# Java Persistence API

# JSR-220, EJB 3.0

Molnár István, Zsemlye Tamás
istvan.molnar@sun.hu
tamas.zsemlye@sun.com

# Témák

- Háttér és kulcsfogalmak
- Entitások és az Entitás Manager API
- Relációk
- Lekérdezések
- Objektum/Relációs megfeleltetés
- Java perzisztencia Java EE és Java SE környezetben
- Összefoglaló

# Java Persistence API

- JSR-220 (EJB 3.0) specifikáció része
- Az Entity Beanek egyszerűsítésének indult
- POJO perzisztencia technológiává fejlődött
  - Gazdag modellezési lehetőségek, öröklés, polimorfizmus
  - Szabványos objektum/relációs megfeleltetés
  - Erős lekérdezési képesség
- A specifikáció a közösség kérésére bővült ki
  - a Java EE rendszerek perzisztencia technológiájává
  - a Java SE környezet technológiájává
  - cserélhető perzisztencia implementációk támogatásával

# Java Persistence API: kulcsfogalmak

- Entitások
- Perzisztencia egység (Persistence Unit)
- Perzisztencia környezet (Persistence Context)

# Entitások

- Egyszerű java osztályok (POJO-k)
  - new() operátorral jönnek létre
  - nincs kötelező interface amit implementálni kell
  - támogatják az öröklést és a polimorfikusságot
- Konténeren kívül is használhatóak
  - Szerializálhatóak, lecsatolt objektumként is használhatóak
- Lekérdezés Java Perzisztencia lekérdező nyelvvel
- Futásidőben az Entitás Manager API-n keresztül érhetők el

# Perzisztencia Unit

- A csomagolás és telepítés alapegysége
- Menedzselt osztályok halmaza (entitások és kapcsolódó osztályok)
- Objektum/relációs leképzési információk
  - Java programnyelvből a relációs adatbázis nézete
  - Java annotációk és/vagy XML
- Definiálja a hatáskörét a:
  - lekérdezéseknek
  - az entitás relációknak
- Konfigurációs információ a perzisztence provider számára: persistence.xml

# Perzisztencia Context

- Futás idejű alkalmazás környezet
- A menedzselt entitás példányok halmaza, amelyek egy perzisztencia unithoz tartoznak
  - Entitások, amelyeket az adatbázisból olvastak be
  - Entitások, amelyeket az adatbázisba fognak kiírni
    - Az éppen most perzisztált entitásokat is beleértve
- Perzisztencia context élettartama lehet:
  - Tranzakció alapú: egy tranzakcióra kiterjedő
  - Bővített: több egymás utáni tranzakción átívelő
  - A context élettartamát az alkalmazás vagy a konténer menedzselheti

# Témák

- Háttér és kulcsfogalmak
- Entitások és az Entitás Manager API
- Relációk
- Lekérdezések
- Objektum/Relációs megfeleltetés
- Java perzisztencia Java EE és Java SE környezetben
- Összefoglaló

# Entitás osztály

- A @Entity annotációval vagy XML leíróval vannak kijelölve
- Elsődleges kulcsuk vagy van (vagy örökölniük kell)
- Lehetnek konkrét vagy absztrakt osztályok
- Leszármazhatnak másik entitásból, „mapped” ősosztályból vagy egyszerű java osztályból
- Lehet perzisztens és nem perzisztens állapotuk
- Szerializálhatóak lehetnek

# Entitás állapot

- Perzisztens állapot:
  - lehetnek egyszerű típusok: primitív/wrapper, szerializálható, enum, byte[], char[], …
    - Annotáció: @Basic
  - lehetnek összetett, beágyazott objektumok: pl. Cím
    - Annotáció: @Embedded
  - Elsődleges kulcs kötelező
    - Annotáció: @Id, @EmbeddedId, @IdClass
  - Entitások állapota perzisztens kivéve a:
    - @Transient vagy transient attribútumokat

# Entitás állapot

- Mind a mező, mind a property alapú elérés támogatott
  - A perzisztencia szolgáltató használja az elérési típusokat
  - Egy entitás hierarchiára csak egyféle elérés lehetséges
    - Amelyet az annotáció elhelyezése vagy az XML határoz meg
- Mind a mohó (EAGER) mind a lusta (LAZY) betöltés támogatott
  - a LAZY opcionális javaslat; az EAGER kötelező
    - Az EAGER az alapértelmezett az egy-több és a több-több relációk kivételével

# Példa: Entitás

```
@Entity
public class Customer implements Serializable {
   @Id protected Long id;
   protected String name;

   protected Address address;
   protected PreferredStatus status;

   @Transient protected int orderCount;

   public Customer() {}

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

   public String getName() {return name;}
   public void setName(String name) {this.name = name;}

   ...
}
```

# Példa: beágyazható osztály

```
@Embeddable public class Address {
   private String street;
   private String city;
   private String zip;

   public Address() {}

   public String getStreet() {
      return street;
   }
```

…

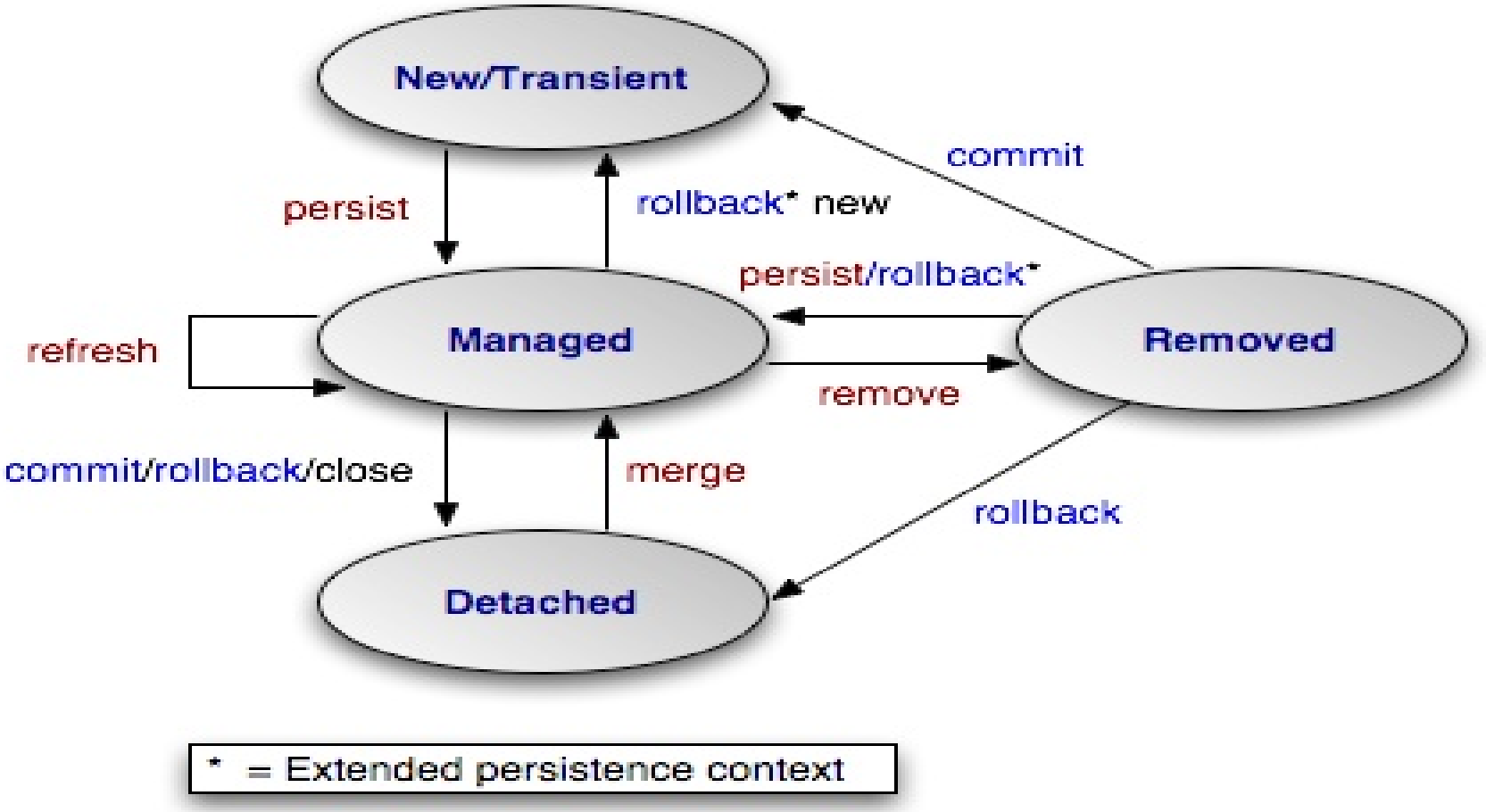
# Entitások identitása

- Minden entitásnak van egy perzisztens identitása
  - Az elsődleges kulcs a megfeleltetés az adatbázisban
- Megfeleltethető egy egyszerű típusnak
  - Annotáció:
    - @Id – egy darab mező vagy property esetén
    - @GeneratedValues – az azonosító automatikus generálásához
- Lehet egy felhasználó által definiált típus
  - Annotáció:
    - @EmbeddedId – egy mező vagy property esetén
    - @IdClass – ha több @Id mező is van az entitás osztályban

# Entitás manager API

- Entitások életciklusát és a perzisztencia contextust kezeli
- Életciklus műveletek:
  - persist, remove, refresh, merge
- Finder műveletek
  - find, getReference
- Query objektumok előállítását segítő metódusok
  - createNamedQuery, createQuery, createNativeQuery
- Egyéb műveletek
  - flush, clear, contains, lock, ...

# Entitások életciklusa



# Entitás életciklus műveletek

- new
  - Új entitás jön létre
  - Entitás még nem menedzselt és nem is perzisztens
- persist
  - Az entitás menedzselt állapotba kerül
  - Az entitás a tranzakció commitjával kerül be az adatbázisba
- remove
  - Törli a perzisztens entitást
  - Tranzakció commit esetén az adatbázisból törlődik
- refresh
  - Az entitás állapota frissül az adatbázisból

# Entitás életciklus (folytatás)

- Az entitások lekapcsolódnak (detach) a perzisztencia contextből
  - Amikor a perzisztencia context befejeződik vagy törlődik
  - Amikor szerializációval eltávolítják őket a perzisztencia contextből
- merge
  - Egy lekapcsolódott entitás állapotát átmásolja egy menedzselt entitásba
  - Visszatér a menedzselt példánnyal, melynek perzisztens állapota megegyezik az eredetivel

# Példa: entitás menedzser használata

```
@Stateless public class OrderManagementBean
    implements OrderManagement {

    ...
    @PersistenceContext EntityManager em;
    ...

    public Order addNewOrder(Customer customer, Product
  product) {
        Order order = new Order(product);
        customer.addOrder(order);

        em.persist(order);

        return order;
    }

}
```

# Példa: cascade persist

```
@Entity
public class Customer {
    @Id protected Long id;

    ...

    @OneToMany(cascade=PERSIST)
    protected Set<Order> orders = new HashSet();
}

...

public Order addNewOrder(Customer customer, Product product) {
    Order order = new Order(product);
    customer.addOrder(order);
    return order;
}
```

# Példa: cascade remove

```
@Entity
public class Order {
    @Id protected Long id;

    ...

    @OneToMany(cascade=PERSIST, REMOVE)
    protected Set<LineItem> lineItems = new HashSet();
}

...

@PersistenceContext EntityManager em;
...

public void deleteOrder(Long orderId) {
    Order order = em.find(Order.class, orderId);
    em.remove(order);
}
```

# Példa: cascade merge

```
@Entity
public class Order {
    @Id protected Long id;

    ...

    @OneToMany(cascade=PERSIST, REMOVE, MERGE)
    protected Set<LineItem> lineItems = new HashSet();
}

...

@PersistenceContext EntityManager em;
...

public Order updateOrder(Order changedOrder) {
    return em.merge(changedOrder);
}
```

# Témák

- Háttér és kulcsfogalmak
- Entitások és az Entitás Manager API
- Relációk
- Lekérdezések
- Objektum/Relációs megfeleltetés
- Java perzisztencia Java EE és Java SE környezetben
- Összefoglaló

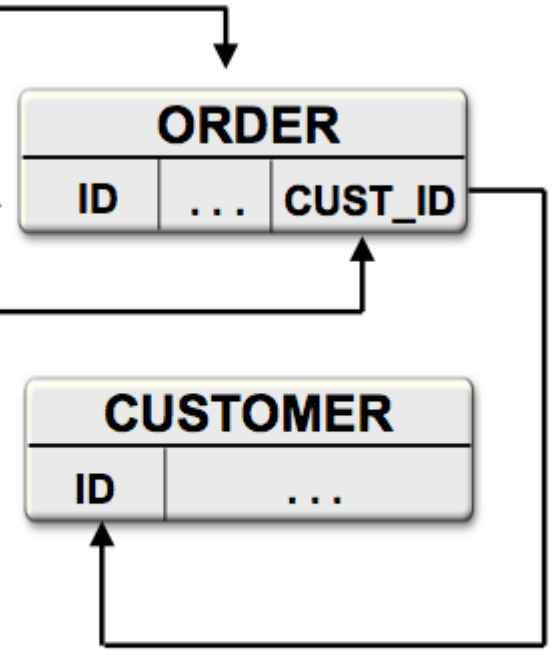
# Entitások relációja

- 1-1, 1-több, több-1, több-több entitások között
  - Támogatás Collection, Set, List, Map típusokra
  - Annotáció:
    - @OneToOne, @OneToMany, @ManyToOne, @ManyToMany
- Kétirányú és egyirányú kapcsolat
  - Kétirányú kapcsolatok kezelése az alkalmazás feladata
  - Kétirányú esetben van tulajdonos és inverz oldal
    - Tulajdonos oldal megfelel az idegen kulcsnak
    - Inverz oldalnak hivatkoznia kell a tulajdonos oldalra (mapped-by)

# Több-egy

```
@Entity
public class Order {
    @Id
    int id;

    ...
    @ManyToOne
    Customer cust;
}
```



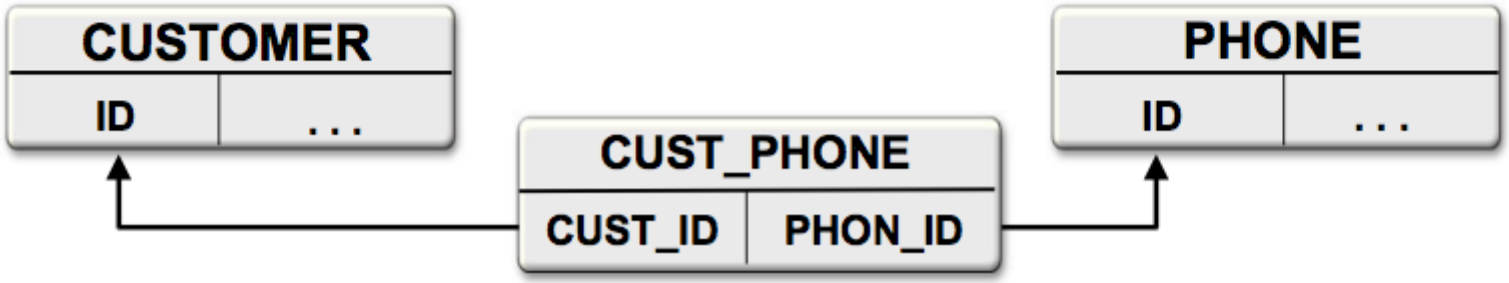
# Több-több

```
@Entity
public class Customer {
   @Id
   int id;
   ...
   @ManyToMany
   Collection<Phone> phones;
}
```

```
@Entity
public class Phone {
   @Id
   int id;
   ...
   @ManyToMany(mappedBy="phones"
   Collection<Customer> custs;
}
```

# Több-több

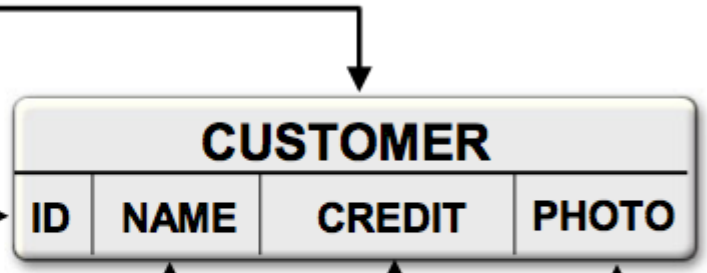
```
@Entity
public class Customer {
  ...
  @ManyToMany
  @JoinTable(table="CUST_PHONE"),
    joinColumns=@JoinColumn(name="CUST_ID"),
    inverseJoinColumns=@JoinColumn(name="PHON_ID"))
  Collection<Phone> phones;
}
```



# Beágyazott

```
@Entity
public class Customer {
   @Id
   int id;
   @Embedded
   CustomerInfo info;
}

@Embeddable
public class CustomerInfo {
   String name;
   int credit;
   @Lob
   Image photo;
}
```



# Példa: reláció

```
@Entity public class Customer {
     @Id protected Long id;
     ...
     @OneToMany protected Set<Order> orders = new HashSet();
     @ManyToOne protected SalesRep rep;
     ...
     public Set<Order> getOrders() {return orders;}
     public SalesRep getSalesRep() {return rep;}
     public void setSalesRep(SalesRep rep) {this.rep = rep;}
}

@Entity public class SalesRep {
     @Id protected Long id;
     ...
     @OneToMany(mappedBy="rep")
     protected Set<Customer> customers = new HashSet();
     ...
     public Set<Customer> getCustomers() {return customers;}
     public void addCustomer(Customer customer) {
        getCustomers().add(customer);
        customer.setSalesRep(this);}
}
```

# Öröklés

- Entitásokból lehet leszármaztatni
  - Más entitások
    - Konkrét vagy absztrakt
  - Mapped ősosztály
    - közös entitás állapot
  - Közönséges (nem entitás) Java osztály
    - viselkedést definiál, de perzisztens állapotot nem

# Példa: mapped ősosztály

```
@MappedSuperclass public class Person {
    @Id protected Long id;
    protected String name;
    @Embedded protected Address address;
}

@Entity public class Customer extends Person {
    @Transient protected int orderCount;
    @OneToMany
    protected Set<Order> orders = new HashSet();
}

@Entity public class Employee extends Person {
    @ManyToOne
    protected Department dept;
}
```

# Példa: absztrakt entitás

```
@Entity public abstract class Person {
    @Id protected Long id;
    protected String name;
    @Embedded protected Address address;
}

@Entity public class Customer extends Person {
    @Transient protected int orderCount;
    @OneToMany
    protected Set<Order> orders = new HashSet();
}

@Entity public class Employee extends Person {
    @ManyToOne
    protected Department dept;
}
```

# Témák

- Háttér és kulcsfogalmak
- Entitások és az Entitás Manager API
- Relációk
- Lekérdezések
- Objektum/Relációs megfeleltetés
- Java perzisztencia Java EE és Java SE környezetben
- Összefoglaló

# Java Persistemce Query Language

- Az EJB QL nyelv bővítése
  - az EJB QL-hez hasonlóan és is SQL szerű nyelv
- Plusz funkcionalitás
  - projekciós lista (SELECT)
  - explicit JOINS
    - inner, outer, fetch
  - subquery
  - GROUP BY, HAVING
  - EXISTS, ALL, SOME/ANY
  - UPDATE, DELETE
  - további függvények

# Példa: projekció

```
SELECT e.name, d.name
FROM Employee e JOIN e.department d
WHERE e.status = 'FULLTIME'
```

```
SELECT new com.example.EmployeeInfo(e.id, e.name,
e.salary, e.status, d.name)
FROM Employee e JOIN e.department d
WHERE e.address.state = 'CA'
```

# Példa: subquery

```
SELECT DISTINCT emp
FROM Employee emp
WHERE EXISTS (
    SELECT mgr
    FROM Manager mgr
    WHERE emp.manager = mgr
        AND emp.salary > mgr.salary)
```

# Példa: inner és outer join

```
SELECT DISTINCT o
FROM Order o JOIN o.lineItems l JOIN l.product p
WHERE p.productType = 'shoes'
```

```
SELECT DISTINCT c
FROM Customer c JOIN FETCH c.orders
WHERE c.address.city = 'San Francisco'
```

```
SELECT DISTINCT c
FROM Customer c LEFT JOIN FETCH c.orders
WHERE c.address.city = 'San Francisco'
```

# Példa: update, delete

```
UPDATE Employee e
SET e.salary = e.salary * 1.1
WHERE e.department.name = 'Engineering'
```

```
DELETE
FROM Customer c
WHERE c.status = 'inactive'
     AND c.orders IS EMPTY
    AND c.balance = 0
```

# Query

- Statikus lekérdezések
  - Java metadata vagy XML segítségével definiálhatóak
    - Annotációk: @NamedQuery, @NamedNativeQuery
- Dinamikus lekérdezések
  - A query sztringet futás időben kell létrehozni
- Java persistence query language vagy natív SQL használható
- Névvel vagy pozícióval lehet a paraméterekre hivatkozni
- A Query objektum metódusaival állíthatóak
  - maximális elemszám, lapozás, flush mód

# Példa: dinamikus lekérdezés

```
@PersistenceContext EntityManager em;

...

public List findByZipcode(String personType, int zip) {
    return em.createQuery (
        "SELECT p FROM " + personType + "p WHERE p.address.zip
            = :zipcode")
        .setParameter("zipcode", zip)
        .setMaxResults(20)
        .getResultList();
}
```

# Példa: statikus lekérdezés

```
@NamedQuery(name="customerFindByZipcode", query =
    "SELECT c FROM Customer c WHERE c.address.zipcode = :zip")
@Entity public class Customer {...}

...

public List findCustomerByZipcode(int zipcode) {
    return em.createNamedQuery ("customerFindByZipcode")
       .setParameter("zip", zipcode)
       .setMaxResults(20)
       .getResultList();
}

...
```

# Témák

- Háttér és kulcsfogalmak
- Entitások és az Entitás Manager API
- Relációk
- Lekérdezések
- Objektum/Relációs megfeleltetés
- Java perzisztencia Java EE és Java SE környezetben
- Összefoglaló

# Objektum/Relációs megfeleltetés

- Könnyen használható mapping lehetőség Java fejlesztők számára, a domain modellből a relációs adatbázisba
- Mapping annotációk és/vagy XML metaadatok
  - Alapértelmezés a könnyű és gyors prototípus készítéshez
- Mapping annotációk:
  - @Table, @SecondaryTable, @JoinTable, …
  - @Column, @JoinColumn, @PrimaryKeyJoinColumn, …
  - @Inheritance, @DiscriminatorColumn, ...

# Egyszerű Mapping

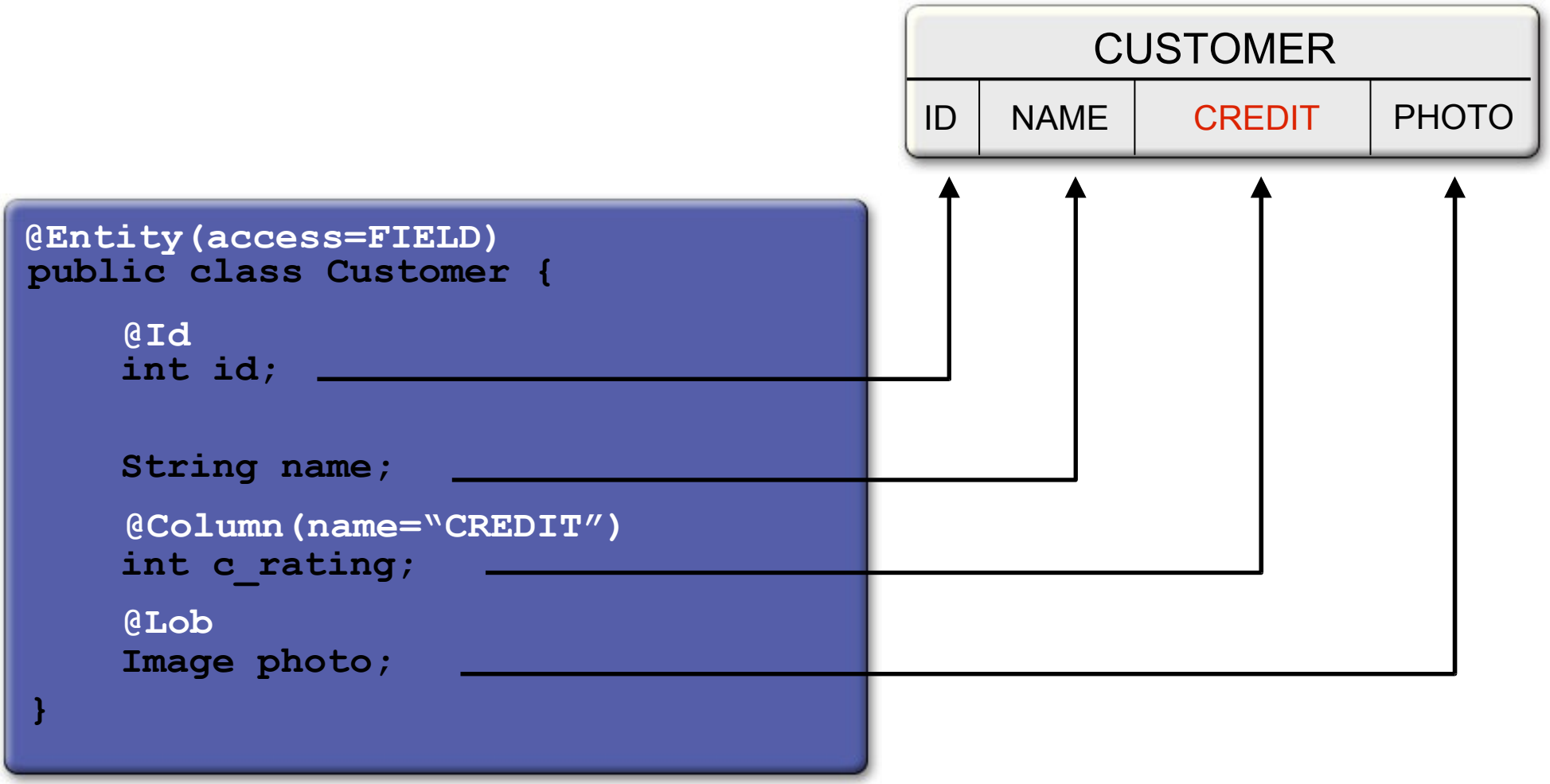

```
@Entity(access=FIELD)
public class Customer {
    @Id
    int id;

    String name;

    int c_rating;

    @Lob
    Image photo;
}
```

# Egyszerű Mapping



# O/R Mapping Példák

```
@Entity
@Table(name="EMPLOYEE", schema="EMPLOYEE_SCHEMA")
uniqueConstraints=
{@UniqueConstraint(columnNames={"EMP_ID", "EMP_NAME"})}
public class EMPLOYEE {
    ...
    @Column(name="NAME", nullable=false, length=30)
    public String getName() { return name; }
}
```

---

```
@Version
@Column("OPTLOCK")
protected int getVersionNum() { return versionNum; }
```

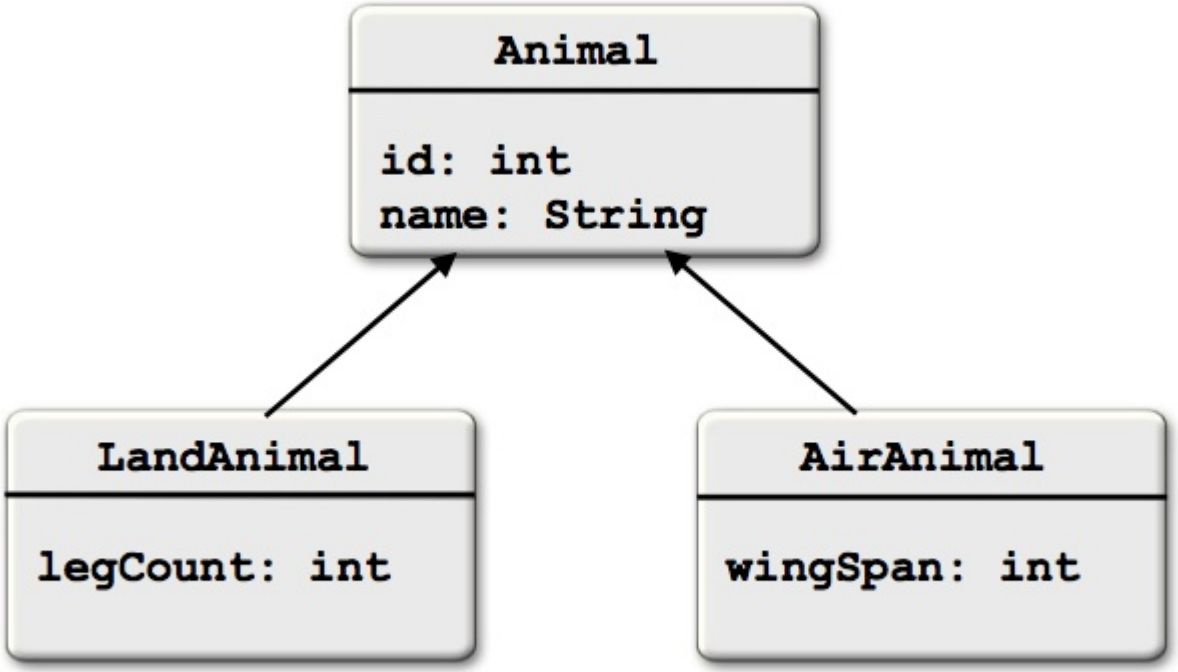
---

```
@ManyToOne
@JoinColumn(name="ADDR_ID")
public Address getAddress() { return address; }
```

# Leszámaztatás

- Entitások származhatnak
  - Más entitásokból (akár konkrét akár absztrakt)
  - Nem entitásokból
  - „mapped ősosztályból”
- Támogatás a megszokott öröklési stratégiákhoz
  - Egy tábla egy öröklési hierarchiához
  - Joinolt leszármazott osztály stratégia
  - Tábla minden konkrét osztályhoz (opcionális)

# Objektum modell



# Adatmodell

- Single table:

| ANIMAL | | | | |
|---|---|---|---|---|
| ID | DISC | NAME | LEG_COUNT | WING_SPAN |

- Joined:

| ANIMAL | |
|---|---|
| ID | NAME |

| LAND_ANIMAL | |
|---|---|
| ID | LEG_COUNT |

| AIR_ANIMAL | |
|---|---|
| ID | WING_SPAN |

- Table per Class:

| LAND_ANIMAL | | |
|---|---|---|
| ID | NAME | LEG_COUNT |

| AIR_ANIMAL | | |
|---|---|---|
| ID | NAME | WING_SPAN |

# Stratégiák jellemzői

- Egy tábla egy öröklési hierarchiához
  - Nem normalizált (null mezők!)
  - Polimorfikus lekérdezések, relációk jó támogatása
- Joinolt leszármaztatott osztály
  - Leszármazott osztály specifikus állapot külön táblában tárolódik
  - Normalizált
  - Teljesítmény problémák közepesen mély öröklési hierarchiák esetén
- Tábla minde konkrét osztályhoz
  - Normalizált
  - Polimorfikus lekérdezések és relációk gyengén támogatottak

# Témák

- Háttér és kulcsfogalmak
- Entitások és az Entitás Manager API
- Relációk
- Lekérdezések
- Objektum/Relációs megfeleltetés
- Java perzisztencia Java EE és Java SE környezetben
- Összefoglaló

# Persistence Context

- A perzisztencia contextet kezelheti az alkalmazás vagy a konténer
- Konténer menedzselt perzisztencia context egyszerű használatot jelent Java EE környezetben

# Konténer általt menedzselt Perzisztencia Context

- A perzisztencia context életciklusát a konténer kezeli
- Lehet egy JTA tranzakción át tartó, vagy lehet extended hatályú
  - Alapértelmezett a tranzakcióval határolt
  - Extended perzisztencia context használható arra, hogy az entitásokat több egymást követő tranzakción át is managed állapotban tartsa
- A komponensek között a JTA tranzakcióval terjed
  - Az ugyanahoz a perzisztencia unithoz tartozó entitás menedzserek számára közösen elérhető
- Injection vagy JNDI lookup segítségével elérhető
  - Annotáció: @PersistenceContext

# Példa

```
@Stateless public class OrderManagementBean
    implements OrderManagement {

    ...
    @PersistenceContext EntityManager em;
    ...

    public Order addNewOrder(Customer customer, Product
  product) {
        Order order = new Order(product);
        customer.addOrder(order);

        em.persist(order);

        return order;
    }

}
```

# Alkalmazás által menedzselt perzisztencia context

- A perzisztencia context életciklusát az alkalmazás vezérli
  - Az alkalmazás az EntityManagert az EntityManagerFactory segítségével hozza létre a kívánt perzisztencia unithoz
    - Annotáció: @PersistenceUnit
  - A perzisztencia context mindaddig létezik amíg az alkalmazás be nem zárja
  - Nincs automatikus propagáció (referenciát kell átadni)
- Java EE konténeren kívül van rá szükség
  - Lehet Java EE környezetben is használni
  - JTA támogatás nélküli környezetekben az EntityTransaction API-t lehet az erőforrás-lokális tranzakciók kezelésére használni

# Java SE kömyezet

- Bootstrapping API használatára van szükség
  - javax.persistence.Persistence osztály segítségével lehet EntityManagerFactory osztályt létrehozni
- EntityManagerFactory API segítségével lehet alkalmazás menedzselt entitás menedzsert létrehozni
- Csak resource-local tranzakciók támogatottak

# Példa Java SE programra

```
public class SalaryChanger {

  public static void main(String[] args) {
    EntityManagerFactory emf = Persistence
        .createEntityManagerFactory("HRSystem");
    EntityManager em = emf.createEntityManager();
    em.getTransaction().begin();
    Employee emp = em.find(
        Employee.class, new Integer(args[0]));
    emp.setSalary(new Integer(args[1]));
    em.getTransaction().commit();
    em.close();
    emf.close();
  }

}
```

# Témák

- Háttér és kulcsfogalmak
- Entitások és az Entitás Manager API
- Relációk
- Lekérdezések
- Objektum/Relációs megfeleltetés
- Java perzisztencia Java EE és Java SE környezetben
- Összefoglaló

# Összefoglaló

- Entitások egyszerű Java osztályok
  - Egyszerű fejleszteni és használni
  - Minden alkalmazás rétegben használható, a standalone klienstől az alkalmazásszerver konténeréig mindenhol
- EntityManager API
  - Támogatja a Java EE és az azon kívüli használatot is
  - Java EE környezetben a konténer menedzslet perzisztencia context használata tovább egyszerűsíti a fejlesztést
- Szabványosított O/R mapping megoldás
- Statikus és dinamikus lekérdezések a Java Persistence query language és native SQL segítségével

# Linkek

JSR-220:
http://jcp.org/en/jsr/detail?id=220
Java EE tutorial:
http://java.sun.com/javaee/5/docs/tutorial/doc/
Netbeans tutorial:
http://www.netbeans.org/kb/trails/java-ee.html
Glassfish:
https://glassfish.dev.java.net/javaee5/persistence
Sun Educational Services:
http://www.sun.com/training/catalog/courses/SL-351.xml
http://hu.sun.com/ > Oktatás

# Java Persistence API

# JSR-220, EJB 3.0

Molnár István, Zsemlye Tamás
istvan.molnar@sun.hu
tamas.zsemlye@sun.com