

# Tutoriel

## (Java Persistence API)



**Amadou DIAW**

[jodyama@gmail.com](mailto:jodyama@gmail.com)

### Definition

**JPA** (*Java Persistence API*) version 2 est la partie de la spécification EJB 3.1 qui concerne la persistance des composants dans une base de données relationnelle

Un des grands avantages de JPA est que c'est une API indépendante et peut très bien s'intégrer avec J2EE, ainsi que les applications J2SE.

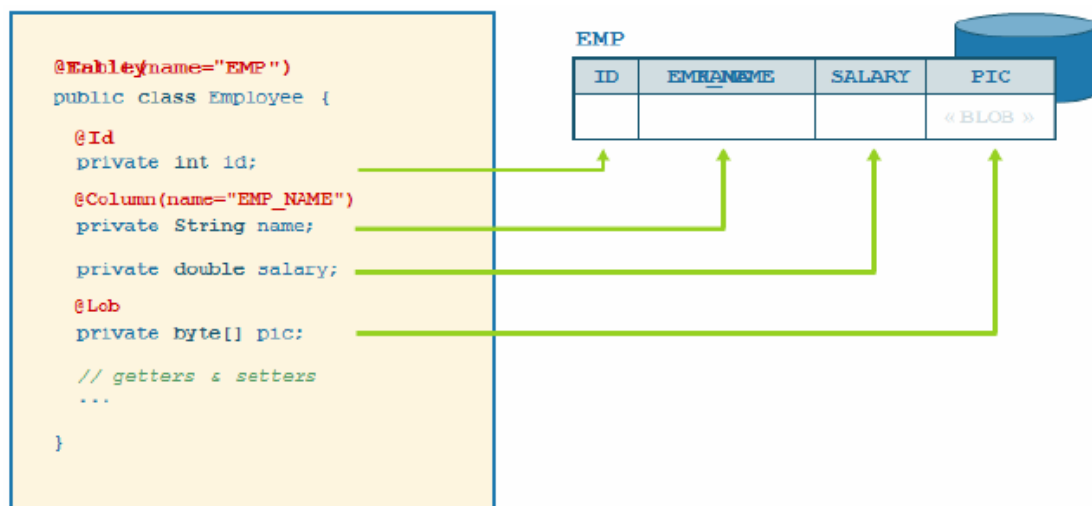
C'est l'API ORM (*object-relational mapping*) standard pour la persistance des objets Java

La couche JPA a sa place dans une architecture multicouche.

L'API de persistance est accessible en utilisant le paquetage Java `javax.persistence`.

### Le Mapping Objet Relationnel

La communication entre les mondes objet et relationnel suppose une transformation (ou mapping) pour adapter la structure des données relationnelles au modèle objet.



## Quelques annotations importantes

Annotation	Description
@Entity	La classe est un bean entité
@Id	L'objet ou la fonction qui suit est la clé primaire de l'entité
@GeneratedValue[strategy="..."]	Définit une politique de génération automatique de la clé-primaire
@Table[name="..."]	Assigne l'entité à une table spécifique
@Column[name="..."]	Assigne l'attribut qui suit à une colonne spécifique de la table

## Principes JPA

Cette technologie est basée sur :

- ✓ un jeu d'interfaces et de classes permettant de séparer l'utilisateur d'un service de persistance (votre application) et le fournisseur d'un service de persistance,
- ✓ un jeu d'annotations pour préciser la mise en correspondance entre classes Java et tables relationnelles,
- ✓ un fournisseur de persistance (par exemple **Hibernate** ou **TopLink**),
- ✓ un fichier XML « **persistence.xml** » décrivant les moyens de la persistance (fournisseur, datasource, etc.)

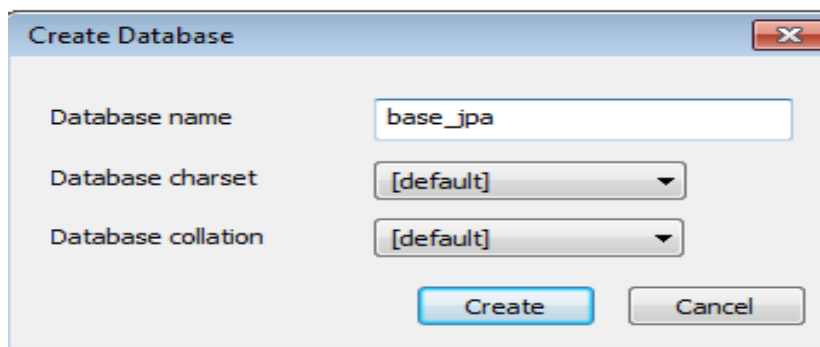
## Mise en œuvre

On se propose dans la suite de créer une application utilisant JPA pour l'accès à la base de données. Cette application va permettre de réaliser quelques fonctionnalités

- ✓ L'ajout d'une nouvelle personne.
- ✓ La mise à jour
- ✓ La suppression
- ✓ La recherche d'une personne existante dans la base et l'affichage de ses informations.
- ✓ Lister toutes les personnes présentes dans la base.

### Partie 1 : Création de la base de données

- ✓ Créer une base de données avec MySQL «**base\_jpa** »

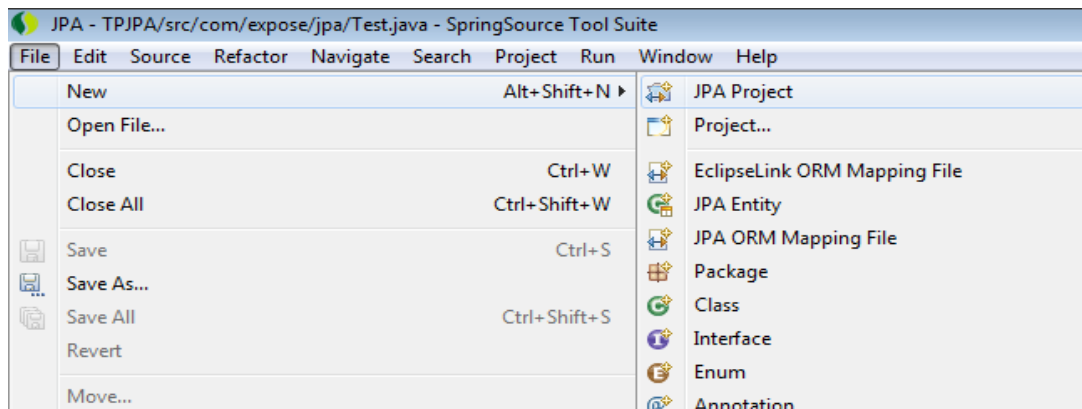


- ✓ Créer ensuite une table **personne** ayant la structure suivante

	Field Name	Datatype	Len	Default	PK?
*	ID	varchar	30		<input checked="" type="checkbox"/>
	NOM	varchar	80		<input type="checkbox"/>
	PRENOM	varchar	80		<input type="checkbox"/>

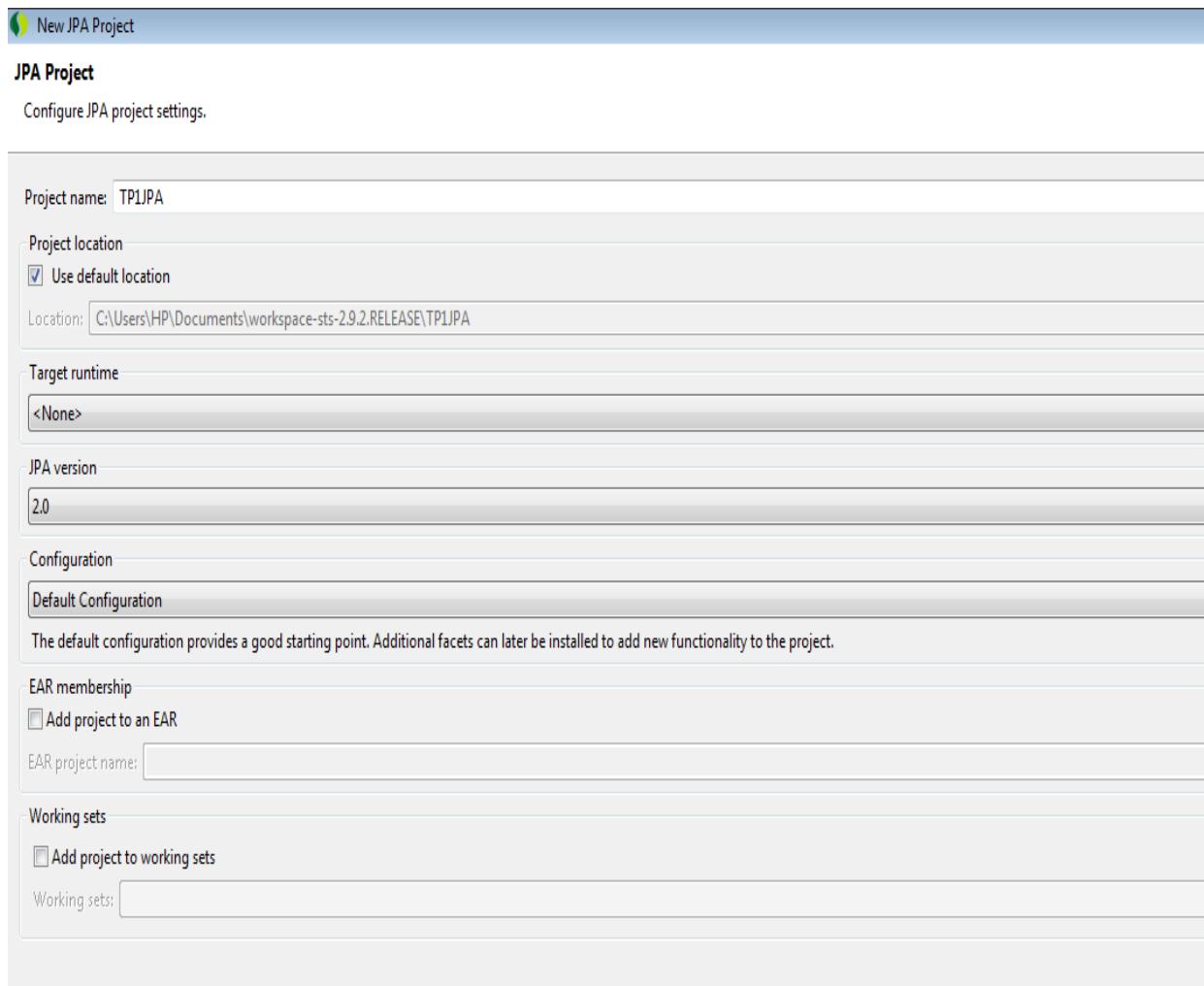
## Partie 2 : Création du projet sous eclipse

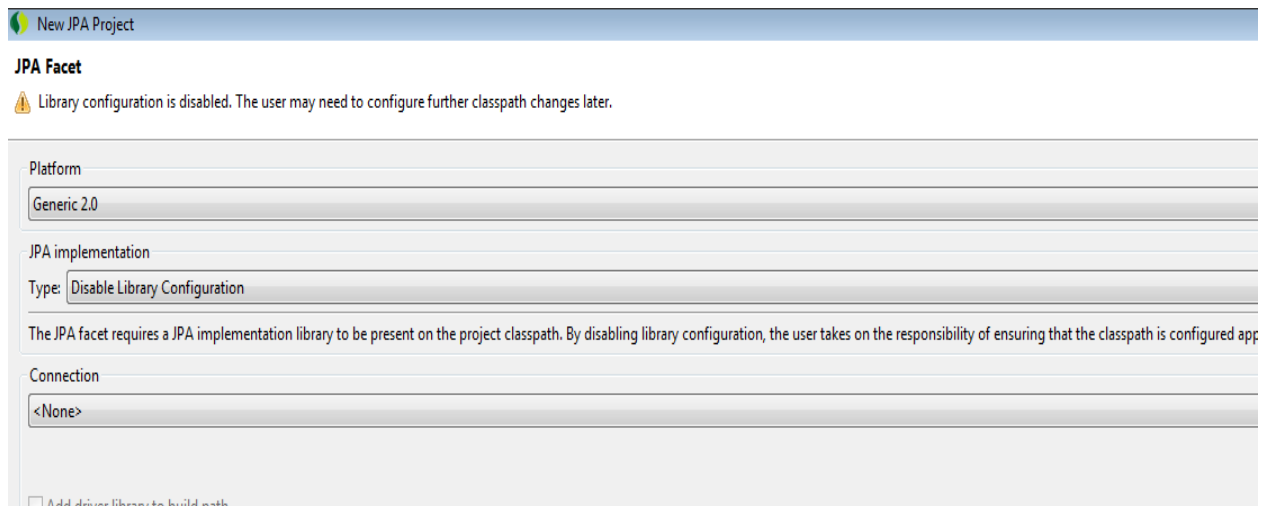
Créer un projet jpa et nommez le « TP1JPA »



Donner un nom au projet puis cliquez sur « Next »

**Selectionner la version 1.0 de JPA**





Cliquez ensuite sur « **finish** »

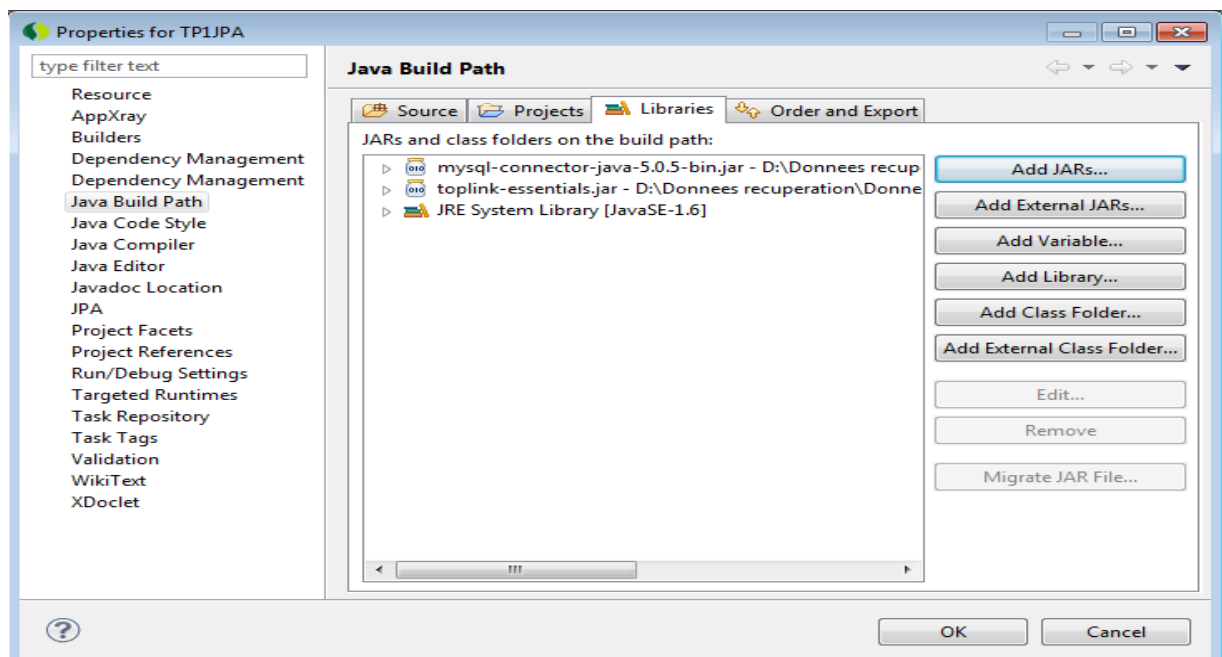
## Eléments nécessaires pour ce projet

- ✓ Java Persistence API : (Implémentation de référence (toplink-essentials.jar) d'ORACLE.
- ✓ Le pilote de connexion: *mysql-connector-java-3.1.10-bin.jar*.

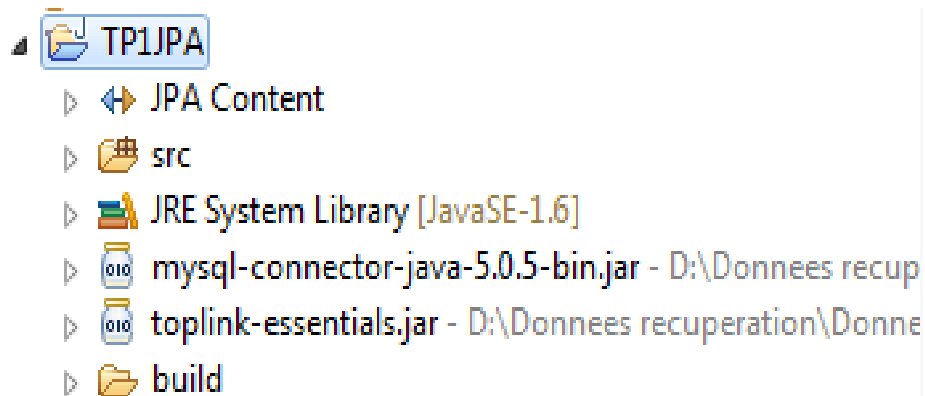
Ajouter les fichiers jars au CLASSPATH du projet

**Button droit** sur le projet --> **Build Path** --> **Configure Build Path** --> **Libraries** -->

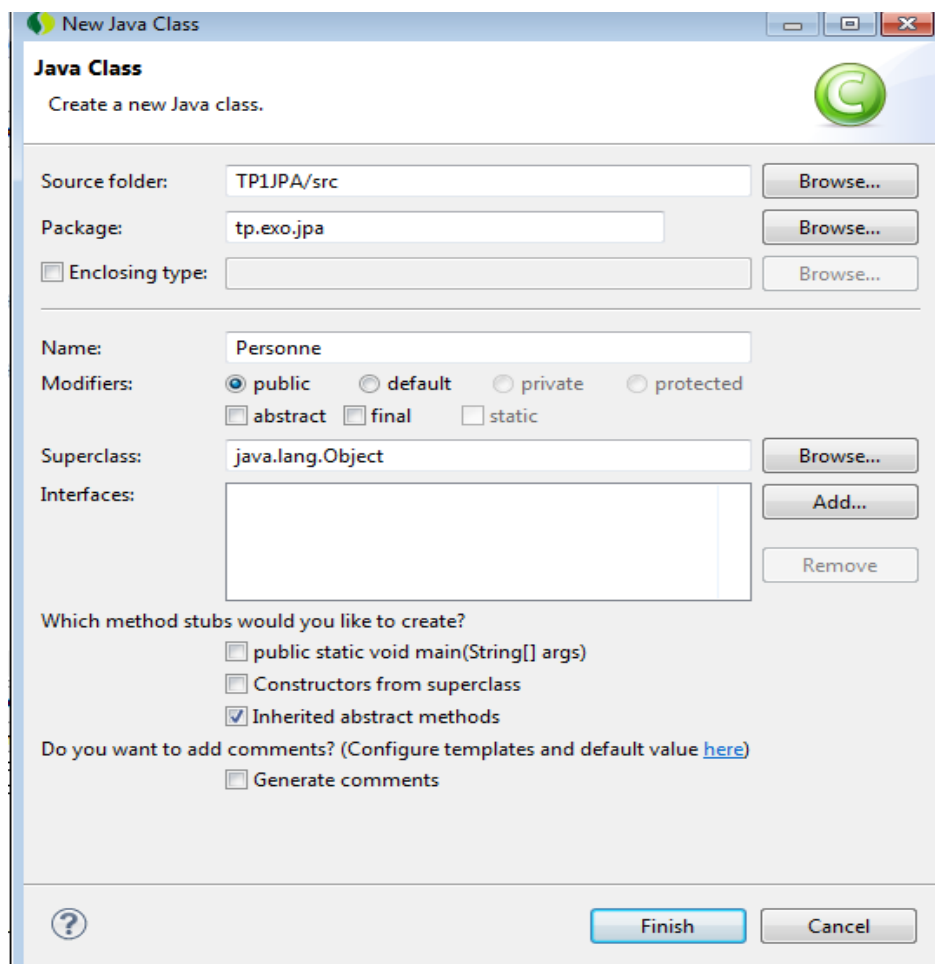
### Add External JARS



Ci-dessous la structure de notre projet



Créer une classe **Personne**



## Ajouter une entité persistante

Les classes dont les instances peuvent être persistantes sont appelées des entités dans la spécification de JPA

La configuration se fait à l'aide d'annotations Java **@Annotation**. Les annotations Java sont soit exploitées par le compilateur, soit par des outils spécialisés au moment de l'exécution.

L'annotation **@Entity** est la première annotation indispensable. Elle se place avant la ligne qui déclare la classe et indique que la classe en question doit être gérée par la couche de persistance JPA. En l'absence de cette annotation, toutes les autres annotations JPA seraient ignorées.

L'annotation **@Table** désigne la table de la base de données dont la classe est une représentation. Son principal argument est **name** qui désigne le nom de la table. En l'absence de cet argument, la table portera le nom de la classe.

L'annotation **@Id** sert à désigner le champ dans la classe qui est image de la clé primaire de la table. Cette annotation est obligatoire. Elle indique que le champ *id* est l'image de la clé primaire de la table.

L'annotation **@Column** sert à faire le lien entre un champ de la classe et la colonne de la table dont le champ est l'image. L'attribut **name** indique le nom de la colonne dans la table. En l'absence de cet attribut, la colonne porte le même nom que le champ.

```

package tp.exo.jpa;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

@Entity
@Table(name = "personne")

public class Personne implements Serializable {
    @Id
    @Column(table = "personne", name = "ID")
    private String id;
    @Column( table = "personne",name = "NOM")
    private String nom;
    @Column(table = "personne", name = "PRENOM")
    private String prenom;

    private static final long serialVersionUID = 1L;

    public Personne() {
        super();
    }

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public String getPrenom() {
        return prenom;
    }

    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
}

```



## Le fichier persistence.xml

Le fichier **persistence.xml** est utilisé pour gérer une ou plusieurs unités de persistance. Comme le fichier WEB.XML pour les **Servets**, et **ejb-jar.xml** pour les applications EJB, le fichier **persistence.xml** est indispensable pour réaliser le mapping dans JPA.

A un très haut niveau, nous pouvons réaliser la persistance de plusieurs unités au sein d'un même fichier unique persistence.xml identifié par son nom de balise. Chaque unité est configuré par son fournisseur de persistance, sa datasource et avec un certain nombre d'entités.

Donc, nous pouvons dire que la persistance d'une unité fournit un groupement logique d'un certain ensemble d'entités qui partagent une même configuration.

Chaque nouvelle entité définie dans les sources est automatiquement ajoutée dans un tag `<class>` du fichier persistence.xml.

## Mise à jour du fichier persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" 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">
  <persistence-unit name="TP1JPA">
    <provider>oracle.toplink.essentials.PersistenceProvider</provider>
    <class>tp.exo.jpa.Personne</class>
    <properties>
      <property name="toplink.jdbc.driver" value="com.mysql.jdbc.Driver"/>
      <property name="toplink.jdbc.url" value="jdbc:mysql://localhost:3306/base_jpa"/>
      <property name="toplink.jdbc.user" value="root"/>
    </properties>
  </persistence-unit>
</persistence>
```

## Créer une classe PersonneTools

```
package tp.exo.jpa;

import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;
import javax.persistence.Query;

public class PersonneTools {
    private static EntityManagerFactory entityManagerFactory = Persistence.createEntityManagerFactory("TP1JPA");
    public EntityManager em;

    public PersonneTools(){
        em=entityManagerFactory.createEntityManager();
    }

    public void beginTransaction(){
        em.getTransaction().begin();
    }

    public void commiTransaction(){
        em.getTransaction().commit();
    }

    public void failTransaction(){
        em.getTransaction().rollback();
    }

    public void insertPersonne(String id,String nom,String prenom){
        try{
            beginTransaction();
            Personne p=createPersonne(id, nom, prenom);
            em.persist(p);
        }

        catch(Exception e){
            failTransaction();
        }

        finally{
            commiTransaction();
        }
    }

    public List<Personne> findAllPersonne(){
        Query query=em.createQuery("SELECT p FROM Personne p");
        List personnes=query.getResultList();
        return (List<Personne>)personnes;
    }

    public Personne findByName(String nomPersonne){
        String requete="SELECT p FROM Personne p WHERE p.nom=:nom";
        Query query=em.createQuery(requete);
        query.setParameter("nom", nomPersonne);
        return (Personne)query.getSingleResult();
    }
}
```

```

private Personne createPersonne(String ID,String nom, String prenom){
    Personne personne = new Personne();
    personne.setId(ID);
    personne.setNom(nom);
    personne.setPrenom(prenom);
    return personne;
}
}

```

Tout d'abord, on demande un objet **EntityManagerFactory** *entityManagerFactory* pour l'unité de persistance « *TP1JPA*. »

Cette opération n'est faite normalement qu'une fois dans la vie d'une application.

Ensuite, on demande un objet **EntityManager** *entityManager* pour gérer un contexte de persistance.

On demande, aussi, un objet **Transaction** pour gérer une transaction. Les opérations sur le contexte de persistance se font à l'intérieur d'une transaction.

Pour ajouter une nouvelle personne, on procède comme suit :

- ✓ On commence la transaction
- ✓ On créer une nouvelle instance de l'entité Personne, qui n'est pas encore gérée par le contexte de persistance.
- ✓ Ce nouvel objet est intégré au contexte de persistance par l'opération **entityManager.persist(p)**, Il devient alors un objet persistant.

Pour rechercher une personne donnée, suivant son nom, on procède comme suit :

On exécute un ordre JPQL " SELECT p FROM Personne p WHERE p.nom = :nom"

*Personne* n'est pas la table mais l'objet @Entity associé à la table.

On a ici une requête JPQL (Java Persistence Query Language) sur le contexte de persistance et non un ordre SQL sur la base de données. Ceci dit, en-dehors de l'objet Personne qui a remplacé la table personne, les syntaxes sont identiques.

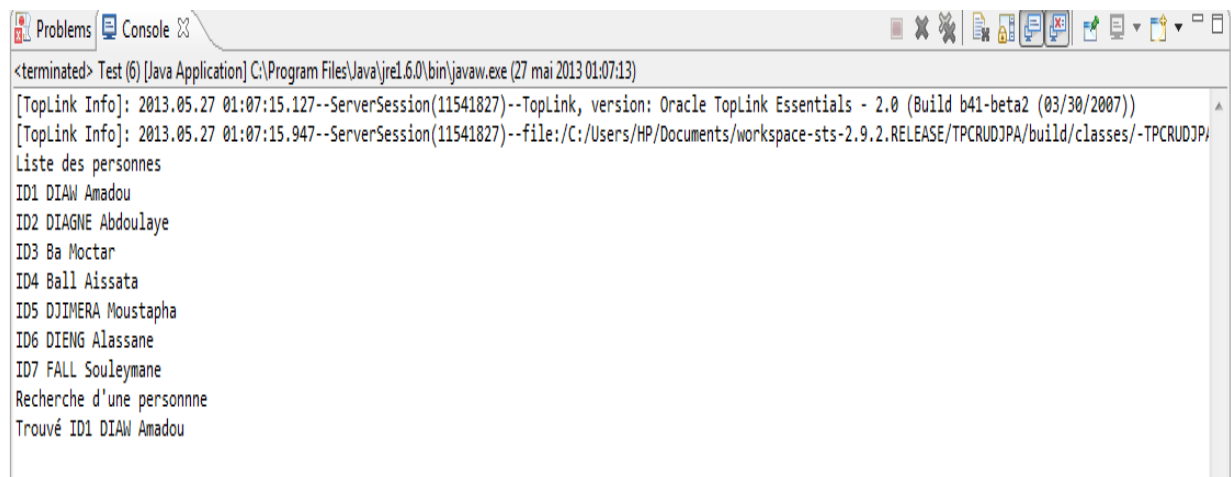
De façon transparente, une synchronisation du contexte de persistance avec la base va avoir lieu.

En effet, une requête select va être émise. C'est donc à ce moment, qu'en arrière-plan, JPA / TopLink va émettre l'ordre SQL insert qui va insérer la personne dans la table personne.

## Créer enfin la classe Test

```
package tp.exo.jpaa;  
  
import java.util.List;  
  
public class Test {  
  
    /**  
     * @param args  
     */  
    public static void main(String[] args) {  
        PersonneTools p=new PersonneTools();  
        p.insertPersonne("ID1", "DIAW", "Amadou");  
        p.insertPersonne("ID2", "DIAGNE", "Abdoulaye");  
        p.insertPersonne("ID3", "Ba", "Moctar");  
        p.insertPersonne("ID4", "Ball", "Aissata");  
        p.insertPersonne("ID5", "DJIMERA", "Moustapha");  
        p.insertPersonne("ID6", "DIENG", "Alassane");  
        p.insertPersonne("ID7", "FALL", "Souleymane");  
  
        System.out.println("Liste des personnes");  
  
        List<Personne>liste=p.findAllPersonne();  
        for(Personne p1:liste)  
            System.out.println(p1.getId()+" "+p1.getNom()+" "+p1.getPrenom());  
  
        System.out.println("Recherche d'une personne");  
  
        //p.insertPersonne("ID6", "BALL", "Adama");  
        Personne pchechr=p.findByName("DIAW");  
        System.out.println("Trouvé "+pchechr.getId()+" "+pchechr.getNom()+" "+pchechr.getPrenom());  
    }  
}
```

## Résultat



```
<terminated> Test (6) [Java Application] C:\Program Files\Java\jre1.6.0\bin\javaw.exe (27 mai 2013 01:07:13)  
[TopLink Info]: 2013.05.27 01:07:15.127--ServerSession(11541827)--TopLink, version: Oracle TopLink Essentials - 2.0 (Build b41-beta2 (03/30/2007))  
[TopLink Info]: 2013.05.27 01:07:15.947--ServerSession(11541827)--file:/C:/Users/HP/Documents/workspace-sts-2.9.2.RELEASE/TPCRUDJPA/build/classes/-TPCRUDJPA/  
Liste des personnes  
ID1 DIAW Amadou  
ID2 DIAGNE Abdoulaye  
ID3 Ba Moctar  
ID4 Ball Aissata  
ID5 DJIMERA Moustapha  
ID6 DIENG Alassane  
ID7 FALL Souleymane  
Recherche d'une personne  
Trouvé ID1 DIAW Amadou
```

Nous verrons dans un prochain tuto, les relations entre les entités et la partie JPAQL