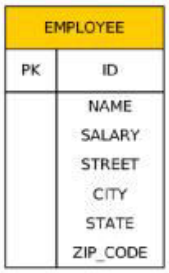
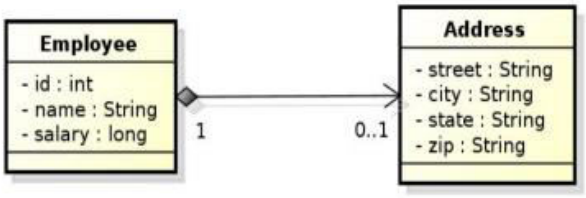
Pokročilá témata JPA, Spring

Pokročilé JPA

Embedded objekty

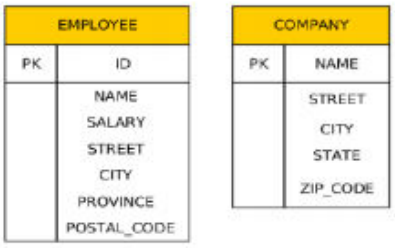
Embedded Objects

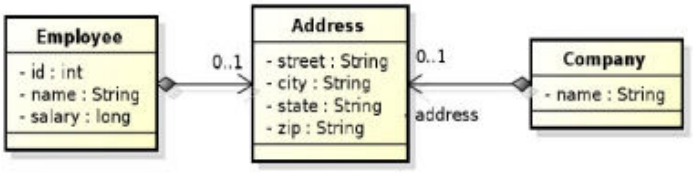




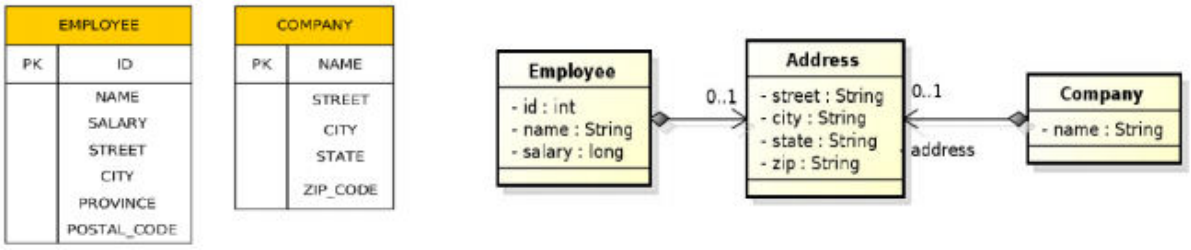

```
@Embeddable
@Access(AccessType.FIELD)
public class Address {
   private String street;
   private String city;
   private String state;
   @Column(name="ZIP_CODE")
   private String zip;
}
```

```
@Entity
public class Employee {
   @Id private int id;
   private String name;
   private long salary;
   @Embedded
    private Address address;
}
```

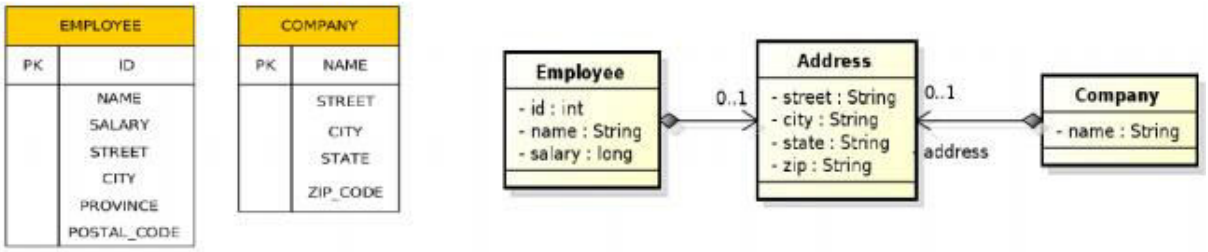





```
@Embeddable
@Access(AccessType.FIELD)
public class Address {
   private String street;
   private String city;
   private String state;
   @Column(name="ZIP_CODE")
   private String zip;
}
```




```
@Entity
public class Employee {
    @Id private int id;
    private String name;
    private long salary;
    @Embedded
    @AttributeOverrides({
        @AttributeOverride(name="state", column=@Column(name="PROVINCE")),
        @AttributeOverride(name="zip", column=@Column(name="POSTAL_CODE"))
    })
    private Address address;
}
```




```
@Entity
public class Company {
    @Id private String name;
    @Embedded
    private Address address;
}
```

Mapování legacy DB

One entity to many tables

- @SecondaryTable, @Column(table=…)

```
@SecondaryTables({
  @SecondaryTable(name="ADDRESS")
})
public class Person {

  @Id
  private Long id;

  @Column(table="ADDRESS")
  private String city;

  // getters + setters
}
```

```
PERSON
================================
ID bigint PRIMARY KEY NOT NULL
HASNAME varchar(255)
```

```
ADDRESS
================================
ID bigint
     PRIMARY KEY NOT NULL
CITY varchar(255)
FOREIGN KEY (id)
     REFERENCES person (id)
```

Multiple entities to one table

- @Embedded, @EmbeddedId, @Embeddable

```
@Entity
public class Person {
  @Id
  private Long id;
  private String hasName;

  @Embedded
  private Birth birth;
  // getters + setters
}
```

```
@Embeddable
public class Birth {
  private String hasPlace;

  @Temporal(value=TemporalType.DATE)
  private Date hasDateOfBirth;
  // getters + setters
}
```

```
PERSON
================================
ID bigint PRIMARY KEY NOT NULL
HASNAME varchar(255)
HASDATEOFBIRTH date
HASPLACE varchar(255)
```

Cascade-persist

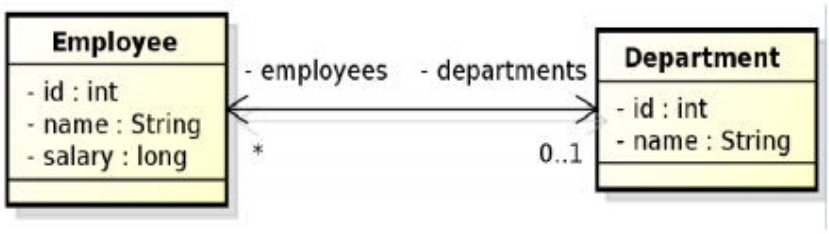
```
@Entity
public class Employee {
    // …
    @ManyToOne(cascade=cascadeType.PERSIST)
    Address address;
    // …
}
```

```
Employee emp = new Employee();
emp.setId(2);
emp.setName("Rob");
Address addr = new Address();
addr.setStreet("164 Brown Deer Road");
addr.steCity("Milwaukee");
addr.setState("WI");
emp.setAddress(addr);
em.persist(addr);
em.persist(emp);
```

List of operations supporting cascading:

- cascadeType.ALL
- cascadeType.DETACH
- cascadeType.MERGE
- cascadeType.PERSIST
- cascadeType.REFRESH
- cascadeType.REMOVE

Persisting bidirectional relationship




```
…
Department dept = em.find(Deprtment.class, 101);
Employee emp = new Employee();
emp.setId(2);
emp.setName("Rob");
emp.setSalary(25000);
dept.employees.add(emp);   // @ManyToOne(cascade=cascadeType.PERSIST)
em.persist(dept);

!!! emp.departments still doesn't contain dept !!!

em.refresh(dept);

!!! emp.departments does contain dept  now !!!
```

Collection mapping, Compound/Shared keys, Beyond JPA

viz https://cw.fel.cvut.cz/wiki/_media/courses/b6b33ear/lectures/lecture-08-advancedtopics-s.pdf

Queries

- JPQL (Java Persistence Query Language)
- Native queries (SQL)
- Criteria API (queries = Java objects, not strings)

JPQL

- velmi podobné SQL, jsou to stringy
- možnost dynamické tvorby dotazů
  - vytvoření pomocí String → … return em.createQuery(query,x.class).getSingleResult();
- parametry dotazu – positional (?1, ?2,…) x named (:dept, :name,…)
  - createNamedQuery(…).setParameter(„name“, name).getSingleResult();
  - ……..setParameter(1, dept)…….
- @NamedQuery(name="name“, query="…“)

JPQL vs. SQL

- v SQL jsou dotazy vykonávány nad tabulkami databáze, u JPQL nad objekty aplikace (entity)
- v JPQL stále stejné dotazy (odstínění od konkrétní databáze), dotaz se pak převede

JPQL vs. Criteria API

- query u JPQL = string, query u CAPI = instance objektů reprezentující elementy dotazu
- u CAPI lze chyby detekovat dříve (již během kompilace)
- JPQL může být preferováno u statických string dotazů, CAPI u dynamických dotazů