



Persistence Techniques in EJB 3.0

Amanda Waite

Senior Software Engineer

Sun Microsystems

Agenda

- EJB 3.0 overview
- Persistence Context
- O/R Mapping
- Entity Relationships
- Entity Inheritance
- Query API
- Summary and Resources



EJB 3.0 Overview

Original Goals for Persistence

- Simplification of EJB Entity Bean programming model
- Improved support for domain modelling
- Support for object/relational mapping
- Allow entities to be used outside of the EJB container
- Expanded query capabilities

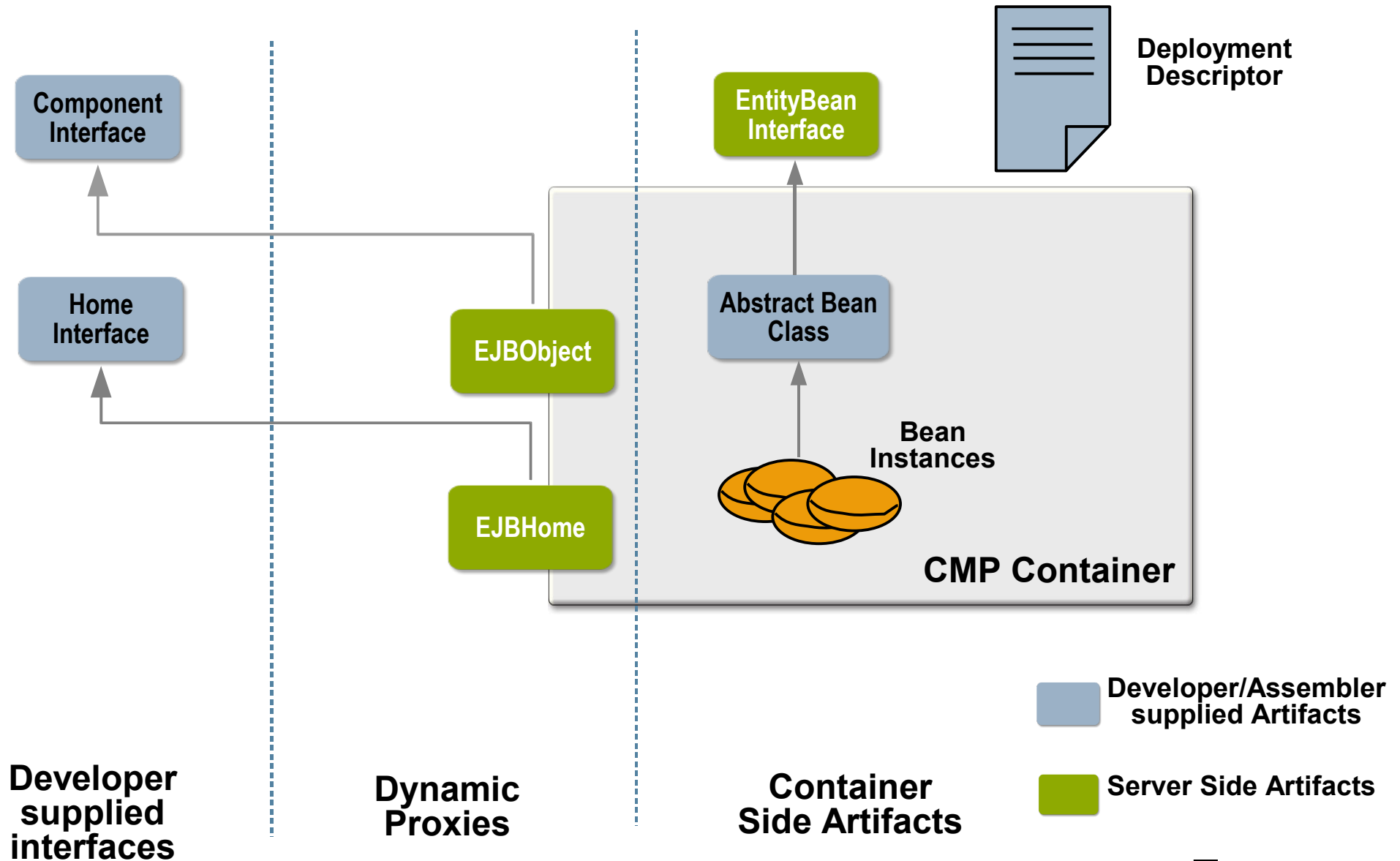
Results Today

- Possible to write very simple and reusable Entities
- Possible to model very complex relationships using advanced OO constructs
- EJB Persistence API extended to include use in Java SE
- Evolved into “common” Java Persistence API
- Support for pluggable, third-party persistence providers

EJB 3.0 Specification

- 3 Part specification:
 - > EJB Core Contracts and Requirements
 - > EJB 3.0 Simplified API
 - > Java Persistence API
- Specified under Java Community Process – JSR-220
 - > Public Final Draft – Dec 21st 2005
 - > Target Final release – JavaOne '06 (May 16-19 2006)
- Java EE 5 SDK Beta available today

EJB 2.1 Entity Bean artifacts



EJB 3.0 Entities

- Are simple objects, not components
- Are concrete Java Classes, entity instances created with the java “new” keyword (i.e.: `new Customer();`)
- No required bean (component/home) or callback interfaces
- Support domain modelling through inheritance and polymorphism
- Have state, accessed by either Fields or by Properties
- Use of annotations to describe Persistence and O/R mapping

Annotated Entity code

@Entity

```
public class Customer {  
    private Long custID;  
    private String name;  
    private Collection<Order> orders = new HashSet();  
  
    public Customer() {} // No argument constructor  
  
    public Customer(Long custID, String name) {  
        this.custID = custID;  
        this.name = name;  
    }  
}
```

@Id

```
public Long getCustID() {  
    return custID;  
}  
  
protected void setCustID (Long id) {  
    this.custID = id;  
}
```

Annotated Entity code (cont)

```
@Column(name = "CUST_NAME")
```

```
public String getName() {  
    return name;  
}
```

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

```
@OneToMany
```

```
public Collection<Order> getOrders() {  
    return orders;  
}
```

```
public void setOrders(Collection<Order> orders) {  
    this.orders = orders;  
}
```

```
// Other business methods  
}
```

Annotated Entity code

@Entity

@Entity denotes Entity class

```
public class Customer {
    private Long custID;
    private String name;
    private Collection<Order> orders = new HashSet();

    public Customer() {} // No argument constructor
```

```
public Customer(Long custID, String name) {
    this.custID = custID;
    this.name = name;
}
```

@Id denotes PK field(s) (annotation of getter method denotes property based access)

@Id

```
public Long getCustID() {
    return custID;
}
```

getters/setters to access state

```
protected void setCustID (Long id) {
    this.custID = id;
}
```

Annotated Entity code (cont)

```
@Column(name = "CUST_NAME")
```

```
public String getName() {
    return name;
}
```

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

Explicitly specifies column name to map to (default would be "name")

```
@OneToMany
```

```
public Collection<Order> getOrders() {
    return orders;
}
```

Denotes Entity relationship (One-To-Many between Customer and Order)

```
public void setOrders(Collection<Order> orders) {
    this.orders = orders;
}
```

```
// Other business methods
}
```

Creating an Entity instance

@Stateless



Indicates that this class is a
Stateless Session Bean

```
public class OrderEntry {
```

```
    public void addCustomer(Long id, String name) {
```

```
        Customer customer = new Customer(id, name);
```

```
    }
```

```
    ...  
}
```



Persistence Context

Persistent Identity

- 'new' entities do not have Persistent Identity
- An entity has a Persistent Identity after a persistence operation has been performed with it
- “Persistent Identity” simply means that a row corresponding to this entity exists in the datastore
- You need to know this to make sense of the spec

Persistence Context (PC)

- A set of managed entity instances
- Analogous to a JTA “transaction context”
- Persistence Context scoped to transaction
- Extended Persistence Context to span multiple sequential transactions

Entity instance Life Cycle

New

New Entity instance

Created using “new”
No persistent identity, not associated with a PC

Managed

Managed Entity instance

Has a persistent identity
Associated with a persistence context

Detached

Detached Entity instance

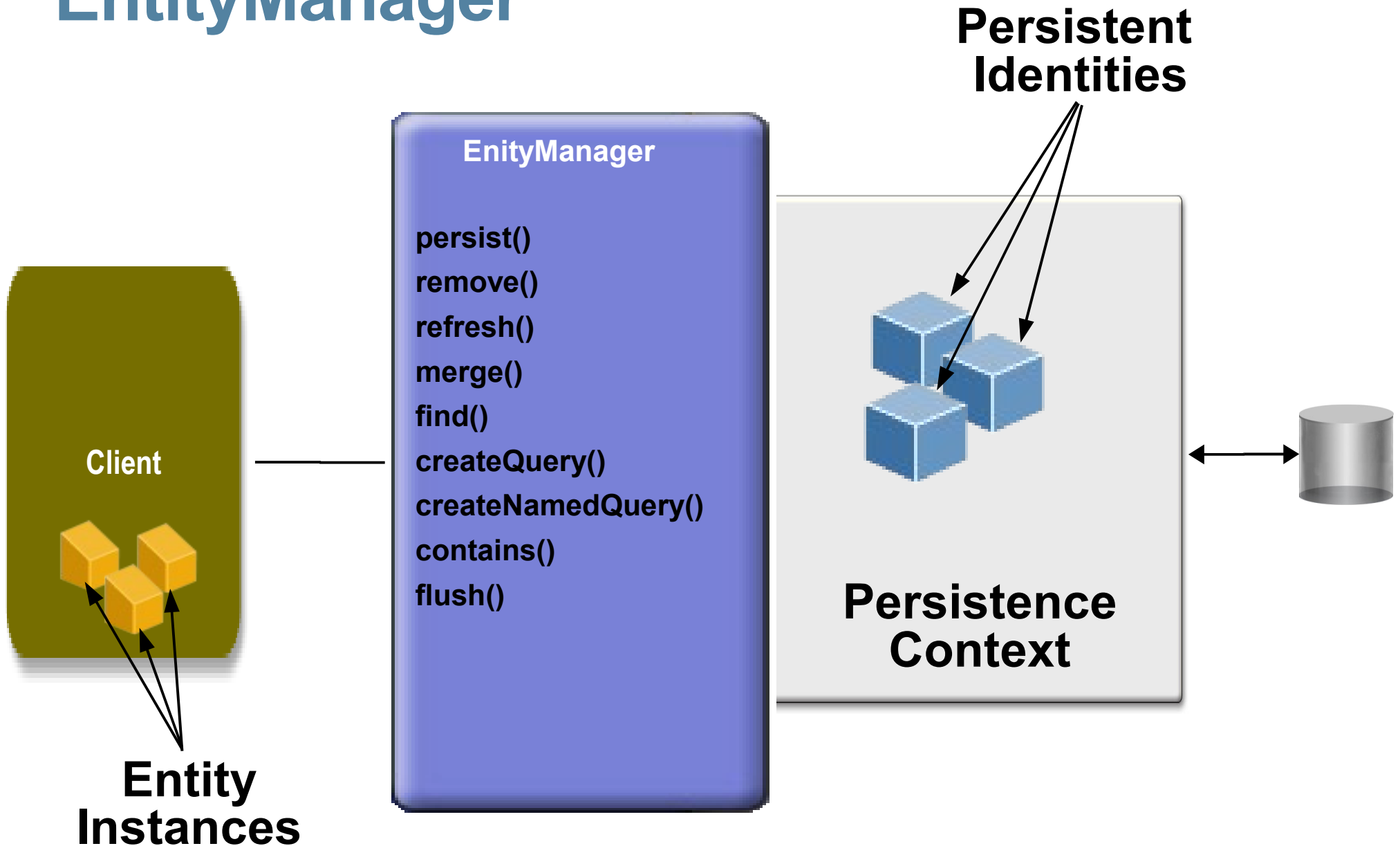
Has a persistent Identity
No longer associated with a PC
Occurs when a PC ends or if passed-by-value

Removed

Removed Entity instance

Has a persistent identity
Associated with a persistence context
Scheduled to be removed from the database

EntityManager



EntityManager

- Programatic access to Persistence Context (PC)
- Controls lifecycle of entities and creates queries
 - > `persist()` - insert the identity of an entity into the DB (making it persistent)
 - > `remove()` - Delete the persistent identity of the entity from the DB
 - > `merge()` - Synchronize the state of detached entity with the PC
 - > `flush()` - Force sync of PC to the DB
 - > `refresh()` - Reload the state of an entity instance from the DB
 - > `find()` - execute a simple PK query
 - > `createQuery()` - Create query instance using EJB QL
 - > `createNamedQuery()` - create predefined query instance
 - > `createNativeQuery()` - Create query instance using SQL

Persistence Unit

- A persistence unit is a logical grouping that includes:
 - > An entity manager factory and its entity managers, together with their configuration information
 - > The set of managed classes included in the persistence unit and managed by the entity managers of the entity manager factory
 - > Mapping metadata that specifies the mapping of the classes to the database

Persistence Unit (contd)

- A Persistence Unit (PU) is defined by a *persistence.xml* file
- A PU can be packaged in a WAR or EJB JAR or in a JAR as part of an EAR or WAR
- persistence.xml should be located as follows
 - > WAR: WEB-INF/classes/META-INF/persistence.xml
 - > EJB JAR: META-INF/persistence.xml
- A PU must have a name which is unique within its scope
- The persistence.xml file may be used to define more than one PU

Persistence Unit example

Application Managed Persistence Context

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence">
  <persistence-unit name="em" transaction-type="RESOURCE_LOCAL">
    <provider>cmp3.EntityManagerFactoryProvider</provider>
    <class>addressbook.Address</class>
    <class>addressbook.Person</class>
    <properties>
      <property name="jdbc.connection.string"
        value="jdbc:derby://myhost:1527/db"/>
      <property name="jdbc.driver"
        value="org.apache.derby.jdbc.ClientDriver"/>
      <property name="jdbc.user" value="app"/>
      <property name="jdbc.password" value="app"/>
      <property name="ddl-generation" value="create"/>
    </properties>
  </persistence-unit>
</persistence>
```

Persistence Unit example

Container Managed Persistence Context

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns="http://java.sun.com/xml/ns/persistence">
  <persistence-unit name="Order" transaction-type="JTA">
    <description>
      This unit manages orders and customers.
      It does not rely on any vendor-specific features and can
      therefore be deployed to any persistence provider.
    </description>
    <jta-data-source>jdbc/MyOrderDB</jta-data-source>
    <mapping-file>ormap.xml</mapping-file>
    <jar-file>MyOrderApp.jar</jar-file>
    <class>com.widgets.Order</class>
    <class>com.widgets.Customer</class>
  </persistence-unit>
</persistence>
```

Transactions and EntityManager

- Transaction type can be either JTA or resource-local
- JTA entity managers
 - > Used in Java EE i.e. Managed environments
 - > Container-managed entity managers must support JTA
 - > Support Local and global transactions
- resource-local entity managers
 - > **EntityTransaction** API used for application-controlled local transactions
 - > Cannot participate in global transactions

Client View: Persisting an Entity instance

Transaction Attribute for all methods in Class (REQUIRED is the default for an SLSB)



```
@Stateless  
@TransactionAttribute(REQUIRED)  
public class OrderEntry {
```

Dependency Injection of EntityManager (defined as name "order" in PU)



```
@PersistenceContext(unitName="Order")  
EntityManager orderEm;
```

```
Public void addCustomer(Long id, String name) {  
    Customer customer = new Customer(id, name);  
    orderEm.persist(customer);  
}
```

Client view: Using find()

@Stateless

```
public class OrderEntry {
```

```
    @PersistenceContext(name="Order")
```

```
    EntityManager em;
```

```
    public void enterOrder(int custID, Order newOrder){
```

```
        //Use find method to locate customer entity
```

```
        Customer c = em.find(Customer.class, custID);
```

```
        c.getOrders().add(newOrder);
```

```
        newOrder.setCustomer(c);
```

```
    }
```

```
    // other business methods
```

```
}
```

PCs and Web Applications

- EntityManager is not Thread Safe so don't use it in a Servlet or a Listener, use EntityManagerFactory in these cases and pass in the EntityManager to request-scoped objects
- Container Managed request-scoped objects (i.e.: JSF managed beans) can use dependency injection for the EntityManager



O/R Mapping

O/R Mapping

- Specified using standard description elements in a separate mapping file or within the code as annotations
- Rules for defaulting of DB table and column names
- Comprehensive set of annotations defined for mapping
 - > Relationships
 - > Joins
 - > Database tables and columns
 - > Database sequence generators
 - > Much more

Simple Mapping

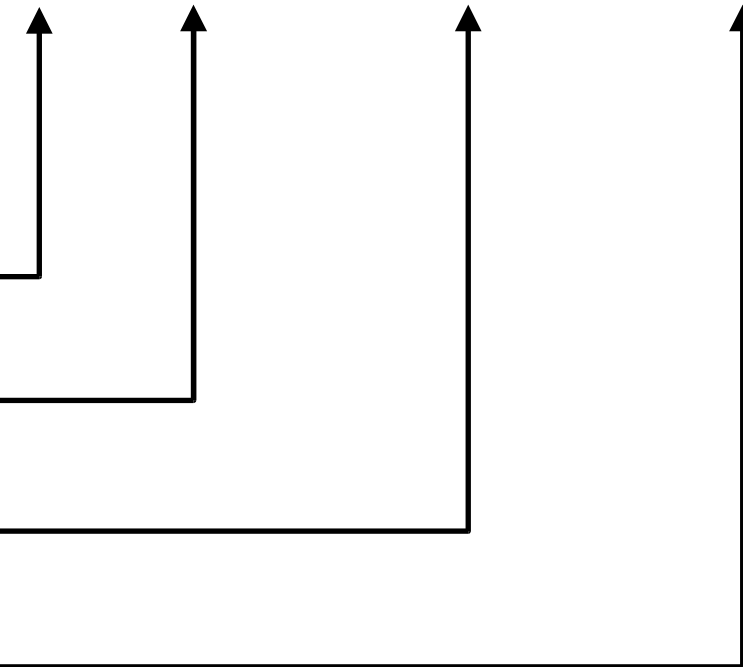
```

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

    String name;

    int c_rating;
    @Lob
    Image photo;
}
    
```

CUSTOMER			
ID	NAME	C_RATING	PHOTO



Column Mapping

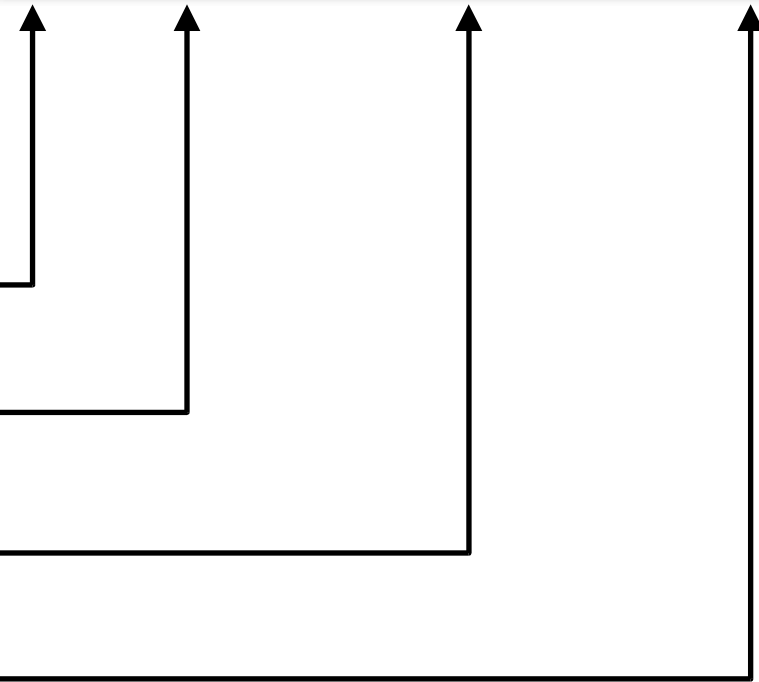
```

@Entity(access=FIELD)
public class Customer {

    @Id
    int id;

    String name;
    @Column(name="CREDIT")
    int c_rating;
    @Lob
    Image photo;
}
    
```

CUSTOMER			
ID	NAME	CREDIT	PHOTO





Entity Relationships

Entity Relationships

- Models association between entities
- Supports unidirectional as well as bidirectional relationships
 - > Unidirectional relationship: Entity A references B, but B doesn't reference A
- Cardinalities
 - > OneToOne
 - > OneToMany
 - > ManyToOne
 - > ManyToMany

O/R Mapping: ManyToOne

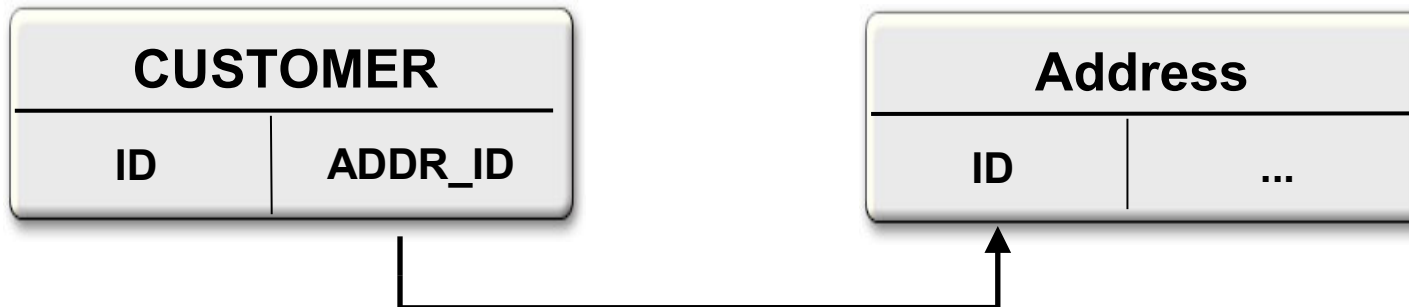
```
@Entity(access=FIELD)
public class Customer {

    @Id
    int id;

    ...

    @ManyToOne
    Address addr;

}
```



O/R Mapping: OneToMany

```

@Entity(access=FIELD)
public class Customer {

    @Id
    int id;

    ...

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

}
    
```

```

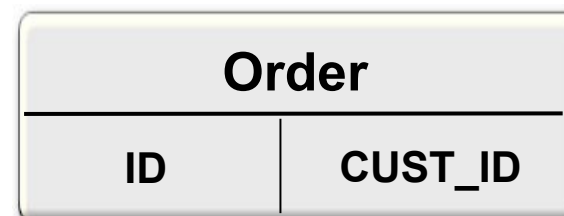
@Entity(access=FIELD)
public class Order {

    @Id
    int id;

    ...

    @ManyToOne
    Customer cust;

}
    
```



O/R Mapping: ManyToMany

```

@Entity(access=FIELD)
public class Customer {

    @Id
    int id;

    ...

    @ManyToMany
    Collection<phone> phones;
}
    
```

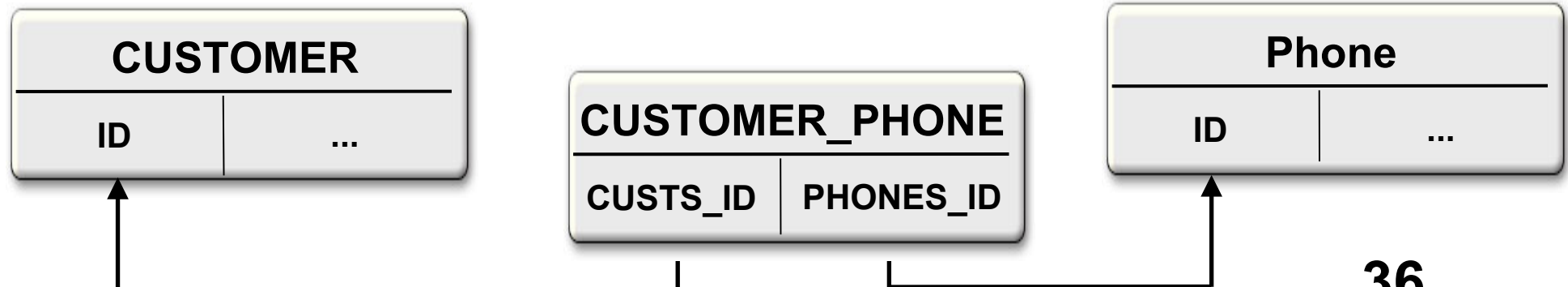
```

@Entity(access=FIELD)
public class Phone {

    @Id
    int id;

    ...

    @ManyToMany(mappedBy="phones")
    Collection<Customer> custs;
}
    
```



Cascading Behaviour

- Cascading is used to propagate the effect of an operation across relationships
- Specified via MetaData i.e.: `@OneToMany(Cascade=ALL)`
- Can cascade combinations of:
 - > PERSIST, REMOVE, MERGE, REFRESH, ALL
- Default is to not Cascade
- Cascading operations will work only when entities are associated to the persistence context
 - > If a cascaded operation takes place on detached entity, `IllegalArgumentException` is thrown



Entity Inheritance

Entity Inheritance

- Entities can Inherit from or extend:
 - > Non Entity Classes concrete or abstract
 - > Entity Classes – concrete or abstract
- Different strategies are available for mapping inheritance hierarchies to Database tables

Entity Inheritance

Mapping Classes to Tables

- Use Java™ application metadata to specify mapping
- Support for various inheritance mapping strategies
 - > **Single table (default)**
 - > All the classes in a hierarchy are mapped to a single table
 - > Root table has a discriminator column whose value identifies the specific subclass to which the instance represented by row belongs
 - > **Table per class (optional)**
 - > Each class in a hierarchy mapped to a separate table, all properties of the class (incl. inherited properties) are mapped to columns of this table
 - > **Joined**
 - > The root of the hierarchy is represented by a single table
 - > Each subclass represented by separate table that contains fields specific to the subclass as well as columns that represent its primary key(s)

Entity Inheritance Example

```

@Entity
@Table(name="CUST")
@Inheritance(strategy=SINGLE_TABLE)
@DiscriminatorColumn(name="STATUS",
    discriminatorType=STRING)
@DiscriminatorValue("CUST")
public class Customer {...}

```

```

@Entity
@DiscriminatorValue("VCUST")
public class ValuedCustomer extends Customer{
    private String privilegeProgramNo;
}

```

CUST			
ID	NAME	STATUS	PRIV_PROG_NO
1234	Matt	CUST	
6789	Mandy	VCUST	56634604603



Query API

EJB QL Enhancements

- Bulk update and delete operations
- Group By / Having
- Subqueries
- Additional SQL functions
 - > UPPER, LOWER, TRIM, CURRENT_DATE, ...
- Polymorphic queries
- Support for dynamic queries or ad-hoc queries in addition to named queries or static queries

Polymorphic Queries

- All Queries are polymorphic by default
 - > That is to say that the FROM clause of a query designates not only instances of the specific entity class(es) to which it explicitly refers but of subclasses as well

```
select avg(e.salary) from Employee e where e.salary > 80000
```

This example returns average salaries of all employees, including subtypes of Employee, such as Manager

Joins

- Adds keyword JOIN in EJB-QL
- Supports
 - > Inner Joins
 - > Left Joins/Left outer joins
 - > Fetch join
 - > Enables pre-fetching of association data as a side-effect of the query

```
SELECT DISTINCT c FROM Customer c LEFT JOIN FETCH c.orders WHERE  
c.address.state = 'MA'
```

Dynamic Queries

- Dynamically created in application Business Logic using the `EntityManager.createQuery()` method

```
public List findWithName(String name) {  
  
    return em.createQuery(  
        "SELECT c from Customer c WHERE c.name LIKE :custName")  
        .setParameter("custName", name)  
        .setMaxResults(10)  
        .getResultList();  
}
```

Named Queries

- Defined in MetaData using `@NamedQuery`
- Example (as per Dynamic Query example):

```
@NamedQuery (name="findAllCustomersWithName",  
    query="SELECT c FROM Customer c WHERE c.name LIKE ?1"  
)
```

- Used with `EntityManager.createNamedQuery()`

```
customers = em.createNamedQuery("findAllCustomersWithName")  
    .setParameter(1, "Smith")  
    .getResultList();
```



Summary and Resources

Java/EJB 3.0 Persistence Summary

- Java Persistence technology
 - > Will be implemented by open-source as well as closed-source products
- Future of light-weight modelling
- Defined by the EJB 3.0 Expert Group
 - > However, it is a separate specification
- Is designed to accommodate non-managed applications

Resources

- JCP Page for JSR-220
 - > <http://jcp.org/en/jsr/detail?id=220>
- Java EE Home on java.sun.com
 - > <http://java.sun.com/javaee/>
- Netbeans 5.5 Enterprise Pack Technology Preview
 - > <http://www.netbeans.org/kb/55/index.html> and <http://www.netbeans.org>
- Amis Technology Blog
 - > <http://technology.amis.nl/blog/>
- Oracle EJB How-Tos
 - > <http://www.oracle.com/technology/tech/java/ejb30.html>

Resources

- Glassfish persistence homepage
 - > <https://glassfish.dev.java.net/javaee5/persistence>
- Persistence support page
 - > <https://glassfish.dev.java.net/javaee5/persistence/entity-persistence-support.html>
- Blog on using persistence in Web applications
 - > http://weblogs.java.net/blog/ss141213/archive/2005/12/using_java_per_s.html
- Blog on schema generation
 - > http://blogs.sun.com/roller/page/java2dbInGlassFish#automatic_table_generation_feature_in

Sun Developer Network

- Free resources for all developers
 - > Tools (Creator, Java Studio Enterprise Edition)
 - > Forums
 - > FAQs
 - > Early Access
 - > Newsletters
- Join Today!
 - > <http://developers.sun.com>



Get Connected

Sun Developer Network connects you to what you need, when you need it.

Join Now >>

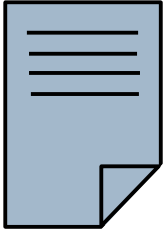


Thank You

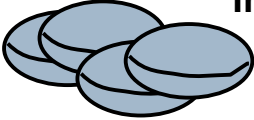
Amanda Waite

amanda.waite@sun.com

EJB 3.0 Entity Bean artifacts



**Deployment
Descriptor
(optional)**



**Bean
Instances**



**Developer/Assembler
supplied Artifacts**

Client view: Java Application

```
public class OrderClient {  
  
    // Find a factory for a Persistence Unit named "em"  
    EntityManagerFactory emf =  
        Persistence.createEntityManagerFactory("em");  
  
    // Get the EntityManager  
    EntityManager em = emf.createEntityManager();  
  
    public void addCust(int ID, String fName, String lName)  
    {  
  
        // Create new customer entity  
        Customer c = new Customer(ID, fName, lName);  
  
        // get a new transaction  
        EntityTransaction tx = em.getTransaction();  
  
        tx.begin();           // Begin transaction  
        em.persist(c);       // persist Customer entity  
        tx.commit();         // Commit transaction  
  
    }  
}
```

O/R Mapping Examples

@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; }

Entity Inheritance Example

```
@Entity  
@Inheritance(strategy=JOINED),  
public class Employee {...}
```

```
@Entity  
@PrimaryKeyJoinColumn(name="ID",  
    referencedColumnName="ID")  
public class Manager extends Employee {...}
```