# Java Persitence API (JPA)

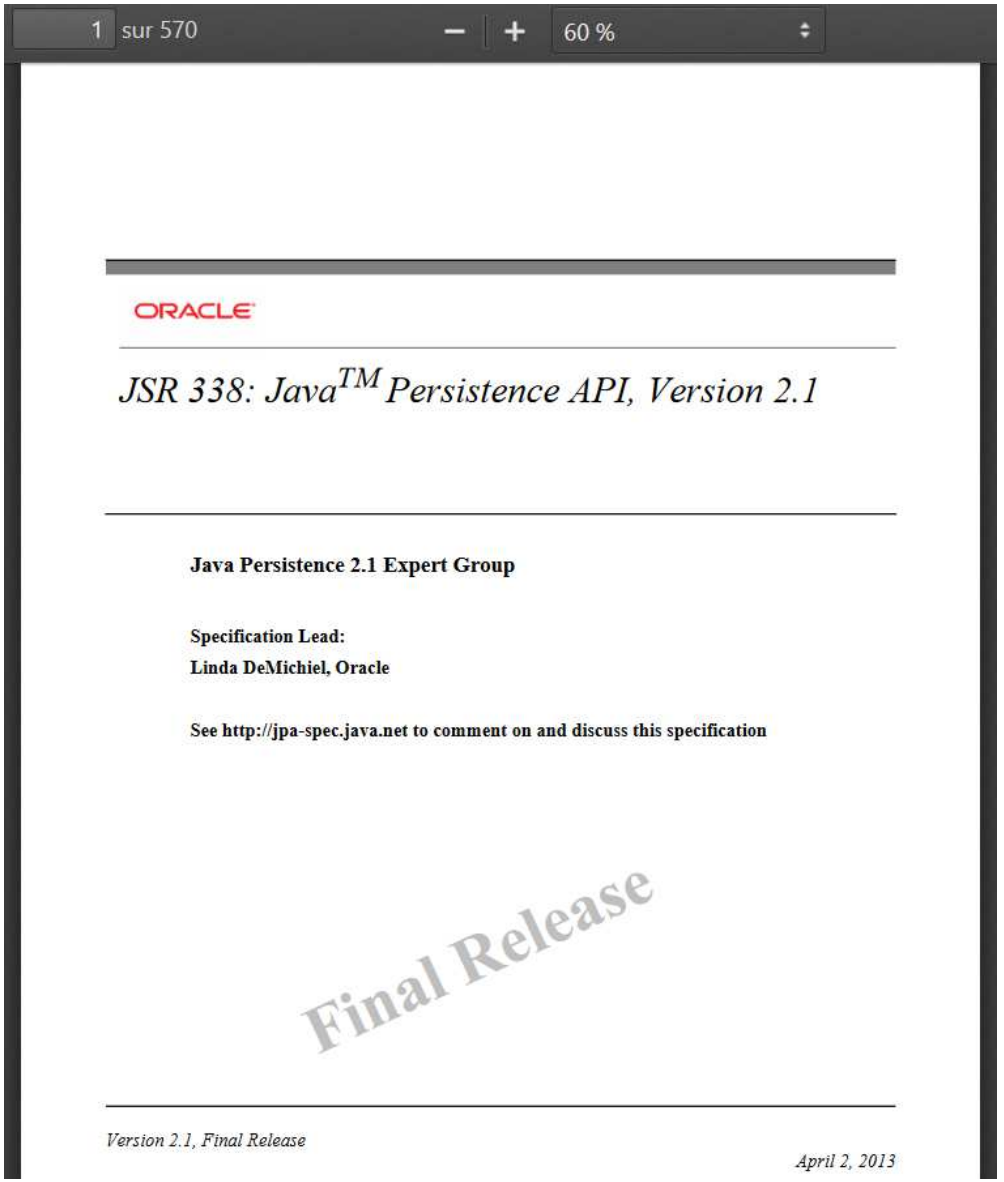
RAMZI FARHAT

- ❑ Introduction
- ❑ Entité
- ❑ Unité de persistance
- ❑ Entité Manager
- ❑ JPQL

# Introduction

## JPA (Java Persistence API)

- API (Application Programming Interface) permettant
  - *Assurer la correspondance Objet/Relationnel (**ORM**)*
  - *Offre un langage d'interrogation objet : **JPQL** (Java Persistence Query Language)*
- Implémentations
  - *EclipseLink*
  - *Hibernate*
  - *TopLink*
  - *etc.*




https://download.oracle.com/otn-pub/jcp/persistence-2_1-fr-eval-spec/JavaPersistence.pdf

# Composantes JPA

- EJB Entity

  Entreprise Java Beans qui contiennent la correspondance O/R via des annotations.

- Unité de persistance

  Fichier de configuration XML permettant de définir la source de données concernée.

- Entity Manager

  Objet responsable des opérations CRUD (Create, Read, Update et Delete) sur les données.

# Entité

- Type d'EJB :
  - Représente les données persistantes dans une BDR (typiquement la classe représente une table relationnelle et une instance représente un enregistrement)
  - Identifié obligatoirement par une clé primaire
- Variantes :
  - BMP (Bean Managed Persistence) : persistance développée par le développeur
  - CMP (Container Managed Persistence) : persistance gérée par le conteneur (via les descripteurs de déploiement)

# Entité

Etudiant

| id | nom | dateNaissance | niveauEtude |
|---|---|---|---|
| | | | |
| | | | |

```
@Entity (name="Etudiant")
public class Etudiant{
     @Id
     @GeneratedValue(strategy = GenerationType.AUTO)
     private Long id;
     @Column (name="nom", length=80)
     private String nom;
     @Column (name="dateNaissance")
     @Temporal (TemporalType.DATE)
     private Date dateNaissance;
     @Column (name="niveauEtude", length=30)
     @Enumerated (EnumType.STRING)
     private NiveauEtude niveauEtude;
     // Getters and Setters
     …
}
```

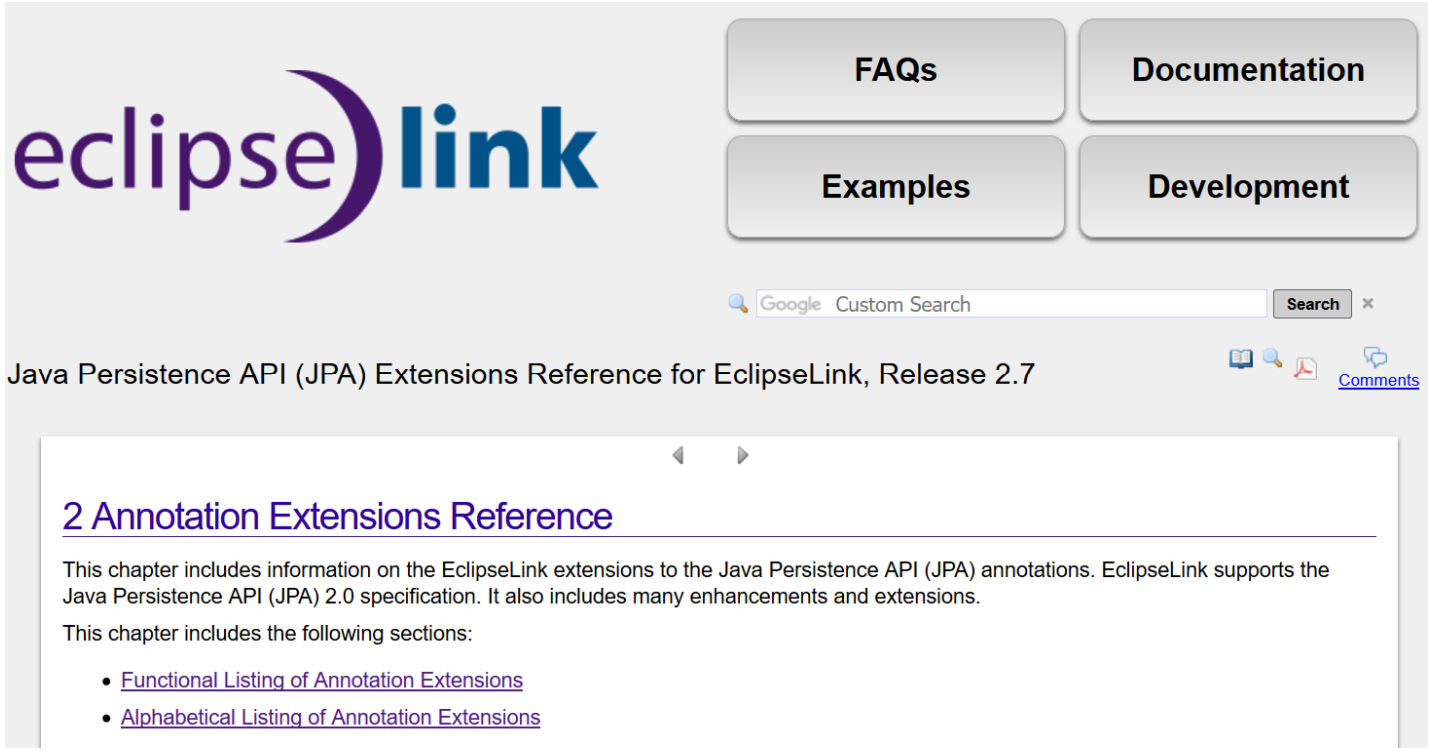
# Entité - Annotations

| | |
|---|---|
| @Entity (name="…") | Permet de marquer la classe comme **EJB Entity** (Obligatoire).<br>Possibilité d'attribuer un nom via l'attribut **name** (qui sera utilisé dans les requêtes JPQL) |
| @Table (name="…") | Permet de spécifier le nom de la table correspondante si différent du nom de la classe |
| @Id | Obligatoire avant l'attribut de la classe qui correspond à la **clé primaire**.<br>On peut spécifier la stratégie de génération de la clé primaire en ajoutant l'annotation permettant la délégation de cette tâche à JPA<br>**@GeneratedValue(** strategy=GenerationType.Auto**)** |

# Entité - Annotations

| | |
|---|---|
| @Column(<br>name="…",<br>length=…<br>) | Permet de spécifier le nom de la colonne si différent du nom de l'attribut et de modifier la longueur du champs chaîne de caractère (par défaut 255)<br>Remarque :<br>Les attributs de type non primitif et non enveloppe ont besoins d'annotations pour qu'ils soient pris en compte. Exemple de type DATE : @Temporal(TemporalType.DATE) |
| @Enumerated(<br>EnumType.String<br>) | Permet à un champs de type Enumération d'être considéré comme chaîne de caractères (ou comme entier qui correspond à l'ordre de définition en commençant de 0) |

# Entité - Annotations




http://www.eclipse.org/eclipselink/documentation/2.7/jpa/extensions/annotations_ref.htm

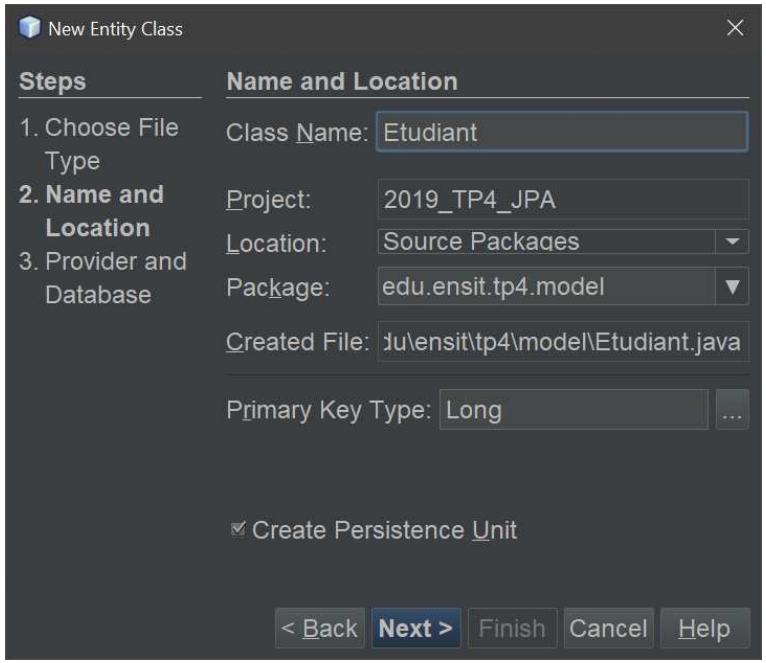
# Entité

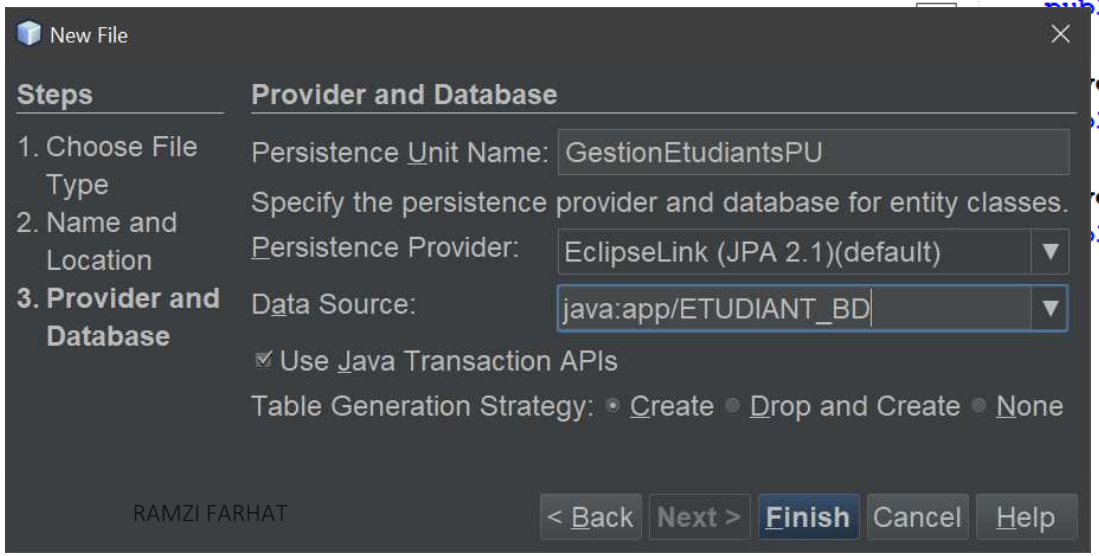
Etudiant

| id | nom | dateNaissance | niveauEtude |
|---|---|---|---|
| | | | |
| | | | |

```
@Entity (name="Etudiant")
public class Etudiant implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    @Column (name="nom", length=80)
    private String nom;
    @Column (name="dateNaissance")
    @Temporal (TemporalType.DATE)
    private Date dateNaissance;
    @Column (name="niveauEtude", length=30)
    @Enumerated (EnumType.STRING)
    private NiveauEtude niveauEtude;
    // Getters and Setters
    …
}
```

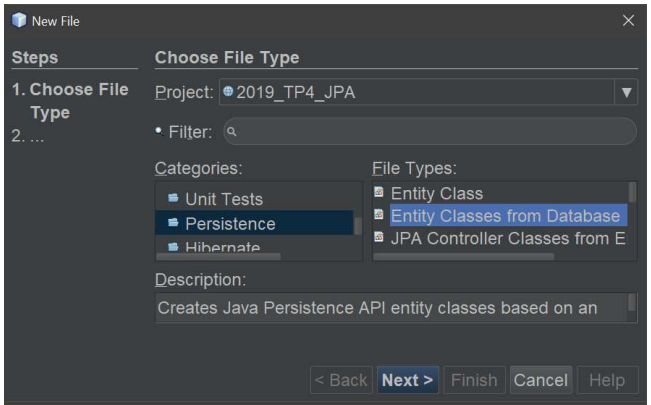
# Création d'entité dans NetBeans

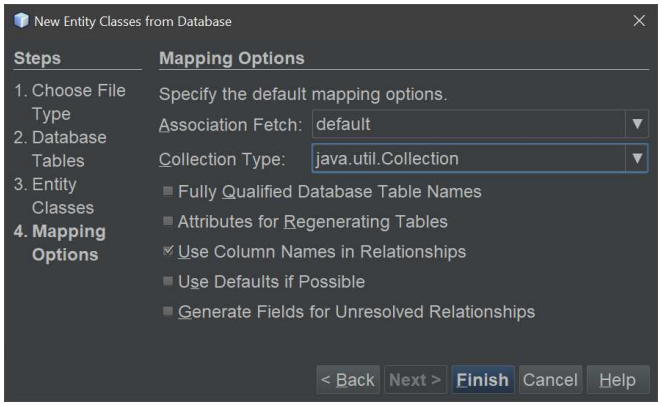


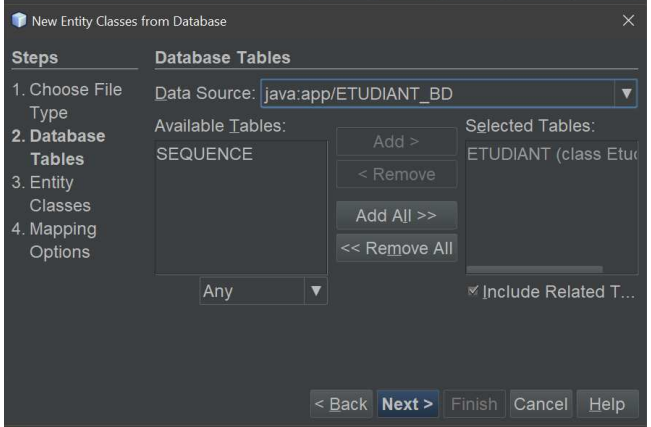



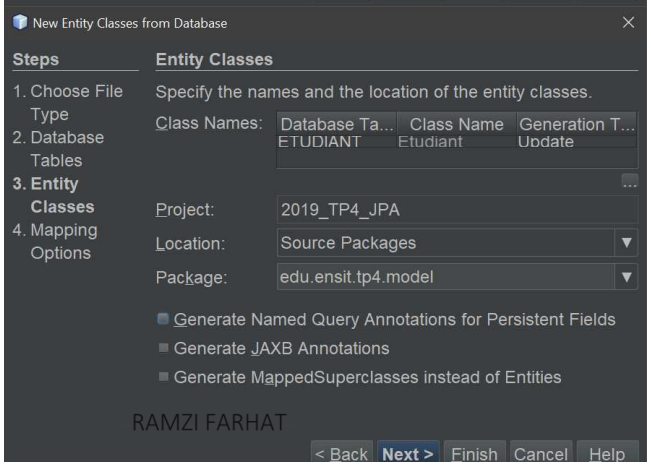

```
package edu.ensit.tp4.model;

import ...5 lines

@Entity
public class Etudiant implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    public Long getId() {...3 lines }

    public void setId(Long id) {...3 lines }

    @Override
    public int hashCode() {...5 lines }

    @Override
    public boolean equals(Object object) {...11 lines }

    @Override
    public String toString() {...3 lines }
```

# Création d'entité à partir de BD

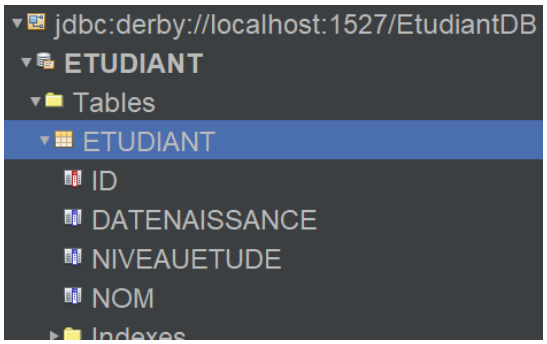



```java
package edu.ensit.tp4.model;

import ...13 lines

@Entity
@Table(name = "ETUDIANT")
@NamedQueries({
    @NamedQuery(name = "Etudiant.findAll", query = "SELECT e FROM Etudiant e")})
public class Etudiant implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @NotNull
    @Column(name = "ID")
    private Long id;
    @Column(name = "DATENAISSANCE")
    @Temporal(TemporalType.DATE)
    private Date datenaissance;
    @Size(max = 30)
    @Column(name = "NIVEAUETUDE")
    private String niveauetude;
    @Size(max = 80)
    @Column(name = "NOM")
    private String nom;

    public Etudiant() { ...2 lines }
```



# Composantes JPA

- EJB Entity

  Entreprise Java Beans qui contiennent la correspondance O/R via des annotations.

- Unité de persistance

  Fichier de configuration XML permettant de définir la source de données concernée.

- Entity Manager

  Objet responsable des opérations CRUD (Create, Read, Update et Delete) sur les données.

# Unité de persistance

- Fichier de configuration permettant d'indiquer au JPA :
  - A quel BD il doit s'adresser
  - Dans quel module de notre application nos classes sont rangés
  - etc.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns=http://java.sun.com/xml/ns/persistence … >

   <!-- transaction type : JTA ou RESOURCE_LOCAL -->
   <persistence-unit name="GestionEtudiantsPU" transaction-type="JTA">
   <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
   <!-- presistent classes list, can be JARs -->
   <class>edu.ensit.gestionetudiants.model.Etudiant</class>

   <properties>
         <property name="javax.persistence.jdbc.url"
                   value="jdbc:derby://localhost:1527/EtudiantDB" />
         <property name="javax.persistence.jdbc.driver"
                   value="org.apache.derby.jdbc.ClientDriver" />
         <property name="javax.persistence.jdbc.user"          value="USER_NAME" />
         <property name="javax.persistence.jdbc.password"      value="PASSWORD" />
   </properties>

</persistence>
```

# Unité de persistance

ensemble de classes persistantes (dont les entités JPA) qui dispose d'un nom

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns=http://java.sun.com/xml/ns/persistence … >

   <!-- transaction type : JTA ou RESOURCE_LOCAL -->
   <persistence-unit name="GestionEtudiantsPU" transaction-type="JTA">
   <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
   <!-- presistent classes list, can be JARs -->
   <class>edu.ensit.gestionetudiants.model.Etudiant</class>

   <properties>
         <property name="javax.persistence.jdbc.url"
                   value="jdbc:derby://localhost:1527/EtudiantDB" />
         <property name="javax.persistence.jdbc.driver"
                   value="org.apache.derby.jdbc.ClientDriver" />
         <property name="javax.persistence.jdbc.user"          value="USER_NAME" />
         <property name="javax.persistence.jdbc.password"      value="PASSWORD" />
   </properties>

</persistence>
```

# Unité de persistance

Implémentation de la spécification JPA
(Glassfish propose en standard l'implémentation EclipseLink)

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns=http://java.sun.com/xml/ns/persistence … >

   <!-- transaction type : JTA ou RESOURCE_LOCAL -->
   <persistence-unit name="GestionEtudiantsPU" transaction-type="JTA">
   <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
   <!-- presistent classes list, can be JARs -->
   <class>edu.ensit.gestionetudiants.model.Etudiant</class>

   <properties>
         <property name="javax.persistence.jdbc.url"
                   value="jdbc:derby://localhost:1527/EtudiantDB" />
         <property name="javax.persistence.jdbc.driver"
                   value="org.apache.derby.jdbc.ClientDriver" />
         <property name="javax.persistence.jdbc.user"          value="USER_NAME" />
         <property name="javax.persistence.jdbc.password"      value="PASSWORD" />
   </properties>

</persistence>
```

# Unité de persistance

Permettent de personnaliser le comportement du mapping O/R (exemples : URL de connexion à la BD, Driver JDBC, Login, Mot de passe, etc.)

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence xmlns=http://java.sun.com/xml/ns/persistence … >

   <!-- transaction type : JTA ou RESOURCE_LOCAL -->
   <persistence-unit name="GestionEtudiantsPU" transaction-type="JTA">
   <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
   <!-- presistent classes list, can be JARs -->
   <class>edu.ensit.gestionetudiants.model.Etudiant</class>

   <properties>
         <property name="javax.persistence.jdbc.url"
                   value="jdbc:derby://localhost:1527/EtudiantDB" />
         <property name="javax.persistence.jdbc.driver"
                   value="org.apache.derby.jdbc.ClientDriver" />
         <property name="javax.persistence.jdbc.user"          value="USER_NAME" />
         <property name="javax.persistence.jdbc.password"      value="PASSWORD" />
   </properties>

</persistence>
```

# Composantes JPA

- EJB Entity

  Entreprise Java Beans qui contiennent la correspondance O/R via des annotations.

- Unité de persistance

  Fichier de configuration XML permettant de définir la source de données concernée.

- Entity Manager

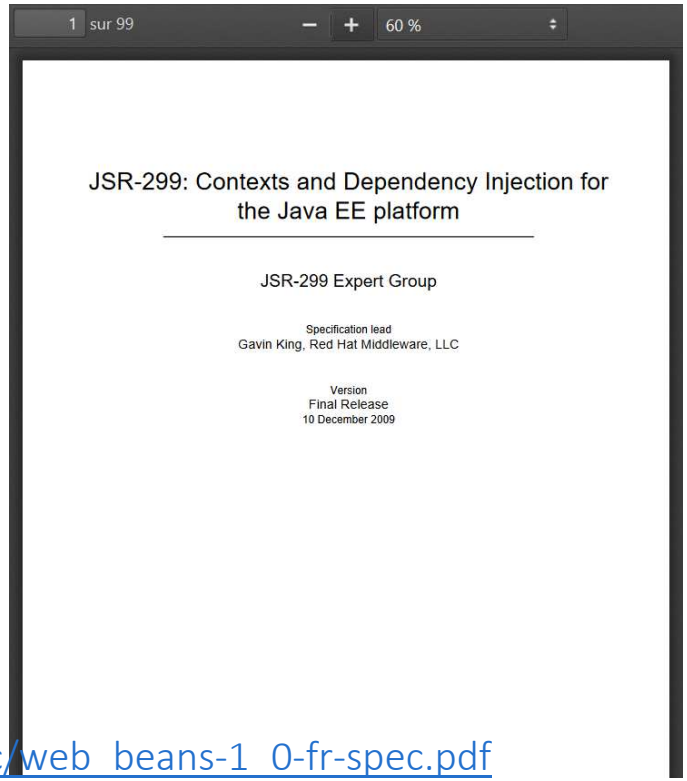
  Objet responsable des opérations CRUD (Create, Read, Update et Delete) sur les données.

# Entity Manager

- Assure la correspondance entre les entités (Objets) et la base de données (Tables)
- Permet d'assurer les opération CRUD
- Accessible via une **injection de dépendance** :

```
@PersistenceContext(unitName=" GestionEtudiantsPU ")
EntityManager em;
```

- **Injection de dépendance :**
  - Implémentation du principe d'inversion de contrôle (IoC) :
    - Dépendance entre les objets exprimée d'une façon statique dans le code
    - Dépendance déterminée dynamiquement au moment de l'exécution




https://download.oracle.com/otn-pub/jcp/web_beans-1.0-fr-oth-JSpec/web_beans-1_0-fr-spec.pdf

# Entity Manager

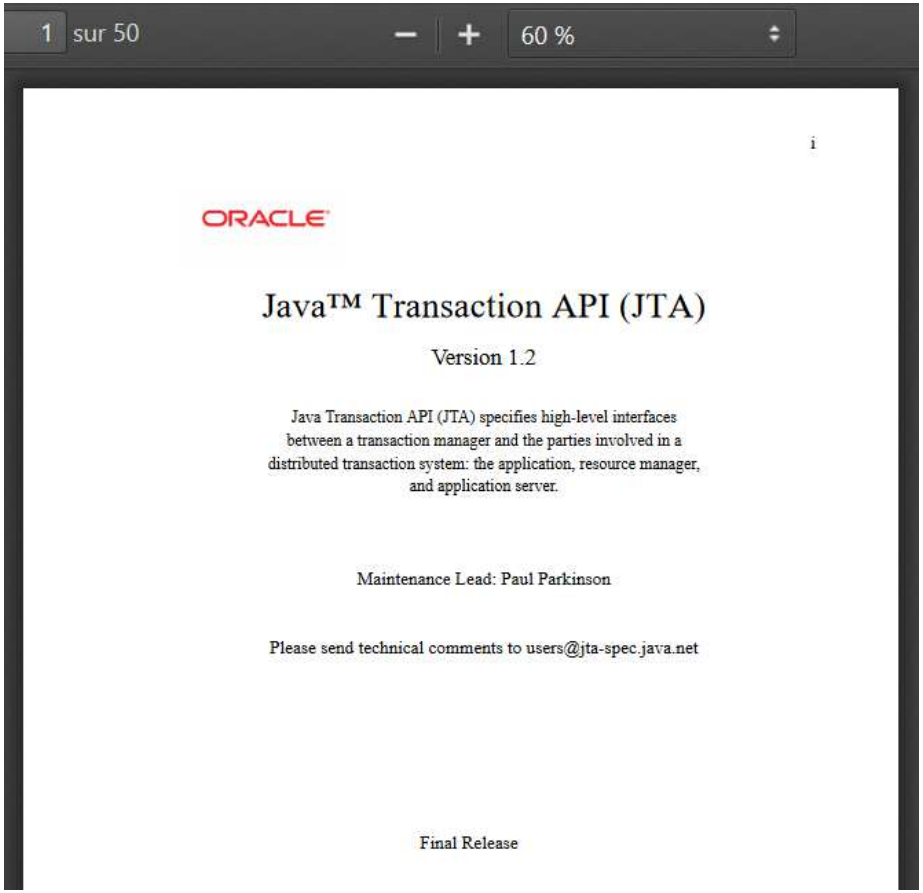
- Gestion des transaction
  - JTA : Transactions gérées automatiquement par le serveur
  - RESOURCE-LOCAL : Transactions gérées manuellement
- Exemple (RESOURCE-LOCAL)

```
em.getTransaction().begin();
em.getTransaction().commit();
em.getTransaction().rollback();
em.close(); // Pour récupérer les ressources
```

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
             xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
  <persistence-unit name="GestionEtudiantsPU" transaction-type="RESOURCE_LOCAL">
```

# JTA

- JTA : Java Transaction API
  - Simplifie la gestion des transaction réparties
  - Permet de gérer plusieurs ressources (exemple : plusieurs SGBDR) au sein d'une seul transaction
  - Exemple :




```
@Transactional
public void transferBancaire(String deCompte, String versCopmpte, Double somme){
    // Comptes dans Oracle
    gestionCompte.transfer(deCompte, deCompte, somme);
    // Audit des mouvements dans MySQL
    gestionAudit.log(deCompte, deCompte, somme);
}
```

# Entity Manager

## ❑Création

- ❑Création d'une instance de l'entité

```
Exemple : Etudiand etudiant= new Etudiant();
```

- ❑Initialisation des valeurs aves les mutateurs

```
Exemple : etudiant.setNom("Ramzi");
```

- ❑Persister l'entité via l'Entity Manager (au sein d'une transaction)

```
Exemple : em.persist(etudiant);
```

# Entity Manager

## Lecture de la BD

- Récupérer un objet via sa clé primaire via la méthode find()

Exemple :

Etudiant etudiant=em.find(Etudiant.class, id);

(Retourne null s'il n'y a pas d'enregistrement qui correspond à la clé primaire)

- Récupérer les résultats d'une requête (Requête nommée définie dans l'Entité ou requête non nommée)

Exemple :

```
Query q = em.createQuery("
SELECT OBJECT(e) FROM Etudiant e WHERE e.niveauEtude = :ne
");
q.setParameter("ne", NiveauEtude.G_INFO_2);
List<Etudiant> etudiants= (List<Etudiant>) q.getResultList();
```

# Entity Manager

## ❑Modifier la BD

- ❑Approche 1 : Objet récupérer par une requête d'un EM, toute modification est propagée dans la BD
- ❑Approche 2 : Objet créer avec une clé primaire existante dans la BD, il suffit de l'utiliser dans une transaction avec **merge()** pour MAJ la ligne correspondante dans le BD

Exemple :

```
Etudiant etudiant=em.find(Etudiant.class, id);
em.getTransaction().begin();
etudiant.setNiveauEtude("G_INFO_2");
em.getTransaction.commit();
em.close();
```

Exemple :

```
Etudiant etudiant= ... ;
EntityManager em=emf.createEntityManager();
em.getTransaction().begin();
etudiant=em.merge(etudiant);
em.getTransaction.commit();
em.close();
```

❑Autres méthodes utiles

- ❑void **refresh**(Object entité) Rafraîchit l'état de l'entité à parrtir de la BD (en écrasant les modifications apportées)
- ❑void **flush**() Synchronise le contexte de persistance avec la BD

# EntityManager

## ❑Supprimer de la BD

- ❑Récupérer un objet de la BD
- ❑Passer l'objet comme paramètre à la méthode **remove()** de l'EM

Exemple :

```
Etudiant etudiant=em.find(Etudiant.class, 1L);
em.getTransaction().begin();
em.remove(etudiant);
em.getTransaction().commit();
em.close();
```

# JPQL

- JPQL : JPA Qurey Language
  - Requêtes décrites par des annotations
  - Déclaration de requêtes nommées dans un Entity Bean :

    *@NamedQueries({ …})*
  - Déclaration d'une Requête nommée :

    *@NamedQuery(name= …, query="…")*

  Exemple :

```
@Entity
@NamedQueries({
   @NamedQuery(name="Etudiant.findAll", query="SELECT e FROM Etudiant e"),
   @NamedQuery(name="Etudiant.findByNom", query="SELECT e FROM Etudiant
e WHERE e.nom=:nom")
})
public class Etudiant{ … }
```

# JPQL

## ❑Utilisation des requêtes nommées

- ❑Créer un Objet requête qui représente la requête nommée

  Exemple :

```
Query query = em.createNamedQuery(Etudiant.findByNom);
```

- ❑Fixer les éventuels paramètres de la requête

  Exemple :

```
query.setParameter("nom","Ramzi");
```

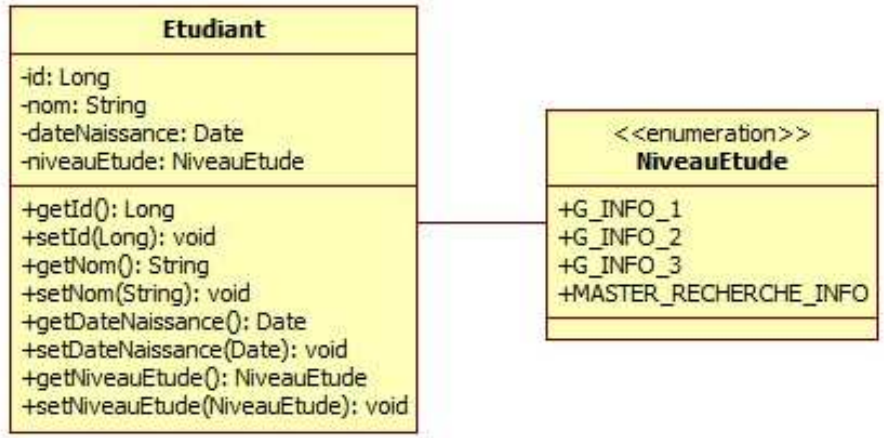
- ❑Exécuter la requête

  Exemple :

```
List<Etudiant> result = (List <Etudiant>) query.getResultList();
```

# Exercice

1. *Création d'une BD*
2. *Création de l'entité Etudiant*




3. *Création des requêtes nommées*

*Etudiant.findAll* permettant de retourner tous les étudiants dans la BD

*Etudiant.findByNom* permettant de trouver des étudiants à partir du nom

- Introduction JPA
- Entité
  - Présentation
  - Annotation
  - Création
- Unité de persistance
- Entité Manager
  - Présentation
  - CDI
  - Gestion de transaction
  - JTA
  - Création
  - Lecture
  - Modification
  - Suppression
- JPQL
  - Définition
  - Utilisation