Java™ Persistence API

Linda DeMichiel
Sun Microsystems

Michael Keith
Oracle Corporation

TS-3395

Agenda

Background

Entities and the EntityManager API

Queries

Object/Relational Mapping

Java Persistence API in Java EE and Java SE

Summary

Background

- Part of JSR-220 (Enterprise JavaBeans™ 3.0)
- Began as simplification of entity beans
  - Evolved into POJO persistence technology
- Scope expanded at request of community to support general use in Java™ EE and Java SE environments
- Reference implementation under Project GlassFish
  - Oracle TopLink Essentials
  - Oracle joined Sun as co-speclead in June 2005

Primary Features

- POJO-based persistence model
  - Simple Java classes—not components
- Support for enriched domain modelling
  - Inheritance, polymorphism, etc.
- Expanded query language
- Standardized object/relational mapping
  - Using annotations and/or XML
- Usable in Java EE and Java SE environments
- Support for pluggable persistence providers

Entities

- Plain old Java objects
  - Created by means of new
  - No required interfaces
  - Have persistent identity
  - May have both persistent and non-persistent state
    - Simple types (e.g., primitives, wrappers, enums, serializables)
    - Composite dependent object types (e.g., Address)
    - Non-persistent state (transient or @Transient)
  - Can extend other entity and non-entity classes
  - Serializable; usable as detached objects in other tiers
    - No need for data transfer objects

Example: Entity

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

    …
}
```

Entity Identity

- Every entity has a persistence identity
  - Maps to primary key in database
- Can correspond to simple type
  - Annotations
    - @Id—single field/property in entity class
    - @GeneratedValue—value can be generated automatically
- Can correspond to user-defined class
  - Annotations
    - @EmbeddedId—single field/property in entity class
    - @IdClass—corresponds to multiple Id fields in entity class
- Must be defined on root of entity hierarchy or mapped superclass

Entity Relationships

- One-to-one, one-to-many, many-to-many, many-to-one relationships among entities
  - Support for Collection, Set, List, Map types
- May be unidirectional or bidirectional
  - Bidirectional relationships are managed by application, not container
  - Bidirectional relationships have owning side and inverse side

Example: Relationships

```
@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);}
}
```

Inheritance

- Entities can extend
  - Other entities
    - Either concrete or abstract
  - Mapped superclasses
    - Supply common entity state
  - Ordinary (non-entity) Java classes
    - Supply behavior and/or non-persistent state

Example: Mapped Superclass

```
@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;
}
```

Example: Abstract Entity

```
@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;
}
```

Persistence Context

- Set of managed entity instances at runtime
- Unique entity identity for any persistent identity
- Entity instances all belong to same persistence unit; all mapped to same database
  - Persistence unit is unit of packaging and deployment
- EntityManager API is used to manage persistence context, control lifecycle of entities, find entities by id, create queries

Entity Lifecycle

- new
  - New entity instance is created
  - Entity is not yet managed or persistent
- persist
  - Entity becomes managed
  - Entity becomes persistent in database on transaction commit
- remove
  - Entity is removed
  - Entity is deleted from database on transaction commit
- refresh
  - Entity's state is reloaded from database
- merge
  - State of detached entity is merged back into managed entity

Persist

```
@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;
    }
}
```

Cascading 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;
}
```

Remove

```
@Entity
public class Order {
    @Id protected Long orderId;
    …
    @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);
}
```

Merge

```
@Entity
public class Order {
    @Id protected Long orderId;
    …
    @OneToMany(cascade={PERSIST, REMOVE, MERGE})
    protected Set<LineItem> lineItems = new HashSet();
}
```

```
…
@PersistenceContext EntityManager em;
…
public Order updateOrder(Order changedOrder) {
    return em.merge(changedOrder);
}
```

Agenda

Background

Entities and the EntityManager API

Queries

Object/Relational Mapping

Java Persistence API in Java EE and Java SE

Summary

Java Persistence Query Language

- An extension of EJB™ QL
  - Like EJB QL, a SQL-like language
- Added functionality
  - Projection list (SELECT clause)
  - Explicit JOINS
  - Subqueries
  - GROUP BY, HAVING
  - EXISTS, ALL, SOME/ANY
  - UPDATE, DELETE operations
  - Additional functions

Projection

```
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'
```

Subqueries

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

Joins

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

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

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
```

Queries

- Static queries
  - Defined with Java language metadata or XML
    - Annotations: @NamedQuery, @NamedNativeQuery
- Dynamic queries
  - Query string is specified at runtime
- Use Java Persistence query language or SQL
- Named or positional parameters
- EntityManager is factory for Query objects
  - createNamedQuery, createQuery, createNativeQuery
- Query methods for controlling max results, pagination, flush mode

Dynamic Query

```
@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();
}
```

Static Query

```
@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();
}

…
```

Agenda

Background

Entities and the EntityManager API

Queries

Object/Relational Mapping

Java Persistence API in Java EE and Java SE

Summary

Object/Relational Mapping

- Map persistent object state to relational database
- Map relationships to other entities
- Mapping metadata may be annotations or XML (or both)
- Annotations
  - Logical—object model (e.g., @OneToMany, @Id, @Transient)
  - Physical—DB tables and columns (e.g., @Table, @Column)
- XML
  - Elements for mapping entities and their fields or properties
  - Can specify metadata for different scopes
- Rules for defaulting of database table and column names

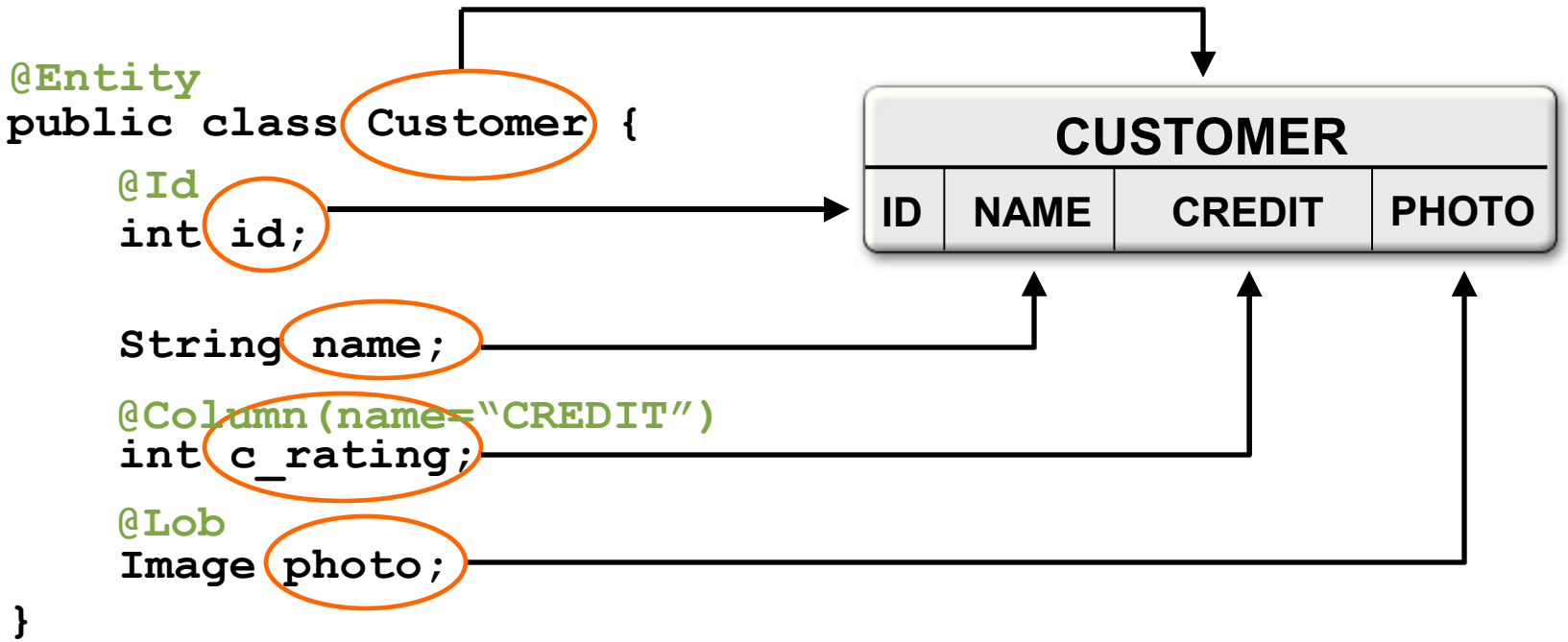
Object/Relational Mapping

- State or relationships may be loaded or "fetched" as EAGER or LAZY
  - LAZY is a hint to the Container to defer loading until the field or property is accessed
  - EAGER requires that the field or relationship be loaded when the referencing entity is loaded
- Cascading of entity operations to related entities
  - Setting may be defined per relationship
  - Configurable globally in mapping file for persistence-by-reachability

Simple Mappings

- Direct mappings of fields/properties to columns
  - @Basic—optional annotation to indicate simple mapped attribute
- Maps any of the common simple Java types
  - Primitives, wrapper types, Date, Serializable, byte[ ], ...
- Used in conjunction with @Column
- Defaults to the type deemed most appropriate if no mapping annotation is present
- Can override any of the defaults

Simple Mappings



Simple Mappings

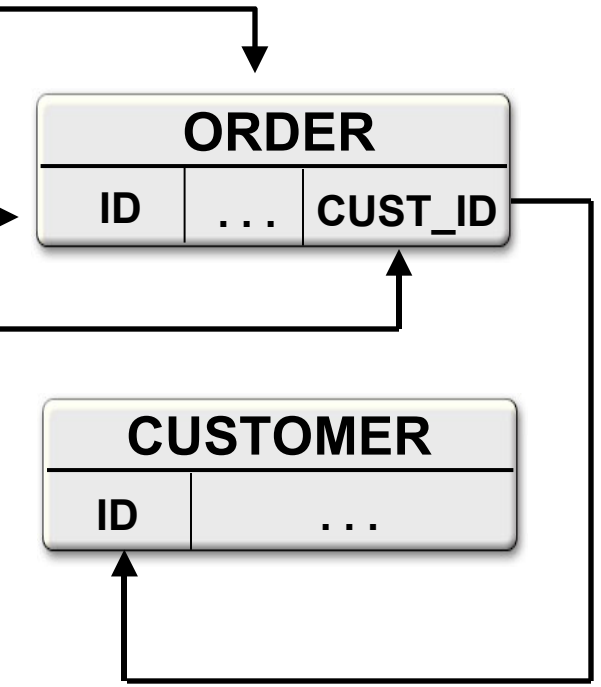
```
<entity class="com.acme.Customer">
  <attributes>
    <id name="id"/>
    <basic name="c_rating">
      <column name="CREDIT"/>
    </basic>
    <basic name="photo"><lob/></basic>
  </attributes>
</entity>
```

Relationship Mappings

- Common relationship mappings supported
  - @ManyToOne, @OneToOne—single entity
  - @OneToMany, @ManyToMany—collection of entities
- Unidirectional or bidirectional
- Owning and inverse sides of every bidirectional relationship
- Owning side specifies the physical mapping
  - @JoinColumn to specify foreign key column
  - @JoinTable decouples physical relationship mappings from entity tables

Many-to-One Mapping

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

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

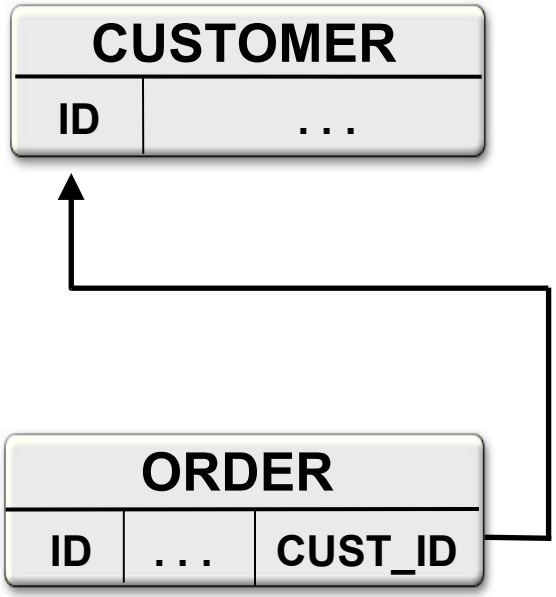


Many-to-One Mapping

```
<entity class="com.acme.Order">
  <attributes>
    <id name="id"/>
    ...
    <many-to-one name="cust"/>
  </attributes>
</entity>
```

One-to-Many Mapping

```
@Entity
public class Customer {
   @Id
   int id;
   ...

   @OneToMany(mappedBy="cust")
   Set<Order> orders;
}

@Entity
public class Order {
   @Id
   int id;
   ...
   @ManyToOne
   Customer cust;
}
```



One-to-Many Mapping

```
<entity class="com.acme.Customer">
  <attributes>
    <id name="id"/>
      ...
    <one-to-many name="orders" mapped-by="cust"/>
  </attributes>
</entity>
```

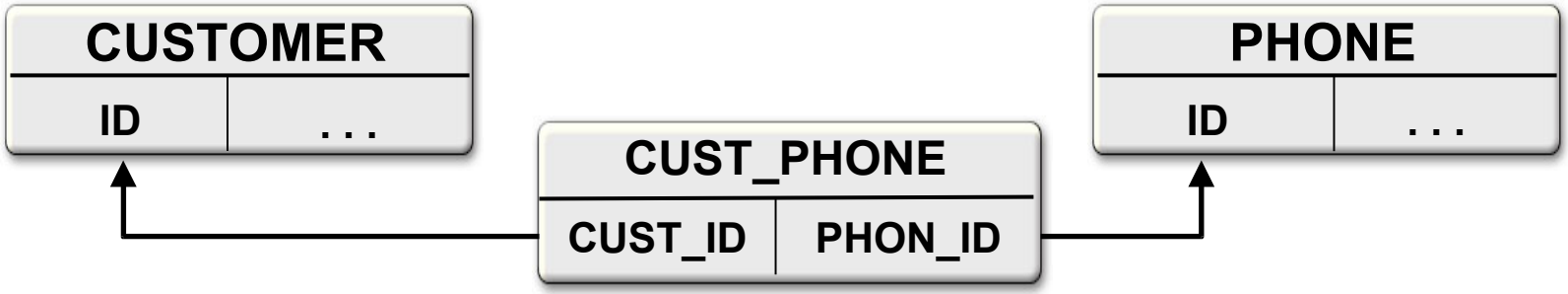
Many-to-Many Mapping

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

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

Many-to-Many Mapping

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



Many-to-Many Mapping

```
<entity class="com.acme.Customer">
  <attributes>
    ...
    <many-to-many name="phones"
      <join-table name="CUST_PHONE">
        <join-column name="CUST_ID"/>
        <inverse-join-column name="PHON_ID"/>
      </join-table>
    </many-to-many>
  </attributes>
</entity>
```

Embedded Objects

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

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

Embedded Objects

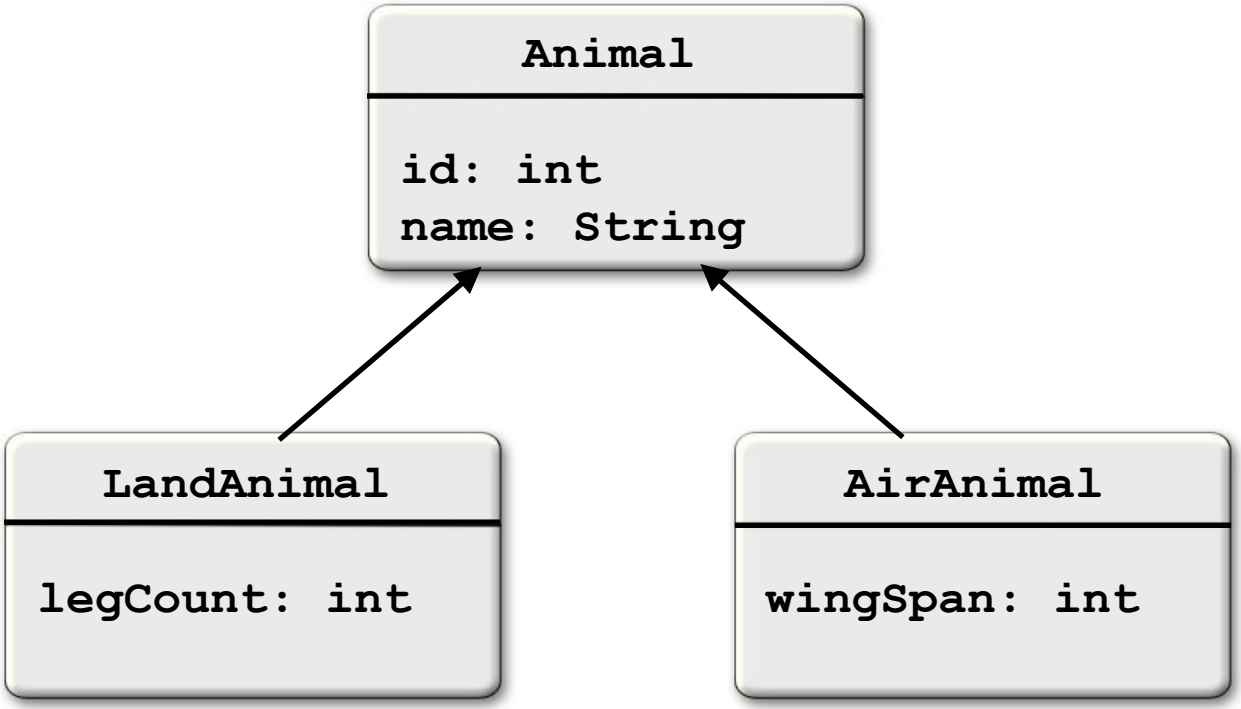
```
<entity class="com.acme.Customer">
   <attributes>
        ...
     <embedded name="info"/>
  </attributes>
</entity>

<embeddable class="com.acme.CustomerInfo">
  <attributes>
    <basic name="photo"><lob/></basic>
  </attributes>
</embeddable>
```

Inheritance

- Entities can extend
  - Other entities—concrete or abstract
  - Non-entity classes—concrete or abstract

- Map inheritance hierarchies in three ways
  1. SINGLE_TABLE
  2. JOINED
  3. TABLE_PER_CLASS

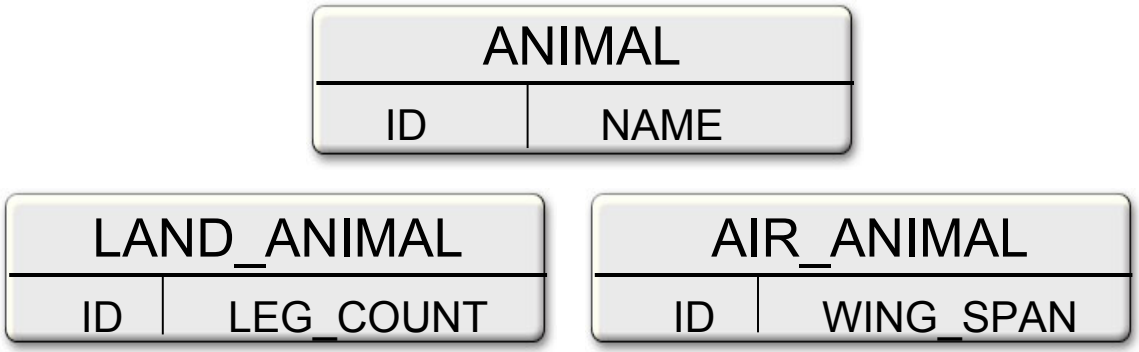
Object Model



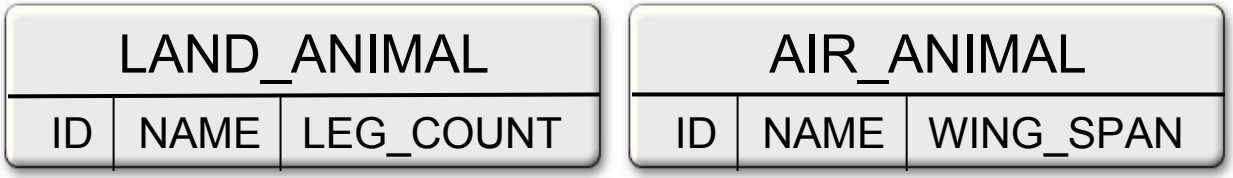
Data Models

- Single table:

- Joined:



- Table per Class:



Persistence in Java SE

- No deployment phase
  - Application must use a “Bootstrap API” to obtain an EntityManagerFactory
- Typically use resource-local EntityManagers
  - Application uses a local EntityTransaction obtained from the EntityManager
- New persistence context for each and every EntityManager that is created
  - No propagation of persistence contexts

Entity Transactions

- Resource-level transaction akin to a JDBC transaction
  - Isolated from transactions in other EntityManagers
- Transaction demarcation under explicit application control using EntityTransaction API
  - begin(), commit(), setRollbackOnly(), rollback(), isActive()
- Underlying (JDBC™) resources allocated by EntityManager as required

Bootstrap Classes

`javax.persistence.Persistence`

- Root class for bootstrapping an EntityManager
- Locates a provider service for a named persistence unit
- Invokes on the provider to obtain an EntityManagerFactory

`javax.persistence.EntityManagerFactory`

- Creates EntityManagers for a named persistence unit or configuration

Example

```
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();
  }
}
```

Current Status

- Final Release last week as part of EJB 3.0
- Java Persistence API specification
  http://jcp.org/en/jsr/detail?id=220
- Reference Implementation
  - Oracle TopLink Essentials
  - Part of Sun open source Project GlassFish
    http://glassfish.dev.java.net

Summary

- Entities are simple Java classes
  - Easy to develop and intuitive to use
  - Can be moved to other server and client tiers
- EntityManager
  - Simple API for operating on entities
  - Supports use inside and outside Java EE containers
- Standardization
  - O/R mapping using annotations or XML
  - Named and dynamic query definition
  - SPI for pluggable persistence providers

For More Information

Technical Sessions

- TS-3616 Building EJB 3.0 Applications: A Simple Matter of Point and Squish
  Thurs @ 9:45
- TS-1887 Java Persistence in the Web Tier
  Fri @ 10:45
- TS-9056 Java Persistence API in 60 Minutes
  Fri @ 2:30

For More Information

Books

- Pro EJB 3: Java Persistence API (Apress)
  Mike Keith & Merrick Schincariol
- Enterprise JavaBeans 5th Edition (O'Reilly)
  Bill Burke, Richard Monson-Haefel

More books coming…

the POWER of JAVA™

Q&A

Java™ Persistence API

Linda DeMichiel
Sun Microsystems

Michael Keith
Oracle Corporation

TS-3395