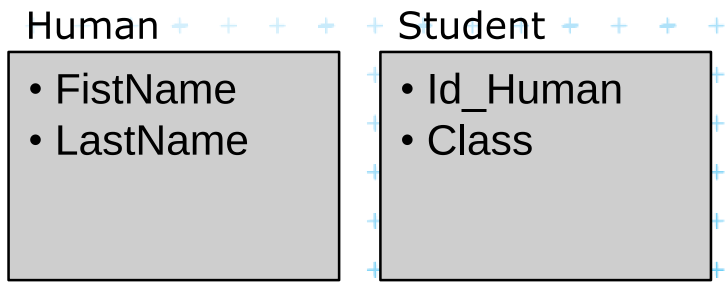
SINGLE_TABLE
all is in one table,
distinguishing is one column



TABLE_PER_CLASS
every entity has its own
table with a full set of
attributes



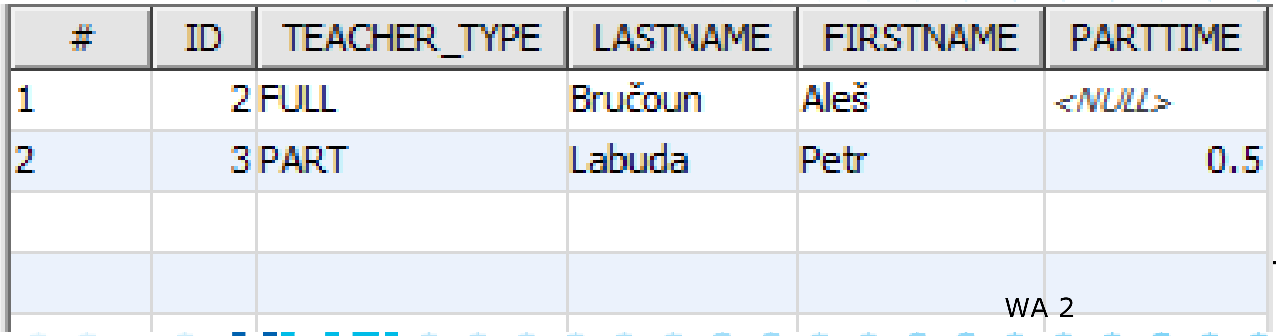
JOINED
main table has the basic set
of attributes, others have
the extra ones. The others
are weak entities.



Inheritance – implementation in JPA

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="TEACHER_TYPE", discriminatorType=DiscriminatorType.STRING,
length=4)
@DiscriminatorValue(value="FULL")
public class TeacherEntity implements Serializable {
…
}
```

```
@Entity
@DiscriminatorValue(value="PART")
public class PartTimeTeacherEntity extends TeacherEntity implements Serializable {
…
}
```



#	ID	TEACHER_TYPE	LASTNAME	FIRSTNAME	PARTTIME
1	2	FULL	Bručoun	Aleš	<NULL>
2	3	PART	Labuda	Petr	0.5

Query language

- Language similar to SQL
- Java Persistence query language

```
public Collection<TeacherEntity> findTeacherByLastName(final String lastName) {
        return em.createQuery(
                "SELECT t FROM TeacherEntity t WHERE t.lastName LIKE :paramName")
                .setParameter("paramName", lastName)
                .setMaxResults(10).getResultList();
    }
```

- Language is dialect independent.

Named Query

using annotation

```
@NamedQueries({
   @NamedQuery(name = "Teacherentity.findAll", query = "SELECT t FROM Teacherentity t"),
   @NamedQuery(name = "Teacherentity.findById", query = "SELECT t FROM Teacherentity t WHERE t.id = :id"),
   @NamedQuery(name = "Teacherentity.findByLastname", query = "SELECT t FROM Teacherentity t WHERE
t.lastname = :lastname"),
   @NamedQuery(name = "Teacherentity.findByFirstname", query = "SELECT t FROM Teacherentity t WHERE
t.firstname = :firstname")})
```

usage

```
 public Collection<TeacherEntity> findTeacherByFirstname(final String firstName){
       return em.createNamedQuery("Teacherentity.findByFirstname")
              .setParameter("firstname", firstName)
              .getResultList();
   }
```