Java Persistence API JSR-220 EJB3 Persistence

Banu Prakash

Where are we now?

- JPA 1.0 finalized in May 2006
  - Released as part of Java EE 5
- 80% of useful ORM features specifed
- Most major vendors have implemented JPA
- JPA 2.0 beta released

Implementations

- Persistence provider vendors include:
  - Oracle,Sun / TopLink Essentials (RI)
  - Eclipse JPA – EclipseLink Project
  - BEA Kodo / Apache OpenJPA
  - RedHat / JBoss Hibernate
  - SAP JPA
- JPA containers:
  - Sun, Oracle, SAP, BEA, JBoss, Spring 2.0

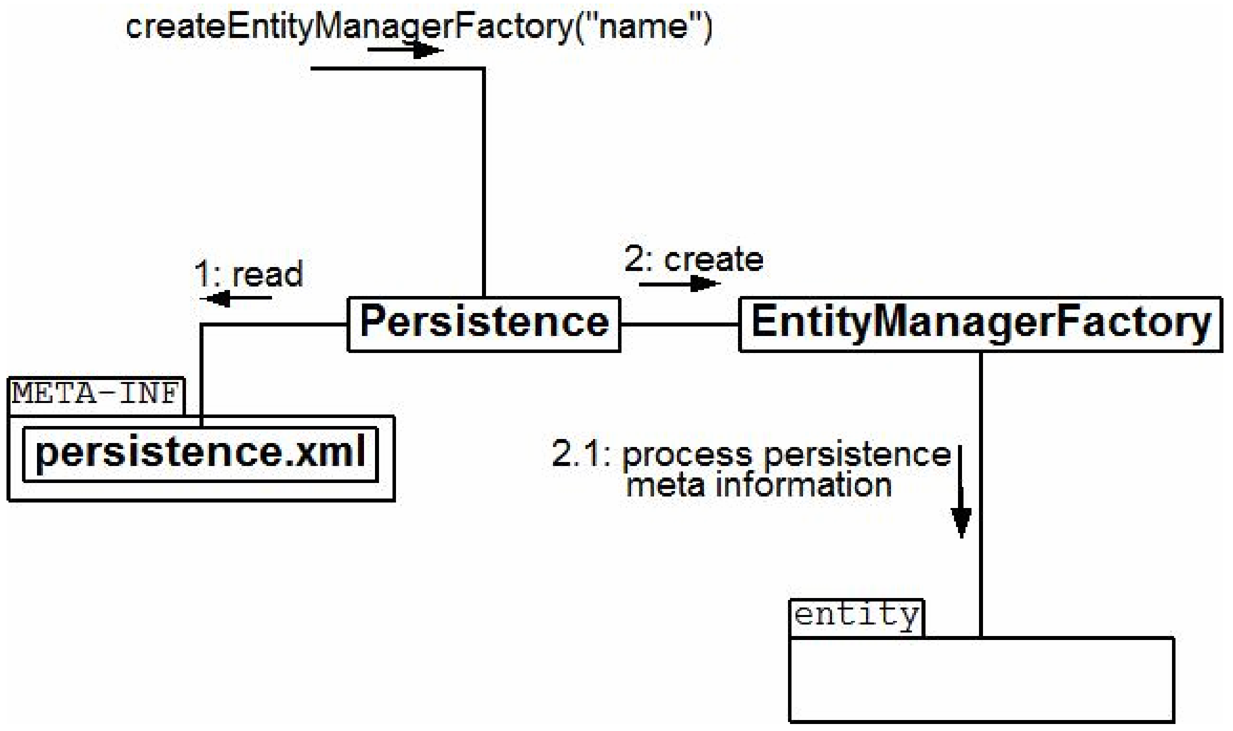
javax.persistence.Persistence

Entry point for using JPA.

The method you'll use on this class is createEntityManagerFactory(“name")   to retrieve an entity manager factory with the name "someName".

This class ***requires*** a file called **persistence.xml** to be in the class path under a directory called **META-INF**

EntityManagerFactory

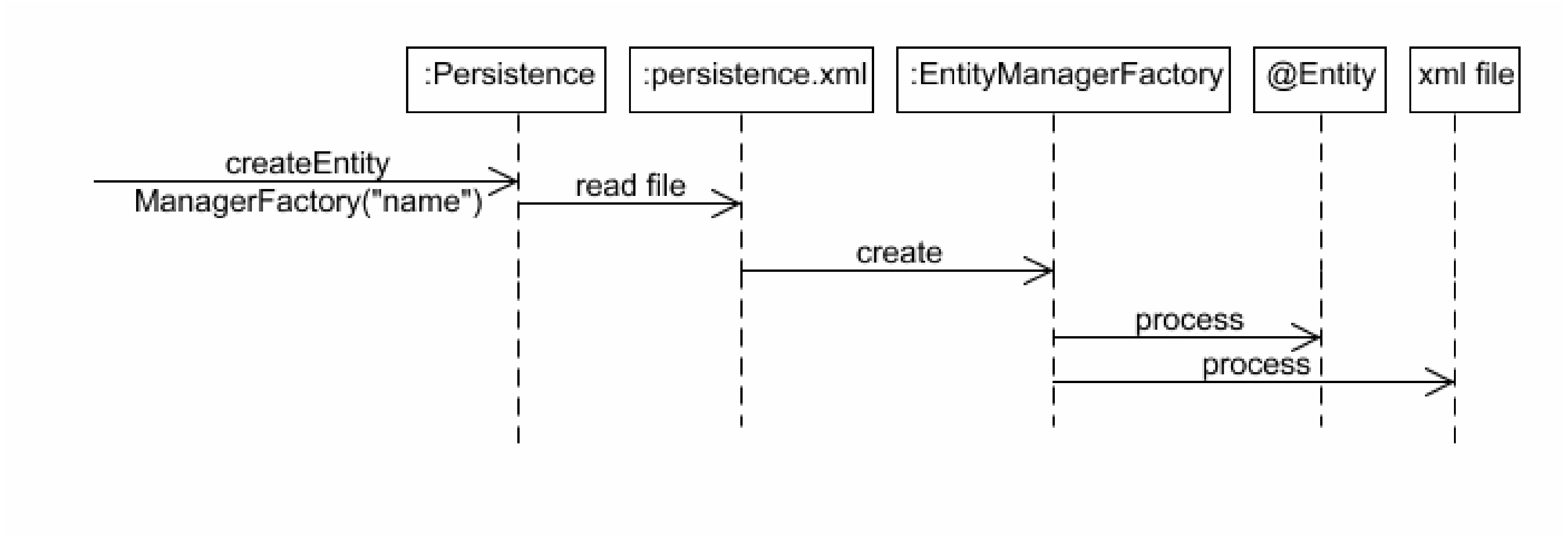


private static EntityManagerFactory *entityManagerFactory*
= Persistence.*createEntityManagerFactory*("helloworld");

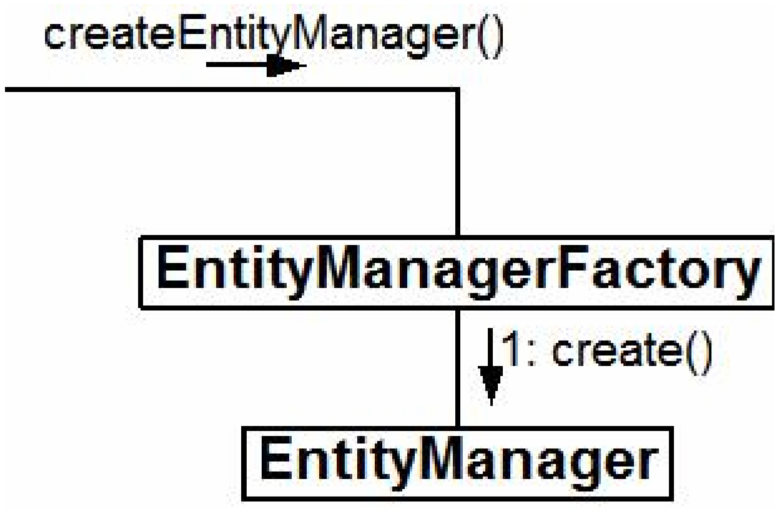
An instance of this class provides a way to create entity managers.

The Entity Manager Factory is the in-memory representation of a Persistence Unit.

Sequence


:Persistence
:persistence.xml
:EntityManagerFactory
@Entity
xml file
createEntity ManagerFactory("name")
read file
create
process
process

Entity Manager



An Entity Manager is **the** interface in your underlying storage mechanism.

It provides methods for persisting, merging, removing, retrieving and querying objects. It is **not** thread safe so we need one per thread.

The Entity Manager also serves as a first level cache.

It maintains changes and then attempts to optimize changes to the database by batching them up when the transaction completes.

EntityManager

- Similar in functionality to Hibernate Session,
- JDO PersistenceManager, etc.

Controls life-cycle of entities

- persist() - insert an entity into the DB
- remove() - remove an entity from the DB
- merge() - synchronize the state of detached entities
- refresh() - reloads state from the database

Types of Entity Managers

- Container-Managed Entity Manager (Java EE environment)
  - Transaction scope entity manager
  - Extended scope entity manager
- Application-Managed Entity Manager (Java SE environment)

Transaction-Scope Entity Manager

- Persistence context is created when a transaction gets started and is removed when the transaction is finished (committed or rolled-back)

  The life-cycle of the persistence context is tied up with transactional scope

Persistence context is propagated

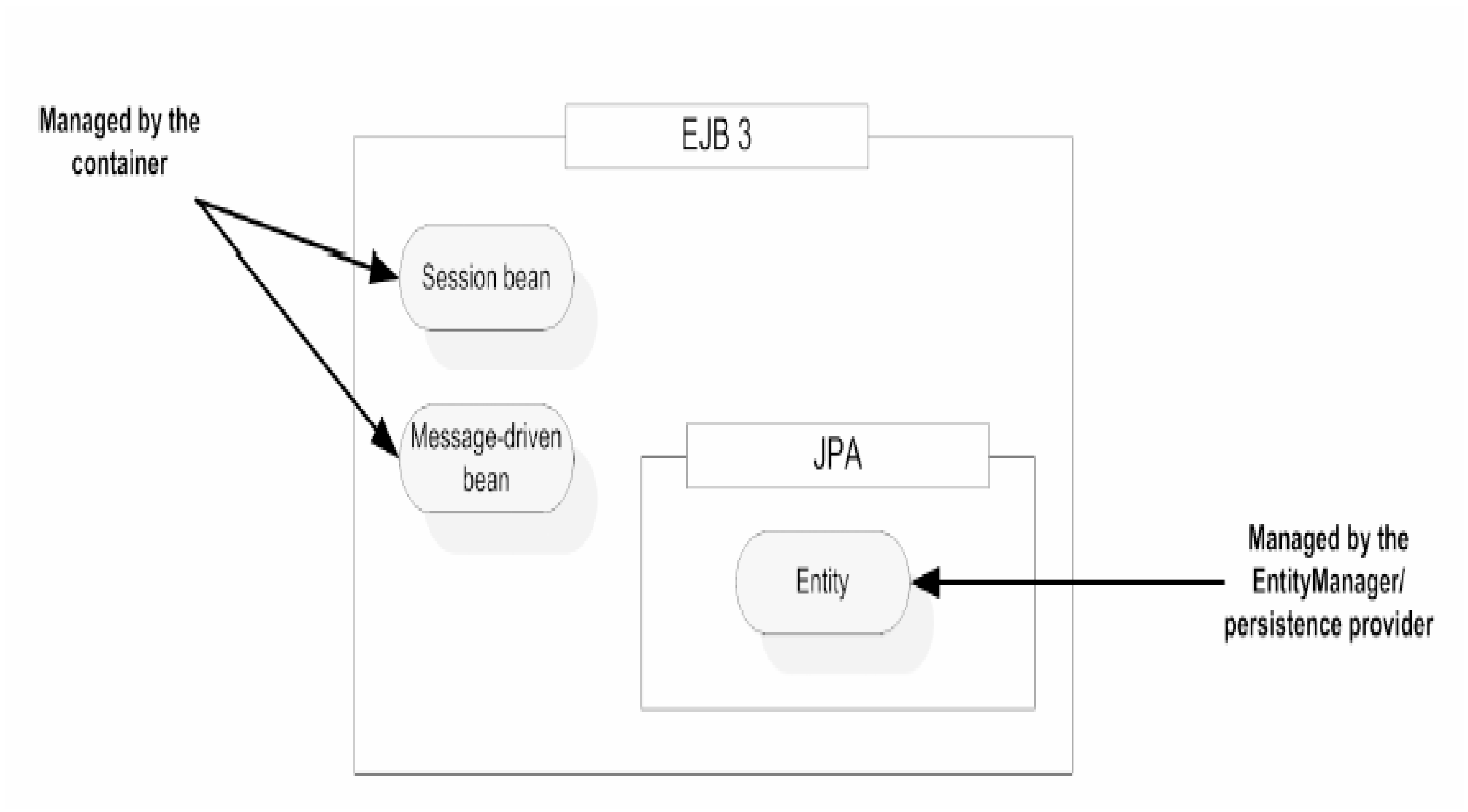
- The same persistence context is used for operations that are being performed in a same transaction

The most common entity manager in Java EE environment

Extended-Scope Entity Manager

- Extended-scope Entity manager work with a single persistent context that is tied to the life-cycle of a stateful session bean

EJB


Managed by the container
EJB 3
Session bean
Message-driven bean
JPA
Entity
Managed by the EntityManager/ persistence provider

jar files

- hibernate3.jar
- hibernate-all.jar
- hibernate-commons-annotations.jar
- hibernate-entitymanager.jar
- thridparty-all.jar
- jboss-ejb3-all.jar
- jboss-archive-browsing.jar
- Add all other required libraries

Entity example

```
@Entity
public class Person  implements Serializable
{
@Id
    private Long id;
    private String firstName;
    private String lastName;
```

- @Entity is required
- Primary key (@Id) is required
- Must be persisted by EntityManager
- Serializable is recommended

Entity example

```
@Entity(name=”USER”)
public class Person  implements Serializable
{
```

Name attribute is used by queries

(SELECT u FROM USER)

Defaults to class name

Primary Key Generation

SEQUENCE indicates that a database sequence should be used to generate the identifier.

```
@Id
        @GeneratedValue(strategy=GenerationType.SEQUENCE)
        private long id;
```

To use a specific named sequence object, whether it is generated by schema generation or already exists in the database, you must define a sequence generator using a @SequenceGenerator annotation.

```
@Id
     @GeneratedValue(generator="InvSeq")
      @SequenceGenerator(name="InvSeq",sequenceName="INV_SEQ",
    allocationSize=5)
     private long id;
```

Primary Key Generation

- **Using Identity Columns**
- When using a database that does not support sequences, but does support identity columns (such as SQL Server database),

```
@Id
  @GeneratedValue(strategy=GenerationType.IDENTITY)
  private long id;
```

Primary Key Generation **Using a Table**

```
@Id
    @GeneratedValue(generator="InvTab")
    @TableGenerator(name="InvTab", table="ID_GEN",
       pkColumnName="ID_NAME", valueColumnName="ID_VAL",
       pkColumnValue="INV_GEN")
    private long id;
```

Table
ID_GEN

ID_NAME	**ID_VAL**
INV_GEN	<last generated value >

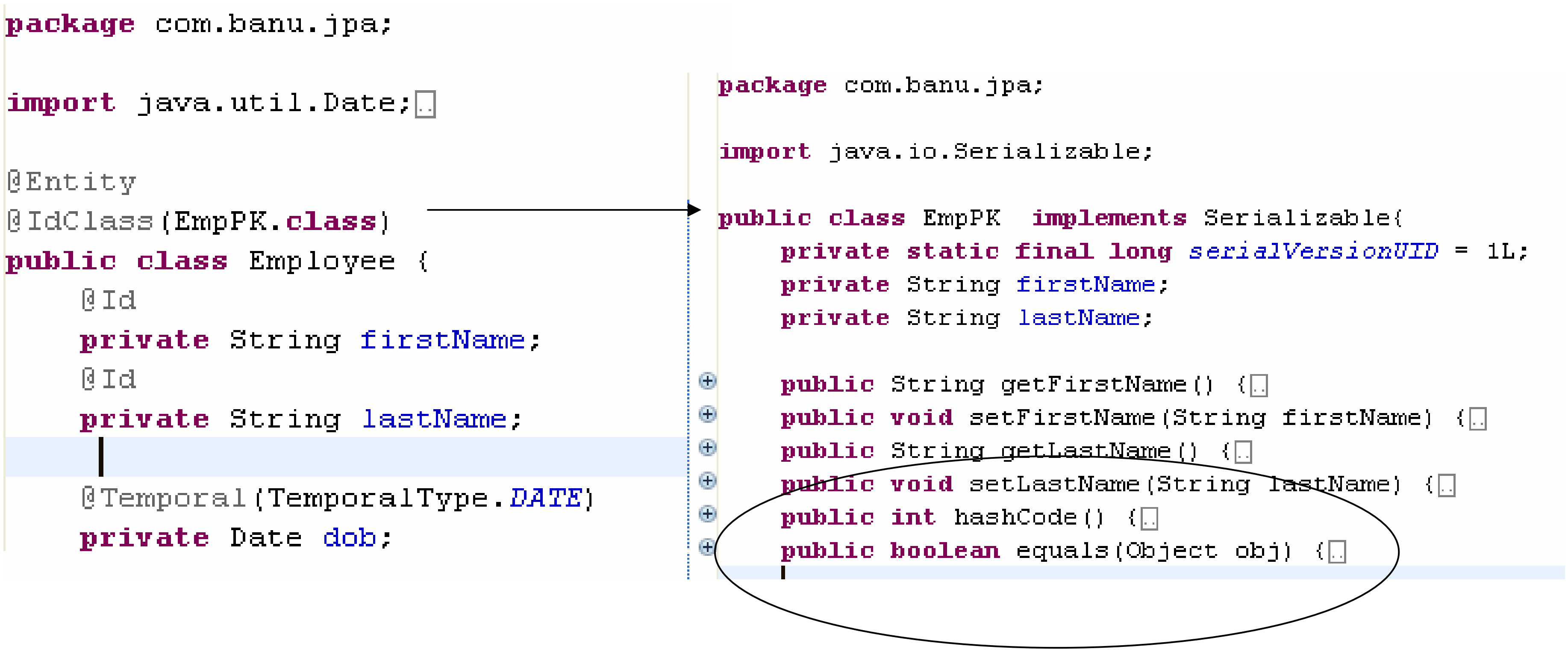
Primary Key Generation

Using a Default Generation Strategy

Provider will select appropriate strategy

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

Primary Key Generation @IdClass

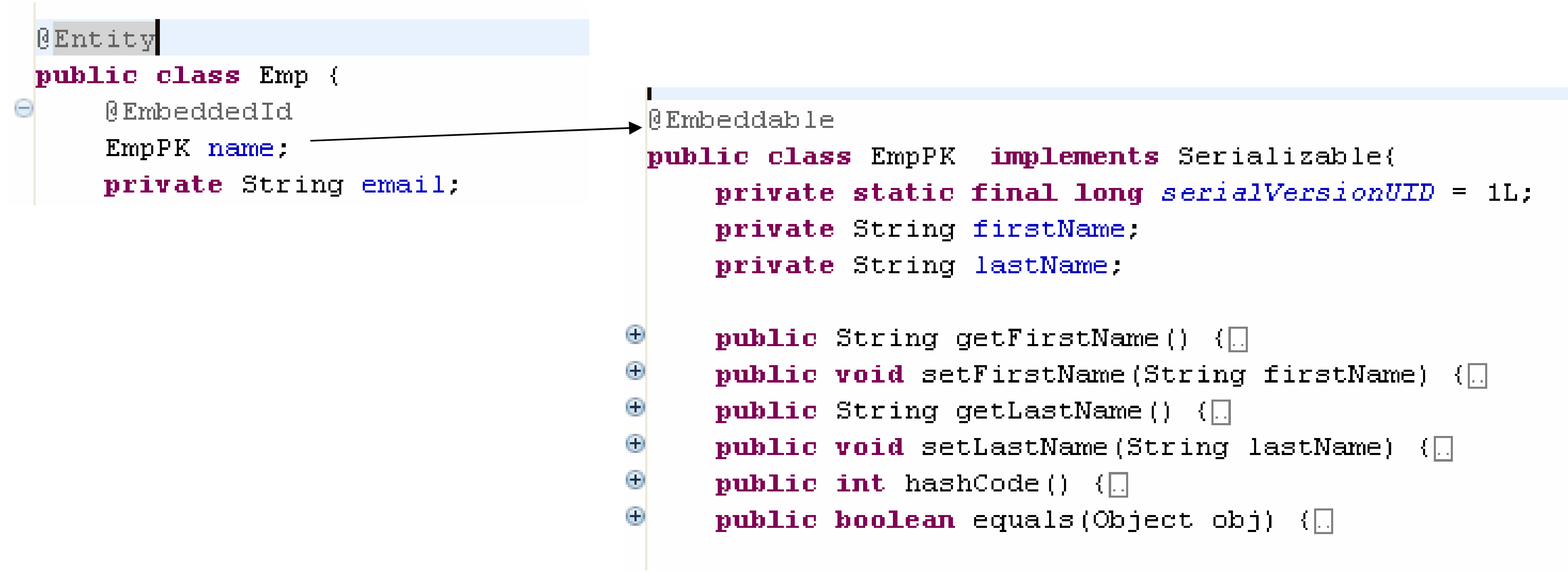


```java
package com.banu.jpa;

import java.util.Date;[..]

@Entity
@IdClass(EmpPK.class)
public class Employee {
    @Id
    private String firstName;
    @Id
    private String lastName;

    @Temporal(TemporalType.DATE)
    private Date dob;
```

```java
package com.banu.jpa;

import java.io.Serializable;

public class EmpPK  implements Serializable{
    private static final long serialVersionUID = 1L;
    private String firstName;
    private String lastName;

    public String getFirstName() {[..]
    public void setFirstName(String firstName) {[..]
    public String getLastName() {[..]
    public void setLastName(String lastName) {[..]
    public int hashCode() {[..]
    public boolean equals(Object obj) {[..]
```

@EmbeddedId

Primary key is formal member of persistent entity



```
@Entity
public class Emp {
    @EmbeddedId
    EmpPK name;
    private String email;
```

```
@Embeddable
public class EmpPK  implements Serializable{
    private static final long serialVersionUID = 1L;
    private String firstName;
    private String lastName;

    public String getFirstName() {
    public void setFirstName(String firstName) {
    public String getLastName() {
    public void setLastName(String lastName) {
    public int hashCode() {
    public boolean equals(Object obj) {
```

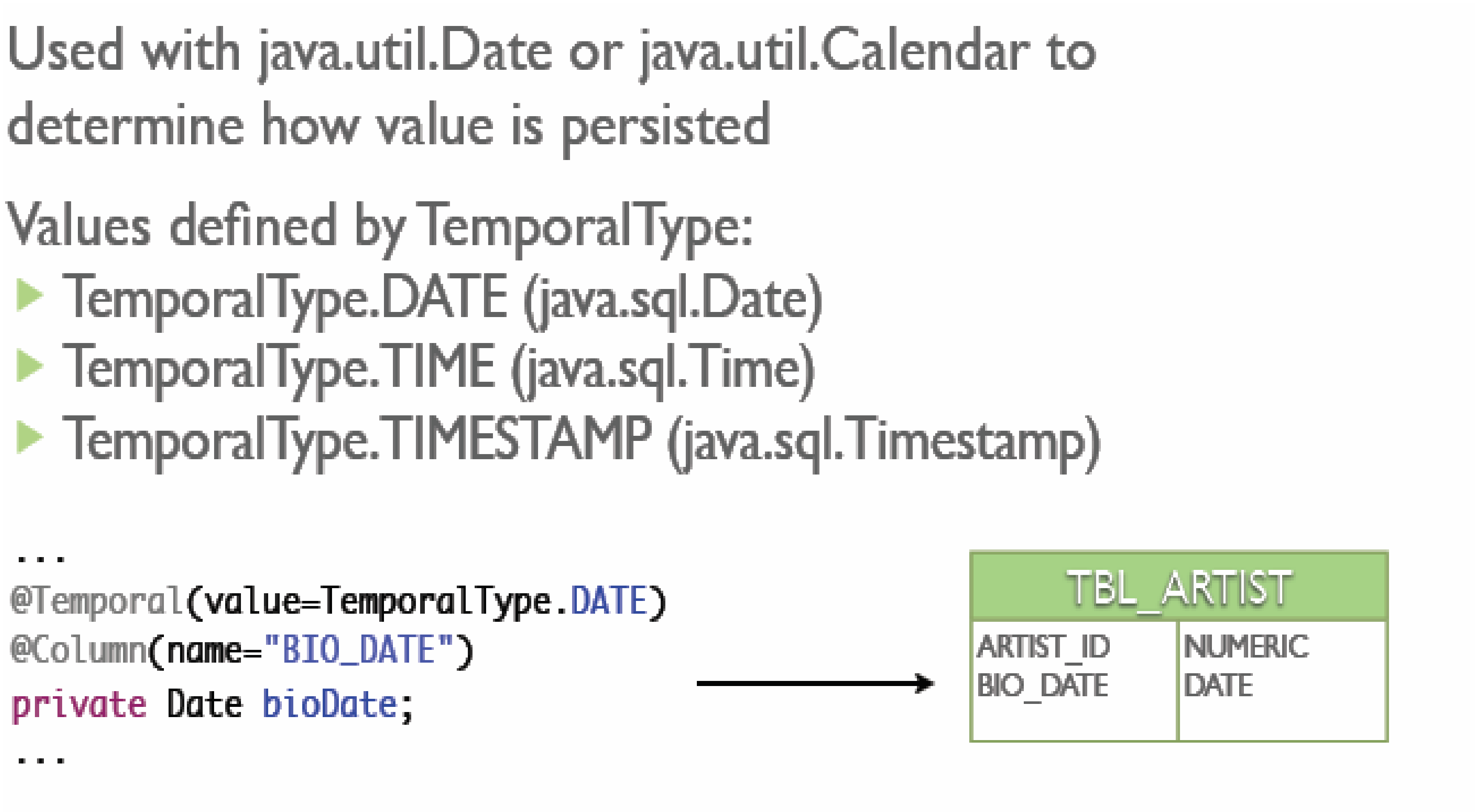
@Column and @Table

- @Column annotation is used to fine-tune the relational database column for field

```
@Id
@Column(name="ITEM_ID", insertable=false, updatable=false)
    private long id;
```

```
@Entity
@Table(name="ORDER_TABLE")
public class Order { ... }
```

@Temporal



Used with java.util.Date or java.util.Calendar to determine how value is persisted

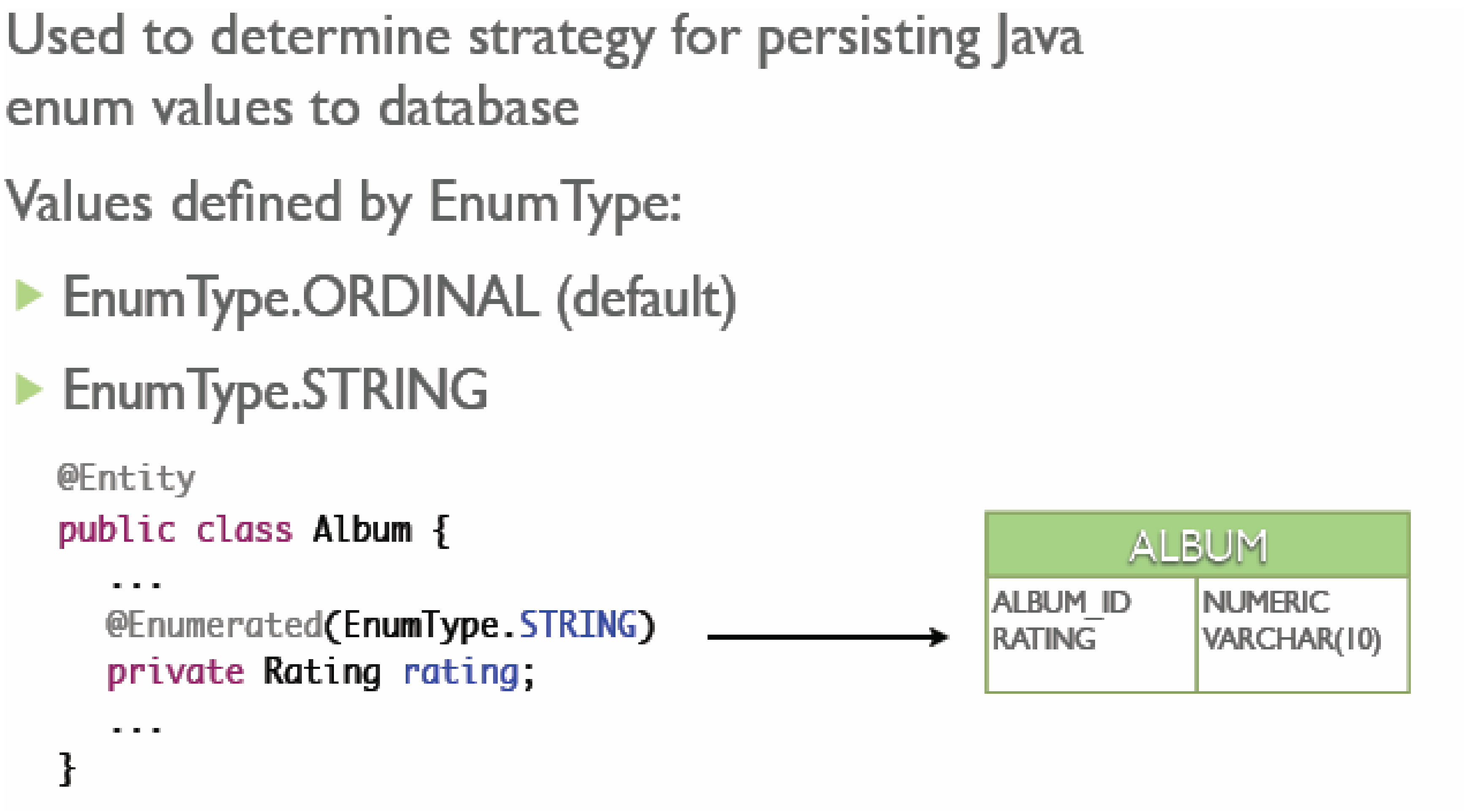
Values defined by TemporalType:

- TemporalType.DATE (java.sql.Date)
- TemporalType.TIME (java.sql.Time)
- TemporalType.TIMESTAMP (java.sql.Timestamp)

```
...
@Temporal(value=TemporalType.DATE)
@Column(name="BIO_DATE")
private Date bioDate;
...
```

TBL_ARTIST	
ARTIST_ID	NUMERIC
BIO_DATE	DATE

@Enumerated



Used to determine strategy for persisting Java enum values to database

Values defined by EnumType:

- EnumType.ORDINAL (default)
- EnumType.STRING

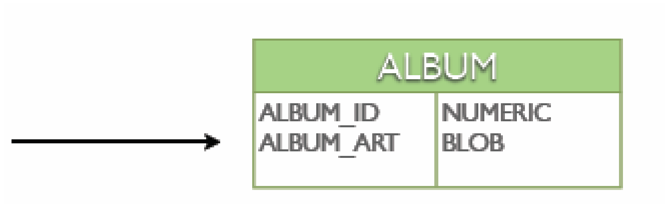
```
@Entity
public class Album {
   ...
   @Enumerated(EnumType.STRING)
   private Rating rating;
   ...
}
```

ALBUM	
ALBUM_ID	NUMERIC
RATING	VARCHAR(10)

@Lob

Used to persist values to BLOB/CLOB fields

```
@Entity
public class Album {
...
@Lob
 private byte[] artwork;
       ...
}
```

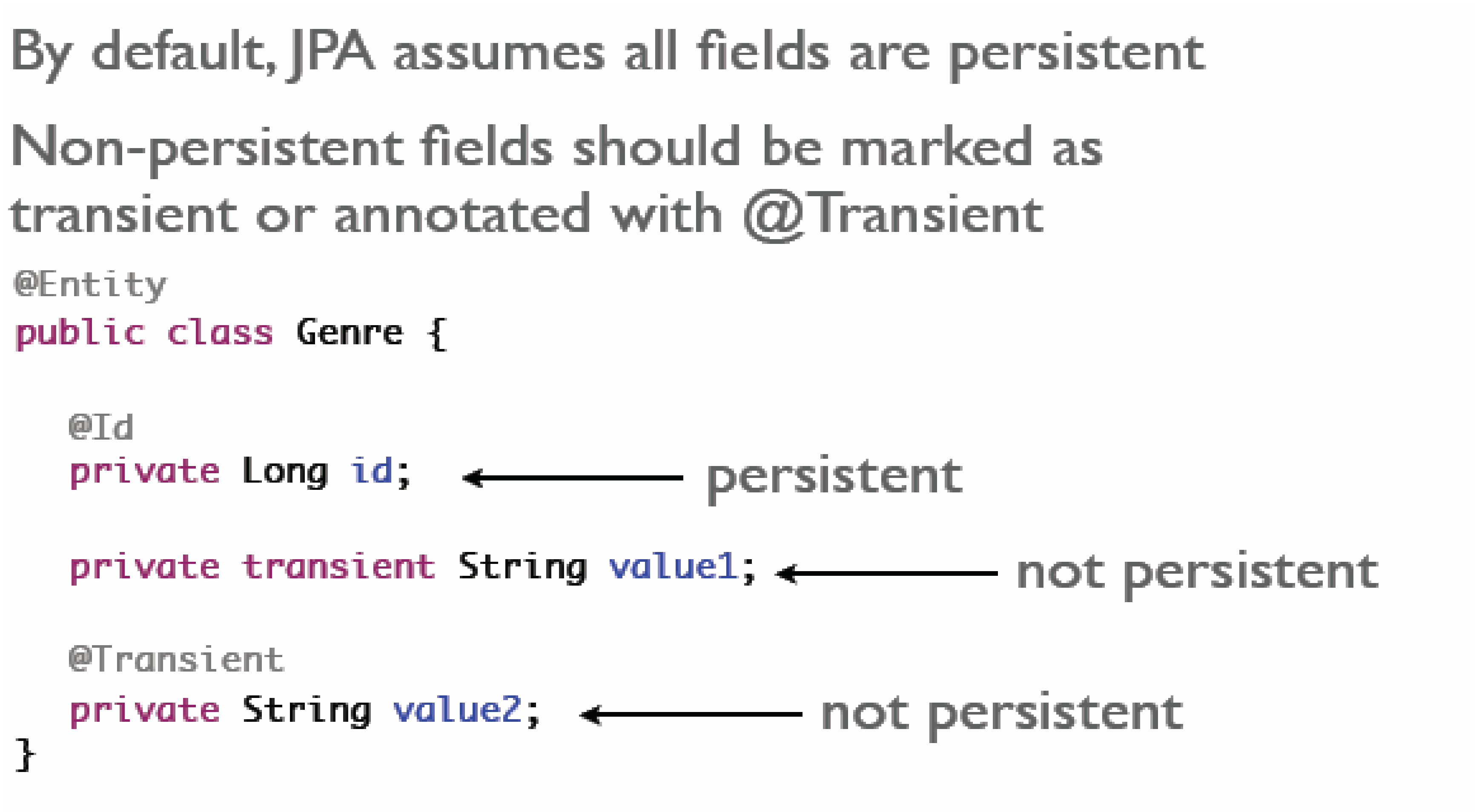


@Version

- JPA has automatic versioning support to assist optimistic locking
- Version field should not be modified by the application
- Value can be primitive or wrapper type of short,int, long or java.sql.Timestamp field

```
@Version
private Integer version;
```

@Transient

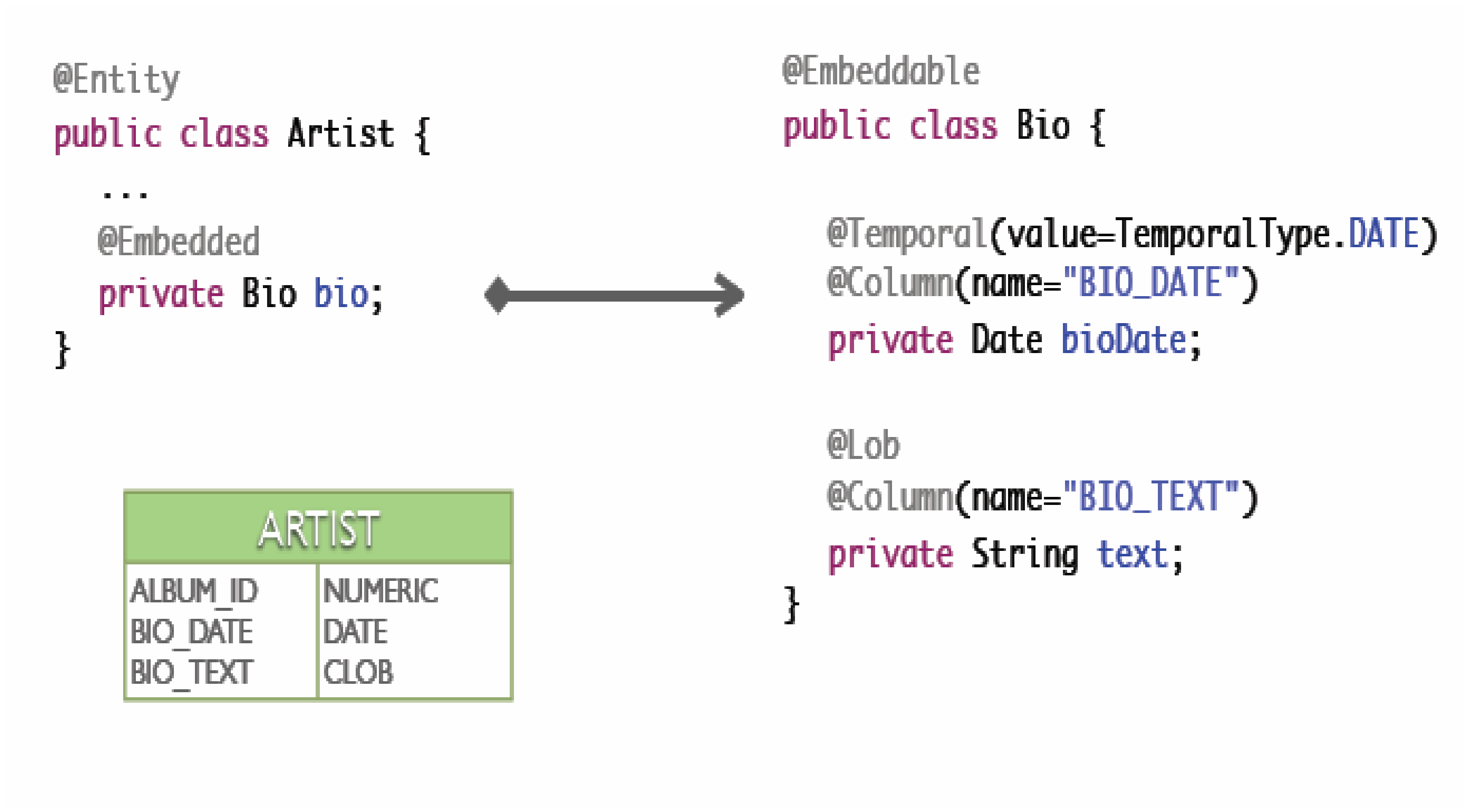


By default, JPA assumes all fields are persistent

Non-persistent fields should be marked as transient or annotated with @Transient

```
@Entity
public class Genre {

   @Id
   private Long id;  ←——— persistent

   private transient String value1; ←——— not persistent

   @Transient
   private String value2; ←——— not persistent
}
```

@Embedded and @Embeddable



```
@Entity
public class Artist {
   ...
   @Embedded
   private Bio bio;
}
```

```
@Embeddable
public class Bio {

   @Temporal(value=TemporalType.DATE)
   @Column(name="BIO_DATE")
   private Date bioDate;

   @Lob
   @Column(name="BIO_TEXT")
   private String text;
}
```

ARTIST	
ALBUM_ID	NUMERIC
BIO_DATE	DATE
BIO_TEXT	CLOB

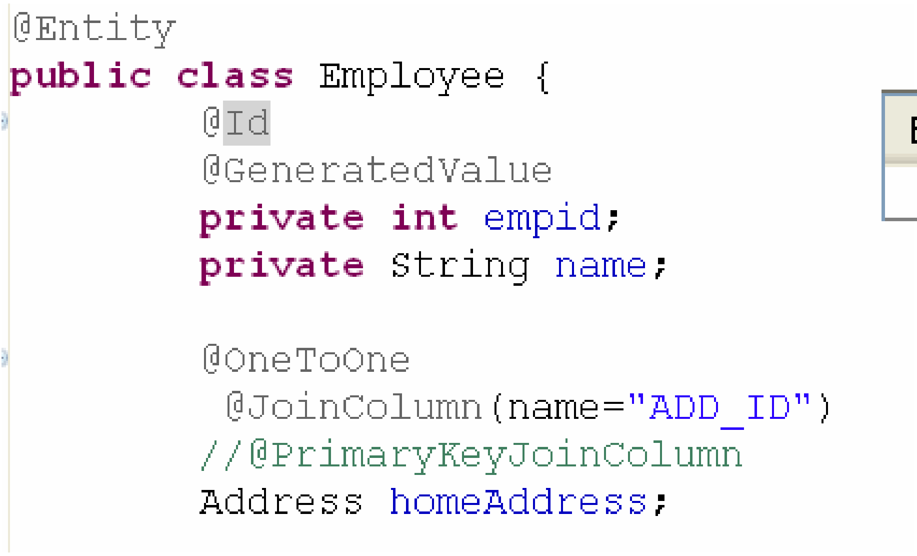
Annotating Relationships

Relationships

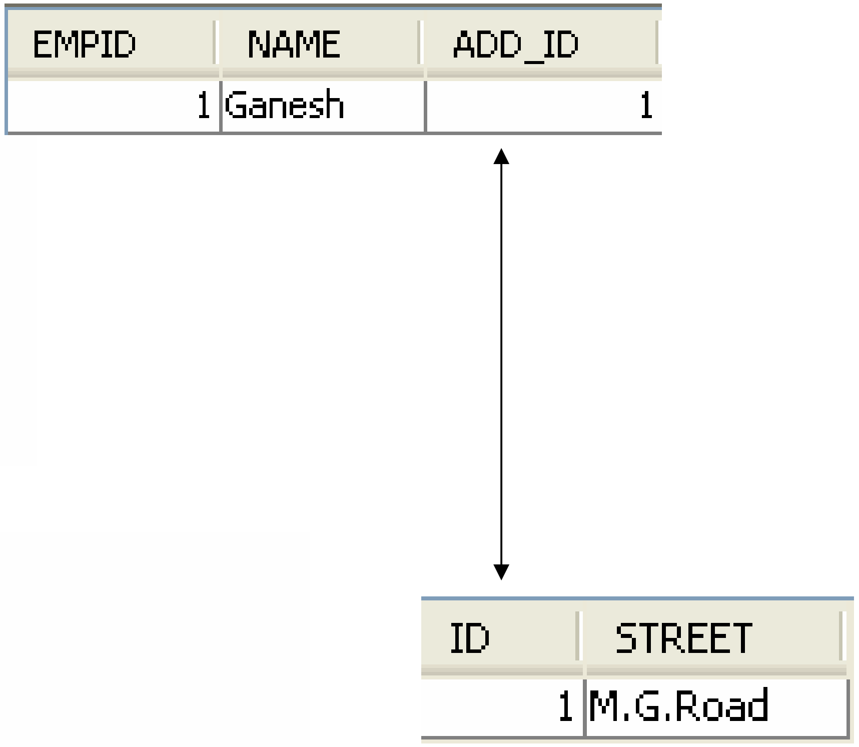
- JPA supports all standard relationships
  - One-To-One
  - One-To-Many
  - Many-To-One
  - Many-To-Many

- Supports unidirectional and bidirectional relationships

@OneToOne using @JoinColumn

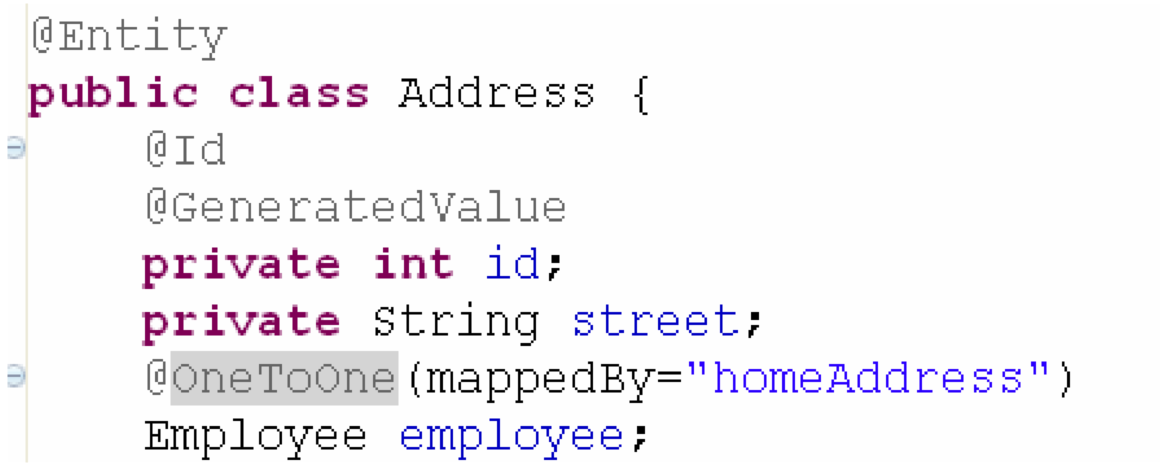


```
@Entity
public class Employee {
        @Id
        @GeneratedValue
        private int empid;
        private String name;

        @OneToOne
         @JoinColumn(name="ADD_ID")
        //@PrimaryKeyJoinColumn
        Address homeAddress;
```



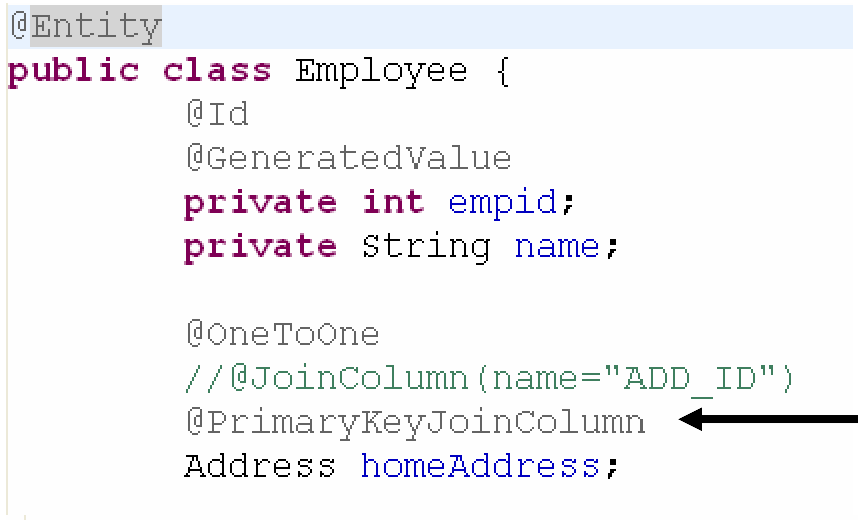
EMPID	NAME	ADD_ID
1	Ganesh	1

ID	STREET
1	M.G.Road

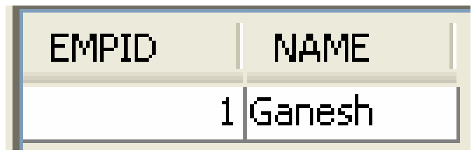


```
@Entity
public class Address {
     @Id
     @GeneratedValue
     private int id;
     private String street;
     @OneToOne(mappedBy="homeAddress")
     Employee employee;
```

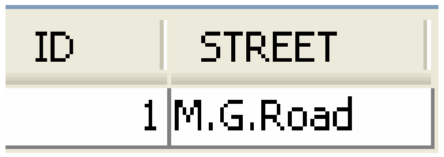
@OneToOne @PrimaryKeyJoinColumn



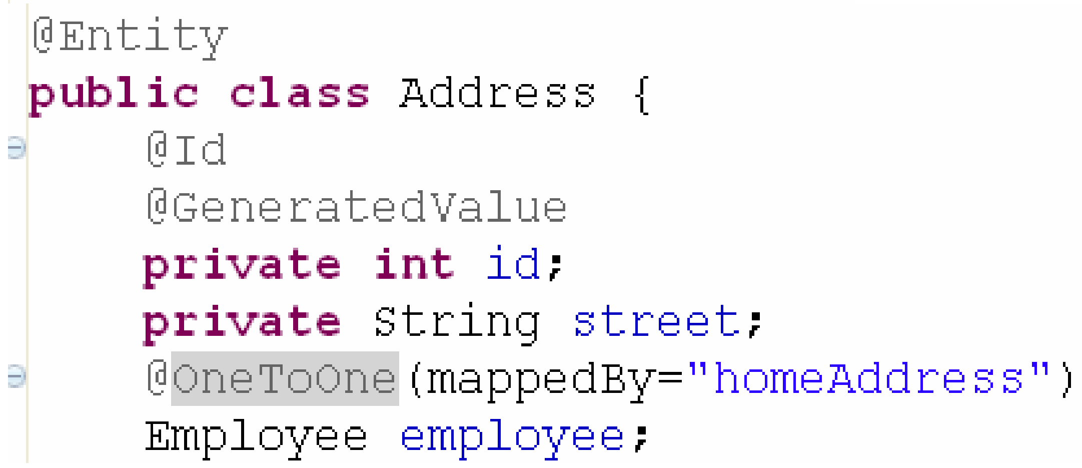
```
@Entity
public class Employee {
        @Id
        @GeneratedValue
        private int empid;
        private String name;

        @OneToOne
        //@JoinColumn(name="ADD_ID")
        @PrimaryKeyJoinColumn
        Address homeAddress;
```



EMPID	NAME
1	Ganesh



ID	STREET
1	M.G.Road



```
@Entity
public class Address {
    @Id
    @GeneratedValue
    private int id;
    private String street;
    @OneToOne(mappedBy="homeAddress")
    Employee employee;
```

CASCADE types

- PERSIST: When the owning entity is persisted, all its related data is also persisted.
- MERGE: When a detached entity is merged back to an active persistence context, all its related data is also merged.
- REMOVE: When an entity is removed, all its related data is also removed.
- ALL: All the above applies.

@OneToMany

- @OneToMany defines the *one* side of a one-tomany relationship
- The *mappedBy* element of the annotation defines the object reference used by the *child* entity
- @OrderBy defines an collection ordering required when relationship is retrieved
- The child (many) side will be represented using an implementation of the java.util.Collection interface

One-to-Many unidirectional

```
@Entity()
public class Trainer {
    private Integer id;
    private String name;
    private Set<Course> courses;

@OneToMany
@JoinColumn(name="trainer_id")
    public Set<Course> getCourses() {
    return courses;
    }

@Entity
public class Course {
    private Integer id;
    private String name;
```

```
Trainer trainer = new Trainer();
                trainer.setName( "Banu Prakash" );
                session.persist( trainer );

 Course c1 = new Course( );
 c1.setName("Java");

 Course c2 = new Course( );
 c2.setName("Hibernate");

 Course c3 = new Course ();
 c3.setName("Spring");


 trainer.setCourses(new HashSet<Course>() );
                trainer.getCourses().add(c1);
                trainer.getCourses().add(c2);
                trainer.getCourses().add(c3);

session.persist(c1);
 session.persist(c2);
 session.persist(c3);
```

Not Required if @OneToMany(cascade = CascadeType.*ALL*)

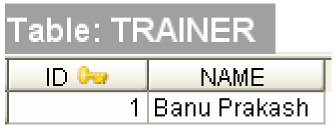


Table: TRAINER

ID	NAME
1	Banu Prakash

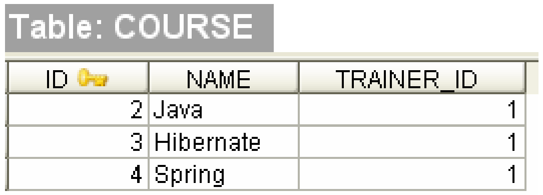


Table: COURSE

ID	NAME	TRAINER_ID
2	Java	1
3	Hibernate	1
4	Spring	1

One-to-Many unidirectional Without Join Column

```
@Entity()
public class Trainer {
    private Integer id;
    private String name;
    private Set<Course> courses;

@OneToMany
//@JoinColumn(name="trainer_id")
public Set<Course> getCourses() {
    return courses;
    }

@Entity
public class Course {
    private Integer id;
    private String name;
```

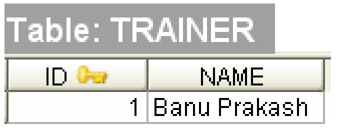


Table: TRAINER

ID	NAME
1	Banu Prakash

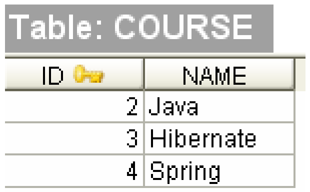


Table: COURSE

ID	NAME
2	Java
3	Hibernate
4	Spring

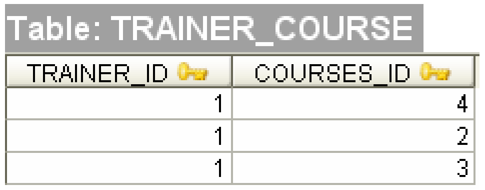


Table: TRAINER_COURSE

TRAINER_ID	COURSES_ID
1	4
1	2
1	3

One-to-many Bidirectional

```
@Entity public class Customer {
@Id String name;

@OneToMany(mappedBy = "customer", cascade = CascadeType.ALL)
Set<Order> orders = new HashSet<Order>();
```

```
@Entity
@Table(name ="OrderTable")
public class Order {
@Id
@GeneratedValue(strategy=GenerationType.SEQUENCE)
int orderId;

@ManyToOne
@JoinColumn(name = "customer_fk")
Customer customer;
```

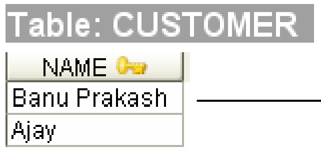


Table: CUSTOMER

NAME
Banu Prakash
Ajay

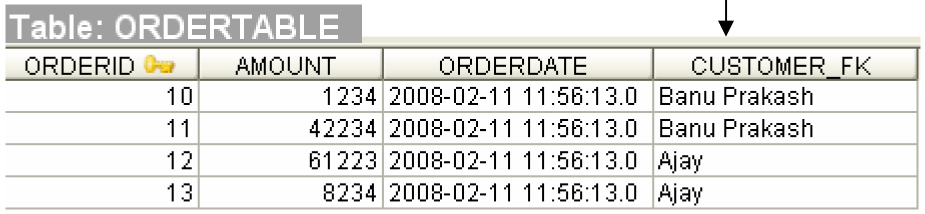


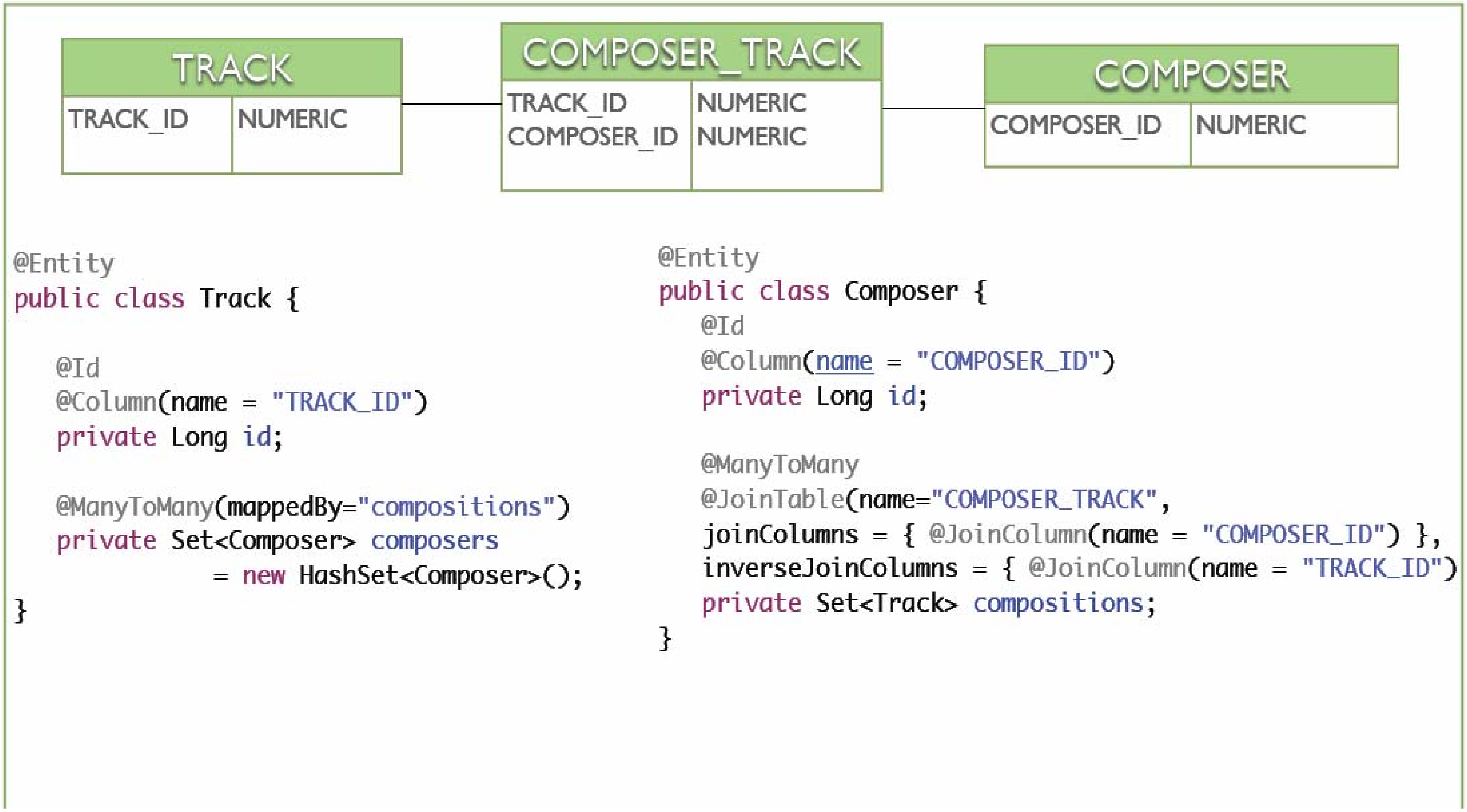
Table: ORDERTABLE

ORDERID	AMOUNT	ORDERDATE	CUSTOMER_FK
10	1234	2008-02-11 11:56:13.0	Banu Prakash
11	42234	2008-02-11 11:56:13.0	Banu Prakash
12	61223	2008-02-11 11:56:13.0	Ajay
13	8234	2008-02-11 11:56:13.0	Ajay

FETCH types

- FetchType.EAGER
  - Ex :
    - @OneToMany(mappedBy = "registration", fetch = FetchType.EAGER)
- FetchType.LAZY
- With the FETCH type set to LAZY, we'd have to make repetitive calls to the database to obtain data

@ManyToMany



TRACK	
TRACK_ID	NUMERIC

COMPOSER_TRACK	
TRACK_ID	NUMERIC
COMPOSER_ID	NUMERIC

COMPOSER	
COMPOSER_ID	NUMERIC

```
@Entity
public class Track {

    @Id
    @Column(name = "TRACK_ID")
    private Long id;

    @ManyToMany(mappedBy="compositions")
    private Set<Composer> composers
                = new HashSet<Composer>();
}
```

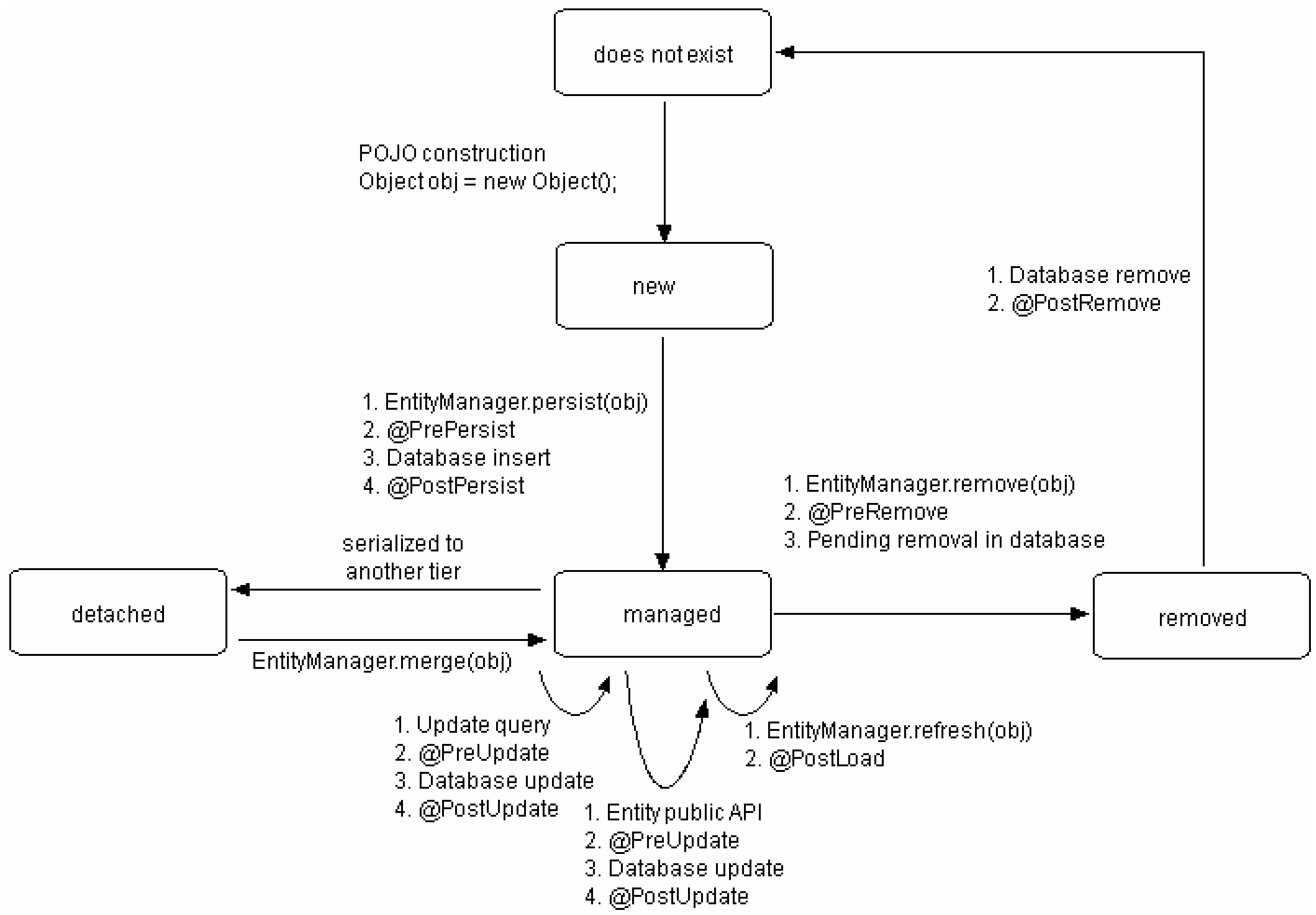
```
@Entity
public class Composer {
    @Id
    @Column(name = "COMPOSER_ID")
    private Long id;

    @ManyToMany
    @JoinTable(name="COMPOSER_TRACK",
    joinColumns = { @JoinColumn(name = "COMPOSER_ID") },
    inverseJoinColumns = { @JoinColumn(name = "TRACK_ID")
    private Set<Track> compositions;
}
```

Lifecycle Callbacks

- PrePersist
- PostPersist
- PreRemove
- PostRemove
- PreUpdate
- PostUpdate
- PostLoad

```
@Entity
 public class Account
     {
     …..
      @PrePersist
     protected void validateCreate()
     {
               if (getBalance() < MIN_REQUIRED_BALANCE)
               {
                              throw new
                              AccountException("Insufficient balance to open an account");
               }
      }

 }
```

JPA Entity Lifecycle Callback Event Annotations



Using Entity Listeners

```
  @Entity
@EntityListeners({ MagazineLogger.class, ... })
public class Magazine { // ... // }

 /** * Example entity listener. */
public class MagazineLogger
      {
                @PostPersist
                public void logAddition (Object pc)
                {
                                debug ("Added :" + ((Magazine) pc).getTitle ());
      }
      @PreRemove
      public void logDeletion (Object pc)
                {
                                debug ("Removing :" + ((Magazine) pc).getTitle ());
                }
                }
```

JPQL SELECT Queries

SELECT Syntax

```
SELECT [<result>]
[FROM <candidate-class(es)>]
[WHERE <filter>]
[GROUP BY <grouping>]
[HAVING <having>]
[ORDER BY <ordering>]
```

```
Query q = em.createQuery("SELECT p FROM Person p WHERE
    p.lastName = 'Jones'");
List results = (List)q.getResultsList();
```

Input Parameters

- **Named Parameters :**

  Query q = em.createQuery("SELECT p FROM Person p WHERE p.lastName = :surname AND o.firstName = :forename");

  q.setParameter("surname", theSurname);
  q.setParameter("forename", theForename");

  Numbered Parameters :

  Query q = em.createQuery("SELECT p FROM Person p WHERE p.lastName = ?1 AND p.firstName = ?2");
  q.setParameter(1, theSurname);

  q.setParameter(2, theForename);

Range of Results

```
Query q =
    em.createQuery("SELECT p FROM Person p WHERE p.age >  20");
    q.setFirstResult(0);
    q.setMaxResults(20);
```

JPQL DELETE Queries

```
    DELETE FROM [<candidate-class>] [WHERE <filter>]

    Query query = em.createQuery("DELETE FROM Product p");
    int number = q.executeUpdate();
```

JPQL UPDATE Queries

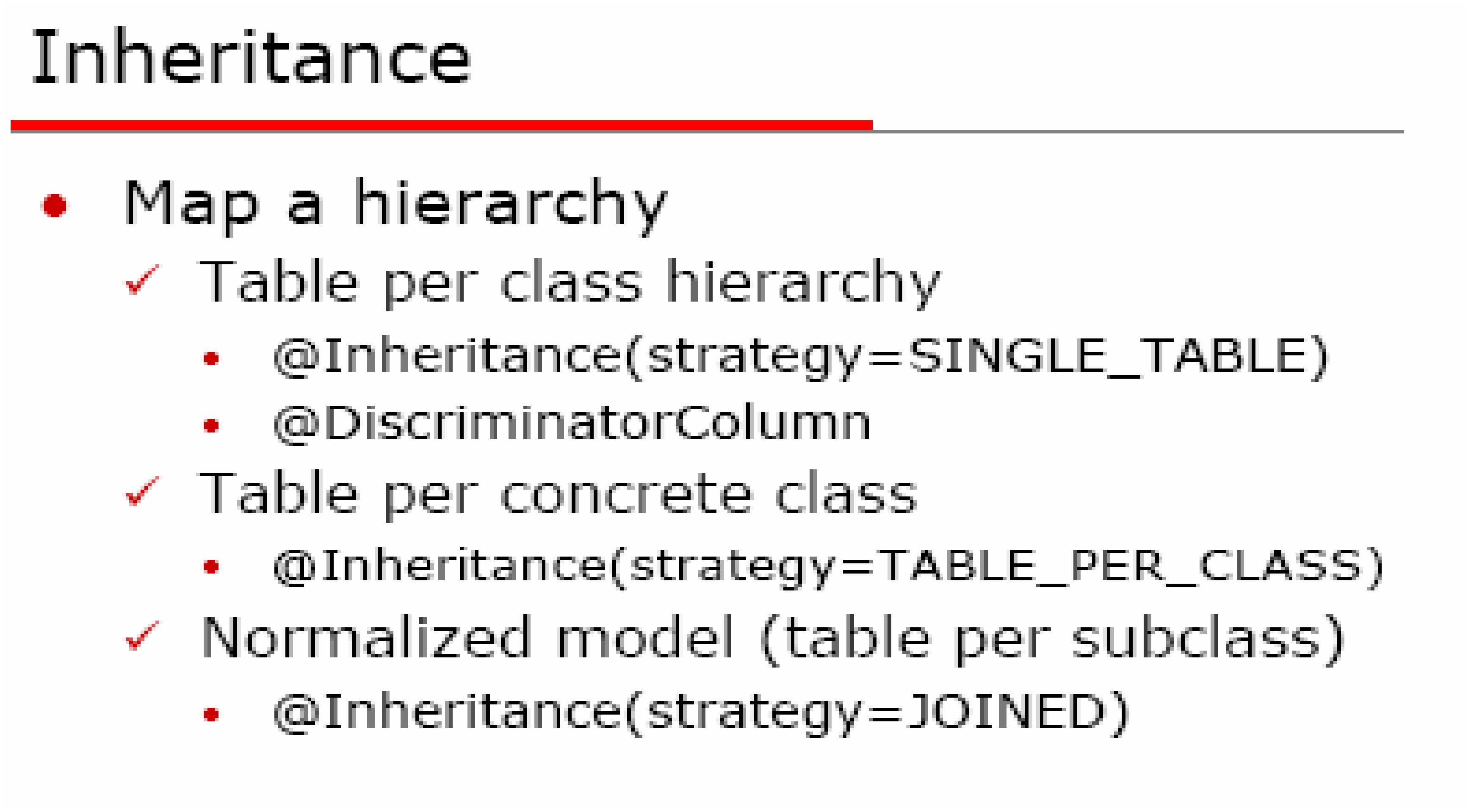
```
    UPDATE [<candidate-class>] SET item1=value1, item2=value2
        [WHERE <filter>]
```

Named Queries

```
@Entity
    @NamedQueries(
    {
    @NamedQuery(name="magsOverPrice", query="SELECT x FROM Magazine x
    WHERE x.price > ?1"),
    @NamedQuery(name="magsByTitle", query="SELECT x FROM Magazine x WHERE
    x.title = :titleParam")
    })
     public class Magazine { ... }
```

```
EntityManager em = ...
    Query q = em.createNamedQuery ("magsOverPrice");
    q.setParameter (1, 5.0f);
    List<Magazine> results =
                    (List<Magazine>) q.getResultList ();
```

Inheritance



Inheritance

- Map a hierarchy
  - ✓ Table per class hierarchy
    - @Inheritance(strategy=SINGLE_TABLE)
    - @DiscriminatorColumn
  - ✓ Table per concrete class
    - @Inheritance(strategy=TABLE_PER_CLASS)
  - ✓ Normalized model (table per subclass)
    - @Inheritance(strategy=JOINED)

Typical Session Bean

```
@Stateless @TransactionAttribute(REQUIRED)
public EditDocumentBean implements EditDocument {
    @PersistenceContext(name="sample")
    private EntityManager em;

    public Document get(Long id) {
        return em.find(Document.class, id);
    }

    public Document save(Document doc) {
        return em.merge(doc);
    }
}
```

JBoss Embeddable

- I want Java EE ease of use in Java SE
- JBoss Embeddable runs in
  - ✓ Unit tests
  - ✓ Main apps
  - ✓ Weblogic
  - ✓ Websphere
  - ✓ Tomcat
- JBoss Embeddable is
  - ✓ EJB3 container
  - ✓ JTA
  - ✓ ...