JPA: Java Persistance API*

Topics:

- Become acquainted with the Java Persistence API (JPA)
- Compare Hibernate and JPA
- Learn how to setup and use Hibernate as a JPA provider

* This section supposes that you are familiar with Hibernate concepts

JPA: Goal

- Part of Java Specification Request (JSR) 220
  - ✓ **Original goal:** to simplify EJB CMP entity beans
- Simplifies the development of Java EE /Java SE applications
- Provides a standard persistence API
- Draws upon the best ideas from existing persistence technologies
  - ✓ Hibernate, TopLink, and JDO
- Usable both within Java SE environments as well as Java EE
  - ✓ POJO based
  - ✓ Works with XML descriptors and annotations

JPA: Main Components

- **Entity Classes**
- **Entity Manager**
  - ✓ Persistence Context
- **EntityManagerFactory**
- **EntityTransaction**
- **Persistence Unit**
  - ✓ persistence.xml
- **Java Persistence Query Language (JPAQL)**
  - ✓ Query

JPA- Hibernate: Mapping

- **Entity Classes => Persistent Classes**
- **EntityManagerFactory => SessionFactory**
- **EntityManager => Session**
- **Persistence => Configuration**
- **EntityTransaction => Transaction**
- **Query => Query**
- **Persistence Unit => Hibernate Config**

JPA: Persistence Unit

- Defines all **entity classes** that are managed by JPA

- **Identified in the persistence.xml configuration file**

- Entity classes and configuration files are packaged together
  - ✓ The JAR or directory that contains the persistence.xml is called the root of the persistence unit
  - ✓ **Needs to be inside a META-INF directory**
    - Whether or not inside a jar

JPA Persistence Unit: persistence.xml

```
<persistence xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd"version="1.0">

<persistence-unit name="BankingApp">
<provider> org.hibernate.ejb.HibernatePersistence </provider>
<mapping-file>orm.xml</mapping-file>
<class>courses.hibernate.vo.Account</class>
<class>courses.hibernate.vo.AccountOwner</class>
<class>courses.hibernate.vo.AccountTransaction</class>
<class>courses.hibernate.vo.EBill</class>
<class>courses.hibernate.vo.EBiller</class>
...
```

Used by the EntityManagerFactory

Can be any JPA provider

Knows the Entity mappings
eqv. To .hbm.xml
Optional when using annotations

Entity classes to map
Optional when using hibernate

JPA Persistence Unit: persistence.xml

```
…
<properties>
    <!-- VENDOR SPECIFIC TAGS -->
    <property name="hibernate.connection.driver_class“
       value="oracle.jdbc.driver.OracleDriver"/>
    <property name="hibernate.connection.url"
       value="jdbc:oracle:thin:@localhost:1521:XE"/>
    <property name="hibernate.connection.username" value="lecture10"/>
    <property name="hibernate.connection.password"value="lecture10"/>
    <property name="hibernate.dialect"
       value="org.hibernate.dialect.Oracle10gDialect"/>
    <property name="hibernate.show_sql" value="true"/>
</properties>
</persistence-unit>
</persistence>
```

You can point the hibernate.cfg.xml instead

JPA Persistence Unit: persistence.xml

```
<persistence-unit name="BankingApp">
<properties>
    <property name="hibernate.ejb.cfgfile value="/hibernate.cfg.xml"/>
</properties>
</persistence-unit>
</persistence
```

JPA Persistence Unit: persistence.xml

- **JPA provides for auto detection**
  - ✓ No need to list individual Entity classes in persistence.xml. Looks for annotated classes and mapping files
  - ✓ Specification requires use of <class> tags in non–EE environment, but Hibernate supports the functionality in both (Enabled by default)
  - ✓ Does NOT work with non–JPA Hibernate

```
<persistence-unit name="BankingApp">...
    <property name="hibernate.archive.autodetection"value="class, hbm*"/>
    ...
</persistence-unit>
```

JPA: Entity Classes

- **Managed objects mapped in one of two ways**
  - ✓ Described in the **orm.xml** mapping file
  - ✓ **Marked with annotations in individual classes**
    - Identified as managed with **@Entity**
    - Primary key identified through the **@Id**

JPA: Entity Classes

- **Contains persistent fields or properties**
  - ✓ **Attributes accessed through getters/setters are 'properties'**
  - ✓ Directly accessed attributes are referred to as 'fields'
  - ✓ <u>Can not combine fields and properties in a single entity</u>
  - ✓ Must define ALL attributes in your entity, even if they’re not persisted

- **Mark as ‘transient’ if attribute is not managed**

JPA: Entity Classes

```
<entity-mappings xmlns="http://java.sun.com/xml/ns/persistence/orm"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/persistence/orm_1_0.xsd"
version="1.0">

<persistence-unit-metadata>
    <!-- identifies the orm.xml as the only source for class definition, telling the
       engine to ignore annotations in classes -->
    <xml-mapping-metadata-complete/>
<persistence-unit-defaults>
   <cascade-persist/>  <------------------  Set any defaults across the
                                            persistence unit entities
</persistence-unit-defaults>
</persistence-unit-metadata>
...
```

JPA: orm.xml mapping file

```
</entity-mappings…….>
….
<package>courses.hibernate.vo</package>
<entity class="Account" access="FIELD">
        <table name="ACCOUNT" />
        <attributes>
        <id name="accountId">
                <column name="ACCOUNT_ID" />
                <generated-value strategy="AUTO" />
        </id>
        <basic name="balance" optional="false">
             <column name="BALANCE" />
        </basic>
        <version name="version">
             <column name="VERSION" />
        </version>
        <attributes>
</entity>
</entity-mappings>
```

JPA Annotations

Property Access Vs Field Access

```
@Entity
public class Account {
private long accountId;
...
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
@Column(name="ACCOUNT_ID")
public long getAccountId() {...}
public void setAccountId(long newId) {...}
...
}
```

Account

Property Access Vs Field Access

```
@Entity
public class Account {
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
@Column(name="ACCOUNT_ID")
private long accountId;
...
public long getAccountId() {...}
public void setAccountId(long newId) {...}
...
}
```

JPA: EntityManagerFactory

- **Used to create EntityManager in JavaSE environment**
  - ✓ Similar to Hibernate **SessionFactory**
- Created through a static method on Persistence

```
EntityManagerFactory emf =
   Persistence.createEntityManagerFactory("BankingApp");
```

Remember the name of the persistence unit in persistence.xml file

JPA: EntityManager

- ***Similar to Hibernate Session***
- Creates and removes persistent entity instances
- Finds entities by their primary key
- Allows for data querying
- Interacts with the persistence context

EntityManager: Methods

– clear()	// clears the context
– close()	// closes the manager
– contains()	// checks for existing object
– createNamedQuery()	// create named query
– createNativeQuery()	// create SQL query
– getTransaction()	// returns the current transaction
– lock()	// locks an object
– persist()	// makes an object persisten
– refresh()	// refreshes an object from the database
– remove()	// deletes an object from the database
– find()	// retrieves an object from the database
– setFlushMode()	// like Hibernate, but missing MANUAL

EntityManager: Application-managed context

- Created and destroyed explicitly by the application
- Created through the EntityManagerFactory class

```
EntityManagerFactory emf
    =Persistence.createEntityManagerFactory("BankingApp");

EntityManager em = emf.createEntityManager();
```

EntityManager: Container-managed context

- Used with Enterprise Java Beans

- Automatically propagated to all application components within a single Java Transaction API (JTA) transaction
  - ✓ Need to identify data source in persistence.xml file

- **Injected into classes with (Dependency injection & IOC)**

  @PersistenceContext

  EntityManger em;

EntityManager: Container-managed context

- Example using JPA within EJB3

```
public class AccountSessionBean {
@PersistenceContext
EntityManager em;

public Account getAccount(int accountId){
    Account account = em.find(Account.class, accountId);
    return account;
}
}
```

Note: need to define the **data source** in persistence.xml

EntityManager: Save an Entity

- **In the context of JavaSE applications**

```
public void saveAccount(Account account) {

EntityManager em =JPAUtil.getEntityManager();

EntityTransaction tx = em.getTransaction();

tx.begin();

em.persist(account);

tx.commit();

em.close();
}
```

Equivalent to HibernateUtil class,

Singleton pattern

EntityManager: Find or Remove an Entity

- **Examples**:

Remove

```
public void deleteAccount(Account account) {
EntityManager manager = JPAUtil.getEntityManager();
account = manager.getReference(Account.class, account.getAccountId());
manager.remove(account);
}
```

Find

```
public Account getAccount(long accountId) {
EntityManager manager = JPAUtil.getEntityManager();
Account account = (Account) manager.find(Account.class, accountId);
return account;
}
```

EntityManager: get an Entity

- **Similar to 'load()' method in Hibernate**
  - ✓ Lazily loads using a proxy

```
@PersistenceContext
EntityManager em;
public Account getAccount(int accountId) {
Account account =em.getReference(Account.class, accountId);
return account;
}
```

load() Vs get() in Hibernate?

- load() throws an exception or return a proxy if the object is not found in the cache or in DB
- get() returns null or a FULLY initialized object (performance down!)

EntityManager : Associations

- **Association Multiplicities**

  ✓ 1 - 1 (**one-to-one**)
  ✓ 1 - n (**one-to-many**)
  ✓ n - 1 (**many-to-one**)
  ✓ n - n (**many-to-many**)

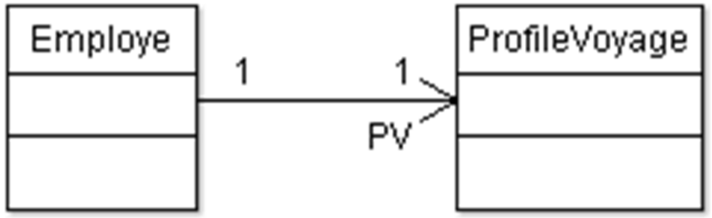
- **Navigability of Associations (Direction)**

  ✓ **Bi-directional** : **two sides** : an **owner** side and an **inverse** side
  ✓ **Unidirectional** : **one side** : the **owner**

Unidirectional : OneToOne

```
@Entity
public class Employe {
private ProfilVoyage pv;
@OneToOne
public ProfilVoyage getPv() {
return pv; }
public void setPv(ProfilVoyage profil) {
this.pv = profil; }
...
}
```

```
@Entity
public class ProfilVoyage
{
...
}
```



Employe entity ➔ ***Employe*** table

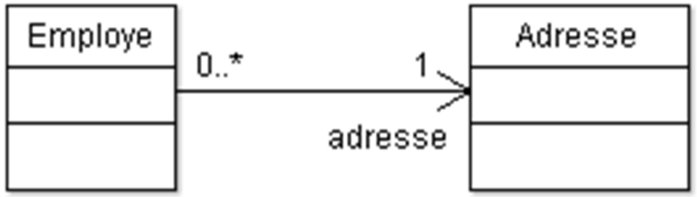
ProfilVoyage entity ➔ ***ProfilVoyage*** table with ***pf_Id*** as **PK**

Employe table owns a foreign key to ***ProfilVoyage***, ***PV***

Unidirectional : ManyToOne

```
@Entity
public class Employe {
private Adresse ad;
@ManyToOne
public Adresse getAd() {
return ad; }
public void setAd(Adresse a) {this.ad = a ; }
...
}
```

```
@Entity
public class Adresse
{
...
}
```



Employe entity ➔ *Employe* table

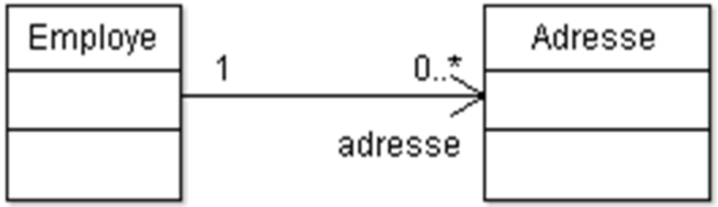
Adresse entity ➔ ***Adresse*** table with ***Id_ad*** as **PK**

Employe table owns a foreign key to ***Adresse*** , ***ad***

Unidirectional : OneToMany

```
@Entity
public class Employe {
private Collection<Adresse> adresses;
@OneToMany
public Collection<Adresse> getAdresses() {return adresses; }
public void setAdresses
        (Collection<Adresse> adresses) {this.adresses = adresses;}
...
}
```

```
@Entity
public class Adresse
{
...
}
```



Employe entity ➔ ***Employe*** table

Adresse entity ➔ ***Adresse*** table with ***Id_ad*** as **PK**

Creation of a join table Employe_Adresse with two columns (i.e. Employe_Pkemploye & Adresse_PKAdresse, each column represents a PK to each table

Unidirectional : ManyToMany

```
@Entity
public class Employe {
private Collection<Adresse> adresses;
@ManyToMany
public Collection<Adresse> getAdresses() {return adresses; }
public void setAdresses
	(Collection<Adresse> adresses) {
this.adresses = adresses; }
...
}
```

```
@Entity
public class Adresse
{
...
}
```

Employe entity ➔ ***Employe*** table

Adresse entity ➔ ***Adresse*** table with ***Id_ad*** as **PK**

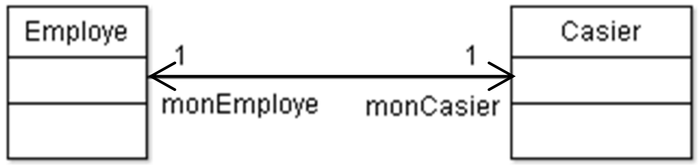
Creation of a join table Employe_Adresse with two columns (i.e. Employe_Pkemploye & Adresse_PKAdresse, each column represents a PK to each table

Bidirectional : OneToOne/OneToOne

```
@Entity
public class Employe {
private Casier monCasier;
@OneToOne
public Casier getMonCasier()
{ return monCasier; }
public void setMoncasier(Casier c)
{ this.monCasier = c; }
...
}
```

```
@Entity
public class Casier {
private Employe monEmploye;
@OneToOne(mappedBy="monCasier")
public Employe getMonEmploye()
{ return monEmploye; }
public void setMonEmploye(Employe e)
{ this.monEmploye = e; }
...
}
```

Employe entity ➔ *Employe* table



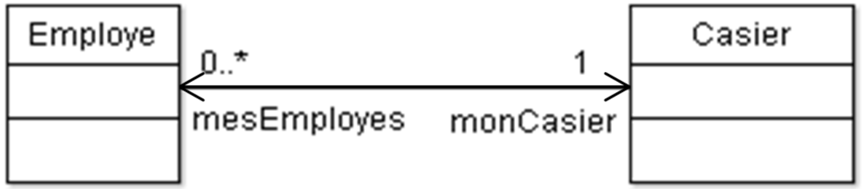
Casier entity ➔ *Casier* table with ***Id_ad*** as **PK**

Employe table owns a foreign key to *Casier* , ***monCasier***

Bidirectional : ManyToOne/OneToMany

```
@Entity
public class Employe {
private Casier monCasier;
@ManyToOne
public Casier getMonCasier()
{ return monCasier; }
public void setMoncasier(Casier c)
{ this.monCasier = c; }
...
}
```

```
@Entity
public class Casier {
private Collection<Employe> mesEmployes;
@OneToMany(mappedBy="monCasier")
public Collection<Employe> getMesEmployes()
{ return mesEmployes; }
public void setMesEmployes
        (Collection<Employe> e)
{ this.mesEmployes = e; }
...
}
```



Employe entity ➔ ***Employe*** table

Casier entity ➔ ***Casier*** table with ***Id_ad*** as **PK**

Employe table owns a foreign key to ***Casier*** , ***monCasier***

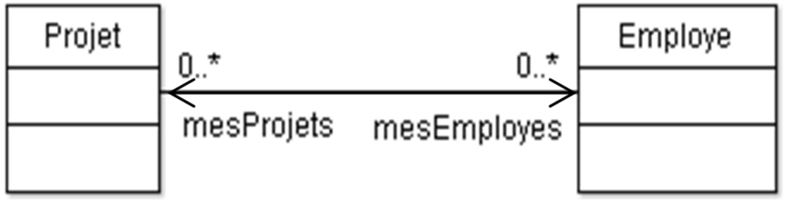
Bidirectional : ManyToMany/ManyToMany

```
@Entity
public class Projet {
Collection<Employe> mesEmployes;
@ManyToMany
public Collection<Employe> getMesEmployes()
{ return mesEmployes; }
public void setMesEmployes
        (Collection<Employe> e)
{ this.mesEmployes = e; }
...
}
```

```
@Entity
public class Employe {
private Collection<Projet> mesProjets;
@ManyToMany(mappedBy= "mesEmployes")
public Collection<Projet> getMesProjets()
{ return mesProjets; }
public void setMesProjets
        (Collection< Projet > p)
{ this.mesProjets = p; }
...
}
```

Projet entity ➔ ***Projet*** table

Employe entity ➔ ***Employe*** table

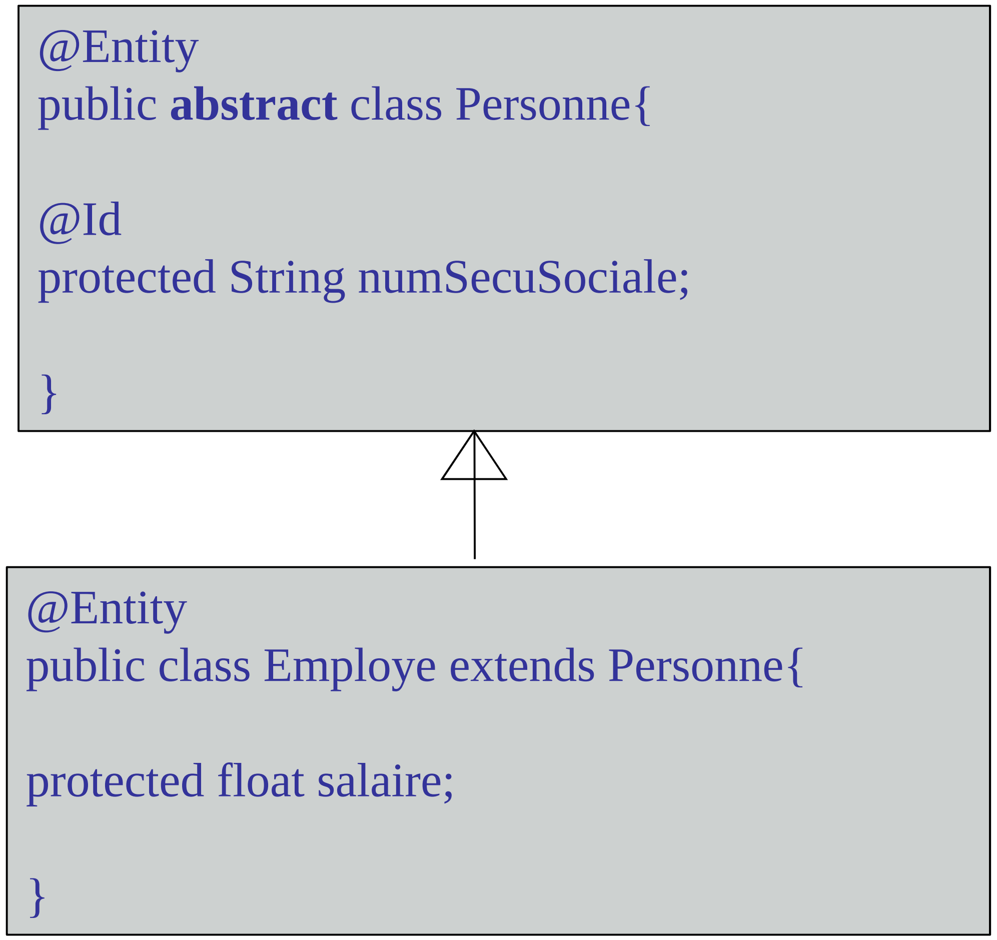


Creation of a join table **Projet_Employe** with two columns (i.e. mesProjets_PKProjet & mesEmployes_Pkemploye, each column represents a PK to each table

JPA: Inheritance

- Entities support inheritance and polymorphism
- Entities may be concrete or abstract
- An Entity can inherit a non-entity class
- A non-entity class can inherit an entity class

Inheriting abstract class



```
@Entity
public abstract class Personne{

@Id
protected String numSecuSociale;

}
```

```
@Entity
public class Employe extends Personne{

protected float salaire;

}
```

JPA: Inheritance Strategies

- :
- One Table by classes hierarchy (Default)

```
@Inheritance(strategy=SINGLE_TABLE)
```

- One Table by concrete class

```
@Inheritance(strategy=TABLE_PER_CLASS)
```

- Join Strategy: a join between the concrete class and the super class tables
  - No duplication of the fields, a Join operation to get the info

```
@Inheritance(strategy=JOINED)
```

JPA: Inheritance Strategies

- One Table by classes hierarchy (Default)
  - ✓ Implemented in most tooling solutions
  - ✓ Good support of polymorphism
  - ✓ Columns proper to sub-classes set at null
- One Table by concrete class
  - ✓ Some issues remain regarding polymorphism
- Join Strategy
  - ✓ Good support of polymorphism
  - ✓ Not always implemented
  - ✓ Join operation can be costly

Inheritance Strategies : One table

- A discriminator column is used

- Possible Types
  - ✓ DiscriminatorType.STRING (Default)
  - ✓ DiscriminatorType.CHAR
  - ✓ DiscriminatorType.INTEGER.

- Example

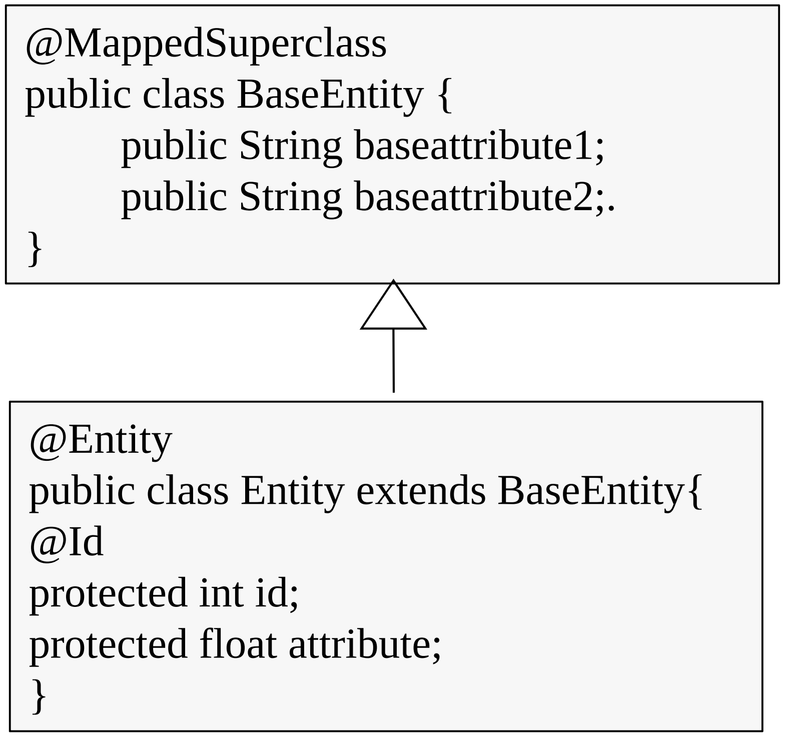
```
@Entity
@DiscriminatorColumn(name="DISCRIMINATEUR_PERSONNE"
discriminatorType=DiscriminatorType.INTEGER)
@DiscriminatorValue("Personne")
public class Personne{
...
}
```

```
@Entity
@DiscriminatorValue("Player")
public class Player extends Personne
{
…}
```

Inheritance : *MappedSuperClass*

- Entities can inherit non persistent entities

- **MappedSuperClasses** are not accessible to the Entity Manager
- Not considered as an Entity (no table in the DB)



```
@MappedSuperclass
public class BaseEntity {
        public String baseattribute1;
        public String baseattribute2;.
}
```

```
@Entity
public class Entity extends BaseEntity{
@Id
protected int id;
protected float attribute;
}
```

Entity : Composed Primary Key

```
public class ClefEtudiant implements
java.io.Serializable{
private String nomId;
private String prenomId;

public String getNomId(){
        return nomId;
}
public void setNomId( String nomId ){
        this.nomId = nomId;
}
public String getPrenomId(){
        return prenomId;
}
public void setPrenomId( String prenomId ){
        this.prenomId = prenomId;
}
public int hashCode(){
        return ...
}
public boolean equals(Object otherOb) {
        ...
}}
```

```
@IdClass(ClefEtudiant.class)
@Entity
public class Etudiant{

private String nomId;
@Id
public String getNomId(){
        return nomId;
}
public void setNomId( String nomId ){
        this.nomId = nomId;
}

private String prenomId;
@Id
public String getPrenomId(){
        return prenomId;
}
public void setPrenom( String prenomId ){
        this.prenomId = prenomId;
}
}
```

Entity: Two classes in one table

- **@Embeddable** & **@Embedded** : fields of two classes into one table

```
@Embeddable
public class Address implements Serializable {
private String rue; private int codePostal;
}
```

```
@Entity
public class User {
private String nom;
@Embedded
private Address adresse;
}
```

JPA : Cascading

- **Achieved through the "cascade" attribute on the multiplicity annotation**

- **Multiple cascading options**
  - ✓ Persist
  - ✓ Merge
  - ✓ Remove
  - ✓ Refresh
  - ✓ All

- **Does not currently provide these Hibernate additional cascading options**
  - ✓ save-update
  - ✓ delete
  - ✓ lock
  - ✓ evict
  - ✓ delete-orphan

JPA : Cascading annotation

```
@Entity
public class Account {
@OneToMany(mappedBy="account",
cascade="CascadeType.REMOVE")
private Set ebills;
...
}
```

JPA Query Language (JPQL)

- Subset of Hibernate Query Language
  - ✓ Same syntax
- Provides the @NamedQuery and @NamedNativeQuery annotations
- Does not support the following:
  - ✓ Updating the version of an entity with the ‘versioning’ keyword
  - ✓ Some batch functionality
  - ✓ Additional syntactical functions available in HQL

JPQL: Query Annotation

```
import javax.persistence.*;
@NamedQueries( {
@NamedQuery(name = "getAllAccounts" query = "from Account")
@NamedQuery(name = "getAccountByBalance"
query = "from Account where
balance = :balance")
})
```

Some Hibernate functions are not provided by JPA

- **CURRENT_DATE(), CURRENT_TIME(), INDEX(joinedCollection), ELEMENTS(c), etc.**

JPA with Hibernate

Does not come with default Hibernate distribution

Additional jar required for compile time

- ✓ javaee.zip
  - Standard jar, downloadable from Java site

Also need to download Hibernate implementation

- ✓ hibernate-entitymanager-3.4.0.ja.zip
- ✓ Contains additional required jars
  - hibernate-entitymanager.jar
  - hibernate-annotations jar
  - hibernate-annotations.jar
  - hibernate-commons-annotations.jar

JPA : Benefits

- **Standardized configuration**
  - ✓ Persistence Unit
- **Standardized data access code, lifecycle, and querying capability that is fully portable**
- **Can override annotations with descriptor file**

JPA : disadvantages

- **Though standard interfaces are nice, some-what lenient spec may present gaps when switching vendor implementations**
  - ✓ Not all inheritance strategies supported
  - ✓ ‘Standardized’ descriptor file is basically a wrapper around vendor specific implementations

Missing some beneficial aspects from Hibernate

- ✓ Query by Example, Query by Criteria (expected later)
- ✓ EntityManager propagation across methods/objects
- ✓ Collection Filters
- ✓ 2nd level Cache
- ✓ Other minor items that developers may come to rely on
  - More-so than with most vendor-specific implementations, the temptation is there to use the vendor-specific features to fill the gap – but then, no longer portable